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
```

```
C:\Users\ADMIN\Anaconda3\lib\site-packages\gensim\utils.py:1197: UserWarning: detected Windows; al
iasing 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]:

```
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
	LC652 - Lakeshore Double-Space Mobile Drying		

	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)
```

```
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

In [9]:

```
price_data = resource_data.groupby('id').agg({'price':'sum', 'quantity':'sum'}).reset_index()
price_data.head(2)
project_data = pd.merge(project_data, price_data, on='id', how='left')
```

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**

2. TruncatedSVD

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

In [10]:

```
# 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
sample_data=project_data.sample(50000)
sample_data.shape
from sklearn.model_selection import train_test_split
data=sample_data
data.head()
y=data.project_is_approved
x=data.drop('project_is_approved',axis=1)
x_train,x_test,y_train,y_test=train_test_split(x,y,train_size=0.8,test_size=0.2,stratify=y)
x_train,x_cv,y_train,y_cv=train_test_split(x_train,y_train,train_size=0.75,test_size=0.25,stratify
=y_train)


print("shape of train data ")
print(x_train.shape)
print(y_train.shape)
print("shape of test data ")
print(x_test.shape)
print(y_test.shape)
print("shape of crossvalidation data ")
print(x_cv.shape)
print(y_cv.shape)
```

```
shape of train data
(30000, 19)
(30000,)
shape of test data
(10000, 19)
(10000,)
shape of crossvalidation data
(10000, 19)
(10000,)
```

In [11]:

```
x_train['essay_title'] = x_train['essay'] + x_train['project_title']
x_test['essay_title'] = x_test['essay'] + x_test['project_title']
x_cv['essay_title'] = x_cv['essay'] + x_cv['project_title']
```

In [12]:

```
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]:

```
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', "d
esn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn',
"mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn',
"wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"]
```

In [14]:

```
def preprocessing(x):
    import nltk
    nltk.download('stopwords')
    from tqdm import tqdm
    preprocessed_essays = []
    # tqdm is for printing the status bar
    for sentence in tqdm(x.values):
        sent = decontracted(sentence)
        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())
    return preprocessed_essays
```

In [15]:

```
train_text=[]
test_text=[]
cv_text=[]
train_text=preprocessing(x_train['essay_title'])
test_text=preprocessing(x_test['essay_title'])
cv_text=preprocessing(x_cv['essay_title'])
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\ADMIN\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
100%|████████████████████████████████████████████████████████████████████████████| 30000/30000
[00:24<00:00, 1218.85it/s]
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\ADMIN\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
100%|████████████████████████████████████████████████████████████████████████████| 10000/10000
[00:07<00:00, 1250.54it/s]
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\ADMIN\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
100%|████████████████████████████████████████████████████████████████████████████| 10000/10000
[00:08<00:00, 1234.45it/s]
```

In [16]:

```
def tfidf_text(train_text):
    from sklearn.feature_extraction.text import TfidfVectorizer
    vectorizer = TfidfVectorizer(ngram_range=(1,1),min_df=10,max_features=5000)
    vectorizer.fit(train_text)

    feature_tfidf=np.array(vectorizer.get_feature_names())
    idf_values=vectorizer.idf_
    idf_values = np.argsort(idf_values)[::-1]
    top_idf = idf_values[:2000]
    top2000 = [feature_tfidf[i] for i in top_idf[:]]
    return top2000
top2000=tfidf_text(train_text)
top2000
```

