DonorsChoose

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

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

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

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

About the DonorsChoose Data Set

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

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

- Literature & Writing, Social Sciences

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

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

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

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

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

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

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

Notes on the Essay Data

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

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

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 [257]:

```
%matplotlib inline
import warnings
warnings.filterwarnings(action='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
from tqdm import tqdm_notebook
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 [260]:

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

In [261]:

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

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

In [262]:

```
# 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[262]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_cate
55660	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	CA	2016-04-27 00:27:36	Grades PreK-2
76127	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	UT	2016-04-27 00:31:25	Grades 3-5

In [263]:

```
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[263]:

	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 [264]:

```
project_grade_category = []

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

In [265]:

```
project_grade_category[0:5]
```

Out[265]:

```
['Grades_PreK-2', 'Grades_6-8', 'Grades_6-8', 'Grades_PreK-2', 'Grades_PreK-2']
```

In [266]:

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

project_data["project_grade_category"] = project_grade_category
```

In [267]:

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

Out[267]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_subject_cat
55660	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	CA	2016-04-27 00:27:36	Math & Science
76127	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	UT	2016-04-27 00:31:25	Special Needs
51140	74477	p189804	4a97f3a390bfe21b99cf5e2b81981c73	Mrs.	CA	2016-04-27 00:46:53	Literacy & Language
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	GA	2016-04-27 00:53:00	Applied Learning
41558	33679	p137682	06f6e62e17de34fcf81020c77549e1d5	Mrs.	WA	2016-04-27 01:05:25	Literacy & Language

1.2 preprocessing of `project_subject_categories`

In [268]:

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

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

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

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

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

1.3 preprocessing of `project_subject_subcategories`

In [269]:

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

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

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

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

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

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

preprocessing of project_grade_category

In [270]:

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

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

grade_cat_list = []
for category in grade_catogories:
    category =str(category)
    category=category.replace('-','_')

    grade_cat_list.append(category.strip())

project_data['clean_grade_categories'] = grade_cat_list
project_data.drop(['project_grade_category'], 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_grade_categories'].values:
    my_counter.update(word.split())

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

preprocessing of teacher_prefix

In [271]:

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

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

teacher_prefix_list = []
for prefix in teacher_prefixes:
    prefix =str(prefix)
    prefix=prefix.replace('.','')
    teacher_prefix_list.append(prefix.strip())

project_data['clean_teacher_prefix'] = teacher_prefix_list
project_data.drop(['teacher_prefix'], 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_teacher_prefix'].values:
    my_counter.update(word.split())

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

1.4 New feature "Number of Words in Title"

In [272]:

```
title_word_count = []
```

In [273]:

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

In [274]:

```
project_data["title_word_count"] = title_word_count
```

In [275]:

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

Out[275]:

	Unnamed: 0	id	teacher_id	school_state	Date	project_title	project_essay_1	proj
55660	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	CA	2016-04-27 00:27:36	Engineering STEAM into the Primary Classroom	I have been fortunate enough to use the Fairy ...	My s com varie back
					2016-	Sensory	Imagine being 8-	Mos

	Unnamed: 0	id	teacher_id	school_state	Date	project_title	project_essay_1	proj
76127	37728	p043609	3f60494c61921b3b43ab61bdde2904df	UT	04-27 00:31:25	Tools for Focus	9 years old. You're in your th...	stud autis anot
51140	74477	p189804	4a97f3a390bfe21b99cf5e2b81981c73	CA	2016-04-27 00:46:53	Mobile Learning with a Mobile Listening Center	Having a class of 24 students comes with diver...	I hav twer kind stu..
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	GA	2016-04-27 00:53:00	Flexible Seating for Flexible Learning	I recently read an article about giving studen...	I tea inco scho
41558	33679	p137682	06f6e62e17de34fcf81020c77549e1d5	WA	2016-04-27 01:05:25	Going Deep: The Art of Inner Thinking!	My students crave challenge, they eat obstacle...	We a publ elem scho

In [276]:

```
# 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 [277]:

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

Out[277]:

	Unnamed: 0	id	teacher_id	school_state	Date	project_title	project_essay_1	proj
55660	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	CA	2016-04-27 00:27:36	Engineering STEAM into the Primary Classroom	I have been fortunate enough to use the Fairy ...	My s com varie back
76127	37728	p043609	3f60494c61921b3b43ab61bdde2904df	UT	2016-04-27 00:31:25	Sensory Tools for Focus	Imagine being 8-9 years old. You're in your th...	Mos stud autis anot

In [278]:

```
essay_word_count = []
```

In [279]:

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

In [280]:

```
project_data["essay_word_count"] = essay_word_count
```

In [281]:

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

Out[281]:

	Unnamed: 0	id	teacher_id	school_state	Date	project_title	project_essay_1	proj
55660	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	CA	2016-04-27 00:27:36	Engineering STEAM into the Primary Classroom	I have been fortunate enough to use the Fairy ...	My s com varie back
76127	37728	p043609	3f60494c61921b3b43ab61bdde2904df	UT	2016-04-27 00:31:25	Sensory Tools for Focus	Imagine being 8-9 years old. You're in your th...	Mos stud autis anot
51140	74477	p189804	4a97f3a390bfe21b99cf5e2b81981c73	CA	2016-04-27 00:46:53	Mobile Learning with a Mobile Listening Center	Having a class of 24 students comes with diver...	I hav twer kind stu..
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	GA	2016-04-27 00:53:00	Flexible Seating for Flexible Learning	I recently read an article about giving studen...	I tea incoı schc
41558	33679	p137682	06f6e62e17de34fcf81020c77549e1d5	WA	2016-04-27 01:05:25	Going Deep: The Art of Inner Thinking!	My students crave challenge, they eat obstacle...	We ; publ elem schc

Splitting data into Test - Train

In [282]:

```
# 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_approved'], test_size=0.33, stratify = project_data['project_is_approved'
])
#X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33,
stratify=y_train)
```

In [283]:

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

1.3 Text preprocessing

In [284]:

```
#### 1.4.2.3 Using Pretrained Models: TFIDF weighted W2V
```

In [285]:

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

```
I have an amazing class of enthusiastic 3rd graders! We are a diverse group including multiple eth
nic backgrounds, special education, gifted, and ESOL. A look into my classroom reminds you that re
aders come in all shapes and sizes with varying needs, interests, and aspirations.\r\n\r\nThe firs
t weeks of school are all about creating a community of readers, writers, thinkers, and friends.
And we are well on our way! We believe in working hard, setting goals, and celebrating successes,
great and small. It's going to be a great year!\r\n\r\nI want to ensure that my classroom library
reflects the diversity of my students. It is my passion to grow readers and writers and to do that
I need to get the right kinds of texts in their hands. If we are to help children develop and keep
a love of reading, then we need to provide access to books that they love.\r\n\r\nThird graders
crave series books. This project  will provide diverse books for all of my students, taking into c
onsideration reading levels and interests. I've chosen texts for the reluctant readers, the girly
girls, the advanced readers, and many other reading personalities.\r\n\r\nTime, choice, and access
to great books are the ingredients needed to  grow confident, joyful readers. I commit to
providing the time and allow for choice. This project will ensure the access needed to read, thriv
e, and grow.\r\nnannan
==================================================
Children are hands-on learners. It is essential that teachers can provide their students materials
that allow them to learn in ways that meet their various unique needs. Children need a chance to e
xperiment as they learn. \r\n\r\nMy classroom is in Michigan, bordering Detroit just to the
south.\r\nMy students come from a wide range of backgrounds, experiences, and family structures. F
or some of my students, our school is their first structured educational environment, while other
students come from many preschool experiences. \r\nNo one likes to sit all day-especially young ch
ildren!  More and more children are expected to sit to write and work during the school day.  Prov
iding alternative seating options such as these ball chairs will help children who need to move, s
hake, and wobble during the school day as they complete their reading or writing. Both their acade
mic requirements and their body's needs will be met. \r\n\r\nChildren are looking for sensory inpu
t and these ball chairs can provide appropriate sensory input for students which allows for childr
en to focus on their work while moving their bodies.  Having chairs with legs will help the
children keep the bouncy ball under control versus having it go all around the room or used inappr
opriately.  It is vital that teachers provide the tools children need to meet their maximum
potential!nannan
==================================================
I am a 1st grade teacher working in a high needs school. Many of my students receive free or
reduced breakfast and lunch. Some also receive \"powerpacks\" on Fridays to ensure they have food
to eat over the weekend. \r\n\r\nDespite all the obstacles surrounding my students, they come to s
chool enthusiastic and eager to learn each day. The excitement and smiles on their faces when they
are learning and discovering inside our classroom is priceless. These wonderful 6 year olds deserv
e everything I can do for them at school to help them reach their full potential!Understanding the
world around us isn't always easy for 6 and 7 year olds to grasp, especially when we're simply rea
ding about it in a book. I want to provide my students with hands on science materials which will
allow them to learn, explore and understand our world in a very meaningful way. \r\nHaving these m
aterials will provide enriching learning experiences that these children can carry with them for a
lifetime.\r\nWe will learn about weather, volcanoes, plants, force, motion and witness first hand
the life cycles of butterflies and ladybugs. As the Chinese Proverb goes, \"Tell me and I'll forge
t, show me and I may remember, involve me and I'll understand\". I want my students to understand
as much as possible about the world around them.nannan
==================================================
As a teacher in a low-income-high poverty school, my students are faced with several challenges in
and out of the classroom. In spite of the many challenges they face, I am looking to keep things s
imple and provide my students with flexible and meaningful learning experiences.\r\n\r\n\r\nMy stu
dents attend a Title 1 school in the eastern part of Florida. They like to move, they love to read
and love and need lots of positive attention. Most of them are being raised by a single parent and
/or live with grandparents and receive a free breakfast and lunch based on their socioeconomic
status. Many of my students consider school their safe place.   From the minute they walk in the d
oor of my classroom I focus on their potential and growth while they are with me. I may not be abl
e to control their home lives, however, I can certainly control their experience during the school
day.I am requesting 4 Hokki Stools for my flexible seating. Students are not all the same and they
do not learn the same. My job as a teacher is to find what works for each and everyone of my stude
nts. I get to stand, walk and move around the class as I please. My students are very active and n
nts. I get to stand, walk and move around the class as I please. My students are very active and n
eed to move as well. This year I am starting off with flexible seating to allow my students the mo
vement that we all need. \r\n\r\nBy adding flexible seating such as Hokki Stools to our small
group area and/or computer area, students will be allow to move and continue their active
learning.  Flexible seating and the Hokki Stools will allow the students to choose the type of sea
ting they need at that time to allow them to stay focused on their learning. Six hours in a
classroom, who wouldn't want to be comfortable .\r\nnannan
==================================================
```

In [286]:

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

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

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

In [287]:

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

```
As a teacher in a low-income-high poverty school, my students are faced with several challenges in
and out of the classroom. In spite of the many challenges they face, I am looking to keep things s
imple and provide my students with flexible and meaningful learning experiences.\r\n\r\n\r\nMy stu
dents attend a Title 1 school in the eastern part of Florida. They like to move, they love to read
and love and need lots of positive attention. Most of them are being raised by a single parent and
/or live with grandparents and receive a free breakfast and lunch based on their socioeconomic
status. Many of my students consider school their safe place.   From the minute they walk in the d
oor of my classroom I focus on their potential and growth while they are with me. I may not be abl
e to control their home lives, however, I can certainly control their experience during the school
day.I am requesting 4 Hokki Stools for my flexible seating. Students are not all the same and they
do not learn the same. My job as a teacher is to find what works for each and everyone of my stude
nts. I get to stand, walk and move around the class as I please. My students are very active and n
eed to move as well. This year I am starting off with flexible seating to allow my students the mo
vement that we all need. \r\n\r\nBy adding flexible seating such as Hokki Stools to our small
group area and/or computer area, students will be allow to move and continue their active
learning.  Flexible seating and the Hokki Stools will allow the students to choose the type of sea
ting they need at that time to allow them to stay focused on their learning. Six hours in a
classroom, who would not want to be comfortable .\r\nnannan
==================================================
```

In [288]:

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

```
As a teacher in a low-income-high poverty school, my students are faced with several challenges in
and out of the classroom. In spite of the many challenges they face, I am looking to keep things s
imple and provide my students with flexible and meaningful learning experiences.      My students
attend a Title 1 school in the eastern part of Florida. They like to move, they love to read and l
ove and need lots of positive attention. Most of them are being raised by a single parent and/or l
ive with grandparents and receive a free breakfast and lunch based on their socioeconomic status.
Many of my students consider school their safe place.   From the minute they walk in the door of m
y classroom I focus on their potential and growth while they are with me. I may not be able to con
trol their home lives, however, I can certainly control their experience during the school day.I a
```

```
m requesting 4 Hokki Stools for my flexible seating. Students are not all the same and they do not
learn the same. My job as a teacher is to find what works for each and everyone of my students. I
get to stand, walk and move around the class as I please. My students are very active and need to
move as well. This year I am starting off with flexible seating to allow my students the movement
that we all need.     By adding flexible seating such as Hokki Stools to our small group area
and/or computer area, students will be allow to move and continue their active learning.  Flexible
seating and the Hokki Stools will allow the students to choose the type of seating they need at th
at time to allow them to stay focused on their learning. Six hours in a classroom, who would not w
ant to be comfortable .  nannan
```

In [289]:

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

```
As a teacher in a low income high poverty school my students are faced with several challenges in
and out of the classroom In spite of the many challenges they face I am looking to keep things sim
ple and provide my students with flexible and meaningful learning experiences My students attend a
Title 1 school in the eastern part of Florida They like to move they love to read and love and nee
d lots of positive attention Most of them are being raised by a single parent and or live with gra
ndparents and receive a free breakfast and lunch based on their socioeconomic status Many of my st
udents consider school their safe place From the minute they walk in the door of my classroom I fo
cus on their potential and growth while they are with me I may not be able to control their home l
ives however I can certainly control their experience during the school day I am requesting 4 Hokk
i Stools for my flexible seating Students are not all the same and they do not learn the same My j
ob as a teacher is to find what works for each and everyone of my students I get to stand walk and
move around the class as I please My students are very active and need to move as well This year I
am starting off with flexible seating to allow my students the movement that we all need By adding
flexible seating such as Hokki Stools to our small group area and or computer area students will b
e allow to move and continue their active learning Flexible seating and the Hokki Stools will allo
w the students to choose the type of seating they need at that time to allow them to stay focused
on their learning Six hours in a classroom who would not want to be comfortable nannan
```

In [290]:

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

In [291]:

```python
# 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['essay'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e.lower() not in stopwords)
    preprocessed_essays_train.append(sent.lower().strip())
```

```
100%|██████████████████████████████████████████████████████████████████████████████| 73196/73196 [01:
18<00:00, 937.76it/s]
```

