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`

Feature	Description
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 (https://en.wikipedia.org/wiki/List_of_U.S._state_abbreviations#Postal_codes)). **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!</code`
project_essay_1	First application essay*
project_essay_2	Second application essay*
project_essay_3	Third application essay*
project_essay_4	Fourth application essay*
project_submitted_datetime	Datetime when project application was submitted. **Example:** `2016-04-28 12:43:56.245`
teacher_id	A unique identifier for the teacher of the proposed project. **Example:** `bdf8baa8fedef6bfeec7ae4ff1c15c56`

Feature	Description
`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
```

```
D:\Softwares\Anaconda\envs\AAIC\lib\site-packages\gensim\utils.py:1197: UserWarning: detected Windows; aliasin
g chunkize to chunkize_serial
  warnings.warn("detected Windows; aliasing chunkize to chunkize_serial")
```

1.1 Reading Data

In [2]:
```
project_data = pd.read_csv('train_data.csv')
resource_data = pd.read_csv('resources.csv')
```

In [3]:
```
import random
project_data = project_data.loc[random.sample(list(project_data.index), 50000)]
```

In [4]:
```
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 (50000, 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 [5]:
```
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[5]:

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

1.2 preprocessing of `project_subject_categories`

In [6]:
```
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 & Hunger"]
        if 'The' in j.split(): # this will split each of the catogory based on space "Math & Science"=> "Math","
            j=j.replace('The','') # if we have the words "The" we are going to replace it with ''(i.e removing
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"Math & Science"=>"Math&Sci
        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 [7]:
```
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 & Hunger"]
        if 'The' in j.split(): # this will split each of the catogory based on space "Math & Science"=> "Math","
            j=j.replace('The','') # if we have the words "The" we are going to replace it with ''(i.e removing
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"Math & Science"=>"Math&Sci
        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 [8]:
```
# 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 [9]:
```
project_data.head(2)
```

Out[9]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime	project_grade_ca
42895	98274	p099404	7c8ea02b56e468ecef1a18e94cd2464b	Mrs.	GA	2016-09-01 00:00:39	Grades
103967	157886	p077370	1ae060ae651839bef95baf4b49880ce9	Mrs.	AL	2016-05-29 19:03:02	Grad

In [10]:
```
#### 1.4.2.3 Using Pretrained Models: TFIDF weighted W2V
```

In [11]:
```
# 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)
```

```
My students are special because they have a unique desire to want to learn.  They face many challenges coming
to school, but despite all the challenges our students and our school succeeds.  Many of our students live in
poverty and lack the many resources they need at home and at school to meet the needs of our changing society.
We have students who attend our school who still don't have access to personal computers in their homes.  Many
students are able to use their parents smartphone as a means to connect with technology, but it is not a subst
itute for having a personal computer to complete homework assignments and learn more about the world around th
em.  \r\n\r\nMy students have a desire to want to know more about our world and how it works.  I'm empowered t
o help teach them all they need to know to be able to interact in a society that focuses on Math, Technology a
nd Science.  When students have a desire to learn it helps to motivate me to make sure they have the tools the
y need, so they will be able to make an impact on their future.My students will use iPads to explore learning
outside of the classroom.  They will be able to connect to virtual classes and activities that will spark thei
r interest.  My students will use the Legos and Magnetic building blocks to use their imagination as they cons
truct structures that will be three dimensional in size.  \r\n\r\nThree Dimensional shapes are a part of our c
urriculum in which students have the opportunity to use their engineering skills to build what they have envis
ioned. We will also use the additional materials to create math and science journals as a mean of journaling o
ur experiences.  \r\n\r\nThese materials will allow my students to be creative with their learning.  They will
be able to explore some of the unique building materials and technology that our school does not have the priv
ilege to own.  Technology and engineering is part of our future and I want my students to have access to what
they need to be successful.nannan
==================================================
The students at Davis/Northside High School are creative thinkers that strive to reach new potentials. They co
me from diverse economic backgrounds but that doesn't mean they wont work for you. It is my goal to get them i
nterested and passionate about reading, writing, and communicating in such a way that they earn the respect of
others. These scholars push through some of the hardest trials in life to come to school to get the education
that many have forfeited. I applaud their tenacity.These donations will help inner city students become profic
ient readers, critical thinkers, and explore the world through lenses of reality. They will engage teenagers w
ith real concepts that channel them to forge ahead through testimonies of other teens that have overcome simil
ar obstacles. By having this library of reading materials my students will gain exposure that they are not nor
mally afforded. \r\n They need this opportunity because Exposure Expands Expectations. These resources will be
a great blessing to struggling readers and those desiring to see the outside world around them for more than w
hat they see in their neighborhood. Let's Motivate our children to read because reading takes you places.nanna
n
==================================================
```

```
A typical day in my classroom starts out with greeting my 30 wonderful students. They come in ready and eager
to learn. For many of them, English is their second language. My goal is to get them as ready as I can for the
ir futures. \r\n\r\nThey are the sweetest and hardest-working students I have ever had. I work in an inner cit
y school. The majority of my students come from low income homes. We are very underfunded, and it has become e
xtremely difficult to provide my students with even the bare necessities that they need and deserve.\r\nContri
buting to this project will help my Kindergarten students take the first steps to becoming independent and avi
d readers. My Kindergarteners are bilingual students whose exposure to the English language is limited. \r\n\r
\nThe more that you read, the more things you will know, The more things that you learn, the more places you w
ill go.- Dr. Suess\r\n\r\nAfter they have mastered letter and sound recognition, our goal is to learn sight wo
rds. It takes various methods and practice with sight words for them to become automatic. Your donation will m
ake this possible. My students can practice their sight words in many different and fun ways with these materi
als. Then they will feel confident and be successful at reading their sight word readers. The love for reading
has begun!\r\n\r\nnannan
==================================================
My students are from a Title I school and are from all over the world. They love to read and keep several book
s in their desk at any given time. My students are eager learners and love using technology in the classroom o
r whenever they can. My students have wonderful imaginations that create wonderful works of art and let them b
e transported to any place imaginable when they are reading. My students have all different levels of reading
and mathematical abilities, but are all excited to learn.Math is frequently an abstract subject. Math involves
numbers that represent \"things.\" Being able to see and work with the \"things\" gives the students a much be
tter understanding of the concepts they are learning. My students have in the past drawn pictures to help them
understand the math concept, which helps, but only so far. Being able to build equal groups with their hands a
nd see them gives the students a concrete understanding of multiplication and division. Building shapes on geo
boards means the students have to apply what they are learning about shape characteristics. Also, turning any
math subject into a game automatically increases engagement and means the student WANTS to learn!nannan
==================================================
```

In [12]:
```
# 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 [13]:
```
sent = decontracted(project_data['essay'].values[20000])
print(sent)
print("="*50)
```

