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.

11th Assinment Truncated SVD

In [1]:

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

import sqlite3
```

```
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
print('done')

!pip install -U -q PyDrive
```

```
done
     |████████████████████████████████| 993kB 2.8MB/s
  Building wheel for PyDrive (setup.py) ... done
```

In [3]:

```
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials
# Authenticate and create the PyDrive client.
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)


# links to google drive
link='https://drive.google.com/open?id=18VAiuw3vfETGcuJOdicvkgQT0pSxF7Wy'
link3='https://drive.google.com/open?id=1Z6bjXmyCaoEzXYo_tRDwLTsfeA2F3K3j'
flufff, id2 = link3.split('=')
print (id2) # Verify that you have everything after '='
downloaded = drive.CreateFile({'id':id2})
downloaded.GetContentFile('glove_vectors')
```

```
1Z6bjXmyCaoEzXYo_tRDwLTsfeA2F3K3j
```

1.1 Reading Data

In [4]:

```
fluff, id = link.split('=')
print (id) # Verify that you have everything after '='
```

```
# for project data
downloaded = drive.CreateFile({'id':id})
downloaded.GetContentFile('train_data.csv')
project_data = pd.read_csv('train_data.csv',nrows=40000)

print(project_data.shape)



#-----------------------------------------------------------------------------------------------------


link1='https://drive.google.com/open?id=11uHEj9KOgWD9SU-CPgKyb6VrWqVos4uV'
print('\n-----------------------------------------------------------------')

# for resource data
fluff1, idi = link1.split('=')
print (idi) # Verify that you have everything after '='
downloaded = drive.CreateFile({'id':idi})
downloaded.GetContentFile('resources .csv')
resource_data = pd.read_csv('resources .csv')


print(resource_data .head(3))
```

```
18VAiuw3vfETGcuJOdicvkgQT0pSxF7Wy
(40000, 17)

-----------------------------------------------------------------
11uHEj9KOgWD9SU-CPgKyb6VrWqVos4uV
        id                                          description  quantity   price
0  p233245  LC652 - Lakeshore Double-Space Mobile Drying Rack         1  149.00
1  p069063         Bouncy Bands for Desks (Blue support pipes)         3   14.95
2  p069063  Cory Stories: A Kid's Book About Living With Adhd         1    8.45
```

In [5]:

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

print(resource_data.shape)
print(resource_data.columns.values)
```

```
Number of data points in train data (40000, 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']
(1541272, 4)
['id' 'description' 'quantity' 'price']
```

In [6]:

```
#sort the datapoints by date  <-

# how to replace elements in list python: https://stackoverflow.com/a/2582163/4084039
cols = ['Date' if x=='project_submitted_datetime' else x for x in list(project_data.columns)]


#sort dataframe based on time pandas python: https://stackoverflow.com/a/49702492/4084039
project_data['Date'] = pd.to_datetime(project_data['project_submitted_datetime'])
project_data.drop('project_submitted_datetime', axis=1, inplace=True)# we drop the col
project_data.sort_values(by=['Date'], inplace=True)# sort the values y date


# how to reorder columns pandas python: https://stackoverflow.com/a/13148611/4084039
project_data = project_data[cols]
```

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

Out[6]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_category	project_s
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	GA	2016-04-27 00:53:00	Grades PreK-2	
29891	146723	p099708	c0a28c79fe8ad5810da49de47b3fb491	Mrs.	CA	2016-04-27 01:10:09	Grades 3-5	Math &

1.3 Text preprocessing

In [0]:

```
# merge two column text dataframe:
project_data["essay_title_combined"] = 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) + \
                        project_data["project_title"].map(str)
```

In [0]:

```
# 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	Date	project_grade_category	project_s
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	GA	2016-04-27 00:53:00	Grades PreK-2	
29891	146723	p099708	c0a28c79fe8ad5810da49de47b3fb491	Mrs.	CA	2016-04-27 01:10:09	Grades 3-5	Math &

In [0]:

```
# 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 [0]:

```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
stopwords= ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', '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', "doesn'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"]
```

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

Preprocessing of project_subject_categories

In [0]:

```
categories = 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 categories:
    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]))
```

Preprocessing of project_subject_subcategories

In [0]:

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

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

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

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

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

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

Preprocessing of project_grade_category

In [14]:

```
print(project_data['project_grade_category'][:20])#   we have to remove the graddes from every row
```

```
473       Grades PreK-2
29891        Grades 3-5
23374     Grades PreK-2
7176      Grades PreK-2
35006        Grades 3-5
5145         Grades 3-5
36468     Grades PreK-2
36358     Grades PreK-2
39438     Grades PreK-2
2521      Grades PreK-2
25460        Grades 6-8
34399        Grades 3-5
5364         Grades 6-8
29183        Grades 3-5
33043        Grades 3-5
37160        Grades 6-8
27157       Grades 9-12
38830        Grades 3-5
10985     Grades PreK-2
15560     Grades PreK-2
Name: project_grade_category, dtype: object
```