In [292]:

```
# after preprocesing
preprocessed_essays_train[100]
```

Out[292]:

```
'teach student lesson day teach learn creating curiosity continue learning process long lives clay
p bedford students five six years old beginning explore world live school title one school nearly
students receiving free reduced price lunch many students grade level teachers students working ha
rd drastically change statistic end school year many students face lot adversity lives still come
school day ready work hard school place feel safe loved valued motivating students read learn
crucial particularly higher demands new common core standards kindergartners love able move around
classroom students love moving around variety seating options allow students one place still allow
move improve focus learning project greatly improve classroom every student find right type seat s
eats also easily moveable allow students move work groups collaborate work time flexible seating h
uge impact students bettering physical mental fitness nannan'
```

Preprocessed test data

In [293]:

```
preprocessed_essays_test = []
# tqdm is for printing the status bar
for sentence in tqdm(X_test['essay'].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
[00:34<00:00, 1044.46it/s]
```

In [294]:

```
preprocessed_essays_test[1000]
```

Out[294]:

```
'students walk classroom every day full life ready learn excited store day diverse group 12 13
year old journalists videographers digital story tellers embrace new challenges face step journey
midst changing time lives students endless source creativity energy dedication find tell amazing s
tories using cameras student smartphones well classroom video cameras shoot video public service
announcements news stories digital storytelling projects shoot creatively need 12 flexible tripods
allow us get great low angle shots well attach cameras trees fences structures mic cords needed sc
hool news room smartphone mounts allow shots taken phones steady professional accessories allow st
udents get creative ideas storytelling process nannan'
```

1.4 Preprocessing of `project_title`

In [295]:

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

In [296]:

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

```
Engineering STEAM into the Primary Classroom
==================================================
Building Blocks for Learning
==================================================
Empowering Students Through Art:Learning About Then and Now
==================================================
Health Nutritional Cooking in Kindergarten
==================================================
```

Preprocessing of Project Title for Train data

In [297]:

```
preprocessed_titles_train = []

for titles in tqdm(X_train["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.lower() not in stopwords)
    preprocessed_titles_train.append(title.lower().strip())
```

```
100%|████████████████████████████████████████████████████████████████████████| 73196/73196
[00:02<00:00, 24765.88it/s]
```

In [298]:

```
preprocessed_titles_train[1000]
```

Out[298]:

```
'growing little scientists'
```

Preprocessing of Project Title for Test data

In [299]:

```
preprocessed_titles_test = []

for titles in tqdm(X_test["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.lower() not in stopwords)
    preprocessed_titles_test.append(title.lower().strip())
```

```
100%|████████████████████████████████████████████████████████████████████████| 36052/36052
[00:01<00:00, 23517.50it/s]
```

In [300]:

```
preprocessed_titles_test[1000]
```

Out[300]:

```
'accessories open door possibilities'
```

Preprocessing of teacher_prefix for train data

In [301]:

```
preprocessed_teachers_prefix_train=[]

for prefix in tqdm(X_train['clean_teacher_prefix']):
    prefix=str(prefix)
    prefix = prefix.replace(".", " ")
    preprocessed_teachers_prefix_train.append(prefix.strip())
```

```
100%|████████████████████████████████████████████████████████████████████████| 73196/73196
[00:00<00:00, 1060881.15it/s]
```

Preprocessing of teacher_prefix for test data

In [302]:

```
preprocessed_teachers_prefix_test=[]

for prefix in tqdm(X_test['clean_teacher_prefix']):
    prefix=str(prefix)
    prefix = prefix.replace(".", " ")
    preprocessed_teachers_prefix_test.append(prefix.strip())
```

```
100%|█████████████████████████████████████████████████████████████████████████| 36052/36052
[00:00<00:00, 838502.62it/s]
```

Preprocessing of project_category for train data

In [303]:

```
preprocessed_project_category_train=[]

for category in tqdm(X_train['clean_grade_categories']):
    category=str(category)
    category = category.replace("-","_")
    preprocessed_project_category_train.append(category.strip())
```

```
100%|████████████████████████████████████████████████████████████████████████| 73196/73196
[00:00<00:00, 1074940.65it/s]
```

Preprocessing of project_category for test data

In [304]:

```
preprocessed_project_category_test=[]

for category in tqdm(X_test['clean_grade_categories']):
    category=str(category)
    category = category.replace("-","_")
    preprocessed_project_category_test.append(category.strip())
```

```
100%|████████████████████████████████████████████████████████████████████████| 36052/36052
```

```
[00:00<00:00, 1000768.04it/s]
```

1.5 Preparing data for models

In [305]:

```
project_data.columns
```

Out[305]:

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'school_state', 'Date',
       'project_title', 'project_essay_1', 'project_essay_2',
       'project_essay_3', 'project_essay_4', 'project_resource_summary',
       'teacher_number_of_previously_posted_projects', 'project_is_approved',
       'clean_categories', 'clean_subcategories', 'clean_grade_categories',
       'clean_teacher_prefix', 'title_word_count', 'essay',
       'essay_word_count'],
      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/

One hot encode clean_categories

In [307]:

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

from sklearn.feature_extraction.text import CountVectorizer

vectorizer_proj = CountVectorizer(lowercase=False, binary=True)
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.shape)
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)
```

```
['AppliedLearning', 'Care_Hunger', 'Health_Sports', 'History_Civics', 'Literacy_Language',
'Math_Science', 'Music_Arts', 'SpecialNeeds', 'Warmth']
Shape of matrix of Train data after one hot encoding  (73196, 9)
Shape of matrix of Test data after one hot encoding  (36052, 9)
```

One hot encode clean_subcategories

In [308]:

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

vectorizer_sub_proj = CountVectorizer(lowercase=False, binary=True)
vectorizer_sub_proj.fit(X_train['clean_subcategories'].values)

sub_categories_one_hot_train = vectorizer_sub_proj.transform(X_train['clean_subcategories'].values
)
sub_categories_one_hot_test = vectorizer_sub_proj.transform(X_test['clean_subcategories'].values)
#sub_categories_one_hot_cv = vectorizer_sub_proj.transform(X_cv['clean_subcategories'].values)


print(vectorizer_sub_proj.get_feature_names())

print("Shape of matrix of Train data after one hot encoding ",sub_categories_one_hot_train.shape)
print("Shape of matrix of Test data after one hot encoding ",sub_categories_one_hot_test.shape)
#print("Shape of matrix of Cross Validation data after one hot encoding
",sub_categories_one_hot_cv.shape)
```

```
['AppliedSciences', 'Care_Hunger', 'CharacterEducation', 'Civics_Government',
'College_CareerPrep', 'CommunityService', 'ESL', 'EarlyDevelopment', 'Economics',
'EnvironmentalScience', 'Extracurricular', 'FinancialLiteracy', 'ForeignLanguages', 'Gym_Fitness',
'Health_LifeScience', 'Health_Wellness', 'History_Geography', 'Literacy', 'Literature_Writing', 'M
athematics', 'Music', 'NutritionEducation', 'Other', 'ParentInvolvement', 'PerformingArts', 'Socia
lSciences', 'SpecialNeeds', 'TeamSports', 'VisualArts', 'Warmth']
Shape of matrix of Train data after one hot encoding  (73196, 30)
Shape of matrix of Test data after one hot encoding  (36052, 30)
```

In [309]:

```
# you can do the similar thing with state, teacher_prefix and project_grade_category also
```

One hot encode on state

In [310]:

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

In [311]:

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

In [312]:

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

vectorizer_states = CountVectorizer(lowercase=False,binary=True)
vectorizer_states.fit(X_train['school_state'].values)

school_state_categories_one_hot_train = vectorizer_states.transform(X_train['school_state'].values
)
school_state_categories_one_hot_test = vectorizer_states.transform(X_test['school_state'].values)
#school_state_categories_one_hot_cv = vectorizer_states.transform(X_cv['school_state'].values)

print(vectorizer_states.get_feature_names())

print("Shape of matrix of Train data after one hot encoding
",school_state_categories_one_hot_train.shape)
print("Shape of matrix of Test data after one hot encoding ",school_state_categories_one_hot_test.
shape)
#print("Shape of matrix of Cross Validation data after one hot encoding
",school_state_categories_one_hot_cv.shape)
```

```
['AK', 'AL', 'AR', 'AZ', 'CA', 'CO', 'CT', 'DC', 'DE', 'FL', 'GA', 'HI', 'IA', 'ID', 'IL', 'IN', 'K
S', 'KY', 'LA', 'MA', 'MD', 'ME', 'MI', 'MN', 'MO', 'MS', 'MT', 'NC', 'ND', 'NE', 'NH', 'NJ', 'NM',
'NV', 'NY', 'OH', 'OK', 'OR', 'PA', 'RI', 'SC', 'SD', 'TN', 'TX', 'UT', 'VA', 'VT', 'WA', 'WI', 'WV
', 'WY']
Shape of matrix of Train data after one hot encoding  (73196, 51)
Shape of matrix of Test data after one hot encoding  (36052, 51)
```

One hot encode - project category

In [313]:

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

In [314]:

```
#project_grade_cat_dict = dict(my_counter)
#sorted_project_grade_cat_dict = dict(sorted(project_grade_cat_dict.items(), key=lambda kv: kv[1])
)
```

In [315]:

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

vectorizer_grade = CountVectorizer(lowercase=False, binary=True)
vectorizer_grade.fit(X_train['clean_grade_categories'].values.astype("U"))

project_grade_categories_one_hot_train =
vectorizer_grade.transform(X_train['clean_grade_categories'].values.astype("U"))
project_grade_categories_one_hot_test = vectorizer_grade.transform(X_test['clean_grade_categories'
].values.astype("U"))
#project_grade_categories_one_hot_cv =
vectorizer_grade.transform(X_cv['clean_grade_categories'].values.astype("U"))

print(vectorizer_grade.get_feature_names())

print("Shape of matrix of Train data after one hot encoding
",project_grade_categories_one_hot_train.shape)
print("Shape of matrix of Test data after one hot encoding ",project_grade_categories_one_hot_test
.shape)
#print("Shape of matrix of Cross Validation data after one hot encoding
",project_grade_categories_one_hot_cv.shape)
```

```
['Grades_3_5', 'Grades_6_8', 'Grades_9_12', 'Grades_PreK_2']
Shape of matrix of Train data after one hot encoding  (73196, 4)
Shape of matrix of Test data after one hot encoding  (36052, 4)
```

One hot encode - Teacher Prefix

In [316]:

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

In [317]:

```
#teacher_prefix_cat_dict = dict(my_counter)
#sorted_teacher_prefix_cat_dict = dict(sorted(teacher_prefix_cat_dict.items(), key=lambda kv: kv[1
]))
```

In [318]:

```
## 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-np-nan-
is-an-invalid-document/39308809#39308809

vectorizer_teacher = CountVectorizer(lowercase=False,binary=True)
vectorizer_teacher.fit(X_train['clean_teacher_prefix'].values.astype("U"))

teacher_prefix_categories_one_hot_train =
vectorizer_teacher.transform(X_train['clean_teacher_prefix'].values.astype("U"))
teacher_prefix_categories_one_hot_test =
vectorizer_teacher.transform(X_test['clean_teacher_prefix'].values.astype("U"))
#teacher_prefix_categories_one_hot_cv =
vectorizer_teacher.transform(X_cv['clean_teacher_prefix'].values.astype("U"))

print(vectorizer_teacher.get_feature_names())
#print(teacher_prefix_categories_one_hot_train[:,1:5])
print("Shape of matrix after one hot encoding ",teacher_prefix_categories_one_hot_train.shape)
print("Shape of matrix after one hot encoding ",teacher_prefix_categories_one_hot_test.shape)
#print("Shape of matrix after one hot encoding ",teacher_prefix_categories_one_hot_cv.shape)
```

```
['Dr', 'Mr', 'Mrs', 'Ms', 'Teacher', 'nan']
Shape of matrix after one hot encoding  (73196, 6)
Shape of matrix after one hot encoding  (36052, 6)
```

1.5.2 Vectorizing Text data

1.5.2.1 Bag of words

Train Data - Essays

In [319]:

```
# We are considering only the words which appeared in at least 10 documents(rows or projects).

vectorizer_bow_essay = CountVectorizer(min_df=10)
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  (73196, 14126)
```

Test Data - Essays

In [320]:

```
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, 14126)
```

In [321]:

```
# you can vectorize the title also
# before you vectorize the title make sure you preprocess it
```

Train Data - Title

In [322]:

```
vectorizer_bow_title = CountVectorizer(min_df=10)

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  (73196, 2533)
```

Test Data - Title

In [323]:

```
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, 2533)
```

1.5.2.2 TFIDF vectorizer

Train Data - Essays

In [324]:

```
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer_tfidf_essay = TfidfVectorizer(min_df=10)
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  (73196, 14126)
```

Test Data - Essays

In [325]:

```
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, 14126)
```

Train Data - Title

In [326]:

```
vectorizer_tfidf_titles = TfidfVectorizer(min_df=10)

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  (73196, 2533)
```

Test Data - Title

In [327]:

```
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, 2533)
```

1.5.2.3 Using Pretrained Models: Avg W2V

In [328]:

```
'''
# 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')

# ============================
Output:

Loading Glove Model
1917495it [06:32, 4879.69it/s]
Done. 1917495  words loaded!

# ============================

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

for i in preproced_titles:
    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-pickle-to-sa
ve-and-load-variables-in-python/

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


'''
```

Out[328]:

```
'\n# Reading glove vectors in python: https://stackoverflow.com/a/38230349/4084039\ndef
loadGloveModel(gloveFile):\n    print ("Loading Glove Model")\n    f = open(gloveFile,\'r\',
encoding="utf8")\n    model = {}\n    for line in tqdm(f):\n        splitLine = line.split()\n
word = splitLine[0]\n        embedding = np.array([float(val) for val in splitLine[1:]])\n        m
odel[word] = embedding\n    print ("Done.",len(model)," words loaded!")\n    return model\nmodel =
loadGloveModel(\'glove.42B.300d.txt\')\n\n# ============================\nOutput:\n    \nLoading G
love Model\n1917495it [06:32, 4879.69it/s]\nDone. 1917495  words loaded!\n\n#
```

```
============================\n\nwords = []\nfor i in preproced_texts:\n    words.extend(i.split(\'
\'))\n\nfor i in preproced_titles:\n    words.extend(i.split(\' \'))\nprint("all the words in the
coupus", len(words))\nwords = set(words)\nprint("the unique words in the coupus",
len(words))\n\ninter_words = set(model.keys()).intersection(words)\nprint("The number of words tha
t are present in both glove vectors and our coupus",       len(inter_words),"
(",np.round(len(inter_words)/len(words)*100,3),"%)")\n\nwords_courpus = {}\nwords_glove =
set(model.keys())\nfor i in words:\n    if i in words_glove:\n        words_courpus[i] = model[i]\r
print("word 2 vec length", len(words_courpus))\n\n\n# stronging variables into pickle files python
: http://www.jessicayung.com/how-to-use-pickle-to-save-and-load-variables-in-python/\n\nimport pic
kle\nwith open(\'glove_vectors\', \'wb\') as f:\n    pickle.dump(words_courpus, f)\n\n\n'
```

In [329]:

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

Train Data - Essays

In [330]:

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

avg_w2v_vectors_train = [];

for sentence in tqdm(preprocessed_essays_train): # 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%|██████████████████████████████████████████████████████████| 73196/73196
[00:40<00:00, 1807.59it/s]
```

```
73196
300
```

Test Data - Essays

In [331]:

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

avg_w2v_vectors_test = [];

for sentence in tqdm(preprocessed_essays_test): # 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
[00:19<00:00, 1855.41it/s]
```

```
36052
300
```

Train Data - Title

In [332]:

```
# Similarly you can vectorize for title also

avg_w2v_vectors_titles_train = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_titles_train): # 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%|█████████████████████████████████████████████████████████████████████████| 73196/73196
[00:01<00:00, 47684.06it/s]
```

```
73196
300
```

Test - Titles

In [333]:

```
# Similarly you can vectorize for title also

avg_w2v_vectors_titles_test = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_titles_test): # 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, 42068.04it/s]
```

```
36052
300
```

1.5.2.3 Using Pretrained Models: TFIDF weighted W2V

Train Data - Essays

In [339]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(preprocessed_essays_train)
# 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 [340]:

```
# 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(preprocessed_essays_train): # 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((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += 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%|████████████████████████████████████████████████████████████████████████████| 73196/73196 [04:
51<00:00, 250.69it/s]
```

```
73196
300
```

Test Data - Essays

In [341]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_test = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_essays_test): # 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((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += 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 [02:
22<00:00, 253.29it/s]
```

```
36052
300
```

In [342]:

```
# Similarly you can vectorize for title also
```

Train Data - title

In [343]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(preprocessed_titles_train)
# 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 [344]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_titles_train = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_titles_train): # 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((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += 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%|█████████████████████████████████████████████████████████████████████████████| 73196/73196
[00:02<00:00, 27422.63it/s]
```

```
73196
300
```

Test Data - Title

In [345]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_titles_test = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_titles_test): # 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((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += 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, 23184.50it/s]
```

```
36052
300
```

1.5.3 Vectorizing Numerical features

Price

In [346]:

```
price_data = resource_data.groupby('id').agg({'price':'sum', 'quantity':'sum'}).reset_index()
price_data.head(2)
```

Out[346]:

	id	price	quantity
0	p000001	459.56	7
1	p000002	515.89	21

In [347]:

```
# 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 [348]:

```
from sklearn.preprocessing import Normalizer

normalizer = Normalizer()

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

normalizer.fit(X_train['price'].values.reshape(1,-1))

price_train = normalizer.transform(X_train['price'].values.reshape(1,-1))
#price_cv = normalizer.transform(X_cv['price'].values.reshape(1,-1))
price_test = normalizer.transform(X_test['price'].values.reshape(1,-1))

print("After vectorizations")
print(price_train.shape, y_train.shape)
#print(price_cv.shape, y_cv.shape)
print(price_test.shape, y_test.shape)
print("="*100)
print(price_train)
```

```
After vectorizations
(1, 73196) (73196,)
(1, 36052) (36052,)
====================================================================================================

[[0.00302608 0.00054236 0.00181386 ... 0.00256778 0.00252959 0.00018497]]
```

Quantity

In [349]:

```
normalizer = Normalizer()

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

normalizer.fit(X_train['quantity'].values.reshape(1,-1))

quantity_train = normalizer.transform(X_train['quantity'].values.reshape(1,-1))
#quantity_cv = normalizer.transform(X_cv['quantity'].values.reshape(1,-1))
quantity_test = normalizer.transform(X_test['quantity'].values.reshape(1,-1))

print("After vectorizations")
print(quantity_train.shape, y_train.shape)
#print(quantity_cv.shape, y_cv.shape)
print(quantity_test.shape, y_test.shape)
print("="*100)
print(quantity_train)
```

```
After vectorizations
(1, 73196) (73196,)
(1, 36052) (36052,)
====================================================================================================

[[0.00716097 0.00907056 0.00143219 ... 0.00358049 0.0004774  0.00381918]]
```

Number of Projects previously proposed by Teacher

In [350]:

```
normalizer = Normalizer()

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

normalizer.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(1,-1))

prev_projects_train = normalizer.transform(X_train['teacher_number_of_previously_posted_projects']
.values.reshape(1,-1))
#prev_projects_cv =
normalizer.transform(X_cv['teacher_number_of_previously_posted_projects'].values.reshape(1,-1))
prev_projects_test = normalizer.transform(X_test['teacher_number_of_previously_posted_projects'].v
alues.reshape(1,-1))

print("After vectorizations")
print(prev_projects_train.shape, y_train.shape)
#rint(prev_projects_cv.shape, y_cv.shape)
print(prev_projects_test.shape, y_test.shape)
print("="*100)
print(prev_projects_train)
```

```
After vectorizations
(1, 73196) (73196,)
(1, 36052) (36052,)
====================================================================================================

[[0.00024594 0.         0.         ... 0.         0.00012297 0.00024594]]
```

Title word Count

In [351]:

```
normalizer = Normalizer()

normalizer.fit(X_train['title_word_count'].values.reshape(1,-1))

title_word_count_train = normalizer.transform(X_train['title_word_count'].values.reshape(1,-1))
#title_word_count_cv = normalizer.transform(X_cv['title_word_count'].values.reshape(1,-1))
title_word_count_test = normalizer.transform(X_test['title_word_count'].values.reshape(1,-1))

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)
print("="*100)
print(title_word_count_train)
```

```
After vectorizations
(1, 73196) (73196,)
(1, 36052) (36052,)
====================================================================================================

[[0.00199201 0.00332001 0.00265601 ... 0.00265601 0.00199201 0.00199201]]
```

Essay word Count

In [352]:

```
normalizer = Normalizer()

normalizer.fit(X_train['essay_word_count'].values.reshape(1,-1))

essay_word_count_train = normalizer.transform(X_train['essay_word_count'].values.reshape(1,-1))
#essay_word_count_cv = normalizer.transform(X_cv['essay_word_count'].values.reshape(1,-1))
essay_word_count_test = normalizer.transform(X_test['essay_word_count'].values.reshape(1,-1))

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)
print("="*100)
print(essay_word_count_train )
```

```
After vectorizations
(1, 73196) (73196,)
(1, 36052) (36052,)
====================================================================================================

[[0.00317075 0.00270776 0.00474209 ... 0.00252537 0.00269373 0.00312866]]
```

Assignment 4: Naive Bayes

1. **Apply Multinomial NaiveBayes 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)

2. **The hyper paramter tuning(find best Alpha)**

   - Find the best hyper parameter which will give the maximum AUC value
   - Consider a wide range of alpha values for hyperparameter tuning, start as low as 0.00001
   - 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. **Feature importance**

   - Find the top 10 features of positive class and top 10 features of negative class for both feature sets Set 1 and Set 2 using values of `feature_log_prob_` parameter of MultinomialNB and print their corresponding feature names

4. **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. Here on X-axis you will have alpha values, since they have a wide range, just to represent those alpha values on the graph, apply log function on those alpha values.
   - 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 with predicted and original labels of test data points. Please visualize your confusion matrices using seaborn heatmaps.