Out[16]:

```
['archery',
 'lacrosse',
 'oils',
 'mallets',
 'pottery',
 'printmaking',
 'dell',
 'golf',
 'calculus',
 'hockey',
 'coaster',
 'drone',
 'watches',
 'easels',
 'aquarium',
 'harry',
 'bot',
 'holocaust',
 'monitors',
 'cricut',
 'signing',
 'potter',
 'ceramics',
 'cutter',
 'pellets',
 'dolls',
 'tournament',
 'flipped',
 'renewable',
 'brainpop',
 'partitions',
 'orff',
 'littlebits',
 'blankets',
```

```
'compost',
'shirt',
'dancers',
'expeditions',
'cube',
'fluorescent',
'disc',
'psychology',
'echo',
'cafe',
'flight',
'watercolors',
'vacuum',
'licenses',
'bots',
'marketing',
'refrigerator',
'mac',
'violin',
'ants',
'whisper',
'bells',
'osmos',
'dvds',
'asd',
'chorus',
'tub',
'sewing',
'diary',
'dojo',
'scales',
'reeds',
'bricks',
'hats',
'wire',
'guitars',
'recorders',
'drawer',
'pots',
'lounge',
'meditation',
'caddies',
'tags',
'dinosaurs',
'hawaiian',
'greenhouse',
'greek',
'birds',
'maze',
'storyworks',
'shakespeare',
'bench',
'shelving',
'fabric',
'mail',
'benches',
'beds',
'frogs',
'portraits',
'stained',
'bowling',
'pollution',
'glass',
'audiobooks',
'gadgets',
'artifacts',
'holders',
'herbs',
'prints',
'chess',
'recorder',
'bones',
'trash',
'bee',
'louis',
'cleaner',
'commands',
```

```
'salt',
'wet',
'champion',
'tops',
'tank',
'leisure',
'rhymes',
'prekindergarten',
'carpets',
'perimeter',
'phase',
'eraser',
'velcro',
'productions',
'ngss',
'bloom',
'actors',
'desert',
'injury',
'bucket',
'dissection',
'duct',
'ipods',
'measurements',
'bear',
'newspapers',
'360',
'dog',
'electives',
'architects',
'tag',
'dna',
'sculptures',
'industrial',
'cat',
'comics',
'fishing',
'nets',
'mass',
'geographic',
'fundraisers',
'flooded',
'mentors',
'package',
'paperless',
'expenses',
'drying',
'elmo',
'tangrams',
'crates',
'classical',
'trampoline',
'recycle',
'arm',
'softball',
'rhythms',
'monster',
'www',
'sanitizer',
'camp',
'roller',
'nex',
'iready',
'doh',
'samsung',
'drafts',
'snow',
'sell',
'zones',
'bulb',
'tubes',
'bill',
'launch',
'wires',
'phoenix',
'documenting',
'owl',
```

```
'recycled',
'disease',
'flex',
'magna',
'animated',
'creatures',
'stomach',
'stimuli',
'biological',
'las',
'bass',
'trackers',
'teeth',
'dark',
'pants',
'election',
'mixing',
'equals',
'phrases',
'ozobot',
'grip',
'leap',
'hero',
'powered',
'expecting',
'architecture',
'reduces',
'wave',
'prices',
'console',
'dirt',
'infused',
'http',
'flashcards',
'battery',
'expressions',
'therapist',
'streets',
'bits',
'drinking',
'makeup',
'pursuit',
'stopped',
'ti',
'connecticut',
'activate',
'relatable',
'index',
'socializing',
'somalia',
'lined',
'goggles',
'ebooks',
'surfaces',
'hooks',
'vehicles',
'wi',
'adjustable',
'sculpture',
'stream',
'zero',
'soaking',
'laser',
'generational',
'lexia',
'stakes',
'worse',
'pta',
'laminated',
'resistance',
'delight',
'volunteers',
'recreate',
'constructive',
'conservation',
'disney',
'company',
```

```
'collar',
'coaching',
'renaissance',
'drawers',
'calculations',
'drumming',
'residents',
'biographies',
'restricted',
'bonds',
'pk',
'maryland',
'looping',
'scaffold',
'paid',
'david',
'retelling',
'partnerships',
'links',
'interview',
'inventive',
'loop',
'imovie',
'messages',
'max',
'jazz',
'joining',
'press',
'vs',
'buzz',
'diversified',
'disturbances',
'lift',
'dances',
'cubbies',
'ranked',
'correlate',
'comprehending',
'reflecting',
'remote',
'blogs',
'edition',
'bills',
'ribbon',
'workshops',
'worthwhile',
'suffering',
'shakers',
'advancing',
'sink',
'adaptations',
'34',
'pizza',
'liked',
'phrase',
'failed',
'ninth',
'freshman',
'firmly',
'textures',
'files',
'fictional',
'nannanorganization',
'nannanone',
'hair',
'entice',
'gate',
'mode',
'mo',
'modify',
'returned',
'shot',
'network',
'starter',
'sc',
'rush',
'rubber',
```

```
'nannankids',
'sustainable',
'noodle',
'massive',
'sustained',
'removed',
'refocus',
'recipes',
'participated',
'quotes',
'pathways',
'nannanbouncing',
'propel',
'promethean',
'tenacity',
'march',
'multiply',
'zoo',
'lie',
'crisis',
'visited',
'aides',
'trends',
'displaced',
'indeed',
'descent',
'illustrators',
'asia',
'inquiring',
'hp',
'consuming',
'ukuleles',
'bathroom',
'beginner',
'hence',
'bi',
'addressing',
'anatomy',
'changer',
'250',
'isolated',
'insure',
'42',
'edu',
'essentially',
'chips',
'climb',
'ukulele',
'facilitating',
'crackers',
'participates',
'exists',
'coins',
'comments',
'consequently',
'party',
'nutritional',
'crew',
'disruptions',
'rebuild',
'optimistic',
'greeting',
'puppet',
'dense',
'performers',
'proactive',
'captivating',
'epic',
'ok',
'frog',
'utmost',
'painted',
'articulation',
'mistake',
'sits',
'insects',
'shut',
```

```
'amazes',
'smoother',
'angles',
'impacting',
'appear',
'joys',
'treats',
'saved',
'maintenance',
'hoop',
'tutorials',
'behaviorally',
'paintings',
'locally',
'attributes',
'headsets',
'pastels',
'skip',
'palsy',
'zest',
'embraces',
'abc',
'48',
'yesterday',
'performed',
'stays',
'93',
'rent',
'dialogue',
'verbally',
'president',
'dictionary',
'surrounds',
'seesaw',
'seasons',
'decent',
'rainbow',
'scholarships',
'attainable',
'basket',
'broke',
'relating',
'cater',
'category',
'requirement',
'venture',
'indianapolis',
'mystery',
'glasses',
'unhealthy',
'negatively',
'needless',
'neatly',
'nannansteam',
'instruct',
'nannanmore',
'kansas',
'mandela',
'nannanextra',
'nannancolor',
'nannanchrome',
'frequency',
'kickball',
'oldest',
'miami',
'ice',
'fitbits',
'overrated',
'bottle',
'replenish',
'uncommon',
'toon',
'suggestions',
'invention',
'sea',
'buttons',
'mexican',
```

