Donors Choose - Truncated SVD

Importing packages

In [2]:

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

Reading the data

In [3]:

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

Project data

In [4]:

```
print("Number of data points in train data", project_data.shape)
print('-'*50)
print("Attributes :", project_data.columns.values)
project_data.head(2)
```

```
Number of data points in train data (109248, 17)
--------------------------------------------------
Attributes : ['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']
```

Out[4]:

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

In [5]:

```
# https://matplotlib.org/gallery/pie_and_polar_charts/pie_and_donut_labels.html#sphx-glr-ga

y_value_counts = project_data['project_is_approved'].value_counts()
print("Number of projects thar are approved for funding ", y_value_counts[1], ", (", (y_val
print("Number of projects thar are not approved for funding ", y_value_counts[0], ", (", (y
```

```
Number of projects thar are approved for funding  92706 , ( 84.8583040421792
7 %)
Number of projects thar are not approved for funding  16542 , ( 15.141695957
820739 %)
```

Sampling out the data points: Considering 50k Random samples

In [6]:

```
approved_project=project_data[project_data["project_is_approved"]==1].sample(n=35000,random
rejected_project=project_data[project_data["project_is_approved"]==0].sample(n=15000,random
project_data=pd.concat([approved_project,rejected_project] )
```

Handling Missing Value in "Teacher prefix" column

In [7]:

```
a = project_data['teacher_prefix'].mode().values
```

In [8]:

```
project_data['teacher_prefix'] = project_data['teacher_prefix'].fillna(a[0])
```

Total number of null values in each column

In [9]:

```
#Total number of null values in each column
project_data.isnull().sum(axis = 0)
```

Out[9]:

```
Unnamed: 0                                      0
id                                              0
teacher_id                                      0
teacher_prefix                                  0
school_state                                    0
project_submitted_datetime                      0
project_grade_category                          0
project_subject_categories                      0
project_subject_subcategories                   0
project_title                                   0
project_essay_1                                 0
project_essay_2                                 0
project_essay_3                             48346
project_essay_4                             48346
project_resource_summary                        0
teacher_number_of_previously_posted_projects    0
project_is_approved                             0
dtype: int64
```

Resource data

In [10]:

```
print("Number of data points in train data", resource_data.shape)
print('-'*50)
print("Attributes: ", resource_data.columns.values)
resource_data.head(2)
```

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

Out[10]:

	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

Replacing date-time with date

In [11]:

```
# 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)
project_data.sort_values(by=['Date'], inplace=True)


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


project_data.head(2)
```

Out[11]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	
55660	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	CA	00:
76127	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	UT	00:

Preprocessing Categorical Data

Project Subject Categories

In [12]:

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

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

In [13]:

```
sorted_cat_dict
```

Out[13]:

```
{'warmth': 609,
 'care_hunger': 609,
 'history_civics': 2635,
 'music_arts': 4755,
 'appliedlearning': 5765,
 'specialneeds': 6331,
 'health_sports': 6486,
 'math_science': 19015,
 'literacy_language': 23496}
```

Project Subject Sub-Categories

In [14]:

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

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

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

In [15]:

```
sorted_sub_cat_dict
```

Out[15]:

```
{'economics': 119,
 'communityservice': 214,
 'financialliteracy': 263,
 'parentinvolvement': 307,
 'civics_government': 376,
 'extracurricular': 392,
 'foreignlanguages': 456,
 'warmth': 609,
 'care_hunger': 609,
 'nutritioneducation': 678,
 'socialsciences': 850,
 'performingarts': 905,
 'charactereducation': 1023,
 'teamsports': 1058,
 'other': 1105,
 'college_careerprep': 1188,
 'music': 1383,
 'history_geography': 1397,
 'earlydevelopment': 2024,
 'esl': 2034,
 'health_lifescience': 2037,
 'gym_fitness': 2088,
 'environmentalscience': 2626,
 'visualarts': 2955,
 'health_wellness': 4569,
 'appliedsciences': 5037,
 'specialneeds': 6331,
 'literature_writing': 10035,
 'mathematics': 12726,
 'literacy': 15028}
```

Teacher prefix categories

In [16]:

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

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

prefix_list = []
for i in prefix:
    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", "
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"Math
        temp +=j.strip()+" "#" abc ".strip() will return "abc", remove the trailing spaces
        temp = temp.replace('.','')
        temp = temp.lower()
    prefix_list.append(temp.strip())

project_data['prefix_teacher'] = prefix_list
project_data.drop(['teacher_prefix'], 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['prefix_teacher'].values:
    my_counter.update(word.split())

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

In [17]:

```
sorted_prefix_dict
```

Out[17]:

```
{'dr': 5, 'teacher': 1134, 'mr': 4967, 'ms': 18029, 'mrs': 25865}
```

Project Grade categories

In [18]:

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

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

grades_list = []
for i in grades:
    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", "
        j = j.replace(' ','_') # we are placeing all the ' '(space) with ''(empty) ex:"Math
        temp +=j.strip()+" "#" abc ".strip() will return "abc", remove the trailing spaces
        temp = temp.replace('-','_')
        temp = temp.lower()
    grades_list.append(temp.strip())

project_data['project_grade'] = grades_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['project_grade'].values:
    my_counter.update(word.split())

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

In [19]:

```
sorted_grade_dict
```

Out[19]:

```
{'grades_9_12': 5153,
 'grades_6_8': 7805,
 'grades_3_5': 16765,
 'grades_prek_2': 20277}
```

Preprocessing Text Data

Project Essay

In [20]:

```
# 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)
```

Compound Sentiment score of Project essay

In [21]:

```
#https://medium.com/analytics-vidhya/simplifying-social-media-sentiment-analysis-using-vade

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

sid = SentimentIntensityAnalyzer()

essays = project_data['essay']
essays_sentiments = []

for essay in tqdm(essays):
    res = sid.polarity_scores(essay)
    essays_sentiments.append(res['compound']) #Considering compound as a criteria.

project_data['essay_sentiment'] = essays_sentiments
```

```
[nltk_data] Downloading package vader_lexicon to
[nltk_data]     C:\Users\vansh\AppData\Roaming\nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!
100%|██████████████████████████████████████████████████████████████████████████████
████| 50000/50000 [03:19<00:00, 250.47it/s]
```