```
My students are from a Title I school and are from all over the world. They love to read and keep several book
s in their desk at any given time. My students are eager learners and love using technology in the classroom o
r whenever they can. My students have wonderful imaginations that create wonderful works of art and let them b
e transported to any place imaginable when they are reading. My students have all different levels of reading
and mathematical abilities, but are all excited to learn.Math is frequently an abstract subject. Math involves
numbers that represent \"things.\" Being able to see and work with the \"things\" gives the students a much be
tter understanding of the concepts they are learning. My students have in the past drawn pictures to help them
understand the math concept, which helps, but only so far. Being able to build equal groups with their hands a
nd see them gives the students a concrete understanding of multiplication and division. Building shapes on geo
boards means the students have to apply what they are learning about shape characteristics. Also, turning any
math subject into a game automatically increases engagement and means the student WANTS to learn!nannan
==================================================
```

In [14]:
```
# \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 students are from a Title I school and are from all over the world. They love to read and keep several book
s in their desk at any given time. My students are eager learners and love using technology in the classroom o
r whenever they can. My students have wonderful imaginations that create wonderful works of art and let them b
e transported to any place imaginable when they are reading. My students have all different levels of reading
and mathematical abilities, but are all excited to learn.Math is frequently an abstract subject. Math involves
numbers that represent  things.  Being able to see and work with the  things  gives the students a much better
understanding of the concepts they are learning. My students have in the past drawn pictures to help them unde
rstand the math concept, which helps, but only so far. Being able to build equal groups with their hands and s
ee them gives the students a concrete understanding of multiplication and division. Building shapes on geoboar
ds means the students have to apply what they are learning about shape characteristics. Also, turning any math
subject into a game automatically increases engagement and means the student WANTS to learn!nannan
```

In [15]:
```
#remove spacial character: https://stackoverflow.com/a/5843547/4084039
sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
print(sent)
```

```
My students are from a Title I school and are from all over the world They love to read and keep several books
in their desk at any given time My students are eager learners and love using technology in the classroom or w
henever they can My students have wonderful imaginations that create wonderful works of art and let them be tr
ansported to any place imaginable when they are reading My students have all different levels of reading and m
athematical abilities but are all excited to learn Math is frequently an abstract subject Math involves number
s that represent things Being able to see and work with the things gives the students a much better understand
ing of the concepts they are learning My students have in the past drawn pictures to help them understand the
math concept which helps but only so far Being able to build equal groups with their hands and see them gives
the students a concrete understanding of multiplication and division Building shapes on geoboards means the st
udents have to apply what they are learning about shape characteristics Also turning any math subject into a g
ame automatically increases engagement and means the student WANTS to learn nannan
```

In [16]:
```
# 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', 'af
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'f
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few',
            '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',
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'had
            "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',
            'won', "won't", 'wouldn', "wouldn't", 'nannan', 'nan']
```

In [17]:
```
# 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.lower() for e in sent.split() if e.lower() not in stopwords)
    preprocessed_essays.append(sent.strip())
```

```
100%|██████████████████████████████████████████████████████████████████████████████| 50000/50000 [00:34<00:00, 146
4.89it/s]
```

In [18]:
```
# after preprocesing
preprocessed_essays[20000]
```

Out[18]:
```
'students title school world love read keep several books desk given time students eager learners love using t
echnology classroom whenever students wonderful imaginations create wonderful works art let transported place
imaginable reading students different levels reading mathematical abilities excited learn math frequently abst
ract subject math involves numbers represent things able see work things gives students much better understand
ing concepts learning students past drawn pictures help understand math concept helps far able build equal gro
ups hands see gives students concrete understanding multiplication division building shapes geoboards means st
udents apply learning shape characteristics also turning math subject game automatically increases engagement
means student wants learn'
```

1.4 Preprocessing of `project_title`

In [19]:
```
# similarly you can preprocess the titles also
```

Following Code blocks provided by me.

In [20]:
```
# Code took from original code provided.
# Also function used from original code.
preprocessed_titles = []

for sent in tqdm(project_data['project_title'].values):
    sent = decontracted(sent)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    sent = ' '.join(e.lower() for e in sent.split() if e.lower() not in stopwords)
    preprocessed_titles.append(sent.strip())
```

```
100%|██████████████████████████████████████████████████████████████████████████| 50000/50000 [00:01<00:00, 3064
2.81it/s]
```

In [21]:
```
preprocessed_titles[20000]
```

Out[21]:
```
'hands math centers'
```

Following Code blocks present in original notebook.

1.5 Preparing data for models

In [22]:
```
project_data.columns
```

Out[22]:
```
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

- 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 [23]:
```
# 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  (50000, 9)
```

In [24]:
```
# 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_Governm
ent', 'ForeignLanguages', 'NutritionEducation', 'Warmth', 'Care_Hunger', 'SocialSciences', 'CharacterEducatio
n', 'PerformingArts', 'TeamSports', 'Other', 'College_CareerPrep', 'Music', 'History_Geography', 'Health_LifeS
cience', 'EarlyDevelopment', 'Gym_Fitness', 'ESL', 'EnvironmentalScience', 'VisualArts', 'Health_Wellness', 'A
ppliedSciences', 'SpecialNeeds', 'Literature_Writing', 'Mathematics', 'Literacy']
Shape of matrix after one hot encodig  (50000, 30)
```

In [25]:
```
# you can do the similar thing with state, teacher_prefix and project_grade_category also
```

Following Code blocks provided by me.

In [26]:
```
# Code took from original code provided.
states = project_data['school_state'].unique()
vectorizer = CountVectorizer(vocabulary=list(states), lowercase=False, binary=True)
vectorizer.fit(project_data['school_state'].values)
print(vectorizer.get_feature_names())

school_state_one_hot = vectorizer.transform(project_data['school_state'].values)
print("Shape of matrix after one hot encoding", school_state_one_hot.shape)
```

```
['GA', 'AL', 'SC', 'OK', 'CT', 'MN', 'MO', 'OR', 'NY', 'LA', 'CA', 'NJ', 'IN', 'VA', 'WA', 'IL', 'ID', 'TN',
'TX', 'MI', 'WI', 'PA', 'MA', 'FL', 'OH', 'MD', 'NV', 'NC', 'MS', 'KY', 'RI', 'DC', 'NH', 'AR', 'NM', 'AK', 'C
O', 'KS', 'UT', 'AZ', 'ME', 'IA', 'VT', 'NE', 'WV', 'HI', 'MT', 'DE', 'WY', 'SD', 'ND']
Shape of matrix after one hot encoding (50000, 51)
```

There are some NaN's in teacher_prefix column. replacing them with 'Mrs.' as that has high occurance in that column.