'additions',
'nannanbring',
'yearbook',
'clipboard',
'graduated',
'classmate',
'bounds',
'checked',
'makey',
'tirelessly',
'handed',
'adore',
'adorable',
'adequately',
'heartbreaking',
'underfunded',
'fantasy',
'embody',
'tears',
'downloaded',
'poses',
'inventory',
'accustomed',
'disruption',
'pet',
'khan',
'equitable',
'political',
'gardens',
'disappointed',
'plate',
'steady',
'posted',
'digging',
'noises',
'experiential',
'destroyed',
'linguistically',
'vastly',
'opposed',
'450',
'profession',
'organisms',
'adolescents',
'la',
'shooting',
'surpass',
'shortage',
'progresses',
'peak',
'tinker',
'beanbags',
'hilarious',
'fold',
'unsafe',
'satisfy',
'ladies',
'athlete',
'humor',
'sharpening',
'masters',
'masterpiece',
'explained',
'farther',
'beautifully',
'ecosystem',
'surround',
'yearning',
'delay',
'newcomers',
'proves',
'debates',
'sparks',
'culminating',
'cues',
'reaction',
'receptive',

```
'needy',
'thereby',
'solved',
'nurtured',
'yearly',
'predict',
'nannanon',
'mainstream',
'reminder',
'cabinet',
'manners',
'repeat',
'repeated',
'recall',
'ambassadors',
'southeast',
'affordable',
'specials',
'nannanliteracy',
'india',
'oftentimes',
'cry',
'neither',
'de',
'programmers',
'nutrient',
'occasion',
'leaps',
'nannantime',
'eclectic',
'pitch',
'electrical',
'footballs',
'partnership',
'films',
'recordings',
'nannantake',
'impairment',
'bikes',
'string',
'medicine',
'hinders',
'hearted',
'bellies',
'revolution',
'nannanchromebook',
'communicators',
'nannanempowering',
'cerebral',
'facilities',
'refuse',
'habitat',
'combines',
'immigrated',
'00',
'views',
'thrown',
'vegetable',
'tries',
'toss',
'wasting',
'suit',
'wisconsin',
'applies',
'northeast',
'understandings',
'failing',
'nannando',
'wealthy',
'utah',
'ese',
'qualified',
'expo',
'contemporary',
'inquire',
'forgotten',
'fed',
```

```
'led',
'feeding',
'ticket',
'tricky',
'momentum',
'formats',
'reactions',
'behaved',
'craving',
'river',
'simulations',
'ring',
'demonstrated',
'84',
'borrowed',
'skin',
'cones',
'toy',
'nannanstand',
'strings',
'dissect',
'puerto',
'sequence',
'trained',
'lectures',
'conferences',
'pulling',
'pursuing',
'hop',
'dvd',
'thru',
'regards',
'email',
'traumatic',
'embraced',
'74',
'plug',
'gonoodle',
'nannanback',
'nevada',
'grocery',
'northwest',
'recorded',
'autonomy',
'loose',
'george',
'align',
'dictionaries',
'vivid',
'tapping',
'downs',
'soap',
'eggs',
'describes',
'sketch',
'plates',
'demonstrations',
'worms',
'establishing',
'exact',
'exchange',
'pennsylvania',
'optimize',
'uniform',
'remainder',
'slightly',
'christmas',
'responsive',
'scan',
'trend',
'scheduled',
'intrinsic',
'suitable',
'candy',
'instilling',
'transformed',
'breathing',
'sending',
```

'sending',
'cloud',
'treated',
'lists',
'historic',
'visits',
'tip',
'agreed',
'follows',
'helpers',
'enjoys',
'firsties',
'compliment',
'pain',
'accessibility',
'92',
'workstations',
'pbl',
'million',
'stimulated',
'approved',
'user',
'voracious',
'mindful',
'mirror',
'vocal',
'decode',
'earbuds',
'observed',
'contributes',
'designers',
'blast',
'understands',
'preferences',
'nannanmrs',
'counseling',
'sharp',
'divided',
'educationally',
'disposal',
'freshmen',
'3doodler',
'disciplines',
'provoking',
'blog',
'traditionally',
'demanding',
'kinetic',
'lock',
'piano',
'intended',
'desktops',
'arise',
'lake',
'inventions',
'empathetic',
'invited',
'journalism',
'watercolor',
'scene',
'dallas',
'hydrated',
'weighted',
'reuse',
'ties',
'honestly',
'samples',
'reaches',
'civil',
'worthy',
'gateway',
'guarantee',
'upgrade',
'nc',
'studied',
'harness',
'guess',
'facility'

```
'facility',
'tennis',
'nannanfuture',
'desires',
'yard',
'net',
'noisy',
'driving',
'nannanno',
'association',
'conscious',
'nights',
'ecosystems',
'emergent',
'somewhat',
'excuses',
'shake',
'initial',
'pedometers',
'dated',
'deployed',
'theatre',
'scope',
'headphone',
'modifications',
'accomplishing',
'impress',
'efficiency',
'bluetooth',
'returning',
'damage',
'boogie',
'rope',
'counters',
'similarities',
'bike',
'pilot',
'laughing',
'houston',
'followed',
'mornings',
'therapeutic',
'guardians',
'marching',
'latin',
'timely',
'narratives',
'specially',
'mandarin',
'police',
'assure',
'vietnam',
'linguistic',
'lovable',
'adventurous',
'combat',
'hole',
'holes',
'honest',
'survival',
'clip',
'remains',
'devoted',
'monday',
'myriad',
'sales',
'mood',
'beliefs',
'instance',
'travelers',
'tolerance',
'brainstormed',
'league',
'lies',
'ses',
'nelson',
'experimentation',
'grouping'
```

```
'grouping',
'quicker',
'expanded',
'explored',
'parachute',
'unstable',
'baltimore',
'entry',
'planting',
'awarded',
'forming',
'francisco',
'eleven',
'uniqueness',
'basically',
'spin',
'attach',
'spirits',
'nonverbal',
'dividers',
'diego',
'male',
'length',
'presence',
'playful',
'limiting',
'error',
'teen',
'microsoft',
'moreover',
'quiz',
'nannancoding',
'guides',
'diagnosis',
'globally',
'observation',
'coats',
'obvious',
'counts',
'injuries',
'oregon',
'mountain',
'imagined',
'chalk',
'chain',
'grandparent',
'concerts',
'prevents',
'worried',
'smell',
'assortment',
'apartment',
'55',
'ordinary',
'believer',
'nannanplease',
'dress',
'holiday',
'dollar',
'nannanbook',
'citizen',
'suited',
'awe',
'inability',
'produced',
'neighbors',
'raz',
'rare',
'closet',
'analytical',
'flowers',
'fitbit',
'symbols',
'sick',
'upset',
'constructing',
'180',
'explanations',
```

```
 'explorations',
 'totally',
 'discouraged',
 'cooperate',
 'mississippi',
 'differing',
 'beloved',
 'attempting',
 'chemical',
 'relatives',
 'impressed',
 'personalize',
 'calculate',
 'homeroom',
 'lean',
 'embarrassed',
 'assisting',
 'dc',
 'weapon',
 '1000',
 'conflict',
 'shoot',
 'reply',
 'laboratory',
 'emphasizes',
 'kidney',
 'covering',
 'tied',
 'sedentary',
 'vietnamese',
 'takers',
 'crowded',
 'confined',
 'sticky',
 'modeled',
 'montessori',
 'nannanmusic',
 'delayed',
 'controlled',
 'conclusions',
 'selecting',
 'selections',
 'potentially',
 ...]
```