In [0]:

```
d= list(project_data['project_grade_category'].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

grade_cat_list = []
for i in d:
    # consider we have text like this:
    for j in i.split(' '): #    # split by spae
        j=j.replace('Grades','')# clean grades from the row
    grade_cat_list.append(j.strip())



project_data['clean_grade'] = grade_cat_list
project_data.drop(['project_grade_category'], 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_grade'].values:
    my_counter.update(word.split())

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

2. Preparing our data for the models

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

In [0]:

```
#Splitting Data into train and Test sklearn https://scikit-
learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(project_data,
                                                    project_data['project_is_approved'],
                                                      stratify=  project_data['project_is_approved'],
                                                      test_size = 0.33
                                                     )
```

In [17]:

```
print(y_train.value_counts())
print(y_test.value_counts())
# huge imbalance
print(X_train.head(1))
```

```
1    22663
0     4137
Name: project_is_approved, dtype: int64
1    11163
0     2037
Name: project_is_approved, dtype: int64
      Unnamed: 0       id  ... clean_subcategories clean_grade
3013      130430  p129765  ...  Literature_Writing      PreK-2

[1 rows x 19 columns]
```

In [0]:

```
#droping the y labels
#https://stackoverflow.com/questions/13411544/delete-column-from-pandas-dataframe-by-column-name
#x_train =
X_train.drop(["project_is_approved"], axis = 1, inplace = True)
#x_test =
X_test.drop(["project_is_approved"], axis = 1, inplace = True)
```

Text preprocessing of train and test

In [19]:

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

```
100%|██████████| 26800/26800 [00:14<00:00, 1822.94it/s]
```

In [20]:

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

```
100%|██████████| 13200/13200 [00:07<00:00, 1852.69it/s]
```

In [21]:

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

```
100%|██████████| 26800/26800 [00:00<00:00, 36155.19it/s]
```

In [22]:

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

```
100%|██████████| 13200/13200 [00:00<00:00, 37010.44it/s]
```

In [23]:

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

```
100%|██████████| 26800/26800 [00:14<00:00, 1790.21it/s]
```

In [24]:

```
#Proprocessing for essay_titles_cobined
# Combining all the above stundents
from tqdm import tqdm
preprocessed_essays_titles_test = []
# tqdm is for printing the status bar
for sentance in tqdm(X_test['essay_title_combined'].values):
  sent = decontracted(sentance)
  sent = sent.replace('\\r', ' ')
```

```
  sent = sent.replace('\\"', ' ')
  sent = sent.replace('\\n', ' ')
  sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
   # https://gist.github.com/sebleier/554280
  sent = ' '.join(e for e in sent.split() if e.lower() not in stopwords)
  preprocessed_essays_titles_test.append(sent.lower().strip())
```

```
100%|██████████| 13200/13200 [00:07<00:00, 1809.03it/s]
```

2.2 Make Data Model Ready: encoding numerical, categorical features

1.*project*_subject_categories convert categorical to vectors*

In [25]:

```
# convert train and test data of clean_categories into vectors


# we use count vectorizer to convert the values into one
from sklearn.feature_extraction.text import CountVectorizer
vectorizer1 = CountVectorizer(vocabulary=list(sorted_cat_dict.keys()), lowercase=False,
binary=True)
vectorizer1.fit(X_train['clean_categories'].values)

# firstly convert fit the train data into the vectoriaer then it learn hte vocablery

# we use the fitted CountVectorizer to convert the text to vector
X_train_cat = vectorizer1.transform(X_train['clean_categories'].values)
X_test_cat = vectorizer1.transform(X_test['clean_categories'].values)

print(vectorizer1.get_feature_names())
```

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

In [26]:

```
print("After vectorizations")
print(X_train_cat.shape, y_train.shape)
print(X_test_cat.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(26800, 9) (26800,)
(13200, 9) (13200,)
====================================================================================================
```

2.*project*_subject_subcategories convert categorical to vectors*

In [27]:

```
# convert train and test data of clean_categories into vectors


# we use count vectorizer to convert the values into one
from sklearn.feature_extraction.text import CountVectorizer
vectorizer2 = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()), lowercase=False, binary
=True)
vectorizer2.fit(X_train['clean_subcategories'].values)

# firstly convert fit the train data into the vectoriaer then it learn hte vocablery
```

```
# we use the fitted CountVectorizer to convert the text to vector
X_train_subcat = vectorizer2.transform(X_train['clean_subcategories'].values)
X_test_subcat = vectorizer2.transform(X_test['clean_subcategories'].values)

print(vectorizer2.get_feature_names())
```

```
['Economics', 'CommunityService', 'FinancialLiteracy', 'ParentInvolvement', 'Extracurricular',
'ForeignLanguages', 'Civics_Government', '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']
```

In [28]:

```
print("After vectorizations")
print(X_train_subcat.shape, y_train.shape)
#print(X_cv_subcat.shape, y_cv.shape)
print(X_test_subcat.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(26800, 30) (26800,)
(13200, 30) (13200,)
====================================================================================================
```

*3 school_state convert categorical to vectors**

In [29]:

```
# now time to cont the each words
from collections import Counter
my_counter = Counter()
for word in project_data['school_state'].values:
    my_counter.update(word.split())# count the words

school_state_dict = dict(my_counter)# store in dicionary
sorted_school_state_dict = dict(sorted(school_state_dict.items(), key=lambda kv: kv[1]))# sor it
print(sorted_school_state_dict)
```

```
{'VT': 22, 'WY': 39, 'ND': 54, 'MT': 85, 'RI': 107, 'NH': 107, 'SD': 115, 'AK': 116, 'NE': 121,
'DE': 130, 'WV': 181, 'HI': 183, 'ME': 184, 'NM': 187, 'DC': 204, 'KS': 228, 'ID': 238, 'IA': 241,
'AR': 344, 'CO': 422, 'MN': 443, 'MS': 461, 'OR': 461, 'KY': 492, 'MD': 526, 'NV': 539, 'AL': 620,
'CT': 630, 'UT': 631, 'TN': 632, 'WI': 663, 'VA': 739, 'NJ': 813, 'AZ': 816, 'OK': 836, 'MA': 858,
'LA': 872, 'WA': 891, 'MO': 924, 'IN': 936, 'OH': 960, 'PA': 1139, 'MI': 1185, 'SC': 1449, 'GA': 14
53, 'IL': 1598, 'NC': 1872, 'FL': 2238, 'TX': 2673, 'NY': 2730, 'CA': 5612}
```

In [30]:

```
# convert train,cv and test data of clean_categories into vectors


# we use count vectorizer to convert the values into one
from sklearn.feature_extraction.text import CountVectorizer
vectorizer3 = CountVectorizer(vocabulary=list(sorted_school_state_dict.keys()), lowercase=False, b
inary=True)
vectorizer3.fit(project_data['school_state'].values)

# firstly convert fit the train data into the vectoriaer then it learn hte vocablery

# we use the fitted CountVectorizer to convert the text to vector
X_train_school_state = vectorizer3.transform(X_train['school_state'].values)
#X_cv_school_state = vectorizer3.transform(X_cv['school_state'].values)
X_test_school_state = vectorizer3.transform(X_test['school_state'].values)

print(vectorizer3.get_feature_names())
```

```
['VT', 'WY', 'ND', 'MT', 'RI', 'NH', 'SD', 'AK', 'NE', 'DE', 'WV', 'HI', 'ME', 'NM', 'DC', 'KS', 'I
D', 'IA', 'AR', 'CO', 'MN', 'MS', 'OR', 'KY', 'MD', 'NV', 'AL', 'CT', 'UT', 'TN', 'WI', 'VA', 'NJ',
'AZ', 'OK', 'MA', 'LA', 'WA', 'MO', 'IN', 'OH', 'PA', 'MI', 'SC', 'GA', 'IL', 'NC', 'FL', 'TX', 'NY
', 'CA']
```

In [31]:

```
print("After vectorizations")
print(X_train_school_state .shape, y_train.shape)
#print(X_cv_school_state .shape, y_cv.shape)
print(X_test_school_state .shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(26800, 51) (26800,)
(13200, 51) (13200,)
====================================================================================================
```

In [32]:

```
#https://stackoverflow.com/questions/42224700/attributeerror-float-object-has-no-attribute-split
project_data['clean_grade']=project_data['clean_grade'].fillna("")# fill the nulll values with
space

# convert train,cv and test data of clean_categories into vectors


# we use count vectorizer to convert the values into one
from sklearn.feature_extraction.text import CountVectorizer
vectorizer4 = CountVectorizer(vocabulary=list(sorted_project_grade_category_dict.keys()),
lowercase=False, binary=True)
vectorizer4.fit(project_data['clean_grade'].values)

# firstly convert fit the train data into the vectoriaer then it learn hte vocablery

# we use the fitted CountVectorizer to convert the text to vector
X_train_project_grade_category = vectorizer4.transform(X_train['clean_grade'].values)
#X_cv_project_grade_category = vectorizer4.transform(X_cv['clean_grade'].values)
X_test_project_grade_category = vectorizer4.transform(X_test['clean_grade'].values)

print(vectorizer4.get_feature_names())
```

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

In [33]:

```
print("After vectorizations")
print(X_train_project_grade_category  .shape, y_train.shape)
#print(X_cv_project_grade_category  .shape, y_cv.shape)
print(X_test_project_grade_category  .shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(26800, 4) (26800,)
(13200, 4) (13200,)
====================================================================================================
```

In [0]:

```
#https://stackoverflow.com/questions/42224700/attributeerror-float-object-has-no-attribute-split
project_data['teacher_prefix']=project_data['teacher_prefix'].fillna(" ")# filll the null values
with space
```

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


# dict sort by value python: https://stackoverflow.com/a/613218/4084039
teacher_cat_dict = dict(my_counter)
sorted_teacher_prefix_dict = dict(sorted(teacher_cat_dict.items(), key=lambda kv: kv[1]))
```

In [35]:

```
# convert train,cv and test data of clean_categories into vectors


# we use count vectorizer to convert the values into one
from sklearn.feature_extraction.text import CountVectorizer
vectorizer5 = CountVectorizer(vocabulary=list(sorted_teacher_prefix_dict.keys()), lowercase=False,
binary=True)
vectorizer5.fit(project_data['teacher_prefix'].values.astype('U'))

# firstly convert fit the train data into the vectoriaer then it learn hte vocablery

# we use the fitted CountVectorizer to convert the text to vector
X_train_teacher_prefix = vectorizer5.transform(X_train['teacher_prefix'].values.astype('U'))
#X_cv_teacher_prefix= vectorizer5.transform(X_cv['teacher_prefix'].values.astype('U'))
X_test_teacher_prefix = vectorizer5.transform(X_test['teacher_prefix'].values.astype('U'))

print(vectorizer5.get_feature_names())


# when i executeed this error comes
#np.nan is an invalid document, expected byte or unicode string.
# then iconvert to unicode     just writ .astype('U') after the .values in fit and trainform
#https://stackoverflow.com/questions/39303912/tfidfvectorizer-in-scikit-learn-valueerror-np-nan-is
-an-invalid-document
```

```
['Dr.', 'Teacher', 'Mr.', 'Ms.', 'Mrs.']
```

In [36]:

```
print("After vectorizations")
print(X_train_teacher_prefix   .shape, y_train.shape)
#print(X_cv_teacher_prefix   .shape, y_cv.shape)
print(X_test_teacher_prefix   .shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(26800, 5) (26800,)
(13200, 5) (13200,)
====================================================================================================
```

1.5.3 Vectorizing Numerical features¶

In [37]:

```
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')
print(price_data.head(2))


# we also have to do this in tran,test and cv
# so also merge the resource data with the trian,cv and test

X_train = pd.merge(X_train, price_data, on = "id", how = "left")
#print(x_train.columns)
X_test = pd.merge(X_test, price_data, on = "id", how = "left")
#X_cv = pd.merge(X_cv, price_data, on = "id", how = "left")
```

```
        id   price  quantity
```

```
0  p000001  459.56         7
1  p000002  515.89        21
```

Standadized price for the train,test and cv

In [0]:

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


price_scalar =  StandardScaler()

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

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



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



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

Stadadized Previous_year_tecaher_projects train,test and cv

In [0]:

```
# previous_year_projects
price_scalar.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1)) # fi
nding the mean and standard deviation of this data
#print(f"Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(price_scalar.var_[0])}")

# Now standardize the data with above maen and variance.
train_prev_proj_standar =
price_scalar.transform(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,
1))
# Now standardize the data with above maen and variance.
test_prev_proj_standar =
price_scalar.transform(X_test['teacher_number_of_previously_posted_projects'].values.reshape(-1, 1)
)
# Now standardize the data with above maen and variance.
#cv_prev_proj_standar =
price_scalar.transform(X_cv['teacher_number_of_previously_posted_projects'].values.reshape(-1, 1))
```

Standaized the Quantity column of the train,test and cv

In [0]:

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

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

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


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

In [41]:

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

```
(26800, 20)
(13200, 20)
```

New feature(No. of words in title)

In [0]:

```
# For train data
title_length_train=[]
for i in range(0,X_train.shape[0]):
  title_length_train.append(len(X_train["project_title"][i].split()))


title_length_train=np.array(title_length_train)

#for test data titles
title_length_test=[]
for i in range(0,X_test.shape[0]):
  title_length_test.append(len(X_test["project_title"][i].split()))

title_length_test=np.array(title_length_test)
```

New feature(No. of words in combined essays)

In [0]:

```
#for test data esssay
essay_length_test=[]
for i in range(0,X_test.shape[0]):
  essay_length_test.append(len(X_test["essay"][i].split()))

essay_length_test=np.array(essay_length_test)


#for train data essay

essay_length_train=[]
for i in range(0,X_train.shape[0]):
  essay_length_train.append(len(X_train["essay"][i].split()))

essay_length_train=np.array(essay_length_train)
```

New feature(Sentiment scores of each combined essay's)

In [44]:

```
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer
nltk.download('vader_lexicon')


#https://www.programcreek.com/python/example/100005/nltk.sentiment.vader.SentimentIntensityAnalyze

def analyze_sentiment(df):
```

```
    sentiments = []
    sid = SentimentIntensityAnalyzer()
    for i in range(df.shape[0]):
        line = df['essay'][i]# take one essay
        sentiment = sid.polarity_scores(line)# calculate the sentiment
        sentiments.append([sentiment['neg'], sentiment['pos'],
                           sentiment['neu'], sentiment['compound']])# list of lists
    df[['neg', 'pos', 'neu', 'compound']] = pd.DataFrame(sentiments)
    df['Negative'] = df['compound'] < -0.1
    df['Positive'] = df['compound'] > 0.1
    return df
```

```
/usr/local/lib/python3.6/dist-packages/nltk/twitter/__init__.py:20: UserWarning:

The twython library has not been installed. Some functionality from the twitter package will not b
e available.
```

```
[nltk_data] Downloading package vader_lexicon to /root/nltk_data...
```

In [0]:

```
X_train=analyze_sentiment(X_train)
X_test=analyze_sentiment(X_test)
```

Assinment 11: Truncated SVD

- 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**

Step 1: Selecting top 2000 words from `essay` and `project_title` based on their `idf` values.

project_titles based on their idf values.

In [66]:

```
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import CountVectorizer
#Select the top 2k words from essay text and project_title (concatinate essay text with project ti
tle and then find the top 2k words) based on their idf_ values

#instantiate CountVectorizer()
cv=CountVectorizer()
word_count_vector=cv.fit_transform(preprocessed_essays_titles_train)
word_count_vector.shape
# this means    37022 unique words we have
```

Out[66]:

```
(26800, 37022)
```

In [67]:

```
 tfidf_transformer=TfidfTransformer(smooth_idf=True,use_idf=True)
tfidf_transformer.fit(word_count_vector)
```

Out[67]:

```
TfidfTransformer(norm='l2', smooth_idf=True, sublinear_tf=False, use_idf=True)
```

idf_ values and his coresponding feature name:

In [114]:

```
# we want descending order , because basically that means if the idf high that means its occurence
#in the corpus is very very less, that means -> most important and unique words that can define ou
r statement
# print idf values
df_idf = pd.DataFrame(tfidf_transformer.idf_,
index=cv.get_feature_names(),columns=["tf_idf_weights"])

# sort ascending
sorted_descending_idf=df_idf.sort_values(by=['tf_idf_weights'],ascending=False)
print(sorted_descending_idf.head(5))
idf_features=sorted_descending_idf.index[:2000]
```

```
                tf_idf_weights
mcauliffe            10.503047
nannanafternoon      10.503047
nannanadvocating     10.503047
nannanadventure      10.503047
nannanadvancement    10.503047
```

step 2:Computing Co-occurance matrix

In [0]:

```
# # For reference:
# # https://gist.github.com/nkt1546789/e9fc84579b9c8356f1e5
# # https://stackoverflow.com/questions/41661801/python-calculate-the-co-occurrence-matrix


# #                co-occurence matrix code   for train_data

# #   ->window size
# window=5
#  #  -> 2000*2000 matrix of zeros
# coocc_train = np.zeros((len(idf_features),len(idf_features)),np.float64)
```

```
# cooc_train = np.zeros((len(idf_features),len(idf_features)),np.float64)
# #    -> Make a dictionary in which keys are your features, values are the indexes.

# list_of_words_dict = {idf_features[i]:i for i in range(len(idf_features))}
# #    ->Loop to the total 2k words


# for i in (list_of_words_dict.keys()):
#   #    ->Loop to the corpus
#   for j in preprocessed_essays_titles_train:
#     #split each word
#       j = j.split()
#       #->  condition if our which are in 2k words  not in this sentence then just skip this
sentence

#       if str(i) not in j:
#         continue

#         # Loop to 2k words indexes, if our word is under window of some word  then add it to the
co-occurence matrix;

#       else:
#         for x in list_of_words_dict.values():
#           if abs(list_of_words_dict[i]-x)<window:
#             cooc_train[list_of_words_dict[i],x]+=1

# # end loop:)
```

In [0]:

```
# # start func:)




#  #  -> 2000*2000 matrix of zeros
cooc_train=np.zeros([2000,2000])#    2k*2k features
def co_occurence_matrix(win,vocab,corpus,coo_matrix):
  # #   ->window size
  window=win
  # #  -> total 2k words
  a=vocab
  #  # -> loop to the each word in the top2k features
  for q,word in enumerate(vocab):
    #   #    ->Loop to the corpus
    for i in corpus:
     ## -> if word present in corpus  then go further  othersize  skip it go to the next essay
      if word in i:
        # # ->  split it into teh words in a list
        arr=[g for g in i.split(' ')]

        # for each word
        for j,d in enumerate(arr):
          arrr=[]
          # just make a loop to the and append all the elements which are under current word's win
dow
          for i in range(max(0,j-window),min(j+window,len(arr)-1)):# window size
            arrr.append(arr[x])


            #loop to the window start to end
            for f,w in  enumerate(arrr):
              if wd in vocab:
                if wd!=word:
                  index=vocab.index(wd)
                  coo_matrix[q,index]+=1
    return coo_matrix


# # end func:)
```

In [0]:

```
cooc_train=co_occurence_matrix(5,idf_features,preprocessed_essays_titles_train,cooc_train)
print(cooc_train.shape)# <=  for train
```

```
(2000, 2000)
```

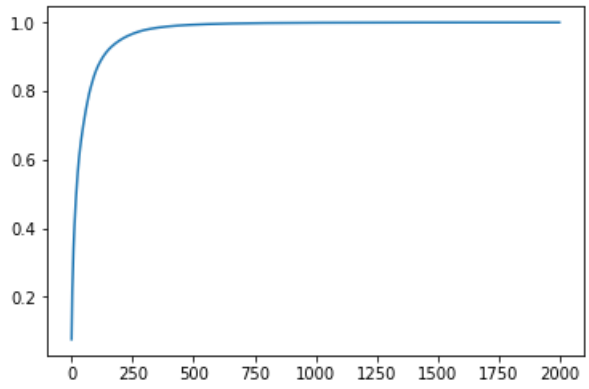
Step 3: Applying TruncatedSVD and Calculating Vectors for `essay` and `project_title`

In [0]:

```
from sklearn.decomposition import TruncatedSVD

svd = TruncatedSVD(n_components=1999, n_iter=7, random_state=42)
svd.fit(cooc_train)
ratio=svd.explained_variance_ratio_
cum_sum=np.cumsum(ratio)
plt.plot(cum_sum)
```