In [27]:
```
print("Number of NaN's before replacement in column: ", sum(project_data['teacher_prefix'].isna()))
project_data['teacher_prefix'] = project_data['teacher_prefix'].replace(np.nan, 'Mrs.', regex=True)
print("Number of NaN's after replacement in column: ", sum(project_data['teacher_prefix'].isna()))

# Output may show both zeros as I re-run this several times. But there are 3 zeros in original column.
```

```
Number of NaN's before replacement in column:  2
Number of NaN's after replacement in column:  0
```

In [28]:
```
# Code took from original code provided.
prefixes = project_data['teacher_prefix'].unique()
vectorizer = CountVectorizer(vocabulary=list(prefixes), lowercase=False, binary=True)
vectorizer.fit(project_data['teacher_prefix'].values)
print(vectorizer.get_feature_names())

teacher_prefix_one_hot = vectorizer.transform(project_data['teacher_prefix'].values)
print("Shape of matrix after one hot encoding", teacher_prefix_one_hot.shape)
```

```
['Mrs.', 'Ms.', 'Mr.', 'Teacher', 'Dr.']
Shape of matrix after one hot encoding (50000, 5)
```

In [29]:
```
grades = project_data['project_grade_category'].unique()
vectorizer = CountVectorizer(vocabulary=list(grades), lowercase=False, binary=True)
vectorizer.fit(project_data['project_grade_category'].values)
print(vectorizer.get_feature_names())

project_grade_category_one_hot = vectorizer.transform(project_data['project_grade_category'].values)
print("Shape of matrix after one hot encoding", project_grade_category_one_hot.shape)
```

```
['Grades PreK-2', 'Grades 9-12', 'Grades 3-5', 'Grades 6-8']
Shape of matrix after one hot encoding (50000, 4)
```

Following Code blocks present in original notebook.

1.5.2 Vectorizing Text data

1.5.2.1 Bag of words

In [30]:
```
# 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  (50000, 12156)
```

In [31]:
```
# you can vectorize the title also
# before you vectorize the title make sure you preprocess it
```

Following Code blocks provided by me.

In [32]:
```
# Code took from original code provided.
# We are considering only the words which appeared in at least 5 documents(rows or projects).
# Reduced number as title has less words
vectorizer = CountVectorizer(min_df=10)
titles_bow = vectorizer.fit_transform(preprocessed_titles)
print("Shape of matrix after one hot encodig ", titles_bow.shape)
```

```
Shape of matrix after one hot encodig  (50000, 2024)
```

Following Code blocks present in original notebook.

1.5.2.2 TFIDF vectorizer

In [33]:
```
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  (50000, 12156)
```

1.5.2.3 Using Pretrained Models: Avg W2V

In [34]:
```
# stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-pickle-to-save-and-load-va
# 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 [35]:
```
# 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%|████████████████████████████████████████████████████████████████████████| 50000/50000 [00:16<00:00, 297
5.16it/s]

50000
300
```

1.5.2.3 Using Pretrained Models: TFIDF weighted W2V

In [36]:
```
# 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 [37]:
```
# 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(ser
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf value for
            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%|██████████████████████████████████████████████████████████████████████████████| 50000/50000 [01:52<00:00, 44
6.27it/s]

50000
300
```

In [38]:
```
# Similarly you can vectorize for title also
```

Following Code blocks provided by me.

In [39]:
```
# Code took from original code provided.
# tfidf of project titles
vectorizer = TfidfVectorizer(min_df=10)
titles_tfidf = vectorizer.fit_transform(preprocessed_titles)
print("Shape of matrix after one hot encodig ",titles_tfidf.shape)
```

```
Shape of matrix after one hot encodig  (50000, 2024)
```

In [40]:
```
# Code took from original code provided.
# avg-w2v for project titles
avg_w2v_titles = []
for sentence in tqdm(preprocessed_titles):
    vector = np.zeros(300)
    cnt_words =0;
    for word in sentence.split():
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_titles.append(vector)

print(len(avg_w2v_titles))
print(len(avg_w2v_titles[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████████| 50000/50000 [00:00<00:00, 5166
5.94it/s]

50000
300
```

In [41]:
```
# Code took from original code provided.
tfidf_model = TfidfVectorizer()
tfidf_model.fit(preprocessed_titles)
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())
```

In [42]:
```
# Code took from original code provided.
# tfidf-w2v for project titles
tfidf_w2v_titles = []
for sentence in tqdm(preprocessed_titles):
    vector = np.zeros(300)
    tf_idf_weight =0
    for word in sentence.split():
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word]
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split()))
            vector += (vec * tf_idf)
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_titles.append(vector)

print(len(tfidf_w2v_titles))
print(len(tfidf_w2v_titles[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████████| 50000/50000 [00:01<00:00, 2929
1.62it/s]

50000
300
```

Following Code blocks present in original notebook.

1.5.3 Vectorizing Numerical features

In [43]:
```
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 [44]:
```
# check this one: https://www.youtube.com/watch?v=0HOqOcln3Z4&t=530s
# standardization sklearn: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScale
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 d
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 : 297.1031732, Standard deviation : 349.3284957374173
```

In [45]:
```
price_standardized
```

Out[45]:
```
array([[ 1.23576185],
       [ 1.33412199],
       [-0.07217611],
       ...,
       [ 0.03434254],
       [-0.51674334],
       [-0.54204903]])
```

Following Code blocks provided by me.

In [46]:
```
warnings.filterwarnings("ignore")
# Code took from original code provided
scalar = StandardScaler()
scalar.fit(project_data['teacher_number_of_previously_posted_projects'].values.reshape(-1, 1))
print(f"Mean : {scalar.mean_[0]}, Standard deviation : {np.sqrt(scalar.var_[0])}")

# Now standardize the data with above maen and variance.
previously_posted_projects_standardized = \
                scalar.transform(project_data['teacher_number_of_previously_posted_projects'].values.reshape(-1,
print(previously_posted_projects_standardized)
```

```
Mean : 11.07976, Standard deviation : 27.467873567904743
[[-0.18493459]
 [-0.33055926]
 [-0.4033716 ]
 ...
 [ 0.32475175]
 [-0.4033716 ]
 [-0.33055926]]
```

Following Code blocks present in original notebook.

1.5.4 Merging all the above features

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

In [47]:
```
print(categories_one_hot.shape)
print(sub_categories_one_hot.shape)
print(text_bow.shape)
print(price_standardized.shape)
```

```
(50000, 9)
(50000, 30)
(50000, 12156)
(50000, 1)
```

In [48]:
```
# 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[48]:
```
(50000, 12196)
```

In [49]:
```
# 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__

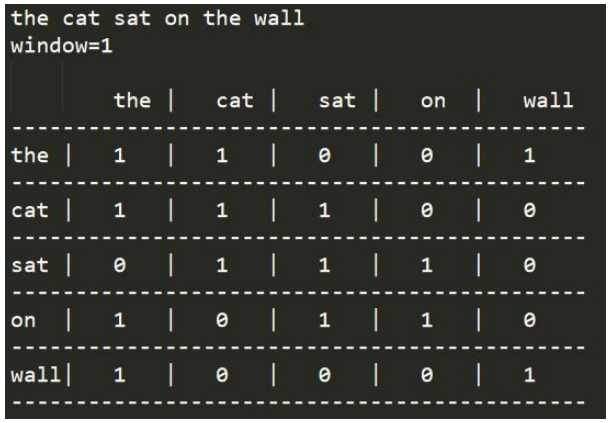
In [50]:
```
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer

# 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 bigges
for learning my students learn in many different ways using all of our senses and multiple intelligences i use a
of techniques to help all my students succeed students in my class come from a variety of different backgrounds
for wonderful sharing of experiences and cultures including native americans our school is a caring community of
learners which can be seen through collaborative student project based learning in and out of the classroom kind
in my class love to work with hands on materials and have many different opportunities to practice a skill befor
mastered having the social skills to work cooperatively with friends is a crucial aspect of the kindergarten cur
montana is the perfect place to learn about agriculture and nutrition my students love to role play in our prete
in the early childhood classroom i have had several kids ask me can we try cooking with real food i will take th
and create common core cooking lessons where we learn important math and writing concepts while cooking deliciou
food for snack time my students will have a grounded appreciation for the work that went into making the food ar
of where the ingredients came from as well as how it is healthy for their bodies this project would expand our l
nutrition and agricultural cooking recipes by having us peel our own apples to make homemade applesauce make our
and mix up healthy plants from our classroom garden in the spring we will also create our own cookbooks to be pr
shared with families students will gain math and literature skills as well as a life long enjoyment for healthy
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_` (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

In [51]:
```
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_round, verbose_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


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.grid_scores_)
best_parameters, score = clf.best_estimator_ , clf.best_score_
print('score:', score)
# for param_name in sorted(best_parameters.keys()):
#     print("%s: %r" % (param_name, best_parameters[param_name]))
```

```
score: 1.0
```

Following Code blocks provided by me.

