DonorsChoose

DonorsChoose.org receives hundreds of thousands of project proposals each year for classroom projects in need of funding. Right now, a large number of volunteers is needed to manually screen each submission before it's approved to be posted on the DonorsChoose.org website.

Next year, DonorsChoose.org expects to receive close to 500,000 project proposals. As a result, there are three main problems they need to solve:

- How to scale current manual processes and resources to screen 500,000 projects so that they can be posted as quickly and as efficiently as possible
- How to increase the consistency of project vetting across different volunteers to improve the experience for teachers
- How to focus volunteer time on the applications that need the most assistance

The goal of the competition is to predict whether or not a DonorsChoose.org project proposal submitted by a teacher will be approved, using the text of project descriptions as well as additional metadata about the project, teacher, and school. DonorsChoose.org can then use this information to identify projects most likely to need further review before approval.

About the DonorsChoose Data Set

The `train.csv` data set provided by DonorsChoose contains the following features:

Feature	Description
`project_id`	A unique identifier for the proposed project. **Example:** `p036502`
`project_title`	Title of the project. **Examples:** • `Art Will Make You Happy!` • `First Grade Fun`
`project_grade_category`	Grade level of students for which the project is targeted. One of the following enumerated values: • `Grades PreK-2` • `Grades 3-5` • `Grades 6-8` • `Grades 9-12`
`project_subject_categories`	One or more (comma-separated) subject categories for the project from the following enumerated list of values: • `Applied Learning` • `Care & Hunger` • `Health & Sports` • `History & Civics` • `Literacy & Language` • `Math & Science` • `Music & The Arts` • `Special Needs` • `Warmth` **Examples:** • `Music & The Arts` • `Literacy & Language, Math & Science`
`school_state`	State where school is located (Two-letter U.S. postal code). **Example:** `WY`
`project_subject_subcategories`	One or more (comma-separated) subject subcategories for the project. **Examples:** • `Literacy` • `Literature & Writing, Social Sciences`
`project_resource_summary`	An explanation of the resources needed for the project. **Example:** • `My students need hands on literacy materials to manage sensory needs!`
`project_essay_1`	First application essay*
`project_essay_2`	Second application essay*
`project_essay_3`	Third application essay*

Feature	Description
`project_essay_4`	Fourth application essay
`project_submitted_datetime`	Datetime when project application was submitted. **Example:** `2016-04-28 12:43:56.245`
`teacher_id`	A unique identifier for the teacher of the proposed project. **Example:** `bdf8baa8fedef6bfeec7ae4ff1c15c56`
`teacher_prefix`	Teacher's title. One of the following enumerated values: • `nan` • `Dr.` • `Mr.` • `Mrs.` • `Ms.` • `Teacher.`
`teacher_number_of_previously_posted_projects`	Number of project applications previously submitted by the same teacher. **Example:** `2`

* See the section **Notes on the Essay Data** for more details about these features.

Additionally, the `resources.csv` data set provides more data about the resources required for each project. Each line in this file represents a resource required by a project:

Feature	Description
`id`	A `project_id` value from the `train.csv` file. **Example:** `p036502`
`description`	Desciption of the resource. **Example:** `Tenor Saxophone Reeds, Box of 25`
`quantity`	Quantity of the resource required. **Example:** `3`
`price`	Price of the resource required. **Example:** `9.95`

Note: Many projects require multiple resources. The `id` value corresponds to a `project_id` in train.csv, so you use it as a key to retrieve all resources needed for a project:

The data set contains the following label (the value you will attempt to predict):

Label	Description
`project_is_approved`	A binary flag indicating whether DonorsChoose approved the project. A value of `0` indicates the project was not approved, and a value of `1` indicates the project was approved.

Notes on the Essay Data

Prior to May 17, 2016, the prompts for the essays were as follows:

- __project_essay_1:__ "Introduce us to your classroom"
- __project_essay_2:__ "Tell us more about your students"
- __project_essay_3:__ "Describe how your students will use the materials you're requesting"
- __project_essay_3:__ "Close by sharing why your project will make a difference"

Starting on May 17, 2016, the number of essays was reduced from 4 to 2, and the prompts for the first 2 essays were changed to the following:

- __project_essay_1:__ "Describe your students: What makes your students special? Specific details about their background, your neighborhood, and your school are all helpful."
- __project_essay_2:__ "About your project: How will these materials make a difference in your students' learning and improve their school lives?"

For all projects with project_submitted_datetime of 2016-05-17 and later, the values of project_essay_3 and project_essay_4 will be NaN.

In [1]:

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

import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
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 Data

In [2]:

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

In [3]:

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

```
Number of data points in train data (109248, 17)
--------------------------------------------------
The attributes of data : ['Unnamed: 0' 'id' 'teacher_id' 'teacher_prefix' 'school_state'
 'project_submitted_datetime' 'project_grade_category'
 'project_subject_categories' 'project_subject_subcategories'
 'project_title' 'project_essay_1' 'project_essay_2' 'project_essay_3'
 'project_essay_4' 'project_resource_summary'
 'teacher_number_of_previously_posted_projects' 'project_is_approved']
```

In [4]:

```
print("Number of data points in train data", resource_data.shape)
print(resource_data.columns.values)
resource_data.head(2)
```

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

Out[4]:

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

Preprocessing of Project Subject Categories

In [5]:

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

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

project_data['clean_categories'] = cat_list
project_data.drop(['project_subject_categories'], axis=1, inplace=True)

from collections import Counter
my_counter = Counter()
for word in project_data['clean_categories'].values:
    my_counter.update(word.split())

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

Preprocessing of Project Subject Subcategories

In [6]:

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

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

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

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

# 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())
    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]))
```

Text Preprocessing

In [7]:

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

In [8]:

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

Out[8]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime	project_grade_cate
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	2016-12-05 13:43:57	Grades P
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL	2016-10-25 09:22:10	Grade

In [9]:

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

In [10]:

```
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 [11]:

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

In [12]:

```
from tqdm import tqdm
preprocessed_essays = []
# tqdm is for printing the status bar
for sentance in tqdm(project_data['essay'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    sent = re.sub('nannan', '', 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%|██████████| 109248/109248 [00:52<00:00, 2081.16it/s]
```

In [13]:

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

```
100%|██████████| 109248/109248 [00:02<00:00, 40906.31it/s]
```

In [14]:

```
project_data.drop(['project_essay_1','project_essay_2','project_essay_3','project_essay_4'], axis=
1, inplace=True)
project_data.head(2)
```

Out[14]:

Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime	project_grade_cate

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime	project_grade_cate
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	2016-12-05 13:43:57	Grades P
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL	2016-10-25 09:22:10	Grade

In [15]:

```
project_data['teacher_prefix'].value_counts().argmax()
project_data.fillna(value=project_data['teacher_prefix'].value_counts().argmax(),axis=1,inplace=Tru
e)
```

In [16]:

```
project_data['preprocessed_essays'] = preprocessed_essays
project_data['preprocessed_titles'] = preprocessed_titles
```

In [17]:

```
project_data.columns
```

Out[17]:

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'project_submitted_datetime', 'project_grade_category', 'project_title',
       'project_resource_summary',
       'teacher_number_of_previously_posted_projects', 'project_is_approved',
       'clean_categories', 'clean_subcategories', 'essay', 'price', 'quantity',
       'preprocessed_essays', 'preprocessed_titles'],
      dtype='object')
```

In [18]:

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

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

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

```
100%|██████████| 109248/109248 [00:00<00:00, 720394.74it/s]
```

In [19]:

```
project_data['clean_project_grade_category'] = project_grade_cat_list
project_data.drop(['project_grade_category'], axis=1, inplace=True)
project_data.head(2)
```

Out[19]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime	project_title	proje
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	2016-12-05 13:43:57	Educational Support for English Learners at Home	
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL	2016-10-25 09:22:10	Wanted: Projector for Hungry Learners	My s

In [20]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(project_data,project_data['project_is_approved'
], test_size=0.33, stratify = project_data['project_is_approved'])
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33, stratify=y_train)

X_train.drop(['project_is_approved'], axis=1, inplace=True)
X_test.drop(['project_is_approved'], axis=1, inplace=True)
X_cv.drop(['project_is_approved'], axis=1, inplace=True)
print(X_train.shape)
print(X_cv.shape)
print(X_test.shape)
```

```
(49041, 17)
(24155, 17)
(36052, 17)
```

Catagorical Data :

In [21]:

```
# we use count vectorizer to convert the values into one hot encoded features
from sklearn.feature_extraction.text import CountVectorizer
vectorizer_cat = CountVectorizer(vocabulary=list(sorted_cat_dict.keys()), lowercase=False, binary=
True)
vectorizer_cat.fit(X_train['clean_categories'].values)
print(vectorizer_cat.get_feature_names())
categories_one_hot_train = vectorizer_cat.transform(X_train['clean_categories'].values)
categories_one_hot_cv = vectorizer_cat.transform(X_cv['clean_categories'].values)
categories_one_hot_test = vectorizer_cat.transform(X_test['clean_categories'].values)
print("Shape of matrix after one hot encodig_train ",categories_one_hot_train.shape)
print("Shape of matrix after one hot encodig_cv ",categories_one_hot_cv.shape)
print("Shape of matrix after one hot encodig_test ",categories_one_hot_test.shape)
```

```
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'SpecialNeeds',
'Health_Sports', 'Math_Science', 'Literacy_Language']
Shape of matrix after one hot encodig_train  (49041, 9)
Shape of matrix after one hot encodig_cv  (24155, 9)
Shape of matrix after one hot encodig_test  (36052, 9)
```

In [22]:

```
vectorizer_sub_cat = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()), lowercase=False,
binary=True)
vectorizer_sub_cat.fit(X_train['clean_subcategories'].values)
print(vectorizer_sub_cat.get_feature_names())
sub_categories_one_hot_train = vectorizer_sub_cat.transform(X_train['clean_subcategories'].values)
sub_categories_one_hot_cv = vectorizer_sub_cat.transform(X_cv['clean_subcategories'].values)
sub_categories_one_hot_test = vectorizer_sub_cat.transform(X_test['clean_subcategories'].values)
print("Shape of matrix after one hot encodig_train ",sub_categories_one_hot_train.shape)
print("Shape of matrix after one hot encodig_cv ",sub_categories_one_hot_cv.shape)
print("Shape of matrix after one hot encodig_test ",sub_categories_one_hot_test.shape)
```

```
['Economics', 'CommunityService', 'FinancialLiteracy', 'ParentInvolvement', 'Extracurricular',
'Civics_Government', 'ForeignLanguages', 'NutritionEducation', 'Warmth', 'Care_Hunger',
'SocialSciences', 'PerformingArts', 'CharacterEducation', 'TeamSports', 'Other',
'College_CareerPrep', 'Music', 'History_Geography', 'Health_LifeScience', 'EarlyDevelopment', 'ESL
', 'Gym_Fitness', 'EnvironmentalScience', 'VisualArts', 'Health_Wellness', 'AppliedSciences',
'SpecialNeeds', 'Literature_Writing', 'Mathematics', 'Literacy']
Shape of matrix after one hot encodig_train  (49041, 30)
Shape of matrix after one hot encodig_cv  (24155, 30)
Shape of matrix after one hot encodig_test  (36052, 30)
```

In [23]:

```
# we use count vectorizer to convert the values into one hot encoded features
from sklearn.feature_extraction.text import CountVectorizer
vectorizer_state = CountVectorizer( lowercase=False, binary=True)
vectorizer_state.fit(X_train['school_state'].values)
print(vectorizer_state.get_feature_names())
school_state_one_hot_train = vectorizer_state.transform(X_train['school_state'].values)
school_state_one_hot_cv = vectorizer_state.transform(X_cv['school_state'].values)
school_state_one_hot_test = vectorizer_state.transform(X_test['school_state'].values)
print("Shape of matrix after one hot encodig_train ",school_state_one_hot_train.shape)
print("Shape of matrix after one hot encodig_cv ",school_state_one_hot_cv.shape)
print("Shape of matrix after one hot encodig_test ",school_state_one_hot_test.shape)
```

```
['AK', 'AL', 'AR', 'AZ', 'CA', 'CO', 'CT', 'DC', 'DE', 'FL', 'GA', 'HI', 'IA', 'ID', 'IL', 'IN', 'K
S', 'KY', 'LA', 'MA', 'MD', 'ME', 'MI', 'MN', 'MO', 'MS', 'MT', 'NC', 'ND', 'NE', 'NH', 'NJ', 'NM',
'NV', 'NY', 'OH', 'OK', 'OR', 'PA', 'RI', 'SC', 'SD', 'TN', 'TX', 'UT', 'VA', 'VT', 'WA', 'WI', 'WV
', 'WY']
Shape of matrix after one hot encodig_train  (49041, 51)
Shape of matrix after one hot encodig_cv  (24155, 51)
Shape of matrix after one hot encodig_test  (36052, 51)
```

In [24]:

```
# we use count vectorizer to convert the values into one hot encoded features
from sklearn.feature_extraction.text import CountVectorizer
vectorizer_teacherprefix = CountVectorizer( lowercase=False, binary=True)
vectorizer_teacherprefix.fit(X_train['teacher_prefix'].values)
print(vectorizer_teacherprefix.get_feature_names())
#https://stackoverflow.com/a/39308809/8089731
teacher_prefix_one_hot_train = vectorizer_teacherprefix.transform(X_train['teacher_prefix'].values
.astype('U'))
teacher_prefix_one_hot_cv =
vectorizer_teacherprefix.transform(X_cv['teacher_prefix'].values.astype('U'))
teacher_prefix_one_hot_test =
vectorizer_teacherprefix.transform(X_test['teacher_prefix'].values.astype('U'))
print("Shape of matrix after one hot encodig_train ",teacher_prefix_one_hot_train.shape)
print("Shape of matrix after one hot encodig_cv ",teacher_prefix_one_hot_cv.shape)
print("Shape of matrix after one hot encodig_test ",teacher_prefix_one_hot_test.shape)
```

```
['Dr', 'Mr', 'Mrs', 'Ms', 'Teacher']
Shape of matrix after one hot encodig_train  (49041, 5)
Shape of matrix after one hot encodig_cv  (24155, 5)
Shape of matrix after one hot encodig_test  (36052, 5)
```

In [25]:

```
print(project_data['clean_project_grade_category'].unique())# we use count vectorizer to convert t
he values into one hot encoded features
from sklearn.feature_extraction.text import CountVectorizer
# https://stackoverflow.com/a/38161028/8089731
pattern = "(?u)\\b[\\w-]+\\b"
vectorizer_projectgrade = CountVectorizer(token_pattern=pattern, lowercase=False, binary=True)
vectorizer_projectgrade.fit(X_train['clean_project_grade_category'].values)
print(vectorizer_projectgrade.get_feature_names())
pgc_one_hot_train = vectorizer_projectgrade.transform(X_train['clean_project_grade_category'].valu
es)
pgc_one_hot_cv = vectorizer_projectgrade.transform(X_cv['clean_project_grade_category'].values)
pgc_one_hot_test = vectorizer_projectgrade.transform(X_test['clean_project_grade_category'].values
)
```

```
print("Shape of matrix after one hot encodig_train ",pgc_one_hot_train.shape)
print("Shape of matrix after one hot encodig_cv ",pgc_one_hot_cv.shape)
print("Shape of matrix after one hot encodig_test ",pgc_one_hot_test.shape)
```

```
['GradesPreK-2' 'Grades6-8' 'Grades3-5' 'Grades9-12']
['Grades3-5', 'Grades6-8', 'Grades9-12', 'GradesPreK-2']
Shape of matrix after one hot encodig_train  (49041, 4)
Shape of matrix after one hot encodig_cv  (24155, 4)
Shape of matrix after one hot encodig_test  (36052, 4)
```

In [26]:

```
pgc_show=pgc_one_hot_train[:].toarray()
```

In [28]:

```
pgc_show
```

Out[28]:

```
array([[1, 0, 0, 0],
       [1, 0, 0, 0],
       [0, 1, 0, 0],
       ...,
       [0, 0, 1, 0],
       [0, 0, 1, 0],
       [1, 0, 0, 0]], dtype=int64)
```

Numerical Features :

In [29]:

```
from sklearn.preprocessing import StandardScaler
from sklearn import preprocessing
price_scalar = StandardScaler()
price_scalar.fit(X_train['price'].values.reshape(-1,1))
price_train= price_scalar.transform(X_train['price'].values.reshape(-1, 1))
price_test= price_scalar.transform(X_test['price'].values.reshape(-1, 1))
price_cv = price_scalar.transform(X_cv['price'].values.reshape(-1, 1))
print(price_train.shape, y_train.shape)
print(price_cv.shape, y_cv.shape)
print(price_test.shape, y_test.shape)
```

```
(49041, 1) (49041,)
(24155, 1) (24155,)
(36052, 1) (36052,)
```

In [30]:

```
quantity_scaler = StandardScaler()
quantity_scaler.fit(X_train['quantity'].values.reshape(-1,1))
quantity_train = quantity_scaler.transform(X_train['quantity'].values.reshape(-1,1))
quantity_cv = quantity_scaler.transform(X_cv['quantity'].values.reshape(-1,1))
quantity_test = quantity_scaler.transform(X_test['quantity'].values.reshape(-1,1))
print("After vectorization")
print(quantity_train.shape, y_train.shape)
print(quantity_cv.shape, y_cv.shape)
print(quantity_test.shape, y_test.shape)
```

```
After vectorization
(49041, 1) (49041,)
(24155, 1) (24155,)
(36052, 1) (36052,)
```

In [31]:

```
noofprojects = StandardScaler()
noofprojects.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))
```

```
prev_projects_train =
noofprojects.transform(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1
))
prev_projects_cv = noofprojects.transform(X_cv['teacher_number_of_previously_posted_projects'].val
ues.reshape(-1,1))
prev_projects_test = noofprojects.transform(X_test['teacher_number_of_previously_posted_projects']
.values.reshape(-1,1))
print(prev_projects_train.shape, y_train.shape)
print(prev_projects_cv.shape, y_cv.shape)
print(prev_projects_test.shape, y_test.shape)
```

```
(49041, 1) (49041,)
(24155, 1) (24155,)
(36052, 1) (36052,)
```

TF-IDF Vectorizer :

In [32]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizertie = TfidfVectorizer(min_df=10,ngram_range=(1,2), max_features=5000)
vectorizertie.fit(X_train['preprocessed_essays'])
text_tfidf_train = vectorizertie.transform(X_train['preprocessed_essays'])
text_tfidf_cv = vectorizertie.transform(X_cv['preprocessed_essays'])
text_tfidf_test = vectorizertie.transform(X_test['preprocessed_essays'])
print("Shape of matrix after one hot encoding ",text_tfidf_train.shape)
print("Shape of matrix after one hot encoding ",text_tfidf_cv.shape)
print("Shape of matrix after one hot encoding ",text_tfidf_test.shape)
```

```
Shape of matrix after one hot encoding  (49041, 5000)
Shape of matrix after one hot encoding  (24155, 5000)
Shape of matrix after one hot encoding  (36052, 5000)
```

In [33]:

```
vectorizertit = TfidfVectorizer(min_df=10)
vectorizertit.fit(X_train['preprocessed_titles'])
title_tfidf_train = vectorizertit.transform(X_train['preprocessed_titles'])
title_tfidf_cv = vectorizertit.transform(X_train['preprocessed_titles'])
title_tfidf_test = vectorizertit.transform(X_train['preprocessed_titles'])
print("Shape of matrix after one hot encoding ",title_tfidf_train.shape)
print("Shape of matrix after one hot encoding ",title_tfidf_cv.shape)
print("Shape of matrix after one hot encoding ",title_tfidf_test.shape)
```

```
Shape of matrix after one hot encoding  (49041, 2108)
Shape of matrix after one hot encoding  (49041, 2108)
Shape of matrix after one hot encoding  (49041, 2108)
```

In [34]:

```
concat_essaystitles= (list(X_train['preprocessed_essays'])+list(X_test['preprocessed_titles']))
```

In [35]:

```
tf_idf_vectorizer = TfidfVectorizer()
tf_idf_vectorizer.fit_transform(concat_essaystitles)
idf_score = tf_idf_vectorizer.idf_
feature_names  = tf_idf_vectorizer.get_feature_names()
idf_score_features=[]
for i in range(len(idf_score)):
    idf_score_features.append([idf_score[i],feature_names[i]])
idf_score_features.sort(reverse=True)
idf_score_features=idf_score_features[:2000]
```

Co-Occurance Matrix :

In [37]:

In [37]:

```
coo_matrix=np.zeros((2000,2000))
window=2
```

In [38]:

```
final_2000_features=[]
for i in range(2000):
    final_2000_features.append(idf_score_features[i][1])
```

In [39]:

```
for sentance in concat_essaystitles:
    word_sen=sentance.split()
    for idss,word in enumerate(word_sen):
        if word in final_2000_features:
            for i in range(max(0,idss-window),min(idss+window,len(word_sen))):
                if word_sen[i] in final_2000_features:

coo_matrix[final_2000_features.index(word_sen[i]),final_2000_features.index(word)]+=1
```

In [40]:

```
coo_matrix
```

Out[40]:

```
array([[1., 0., 0., ..., 0., 0., 0.],
       [0., 1., 0., ..., 0., 0., 0.],
       [0., 0., 1., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 1., 0., 0.],
       [0., 0., 0., ..., 0., 1., 0.],
       [0., 0., 0., ..., 0., 0., 1.]])
```

Applying Truncated SVD :

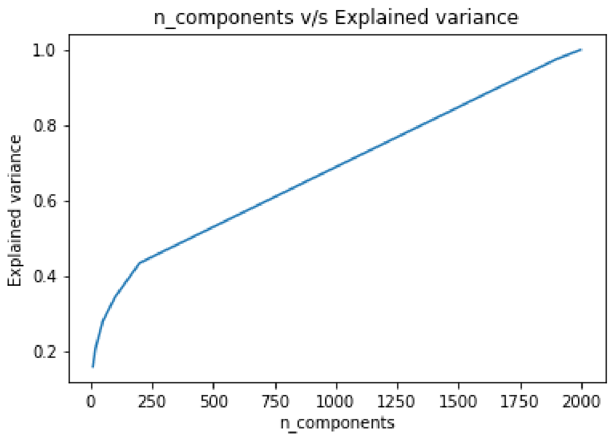
In [41]:

```
from sklearn.decomposition import TruncatedSVD
n_components=[10,20,50,60,100,200,300,400,500,1000,1200,1500,1600,1700,1800,1900,1999]
explained_variance=[]
for n in n_components:
    svd=TruncatedSVD(n_components=n,random_state=42)
    svd.fit(coo_matrix)
    exvar=svd.explained_variance_ratio_.sum()
    explained_variance.append(exvar)

    print('n_components=',n,'variance=',exvar)
```

```
n_components= 10 variance= 0.15859033693394686
n_components= 20 variance= 0.2063065895906763
n_components= 50 variance= 0.27979142866010104
n_components= 60 variance= 0.2925981583614415
n_components= 100 variance= 0.3435444470332804
n_components= 200 variance= 0.43386545118529707
n_components= 300 variance= 0.4656908117822521
n_components= 400 variance= 0.4975114973499593
n_components= 500 variance= 0.5293250551496524
n_components= 1000 variance= 0.6884210293543
n_components= 1200 variance= 0.7520424821050287
n_components= 1500 variance= 0.8475082563322406
n_components= 1600 variance= 0.8793204385522709
n_components= 1700 variance= 0.911135458515578
n_components= 1800 variance= 0.9429615845030164
n_components= 1900 variance= 0.9747751288190477
n_components= 1999 variance= 0.9999999999999802
```

In [42]:

```
#plotting curve between n_components and explained variance
plt.plot(n_components, explained_variance)
plt.xlabel('n_components')
plt.ylabel("Explained variance")
plt.title("n_components v/s Explained variance")
plt.show()
```



In [43]:

```
from sklearn.decomposition import TruncatedSVD
tsvd=TruncatedSVD(n_components=1900,random_state=42)
final_coo_matrix=tsvd.fit_transform(coo_matrix)
```

In [44]:

```
model = {}
for i in range(len(final_2000_features)):
    model[final_2000_features[i]] = final_coo_matrix[i]
```

In [45]:

```
# model = final_2000_features
glove_words =  set(model.keys())
```

In [46]:

```
# compute average word2vec for each review.
avg_w2v_titles_vectors_train = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['preprocessed_titles']): # for each review/sentence
    vector = np.zeros(1900) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_titles_vectors_train.append(vector)
```

```
100%|██████████| 49041/49041 [00:00<00:00, 97964.80it/s]
```

In [47]:

```
avg_w2v_essays_vectors_train = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['preprocessed_essays']): # for each review/sentence
    vector = np.zeros(1900) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_essays_vectors_train.append(vector)
```

```
100%|██████████| 49041/49041 [00:01<00:00, 26537.30it/s]
```

In [48]:

```
avg_w2v_titles_vectors_test = [];
for sentence in tqdm(X_test['preprocessed_titles']): # for each review/sentence
    vector = np.zeros(1900) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_titles_vectors_test.append(vector)
```

```
100%|██████████| 36052/36052 [00:00<00:00, 120683.02it/s]
```

In [49]:

```
avg_w2v_essays_vectors_test = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['preprocessed_essays']): # for each review/sentence
    vector = np.zeros(1900) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_essays_vectors_test.append(vector)
```

```
100%|██████████| 36052/36052 [00:01<00:00, 32112.49it/s]
```

In [50]:

```
avg_w2v_essays_vectors_cv = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['preprocessed_essays']): # for each review/sentence
    vector = np.zeros(1900) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_essays_vectors_cv.append(vector)
```

```
100%|██████████| 24155/24155 [00:00<00:00, 29826.40it/s]
```

In [51]:

```
avg_w2v_title_vectors_cv = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['preprocessed_titles']): # for each review/sentence
    vector = np.zeros(1900) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_title_vectors_cv.append(vector)
```

```
100%|██████████| 24155/24155 [00:00<00:00, 124058.25it/s]
```

Number Of Words in Essay & Title:

In [52]:

```
noofwordsessaytrain=[]
for i in tqdm(X_train['preprocessed_essays']):
    s = i.split(" ")
    noofwordsessaytrain.append(len(s))
noofwordsessaytrain[1]
X_train['noofwordsessay']=noofwordsessaytrain
noofwordsessaycv=[]
for i in tqdm(X_cv['preprocessed_essays']):
    s = i.split(" ")
    noofwordsessaycv.append(len(s))
noofwordsessaycv[1]
X_cv['noofwordsessay']=noofwordsessaycv
noofwordsessaytest=[]
for i in tqdm(X_test['preprocessed_essays']):
    s = i.split(" ")
    noofwordsessaytest.append(len(s))
noofwordsessaytest[1]
X_test['noofwordsessay']=noofwordsessaytest
```

```
100%|██████████| 49041/49041 [00:00<00:00, 123617.73it/s]
100%|██████████| 24155/24155 [00:00<00:00, 120912.19it/s]
100%|██████████| 36052/36052 [00:00<00:00, 124630.18it/s]
```

In [53]:

```
noofwordstitletrain=[]
for i in tqdm(X_train['preprocessed_titles']):
    s = i.split(" ")
    noofwordstitletrain.append(len(s))
X_train['noofwordstitle']=noofwordstitletrain

noofwordstitlecv=[]
for i in tqdm(X_cv['preprocessed_titles']):
    s = i.split(" ")
    noofwordstitlecv.append(len(s))
X_cv['noofwordstitle']=noofwordstitlecv

noofwordstitletest=[]
for i in tqdm(X_test['preprocessed_titles']):
    s = i.split(" ")
    noofwordstitletest.append(len(s))
X_test['noofwordstitle']=noofwordstitletest
```

```
100%|██████████| 49041/49041 [00:00<00:00, 982737.53it/s]
100%|██████████| 24155/24155 [00:00<00:00, 931497.67it/s]
100%|██████████| 36052/36052 [00:00<00:00, 977106.20it/s]
```

Computing Sentiment Scores :

In [54]:

```
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer
nltk.download('vader_lexicon')
essay_neg_train=[]
essay_pos_train=[]
essay_neu_train=[]
essay_com_train=[]
sid = SentimentIntensityAnalyzer()
for i in X_train['preprocessed_essays']:
    for_sentiment = i
    ss=sid.polarity_scores(for_sentiment)
    essay_neg_train.append(ss['neg'])
    essay_pos_train.append(ss['pos'])
    essay_neu_train.append(ss['neu'])
    essay_com_train.append(ss['compound'])
len(essay_neg_train)
```

```
X_train['essay_neg_train']=essay_neg_train
X_train['essay_pos_train']=essay_pos_train
X_train['essay_neu_train']=essay_neu_train
X_train['essay_com_train']=essay_com_train
```

```
[nltk_data] Downloading package vader_lexicon to
[nltk_data]     C:\Users\mitad\Anaconda3\lib\nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!
```

In [55]:

```
essay_neg_cv=[]
essay_pos_cv=[]
essay_neu_cv=[]
essay_com_cv=[]
sid = SentimentIntensityAnalyzer()
for i in X_cv['preprocessed_essays']:
    for_sentiment = i
    ss=sid.polarity_scores(for_sentiment)
    essay_neg_cv.append(ss['neg'])
    essay_pos_cv.append(ss['pos'])
    essay_neu_cv.append(ss['neu'])
    essay_com_cv.append(ss['compound'])
len(essay_neg_cv)
X_cv['essay_neg_cv']=essay_neg_cv
X_cv['essay_pos_cv']=essay_pos_cv
X_cv['essay_neu_cv']=essay_neu_cv
X_cv['essay_com_cv']=essay_com_cv
```

In [56]:

```
essay_neg_test=[]
essay_pos_test=[]
essay_neu_test=[]
essay_com_test=[]
sid = SentimentIntensityAnalyzer()
for i in X_test['preprocessed_essays']:
    for_sentiment = i
    ss=sid.polarity_scores(for_sentiment)
    essay_neg_test.append(ss['neg'])
    essay_pos_test.append(ss['pos'])
    essay_neu_test.append(ss['neu'])
    essay_com_test.append(ss['compound'])
len(essay_neg_test)
X_test['essay_neg_test']=essay_neg_test
X_test['essay_pos_test']=essay_pos_test
X_test['essay_neu_test']=essay_neu_test
X_test['essay_com_test']=essay_com_test
```

In [57]:

```
essay_neg_test=X_test['essay_neg_test'].values.reshape(-1,1)
essay_pos_test=X_test['essay_pos_test'].values.reshape(-1,1)
essay_neu_test=X_test['essay_pos_test'].values.reshape(-1,1)
essay_com_test=X_test['essay_pos_test'].values.reshape(-1,1)
essay_neg_cv=X_cv['essay_pos_cv'].values.reshape(-1,1)
essay_pos_cv=X_cv['essay_pos_cv'].values.reshape(-1,1)
essay_neu_cv=X_cv['essay_pos_cv'].values.reshape(-1,1)
essay_com_cv=X_cv['essay_pos_cv'].values.reshape(-1,1)
essay_neg_train=X_train['essay_neg_train'].values.reshape(-1,1)
essay_pos_train=X_train['essay_pos_train'].values.reshape(-1,1)
essay_neu_train=X_train['essay_neu_train'].values.reshape(-1,1)
essay_com_train=X_train['essay_com_train'].values.reshape(-1,1)
```

In [58]:

```
noofwordse = StandardScaler()
noofwordse.fit(X_train['noofwordsessay'].values.reshape(-1,1))
noessay_train = noofwordse.transform(X_train['noofwordsessay'].values.reshape(-1,1))
noessay_cv = noofwordse.transform(X_cv['noofwordsessay'].values.reshape(-1,1))
noessay_test = noofwordse.transform(X_test['noofwordsessay'].values.reshape(-1,1))
print(noessay_train.shape, y_train.shape)
print(noessay_cv.shape, y_cv.shape)
print(noessay_test.shape, y_test.shape)
```

```
(49041, 1) (49041,)
(24155, 1) (24155,)
(36052, 1) (36052,)
```

In [59]:

```
noofwordst = StandardScaler()
noofwordst.fit(X_train['noofwordstitle'].values.reshape(-1,1))
notitle_train = noofwordst.transform(X_train['noofwordstitle'].values.reshape(-1,1))
notitle_cv = noofwordst.transform(X_cv['noofwordstitle'].values.reshape(-1,1))
notitle_test = noofwordst.transform(X_test['noofwordstitle'].values.reshape(-1,1))
print(notitle_train.shape, y_train.shape)
print(notitle_cv.shape, y_cv.shape)
print(notitle_test.shape, y_test.shape)
```

```
(49041, 1) (49041,)
(24155, 1) (24155,)
(36052, 1) (36052,)
```

Merging all the above features

In [60]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr =
hstack((avg_w2v_titles_vectors_train,avg_w2v_essays_vectors_train,essay_neg_train,essay_pos_train,
essay_neu_train,essay_com_train,noessay_train,notitle_train,school_state_one_hot_train,pgc_one_hot_
train,teacher_prefix_one_hot_train,categories_one_hot_train,sub_categories_one_hot_train,prev_proje
cts_train,price_train,quantity_train)).tocsr()
X_cr =
hstack((avg_w2v_title_vectors_cv,avg_w2v_essays_vectors_cv,essay_neg_cv,essay_pos_cv,essay_neu_cv,
essay_com_cv,noessay_cv,notitle_cv,school_state_one_hot_cv,pgc_one_hot_cv,teacher_prefix_one_hot_cv
,categories_one_hot_cv,sub_categories_one_hot_cv,prev_projects_cv,price_cv,quantity_cv)).tocsr()
X_te =
hstack((avg_w2v_titles_vectors_test,avg_w2v_essays_vectors_test,essay_neg_test,essay_pos_test,essa
y_neu_test,essay_com_test,noessay_test,notitle_test,school_state_one_hot_test,pgc_one_hot_test,tea
cher_prefix_one_hot_test,categories_one_hot_test,sub_categories_one_hot_test,prev_projects_test,pr
ice_test,quantity_test)).tocsr()

print("Final Data matrix")
print(X_tr.shape, y_train.shape)
print(X_cr.shape, y_cv.shape)
print(X_te.shape, y_test.shape)
```

```
Final Data matrix
(49041, 3908) (49041,)
(24155, 3908) (24155,)
(36052, 3908) (36052,)
```

In [61]:

```
X_tr
```

Out[61]:

```
<49041x3908 sparse matrix of type '<class 'numpy.float64'>'
 with 4127051 stored elements in Compressed Sparse Row format>
```

Assignment 11: TruncatedSVD

- step 1 Select the top 2k words from essay text and project_title (concatinate essay text with project title and then find the top 2k words) based on their `idf_` values
- step 2 Compute the co-occurance matrix with these 2k words, with window size=5 (ref)

- step 2 Compute the co-occurance matrix with these 2k words, with window size=5 (ref)

- step 3 Use TruncatedSVD on calculated co-occurance matrix and reduce its dimensions, choose the number of components (`n_components`) using elbow method

  > - The shape of the matrix after TruncatedSVD will be 2000*n, i.e. each row represents a vector form of the corresponding word.
  > - Vectorize the essay text and project titles using these word vectors. (while vectorizing, do ignore all the words which are not in top 2k words)

- step 4 Concatenate these truncatedSVD matrix, with the matrix with features
  - **school_state** : categorical data
  - **clean_categories** : categorical data
  - **clean_subcategories** : categorical data
  - **project_grade_category** :categorical data
  - **teacher_prefix** : categorical data
  - **quantity** : numerical data
  - **teacher_number_of_previously_posted_projects** : numerical data
  - **price** : numerical data
  - **sentiment score's of each of the essay** : numerical data
  - **number of words in the title** : numerical data
  - **number of words in the combine essays** : numerical data
  - **word vectors calculated in** step 3 : numerical data
- step 5: Apply GBDT on matrix that was formed in step 4 of this assignment, **DO REFER THIS BLOG: XGBOOST DMATRIX**
- **step 6:Hyper parameter tuning (Consider any two hyper parameters)**
  - **Find the best hyper parameter which will give the maximum AUC value**
  - **Find the best hyper paramter using k-fold cross validation or simple cross validation data**
  - **Use gridsearch cv or randomsearch cv or you can also write your own for loops to do this task of hyperparameter tuning**

In [0]:

```
import sys
import math

import numpy as np
from sklearn.grid_search import GridSearchCV
from sklearn.metrics import roc_auc_score

# you might need to install this one
import xgboost as xgb

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

    def fit(self, X, y, num_boost_round=None):
        num_boost_round = num_boost_round or self.num_boost_round
        self.label2num = {label: i for i, label in enumerate(sorted(set(y)))}
        dtrain = xgb.DMatrix(X, label=[self.label2num[label] for label in y])
        self.clf = xgb.train(params=self.params, dtrain=dtrain, num_boost_round=num_boost_round, ve
rbose_eval=1)

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

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

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

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

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


clf = XGBoostClassifier(eval_metric = 'auc', num_class = 2, nthread = 4,)
###################################################################
#               Change from here                                  #
###################################################################
parameters = {
    'num_boost_round': [100, 250, 500],
    'eta': [0.05, 0.1, 0.3],
    'max_depth': [6, 9, 12],
    'subsample': [0.9, 1.0],
    'colsample_bytree': [0.9, 1.0],
}

clf = GridSearchCV(clf, parameters)
X = np.array([[1,2], [3,4], [2,1], [4,3], [1,0], [4,5]])
Y = np.array([0, 1, 0, 1, 0, 1])
clf.fit(X, Y)

# print(clf.grid_scores_)
best_parameters, score, _ = max(clf.grid_scores_, key=lambda x: x[1])
print('score:', score)
for param_name in sorted(best_parameters.keys()):
    print("%s: %r" % (param_name, best_parameters[param_name]))
```

```
score: 0.8333333333333334
colsample_bytree: 0.9
eta: 0.05
max_depth: 6
num_boost_round: 100
subsample: 0.9
```

2. TruncatedSVD

Apply XGBoost on the Final Features from the above section

In [62]:

```
from xgboost import XGBClassifier
from sklearn.metrics import roc_auc_score
from sklearn.metrics import accuracy_score
main_score_trdata = []
main_score_cvdata = []
n_estimators = [10, 50, 100, 150, 200,300]
learning_rate = [0.0001, 0.001, 0.01, 0.1, 0.2, 0.3]
for i in tqdm(n_estimators):
    train_auc = []
    cv_auc = []
    for j in learning_rate:
        rf = XGBClassifier(n_estimators=i,learning_rate=j,booster='gbtree',n_jobs=-1,class_weight='
balanced')
        rf.fit(X_tr, y_train)
        y_train_pred=rf.predict_proba(X_tr)[:, 1]
        y_cv_pred=rf.predict_proba(X_cr)[:, 1]
        train_auc.append(roc_auc_score(y_train,y_train_pred))
        cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
    main_score_trdata.append(train_auc)
    main_score_cvdata.append(cv_auc)
```

```
100%|██████████| 6/6 [11:52<00:00, 118.76s/it]
```

In [63]:

In [63]:

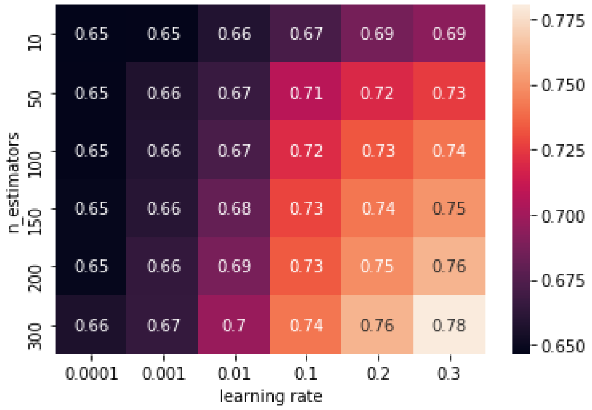
```
train_auc_df = pd.DataFrame(data=main_score_trdata, index=n_estimators, columns=learning_rate)
cv_auc_df = pd.DataFrame(data=main_score_cvdata, index=n_estimators, columns=learning_rate)
```

In [64]:

```
heat_train=sns.heatmap(data=train_auc_df,annot=True)
heat_train.set(xlabel='learning rate', ylabel='n_estimators')
```

Out[64]:

```
[Text(33.0, 0.5, 'n_estimators'), Text(0.5, 15.0, 'learning rate')]
```

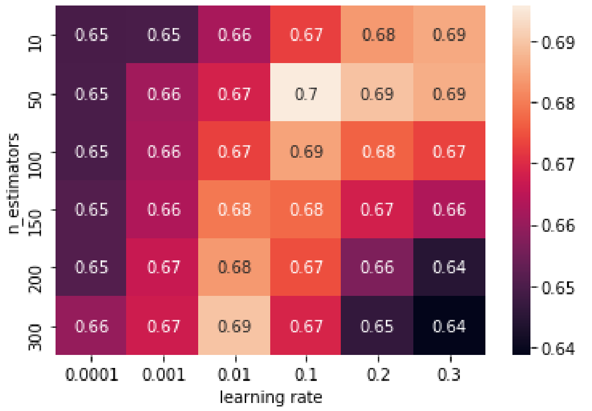


In [65]:

```
heat_cv=sns.heatmap(data=cv_auc_df,annot=True)
heat_cv.set(xlabel='learning rate', ylabel='n_estimators')
```

Out[65]:

```
[Text(33.0, 0.5, 'n_estimators'), Text(0.5, 15.0, 'learning rate')]
```



In [66]:

```
best_n_estimators = 50
best_learning_rate = 0.1
```

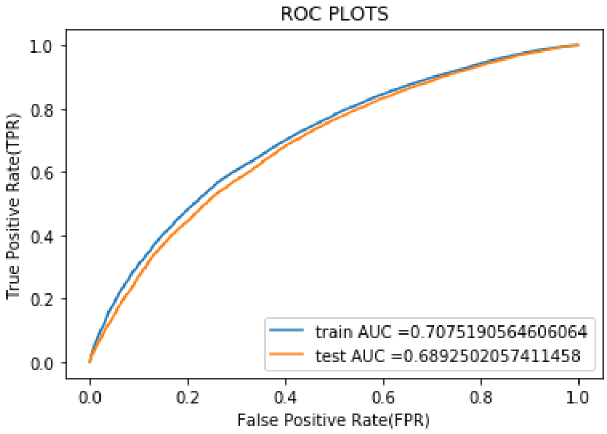
In [67]:

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

    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000
    # consider you X_tr shape is 49041, then your cr_loop will be 49041 - 49041%1000 = 49000
    # in this for loop we will iterate unti the last 1000 multiplier
    for i in range(0, tr_loop, 1000):
```

```
    for i in range(0, tr_loop, 1000):
        y_data_pred.extend(clf.predict_proba(data[i:i+1000])[:,1])
    # we will be predicting for the last data points
    y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

    return y_data_pred
```

In [68]:

```
from sklearn.metrics import roc_curve, auc
rf =
XGBClassifier(n_estimators=best_n_estimators,learning_rate=best_learning_rate,booster='gbtree',n_j
obs=-1,class_weight='balanced')
rf.fit(X_tr, y_train)
y_train_pred = batch_predict(rf,X_tr)
y_test_pred = batch_predict(rf,X_te)
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(FPR)")
plt.ylabel("True Positive Rate(TPR)")
plt.title("ROC PLOTS")
plt.show()
print("="*100)
```



```
====================================================================================================
```

Confusion Matrix :

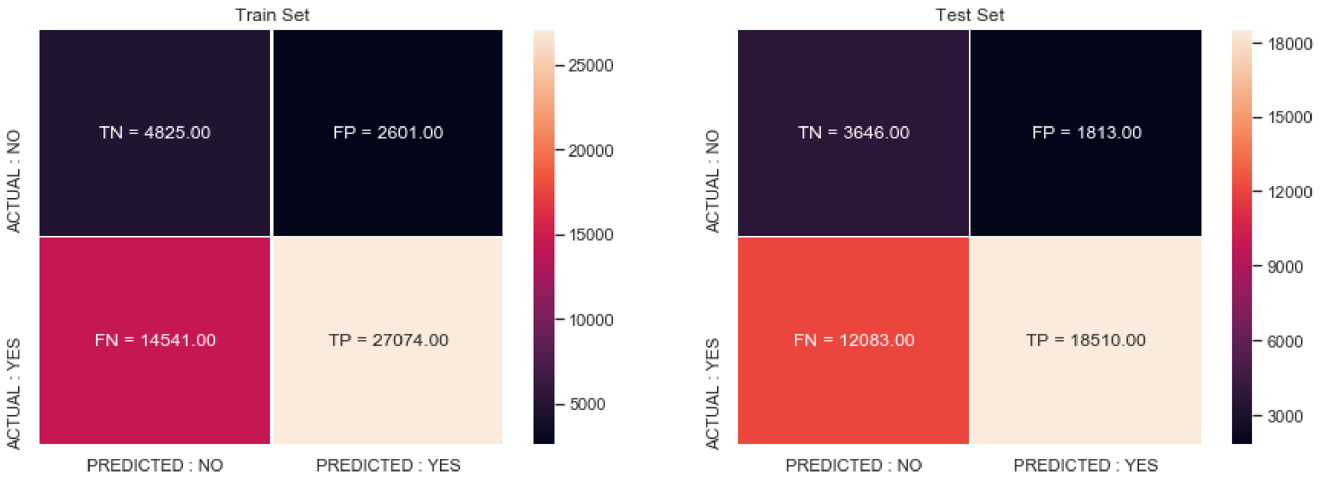
In [69]:

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

    t = threshould[np.argmax(fpr*(1-tpr))]
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
    predictions = []
    global predictions1
    for i in proba:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)
    predictions1 = predictions
    return predictions
```

In [70]:

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

con_m_train = confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr,
train_tpr))
con_m_test = confusion_matrix(y_test, predict(y_test_pred, te_thresholds, test_fpr, test_tpr))

key = (np.asarray([['TN','FP'], ['FN', 'TP']]))
fig, ax = plt.subplots(1,2, figsize=(15,5))

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

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

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

plt.show()
```

```
the maximum value of tpr*(1-fpr) 0.42472620803364713 for threshold 0.842
the maximum value of tpr*(1-fpr) 0.40982503345799126 for threshold 0.824
```



3. Conclusion

In [71]:

```
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Vectorizer", "Model", "Hyper Parameter", "AUC"]
x.add_row(["AVG W2V", "XGBoost", "learning rate:0.1 , n_estimators:150", 0.68])
print(x)
```

```
+------------+---------+--------------------------------------+------+
| Vectorizer |  Model  |           Hyper Parameter            | AUC  |
+------------+---------+--------------------------------------+------+
|  AVG W2V   | XGBoost | learning rate:0.1 , n_estimators:150 | 0.68 |
+------------+---------+--------------------------------------+------+
```