Out[0]:

```
[<matplotlib.lines.Line2D at 0x7ff43e24aac8>]
```



Observations: From the 250 dimensions onwards, graph is not increasing so much, so we choose 250

In [0]:

```
svd = TruncatedSVD(n_components=250, n_iter=7, random_state=42)
result_train=svd.fit_transform(cooc_train)
```

In [0]:

```
print(result_train.shape)
```

```
(2000, 250)
```

In [0]:

```
model = result_train
glove_words = idf_features
keys={}
for i,j in enumerate(glove_words):
  keys[j]=i
```

In [0]:

```
""#for essay
# average Word2Vec
# compute average word2vec for each review.
def func(wordlist):


  train_avg_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
```

```
  for sentence in tqdm(wordlist): # for each review/sentence
    vector = np.zeros(250) # as word vectors are of zero length    # we are taking the 300
dimensions  very large
    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[keys[word]]
          cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    train_avg_w2v_vectors.append(vector)

  print(len(train_avg_w2v_vectors))
  print(len(train_avg_w2v_vectors[0]))
  return train_avg_w2v_vectors
```

In [0]:

```
train_avg_w2v_vectors=func(preprocessed_essays_train)
test_avg_w2v_vectors=func(preprocessed_essays_test)

test_avg_w2v_vectors_title=func(preprocessed_titles_test)
train_avg_w2v_vectors_title=func(preprocessed_titles_train)
```

```
100%|██████████| 26800/26800 [01:12<00:00, 370.07it/s]
  1%|          | 76/13200 [00:00<00:34, 377.01it/s]

26800
250

100%|██████████| 13200/13200 [00:35<00:00, 368.11it/s]
  9%|▉         | 1162/13200 [00:00<00:01, 11618.74it/s]

13200
250

100%|██████████| 13200/13200 [00:01<00:00, 10828.80it/s]
  7%|▋         | 1867/26800 [00:00<00:02, 9620.25it/s]

13200
250

100%|██████████| 26800/26800 [00:02<00:00, 11022.08it/s]

26800
250
```

Step 4: Concatenate these truncatedSVD matrix, with the matrix with features

In [0]:

```
#for train

pos=list(X_train['pos'])
pos=np.array(pos)
neg=list(X_train['neg'])
neg=np.array(neg)
com=list(X_train['compound'])
com=np.array(com)

#combine all
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X_set5_train = hstack((
                      X_train_teacher_prefix,X_train_cat,X_train_subcat
```

```
,X_train_project_grade_category,X_train_school_state,#all categorials
                      train_qnty_standar,train_price_standar,train_prev_proj_standar,
                      essay_length_train.reshape(-1,1),title_length_train.reshape(-1,1),
                      pos.reshape(-1,1),neg.reshape(-1,1),com.reshape(-1,1),train_avg_w2v_vectors,t
ain_avg_w2v_vectors_title
                                                                                  ))# all numericals

print(X_set5_train.shape, y_train.shape)

#X_train['pos'],X_train['neg'],X_train['neu'],
```

```
(26800, 607) (26800,)
```

In [0]:

```
#for test
pos=list(X_test['pos'])
pos=np.array(pos)

neg=list(X_test['neg'])
neg=np.array(neg)

com=list(X_test['compound'])
com=np.array(com)

# combine all
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X_set5_test = hstack((
                      X_test_teacher_prefix,X_test_cat,X_test_subcat ,X_test_project_grade_category
,X_test_school_state,#all categorials
                      test_qnty_standar,test_price_standar,test_prev_proj_standar,
                      essay_length_test.reshape(-1,1),title_length_test.reshape(-1,1),
                      pos.reshape(-1,1),neg.reshape(-1,1),com.reshape(-1,1),test_avg_w2v_vectors,te
t_avg_w2v_vectors_title
                                                                                  ))# all numericals


print(X_set5_test.shape, y_test.shape)

#X_train['pos'],X_train['neg'],X_train['neu'],
```

```
(13200, 607) (13200,)
```

Step 5: Apply XgBoost on you matrix

In [0]:

```
 #Define  a class

  #DO REFER THIS BLOG: XGBOOST DMATRIX

from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestClassifier
import xgboost as xgb
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import GradientBoostingClassifier

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

    def fit(self, X, y, num_boost_round=None):
        num_boost_round = num_boost_round or self.num_boost_round
        self.label2num = {label: i for i, label in enumerate(sorted(set(y)))}
```

```
        dtrain = xgb.DMatrix(X, label=[self.label2num[label] for label in y])
        self.clf = xgb.train(params=self.params, dtrain=dtrain, num_boost_round=num_boost_round, ve
rbose_eval=1)

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

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

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

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

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

Hyperparameter tuning:

In [0]:

```
dt7= XGBoostClassifier( num_class = 2, nthread = 4)

#------------------------------------------------------------------------------------------------
----------------------------------------#