Bulding data matrix with required columns

Adding a column `summary_numeric_bool` instead of `project_resource_summary` column which tells if resource summary has a number in it

In [131]:
```
# ref: https://stackoverflow.com/questions/4138202/using-isdigit-for-floats
def nums_in_str(text):
    """
    Returns list of numbers present in the given string. Numbers := floats ints etc.
    """
    result = []
    for s in text.split():
        try:
            x = float(s)
            result.append(x)
        except:
            continue
    return result
```

In [132]:
```
print(nums_in_str('HE44LLo 56 are -89 I 820.353 in -78.39 what .293 about 00'))
```

```
[56.0, -89.0, 820.353, -78.39, 0.293, 0.0]
```

In [133]:
```
numbers_in_summary = np.array([len(nums_in_str(s)) for s in project_data['project_resource_summary']])
project_data['summary_numeric_bool'] = list(map(int, numbers_in_summary>0))
```

Taking Relevant columns as X (input data to model)

In [134]:
```
project_data.columns
```

Out[134]:
```
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',
       'summary_numeric_bool'],
      dtype='object')
```

In [135]:
```
project_data.head(2)
```

Out[135]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime	project_grade_categor
0	98274	p099404	7c8ea02b56e468ecef1a18e94cd2464b	Mrs.	GA	2016-09-01 00:00:39	Grades PreK-
1	157886	p077370	1ae060ae651839bef95baf4b49880ce9	Mrs.	AL	2016-05-29 19:03:02	Grades 9-1

2 rows × 21 columns

In [136]:
```
# Categorical and numerical columns are listed below.
X_columns = ['teacher_prefix', 'school_state', 'project_grade_category', 'summary_numeric_bool',\
             'teacher_number_of_previously_posted_projects', 'clean_categories', 'clean_subcategories',\
             'price', 'quantity']
X = project_data[X_columns]
y = project_data['project_is_approved']
```

Adding preprocessed_essays and preprocessed_titles as columns to X before splitting

In [137]:
```
X['essay'] = preprocessed_essays
X['project_title'] = preprocessed_titles
X_columns.append('essay')
X_columns.append('project_title')
X['essay_and_title'] = X['essay'] + ' ' + X['project_title']
X_columns.append('essay_and_title')
print('columns of X: ', X_columns)
```

```
columns of X:  ['teacher_prefix', 'school_state', 'project_grade_category', 'summary_numeric_bool', 'teacher_n
umber_of_previously_posted_projects', 'clean_categories', 'clean_subcategories', 'price', 'quantity', 'essay',
'project_title', 'essay_and_title']
```

In [138]:
```
X['essay_1'] = project_data['project_essay_1']
X['essay_2'] = project_data['project_essay_2']
X['essay_3'] = project_data['project_essay_3']
X['essay_4'] = project_data['project_essay_4']
```

In [139]:
```
sia = SentimentIntensityAnalyzer()
for esnum in range(1, 5):
    sentim_data = []
    for es in project_data['project_essay_' + str(esnum)]:
        sentim_data.append(list(sia.polarity_scores(str(es)).values()))
    df_cols = ['essay' + str(esnum) + '_neg', 'essay' + str(esnum) + '_nue',\
    'essay' + str(esnum) + '_pos', 'essay' + str(esnum) + '_comp']
    sentim_data = pd.DataFrame(sentim_data, columns=df_cols)
    X = pd.concat([X, sentim_data], axis=1)

X['essay_word_count'] = [len(es.split()) for es in X['essay']]
X['title_word_count'] = [len(title.split()) for title in X['project_title']]
```

Splitting the data into Train and test

In [140]:
```
from sklearn.model_selection import train_test_split
```

In [141]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20)
```

In [142]:
```
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)
```

```
(40000, 34) (40000,)
(10000, 34) (10000,)
```

2. TruncatedSVD

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

In [143]:
```
# 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 [144]:
```
vectorizer = TfidfVectorizer()
vectorizer.fit(X['essay_and_title'])
idf_vals = dict(zip(vectorizer.get_feature_names(), vectorizer.idf_))
idf_vals = dict(sorted(idf_vals.items(), key=lambda x: x[1])[:2000])
```

In [145]:
```
imp_words = dict(zip(idf_vals.keys(), range(2000)))
```

2.2 Computing Co-occurance matrix

In [146]:
```
# 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 [147]:
```
co_occ_mat = np.zeros((2000, 2000)) # initializing the co-occurance matrix
```

In [148]:
```
for sentence in tqdm(X['essay_and_title']):
    lw = sentence.split()
    length = len(lw)-1
    for i, m in enumerate(lw):
        if m in imp_words.keys():
            for j in range(max(i-5,0),min(i+5,length)):
                if lw[j] in imp_words.keys():
                    co_occ_mat[imp_words[m], imp_words[lw[j]]] += 1
```

```
100%|██████████████████████████████████████████████████████████████████████████████| 50000/50000 [01:22<00:00, 606.59it/s]
```

In [231]:
```
print(f'{np.count_nonzero(co_occ_mat)}/4000000 are non zeros in the co-occurance matrix')
```