2.2 Computing Co-occurance matrix

In [17]:

```
# 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

#https://stackoverflow.com/questions/41661801/python-calculate-the-co-occurrence-matrix

cooc=np.zeros([2000,2000])
for sentence in train_text:
    sent=sentence.split()
    for i,word in enumerate(sent):
        if word in top2000:
            for j in range(max(i-5,0),min(i+5,len(sent))):
                if sent[j] in top2000 and sent[j]!=word:
                    cooc[top2000.index(word),top2000.index(sent[j])]+=1
cooc
```

Out[17]:

```
array([[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.]])
```

In [ ]:

```
```

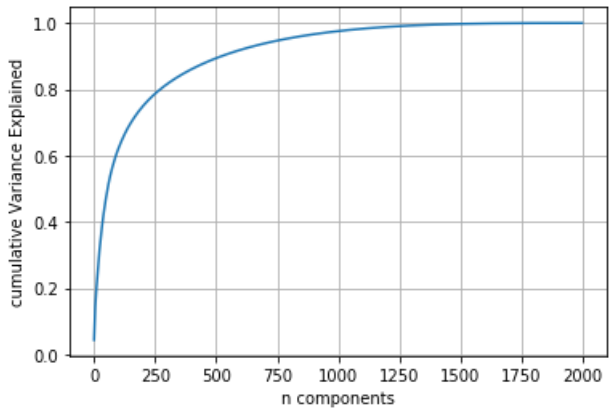
2.3 Applying TruncatedSVD and Calculating Vectors for `essay` and `project_title`

In [18]:

```
# 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
from sklearn.decomposition import TruncatedSVD
svd = TruncatedSVD(n_components=1999,  random_state=42)
svd.fit(cooc)
cumvar = np.cumsum( svd.explained_variance_ratio_ )
```

In [19]:

```
plt.plot( cumvar )
plt.grid()
plt.xlabel('n components')
plt.ylabel('cumulative Variance Explained')
plt.show()
```



In [20]:

```
n=600
svd = TruncatedSVD(n_components=600,  random_state=42)
svd.fit(cooc)
finalcooc=svd.transform(cooc)
finalcooc.shape
```

Out[20]:

```
(2000, 600)
```

In [21]:

```
model=dict()
j=0
for i in finalcooc:
    model[top2000[j]]=i
    j=j+1
```

In [22]:

```
glove_words =  set(model.keys())
```

In [23]:

```
def avgword2vec(text):


    avg_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
    for sentence in tqdm(text):# for each review/sentence
        vector = np.zeros(600) # 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)
    return avg_w2v_vectors
```

2.4 Merge the features from step 3 and step 4

In [24]:

```
def totalWords( col,x ):

    words = []
    for sent in tqdm( x[col].values ) :

        words.append(len(sent.split()))

    return words
```

In [25]:

```
train_essay=totalWords('essay',x_train)
test_essay=totalWords('essay',x_test)
cv_essay=totalWords('essay',x_cv)
train_title=totalWords('project_title',x_train)
test_title=totalWords('project_title',x_test)
cv_title=totalWords('project_title',x_cv)
```

```
100%|██████████| 30000/30000
[00:00<00:00, 43778.01it/s]
100%|██████████| 10000/10000
[00:00<00:00, 40618.98it/s]
100%|██████████| 10000/10000
[00:00<00:00, 41270.37it/s]
100%|██████████| 30000/30000
[00:00<00:00, 435029.09it/s]
100%|██████████| 10000/10000
[00:00<00:00, 625427.43it/s]
100%|██████████| 10000/10000
[00:00<00:00, 435035.11it/s]
```

