SVD Algorithm on Donors_Choose dataset

In [1]:
```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
import sqlite3
import pandas as pd
import numpy as np
import nltk
import gc
gc.enable()
gc.DEBUG_SAVEALL
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
import math
from plotly import plotly
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
from collections import Counter
gc.set_threshold(2, 1, 1)
```

2.1 Loading Input Data

In [2]:
```
# %load_ext memory_profiler


s=0
# We are taking samples of 0's and 1's and appending them to overcome memory erro


project_data = pd.read_csv('train_data.csv')
# project_data=project_data.dropna(how='any')
project_data=project_data.fillna("")
project_data_1 = project_data[project_data['project_is_approved'] == s+1]
project_data_0 = project_data[project_data['project_is_approved'] == s]

project_data_1=project_data_1.head(12000)
project_data_0=project_data_0.tail(8000)
project_data_1=project_data_1.append(project_data_0)
project_data=project_data_1
resource_data = pd.read_csv('resources.csv')

#Sorting them by columns to spread the zeros and one's unevenly in the 'project_
project_data.sort_values(by=['project_essay_1'])
project_data.sort_values(by=['project_essay_2'], ascending=False)
project_data.sort_values(by=['project_essay_3'])
project_data.sort_values(by=['project_essay_4'], ascending=False)
project_data_1=None
project_data_0=None
```

In [3]:
```
print("Number of data points in train data", project_data.shape)
print('-'*50)
print("The attributes of data :", project_data.columns.values)
```

```
Number of data points in train data (20000, 17)
--------------------------------------------------
The attributes of data : ['Unnamed: 0' 'id' 'teacher_id' 'teacher_prefix' 'scho
ol_state'
 'project_submitted_datetime' 'project_grade_category'
 'project_subject_categories' 'project_subject_subcategories'
 'project_title' 'project_essay_1' 'project_essay_2' 'project_essay_3'
 'project_essay_4' 'project_resource_summary'
 'teacher_number_of_previously_posted_projects' 'project_is_approved']
```

In [4]:
```
print("Number of data points in resource data", resource_data.shape)
print(resource_data.columns.values)
resource_data.head(1)
# project_data.head(2)
```

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

Out[4]:

	id	description	quantity	price
0	p233245	LC652 - Lakeshore Double-Space Mobile Drying Rack	1	149.0

In [5]:
```
y = project_data['project_is_approved'].values
X = project_data.drop(['project_is_approved'], axis=1)
X.head(1)
project_data=None
gc.collect()
gc.enable()
gc.DEBUG_SAVEALL
```

Out[5]: 32

2.2 Getting the Data Model Ready:Preprocessing and Vectorizing categorical features

2.2.1 Preprocessing:project_grade_category

In [6]:
```
sub_catogories = list(X['project_grade_category'].values)
# remove special characters from list of strings python: https://stackoverflow.co

# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-1

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",
        if 'The' in j.split(): # this will split each of the catogory based on sp
            j=j.replace('The','') # if we have the words "The" we are going to re
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty)
        temp +=j.strip()+" "#" abc ".strip() will return "abc", remove the trail
        temp = temp.replace('&','_')
    sub_cat_list.append(temp.strip())
X['project_grade_category'] = sub_cat_list
```

In [7]:
```
sub_catogories=None
sub_cat_list=None
temp=None
i=None
j=None
catogories=None
cat_list=None
temp=None
my_counter=None
word=None
cat_dict=None
gc.collect()
gc.enable()
gc.DEBUG_SAVEALL
```

Out[7]: 32

2.2.2 Preprocessing:project_subject_categories

In [8]:
```
catogories = list(X['project_subject_categories'].values)
# remove special characters from list of strings python: https://stackoverflow.co

# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-
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",
        if 'The' in j.split(): # this will split each of the catogory based on sp
            j=j.replace('The','') # if we have the words "The" we are going to re
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty)
        temp+=j.strip()+" " #" abc ".strip() will return "abc", remove the trail
        temp = temp.replace('&','_') # we are replacing the & value into
    cat_list.append(temp.strip())

X['clean_categories'] = cat_list
X.drop(['project_subject_categories'], axis=1, inplace=True)
X.head(2)
```

Out[8]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_sul
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL	
3	45	p246581	f3cb9bffbba169bef1a77b243e620b60	Mrs.	KY	

2.2.3 Preprocessing:project_subject_subcategories

In [9]:
```
sub_catogories = list(X['project_subject_subcategories'].values)
# remove special characters from list of strings python: https://stackoverflow.co

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

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",
        if 'The' in j.split(): # this will split each of the catogory based on sp
            j=j.replace('The','') # if we have the words "The" we are going to re
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty)
        temp +=j.strip()+" "#" abc ".strip() will return "abc", remove the trail
        temp = temp.replace('&','_')
    sub_cat_list.append(temp.strip())
```

In [10]:
```
X['clean_subcategories'] = sub_cat_list
X.drop(['project_subject_subcategories'], axis=1, inplace=True)
X.head(2)
```

Out[10]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_sul
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL	
3	45	p246581	f3cb9bffbba169bef1a77b243e620b60	Mrs.	KY	

2.2.4 New Column:digits in summary

In [11]:
```
# Creating a new column 'digits_in_summary' which contains flags of 1 for /
# 'project_resource_summary' containing numeric specification in their requiremn
project_resource_summary = []
new=[]

project_resource_summary = list(X['project_resource_summary'].values)

for i in project_resource_summary:
    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
    for j in i.split(' '):
        if j.isdigit():
            new.append(1)
            break
        else:
            continue
    else:
        new.append(0)


X['digits_in_summary']=new
```

In [12]:
```
#To make best use of the memory we are setting the variable names to 'None' and p
project_resource_summary=None
new=None
new1=None
i=None
j=None
a=None

gc.collect()
gc.enable()
gc.DEBUG_SAVEALL
```

Out[12]: 32

2.2.5 Preprocessing:Text features (Project Essay's)

In [13]:
```
X['project_essay_1'] = X['project_essay_1'].fillna('')
X['project_essay_2'] = X['project_essay_2'].fillna('')
X['project_essay_3'] = X['project_essay_3'].fillna('')
X['project_essay_4'] = X['project_essay_4'].fillna('')
```

In [14]:
```
# merge two column text dataframe:
X["essay"] = X["project_essay_1"].map(str) +\
                        X["project_essay_2"].map(str) + \
                        X["project_essay_3"].map(str) + \
                        X["project_essay_4"].map(str)
```

In [15]:
```
X = X.drop(['project_essay_1', 'project_essay_2','project_essay_3', 'project_ess
X.shape
```

Out[15]:
```
(20000, 14)
```

2.2.6 Adding column Cost per project in dataset

In [16]:
```
# https://stackoverflow.com/questions/22407798/how-to-reset-a-dataframes-indexes
price_data = resource_data.groupby('id').agg({'price':'sum', 'quantity':'sum'}).
price_data.head(2)
type(price_data)
```

Out[16]:
```
pandas.core.frame.DataFrame
```

In [17]:
```
# join two dataframes in python:
X = pd.merge(X, price_data, on='id', how='left')
X.head(2)
```

Out[17]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_sul
0	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL	
1	45	p246581	f3cb9bffbba169bef1a77b243e620b60	Mrs.	KY	

In [18]:
```
#To make best use of the memory we are setting the variable names to 'None' and 
resource_data=None
price_data=None
gc.collect()
gc.enable()
gc.DEBUG_SAVEALL
```

Out[18]:
```
32
```

2.2.7 Text Preprocessing:Essay Text

In [19]:
```
# 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 [20]:
```
sent = decontracted(X['essay'].values[99])
print(sent)
print("="*50)
```