In [22]:

```
# 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
```

Word count of project essay and title

In [23]:

```
#https://www.geeksforgeeks.org/python-program-to-count-words-in-a-sentence/
for col_type, new_col in [('project_title', 'title_size'), ('essay', 'essay_size')]:
    col_data = project_data[col_type]
    col_size = []
    for sen in col_data:
        sen = decontracted(sen)
        col_size.append(len(sen.split()))
    project_data[new_col] = col_size
```

In [24]:

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

Out[24]:

	Unnamed: 0	id	teacher_id	school_state	Date	project_t
55660	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	CA	2016-04-27 00:27:36	Engineer STEAM i the Prim Classro
76127	37728	p043609	3f60494c61921b3b43ab61bdde2904df	UT	2016-04-27 00:31:25	Sens Tools Fo

2 rows × 21 columns

In [25]:

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


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

```
100%|████████████████████████████████████████████████████████████████
███| 50000/50000 [00:40<00:00, 1235.73it/s]
```

In [26]:

```
# placing the preprocessed essay into the dataframe
project_data['clean_essays'] = preprocessed_essays
project_data.drop(['essay'], axis=1, inplace=True)
project_data.drop(['project_essay_1'], axis=1, inplace=True)
project_data.drop(['project_essay_2'], axis=1, inplace=True)
project_data.drop(['project_essay_3'], axis=1, inplace=True)
project_data.drop(['project_essay_4'], axis=1, inplace=True)
```

In [27]:

```
#Printing random cleaned essay
project_data['clean_essays'].values[23]
```

Out[27]:

```
'teach science technology 3 8 grades recently middle schoolers told would go
od idea real technology stem classes students since third grade seen grow ph
ysically mature emotionally students school encouraged lead example inspire
creativity voice opinions actions master academics working project alone evi
dence desire master academics working team 2014 ten percent jobs us required
stem skills hope continued support encouragement students begin see stem car
eer option future students needed scan print park designs third term enginee
ring project design paper park also share work others students believed acce
ss color printer scanner sharing work would easier students trying visualize
work better even design something computers actually would take things home
continue working printer class not extra one period stem classroom lower gra
des hands science classroom resources learn technology engineering need tech
nology classroom order good stem classroom'
```

Project title

In [28]:

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

```
100%|██████████████████████████████████████████████████████████████████████
██| 50000/50000 [00:01<00:00, 27569.82it/s]
```

In [29]:

```
# placing the preprocessed essay into the dataframe
project_data['clean_titles'] = preprocessed_titles
project_data.drop(['project_title'], axis=1, inplace=True)
```

In [30]:

```
#Printing random cleaned title
project_data['clean_titles'].values[12]
```

Out[30]:

```
'duct duct craft spring inspiration'
```

Merging Price and quantity data to Project data (left joining price data)

In [31]:

```
# reference : https://stackoverflow.com/questions/22407798/how-to-reset-a-dataframes-indexe
price_data = resource_data.groupby('id').agg({'price':'sum', 'quantity':'sum'}).reset_index
price_data.head(2)
```

Out[31]:

	id	price	quantity
0	p000001	459.56	7
1	p000002	515.89	21

In [32]:

```
# join two dataframes(project_data and price_data) in python
# reference : https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.m
project_data = pd.merge(project_data, price_data, on='id', how='left')
```

Splitting Data and Starifying the sampling

In [33]:

```
y = project_data['project_is_approved'].values
project_data.drop(['project_is_approved'], axis=1, inplace=True)
X = project_data

print(X.shape)
print(y.shape)
```

```
(50000, 18)
(50000,)
```

In [34]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_spli
#https://stackoverflow.com/questions/34842405/parameter-stratify-from-method-train-test-spl
from sklearn.model_selection import train_test_split

# X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.33, shuffle=Flase)#
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, stratify = y) # t
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33, stratify

print(X_train.shape, y_train.shape)
print(X_cv.shape, y_cv.shape)
print(X_test.shape, y_test.shape)
```

```
(22445, 18) (22445,)
(11055, 18) (11055,)
(16500, 18) (16500,)
```

Vectorizing Categorical Data

Clean Categories

In [35]:

```
# we use count vectorizer to convert the values into one hot encoded features

# Vectorizing "clean_categories"
from sklearn.feature_extraction.text import CountVectorizer
vectorizer_sbj = CountVectorizer(vocabulary=list(sorted_cat_dict.keys()), lowercase=False,
vectorizer_sbj.fit(X_train['clean_categories'].values)

X_train_categories_one_hot = vectorizer_sbj.transform(X_train['clean_categories'].values)
X_cv_categories_one_hot = vectorizer_sbj.transform(X_cv['clean_categories'].values)
X_test_categories_one_hot = vectorizer_sbj.transform(X_test['clean_categories'].values)

print("After verctorizing")
print(X_train_categories_one_hot.shape, y_train.shape)
print(X_cv_categories_one_hot.shape, y_cv.shape)
print(X_test_categories_one_hot.shape, y_test.shape)

print(vectorizer_sbj.get_feature_names())
```

```
After verctorizing
(22445, 9) (22445,)
(11055, 9) (11055,)
(16500, 9) (16500,)
['warmth', 'care_hunger', 'history_civics', 'music_arts', 'appliedlearning',
'specialneeds', 'health_sports', 'math_science', 'literacy_language']
```

Clean sub Categories

In [36]:

```
# Vectorizing "clean_subcategories"
vectorizer_sub_sbj = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()), lowercase
vectorizer_sub_sbj.fit(X_train['clean_subcategories'].values)