In [26]:

```
x_train['essay_words']=train_essay
x_test['essay_words']=test_essay
```

```
x_cv['essay_words']=cv_essay
x_train['title_words']=train_title
x_test['title_words']=test_title
x_cv['title_words']=cv_title
```

In [27]:

```
from textblob import TextBlob
def senti(x):
    return TextBlob(x).sentiment.polarity

x_train['senti_score'] = x_train['essay'].apply(senti)
x_test['senti_score'] = x_test['essay'].apply(senti)
x_cv['senti_score'] = x_cv['essay'].apply(senti)

x_train['senti_score']
```

Out[27]:

```
58872     0.316850
71399     0.147348
19182     0.195877
77811     0.047059
7655      0.139320
45970     0.197826
817       0.228947
40919     0.251244
84750     0.161063
68606     0.224459
3673      0.245242
86797     0.094657
43076     0.067717
295       0.209801
93972     0.123578
12093     0.325000
8045      0.214731
96441     0.330768
98856     0.223387
108735    0.205914
72301     0.095867
13230     0.173810
22959     0.187515
12752     0.127062
42430     0.288322
18858     0.175000
51181     0.277092
95069     0.205795
36109     0.191829
39264     0.228587
            ...
99387     0.117932
64399     0.261538
32383     0.199476
8446      0.211042
46496     0.284909
16140     0.141111
53035     0.318290
17784     0.299198
70729     0.248810
23090     0.222850
107104    0.217857
8322      0.240298
72453     0.177296
35253     0.247165
89408     0.210591
24275     0.222212
48157     0.208648
86913     0.326732
56710     0.159820
64120     0.254673
76014     0.219113
83607     0.417165
16493     0.370657
100907    0.255441
13645     0.188297
97834     0.209722
```

```
2231      0.313112
32455     0.243700
7303      0.269507
18262     0.354327
Name: senti_score, Length: 30000, dtype: float64
```

In [28]:

```
# 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
def veccat(x):
    from sklearn.feature_extraction.text import CountVectorizer
    from sklearn.preprocessing import Normalizer
    from scipy.sparse import hstack
    vectorizer = CountVectorizer(vocabulary=list(sorted_cat_dict.keys()), lowercase=False, binary=T
rue)
    vectorizer.fit(x['clean_categories'].values)
    print(vectorizer.get_feature_names())



    categories_one_hot = vectorizer.transform(x['clean_categories'].values)
    print("Shape of matrix after one hot encodig ",categories_one_hot.shape)

    vectorizer = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()), lowercase=False,
binary=True)
    vectorizer.fit(x['clean_subcategories'].values)
    print(vectorizer.get_feature_names())



    subcategories_one_hot = vectorizer.transform(x['clean_subcategories'].values)
    print("Shape of matrix after one hot encodig ",subcategories_one_hot.shape)


    vectorizer = CountVectorizer(lowercase=False, binary=True)
    vectorizer.fit(x['school_state'].values)
    print(vectorizer.get_feature_names())



    state_one_hot = vectorizer.transform(x['school_state'].values)
    print("Shape of matrix after one hot encodig ",state_one_hot.shape)

    x = x.replace(np.nan, '', regex=True)
    vectorizer = CountVectorizer( vocabulary=['Mrs.', 'Ms.','Mr.','Teacher','Dr.'],lowercase=False)
    vectorizer.fit(x['teacher_prefix'].values)
    print(vectorizer.get_feature_names())



    prefix_one_hot = vectorizer.transform(x['teacher_prefix'].values)
    print("Shape of matrix after one hot encodig ",prefix_one_hot.shape)

    vocab=[]
    for i in x['project_grade_category'].values:
        vocab.append(i)
    v_set=set(vocab)
    vocab=list(v_set)
    vectorizer = CountVectorizer(vocabulary=vocab,lowercase=False)
    vectorizer.fit(x['project_grade_category'].values)




    grade_one_hot = vectorizer.transform(x['project_grade_category'].values)
    print("Shape of matrix after one hot encodig ",grade_one_hot.shape)

    price_scalar =  Normalizer(copy=False,norm='l2')
```

```
    price_scalar.fit(x['price'].values.reshape(1,-1)) # finding the mean and standard deviation of
this data



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

    projects_scalar = Normalizer(copy=False,norm='l2')
    projects_scalar.fit(x['teacher_number_of_previously_posted_projects'].values.reshape(1,-1)) # f
inding the mean and standard deviation of this data

    # Now standardize the data with above maen and variance.
    projects_standardized =
projects_scalar.transform(x['teacher_number_of_previously_posted_projects'].values.reshape(1, -1))
    projects_standardized =np.transpose(projects_standardized)

    qty_scalar= Normalizer(copy=False,norm='l2')
    qty_scalar.fit(x['quantity'].values.reshape(1,-1)) # finding the mean and standard deviation of
this data


    # Now standardize the data with above maen and variance.
    qty_standardized = qty_scalar.transform(x['quantity'].values.reshape(1, -1))
    qty_standardized=np.transpose(qty_standardized)

    essay_words_scalar= Normalizer(copy=False,norm='l2')
    essay_words_scalar.fit(x['essay_words'].values.reshape(1,-1))

    essay_words_standardized = essay_words_scalar.transform(x['essay_words'].values.reshape(1, -1))
    essay_words_standardized=np.transpose(essay_words_standardized)

    title_words_scalar= Normalizer(copy=False,norm='l2')
    title_words_scalar.fit(x['title_words'].values.reshape(1,-1))

    title_words_standardized = title_words_scalar.transform(x['title_words'].values.reshape(1, -1))
    title_words_standardized=np.transpose(title_words_standardized)

    senti_scalar= Normalizer(copy=False,norm='l2')
    senti_scalar.fit(x['senti_score'].values.reshape(1,-1))

    senti_standardized = senti_scalar.transform(x['senti_score'].values.reshape(1, -1))
    senti_standardized=np.transpose(senti_standardized)

    X1 = hstack((categories_one_hot, subcategories_one_hot,state_one_hot,prefix_one_hot,grade_one_h
ot,
price_standardized,projects_standardized,qty_standardized,essay_words_standardized,title_words_stan
dardized,senti_standardized))
    print(X1.shape)
    return(X1)
```