```
2560775/4000000 are non zeros in the co-occurance matrix
```

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

In [150]:
```
# 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
```

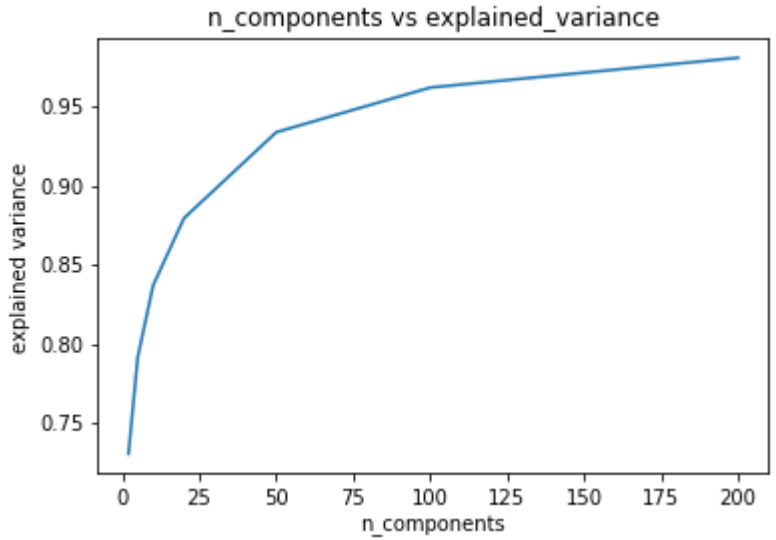
We have `imp_words` which is a dict indicating which word represents which row. And `co_occ_mat` which is the co-occurance matrix.

In [151]:
```
from sklearn.decomposition import TruncatedSVD
```

In [152]:
```
n_comps = [2, 5, 10, 20, 50, 100, 200]
svd_models = {}
variances = []
for comp in n_comps:
    svd = TruncatedSVD(n_components=comp)
    svd.fit(co_occ_mat)
    var = svd.explained_variance_ratio_.sum()
    variances.append(var)
    svd_models[comp] = svd
    print(f"variance for n_components = {comp} is {var}")
```

```
variance for n_components = 2 is 0.7306607346174743
variance for n_components = 5 is 0.7917173311647453
variance for n_components = 10 is 0.8370578545944767
variance for n_components = 20 is 0.8794634282743422
variance for n_components = 50 is 0.933845339820625
variance for n_components = 100 is 0.9620579422273324
variance for n_components = 200 is 0.9808569394492531
```

In [153]:
```
plt.plot(n_comps, variances)
plt.title('n_components vs explained_variance')
plt.xlabel('n_components')
plt.ylabel('explained variance')
plt.show()
```



Taking n_components = 50 as our best by elbow method. Now we use the model to get the vectors for each word.

In [154]:
```
vec_matrix = svd_models[50].transform(co_occ_mat)
```

In [155]:
```
vec_matrix.shape
```

Out[155]:
```
(2000, 50)
```

In [156]:
```
word_vec = pd.DataFrame(vec_matrix, index=imp_words.keys())
word_vec.head()
```

Out[156]:

	0	1	2	3	4	5	6	7	
students	450662.737858	-33272.854848	-30631.236936	4803.935315	8484.457436	-97.408272	-836.811994	-1399.214171	-2564.2
school	94595.740674	130525.194656	-6657.877739	3438.132983	5112.093527	-2406.701995	-7118.673945	451.676843	-4155.7
learning	64335.874826	-2843.974721	66629.769370	55129.551128	14701.710437	5609.668619	3500.043311	-1195.455732	1884.5
classroom	55312.415214	-2093.746568	48259.925475	-54322.745620	37457.643263	-1125.764959	1120.263249	-392.599724	1087.0
not	41252.348402	7334.844552	13964.186574	-13229.083177	-29218.849531	47660.376263	12384.310788	-9467.514938	-10712.7

5 rows × 50 columns

The above dataframe has word vectors for 2000 words each having a dimention of 50. Now we vectorize our essay and project title with these word vectors.

In [157]:
```
essay_vect = []
for sentence in tqdm(X_train['essay']):
    vector = np.zeros(50)
    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 imp_words.keys():
            vector += word_vec.loc[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    essay_vect.append(vector)
essay_train_vect = np.array(essay_vect)
print(essay_train_vect.shape)
```

```
100%|████████████████████████████████████████████████████████████████| 40000/40000 [21:39<00:00, 30.78it/s]

(40000, 50)
```

In [158]:
```
essay_vect = []
for sentence in tqdm(X_test['essay']):
    vector = np.zeros(50)
    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 imp_words.keys():
            vector += word_vec.loc[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    essay_vect.append(vector)
essay_test_vect = np.array(essay_vect)
print(essay_test_vect.shape)
```

```
100%|████████████████████████████████████████████████████████████████████████████| 10000/10000 [05:23<00:00, 3
0.91it/s]

(10000, 50)
```

In [159]:
```
title_vect = []
for sentence in tqdm(X_train['project_title']):
    vector = np.zeros(50)
    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 imp_words.keys():
            vector += word_vec.loc[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    title_vect.append(vector)
title_train_vect = np.array(title_vect)
print(title_train_vect.shape)
```

```
100%|████████████████████████████████████████████████████████████████████████████| 40000/40000 [00:36<00:00, 110
8.04it/s]

(40000, 50)
```

In [160]:
```
title_vect = []
for sentence in tqdm(X_test['project_title']):
    vector = np.zeros(50)
    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 imp_words.keys():
            vector += word_vec.loc[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    title_vect.append(vector)
title_test_vect = np.array(title_vect)
print(title_test_vect.shape)
```

```
100%|██████████████████████████████████████████████████████████████████████████| 10000/10000 [00:09<00:00, 107
6.61it/s]

(10000, 50)
```

2.4 Merge the features from step 3 and step 4

In [161]:
```
# 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
```

encoding numerical, categorical features

In [162]:
```
# 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
```

numerical columns

- teacher_number_of_previously_posted_projects
- price
- quantity

Leaving summary_numeric_bool as it is because it only has 0's and 1's in it.

categorical columns

- teacher_prefix
- school_state
- project_grade_category
- clean_categories
- clean_subcategories

Normalizing teacher_number_of_previously_posted_projects column

In [163]:
```
warnings.filterwarnings("ignore")
# Code took from original Code provided.
scaler = StandardScaler()
scaler.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))
print(f"Mean : {scaler.mean_[0]}, Standard deviation : {np.sqrt(scaler.var_[0])}")
```

```
Mean : 11.030425, Standard deviation : 27.291773473326625
```

In [164]:
```
warnings.filterwarnings("ignore")
X_train_tnppp_norm = scaler.transform(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,
X_test_tnppp_norm = scaler.transform(X_test['teacher_number_of_previously_posted_projects'].values.reshape(-1,1)
```

Normalizing price column

In [165]:
```
# Code took from original Code provided.
scaler = StandardScaler()
scaler.fit(X_train['price'].values.reshape(-1,1))
print(f"Mean : {scaler.mean_[0]}, Standard deviation : {np.sqrt(scaler.var_[0])}")
```

```
Mean : 296.85083325, Standard deviation : 350.3734509204403
```

In [166]:
```
X_train_price_norm = scaler.transform(X_train['price'].values.reshape(-1,1))
X_test_price_norm = scaler.transform(X_test['price'].values.reshape(-1,1))
```

Normalizing quantity column

In [167]:
```
warnings.filterwarnings("ignore")
# Code took from original Code provided.
scaler = StandardScaler()
scaler.fit(X_train['quantity'].values.reshape(-1,1))
print(f"Mean : {scaler.mean_[0]}, Standard deviation : {np.sqrt(scaler.var_[0])}")
```

