DonorsChoose

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

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

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

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

About the DonorsChoose Data Set

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

Feature	Description
`project_id`	A unique identifier for the proposed project. **Example:** `p036502`
`project_title`	Title of the project. **Examples:** • `Art Will Make You Happy!` • `First Grade Fun`
`project_grade_category`	Grade level of students for which the project is targeted. One of the following enumerated values: • `Grades PreK-2` • `Grades 3-5` • `Grades 6-8` • `Grades 9-12`
`project_subject_categories`	One or more (comma-separated) subject categories for the project from the following enumerated list of values: • `Applied Learning` • `Care & Hunger` • `Health & Sports` • `History & Civics` • `Literacy & Language` • `Math & Science` • `Music & The Arts` • `Special Needs` • `Warmth` **Examples:** • `Music & The Arts` • `Literacy & Language, Math & Science`
`school_state`	State where school is located (Two-letter U.S. postal code). **Example:** `WY`
`project_subject_subcategories`	One or more (comma-separated) subject subcategories for the project. **Examples:** • `Literacy`

Feature	Description
	• Literature & Writing, Social Sciences
project_resource_summary	An explanation of the resources needed for the project. **Example:** • `My students need hands on literacy materials to manage sensory needs!`
project_essay_1	First application essay*
project_essay_2	Second application essay*
project_essay_3	Third application essay*
project_essay_4	Fourth application essay*
project_submitted_datetime	Datetime when project application was submitted. **Example:** `2016-04-28 12:43:56.245`
teacher_id	A unique identifier for the teacher of the proposed project. **Example:** `bdf8baa8fedef6bfeec7ae4ff1c15c56`
teacher_prefix	Teacher's title. One of the following enumerated values: • `nan` • `Dr.` • `Mr.` • `Mrs.` • `Ms.` • `Teacher.`
teacher_number_of_previously_posted_projects	Number of project applications previously submitted by the same teacher. **Example:** `2`

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

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

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

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

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

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

Notes on the Essay Data

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

- __project_essay_1:__ "Introduce us to your classroom"
- __project_essay_2:__ "Tell us more about your students"
- __project_essay_3:__ "Describe how your students will use the materials you're requesting"
- __project_essay_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."

- __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 [132]:

```
%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 [134]:

```
project_data.columns
```

Out[134]:

```
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_essay_3', 'project_essay_4', 'project_resource_summary',
       'teacher_number_of_previously_posted_projects', 'project_is_approved'],
      dtype='object')
```

In [135]:

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

Out[135]:

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

In [136]:

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

Out[136]:

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

In [137]:

```
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 [138]:

```
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[138]:

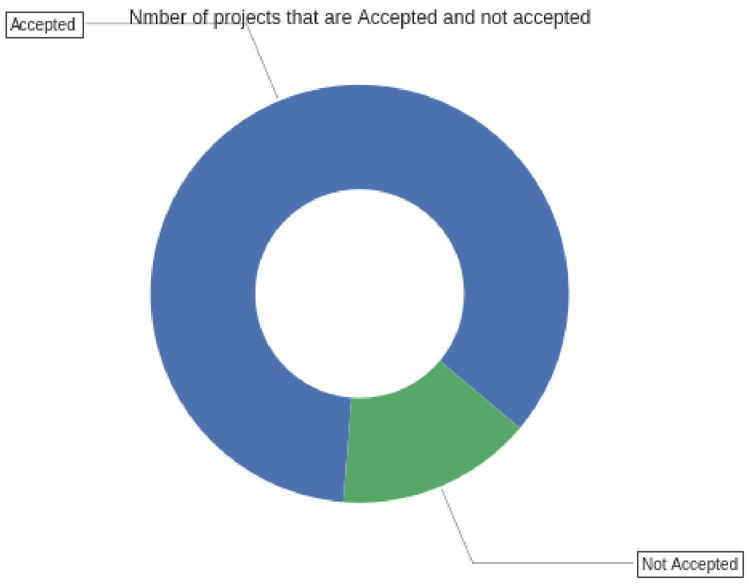
	id	description	quantity	price
0	p233245	LC652 - Lakeshore Double-Space Mobile Drying Rack	1	149.00
1	p069063	Bouncy Bands for Desks (Blue support pipes)	3	14.95

1.2 Data Analysis

In [139]:

```
# 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 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 aprooved 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 [140]:

```
# 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)
```

```
temp.columns = ['state_code', 'num_proposals']

'''# How to plot US state heatmap: https://datascience.stackexchange.com/a/9620

scl = [[0.0, 'rgb(242,240,247)'],[0.2, 'rgb(218,218,235)'],[0.4, 'rgb(188,189,220)'],\
            [0.6, 'rgb(158,154,200)'],[0.8, 'rgb(117,107,177)'],[1.0, 'rgb(84,39,143)']]

data = [ dict(
        type='choropleth',
        colorscale = scl,
        autocolorscale = False,
        locations = temp['state_code'],
        z = temp['num_proposals'].astype(float),
        locationmode = 'USA-states',
        text = temp['state_code'],
        marker = dict(line = dict (color = 'rgb(255,255,255)',width = 2)),
        colorbar = dict(title = "% of pro")
    ) ]

layout = dict(
        title = 'Project Proposals % of Acceptance Rate by US States',
        geo = dict(
            scope='usa',
            projection=dict( type='albers usa' ),
            showlakes = True,
            lakecolor = 'rgb(255, 255, 255)',
        ),
    )

fig = go.Figure(data=data, layout=layout)
offline.iplot(fig, filename='us-map-heat-map')
'''
```

Out[140]:

```
'# How to plot US state heatmap: https://datascience.stackexchange.com/a/9620\n\nscl = [[0.0, \'rg
b(242,240,247)\'],[0.2, \'rgb(218,218,235)\'],[0.4, \'rgb(188,189,220)\'],            [0.6, \'rgb(1
58,154,200)\'],[0.8, \'rgb(117,107,177)\'],[1.0, \'rgb(84,39,143)\']]\n\ndata = [ dict(\n        ty
pe=\'choropleth\',\n        colorscale = scl,\n        autocolorscale = False,\n        locations =
temp[\'state_code\'],\n        z = temp[\'num_proposals\'].astype(float),\n        locationmode = \
'USA-states\',\n        text = temp[\'state_code\'],\n        marker = dict(line = dict (color = \'
rgb(255,255,255)\',width = 2)),\n        colorbar = dict(title = "% of pro")\n    ) ]\n\nlayout = d
ict(\n        title = \'Project Proposals % of Acceptance Rate by US States\',\n        geo = dict(
\n            scope=\'usa\',\n            projection=dict( type=\'albers usa\' ),\n            show
akes = True,\n            lakecolor = \'rgb(255, 255, 255)\',\n        ),\n    )\n\nfig =
go.Figure(data=data, layout=layout)\noffline.iplot(fig, filename=\'us-map-heat-map\')\n'
```

In [141]:

```
# https://www.csi.cuny.edu/sites/default/files/pdf/administration/ops/2letterstabbrev.pdf
temp.sort_values(by=['num_proposals'], inplace=True)
print("States with lowest % approvals")
print(temp.head(5))
print('='*50)
print("States with highest % approvals")
print(temp.tail(5))
```

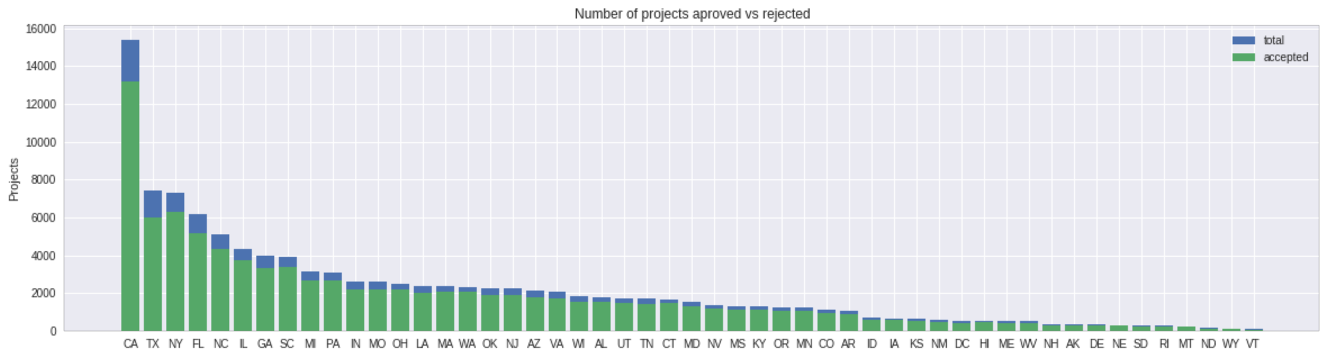
```
States with lowest % approvals
   state_code  num_proposals
46         VT       0.800000
7          DC       0.802326
43         TX       0.813142
26         MT       0.816327
18         LA       0.831245
==================================================
States with highest % approvals
   state_code  num_proposals
30         NH       0.873563
35         OH       0.875152
47         WA       0.876178
28         ND       0.888112
8          DE       0.897959
```

In [0]:

```
#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 [143]:

```
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
==================================================
   school_state  project_is_approved  total       Avg
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 perentage of projects submitted and accepted and rejected from particular state and we sorted the acceptance rate

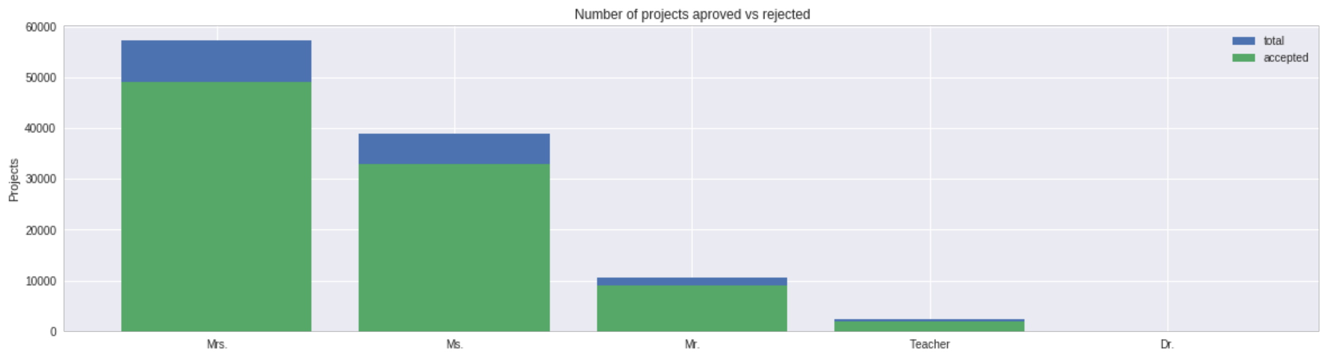
1. projects from CA have been accepted and rejected more

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

1.2.2 Univariate Analysis: teacher_prefix

In [144]:

```
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

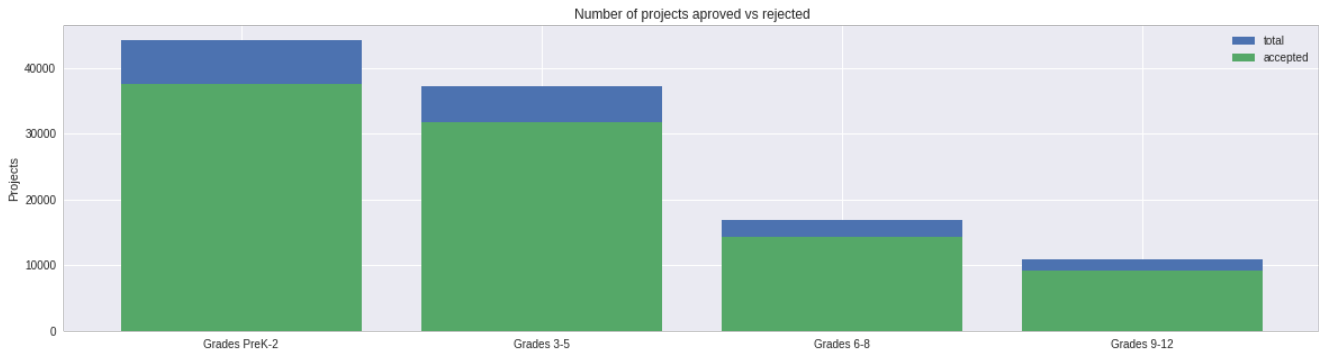
3.projects submitted by teacher of prefix-MRS are accepted more

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

1.2.3 Univariate Analysis: project_grade_category

In [145]:

```
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]:

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

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

In [147]:

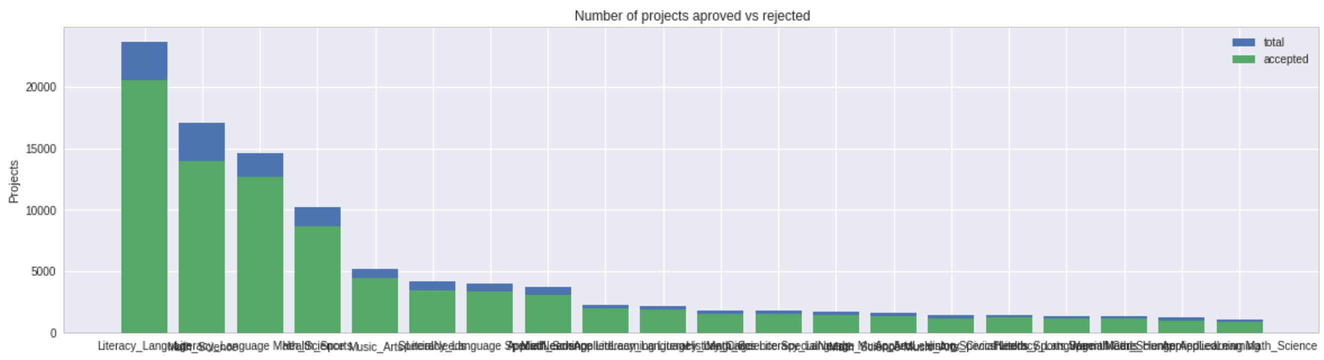
```
project_data['clean_categories'] = cat_list
project_data.drop(['project_subject_categories'], axis=1, inplace=True)
project_data.head(2)
```

Out[147]:

	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 [148]:

```
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
```

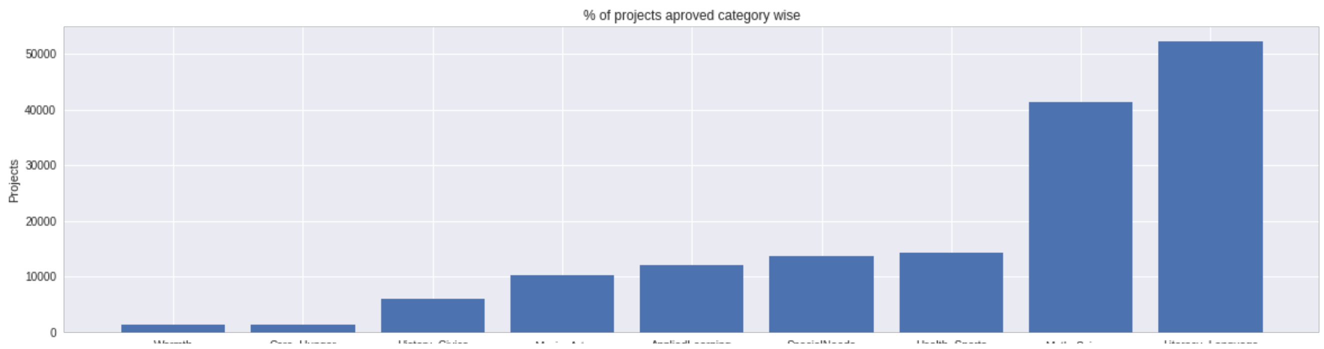
1. projects from literacy_language subcategories have been acceptd more
2. we calculated projects submitted of a specific sub_category
3. We calculated what are the total no.of.projets submitted from specific catgory and how many are accepted and rejected from that category

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 [150]:

```
# 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 categoryand we do plot how many are rejected and how many are submitted

we can say that literacy_language projects are more accepted

In [151]:

```python
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]:

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

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

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

In [153]:

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

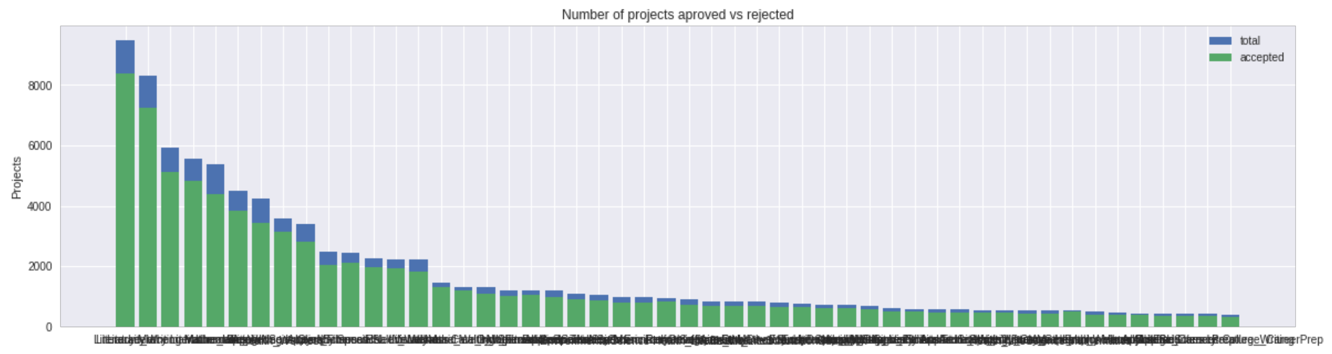
Out[153]:

	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

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime	pro

In [154]:

```
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.

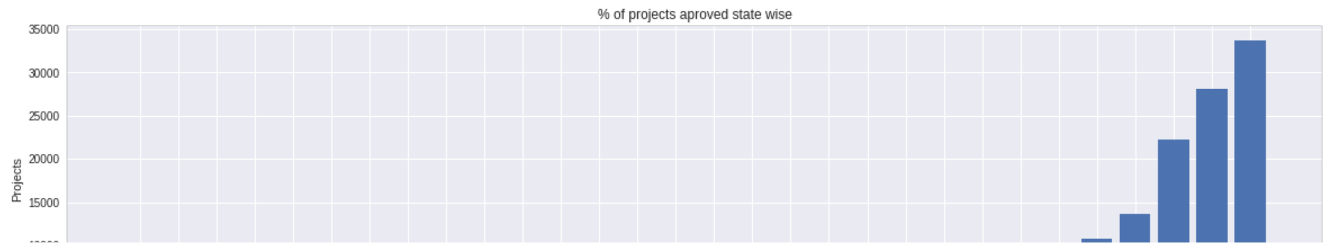
2.literacy sub_category are submitted more and are accepted more

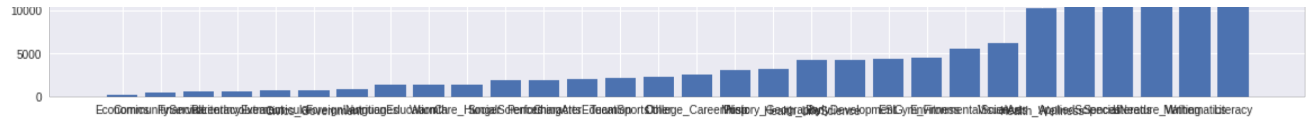
In [0]:

```
# 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 [156]:

```
# 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 [157]:

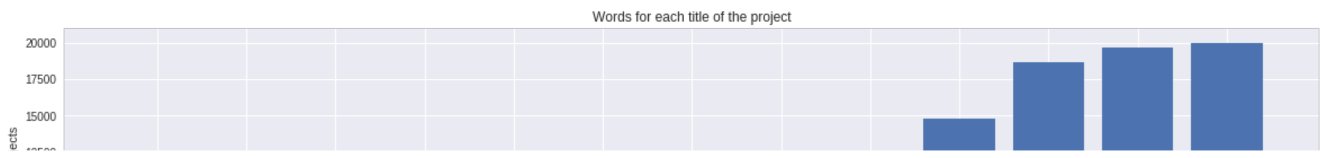
```
for i, j in sorted_sub_cat_dict.items():
    print("{:20} :{:10}".format(i,j))
```

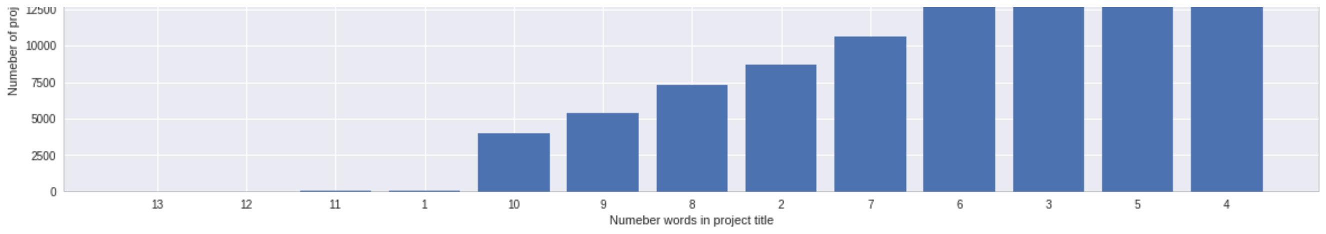
```
Economics            :       269
CommunityService     :       441
FinancialLiteracy    :       568
ParentInvolvement    :       677
Extracurricular      :       810
Civics_Government    :       815
ForeignLanguages     :       890
NutritionEducation   :      1355
Warmth               :      1388
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 [158]:

```
#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 oon 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
```

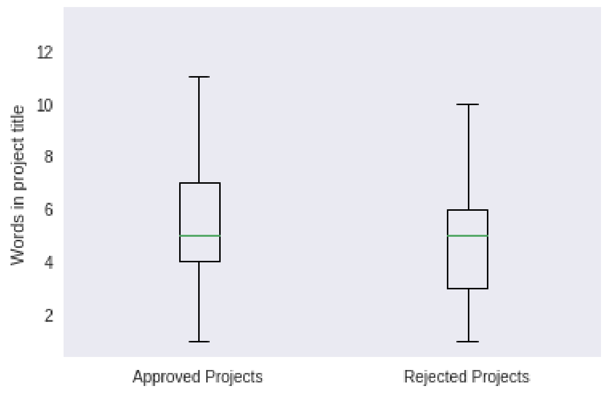
In [160]:

```
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 [161]:

```
# 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 fo approved projects and rejected projects under project_title category

we found few outliers for approved projects and rejected projects

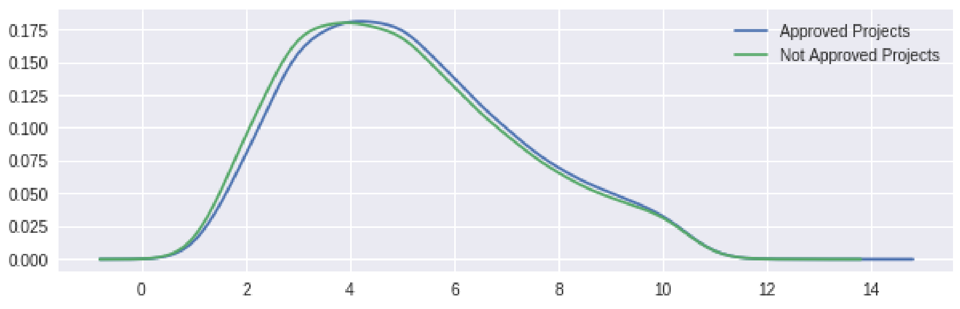
we found the Iqr range for approved projects and rejected projects and found the range for approved projects under project_title category is High and the for rejected projects is low.

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

In [162]:

```
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.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 calcuated 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 whther the project is accepted or rejected.

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

1.2.7 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 [165]:

```
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 [166]:

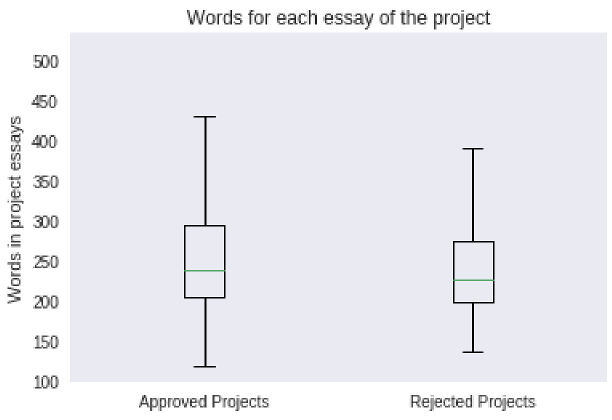
```
word_count_resource_summary = project_data['project_resource_summary'].str.split().apply(len)
word_count_resource_summary = word_count_resource_summary.values
print(word_count_resource_summary)
```

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

In [167]:

```
# 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

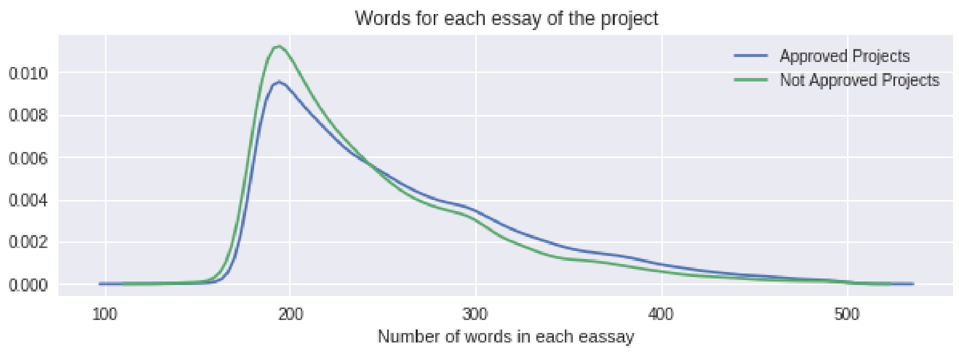
The numerical data is whether the project is approved and rejected based on no.of.words present in the essay.

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 [168]:

```
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 calcuated 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 [169]:

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

Out[169]:

	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 [170]:

```
# 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[170]:

	id	price	quantity
0	p000001	459.56	7
1	p000002	515.89	21

In [0]:

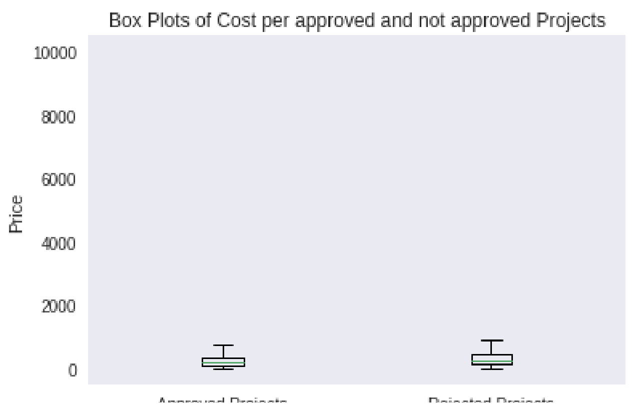
```
# join two dataframes in python:
project_data = pd.merge(project_data, price_data, on='id', how='left')
```

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 [173]:

```
# 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()
```



Approved Projects Rejected Projects

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

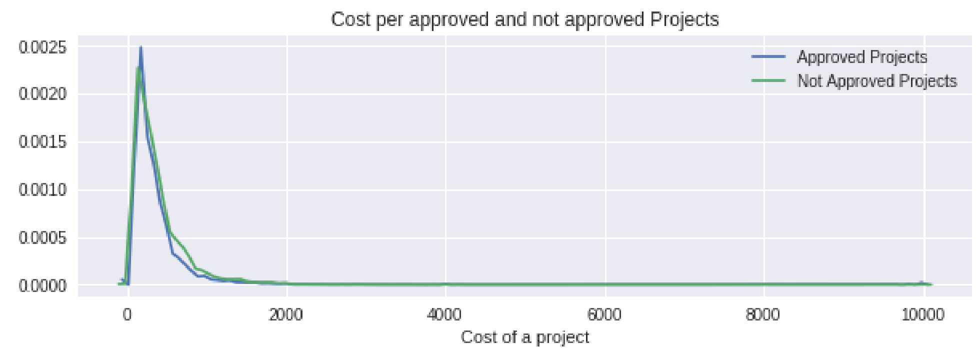
The numerical data is whether the project is approved and rejected based on the price of project

there too many outliers for approved and rejected projects data based on the price of project and the IQR 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 based on the price of project.

In [174]:

```
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 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 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 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

In [175]:

```
# 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_pric
e,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

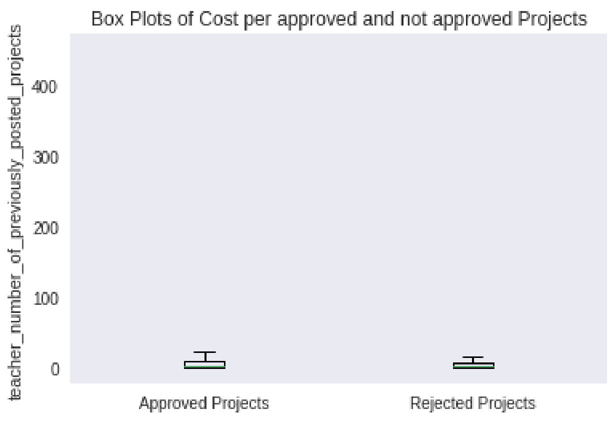
1.2.9 Univariate Analysis: teacher_number_of_previously_posted_projects

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 [177]:

```
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 [178]:

```
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         |
+------------+-------------------+-----------------------+
```

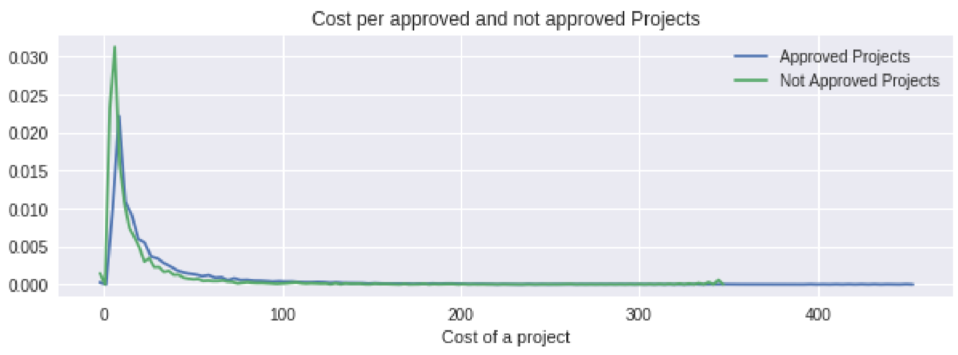
Here we created a table and calculated the percentiles for the approved and rejected data No.of.previously posted projects by a teacher

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 [179]:

```
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 Lex
ia 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	897464ce9ddc600bced1151f324dd63a	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	f3cb9bffbba169bef1a77b243e620b60	Mrs.	KY	2016-10-06 21:16:17	Gra

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime	pro
4	172407	p104768	be1f7507a41f8479dc06f047086a39ec	Mrs.	TX	2016-07-11 01:10:09	Gra

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 Pro
jects')
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 [180]:

```
project_data.shape
```

Out[180]:

```
(109248, 20)
```

In [181]:

```
# 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. W
e are a melting pot of refugees, immigrants, and native-born Americans bringing the gift of langua
ge to our school. \r\n\r\n We have over 24 languages represented in our English Learner program wi
th 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 o
f your world.\"-Ludwig Wittgenstein  Our English learner's have a strong support system at home th
at begs for more resources.  Many times our parents are learning to read and speak English along s
ide 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 hom
e is able to assist.  All families with students within the Level 1 proficiency status, will be a
offered to be a part of this program.  These educational videos will be specially chosen by the En
glish 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 ed
ucational dvd's for the years to come for other EL students.\r\nnannan
==================================================
The 51 fifth grade students that will cycle through my classroom this year all love learning, at l
east most of the time. At our school, 97.3% of the students receive free or reduced price lunch. O
f 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 bea
utiful 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 t
he hard work put in during the school year, with a dunk tank being the most popular activity.My st
udents 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 hav
e 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 us
ed 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 i
n 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 ta
ken. 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 th
e 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 a
nd reduced-price lunch to qualify. Our school is an \"open classroom\" concept, which is very uniq
ue as there are no walls separating the classrooms. These 9 and 10 year-old students are very eage
r learners; they are like sponges, absorbing all the information and experiences and keep on wanti
ng more.With these resources such as the comfy red throw pillows and the whimsical nautical hangin
g 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 pic
tures 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 lost of money out of
my own pocket on resources to get our classroom ready. Please consider helping with this project t
o make our new school year a very successful one. Thank you!nannan
==================================================
My kindergarten students have varied disabilities ranging from speech and language delays, cogniti
ve delays, gross/fine motor delays, to autism. They are eager beavers and always strive to work th
eir 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 pr
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 gr
oove and move as you were in a meeting? This is how my kids feel all the time. The want to be able
to move as they learn or so they say.Wobble chairs are the answer and I love then because they dev
elop their core, which enhances gross motor and in Turn fine motor skills. \r\nThey also want to l
earn through games, my kids don't want to sit and do worksheets. They want to learn to count by ju
mping and playing. Physical engagement is the key to our success. The number toss and color and sh
ape 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]:

```python
# 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 [183]:

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

```
My kindergarten students have varied disabilities ranging from speech and language delays, cogniti
ve delays, gross/fine motor delays, to autism. They are eager beavers and always strive to work th
eir 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 pr
```

```
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 gr
oove and move as you were in a meeting? This is how my kids feel all the time. The want to be able
to move as they learn or so they say.Wobble chairs are the answer and I love then because they dev
elop their core, which enhances gross motor and in Turn fine motor skills. \r\nThey also want to l
earn through games, my kids do not want to sit and do worksheets. They want to learn to count by j
umping and playing. Physical engagement is the key to our success. The number toss and color and s
hape 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 [184]:

```python
# \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', ' ')
print(sent)
```

```
My kindergarten students have varied disabilities ranging from speech and language delays, cogniti
ve delays, gross/fine motor delays, to autism. They are eager beavers and always strive to work th
eir 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 eag
er to learn and explore.Have you ever felt like you had ants in your pants and you needed to groov
e and move as you were in a meeting? This is how my kids feel all the time. The want to be able to
move as they learn or so they say.Wobble chairs are the answer and I love then because they develo
p their core, which enhances gross motor and in Turn fine motor skills.   They also want to learn t
hrough 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 ma
ts can make that happen. My students will forget they are doing work and just have the fun a 6 yea
r old deserves.nannan
```

In [185]:

```python
#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 cognitiv
e delays gross fine motor delays to autism They are eager beavers and always strive to work their
hardest working past their limitations The materials we have are the ones I seek out for my studen
ts 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 lea
rn and explore Have you ever felt like you had ants in your pants and you needed to groove and mov
e as you were in a meeting This is how my kids feel all the time The want to be able to move as th
ey learn or so they say Wobble chairs are the answer and I love then because they develop their co
re 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 Ph
ysical 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 nan
nan
```

In [0]:

```python
# 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', "do
esn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn',
"mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn',
"wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't","nan"]
```

In [187]:

```python
# Combining all the above statemennts
from tqdm import tqdm
preprocessed_essays = []
# tqdm is for printing the status bar
for sentance in tqdm(project_data['essay'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # 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())
```

```
  0%|          | 0/109248 [00:00<?, ?it/s]

  0%|          | 74/109248 [00:00<02:28, 737.10it/s]

  0%|          | 222/109248 [00:00<02:05, 866.78it/s]

  0%|          | 374/109248 [00:00<01:49, 994.49it/s]

  0%|          | 526/109248 [00:00<01:37, 1109.51it/s]

  1%|          | 664/109248 [00:00<01:32, 1177.20it/s]

  1%|          | 812/109248 [00:00<01:26, 1253.00it/s]

  1%|          | 950/109248 [00:00<01:24, 1288.07it/s]

  1%|          | 1098/109248 [00:00<01:20, 1339.99it/s]

  1%|          | 1248/109248 [00:00<01:18, 1382.62it/s]

  1%|▏         | 1404/109248 [00:01<01:15, 1429.81it/s]

  1%|▏         | 1556/109248 [00:01<01:14, 1452.58it/s]

  2%|▏         | 1707/109248 [00:01<01:13, 1468.40it/s]

  2%|▏         | 1858/109248 [00:01<01:12, 1480.61it/s]

  2%|▏         | 2012/109248 [00:01<01:11, 1497.38it/s]

  2%|▏         | 2173/109248 [00:01<01:10, 1527.98it/s]

  2%|▏         | 2327/109248 [00:01<01:09, 1530.16it/s]

  2%|▏         | 2481/109248 [00:01<01:12, 1477.04it/s]

  2%|▏         | 2631/109248 [00:01<01:11, 1482.89it/s]

  3%|▎         | 2780/109248 [00:01<01:11, 1483.44it/s]

  3%|▎         | 2939/109248 [00:02<01:10, 1511.61it/s]

  3%|▎         | 3098/109248 [00:02<01:09, 1532.80it/s]

  3%|▎         | 3252/109248 [00:02<01:09, 1523.31it/s]
  3%|▎         | 3406/109248 [00:02<01:09, 1526.54it/s]
  3%|▎         | 3559/109248 [00:02<01:09, 1523.88it/s]
  3%|▎         | 3713/109248 [00:02<01:09, 1528.30it/s]
  4%|▎         | 3866/109248 [00:02<01:10, 1492.75it/s]
  4%|▎         | 4016/109248 [00:02<01:11, 1472.92it/s]
  4%|▍         | 4174/109248 [00:02<01:09, 1501.84it/s]
  4%|▍         | 4325/109248 [00:02<01:11, 1473.01it/s]
  4%|▍         | 4487/109248 [00:03<01:09, 1512.40it/s]
  4%|▍         | 4639/109248 [00:03<01:09, 1507.52it/s]
  4%|▍         | 4793/109248 [00:03<01:08, 1516.45it/s]
  5%|▍         | 4950/109248 [00:03<01:08, 1530.67it/s]
  5%|▍         | 5104/109248 [00:03<01:08, 1522.58it/s]
  5%|▍         | 5259/109248 [00:03<01:08, 1529.18it/s]
  5%|▍         | 5419/109248 [00:03<01:07, 1548.60it/s]
  5%|▌         | 5575/109248 [00:03<01:10, 1472.99it/s]
  5%|▌         | 5728/109248 [00:03<01:09, 1489.39it/s]
  5%|▌         | 5881/109248 [00:03<01:08, 1499.76it/s]
  6%|▌         | 6042/109248 [00:04<01:07, 1530.60it/s]
  6%|▌         | 6198/109248 [00:04<01:06, 1538.91it/s]
  6%|▌         | 6355/109248 [00:04<01:06, 1547.67it/s]
  6%|▌         | 6511/109248 [00:04<01:06, 1542.32it/s]
  6%|▌         | 6669/109248 [00:04<01:06, 1551.98it/s]
  6%|▌         | 6825/109248 [00:04<01:06, 1534.88it/s]
  6%|▋         | 6985/109248 [00:04<01:05, 1551.35it/s]
  7%|▋         | 7141/109248 [00:04<01:08, 1493.83it/s]
  7%|▋         | 7291/109248 [00:04<01:10, 1454.90it/s]
  7%|▋         | 7438/109248 [00:04<01:10, 1453.79it/s]
  7%|▋         | 7597/109248 [00:05<01:08, 1491.73it/s]
  7%|▋         | 7758/109248 [00:05<01:06, 1524.13it/s]
  7%|▋         | 7913/109248 [00:05<01:06, 1528.31it/s]
  7%|▋         | 8073/109248 [00:05<01:05, 1547.30it/s]
  8%|▊         | 8229/109248 [00:05<01:05, 1533.03it/s]
  8%|▊         | 8390/109248 [00:05<01:04, 1554.40it/s]
  8%|▊         | 8555/109248 [00:05<01:03, 1579.28it/s]
  8%|▊         | 8714/109248 [00:05<01:05, 1543.26it/s]
  8%|▊         | 8869/109248 [00:05<01:05, 1542.10it/s]
  8%|▊         | 9032/109248 [00:06<01:03, 1566.31it/s]
  8%|▊         | 9189/109248 [00:06<01:04, 1556.22it/s]
  9%|▊         | 9354/109248 [00:06<01:03, 1582.65it/s]
  9%|▉         | 9354/109248 [00:06<01:03, 1582.65it/s]
  9%|▉         | 9519/109248 [00:06<01:02, 1598.77it/s]
  9%|▉         | 9680/109248 [00:06<01:02, 1584.11it/s]
  9%|▉         | 9839/109248 [00:06<01:04, 1548.94it/s]
  9%|▉         | 9995/109248 [00:06<01:04, 1537.04it/s]
  9%|▉         | 10150/109248 [00:06<01:04, 1539.70it/s]
  9%|▉         | 10305/109248 [00:06<01:06, 1489.03it/s]
 10%|▉         | 10460/109248 [00:06<01:05, 1502.68it/s]
 10%|▉         | 10612/109248 [00:07<01:05, 1506.79it/s]
 10%|▉         | 10772/109248 [00:07<01:04, 1533.00it/s]
 10%|█         | 10929/109248 [00:07<01:03, 1543.51it/s]
 10%|█         | 11090/109248 [00:07<01:02, 1558.58it/s]
 10%|█         | 11247/109248 [00:07<01:04, 1518.07it/s]
 10%|█         | 11400/109248 [00:07<01:04, 1516.84it/s]
 11%|█         | 11563/109248 [00:07<01:03, 1547.62it/s]
 11%|█         | 11722/109248 [00:07<01:02, 1559.48it/s]
 11%|█         | 11879/109248 [00:07<01:03, 1532.47it/s]
 11%|█         | 12037/109248 [00:07<01:02, 1545.99it/s]
 11%|█         | 12198/109248 [00:08<01:02, 1563.80it/s]
 11%|█▏        | 12362/109248 [00:08<01:01, 1583.22it/s]
 11%|█▏        | 12530/109248 [00:08<01:00, 1609.87it/s]
 12%|█▏        | 12696/109248 [00:08<00:59, 1624.11it/s]
 12%|█▏        | 12859/109248 [00:08<01:00, 1596.95it/s]
 12%|█▏        | 13019/109248 [00:08<01:00, 1579.44it/s]
 12%|█▏        | 13178/109248 [00:08<01:01, 1567.96it/s]
 12%|█▏        | 13342/109248 [00:08<01:00, 1587.69it/s]
 12%|█▏        | 13501/109248 [00:08<01:02, 1532.01it/s]
 12%|█▏        | 13656/109248 [00:08<01:02, 1535.77it/s]
 13%|█▎        | 13815/109248 [00:09<01:01, 1551.06it/s]
 13%|█▎        | 13972/109248 [00:09<01:01, 1554.93it/s]
 13%|█▎        | 14128/109248 [00:09<01:01, 1547.58it/s]
 13%|█▎        | 14291/109248 [00:09<01:00, 1571.25it/s]
 13%|█▎        | 14449/109248 [00:09<01:00, 1565.75it/s]
 13%|█▎        | 14612/109248 [00:09<00:59, 1583.79it/s]
 14%|█▎        | 14771/109248 [00:09<00:59, 1583.50it/s]
 14%|█▎        | 14930/109248 [00:09<00:59, 1572.85it/s]
 14%|█▍        | 15088/109248 [00:09<01:01, 1540.25it/s]
 14%|█▍        | 15246/109248 [00:10<01:00, 1551.72it/s]
 14%|█▍        | 15407/109248 [00:10<00:59, 1568.67it/s]
```

```
14%|█▍        | 15565/109248 [00:10<00:59, 1567.25it/s]
14%|█▍        | 15728/109248 [00:10<00:59, 1584.33it/s]
15%|█▍        | 15889/109248 [00:10<00:58, 1589.36it/s]
15%|█▍        | 16055/109248 [00:10<00:57, 1609.32it/s]
15%|█▍        | 16217/109248 [00:10<00:57, 1610.02it/s]
15%|█▍        | 16379/109248 [00:10<00:59, 1568.78it/s]
15%|█▌        | 16537/109248 [00:10<00:59, 1550.24it/s]
15%|█▌        | 16693/109248 [00:10<01:02, 1489.06it/s]
15%|█▌        | 16843/109248 [00:11<01:02, 1476.61it/s]
16%|█▌        | 16996/109248 [00:11<01:01, 1490.44it/s]
16%|█▌        | 17146/109248 [00:11<01:02, 1485.19it/s]
16%|█▌        | 17295/109248 [00:11<01:02, 1479.63it/s]
16%|█▌        | 17444/109248 [00:11<01:02, 1460.11it/s]
16%|█▌        | 17591/109248 [00:11<01:02, 1461.82it/s]
16%|█▌        | 17743/109248 [00:11<01:01, 1478.61it/s]
16%|█▋        | 17897/109248 [00:11<01:01, 1495.86it/s]
17%|█▋        | 18051/109248 [00:11<01:00, 1508.08it/s]
17%|█▋        | 18202/109248 [00:11<01:01, 1489.41it/s]
17%|█▋        | 18357/109248 [00:12<01:00, 1506.60it/s]
17%|█▋        | 18517/109248 [00:12<00:59, 1532.46it/s]
17%|█▋        | 18671/109248 [00:12<00:59, 1532.96it/s]
17%|█▋        | 18831/109248 [00:12<00:58, 1549.77it/s]
17%|█▋        | 18992/109248 [00:12<00:57, 1567.09it/s]
18%|█▊        | 19153/109248 [00:12<00:57, 1576.97it/s]
18%|█▊        | 19312/109248 [00:12<00:56, 1578.53it/s]
18%|█▊        | 19470/109248 [00:12<00:56, 1576.72it/s]
18%|█▊        | 19628/109248 [00:12<00:56, 1573.59it/s]
18%|█▊        | 19786/109248 [00:12<00:58, 1525.60it/s]
18%|█▊        | 19953/109248 [00:13<00:57, 1565.16it/s]
18%|█▊        | 20118/109248 [00:13<00:56, 1588.83it/s]
19%|█▊        | 20287/109248 [00:13<00:55, 1617.24it/s]
19%|█▊        | 20450/109248 [00:13<00:55, 1612.06it/s]
19%|█▉        | 20612/109248 [00:13<00:55, 1607.61it/s]
19%|█▉        | 20773/109248 [00:13<00:55, 1607.61it/s]
19%|█▉        | 20939/109248 [00:13<00:54, 1621.88it/s]
19%|█▉        | 21102/109248 [00:13<00:54, 1609.50it/s]
19%|█▉        | 21264/109248 [00:13<00:55, 1584.45it/s]
20%|█▉        | 21423/109248 [00:13<00:57, 1535.82it/s]
```

```
20%|█▉        | 21579/109248 [00:14<00:56, 1541.47it/s]
20%|█▉        | 21736/109248 [00:14<00:56, 1549.24it/s]
20%|██        | 21892/109248 [00:14<00:56, 1551.52it/s]
20%|██        | 22056/109248 [00:14<00:55, 1576.85it/s]
20%|██        | 22214/109248 [00:14<00:56, 1549.44it/s]
20%|██        | 22370/109248 [00:14<00:56, 1532.10it/s]
21%|██        | 22526/109248 [00:14<00:56, 1539.78it/s]
21%|██        | 22686/109248 [00:14<00:55, 1556.40it/s]
21%|██        | 22842/109248 [00:14<00:56, 1534.07it/s]
21%|██        | 22996/109248 [00:15<00:58, 1475.96it/s]
21%|██        | 23151/109248 [00:15<00:57, 1497.13it/s]
21%|██▏       | 23316/109248 [00:15<00:55, 1537.68it/s]
21%|██▏       | 23474/109248 [00:15<00:55, 1548.81it/s]
22%|██▏       | 23630/109248 [00:15<00:56, 1522.23it/s]
22%|██▏       | 23785/109248 [00:15<00:55, 1529.70it/s]
22%|██▏       | 23945/109248 [00:15<00:55, 1548.72it/s]
22%|██▏       | 24101/109248 [00:15<00:55, 1546.67it/s]
22%|██▏       | 24256/109248 [00:15<00:55, 1544.30it/s]
22%|██▏       | 24411/109248 [00:15<00:55, 1539.43it/s]
22%|██▏       | 24566/109248 [00:16<00:56, 1510.60it/s]
23%|██▎       | 24719/109248 [00:16<00:55, 1515.55it/s]
23%|██▎       | 24878/109248 [00:16<00:54, 1536.72it/s]
23%|██▎       | 25032/109248 [00:16<00:57, 1471.24it/s]
23%|██▎       | 25183/109248 [00:16<00:56, 1481.90it/s]
23%|██▎       | 25340/109248 [00:16<00:55, 1506.84it/s]
23%|██▎       | 25501/109248 [00:16<00:54, 1535.79it/s]
23%|██▎       | 25660/109248 [00:16<00:53, 1550.67it/s]
24%|██▎       | 25816/109248 [00:16<00:54, 1544.17it/s]
24%|██▍       | 25971/109248 [00:16<00:53, 1544.71it/s]
24%|██▍       | 26126/109248 [00:17<00:58, 1422.59it/s]
24%|██▍       | 26287/109248 [00:17<00:56, 1474.05it/s]
24%|██▍       | 26442/109248 [00:17<00:55, 1495.43it/s]
24%|██▍       | 26598/109248 [00:17<00:54, 1513.01it/s]
24%|██▍       | 26751/109248 [00:17<00:54, 1512.36it/s]
25%|██▍       | 26903/109248 [00:17<00:54, 1509.23it/s]
25%|██▍       | 27055/109248 [00:17<00:54, 1498.08it/s]
25%|██▍       | 27207/109248 [00:17<00:54, 1500.19it/s]
25%|██▌       | 27359/109248 [00:17<00:54, 1504.55it/s]
25%|██▌       | 27520/109248 [00:18<00:53, 1531.38it/s]
```

```
 25%|██▌       | 27674/109248 [00:18<00:55, 1469.44it/s]
 25%|██▌       | 27836/109248 [00:18<00:53, 1510.31it/s]
 26%|██▌       | 27989/109248 [00:18<00:53, 1515.62it/s]
 26%|██▌       | 28153/109248 [00:18<00:52, 1549.89it/s]
 26%|██▌       | 28313/109248 [00:18<00:51, 1564.53it/s]
 26%|██▌       | 28477/109248 [00:18<00:50, 1584.38it/s]
 26%|██▌       | 28636/109248 [00:18<00:50, 1581.89it/s]
 26%|██▋       | 28795/109248 [00:18<00:51, 1568.69it/s]
 27%|██▋       | 28953/109248 [00:18<00:51, 1556.84it/s]
 27%|██▋       | 29116/109248 [00:19<00:50, 1576.50it/s]
 27%|██▋       | 29274/109248 [00:19<00:52, 1517.73it/s]
 27%|██▋       | 29431/109248 [00:19<00:52, 1531.63it/s]
 27%|██▋       | 29598/109248 [00:19<00:50, 1567.98it/s]
 27%|██▋       | 29756/109248 [00:19<00:51, 1549.38it/s]
 27%|██▋       | 29912/109248 [00:19<00:51, 1546.17it/s]
 28%|██▊       | 30068/109248 [00:19<00:51, 1547.57it/s]
 28%|██▊       | 30223/109248 [00:19<00:51, 1540.23it/s]
 28%|██▊       | 30378/109248 [00:19<00:51, 1537.01it/s]
 28%|██▊       | 30536/109248 [00:19<00:50, 1547.31it/s]
 28%|██▊       | 30692/109248 [00:20<00:50, 1548.41it/s]
 28%|██▊       | 30847/109248 [00:20<00:51, 1509.33it/s]
 28%|██▊       | 31006/109248 [00:20<00:51, 1531.49it/s]
 29%|██▊       | 31163/109248 [00:20<00:50, 1540.32it/s]
 29%|██▊       | 31319/109248 [00:20<00:50, 1542.55it/s]
 29%|██▉       | 31474/109248 [00:20<00:50, 1538.47it/s]
 29%|██▉       | 31640/109248 [00:20<00:49, 1570.89it/s]
 29%|██▉       | 31799/109248 [00:20<00:49, 1573.80it/s]
 29%|██▉       | 31957/109248 [00:20<00:49, 1550.54it/s]
 29%|██▉       | 32113/109248 [00:20<00:50, 1517.15it/s]
 30%|██▉       | 32266/109248 [00:21<00:50, 1520.79it/s]
 30%|██▉       | 32419/109248 [00:21<00:52, 1459.58it/s]
 30%|██▉       | 32576/109248 [00:21<00:51, 1490.44it/s]
 30%|██▉       | 32732/109248 [00:21<00:50, 1508.74it/s]
 30%|███       | 32884/109248 [00:21<00:50, 1511.82it/s]
 30%|███       | 33038/109248 [00:21<00:50, 1519.44it/s]
 30%|███       | 33191/109248 [00:21<00:50, 1517.96it/s]
 31%|███       | 33346/109248 [00:21<00:49, 1527.38it/s]
 31%|███       | 33499/109248 [00:21<00:49, 1523.49it/s]
```

```
31%|███       | 33654/109248 [00:21<00:49, 1529.54it/s]
31%|███       | 33810/109248 [00:22<00:49, 1538.10it/s]
31%|███       | 33964/109248 [00:22<00:49, 1514.10it/s]
31%|███       | 34123/109248 [00:22<00:48, 1533.63it/s]
31%|███▏      | 34286/109248 [00:22<00:48, 1559.92it/s]
32%|███▏      | 34443/109248 [00:22<00:49, 1515.17it/s]
32%|███▏      | 34603/109248 [00:22<00:48, 1538.81it/s]
32%|███▏      | 34772/109248 [00:22<00:47, 1578.25it/s]
32%|███▏      | 34934/109248 [00:22<00:46, 1586.59it/s]
32%|███▏      | 35094/109248 [00:22<00:47, 1561.57it/s]
32%|███▏      | 35255/109248 [00:23<00:46, 1575.10it/s]
32%|███▏      | 35413/109248 [00:23<00:47, 1547.34it/s]
33%|███▎      | 35569/109248 [00:23<00:48, 1524.13it/s]
33%|███▎      | 35724/109248 [00:23<00:48, 1528.64it/s]
33%|███▎      | 35885/109248 [00:23<00:47, 1550.84it/s]
33%|███▎      | 36041/109248 [00:23<00:47, 1538.44it/s]
33%|███▎      | 36196/109248 [00:23<00:48, 1519.82it/s]
33%|███▎      | 36352/109248 [00:23<00:47, 1529.55it/s]
33%|███▎      | 36506/109248 [00:23<00:47, 1521.94it/s]
34%|███▎      | 36659/109248 [00:23<00:48, 1505.56it/s]
34%|███▎      | 36822/109248 [00:24<00:47, 1538.96it/s]
34%|███▍      | 36979/109248 [00:24<00:46, 1548.09it/s]
34%|███▍      | 37135/109248 [00:24<00:47, 1511.15it/s]
34%|███▍      | 37291/109248 [00:24<00:47, 1524.61it/s]
34%|███▍      | 37444/109248 [00:24<00:47, 1516.80it/s]
34%|███▍      | 37609/109248 [00:24<00:46, 1552.58it/s]
35%|███▍      | 37766/109248 [00:24<00:45, 1555.41it/s]
35%|███▍      | 37922/109248 [00:24<00:46, 1527.90it/s]
35%|███▍      | 38076/109248 [00:24<00:46, 1526.83it/s]
35%|███▍      | 38236/109248 [00:24<00:45, 1545.88it/s]
35%|███▌      | 38394/109248 [00:25<00:45, 1555.56it/s]
35%|███▌      | 38550/109248 [00:25<00:45, 1538.95it/s]
35%|███▌      | 38705/109248 [00:25<00:46, 1506.66it/s]
36%|███▌      | 38862/109248 [00:25<00:46, 1523.29it/s]
36%|███▌      | 39023/109248 [00:25<00:45, 1547.61it/s]
36%|███▌      | 39179/109248 [00:25<00:45, 1543.38it/s]
36%|███▌      | 39334/109248 [00:25<00:45, 1542.88it/s]
36%|███▌      | 39489/109248 [00:25<00:45, 1526.67it/s]
36%|███▋      | 39642/109248 [00:25<00:45, 1526.48it/s]
```

```
 36%|███▋      | 39795/109248 [00:25<00:45, 1523.07it/s]
 37%|███▋      | 39948/109248 [00:26<00:45, 1516.64it/s]
 37%|███▋      | 40100/109248 [00:26<00:45, 1506.33it/s]
 37%|███▋      | 40251/109248 [00:26<00:46, 1497.14it/s]
 37%|███▋      | 40401/109248 [00:26<00:46, 1496.57it/s]
 37%|███▋      | 40560/109248 [00:26<00:45, 1522.91it/s]
 37%|███▋      | 40713/109248 [00:26<00:45, 1520.81it/s]
 37%|███▋      | 40866/109248 [00:26<00:44, 1522.83it/s]
 38%|███▊      | 41019/109248 [00:26<00:45, 1514.47it/s]
 38%|███▊      | 41171/109248 [00:26<00:45, 1509.51it/s]
 38%|███▊      | 41322/109248 [00:27<00:45, 1502.47it/s]
 38%|███▊      | 41477/109248 [00:27<00:44, 1514.40it/s]
 38%|███▊      | 41638/109248 [00:27<00:43, 1540.92it/s]
 38%|███▊      | 41793/109248 [00:27<00:44, 1506.29it/s]
 38%|███▊      | 41952/109248 [00:27<00:44, 1527.63it/s]
 39%|███▊      | 42106/109248 [00:27<00:43, 1526.46it/s]
 39%|███▊      | 42261/109248 [00:27<00:43, 1532.76it/s]
 39%|███▉      | 42417/109248 [00:27<00:43, 1538.42it/s]
 39%|███▉      | 42571/109248 [00:27<00:43, 1523.36it/s]
 39%|███▉      | 42724/109248 [00:27<00:44, 1509.93it/s]
 39%|███▉      | 42876/109248 [00:28<00:43, 1510.10it/s]
 39%|███▉      | 43028/109248 [00:28<00:43, 1511.54it/s]
 40%|███▉      | 43188/109248 [00:28<00:43, 1534.68it/s]
 40%|███▉      | 43342/109248 [00:28<00:43, 1506.24it/s]
 40%|███▉      | 43493/109248 [00:28<00:44, 1493.17it/s]
 40%|███▉      | 43645/109248 [00:28<00:43, 1498.40it/s]
 40%|████      | 43795/109248 [00:28<00:44, 1482.52it/s]
 40%|████      | 43948/109248 [00:28<00:43, 1496.15it/s]
 40%|████      | 44105/109248 [00:28<00:42, 1516.85it/s]
 41%|████      | 44257/109248 [00:28<00:43, 1509.24it/s]
 41%|████      | 44417/109248 [00:29<00:42, 1533.71it/s]
 41%|████      | 44571/109248 [00:29<00:42, 1528.06it/s]
 41%|████      | 44724/109248 [00:29<00:42, 1526.32it/s]
 41%|████      | 44877/109248 [00:29<00:43, 1495.20it/s]
 41%|████      | 45036/109248 [00:29<00:42, 1522.25it/s]
 41%|████▏     | 45199/109248 [00:29<00:41, 1550.14it/s]
 42%|████▏     | 45355/109248 [00:29<00:41, 1530.60it/s]
 42%|████▏     | 45509/109248 [00:29<00:41, 1531.05it/s]
 42%|████      | 45664/109248 [00:29<00:41, 1534.64it/s]
 42%|████      | 45818/109248 [00:29<00:41, 1531.69it/s]
 42%|████      | 45975/109248 [00:30<00:41, 1542.66it/s]
 42%|████      | 46130/109248 [00:30<00:41, 1530.46it/s]
 42%|████      | 46287/109248 [00:30<00:40, 1541.90it/s]
 43%|████▎     | 46442/109248 [00:30<00:41, 1512.53it/s]
 43%|████▎     | 46598/109248 [00:30<00:41, 1525.73it/s]
 43%|████▎     | 46756/109248 [00:30<00:40, 1539.44it/s]
 43%|████▎     | 46920/109248 [00:30<00:39, 1566.33it/s]
 43%|████▎     | 47086/109248 [00:30<00:39, 1593.17it/s]
 43%|████▎     | 47246/109248 [00:30<00:39, 1587.31it/s]
 43%|████▎     | 47405/109248 [00:30<00:39, 1581.97it/s]
 44%|████▎     | 47564/109248 [00:31<00:40, 1536.33it/s]
 44%|████▎     | 47719/109248 [00:31<00:40, 1513.00it/s]
 44%|████▍     | 47876/109248 [00:31<00:40, 1527.73it/s]
 44%|████▍     | 48030/109248 [00:31<00:41, 1492.79it/s]
 44%|████▍     | 48180/109248 [00:31<00:41, 1488.92it/s]
 44%|████▍     | 48339/109248 [00:31<00:40, 1516.79it/s]
 44%|████▍     | 48491/109248 [00:31<00:40, 1509.37it/s]
 45%|████▍     | 48643/109248 [00:31<00:41, 1470.06it/s]
 45%|████▍     | 48799/109248 [00:31<00:40, 1494.13it/s]
 45%|████▍     | 48952/109248 [00:32<00:40, 1502.86it/s]
 45%|████▍     | 49107/109248 [00:32<00:39, 1515.66it/s]
 45%|████▌     | 49266/109248 [00:32<00:39, 1536.93it/s]
 45%|████▌     | 49420/109248 [00:32<00:39, 1512.30it/s]
 45%|████▌     | 49572/109248 [00:32<00:40, 1487.44it/s]
 46%|████▌     | 49722/109248 [00:32<00:39, 1491.16it/s]
 46%|████▌     | 49878/109248 [00:32<00:39, 1508.27it/s]
 46%|████▌     | 50036/109248 [00:32<00:38, 1527.20it/s]
 46%|████▌     | 50189/109248 [00:32<00:38, 1517.35it/s]
 46%|████▌     | 50341/109248 [00:32<00:38, 1511.98it/s]
 46%|████▌     | 50493/109248 [00:33<00:39, 1498.75it/s]
 46%|████▋     | 50647/109248 [00:33<00:38, 1509.99it/s]
 47%|████▋     | 50807/109248 [00:33<00:38, 1534.58it/s]
 47%|████▋     | 50964/109248 [00:33<00:37, 1543.30it/s]
 47%|████▋     | 51119/109248 [00:33<00:38, 1493.42it/s]
 47%|████▋     | 51269/109248 [00:33<00:39, 1457.22it/s]
 47%|████▋     | 51424/109248 [00:33<00:39, 1481.06it/s]
 47%|████▋     | 51575/109248 [00:33<00:38, 1488.83it/s]
```

```
47%|████▋     | 51726/109248 [00:33<00:38, 1492.88it/s]
47%|████▋     | 51876/109248 [00:33<00:38, 1480.27it/s]
48%|████▊     | 52030/109248 [00:34<00:38, 1496.56it/s]
48%|████▊     | 52180/109248 [00:34<00:38, 1488.26it/s]
48%|████▊     | 52336/109248 [00:34<00:37, 1507.58it/s]
48%|████▊     | 52490/109248 [00:34<00:37, 1516.55it/s]
48%|████▊     | 52642/109248 [00:34<00:38, 1486.58it/s]
48%|████▊     | 52791/109248 [00:34<00:38, 1482.35it/s]
48%|████▊     | 52944/109248 [00:34<00:37, 1494.43it/s]
49%|████▊     | 53094/109248 [00:34<00:37, 1496.07it/s]
49%|████▊     | 53244/109248 [00:34<00:37, 1493.69it/s]
49%|████▉     | 53394/109248 [00:34<00:37, 1489.85it/s]
49%|████▉     | 53547/109248 [00:35<00:37, 1499.53it/s]
49%|████▉     | 53700/109248 [00:35<00:36, 1506.27it/s]
49%|████▉     | 53851/109248 [00:35<00:37, 1482.19it/s]
49%|████▉     | 54011/109248 [00:35<00:36, 1515.08it/s]
50%|████▉     | 54163/109248 [00:35<00:37, 1476.72it/s]
50%|████▉     | 54317/109248 [00:35<00:36, 1493.85it/s]
50%|████▉     | 54478/109248 [00:35<00:35, 1525.67it/s]
50%|█████     | 54636/109248 [00:35<00:35, 1539.79it/s]
50%|█████     | 54791/109248 [00:35<00:35, 1532.90it/s]
50%|█████     | 54945/109248 [00:35<00:35, 1527.93it/s]
50%|█████     | 55098/109248 [00:36<00:35, 1510.51it/s]
51%|█████     | 55251/109248 [00:36<00:35, 1513.80it/s]
51%|█████     | 55407/109248 [00:36<00:35, 1526.23it/s]
51%|█████     | 55567/109248 [00:36<00:34, 1547.06it/s]
51%|█████     | 55722/109248 [00:36<00:35, 1495.14it/s]
51%|█████     | 55875/109248 [00:36<00:35, 1505.05it/s]
51%|█████▏    | 56034/109248 [00:36<00:34, 1526.87it/s]
51%|█████▏    | 56188/109248 [00:36<00:35, 1507.51it/s]
52%|█████▏    | 56340/109248 [00:36<00:35, 1502.44it/s]
52%|█████▏    | 56491/109248 [00:37<00:35, 1469.45it/s]
52%|█████▏    | 56639/109248 [00:37<00:35, 1466.31it/s]
52%|█████▏    | 56790/109248 [00:37<00:35, 1478.95it/s]
52%|█████▏    | 56943/109248 [00:37<00:35, 1493.37it/s]
52%|█████▏    | 57101/109248 [00:37<00:34, 1515.49it/s]
52%|█████▏    | 57253/109248 [00:37<00:35, 1473.09it/s]
53%|█████▎    | 57402/109248 [00:37<00:35, 1477.66it/s]
```

```
 53%|█████▎    | 57559/109248 [00:37<00:34, 1502.84it/s]
 53%|█████▎    | 57716/109248 [00:37<00:33, 1520.70it/s]
 53%|█████▎    | 57869/109248 [00:37<00:33, 1515.91it/s]
 53%|█████▎    | 58026/109248 [00:38<00:33, 1530.21it/s]
 53%|█████▎    | 58180/109248 [00:38<00:33, 1502.66it/s]
 53%|█████▎    | 58342/109248 [00:38<00:33, 1534.90it/s]
 54%|█████▎    | 58505/109248 [00:38<00:32, 1561.86it/s]
 54%|█████▎    | 58662/109248 [00:38<00:32, 1559.50it/s]
 54%|█████▍    | 58819/109248 [00:38<00:33, 1520.49it/s]
 54%|█████▍    | 58978/109248 [00:38<00:32, 1538.54it/s]
 54%|█████▍    | 59134/109248 [00:38<00:32, 1543.67it/s]
 54%|█████▍    | 59290/109248 [00:38<00:32, 1547.31it/s]
 54%|█████▍    | 59445/109248 [00:38<00:32, 1536.70it/s]
 55%|█████▍    | 59602/109248 [00:39<00:32, 1545.95it/s]
 55%|█████▍    | 59763/109248 [00:39<00:31, 1562.16it/s]
 55%|█████▍    | 59920/109248 [00:39<00:31, 1563.62it/s]
 55%|█████▍    | 60081/109248 [00:39<00:31, 1575.93it/s]
 55%|█████▌    | 60239/109248 [00:39<00:31, 1562.25it/s]
 55%|█████▌    | 60396/109248 [00:39<00:32, 1518.05it/s]
 55%|█████▌    | 60558/109248 [00:39<00:31, 1546.18it/s]
 56%|█████▌    | 60715/109248 [00:39<00:31, 1553.20it/s]
 56%|█████▌    | 60872/109248 [00:39<00:31, 1558.18it/s]
 56%|█████▌    | 61029/109248 [00:39<00:31, 1544.87it/s]
 56%|█████▌    | 61184/109248 [00:40<00:31, 1523.00it/s]
 56%|█████▌    | 61342/109248 [00:40<00:31, 1537.63it/s]
 56%|█████▋    | 61497/109248 [00:40<00:30, 1540.53it/s]
 56%|█████▋    | 61652/109248 [00:40<00:30, 1536.47it/s]
 57%|█████▋    | 61806/109248 [00:40<00:31, 1499.60it/s]
 57%|█████▋    | 61957/109248 [00:40<00:32, 1462.38it/s]
 57%|█████▋    | 62104/109248 [00:40<00:32, 1458.50it/s]
 57%|█████▋    | 62261/109248 [00:40<00:31, 1489.89it/s]
 57%|█████▋    | 62415/109248 [00:40<00:31, 1502.83it/s]
 57%|█████▋    | 62566/109248 [00:41<00:31, 1490.04it/s]
 57%|█████▋    | 62725/109248 [00:41<00:30, 1517.58it/s]
 58%|█████▊    | 62883/109248 [00:41<00:30, 1534.00it/s]
 58%|█████▊    | 63043/109248 [00:41<00:29, 1553.17it/s]
 58%|█████▊    | 63199/109248 [00:41<00:29, 1543.81it/s]
 58%|█████▊    | 63355/109248 [00:41<00:29, 1547.44it/s]
 58%|█████▊    | 63510/109248 [00:41<00:30, 1503.38it/s]
 58%|█████▊    | 63669/109248 [00:41<00:29, 1527.11it/s]
 58%|█████▊    | 63832/109248 [00:41<00:29, 1556.31it/s]
 59%|█████▊    | 63991/109248 [00:41<00:28, 1566.06it/s]
 59%|█████▊    | 64148/109248 [00:42<00:28, 1566.51it/s]
 59%|█████▉    | 64306/109248 [00:42<00:28, 1569.48it/s]
 59%|█████▉    | 64464/109248 [00:42<00:28, 1572.55it/s]
 59%|█████▉    | 64625/109248 [00:42<00:28, 1582.60it/s]
 59%|█████▉    | 64787/109248 [00:42<00:27, 1591.77it/s]
 59%|█████▉    | 64947/109248 [00:42<00:28, 1569.70it/s]
 60%|█████▉    | 65105/109248 [00:42<00:29, 1510.31it/s]
 60%|█████▉    | 65257/109248 [00:42<00:29, 1508.51it/s]
 60%|█████▉    | 65411/109248 [00:42<00:28, 1516.16it/s]
 60%|██████    | 65567/109248 [00:42<00:28, 1528.16it/s]
 60%|██████    | 65729/109248 [00:43<00:28, 1551.74it/s]
 60%|██████    | 65887/109248 [00:43<00:27, 1557.60it/s]
 60%|██████    | 66043/109248 [00:43<00:27, 1557.63it/s]
 61%|██████    | 66202/109248 [00:43<00:27, 1565.72it/s]
 61%|██████    | 66359/109248 [00:43<00:27, 1563.62it/s]
 61%|██████    | 66516/109248 [00:43<00:27, 1547.16it/s]
 61%|██████    | 66671/109248 [00:43<00:27, 1520.93it/s]
 61%|██████    | 66829/109248 [00:43<00:27, 1536.88it/s]
 61%|██████▏   | 66990/109248 [00:43<00:27, 1557.24it/s]
 61%|██████▏   | 67146/109248 [00:43<00:27, 1548.36it/s]
 62%|██████▏   | 67307/109248 [00:44<00:26, 1566.14it/s]
 62%|██████▏   | 67464/109248 [00:44<00:26, 1562.70it/s]
 62%|██████▏   | 67626/109248 [00:44<00:26, 1577.08it/s]
 62%|██████▏   | 67784/109248 [00:44<00:26, 1536.14it/s]
 62%|██████▏   | 67938/109248 [00:44<00:27, 1509.69it/s]
 62%|██████▏   | 68090/109248 [00:44<00:27, 1472.58it/s]
 62%|██████▏   | 68238/109248 [00:44<00:28, 1448.73it/s]
 63%|██████▎   | 68402/109248 [00:44<00:27, 1498.80it/s]
 63%|██████▎   | 68553/109248 [00:44<00:27, 1489.02it/s]
 63%|██████▎   | 68708/109248 [00:44<00:26, 1506.69it/s]
 63%|██████▎   | 68860/109248 [00:45<00:26, 1506.61it/s]
 63%|██████▎   | 69021/109248 [00:45<00:26, 1535.66it/s]
 63%|██████▎   | 69179/109248 [00:45<00:25, 1546.36it/s]
 63%|██████▎   | 69334/109248 [00:45<00:26, 1531.77it/s]
 64%|██████▎   | 69491/109248 [00:45<00:25, 1542.37it/s]
 64%|██████▎   | 69650/109248 [00:45<00:25, 1555.68it/s]
 64%|██████▍   | 69806/109248 [00:45<00:25, 1536.87it/s]
 64%|██████▍   | 69960/109248 [00:45<00:25, 1514.08it/s]
 64%|██████▍   | 70112/109248 [00:45<00:25, 1506.38it/s]
 64%|██████▍   | 70273/109248 [00:46<00:25, 1535.74it/s]
 64%|██████▍   | 70431/109248 [00:46<00:25, 1548.50it/s]
 65%|██████▍   | 70588/109248 [00:46<00:24, 1554.32it/s]
 65%|██████▍   | 70744/109248 [00:46<00:24, 1544.55it/s]
 65%|██████▍   | 70899/109248 [00:46<00:24, 1538.41it/s]
 65%|██████▌   | 71058/109248 [00:46<00:24, 1552.14it/s]
 65%|██████▌   | 71222/109248 [00:46<00:24, 1575.37it/s]
 65%|██████▌   | 71380/109248 [00:46<00:24, 1540.47it/s]
 65%|██████▌   | 71535/109248 [00:46<00:25, 1503.21it/s]
 66%|██████▌   | 71686/109248 [00:46<00:24, 1503.62it/s]
 66%|██████▌   | 71845/109248 [00:47<00:24, 1528.14it/s]
 66%|██████▌   | 71999/109248 [00:47<00:24, 1518.85it/s]
 66%|██████▌   | 72152/109248 [00:47<00:24, 1519.35it/s]
 66%|██████▌   | 72305/109248 [00:47<00:24, 1521.82it/s]
 66%|██████▋   | 72464/109248 [00:47<00:23, 1541.61it/s]
 66%|██████▋   | 72619/109248 [00:47<00:24, 1515.89it/s]
 67%|██████▋   | 72778/109248 [00:47<00:23, 1534.56it/s]
 67%|██████▋   | 72932/109248 [00:47<00:23, 1515.21it/s]
 67%|██████▋   | 73084/109248 [00:47<00:24, 1479.89it/s]
 67%|██████▋   | 73233/109248 [00:47<00:24, 1478.01it/s]
 67%|██████▋   | 73387/109248 [00:48<00:24, 1489.60it/s]
 67%|██████▋   | 73544/109248 [00:48<00:23, 1511.78it/s]
 67%|██████▋   | 73699/109248 [00:48<00:23, 1521.74it/s]
 68%|██████▊   | 73855/109248 [00:48<00:23, 1530.69it/s]
 68%|██████▊   | 74009/109248 [00:48<00:23, 1529.61it/s]
 68%|██████▊   | 74163/109248 [00:48<00:22, 1526.45it/s]
 68%|██████▊   | 74320/109248 [00:48<00:22, 1536.87it/s]
 68%|██████▊   | 74474/109248 [00:48<00:22, 1528.90it/s]
 68%|██████▊   | 74627/109248 [00:48<00:23, 1481.72it/s]
 68%|██████▊   | 74776/109248 [00:48<00:23, 1471.70it/s]
 69%|██████▊   | 74925/109248 [00:49<00:23, 1475.89it/s]
 69%|██████▊   | 75078/109248 [00:49<00:22, 1490.88it/s]
 69%|██████▉   | 75237/109248 [00:49<00:22, 1518.15it/s]
 69%|██████▉   | 75392/109248 [00:49<00:22, 1525.47it/s]
 69%|██████▉   | 75545/109248 [00:49<00:22, 1521.96it/s]
```

```
69%|██████▉   | 75702/109248 [00:49<00:21, 1534.68it/s]
69%|██████▉   | 75861/109248 [00:49<00:21, 1549.69it/s]
70%|██████▉   | 76017/109248 [00:49<00:21, 1547.61it/s]
70%|██████▉   | 76172/109248 [00:49<00:22, 1501.27it/s]
70%|██████▉   | 76323/109248 [00:49<00:22, 1493.19it/s]
70%|███████   | 76480/109248 [00:50<00:21, 1514.29it/s]
70%|███████   | 76640/109248 [00:50<00:21, 1536.43it/s]
70%|███████   | 76799/109248 [00:50<00:20, 1550.69it/s]
70%|███████   | 76955/109248 [00:50<00:20, 1551.79it/s]
71%|███████   | 77114/109248 [00:50<00:20, 1563.06it/s]
71%|███████   | 77272/109248 [00:50<00:20, 1566.00it/s]
71%|███████   | 77430/109248 [00:50<00:20, 1567.69it/s]
71%|███████   | 77591/109248 [00:50<00:20, 1577.93it/s]
71%|███████   | 77749/109248 [00:50<00:20, 1532.50it/s]
71%|███████▏  | 77903/109248 [00:51<00:20, 1508.81it/s]
71%|███████▏  | 78055/109248 [00:51<00:20, 1497.34it/s]
72%|███████▏  | 78211/109248 [00:51<00:20, 1513.44it/s]
72%|███████▏  | 78371/109248 [00:51<00:20, 1535.82it/s]
72%|███████▏  | 78527/109248 [00:51<00:19, 1541.59it/s]
72%|███████▏  | 78682/109248 [00:51<00:19, 1531.27it/s]
72%|███████▏  | 78836/109248 [00:51<00:20, 1518.10it/s]
72%|███████▏  | 78988/109248 [00:51<00:20, 1512.42it/s]
72%|███████▏  | 79140/109248 [00:51<00:19, 1514.04it/s]
73%|███████▎  | 79292/109248 [00:51<00:20, 1475.21it/s]
73%|███████▎  | 79443/109248 [00:52<00:20, 1484.26it/s]
73%|███████▎  | 79598/109248 [00:52<00:19, 1502.36it/s]
73%|███████▎  | 79764/109248 [00:52<00:19, 1545.33it/s]
73%|███████▎  | 79921/109248 [00:52<00:18, 1550.21it/s]
73%|███████▎  | 80077/109248 [00:52<00:18, 1544.17it/s]
73%|███████▎  | 80232/109248 [00:52<00:18, 1541.85it/s]
74%|███████▎  | 80387/109248 [00:52<00:18, 1528.44it/s]
74%|███████▎  | 80545/109248 [00:52<00:18, 1541.92it/s]
74%|███████▍  | 80700/109248 [00:52<00:18, 1523.61it/s]
74%|███████▍  | 80853/109248 [00:52<00:19, 1484.56it/s]
74%|███████▍  | 81005/109248 [00:53<00:18, 1493.87it/s]
74%|███████▍  | 81169/109248 [00:53<00:18, 1533.16it/s]
74%|███████▍  | 81324/109248 [00:53<00:18, 1535.00it/s]
75%|███████▍  | 81480/109248 [00:53<00:18, 1541.03it/s]
```

```
 75%|███████▌  | 81636/109248 [00:53<00:17, 1544.33it/s]
 75%|███████▌  | 81791/109248 [00:53<00:18, 1511.55it/s]
 75%|███████▌  | 81946/109248 [00:53<00:17, 1521.52it/s]
 75%|███████▌  | 82105/109248 [00:53<00:17, 1538.17it/s]
 75%|███████▌  | 82264/109248 [00:53<00:17, 1553.34it/s]
 75%|███████▌  | 82420/109248 [00:53<00:17, 1517.90it/s]
 76%|███████▌  | 82578/109248 [00:54<00:17, 1534.42it/s]
 76%|███████▌  | 82732/109248 [00:54<00:17, 1516.09it/s]
 76%|███████▌  | 82885/109248 [00:54<00:17, 1518.67it/s]
 76%|███████▌  | 83038/109248 [00:54<00:17, 1510.96it/s]
 76%|███████▌  | 83192/109248 [00:54<00:17, 1518.90it/s]
 76%|███████▋  | 83352/109248 [00:54<00:16, 1540.99it/s]
 76%|███████▋  | 83513/109248 [00:54<00:16, 1560.68it/s]
 77%|███████▋  | 83670/109248 [00:54<00:16, 1557.69it/s]
 77%|███████▋  | 83835/109248 [00:54<00:16, 1582.65it/s]
 77%|███████▋  | 83994/109248 [00:54<00:16, 1525.42it/s]
 77%|███████▋  | 84156/109248 [00:55<00:16, 1551.64it/s]
 77%|███████▋  | 84312/109248 [00:55<00:16, 1539.24it/s]
 77%|███████▋  | 84467/109248 [00:55<00:16, 1534.70it/s]
 77%|███████▋  | 84625/109248 [00:55<00:15, 1547.44it/s]
 78%|███████▊  | 84780/109248 [00:55<00:15, 1537.80it/s]
 78%|███████▊  | 84939/109248 [00:55<00:15, 1549.09it/s]
 78%|███████▊  | 85095/109248 [00:55<00:15, 1549.93it/s]
 78%|███████▊  | 85251/109248 [00:55<00:15, 1547.68it/s]
 78%|███████▊  | 85413/109248 [00:55<00:15, 1565.49it/s]
 78%|███████▊  | 85570/109248 [00:56<00:15, 1520.28it/s]
 78%|███████▊  | 85729/109248 [00:56<00:15, 1540.42it/s]
 79%|███████▊  | 85892/109248 [00:56<00:14, 1565.50it/s]
 79%|███████▉  | 86049/109248 [00:56<00:14, 1554.49it/s]
 79%|███████▉  | 86205/109248 [00:56<00:14, 1545.20it/s]
 79%|███████▉  | 86360/109248 [00:56<00:14, 1529.04it/s]
 79%|███████▉  | 86516/109248 [00:56<00:14, 1537.55it/s]
 79%|███████▉  | 86671/109248 [00:56<00:14, 1539.13it/s]
 79%|███████▉  | 86826/109248 [00:56<00:14, 1527.40it/s]
 80%|███████▉  | 86979/109248 [00:56<00:14, 1518.31it/s]
 80%|███████▉  | 87131/109248 [00:57<00:14, 1478.94it/s]
 80%|███████▉  | 87285/109248 [00:57<00:14, 1494.86it/s]
 80%|████████  | 87436/109248 [00:57<00:14, 1497.55it/s]
 80%|████████  | 87596/109248 [00:57<00:14, 1525.23it/s]
```

```
80%|███████   | 87598/109248 [00:57<00:14, 1525.25it/s]
80%|███████   | 87758/109248 [00:57<00:13, 1551.28it/s]
80%|███████   | 87917/109248 [00:57<00:13, 1560.65it/s]
81%|███████   | 88078/109248 [00:57<00:13, 1573.47it/s]
81%|███████   | 88236/109248 [00:57<00:13, 1551.37it/s]
81%|███████   | 88392/109248 [00:57<00:13, 1551.70it/s]
81%|███████   | 88548/109248 [00:57<00:13, 1535.51it/s]
81%|███████   | 88702/109248 [00:58<00:13, 1474.96it/s]
81%|███████   | 88862/109248 [00:58<00:13, 1509.84it/s]
81%|███████   | 89025/109248 [00:58<00:13, 1543.84it/s]
82%|███████   | 89181/109248 [00:58<00:12, 1547.29it/s]
82%|███████   | 89337/109248 [00:58<00:12, 1538.23it/s]
82%|███████   | 89497/109248 [00:58<00:12, 1555.40it/s]
82%|███████   | 89653/109248 [00:58<00:12, 1542.42it/s]
82%|███████   | 89808/109248 [00:58<00:12, 1542.74it/s]
82%|███████   | 89967/109248 [00:58<00:12, 1555.40it/s]
83%|███████   | 90130/109248 [00:58<00:12, 1573.52it/s]
83%|███████   | 90288/109248 [00:59<00:12, 1523.37it/s]
83%|███████   | 90441/109248 [00:59<00:12, 1488.55it/s]
83%|███████   | 90594/109248 [00:59<00:12, 1500.15it/s]
83%|███████   | 90756/109248 [00:59<00:12, 1531.92it/s]
83%|███████   | 90910/109248 [00:59<00:12, 1526.75it/s]
83%|███████   | 91067/109248 [00:59<00:11, 1536.81it/s]
84%|███████   | 91228/109248 [00:59<00:11, 1556.12it/s]
84%|███████   | 91384/109248 [00:59<00:11, 1551.71it/s]
84%|███████   | 91540/109248 [00:59<00:11, 1534.81it/s]
84%|███████   | 91694/109248 [01:00<00:11, 1524.89it/s]
84%|███████   | 91847/109248 [01:00<00:11, 1486.87it/s]
84%|███████   | 92003/109248 [01:00<00:11, 1505.22it/s]
84%|███████   | 92154/109248 [01:00<00:11, 1499.45it/s]
84%|███████   | 92305/109248 [01:00<00:11, 1490.18it/s]
85%|███████   | 92455/109248 [01:00<00:11, 1462.45it/s]
85%|███████   | 92611/109248 [01:00<00:11, 1489.13it/s]
85%|███████   | 92772/109248 [01:00<00:10, 1521.75it/s]
85%|███████   | 92927/109248 [01:00<00:10, 1528.15it/s]
85%|███████   | 93081/109248 [01:00<00:10, 1525.01it/s]
85%|███████   | 93235/109248 [01:01<00:10, 1529.30it/s]
85%|███████   | 93389/109248 [01:01<00:10, 1507.84it/s]
86%|███████   | 93540/109248 [01:01<00:10, 1466.60it/s]
```

```
 86%|████████▌ | 93690/109248 [01:01<00:10, 1476.17it/s]
 86%|████████▌ | 93845/109248 [01:01<00:10, 1496.10it/s]
 86%|████████▌ | 93995/109248 [01:01<00:10, 1482.92it/s]
 86%|████████▌ | 94149/109248 [01:01<00:10, 1496.69it/s]
 86%|████████▋ | 94306/109248 [01:01<00:09, 1514.34it/s]
 86%|████████▋ | 94463/109248 [01:01<00:09, 1526.48it/s]
 87%|████████▋ | 94622/109248 [01:01<00:09, 1544.27it/s]
 87%|████████▋ | 94777/109248 [01:02<00:09, 1522.05it/s]
 87%|████████▋ | 94930/109248 [01:02<00:09, 1480.91it/s]
 87%|████████▋ | 95080/109248 [01:02<00:09, 1485.75it/s]
 87%|████████▋ | 95241/109248 [01:02<00:09, 1518.24it/s]
 87%|████████▋ | 95399/109248 [01:02<00:09, 1535.31it/s]
 87%|████████▋ | 95559/109248 [01:02<00:08, 1552.26it/s]
 88%|████████▊ | 95716/109248 [01:02<00:08, 1557.07it/s]
 88%|████████▊ | 95872/109248 [01:02<00:08, 1557.41it/s]
 88%|████████▊ | 96028/109248 [01:02<00:08, 1533.91it/s]
 88%|████████▊ | 96182/109248 [01:02<00:08, 1511.45it/s]
 88%|████████▊ | 96334/109248 [01:03<00:08, 1496.69it/s]
 88%|████████▊ | 96484/109248 [01:03<00:08, 1441.86it/s]
 88%|████████▊ | 96632/109248 [01:03<00:08, 1453.08it/s]
 89%|████████▊ | 96784/109248 [01:03<00:08, 1471.70it/s]
 89%|████████▊ | 96937/109248 [01:03<00:08, 1485.64it/s]
 89%|████████▉ | 97086/109248 [01:03<00:08, 1470.86it/s]
 89%|████████▉ | 97237/109248 [01:03<00:08, 1481.38it/s]
 89%|████████▉ | 97388/109248 [01:03<00:07, 1489.42it/s]
 89%|████████▉ | 97542/109248 [01:03<00:07, 1502.67it/s]
 89%|████████▉ | 97696/109248 [01:04<00:07, 1511.58it/s]
 90%|████████▉ | 97851/109248 [01:04<00:07, 1520.46it/s]
 90%|████████▉ | 98004/109248 [01:04<00:07, 1474.16it/s]
 90%|████████▉ | 98154/109248 [01:04<00:07, 1481.66it/s]
 90%|████████▉ | 98306/109248 [01:04<00:07, 1491.52it/s]
 90%|█████████ | 98456/109248 [01:04<00:07, 1493.80it/s]
 90%|█████████ | 98606/109248 [01:04<00:07, 1488.19it/s]
 90%|█████████ | 98756/109248 [01:04<00:07, 1491.52it/s]
 91%|█████████ | 98906/109248 [01:04<00:06, 1492.84it/s]
 91%|█████████ | 99060/109248 [01:04<00:06, 1504.59it/s]
 91%|█████████ | 99211/109248 [01:05<00:06, 1496.72it/s]
 91%|█████████ | 99361/109248 [01:05<00:06, 1484.62it/s]
 91%|█████████ | 99510/109248 [01:05<00:06, 1452.85it/s]
```

```
91%|█████████▏| 99510/109248 [01:05<00:06, 1452.85it/s]
91%|█████████▏| 99660/109248 [01:05<00:06, 1466.53it/s]
91%|█████████▏| 99816/109248 [01:05<00:06, 1490.54it/s]
92%|█████████▏| 99966/109248 [01:05<00:06, 1474.23it/s]
92%|█████████▏| 100118/109248 [01:05<00:06, 1486.65it/s]
92%|█████████▏| 100267/109248 [01:05<00:06, 1478.10it/s]
92%|█████████▏| 100422/109248 [01:05<00:05, 1496.91it/s]
92%|█████████▏| 100572/109248 [01:05<00:05, 1486.38it/s]
92%|█████████▏| 100721/109248 [01:06<00:05, 1444.57it/s]
92%|█████████▏| 100872/109248 [01:06<00:05, 1460.45it/s]
92%|█████████▏| 101019/109248 [01:06<00:05, 1436.04it/s]
93%|█████████▎| 101166/109248 [01:06<00:05, 1443.73it/s]
93%|█████████▎| 101311/109248 [01:06<00:05, 1442.51it/s]
93%|█████████▎| 101456/109248 [01:06<00:05, 1443.57it/s]
93%|█████████▎| 101609/109248 [01:06<00:05, 1468.31it/s]
93%|█████████▎| 101762/109248 [01:06<00:05, 1484.11it/s]
93%|█████████▎| 101911/109248 [01:06<00:05, 1464.76it/s]
93%|█████████▎| 102062/109248 [01:06<00:04, 1477.95it/s]
94%|█████████▎| 102214/109248 [01:07<00:04, 1489.05it/s]
94%|█████████▎| 102364/109248 [01:07<00:04, 1481.49it/s]
94%|█████████▍| 102513/109248 [01:07<00:04, 1423.79it/s]
94%|█████████▍| 102664/109248 [01:07<00:04, 1448.46it/s]
94%|█████████▍| 102821/109248 [01:07<00:04, 1482.44it/s]
94%|█████████▍| 102974/109248 [01:07<00:04, 1495.72it/s]
94%|█████████▍| 103124/109248 [01:07<00:04, 1483.05it/s]
95%|█████████▍| 103279/109248 [01:07<00:03, 1500.84it/s]
95%|█████████▍| 103434/109248 [01:07<00:03, 1513.81it/s]
95%|█████████▍| 103586/109248 [01:07<00:03, 1515.60it/s]
95%|█████████▍| 103738/109248 [01:08<00:03, 1514.77it/s]
95%|█████████▌| 103890/109248 [01:08<00:03, 1501.94it/s]
95%|█████████▌| 104041/109248 [01:08<00:03, 1465.39it/s]
95%|█████████▌| 104196/109248 [01:08<00:03, 1487.68it/s]
96%|█████████▌| 104346/109248 [01:08<00:03, 1484.98it/s]
96%|█████████▌| 104496/109248 [01:08<00:03, 1487.60it/s]
96%|█████████▌| 104659/109248 [01:08<00:03, 1526.46it/s]
96%|█████████▌| 104812/109248 [01:08<00:02, 1509.14it/s]
96%|█████████▌| 104964/109248 [01:08<00:02, 1501.80it/s]
96%|█████████▌| 105115/109248 [01:09<00:02, 1496.00it/s]
96%|█████████▋| 105266/109248 [01:09<00:02, 1498.32it/s]
```

```
 96%|█████████▋| 105422/109248 [01:09<00:02, 1515.12it/s]
 97%|█████████▋| 105574/109248 [01:09<00:02, 1470.75it/s]
 97%|█████████▋| 105728/109248 [01:09<00:02, 1490.79it/s]
 97%|█████████▋| 105882/109248 [01:09<00:02, 1503.73it/s]
 97%|█████████▋| 106040/109248 [01:09<00:02, 1525.57it/s]
 97%|█████████▋| 106193/109248 [01:09<00:02, 1520.33it/s]
 97%|█████████▋| 106346/109248 [01:09<00:01, 1504.40it/s]
 97%|█████████▋| 106501/109248 [01:09<00:01, 1516.39it/s]
 98%|█████████▊| 106654/109248 [01:10<00:01, 1520.20it/s]
 98%|█████████▊| 106812/109248 [01:10<00:01, 1536.21it/s]
 98%|█████████▊| 106969/109248 [01:10<00:01, 1543.21it/s]
 98%|█████████▊| 107124/109248 [01:10<00:01, 1498.89it/s]
 98%|█████████▊| 107280/109248 [01:10<00:01, 1516.46it/s]
 98%|█████████▊| 107437/109248 [01:10<00:01, 1530.11it/s]
 98%|█████████▊| 107593/109248 [01:10<00:01, 1536.32it/s]
 99%|█████████▊| 107755/109248 [01:10<00:00, 1558.96it/s]
 99%|█████████▉| 107913/109248 [01:10<00:00, 1560.77it/s]
 99%|█████████▉| 108070/109248 [01:10<00:00, 1552.65it/s]
 99%|█████████▉| 108226/109248 [01:11<00:00, 1527.38it/s]
 99%|█████████▉| 108379/109248 [01:11<00:00, 1521.37it/s]
 99%|█████████▉| 108536/109248 [01:11<00:00, 1534.97it/s]
 99%|█████████▉| 108690/109248 [01:11<00:00, 1476.42it/s]
100%|█████████▉| 108841/109248 [01:11<00:00, 1486.06it/s]
100%|█████████▉| 108994/109248 [01:11<00:00, 1496.59it/s]
100%|█████████▉| 109150/109248 [01:11<00:00, 1513.52it/s]
100%|██████████| 109248/109248 [01:11<00:00, 1523.05it/s]
```

In [188]:

```
# after preprocesing
preprocessed_essays[20000]
```

Out[188]:

```
'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 sentance in tqdm(project_data['project_essay_1'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e not in stopwords)
    preprocessed_essays_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 sentance in tqdm(project_data['project_essay_2'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e not in stopwords)
    preprocessed_essays_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 [189]:

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

Out[189]:

```
'Be Active! Be Energized!'
```

In [190]:

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

Out[190]:

```
'Educational Support for English Learners at Home'
```

In [191]:

```
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 [192]:

```
# \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 [193]:

```
#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 [194]:

```
# Combining all the above statemennts
from tqdm import tqdm
preprocessed_titles = []
# tqdm is for printing the status bar
for sentance in tqdm(project_data['project_title'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = sent.replace(',',' ')
    sent = sent.replace('!',' ')
    sent = sent.replace('*',' ')
```

```python
    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())
```

```
  0%|          | 0/109248 [00:00<?, ?it/s]

  3%|▎         | 3135/109248 [00:00<00:03, 31346.29it/s]

  6%|▌         | 6145/109248 [00:00<00:03, 30958.09it/s]

  9%|▊         | 9340/109248 [00:00<00:03, 31248.84it/s]

 11%|█         | 12080/109248 [00:00<00:03, 29982.58it/s]

 14%|█▍        | 15058/109248 [00:00<00:03, 29919.93it/s]

 17%|█▋        | 18131/109248 [00:00<00:03, 30157.74it/s]

 20%|█▉        | 21331/109248 [00:00<00:02, 30686.40it/s]

 22%|██▏       | 24487/109248 [00:00<00:02, 30941.89it/s]

 25%|██▌       | 27620/109248 [00:00<00:02, 31056.31it/s]

 28%|██▊       | 30661/109248 [00:01<00:02, 30858.48it/s]

 31%|███       | 33782/109248 [00:01<00:02, 30961.73it/s]

 34%|███▍      | 36877/109248 [00:01<00:02, 30956.38it/s]

 37%|███▋      | 39946/109248 [00:01<00:02, 30873.46it/s]

 39%|███▉      | 43005/109248 [00:01<00:02, 30246.64it/s]

 42%|████▏     | 46129/109248 [00:01<00:02, 30536.79it/s]

 45%|████▌     | 49253/109248 [00:01<00:01, 30744.28it/s]

 48%|████▊     | 52434/109248 [00:01<00:01, 31053.40it/s]

 51%|█████     | 55535/109248 [00:01<00:01, 30865.27it/s]

 54%|█████▎    | 58619/109248 [00:01<00:01, 30759.25it/s]

 57%|█████▋    | 61815/109248 [00:02<00:01, 31109.84it/s]

 59%|█████▉    | 64926/109248 [00:02<00:01, 30781.97it/s]

 62%|██████▏   | 68005/109248 [00:02<00:01, 30756.79it/s]

 65%|██████▌   | 71169/109248 [00:02<00:01, 31013.84it/s]

 68%|██████▊   | 74272/109248 [00:02<00:01, 30118.67it/s]

 71%|███████   | 77291/109248 [00:02<00:01, 30047.74it/s]

 74%|███████▎  | 80301/109248 [00:02<00:00, 30018.65it/s]

 76%|███████▋  | 83504/109248 [00:02<00:00, 30588.32it/s]

 79%|███████▉  | 86568/109248 [00:02<00:00, 29763.84it/s]

 82%|████████▏ | 89583/109248 [00:02<00:00, 29877.49it/s]

 85%|████████▍ | 92592/109248 [00:03<00:00, 29939.91it/s]

 87%|████████▋ | 95591/109248 [00:03<00:00, 29855.64it/s]

 90%|█████████ | 98715/109248 [00:03<00:00, 30256.60it/s]

 93%|█████████▎| 101791/109248 [00:03<00:00, 30403.68it/s]

 96%|█████████▌| 104834/109248 [00:03<00:00, 29476.41it/s]
 99%|█████████▉| 107790/109248 [00:03<00:00, 29439.14it/s]

100%|██████████| 109248/109248 [00:03<00:00, 30303.87it/s]
```

In [195]:

```
preprocessed_titles[20000]
```

Out[195]:

```
'we need to move it while we input it'
```

Resource_summary_Text

In [196]:

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

Out[196]:

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

In [0]:

```
# Combining all the above statemennts
from tqdm import tqdm
preprocessed_resource_summary = []
# tqdm is for printing the status bar
for sentance in project_data['project_resource_summary'].values:
    sent = decontracted(sentance)
    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_resource_summary.append(sent.lower().strip())
```

In [0]:

```
!pip install autocorrect
```

```
Collecting autocorrect
  Downloading
https://files.pythonhosted.org/packages/ec/b6/6c74ff19249dc6d7285541cd59f5a3edbbd0f7209362a63e314fc
636/autocorrect-0.3.0.tar.gz (3.6MB)
    100% |████████████████████████████████| 3.6MB 9.1MB/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]:

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 [198]:

```
## 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', 'K
S', 'KY', 'LA', 'MA', 'MD', 'ME', 'MI', 'MN', 'MO', 'MS', 'MT', 'NC', 'ND', 'NE', 'NH', 'NJ', 'NM',
'NV', 'NY', 'OH', 'OK', 'OR', 'PA', 'RI', 'SC', 'SD', 'TN', 'TX', 'UT', 'VA', 'VT', 'WA', 'WI', 'WV
', 'WY']
Shape of matrix after one hot encodig  (109248, 51)
```

Clean_Categories-One-Hot-Encoding

In [199]:

```
# 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', 'K
S', 'KY', 'LA', 'MA', 'MD', 'ME', 'MI', 'MN', 'MO', 'MS', 'MT', 'NC', 'ND', 'NE', 'NH', 'NJ', 'NM',
'NV', 'NY', 'OH', 'OK', 'OR', 'PA', 'RI', 'SC', 'SD', 'TN', 'TX', 'UT', 'VA', 'VT', 'WA', 'WI', 'WV
', 'WY', 'Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning',
'SpecialNeeds', 'Health_Sports', 'Math_Science', 'Literacy_Language']
```

Clean_SubCategories

In [200]:

```
# 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 [201]:

```
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 [202]:

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

Out[202]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime
7820	17809	p180947	834f75f1b5e24bd10abe9c3dbf7ba12f	NaN	CA	2016-11-04 00:15:45
30368	22174	p002730	339bd5a9e445d68a74d65b99cd325397	NaN	SC	2016-05-09 09:38:40
57654	158692	p197901	e4be6aaaa887d4202df2b647fbfc82bb	NaN	PA	2016-06-03 10:15:05

3 rows × 22 columns

We contain nan values in the teacher_prefix column

In [203]:

```
#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[203]:

```
False
```

replaced nan values in teacher_prefix with "Mrs." as Mrs. is majority vote

In [204]:

```
# 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=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)

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 maen 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 = 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)

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 maen 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 = 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)

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 maen 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=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(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 th
is 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 maen 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 maen 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 maen 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 maen 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 concatinating a sparse matrix and a dense matirx :)
X_cn = hstack((school_state_one_hot[:69999],categories_one_hot[:69999],sub_categories_one_hot[:699
99],project_grade_one_hot[:69999],teacher_prefix_one_hot[:69999],price_standardized[:69999],teache
r_number_of_previously_posted_projects_standardized[:69999],quantity_standardised[:69999],digits_s
tandardised[:69999],essays_words_standardised[:69999],summary_words_standardised[:69999],titles_wo
rds_standardised[:69999],neutrality_polarity_standardised[:69999],compound_polarity_standardised[:
69999],positive_polarity_standardised[:69999],negative_polarity_standardised[:69999],spellingmista
ke_essays[:69999],spellingmistake_titles[:69999],spellingmistake_resource[:69999],similarity_stand
rdised[: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]:

```
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 [222]:

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

Out[222]:

```
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]:

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

1.1Vectorizers 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 [226]:

```
# 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 [234]:

```
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

1.2Extract train,test of only numerical and categorical features

In [235]:

```
import scipy
X_train_cn =
X_train.drop(['y','preprocessed_essays','preprocessed_titles','preprocessed_resource_summary'],axi
s=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 [238]:

```
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 gievn to the Logistic-Regression should be of Sparse Matrix and Not Dataframe as DF taes more time to Run.

1.4-Applying 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
```

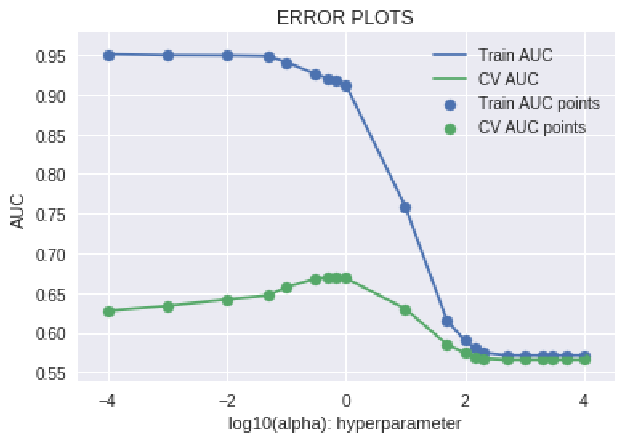
1.5-AUC with trainset and CV-set using Dataset after CV-spliting

In [240]:

```
train_auc = []
cv_auc = []

alpha_values =[0.0001,0.001,0.01,0.05,0.1,0.3,0.5,0.7,1,10,50,100,150,200,500,1000,2000,3000,5000,1
0000]

for i in tqdm(alpha_values):


  model_Logistic = SGDClassifier(loss='hinge',alpha=i,penalty='l2')
  model_Logistic.fit(X_train_bow, y_train)
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive 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.decision_function(X_train_bow[k:k+1000]))

  y_cv_pred = []


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


    y_cv_pred.extend(model_Logistic.decision_function(X_cv_bow[k:k+500]))



  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()
```

```
  0%|          | 0/20 [00:00<?, ?it/s]
  5%|▌         | 1/20 [00:00<00:06,  2.92it/s]
 10%|█         | 2/20 [00:00<00:06,  2.92it/s]
 15%|█▌        | 3/20 [00:01<00:05,  2.94it/s]
 20%|██        | 4/20 [00:01<00:05,  2.90it/s]
 25%|██▌       | 5/20 [00:01<00:04,  3.01it/s]
 30%|███       | 6/20 [00:02<00:04,  2.94it/s]
 35%|███▌      | 7/20 [00:02<00:04,  3.01it/s]
 40%|████      | 8/20 [00:02<00:03,  3.04it/s]
 45%|████▌     | 9/20 [00:02<00:03,  3.09it/s]
 50%|█████     | 10/20 [00:03<00:03,  3.12it/s]
 55%|█████▌    | 11/20 [00:03<00:02,  3.06it/s]
 60%|██████    | 12/20 [00:03<00:02,  2.99it/s]
 65%|██████▌   | 13/20 [00:04<00:02,  2.92it/s]
 70%|███████   | 14/20 [00:04<00:02,  2.82it/s]
 75%|███████▌  | 15/20 [00:05<00:01,  2.76it/s]
 80%|████████  | 16/20 [00:05<00:01,  2.71it/s]
 85%|████████▌ | 17/20 [00:05<00:01,  2.67it/s]
 90%|█████████ | 18/20 [00:06<00:00,  2.64it/s]
 95%|█████████▌| 19/20 [00:06<00:00,  2.62it/s]
100%|██████████| 20/20 [00:07<00:00,  2.61it/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 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. Claculated 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

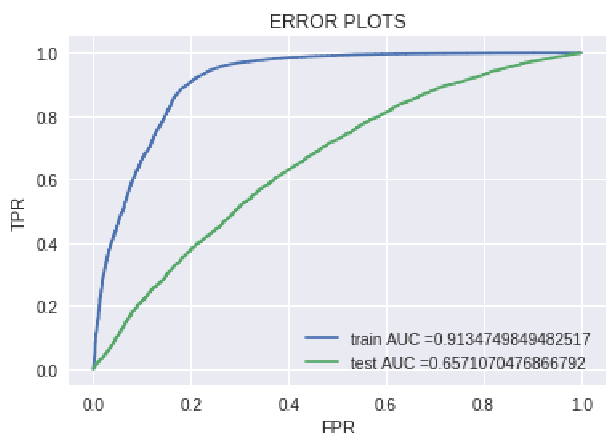
8.We are plotting log(alpha) because we need to accumodate large and small values of alpha

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 [241]:

```
from sklearn.metrics import roc_curve, auc

optimal_alpha = 1
model_Logistic_bow = SGDClassifier(loss='hinge',alpha=optimal_alpha,penalty='l2')
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.decision_function(X_train_bow[k:k+100]))

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

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()
```



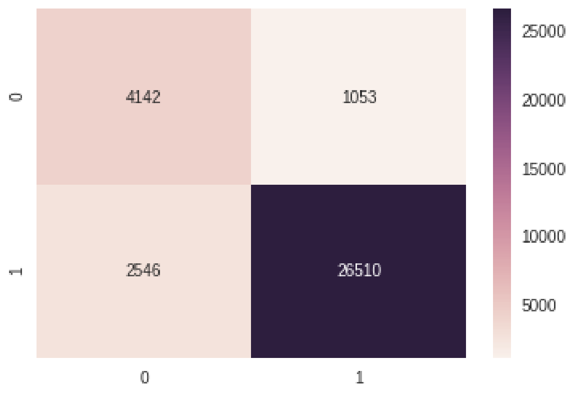
In [242]:

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

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

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

    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,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) 0.7274421196677592 for threshold 0.98
```

Out[242]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb1fcdf0780>
```



In [243]:

```
y_train.value_counts()
```

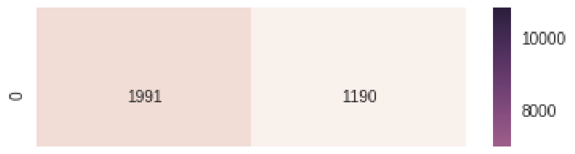
Out[243]:

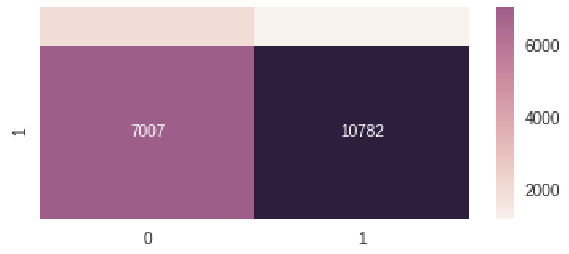
```
1    29056
0     5195
Name: project_is_approved, dtype: int64
```

In [244]:

```
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) 0.3793633601074767 for threshold 1.08
AxesSubplot(0.125,0.125;0.62x0.755)
```





In [245]:

```
y_test.value_counts()
```

Out[245]:

```
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 trian 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-lear
n-classifiers

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

In [247]:

```
top_20_features
```

Out[247]:

```
[(0.0480254184757742, 'my'),
 (0.023075633248609655, 'nannan'),
 (0.015247024745729925, 'need'),
 (0.010997370053558839, 'students'),
 (0.007900627686136892, 'Mrs'),
 (0.007897722115532132, 'use'),
 (0.007215223500671257, 'students'),
 (0.007152386164926986, 'previously_posted_projects'),
 (0.006765025830071498, 'allow'),
 (0.006640541560948006, 'used'),
 (0.0064722064601951635, 'WA'),
 (0.0064526159217937525, 'Literacy'),
 (0.006445607667825487, 'balls'),
 (0.006094689657004158, 'kits'),
 (0.006070669636555577, 'storage'),
 (0.005994798689852919, 'the'),
 (0.005971259708067659, 'Literacy_Language'),
 (0.005911014379005443, 'carpet'),
 (0.0058079154287679615, 'Grades 3-5'),
 (0.005791518057697166, 'using')]
```

2-TFIDF

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

In [248]:

```
# 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_scalar = StandardScaler(with_mean = False)
X_train_tfidf = train_scalar.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_scalar = StandardScaler(with_mean = False)
X_test_tfidf = test_scalar.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_scalar = StandardScaler(with_mean = False)
X_cv_tfidf = cv_scalar.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

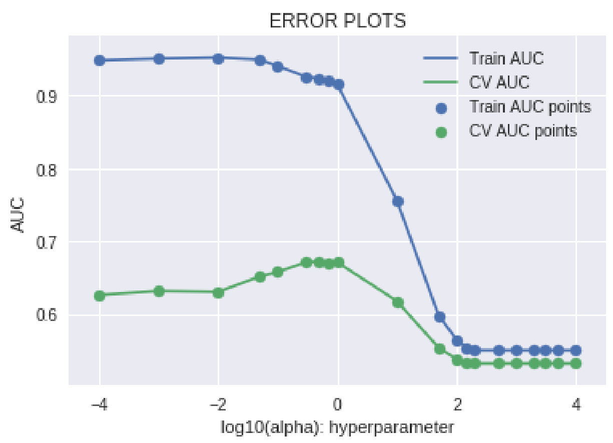
Convert COO-matrix to CSR-Sparse matrix as the input given to the Logistic-Regression should be of Sparse Matrix and Not Dataframe

2.3-AUC with trainset and CV-set using Dataset after CV-spliting

In [258]:

```
train_auc = []
cv_auc = []

alpha_values =[0.0001,0.001,0.01,0.05,0.1,0.3,0.5,0.7,1,10,50,100,150,200,500,1000,2000,3000,5000,1
0000]


for i in tqdm(alpha_values):
    model_logistic_tfidf = SGDClassifier(loss='hinge',alpha=i,penalty='l2')
    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 posi
tive 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.decision_function(X_train_tfidf[k:k+1000]))

    y_cv_pred = []


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



    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()
```

```
  0%|          | 0/20 [00:00<?, ?it/s]

  5%|▌         | 1/20 [00:00<00:05,  3.35it/s]

 10%|█         | 2/20 [00:00<00:05,  3.35it/s]
 10%|█         | 2/20 [00:00<00:05,  3.35it/s]
 15%|█▌        | 3/20 [00:00<00:05,  3.36it/s]
 20%|██        | 4/20 [00:01<00:04,  3.30it/s]
 25%|██▌       | 5/20 [00:01<00:04,  3.29it/s]
 30%|███       | 6/20 [00:01<00:04,  3.26it/s]
 35%|███▌      | 7/20 [00:02<00:04,  3.20it/s]
 40%|████      | 8/20 [00:02<00:03,  3.21it/s]
 45%|████▌     | 9/20 [00:02<00:03,  3.20it/s]
 50%|█████     | 10/20 [00:03<00:03,  3.16it/s]
 55%|█████▌    | 11/20 [00:03<00:02,  3.08it/s]
 60%|██████    | 12/20 [00:03<00:02,  2.97it/s]
 65%|██████▌   | 13/20 [00:04<00:02,  2.85it/s]
 70%|███████   | 14/20 [00:04<00:02,  2.76it/s]
 75%|███████▌  | 15/20 [00:04<00:01,  2.70it/s]
 80%|████████  | 16/20 [00:05<00:01,  2.66it/s]
 85%|████████▌ | 17/20 [00:05<00:01,  2.66it/s]
 90%|█████████ | 18/20 [00:06<00:00,  2.64it/s]
 95%|█████████▌| 19/20 [00:06<00:00,  2.63it/s]
100%|██████████| 20/20 [00:06<00:00,  2.65it/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. Claculated 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

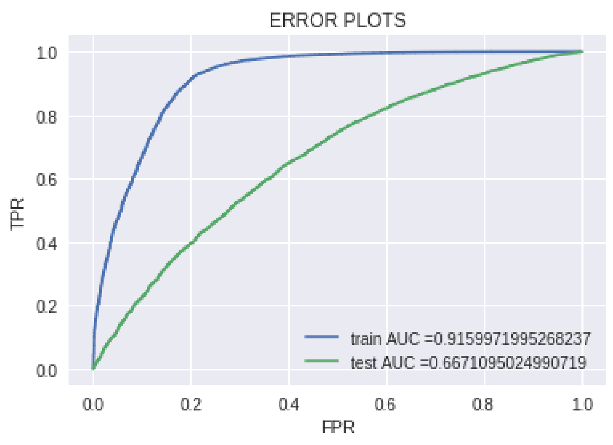
regarding the misclassified labels,SO use AUC as the best metric

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

2.5-ROC-Curve with optimal_alpha for train and test-sets

In [259]:

```
from sklearn.metrics import roc_curve, auc

optimal_alpha =1
model_logistic_tfidf = SGDClassifier(loss='hinge',alpha=optimal_alpha,penalty='l2')
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.decision_function(X_train_tfidf[k:k+1000]))

y_test_pred = []


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

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 [260]:

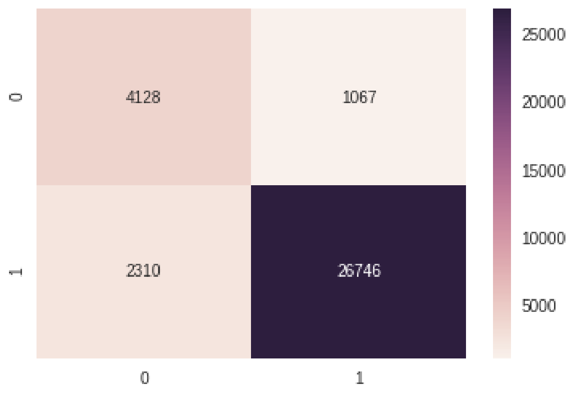
```
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def predict(proba, threshould, fpr, tpr):

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

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

    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,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) 0.7314373783670337 for threshold 0.978
```

Out[260]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb1d2d72320>
```



In [261]:

```
y_train.value_counts()
```

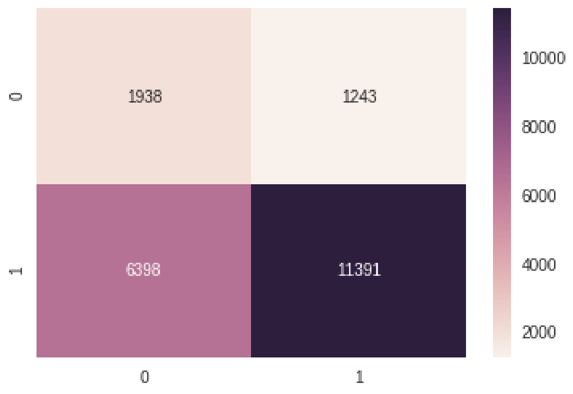
Out[261]:

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

In [262]:

```
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) 0.39012198054850555 for threshold 1.063
AxesSubplot(0.125,0.125;0.62x0.755)
```



In [263]:

```
y_test.value_counts()
```

Out[263]:

```
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-lear
n-classifiers

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

In [265]:

```
top_20_features
```

Out[265]:

```
[(0.01914783238580652, 'nannan'),
 (0.015106810971058381, 'my'),
 (0.012825674116640692, 'students'),
 (0.010022441246618104, 'students'),
 (0.009955762102341479, 'need'),
 (0.009451403123486711, 'use'),
 (0.008866518632974708, 'the'),
 (0.008613554148939214, 'Mrs'),
 (0.00800717785314913, 'allow'),
 (0.007533838623711967, 'used'),
 (0.007436763406283051, 'books'),
 (0.0073538731610499125, 'kits'),
 (0.007329707739790897, 'using'),
 (0.007318383306622204, 'balls'),
 (0.007142532770398568, 'previously_posted_projects'),
 (0.0068038580216761655, 'sand'),
 (0.00665386710064112, 'Literacy'),
 (0.006583124479641601, 'Grades 3-5'),
 (0.006557919044919102, 'school'),
 (0.006451806364604535, 'carpet')]
```

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 [0]:

```
# 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 [01:15<00:00, 456.05it/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 [0]:

```
# 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, 23599.69it/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 [0]:

```
# 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
    if 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 [00:47<00:00, 440.65it/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 [0]:

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

```
100%|██████████| 20970/20970 [00:00<00:00, 23932.25it/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 [0]:

```
# 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
    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):
            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.append(vector)

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

```
100%|██████████| 14679/14679 [00:33<00:00, 439.03it/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 [0]:

```
# 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
    tf_idf_weight_cv_titles =0; # num of words with a valid vector in the sentence/review
    tf_idf_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())))
            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_titles += tf_idf
    if tf_idf_weight_cv_titles != 0:
        vector /= tf_idf_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, 22990.01it/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 [0]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_train_summary = []; # the avg-w2v for each sentence/review is stored in this lis
t
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, 7782.41it/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 = 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 [0]:

```
# 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
            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, 7809.59it/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 [0]:

```
# 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, 7951.95it/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,tfid
f_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_summ
ary))
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 [0]:

```
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)
```

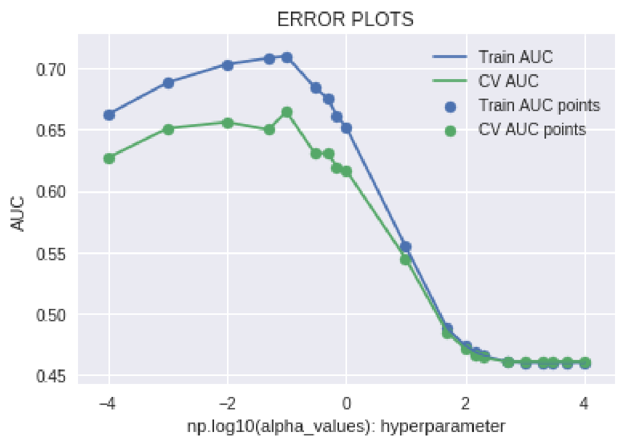
```
(34251, 1014) (20970, 1014) (14679, 1014)
```

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

In [0]:

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


train_auc = []
cv_auc = []
alpha_values =[0.0001,0.001,0.01,0.05,0.1,0.3,0.5,0.7,1,10,50,100,150,200,500,1000,2000,3000,5000,1
0000]

for i in tqdm(alpha_values):
    Model_tfidf_w2v = SGDClassifier(loss='hinge',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 posi
tive 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.decision_function(X_train_tfidf_w2v[k:k+1000]))

    y_cv_pred = []


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

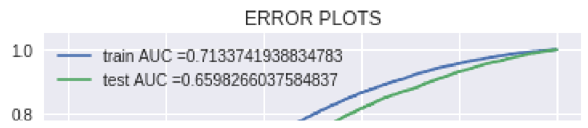
```
100%|██████████| 20/20 [00:24<00:00,  1.40s/it]
```

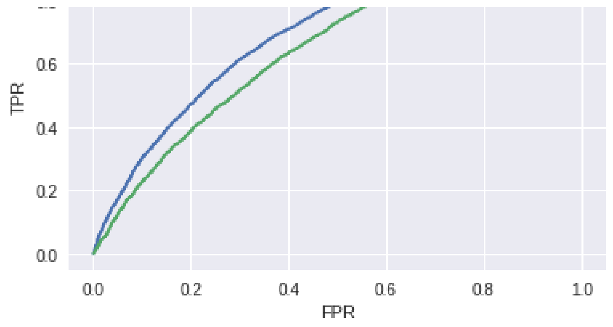


3.3-ROC-Curve with optimal_alpha for train and test-sets

In [0]:

```
from sklearn.metrics import roc_curve, auc

best_alpha = 0.1
model_tfidf_w2v = SGDClassifier(loss='hinge',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.decision_function(X_train_tfidf_w2v[k:k+1000]))

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



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()
```





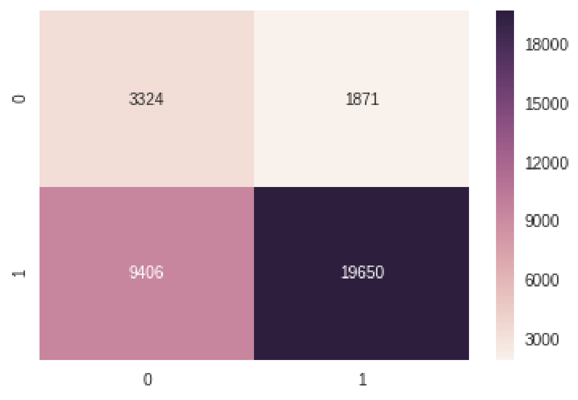
In [0]:

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

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

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

    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,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) 0.4327152400011872 for threshold 0.975
```

Out[0]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7feb63b6c588>
```



In [0]:

```
y_train.value_counts()
```

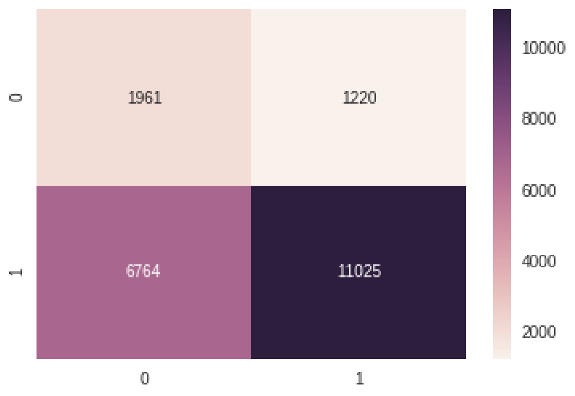
Out[0]:

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

In [0]:

```
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) 0.38206828379384317 for threshold 1.017
AxesSubplot(0.125,0.125;0.62x0.755)
```



In [0]:

```
y_test.value_counts()
```

Out[0]:

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

4-AVG-W2v

Train_Essay

In [0]:

```
# 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, 2959.48it/s]
```

```
34251
300
```

Train_titles

In [0]:

```
# 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, 56595.80it/s]
```

```
34251
300
```

Test_Essay

In [0]:

```
# 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, 2878.02it/s]
```

```
20970
300
```

Test_Titles

In [0]:

```
# 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, 56518.65it/s]
```

```
20970
300
```

CV_Essay

In [0]:

```python
# 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, 2907.67it/s]
```

```
14679
300
```

CV_Titles

In [0]:

```python
# 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, 53115.22it/s]
```

```
14679
300
```

Train_Summary

In [0]:

```python
# 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, 23532.02it/s]
```

```
34251
300
```

Test_Summary

In [0]:

```
# 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, 26316.51it/s]
```

```
20970
300
```

CV_Summary

In [0]:

```
# 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, 22598.28it/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_scalar = StandardScaler(with_mean = False)
X_train_avg_w2v = train_scalar.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]:

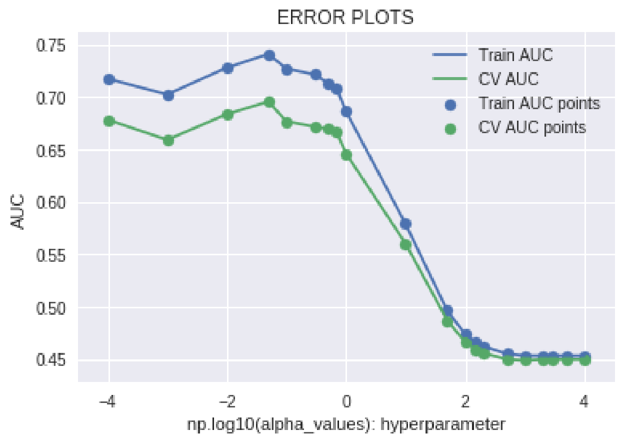
```
X_test_avg_w2v
=hstack((X_test_cn,avg_w2v_essays_vectors_test,avg_w2v_titles_vectors_test,avg_w2v_summary_vectors_
test))
X_test_avg_w2v = X_test_avg_w2v.tocsr()
test_scalar = StandardScaler(with_mean = False)
X_test_avg_w2v = test_scalar.fit_transform(X_test_avg_w2v)
```

4.2-AUC with trainset and CV-set using Dataset after CV-spliting

In [0]:

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


train_auc = []
cv_auc = []
alpha_values =[0.0001,0.001,0.01,0.05,0.1,0.3,0.5,0.7,1,10,50,100,150,200,500,1000,2000,3000,5000,1
0000]

for i in tqdm(alpha_values):
    model_avgw2v = SGDClassifier(loss='hinge',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 posi
tive 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.decision_function(X_train_avg_w2v[k:k+1000]))

    y_cv_pred = []


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

    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()
```

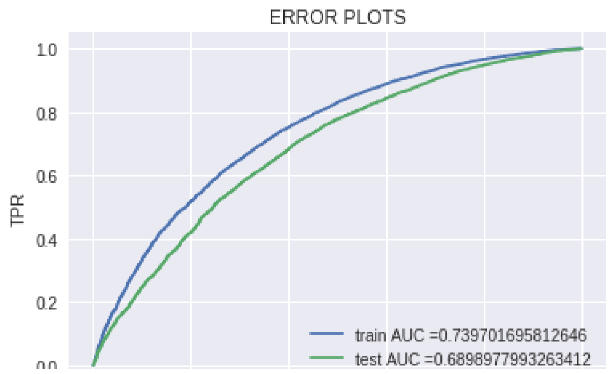
```
100%|██████████| 20/20 [00:24<00:00,  1.42s/it]
```



4.3-ROC-Curve with optimal_alpha for train and test-sets

In [0]:

```
from sklearn.metrics import roc_curve, auc

optimal_alpha =0.05
model_avgw2v = SGDClassifier(loss='hinge',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.decision_function(X_train_avg_w2v[k:k+1000]))

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

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()
```



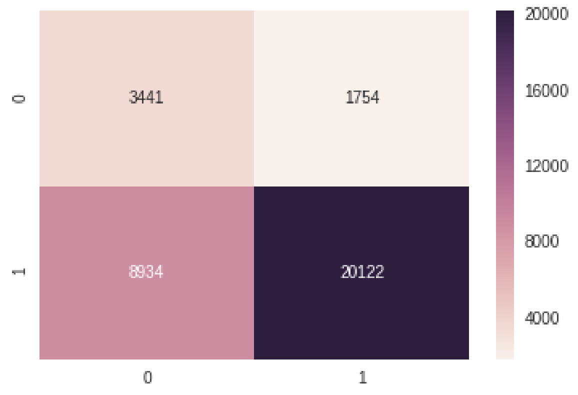
In [0]:

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

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

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

    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,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) 0.4587060186853676 for threshold 1.118
```

Out[0]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7feb633937b8>
```



In [0]:

```
y_train.value_counts()
```

Out[0]:

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

In [0]:

```
y_test.value_counts()
```

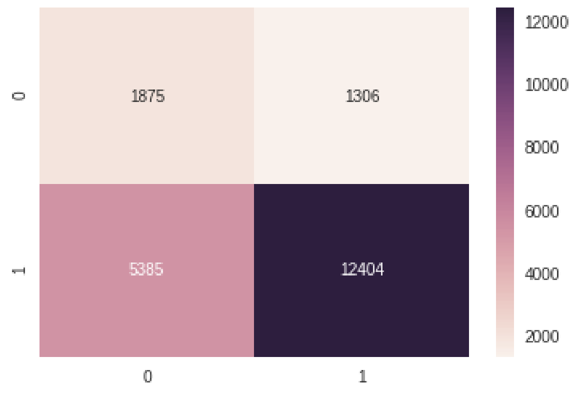
Out[0]:

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

In [0]:

```
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) 0.4110056815538053 for threshold 1.131
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'],axi
s=1)
k_essay = pd.DataFrame({'preprocessed_essays':preprocessed_essays,'y':y})
set5_dataset = pd.concat([X_cn,k_essay],axis=1)
```

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(set5_dataset[:69900], y[:69900], test_size=0.3,random_s
tate=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]:

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

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=(1,3),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)
```

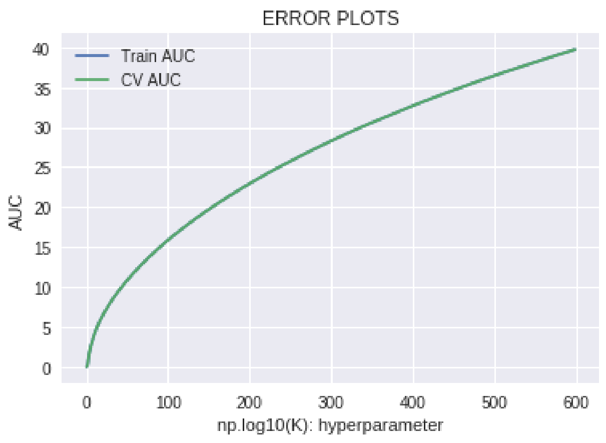
```
(34251, 10000) (14679, 10000) (20970, 10000)
```

In [0]:

```
import numpy as np
from sklearn.decomposition import TruncatedSVD
z = list(range(600))

explained_variances = []
for j in tqdm(range(600)):

  model = TruncatedSVD(n_components=j).fit(train_tfidf_essay)
  X_proj = model.transform(train_tfidf_essay)
  explained_variances.append(model.explained_variance_ratio_.sum() * 100)
```

In [106]:

```
plt.plot(z, explained_variances, label='Train AUC')
plt.plot(z, explained_variances, label='CV AUC')
plt.legend()
plt.xlabel("np.log10(K): hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



In [0]:

```
model = TruncatedSVD(n_components=1000).fit(train_tfidf_essay)
X_essay_tfidf_svd_train = model.transform(train_tfidf_essay)
X_essay_tfidf_svd_test = model.transform(test_tfidf_essay)
X_essay_tfidf_svd_cv = model.transform(cv_tfidf_essay)
```

In [123]:

```
import scipy
X_train_cn = X_train.drop(['y','preprocessed_essays'],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'],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'],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)
```

In [0]:

```
from scipy.sparse import hstack

X_train_tfidf = hstack((X_train_cn,X_essay_tfidf_svd_train))
X_train_tfidf = X_train_tfidf.tocsr()
train_scalar = StandardScaler(with_mean = False)
X_train_tfidf = train_scalar.fit_transform(X_train_tfidf)

X_test_tfidf =hstack((X_test_cn,X_essay_tfidf_svd_test))
X_test_tfidf = X_test_tfidf.tocsr()
test_scalar = StandardScaler(with_mean = False)
X_test_tfidf = test_scalar.fit_transform(X_test_tfidf)

X_cv_tfidf = hstack((X_cv_cn,X_essay_tfidf_svd_cv))
X_cv_tfidf  = X_cv_tfidf.tocsr()
cv_scalar = StandardScaler(with_mean = False)
X_cv_tfidf = cv_scalar.fit_transform(X_cv_tfidf)
```

In [125]:

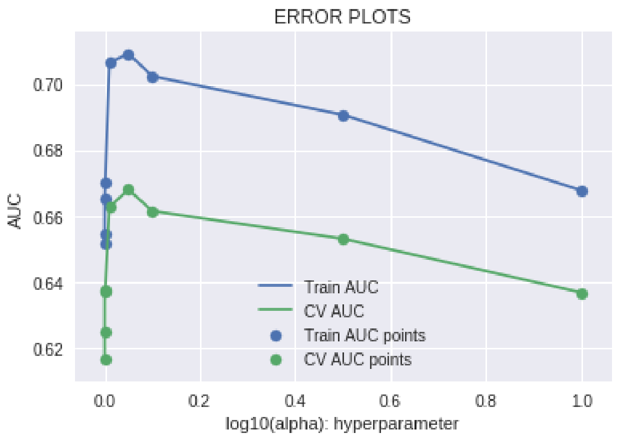
```
print(X_train_tfidf.shape,X_test_tfidf.shape,X_cv_tfidf.shape)
```

```
(34251, 1114) (20970, 1114) (14679, 1114)
```

In [128]:

```
train_auc = []
cv_auc = []

alpha_values = [0.000001,0.00001,0.0001,0.001,0.01,0.05,0.1,0.5,1]

for i in tqdm(alpha_values):


  model_Logistic = SGDClassifier(loss='hinge',alpha=i)
  model_Logistic.fit(X_train_tfidf, y_train)
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive 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.decision_function(X_train_tfidf[k:k+1000]))

  y_cv_pred = []


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


    y_cv_pred.extend(model_Logistic.decision_function(X_cv_tfidf[k:k+500]))



  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()
```

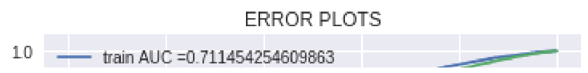
```
  0%|          | 0/9 [00:00<?, ?it/s]

 11%|█         | 1/9 [00:01<00:09,  1.15s/it]

 22%|██        | 2/9 [00:02<00:08,  1.18s/it]

 33%|███       | 3/9 [00:03<00:07,  1.18s/it]

 44%|████      | 4/9 [00:04<00:05,  1.16s/it]

 56%|█████     | 5/9 [00:05<00:04,  1.15s/it]

 67%|██████    | 6/9 [00:06<00:03,  1.15s/it]

 78%|███████   | 7/9 [00:08<00:02,  1.14s/it]

 89%|████████  | 8/9 [00:09<00:01,  1.15s/it]

100%|██████████| 9/9 [00:10<00:00,  1.15s/it]
```

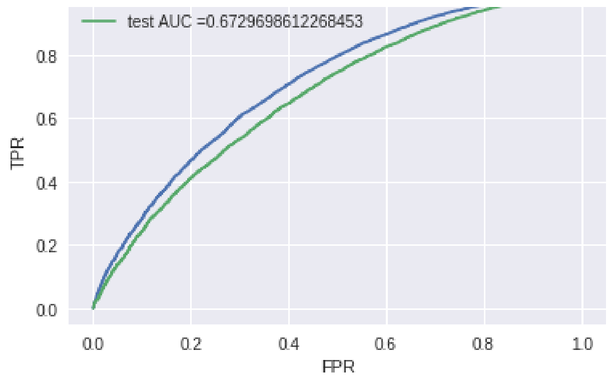


5.3-ROC-Curve with optimal_alpha for train and test-sets

In [129]:

```
from sklearn.metrics import roc_curve, auc

optimal_alpha = 0.05
model_Logistic_bow = SGDClassifier(loss='hinge',alpha=optimal_alpha)
model_Logistic_bow.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],100):
  y_train_pred.extend(model_Logistic_bow.decision_function(X_train_tfidf[k:k+100]))

y_test_pred = []
for k in range(0, X_test_tfidf.shape[0],100):
  y_test_pred.extend(model_Logistic_bow.decision_function(X_test_tfidf[k:k+100]))

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()
```





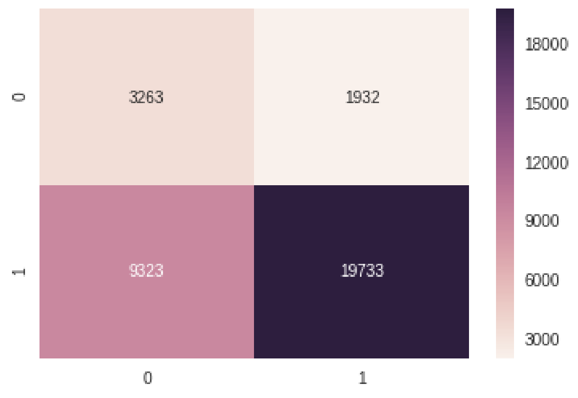
In [130]:

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

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

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

    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,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) 0.42656852864920103 for threshold 1.059
```

Out[130]:

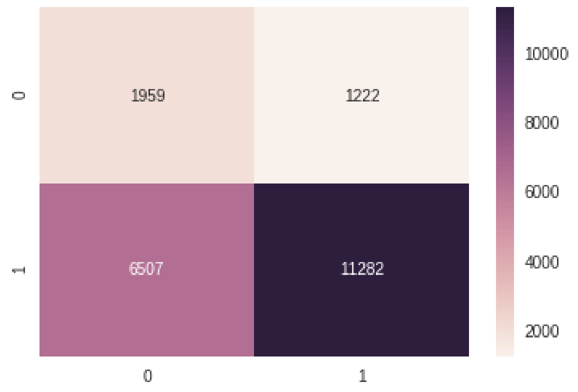
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb1feac39e8>
```



In [131]:

```
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) 0.3905757965606437 for threshold 1.068
AxesSubplot(0.125,0.125;0.62x0.755)
```



Feature-Importance

In [0]:

```
# Pretty-Table
```

In [267]:

```
import numpy as np
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Model", "alpha", "Train-AUC","Test-AUC"]


x.add_row(["BOW",1,0.913,0.657])
x.add_row(["TFidf",1,0.915,0.667])
x.add_row(["TFidf-W2V",0.1,0.713,0.659])
x.add_row(["AVG-W2V",0.1,0.739,0.689])
x.add_row(["SET-5",0.05,0.711,0.672])
print(x)
```

```
+-----------+-------+-----------+----------+
|   Model   | alpha | Train-AUC | Test-AUC |
+-----------+-------+-----------+----------+
|    BOW    |   1   |   0.913   |  0.657   |
|   TFidf   |   1   |   0.915   |  0.667   |
| TFidf-W2V |  0.1  |   0.713   |  0.659   |
|  AVG-W2V  |  0.1  |   0.739   |  0.689   |
|   SET-5   |  0.05 |   0.711   |  0.672   |
+-----------+-------+-----------+----------+
```