# I took ranges of num_boost_round and max_depth very less ,Because  of runtime errors in google c
olab
.
#------------------------------------------------------------------------------------------------
----------------------------------------#

parameters = {'num_boost_round': [5, 8,11,15,20], 'max_depth':[2, 3, 5,  7,  10] }
clf7 = GridSearchCV(dt7, parameters, cv=3, scoring='roc_auc',return_train_score=True)
se7 = clf7.fit(X_set5_train, y_train)


import seaborn as sns; sns.set()
```

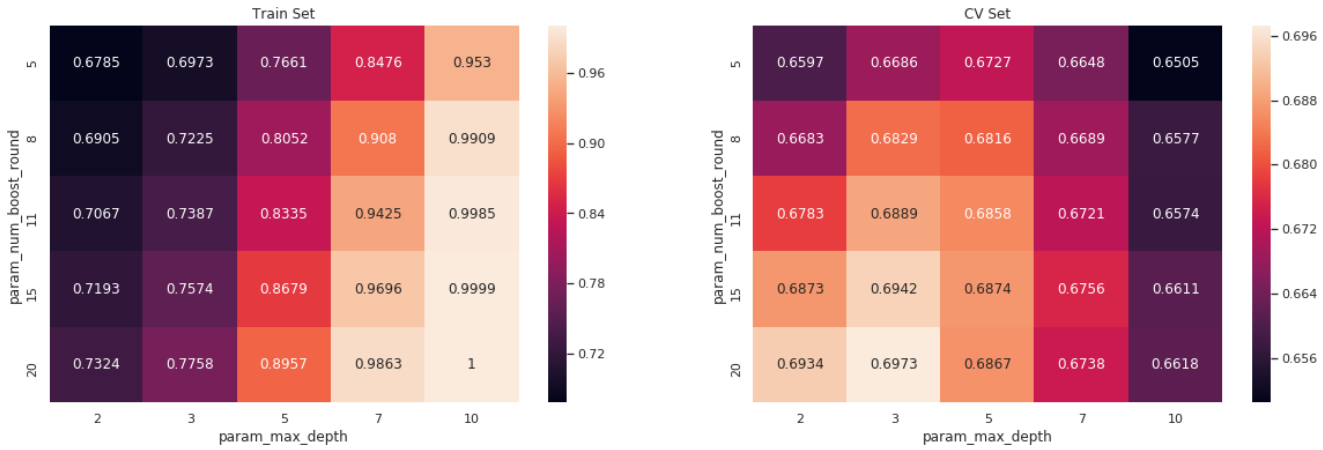
Plot the grid

In [0]:

```
print(clf7.cv_results_.keys())

max_scores1 = pd.DataFrame(clf7.cv_results_).groupby(['param_num_boost_round', 'param_max_depth'])
.max().unstack()[['mean_test_score', 'mean_train_score']]


fig, ax = plt.subplots(1,2, figsize=(20,6))

sns.heatmap(max_scores1.mean_train_score, annot = True, fmt='.4g', ax=ax[0])
sns.heatmap(max_scores1.mean_test_score, annot = True, fmt='.4g', ax=ax[1])

ax[0].set_title('Train Set')
ax[1].set_title('CV Set')
```

```
plt.show()
```

```
dict_keys(['mean_fit_time', 'std_fit_time', 'mean_score_time', 'std_score_time',
'param_max_depth', 'param_num_boost_round', 'params', 'split0_test_score', 'split1_test_score', 's
plit2_test_score', 'mean_test_score', 'std_test_score', 'rank_test_score', 'split0_train_score', '
split1_train_score', 'split2_train_score', 'mean_train_score', 'std_train_score'])
```



Best parameter

In [0]:

```
print(clf7.score(X_set5_train,y_train))
print(clf7.score(X_set5_test,y_test))
print(clf7.best_params_)
print(clf7.best_score_)
```

```
0.7544784550151872
0.7006129900610101
{'max_depth': 3, 'num_boost_round': 20}
0.6972502897659504
```

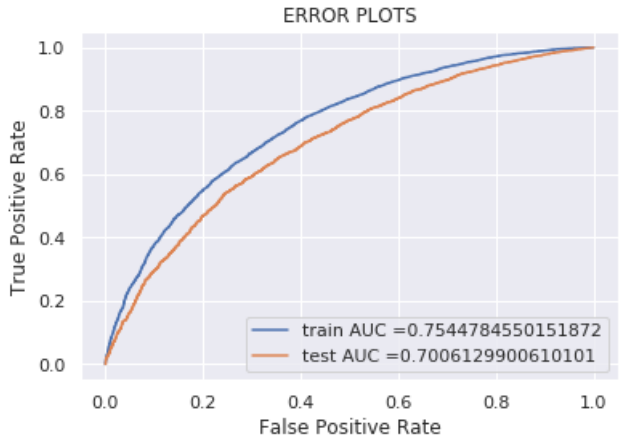
In [0]:

```
#Best tune parameters
best_tune_parameters=[{'num_boost_round': [20],  'max_depth':[3]  }  ]
```

In [0]:

```
# https://scikit-
learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import roc_curve, auc



clf11 = GridSearchCV(XGBoostClassifier( num_class = 2, nthread = 4),best_tune_parameters)
clf11.fit(X_set5_train, y_train)


#https://scikit-
learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html#sklearn.linear_model.SGI
sifier.decision_function


y_train_pred1 = clf11.predict_proba(X_set5_train) [:,1]
y_test_pred1 =  clf11.predict_proba(X_set5_test) [:,1]


train_fpr1, train_tpr1, tr_thresholds1 = roc_curve(y_train, y_train_pred1)
test_fpr1, test_tpr1, te_thresholds1 = roc_curve(y_test, y_test_pred1)
```