```
Mean : 16.935875, Standard deviation : 26.23959532813673
```

In [168]:
```
warnings.filterwarnings("ignore")
X_train_quant_norm = scaler.transform(X_train['quantity'].values.reshape(-1,1))
X_test_quant_norm = scaler.transform(X_test['quantity'].values.reshape(-1,1))
```

Using a array to store column names data to use at last when interpreting the model

In [169]:
```
# when combining the input matrix the order of columns is same as cat_num_columns
cat_num_columns = ['previously_posted_projects', 'price', 'quantity', 'summary_numeric_bool']
```

Encoding teacher_prefix column

In [170]:
```
# Code took from SAMPLE_SOLUTION notebook.
vectorizer = CountVectorizer()
vectorizer.fit(X_train['teacher_prefix'].values)
print(vectorizer.get_feature_names())
```

```
['dr', 'mr', 'mrs', 'ms', 'teacher']
```

In [171]:
```
# Code took from SAMPLE_SOLUTION notebook.
X_train_prefix_ohe = vectorizer.transform(X_train['teacher_prefix'].values)
X_test_prefix_ohe = vectorizer.transform(X_test['teacher_prefix'].values)

print(X_train_prefix_ohe.shape)
print(X_test_prefix_ohe.shape)
```

```
(40000, 5)
(10000, 5)
```

In [172]:
```
cat_num_columns.extend(['prefix_'+i for i in vectorizer.get_feature_names()])
```

Encoding school_state column

In [173]:
```
# Code took from SAMPLE_SOLUTION notebook.
vectorizer = CountVectorizer()
vectorizer.fit(X_train['school_state'].values)
print(vectorizer.get_feature_names())
```

```
['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl', 'ga', 'hi', 'ia', 'id', 'il', 'in', 'ks', 'ky',
'la', 'ma', 'md', 'me', 'mi', 'mn', 'mo', 'ms', 'mt', 'nc', 'nd', 'ne', 'nh', 'nj', 'nm', 'nv', 'ny', 'oh', 'o
k', 'or', 'pa', 'ri', 'sc', 'sd', 'tn', 'tx', 'ut', 'va', 'vt', 'wa', 'wi', 'wv', 'wy']
```

In [174]:
```
# Code took from SAMPLE_SOLUTION notebook.
X_train_school_ohe = vectorizer.transform(X_train['school_state'].values)
X_test_school_ohe = vectorizer.transform(X_test['school_state'].values)

print(X_train_school_ohe.shape)
print(X_test_school_ohe.shape)
```

```
(40000, 51)
(10000, 51)
```

In [175]:
```
cat_num_columns.extend(['state_'+i for i in vectorizer.get_feature_names()])
print(len(cat_num_columns))
```

```
60
```

Encoding project_grade_category column

In [176]:
```
# Code took from original Code provided.
grades = X_train['project_grade_category'].unique()
vectorizer = CountVectorizer(vocabulary=list(grades), lowercase=False, binary=True)
vectorizer.fit(X_train['project_grade_category'].values)
print(vectorizer.get_feature_names())
```

```
['Grades PreK-2', 'Grades 6-8', 'Grades 3-5', 'Grades 9-12']
```

In [177]:
```
# Code took from SAMPLE_SOLUTION notebook.
X_train_grade_ohe = vectorizer.transform(X_train['project_grade_category'].values)
X_test_grade_ohe = vectorizer.transform(X_test['project_grade_category'].values)

print(X_train_grade_ohe.shape)
print(X_test_grade_ohe.shape)
```

```
(40000, 4)
(10000, 4)
```

In [178]:
```
cat_num_columns.extend(vectorizer.get_feature_names())
print(len(cat_num_columns))
```

```
64
```

Encoding clean_categories column

In [179]:
```
# Code took from original Code provided.
vectorizer = CountVectorizer(vocabulary=list(sorted_cat_dict.keys()), lowercase=False, binary=True)
vectorizer.fit(X_train['clean_categories'].values)
print(vectorizer.get_feature_names())
```

```
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'SpecialNeeds', 'Health_Sports',
'Math_Science', 'Literacy_Language']
```

In [180]:
```
# Code took from SAMPLE_SOLUTION notebook.
X_train_categ_ohe = vectorizer.transform(X_train['clean_categories'].values)
X_test_categ_ohe = vectorizer.transform(X_test['clean_categories'].values)

print(X_train_categ_ohe.shape)
print(X_test_categ_ohe.shape)
```

```
(40000, 9)
(10000, 9)
```

In [181]:
```
cat_num_columns.extend(['categ_'+i for i in vectorizer.get_feature_names()])
print(len(cat_num_columns))
```

```
73
```

Encoding clean_subcategories column

In [182]:
```
# Code took from original Code provided.
vectorizer = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()), lowercase=False, binary=True)
vectorizer.fit(X_train['clean_subcategories'].values)
print(vectorizer.get_feature_names())
```

```
['Economics', 'CommunityService', 'FinancialLiteracy', 'ParentInvolvement', 'Extracurricular', 'Civics_Governm
ent', 'ForeignLanguages', 'NutritionEducation', 'Warmth', 'Care_Hunger', 'SocialSciences', 'CharacterEducatio
n', 'PerformingArts', 'TeamSports', 'Other', 'College_CareerPrep', 'Music', 'History_Geography', 'Health_LifeS
cience', 'EarlyDevelopment', 'Gym_Fitness', 'ESL', 'EnvironmentalScience', 'VisualArts', 'Health_Wellness', 'A
ppliedSciences', 'SpecialNeeds', 'Literature_Writing', 'Mathematics', 'Literacy']
```

In [183]:
```
# Code took from SAMPLE_SOLUTION notebook.
X_train_subcat_ohe = vectorizer.transform(X_train['clean_subcategories'].values)
X_test_subcat_ohe = vectorizer.transform(X_test['clean_subcategories'].values)

print(X_train_subcat_ohe.shape)
print(X_test_subcat_ohe.shape)
```

```
(40000, 30)
(10000, 30)
```

In [184]:
```
cat_num_columns.extend(['subcateg_'+i for i in vectorizer.get_feature_names()])
print(len(cat_num_columns))
```

```
103
```

Other numerical columns (Sentiment_analysis, word counts)

Scaling numerical features for better results

In [185]:
```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaler.fit(X_train_tnppp_norm)

X_train_tnppp_norm = scaler.transform(X_train_tnppp_norm)
X_test_tnppp_norm = scaler.transform(X_test_tnppp_norm)
```

In [186]:
```
scaler = MinMaxScaler()
scaler.fit(X_train_price_norm)

X_train_price_norm = scaler.transform(X_train_price_norm)
X_test_price_norm = scaler.transform(X_test_price_norm)
```

In [187]:
```
scaler = MinMaxScaler()
scaler.fit(X_train_quant_norm)