In [29]:

```
x=veccat(x_train)
t=veccat(x_test)
cv=veccat(x_cv)
```

```
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'SpecialNeeds',
'Health_Sports', 'Math_Science', 'Literacy_Language']
Shape of matrix after one hot encodig  (30000, 9)
['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  (30000, 30)
['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']
Shape of matrix after one hot encodig  (30000, 51)
['Mrs.', 'Ms.', 'Mr.', 'Teacher', 'Dr.']
Shape of matrix after one hot encodig  (30000, 5)
```

```
Shape of matrix after one hot encodig  (30000, 4)
(30000, 105)
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'SpecialNeeds',
'Health_Sports', 'Math_Science', 'Literacy_Language']
Shape of matrix after one hot encodig  (10000, 9)
['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  (10000, 30)
['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']
Shape of matrix after one hot encodig  (10000, 51)
['Mrs.', 'Ms.', 'Mr.', 'Teacher', 'Dr.']
Shape of matrix after one hot encodig  (10000, 5)
Shape of matrix after one hot encodig  (10000, 4)
(10000, 105)
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'SpecialNeeds',
'Health_Sports', 'Math_Science', 'Literacy_Language']
Shape of matrix after one hot encodig  (10000, 9)
['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  (10000, 30)
['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']
Shape of matrix after one hot encodig  (10000, 51)
['Mrs.', 'Ms.', 'Mr.', 'Teacher', 'Dr.']
Shape of matrix after one hot encodig  (10000, 5)
Shape of matrix after one hot encodig  (10000, 4)
(10000, 105)
```

Generating final Data Matrix

In [30]:

```
from scipy.sparse import hstack
def w2v(train_essay, test_essay, cv_essay,train_title,test_title,cv_title,x,t,cv):

    text_w2v=avgword2vec(train_essay)
    text_w2v1 = avgword2vec(test_essay)
    text_w2v2 = avgword2vec(cv_essay)

    title_w2v=avgword2vec(train_title)
    title_w2v1 = avgword2vec(test_title)
    title_w2v2 = avgword2vec(cv_title)
    x1 = hstack((x,text_w2v,title_w2v )).tocsr()
    t1= hstack((t,text_w2v1,title_w2v1 )).tocsr()
    cv1 = hstack((cv,text_w2v2,title_w2v2 )).tocsr()
    return x1,t1,cv1
```

In [31]:

```
x1,t1,cv1=w2v(x_train['essay'],x_test['essay'],x_cv['essay'],x_train['project_title'],x_test['proj
ect_title'],x_cv['project_title'],x,t,cv)
print(x1.shape)
print(t1.shape)
print(cv1.shape)
```

```
100%|██████████████████████████████████████████████████████████████████████| 30000/30000
[00:02<00:00, 12701.88it/s]
100%|██████████████████████████████████████████████████████████████████████| 10000/10000
[00:00<00:00, 12782.89it/s]
100%|██████████████████████████████████████████████████████████████████████| 10000/10000
```

```
[00:00<00:00, 12523.71it/s]
100%|████████████████████████████████████████████████████████████| 30000/30000
[00:00<00:00, 130896.24it/s]
100%|████████████████████████████████████████████████████████████| 10000/10000
[00:00<00:00, 131664.08it/s]
100%|████████████████████████████████████████████████████████████| 10000/10000
[00:00<00:00, 125075.04it/s]
```

```
(30000, 1305)
(10000, 1305)
(10000, 1305)
```

2.5 Apply XGBoost on the Final Features from the above section

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

In [32]:

```
def batch_predict(clf, data):
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive class
    # not the predicted outputs

    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000
    # consider you X_tr shape is 49041, then your cr_loop will be 49041 - 49041%1000 = 49000
    # in this for loop we will iterate unti the last 1000 multiplier
    for i in range(0, tr_loop, 1000):
        y_data_pred.extend(clf.predict_proba(data[i:i+1000])[:,1])
    # we will be predicting for the last data points
    #print(len(y_data_pred))
    #print(data.shape[0])
    if(len(y_data_pred)!=data.shape[0]):

        y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])


    return y_data_pred
```