X_train_sub_categories_one_hot = vectorizer_sub_sbj.transform(X_train['clean_subcategories'
X_cv_sub_categories_one_hot = vectorizer_sub_sbj.transform(X_cv['clean_subcategories'].valu
X_test_sub_categories_one_hot = vectorizer_sub_sbj.transform(X_test['clean_subcategories'].

print("After verctorizing")
print(X_train_sub_categories_one_hot.shape, y_train.shape)
print(X_cv_sub_categories_one_hot.shape, y_cv.shape)
print(X_test_sub_categories_one_hot.shape, y_test.shape)

print(vectorizer_sub_sbj.get_feature_names())
```

```
After verctorizing
(22445, 30) (22445,)
(11055, 30) (11055,)
(16500, 30) (16500,)
['economics', 'communityservice', 'financialliteracy', 'parentinvolvement',
'civics_government', 'extracurricular', 'foreignlanguages', 'warmth', 'care_
hunger', 'nutritioneducation', 'socialsciences', 'performingarts', 'characte
reducation', 'teamsports', 'other', 'college_careerprep', 'music', 'history_
geography', 'earlydevelopment', 'esl', 'health_lifescience', 'gym_fitness',
'environmentalscience', 'visualarts', 'health_wellness', 'appliedsciences',
'specialneeds', 'literature_writing', 'mathematics', 'literacy']
```

Teacher Prefix

In [37]:

```
# Vectorizing "teacher_prefix"
prefix = list(set(X_train['prefix_teacher'].values))

vectorizer_teacher = CountVectorizer(vocabulary=prefix, lowercase=False, binary=True)
vectorizer_teacher.fit(X_train['prefix_teacher'].values)

X_train_prefix_one_hot = vectorizer_teacher.transform(X_train['prefix_teacher'])
X_cv_prefix_one_hot = vectorizer_teacher.transform(X_cv['prefix_teacher'])
X_test_prefix_one_hot = vectorizer_teacher.transform(X_test['prefix_teacher'])

print("After verctorizing")
print(X_train_prefix_one_hot.shape, y_train.shape)
print(X_cv_prefix_one_hot.shape, y_cv.shape)
print(X_test_prefix_one_hot.shape, y_test.shape)

print(vectorizer_teacher.get_feature_names())
```

```
After verctorizing
(22445, 5) (22445,)
(11055, 5) (11055,)
(16500, 5) (16500,)
['mrs', 'ms', 'mr', 'teacher', 'dr']
```

school state

In [38]:

```
# Vectorizing "school_state"
from collections import Counter
my_counter = Counter()
for word in X_train['school_state'].values:
    my_counter.update(word.split())

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

vectorizer_state = CountVectorizer(vocabulary=list(sorted_state_dict.keys()), lowercase=Fal
vectorizer_state.fit(X_train['school_state'].values)

X_train_state_one_hot = vectorizer_state.transform(X_train['school_state'].values)
X_cv_state_one_hot = vectorizer_state.transform(X_cv['school_state'].values)
X_test_state_one_hot = vectorizer_state.transform(X_test['school_state'].values)

print("After verctorizing")
print(X_train_state_one_hot.shape, y_train.shape)
print(X_cv_state_one_hot.shape, y_cv.shape)
print(X_test_state_one_hot.shape, y_test.shape)

print(vectorizer_state.get_feature_names())
```

```
After verctorizing
(22445, 51) (22445,)
(11055, 51) (11055,)
(16500, 51) (16500,)
['WY', 'VT', 'ND', 'NE', 'RI', 'SD', 'MT', 'DE', 'AK', 'NH', 'WV', 'NM', 'H
I', 'DC', 'ME', 'KS', 'ID', 'IA', 'AR', 'CO', 'MN', 'KY', 'NV', 'OR', 'MS',
'MD', 'AL', 'CT', 'TN', 'UT', 'WI', 'VA', 'OK', 'NJ', 'WA', 'AZ', 'OH', 'M
A', 'LA', 'MO', 'IN', 'MI', 'PA', 'SC', 'GA', 'IL', 'NC', 'FL', 'NY', 'TX',
'CA']
```

Project Grade Category

In [39]:

```
# Vectorizing "project_grade_category"
prefix = list(set(X_train["project_grade"].values))

vectorizer_grade = CountVectorizer(vocabulary=prefix, lowercase=False, binary=True)
vectorizer_grade.fit(X_train['project_grade'])

X_train_grade_one_hot = vectorizer_grade.transform(X_train['project_grade'])
X_cv_grade_one_hot = vectorizer_grade.transform(X_cv['project_grade'])
X_test_grade_one_hot = vectorizer_grade.transform(X_test['project_grade'])

print("After verctorizing")
print(X_train_grade_one_hot.shape, y_train.shape)
print(X_cv_grade_one_hot.shape, y_cv.shape)
print(X_test_grade_one_hot.shape, y_test.shape)

print(vectorizer_grade.get_feature_names())
```

```
After verctorizing
(22445, 4) (22445,)
(11055, 4) (11055,)
(16500, 4) (16500,)
['grades_9_12', 'grades_prek_2', 'grades_3_5', 'grades_6_8']
```

Normalizing Numerical values

Number of previously posted assignments by Teacher

In [40]:

```
#https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.Normalizer.html
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()

normalizer.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(1,-1)

number_projects_train = normalizer.transform(X_train['teacher_number_of_previously_posted_p
number_projects_cv = normalizer.transform(X_cv['teacher_number_of_previously_posted_project
number_projects_test = normalizer.transform(X_test['teacher_number_of_previously_posted_pro

print("After vectorizations")
print(number_projects_train.shape, y_train.shape)
print(number_projects_cv.shape, y_cv.shape)
print(number_projects_test.shape, y_test.shape)
```

```
After vectorizations
(1, 22445) (22445,)
(1, 11055) (11055,)
(1, 16500) (16500,)
```

In [41]:

```
number_projects_train
```

Out[41]:

```
array([[0.00244294, 0.        , 0.00024429, ..., 0.00244294, 0.00073288,
        0.        ]])
```

In [42]:

```
number_projects_train = np.reshape(number_projects_train, (-1, 1))
number_projects_cv = np.reshape(number_projects_cv, (-1, 1))
number_projects_test = np.reshape(number_projects_test, (-1, 1))
```

Price

In [43]:

```
#https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.Normalizer.html
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()

normalizer.fit(X_train['price'].values.reshape(1,-1))

price_train = normalizer.transform(X_train['price'].values.reshape(1,-1))
price_cv = normalizer.transform(X_cv['price'].values.reshape(1,-1))
price_test = normalizer.transform(X_test['price'].values.reshape(1,-1))

print("After vectorizations")
print(price_train.shape, y_train.shape)
print(price_cv.shape, y_cv.shape)
print(price_test.shape, y_test.shape)
```

```
After vectorizations
(1, 22445) (22445,)
(1, 11055) (11055,)
(1, 16500) (16500,)
```

In [44]:

```
price_train=np.reshape(price_train, (-1, 1))
price_cv=np.reshape(price_cv, (-1, 1))
price_test=np.reshape(price_test, (-1, 1))
```

Resource quantity

In [45]:

```
#https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.Normalizer.html
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()

normalizer.fit(X_train['quantity'].values.reshape(1,-1))

quantity_train = normalizer.transform(X_train['quantity'].values.reshape(1,-1))
quantity_cv = normalizer.transform(X_cv['quantity'].values.reshape(1,-1))
quantity_test = normalizer.transform(X_test['quantity'].values.reshape(1,-1))

print("After vectorizations")
print(quantity_train.shape, y_train.shape)
print(quantity_cv.shape, y_cv.shape)
print(quantity_test.shape, y_test.shape)
```

```
After vectorizations
(1, 22445) (22445,)
(1, 11055) (11055,)
(1, 16500) (16500,)
```

In [46]:

```
quantity_train
```

Out[46]:

```
array([[0.00103749, 0.0002075 , 0.00788492, ..., 0.00207498, 0.00103749,
        0.00207498]])
```

In [47]:

```
quantity_train =np.reshape(quantity_train, (-1, 1))
quantity_cv =np.reshape(quantity_cv, (-1, 1))
quantity_test =np.reshape(quantity_test, (-1, 1))
```

Sentiment score

In [48]:

```
#https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.Normalizer.html

essay_sentiment_train = X_train['essay_sentiment'].values.reshape(1,-1)
essay_sentiment_cv = X_cv['essay_sentiment'].values.reshape(1,-1)
essay_sentiment_test = X_test['essay_sentiment'].values.reshape(1,-1)

essay_sentiment_train = np.reshape(essay_sentiment_train, (-1, 1))
essay_sentiment_cv = np.reshape(essay_sentiment_cv, (-1, 1))
essay_sentiment_test = np.reshape(essay_sentiment_test, (-1, 1))

print(essay_sentiment_train.shape,y_train.shape)
print(essay_sentiment_cv.shape ,y_cv.shape)
print(essay_sentiment_test.shape,  y_test.shape)
```

```
(22445, 1) (22445,)
(11055, 1) (11055,)
(16500, 1) (16500,)
```

Number of words in Project title

In [49]:

```
#https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.Normalizer.html
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()

normalizer.fit(X_train['title_size'].values.reshape(1,-1))

title_size_train = normalizer.transform(X_train['title_size'].values.reshape(1,-1))
title_size_cv = normalizer.transform(X_cv['title_size'].values.reshape(1,-1))
title_size_test = normalizer.transform(X_test['title_size'].values.reshape(1,-1))

print("After normalization")
print(title_size_train.shape, y_train.shape)
print(title_size_cv.shape, y_cv.shape)
print(title_size_test.shape, y_test.shape)
```

```
After normalization
(1, 22445) (22445,)
(1, 11055) (11055,)
(1, 16500) (16500,)
```

In [50]:

```
title_size_train =np.reshape(title_size_train, (-1, 1))
title_size_cv =np.reshape(title_size_cv, (-1, 1))
title_size_test =np.reshape(title_size_test, (-1, 1))

print(title_size_train.shape, y_train.shape)
print(title_size_cv.shape, y_cv.shape)
print(title_size_test.shape, y_test.shape)
```

```
(22445, 1) (22445,)
(11055, 1) (11055,)
(16500, 1) (16500,)
```

Number of words in combined Essay

In [51]:

```
#https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.Normalizer.html
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()

normalizer.fit(X_train['essay_size'].values.reshape(1,-1))

essay_size_train = normalizer.transform(X_train['essay_size'].values.reshape(1,-1))
essay_size_cv = normalizer.transform(X_cv['essay_size'].values.reshape(1,-1))
essay_size_test = normalizer.transform(X_test['essay_size'].values.reshape(1,-1))

essay_size_train =np.reshape(essay_size_train, (-1, 1))
essay_size_cv =np.reshape(essay_size_cv, (-1, 1))
essay_size_test =np.reshape(essay_size_test, (-1, 1))


print(essay_size_train.shape, y_train.shape)
print(essay_size_cv.shape, y_cv.shape)
print(essay_size_test.shape, y_test.shape)
```

```
(22445, 1) (22445,)
(11055, 1) (11055,)
(16500, 1) (16500,)
```

Concateinating train preprocessed essay and title

In [52]:

```
# merge two column text dataframe:
X_train["text_data"]  = X_train['clean_essays'] + ' ' + X_train['clean_titles']
```

In [53]:

```
text_data = X_train["text_data"]
text_data.head()
```

Out[53]:

```
39468    7th 8th grade dedicated group students parents...
8599     students best wayne county offer highly motiva...
12932    third grade students part integrated classroom...
18182    children class need good tools learn wonderful...
32790    work student preschool kindergarten age come h...
Name: text_data, dtype: object
```

Calculating IDF value for each unique word in text_data

In [54]:

```
#https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVec

from sklearn.feature_extraction.text import TfidfVectorizer

tfidf_vectorizer = TfidfVectorizer(min_df = 15)
text_idf_val = tfidf_vectorizer.fit_transform(text_data.values)

print("Shape of matrix after one hot encodig ",text_idf_val.shape)
```

```
Shape of matrix after one hot encodig  (22445, 7377)
```

Selecting top 2k features based on IDF values

In [55]:

```
index = np.argsort(tfidf_vectorizer.idf_)[ ::-1 ]

#get feature names
feature_names = tfidf_vectorizer.get_feature_names()

top_features = [feature_names[i] for i in index[:2000]]
idf_vals = [tfidf_vectorizer.idf_[i] for i in index[:2000]]

feature_idf_dict = list(zip(top_features, idf_vals))

print(feature_idf_dict[:10])
```

```
[('cant', 8.24627898133642), ('correlates', 8.24627898133642), ('underwate
r', 8.24627898133642), ('pod', 8.24627898133642), ('problematic', 8.24627898
133642), ('observant', 8.24627898133642), ('cooler', 8.24627898133642), ('we
lcomes', 8.24627898133642), ('adhere', 8.24627898133642), ('unfair', 8.24627
898133642)]
```

Making Co-occurance matrix with 2k keywords

In [56]:

```
#http://hongleixie.github.io/blog/NLP-words-cooc/

def get_co_matrix(text_data, features, window):
    #creating matrix of top features
    matrix = pd.DataFrame(np.zeros((len(features), len(features))), index=features, columns
    #taking each sentence in text data
    for sent in text_data:
        #splitting sentence into words
        word = sent.split()
        #iterating each word
        for ind in range(len(word)):
            if matrix.get(word[ind]) is None:
                continue
            #For the window size of "window" check the occurance of the word
            for i in range(1, window + 1):
                #to the left
                if ind - i >= 0:
                    if matrix.get(word[ind - i]) is not None:
                        matrix[word[ind-i]].loc[word[ind]] = (matrix.get(word[ind-i]).loc[
                        matrix[word[ind]].loc[word[ind-i]] = (matrix.get(word[ind]).loc[wor
                #to the right
                if ind + i < len(word):
                    if matrix.get(word[ind+i]) is not None:
                        matrix[word[ind+i]].loc[word[ind]] = (matrix.get(word[ind+i]).loc[
                        matrix[word[ind]].loc[word[ind+i]] = (matrix.get(word[ind]).loc[ wo

    #fill diagonal elements of the matrix with 0
    np.fill_diagonal(matrix.values, 0)
    matrix = matrix.div(2)
    return matrix
```

In [57]:

```
co_matrix = get_co_matrix(text_data, top_features, 5)
print(co_matrix.shape)
```

```
(2000, 2000)
```

In [58]:

```
co_matrix
```

Out[58]:

	cant	correlates	underwater	pod	problematic	observant	cooler	welcomes	adh
cant	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
correlates	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
underwater	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
pod	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
problematic	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
...	...	...	...	...	...	...	...	...	
silently	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
excessive	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
satisfaction	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
boosting	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
drawer	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	

2000 rows × 2000 columns

Constructing Co-occurance matrix of sample data

In [60]:

```
sample_top_features =  ['abc','pqr', 'def']
corpus = ['abc def ijk pqr', 'pqr klm opq', 'lmn pqr xyz abc def pqr abc']

Sample_data  = pd.Series(corpus)
Sample_data
```

Out[60]:

```
0                abc def ijk pqr
1                    pqr klm opq
2    lmn pqr xyz abc def pqr abc
dtype: object
```

In [61]:

```
sample_co_matrix = get_co_matrix(Sample_data, sample_top_features, 2)
sample_co_matrix
```

Out[61]:

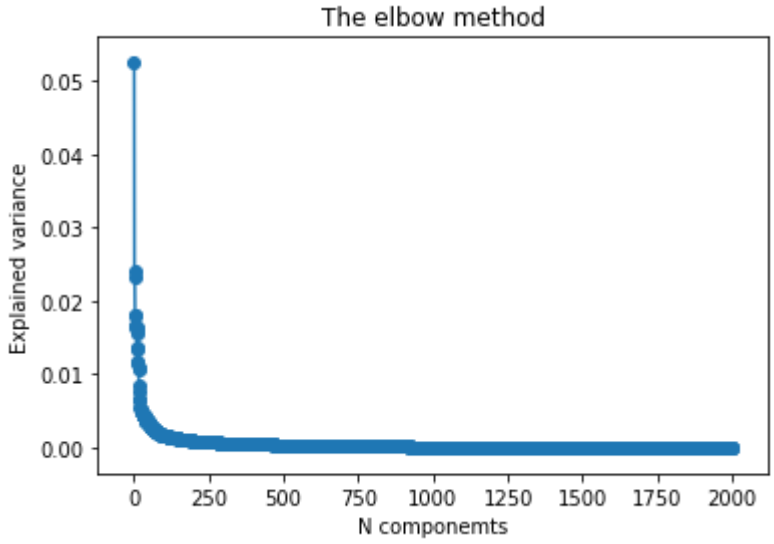
	abc	pqr	def
abc	0.0	3.0	3.0
pqr	3.0	0.0	2.0
def	3.0	2.0	0.0

Applying TSVD

Elbow/knee plot to find optimal number of n_components

In [62]:

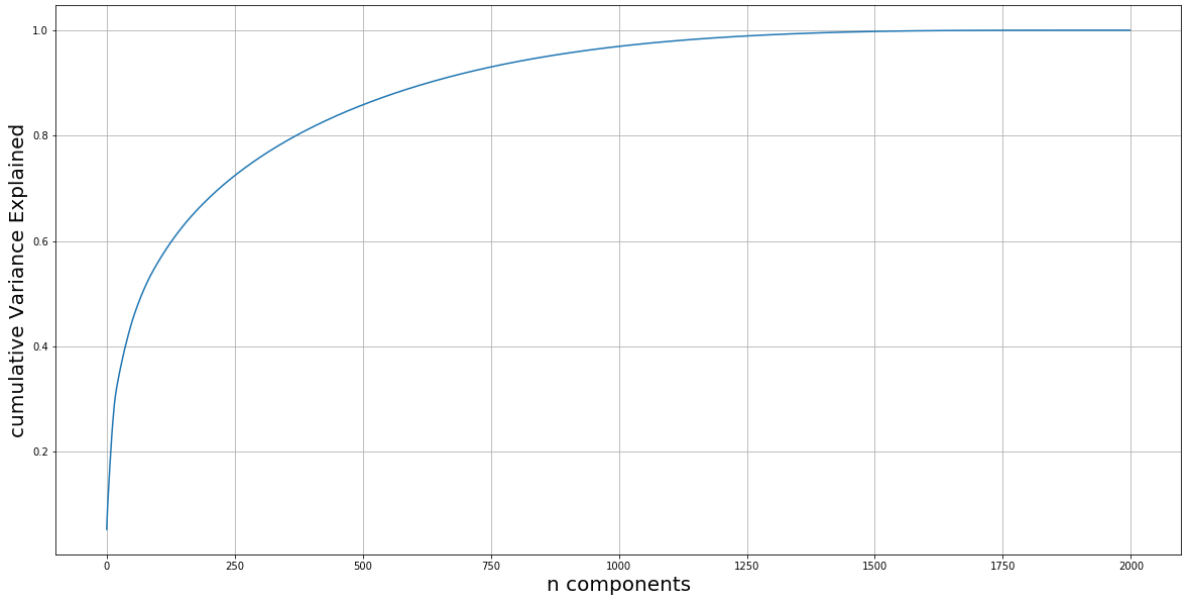
```
#https://chrisalbon.com/machine_learning/feature_engineering/select_best_number_of_componen