X_train_quant_norm = scaler.transform(X_train_quant_norm)
X_test_quant_norm = scaler.transform(X_test_quant_norm)
```

Combining categorical and numerical data for further use.

In [188]:
```
sentim_cols = ['essay1_neg', 'essay1_nue', 'essay1_pos', 'essay1_comp', 'essay2_neg',\
'essay2_nue', 'essay2_pos', 'essay2_comp', 'essay1_neg', 'essay1_nue',\
'essay1_pos', 'essay1_comp', 'essay2_neg', 'essay2_nue', 'essay2_pos',\
'essay2_comp', 'essay3_neg', 'essay3_nue', 'essay3_pos', 'essay3_comp',\
'essay4_neg', 'essay4_nue', 'essay4_pos', 'essay4_comp', 'essay_word_count', 'title_word_count']
# Task2_train = hstack((cat_num_train, np.array(X_train[sentim_cols]), X_train_essay_svd))
# Task2_test = hstack((cat_num_test, np.array(X_test[sentim_cols]), X_test_essay_svd))
# print(Task2_train.shape, y_train.shape)
# print(Task2_test.shape, y_test.shape)
```

In [192]:
```
from scipy.sparse import hstack
cat_num_train = hstack((X_train_tnppp_norm, X_train_price_norm, X_train_quant_norm,\
                        np.array(X_train['summary_numeric_bool']).reshape(-1, 1), np.array(X_train[sentim_cols])
                        X_train_prefix_ohe, X_train_grade_ohe, X_train_school_ohe, X_train_categ_ohe, X_train_su

cat_num_test = hstack((X_test_tnppp_norm, X_test_price_norm, X_test_quant_norm,\
                        np.array(X_test['summary_numeric_bool']).reshape(-1, 1), np.array(X_test[sentim_cols]),\
                        X_test_prefix_ohe, X_test_grade_ohe, X_test_school_ohe, X_test_categ_ohe, X_test_subcat_
```

In [193]:
```
print(cat_num_train.shape)
print(cat_num_test.shape)
```

```
(40000, 129)
(10000, 129)
```

Joining processed `essay` and `project_title` arrays with categorical and numerical data to form matrix

In [194]:
```
X_train = hstack((cat_num_train, essay_train_vect, title_train_vect)).tocsr()
X_test = hstack((cat_num_test, essay_test_vect, title_test_vect)).tocsr()

print(X_train.shape)
print(X_test.shape)
```

```
(40000, 229)
(10000, 229)
```

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 [195]:
```
# 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 [213]:
```
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_round, verbose_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


clf = XGBoostClassifier(eval_metric = 'auc', num_class = 2, nthread = 4,)
```

In [214]:
```
parameters = {
    'n_estimators': [10, 25, 50, 75],
    'max_depth': [3, 5, 8, 10],
}

clf = GridSearchCV(clf, parameters, return_train_score=True)
clf.fit(X_train, y_train)
```

Out[214]:
```
GridSearchCV(cv='warn', error_score='raise-deprecating',
             estimator=<__main__.XGBoostClassifier object at 0x000001FC16E34518>,
             iid='warn', n_jobs=None,
             param_grid={'max_depth': [3, 5, 8, 10],
                         'n_estimators': [10, 25, 50, 75]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring=None, verbose=0)
```

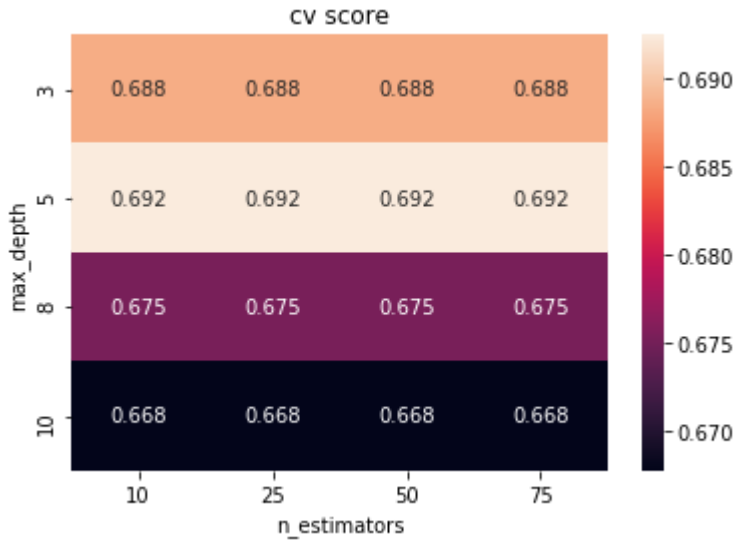
In [219]:
```
clf.cv_results_['params']
```

Out[219]:
```
[{'max_depth': 3, 'n_estimators': 10},
 {'max_depth': 3, 'n_estimators': 25},
 {'max_depth': 3, 'n_estimators': 50},
 {'max_depth': 3, 'n_estimators': 75},
 {'max_depth': 5, 'n_estimators': 10},
 {'max_depth': 5, 'n_estimators': 25},
 {'max_depth': 5, 'n_estimators': 50},
 {'max_depth': 5, 'n_estimators': 75},
 {'max_depth': 8, 'n_estimators': 10},
 {'max_depth': 8, 'n_estimators': 25},
 {'max_depth': 8, 'n_estimators': 50},
 {'max_depth': 8, 'n_estimators': 75},
 {'max_depth': 10, 'n_estimators': 10},
 {'max_depth': 10, 'n_estimators': 25},
 {'max_depth': 10, 'n_estimators': 50},
 {'max_depth': 10, 'n_estimators': 75}]
```

In [215]:
```
clf.cv_results_['mean_test_score'].reshape((4,4))
```

Out[215]:
```
array([[0.68846321, 0.68846321, 0.68846321, 0.68846321],
       [0.69247297, 0.69247297, 0.69247297, 0.69247297],
       [0.67531246, 0.67531246, 0.67531246, 0.67531246],
       [0.66770869, 0.66770869, 0.66770869, 0.66770869]])
```

In [220]:
```
sns.heatmap(clf.cv_results_['mean_test_score'].reshape((4,4)), annot=True, fmt='.3f')
plt.ylabel('max_depth')
plt.xlabel('n_estimators')
plt.title('cv score')
plt.yticks(np.arange(4)+0.5, parameters['max_depth'])
plt.xticks(np.arange(4)+0.5, parameters['n_estimators'])
plt.show()
```



Taking n_estimators=25, max_depth=5 as best hyper-parameters.

Building model with n_estimators=25 and max_depth=5

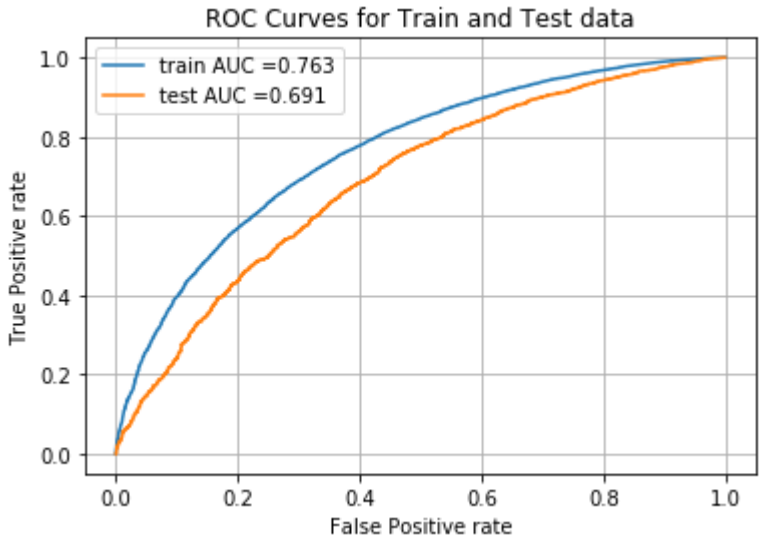
In [221]:
```
best_clf = XGBoostClassifier(eval_metric = 'auc', num_class = 2, nthread = 4, n_estimators=25, max_depth=5)
best_clf.fit(X_train, y_train)
y_pred = best_clf.predict(X_test)
```