Finding the Hyperparameters

Crossvalidating using Gridsearch

In [33]:

```
def grid_search(x_train,y_train,x_cv,y_cv):
    import xgboost as xgb
    from sklearn.model_selection import GridSearchCV
    from sklearn.model_selection import RandomizedSearchCV
    from sklearn.metrics import roc_curve
    from sklearn.metrics import roc_auc_score
    from sklearn import preprocessing

    gbdt = xgb.XGBClassifier(n_jobs=-1,class_weight='balanced')
    parameters = {'learning_rate': [0.01,0.1,1], 'max_depth':[5,10, 50]}
    # Fitted the model on train data
    clf = GridSearchCV(gbdt, parameters, cv=2,verbose=5, scoring='roc_auc')
    clf.fit(x_train, y_train)
    train_auc= clf.cv_results_['mean_train_score']
    cv_auc = clf.cv_results_['mean_test_score']
    max_score= clf.best_score_
    print("Train AUC:",train_auc)
    print("CV AUC:",cv_auc)
    print("Max_Score",max_score)


    # Found out the score for crossvalidated data
    print("Accuracy on crossvalidated data: " , clf.score(x_cv,y_cv))
    return train_auc,cv_auc
```

In [34]:

```
train_auc,cv_auc=grid_search(x1,y_train,cv1,y_cv)
```

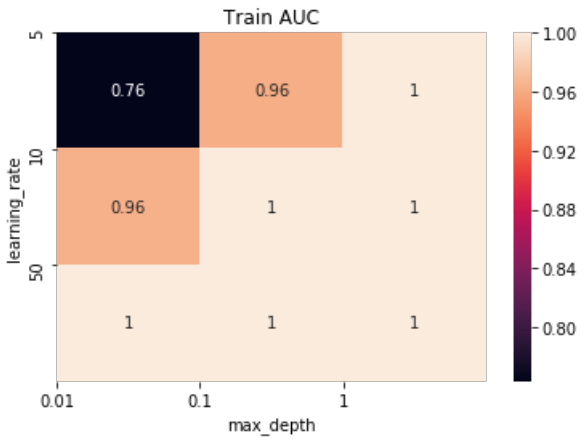
```
Fitting 2 folds for each of 9 candidates, totalling 18 fits

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

[CV] learning_rate=0.01, max_depth=5 .................................
[CV]  learning_rate=0.01, max_depth=5, score=0.6744768941887305, total= 2.1min

[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:  2.2min remaining:    0.0s

[CV] learning_rate=0.01, max_depth=5 .................................
[CV]  learning_rate=0.01, max_depth=5, score=0.6814371795717451, total= 2.1min

[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:  4.3min remaining:    0.0s

[CV] learning_rate=0.01, max_depth=10 ................................
[CV]  learning_rate=0.01, max_depth=10, score=0.6645263635764298, total= 4.2min

[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:  8.5min remaining:    0.0s

[CV] learning_rate=0.01, max_depth=10 ................................
[CV]  learning_rate=0.01, max_depth=10, score=0.6637216175028632, total= 4.1min

[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed: 12.7min remaining:    0.0s

[CV] learning_rate=0.01, max_depth=50 ................................
[CV]  learning_rate=0.01, max_depth=50, score=0.6327945626714153, total= 9.4min
[CV] learning_rate=0.01, max_depth=50 ................................
[CV]  learning_rate=0.01, max_depth=50, score=0.6323229209351986, total= 9.5min
[CV] learning_rate=0.1, max_depth=5 ..................................
[CV]  learning_rate=0.1, max_depth=5, score=0.6714300451550778, total= 2.1min
[CV] learning_rate=0.1, max_depth=5 ..................................
[CV]  learning_rate=0.1, max_depth=5, score=0.6771382418188585, total= 2.2min
[CV] learning_rate=0.1, max_depth=10 .................................
[CV]  learning_rate=0.1, max_depth=10, score=0.6574132859567674, total= 4.0min
[CV] learning_rate=0.1, max_depth=10 .................................
[CV]  learning_rate=0.1, max_depth=10, score=0.6593890498423256, total= 4.0min
[CV] learning_rate=0.1, max_depth=50 .................................
[CV]  learning_rate=0.1, max_depth=50, score=0.6591767217188101, total= 5.8min
[CV] learning_rate=0.1, max_depth=50 .................................
[CV]  learning_rate=0.1, max_depth=50, score=0.6595276194160768, total= 5.9min
[CV] learning_rate=1, max_depth=5 ....................................
[CV]  learning_rate=1, max_depth=5, score=0.6031021186575289, total= 2.1min
[CV] learning_rate=1, max_depth=5 ....................................
[CV]  learning_rate=1, max_depth=5, score=0.6003825572878984, total= 2.1min
[CV] learning_rate=1, max_depth=10 ...................................
[CV]  learning_rate=1, max_depth=10, score=0.6116708745137007, total= 2.9min
[CV] learning_rate=1, max_depth=10 ...................................
[CV]  learning_rate=1, max_depth=10, score=0.6169354268178691, total= 2.9min
[CV] learning_rate=1, max_depth=50 ...................................
[CV]  learning_rate=1, max_depth=50, score=0.6157822786729961, total= 3.1min
[CV] learning_rate=1, max_depth=50 ...................................
[CV]  learning_rate=1, max_depth=50, score=0.6169118790471663, total= 3.1min

[Parallel(n_jobs=1)]: Done  18 out of  18 | elapsed: 72.4min finished

Train AUC: [0.76339225 0.96211591 0.99931027 0.96428653 0.99970943 1.
 0.99984841 1.         1.        ]
CV AUC: [0.6779568  0.66412402 0.63255876 0.67428395 0.6584011  0.65935216
 0.60174243 0.61430298 0.61634704]
Max_Score 0.6779568048707251
Accuracy on crossvalidated data:  0.6641664035518843
```

In [35]:

```
train_auc = train_auc.reshape(3,3)
cv_auc = cv_auc.reshape(3,3)
```