from sklearn.decomposition import TruncatedSVD
n_comp = list(range(1,2000))

tsvd = TruncatedSVD(n_components=1999, algorithm='randomized', n_iter=5, random_state=None,
tsvd.fit(co_matrix)
tsvd_var_ratio = tsvd.explained_variance_ratio_
tsvd_var_ratio = tsvd_var_ratio.tolist()

plt.plot(n_comp,tsvd_var_ratio, label='N component plot')
plt.scatter(n_comp,tsvd_var_ratio, label='N component points')
plt.title("The elbow method")
plt.xlabel("N componemts")
plt.ylabel("Explained variance")
plt.show()
```



Plotting CDF to find number of n_components that explains 95% explained variance

In [63]:

```
from numpy import cumsum

variance_explained = np.cumsum(tsvd_var_ratio)
plt.figure( figsize=(20,10))

plt.plot(variance_explained)
plt.grid()
plt.xlabel('n components',size = 20)
plt.ylabel('cumulative Variance Explained',size = 20)
plt.show()
```



Optimal number of n_components = 850

In [64]:

```
tsvd = TruncatedSVD(n_components=850, algorithm='randomized', n_iter=5, random_state=None,t
truncated = tsvd.fit_transform(co_matrix)

print(truncated.shape)
```

```
(2000, 850)
```

Text Vectorization using co-occurance matrix

In [65]:

```
tsvd_features = top_features[:850]

#Tsvd vector dictionary
tsvd_vec = {}
v = 0
index = list(co_matrix.index)
for i in truncated:
    tsvd_vec[index[v]] = i
    v += 1
```

Vectorizing Essay

In [66]:

```
train_essay_avg_w2v = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['clean_essays']): # for each review/sentence
    vector = np.zeros(850) # 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 tsvd_features:
            vec = tsvd_vec[word]  # Computing it's vector
            vector += vec     # Add it to the svec
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    train_essay_avg_w2v.append(vector)

print(len(train_essay_avg_w2v))
print(len(train_essay_avg_w2v[0]))


cv_essay_avg_w2v = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['clean_essays']): # for each review/sentence
    vector = np.zeros(850) # 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 tsvd_features:
            vec = tsvd_vec[word]  # Computing it's vector
            vector += vec     # Add it to the svec
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    cv_essay_avg_w2v.append(vector)

print(len(cv_essay_avg_w2v))
print(len(cv_essay_avg_w2v[0]))


test_essay_avg_w2v = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['clean_essays']): # for each review/sentence
    vector = np.zeros(850) # 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 tsvd_features:
            vec = tsvd_vec[word]  # Computing it's vector
            vector += vec     # Add it to the svec
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    test_essay_avg_w2v.append(vector)

print(len(test_essay_avg_w2v))
print(len(test_essay_avg_w2v[0]))
```

```
100%|████████████████████████████████████████████████████████████████████████████
████| 22445/22445 [00:51<00:00, 438.38it/s]

22445
850

100%|████████████████████████████████████████████████████████████████████████████
████| 11055/11055 [00:25<00:00, 436.50it/s]
```

```
11055
850

100%|██████████████████████████████████████████████████████████████████
████| 16500/16500 [00:37<00:00, 439.37it/s]

16500
850
```

In [67]:

```
# Changing list to numpy arrays
train_essay_avg_w2v = np.array(train_essay_avg_w2v)
cv_essay_avg_w2v = np.array(cv_essay_avg_w2v)
test_essay_avg_w2v = np.array(test_essay_avg_w2v)
```

vectorizing Title

In [68]:

```
train_titles_avg_w2v = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['clean_titles']): # for each review/sentence
    vector = np.zeros(850) # 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 tsvd_features:
            vec = tsvd_vec[word]  # Computing it's vector
            vector += vec     # Add it to the svec
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    train_titles_avg_w2v.append(vector)

print(len(train_titles_avg_w2v))
print(len(train_titles_avg_w2v[0]))


cv_titles_avg_w2v = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['clean_titles']): # for each review/sentence
    vector = np.zeros(850) # 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 tsvd_features:
            vec = tsvd_vec[word]  # Computing it's vector
            vector += vec     # Add it to the svec
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    cv_titles_avg_w2v.append(vector)

print(len(cv_titles_avg_w2v))
print(len(cv_titles_avg_w2v[0]))


test_titles_avg_w2v = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['clean_titles']): # for each review/sentence
    vector = np.zeros(850) # 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 tsvd_features:
            vec = tsvd_vec[word]  # Computing it's vector
            vector += vec     # Add it to the svec
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    test_titles_avg_w2v.append(vector)

print(len(test_titles_avg_w2v))
print(len(test_titles_avg_w2v[0]))
```

```
100%|████████████████████████████████████████████████████████████████
██| 22445/22445 [00:01<00:00, 14260.76it/s]