5. **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

2. Naive Bayes

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

In [377]:

```
price_train = (price_train.reshape(-1,1))
#price_cv = (price_cv.reshape(-1,1))
price_test = (price_test.reshape(-1,1))

quantity_train =(quantity_train.reshape(-1,1))
#quantity_cv = (quantity_cv.reshape(-1,1))
quantity_test = (quantity_test.reshape(-1,1))

prev_projects_train = (prev_projects_train.reshape(-1,1))
#prev_projects_cv = (prev_projects_cv.reshape(-1,1))
prev_projects_test = (prev_projects_test.reshape(-1,1))

title_word_count_train = (title_word_count_train.reshape(-1,1))
#title_word_count_cv = (title_word_count_cv.reshape(-1,1))
title_word_count_test = (title_word_count_test.reshape(-1,1))


essay_word_count_train = (essay_word_count_train.reshape(-1,1))
#essay_word_count_cv = (essay_word_count_cv.reshape(-1,1))
essay_word_count_test = (essay_word_count_test.reshape(-1,1))

print(price_train)
```

```
[[0.00302608]
 [0.00054236]
 [0.00181386]
 ...
 [0.00256778]
 [0.00252959]
 [0.00018497]]
```

In [378]:

```
# 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_categories_one_hot_train, project_grade_categories_one_hot_train,
teacher_prefix_categories_one_hot_train, price_train, quantity_train, prev_projects_train, title_wo
rd_count_train, essay_word_count_train, title_bow_train, text_bow_train)).tocsr()
X_te = hstack((categories_one_hot_test, sub_categories_one_hot_test,
school_state_categories_one_hot_test, project_grade_categories_one_hot_test,
teacher_prefix_categories_one_hot_test, price_test, quantity_test, prev_projects_test,
title_word_count_test, essay_word_count_test, title_bow_test, text_bow_test)).tocsr()
#X_cr = hstack((categories_one_hot_cv, sub_categories_one_hot_cv,
school_state_categories_one_hot_cv, project_grade_categories_one_hot_cv,
teacher_prefix_categories_one_hot_cv, price_cv, quantity_cv, prev_projects_cv,
title_word_count_cv, essay_word_count_cv, title_bow_cv, text_bow_cv)).tocsr()
```

In [379]:

```
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
(73196, 16764) (73196,)
(36052, 16764) (36052,)
====================================================================================================
```

In [380]:

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

    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000
    # consider you X_tr shape is 49041, then your cr_loop will be 49041 - 49041%1000 = 49000
    # in this for loop we will iterate unti the last 1000 multiplier
    for i in range(0, tr_loop, 1000):
        y_data_pred.extend(clf.predict_proba(data[i:i+1000])[:,1])
    # we will be predicting for the last data points
    y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

    return y_data_pred
```

In [381]:

```
def pred_prob(clf, data):
    y_pred = []
    y_pred = clf.predict_proba(data)[:,1]
    return y_pred
```

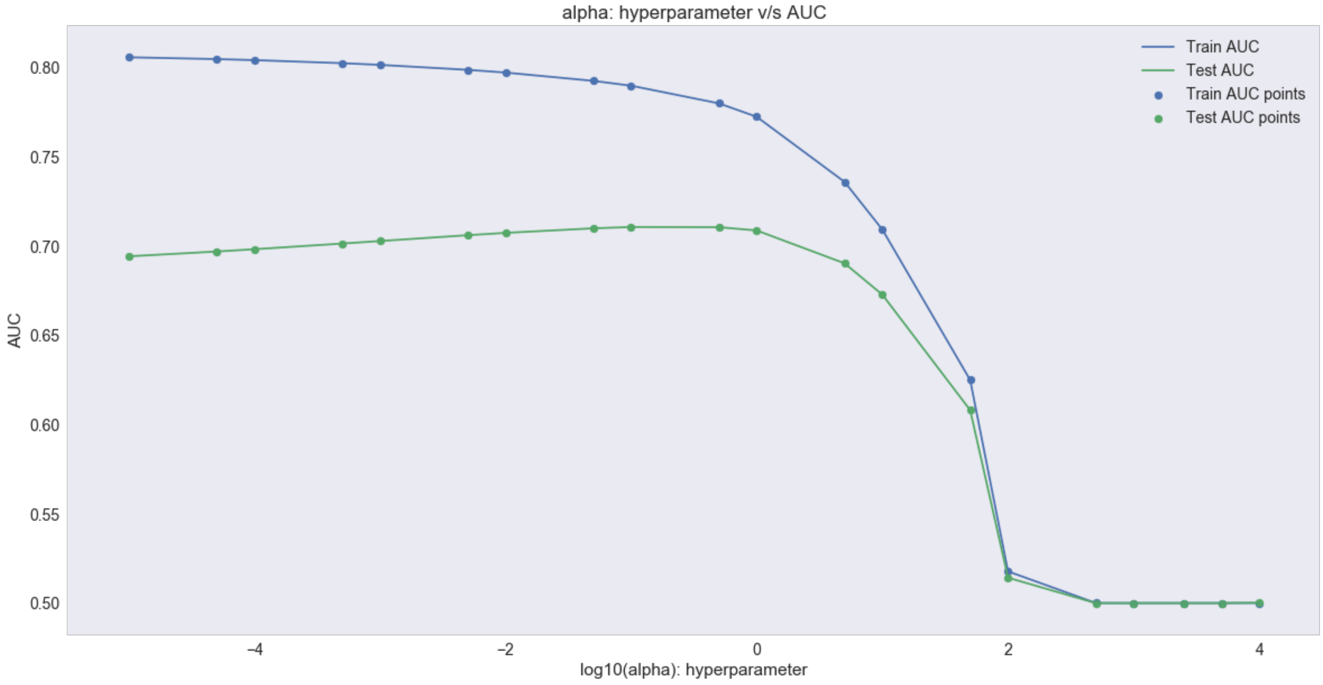
In [384]:

```
import numpy as np

train_auc = []
test_auc = []
log_alphas =[]

alphas = [0.00001, 0.00005, 0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 5, 10, 50, 100, 5
00, 1000, 2500, 5000, 10000]

for i in tqdm(alphas):
    nb = MultinomialNB(alpha = i,class_prior=[0.5,0.5])
    nb.fit(X_tr, y_train)
    y_train_pred = nb.predict_proba(X_tr)[:,1]
    y_test_pred = nb.predict_proba(X_te)[:,1]

    #y_train_pred = batch_predict(nb, X_tr)
    #y_cv_pred = batch_predict(nb, X_cr)


    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive class
tive class
    # not the predicted outputs
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    test_auc.append(roc_auc_score(y_test, y_test_pred))

for a in tqdm(alphas):
    b = np.log10(a)
    log_alphas.append(b)
```

```
100%|█████████████████████████████████████████████████████████████████████████████████| 20/20
[00:05<00:00,  3.53it/s]
100%|█████████████████████████████████████████████████████████████████████████████████████████| 2
0/20 [00:00<?, ?it/s]
```

In [385]:

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

plt.plot(log_alphas, train_auc, label='Train AUC')
plt.plot(log_alphas, test_auc, label='Test AUC')

plt.scatter(log_alphas, train_auc, label='Train AUC points')
plt.scatter(log_alphas, test_auc, label='Test AUC points')

plt.legend()
plt.xlabel("log10(alpha): hyperparameter")
plt.ylabel("AUC")
plt.title("alpha: hyperparameter v/s AUC")
plt.grid()
plt.show()
```



Gridsearch-cv using cv = 10 (K fold cross validation)

In [359]:

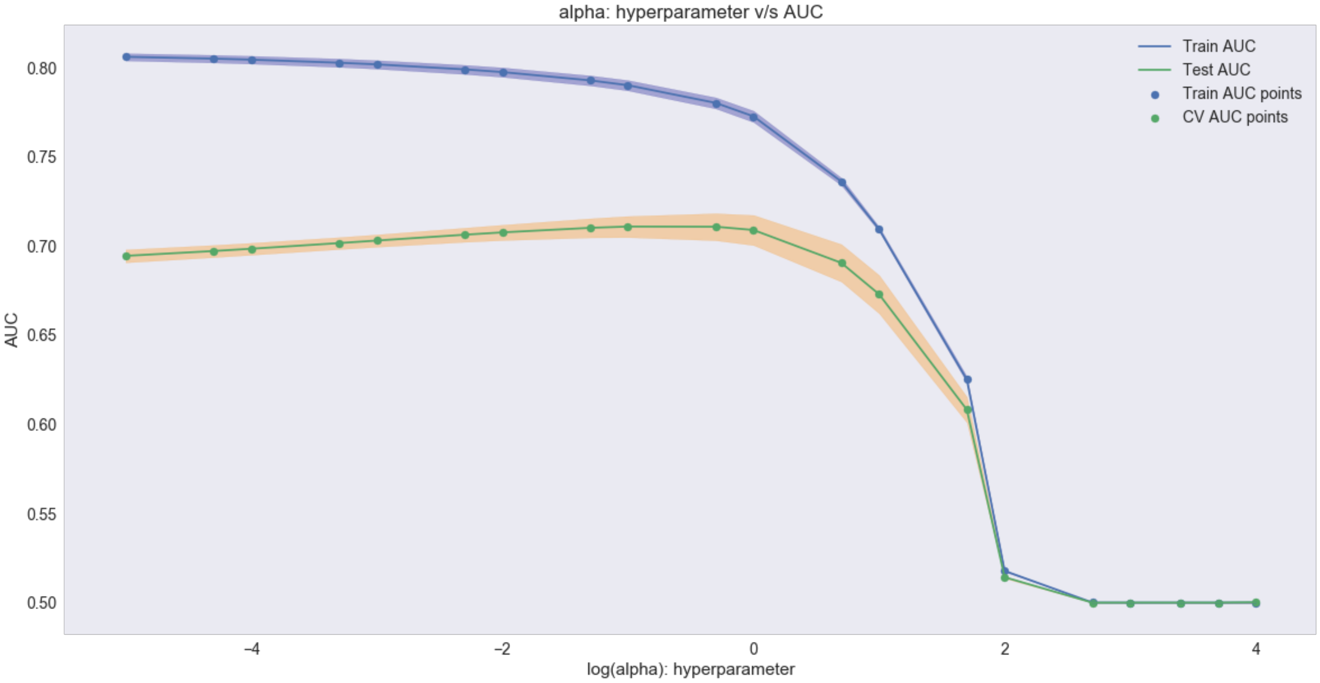
```
from sklearn.model_selection import GridSearchCV

nb = MultinomialNB(class_prior=[0.5,0.5])

parameters = {'alpha':[0.00001, 0.00005, 0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 5,
10, 50, 100, 500, 1000, 2500, 5000, 10000]}

clf = GridSearchCV(nb, parameters,cv=10,  scoring='roc_auc')

clf.fit(X_tr, y_train)
```

```
train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
test_auc = clf.cv_results_['mean_test_score']
test_auc_std= clf.cv_results_['std_test_score']
```

In [387]:

```
alphas = [0.00001, 0.00005, 0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 5, 10, 50, 100, 5
00, 1000, 2500, 5000, 10000]
log_alphas =[]

for a in tqdm(alphas):
    b = np.log10(a)
    log_alphas.append(b)

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

plt.plot(log_alphas, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(log_alphas,train_auc - train_auc_std,train_auc + train_auc_std,alpha=0.3,col
or='darkblue')

plt.plot(log_alphas, test_auc, label='Test AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(log_alphas,test_auc - test_auc_std,test_auc + test_auc_std,alpha=0.3,color='
darkorange')

plt.scatter(log_alphas, train_auc, label='Train AUC points')
plt.scatter(log_alphas, test_auc, label='CV AUC points')


plt.legend()
plt.xlabel("log(alpha): hyperparameter")
plt.ylabel("AUC")
plt.title("alpha: hyperparameter v/s AUC")
plt.grid()
plt.show()
```

```
100%|████████████████████████████████████████████████████████████████████████████████| 20/20
[00:00<00:00, 16905.70it/s]
```



Summary

Alpha values ranging from 0.00001 to 10000.0 was taken and the following results were obtained

obtained

1. 0.00001 as alpha values seemed to work very well on train data and the model seems to not work that efficiently on cross validation data.
2. Values closer to 0 works pretty well both on Train data and Cross Validation data.
3. Values more than 0 also doesnt seem to be effective on both Train and Cross Validation data.

log alpha value chosen 0 and hence alpha value is 1

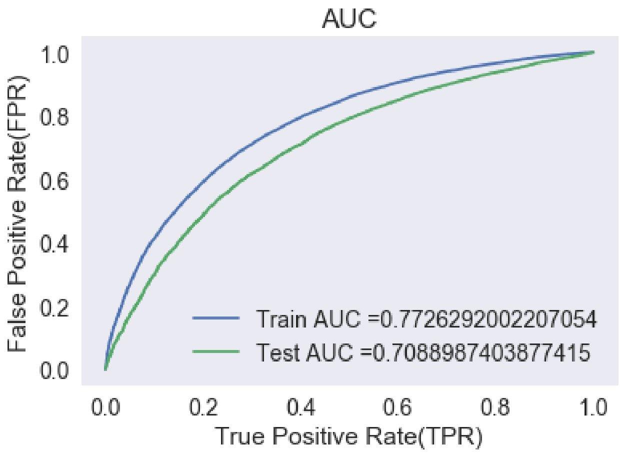
C) Train model using the best hyper-parameter value

In [388]:

```
best_alpha_1 = 1
```

In [389]:

```
# https://scikit-
learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc

nb_bow = MultinomialNB(alpha = best_alpha_1)

nb_bow.fit(X_tr, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs
y_train_pred = nb_bow.predict_proba(X_tr)[:,1]
y_test_pred = nb_bow.predict_proba(X_te)[:,1]
#y_train_pred = batch_predict(nb_bow, X_tr)
#y_test_pred = batch_predict(nb_bow, 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 [390]:

```
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,3))
    predictions = []
    for i in proba:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

Train Data

In [391]:

```
print("="*100)
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) 0.2499999979647145 for threshold 0.163
[[ 5542  5541]
 [ 8799 53314]]
```

In [392]:

```
conf_matr_df_train_1 = pd.DataFrame(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds,
train_fpr, train_fpr)), range(2),range(2))
```

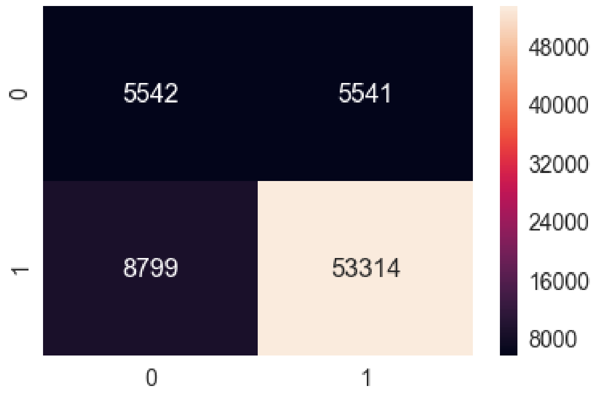
```
the maximum value of tpr*(1-fpr) 0.2499999979647145 for threshold 0.163
```