```
plt.plot(train_fpr1, train_tpr1, label="train AUC ="+str(auc(train_fpr1, train_tpr1)))
plt.plot(test_fpr1, test_tpr1, label="test AUC ="+str(auc(test_fpr1, test_tpr1)))
plt.legend()
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ERROR PLOTS")
plt.grid(True)
plt.show()
```



Confusion matrix:

In [0]:

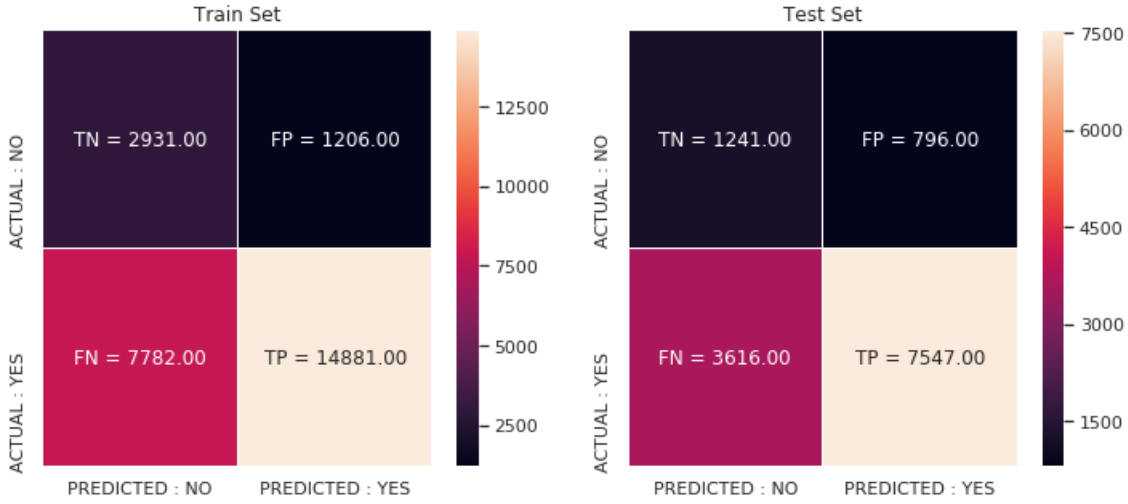
```
def predict(proba, threshould, fpr, tpr):

    t = threshould[np.argmax(fpr*(1-tpr))]
    print("the maximum value of tpr*(1-fpr)", np.round(max(tpr*(1-fpr)),2) , "for threshold", np.ro
und(t,2))
    predictions = []
    global predictions1    # make it global
    for i in proba:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)

    predictions1= predictions
    return predictions
```

In [0]:

```
#https://www.quantinsti.com/blog/creating-heatmap-using-python-seaborn
import seaborn as sns; sns.set()

con_m_train = confusion_matrix(y_train, predict(y_train_pred1, tr_thresholds1, train_fpr1, train_tp
r1))
con_m_test = confusion_matrix(y_test, predict(y_test_pred1, te_thresholds1, test_fpr1, test_tpr1))

key = (np.asarray([['TN','FP'], ['FN', 'TP']]))

fig, ax = plt.subplots(1,2, figsize=(12,5))

labels_train = (np.asarray(["{0} = {1:.2f}" .format(key, value) for key, value in zip(key.flatten()
, con_m_train.flatten())])).reshape(2,2)
labels_test = (np.asarray(["{0} = {1:.2f}" .format(key, value) for key, value in zip(key.flatten(),
con_m_test.flatten())])).reshape(2,2)

sns.heatmap(con_m_train, linewidths=.5, xticklabels=['PREDICTED : NO', 'PREDICTED : YES'],
yticklabels=['ACTUAL : NO', 'ACTUAL : YES'], annot = labels_train, fmt = '', ax=ax[0])
sns.heatmap(con_m_test, linewidths=.5, xticklabels=['PREDICTED : NO', 'PREDICTED : YES'],
yticklabels=['ACTUAL : NO', 'ACTUAL : YES'], annot = labels_test, fmt = '', ax=ax[1])

ax[0].set_title('Train Set')
```

```
ax[1].set_title('Test Set')

plt.show()
```

```
the maximum value of tpr*(1-fpr) 0.47 for threshold 0.85
the maximum value of tpr*(1-fpr) 0.42 for threshold 0.84
```



3. Conclusion

In [0]:

```
from prettytable import PrettyTable

tb = PrettyTable()
tb.field_names= ("Vectorizer", "Model", "num_boost_round","max_depth" ,  "AUC")
tb.add_row(["wordtovec          ", "   XgBoost  ",   20,                                3,
70])
print(tb.get_string(titles = "XgBoost > - Observations"))
#print(tb)
```

```
+--------------------+--------------+-----------------+-----------+-----+
|     Vectorizer     |    Model     | num_boost_round | max_depth | AUC |
+--------------------+--------------+-----------------+-----------+-----+
| wordtovec          |    XgBoost   |        20       |     3     |  70 |
+--------------------+--------------+-----------------+-----------+-----+
```

In [0]:

```

```