22445
850

100%|████████████████████████████████████████████████████████████████
██| 11055/11055 [00:00<00:00, 13837.00it/s]
```

```
11055
850

100%|██████████████████████████████████████████████████████████████
██| 16500/16500 [00:01<00:00, 12709.24it/s]

16500
850
```

In [69]:

```
# Changing list to numpy arrays
train_title_avg_w2v = np.array(train_titles_avg_w2v)
cv_title_avg_w2v = np.array(cv_titles_avg_w2v)
test_title_avg_w2v = np.array(test_titles_avg_w2v)
```

Applying GBDT

Hstacking features

In [70]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)+
X_train = hstack((X_train_categories_one_hot, X_train_sub_categories_one_hot, train_essay_a
X_cv = hstack((X_cv_categories_one_hot, X_cv_sub_categories_one_hot, cv_essay_avg_w2v, quar
X_test = hstack((X_test_categories_one_hot, X_test_sub_categories_one_hot, test_essay_avg_w

print('Final matrix')
print(X_train.shape, y_train.shape)
print(X_cv.shape, y_cv.shape)
print(X_test.shape, y_test.shape)
```

```
Final matrix
(22445, 1805) (22445,)
(11055, 1805) (11055,)
(16500, 1805) (16500,)
```

Hyperparameter tuning (n_estimators and miax_depth)

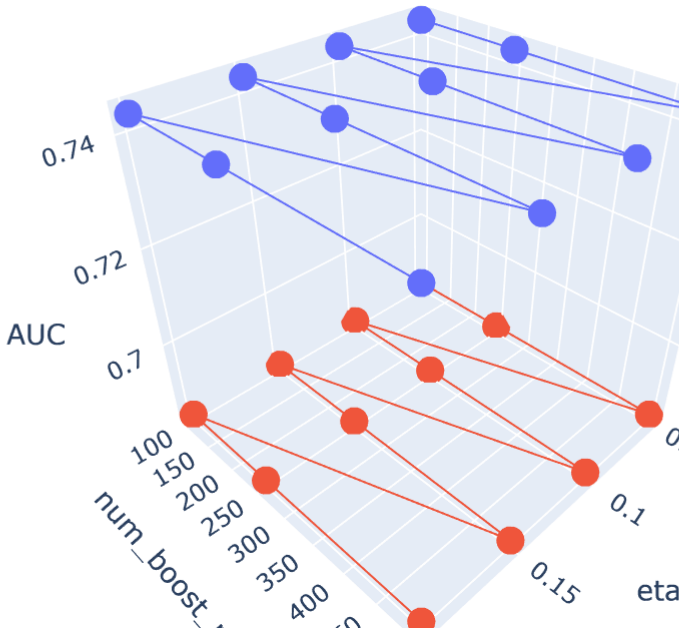
In [72]:

```
#https://scikit-learn.org/stable/modules/generated/sklearn.calibration.CalibratedClassifier
import xgboost as xgb
import matplotlib.pyplot as plt
from sklearn.metrics import roc_auc_score
import math


train_auc = []
cv_auc = []

eta = [0.05, 0.1, 0.15, 0.2]
num_boost_round =  [100, 250, 500]

for i in num_boost_round:
    for j in eta:
        xg = xgb.XGBClassifier(eta=j, num_boost_round=i , class_weight='balanced')
        xg.fit(X_train, y_train)


        y_train_pred = xg.predict_proba(X_train)[:,1]
        y_cv_pred = xg.predict_proba(X_cv)[:,1]


        # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates
        # not the predicted outputs
        train_auc.append(roc_auc_score(y_train,y_train_pred))
        cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
```

AUC 3D Scatter plot

In [73]:

```
#https://pythonprogramming.net/3d-scatter-plot-customizing/?completed=/matplotlib-3d-scatte

num_boost_round_val = []
eta_val = []
for i in (eta):
    for j in (num_boost_round):
        eta_val.append(i)
        num_boost_round_val.append(j)

trace1 = go.Scatter3d(x=eta_val,y=num_boost_round_val,z=train_auc, name = 'train auc')
trace2 = go.Scatter3d(x=eta_val,y=num_boost_round_val,z=cv_auc, name = 'cv auc')
data = [trace1, trace2]
layout = go.Layout(scene = dict(xaxis = dict(title='eta'),
                                yaxis = dict(title='num_boost_round'),
                                zaxis = dict(title='AUC'),))
fig = go.Figure(data=data, layout=layout)
offline.iplot(fig, filename='3d-scatter-auc error')
```

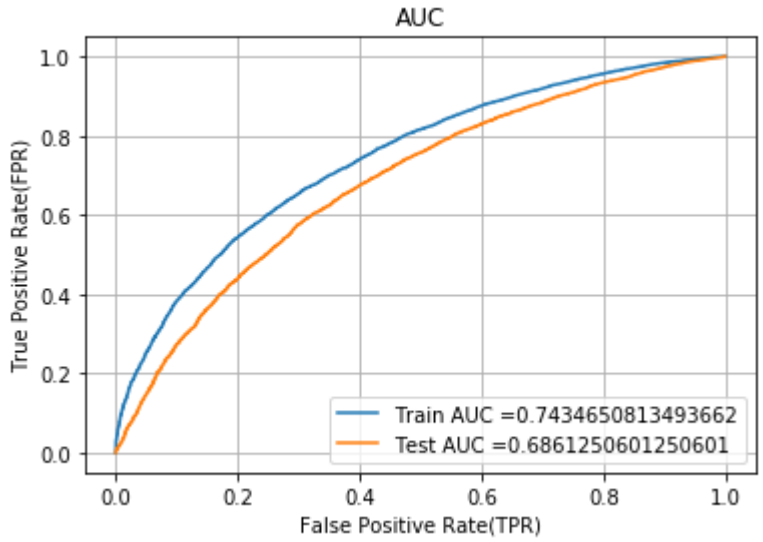


optimal eta = 0.05 || num_boost_round = 200

Training model with best set of hyperparameters

In [77]:

```
xg = xgb.XGBClassifier(eta=0.05 ,num_boost_round =200 ,class_weight='balanced')

xg.fit(X_train, y_train)

y_train_pred = xg.predict_proba(X_train)[:,1]
y_test_pred = xg.predict_proba(X_test)[:,1]

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

plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("False Positive Rate(TPR)")
plt.ylabel("True Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
```



Confusion matrix

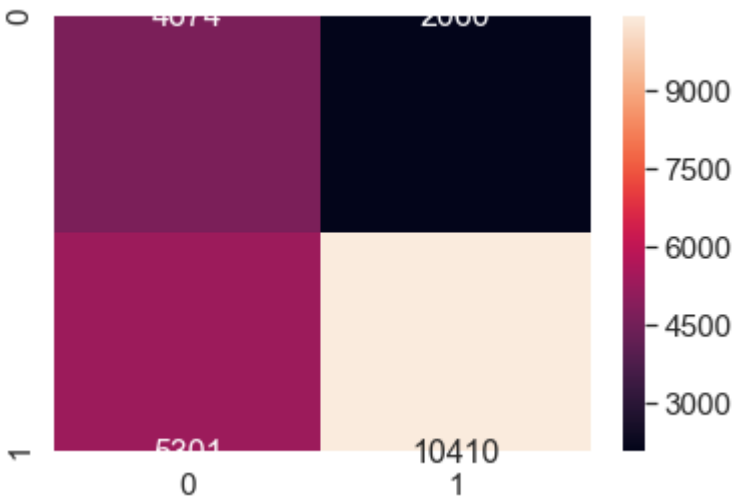
In [78]:

```
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def find_best_threshold(threshould, fpr, tpr):
    t = threshould[np.argmax(tpr*(1-fpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t
    return t

def predict_with_best_t(proba, threshould):
    predictions = []
    for i in proba:
        if i>=threshould:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions

from mlxtend.plotting import plot_confusion_matrix
import matplotlib.pyplot as plt
import numpy as np
```

In [79]:

```
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
df_cmtr = pd.DataFrame(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t))
sns.set(font_scale=1.4)#for label size
sns.heatmap(df_cmtr, annot=True,annot_kws={"size": 16}, fmt='g')
print(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t)))
```

```
the maximum value of tpr*(1-fpr) 0.45989903350987943 for threshold 0.699
Train confusion matrix
[[ 4674  2060]
 [ 5301 10410]]
```



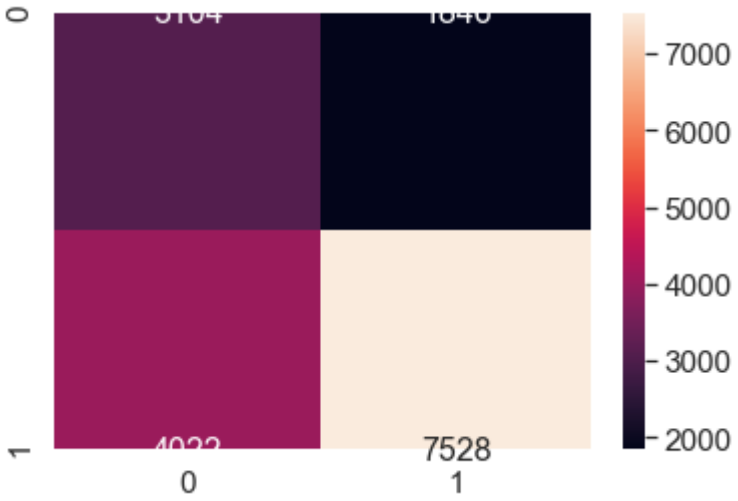
In [80]:

```
best_t = find_best_threshold(te_thresholds, test_fpr, test_tpr)
print("Test confusion matrix")
df_cmte = pd.DataFrame(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)),r
sns.set(font_scale=1.4)#for label size
sns.heatmap(df_cmte, annot=True,annot_kws={"size": 16}, fmt='g')
confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t))
```

```
the maximum value of tpr*(1-fpr) 0.408708942236215 for threshold 0.708
Test confusion matrix
```

Out[80]:

```
array([[3104, 1846],
       [4022, 7528]], dtype=int64)
```



Evaluating model's performance

In [81]:

```
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
from sklearn.metrics import recall_score
y_pred_new = xg.predict(X_test)
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test, y_pred_new)*100))
print("Precision on test set: %0.3f"%(precision_score(y_test, y_pred_new)))
print("Recall on test set: %0.3f"%(recall_score(y_test, y_pred_new)))
print("F1-Score on test set: %0.3f"%(f1_score(y_test, y_pred_new)))
```

```
Accuracy on test set: 71.121%
Precision on test set: 0.716
Recall on test set: 0.973
F1-Score on test set: 0.825
```

1. First top 2000 features from Train essay + train title were selected based on the idf value.
2. Co_occurance matrix of top 2000 features was constructed to find the relation between the words.
3. Truncated SVD applied on the co_occurance matrix...and from that 850 features were lasted after truncation that explains ~95% -explained varinace.
4. Avg W2V vectorization was done on the text data using those 850 features.
5. All the features (Categorical, numerical and text) were hstacked and XGBoostClassifier is applied

----------From the model----------

1. Best num_boost_round =
2. Best eta = 0.05
3. Train AUC = 0.74
4. Test AUC = 0.685
5. Accuracy on test set 71.15%
6. F1 Score 0.823