```
What does \"The message in the music\" mean to you? To my kids, it means so muc
h more! Music is not just for entertainment!  My kids are learning how music ca
n be used as a human rights tool! They are learning to express their feelings o
f the world around them in positive ways.\r\n\r\nMy kids are every stereotype y
ou can think of: poor, underprivileged, rough, angry.  Turn that same prism aro
und there is also joy, happiness, intelligence and creativity. They are learnin
g to navigate the world around them and we are working on showing them how to u
se this energy in a positive way.Our piano lab is aging.  We have 3 classes of
35 kids that are bursting at the gills, which is a great problem to have!  BUT
keyboards are beginning to have the keys break off making them unuseable. We al
so do not have enough keyboards for every student in class, even if you counted
the slightly broken ones!\r\n\r\nWe are in desperate need of more keyboards for
our labs to meet the needs of our students.  A keyboard for each student in cla
ss will facilitate their love of music and help them grow and appreciate the wo
rld around them.  We are down to 24 semi working keyboards and 35 kids in each
piano class.  These keyboards are needed sooner rather than later for our kids
to have a successful year in class.
==================================================
```

In [21]:
```
# \r \n \t remove from string python: http://texthandler.com/info/remove-line-bre
sent = sent.replace('\\r', ' ')
sent = sent.replace('\\"', ' ')
sent = sent.replace('\\n', ' ')
print(sent)
```

```
What does  The message in the music  mean to you? To my kids, it means so much
more! Music is not just for entertainment!  My kids are learning how music can
be used as a human rights tool! They are learning to express their feelings of
the world around them in positive ways.    My kids are every stereotype you can
think of: poor, underprivileged, rough, angry.  Turn that same prism around the
re is also joy, happiness, intelligence and creativity. They are learning to na
vigate the world around them and we are working on showing them how to use this
energy in a positive way.Our piano lab is aging.  We have 3 classes of 35 kids
that are bursting at the gills, which is a great problem to have!  BUT keyboard
s are beginning to have the keys break off making them unuseable. We also do no
t have enough keyboards for every student in class, even if you counted the sli
ghtly broken ones!    We are in desperate need of more keyboards for our labs t
o meet the needs of our students.  A keyboard for each student in class will fa
cilitate their love of music and help them grow and appreciate the world around
them.  We are down to 24 semi working keyboards and 35 kids in each piano clas
s.  These keyboards are needed sooner rather than later for our kids to have a
successful year in class.
```

In [22]:
```
#remove spacial character: https://stackoverflow.com/a/5843547/4084039
sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
print(sent)
```

```
What does The message in the music mean to you To my kids it means so much more
Music is not just for entertainment My kids are learning how music can be used
as a human rights tool They are learning to express their feelings of the world
around them in positive ways My kids are every stereotype you can think of poor
underprivileged rough angry Turn that same prism around there is also joy happi
ness intelligence and creativity They are learning to navigate the world around
them and we are working on showing them how to use this energy in a positive wa
y Our piano lab is aging We have 3 classes of 35 kids that are bursting at the
gills which is a great problem to have BUT keyboards are beginning to have the
keys break off making them unuseable We also do not have enough keyboards for e
very student in class even if you counted the slightly broken ones We are in de
sperate need of more keyboards for our labs to meet the needs of our students A
keyboard for each student in class will facilitate their love of music and help
them grow and appreciate the world around them We are down to 24 semi working k
eyboards and 35 kids in each piano class These keyboards are needed sooner rath
er than later for our kids to have a successful year in class
```

In [23]:
```
# 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'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he',
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itse
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'tha
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'ha
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 't
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'of
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than'
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should'v
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "d
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma'
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn
            'won', "won't", 'wouldn', "wouldn't"]
```

In [24]:
```
# Combining all the above statemennts
from tqdm import tqdm
preprocessed_essays = []
# tqdm is for printing the status bar
for sentance in tqdm(X['essay'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e not in stopwords)
    preprocessed_essays.append(sent.lower().strip())
```

```
100%|████████████████████████████████████| 20000/20000 [00:41<00:00, 485.33it/s]
```

In [25]:
```
# after preprocesing

# X['essay'] = None
X['essay'] = preprocessed_essays

X.head(2)
```

Out[25]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_sul
0	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL	
1	45	p246581	f3cb9bffbba169bef1a77b243e620b60	Mrs.	KY	

In [26]:
```
# Combining all the above statemennts
from tqdm import tqdm
preprocessed_project_title = []
# tqdm is for printing the status bar
for sentance in tqdm(X['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 not in stopwords)
    preprocessed_project_title.append(sent.lower().strip())
```

100%|██████████| 20000/20000 [00:02<00:00, 8665.02it/s]

In [27]:
```
preprocessed_project_title[4999]
# after preprocesing

# X['project_title'] = None
X['project_title'] = preprocessed_project_title

X.head(2)
```

Out[27]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_sul
0	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL	
1	45	p246581	f3cb9bffbba169bef1a77b243e620b60	Mrs.	KY	

In [28]:
```
type(X['project_title'])
X['project_title'] = X['project_title'].fillna('')
project_count = X['project_title'].tolist()
A=[]
B=[]
for i in project_count:
    A=len(i.split())
    B.append(A)
print(len(B))
X['project_title_count'] = B
# X.head(5)
# X['essay']
# project_count
```

```
20000
```

In [29]:
```
type(X['essay'])
X['essay'] = X['essay'].fillna('')
project_count = X['essay'].tolist()
A=[]
B=[]
for i in project_count:
    A=len(i.split())
    B.append(A)
print(len(B))
X['essay_count'] = B
# X.head(5)
```

```
20000
```

In [30]:
```
# https://medium.com/analytics-vidhya/simplifying-social-media-sentiment-analysis
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
analyser = SentimentIntensityAnalyzer()
essay_ = X['essay'].tolist()
essay_score=[]
for i in essay_:
    score = analyser.polarity_scores(i)
    essay_score.append(score)
```

In [31]:
```
essay_score[0]
```

Out[31]:
```
{'neg': 0.049, 'neu': 0.667, 'pos': 0.285, 'compound': 0.9856}
```

In [32]:
```
for i in essay_score:
    X['neg']=i['neg']
    X['neu']=i['neu']
    X['pos']=i['pos']
    X['compound']=i['compound']
```

In [33]:
```
# merge two column text dataframe:
X["essay_pr_title"] = X["essay"].map(str) +\
                        X["project_title"].map(str)
X.head(2)
```

Out[33]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_
0	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL	
1	45	p246581	f3cb9bffbba169bef1a77b243e620b60	Mrs.	KY	

2 rows × 23 columns

In [34]:
```
# X = X.drop(["project_title"], axis=1)
# X.shape
```

In [35]:
```
# train test split(67:33)
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, strati
# X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.
```

In [36]:
```
# count of all the words in corpus python: https://stackoverflow.com/a/22898595/
from collections import Counter
my_counter = Counter()
for word in X_train['clean_categories'].values:
    my_counter.update(word.split())
print(my_counter)
# dict sort by value python: https://stackoverflow.com/a/613218/4084039
cat_dict = dict(my_counter)
sorted_cat_dict_train = dict(sorted(cat_dict.items(), key=lambda kv: kv[1]))
```

```
Counter({'Literacy_Language': 6269, 'Math_Science': 5169, 'Health_Sports': 174
7, 'SpecialNeeds': 1710, 'AppliedLearning': 1513, 'Music_Arts': 1269, 'History_
Civics': 692, 'Warmth': 144, 'Care_Hunger': 144})
```

2.2.9 Vectorizing Categorical data: clean_categories(Project subject categories)

In [37]:
```
print(X_train.shape, y_train.shape)
# print(X_cv.shape, y_cv.shape)
print(X_test.shape, y_test.shape)

print("="*100)

Bow_features_names1=[]

from sklearn.feature_extraction.text import CountVectorizer

vectorizer = CountVectorizer(vocabulary=list(sorted_cat_dict_train.keys()), lowe
vectorizer.fit(X_train['clean_categories'].values) # fit has to happen only on t

# we use the fitted Countvectorizer to convert the text to vector
X_train_clean_cat_ohe = vectorizer.transform(X_train['clean_categories'].values)
# X_cv_clean_cat_ohe = vectorizer.transform(X_cv['clean_categories'].values)
X_test_clean_cat_ohe = vectorizer.transform(X_test['clean_categories'].values)

print("After vectorizations")
print(X_train_clean_cat_ohe.shape, y_train.shape)
# print(X_cv_clean_cat_ohe.shape, y_cv.shape)
print(X_test_clean_cat_ohe.shape, y_test.shape)

print(vectorizer.get_feature_names())
# print(vectorizer_test.get_feature_names())
print("="*100)
```

```
(13400, 23) (13400,)
(6600, 23) (6600,)
============================================================================
====================
After vectorizations
(13400, 9) (13400,)
(6600, 9) (6600,)
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'S
pecialNeeds', 'Health_Sports', 'Math_Science', 'Literacy_Language']
============================================================================
====================
```

2.2.10 Vectorizing Categorical data: clean_subcategories(Project subject subcategories)

In [38]:
```
# count of all the words in corpus python: https://stackoverflow.com/a/22898595/4
from collections import Counter
my_counter = Counter()
for word in X_train['clean_subcategories'].values:
    my_counter.update(word.split())
sub_cat_dict = dict(my_counter)
sorted_sub_cat_dict_train = dict(sorted(sub_cat_dict.items(), key=lambda kv: kv[1
```

In [39]:
```
print(X_train.shape, y_train.shape)
# print(X_cv.shape, y_cv.shape)
print(X_test.shape, y_test.shape)

print("="*100)

vectorizer = CountVectorizer(vocabulary=list(sorted_sub_cat_dict_train.keys()),
vectorizer.fit(X_train['clean_subcategories'].values) # fit has to happen only o

# we use the fitted Countvectorizer to convert the text to vector
X_train_clean_sub_cat_ohe = vectorizer.transform(X_train['clean_subcategories'].
# X_cv_clean_sub_cat_ohe = vectorizer.transform(X_cv['clean_subcategories'].value
X_test_clean_sub_cat_ohe = vectorizer.transform(X_test['clean_subcategories'].va

print("After vectorizations")
print(X_train_clean_sub_cat_ohe.shape, y_train.shape)
# print(X_cv_clean_sub_cat_ohe.shape, y_cv.shape)
print(X_test_clean_sub_cat_ohe.shape, y_test.shape)

print(vectorizer.get_feature_names())
# print(vectorizer_test.get_feature_names())
print("="*100)
```

```
(13400, 23) (13400,)
(6600, 23) (6600,)
============================================================================
====================
After vectorizations
(13400, 30) (13400,)
(6600, 30) (6600,)
['Economics', 'FinancialLiteracy', 'CommunityService', 'ParentInvolvement', 'Ci
vics_Government', 'Extracurricular', 'ForeignLanguages', 'Warmth', 'Care_Hunge
r', 'NutritionEducation', 'SocialSciences', 'PerformingArts', 'CharacterEducati
on', 'Other', 'TeamSports', 'College_CareerPrep', 'Music', 'History_Geography',
'Health_LifeScience', 'ESL', 'EarlyDevelopment', 'Gym_Fitness', 'EnvironmentalS
cience', 'VisualArts', 'Health_Wellness', 'AppliedSciences', 'SpecialNeeds', 'L
iterature_Writing', 'Mathematics', 'Literacy']
============================================================================
====================
```

2.2.11 Vectorizing Categorical data: school_state

In [40]:
```
# count of all the words in corpus python: https://stackoverflow.com/a/22898595/4
from collections import Counter
my_counter = Counter()
for word in X_train['school_state'].values:
    my_counter.update(word.split())
school_state_dict = dict(my_counter)
sorted_school_state_dict_train = dict(sorted(school_state_dict.items(), key=lambd
# sorted_school_state_dict
```

In [41]:
```
print(X_train.shape, y_train.shape)
# print(X_cv.shape, y_cv.shape)
print(X_test.shape, y_test.shape)

print("="*100)


vectorizer = CountVectorizer(vocabulary=list(sorted_school_state_dict_train.keys
vectorizer.fit(X_train['school_state'].values) # fit has to happen only on train

# we use the fitted Countvectorizer to convert the text to vector
X_train_School_state_ohe = vectorizer.transform(X_train['school_state'].values)
# X_cv_School_state_ohe = vectorizer.transform(X_cv['school_state'].values)
X_test_School_state_ohe = vectorizer.transform(X_test['school_state'].values)

print("After vectorizations")
print(X_train_School_state_ohe.shape, y_train.shape)
# print(X_cv_School_state_ohe.shape, y_cv.shape)
print(X_test_School_state_ohe.shape, y_test.shape)

print(vectorizer.get_feature_names())
# print(vectorizer_test.get_feature_names())
print("="*100)
```

```
(13400, 23) (13400,)
(6600, 23) (6600,)
====================================================================================
====================
After vectorizations
(13400, 51) (13400,)
(6600, 51) (6600,)
['VT', 'WY', 'ND', 'MT', 'DE', 'SD', 'RI', 'NH', 'NE', 'AK', 'ME', 'WV', 'NM',
'HI', 'ID', 'DC', 'IA', 'KS', 'AR', 'CO', 'MN', 'MS', 'OR', 'KY', 'NV', 'MD',
'AL', 'CT', 'TN', 'WI', 'UT', 'VA', 'MA', 'NJ', 'WA', 'AZ', 'LA', 'IN', 'OH',
'OK', 'MO', 'PA', 'MI', 'SC', 'GA', 'IL', 'NC', 'FL', 'NY', 'TX', 'CA']
====================================================================================
====================
```

2.2.12 Vectorizing Categorical data: project_grade_category

In [42]:
```
# count of all the words in corpus python: https://stackoverflow.com/a/22898595/4
from collections import Counter
my_counter = Counter()
for word in X_train['project_grade_category'].values:
    my_counter.update(word.split())
project_grade_dict = dict(my_counter)
sorted_project_grade_dict_train = dict(sorted(project_grade_dict.items(), key=la
```

In [43]:
```
print(X_train.shape, y_train.shape)
# print(X_cv.shape, y_cv.shape)
print(X_test.shape, y_test.shape)

print("="*100)

vectorizer= CountVectorizer(vocabulary=list(sorted_project_grade_dict_train.keys
vectorizer.fit(X_train['project_grade_category'].values) # fit has to happen only

# we use the fitted Countvectorizer_pro_gradeto convert the text to vector
X_train_project_grade_category_ohe = vectorizer.transform(X_train['project_grade_
# X_cv_project_grade_category_ohe = vectorizer.transform(X_cv['project_grade_cate
X_test_project_grade_category_ohe = vectorizer.transform(X_test['project_grade_ca


print("After vectorizations")
print(X_train_project_grade_category_ohe.shape, y_train.shape)
# print(X_cv_project_grade_category_ohe.shape, y_cv.shape)
print(X_test_project_grade_category_ohe.shape, y_test.shape)

print(vectorizer.get_feature_names())
# print(vectorizer_test.get_feature_names())
print("="*100)
```

```
(13400, 23) (13400,)
(6600, 23) (6600,)
====================================================================================
====================
After vectorizations
(13400, 4) (13400,)
(6600, 4) (6600,)
['Grades9-12', 'Grades6-8', 'Grades3-5', 'GradesPreK-2']
====================================================================================
====================
```

2.2.13 Vectorizing Categorical data: teacher_prefix

In [44]:
```
#To overcome the blanks in the teacher_prefix categry the .fillna is used
X_train['teacher_prefix']=X_train['teacher_prefix'].fillna("")
# project_data1=project_data.dropna()
```

In [45]:
```
#To overcome the blanks in the teacher_prefix categry the .fillna is used
X_test['teacher_prefix']=X_test['teacher_prefix'].fillna("")
# project_data1=project_data.dropna()
```

In [46]:
```
# count of all the words in corpus python: https://stackoverflow.com/a/22898595/4
from collections import Counter
my_counter = Counter()
my_counter1=[]
# project_data['teacher_prefix']=str(project_data['teacher_prefix'])
for word in X_train['teacher_prefix'].values:
    my_counter.update(word.split())
teacher_prefix_dict = dict(my_counter)
sorted_teacher_prefix_dict_train = dict(sorted(teacher_prefix_dict.items(), key=l
# teacher_prefix_dict
```

In [47]:
```
print(X_train.shape, y_train.shape)
# print(X_cv.shape, y_cv.shape)
print(X_test.shape, y_test.shape)

print("="*100)

vectorizer = CountVectorizer(vocabulary=list(sorted_teacher_prefix_dict_train.key
vectorizer.fit(X_train['teacher_prefix'].values) # fit has to happen only on tra

# we use the fitted Countvectorizer to convert the text to vector
X_train_teacher_prefix_ohe = vectorizer.transform(X_train['teacher_prefix'].valu
# X_cv_teacher_prefix_ohe = vectorizer.transform(X_cv['teacher_prefix'].values)
X_test_teacher_prefix_ohe = vectorizer.transform(X_test['teacher_prefix'].values

print("After vectorizations")
print(X_train_teacher_prefix_ohe.shape, y_train.shape)
# print(X_cv_teacher_prefix_ohe.shape, y_cv.shape)
print(X_test_teacher_prefix_ohe.shape, y_test.shape)

print(vectorizer.get_feature_names())
# print(vectorizer_test.get_feature_names())
print("="*100)
```

```
(13400, 23) (13400,)
(6600, 23) (6600,)
====================================================================================
====================
After vectorizations
(13400, 5) (13400,)
(6600, 5) (6600,)
['Dr.', 'Teacher', 'Mr.', 'Ms.', 'Mrs.']
====================================================================================
====================
```

2.3 Make Data Model Ready: Vectorizing Numerical features

2.3.1 Vectorizing Numerical features--Price

In [48]:
```
from sklearn.preprocessing import Normalizer
from sklearn.preprocessing import StandardScaler
normalizer = StandardScaler() #Normalizer()
# normalizer_test = Normalizer()
# normalizer.fit(X_train['price'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.
normalizer.fit(X_train['price'].values.reshape(-1,1))
# normalizer_test.fit(X_test['price'].values.reshape(1,-1))

X_train_price_norm = normalizer.transform(X_train['price'].values.reshape(-1,1))
# X_cv_price_norm = normalizer.transform(X_cv['price'].values.reshape(-1,1))
X_test_price_norm = normalizer.transform(X_test['price'].values.reshape(-1,1))


# X_train_price_norm=np.reshape(X_train_price_norm,(1,-1))
# X_test_price_norm=np.reshape(X_test_price_norm,(1,-1))

print("After vectorizations")

# np.reshape(X_train_price_norm,
print(X_train_price_norm.shape, y_train.shape)
# print(X_cv_price_norm.shape, y_cv.shape)
print(X_test_price_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(13400, 1) (13400,)
(6600, 1) (6600,)
====================================================================================================
```

In [49]:
```
from sklearn.preprocessing import Normalizer
from sklearn.preprocessing import StandardScaler
normalizer = StandardScaler() #Normalizer()
# normalizer_test = Normalizer()
# normalizer.fit(X_train['essay_count'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.
normalizer.fit(X_train['essay_count'].values.reshape(-1,1))
# normalizer_test.fit(X_test['essay_count'].values.reshape(1,-1))

X_train_essay_count_norm = normalizer.transform(X_train['essay_count'].values.re
# X_cv_essay_count_norm = normalizer.transform(X_cv['essay_count'].values.reshape
X_test_essay_count_norm = normalizer.transform(X_test['essay_count'].values.resh


# X_train_essay_count_norm=np.reshape(X_train_essay_count_norm,(1,-1))
# X_test_essay_count_norm=np.reshape(X_test_essay_count_norm,(1,-1))

print("After vectorizations")

# np.reshape(X_train_essay_count_norm,
print(X_train_essay_count_norm.shape, y_train.shape)
# print(X_cv_essay_count_norm.shape, y_cv.shape)
print(X_test_essay_count_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(13400, 1) (13400,)
(6600, 1) (6600,)
====================================================================================================
```

In [50]:
```
from sklearn.preprocessing import Normalizer
from sklearn.preprocessing import StandardScaler
normalizer = StandardScaler() #Normalizer()
# normalizer_test = Normalizer()
# normalizer.fit(X_train['project_title_count'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.
normalizer.fit(X_train['project_title_count'].values.reshape(-1,1))
# normalizer_test.fit(X_test['project_title_count'].values.reshape(1,-1))

X_train_project_title_count_norm = normalizer.transform(X_train['project_title_c
# X_cv_project_title_count_norm = normalizer.transform(X_cv['project_title_count
X_test_project_title_count_norm = normalizer.transform(X_test['project_title_cou


# X_train_project_title_count_norm=np.reshape(X_train_project_title_count_norm,(1
# X_test_project_title_count_norm=np.reshape(X_test_project_title_count_norm,(1,

print("After vectorizations")

# np.reshape(X_train_project_title_count_norm,
print(X_train_project_title_count_norm.shape, y_train.shape)
# print(X_cv_project_title_count_norm.shape, y_cv.shape)
print(X_test_project_title_count_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(13400, 1) (13400,)
(6600, 1) (6600,)
====================================================================================
====================
```

In [51]:
```
from sklearn.preprocessing import Normalizer
from sklearn.preprocessing import StandardScaler
normalizer = StandardScaler() #Normalizer()
# normalizer_test = Normalizer()
# normalizer.fit(X_train['pos'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.
normalizer.fit(X_train['pos'].values.reshape(-1,1))
# normalizer_test.fit(X_test['pos'].values.reshape(1,-1))

X_train_pos_norm = normalizer.transform(X_train['pos'].values.reshape(-1,1))
# X_cv_pos_norm = normalizer.transform(X_cv['pos'].values.reshape(-1,1))
X_test_pos_norm = normalizer.transform(X_test['pos'].values.reshape(-1,1))


# X_train_pos_norm=np.reshape(X_train_pos_norm,(1,-1))
# X_test_pos_norm=np.reshape(X_test_pos_norm,(1,-1))

print("After vectorizations")

# np.reshape(X_train_pos_norm,
print(X_train_pos_norm.shape, y_train.shape)
# print(X_cv_pos_norm.shape, y_cv.shape)
print(X_test_pos_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(13400, 1) (13400,)
(6600, 1) (6600,)
====================================================================================================
```

In [52]:
```
from sklearn.preprocessing import Normalizer
from sklearn.preprocessing import StandardScaler
normalizer = StandardScaler() #Normalizer()
# normalizer_test = Normalizer()
# normalizer.fit(X_train['neu'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.
normalizer.fit(X_train['neu'].values.reshape(-1,1))
# normalizer_test.fit(X_test['neu'].values.reshape(1,-1))

X_train_neu_norm = normalizer.transform(X_train['neu'].values.reshape(-1,1))
# X_cv_neu_norm = normalizer.transform(X_cv['neu'].values.reshape(-1,1))
X_test_neu_norm = normalizer.transform(X_test['neu'].values.reshape(-1,1))


# X_train_neu_norm=np.reshape(X_train_neu_norm,(1,-1))
# X_test_neu_norm=np.reshape(X_test_neu_norm,(1,-1))

print("After vectorizations")

# np.reshape(X_train_neu_norm,
print(X_train_neu_norm.shape, y_train.shape)
# print(X_cv_neu_norm.shape, y_cv.shape)
print(X_test_neu_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(13400, 1) (13400,)
(6600, 1) (6600,)
====================================================================================================
```

In [53]:
```
from sklearn.preprocessing import Normalizer
from sklearn.preprocessing import StandardScaler
normalizer = StandardScaler() #Normalizer()
# normalizer_test = Normalizer()
# normalizer.fit(X_train['neg'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.
normalizer.fit(X_train['neg'].values.reshape(-1,1))
# normalizer_test.fit(X_test['neg'].values.reshape(1,-1))

X_train_neg_norm = normalizer.transform(X_train['neg'].values.reshape(-1,1))
# X_cv_neg_norm = normalizer.transform(X_cv['neg'].values.reshape(-1,1))
X_test_neg_norm = normalizer.transform(X_test['neg'].values.reshape(-1,1))


# X_train_neg_norm=np.reshape(X_train_neg_norm,(1,-1))
# X_test_neg_norm=np.reshape(X_test_neg_norm,(1,-1))

print("After vectorizations")

# np.reshape(X_train_neg_norm,
print(X_train_neg_norm.shape, y_train.shape)
# print(X_cv_neg_norm.shape, y_cv.shape)
print(X_test_neg_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(13400, 1) (13400,)
(6600, 1) (6600,)
====================================================================================================
```

In [54]:
```
from sklearn.preprocessing import Normalizer
from sklearn.preprocessing import StandardScaler
normalizer = StandardScaler() #Normalizer()
# normalizer_test = Normalizer()
# normalizer.fit(X_train['compound'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.
normalizer.fit(X_train['compound'].values.reshape(-1,1))
# normalizer_test.fit(X_test['compound'].values.reshape(1,-1))

X_train_compound_norm = normalizer.transform(X_train['compound'].values.reshape(
# X_cv_compound_norm = normalizer.transform(X_cv['compound'].values.reshape(-1,1
X_test_compound_norm = normalizer.transform(X_test['compound'].values.reshape(-1


# X_train_compound_norm=np.reshape(X_train_compound_norm,(1,-1))
# X_test_compound_norm=np.reshape(X_test_compound_norm,(1,-1))

print("After vectorizations")

# np.reshape(X_train_compound_norm,
print(X_train_compound_norm.shape, y_train.shape)
# print(X_cv_compound_norm.shape, y_cv.shape)
print(X_test_compound_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(13400, 1) (13400,)
(6600, 1) (6600,)
====================================================================================
====================
```

2.3.2 Vectorizing Numerical features--teacher_number_of_previously_posted_projects

In [55]:
```
from sklearn.preprocessing import Normalizer
normalizer_train = StandardScaler() #Normalizer()
normalizer_test = StandardScaler() #Normalizer()
# normalizer.fit(X_train['teacher_number_of_previously_posted_projects'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.
normalizer.fit(X_train['teacher_number_of_previously_posted_projects'].values.re

X_train_teacher_number_of_previously_posted_projects_norm = normalizer.transform
# X_cv_teacher_number_of_previously_posted_projects_norm = normalizer.transform(
X_test_teacher_number_of_previously_posted_projects_norm = normalizer.transform(



# X_train_teacher_number_of_previously_posted_projects_norm=np.reshape(X_train_te
# X_test_teacher_number_of_previously_posted_projects_norm=np.reshape(X_test_tea


print("After vectorizations")
print(X_train_teacher_number_of_previously_posted_projects_norm.shape, y_train.sh
# print(X_cv_teacher_number_of_previously_posted_projects_norm.shape, y_cv.shape,
print(X_test_teacher_number_of_previously_posted_projects_norm.shape, y_test.shap
print("="*100)
```

```
After vectorizations
(13400, 1) (13400,)
(6600, 1) (6600,)
====================================================================================================
```

2.3.3 Vectorizing Numerical features--digits_in_summary

In [56]:
```
X_train['digits_in_summary'].fillna(X_train['digits_in_summary'].mean(), inplace=
# X_cv['digits_in_summary'].fillna(X_cv['digits_in_summary'].mean(), inplace=True
X_test['digits_in_summary'].fillna(X_test['digits_in_summary'].mean(), inplace=Tr
```

In [57]:
```
from sklearn.preprocessing import Normalizer
normalizer_train = StandardScaler() #Normalizer()
normalizer_test = StandardScaler() #Normalizer()
# normalizer.fit(X_train['digits_in_summary'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.
normalizer.fit(X_train['digits_in_summary'].values.reshape(-1,1))

X_train_digits_in_summary_norm = normalizer.transform(X_train['digits_in_summary
# X_cv_digits_in_summary_norm = normalizer.transform(X_cv['digits_in_summary'].va
X_test_digits_in_summary_norm = normalizer.transform(X_test['digits_in_summary']


# X_train_digits_in_summary_norm=np.reshape(X_train_digits_in_summary_norm,(1,-1)
# X_test_digits_in_summary_norm=np.reshape(X_test_digits_in_summary_norm,(1,-1))

print("After vectorizations")
print(X_train_digits_in_summary_norm.shape, y_train.shape)
# print(X_cv_digits_in_summary_norm.shape, y_cv.shape)
print(X_test_digits_in_summary_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(13400, 1) (13400,)
(6600, 1) (6600,)
====================================================================================
====================
```

In [58]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
total_text = []
vectorizer = TfidfVectorizer(min_df = 10,use_idf = True,stop_words=stopwords)
model_2000 = vectorizer.fit_transform(X["essay_pr_title"].values)

top_2000 = pd.DataFrame({"feature_names":list(vectorizer.get_feature_names()),"i

top_2000_features = top_2000.sort_values(by=['idf_values'],ascending=False)[:200

top_2000_features[:10]
```

Out[58]:

	feature_names	idf_values
2174	diane	8.505642
7103	spider	8.505642
5691	pointers	8.505642
2449	eagle	8.505642
5100	newsletter	8.505642
1115	bye	8.505642
2017	declaration	8.505642
2891	explosive	8.505642
6478	risen	8.505642
7845	tricks	8.505642

In [59]:
```
def COmatrix( data, words, cw=5 ):

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

    for sent in data:

        word = sent.split()

        for ind in range( len(word) ):

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

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

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

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

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

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


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

    return cm
```

In [60]:
```
import pandas as pd
import numpy as np

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

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

df_sample.head()
```

Out[60]:

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

In [61]:
```
top_words = ['ABC','PQR', 'DEF']
a = COmatrix(df_sample['text'],top_words,2)
a
```

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

In [62]:
```
cooccur_matrix_text = COmatrix(X["essay_pr_title"], top_2000_features['feature_na
cooccur_matrix_text
```

Out[62]:

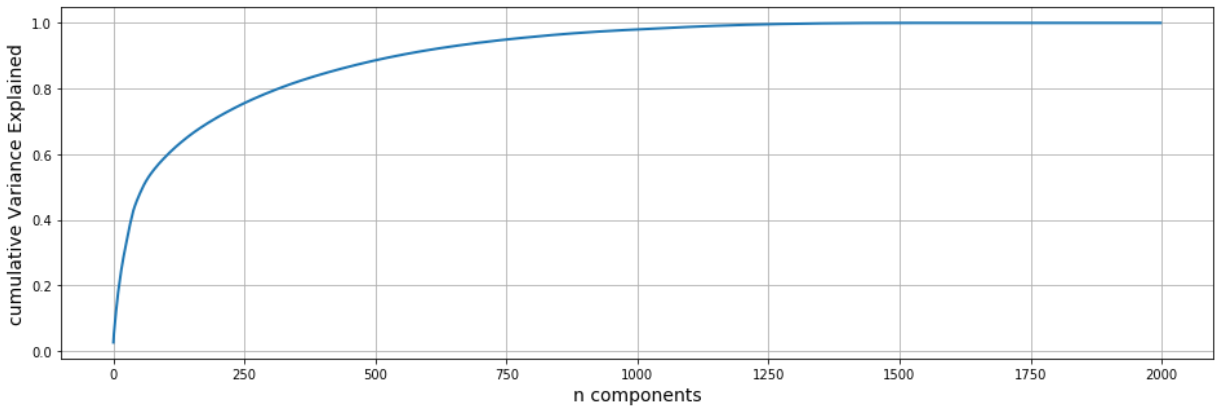
feature_names	diane	spider	pointers	eagle	newsletter	bye	declaration	explosive	risen	tri
feature_names										
diane	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
spider	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
pointers	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
eagle	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
newsletter	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
bye	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
declaration	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
explosive	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
risen	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
tricks	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
trick	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
majors	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
hindrance	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
advantaged	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
helmets	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
frisbee	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
vertical	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
hits	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
roam	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
hobby	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
veterans	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
estimate	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
niche	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
roaring	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
nightmare	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
et	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
ridge	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
richly	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
vacations	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
headstart	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
...	...	...	...	...	...	...	...	...	...	
unconventional	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	

feature_names	diane	spider	pointers	eagle	newsletter	bye	declaration	explosive	risen	tri
feature_names										
carbon	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
references	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
mysteries	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
reduction	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
drinking	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
noses	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
stretches	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
camden	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
frank	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
dread	0.0	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	0.0	
newsletters	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
recommendations	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
osmos	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
furthering	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
trace	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
sunny	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
captures	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
argue	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
carried	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
modules	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
programmable	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
unsuccessful	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
occupied	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
surprisingly	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
stapler	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
upgraded	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
guitars	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
requisite	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	

2000 rows × 2000 columns

In [63]:
```
from sklearn.decomposition import TruncatedSVD
# Dim-reduction using Truncated SVD
svd = TruncatedSVD( n_components = 1999, random_state=42 )
trsvd_text = svd.fit_transform(cooccur_matrix_text)
cumVarianceExplained = np.cumsum( svd.explained_variance_ratio_ )
```

In [64]:
```
import matplotlib.pyplot as plt1

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

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



In [65]:
```
from sklearn.decomposition import TruncatedSVD

# Dim-reduction using Truncated SVD

svd = TruncatedSVD( n_components = 750, random_state = 42 )
textsvd = svd.fit_transform(cooccur_matrix_text)
cumVarianceExplained = np.cumsum( svd.explained_variance_ratio_ )
```

In [66]:
```
top2ktext_df = pd.DataFrame( textsvd, index = cooccur_matrix_text.index)
top2ktext_df
```

Out[66]:

feature_names	0	1	2	3	4	
diane	3.773251e-10	-4.392362e-10	1.129821e-12	-3.542662e-12	8.994877e-09	6.95
spider	2.730588e-09	2.260827e-09	-4.827573e-12	1.271150e-12	1.739699e-05	2.45
pointers	4.627493e-11	1.021050e-11	-3.338777e-13	-3.216661e-13	4.183831e-06	2.04
eagle	5.950640e-09	4.411857e-10	2.637526e-13	1.179010e-12	1.078699e-01	-4.
newsletter	1.512210e-11	3.101994e-12	-1.112041e-12	1.045688e-12	2.302235e-07	7.93
bye	-1.243454e-16	-2.330886e-16	9.661687e-16	-1.499093e-16	-1.051427e-15	-3.

In [67]:
```
index = list(np.arange(1,2000,1))
```

2.4 Make Data Model Ready: Vectorizing Essay and Project_title feature

In [68]:
```
top_feats = list(top_2000_features.feature_names.values)
indx = 0
essay_vectorizer_train = []
for text in tqdm(X_train['essay']):
    vect_sum = np.zeros((1,750))
    count = 0
    for word in tqdm(text.split()):
        if word in top_feats:
            indx = top_feats.index(word)
            count+=1
            vect_sum+=textsvd[indx]
    if count!=0:
        vect_sum = vect_sum/count
    else:
        vect_sum = vect_sum
    essay_vectorizer_train.append(vect_sum)
```

```
  0%|                                                  | 0/13400 [00:00<?, ?it/
s]
  0%|                                                    | 0/139 [00:00<?, ?it/
s]
100%|████████████████████████████████████████| 139/139 [00:00<00:00, 8175.80it/
s]
  0%|                                                    | 0/135 [00:00<?, ?it/
s]
100%|████████████████████████████████████████| 135/135 [00:00<00:00, 8437.11it/
s]
  0%|                                                    | 0/108 [00:00<?, ?it/
s]
100%|████████████████████████████████████████| 108/108 [00:00<00:00, 7713.93it/
s]
  0%|                                                    | 0/154 [00:00<?, ?it/
s]
100%|████████████████████████████████████████| 154/154 [00:00<00:00, 7699.55it/
s]
  0%|                                                    | 0/108 [00:00<?, ?it/
```

In [69]:
```
top_feats = list(top_2000_features.feature_names.values)
indx = 0
essay_vectorizer_test = []
for text in tqdm(X_test['essay']):
    vect_sum = np.zeros((1,750))
    count = 0
    for word in tqdm(text.split()):
        if word in top_feats:
            indx = top_feats.index(word)
            count+=1
            vect_sum+=textsvd[indx]
    if count!=0:
        vect_sum = vect_sum/count
    else:
        vect_sum = vect_sum
    essay_vectorizer_test.append(vect_sum)
```

```
  0%|                                                          | 0/6600 [00:00<?, ?it/
s]
  0%|                                                           | 0/89 [00:00<?, ?it/
s]
100%|████████████████████████████████████████| 89/89 [00:00<00:00, 5562.24it/
s]
  0%|                                                          | 0/155 [00:00<?, ?it/
s]
100%|████████████████████████████████████████| 155/155 [00:00<00:00, 5535.41it/
s]
  0%|                                                          | 0/111 [00:00<?, ?it/
s]
100%|████████████████████████████████████████| 111/111 [00:00<00:00, 5841.80it/
s]
  0%|                                                          | 0/171 [00:00<?, ?it/
s]
  0%|                                                 | 4/6600 [00:00<03:37, 30.30it/
s]
  0%|                                                          | 0/146 [00:00<?, ?it/
```

In [70]:
```
top_feats = list(top_2000_features.feature_names.values)
indx = 0
project_title_vectorizer_train = []
for text in tqdm(X_train["project_title"]):
    vect_sum = np.zeros((750))
    count = 0
    for word in tqdm(text.split()):
        if word in top_feats:
            indx = top_feats.index(word)
            count+=1
            vect_sum+=textsvd[indx]
    if count!=0:
        vect_sum = vect_sum/count
    else:
        vect_sum = vect_sum
    project_title_vectorizer_train.append(vect_sum)
```

```
  0%|                                                  | 0/13400 [00:00<?, ?it/
s]
  0%|                                                     | 0/5 [00:00<?, ?it/
s]
100%|████████████████████████████████████████████████| 5/5 [00:00<00:00, 1250.02it/
s]
  0%|                                                     | 0/3 [00:00<?, ?it/
s]
100%|████████████████████████████████████████████████| 3/3 [00:00<00:00, 499.98it/
s]
  0%|                                                     | 0/2 [00:00<?, ?it/
s]
100%|████████████████████████████████████████████████| 2/2 [00:00<00:00, 666.66it/
s]
  0%|                                                     | 0/4 [00:00<?, ?it/
s]
100%|████████████████████████████████████████████████| 4/4 [00:00<00:00, 1333.11it/
s]
  0%|                                                     | 0/3 [00:00<?, ?it/
```

In [71]:
```
top_feats = list(top_2000_features.feature_names.values)
indx = 0
project_title_vectorizer_test = []
for text in tqdm(X_test["project_title"]):
    vect_sum = np.zeros((1,750))
    count = 0
    for word in tqdm(text.split()):
        if word in top_feats:
            indx = top_feats.index(word)
            count+=1
            vect_sum+=textsvd[indx]
    if count!=0:
        vect_sum = vect_sum/count
    else:
        vect_sum = vect_sum
    project_title_vectorizer_test.append(vect_sum)
```

```
  0%|                                                                | 0/5 [00:00<?, ?it/
s]
100%|████████████████████████████████████████████████████████████| 5/5 [00:00<00:00, 2499.59it/
s]
  0%|                                                                | 0/8 [00:00<?, ?it/
s]
  3%|▉                                                  | 194/6600 [00:02<01:42, 62.35i
t/s]
  0%|                                                                | 0/3 [00:00<?, ?it/
s]
100%|████████████████████████████████████████████████████████████| 3/3 [00:00<00:00, 1499.93it/
s]
  0%|                                                                | 0/6 [00:00<?, ?it/
s]
100%|████████████████████████████████████████████████████████████| 6/6 [00:00<00:00, 2999.86it/
s]
  0%|                                                                | 0/5 [00:00<?, ?it/
s]
100%|████████████████████████████████████████████████████████████| 5/5 [00:00<00:00, 5000.36it/
```

In [72]:
```
essay_vectorizer_train_ = []
title_vectorizer_train_= []
essay_vectorizer_test_ = []
title_vectorizer_test_ = []

r, c = X_train.shape
s, c = X_test.shape
for i in range(r):
    essay_vectorizer_train_.append(essay_vectorizer_train[i][0])
# title_vectorizer_train_ = []
for i in range(r):
    title_vectorizer_train_.append(project_title_vectorizer_train[i][0])
# essay_vectorizer_test_ = []
for i in range(s):
    essay_vectorizer_test_.append(essay_vectorizer_test[i][0])
# title_vectorizer_test_ = []
for i in range(s):
    title_vectorizer_test_.append(project_title_vectorizer_test[i][0])
```

In [73]:
```
print(X_train_digits_in_summary_norm.shape)
print(X_train_teacher_number_of_previously_posted_projects_norm.shape)
print(X_train_price_norm.shape)
print(X_train_teacher_prefix_ohe.shape)
print(X_train_project_grade_category_ohe.shape)
print(X_train_School_state_ohe.shape)
print(X_train_clean_sub_cat_ohe.shape)
print(X_train_clean_cat_ohe.shape)
print(X_train_compound_norm.shape)
print(X_train_neg_norm.shape)
print(X_train_neu_norm.shape)
print(X_train_pos_norm.shape)
print(X_train_project_title_count_norm.shape)
print(X_train_essay_count_norm.shape)
```

```
(13400, 1)
(13400, 1)
(13400, 1)
(13400, 5)
(13400, 4)
(13400, 51)
(13400, 30)
(13400, 9)
(13400, 1)
(13400, 1)
(13400, 1)
(13400, 1)
(13400, 1)
(13400, 1)
```

In [74]:
```
print(X_test_digits_in_summary_norm.shape)
print(X_test_teacher_number_of_previously_posted_projects_norm.shape)
print(X_test_price_norm.shape)
print(X_test_teacher_prefix_ohe.shape)
print(X_test_project_grade_category_ohe.shape)
print(X_test_School_state_ohe.shape)
print(X_test_clean_sub_cat_ohe.shape)
print(X_test_clean_cat_ohe.shape)
print(X_test_compound_norm.shape)
print(X_test_neg_norm.shape)
print(X_test_neu_norm.shape)
print(X_test_pos_norm.shape)
print(X_test_project_title_count_norm.shape)
print(X_test_essay_count_norm.shape)
```

```
(6600, 1)
(6600, 1)
(6600, 1)
(6600, 5)
(6600, 4)
(6600, 51)
(6600, 30)
(6600, 9)
(6600, 1)
(6600, 1)
(6600, 1)
(6600, 1)
(6600, 1)
(6600, 1)
```

In [75]:
```
title_vectorizer_train_ = np.array(title_vectorizer_train_)
title_vectorizer_train_ = np.array(essay_vectorizer_train_)
title_vectorizer_test_ = np.array(title_vectorizer_test_)
essay_vectorizer_test_ = np.array(essay_vectorizer_test_)
```

In [76]:
```
from scipy.sparse import hstack

X_train = hstack((title_vectorizer_train_,
                  essay_vectorizer_train_,
                  X_train_digits_in_summary_norm,
                  X_train_teacher_number_of_previously_posted_projects_norm,
                  X_train_price_norm,
                    X_train_teacher_prefix_ohe,
                  X_train_project_grade_category_ohe,
                  X_train_School_state_ohe,
                  X_train_clean_sub_cat_ohe,
                  X_train_clean_cat_ohe,
                   X_train_compound_norm,
                     X_train_neg_norm,
                     X_train_neu_norm,
                     X_train_pos_norm,
                     X_train_project_title_count_norm,
                     X_train_essay_count_norm


                     ))

# X_train_project_title_count_norm,
# X_train_essay_count_norm

X_train=X_train.todense()
X_train=np.array(X_train)

# X_cv = hstack(( X_cv_project_title,X_cv_essay,X_cv_digits_in_summary_norm,X_cv_
# X_cv=X_cv.todense()
# X_cv=np.array(X_cv)

X_test = hstack((title_vectorizer_test_,
                 essay_vectorizer_test_,
                 X_test_digits_in_summary_norm,
                 X_test_teacher_number_of_previously_posted_projects_norm,
                 X_test_price_norm,
                 X_test_teacher_prefix_ohe,
                 X_test_project_grade_category_ohe,
                 X_test_School_state_ohe,
                 X_test_clean_sub_cat_ohe,
                 X_test_clean_cat_ohe,
                 X_test_compound_norm,
                 X_test_neg_norm,
                 X_test_neu_norm,
                 X_test_pos_norm,
                 X_test_project_title_count_norm,
                 X_test_essay_count_norm))
X_test=X_test.todense()
X_test=np.array(X_test)
```

In [77]:
```
title_vectorizer_train_=None
essay_vectorizer_train_=None
X_train_digits_in_summary_norm=None
X_train_teacher_number_of_previously_posted_projects_norm=None
X_train_price_norm=None
X_train_teacher_prefix_ohe=None
X_train_project_grade_category_ohe=None
X_train_School_state_ohe=None
X_train_clean_sub_cat_ohe=None
X_train_clean_cat_ohe=None
X_train_compound_norm=None
X_train_neg_norm=None
X_train_neu_norm=None
X_train_pos_norm=None
X_train_project_title_count_norm=None
X_train_essay_count_norm=None
```

In [78]:
```
title_vectorizer_test_=None
essay_vectorizer_test_=None
X_test_digits_in_summary_norm=None
X_test_teacher_number_of_previously_posted_projects_norm=None
X_test_price_norm=None
X_test_teacher_prefix_ohe=None
X_test_project_grade_category_ohe=None
X_test_School_state_ohe=None
X_test_clean_sub_cat_ohe=None
X_test_clean_cat_ohe=None
X_test_compound_norm=None
X_test_neg_norm=None
X_test_neu_norm=None
X_test_pos_norm=None
X_test_project_title_count_norm=None
X_test_essay_count_norm=None
```

In [79]:
```
X_train.shape
```

Out[79]:
```
(13400, 1608)
```

2.4 Make Data Model Ready: Vectorizing Essay and Project_title feature into BOW & TFIDF

Vectorizing Text data

2.4.1 Bag of words:Essays

2.5 Applying SVD on `BOW, SET 1`

2.5.1 Applying SVD & GridSearchCV on Train data to obtain the best `C`

In [80]:
```
%%time

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

from sklearn.model_selection import GridSearchCV
from xgboost import XGBClassifier

learning_rate = [0.0001, 0.001, 0.01, 0.1, 0.2, 0.3]
n_estimators = [5,10,20,25]

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

clf_XGBT_finalData = GridSearchCV(XGBClassifier(booster='gbtree', class_weight =
clf_XGBT_finalData.fit(X_train, y_train)
```

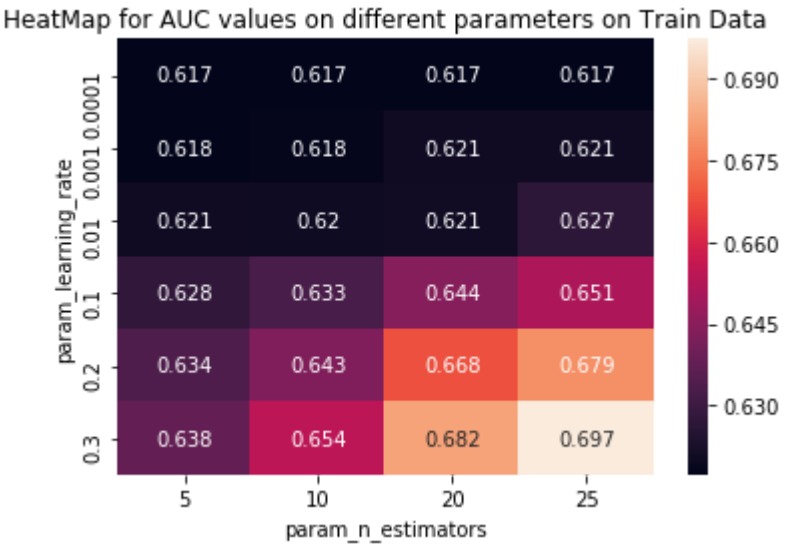
```
Wall time: 27min 51s
Parser   : 104 ms
```

In [81]:
```
print(clf_XGBT_finalData.best_estimator_)
print(clf_XGBT_finalData.best_score_)
```

```
XGBClassifier(base_score=0.5, booster='gbtree', class_weight='balanced',
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
              gamma=0, learning_rate=0.1, max_delta_step=0, max_depth=3,
              min_child_weight=1, missing=None, n_estimators=25, n_jobs=1,
              nthread=None, objective='binary:logistic', random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=None, subsample=1, verbosity=1)
0.6183582089552239
```

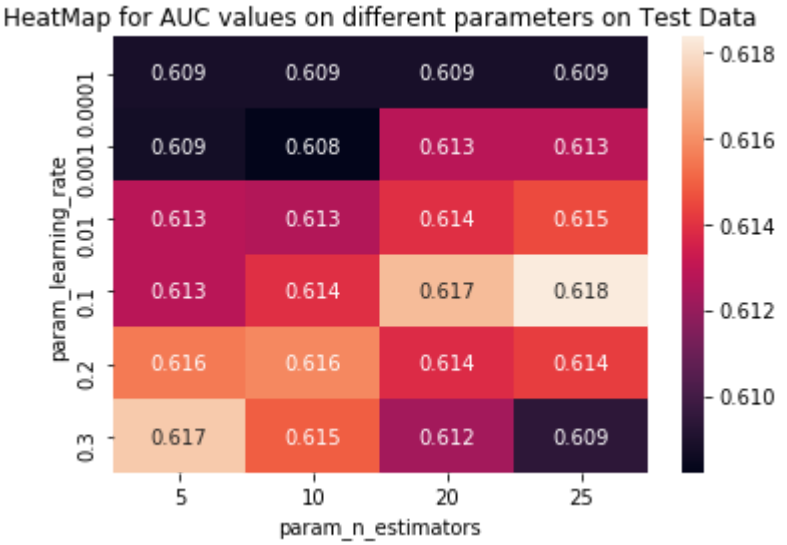
In [82]:
```
df_gridsearch = pd.DataFrame(clf_XGBT_finalData.cv_results_)
max_scores = df_gridsearch.groupby(['param_learning_rate','param_n_estimators'])
max_scores = max_scores.unstack()[['mean_test_score', 'mean_train_score']]
sns.heatmap(max_scores.mean_train_score,annot=True, annot_kws={"size": 10}, fmt=
plt.title('HeatMap for AUC values on different parameters on Train Data')
```

Out[82]:
```
Text(0.5, 1.0, 'HeatMap for AUC values on different parameters on Train Data')
```



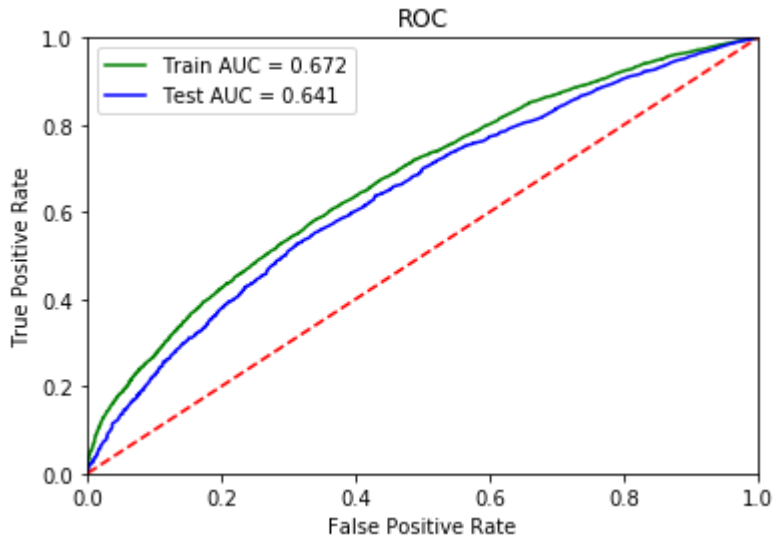
In [83]:
```
df_gridsearch = pd.DataFrame(clf_XGBT_finalData.cv_results_)
max_scores = df_gridsearch.groupby(['param_learning_rate','param_n_estimators'])
max_scores = max_scores.unstack()[['mean_test_score', 'mean_train_score']]
sns.heatmap(max_scores.mean_test_score,annot=True, annot_kws={"size": 10}, fmt='
plt.title('HeatMap for AUC values on different parameters on Test Data')
```

Out[83]:
```
Text(0.5, 1.0, 'HeatMap for AUC values on different parameters on Test Data')
```

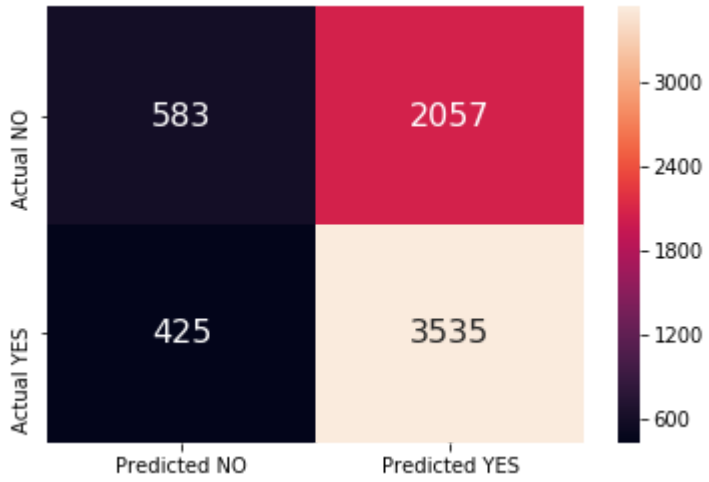


In [85]:
```
clf_XGBT_finalData_best = XGBClassifier(booster='gbtree', class_weight = 'balanc
clf_XGBT_finalData_best.fit(X_train, y_train)

pred_test = clf_XGBT_finalData_best.predict_proba(X_test)[:,1]
pred_train = clf_XGBT_finalData_best.predict_proba(X_train)[:,1]
```

In [86]:
```
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import roc_auc_score


fpr, tpr, thresholds = roc_curve(y_test, pred_test)
fpr2, tpr2, thresholds = roc_curve(y_train, pred_train)

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



In [87]:
```
# https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix
#function to get heatmap confusion matrix
def get_confusion_matrix(clf,X_te,y_test):
    y_pred = clf.predict(X_te)
    df_cm = pd.DataFrame(confusion_matrix(y_test, y_pred), index =['Actual NO','A
    sns.heatmap(df_cm, annot=True,annot_kws={"size": 16}, fmt='g')

# %%time
get_confusion_matrix(clf_XGBT_finalData_best,X_test,y_test)
```



In [1]:
```
#http://zetcode.com/python/prettytable/
from prettytable import PrettyTable

x = PrettyTable()
x.field_names = ["Vectorizer","Model","n_estimators","max_depth","Test-AUC"]
x.add_row([" Truncated SVD","XGBClassifier", '25','3', 0.64])

print(x)
```

```
+----------------+---------------+--------------+-----------+----------+
|   Vectorizer   |     Model     | n_estimators | max_depth | Test-AUC |
+----------------+---------------+--------------+-----------+----------+
|  Truncated SVD | XGBClassifier |      25      |     3     |   0.64   |
+----------------+---------------+--------------+-----------+----------+
```