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	Desc
`project_id`	A unique identifier for the proposed project. **Example:** `p0`
`project_title`	Title of the project. **Exa** • `Art Will Make You H` • `First Grad`
`project_grade_category`	Grade level of students for which the project is targeted. One of the fo enumerated v • `Grades P` • `Grade` • `Grade` • `Grades`
`project_subject_categories`	One or more (comma-separated) subject categories for the project fr following enumerated list of v • `Applied Lea` • `Care & H` • `Health & S` • `History & C` • `Literacy & Lan` • `Math & Sc` • `Music & The` • `Special` • `W` **Exar** • `Music & The` • `Literacy & Language, Math & Sc`

Feature	Desc
`school_state`	State where school is located (Two-letter U.S. posta (https://en.wikipedia.org/wiki/List_of_U.S._state_abbreviations#Postal_c **Exampl**
`project_subject_subcategories`	One or more (comma-separated) subject subcategories for the p **Exan** • Lit • Literature & Writing, Social Sci
`project_resource_summary`	An explanation of the resources needed for the project. **Exa** • My students need hands on literacy materials to ma sensory needs!<
`project_essay_1`	First application
`project_essay_2`	Second application
`project_essay_3`	Third application
`project_essay_4`	Fourth application
`project_submitted_datetime`	Datetime when project application was submitted. **Example:** 2016-0 12:43:5
`teacher_id`	A unique identifier for the teacher of the proposed project. **Exa** bdf8baa8fedef6bfeec7ae4ff1c
`teacher_prefix`	Teacher's title. One of the following enumerated v • • • • • • Tea
`teacher_number_of_previously_posted_projects`	Number of project applications previously submitted by the same te **Exam**

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

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

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

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

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

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

Notes on the Essay Data

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

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

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

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

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

In [2]:

```
import sys
!{sys.executable} -m pip install gensim
```

```
Collecting gensim
  Downloading https://files.pythonhosted.org/packages/ef/65/c90886ac34d4b12d
3ae0bcc7aece1af57e1f30e7138aabbb3e3c027e705a/gensim-3.8.0-cp35-cp35m-manylin
ux1_x86_64.whl (https://files.pythonhosted.org/packages/ef/65/c90886ac34d4b1
2d3ae0bcc7aece1af57e1f30e7138aabbb3e3c027e705a/gensim-3.8.0-cp35-cp35m-manyl
inux1_x86_64.whl) (24.2MB)
    100% |████████████████████████████████| 24.2MB 60kB/s  eta 0:00:01
Collecting scipy>=0.18.1 (from gensim)
  Downloading https://files.pythonhosted.org/packages/7a/0e/3781e028d62a8422
244582abd8f084e6314297026760587c85607f687bf3/scipy-1.3.1-cp35-cp35m-manylinu
x1_x86_64.whl (https://files.pythonhosted.org/packages/7a/0e/3781e028d62a842
2244582abd8f084e6314297026760587c85607f687bf3/scipy-1.3.1-cp35-cp35m-manylin
ux1_x86_64.whl) (25.1MB)
    100% |████████████████████████████████| 25.1MB 58kB/s  eta 0:00:01
Collecting smart-open>=1.7.0 (from gensim)
  Downloading https://files.pythonhosted.org/packages/37/c0/25d19badc495428d
ec6a4bf7782de617ee0246a9211af75b302a2681dea7/smart_open-1.8.4.tar.gz (http
s://files.pythonhosted.org/packages/37/c0/25d19badc495428dec6a4bf7782de617ee
0246a9211af75b302a2681dea7/smart_open-1.8.4.tar.gz) (63kB)
    100% |████████████████████████████████| 71kB 11.8MB/s ta 0:00:01
Collecting six>=1.5.0 (from gensim)
  Downloading https://files.pythonhosted.org/packages/73/fb/00a976f728d0d1fe
cfe898238ce23f502a721c0ac0ecfedb80e0d88c64e9/six-1.12.0-py2.py3-none-any.whl
(https://files.pythonhosted.org/packages/73/fb/00a976f728d0d1fecfe898238ce23
f502a721c0ac0ecfedb80e0d88c64e9/six-1.12.0-py2.py3-none-any.whl)
Collecting numpy>=1.11.3 (from gensim)
  Downloading https://files.pythonhosted.org/packages/69/25/eef8d362bd216b11
e7d005331a3cca3d19b0aa57569bde680070109b745c/numpy-1.17.0-cp35-cp35m-manylin
ux1_x86_64.whl (https://files.pythonhosted.org/packages/69/25/eef8d362bd216b
11e7d005331a3cca3d19b0aa57569bde680070109b745c/numpy-1.17.0-cp35-cp35m-manyl
inux1_x86_64.whl) (20.2MB)
    100% |████████████████████████████████| 20.2MB 69kB/s  eta 0:00:01
Collecting boto>=2.32 (from smart-open>=1.7.0->gensim)
  Downloading https://files.pythonhosted.org/packages/23/10/c0b78c27298029e4
454a472a1919bde20cb182dab1662cec7f2ca1dcc523/boto-2.49.0-py2.py3-none-any.wh
l (https://files.pythonhosted.org/packages/23/10/c0b78c27298029e4454a472a191
9bde20cb182dab1662cec7f2ca1dcc523/boto-2.49.0-py2.py3-none-any.whl) (1.4MB)
    100% |████████████████████████████████| 1.4MB 1.1MB/s eta 0:00:01
Collecting requests (from smart-open>=1.7.0->gensim)
  Downloading https://files.pythonhosted.org/packages/51/bd/23c926cd341ea6b7
dd0b2a00aba99ae0f828be89d72b2190f27c11d4b7fb/requests-2.22.0-py2.py3-none-an
y.whl (https://files.pythonhosted.org/packages/51/bd/23c926cd341ea6b7dd0b2a0
0aba99ae0f828be89d72b2190f27c11d4b7fb/requests-2.22.0-py2.py3-none-any.whl)
 (57kB)
    100% |████████████████████████████████| 61kB 11.1MB/s ta 0:00:01
Collecting boto3 (from smart-open>=1.7.0->gensim)
  Downloading https://files.pythonhosted.org/packages/ff/3e/2262936ad70fd6e7
b8827d79d6508ce33e2ffb49bfca6fedc5fe4abd6f1c/boto3-1.9.215-py2.py3-none-any.
whl (https://files.pythonhosted.org/packages/ff/3e/2262936ad70fd6e7b8827d79d
6508ce33e2ffb49bfca6fedc5fe4abd6f1c/boto3-1.9.215-py2.py3-none-any.whl) (128
kB)
    100% |████████████████████████████████| 133kB 10.6MB/s ta 0:00:01
Collecting chardet<3.1.0,>=3.0.2 (from requests->smart-open>=1.7.0->gensim)
  Downloading https://files.pythonhosted.org/packages/bc/a9/01ffebfb562e4274
b6487b4bb1ddec7ca55ec7510b22e4c51f14098443b8/chardet-3.0.4-py2.py3-none-any.
whl (https://files.pythonhosted.org/packages/bc/a9/01ffebfb562e4274b6487b4bb
```

```
1ddec7ca55ec7510b22e4c51f14098443b8/chardet-3.0.4-py2.py3-none-any.whl) (133
kB)
    100% |████████████████████████████████| 143kB 9.9MB/s eta 0:00:01
Collecting urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 (from requests->smart-ope
n>=1.7.0->gensim)
  Downloading https://files.pythonhosted.org/packages/e6/60/247f23a7121ae632
d62811ba7f273d0e58972d75e58a94d329d51550a47d/urllib3-1.25.3-py2.py3-none-an
y.whl (https://files.pythonhosted.org/packages/e6/60/247f23a7121ae632d62811b
a7f273d0e58972d75e58a94d329d51550a47d/urllib3-1.25.3-py2.py3-none-any.whl)
 (150kB)
    100% |████████████████████████████████| 153kB 8.8MB/s eta 0:00:01
Collecting certifi>=2017.4.17 (from requests->smart-open>=1.7.0->gensim)
  Downloading https://files.pythonhosted.org/packages/69/1b/b853c7a9d4f6a6d0
0749e94eb6f3a041e342a885b87340b79c1ef73e3a78/certifi-2019.6.16-py2.py3-none-
any.whl (https://files.pythonhosted.org/packages/69/1b/b853c7a9d4f6a6d00749e
94eb6f3a041e342a885b87340b79c1ef73e3a78/certifi-2019.6.16-py2.py3-none-any.w
hl) (157kB)
    100% |████████████████████████████████| 163kB 8.3MB/s eta 0:00:01
Collecting idna<2.9,>=2.5 (from requests->smart-open>=1.7.0->gensim)
  Downloading https://files.pythonhosted.org/packages/14/2c/cd551d81dbe15200
be1cf41cd03869a46fe7226e7450af7a6545bfc474c9/idna-2.8-py2.py3-none-any.whl
 (https://files.pythonhosted.org/packages/14/2c/cd551d81dbe15200be1cf41cd038
69a46fe7226e7450af7a6545bfc474c9/idna-2.8-py2.py3-none-any.whl) (58kB)
    100% |████████████████████████████████| 61kB 10.3MB/s ta 0:00:01
Collecting botocore<1.13.0,>=1.12.215 (from boto3->smart-open>=1.7.0->gensi
m)
  Downloading https://files.pythonhosted.org/packages/a1/b0/7a8794d914b95ef3
335a5a4ba20595b46081dbd1e29f13812eceacf091ca/botocore-1.12.215-py2.py3-none-
any.whl (https://files.pythonhosted.org/packages/a1/b0/7a8794d914b95ef3335a5
a4ba20595b46081dbd1e29f13812eceacf091ca/botocore-1.12.215-py2.py3-none-any.w
hl) (5.7MB)
    100% |████████████████████████████████| 5.7MB 270kB/s eta 0:00:01
Collecting s3transfer<0.3.0,>=0.2.0 (from boto3->smart-open>=1.7.0->gensim)
  Downloading https://files.pythonhosted.org/packages/16/8a/1fc3dba0c4923c2a
76e1ff0d52b305c44606da63f718d14d3231e21c51b0/s3transfer-0.2.1-py2.py3-none-a
ny.whl (https://files.pythonhosted.org/packages/16/8a/1fc3dba0c4923c2a76e1ff
0d52b305c44606da63f718d14d3231e21c51b0/s3transfer-0.2.1-py2.py3-none-any.wh
l) (70kB)
    100% |████████████████████████████████| 71kB 9.9MB/s eta 0:00:01
Collecting jmespath<1.0.0,>=0.7.1 (from boto3->smart-open>=1.7.0->gensim)
  Downloading https://files.pythonhosted.org/packages/83/94/7179c3832a6d45b2
66ddb2aac329e101367fbdb11f425f13771d27f225bb/jmespath-0.9.4-py2.py3-none-an
y.whl (https://files.pythonhosted.org/packages/83/94/7179c3832a6d45b266ddb2a
ac329e101367fbdb11f425f13771d27f225bb/jmespath-0.9.4-py2.py3-none-any.whl)
Collecting python-dateutil<3.0.0,>=2.1; python_version >= "2.7" (from botoco
re<1.13.0,>=1.12.215->boto3->smart-open>=1.7.0->gensim)
  Downloading https://files.pythonhosted.org/packages/41/17/c62faccbfbd163c7
f57f3844689e3a78bae1f403648a6afb1d0866d87fbb/python_dateutil-2.8.0-py2.py3-n
one-any.whl (https://files.pythonhosted.org/packages/41/17/c62faccbfbd163c7f
57f3844689e3a78bae1f403648a6afb1d0866d87fbb/python_dateutil-2.8.0-py2.py3-no
ne-any.whl) (226kB)
    100% |████████████████████████████████| 235kB 5.9MB/s eta 0:00:01
Collecting docutils<0.16,>=0.10 (from botocore<1.13.0,>=1.12.215->boto3->sma
rt-open>=1.7.0->gensim)
  Downloading https://files.pythonhosted.org/packages/22/cd/a6aa959dca619918
ccb55023b4cb151949c64d4d5d55b3f4ffd7eee0c6e8/docutils-0.15.2-py3-none-any.wh
l (https://files.pythonhosted.org/packages/22/cd/a6aa959dca619918ccb55023b4c
b151949c64d4d5d55b3f4ffd7eee0c6e8/docutils-0.15.2-py3-none-any.whl) (547kB)
    100% |████████████████████████████████| 552kB 2.7MB/s eta 0:00:01
Building wheels for collected packages: smart-open
  Running setup.py bdist_wheel for smart-open ... done
```

```
  Stored in directory: /home/shanud6711/.cache/pip/wheels/5f/ea/fb/5b1a947b3
69724063b2617011f1540c44eb00e28c3d2ca8692
Successfully built smart-open
Installing collected packages: numpy, scipy, boto, chardet, urllib3, certif
i, idna, requests, six, python-dateutil, jmespath, docutils, botocore, s3tra
nsfer, boto3, smart-open, gensim
Successfully installed boto-2.49.0 boto3-1.9.215 botocore-1.12.215 certifi-2
019.6.16 chardet-3.0.4 docutils-0.15.2 gensim-3.8.0 idna-2.8 jmespath-0.9.4
 numpy-1.17.0 python-dateutil-2.8.0 requests-2.22.0 s3transfer-0.2.1 scipy-
1.3.1 six-1.12.0 smart-open-1.8.4 urllib3-1.25.3
```

In [5]:

```
%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
```

1.1 Reading Data

In [6]:

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

In [7]:

```
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' 's
chool_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 [8]:

```
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[8]:

	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 [9]:

```
# how to replace elements in list python: https://stackoverflow.com/a/2582163/4084039
cols = ['Date' if x=='project_submitted_datetime' else x for x in list(project_data.columns


#sort dataframe based on time pandas python: https://stackoverflow.com/a/49702492/4084039
project_data['Date'] = pd.to_datetime(project_data['project_submitted_datetime'])
project_data.drop('project_submitted_datetime', axis=1, inplace=True)
project_data.sort_values(by=['Date'], inplace=True)


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


project_data.head(2)
```

Out[9]:

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

In [10]:

```
project_grade_category = []

for i in range(len(project_data)):
    a = project_data["project_grade_category"][i].replace(" ", "_")
    project_grade_category.append(a)

project_data.drop(['project_grade_category'], axis=1, inplace=True)
project_data["project_grade_category"] = project_grade_category
project_data.head(5)
```

Out[10]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	
55660	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	CA	00:
76127	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	UT	00:
51140	74477	p189804	4a97f3a390bfe21b99cf5e2b81981c73	Mrs.	CA	00:
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	GA	00:
41558	33679	p137682	06f6e62e17de34fcf81020c77549e1d5	Mrs.	WA	01:

1.2 preprocessing of project_subject_categories

In [11]:

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

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

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

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

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

1.3 preprocessing of project_subject_subcategories

In [12]:

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

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

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

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

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

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

Clean Titles (Text preprocessing)

In [13]:

```
# 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'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they'
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'l
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had',
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'u
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'd
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over',
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', '
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'v
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now',
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'do
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn',
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn'
            'won', "won't", 'wouldn', "wouldn't"]
```

In [14]:

```
# 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 [15]:

```
clean_titles = []

for titles in tqdm(project_data["project_title"]):
    title = decontracted(titles)
    title = title.replace('\\r', ' ')
    title = title.replace('\\"', ' ')
    title = title.replace('\\n', ' ')
    title = re.sub('[^A-Za-z0-9]+', ' ', title)
    title = ' '.join(f for f in title.split() if f not in stopwords)
    clean_titles.append(title.lower().strip())
```

```
100%|██████████| 109248/109248 [00:03<00:00, 35383.16it/s]
```

In [16]:

```
project_data["clean_titles"] = clean_titles
```

In [17]:

```
project_data.drop(['project_title'], axis=1, inplace=True)
```

Feature "Number of Words in Title"

In [18]:

```
title_word_count = []
for a in project_data["clean_titles"] :
    b = len(a.split())
    title_word_count.append(b)

project_data["title_word_count"] = title_word_count
project_data.head(5)
```

Out[18]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	
55660	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	CA	00:
76127	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	UT	00:
51140	74477	p189804	4a97f3a390bfe21b99cf5e2b81981c73	Mrs.	CA	00:
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	GA	00:
41558	33679	p137682	06f6e62e17de34fcf81020c77549e1d5	Mrs.	WA	01:

1.3 Text preprocessing

In [19]:

```
# 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 [20]:

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

Out[20]:

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

Clean Essays (Text preprocessing)

In [21]:

```
clean_essay = []

for ess in tqdm(project_data["essay"]):
    ess = decontracted(ess)
    ess = ess.replace('\\r', ' ')
    ess = ess.replace('\\"', ' ')
    ess = ess.replace('\\n', ' ')
    ess = re.sub('[^A-Za-z0-9]+', ' ', ess)
    ess = ' '.join(f for f in ess.split() if f not in stopwords)
    clean_essay.append(ess.lower().strip())
```

100%|██████████| 109248/109248 [01:11<00:00, 1528.15it/s]

In [22]:

```
project_data["clean_essays"] = clean_essay
```

In [23]:

```
project_data.drop(['essay'], axis=1, inplace=True)
```

Number of Words in Essay

In [24]:

```
essay_word_count = []
for ess in project_data["clean_essays"] :
    c = len(ess.split())
    essay_word_count.append(c)

project_data["essay_word_count"] = essay_word_count

project_data.head(5)
```

Out[24]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	
55660	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	CA	00:
76127	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	UT	00:
51140	74477	p189804	4a97f3a390bfe21b99cf5e2b81981c73	Mrs.	CA	00:
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	GA	00:
41558	33679	p137682	06f6e62e17de34fcf81020c77549e1d5	Mrs.	WA	01:

1.9 Calculate Sentiment Scores for the essays

In [26]:

```
import nltk
```

In [27]:

```
nltk.download('vader_lexicon')
```

```
[nltk_data] Downloading package vader_lexicon to
[nltk_data]     /home/shanud6711/nltk_data...
```

Out[27]:

```
True
```

In [28]:

```
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer
analyser = SentimentIntensityAnalyzer()

neg = []
pos = []
neu = []
compound = []

for a in tqdm(project_data["clean_essays"]) :
    b = analyser.polarity_scores(a)['neg']
    c = analyser.polarity_scores(a)['pos']
    d = analyser.polarity_scores(a)['neu']
    e = analyser.polarity_scores(a)['compound']
    neg.append(b)
    pos.append(c)
    neu.append(d)
    compound.append(e)
```

```
100%|██████████| 109248/109248 [15:40<00:00, 116.21it/s]
```

In [29]:

```
project_data["pos"] = pos
```

In [30]:

```
project_data["neg"] = neg
```

In [31]:

```
project_data["neu"] = neu
```

In [32]:

```
project_data["compound"] = compound
```

In [33]:

```
project_data.head(5)
```

Out[33]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	
55660	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	CA	00:
76127	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	UT	00:
51140	74477	p189804	4a97f3a390bfe21b99cf5e2b81981c73	Mrs.	CA	00:
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	GA	00:
41558	33679	p137682	06f6e62e17de34fcf81020c77549e1d5	Mrs.	WA	01:

5 rows × 24 columns

Type *Markdown* and LaTeX: α^2

In [34]:

```
# train test split

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(project_data, project_data['project_is_
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33, stratify=
```

In [35]:

```
# printing some random reviews
print(project_data['clean_essays'].values[0])
print("="*50)
print(project_data['clean_essays'].values[150])
print("="*50)
print(project_data['clean_essays'].values[1000])
print("="*50)
print(project_data['clean_essays'].values[20000])
print("="*50)
print(project_data['clean_essays'].values[99999])
print("="*50)
```

```
i fortunate enough use fairy tale stem kits classroom well stem journals stu
dents really enjoyed i would love implement lakeshore stem kits classroom ne
xt school year provide excellent engaging stem lessons my students come vari
ety backgrounds including language socioeconomic status many not lot experie
nce science engineering kits give materials provide exciting opportunities s
tudents each month i try several science stem steam projects i would use kit
s robot help guide science instruction engaging meaningful ways i adapt kits
current language arts pacing guide already teach material kits like tall tal
es paul bunyan johnny appleseed the following units taught next school year
i implement kits magnets motion sink vs float robots i often get units not k
now if i teaching right way using right materials the kits give additional i
deas strategies lessons prepare students science it challenging develop high
quality science activities these kits give materials i need provide students
science activities go along curriculum classroom although i things like magn
ets classroom i not know use effectively the kits provide right amount mater
ials show use appropriate way
==================================================
i teach high school english students learning behavioral disabilities my stu
dents vary ability level however ultimate goal increase students literacy le
vels this includes reading writing communication levels i teach really dynam
ic group students however students face lot challenges my students live pove
rty dangerous neighborhood despite challenges i students desire defeat chall
enges my students learning disabilities currently performing grade level my
students visual learners benefit classroom fulfills preferred learning style
the materials i requesting allow students prepared classroom necessary suppl
ies too often i challenged students come school unprepared class due economi
c challenges i want students able focus learning not able get school supplie
s the supplies last year students able complete written assignments maintain
classroom journal the chart paper used make learning visual class create pos
ters aid students learning the students access classroom printer the toner u
sed print student work completed classroom chromebooks i want try remove bar
riers students learning create opportunities learning one biggest barriers s
tudents not resources get pens paper folders my students able increase liter
acy skills project
==================================================
life moves pretty fast if not stop look around awhile could miss movie ferri
s bueller day off think back remember grandparents how amazing would able fl
ip book see day lives my second graders voracious readers they love read fic
tion nonfiction books their favorite characters include pete cat fly guy pig
gie elephant mercy watson they also love read insects space plants my studen
ts hungry bookworms my students eager learn read world around my kids love s
chool like little sponges absorbing everything around their parents work lon
g hours usually not see children my students usually cared grandparents fami
ly friend most students not someone speaks english home thus difficult stude
nts acquire language now think forward would not mean lot kids nieces nephew
s grandchildren able see day life today 30 years memories precious us able s
hare memories future generations rewarding experience as part social studies
```

curriculum students learning changes time students studying photos learn com
munity changed time in particular look photos study land buildings clothing
schools changed time as culminating activity students capture slice history
preserve scrap booking key important events young lives documented date loca
tion names students using photos home school create second grade memories th
eir scrap books preserve unique stories future generations enjoy your donati
on project provide second graders opportunity learn social studies fun creat
ive manner through scrapbooks children share story others historical documen
t rest lives

==========

a person person no matter small dr seuss i teach smallest students biggest e
nthusiasm learning my students learn many different ways using senses multip
le intelligences i use wide range techniques help students succeed students
class come variety different backgrounds makes wonderful sharing experiences
cultures including native americans our school caring community successful l
earners seen collaborative student project based learning classroom kinderga
rteners class love work hands materials many different opportunities practic
e skill mastered having social skills work cooperatively friends crucial asp
ect kindergarten curriculum montana perfect place learn agriculture nutritio
n my students love role play pretend kitchen early childhood classroom i sev
eral kids ask can try cooking real food i take idea create common core cooki
ng lessons learn important math writing concepts cooking delicious healthy f
ood snack time my students grounded appreciation work went making food knowl
edge ingredients came well healthy bodies this project would expand learning
nutrition agricultural cooking recipes us peel apples make homemade applesau
ce make bread mix healthy plants classroom garden spring we also create cook
books printed shared families students gain math literature skills well life
long enjoyment healthy cooking nannan

==========

my classroom consists twenty two amazing sixth graders different cultures ba
ckgrounds they social bunch enjoy working partners working groups they hard
working eager head middle school next year my job get ready make transition
make smooth possible in order students need come school every day feel safe
ready learn because getting ready head middle school i give lots choice choi
ce sit work order complete assignments choice projects etc part students fee
ling safe ability come welcoming encouraging environment my room colorful at
mosphere casual i want take ownership classroom all share together because t
ime limited i want ensure get time enjoy best abilities currently twenty two
desks differing sizes yet desks similar ones students use middle school we a
lso kidney table crates seating i allow students choose spots working indepe
ndently groups more often not move desks onto crates believe not proven succ
essful making stay desks it i looking toward flexible seating option classro
om the students look forward work time move around room i would like get rid
constricting desks move toward fun seating options i requesting various seat
ing students options sit currently i stool papasan chair i inherited previou
s sixth grade teacher well five milk crate seats i made i would like give op
tions reduce competition good seats i also requesting two rugs not seating o
ptions make classroom welcoming appealing in order students able write compl
ete work without desks i requesting class set clipboards finally due curricu
lum requires groups work together i requesting tables fold not using leave r
oom flexible seating options i know seating options much excited coming scho
ol thank support making classroom one students remember forever nannan

==========

In [36]:

```
# 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 [37]:

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

```
a person person no matter small dr seuss i teach smallest students biggest e
nthusiasm learning my students learn many different ways using senses multip
le intelligences i use wide range techniques help students succeed students
class come variety different backgrounds makes wonderful sharing experiences
cultures including native americans our school caring community successful l
earners seen collaborative student project based learning classroom kinderga
rteners class love work hands materials many different opportunities practic
e skill mastered having social skills work cooperatively friends crucial asp
ect kindergarten curriculum montana perfect place learn agriculture nutritio
n my students love role play pretend kitchen early childhood classroom i sev
eral kids ask can try cooking real food i take idea create common core cooki
ng lessons learn important math writing concepts cooking delicious healthy f
ood snack time my students grounded appreciation work went making food knowl
edge ingredients came well healthy bodies this project would expand learning
nutrition agricultural cooking recipes us peel apples make homemade applesau
ce make bread mix healthy plants classroom garden spring we also create cook
books printed shared families students gain math literature skills well life
long enjoyment healthy cooking nannan
==================================================
```

In [38]:

```
# \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)
```

```
a person person no matter small dr seuss i teach smallest students biggest e
nthusiasm learning my students learn many different ways using senses multip
le intelligences i use wide range techniques help students succeed students
class come variety different backgrounds makes wonderful sharing experiences
cultures including native americans our school caring community successful l
earners seen collaborative student project based learning classroom kinderga
rteners class love work hands materials many different opportunities practic
e skill mastered having social skills work cooperatively friends crucial asp
ect kindergarten curriculum montana perfect place learn agriculture nutritio
n my students love role play pretend kitchen early childhood classroom i sev
eral kids ask can try cooking real food i take idea create common core cooki
ng lessons learn important math writing concepts cooking delicious healthy f
ood snack time my students grounded appreciation work went making food knowl
edge ingredients came well healthy bodies this project would expand learning
nutrition agricultural cooking recipes us peel apples make homemade applesau
ce make bread mix healthy plants classroom garden spring we also create cook
books printed shared families students gain math literature skills well life
long enjoyment healthy cooking nannan
```

In [39]:

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

```
a person person no matter small dr seuss i teach smallest students biggest e
nthusiasm learning my students learn many different ways using senses multip
le intelligences i use wide range techniques help students succeed students
class come variety different backgrounds makes wonderful sharing experiences
cultures including native americans our school caring community successful l
earners seen collaborative student project based learning classroom kinderga
rteners class love work hands materials many different opportunities practic
e skill mastered having social skills work cooperatively friends crucial asp
ect kindergarten curriculum montana perfect place learn agriculture nutritio
n my students love role play pretend kitchen early childhood classroom i sev
eral kids ask can try cooking real food i take idea create common core cooki
ng lessons learn important math writing concepts cooking delicious healthy f
ood snack time my students grounded appreciation work went making food knowl
edge ingredients came well healthy bodies this project would expand learning
nutrition agricultural cooking recipes us peel apples make homemade applesau
ce make bread mix healthy plants classroom garden spring we also create cook
books printed shared families students gain math literature skills well life
long enjoyment healthy cooking nannan
```

In [40]:

```
# 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'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they'
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'l
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had',
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'u
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'd
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over',
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any',
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'v
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now',
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'do
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn',
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn'
            'won', "won't", 'wouldn', "wouldn't"]
```

Preprocessed Train data (Essay)

In [41]:

```
# Combining all the above stundents
from tqdm import tqdm
preprocessed_essays_train = []
# tqdm is for printing the status bar
for sentance in tqdm(X_train['clean_essays'].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.lower() not in stopwords)
    preprocessed_essays_train.append(sent.lower().strip())
```

100%|██████████| 49041/49041 [00:23<00:00, 2081.25it/s]

In [42]:

```
# after preprocesing
preprocessed_essays_train[20000]
```

Out[42]:

```
'students walk class smiles faces exhibit eagerness learn something new ever
yday not afraid ask questions truly want know anything class students come c
lass not even knowing speak english deaf hard hearing communication difficul
t never even able experience activities physical education class students co
me low income homes breakfast lunches provided no charge well uniforms fees
students may not worldly possessions possess heart gold eagerness great want
focus teaching students make healthy lifestyle choices comes eating exercisi
ng students truly enjoy receiving handouts completing assignments class take
home teach families classroom kits help inform students make healthier food
choices well teach exercise safely fun help donations not students impacted
become teachers educating others community live happy healthy lifestyle than
k nannan'
```

Preprocessed Test data (Essay)

In [43]:

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

100%|██████████| 36052/36052 [00:17<00:00, 2089.73it/s]

In [44]:

```
preprocessed_essays_test[0]
```

Out[44]:

```
'middle school weird silly crazy weirdos amazing talented aspiring artists h
onestly great kids exciting energy filled engaging crazy fun art classroom f
amily made every race religion socioeconomic status students walks life gath
ered together form little art family middle school well know time discomfort
awkwardness age group difficult students not go store buy art supplies see s
tudents art classroom place everyone equal stuff succeed creative ways pop a
rt cubes cardboard food masking tape shoes super fun projects class would lo
ve get create year sharpies allow us create outlines pop art awesome not ble
ed get wet watercolor xacto knives necessary small little cuts details cardb
oard food finally using masking tape really want create cool shoe sculptures
art projects engage students outside box creativity problem solving allowing
students ability exercise creativity ways far beyond typical classroom reall
y hopeful help supplies create amazing expressive ways nannan'
```

Preprocessed Cross Validation data (essay)

In [45]:

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

100%|██████████| 24155/24155 [00:11<00:00, 2090.18it/s]

In [46]:

```
preprocessed_essays_cv[0]
```

Out[46]:

```
'students active explorers hip techies creative geniuses greatest teachers f
requently blown away kindness intuitiveness humility brain power strong love
learning school shteam driven dig kids love future needs science history tec
hnology engineering arts mathematics using engineering cycle scientific meth
od pure love learning recreate past work present plan bright future students
school learning coding engineering programming year modern technology class
hands way one way practice problem solving skills use dash osmo robotics hop
e prepare students 21st century skills career ready future dash osmo program
mable robots osmo allow younger students practice math phonics shapes dash a
llow older students depth problem solving maneuvering around objects respond
ing stimuli turning ideas adventures also hope incorporate music xylophone a
ccessory students compose simple songs dash nannan'
```

1.4 Preprocessing of `project_title`

Preprocessing of Project Title for Train data

In [47]:

```
# similarly you can preprocess the titles also
preprocessed_titles_train = []

for titles in tqdm(X_train["clean_titles"]):
    title = decontracted(titles)
    title = title.replace('\\r', ' ')
    title = title.replace('\\"', ' ')
    title = title.replace('\\n', ' ')
    title = re.sub('[^A-Za-z0-9]+', ' ', title)
    title = ' '.join(f for f in title.split() if f not in stopwords)
    preprocessed_titles_train.append(title.lower().strip())
```

```
100%|██████████| 49041/49041 [00:01<00:00, 36336.02it/s]
```

In [48]:

```
preprocessed_titles_train[0]
```

Out[48]:

```
'writing mentor texts'
```

Preprocessing of Project Title for Test data

In [49]:

```
preprocessed_titles_test = []

for titles in tqdm(X_test["clean_titles"]):
    title = decontracted(titles)
    title = title.replace('\\r', ' ')
    title = title.replace('\\"', ' ')
    title = title.replace('\\n', ' ')
    title = re.sub('[^A-Za-z0-9]+', ' ', title)
    title = ' '.join(f for f in title.split() if f not in stopwords)
    preprocessed_titles_test.append(title.lower().strip())
```

100%|██████████| 36052/36052 [00:00<00:00, 36297.68it/s]

In [50]:

```
preprocessed_titles_test[0]
```

Out[50]:

```
'pop art cubes cardboard food masking tape shoes'
```

Preprocessing of Project Title for CV data

In [51]:

```
preprocessed_titles_cv = []

for titles in tqdm(X_cv["clean_titles"]):
    title = decontracted(titles)
    title = title.replace('\\r', ' ')
    title = title.replace('\\"', ' ')
    title = title.replace('\\n', ' ')
    title = re.sub('[^A-Za-z0-9]+', ' ', title)
    title = ' '.join(f for f in title.split() if f not in stopwords)
    preprocessed_titles_cv.append(title.lower().strip())
```

100%|██████████| 24155/24155 [00:00<00:00, 36436.06it/s]

In [52]:

```
preprocessed_titles_cv[0]
```

Out[52]:

```
'dash finish'
```

1.5 Preparing data for models

In [53]:

```
project_data.columns
```

Out[53]:

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'Date', 'project_essay_1', 'project_essay_2', 'project_essay_3',
       'project_essay_4', 'project_resource_summary',
       'teacher_number_of_previously_posted_projects', 'project_is_approve
d',
       'project_grade_category', 'clean_categories', 'clean_subcategories',
       'clean_titles', 'title_word_count', 'clean_essays', 'essay_word_coun
t',
       'pos', 'neg', 'neu', 'compound'],
      dtype='object')
```

we are going to consider

```
- school_state : categorical data
- clean_categories : categorical data
- clean_subcategories : categorical data
- project_grade_category : categorical data
- teacher_prefix : categorical data

- project_title : text data
- text : text data
- project_resource_summary: text data (optinal)

- quantity : numerical (optinal)
- teacher_number_of_previously_posted_projects : numerical
- price : numerical
```

1.5.1 Vectorizing Categorical data

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

In [54]:

```
# we use count vectorizer to convert the values into one
from sklearn.feature_extraction.text import CountVectorizer

vectorizer_proj = CountVectorizer(vocabulary=list(sorted_cat_dict.keys()), lowercase=False,
vectorizer_proj.fit(X_train['clean_categories'].values)

categories_one_hot_train = vectorizer_proj.transform(X_train['clean_categories'].values)
categories_one_hot_test = vectorizer_proj.transform(X_test['clean_categories'].values)
categories_one_hot_cv = vectorizer_proj.transform(X_cv['clean_categories'].values)

print(vectorizer_proj.get_feature_names())

print("Shape of matrix of Train data after one hot encoding ",categories_one_hot_train.shap
print("Shape of matrix of Test data after one hot encoding ",categories_one_hot_test.shape)
print("Shape of matrix of CV data after one hot encoding ",categories_one_hot_cv.shape)
```

```
['Health_Sports', 'Music_Arts', 'Care_Hunger', 'History_Civics', 'Math_Scien
ce', 'Warmth', 'SpecialNeeds', 'AppliedLearning', 'Literacy_Language']
Shape of matrix of Train data after one hot encoding  (49041, 9)
Shape of matrix of Test data after one hot encoding  (36052, 9)
Shape of matrix of CV data after one hot encoding  (24155, 9)
```

In [55]:

```
# we use count vectorizer to convert the values into one
vectorizer_sub_proj = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()), lowercas
vectorizer_sub_proj.fit(X_train['clean_subcategories'].values)

sub_categories_one_hot_train = vectorizer_sub_proj.transform(X_train['clean_subcategories']
sub_categories_one_hot_test = vectorizer_sub_proj.transform(X_test['clean_subcategories'].v
sub_categories_one_hot_cv = vectorizer_sub_proj.transform(X_cv['clean_subcategories'].value


print(vectorizer_sub_proj.get_feature_names())

print("Shape of matrix of Train data after one hot encoding ",sub_categories_one_hot_train.
print("Shape of matrix of Test data after one hot encoding ",sub_categories_one_hot_test.sh
print("Shape of matrix of Cross Validation data after one hot encoding ",sub_categories_one
```

```
['AppliedSciences', 'Care_Hunger', 'Gym_Fitness', 'Civics_Government', 'Soci
alSciences', 'SpecialNeeds', 'EnvironmentalScience', 'Health_Wellness', 'Vis
ualArts', 'TeamSports', 'ParentInvolvement', 'Other', 'Literacy', 'Economic
s', 'Warmth', 'EarlyDevelopment', 'Music', 'ForeignLanguages', 'CommunitySer
vice', 'NutritionEducation', 'Mathematics', 'Extracurricular', 'Literature_W
riting', 'CharacterEducation', 'College_CareerPrep', 'Health_LifeScience',
'PerformingArts', 'FinancialLiteracy', 'ESL', 'History_Geography']
Shape of matrix of Train data after one hot encoding  (49041, 30)
Shape of matrix of Test data after one hot encoding  (36052, 30)
Shape of matrix of Cross Validation data after one hot encoding  (24155, 30)
```

In [56]:

```
# you can do the similar thing with state, teacher_prefix and project_grade_category also
my_counter = Counter()
for state in project_data['school_state'].values:
    my_counter.update(state.split())
```

In [57]:

```
school_state_cat_dict = dict(my_counter)
sorted_school_state_cat_dict = dict(sorted(school_state_cat_dict.items(), key=lambda kv: kv
```

In [58]:

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

vectorizer_states = CountVectorizer(vocabulary=list(sorted_school_state_cat_dict.keys()), l
vectorizer_states.fit(X_train['school_state'].values)

school_state_categories_one_hot_train = vectorizer_states.transform(X_train['school_state']
school_state_categories_one_hot_test = vectorizer_states.transform(X_test['school_state'].v
school_state_categories_one_hot_cv = vectorizer_states.transform(X_cv['school_state'].value

print(vectorizer_states.get_feature_names())

print("Shape of matrix of Train data after one hot encoding ",school_state_categories_one_h
print("Shape of matrix of Test data after one hot encoding ",school_state_categories_one_ho
print("Shape of matrix of Cross Validation data after one hot encoding ",school_state_categ
```

```
['TN', 'MO', 'IA', 'TX', 'NH', 'CT', 'AK', 'OK', 'ID', 'IN', 'SC', 'HI', 'M
N', 'NV', 'MI', 'NM', 'NJ', 'AZ', 'KS', 'LA', 'OH', 'OR', 'KY', 'RI', 'NC',
'CA', 'UT', 'VA', 'VT', 'FL', 'WI', 'ND', 'CO', 'GA', 'AL', 'WV', 'WY', 'M
D', 'DC', 'NY', 'ME', 'AR', 'WA', 'MT', 'MS', 'NE', 'DE', 'PA', 'IL', 'MA',
'SD']
Shape of matrix of Train data after one hot encoding  (49041, 51)
Shape of matrix of Test data after one hot encoding  (36052, 51)
Shape of matrix of Cross Validation data after one hot encoding  (24155, 51)
```

In [59]:

```
my_counter = Counter()
for project_grade in project_data['project_grade_category'].values:
    my_counter.update(project_grade.split())
```

In [60]:

```
project_grade_cat_dict = dict(my_counter)
sorted_project_grade_cat_dict = dict(sorted(project_grade_cat_dict.items(), key=lambda kv:
```

In [61]:

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

vectorizer_grade = CountVectorizer(vocabulary=list(sorted_project_grade_cat_dict.keys()), l
vectorizer_grade.fit(X_train['project_grade_category'].values)

project_grade_categories_one_hot_train = vectorizer_grade.transform(X_train['project_grade_
project_grade_categories_one_hot_test = vectorizer_grade.transform(X_test['project_grade_ca
project_grade_categories_one_hot_cv = vectorizer_grade.transform(X_cv['project_grade_catego

print(vectorizer_grade.get_feature_names())

print("Shape of matrix of Train data after one hot encoding ",project_grade_categories_one_
print("Shape of matrix of Test data after one hot encoding ",project_grade_categories_one_h
print("Shape of matrix of Cross Validation data after one hot encoding ",project_grade_cate
```

```
['Grades_9-12', 'Grades_6-8', 'Grades_3-5', 'Grades_PreK-2']
Shape of matrix of Train data after one hot encoding  (49041, 4)
Shape of matrix of Test data after one hot encoding  (36052, 4)
Shape of matrix of Cross Validation data after one hot encoding  (24155, 4)
```

In [62]:

```
my_counter = Counter()
for teacher_prefix in project_data['teacher_prefix'].values:
    teacher_prefix = str(teacher_prefix)
    my_counter.update(teacher_prefix.split())
```

In [63]:

```
teacher_prefix_cat_dict = dict(my_counter)
sorted_teacher_prefix_cat_dict = dict(sorted(teacher_prefix_cat_dict.items(), key=lambda kv
```

In [64]:

```
## we use count vectorizer to convert the values into one hot encoded features
## Unlike the previous Categories this category returns a
## ValueError: np.nan is an invalid document, expected byte or unicode string.
## The link below explains h0w to tackle such discrepancies.
## https://stackoverflow.com/questions/39303912/tfidfvectorizer-in-scikit-learn-valueerror-

vectorizer_teacher = CountVectorizer(vocabulary=list(sorted_teacher_prefix_cat_dict.keys())
vectorizer_teacher.fit(X_train['teacher_prefix'].values.astype("U"))

teacher_prefix_categories_one_hot_train = vectorizer_teacher.transform(X_train['teacher_pre
teacher_prefix_categories_one_hot_test = vectorizer_teacher.transform(X_test['teacher_prefi
teacher_prefix_categories_one_hot_cv = vectorizer_teacher.transform(X_cv['teacher_prefix'].

print(vectorizer_teacher.get_feature_names())

print("Shape of matrix after one hot encoding ",teacher_prefix_categories_one_hot_train.sha
print("Shape of matrix after one hot encoding ",teacher_prefix_categories_one_hot_test.shap
print("Shape of matrix after one hot encoding ",teacher_prefix_categories_one_hot_cv.shape)
```

```
['nan', 'Mrs.', 'Teacher', 'Ms.', 'Mr.', 'Dr.']
Shape of matrix after one hot encoding  (49041, 6)
Shape of matrix after one hot encoding  (36052, 6)
Shape of matrix after one hot encoding  (24155, 6)
```

1.5.2 Vectorizing Text data

a) Bag of words Train Data (Essays)

In [65]:

```
# We are considering only the words which appeared in at least 10 documents(rows or project
vectorizer_bow_essay = CountVectorizer(ngram_range=(2,2), min_df=10, max_features = 5000)

vectorizer_bow_essay.fit(preprocessed_essays_train)

text_bow_train = vectorizer_bow_essay.transform(preprocessed_essays_train)

print("Shape of matrix after one hot encoding ",text_bow_train.shape)
```

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

b) Bag of words Test Data (Essays)

In [66]:

```
text_bow_test = vectorizer_bow_essay.transform(preprocessed_essays_test)
print("Shape of matrix after one hot encoding ",text_bow_test.shape)
```

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

c) Bag of words CV Data (Essays)

In [67]:

```
text_bow_cv = vectorizer_bow_essay.transform(preprocessed_essays_cv)
print("Shape of matrix after one hot encoding ",text_bow_cv.shape)
```

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

d) Bag of words train Data (Titles)

In [68]:

```
# you can vectorize the title also
# before you vectorize the title make sure you preprocess it
vectorizer_bow_title = CountVectorizer(ngram_range=(2,2), min_df=10, max_features = 5000)

vectorizer_bow_title.fit(preprocessed_titles_train)

title_bow_train = vectorizer_bow_title.transform(preprocessed_titles_train)
print("Shape of matrix after one hot encoding ",title_bow_train.shape)
```

```
Shape of matrix after one hot encoding  (49041, 1223)
```

e) Bag of words Test Data (Titles)

In [69]:

```
title_bow_test = vectorizer_bow_title.transform(preprocessed_titles_test)
print("Shape of matrix after one hot encoding ",title_bow_test.shape)
```

```
Shape of matrix after one hot encoding  (36052, 1223)
```

f) Bag of words Data (Titles)

In [70]:

```
title_bow_cv = vectorizer_bow_title.transform(preprocessed_titles_cv)
print("Shape of matrix after one hot encoding ",title_bow_cv.shape)
```

```
Shape of matrix after one hot encoding  (24155, 1223)
```

1.5.2.2 TFIDF vectorizer

a) TFIDF vectorizer Train Data (Essays)

In [71]:

```
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer_tfidf_essay = TfidfVectorizer(ngram_range=(2,2), min_df=10, max_features = 5000)
vectorizer_tfidf_essay.fit(preprocessed_essays_train)

text_tfidf_train = vectorizer_tfidf_essay.transform(preprocessed_essays_train)
print("Shape of matrix after one hot encoding ",text_tfidf_train.shape)
```

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

b) TFIDF vectorizer Test Data (Essays)

In [72]:

```
text_tfidf_test = vectorizer_tfidf_essay.transform(preprocessed_essays_test)
print("Shape of matrix after one hot encoding ",text_tfidf_test.shape)
```

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

c) TFIDF vectorizer CV Data (Essays)

In [73]:

```
text_tfidf_cv = vectorizer_tfidf_essay.transform(preprocessed_essays_cv)
print("Shape of matrix after one hot encoding ",text_tfidf_cv.shape)
```

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

c) TFIDF vectorizer Train Data (Titles)

In [74]:

```
vectorizer_tfidf_titles = TfidfVectorizer(ngram_range=(2,2), min_df=10, max_features = 5000

vectorizer_tfidf_titles.fit(preprocessed_titles_train)
title_tfidf_train = vectorizer_tfidf_titles.transform(preprocessed_titles_train)
print("Shape of matrix after one hot encoding ",title_tfidf_train.shape)
```

```
Shape of matrix after one hot encoding  (49041, 1223)
```

d) TFIDF vectorizer Test Data (Titles)

In [75]:

```
title_tfidf_test = vectorizer_tfidf_titles.transform(preprocessed_titles_test)
print("Shape of matrix after one hot encoding ",title_tfidf_test.shape)
```

```
Shape of matrix after one hot encoding  (36052, 1223)
```

e) TFIDF vectorizer CV Data (Titles)

In [76]:

```
title_tfidf_cv = vectorizer_tfidf_titles.transform(preprocessed_titles_cv)
print("Shape of matrix after one hot encoding ",title_tfidf_cv.shape)
```

```
Shape of matrix after one hot encoding  (24155, 1223)
```

C) Using Pretrained Models : AVG W2V

In [72]:

```
# Reading glove vectors in python: https://stackoverflow.com/a/38230349/4084039
def loadGloveModel(gloveFile):
    print ("Loading Glove Model")
    f = open(gloveFile,'r', encoding="utf8")
    model = {}
    for line in tqdm(f):
        splitLine = line.split()
        word = splitLine[0]
        embedding = np.array([float(val) for val in splitLine[1:]])
        model[word] = embedding
    print ("Done.",len(model)," words loaded!")
    return model

model = loadGloveModel('glove.42B.300d.txt')


words = []
for i in preprocessed_essays_train :
    words.extend(i.split(' '))

print("all the words in the coupus", len(words))
words = set(words)
print("the unique words in the coupus", len(words))

inter_words = set(model.keys()).intersection(words)
print("The number of words that are present in both glove vectors and our coupus", \
      len(inter_words),"(",np.round(len(inter_words)/len(words)*100,3),"%)")

words_courpus = {}
words_glove = set(model.keys())
for i in words:
    if i in words_glove:
        words_courpus[i] = model[i]
print("word 2 vec length", len(words_courpus))


# stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-pickl

import pickle
with open('glove_vectors', 'wb') as f:
    pickle.dump(words_courpus, f)
```

```
Loading Glove Model

1333129it [06:31, 3404.12it/s]

Done. 1333129  words loaded!
all the words in the coupus 6767354
the unique words in the coupus 41263
The number of words that are present in both glove vectors and our coupus 37
422 ( 90.691 %)
word 2 vec length 37422
```

In [77]:

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

Train Essay

In [78]:

```
# average Word2Vec
# compute average word2vec for each review.

avg_w2v_vectors_train = [];

for sentence in tqdm(X_train["clean_essays"]): # 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:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_train.append(vector)

print(len(avg_w2v_vectors_train))
print(len(avg_w2v_vectors_train[0]))
```

```
100%|██████████| 49041/49041 [00:15<00:00, 3112.97it/s]

49041
300
```

Test Essay

In [79]:

```
# average Word2Vec
# compute average word2vec for each review.

avg_w2v_vectors_test = [];

for sentence in tqdm(X_test["clean_essays"]): # 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:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_test.append(vector)

print(len(avg_w2v_vectors_test))
print(len(avg_w2v_vectors_test[0]))
```

```
100%|██████████| 36052/36052 [00:11<00:00, 3104.10it/s]

36052
300
```

Cross validation Essay

In [80]:

```
avg_w2v_vectors_cv = [];

for sentence in tqdm(X_cv["clean_essays"]): # 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:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_cv.append(vector)

print(len(avg_w2v_vectors_cv))
print(len(avg_w2v_vectors_cv[0]))
```

```
100%|██████████| 24155/24155 [00:07<00:00, 3093.67it/s]

24155
300
```

train Titles

In [81]:

```
avg_w2v_vectors_titles_train = []; # the avg-w2v for each sentence/review is stored in this
for sentence in tqdm(X_train["clean_titles"]): # for each title
    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:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_titles_train.append(vector)

print(len(avg_w2v_vectors_titles_train))
print(len(avg_w2v_vectors_titles_train[0]))
```

```
100%|██████████| 49041/49041 [00:00<00:00, 59273.91it/s]

49041
300
```

Test Titles

In [82]:

```
avg_w2v_vectors_titles_test = []; # the avg-w2v for each sentence/review is stored in this
for sentence in tqdm(X_test["clean_titles"]): # for each title
    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:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_titles_test.append(vector)

print(len(avg_w2v_vectors_titles_test))
print(len(avg_w2v_vectors_titles_test[0]))
```

```
100%|██████████| 36052/36052 [00:00<00:00, 57994.96it/s]

36052
300
```

CV Titles

In [83]:

```
avg_w2v_vectors_titles_cv = []; # the avg-w2v for each sentence/review is stored in this li
for sentence in tqdm(X_cv["clean_titles"]): # for each title
    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:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_titles_cv.append(vector)

print(len(avg_w2v_vectors_titles_cv))
print(len(avg_w2v_vectors_titles_cv[0]))
```

```
100%|██████████| 24155/24155 [00:00<00:00, 58699.30it/s]

24155
300
```

D) Using Pretrained Models: TFIDF weighted W2V

Train - Essays

In [84]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train["clean_essays"])
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())
```

In [85]:

```
# 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["clean_essays"]): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =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):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentenc
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # gettin
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_train.append(vector)

print(len(tfidf_w2v_vectors_train))
print(len(tfidf_w2v_vectors_train[0]))
```

```
100%|██████████| 49041/49041 [01:44<00:00, 469.16it/s]

49041
300
```

Test essays

In [86]:

```
# 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["clean_essays"]): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =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):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentenc
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # gettin
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_test.append(vector)

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

```
100%|██████████| 36052/36052 [01:16<00:00, 469.39it/s]

36052
300
```

CV essays

In [87]:

```
# 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["clean_essays"]): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =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):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentenc
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # gettin
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_cv.append(vector)

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

```
100%|██████████| 24155/24155 [00:50<00:00, 478.31it/s]

24155
300
```

Train Titles

In [88]:

```
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train["clean_titles"])
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())
```

In [89]:

```
# compute average word2vec for each review.

tfidf_w2v_vectors_titles_train = [];

for sentence in tqdm(X_train["clean_titles"]): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =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):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentenc
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # gettin
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_titles_train.append(vector)

print(len(tfidf_w2v_vectors_titles_train))
print(len(tfidf_w2v_vectors_titles_train[0]))
```

```
100%|██████████| 49041/49041 [00:01<00:00, 25884.96it/s]

49041
300
```

Test Titles

In [90]:

```
# compute average word2vec for each review.

tfidf_w2v_vectors_titles_test = [];

for sentence in tqdm(X_test["clean_titles"]): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =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):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentenc
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # gettin
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_titles_test.append(vector)

print(len(tfidf_w2v_vectors_titles_test))
print(len(tfidf_w2v_vectors_titles_test[0]))
```

```
100%|██████████| 36052/36052 [00:01<00:00, 29713.90it/s]

36052
300
```

CV Titles

In [91]:

```
# compute average word2vec for each review.

tfidf_w2v_vectors_titles_cv = [];

for sentence in tqdm(X_cv["clean_titles"]): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =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):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentenc
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # gettin
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_titles_cv.append(vector)

print(len(tfidf_w2v_vectors_titles_cv))
print(len(tfidf_w2v_vectors_titles_cv[0]))
```

```
100%|██████████| 24155/24155 [00:00<00:00, 29681.13it/s]

24155
300
```

1.5.3 Vectorizing Numerical features

a) Price

In [92]:

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

In [93]:

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

In [94]:

```
# check this one: https://www.youtube.com/watch?v=0HOqOcln3Z4&t=530s
# standardization sklearn: https://scikitlearn.org/stable/modules/generated/sklearn.preproc
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import Normalizer
from sklearn import preprocessing
price_scalar = MinMaxScaler()
price_scalar.fit(X_train['price'].values.reshape(-1,1)) # finding the mean and standarddevi
#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_train = price_scalar.transform(X_train['price'].values.reshape(-1, 1))
price_train
# Now standardize the data with above maen and variance.
price_test = price_scalar.transform(X_test['price'].values.reshape(-1, 1))
price_test
# Now standardize the data with above maen and variance.
price_cv = price_scalar.transform(X_cv['price'].values.reshape(-1, 1))
price_cv
```

Out[94]:

```
array([[0.03190939],
       [0.04167504],
       [0.01292818],
       ...,
       [0.0023424 ],
       [0.00863146],
       [0.02682353]])
```

In [95]:

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

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

b) Quantity

In [96]:

```
price_scalar.fit(X_train['quantity'].values.reshape(-1,1)) # finding the mean and standard
#print(f"Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(price_scalar.var_[0]
# Now standardize the data with above maen and variance.
quantity_train = price_scalar.transform(X_train['quantity'].values.reshape(-1, 1))
quantity_train
# Now standardize the data with above maen and variance.
quantity_cv = price_scalar.transform(X_cv['quantity'].values.reshape(-1, 1))
quantity_cv
# Now standardize the data with above maen and variance.
quantity_test = price_scalar.transform(X_test['quantity'].values.reshape(-1, 1))
quantity_test
```

Out[96]:

```
array([[0.04783092],
       [0.01334816],
       [0.00111235],
       ...,
       [0.01112347],
       [0.01223582],
       [0.00111235]])
```

In [97]:

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

```
After vectorizations
<built-in method reshape of numpy.ndarray object at 0x7f8ae7b7a440> (49041,)
(24155, 1) (24155,)
(36052, 1) (36052,)
```

c) Number of Projects previously proposed by Teacher

In [98]:

```
price_scalar.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,
#print(f"Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(price_scalar.var_[0]
# Now standardize the data with above maen and variance.
prev_projects_train = price_scalar.transform(X_train['teacher_number_of_previously_posted_p
prev_projects_train
# Now standardize the data with above maen and variance.
prev_projects_cv = price_scalar.transform(X_cv['teacher_number_of_previously_posted_project
prev_projects_cv
# Now standardize the data with above maen and variance.
prev_projects_test = price_scalar.transform(X_test['teacher_number_of_previously_posted_pro
prev_projects_test
```

Out[98]:

```
array([[0.        ],
       [0.        ],
       [0.        ],
       ...,
       [0.07982262],
       [0.02660754],
       [0.02882483]])
```

In [99]:

```
print("After vectorizations")
print(prev_projects_train.shape, y_train.shape)
print(prev_projects_cv.shape, y_cv.shape)
print(prev_projects_test.shape, y_test.shape)
```

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

d) Count of Words in the Title

In [100]:

```
price_scalar.fit(X_train['title_word_count'].values.reshape(-1,1)) # finding the mean and s
#print(f"Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(price_scalar.var_[0]
# Now standardize the data with above maen and variance.
title_word_count_train = price_scalar.transform(X_train['title_word_count'].values.reshape(
title_word_count_train
# Now standardize the data with above maen and variance.
title_word_count_cv = price_scalar.transform(X_cv['title_word_count'].values.reshape(-1, 1)
title_word_count_cv
# Now standardize the data with above maen and variance.
title_word_count_test = price_scalar.transform(X_test['title_word_count'].values.reshape(-1
title_word_count_test
```

Out[100]:

```
array([[0.38888889],
       [0.33333333],
       [0.05555556],
       ...,
       [0.11111111],
       [0.16666667],
       [0.05555556]])
```

In [101]:

```
print("After vectorizations")
print(title_word_count_train.shape, y_train.shape)
print(title_word_count_cv.shape, y_cv.shape)
print(title_word_count_test.shape, y_test.shape)
```

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

e) Count of Words in the Essay

In [102]:

```
price_scalar.fit(X_train['essay_word_count'].values.reshape(-1,1)) # finding the mean and s
#print(f"Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(price_scalar.var_[0]
# Now standardize the data with above maen and variance.
essay_word_count_train = price_scalar.transform(X_train['essay_word_count'].values.reshape(
essay_word_count_train
# Now standardize the data with above maen and variance.
essay_word_count_cv = price_scalar.transform(X_cv['essay_word_count'].values.reshape(-1, 1)
essay_word_count_cv
# Now standardize the data with above maen and variance.
essay_word_count_test = price_scalar.transform(X_test['essay_word_count'].values.reshape(-1
essay_word_count_test


print("After vectorizations")
print(essay_word_count_train.shape, y_train.shape)
print(essay_word_count_cv.shape, y_cv.shape)
print(essay_word_count_test.shape, y_test.shape)
```

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

In [103]:

```
essay_word_count_train
```

Out[103]:

```
array([[0.42748092],
       [0.42748092],
       [0.16412214],
       ...,
       [0.31679389],
       [0.36259542],
       [0.10687023]])
```

Essay Sentiments - Pos

In [104]:

```
price_scalar.fit(X_train['pos'].values.reshape(-1,1)) # finding the mean and standard devia
#print(f"Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(price_scalar.var_[0]
# Now standardize the data with above maen and variance.
essay_sent_pos_train = price_scalar.transform(X_train['pos'].values.reshape(-1, 1))
essay_sent_pos_train
# Now standardize the data with above maen and variance.
essay_sent_pos_cv = price_scalar.transform(X_cv['pos'].values.reshape(-1, 1))
essay_sent_pos_cv
# Now standardize the data with above maen and variance.
essay_sent_pos_test = price_scalar.transform(X_test['pos'].values.reshape(-1, 1))
essay_sent_pos_test
```

Out[104]:

```
array([[0.77684564],
       [0.47483221],
       [0.26845638],
       ...,
       [0.57550336],
       [0.69630872],
       [0.56040268]])
```

In [105]:

```
print("After vectorizations")
print(essay_sent_pos_train.shape, y_train.shape)
print(essay_sent_pos_cv.shape, y_cv.shape)
print(essay_sent_pos_test.shape, y_test.shape)
```

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

Essay Sentiments - Neg

In [106]:

```
price_scalar.fit(X_train['neg'].values.reshape(-1,1)) # finding the mean and standard devia
#print(f"Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(price_scalar.var_[0]
# Now standardize the data with above maen and variance.
essay_sent_neg_train = price_scalar.transform(X_train['neg'].values.reshape(-1, 1))
essay_sent_neg_train
# Now standardize the data with above maen and variance.
essay_sent_neg_cv = price_scalar.transform(X_cv['neg'].values.reshape(-1, 1))
essay_sent_neg_cv
# Now standardize the data with above maen and variance.
essay_sent_neg_test = price_scalar.transform(X_test['neg'].values.reshape(-1, 1))
essay_sent_neg_test
```

Out[106]:

```
array([[0.30322581],
       [0.17741935],
       [0.37741935],
       ...,
       [0.39354839],
       [0.04516129],
       [0.36129032]])
```

In [107]:

```
print("After vectorizations")
print(essay_sent_neg_train.shape, y_train.shape)
print(essay_sent_neg_cv.shape, y_cv.shape)
print(essay_sent_neg_test.shape, y_test.shape)
```

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

Essay Sentiments - Neu

In [108]:

```
price_scalar.fit(X_train['neu'].values.reshape(-1,1)) # finding the mean and standard devia
#print(f"Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(price_scalar.var_[0]
# Now standardize the data with above maen and variance.
essay_sent_neu_train = price_scalar.transform(X_train['neu'].values.reshape(-1, 1))
essay_sent_neu_train
# Now standardize the data with above maen and variance.
essay_sent_neu_cv = price_scalar.transform(X_cv['neu'].values.reshape(-1, 1))
essay_sent_neu_cv
# Now standardize the data with above maen and variance.
essay_sent_neu_test = price_scalar.transform(X_test['neu'].values.reshape(-1, 1))
essay_sent_neu_test
```

Out[108]:

```
array([[0.10144928],
       [0.45410628],
       [0.55394525],
       ...,
       [0.25120773],
       [0.30756844],
       [0.28180354]])
```

In [109]:

```
print("After vectorizations")
print(essay_sent_neu_train.shape, y_train.shape)
print(essay_sent_neu_cv.shape, y_cv.shape)
print(essay_sent_neu_test.shape, y_test.shape)
```

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

Essay Sentiments - Compound

In [110]:

```
price_scalar.fit(X_train['compound'].values.reshape(-1,1)) # finding the mean and standard
#print(f"Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(price_scalar.var_[0]
# Now standardize the data with above maen and variance.
essay_sent_comp_train = price_scalar.transform(X_train['compound'].values.reshape(-1, 1))
essay_sent_comp_train
# Now standardize the data with above maen and variance.
essay_sent_comp_cv = price_scalar.transform(X_cv['compound'].values.reshape(-1, 1))
essay_sent_comp_cv
# Now standardize the data with above maen and variance.
essay_sent_comp_test = price_scalar.transform(X_test['compound'].values.reshape(-1, 1))
essay_sent_comp_test
```

Out[110]:

```
array([[0.99949804],
       [0.9952816 ],
       [0.77065556],
       ...,
       [0.9941271 ],
       [0.99738982],
       [0.99663688]])
```

In [111]:

```
print("After vectorizations")
print(essay_sent_comp_train.shape, y_train.shape)
print(essay_sent_comp_cv.shape, y_cv.shape)
print(essay_sent_comp_test.shape, y_test.shape)
```

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

Assignment 7: SVM

1. **[Task-1] Apply Support Vector Machines(SGDClassifier with hinge loss: Linear SVM) on these feature sets**

   - Set 1: categorical, numerical features + project_title(BOW) + preprocessed_eassay (BOW)
   - Set 2: categorical, numerical features + project_title(TFIDF)+ preprocessed_eassay (TFIDF)
   - Set 3: categorical, numerical features + project_title(AVG W2V)+ preprocessed_eassay (AVG W2V)
   - Set 4: categorical, numerical features + project_title(TFIDF W2V)+ preprocessed_eassay (TFIDF W2V)

2. **The hyper paramter tuning (best alpha in range [10^-4 to 10^4], and the best penalty among 'l1', 'l2')**

   - Find the best hyper parameter which will give the maximum AUC (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/receiver-operating-characteristic-curve-roc-curve-and-auc-1/) value
   - Find the best hyper paramter using k-fold cross validation or simple cross validation data
   - Use gridsearch cv or randomsearch cv or you can also write your own for loops to do this task of hyperparameter tuning

3. **Representation of results**

   - You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure.
   
   - Once after you found the best hyper parameter, you need to train your model with it, and find the AUC on test data and plot the ROC curve on both train and test.
   
   - Along with plotting ROC curve, you need to print the confusion matrix (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/confusion-matrix-tpr-fpr-fnr-tnr-1/) with predicted and original labels of test data points. Please visualize your confusion matrices using seaborn heatmaps.
   
   (https://seaborn.pydata.org/generated/seaborn.heatmap.html)

4. **[Task-2] Apply the Support Vector Machines on these features by finding the best hyper paramter as suggested in step 2 and step 3**

   - Consider these set of features Set 5 :
     - **school_state** : categorical data
     - **clean_categories** : categorical data
     - **clean_subcategories** : categorical data
     - **project_grade_category** :categorical data
     - **teacher_prefix** : categorical data
     - **quantity** : numerical data
     - **teacher_number_of_previously_posted_projects** : numerical data
     - **price** : numerical data
     - **sentiment score's of each of the essay** : numerical data
     - **number of words in the title** : numerical data
     - **number of words in the combine essays** : numerical data
     - **Apply TruncatedSVD (http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.TruncatedSVD.html) on TfidfVectorizer (https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html) of essay text, choose the number of components (`n_components`) using elbow method (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/pca-code-example-using-non-visualization/)** : numerical data

   - **Conclusion**
     - You need to summarize the results at the end of the notebook, summarize it in the table format. To print out a table please refer to this prettytable library link (http://zetcode.com/python/prettytable/)
     

Note: Data Leakage

1. There will be an issue of data-leakage if you vectorize the entire data and then split it into train/cv/test.
2. To avoid the issue of data-leakage, make sure to split your data first and then vectorize it.
3. While vectorizing your data, apply the method fit_transform() on you train data, and apply the method transform() on cv/test data.

4. For more details please go through this link. (https://soundcloud.com/applied-ai-course/leakage-bow-and-tfidf)

2.1 Set 1: categorical, numerical features + project_title(BOW) + preprocessed_essay (BOW)

In [112]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack

X_tr = hstack((categories_one_hot_train, sub_categories_one_hot_train,school_state_categori
X_te = hstack((categories_one_hot_test, sub_categories_one_hot_test,school_state_categories
X_cr = hstack((categories_one_hot_cv, sub_categories_one_hot_cv,school_state_categories_one
```

In [113]:

```
print("Final Data matrix")
print(X_tr.shape, y_train.shape)
print(X_cr.shape, y_cv.shape)
print(X_te.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(49041, 6332) (49041,)
(24155, 6332) (24155,)
(36052, 6332) (36052,)
====================================================================================
========================
```

In [114]:

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

    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000
    # consider you X_tr shape is 49041, then your cr_loop will be 49041 - 49041%1000 = 4900
    # in this for loop we will iterate unti the last 1000 multiplier
    for i in range(0, tr_loop, 1000):
        y_data_pred.extend(clf.predict_proba(data[i:i+1000])[:,1])
    # we will be predicting for the last data points
    y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

    return y_data_pred
```

A) Gridsearch-cv

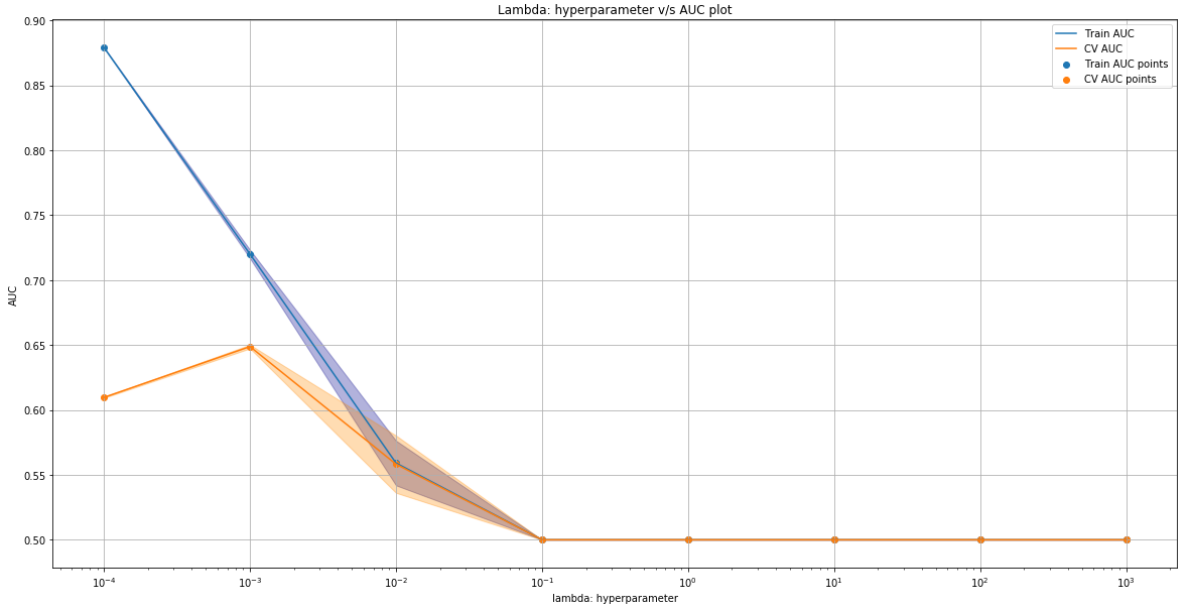
In [115]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import SGDClassifier
```

With L1 Regularizer

In [117]:

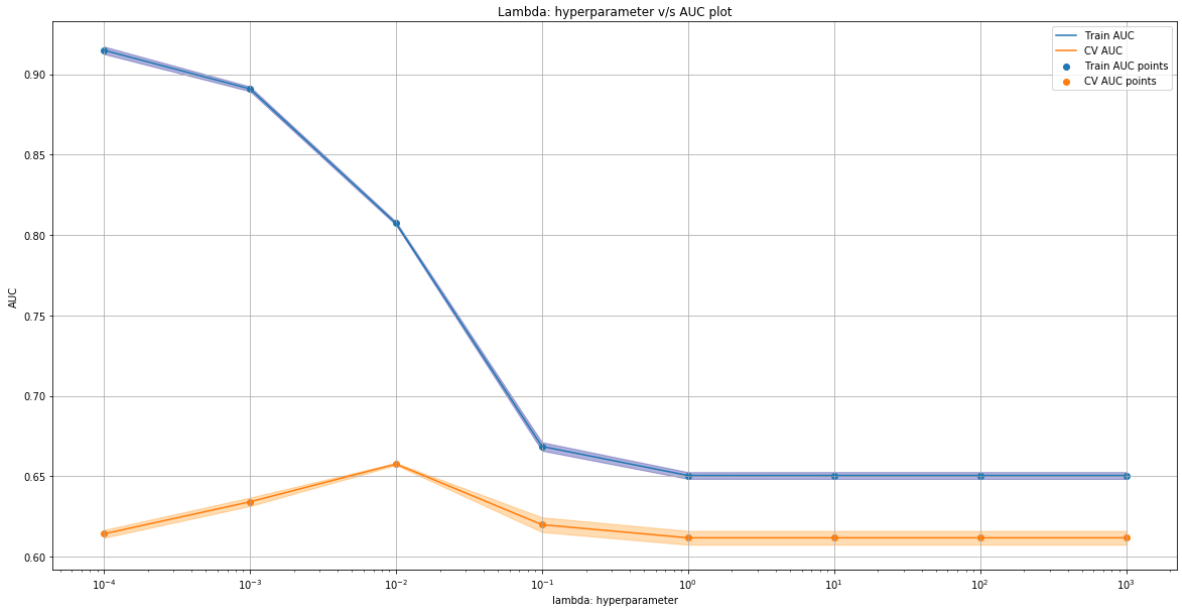
```
sv = SGDClassifier(loss='hinge', penalty='l1',class_weight = 'balanced')
alpha_vals = [10**x for x in range(-4,4)]

parameters = {'alpha':alpha_vals}
clf = GridSearchCV(sv, parameters, cv= 2, scoring='roc_auc',n_jobs=-1,return_train_score=Tr
clf.fit(X_tr, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']

plt.figure(figsize=(20,10))

plt.plot(parameters['alpha'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['alpha'],train_auc - train_auc_std,train_auc + train_auc_

plt.plot(parameters['alpha'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['alpha'],cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.

plt.scatter(parameters['alpha'], train_auc, label='Train AUC points')
plt.scatter(parameters['alpha'], cv_auc, label='CV AUC points')
plt.xscale('log')

plt.legend()
plt.xlabel("lambda: hyperparameter")
plt.ylabel("AUC")
plt.title("Lambda: hyperparameter v/s AUC plot")
plt.grid()
plt.show()
```



with L2 Regularizer

In [118]:

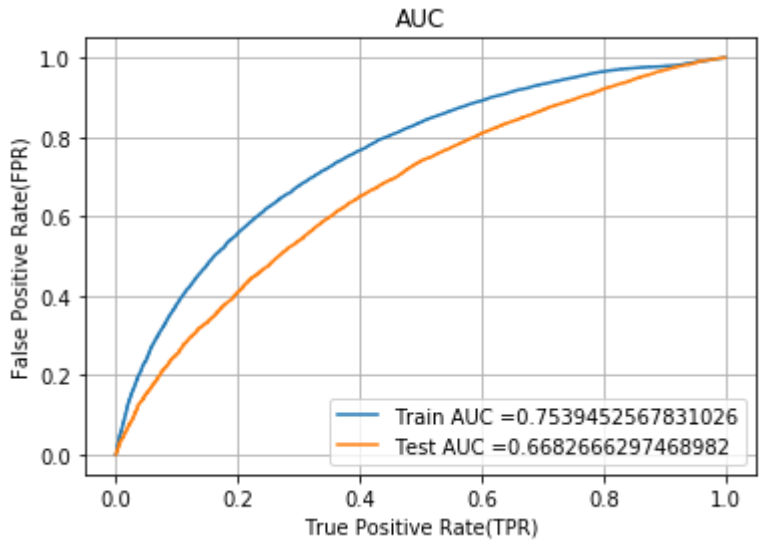
```
sv = SGDClassifier(loss='hinge', penalty='l2',class_weight = 'balanced')
alpha_vals = [10**x for x in range(-4,4)]

parameters = {'alpha':alpha_vals}
clf = GridSearchCV(sv, parameters, cv= 2, scoring='roc_auc',n_jobs=-1,return_train_score=Tr
clf.fit(X_tr, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']

plt.figure(figsize=(20,10))

plt.plot(parameters['alpha'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['alpha'],train_auc - train_auc_std,train_auc + train_auc_

plt.plot(parameters['alpha'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['alpha'],cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.

plt.scatter(parameters['alpha'], train_auc, label='Train AUC points')
plt.scatter(parameters['alpha'], cv_auc, label='CV AUC points')
plt.xscale('log')

plt.legend()
plt.xlabel("lambda: hyperparameter")
plt.ylabel("AUC")
plt.title("Lambda: hyperparameter v/s AUC plot")
plt.grid()
plt.show()
```



B) Train model using the best hyper-parameter value

In [119]:

```
# https://scikitlearn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.m
from sklearn.metrics import roc_curve, auc
model = SGDClassifier(loss='hinge', penalty='l2', alpha=0.01,class_weight = 'balanced')
model.fit(X_tr, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the p
# not the predicted outputs
y_train_pred = model.decision_function(X_tr)
y_test_pred = model.decision_function(X_te)
train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)
plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
```



D) Confusion Matrix

In [120]:

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

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

    # (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
    predictions = []
    for i in proba:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

Train Data

In [121]:

```
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr))
```

```
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.2499999818661462 for threshold -0.412
[[ 3714  3712]
 [ 6773 34842]]
```

In [122]:

```
conf_matr_df_train_1 = pd.DataFrame(confusion_matrix(y_train, predict(y_train_pred, tr_thre
```

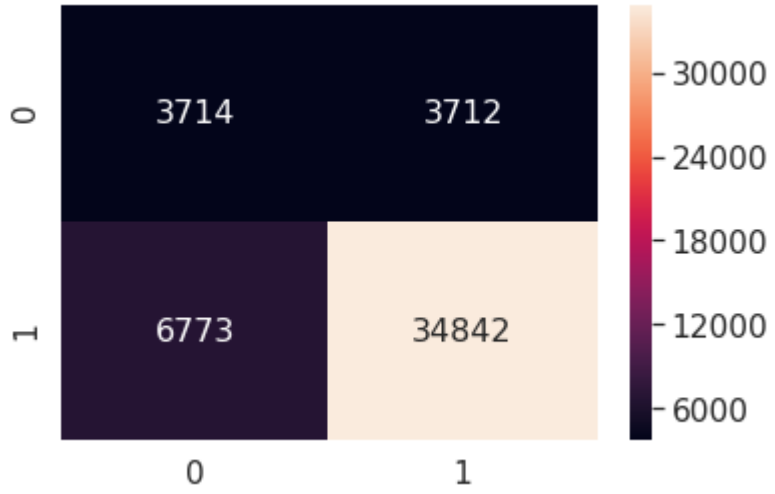
```
the maximum value of tpr*(1-fpr) 0.2499999818661462 for threshold -0.412
```

In [123]:

```
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_train_1, annot=True,annot_kws={"size": 16}, fmt='g')
```

Out[123]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8adae13898>
```



Test Data

In [124]:

```
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr)))
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24999999161092998 for threshold -0.196
[[ 2857  2602]
 [ 8653 21940]]
```

In [125]:

```
conf_matr_df_test_2 = pd.DataFrame(confusion_matrix(y_test, predict(y_test_pred, tr_thresho
```

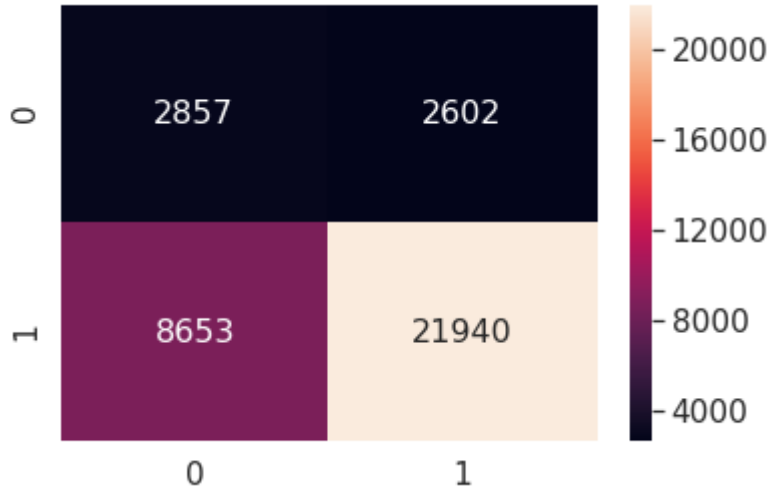
```
the maximum value of tpr*(1-fpr) 0.24999999161092998 for threshold -0.196
```

In [126]:

```
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_test_2, annot=True,annot_kws={"size": 16}, fmt='g')
```

Out[126]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8ad8d6f8d0>
```



Set 2 : categorical, numerical features + project_title(TFIDF) + preprocessed_essay (TFIDF)

In [127]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack

X_tr = hstack((categories_one_hot_train, sub_categories_one_hot_train, school_state_categor
X_te = hstack((categories_one_hot_test, sub_categories_one_hot_test, school_state_categorie
X_cr = hstack((categories_one_hot_cv, sub_categories_one_hot_cv, school_state_categories_or
```

In [128]:

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

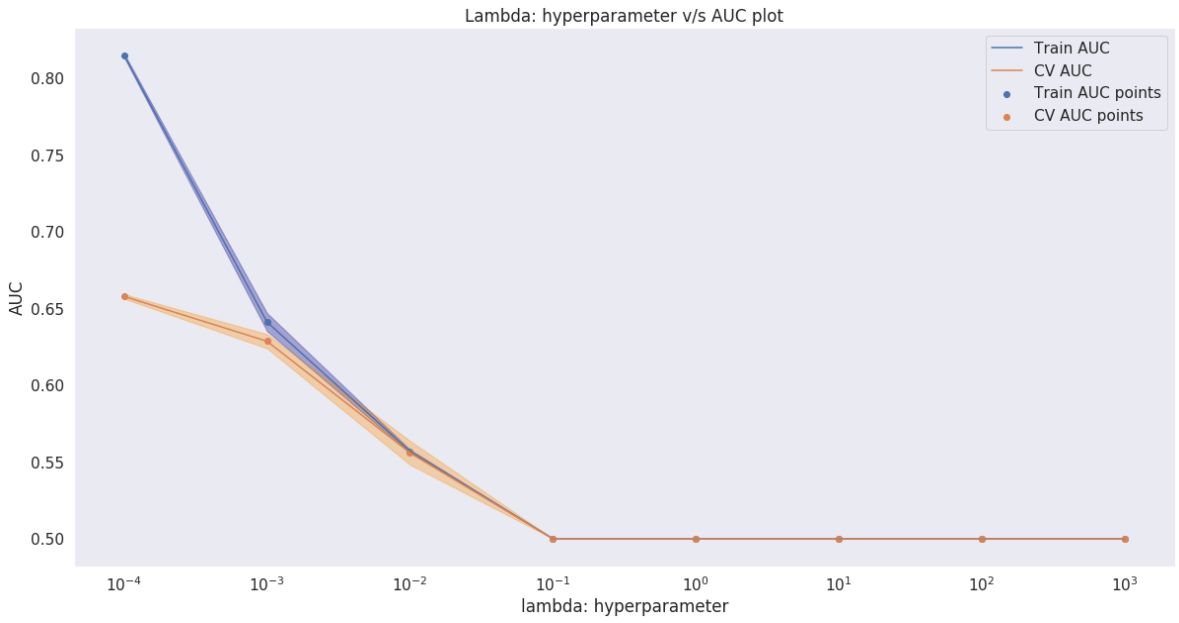
```
Final Data matrix
(49041, 6332) (49041,)
(24155, 6332) (24155,)
(36052, 6332) (36052,)
```

GridSearch CV

with L1 Regularizer

In [129]:

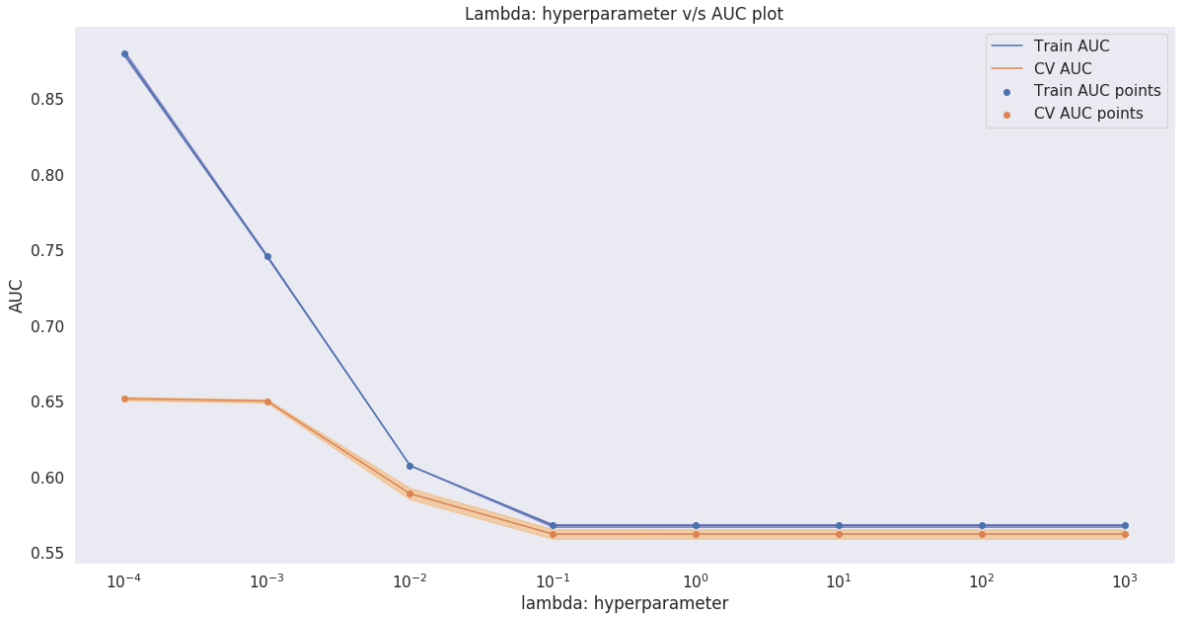
```
sv = SGDClassifier(loss='hinge', penalty='l1',class_weight = 'balanced')
alpha_vals = [10**x for x in range(-4,4)]

parameters = {'alpha':alpha_vals}
clf = GridSearchCV(sv, parameters, cv= 2, scoring='roc_auc',n_jobs=-1,return_train_score=Tr
clf.fit(X_tr, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']

plt.figure(figsize=(20,10))

plt.plot(parameters['alpha'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['alpha'],train_auc - train_auc_std,train_auc + train_auc_

plt.plot(parameters['alpha'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['alpha'],cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.

plt.scatter(parameters['alpha'], train_auc, label='Train AUC points')
plt.scatter(parameters['alpha'], cv_auc, label='CV AUC points')
plt.xscale('log')

plt.legend()
plt.xlabel("lambda: hyperparameter")
plt.ylabel("AUC")
plt.title("Lambda: hyperparameter v/s AUC plot")
plt.grid()
plt.show()
```



With L2 Regualrizer

In [130]:

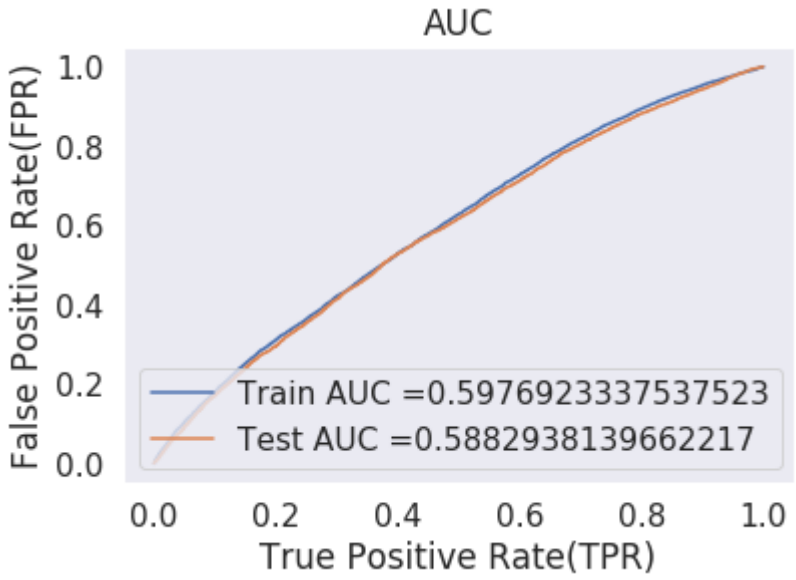
```
sv = SGDClassifier(loss='hinge', penalty='l2',class_weight = 'balanced')
alpha_vals = [10**x for x in range(-4,4)]

parameters = {'alpha':alpha_vals}
clf = GridSearchCV(sv, parameters, cv= 2, scoring='roc_auc',n_jobs=-1,return_train_score=Tr
clf.fit(X_tr, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']

plt.figure(figsize=(20,10))

plt.plot(parameters['alpha'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['alpha'],train_auc - train_auc_std,train_auc + train_auc_

plt.plot(parameters['alpha'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['alpha'],cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.

plt.scatter(parameters['alpha'], train_auc, label='Train AUC points')
plt.scatter(parameters['alpha'], cv_auc, label='CV AUC points')
plt.xscale('log')

plt.legend()
plt.xlabel("lambda: hyperparameter")
plt.ylabel("AUC")
plt.title("Lambda: hyperparameter v/s AUC plot")
plt.grid()
plt.show()
```



Train model using the best hyper-parameter value

In [131]:

```
# https://scikitlearn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.m
from sklearn.metrics import roc_curve, auc
model = SGDClassifier(loss='hinge', penalty='l2', alpha=0.01,class_weight = 'balanced')
model.fit(X_tr, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the p
# not the predicted outputs
y_train_pred = model.decision_function(X_tr)
y_test_pred = model.decision_function(X_te)
train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)
plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
```



Confusion Matrix -Train data

In [132]:

```
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr))
```

```
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.25 for threshold -0.177
[[ 3713  3713]
 [15490 26125]]
```

In [133]:

```
conf_matr_df_train_3 = pd.DataFrame(confusion_matrix(y_train, predict(y_train_pred, tr_thre
```

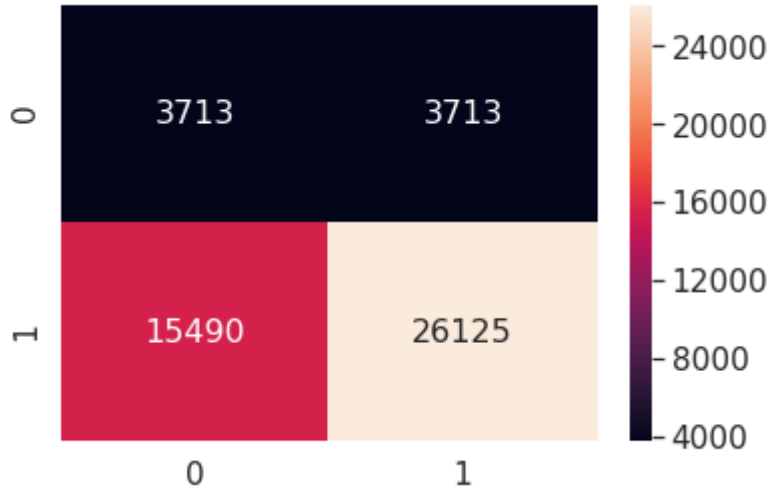
```
the maximum value of tpr*(1-fpr) 0.25 for threshold -0.177
```

In [134]:

```
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_train_3, annot=True,annot_kws={"size": 16}, fmt='g')
```

Out[134]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8ae284bba8>
```



Test Data

In [135]:

```
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr)))
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24999999161092998 for threshold 0.147
[[ 3448  2011]
 [15588 15005]]
```

In [136]:

```
conf_matr_df_test_4 = pd.DataFrame(confusion_matrix(y_test, predict(y_test_pred, tr_thresho
```

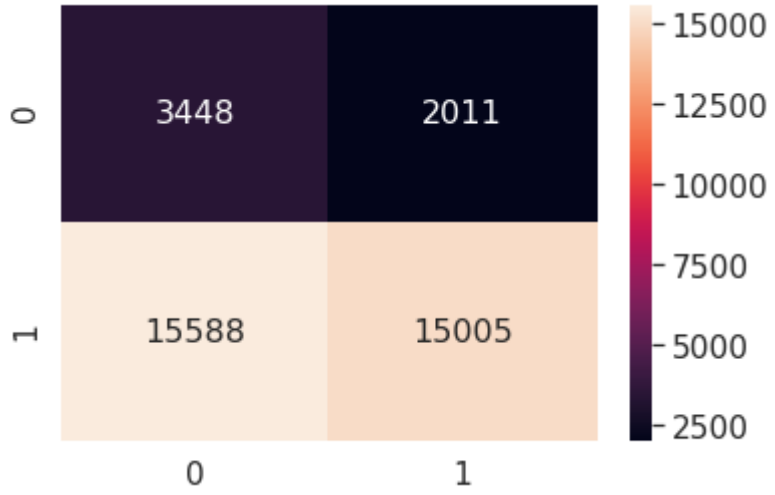
```
the maximum value of tpr*(1-fpr) 0.24999999161092998 for threshold 0.147
```

In [137]:

```
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_test_4, annot=True,annot_kws={"size": 16}, fmt='g')
```

Out[137]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8ae6971f98>
```



Set 3 : Categorical, Numerical features + Project_title(AVG W2V) + Preprocessed_essay (AVG W2V)

In [138]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack

X_tr = hstack((categories_one_hot_train, sub_categories_one_hot_train,school_state_categori
X_te = hstack((categories_one_hot_test, sub_categories_one_hot_test,school_state_categories
X_cr = hstack((categories_one_hot_cv, sub_categories_one_hot_cv,school_state_categories_one
```

In [139]:

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

```
Final Data matrix
(49041, 709) (49041,)
(24155, 709) (24155,)
(36052, 709) (36052,)
```

Gridsearch CV

With L1 Regularizer

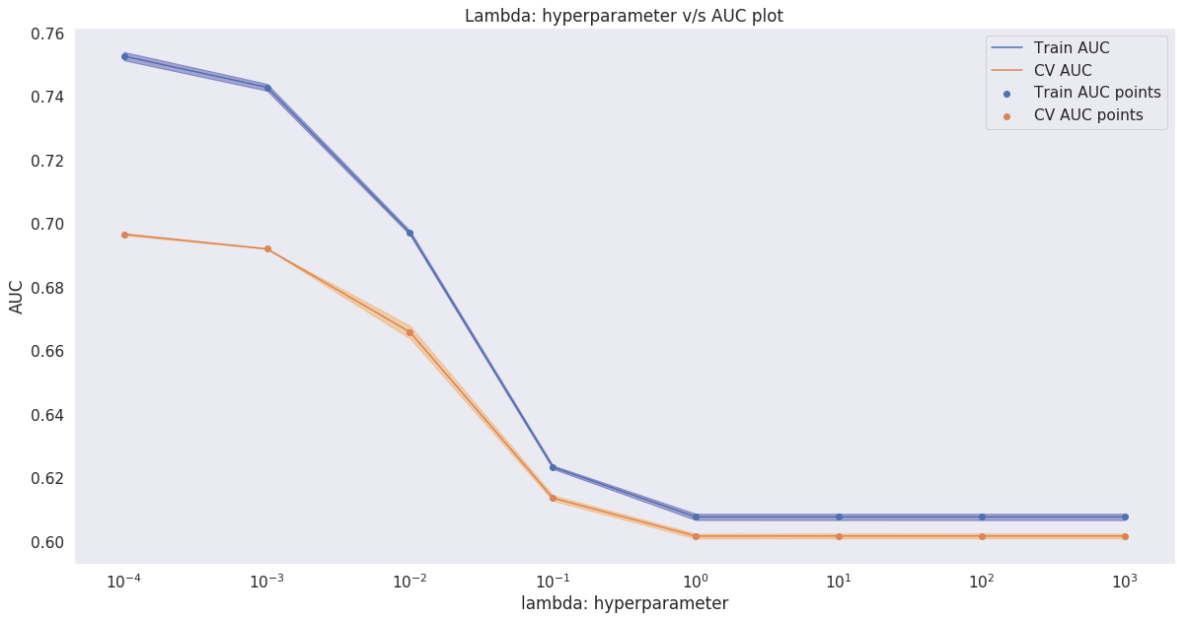
In [141]:

```
sv = SGDClassifier(loss='hinge', penalty='l1',class_weight = 'balanced')
alpha_vals = [10**x for x in range(-4,4)]

parameters = {'alpha':alpha_vals}
clf = GridSearchCV(sv, parameters, cv= 2, scoring='roc_auc',n_jobs=-1,return_train_score=Tr
clf.fit(X_tr, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']

plt.figure(figsize=(20,10))

plt.plot(parameters['alpha'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['alpha'],train_auc - train_auc_std,train_auc + train_auc_

plt.plot(parameters['alpha'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['alpha'],cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.

plt.scatter(parameters['alpha'], train_auc, label='Train AUC points')
plt.scatter(parameters['alpha'], cv_auc, label='CV AUC points')
plt.xscale('log')

plt.legend()
plt.xlabel("lambda: hyperparameter")
plt.ylabel("AUC")
plt.title("Lambda: hyperparameter v/s AUC plot")
plt.grid()
plt.show()
```

With L2 regularizer

In [142]:

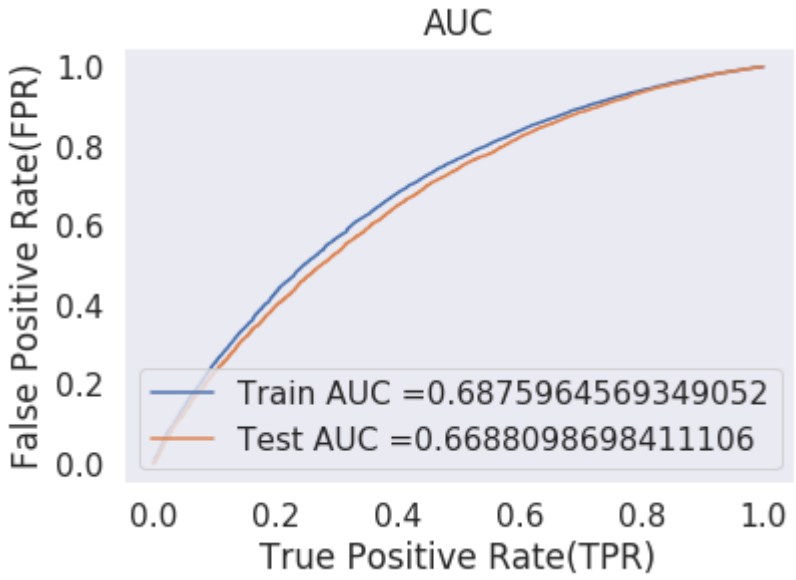
```
sv = SGDClassifier(loss='hinge', penalty='l2',class_weight = 'balanced')
alpha_vals = [10**x for x in range(-4,4)]

parameters = {'alpha':alpha_vals}
clf = GridSearchCV(sv, parameters, cv= 2, scoring='roc_auc',n_jobs=-1,return_train_score=Tr
clf.fit(X_tr, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']

plt.figure(figsize=(20,10))

plt.plot(parameters['alpha'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['alpha'],train_auc - train_auc_std,train_auc + train_auc_

plt.plot(parameters['alpha'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['alpha'],cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.

plt.scatter(parameters['alpha'], train_auc, label='Train AUC points')
plt.scatter(parameters['alpha'], cv_auc, label='CV AUC points')
plt.xscale('log')

plt.legend()
plt.xlabel("lambda: hyperparameter")
plt.ylabel("AUC")
plt.title("Lambda: hyperparameter v/s AUC plot")
plt.grid()
plt.show()
```



B) Train the model using the best hyper parameter value

In [143]:

```
# https://scikitlearn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.m
from sklearn.metrics import roc_curve, auc
model = SGDClassifier(loss='hinge', penalty='l2', alpha=0.01,class_weight = 'balanced')
model.fit(X_tr, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the p
# not the predicted outputs
y_train_pred = model.decision_function(X_tr)
y_test_pred = model.decision_function(X_te)
train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)
plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
```



C) Confusion Matrix

Train data

In [144]:

```
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr))
```

```
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.25 for threshold -0.232
[[ 3713  3713]
 [ 9648 31967]]
```

In [145]:

```
conf_matr_df_train_5 = pd.DataFrame(confusion_matrix(y_train, predict(y_train_pred, tr_thre
```

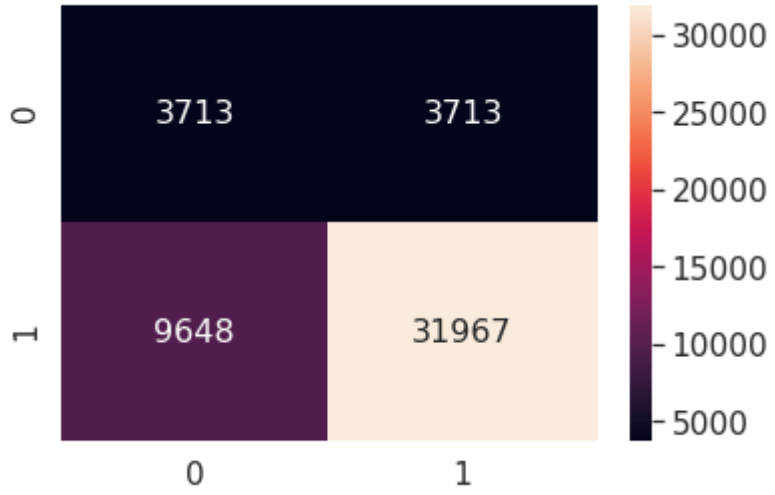
```
the maximum value of tpr*(1-fpr) 0.25 for threshold -0.232
```

In [146]:

```
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_train_5, annot=True,annot_kws={"size": 16}, fmt='g')
```

Out[146]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8ae677b198>
```



Test data

In [147]:

```
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr)))
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24999999161092998 for threshold 0.025
[[ 3394  2065]
 [11473 19120]]
```

In [148]:

```
conf_matr_df_test_6 = pd.DataFrame(confusion_matrix(y_test, predict(y_test_pred, tr_thresho
```

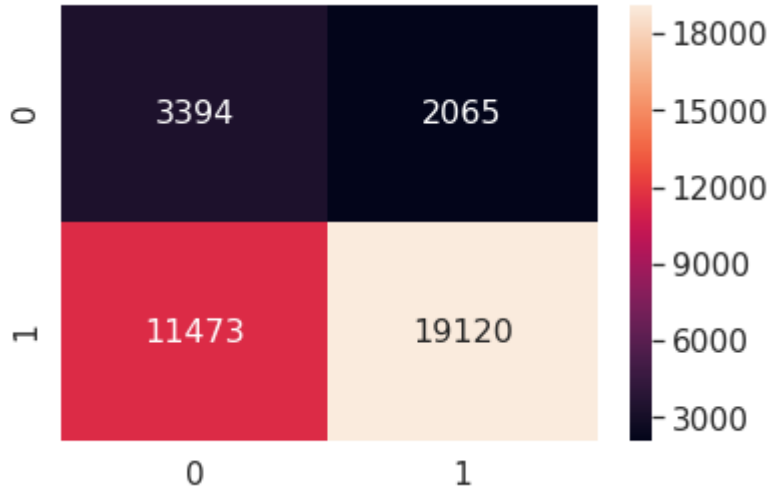
```
the maximum value of tpr*(1-fpr) 0.24999999161092998 for threshold 0.025
```

In [149]:

```
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_test_6, annot=True,annot_kws={"size": 16}, fmt='g')
```

Out[149]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8ad00b6e10>
```



Set 4 : Categorical, Numerical features + Project_title(TFIDF W2V) + Preprocessed_essay (TFIDF W2V)

In [150]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack

X_tr = hstack((categories_one_hot_train, sub_categories_one_hot_train,school_state_categori
X_te = hstack((categories_one_hot_test, sub_categories_one_hot_test,school_state_categories
X_cr = hstack((categories_one_hot_cv, sub_categories_one_hot_cv,school_state_categories_one
```

In [151]:

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

```
Final Data matrix
(49041, 709) (49041,)
(24155, 709) (24155,)
(36052, 709) (36052,)
```

GridSearchCV

With L1 regualrizer

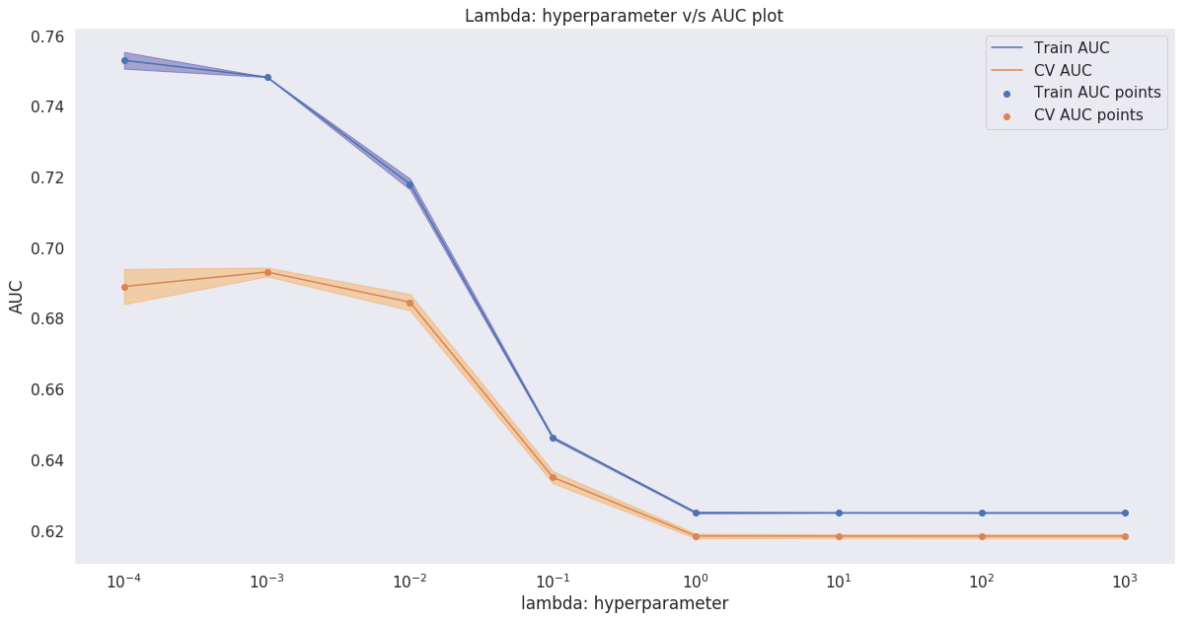
In [152]:

```
sv = SGDClassifier(loss='hinge', penalty='l1',class_weight = 'balanced')
alpha_vals = [10**x for x in range(-4,4)]

parameters = {'alpha':alpha_vals}
clf = GridSearchCV(sv, parameters, cv= 2, scoring='roc_auc',n_jobs=-1,return_train_score=Tr
clf.fit(X_tr, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']

plt.figure(figsize=(20,10))

plt.plot(parameters['alpha'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['alpha'],train_auc - train_auc_std,train_auc + train_auc_

plt.plot(parameters['alpha'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['alpha'],cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.

plt.scatter(parameters['alpha'], train_auc, label='Train AUC points')
plt.scatter(parameters['alpha'], cv_auc, label='CV AUC points')
plt.xscale('log')

plt.legend()
plt.xlabel("lambda: hyperparameter")
plt.ylabel("AUC")
plt.title("Lambda: hyperparameter v/s AUC plot")
plt.grid()
plt.show()
```

With L2 Regularizer

In [153]:

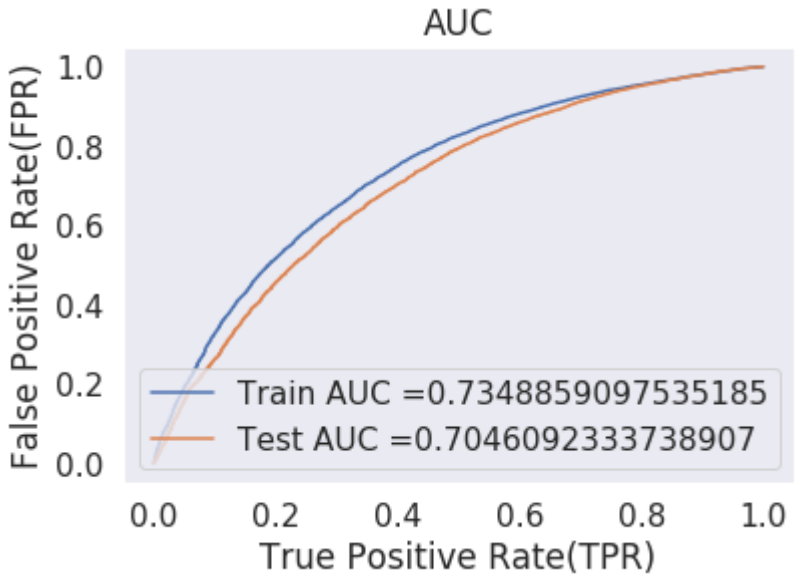
```
sv = SGDClassifier(loss='hinge', penalty='l2',class_weight = 'balanced')
alpha_vals = [10**x for x in range(-4,4)]

parameters = {'alpha':alpha_vals}
clf = GridSearchCV(sv, parameters, cv= 2, scoring='roc_auc',n_jobs=-1,return_train_score=Tr
clf.fit(X_tr, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']

plt.figure(figsize=(20,10))

plt.plot(parameters['alpha'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['alpha'],train_auc - train_auc_std,train_auc + train_auc_

plt.plot(parameters['alpha'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['alpha'],cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.

plt.scatter(parameters['alpha'], train_auc, label='Train AUC points')
plt.scatter(parameters['alpha'], cv_auc, label='CV AUC points')
plt.xscale('log')

plt.legend()
plt.xlabel("lambda: hyperparameter")
plt.ylabel("AUC")
plt.title("Lambda: hyperparameter v/s AUC plot")
plt.grid()
plt.show()
```



Train the model using the best hyper parameter value

In [154]:

```
# https://scikitlearn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.m
from sklearn.metrics import roc_curve, auc
model = SGDClassifier(loss='hinge', penalty='l2', alpha=0.001,class_weight = 'balanced')
model.fit(X_tr, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the p
# not the predicted outputs
y_train_pred = model.decision_function(X_tr)
y_test_pred = model.decision_function(X_te)
train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)
plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
```



Confusion Matrix

In [155]:

```
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr))
```

```
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.25 for threshold -0.331
[[ 3713  3713]
 [ 7200 34415]]
```

In [156]:

```
conf_matr_df_train_7 = pd.DataFrame(confusion_matrix(y_train, predict(y_train_pred, tr_thre
```

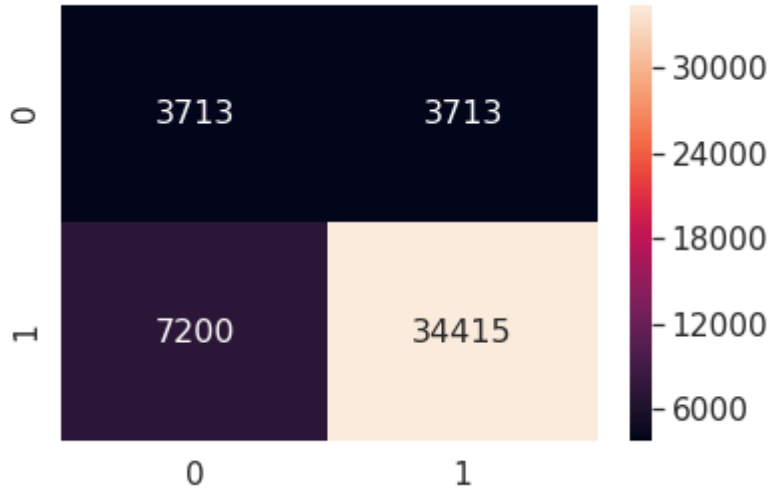
```
the maximum value of tpr*(1-fpr) 0.25 for threshold -0.331
```

In [157]:

```
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_train_7, annot=True,annot_kws={"size": 16}, fmt='g')
```

Out[157]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8ae6bc0128>
```



In [158]:

```
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr)))
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24999999161092998 for threshold -0.027
[[ 3277  2182]
 [ 9071 21522]]
```

In [159]:

```
conf_matr_df_test_8 = pd.DataFrame(confusion_matrix(y_test, predict(y_test_pred, tr_thresho
```

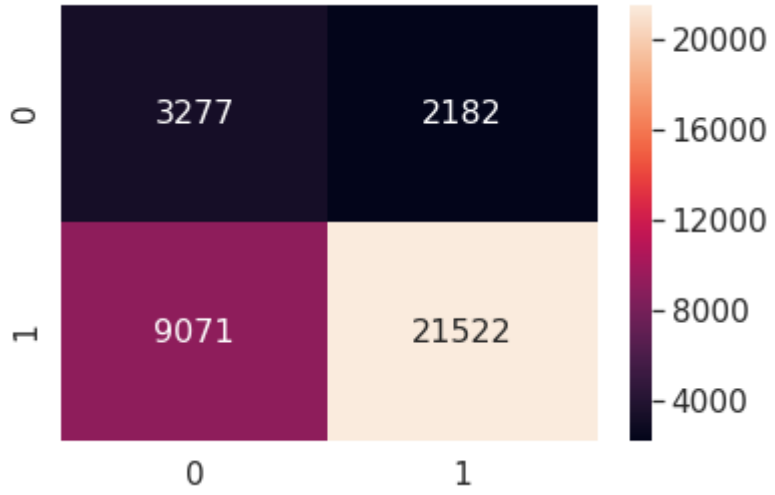
```
the maximum value of tpr*(1-fpr) 0.24999999161092998 for threshold -0.027
```

In [160]:

```
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_test_8, annot=True,annot_kws={"size": 16}, fmt='g')
```

Out[160]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8adbdc4278>
```



Set 5 : : Categorical features, Numerical features by TruncatedSVD on TfidfVectorizer

In [163]:

```
from sklearn.decomposition import TruncatedSVD
index = [5,10,50,100,250,500,1000,2500]
variance_sum = []
for i in tqdm(index):
    svd = TruncatedSVD(n_components= i, n_iter=7, random_state=42)
    svd.fit(text_tfidf_train)
    variance_sum.append(svd.explained_variance_ratio_.sum())
```

```
  0%|          | 0/8 [00:00<?, ?it/s]
 12%|▉         | 1/8 [00:00<00:04,  1.51it/s]
 25%|██        | 2/8 [00:01<00:04,  1.45it/s]
 38%|███▊      | 3/8 [00:05<00:07,  1.56s/it]
 50%|█████     | 4/8 [00:11<00:12,  3.02s/it]
 62%|██████▎   | 5/8 [00:28<00:21,  7.19s/it]
 75%|███████▌  | 6/8 [01:05<00:32, 16.30s/it]
 88%|████████▊ | 7/8 [02:30<00:36, 36.80s/it]
100%|██████████| 8/8 [06:56<00:00, 105.63s/it]
```

In [164]:

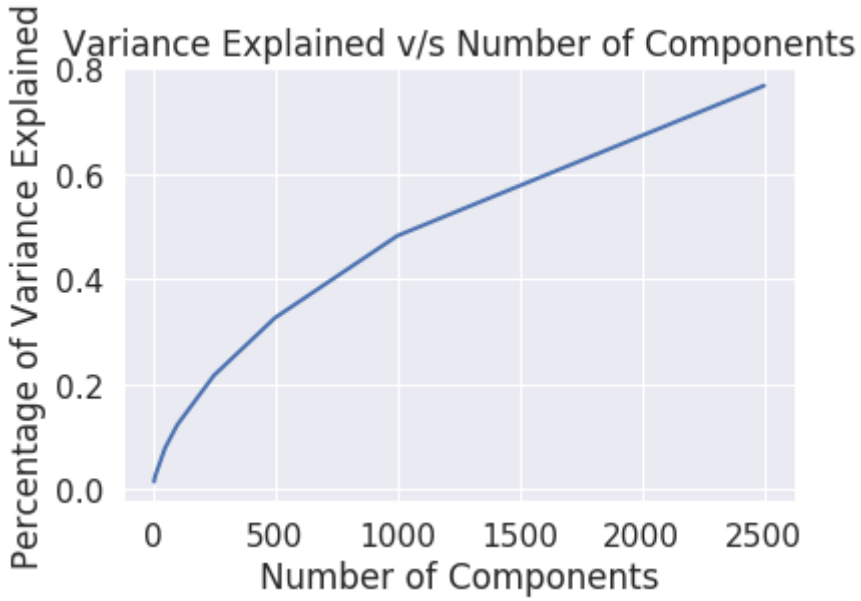
```
variance_sum
```

Out[164]:

```
[0.014779396519832781,
 0.0253627334636365,
 0.07833137540022103,
 0.12257721533390073,
 0.2167237234684663,
 0.32640412405049907,
 0.4823872068082544,
 0.7681659572229157]
```

In [165]:

```
plt.xlabel("Number of Components")
plt.ylabel("Percentage of Variance Explained")
plt.title("Variance Explained v/s Number of Components")
plt.plot(index,variance_sum,lw=2)
plt.show()
```



In [167]:

```
svd = TruncatedSVD(n_components= 2500, n_iter=7, random_state=42)
svd.fit(text_tfidf_train)
svd_train = svd.transform(text_tfidf_train)
svd_test = svd.transform(text_tfidf_test)
svd_cv = svd.transform(text_tfidf_cv)
print("Shape of matrix after Decomposition ",svd_train.shape)
print("Shape of matrix after Decomposition ",svd_test.shape)
print("Shape of matrix after Decomposition ",svd_cv.shape)
```

```
Shape of matrix after Decomposition  (49041, 2500)
Shape of matrix after Decomposition  (36052, 2500)
Shape of matrix after Decomposition  (24155, 2500)
```

In [168]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack

X_tr = hstack((categories_one_hot_train, sub_categories_one_hot_train,school_state_categori
X_te = hstack((categories_one_hot_test, sub_categories_one_hot_test,school_state_categories
X_cr = hstack((categories_one_hot_cv, sub_categories_one_hot_cv,school_state_categories_one
```

In [169]:

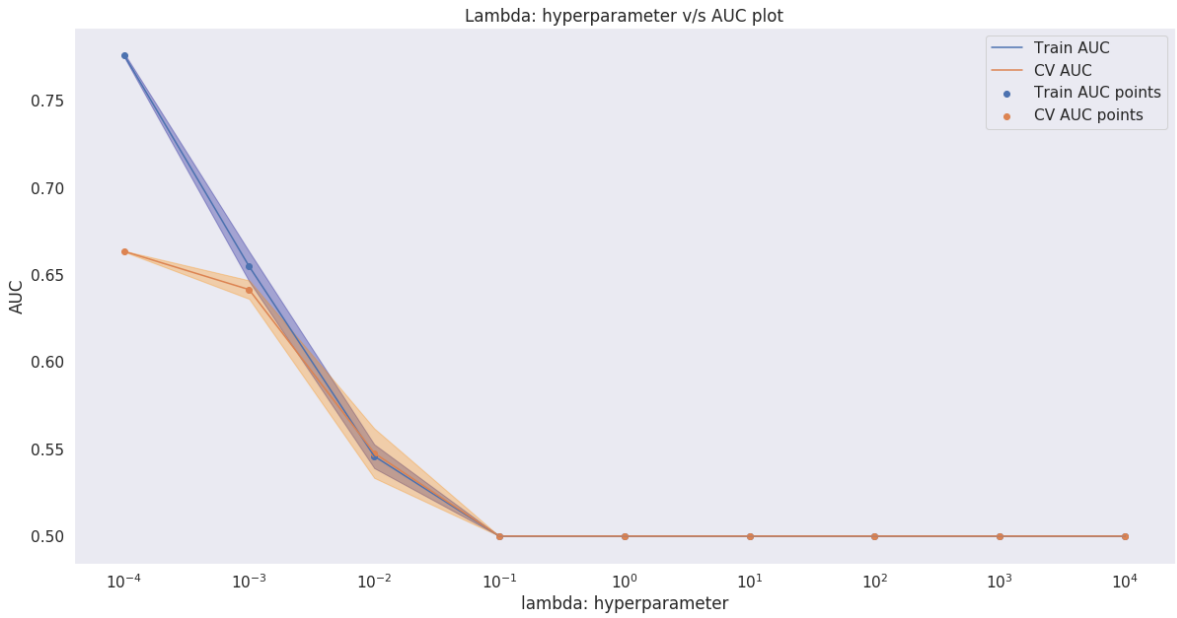
```
print("Final Data matrix")
print(X_tr.shape, y_train.shape)
print(X_cr.shape, y_cv.shape)
print(X_te.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(49041, 2609) (49041,)
(24155, 2609) (24155,)
(36052, 2609) (36052,)
====================================================================================
========================
```

GridSearch CV with L1 regualarizer

In [170]:

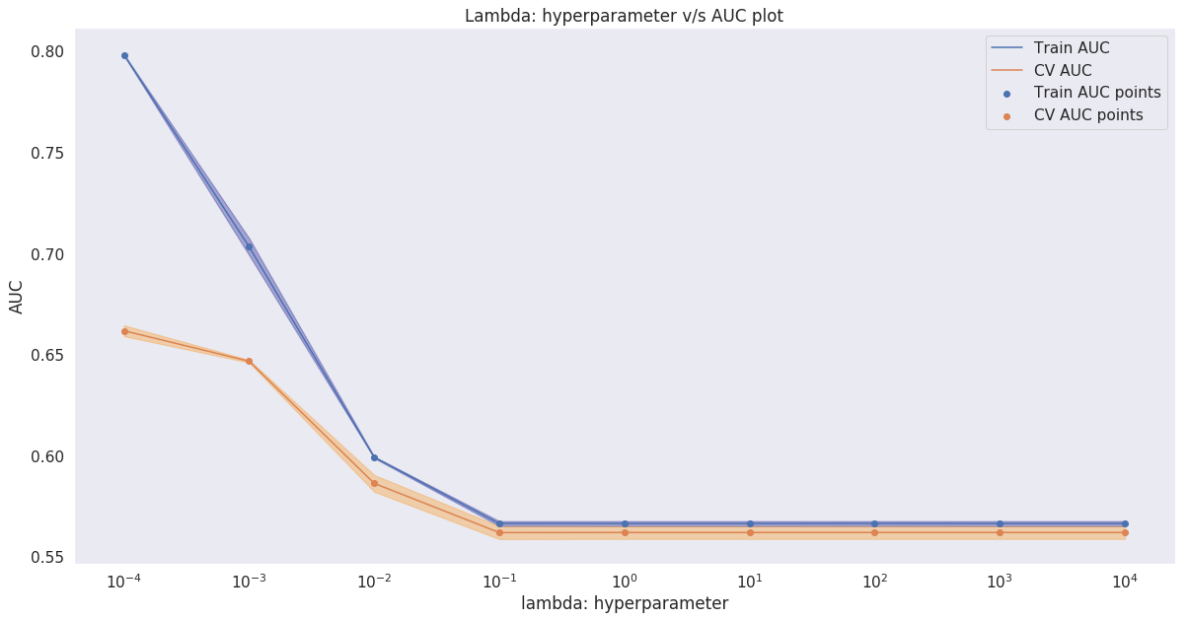
```
sv = SGDClassifier(loss='hinge', penalty='l1',class_weight ='balanced')

parameters = {'alpha':[10**-4, 10**-3, 10**-2, 10**-1, 10**0, 10**1, 10**2, 10**3, 10**4]}

clf = GridSearchCV(sv, parameters, cv=2, scoring='roc_auc',return_train_score=True)
clf.fit(X_tr, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']

plt.figure(figsize=(20,10))

plt.plot(parameters['alpha'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['alpha'],train_auc - train_auc_std,train_auc + train_auc_

plt.plot(parameters['alpha'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['alpha'],cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.

plt.scatter(parameters['alpha'], train_auc, label='Train AUC points')
plt.scatter(parameters['alpha'], cv_auc, label='CV AUC points')

plt.xscale('log')
plt.legend()
plt.xlabel("lambda: hyperparameter")
plt.ylabel("AUC")
plt.title("Lambda: hyperparameter v/s AUC plot")
plt.grid()
plt.show()
```



GridSearch with L2 regularizer

In [171]:

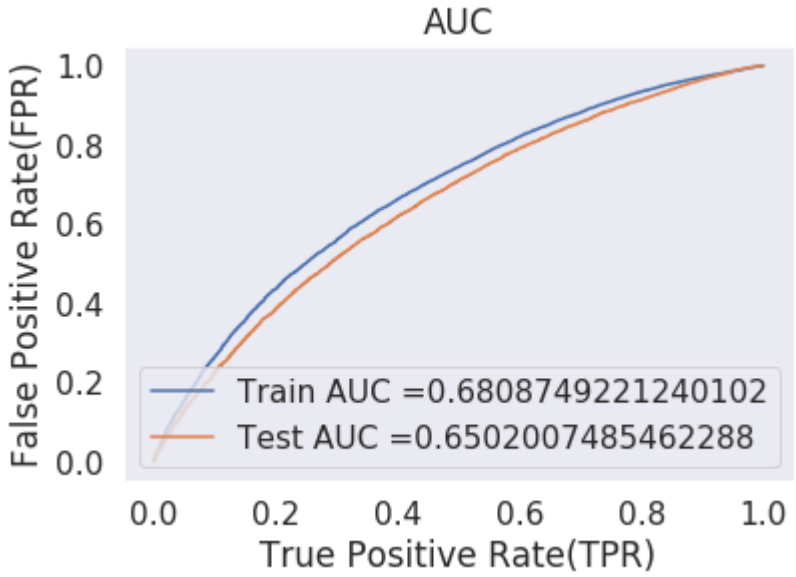
```
sv = SGDClassifier(loss='hinge', penalty='l2',class_weight ='balanced')

parameters = {'alpha':[10**-4, 10**-3, 10**-2, 10**-1, 10**0, 10**1, 10**2, 10**3, 10**4]}

clf = GridSearchCV(sv, parameters, cv=2, scoring='roc_auc',return_train_score=True)
clf.fit(X_tr, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']

plt.figure(figsize=(20,10))

plt.plot(parameters['alpha'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['alpha'],train_auc - train_auc_std,train_auc + train_auc_

plt.plot(parameters['alpha'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['alpha'],cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.

plt.scatter(parameters['alpha'], train_auc, label='Train AUC points')
plt.scatter(parameters['alpha'], cv_auc, label='CV AUC points')

plt.xscale('log')
plt.legend()
plt.xlabel("lambda: hyperparameter")
plt.ylabel("AUC")
plt.title("Lambda: hyperparameter v/s AUC plot")
plt.grid()
plt.show()
```



Train Model using best Hyperparameter Value

In [180]:

```
# https://scikitlearn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.m
from sklearn.metrics import roc_curve, auc
model = SGDClassifier(loss='hinge', penalty='l2', alpha= 0.001,class_weight ='balanced')
model.fit(X_tr, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the p
# not the predicted outputs
y_train_pred = model.decision_function(X_tr)
y_test_pred = model.decision_function(X_te)
train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)
plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
```



Confusion Matrix

In [181]:

```
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr))
```

```
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.25 for threshold -0.243
[[ 3713  3713]
 [10642 30973]]
```

In [182]:

```
conf_matr_df_train_9 = pd.DataFrame(confusion_matrix(y_train, predict(y_train_pred, tr_thre
```

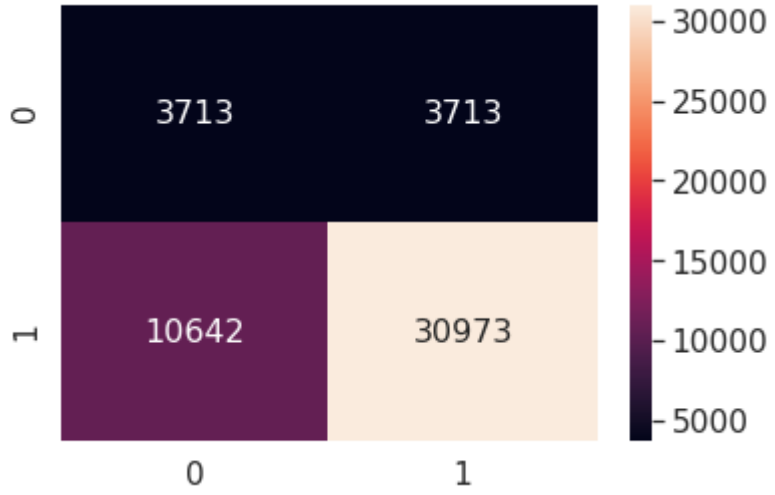
```
the maximum value of tpr*(1-fpr) 0.25 for threshold -0.243
```

In [183]:

```
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_train_9, annot=True,annot_kws={"size": 16}, fmt='g')
```

Out[183]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8ae6ce6c18>
```



In [184]:

```
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr)))
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24999999161092998 for threshold -0.014
[[ 3274  2185]
 [11653 18940]]
```

In [185]:

```
conf_matr_df_test_10 = pd.DataFrame(confusion_matrix(y_test, predict(y_test_pred, tr_thresh
```

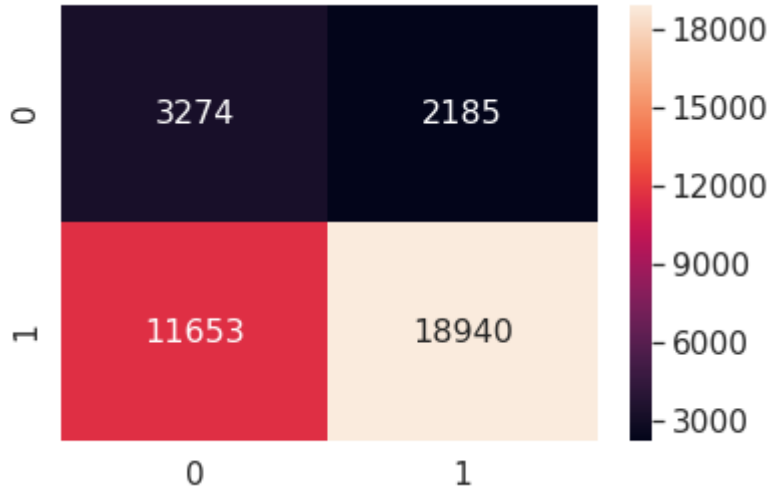
```
the maximum value of tpr*(1-fpr) 0.24999999161092998 for threshold -0.014
```

In [186]:

```
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_test_10, annot=True,annot_kws={"size": 16}, fmt='g')
```

Out[186]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8ada058438>
```



In [187]:

```
from prettytable import PrettyTable

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

x = PrettyTable()
x.field_names = ["Vectorizer", "Model", "Alpha:Hyper Parameter", "AUC"]

x.add_row(["BOW", "SVM", 0.01, 0.75])
x.add_row(["TFIDF", "SVM", 0.01, 0.59])
x.add_row(["AVG W2V", "SVM", 0.01, 0.687])
x.add_row(["TFIDF W2V", "SVM", 0.001, 0.73])
x.add_row(["TRUNCATED SVD", "SVM", 0.001, 0.68])


print(x)
```

```
+---------------+-------+----------------------+-------+
|   Vectorizer  | Model | Alpha:Hyper Parameter |  AUC  |
+---------------+-------+----------------------+-------+
|      BOW      |  SVM  |         0.01         |  0.75 |
|     TFIDF     |  SVM  |         0.01         |  0.59 |
|    AVG W2V    |  SVM  |         0.01         | 0.687 |
|   TFIDF W2V   |  SVM  |        0.001         |  0.73 |
| TRUNCATED SVD |  SVM  |        0.001         |  0.68 |
+---------------+-------+----------------------+-------+
```

In [ ]:

```

```