In [222]:
```
y_train_pred = best_clf.predict_proba(X_train)[:, 1]
y_test_pred = best_clf.predict_proba(X_test)[:, 1]

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)

result = {}

result['train_auc'], result['test_auc'] = (auc(train_fpr, train_tpr), auc(test_fpr, test_tpr))
result['model'] = best_clf

thr_train = tr_thresholds[np.argmax(train_tpr*(1-train_fpr))]
thr_test = te_thresholds[np.argmax(test_tpr*(1-test_fpr))]

train_predictions = []
for i in y_train_pred:
    if i >= thr_train:
        train_predictions.append(1)
    else:
        train_predictions.append(0)

test_predictions = []
for i in y_test_pred:
    if i >= thr_test:
        test_predictions.append(1)
    else:
        test_predictions.append(0)
```

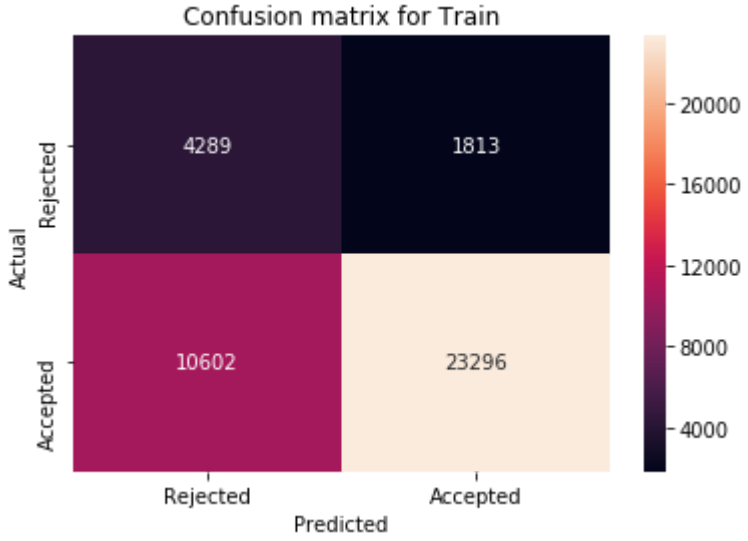
In [223]:
```
from IPython.display import display, Markdown
```

In [224]:
```
display(Markdown(f'**Hyper-Parameters: n_estimators = 25 and max_depth = 5**'))

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(np.round(result['train_auc'], 3)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(np.round(result['test_auc'], 3)))
plt.legend()
plt.xlabel("False Positive rate")
plt.ylabel("True Positive rate")
plt.title("ROC Curves for Train and Test data")
plt.grid()
plt.show()

# Printing confusion matrices code
print(f"\nConfusion matrix for Train data with {thr_train} as threshold:")

ax = sns.heatmap(confusion_matrix(y_train, train_predictions), annot=True, fmt='g')
ax.set_yticklabels(['Rejected', 'Accepted'])
ax.set_xticklabels(['Rejected', 'Accepted'])
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title('Confusion matrix for Train')
plt.show()

print(f"\nConfusion matrix for Test data with {thr_test} as threshold:")

ax = sns.heatmap(confusion_matrix(y_test, test_predictions), annot=True, fmt='g')
ax.set_yticklabels(['Rejected', 'Accepted'])
ax.set_xticklabels(['Rejected', 'Accepted'])
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title('Confusion matrix for Test')
plt.show()
```

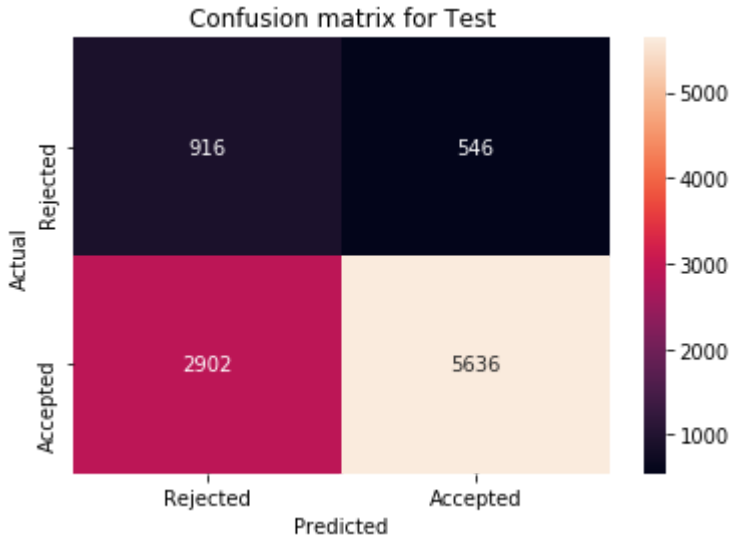
Hyper-Parameters: n_estimators = 25 and max_depth = 5



Confusion matrix for Train data with 0.8339205980300903 as threshold:



Confusion matrix for Test data with 0.8367424011230469 as threshold:



3. Conclusion

In [203]:
```
# Please write down few lines about what you observed from this assignment.
```

Here I plotted ROC curves for only 1 set of hyper-parameters after tuning for 16 sets of hyper-parameters. So printing table for only this model and also included rows for GBDT models in previous notebooks

In [211]:
```
from prettytable import PrettyTable
```

In [230]:
```
table = PrettyTable()
display(Markdown('**GBDT Model Results for this notebook and previous notebooks**'))
table.field_names = ['Vectorizer', 'n_estimators', 'max_depth', 'Train AUC', 'Test AUC']
table.add_row(['TruncatedSVD WordVectors', 25, 5, 0.763, 0.691])
table.add_row(['Bag of Words', 25, 5, 0.76, 0.7])
table.add_row(['Tfidf', 25, 5, 0.78, 0.704])
table.add_row(['Average Word2Vec', 25, 5, 0.798, 0.71])
table.add_row(['Tfidf Weighted Word2Vec', 25, 5, 0.797, 0.707])
print(table)
```

GBDT Model Results for this notebook and previous notebooks

Vectorizer	n_estimators	max_depth	Train AUC	Test AUC
TruncatedSVD WordVectors	25	5	0.763	0.691
Bag of Words	25	5	0.76	0.7
Tfidf	25	5	0.78	0.704
Average Word2Vec	25	5	0.798	0.71
Tfidf Weighted Word2Vec	25	5	0.797	0.707

Only first row of the table is calculated in this notebook. Other rows are manually entered here, just to compare results to previous models

Conclusion:

- **Model which used WordVectors produced from TruncatedSVD did not do good compared to previous GDBT models but the difference is very less.**
- **This suggests that our wordvectors are not bad as we only considered top 2000 words from titles and essays and still got a good test score**
- **If more words from corpus (titles + essays) are considered we might get better models which may give better performance than BOW and other models.**

In [ ]:
```
```