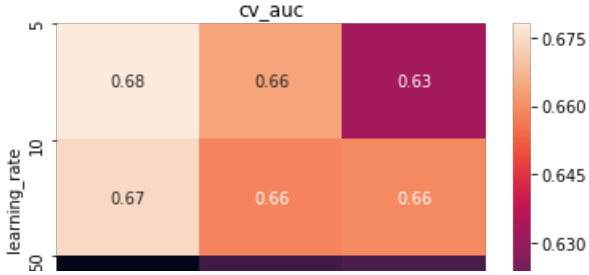
In [36]:

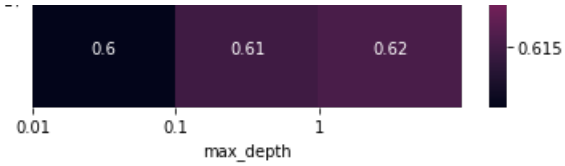
```
import matplotlib.pyplot as plt
# plt.show()

import numpy as np; np.random.seed(0)
import seaborn as sns


sns.heatmap(train_auc,annot=True)

plt.yticks(np.arange(3), [5,10,50])
plt.xticks(np.arange(3), [0.01,0.1,1])

plt.xlabel('max_depth')
plt.ylabel('learning_rate')
plt.title('Train AUC')

plt.show()
```



In [37]:

```
import matplotlib.pyplot as plt
# plt.show()

import numpy as np; np.random.seed(0)
import seaborn as sns


sns.heatmap(cv_auc,annot=True)

plt.yticks(np.arange(3), [5,10,50])
plt.xticks(np.arange(3), [0.01,0.1,1])

plt.xlabel('max_depth')
plt.ylabel('learning_rate')
plt.title('cv_auc')


plt.show()
```





from both train and cv AUC we find that when learning rate=0.01 and depth =10 the AUC is highest. So we take those values for our hyperparameters.

In [38]:

```
def predict1(x_train,y_train,x_test,y_test,depth,rate):

    clf = xgb.XGBClassifier( objective = "binary:logistic", random_state = 42, max_depth = depth, l
earning_rate = rate, class_weight='balanced' )
    clf.fit(x_train,y_train)
    predictions=clf.predict(x_test)
    return predictions
```

Confusion Matrix for Train and Test Data

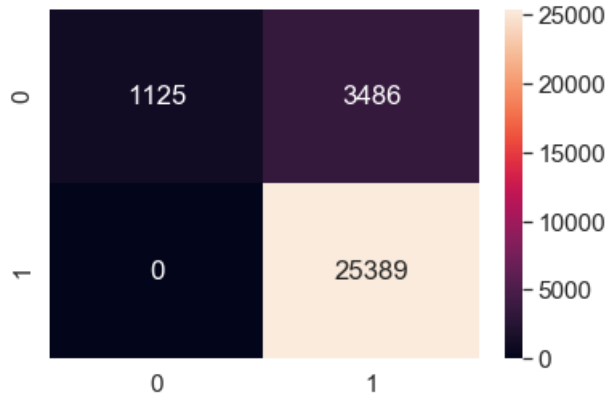
In [39]:

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
import xgboost as xgb
print("Train confusion matrix")
labels=[0,1]
cm=confusion_matrix(y_train, predict1(x1, y_train, x1, y_train,10,0.01),labels)
sns.set(font_scale=1.4)
sns.heatmap(cm,fmt='d',annot=True)
```

Train confusion matrix

Out[39]:

<matplotlib.axes._subplots.AxesSubplot at 0x168825da630>



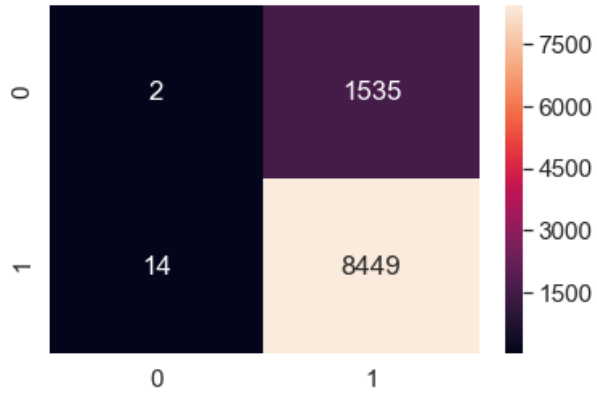
In [40]:

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score


print("Test confusion matrix")
labels=[0,1]
cm=confusion_matrix(y_test, predict1(x1, y_train, t1, y_test,10,0.01),labels)
sns.set(font_scale=1.4)
sns.heatmap(cm,fmt='d',annot=True)
```

Test confusion matrix

Out[40]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x16882fbfda0>
```



3. Conclusion

In [42]:

```
# Please write down few lines about what you observed from this assignment.
from prettytable import PrettyTable
x=PrettyTable()
x.field_names=["Vectorizer","Model","Features","Depth","Learning Rate","Train AUC","CV AUC"]
x.add_row(["AVG W2V","Truncated SVD","1305","10","0.01","0.96","0.67"])
print(x)
```

```
+------------+---------------+----------+-------+---------------+-----------+--------+
| Vectorizer |     Model     | Features | Depth | Learning Rate | Train AUC | CV AUC |
+------------+---------------+----------+-------+---------------+-----------+--------+
|  AVG W2V   | Truncated SVD |   1305   |   10  |      0.01     |    0.96   |  0.67  |
+------------+---------------+----------+-------+---------------+-----------+--------+
```

In [ ]: