

# 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
<code>project_id</code>	A unique identifier for the proposed project. <b>Example:</b> p036502
<code>project_title</code>	Title of the project. <b>Examples:</b> <ul style="list-style-type: none"><li>• Art Will Make You Happy!</li><li>• First Grade Fun</li></ul>
<code>project_grade_category</code>	Grade level of students for which the project is targeted. One of the following enumerated values: <ul style="list-style-type: none"><li>• Grades PreK-2</li><li>• Grades 3-5</li><li>• Grades 6-8</li><li>• Grades 9-12</li></ul>
<code>project_subject_categories</code>	One or more (comma-separated) subject categories for the project from the following enumerated list of values: <ul style="list-style-type: none"><li>• Applied Learning</li><li>• Care &amp; Hunger</li><li>• Health &amp; Sports</li><li>• History &amp; Civics</li><li>• Literacy &amp; Language</li><li>• Math &amp; Science</li><li>• Music &amp; The Arts</li><li>• Special Needs</li><li>• Warmth</li></ul> <b>Examples:</b> <ul style="list-style-type: none"><li>• Music &amp; The Arts</li><li>• Literacy &amp; Language, Math &amp; Science</li></ul>
<code>school_state</code>	State where school is located ( <a href="#">Two-letter U.S. postal code</a> ). <b>Example:</b> WY
<code>project_subject_subcategories</code>	One or more (comma-separated) subject subcategories for the project. <b>Examples:</b> <ul style="list-style-type: none"><li>• Literacy</li></ul>

Feature	Description
<code>project_resource_summary</code>	An explanation of the resources needed for the project. <b>Example:</b> <ul style="list-style-type: none"> <li>• My students need hands on literacy materials to manage sensory needs!</li> </ul>
<code>project_essay_1</code>	First application essay*
<code>project_essay_2</code>	Second application essay*
<code>project_essay_3</code>	Third application essay*
<code>project_essay_4</code>	Fourth application essay*
<code>project_submitted_datetime</code>	Datetime when project application was submitted. <b>Example:</b> 2016-04-28 12:43:56.245
<code>teacher_id</code>	A unique identifier for the teacher of the proposed project. <b>Example:</b> bdf8baa8fedef6bfeec7ae4ff1c15c56
<code>teacher_prefix</code>	Teacher's title. One of the following enumerated values: <ul style="list-style-type: none"> <li>• nan</li> <li>• Dr.</li> <li>• Mr.</li> <li>• Mrs.</li> <li>• Ms.</li> <li>• Teacher.</li> </ul>
<code>teacher_number_of_previously_posted_projects</code>	Number of project applications previously submitted by the same teacher. <b>Example:</b> 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
<code>id</code>	A <code>project_id</code> value from the <code>train.csv</code> file. <b>Example:</b> p036502
<code>description</code>	Description of the resource. <b>Example:</b> Tenor Saxophone Reeds, Box of 25
<code>quantity</code>	Quantity of the resource required. <b>Example:</b> 3
<code>price</code>	Price of the resource required. <b>Example:</b> 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
<code>project_is_approved</code>	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_4__` "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."

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 [3]:

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

import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from nltk.stem.porter import PorterStemmer

import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer

from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle

from tqdm import tqdm
import os

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

from google.colab import drive
drive.mount('/content/gdrive')
```

Drive already mounted at /content/gdrive; to attempt to forcibly remount, call `drive.mount("/content/gdrive", force_remount=True)`.

## 1.1 Reading Data

In [0]:

```
project_data = pd.read_csv('gdrive/My Drive/train_data.csv')
resource_data = pd.read_csv('gdrive/My Drive/resources.csv')
```

In [5]:

```
project_data.columns
```

Out[5]:

```
Index(['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_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'],
dtype='object')
```

In [6]:

```
project_data[0:2000]['project_is_approved'].value_counts()
```

Out[6]:

```
1    1699
0     301
Name: project_is_approved, dtype: int64
```

In [7]:

```
project_data[0:4000]['project_is_approved'].value_counts()
```

Out[7]:

```
1    3392
0     608
Name: project_is_approved, dtype: int64
```

In [8]:

```
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 [9]:

```
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[9]:

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

## 1.2 Data Analysis

In [10]:

```
# PROVIDE CITATIONS TO YOUR CODE IF YOU TAKE IT FROM ANOTHER WEBSITE.
# https://matplotlib.org/gallery/pie_and_polar_charts/pie_and_donut_labels.html#sphx-glr-gallery-p
ie-and-polar-charts-pie-and-donut-labels-py
```

```
y_value_counts = project_data['project_is_approved'].value_counts()
print("Number of projects thar are approved for funding ", y_value_counts[1], ", (",
```

```

(y_value_counts[1]/(y_value_counts[1]+y_value_counts[0]))*100,"%")
print("Number of projects thar are not approved for funding ", y_value_counts[0], ", (",
(y_value_counts[0]/(y_value_counts[1]+y_value_counts[0]))*100,"%")

fig, ax = plt.subplots(figsize=(6, 6), subplot_kw=dict(aspect="equal"))
recipe = ["Accepted", "Not Accepted"]

data = [y_value_counts[1], y_value_counts[0]]

wedges, texts = ax.pie(data, wedgeprops=dict(width=0.5), startangle=-40)

bbox_props = dict(boxstyle="square,pad=0.3", fc="w", ec="k", lw=0.72)
kw = dict(xycoords='data', textcoords='data', arrowprops=dict(arrowstyle="-"),
          bbox=bbox_props, zorder=0, va="center")

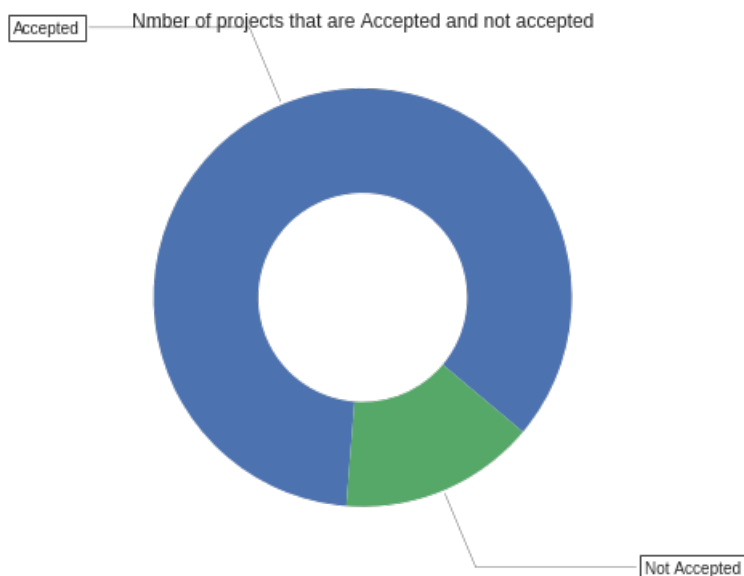
for i, p in enumerate(wedges):
    ang = (p.theta2 - p.theta1)/2. + p.theta1
    y = np.sin(np.deg2rad(ang))
    x = np.cos(np.deg2rad(ang))
    horizontalalignment = {-1: "right", 1: "left"}[int(np.sign(x))]
    connectionstyle = "angle,angleA=0,angleB={}".format(ang)
    kw["arrowprops"].update({"connectionstyle": connectionstyle})
    ax.annotate(recipe[i], xy=(x, y), xytext=(1.35*np.sign(x), 1.4*y),
                horizontalalignment=horizontalalignment, **kw)

ax.set_title("Nmber of projects that are Accepted and not accepted")

plt.show()

```

Number of projects thar are approved for funding 92706 , ( 84.85830404217927 %)  
 Number of projects thar are not approved for funding 16542 , ( 15.141695957820739 %)



1. We took the feature "project is approved" which is output for Project Data and calculated percentages for approved and not approved
2. The above pie chart shows the percentage of projects which are approved in Blue which is 84.85830404217927 % and the percentage of projects which are not approved in Orange which is 15.141695957820739 %.
3. We calculated approved and not approved projects using y\_value\_counts which gives details of no. of projects submitted and how many of them are approved and not approved

## 1.2.1 Univariate Analysis: School State

In [11]:

```

# Pandas dataframe groupby count, mean: https://stackoverflow.com/a/19385591/4084039

temp = pd.DataFrame(project_data.groupby("school_state")
["project_is_approved"].apply(np.mean)).reset_index()
# if you have data which contain only 0 and 1, then the mean = percentage (think about it)

```



```
#stacked bar plots matplotlib:
https://matplotlib.org/gallery/lines_bars_and_markers/bar_stacked.html
def stack_plot(data, xtick, col2='project_is_approved', col3='total'):
    ind = np.arange(data.shape[0])

    plt.figure(figsize=(20,5))
    p1 = plt.bar(ind, data[col3].values)
    p2 = plt.bar(ind, data[col2].values)

    plt.ylabel('Projects')
    plt.title('Number of projects aproved vs rejected')
    plt.xticks(ind, list(data[xtick].values))
    plt.legend((p1[0], p2[0]), ('total', 'accepted'))
    plt.show()
```

In [14]:

```
def univariate_barplots(data, col1, col2='project_is_approved', top=False):
    # Count number of zeros in dataframe python: https://stackoverflow.com/a/51540521/4084039
    temp = pd.DataFrame(project_data.groupby(col1)[col2].agg(lambda x: x.eq(1).sum()).reset_index()
    )

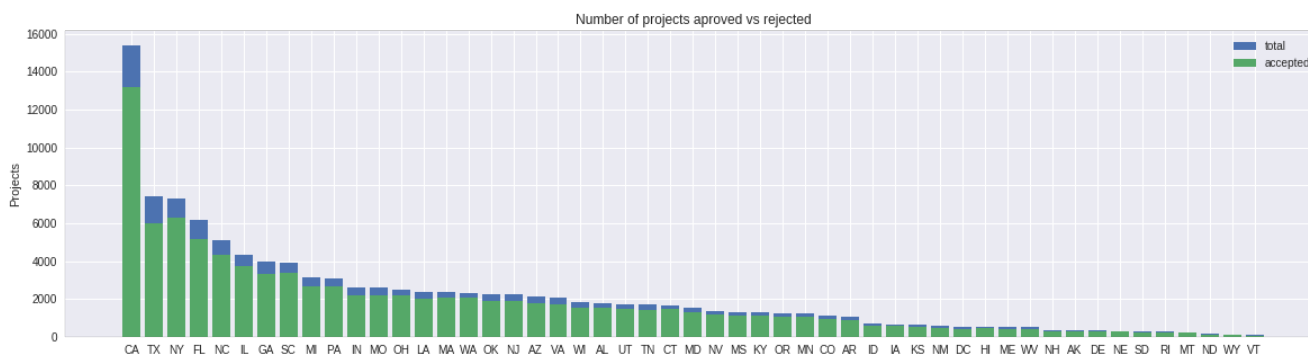
    # Pandas dataframe grouby count: https://stackoverflow.com/a/19385591/4084039
    temp['total'] = pd.DataFrame(project_data.groupby(col1)
    [col2].agg({'total': 'count'})).reset_index()['total']
    temp['Avg'] = pd.DataFrame(project_data.groupby(col1)[col2].agg({'Avg': 'mean'})).reset_index()['Avg']

    temp.sort_values(by=['total'], inplace=True, ascending=False)

    if top:
        temp = temp[0:top]

    stack_plot(temp, xtick=col1, col2=col2, col3='total')
    print(temp.head(5))
    print("="*50)
    print(temp.tail(5))

univariate_barplots(project_data, 'school_state', 'project_is_approved', False)
```



	school_state	project_is_approved	total	Avg
4	CA	13205	15388	0.858136
43	TX	6014	7396	0.813142
34	NY	6291	7318	0.859661
9	FL	5144	6185	0.831690
27	NC	4353	5091	0.855038
=====				
39	RI	243	285	0.852632
26	MT	200	245	0.816327
28	ND	127	143	0.888112
50	WY	82	98	0.836735
46	VT	64	80	0.800000

1. we calculated no. of projects submitted per each state and calculated how many of them are approved and rejected from that state
2. we made a barplot of how many total projects are submitted from each state and how many are accepted and rejected from that state

3.We calculated acceptance rate which is percentage of projects submitted and accepted and rejected from particular state and we sorted the acceptance rate

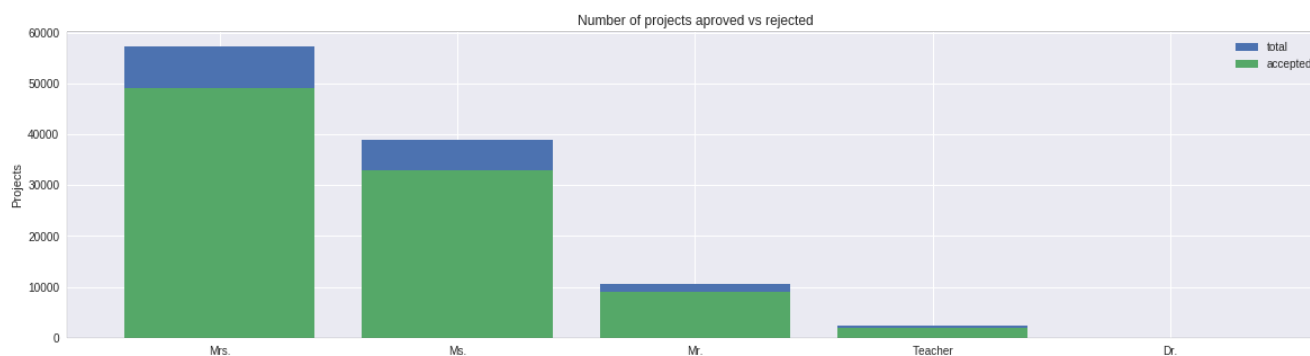
1. projects from CA have been accepted and rejected more

**SUMMARY: Every state has greater than 80% success rate in approval**

### 1.2.2 Univariate Analysis: teacher\_prefix

In [15]:

```
univariate_barplots(project_data, 'teacher_prefix', 'project_is_approved', top=False)
```



	teacher_prefix	project_is_approved	total	Avg
2	Mrs.	48997	57269	0.855559
3	Ms.	32860	38955	0.843537
1	Mr.	8960	10648	0.841473
4	Teacher	1877	2360	0.795339
0	Dr.	9	13	0.692308

=====

	teacher_prefix	project_is_approved	total	Avg
2	Mrs.	48997	57269	0.855559
3	Ms.	32860	38955	0.843537
1	Mr.	8960	10648	0.841473
4	Teacher	1877	2360	0.795339
0	Dr.	9	13	0.692308

1.based on the prefixe's of the teacher's who submitted projects,we calculated what are the total no.of.projects submitted by a teacher of particular prefix and what is the approval rate that that project is approved.

2.we calculated the approval rate of project acceptance of a teacher of a particular prefix

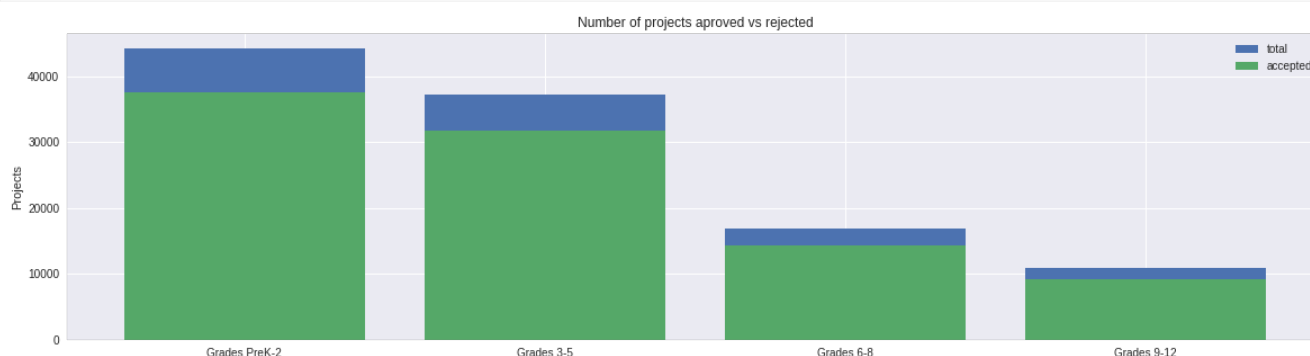
3.projects submitted by teacher of prefix-MRS are accepted more

4.no.of.projects approved and rejected by teacher of certain specific prefix

### 1.2.3 Univariate Analysis: project\_grade\_category

In [16]:

```
univariate_barplots(project_data, 'project_grade_category', 'project_is_approved', top=False)
```





	project_grade_category	project_is_approved	total	Avg
3	Grades PreK-2	37536	44225	0.848751
0	Grades 3-5	31729	37137	0.854377
1	Grades 6-8	14258	16923	0.842522
2	Grades 9-12	9183	10963	0.837636

	project_grade_category	project_is_approved	total	Avg
3	Grades PreK-2	37536	44225	0.848751
0	Grades 3-5	31729	37137	0.854377
1	Grades 6-8	14258	16923	0.842522
2	Grades 9-12	9183	10963	0.837636

1.we calculated the acceptance rate based on the grades of projects submitted and how many are accepted and rejected from that specific grade

2.projects from Grade preK-2 are submitted more

## 1.2.4 Univariate Analysis: project\_subject\_categories

In [0]:

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

# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-string
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-python
cat_list = []
for i in categories:
    temp = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
    for j in i.split(','): # it will split it in three parts ["Math & Science", "Warmth", "Care & Hunger"]
        if 'The' in j.split(): # this will split each of the category based on space "Math & Science" => "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 placing 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())
```

In [18]:

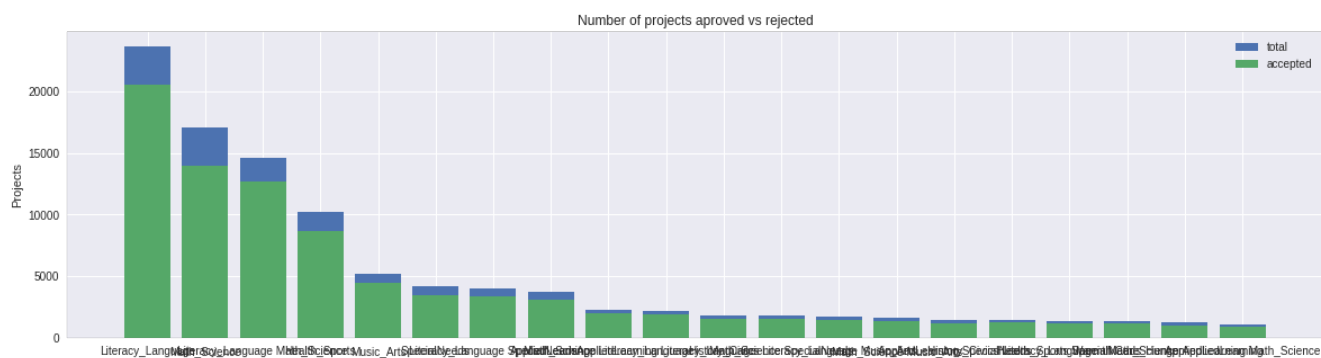
```
project_data['clean_categories'] = cat_list
project_data.drop(['project_subject_categories'], axis=1, inplace=True)
project_data.head(2)
```

Out[18]:

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

In [19]:

```
univariate_barplots(project_data, 'clean_categories', 'project_is_approved', top=20)
```



	clean_categories	project_is_approved	total	Avg
24	Literacy_Language	20520	23655	0.867470
32	Math_Science	13991	17072	0.819529
28	Literacy_Language Math_Science	12725	14636	0.869432
8	Health_Sports	8640	10177	0.848973
40	Music_Arts	4429	5180	0.855019

	clean_categories	project_is_approved	total	Avg
19	History_Civics Literacy_Language	1271	1421	0.894441
14	Health_Sports SpecialNeeds	1215	1391	0.873472
50	Warmth Care_Hunger	1212	1309	0.925898
33	Math_Science AppliedLearning	1019	1220	0.835246
4	AppliedLearning Math_Science	855	1052	0.812738

1. projects from literacy\_language subcategories have been accepted more
2. we calculated projects submitted of a specific sub\_category
3. We calculated what are the total no.of.projects submitted from specific catgory and how many are accepted and rejected from that category

In [0]:

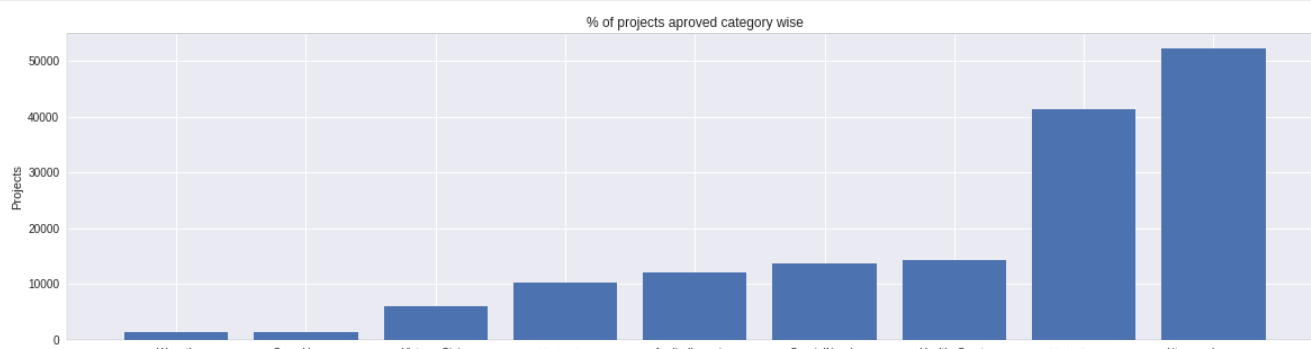
```
# count of all the words in corpus python: https://stackoverflow.com/a/22898595/4084039
from collections import Counter
my_counter = Counter()
for word in project_data['clean_categories'].values:
    my_counter.update(word.split())
```

In [21]:

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

ind = np.arange(len(sorted_cat_dict))
plt.figure(figsize=(20,5))
p1 = plt.bar(ind, list(sorted_cat_dict.values()))

plt.ylabel('Projects')
plt.title('% of projects aproved category wise')
plt.xticks(ind, list(sorted_cat_dict.keys()))
plt.show()
```



We calculated how many projected are approved from a specific category and we do plot how many are rejected and how many are submitted

we can say that literacy\_language projects are more accepted

In [22]:

```
for i, j in sorted_cat_dict.items():
    print("{:20} {:10}".format(i, j))
```

```
Warmth           :      1388
Care_Hunger      :      1388
History_Civics   :       5914
Music_Arts       :     10293
AppliedLearning  :     12135
SpecialNeeds     :     13642
Health_Sports    :     14223
Math_Science     :     41421
Literacy_Language :     52239
```

## 1.2.5 Univariate Analysis: project\_subject\_subcategories

In [0]:

```
sub_categories = 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_categories:
    temp = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
    for j in i.split(','): # it will split it in three parts ["Math & Science", "Warmth", "Care & Hunger"]
        if 'The' in j.split(): # this will split each of the category based on space "Math & Science" => "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 placing 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())
```

In [24]:

```
project_data['clean_subcategories'] = sub_cat_list
project_data.drop(['project_subject_subcategories'], axis=1, inplace=True)
project_data.head(2)
```

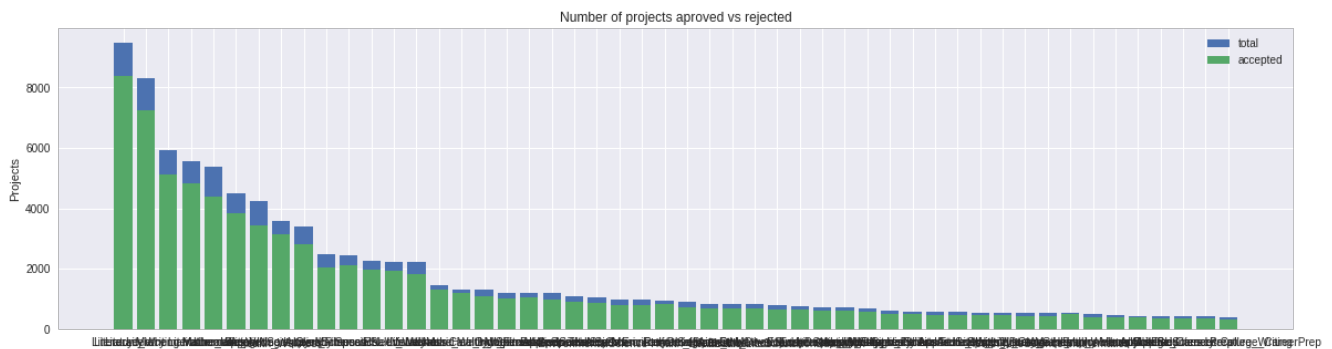
Out[24]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime	pro
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	2016-12-05 13:43:57	Gra
1	140945	p258326	897464ce9ddc600bcd1151f324dd63a	Mr.	FL	2016-10-25 09:22:10	Gra

Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime	pro
------------	----	------------	----------------	--------------	----------------------------	-----

In [25]:

```
univariate_barplots(project_data, 'clean_subcategories', 'project_is_approved', top=50)
```



	clean_subcategories	project_is_approved	total	Avg
317	Literacy	8371	9486	0.882458
319	Literacy Mathematics	7260	8325	0.872072
331	Literature_Writing Mathematics	5140	5923	0.867803
318	Literacy Literature_Writing	4823	5571	0.865733
342	Mathematics	4385	5379	0.815207

=====

	clean_subcategories	project_is_approved	total	Avg
196	EnvironmentalScience Literacy	389	444	0.876126
127	ESL	349	421	0.828979
79	College_CareerPrep	343	421	0.814727
17	AppliedSciences Literature_Writing	361	420	0.859524
3	AppliedSciences College_CareerPrep	330	405	0.814815

1. how many projects are submitted of a particular sub\_category and how many are approved and rejected.

2.literacy sub\_category are submitted more and are accepted more

In [0]:

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

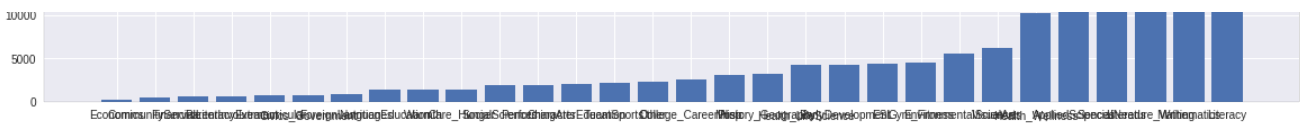
In [27]:

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

ind = np.arange(len(sorted_sub_cat_dict))
plt.figure(figsize=(20,5))
p1 = plt.bar(ind, list(sorted_sub_cat_dict.values()))

plt.ylabel('Projects')
plt.title('% of projects aproved state wise')
plt.xticks(ind, list(sorted_sub_cat_dict.keys()))
plt.show()
```





1. total no.of.projects approved for a specific sub\_category
2. Literacy subcategory projects are accepted more

In [28]:

```
for i, j in sorted_sub_cat_dict.items():
    print("{:20} {:10}".format(i,j))
```

```
Economics          :      269
CommunityService   :      441
FinancialLiteracy   :      568
ParentInvolvement  :      677
Extracurricular    :      810
Civics_Government  :      815
ForeignLanguages    :      890
NutritionEducation :     1355
Warmth             :     1388
Care_Hunger        :     1388
SocialSciences     :     1920
PerformingArts     :     1961
CharacterEducation  :     2065
TeamSports         :     2192
Other              :     2372
College_CareerPrep :     2568
Music              :     3145
History_Geography  :     3171
Health_LifeScience :     4235
EarlyDevelopment   :     4254
ESL                :     4367
Gym_Fitness        :     4509
EnvironmentalScience :    5591
VisualArts         :     6278
Health_Wellness    :    10234
AppliedSciences    :    10816
SpecialNeeds       :    13642
Literature_Writing :    22179
Mathematics        :    28074
Literacy           :   33700
```

## 1.2.6 Univariate Analysis: Text features (Title)

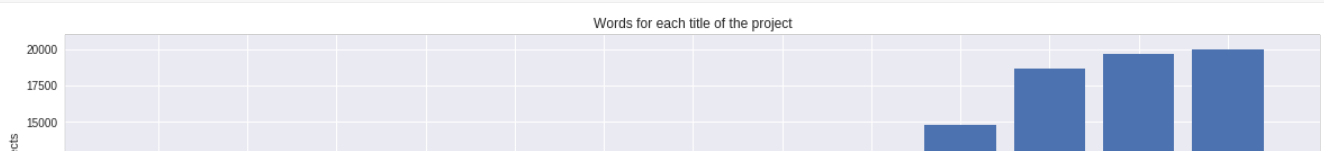
In [29]:

```
#How to calculate number of words in a string in DataFrame:
https://stackoverflow.com/a/37483537/4084039
word_count = project_data['project_title'].str.split().apply(len).value_counts()
word_dict = dict(word_count)

word_dict = dict(sorted(word_dict.items(), key=lambda kv: kv[1]))

ind = np.arange(len(word_dict))
plt.figure(figsize=(20,5))
p1 = plt.bar(ind, list(word_dict.values()))

plt.ylabel('Numeber of projects')
plt.xlabel('Numeber words in project title')
plt.title('Words for each title of the project')
plt.xticks(ind, list(word_dict.keys()))
plt.show()
```





1.No.of.words in the project title and are the projects accepted based on no.of.words in project title

2project title with more no.of.words are accepted more

In [0]:

```
approved_title_word_count = project_data[project_data['project_is_approved']==1]['project_title'].str.split().apply(len)
approved_title_word_count = approved_title_word_count.values

rejected_title_word_count = project_data[project_data['project_is_approved']==0]['project_title'].str.split().apply(len)
rejected_title_word_count = rejected_title_word_count.values
```

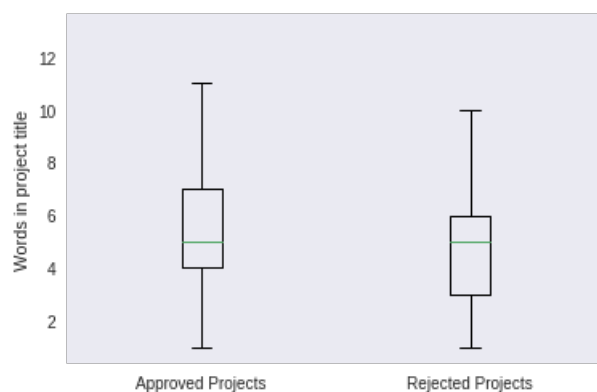
In [31]:

```
word_count_titles = project_data['project_title'].str.split().apply(len)
word_count_titles = word_count_titles.values
print(word_count_titles)
```

[7 5 7 ... 6 5 7]

In [32]:

```
# https://glowingpython.blogspot.com/2012/09/boxplot-with-matplotlib.html
plt.boxplot([approved_title_word_count, rejected_title_word_count])
plt.xticks([1,2], ('Approved Projects', 'Rejected Projects'))
plt.ylabel('Words in project title')
plt.grid()
plt.show()
```



We created a box plot for numerical data for approved projects and rejected projects under project\_title category

we found few outliers for approved projects and rejected projects

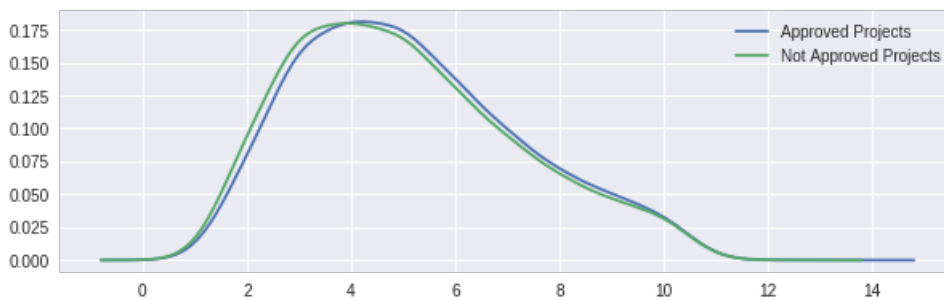
we found the IQR range for approved projects and rejected projects and found the range for approved projects under project\_title category is High and the for rejected projects is low.

The mean for approved projects and rejected projects is Same though the IQR range is different.

In [33]:

```
plt.figure(figsize=(10,3))
sns.kdeplot(approved_title_word_count, label="Approved Projects", bw=0.6)
sns.kdeplot(rejected_title_word_count, label="Not Approved Projects", bw=0.6)
plt.legend()
```

```
plt.legend()
plt.show()
```



We plotted the PDF curve for numerical data of approved projects and rejected projects under project\_title category

the PDF for both is almost similar both approved and rejected projects Numerical data under project\_title category

The PDF will be calculated based on the distribution of data which is Gaussian distribution for both approved and rejected projects numerical data under project\_title category

Approved and rejected projects numerical data do follow gaussian distribution as the PDF for both is Gaussian curve

we can say whether the project will be accepted or rejected based on no.of.words present in the project title as PDF is almost same for both which gives the probability whether the project is accepted or rejected.

So we do need other few features To predict whether the project will be accepted or rejected

## 1.2.7 Univariate Analysis: Text features (Project Essay's)

In [0]:

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

In [0]:

```
approved_word_count = project_data[project_data['project_is_approved']==1]['essay'].str.split().app
ly(len)
approved_word_count = approved_word_count.values

rejected_word_count = project_data[project_data['project_is_approved']==0]['essay'].str.split().app
ly(len)
rejected_word_count = rejected_word_count.values
```

In [36]:

```
word_count_essays = project_data['essay'].str.split().apply(len)
word_count_essays = word_count_essays.values
print(word_count_essays)
```

```
[272 221 361 ... 181 254 263]
```

In [37]:

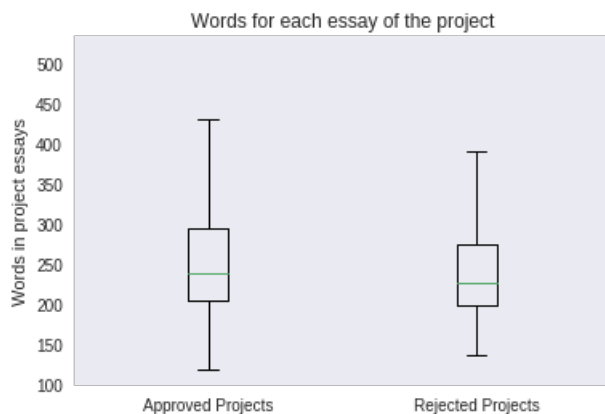
```
word_count_resource_summary = project_data['project_resource_summary'].str.split().apply(len)
word_count_resource_summary = word_count_resource_summary.values
print(word_count_resource_summary)
```

```
[13 11 19 ... 36 15 27]
```

In [38]:

```
# https://glowingpython.blogspot.com/2012/09/boxplot-with-matplotlib.html
```

```
plt.boxplot([approved_word_count, rejected_word_count])
plt.title('Words for each essay of the project')
plt.xticks([1,2], ('Approved Projects', 'Rejected Projects'))
plt.ylabel('Words in project essays')
plt.grid()
plt.show()
```



We calculated boxplots for Approved projects and rejected projects on no.of.words present in essay

There are many outliers for both numerical data

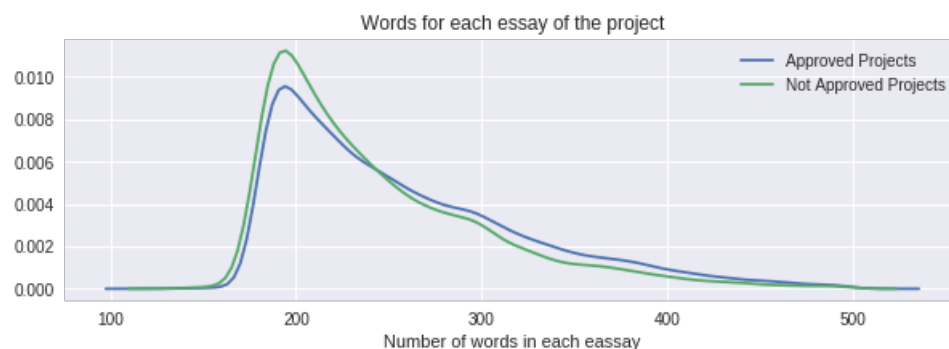
The numerical data is whether the project is approved and rejected based on no.of.words present in the essay.

there many outliers for approved and rejected projects data on no.of.words present in essay

The IQR and median is almost similar for both approved and rejected data under no.of.words present in essay.

In [39]:

```
plt.figure(figsize=(10,3))
sns.distplot(approved_word_count, hist=False, label="Approved Projects")
sns.distplot(rejected_word_count, hist=False, label="Not Approved Projects")
plt.title('Words for each essay of the project')
plt.xlabel('Number of words in each eassay')
plt.legend()
plt.show()
```



We plotted the PDF curve for numerical data of approved projects and rejected projects under no.of.words present in the essay category.

the PDF for both is almost similar both approved and rejected projects Numerical data under no.of.words present in the essay category

The PDF will be calculated based on the distribution of data which is Gaussian distribution for both approved and rejected projects numerical data under no.of.words present in the essay category

Approved and rejected projects numerical data under no.of.words present in the essay do follow gaussian distribution as the PDF for both is Gaussian curve

we cann say whether the project will be accepted or rejected based on no.of.words present in the essay as PDF is almost same for both which gives the probability whther the project is accepted or rejected.



So we do need other few features To predict whether the project will be accepted or rejected

## 1.2.8 Univariate Analysis: Cost per project

In [40]:

```
# we get the cost of the project using resource.csv file
resource_data.head(2)
```

Out[40]:

	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

In [41]:

```
# https://stackoverflow.com/questions/22407798/how-to-reset-a-dataframes-indexes-for-all-groups-in-one-step
price_data = resource_data.groupby('id').agg({'price':'sum', 'quantity':'sum'}).reset_index()
price_data.head(2)
```

Out[41]:

	id	price	quantity
0	p000001	459.56	7
1	p000002	515.89	21

In [0]:

```
# join two dataframes in python:
project_data = pd.merge(project_data, price_data, on='id', how='left')
```

In [0]:

```
approved_price = project_data[project_data['project_is_approved']==1]['price'].values
rejected_price = project_data[project_data['project_is_approved']==0]['price'].values
```

In [44]:

```
# https://glowingpython.blogspot.com/2012/09/boxplot-with-matplotlib.html
plt.boxplot([approved_price, rejected_price])
plt.title('Box Plots of Cost per approved and not approved Projects')
plt.xticks([1,2], ('Approved Projects', 'Rejected Projects'))
plt.ylabel('Price')
plt.grid()
plt.show()
```



We calculated boxplots for Approved projects and rejected projects based on the price of project

There are many outliers for both numerical data for approved and rejected projects based on the price of project

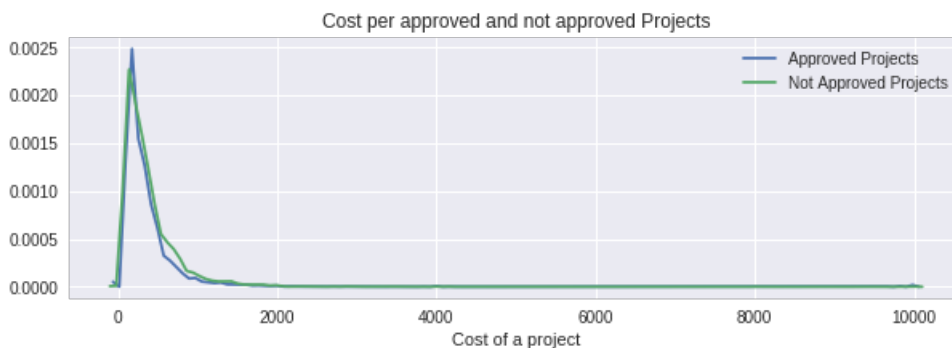
The numerical data is whether the project is approved and rejected based on the price of project

there too many outliers for approved and rejected projects data based on the price of project and the IQR range is too small for the data. As there are too many outliers in the data We could normalize and standard them to scale them under a certain range

The IQR and median is almost similar for both approved and rejected data based on the price of project.

In [45]:

```
plt.figure(figsize=(10,3))
sns.distplot(approved_price, hist=False, label="Approved Projects")
sns.distplot(rejected_price, hist=False, label="Not Approved Projects")
plt.title('Cost per approved and not approved Projects')
plt.xlabel('Cost of a project')
plt.legend()
plt.show()
```



We plotted the PDF curve for numerical data of approved projects and rejected projects based on the price of project and the plot does not follow PDF-Gaussian distribution as the graph increases and decreases linearly at a particular peak

the curve for both is almost similar both approved and rejected projects Numerical data based on the price of project

Approved and rejected projects numerical data based on the price of project do not follow gaussian distribution as the curve is linear.

we can say whether the project will be accepted or rejected based on the price of project as PDF which gives the probability whether the project is accepted or rejected but here the curve is linear, it's difficult to calculate whether the project will be accepted or rejected.

So we do need other few features To predict whether the project will be accepted or rejected

In [46]:

```
# http://zetcode.com/python/prettytable/
from prettytable import PrettyTable

#If you get a ModuleNotFoundError error , install prettytable using: pip3 install prettytable

x = PrettyTable()
x.field_names = ["Percentile", "Approved Projects", "Not Approved Projects"]

for i in range(0,101,5):
    x.add_row([i,np.round(np.percentile(approved_price,i), 3), np.round(np.percentile(rejected_price,i), 3)])
print(x)
```

Percentile	Approved Projects	Not Approved Projects
0	0.66	1.97
5	13.59	41.9
10	33.88	73.67
15	58.0	99.109

	20		77.38		118.56	
	25		99.95		140.892	
	30		116.68		162.23	
	35		137.232		184.014	
	40		157.0		208.632	
	45		178.265		235.106	
	50		198.99		263.145	
	55		223.99		292.61	
	60		255.63		325.144	
	65		285.412		362.39	
	70		321.225		399.99	
	75		366.075		449.945	
	80		411.67		519.282	
	85		479.0		618.276	
	90		593.11		739.356	
	95		801.598		992.486	
	100		9999.0		9999.0	
+-----+-----+-----+-----+-----+						

Here we created a table and calculated the percentiles for the approved and rejected data based on the price of project

We can say that the percentiles are larger for rejected projects as the price is high for rejected projects when compared to same percentile of approved projects

## 1.2.9 Univariate Analysis: teacher\_number\_of\_previously\_posted\_projects

In [0]:

```
project_data['teacher_number_of_previously_posted_projects'].head(10)
approved_ppp = project_data[project_data['project_is_approved']==1]
['teacher_number_of_previously_posted_projects'].values
rejected_ppp = project_data[project_data['project_is_approved']==0]
['teacher_number_of_previously_posted_projects'].values
```

In [48]:

```
plt.boxplot([approved_ppp, rejected_ppp])
plt.title('Box Plots of Cost per approved and not approved Projects')
plt.xticks([1,2], ('Approved Projects', 'Rejected Projects'))
plt.ylabel('teacher_number_of_previously_posted_projects')
plt.grid()
plt.show()
```



We calculated boxplots for Approved projects and rejected projects based on No.of.previously posted projects by a teacher

There are many outliers for both numerical data for approved and rejected projects based on No.of.previously posted projects by a teacher

The numerical data is whether the project is approved and rejected based on No.of.previously posted projects by a teacher

there too many outliers for approved and rejected projects data based on No.of.previously posted projects by a teacher and the IQR rane is too small for the data. As there are to many outliers in the data We could normalize and standard them to scale them under a certain range

The IQR and median is almost similar for both approved and rejected data No.of.previously posted projects by a teacher

In [49]:

```
x = PrettyTable()
x.field_names = ["Percentile", "Approved Projects", "Not Approved Projects"]

for i in range(0,101,5):
    x.add_row([i,np.round(np.percentile(approved_ppp,i), 3), np.round(np.percentile(rejected_ppp,i)
, 3)])
print(x)
```

Percentile	Approved Projects	Not Approved Projects
0	0.0	0.0
5	0.0	0.0
10	0.0	0.0
15	0.0	0.0
20	0.0	0.0
25	0.0	0.0
30	1.0	0.0
35	1.0	1.0
40	1.0	1.0
45	2.0	1.0
50	2.0	2.0
55	3.0	2.0
60	4.0	3.0
65	5.0	3.0
70	7.0	4.0
75	9.0	6.0
80	13.0	8.0
85	19.0	11.0
90	30.0	17.0
95	57.0	31.0
100	451.0	345.0

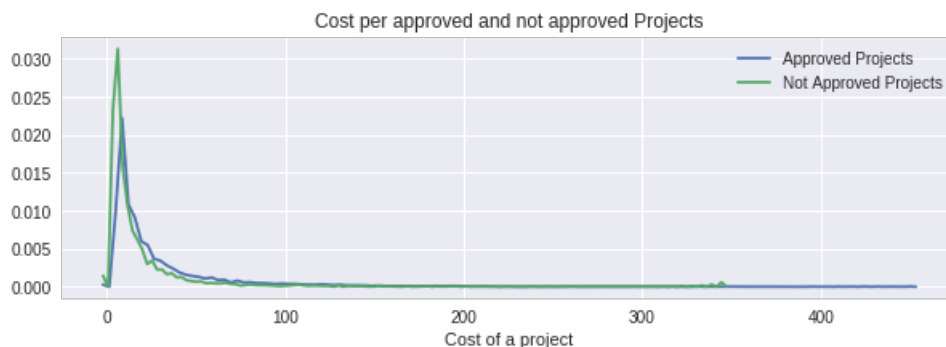
Here we created a table and calculated the percentiles for the approved and rejected data No.of.previously posted projects by a teacher

The No.of.previously posted projects by a teacher are very less for a smaller range and more for a larger range

This means large no.of. teachers posted large no.of. projects previously and few no.of.teachers posted few no.of.projects previously.

In [50]:

```
plt.figure(figsize=(10,3))
sns.distplot(approved_ppp, hist=False, label="Approved Projects")
sns.distplot(rejected_ppp, hist=False, label="Not Approved Projects")
plt.title('Cost per approved and not approved Projects')
plt.xlabel('Cost of a project')
plt.legend()
plt.show()
```



We plotted the PDF curve for numerical data of approved projects and rejected projects No.of.previously posted projects by a teacher and the plot do not follow PDF-Gaussian distribution as the graph increases and decreases linearly at a particular peak

the curve for both is almost similar both approved and rejected projects Numerical data No.of.previously posted projects by a teacher

Approved and rejected projects numerical data No.of.previously posted projects by a teacher do not follow gaussian distribution as the curve is linear .

we can say whether the project will be accepted or rejected No.of.previously posted projects by a teacher as PDF which gives the probability whther the project is accepted or rejected but here the curve is linear, its difficult to calculate whether the project will be calculated or rejected.

So we do need other few features To predict whether the project will be accepted or rejected

### 1.2.10 Univariate Analysis: project\_resource\_summary

Please do this on your own based on the data analysis that was done in the above cells

Check if the presence of the numerical digits in the project\_resource\_summary effects the acceptance of the project or not. If you observe that presence of the numerical digits is helpful in the classification, please include it for further process or you can ignore it.

In [0]:

```
k=project_data['project_resource_summary'].shape
n = k[0]
print(n)
b=project_data['project_resource_summary'][45]
print(b)
project_data['No.of.digits'] = 0
project_data.head(5)
```

109248

My students need 5 Chromebooks to access their differentiated literacy instruction through the Lexia Reading Core5 program. This will individually help fill in their various learning gaps.

Out[0]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime	pro
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	2016-12-05 13:43:57	Gra
1	140945	p258326	897464ce9ddc600bcd1151f324dd63a	Mr.	FL	2016-10-25 09:22:10	Gra
2	21895	p182444	3465aaf82da834c0582ebd0ef8040ca0	Ms.	AZ	2016-08-31 12:03:56	Gra
3	45	p246581	f3cb9bffbb169bef1a77b243e620b60	Mrs.	KY	2016-10-06 21:16:17	Gra

4	1722407 Unnamed: 0	p104768 id	be1f7507a41f8479dc06f047086a39ec teacher_id	Mrs. teacher_prefix	TX school_state	2016-07-11 01:10:09 project_submitted_datetime	Gra pro

5 rows × 21 columns

In [0]:

```
for j in tqdm(range(70000)):
    sample = project_data['project_resource_summary'][j] #sample string
    letters = 0
    numeric = 0

    for i in sample:
        if i.isdigit():
            numeric +=1
        elif i.isalpha():
            letters +=1
        else:
            pass
    project_data['No.of.digits'][j] = numeric
#count no of digits in a string https://stackoverflow.com/questions/24878174/how-to-count-digits-letters-spaces-for-a-string-in-python
```

CALCULATED ANALYSIS on NO.OF.DIGITS in THE PROJECT\_RESOURCE\_SUMMARY TEXT on 10000 points only as its taking HUGE TIME on my LAPTOP

In [0]:

```
project_data[:69999].to_pickle('gdrive/My Drive/naive_bayes_no.of.digits_original.pkl')
```

In [0]:

```
project_data = pd.read_pickle('gdrive/My Drive/naive_bayes_no.of.digits_original.pkl')
```

In [0]:

```
digits_project_resource_summary = project_data['No.of.digits']
```

In [0]:

```
rejected_np[:100]
```

As we can see the no.of.digits in the text are few and the text do contain any few digits

Approved\_np and rejected\_np is very sparse as we can see

In [0]:

```
plt.boxplot([approved_np[:49999], rejected_np[:49999]])
plt.title('Box Plots of No.of.digits in project_resource_summary per approved and not approved Projects')
plt.xticks([1,2], ('Approved Projects', 'Rejected Projects'))
plt.ylabel('No.of.digits in project_resource_summary')
plt.grid()
plt.show()
```

FOR 10000 DATAPOINTS ONLY

The no.of.digits in No.of.digits in project\_resource\_summary text are few and very less, So we can IQR is almost zero and the text which has values are treated as outliers mostly for both approved and rejected projects

In [0]:

```
plt.figure(figsize=(10,3))
sns.distplot(approved_np, hist=False, label="Approved Projects")
```

```
sns.distplot(rejected_np, hist=False, label="Not Approved Projects")
plt.title('No.of.digits in project_resource_summary per approved and not approved Projects')
plt.xlabel('No.of.digits in project_resource_summary')
plt.legend()
plt.show()
```

This do not follow any gaussian distribution and the curve for approved and rejected is non-linear and non-symmetric and do have any local maximum and local minimum

we need to look for other features in the Project\_data and check whether other features data follow any distributions

## 1.3 Text preprocessing

### 1.3.1 Essay Text

In [51]:

```
project_data.shape
```

Out[51]:

```
(109248, 20)
```

In [52]:

```
# printing some random essays.
print(project_data['essay'].values[0])
print("="*50)
print(project_data['essay'].values[150])
print("="*50)
print(project_data['essay'].values[1000])
print("="*50)
print(project_data['essay'].values[20000])
print("="*50)
```

My students are English learners that are working on English as their second or third languages. We are a melting pot of refugees, immigrants, and native-born Americans bringing the gift of language to our school. \r\n\r\n We have over 24 languages represented in our English Learner program with students at every level of mastery. We also have over 40 countries represented with the families within our school. Each student brings a wealth of knowledge and experiences to us that open our eyes to new cultures, beliefs, and respect.\"The limits of your language are the limits of your world.\"-Ludwig Wittgenstein Our English learner's have a strong support system at home that begs for more resources. Many times our parents are learning to read and speak English alongside of their children. Sometimes this creates barriers for parents to be able to help their child learn phonetics, letter recognition, and other reading skills.\r\n\r\nBy providing these dvd's and players, students are able to continue their mastery of the English language even if no one at home is able to assist. All families with students within the Level 1 proficiency status, will be offered to be a part of this program. These educational videos will be specially chosen by the English Learner Teacher and will be sent home regularly to watch. The videos are to help the child develop early reading skills.\r\n\r\nParents that do not have access to a dvd player will have the opportunity to check out a dvd player to use for the year. The plan is to use these videos and educational dvd's for the years to come for other EL students.\r\nnnnnnn

=====

The 51 fifth grade students that will cycle through my classroom this year all love learning, at least most of the time. At our school, 97.3% of the students receive free or reduced price lunch. Of the 560 students, 97.3% are minority students. \r\nThe school has a vibrant community that loves to get together and celebrate. Around Halloween there is a whole school parade to show off the beautiful costumes that students wear. On Cinco de Mayo we put on a big festival with crafts made by the students, dances, and games. At the end of the year the school hosts a carnival to celebrate the hard work put in during the school year, with a dunk tank being the most popular activity. My students will use these five brightly colored Hokki stools in place of regular, stationary, 4-legged chairs. As I will only have a total of ten in the classroom and not enough for each student to have an individual one, they will be used in a variety of ways. During independent reading time they will be used as special chairs students will each use on occasion. I will utilize them in place of chairs at my small group tables during math and reading times. The rest of the day they will be used by the students who need the highest amount of movement in their life in order to stay focused on school.\r\n\r\nWhenever asked what the classroom is missing, my students always say more Hokki Stools. They can't get their fill of the 5 stools we already have. When the students are sitting in group with me on the Hokki Stools, they are always moving, but at the same time doing their work. Anytime the students get to pick where they can sit, the Hokki Stools are the first to be taken. There are always students who head over to the kidney table to get one of the stools who are

disappointed as there are not enough of them. \r\n\r\nWe ask a lot of students to sit for 7 hours a day. The Hokki stools will be a compromise that allow my students to do desk work and move at the same time. These stools will help students to meet their 60 minutes a day of movement by allowing them to activate their core muscles for balance while they sit. For many of my students, these chairs will take away the barrier that exists in schools for a child who can't sit still.nannan

=====  
How do you remember your days of school? Was it in a sterile environment with plain walls, rows of desks, and a teacher in front of the room? A typical day in our room is nothing like that. I work hard to create a warm inviting themed room for my students look forward to coming to each day.\r\n\r\nMy class is made up of 28 wonderfully unique boys and girls of mixed races in Arkansas.\r\nThey attend a Title I school, which means there is a high enough percentage of free and reduced-price lunch to qualify. Our school is an "open classroom" concept, which is very unique as there are no walls separating the classrooms. These 9 and 10 year-old students are very eager learners; they are like sponges, absorbing all the information and experiences and keep on wanting more. With these resources such as the comfy red throw pillows and the whimsical nautical hanging decor and the blue fish nets, I will be able to help create the mood in our classroom setting to be one of a themed nautical environment. Creating a classroom environment is very important in the success in each and every child's education. The nautical photo props will be used with each child as they step foot into our classroom for the first time on Meet the Teacher evening. I'll take pictures of each child with them, have them developed, and then hung in our classroom ready for their first day of 4th grade. This kind gesture will set the tone before even the first day of school! The nautical thank you cards will be used throughout the year by the students as they create thank you cards to their team groups.\r\n\r\nYour generous donations will help me to help make our classroom a fun, inviting, learning environment from day one.\r\n\r\nIt costs a lot of money out of my own pocket on resources to get our classroom ready. Please consider helping with this project to make our new school year a very successful one. Thank you!nannan

=====  
My kindergarten students have varied disabilities ranging from speech and language delays, cognitive delays, gross/fine motor delays, to autism. They are eager beavers and always strive to work their hardest working past their limitations. \r\n\r\nThe materials we have are the ones I seek out for my students. I teach in a Title I school where most of the students receive free or reduced price lunch. Despite their disabilities and limitations, my students love coming to school and come eager to learn and explore. Have you ever felt like you had ants in your pants and you needed to groove and move as you were in a meeting? This is how my kids feel all the time. They want to be able to move as they learn or so they say. Wobble chairs are the answer and I love them because they develop their core, which enhances gross motor and in turn fine motor skills. \r\nThey also want to learn through games, my kids don't want to sit and do worksheets. They want to learn to count by jumping and playing. Physical engagement is the key to our success. The number toss and color and shape mats can make that happen. My students will forget they are doing work and just have the fun a 6 year old deserves.nannan

In [0]:

```
# https://stackoverflow.com/a/47091490/4084039
import re

def decontracted(phrase):
    # specific
    phrase = re.sub(r"won't", "will not", phrase)
    phrase = re.sub(r"can't", "can not", phrase)

    # general
    phrase = re.sub(r"n't", " not", phrase)
    phrase = re.sub(r"\ 're", " are", phrase)
    phrase = re.sub(r"\ 's", " is", phrase)
    phrase = re.sub(r"\ 'd", " would", phrase)
    phrase = re.sub(r"\ 'll", " will", phrase)
    phrase = re.sub(r"\ 't", " not", phrase)
    phrase = re.sub(r"\ 've", " have", phrase)
    phrase = re.sub(r"\ 'm", " am", phrase)
    return phrase
```

In [54]:

```
sent = decontracted(project_data['essay'].values[20000])
print(sent)
print("="*50)
```

My kindergarten students have varied disabilities ranging from speech and language delays, cognitive delays, gross/fine motor delays, to autism. They are eager beavers and always strive to work their hardest working past their limitations. \r\n\r\nThe materials we have are the ones I seek out for my students. I teach in a Title I school where most of the students receive free or reduced price



ice lunch. Despite their disabilities and limitations, my students love coming to school and come eager to learn and explore. Have you ever felt like you had ants in your pants and you needed to groove and move as you were in a meeting? This is how my kids feel all the time. They want to be able to move as they learn or so they say. Wobble chairs are the answer and I love them because they develop their core, which enhances gross motor and in turn fine motor skills. \r\nThey also want to learn through games, my kids do not want to sit and do worksheets. They want to learn to count by jumping and playing. Physical engagement is the key to our success. The number toss and color and shape mats can make that happen. My students will forget they are doing work and just have the fun a 6 year old deserves. nannan

=====

In [55]:

```
# \r \n \t remove from string python: http://texthandler.com/info/remove-line-breaks-python/
sent = sent.replace('\\r', ' ')
sent = sent.replace('\\n', ' ')
sent = sent.replace('\\t', ' ')
print(sent)
```

My kindergarten students have varied disabilities ranging from speech and language delays, cognitive delays, gross/fine motor delays, to autism. They are eager learners and always strive to work their hardest working past their limitations. The materials we have are the ones I seek out for my students. I teach in a Title I school where most of the students receive free or reduced price lunch. Despite their disabilities and limitations, my students love coming to school and come eager to learn and explore. Have you ever felt like you had ants in your pants and you needed to groove and move as you were in a meeting? This is how my kids feel all the time. They want to be able to move as they learn or so they say. Wobble chairs are the answer and I love them because they develop their core, which enhances gross motor and in turn fine motor skills. They also want to learn through games, my kids do not want to sit and do worksheets. They want to learn to count by jumping and playing. Physical engagement is the key to our success. The number toss and color and shape mats can make that happen. My students will forget they are doing work and just have the fun a 6 year old deserves. nannan

In [56]:

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

My kindergarten students have varied disabilities ranging from speech and language delays cognitive delays gross fine motor delays to autism They are eager learners and always strive to work their hardest working past their limitations The materials we have are the ones I seek out for my students I teach in a Title I school where most of the students receive free or reduced price lunch Despite their disabilities and limitations my students love coming to school and come eager to learn and explore Have you ever felt like you had ants in your pants and you needed to groove and move as you were in a meeting This is how my kids feel all the time They want to be able to move as they learn or so they say Wobble chairs are the answer and I love them because they develop their core which enhances gross motor and in turn fine motor skills They also want to learn through games my kids do not want to sit and do worksheets They want to learn to count by jumping and playing Physical engagement is the key to our success The number toss and color and shape mats can make that happen My students will forget they are doing work and just have the fun a 6 year old deserves nannan

In [0]:

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

```

        'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', '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', "dc
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", "nan"]

```

In [58]:

```

# Combining all the above statemennts
from tqdm import tqdm
preprocessed_essays = []
# tqdm is for printing the status bar
for sentence in tqdm(project_data['essay'].values):
    sent = decontracted(sentence)
    sent = sent.replace('\r', ' ')
    sent = sent.replace('\n', ' ')
    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%|██████████| 109248/109248 [01:10<00:00, 1542.50it/s]

In [59]:

```

# after preprocessing
preprocessed_essays[20000]

```

Out[59]:

'my kindergarten students varied disabilities ranging speech language delays cognitive delays gros  
s fine motor delays autism they eager beavers always strive work hardest working past limitations  
the materials ones i seek students i teach title i school students receive free reduced price lunc  
h despite disabilities limitations students love coming school come eager learn explore have ever  
felt like ants pants needed groove move meeting this kids feel time the want able move learn say w  
obble chairs answer i love develop core enhances gross motor turn fine motor skills they also want  
learn games kids not want sit worksheets they want learn count jumping playing physical engagement  
key success the number toss color shape mats make happen my students forget work fun 6 year old de  
serves nannan'

## Essays\_1-Text

In [0]:

```

# Combining all the above statemennts
from tqdm import tqdm
preprocessed_essays_1 = []
# tqdm is for printing the status bar
for sentence in tqdm(project_data['project_essay_1'].values):
    sent = decontracted(sentence)
    sent = sent.replace('\r', ' ')
    sent = sent.replace('\n', ' ')
    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_1.append(sent.lower().strip())

```

100%|██████████| 69999/69999 [00:18<00:00, 3783.79it/s]

## Essays\_2\_Text

In [0]:

```
# Combining all the above statemennts
from tqdm import tqdm
preprocessed_essays_2 = []
# tqdm is for printing the status bar
for sentence in tqdm(project_data['project_essay_2'].values):
    sent = decontracted(sentence)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e not in stopwords)
    preprocessed_essays_2.append(sent.lower().strip())
```

100%|██████████| 69999/69999 [00:23<00:00, 3036.19it/s]

In [0]:

```
# We are considering only the words which appeared in at least 10 documents(rows or projects).
vectorizer = CountVectorizer(min_df=10,max_features=10000)
train_bow_essay_2 = vectorizer.fit_transform(preprocessed_essays_2)
```

In [0]:

```
vectorizer_1 = CountVectorizer(min_df=10,max_features=10000)
train_bow_essay_1 = vectorizer_1.fit_transform(preprocessed_essays_1)
```

### Similarity Between Two essays

In [0]:

```
from sklearn.metrics.pairwise import cosine_similarity
similarity = []
for i in tqdm(range(70000)):

    similarity.append(cosine_similarity(train_bow_essay_1[i],train_bow_essay_2[i]).item(0))
```

In [0]:

```
similarity = pd.Series(similarity)
```

## 1.3.2 Project title Text-Cleaning

In [0]:

```
# similarly you can preprocess the titles also
```

In [60]:

```
project_data['project_title'].values[50]
```

Out[60]:

'Be Active! Be Energized!'

In [61]:

```
project_data['title'] = project_data['project_title'].map(str)
project_data['title'].values[0]
```

Out[61]:

'Educational Support for English Learners at Home'

In [62]:

```
sent = decontracted(project_data['title'].values[20000])
print(sent)
print("="*50)
```

We Need To Move It While We Input It!

=====

We are checking is decontracted applied to 20000th value of project\_title

In [63]:

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

We Need To Move It While We Input It

we are replacing special characters in the project\_title with space

In [64]:

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

We Need To Move It While We Input It

we are replacing numerical charcaters in the text with space

In [65]:

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

100%|██████████| 109248/109248 [00:03<00:00, 31074.24it/s]

In [66]:

```
preprocessed_titles[20000]
```

Out[66]:

'we need to move it while we input it'

## Resource\_summary\_Text

In [67]:

```
project_data['resources_summary'] = project_data['project_resource_summary'].map(str)
project_data['resources_summary'].values[0]
```

Out[67]:

'My students need opportunities to practice beginning reading skills in English at home.'

In [0]:

```
# Combining all the above statements
from tqdm import tqdm
preprocessed_resource_summary = []
# tqdm is for printing the status bar
for sentence in project_data['project_resource_summary'].values:
    sent = decontracted(sentence)
    sent = sent.replace('\r', ' ')
    sent = sent.replace('\n', ' ')
    sent = sent.replace('\n', ' ')
    sent = sent.replace(',', ' ')
    sent = sent.replace('!', ' ')
    sent = sent.replace('*', ' ')
    sent = sent.replace('.', ' ')
    sent = sent.replace(':', ' ')
    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_resource_summary.append(sent.lower().strip())
```

In [69]:

```
!pip install autocorrect
```

Collecting autocorrect

Downloading

<https://files.pythonhosted.org/packages/ec/b6/6c74ff19249dc6d7285541cd59f5a3edbbd0f7209362a63e314fc636/autocorrect-0.3.0.tar.gz> (3.6MB)

100% |██| 3.6MB 7.6MB/s

Building wheels for collected packages: autocorrect

Building wheel for autocorrect (setup.py) ... done

Stored in directory:

/root/.cache/pip/wheels/bf/b8/ae/704d5643f1d0637c5b87d9feccf2ee923c492b703bb0bfbb19

Successfully built autocorrect

Installing collected packages: autocorrect

Successfully installed autocorrect-0.3.0

## SpellingMistakes Count

### Resource\_summary

In [0]:

```
from autocorrect import spell
count_spellingmistake_resource = []
for i in tqdm(range(70000)):

    my_words = []

    my_words = preprocessed_resource_summary[i].split()
    my_words = list(set(my_words))
    count = 0
    for j in range(len(my_words)):

        if(my_words[j] != spell(my_words[j])):

            count += 1
```

```
        else:

            pass

count_spellingmistake_resource.append(count)
```

## Titles

In [0]:

```
from autocorrect import spell
count_spellingmistake_titles = []
for i in tqdm(range(70000)):

    my_words = []

    my_words = preprocessed_titles[i].split()
    my_words = list(set(my_words))
    count = 0
    for j in range(len(my_words)):

        if(my_words[j] != spell(my_words[j])):

            count += 1
        else:
            pass
    count_spellingmistake_titles.append(count)
```

## Essays

In [0]:

```
from autocorrect import spell
count_spellingmistake_essays = []
for i in tqdm(range(70000)):

    my_words = []

    my_words = preprocessed_essays[i].split()
    my_words = list(set(my_words))
    count = 0
    for j in range(len(my_words)):

        if(my_words[j] != spell(my_words[j])):

            count += 1
        else:
            pass
    count_spellingmistake_essays.append(count)
```

100%|██████████| 70000/70000 [1:35:25<00:00, 12.23it/s]

## Sentimental Analysis of Essays

In [0]:

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

```

for_sentiment = 'a person is a person no matter how small dr seuss i teach the smallest students w
ith the biggest enthusiasm \
for learning my students learn in many different ways using all of our senses and multiple intelli
gences i use a wide range\
of techniques to help all my students succeed students in my class come from a variety of differen
t backgrounds which makes\
for wonderful sharing of experiences and cultures including native americans our school is a carin
g community of successful \
learners which can be seen through collaborative student project based learning in and out of the
classroom kindergarteners \
in my class love to work with hands on materials and have many different opportunities to practice
a skill before it is\
mastered having the social skills to work cooperatively with friends is a crucial aspect of the ki
ndergarten curriculum\
montana is the perfect place to learn about agriculture and nutrition my students love to role pla
y in our pretend kitchen\
in the early childhood classroom i have had several kids ask me can we try cooking with real food
i will take their idea \
and create common core cooking lessons where we learn important math and writing concepts while co
oking delicious healthy \
food for snack time my students will have a grounded appreciation for the work that went into maki
ng the food and knowledge \
of where the ingredients came from as well as how it is healthy for their bodies this project woul
d expand our learning of \
nutrition and agricultural cooking recipes by having us peel our own apples to make homemade apple
sauce make our own bread \
and mix up healthy plants from our classroom garden in the spring we will also create our own cook
books to be printed and \
shared with families students will gain math and literature skills as well as a life long enjoymen
t for healthy cooking \
nannan'
ss = sid.polarity_scores(for_sentiment)

```

```

for k in ss:
    print('{0}: {1}', .format(k, ss[k]), end='')

# we can use these 4 things as features/attributes (neg, neu, pos, compound)
# neg: 0.0, neu: 0.753, pos: 0.247, compound: 0.93

```

[nltk\_data] Downloading package vader\_lexicon to /root/nltk\_data...

neg: 0.01, neu: 0.745, pos: 0.245, compound: 0.9975,

In [0]:

```

ss = sid.polarity_scores(preprocessed_essays[5])
for k in ss:
    print(k,ss[k])

```

```

neg 0.111
neu 0.647
pos 0.242
compound 0.9776

```

In [0]:

```

negative_polarity = []
neutrality_polarity = []
positive_polarity = []
compound_polarity = []

for i in range(70000):

    ss = sid.polarity_scores(preprocessed_essays[i])

    negative_polarity.append(ss['neg'])
    neutrality_polarity.append(ss['neu'])
    positive_polarity.append(ss['pos'])
    compound_polarity.append(ss['compound'])

```

In [0]:

```
negative_polarity = pd.Series(negative_polarity)
neutrality_polarity = pd.Series(neutrality_polarity)
positive_polarity = pd.Series(positive_polarity)
compound_polarity = pd.Series(compound_polarity)
```

In [0]:

```
count_spellingmistake_essays = pd.Series(count_spellingmistake_essays)
count_spellingmistake_titles = pd.Series(count_spellingmistake_titles)
count_spellingmistake_resource = pd.Series(count_spellingmistake_resource)
```

## 1. 4 Preparing data for models

In [0]:

```
project_data.columns
```

Out[0]:

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
      'project_submitted_datetime', 'project_grade_category', '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',
      'clean_categories', 'clean_subcategories', 'essay', 'price', 'quantity',
      'title', 'resources_summary'],
      dtype='object')
```

### 1.4.1 Vectorizing Categorical data

- <https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/handling-categorical-and-numerical-features/>

#### School\_state- One hot encoding

In [70]:

```
## School_state- One hot Encoding
# we use count vectorizer to convert the values into one hot encoded features
#https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html
vectorizer = CountVectorizer(lowercase=False, binary=True)
vectorizer.fit(project_data['school_state'].values)
print(vectorizer.get_feature_names())
X= vectorizer.get_feature_names()

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

```
['AK', 'AL', 'AR', 'AZ', 'CA', 'CO', 'CT', 'DC', 'DE', 'FL', 'GA', 'HI', 'IA', 'ID', 'IL', 'IN', 'KS', '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 (109248, 51)
```

#### Clean\_Categories-One-Hot-Encoding

In [71]:

```
# we use count vectorizer to convert the values into one hot encoded features
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(vocabulary=list(sorted_cat_dict.keys()), lowercase=False, binary=True)
vectorizer.fit(project_data['clean_categories'].values)
print(vectorizer.get_feature_names())
```



```
X = X + vectorizer.get_feature_names()
```

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

```
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'SpecialNeeds',
'Health_Sports', 'Math_Science', 'Literacy_Language']
Shape of matrix after one hot encodig (109248, 9)
['AK', 'AL', 'AR', 'AZ', 'CA', 'CO', 'CT', 'DC', 'DE', 'FL', 'GA', 'HI', 'IA', 'ID', 'IL', 'IN', 'KS',
'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', 'Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning',
'SpecialNeeds', 'Health_Sports', 'Math_Science', 'Literacy_Language']
```

## Clean\_SubCategories

In [72]:

```
# we use count vectorizer to convert the values into one hot encoded features
vectorizer = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()), lowercase=False, binary=
True)
vectorizer.fit(project_data['clean_subcategories'].values)
print(vectorizer.get_feature_names())
X = X + vectorizer.get_feature_names()

sub_categories_one_hot = vectorizer.transform(project_data['clean_subcategories'].values)
print("Shape of matrix after one hot encodig ",sub_categories_one_hot.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 (109248, 30)
```

## Project\_Grade\_Category

In [73]:

```
import pandas as pd
s = pd.Series(project_data['project_grade_category'])
project_grade_one_hot = pd.get_dummies(s)
print(project_grade_one_hot.shape,project_grade_one_hot.columns)
X = X + list(project_grade_one_hot.columns)
```

```
(109248, 4) Index(['Grades 3-5', 'Grades 6-8', 'Grades 9-12', 'Grades PreK-2'], dtype='object')
```

## Teahcer\_prefix-One Hot Encoding

In [74]:

```
project_data[project_data['teacher_prefix'].isnull()]
```

Out [74]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime
7820	17809	p180947	834f75f1b5e24bd10abe9c3dbf7ba12f	NaN	CA	2016-11-04 00:15:45

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime
30368	22174	p002730	339bd5a9e445d68a74d65b99cd325397	NaN	SC	2016-05-09 09:38:40
57654	158692	p197901	e4be6aaaa887d4202df2b647bfc82bb	NaN	PA	2016-06-03 10:15:05

3 rows × 22 columns



We contain nan values in the teacher\_prefix column

In [75]:

```
#replacing nan values in pandas https://stackoverflow.com/questions/13295735/how-can-i-replace-all-the-nan-values-with-zeros-in-a-column-of-a-pandas-datafra
project_data['teacher_prefix'].value_counts()
project_data['teacher_prefix'] = project_data['teacher_prefix'].fillna('Mrs.')
project_data['teacher_prefix'].isnull().any()
```

Out[75]:

False

replaced nan values in teacher\_prefix with "Mrs." as Mrs. is majority vote

In [76]:

```
# we use count vectorizer to convert the values into one hot encoded features
#https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html
vectorizer = CountVectorizer(lowercase=False, binary=True)
vectorizer.fit(project_data['teacher_prefix'].values)
print(vectorizer.get_feature_names())
X = X + vectorizer.get_feature_names()
```

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

```
['Dr', 'Mr', 'Mrs', 'Ms', 'Teacher']
Shape of matrix after one hot encodig (109248, 5)
```

## Vectorizing Numerical Features

### Prize

In [0]:

```
# check this one: https://www.youtube.com/watch?v=0H0qOcln3Z4&t=530s
# standardization sklearn: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
from sklearn.preprocessing import StandardScaler

# price_standardized = standardScalar.fit(project_data['price'].values)
# this will rise the error
# ValueError: Expected 2D array, got 1D array instead: array=[725.05 213.03 329. ... 399. 287.73 5.5 ].
# Reshape your data either using array.reshape(-1, 1)

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

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

Mean : 298.1193425966608, Standard deviation : 367.49634838483496

In [0]:

```
X = X + ['price_standardized']
```

## teacher\_number\_of\_previously\_posted\_projects

In [0]:

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

# price_standardized = standardScaler.fit(project_data['price'].values)
# this will rise the error
# ValueError: Expected 2D array, got 1D array instead: array=[725.05 213.03 329. ... 399. 287.73 5.5 ].
# Reshape your data either using array.reshape(-1, 1)

teacher_number_of_previously_posted_projects_scalar = StandardScaler()
teacher_number_of_previously_posted_projects_scalar.fit(project_data['teacher_number_of_previously_
osted_projects'].values.reshape(-1,1)) # finding the mean and standard deviation of this data
print(f"Mean : {teacher_number_of_previously_posted_projects_scalar.mean_[0]}, Standard deviation
: {np.sqrt(teacher_number_of_previously_posted_projects_scalar.var_[0])}")

# Now standardize the data with above mean and variance.
teacher_number_of_previously_posted_projects_standardized =
teacher_number_of_previously_posted_projects_scalar.transform(project_data['teacher_number_of_previ
ously_posted_projects'].values.reshape(-1, 1))
```

Mean : 11.153165275336848, Standard deviation : 27.77702641477403

In [0]:

```
X = X + ['previously_posted_projects']
```

## Quantity

In [0]:

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

# price_standardized = standardScaler.fit(project_data['price'].values)
# this will rise the error
# ValueError: Expected 2D array, got 1D array instead: array=[725.05 213.03 329. ... 399. 287.73 5.5 ].
# Reshape your data either using array.reshape(-1, 1)

quantity_scalar = StandardScaler()
quantity_scalar.fit(resource_data['quantity'][:109248].values.reshape(-1,1)) # finding the mean and
standard deviation of this data
print(f"Mean : {quantity_scalar.mean_[0]}, Standard deviation :
{np.sqrt(quantity_scalar.var_[0])}")

# Now standardize the data with above mean and variance.
quantity_standardised = quantity_scalar.transform(resource_data['quantity']
[:109248].values.reshape(-1, 1))
```

Mean : 3.159179115407147, Standard deviation : 9.225773323106056

In [0]:

```
X = X + ['quantity_standardised']
```

## Digits in Resource\_Summary

In [0]:

```
# check this one: https://www.youtube.com/watch?v=0H0qOcln3Z4&t=530s
# standardization sklearn: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
from sklearn.preprocessing import StandardScaler

# price_standardized = standardScaler.fit(project_data['price'].values)
# this will rise the error
# ValueError: Expected 2D array, got 1D array instead: array=[725.05 213.03 329. ... 399. 287. 73 5.5 ].
# Reshape your data either using array.reshape(-1, 1)

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

# Now standardize the data with above maen and variance.
digits_standardised = digits_scalar.transform(digits_project_resource_summary.values.reshape(-1, 1))
)
```

Mean : 0.2779039700567151, Standard deviation : 0.8994739980674412

In [0]:

```
X = X + ['digits_standardised']
```

## Words in Essays

In [0]:

```
from sklearn.preprocessing import StandardScaler

essays_scalar = StandardScaler()
essays_scalar.fit(word_count_essays.reshape(-1,1)) # finding the mean and standard deviation of this data
print(f"Mean : {essays_scalar.mean_[0]}, Standard deviation : {np.sqrt(essays_scalar.var_[0])}")

# Now standardize the data with above maen and variance.
essays_words_standardised = essays_scalar.transform(word_count_essays.reshape(-1, 1))
```

Mean : 255.25869581136496, Standard deviation : 65.50050244330986

In [0]:

```
X = X + ['essays_words_standardised']
```

## Resource\_summary-No.of.Words

In [0]:

```
essays_scalar = StandardScaler()
essays_scalar.fit(word_count_resource_summary.reshape(-1,1)) # finding the mean and standard deviation of this data
print(f"Mean : {essays_scalar.mean_[0]}, Standard deviation : {np.sqrt(essays_scalar.var_[0])}")

# Now standardize the data with above maen and variance.
summary_words_standardised = essays_scalar.transform(word_count_resource_summary.reshape(-1, 1))
```

Mean : 20.219271748681898, Standard deviation : 7.778158320755454

In [0]:

```
X = X + ['summary_words_standardised']
```

## Titles-No.of.Words

In [0]:

```
titles_scalar = StandardScaler()
titles_scalar.fit(word_count_titles.reshape(-1,1)) # finding the mean and standard deviation of th
is data
print(f"Mean : {titles_scalar.mean_[0]}, Standard deviation : {np.sqrt(titles_scalar.var_[0])}")

# Now standardize the data with above maen and variance.
titles_words_standardised = titles_scalar.transform(word_count_titles.reshape(-1, 1))
```

Mean : 5.145595342706502, Standard deviation : 2.099338404931425

In [0]:

```
X = X + ['titles_words_standardised']
```

## Sentimental\_score's of Essays

In [0]:

```
neutrality_scalar = StandardScaler()
neutrality_scalar.fit(neutrality_polarity.reshape(-1,1)) # finding the mean and standard deviation
of this data
print(f"Mean : {neutrality_scalar.mean_[0]}, Standard deviation :
{np.sqrt(neutrality_scalar.var_[0])}")

# Now standardize the data with above maen and variance.
neutrality_polarity_standardised = neutrality_scalar.transform(neutrality_polarity.reshape(-1, 1))
```

Mean : 0.6879833, Standard deviation : 0.07240191981251996

In [0]:

```
X = X + ['neutrality_polarity_standardised']
```

In [0]:

```
compound_scalar = StandardScaler()
compound_scalar.fit(compound_polarity.reshape(-1,1)) # finding the mean and standard deviation of
this data
print(f"Mean : {compound_scalar.mean_[0]}, Standard deviation :
{np.sqrt(compound_scalar.var_[0])}")

# Now standardize the data with above maen and variance.
compound_polarity_standardised = compound_scalar.transform(compound_polarity.reshape(-1, 1))
```

Mean : 0.9596761114285715, Standard deviation : 0.1494520866877949

In [0]:

```
X = X + ['compound_polarity_standardised']
```

In [0]:

```
positive_scalar = StandardScaler()
positive_scalar.fit(positive_polarity.reshape(-1,1)) # finding the mean and standard deviation of
```

```
this data
print(f"Mean : {positive_scalar.mean_[0]}, Standard deviation :
{np.sqrt(positive_scalar.var_[0])}")

# Now standardize the data with above mean and variance.
positive_polarity_standardised = positive_scalar.transform(positive_polarity.reshape(-1, 1))
```

Mean : 0.26698345714285715, Standard deviation : 0.07404377034115617

In [0]:

```
X = X + ['positive_polarity_standardised']
```

In [0]:

```
negative_scalar = StandardScaler()
negative_scalar.fit(negative_polarity.reshape(-1,1)) # finding the mean and standard deviation of
this data
print(f"Mean : {negative_scalar.mean_[0]}, Standard deviation :
{np.sqrt(negative_scalar.var_[0])}")

# Now standardize the data with above mean and variance.
negative_polarity_standardised = negative_scalar.transform(negative_polarity.reshape(-1, 1))
```

Mean : 0.045034057142857145, Standard deviation : 0.03379663821146616

In [0]:

```
X = X + ['negative_polarity_standardised']
```

In [0]:

```
### Spelling Mistakes-Count in Essays
```

In [0]:

```
essays_spelling_scalar = StandardScaler()
essays_spelling_scalar.fit(count_spellingmistake_essays.reshape(-1,1)) # finding the mean and
standard deviation of this data
print(f"Mean : {essays_spelling_scalar.mean_[0]}, Standard deviation :
{np.sqrt(essays_spelling_scalar.var_[0])}")

# Now standardize the data with above mean and variance.
spellingmistake_essays = essays_spelling_scalar.transform(count_spellingmistake_essays.reshape(-1,
1))
```

Mean : 2.9424285714285716, Standard deviation : 1.9405742417529668

In [0]:

```
X = X + ['spellingmistake_essays']
```

## Spelling Mistakes-Count in Titles

In [0]:

```
titles_spelling_scalar = StandardScaler()
titles_spelling_scalar.fit(count_spellingmistake_titles.reshape(-1,1)) # finding the mean and
standard deviation of this data
print(f"Mean : {titles_spelling_scalar.mean_[0]}, Standard deviation :
{np.sqrt(titles_spelling_scalar.var_[0])}")

# Now standardize the data with above mean and variance.
spellingmistake_titles = titles_spelling_scalar.transform(count_spellingmistake_titles.reshape(-1,
1))
```

Mean : 0.15121428571428572, Standard deviation : 0.4267886192369756

In [0]:

```
X = X + ['spellingmistake_titles']
```

## Spelling Mistakes-Count in Resource\_suumary

In [0]:

```
resource_spelling_scalar = StandardScaler()
resource_spelling_scalar.fit(count_spellingmistake_resource.reshape(-1,1)) # finding the mean and
standard deviation of this data
print(f"Mean : {resource_spelling_scalar.mean_[0]}, Standard deviation :
{np.sqrt(resource_spelling_scalar.var_[0])}")
```

```
# Now standardize the data with above maen and variance.
spellingmistake_resource =
resource_spelling_scalar.transform(count_spellingmistake_resource.reshape(-1, 1))
```

Mean : 0.3830142857142857, Standard deviation : 0.7296574928956568

In [0]:

```
X = X + ['spellingmistake_resource']
```

## Similarity Between Essay\_1 and Essay\_2

In [0]:

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

# price_standardized = standardScalar.fit(project_data['price'].values)
# this will rise the error
# ValueError: Expected 2D array, got 1D array instead: array=[725.05 213.03 329. ... 399. 287.
73 5.5 ].
# Reshape your data either using array.reshape(-1, 1)

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

# Now standardize the data with above maen and variance.
similarity_standardised = digits_scalar.transform(similarity.values.reshape(-1, 1))
```

Mean : 0.00477685983983777, Standard deviation : 0.009890084992979064

In [0]:

```
X = X + ['similarity']
```

In [0]:

```
X_features_cn = X
```

In [0]:

```
X_cat_num = X_features_cn
```

X-features\_cn conatains all feature-names of only categorial and Numerical-Features

we need to append this `X_features_cn` to BOW when performing BOW and to TFIDF when performing TFIDF

### Converting each feature-vector to Dataframe

In [0]:

```
print(type(teacher_prefix_one_hot))
teacher_prefix_one_hot.shape
df = pd.DataFrame(teacher_prefix_one_hot.toarray().astype(np.float64))
type(df)

print(type(school_state_one_hot))
school_state_one_hot.shape
df1 = pd.DataFrame(school_state_one_hot.toarray().astype(np.float64))
type(df1)

print(type(sub_categories_one_hot))
sub_categories_one_hot.shape
df2 = pd.DataFrame(sub_categories_one_hot.toarray().astype(np.float64))
type(df2)

print(type(categories_one_hot))
categories_one_hot.shape
df3 = pd.DataFrame(categories_one_hot.toarray().astype(np.float64))
type(df3)

type(teacher_number_of_previously_posted_projects_standardized.tolist())
df4=teacher_number_of_previously_posted_projects_standardized.tolist()
type(df4)

type(price_standardized.tolist())
df5=price_standardized.tolist()
type(df5)
```

## Combine all numerical and categorical features

In [0]:

```
from scipy.sparse import hstack
# with the same hstack function we are concatenating a sparse matrix and a dense matrix :)
X_cn = hstack((school_state_one_hot[:69999],categories_one_hot[:69999],sub_categories_one_hot[:69999],project_grade_one_hot[:69999],teacher_prefix_one_hot[:69999],price_standardized[:69999],teacher_number_of_previously_posted_projects_standardized[:69999],quantity_standardised[:69999],digits_standardised[:69999],essays_words_standardised[:69999],summary_words_standardised[:69999],titles_words_standardised[:69999],neutrality_polarity_standardised[:69999],compound_polarity_standardised[:69999],positive_polarity_standardised[:69999],negative_polarity_standardised[:69999],spellingmistake_essays[:69999],spellingmistake_titles[:69999],spellingmistake_resource[:69999],similarity_standardised[:69999]))
X_cn.shape
type(X_cn)
```

Out[0]:

```
scipy.sparse.coo.coo_matrix
```

we are combining all the categorical and numerical features into a single `X_cn` Sparse Matrix

We are ignoring text features here

In [0]:

```
dk= pd.DataFrame(X_cn.toarray())
type(dk)
```

Out[0]:

```
pandas.core.frame.DataFrame
```

In [0]:



```
In [93]:
```

```
dk.columns = X
```

Creating Datframe for X\_cn as we need to concatenate text features into the dataframe

We can not add Text columns to Sparse matrix as the type of text is 'str and numerical columns as 'int'

dk is the dataframe conatining all categorical and numerical features

```
In [94]:
```

```
y = project_data['project_is_approved']  
type(y)
```

```
Out[94]:
```

```
pandas.core.series.Series
```

Taking the output into a series-(y)

```
In [0]:
```

```
k =  
pd.DataFrame({'preprocessed_essays':preprocessed_essays[:69999], 'preprocessed_titles':preprocessed_  
titles[:69999], 'preprocessed_resource_summary':preprocessed_resource_summary[:69999], 'y':y[:69999]  
})
```

k is the dataframe conatining all the text features and the output-(y) feature.

We should not be using hstack as the features are of strings and could not concatenate them

k is the dataframe conatining all text and output-y

## dataset - contains all features

Dataset is the Dataframe containing all the features text,Categorical and Numerical Features

we need to vectorize the text features only after splitting Dataset into train,test,split

dataset conatins all features with text in raw format and also output-y

```
In [0]:
```

```
dataset = pd.concat([dk,k],axis=1)
```

```
In [0]:
```

```
dataset.to_pickle('gdrive/My Drive/dataset_naivebayes_original.pkl')
```

```
In [0]:
```

```
dataset = pd.read_pickle('gdrive/My Drive/dataset_naivebayes_original.pkl')
```

X = Contains all the features-Names of the Dataset..We need to store them because when we convert them to sparse matrice's,We loose the column-Names

X needs to be stored in the order in which we are storing all the features(Numerical + Categorical)

## Train-Test-Split of Dataset

```
In [0]:
```

```

from sklearn.model_selection import train_test_split
X_1, X_test, y_1, y_test = train_test_split(dataset[:69900], y[:69900], test_size=0.3, random_state=
0, stratify=y[:69900])

# split the train data set into cross validation train and cross validation test
X_train, X_cv, y_train, y_cv = train_test_split(X_1, y_1, test_size=0.3, random_state=0, stratify=y_1
)

```

Split the essays into Train\_test\_split

We are splitting the dataset Randomly and by using stratify which means train and test-set contains equal no.of.y values

Stratify means train and test contain same proportion of 1 and 0 -samples or same ratio

while stratifying using cv, we need to stratify using y\_train

X\_train = train set of essay after cross=validation

X\_test = test set of essay

X\_cv = cv set of essay

X\_1 = train set before cross-Validation

# 1-BOW

## 1.1 Vectorizers of train, test, split of only Raw test-Features

### Essays\_Vectorizers\_BOW

In [0]:

```

X_train_essay = X_train[:, ['preprocessed_essays']]
X_cv_essay = X_cv[:, ['preprocessed_essays']]
X_test_essay = X_test[:, ['preprocessed_essays']]

```

In [98]:

```

# We are considering only the words which appeared in at least 10 documents (rows or projects).
vectorizer = CountVectorizer(min_df=10, ngram_range=(1,2), max_features=10000)
train_bow_essay = vectorizer.fit_transform(X_train_essay)
cv_bow_essay = vectorizer.transform(X_cv_essay)
test_bow_essay = vectorizer.transform(X_test_essay)

print(train_bow_essay.shape, cv_bow_essay.shape, test_bow_essay.shape)

```

(34251, 10000) (14679, 10000) (20970, 10000)

In [0]:

```
X = X + vectorizer.get_feature_names()
```

Vectorizing the train, test, cv sets of essays-Text features

min\_df=10 means, we are using all the words present only in min of 10 documents

We need to fit the vectorizer with train set and then transform to cv, test using the same vectorizer

It is because test and cv should contain the same words as Train-set

we are appending X = which contain BOW-features\_names of Essays to Categorical and Numerical-Features

### Titles\_Vectorizers\_bow

In [0]:

```
X_train_titles = X_train[:,['preprocessed_titles']]
X_cv_titles = X_cv[:,['preprocessed_titles']]
X_test_titles = X_test[:,['preprocessed_titles']]
X_1_titles = X_1[:,['preprocessed_titles']]
```

In [0]:

```
vectorizer3 = CountVectorizer(min_df=10)
train_bow_titles = vectorizer3.fit_transform(X_train_titles)
bow_titles_cv = vectorizer3.transform(X_cv_titles)
test_bow_titles = vectorizer3.transform(X_test_titles)
```

In [0]:

```
X= X + vectorizer3.get_feature_names()
```

## Vectorizing the train,test,cv sets of titles-Text features

min\_df=10 means, we are using all the words present only in min of 10 documents

Similar Vectorizing has to be done to Titles and titles are also the text vectors

In [0]:

```
X_train_summary = X_train[:,['preprocessed_resource_summary']]
X_cv_summary = X_cv[:,['preprocessed_resource_summary']]
X_test_summary = X_test[:,['preprocessed_resource_summary']]
```

In [0]:

```
vectorizer4 = CountVectorizer(min_df=10)
train_bow_summary = vectorizer4.fit_transform(X_train_summary)
bow_summary_cv = vectorizer4.transform(X_cv_summary)
test_bow_summary = vectorizer4.transform(X_test_summary)
```

In [0]:

```
X= X + vectorizer4.get_feature_names()
```

In [106]:

```
print(train_bow_essay.shape,test_bow_essay.shape,cv_bow_essay.shape)
print(train_bow_titles.shape,test_bow_titles.shape,bow_titles_cv.shape)
print(y_train.shape,y_test.shape,y_cv.shape)
```

```
(34251, 10000) (20970, 10000) (14679, 10000)
(34251, 1652) (20970, 1652) (14679, 1652)
(34251,) (20970,) (14679,)
```

we are using BOW of the text here

As we need to use `fit_transform` for train of essays and titles and their respective test-set/cv-set should be transformed because they both should have the same no.of.features (train/test and 1/cv-sets).

When transforming CV, Test features , they should have same no.of features/vectorizers similar to Train-set

We need to vectorize the Each train and testset separately and fit the train data and then transform the test data

## 1.2 Extract train, test of only numerical and categorical features

In [107]:

```
import scipy
X_train_cn =
Y_train_drop[['Unl. Inprocessed, casual', 'Inprocessed, titles', 'Inprocessed, resource, summary']] and
```

```

X_train.drop(['y', 'preprocessed_essays', 'preprocessed_titles', 'preprocessed_resource_summary'], axis=1)
print(X_train_cn.shape)
X_train_cn = scipy.sparse.csr_matrix(X_train_cn)
print(X_train_cn.shape)

X_test_cn =
X_test.drop(['y', 'preprocessed_essays', 'preprocessed_titles', 'preprocessed_resource_summary'], axis=1)
print(X_test_cn.shape)
X_test_cn = scipy.sparse.csr_matrix(X_test_cn)
print(X_test_cn.shape)

X_cv_cn =
X_cv.drop(['y', 'preprocessed_essays', 'preprocessed_titles', 'preprocessed_resource_summary'], axis=1)
print(X_cv_cn.shape)
X_cv_cn = scipy.sparse.csr_matrix(X_cv_cn)
print(X_cv_cn.shape)

```

```

(34251, 114)
(34251, 114)
(20970, 114)
(20970, 114)
(14679, 114)
(14679, 114)

```

From the original TrainTest,Cv sets of dataset, we need to drop text of essays and Titles and replace them with Vectorizers of Text of Essays and Titles

We need to keep the Categorical and Numerical columns also along with the vectors of essays and titles

### 1.3-Train,test,cv sets of ALL features

In [0]:

```

from scipy.sparse import hstack
from sklearn.preprocessing import StandardScaler

X_train_bow = hstack((X_train_cn,train_bow_essay,train_bow_titles,train_bow_summary))
X_train_bow = X_train_bow.tocsr()
train_scalar = StandardScaler(with_mean = False)
X_train_bow = train_scalar.fit_transform(X_train_bow)

```

In [0]:

```

X_test_bow =hstack((X_test_cn,test_bow_essay,test_bow_titles,test_bow_summary))
X_test_bow = X_test_bow.tocsr()
test_scalar = StandardScaler(with_mean = False)
X_test_bow = test_scalar.fit_transform(X_test_bow)

```

In [110]:

```

X_cv_bow = hstack((X_cv_cn,cv_bow_essay,bow_titles_cv,bow_summary_cv))
X_cv_bow = X_cv_bow.tocsr()
cv_scalar = StandardScaler(with_mean = False)
X_cv_bow = cv_scalar.fit_transform(X_cv_bow)

print(X_train_bow.shape,X_test_bow.shape,X_cv_bow.shape)

```

```

(34251, 14994) (20970, 14994) (14679, 14994)

```

Now using hstack concatenate all train sets of categorical,numerical,vectors of essays and vectors of titles -Features

Similarly concatenate all the test sets and cv sets with their respective features

Convert COO-matrix to CSR-Sparse matrix as the input given to the Logistic-Regression should be of Sparse Matrix and Not Dataframe as DF takes more time to Run.

## 1.4-Appling Logistic-Regression on BOW, SET 1

In [0]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score
from collections import Counter
from sklearn.metrics import accuracy_score
from sklearn.linear_model import SGDClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
```

## 1.5-AUC with trainset and CV-set using Dataset after CV-splitting

In [112]:

```
train_auc = []
cv_auc = []

alpha_values = [0.0001, 0.001, 0.01, 0.05, 0.1, 0.3, 0.5, 0.7, 1, 10, 50, 100, 150, 200, 500, 1000, 2000, 3000, 5000, 10000]

for i in tqdm(alpha_values):

    model_Logistic = SGDClassifier(loss='log', alpha=i)
    model_Logistic.fit(X_train_bow, y_train)
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
    # not the predicted outputs

    y_train_pred = []
    for k in range(0, X_train_bow.shape[0], 1000):

        y_train_pred.extend(model_Logistic.predict_proba(X_train_bow[k:k+1000])[:, 1])

    y_cv_pred = []

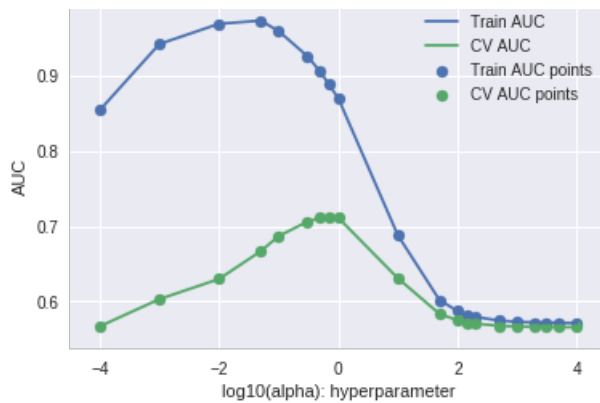
    for k in range(0, X_cv_bow.shape[0], 500):

        y_cv_pred.extend(model_Logistic.predict_proba(X_cv_bow[k:k+500])[:, 1])

    train_auc.append(roc_auc_score(y_train, y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

plt.plot(np.log10(alpha_values), train_auc, label='Train AUC')
plt.plot(np.log10(alpha_values), cv_auc, label='CV AUC')
plt.scatter(np.log10(alpha_values), train_auc, label='Train AUC points')
plt.scatter(np.log10(alpha_values), cv_auc, label='CV AUC points')
plt.legend()
plt.xlabel("log10(alpha): hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

100%|██████████| 20/20 [00:08<00:00, 2.42it/s]



We can check the values from 0 and below such that third and fourth point left to zero's are Considered the best-Alpha = 1, as cv-train is Max and Gap between cv & train-AUC is less

Check the Alpha-Values from the input Given ALpha-Range

1. we are using AUC-score as metric to predict the best-alpha using Roc by Logistic\_Regression

2. AUC-score we calculated using the train set after-cv and cv set

1. We trained the model using the train set and predicted the model on CV set and also the train-set to find the train error and cv-error, but we are using AUC as a metric to find the best-alpha
2. Calculated AUC-score using the both train and cv sets.
3. The best-alpha is where AUC of cv is MAX at particular-alpha and nearest to the train-AUC graph
4. If we do take extreme values of alpha into consideration, we are underfitting the data.
5. We do not consider accuracy as the metric as Accuracy gives only details about correctly labeled, but there will be no info regarding the misclassified labels, so use AUC as the best metric

8. We are plotting log(alpha) because we need to accommodate large and small values of alpha

We need to form a set containing cv set and train set, We will be using this set to train the model with optimal\_alpha and predict the test-data

## 1.7-ROC-Curve with optimal\_alpha for train and test-sets

In [113]:

```
from sklearn.metrics import roc_curve, auc

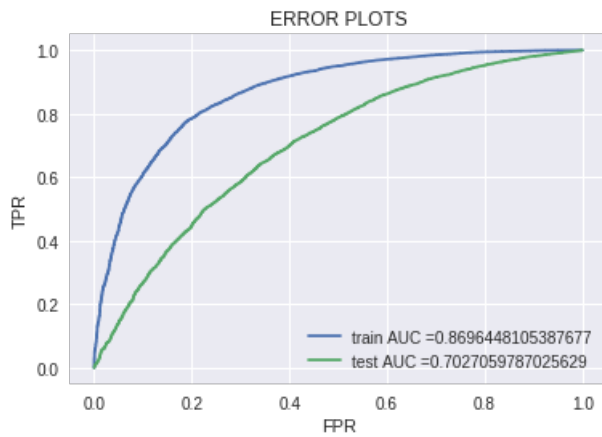
optimal_alpha = 1
model_Logistic_bow = SGDClassifier(loss='log', alpha=optimal_alpha)
model_Logistic_bow.fit(X_train_bow, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
# not the predicted outputs

y_train_pred = []
for k in range(0, X_train_bow.shape[0], 100):
    y_train_pred.extend(model_Logistic_bow.predict_proba(X_train_bow[k:k+100])[:, 1])

y_test_pred = []
for k in range(0, X_test_bow.shape[0], 100):
    y_test_pred.extend(model_Logistic_bow.predict_proba(X_test_bow[k:k+100])[:, 1])

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

plt.plot(train_fpr, train_tpr, label="train AUC =" + str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC =" + str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ERROR PLOTS")
plt.show()
```



In [114]:

```
# we are writing our own function for predict, with defined threshold
# we will pick a threshold that will give the least fpr
def predict(proba, threshold, fpr, tpr):

    t = threshold[np.argmax(tpr*(1-fpr))]

    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high

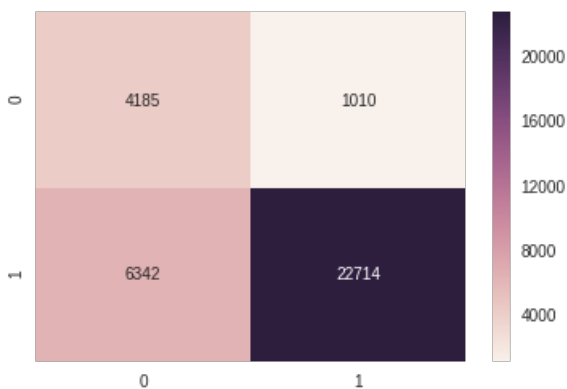
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
    predictions = []
    for i in proba:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions

import seaborn as sns
from sklearn.metrics import confusion_matrix
print("train Confusion Matrix")
cm_train=confusion_matrix(y_train,predict( y_train_pred, tr_thresholds, train_fpr, train_tpr))
sns.heatmap(cm_train,annot=True,fmt='.5g')
```

train Confusion Matrix  
the maximum value of tpr\*(1-fpr) 0.6297493168414224 for threshold 0.816

Out[114]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x7f8c25e49d68>



In [115]:

```
y_train.value_counts()
```

Out[115]:

```
1    29056
0     5195
Name: project is approved. dtype: int64
```

```
Name: project_is_approved, dtype: int64
```

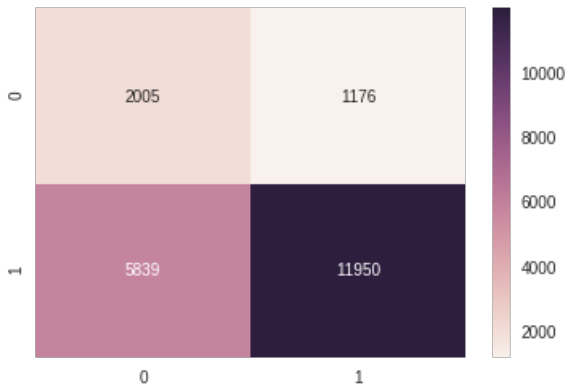
In [116]:

```
print("Test confusion matrix")
cm=confusion_matrix(y_test, predict( y_test_pred, te_thresholds, test_fpr, test_tpr))
print(sns.heatmap(cm,annot=True,fmt='.5g'))
```

Test confusion matrix

the maximum value of  $tpr \cdot (1 - fpr)$  0.4234158176333993 for threshold 0.828

AxesSubplot(0.125,0.125;0.62x0.755)



In [117]:

```
y_test.value_counts()
```

Out[117]:

```
1    17789
```

```
0     3181
```

Name: project\_is\_approved, dtype: int64

We are calculating Train-AUC and Test\_AUC for train data and test data

The confusion Matrix for train data and Test data is calculated using the train data and test data

The ROC-plots are also plotted.

best-alpha=1,the AUC of best model is 0.701 for the test-set which is unseen data-points, but for the train set- it is 0.86

## 1.8-BOW-Feature-Importance(Top 20 Features)

In [0]:

```
#https://stackoverflow.com/questions/11116697/how-to-get-most-informative-features-for-scikit-learn-classifiers
```

```
top_20_features=sorted(zip(model_Logistic_bow.coef_[0],X),reverse=True)[:20]
```

In [119]:

```
top_20_features
```

Out[119]:

```
[(0.04683566919977522, 'my'),
 (0.029402168450956535, 'nannan'),
 (0.01741259171079789, 'need'),
 (0.01719124034260788, 'students'),
 (0.015435698916988556, 'previously_posted_projects'),
 (0.015300820409651026, 'use'),
 (0.015084354578120587, 'students'),
 (0.011963999721355185, 'chromebooks'),
 (0.011827060873267523, 'books'),
 (0.011734022897922632, 'used'),
```



```
(0.011096867610707457, 'the'),
(0.010641379022244168, 'allow'),
(0.010318115459644646, 'Literacy_Language'),
(0.010217609420069015, 'Literacy'),
(0.01020771555878269, 'using'),
(0.010177004509169314, 'balls'),
(0.010111669553411555, 'neutrality_polarity_standardised'),
(0.009315130712060821, 'chromebooks'),
(0.009148220712146325, 'currently'),
(0.009040564714986246, 'Mrs')]
```

## 2-TFIDF

### 2.1-Vectorizer of train,test,split with TFIDF-

In [120]:

```
# We are considering only the words which appeared in at least 10 documents(rows or projects).
vectorizer3 = TfidfVectorizer(min_df=10,ngram_range=(1,2),max_features=10000)
train_tfidf_essay = vectorizer3.fit_transform(X_train_essay)
cv_tfidf_essay = vectorizer3.transform(X_cv_essay)
test_tfidf_essay = vectorizer3.transform(X_test_essay)
print(train_tfidf_essay.shape,cv_tfidf_essay.shape,test_tfidf_essay.shape)
```

```
(34251, 10000) (14679, 10000) (20970, 10000)
```

In [0]:

```
X= X_cat_num + vectorizer3.get_feature_names()
```

Vectorizing using TFIDF the train,test,cv sets of Essay-Text features

min\_df=10 means, we are using all the words present only in min of 10 documents

In [0]:

```
vectorizer4 = TfidfVectorizer(min_df=10)
train_tfidf_titles = vectorizer4.fit_transform(X_train_titles)
tfidf_titles_cv = vectorizer4.transform(X_cv_titles)
test_tfidf_titles = vectorizer4.transform(X_test_titles)
```

In [0]:

```
X= X + vectorizer4.get_feature_names()
```

In [0]:

```
vectorizer5 = TfidfVectorizer(min_df=10)
train_tfidf_summary = vectorizer5.fit_transform(X_train_summary)
tfidf_summary_cv = vectorizer5.transform(X_cv_summary)
test_tfidf_summary = vectorizer5.transform(X_test_summary)
```

In [0]:

```
X = X + vectorizer5.get_feature_names()
```

we are using TFIDF of the text here

As we need to use fit\_transform for train of essays and titles and their respective test-set/cv-set should be transformed because they both should have the same no.of.features (train/test and 1/cv-sets).

When transforming CV,Test features \, they should have same no.of features/vectorizers similar to Train-set

Vectorizing using TFIDF the train,test,cv sets of titles-Text features

min\_df=10 means, we are using all the words present only in min of 10 documents

## 2.2-Train,test,cv sets of ALL features -Concatenating

In [0]:

```
from scipy.sparse import hstack
from sklearn import preprocessing

X_train_tfidf = hstack((X_train_cn,train_tfidf_essay,train_tfidf_titles,train_tfidf_summary))
X_train_tfidf = X_train_tfidf.tocsr()
train_scaler = StandardScaler(with_mean = False)
X_train_tfidf = train_scaler.fit_transform(X_train_tfidf)
```

In [0]:

```
X_test_tfidf =hstack((X_test_cn,test_tfidf_essay,test_tfidf_titles,test_tfidf_summary))
X_test_tfidf = X_test_tfidf.tocsr()
test_scaler = StandardScaler(with_mean = False)
X_test_tfidf = test_scaler.fit_transform(X_test_tfidf)
```

In [0]:

```
X_cv_tfidf = hstack((X_cv_cn,cv_tfidf_essay,tfidf_titles_cv,tfidf_summary_cv))
X_cv_tfidf = X_cv_tfidf.tocsr()
cv_scaler = StandardScaler(with_mean = False)
X_cv_tfidf = cv_scaler.fit_transform(X_cv_tfidf)
```

Now using hstack concatenate all train sets of categorical,numerical,vectors of essays and vectors of titles -Features

Similarly concatenate all the test sets and cv sets with their respective features

Convert COO-matrix to CSR-Sparse matrix as the input given to the Logistic-Regression should be of Sparse Matrix and Not Dataframe

## 2.3-AUC with trainset and CV-set using Dataset after CV-splitting

In [129]:

```
train_auc = []
cv_auc = []

alpha_values =[0.0001,0.001,0.01,0.05,0.1,0.3,0.5,0.7,1,10,50,100,150,200,500,1000,2000,3000,5000,10000]

for i in tqdm(alpha_values):
    model_logistic_tfidf = SGDClassifier(loss='log',alpha=i)
    model_logistic_tfidf.fit(X_train_tfidf, y_train)
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
    # not the predicted outputs

    y_train_pred = []
    for k in range(0,X_train_tfidf.shape[0],1000):
        y_train_pred.extend(model_logistic_tfidf.predict_proba(X_train_tfidf[k:k+1000])[:,1])

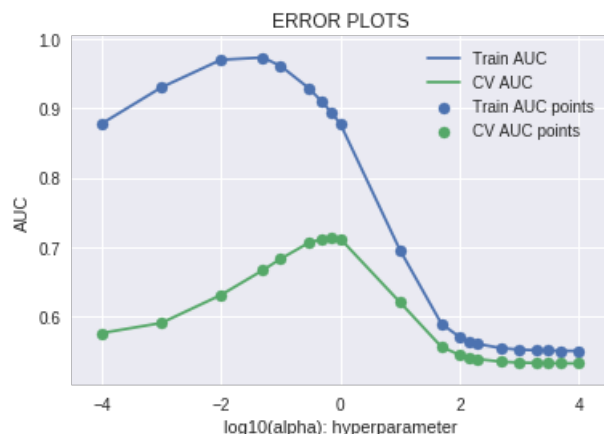
    y_cv_pred = []

    for k in range(0, X_cv_tfidf.shape[0],500):
        y_cv_pred.extend(model_logistic_tfidf.predict_proba(X_cv_tfidf[k:k+500])[:,1])

    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
```

```
plt.plot(np.log10(alpha_values), train_auc, label='Train AUC')
plt.plot(np.log10(alpha_values), cv_auc, label='CV AUC')
plt.scatter(np.log10(alpha_values), train_auc, label='Train AUC points')
plt.scatter(np.log10(alpha_values), cv_auc, label='CV AUC points')
plt.legend()
plt.xlabel("log10(alpha): hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

100%|██████████| 20/20 [00:09<00:00, 2.18it/s]



We can check the values from 0 and below such that third and fourth point left to zero's are Considered the best-Alpha = 1, as cv-train is Max and Gap between cv & train-AUC is less

Check the Alpha-Values from the input Given Alpha-Range

If not sure about the best-Alpha, Count the point in the alpha\_range and check the values in the alpha\_range

1. we are using AUC-score as metric to predict the best-alpha using Roc by Logistic-Regression

2. AUC-score we calculated using the train set after-cv and cv set

1. We trained the model using the train set and predicted the model on CV set and also the train-set to find the train error and cv-error, but we are using AUC as a metric to find the best-alpha
2. Calculated AUC-score using the both train and cv sets.
3. The best-alpha is where AUC of cv is MAX at particular-alpha and nearest to the train-AUC graph
4. If we do take extreme values of alpha into consideration, we are underfitting the data.
5. We do not consider accuracy as the metric as Accuracy gives only details about correctly labeled, but there will be no info regarding the misclassified labels, SO use AUC as the best metric

8. We are plotting log(alpha) because we need to accommodate large and small values of alpha

## 2.5-ROC-Curve with optimal\_alpha for train and test-sets

In [130]:

```
from sklearn.metrics import roc_curve, auc

optimal_alpha = 1
model_logistic_tfidf = SGDClassifier(loss='log', alpha=optimal_alpha)
model_logistic_tfidf.fit(X_train_tfidf, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs

y_train_pred = []
for k in range(0, X_train_tfidf.shape[0], 1000):
    y_train_pred.extend(model_logistic_tfidf.predict_proba(X_train_tfidf[k:k+1000])[:, 1])

y_test_pred = []
```

```

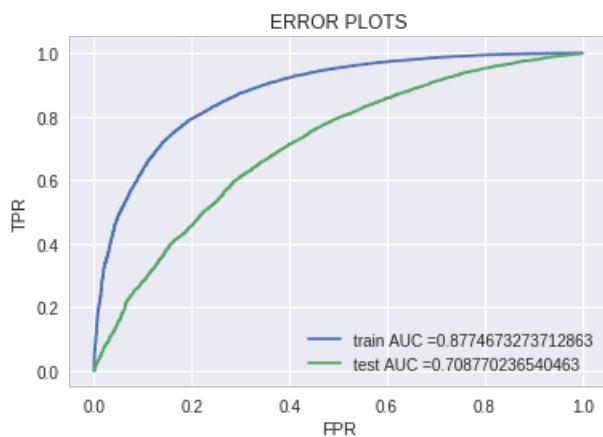
y_test_pred = []

for k in range(0, X_test_tfidf.shape[0],100):
    y_test_pred.extend(model_logistic_tfidf.predict_proba(X_test_tfidf[k:k+100]))[:,1])

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

plt.plot(train_fpr, train_tpr, label="train AUC =" +str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC =" +str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ERROR PLOTS")
plt.show()

```



In [131]:

```

# we are writing our own function for predict, with defined threshold
# we will pick a threshold that will give the least fpr
def predict(proba, threshold, fpr, tpr):

    t = threshold[np.argmax(tpr*(1-fpr))]

    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high

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

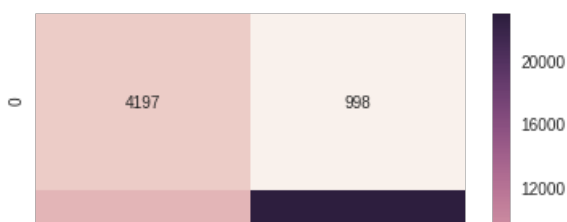
import seaborn as sns
from sklearn.metrics import confusion_matrix
print("train Confusion Matrix")
cm_train=confusion_matrix(y_train,predict( y_train_pred, tr_thresholds, train_fpr, train_tpr))
sns.heatmap(cm_train,annot=True,fmt='.5g')

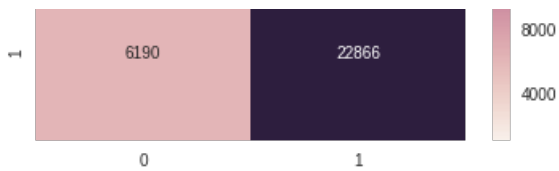
```

train Confusion Matrix  
the maximum value of  $tpr \cdot (1 - fpr)$  0.6357813579856945 for threshold 0.821

Out[131]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x7f8c166b1438>





In [132]:

```
y_train.value_counts()
```

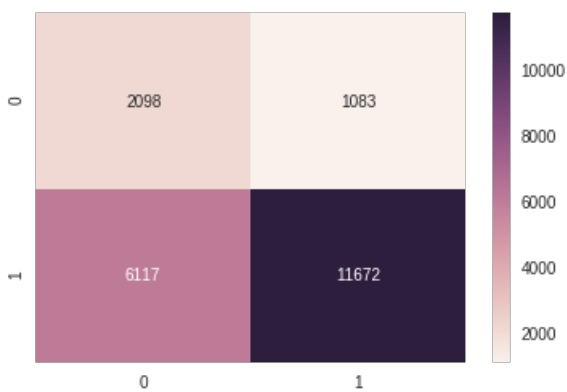
Out[132]:

```
1    29056
0     5195
Name: project_is_approved, dtype: int64
```

In [133]:

```
print("Test confusion matrix")
cm=confusion_matrix(y_test, predict(y_test_pred, te_thresholds, test_fpr, test_tpr))
print(sns.heatmap(cm,annot=True,fmt='.5g'))
```

Test confusion matrix  
the maximum value of  $tpr \cdot (1 - fpr)$  0.43274848737273736 for threshold 0.835  
AxesSubplot(0.125,0.125;0.62x0.755)



In [134]:

```
y_test.value_counts()
```

Out[134]:

```
1    17789
0     3181
Name: project_is_approved, dtype: int64
```

## 2.6- TFIDF-Feature-Importance

In [0]:

```
#https://stackoverflow.com/questions/11116697/how-to-get-most-informative-features-for-scikit-learn-classifiers
```

```
top_20_features=sorted(zip(model_logistic_tfidf.coef_[0],X),reverse=True)[:20]
```

In [136]:

```
top_20_features
```

Out[136]:

```
[(0.02399036104559054, 'students'),
 (0.023464520000000005, 'math')]
```

```
(0.023464526852058954, 'my'),
(0.022054144992075482, 'nannan'),
(0.019072542944332818, 'students'),
(0.018343324329663256, 'need'),
(0.017288367622365956, 'use'),
(0.015426344252330863, 'previously_posted_projects'),
(0.013836823413395184, 'the'),
(0.013406428057353969, 'books'),
(0.01302536074663901, 'used'),
(0.012742296214341762, 'essays_words_standardised'),
(0.012686917137134691, 'chromebooks'),
(0.011886079475032613, 'using'),
(0.011830150900168344, 'allow'),
(0.011574804181903714, 'Literacy_Language'),
(0.011443531663098831, 'Mrs'),
(0.011431782759844814, 'balls'),
(0.011101971422770031, 'Literacy'),
(0.010863250110541418, 'also'),
(0.010615814891236655, 'neutrality_polarity_standardised')]
```

## Glove-Vector-Import

In [0]:

```
# stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-pickle-to-sa
ve-and-load-variables-in-python/
# make sure you have the glove_vectors file
with open('gdrive/My Drive/glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words = set(model.keys())
```

## 3-TFIDF-W2v

### Train\_essay

In [0]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model_train = TfidfVectorizer()
tfidf_model_train.fit_transform(X_train_essay)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model_train.get_feature_names(), list(tfidf_model_train.idf_)))
tfidf_words_train = set(tfidf_model_train.get_feature_names())
```

In [139]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_train = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train_essay): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight_train = 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) and (word in tfidf_words_train):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
            value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
            idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight_train += tf_idf
    if tf_idf_weight_train != 0:
        vector /= tf_idf_weight_train
    tfidf_w2v_vectors_train.append(vector)
```

100%|██████████| 34251/34251 [01:14<00:00, 456.92it/s]

## Train\_Titles

In [0]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model_train_titles = TfidfVectorizer()
tfidf_model_train_titles.fit_transform(X_train_titles)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model_train_titles.get_feature_names(), list(tfidf_model_train_titles.idf_)))
tfidf_words_train_titles = set(tfidf_model_train_titles.get_feature_names())
```

In [141]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_train_titles = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train_titles): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight_train_titles = 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) and (word in tfidf_words_train_titles):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
            value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
            idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight_train_titles += tf_idf
    if tf_idf_weight_train_titles != 0:
        vector /= tf_idf_weight_train_titles
    tfidf_w2v_vectors_train_titles.append(vector)

print(len(tfidf_w2v_vectors_train_titles))
print(len(tfidf_w2v_vectors_train_titles[0]))
```

100%|██████████| 34251/34251 [00:01<00:00, 24598.01it/s]

34251  
300

## Test\_Essay

In [0]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model_train = TfidfVectorizer()
tfidf_model_train.fit_transform(X_train_essay)

tfidf_model_train.transform(X_test_essay)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model_train.get_feature_names(), list(tfidf_model_train.idf_)))
tfidf_words_test = set(tfidf_model_train.get_feature_names())
```

In [143]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_test = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test_essay): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight_test = 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) and (word in tfidf_words_train):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
            value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
            idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight_test += tf_idf
```

```

tf_idf_weight_test != 0:
    vector /= tf_idf_weight_test
tfidf_w2v_vectors_test.append(vector)

print(len(tfidf_w2v_vectors_test))
print(len(tfidf_w2v_vectors_test[0]))

```

100%|██████████| 20970/20970 [00:46<00:00, 453.02it/s]

20970  
300

## Test\_Titles

In [0]:

```

# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model_train_titles = TfidfVectorizer()
tfidf_model_train_titles.fit_transform(X_train_titles)

tfidf_model_train_titles.transform(X_test_titles)
# we are converting aa dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model_train_titles.get_feature_names(), list(tfidf_model_train_titles.idf_)))
tfidf_words_test_titles = set(tfidf_model_train_titles.get_feature_names())

```

In [145]:

```

# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_test_titles = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test_titles): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight_test_titles = 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) and (word in tfidf_words_train_titles):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
            value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
            idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight_test_titles += tf_idf
    if tf_idf_weight_test_titles != 0:
        vector /= tf_idf_weight_test_titles
    tfidf_w2v_vectors_test_titles.append(vector)

print(len(tfidf_w2v_vectors_test_titles))
print(len(tfidf_w2v_vectors_test_titles[0]))

```

100%|██████████| 20970/20970 [00:00<00:00, 21756.03it/s]

20970  
300

## CV\_Essay

In [0]:

```

# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model_train = TfidfVectorizer()
tfidf_model_train.fit_transform(X_train_essay)
tfidf_model_train.transform(X_cv_essay)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model_train.get_feature_names(), list(tfidf_model_train.idf_)))
tfidf_words_cv = set(tfidf_model_train.get_feature_names())

```



In [147]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_cv = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv_essay): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tfidf_weight_cv = 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) and (word in tfidf_words_train):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
            value((sentence.count(word)/len(sentence.split())))
            tfidf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
            idf value for each word
            vector += (vec * tfidf) # calculating tfidf weighted w2v
            tfidf_weight_cv += tfidf
    if tfidf_weight_cv != 0:
        vector /= tfidf_weight_cv
    tfidf_w2v_vectors_cv.append(vector)

print(len(tfidf_w2v_vectors_cv))
print(len(tfidf_w2v_vectors_cv[0]))
```

100%|██████████| 14679/14679 [00:32<00:00, 451.79it/s]

14679

300

## CV\_Titles

In [0]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model_1_titles = TfidfVectorizer()
tfidf_model_1_titles.fit_transform(X_train_titles)
tfidf_model_1_titles.transform(X_cv_titles)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model_1_titles.get_feature_names(), list(tfidf_model_1_titles.idf_)))
tfidf_words_cv_titles = set(tfidf_model_1_titles.get_feature_names())
```

In [149]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_cv_titles = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv_titles): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tfidf_weight_cv_titles = 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) and (word in tfidf_words_train_titles):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
            value((sentence.count(word)/len(sentence.split())))
            tfidf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
            idf value for each word
            vector += (vec * tfidf) # calculating tfidf weighted w2v
            tfidf_weight_cv_titles += tfidf
    if tfidf_weight_cv_titles != 0:
        vector /= tfidf_weight_cv_titles
    tfidf_w2v_vectors_cv_titles.append(vector)

print(len(tfidf_w2v_vectors_cv_titles))
print(len(tfidf_w2v_vectors_cv_titles[0]))
```

100%|██████████| 14679/14679 [00:00<00:00, 24725.14it/s]

14679

300

## Train\_Summary

In [0]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model_train = TfidfVectorizer()
tfidf_model_train.fit_transform(X_train_summary)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model_train.get_feature_names(), list(tfidf_model_train.idf_)))
tfidf_words_train_summary = set(tfidf_model_train.get_feature_names())
```

In [151]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_train_summary = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train_summary): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight_train = 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) and (word in tfidf_words_train_summary):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
            value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
            idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight_train += tf_idf
    if tf_idf_weight_train != 0:
        vector /= tf_idf_weight_train
    tfidf_w2v_vectors_train_summary.append(vector)

print(len(tfidf_w2v_vectors_train_summary))
print(len(tfidf_w2v_vectors_train_summary[0]))
```

100%|██████████| 34251/34251 [00:04<00:00, 7758.62it/s]

34251  
300

## Test\_Summary

In [0]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model_train = TfidfVectorizer()
tfidf_model_train.fit_transform(X_train_summary)

tfidf_model_train.transform(X_test_summary)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model_train.get_feature_names(), list(tfidf_model_train.idf_)))
tfidf_words_test = set(tfidf_model_train.get_feature_names())
```

In [153]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_test_summary = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test_summary): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight_test = 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) and (word in tfidf_words_train_summary):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
            value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
            idf value for each word
```

```

for value in each word
    vector += (vec * tf_idf) # calculating tfidf weighted w2v
    tf_idf_weight_test += tf_idf
if tf_idf_weight_test != 0:
    vector /= tf_idf_weight_test
tfidf_w2v_vectors_test_summary.append(vector)

```

100%|██████████| 20970/20970 [00:02<00:00, 8101.92it/s]

## CV\_Summary

In [0]:

```

# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model_1 = TfidfVectorizer()
tfidf_model_1.fit_transform(X_train_summary)
tfidf_model_1.transform(X_cv_summary)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model_1.get_feature_names(), list(tfidf_model_1.idf_)))
tfidf_words_cv = set(tfidf_model_1.get_feature_names())

```

In [155]:

```

# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_cv_summary = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv_summary): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight_cv = 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) and (word in tfidf_words_train_summary):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
            value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
            idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight_cv += tf_idf
    if tf_idf_weight_cv != 0:
        vector /= tf_idf_weight_cv
    tfidf_w2v_vectors_cv_summary.append(vector)

print(len(tfidf_w2v_vectors_cv_summary))
print(len(tfidf_w2v_vectors_cv_summary[0]))

```

100%|██████████| 14679/14679 [00:01<00:00, 8823.88it/s]

14679

300

## 3.1-Train,test,cv sets of ALL features

In [0]:

```

from scipy.sparse import hstack
from sklearn.preprocessing import StandardScaler

X_train_tfidf_w2v = hstack((X_train_cn,tfidf_w2v_vectors_train,tfidf_w2v_vectors_train_titles,tfidf_w2v_vectors_train_summary))
X_train_tfidf_w2v = X_train_tfidf_w2v.tocsr()
train_scalar = StandardScaler(with_mean = False)
X_train_tfidf_w2v = train_scalar.fit_transform(X_train_tfidf_w2v)

```

In [0]:

```

X_test_tfidf_w2v
= hstack((X_test_cn,tfidf_w2v_vectors_test,tfidf_w2v_vectors_test_titles,tfidf_w2v_vectors_test_summary))

```

```
X_test_tfidf_w2v = X_test_tfidf_w2v.tocsr()
test_scalar = StandardScaler(with_mean = False)
X_test_tfidf_w2v = test_scalar.fit_transform(X_test_tfidf_w2v)
```

In [158]:

```
X_cv_tfidf_w2v=
hstack((X_cv_cn,tfidf_w2v_vectors_cv,tfidf_w2v_vectors_cv_titles,tfidf_w2v_vectors_cv_summary))
X_cv_tfidf_w2v = X_cv_tfidf_w2v.tocsr()
cv_scalar = StandardScaler(with_mean = False)
X_cv_tfidf_w2v = cv_scalar.fit_transform(X_cv_tfidf_w2v)

print(X_train_tfidf_w2v.shape,X_test_tfidf_w2v.shape,X_cv_tfidf_w2v.shape)

(34251, 1014) (20970, 1014) (14679, 1014)
```

## 3.2-Applying Logistic-Regression on TFIDF-W2v, SET 3

In [159]:

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt

train_auc = []
cv_auc = []
alpha_values =[0.0001,0.001,0.01,0.05,0.1,0.3,0.5,0.7,1,10,50,100,150,200,500,1000,2000,3000,5000,10000]

for i in tqdm(alpha_values):
    Model_tfidf_w2v = SGDClassifier(loss='log',alpha=i)
    Model_tfidf_w2v.fit(X_train_tfidf_w2v, y_train)
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
    # not the predicted outputs

    y_train_pred = []
    for k in range(0,X_train_tfidf_w2v.shape[0],1000):
        y_train_pred.extend(Model_tfidf_w2v.predict_proba(X_train_tfidf_w2v[k:k+1000])[:,1])

    y_cv_pred = []

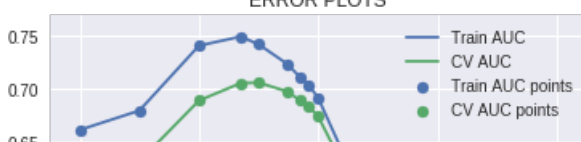
    for k in range(0, X_cv_tfidf_w2v.shape[0],100):
        y_cv_pred.extend(Model_tfidf_w2v.predict_proba(X_cv_tfidf_w2v[k:k+100])[:,1])

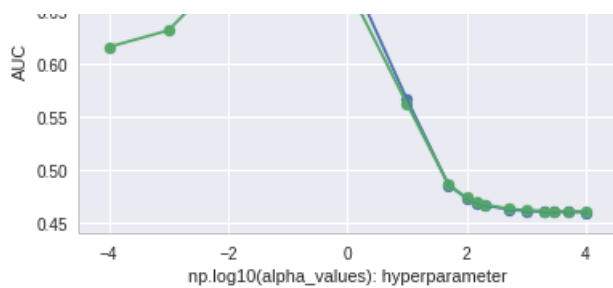
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

plt.plot(np.log10(alpha_values), train_auc, label='Train AUC')
plt.plot(np.log10(alpha_values), cv_auc, label='CV AUC')
plt.scatter(np.log10(alpha_values), train_auc, label='Train AUC points')
plt.scatter(np.log10(alpha_values), cv_auc, label='CV AUC points')
plt.legend()
plt.xlabel("np.log10(alpha_values): hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

100% |██████████| 20/20 [00:29<00:00, 1.46s/it]

ERROR PLOTS





### 3.3-ROC-Curve with optimal\_alpha for train and test-sets

In [162]:

```
from sklearn.metrics import roc_curve, auc

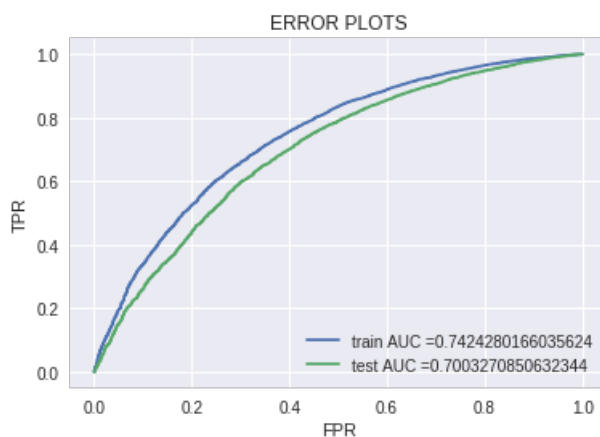
best_alpha = 0.1
model_tfidf_w2v = SGDClassifier(loss='log', alpha=best_alpha)
model_tfidf_w2v.fit(X_train_tfidf_w2v, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
# class
# not the predicted outputs

y_train_pred = []
for k in range(0, X_train_tfidf_w2v.shape[0], 1000):
    y_train_pred.extend(model_tfidf_w2v.predict_proba(X_train_tfidf_w2v[k:k+1000])[:,1])

y_test_pred = []
for k in range(0, X_test_tfidf_w2v.shape[0], 1000):
    y_test_pred.extend(model_tfidf_w2v.predict_proba(X_test_tfidf_w2v[k:k+1000])[:,1])

test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)
train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_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("FPR")
plt.ylabel("TPR")
plt.title("ERROR PLOTS")
plt.show()
```



In [163]:

```
# we are writing our own function for predict, with defined threshold
# we will pick a threshold that will give the least fpr
def predict(proba, threshold, fpr, tpr):

    t = threshold[np.argmax(tpr*(1-fpr))]

    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high
```

```

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

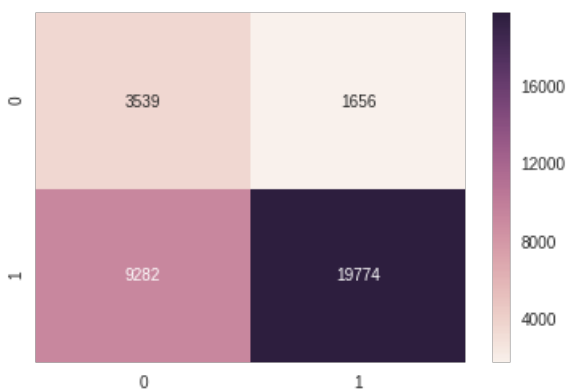
import seaborn as sns
from sklearn.metrics import confusion_matrix
print("train Confusion Matrix")
cm_train=confusion_matrix(y_train,predict( y_train_pred, tr_thresholds, train_fpr, train_tpr))
sns.heatmap(cm_train,annot=True,fmt='.5g')

```

train Confusion Matrix  
the maximum value of tpr\*(1-fpr) 0.4636109806744031 for threshold 0.836

Out[163]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x7f8c1106f828>



In [164]:

```
y_train.value_counts()
```

Out[164]:

```

1    29056
0     5195
Name: project_is_approved, dtype: int64

```

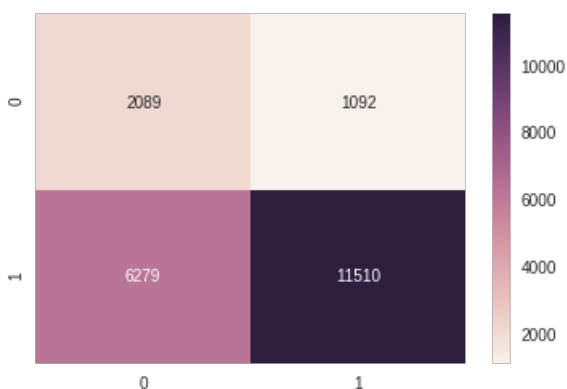
In [165]:

```

print("Test confusion matrix")
cm=confusion_matrix(y_test, predict( y_test_pred, te_thresholds, test_fpr, test_tpr))
print(sns.heatmap(cm,annot=True,fmt='.5g'))

```

Test confusion matrix  
the maximum value of tpr\*(1-fpr) 0.42491157258929374 for threshold 0.847  
AxesSubplot(0.125,0.125;0.62x0.755)



In [166]:

```
y_test.value_counts()
```

Out[166]:

```
1    17789
0     3181
```

Name: project\_is\_approved, dtype: int64

## 4-AVG-W2v

### Train\_Essay

In [167]:

```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_essays_vectors_train = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train_essay): # for each review/sentence
    vector = np.zeros(300) # 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) and (word in tfidf_words_train):
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_essays_vectors_train.append(vector)

print(len(avg_w2v_essays_vectors_train))
print(len(avg_w2v_essays_vectors_train[0]))
```

100%|██████████| 34251/34251 [00:11<00:00, 2884.67it/s]

```
34251
300
```

### Train\_titles

In [168]:

```
# average Word2Vec
# 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_titles): # for each review/sentence
    vector = np.zeros(300) # 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) and (word in tfidf_words_train_titles):
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_titles_vectors_train.append(vector)

print(len(avg_w2v_titles_vectors_train))
print(len(avg_w2v_titles_vectors_train[0]))
```

100%|██████████| 34251/34251 [00:00<00:00, 59217.57it/s]

```
34251
300
```

## Test\_Essay

In [169]:

```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_essays_vectors_test = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test_essay): # for each review/sentence
    vector = np.zeros(300) # 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) and (word in tfidf_words_train):
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_essays_vectors_test.append(vector)

print(len(avg_w2v_essays_vectors_test))
print(len(avg_w2v_essays_vectors_test[0]))
```

100%|██████████| 20970/20970 [00:07<00:00, 2916.57it/s]

20970  
300

## Test\_Titles

In [170]:

```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_titles_vectors_test = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test_titles): # for each review/sentence
    vector = np.zeros(300) # 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) and (word in tfidf_words_train_titles):
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_titles_vectors_test.append(vector)

print(len(avg_w2v_titles_vectors_test))
print(len(avg_w2v_titles_vectors_test[0]))
```

100%|██████████| 20970/20970 [00:00<00:00, 58743.48it/s]

20970  
300

## CV\_Essay

In [171]:

```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_essays_vectors_cv = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv_essay): # for each review/sentence
    vector = np.zeros(300) # 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) and (word in tfidf_words_train):
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
```



```
vector /= cnt_words
avg_w2v_essays_vectors_cv.append(vector)
```

```
print(len(avg_w2v_essays_vectors_cv))
print(len(avg_w2v_essays_vectors_cv[0]))
```

100%|██████████| 14679/14679 [00:05<00:00, 2920.06it/s]

14679  
300

## CV\_Titles

In [172]:

```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_titles_vectors_cv = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv_titles): # for each review/sentence
    vector = np.zeros(300) # 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) and (word in tfidf_words_train_titles):
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_titles_vectors_cv.append(vector)

print(len(avg_w2v_titles_vectors_cv))
print(len(avg_w2v_titles_vectors_cv[0]))
```

100%|██████████| 14679/14679 [00:00<00:00, 56360.27it/s]

14679  
300

## Train\_Summary

In [173]:

```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_summary_vectors_train = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train_summary): # for each review/sentence
    vector = np.zeros(300) # 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) and (word in tfidf_words_train_summary):
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_summary_vectors_train.append(vector)

print(len(avg_w2v_summary_vectors_train))
print(len(avg_w2v_summary_vectors_train[0]))
```

100%|██████████| 34251/34251 [00:01<00:00, 25803.13it/s]

34251  
300

## Test\_Summary

In [174]:

```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_summary_vectors_test = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test_summary): # for each review/sentence
    vector = np.zeros(300) # 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) and (word in tfidf_words_train_summary):
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_summary_vectors_test.append(vector)

print(len(avg_w2v_summary_vectors_test))
print(len(avg_w2v_summary_vectors_test[0]))
```

100%|██████████| 20970/20970 [00:00<00:00, 26078.71it/s]

20970  
300

## CV\_Summary

In [175]:

```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_summary_vectors_cv = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv_summary): # for each review/sentence
    vector = np.zeros(300) # 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) and (word in tfidf_words_train_summary):
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_summary_vectors_cv.append(vector)

print(len(avg_w2v_summary_vectors_cv))
print(len(avg_w2v_summary_vectors_cv[0]))
```

100%|██████████| 14679/14679 [00:00<00:00, 25234.05it/s]

14679  
300

## 4.1-Train,test,cv sets of ALL features

In [0]:

```
from scipy.sparse import hstack
from sklearn import preprocessing

X_train_avg_w2v = hstack((X_train_cn, avg_w2v_essays_vectors_train, avg_w2v_titles_vectors_train, avg_w2v_summary_vectors_train))
X_train_avg_w2v = X_train_avg_w2v.tocsr()
train_scaler = StandardScaler(with_mean = False)
X_train_avg_w2v = train_scaler.fit_transform(X_train_avg_w2v)
```

In [0]:

```
X_cv_avg_w2v =
hstack((X_cv_cn, avg_w2v_essays_vectors_cv, avg_w2v_titles_vectors_cv, avg_w2v_summary_vectors_cv))
X_cv_avg_w2v = X_cv_avg_w2v.tocsr()
```

```
cv_scalar = StandardScaler(with_mean = False)
X_cv_avg_w2v = cv_scalar.fit_transform(X_cv_avg_w2v)
```

In [0]:

```
X_test_avg_w2v
=hstack((X_test_cn,avg_w2v_essays_vectors_test,avg_w2v_titles_vectors_test,avg_w2v_summary_vectors_test))
X_test_avg_w2v = X_test_avg_w2v.tocsr()
test_scalar = StandardScaler(with_mean = False)
X_test_avg_w2v = test_scalar.fit_transform(X_test_avg_w2v)
```

## 4.2-AUC with trainset and CV-set using Dataset after CV-splitting

In [179]:

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt

train_auc = []
cv_auc = []
alpha_values =[0.0001,0.001,0.01,0.05,0.1,0.3,0.5,0.7,1,10,50,100,150,200,500,1000,2000,3000,5000,10000]

for i in tqdm(alpha_values):
    model_avgw2v = SGDClassifier(loss='log',alpha=i)
    model_avgw2v.fit(X_train_avg_w2v, y_train)
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
    # not the predicted outputs

    y_train_pred = []
    for k in range(0,X_train_avg_w2v.shape[0],1000):
        y_train_pred.extend(model_avgw2v.predict_proba(X_train_avg_w2v[k:k+1000])[:,1])

    y_cv_pred = []

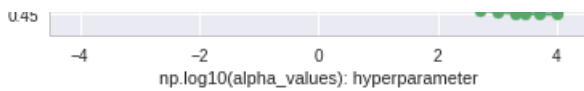
    for k in range(0, X_cv_avg_w2v.shape[0],1000):
        y_cv_pred.extend(model_avgw2v.predict_proba(X_cv_avg_w2v[k:k+1000])[:,1])

    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

plt.plot(np.log10(alpha_values), train_auc, label='Train AUC')
plt.plot(np.log10(alpha_values), cv_auc, label='CV AUC')
plt.scatter(np.log10(alpha_values), train_auc, label='Train AUC points')
plt.scatter(np.log10(alpha_values), cv_auc, label='CV AUC points')
plt.legend()
plt.xlabel("np.log10(alpha_values): hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

100% |██████████| 20/20 [00:28<00:00, 1.42s/it]





### 4.3-ROC-Curve with optimal\_alpha for train and test-sets

In [180]:

```
from sklearn.metrics import roc_curve, auc

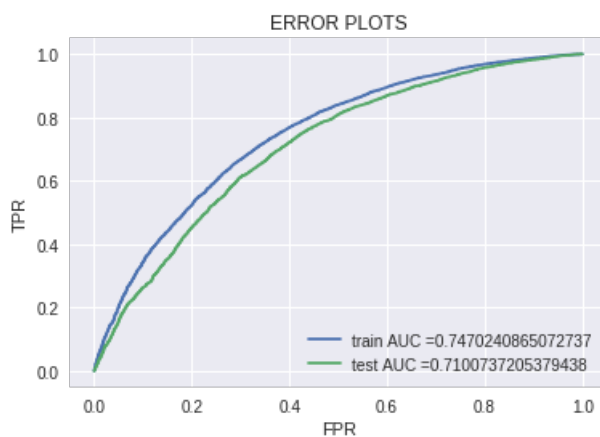
optimal_alpha = 0.1
model_avgw2v = SGDClassifier(loss='log', alpha=optimal_alpha)
model_avgw2v.fit(X_train_avg_w2v, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs

y_train_pred = []
for k in range(0, X_train_avg_w2v.shape[0], 1000):
    y_train_pred.extend(model_avgw2v.predict_proba(X_train_avg_w2v[k:k+1000])[:,1])

y_test_pred = []
for k in range(0, X_test_avg_w2v.shape[0], 100):
    y_test_pred.extend(model_avgw2v.predict_proba(X_test_avg_w2v[k:k+100])[:,1])

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

plt.plot(train_fpr, train_tpr, label="train AUC =" + str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC =" + str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ERROR PLOTS")
plt.show()
```



In [181]:

```
# we are writing our own function for predict, with defined threshold
# we will pick a threshold that will give the least fpr
def predict(proba, threshold, fpr, tpr):

    t = threshold[np.argmax(tpr*(1-fpr))]

    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high

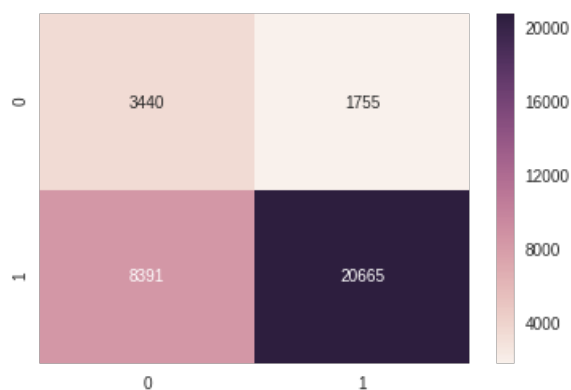
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
    predictions = []
    for i in proba:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

```
import seaborn as sns
from sklearn.metrics import confusion_matrix
print("train Confusion Matrix")
cm_train=confusion_matrix(y_train,predict(y_train_pred, tr_thresholds, train_fpr, train_tpr))
sns.heatmap(cm_train,annot=True,fmt='.5g')
```

train Confusion Matrix  
the maximum value of  $tpr \cdot (1-fpr)$  0.47094747575820534 for threshold 0.852

Out[181]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x7f8c0e393b38>



In [182]:

```
y_train.value_counts()
```

Out[182]:

```
1    29056
0     5195
Name: project_is_approved, dtype: int64
```

In [183]:

```
y_test.value_counts()
```

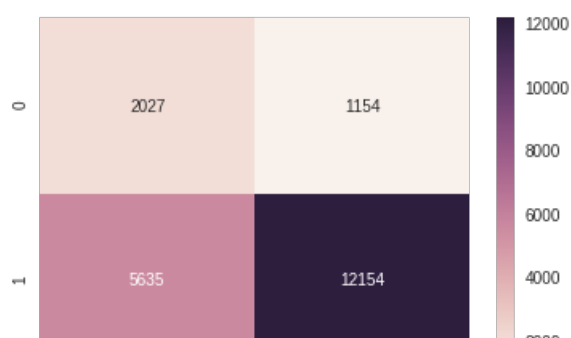
Out[183]:

```
1    17789
0     3181
Name: project_is_approved, dtype: int64
```

In [184]:

```
print("Test confusion matrix")
cm=confusion_matrix(y_test, predict(y_test_pred, te_thresholds, test_fpr, test_tpr))
print(sns.heatmap(cm,annot=True,fmt='.5g'))
```

Test confusion matrix  
the maximum value of  $tpr \cdot (1-fpr)$  0.4353692748428349 for threshold 0.862  
AxesSubplot(0.125,0.125;0.62x0.755)





## 5-SET-5

X\_cn = contain all the features with the exception of Text-Features-Essays, Titles, Resource\_Summary

In [0]:

```
X_cn =
dataset.drop(['y', 'preprocessed_resource_summary', 'preprocessed_essays', 'preprocessed_titles'], axis=1)
```

In [0]:

```
X_cn = scipy.sparse.csr_matrix(X_cn)
```

In [0]:

```
from scipy.sparse import hstack
from sklearn.preprocessing import StandardScaler

X_cn = X_cn.tocsr()
test_scalar = StandardScaler(with_mean = False)
X_cn = test_scalar.fit_transform(X_cn)
```

### 5.1-Train,test,cv sets of ALL features

In [0]:

```
from sklearn.model_selection import train_test_split
X_1, X_test, y_1, y_test = train_test_split(X_cn[:69900], y[:69900], test_size=0.3, random_state=0, stratify=y[:69900])

# split the train data set into cross validation train and cross validation test
X_train, X_cv, y_train, y_cv = train_test_split(X_1, y_1, test_size=0.3, random_state=0, stratify=y_1)
```

In [0]:

```
# 5.2-AUC with trainset and CV-set using Dataset after CV-splitting
```

In [189]:

```
train_auc = []
cv_auc = []

alpha_values = [0.000001, 0.00001, 0.0001, 0.001, 0.01, 0.04]

for i in tqdm(alpha_values):

    model_Logistic = SGDClassifier(loss='log', alpha=i)
    model_Logistic.fit(X_train, y_train)
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
    # not the predicted outputs

    y_train_pred = []
    for k in range(0, X_train.shape[0], 1000):

        y_train_pred.extend(model_Logistic.predict_proba(X_train[k:k+1000])[:, 1])

    y_cv_pred = []
```

```

for k in range(0, X_cv.shape[0],500):

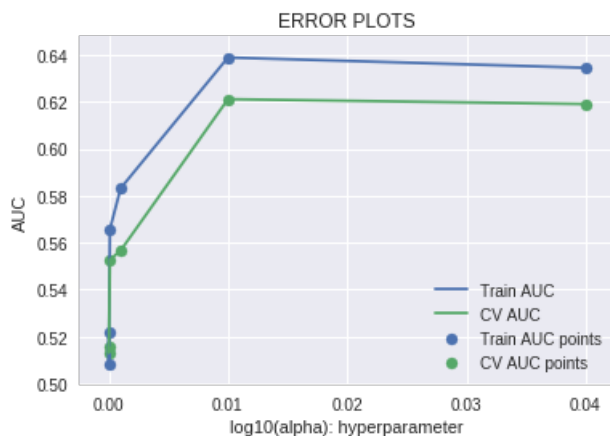
    y_cv_pred.extend(model_Logistic.predict_proba(X_cv[k:k+500])[:,1])

train_auc.append(roc_auc_score(y_train,y_train_pred))
cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

plt.plot(alpha_values, train_auc, label='Train AUC')
plt.plot(alpha_values, cv_auc, label='CV AUC')
plt.scatter(alpha_values, train_auc, label='Train AUC points')
plt.scatter(alpha_values, cv_auc, label='CV AUC points')
plt.legend()
plt.xlabel("log10(alpha): hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()

```

100%|██████████| 6/6 [00:00<00:00, 9.20it/s]



### 5.3-ROC-Curve with optimal\_alpha for train and test-sets

In [190]:

```

from sklearn.metrics import roc_curve, auc

optimal_alpha = 0.01
model_Logistic_bow = SGDClassifier(loss='log',alpha=optimal_alpha)
model_Logistic_bow.fit(X_train, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs

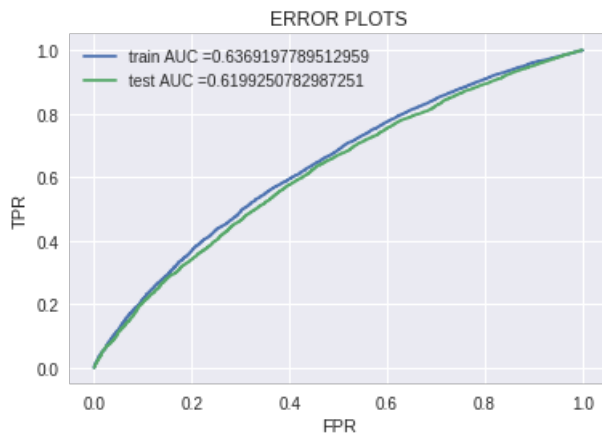
y_train_pred = []
for k in range(0,X_train.shape[0],100):
    y_train_pred.extend(model_Logistic_bow.predict_proba(X_train[k:k+100])[:,1])

y_test_pred = []
for k in range(0, X_test.shape[0],100):
    y_test_pred.extend(model_Logistic_bow.predict_proba(X_test[k:k+100])[:,1])

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

plt.plot(train_fpr, train_tpr, label="train AUC =" +str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC =" +str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ERROR PLOTS")
plt.show()

```



In [191]:

```
# we are writing our own function for predict, with defined threshold
# we will pick a threshold that will give the least fpr
def predict(proba, threshold, fpr, tpr):

    t = threshold[np.argmax(tpr*(1-fpr))]

    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high

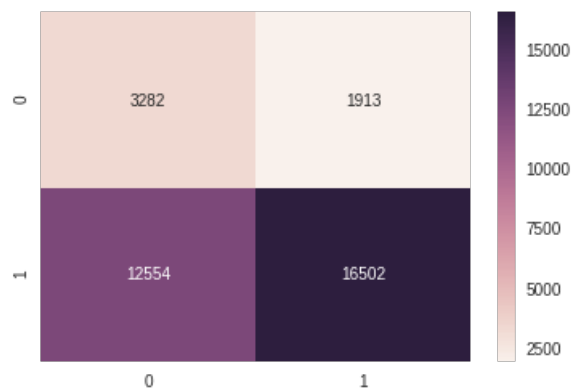
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
    predictions = []
    for i in proba:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions

import seaborn as sns
from sklearn.metrics import confusion_matrix
print("train Confusion Matrix")
cm_train=confusion_matrix(y_train,predict( y_train_pred, tr_thresholds, train_fpr, train_tpr))
sns.heatmap(cm_train,annot=True,fmt='.5g')
```

train Confusion Matrix  
the maximum value of tpr\*(1-fpr) 0.3588011123454016 for threshold 0.852

Out[191]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x7f8c22559c18>



In [192]:

```
print("Test confusion matrix")
cm=confusion_matrix(y_test, predict(y_test_pred, te_thresholds, test_fpr, test_tpr))
print(sns.heatmap(cm,annot=True,fmt='.5g'))
```

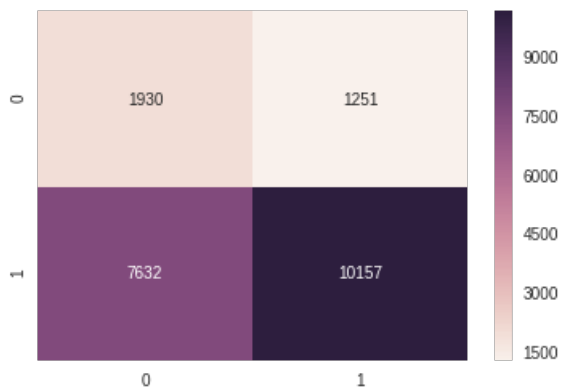
Test confusion matrix



test confusion matrix

the maximum value of  $tpr \cdot (1 - fpr)$  0.34642366916289624 for threshold 0.851

AxesSubplot(0.125,0.125;0.62x0.755)



## Feature-Importance

In [0]:

```
#https://stackoverflow.com/questions/11116697/how-to-get-most-informative-features-for-scikit-learn-classifiers
```

```
top_20_features=sorted(zip(model_Logistic_bow.coef_[0],X_cat_num),reverse=True)[:20]
```

In [194]:

```
top_20_features
```

Out[194]:

```
[(0.30311944413091557, 'Mrs'),  
 (0.24936303122283737, 'Grades 3-5'),  
 (0.23578175912388474, 'previously_posted_projects'),  
 (0.22669479486039085, 'Ms'),  
 (0.20127567972457353, 'Grades PreK-2'),  
 (0.20095113381085317, 'essays_words_standardised'),  
 (0.17315112771403357, 'Literacy_Language'),  
 (0.17288533769993883, 'Grades 6-8'),  
 (0.15061091275004085, 'Mr'),  
 (0.13535760169301755, 'CA'),  
 (0.12741957298497172, 'Grades 9-12'),  
 (0.11572996959802499, 'WA'),  
 (0.11154626634016428, 'Literacy'),  
 (0.09610676996961982, 'spellingmistake_resource'),  
 (0.09491836071146344, 'OH'),  
 (0.0909694722255284, 'Teacher'),  
 (0.08792724898726681, 'NY'),  
 (0.0832324102608007, 'KY'),  
 (0.08035419759391782, 'Health_Wellness'),  
 (0.07580234782634575, 'MA')]
```

In [0]:

```
# Pretty-Table
```

In [196]:

```
import numpy as np  
from prettytable import PrettyTable  
x = PrettyTable()  
x.field_names = ["Model", "alpha", "Train-AUC", "Test-AUC"]  
  
x.add_row(["BOW", 1, 0.869, 0.702])  
x.add_row(["TFidf", 1, 0.877, 0.708])  
x.add_row(["TFidf-W2V", 0.1, 0.74, 0.701])  
x.add_row(["AVG-W2V", 0.1, 0.747, 0.713])
```

```
x.add_row(["SET-5",0.01,0.636,0.619])
print(x)
```

Model	alpha	Train-AUC	Test-AUC
BOW	1	0.869	0.702
TFidf	1	0.877	0.708
TFidf-W2V	0.1	0.74	0.701
AVG-W2V	0.1	0.747	0.713
SET-5	0.01	0.636	0.619