In [393]:

```
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[393]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x21fab242b38>
```



Test Data

In [394]:

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

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.2499999979647145 for threshold 0.163
[[ 2154  3305]
 [ 4538 26055]]
```

In [395]:

```
conf_matr_df_test_1 = pd.DataFrame(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, train_fpr, train_fpr)), range(2),range(2))
```

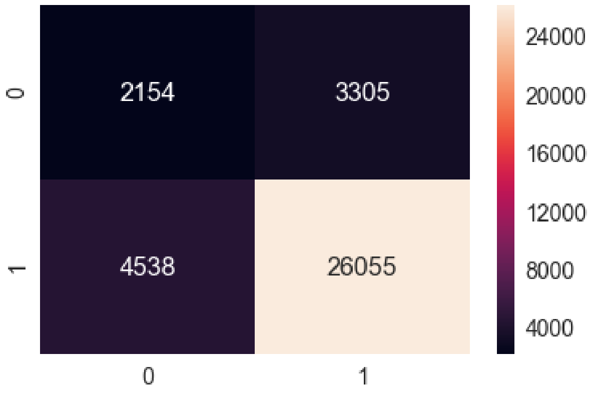
```
the maximum value of tpr*(1-fpr) 0.2499999979647145 for threshold 0.163
```

In [396]:

```
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_test_1, annot=True,annot_kws={"size": 16}, fmt='g')
```

Out[396]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x21fbae7d828>
```



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

In [397]:

```
# 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_categories_one_hot_train,
               project_grade_categories_one_hot_train, teacher_prefix_categories_one_hot_train, price_train, quantity_train,
               prev_projects_train, title_word_count_train, essay_word_count_train,
text_tfidf_train, title_tfidf_train)).tocsr()
X_te = hstack((categories_one_hot_test, sub_categories_one_hot_test,
school_state_categories_one_hot_test,
               project_grade_categories_one_hot_test, teacher_prefix_categories_one_hot_test, price_test, quantity_test,
               prev_projects_test, title_word_count_test, essay_word_count_test, text_tfidf_test, title_tfidf_test)).tocsr()
#X_cr = hstack((categories_one_hot_cv, sub_categories_one_hot_cv,
school_state_categories_one_hot_cv,
               #project_grade_categories_one_hot_cv, teacher_prefix_categories_one_hot_cv, price_cv, quantity_cv,
               #prev_projects_cv, title_word_count_cv, essay_word_count_cv, text_tfidf_cv, title_tfidf_cv)).tocsr()
```

In [398]:

```
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
(73196, 16764) (73196,)
(36052, 16764) (36052,)
====================================================================================================
```

A) Random alpha values

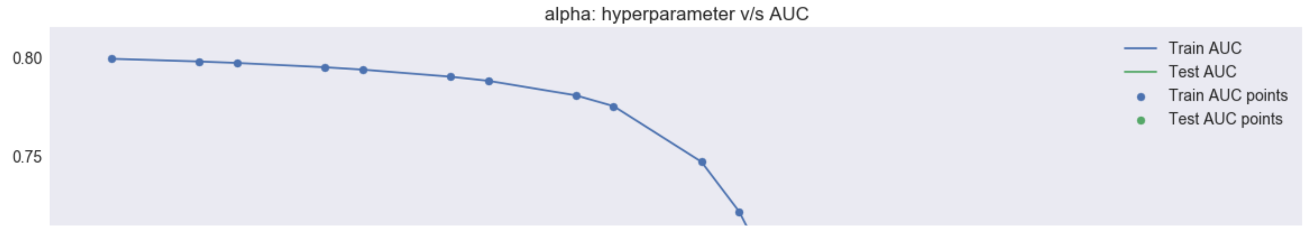
In [399]:

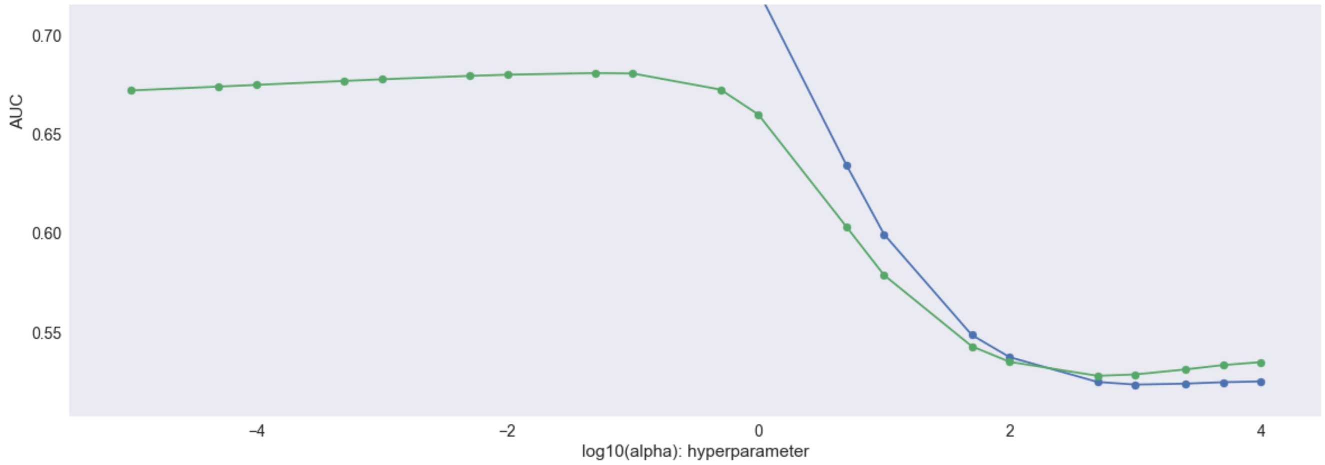
```
import numpy as np

train_auc = []
test_auc = []
log_alphas =[]

alphas = [0.00001, 0.00005, 0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 5, 10, 50, 100, 5
00, 1000, 2500, 5000, 10000]

for i in tqdm(alphas):
    nb = MultinomialNB(alpha = i,class_prior=[0.5,0.5])
    nb.fit(X_tr, y_train)
    y_train_pred = nb.predict_proba(X_tr)[:,1]
    y_test_pred = nb.predict_proba(X_te)[:,1]

    #y_train_pred = batch_predict(nb, X_tr)
    #y_cv_pred = batch_predict(nb, X_cr)


    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive class
    # not the predicted outputs
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    test_auc.append(roc_auc_score(y_test, y_test_pred))

for a in tqdm(alphas):
    b = np.log10(a)
    log_alphas.append(b)
```

```
100%|████████████████████████████████████████████████████████████████| 20/20
[00:05<00:00,  3.41it/s]
100%|██████████████████████████████████████████████████████████████████████| 2
0/20 [00:00<?, ?it/s]
```

In [400]:

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

plt.plot(log_alphas, train_auc, label='Train AUC')
plt.plot(log_alphas, test_auc, label='Test AUC')

plt.scatter(log_alphas, train_auc, label='Train AUC points')
plt.scatter(log_alphas, test_auc, label='Test AUC points')

plt.legend()
plt.xlabel("log10(alpha): hyperparameter")
plt.ylabel("AUC")
plt.title("alpha: hyperparameter v/s AUC")
plt.grid()
plt.show()
```





Summary

Alpha values ranging from 0.00001 to 10000.0 was taken and the following results were obtained :

1. Values closer to 0.01 works pretty well both on Train data and Cross Validation data.
2. Values more than 0.01 also doesnt seem to be effective on both Train and Cross Validation data

B) Gridsearch-cv using cv = 10 (K fold cross validation)

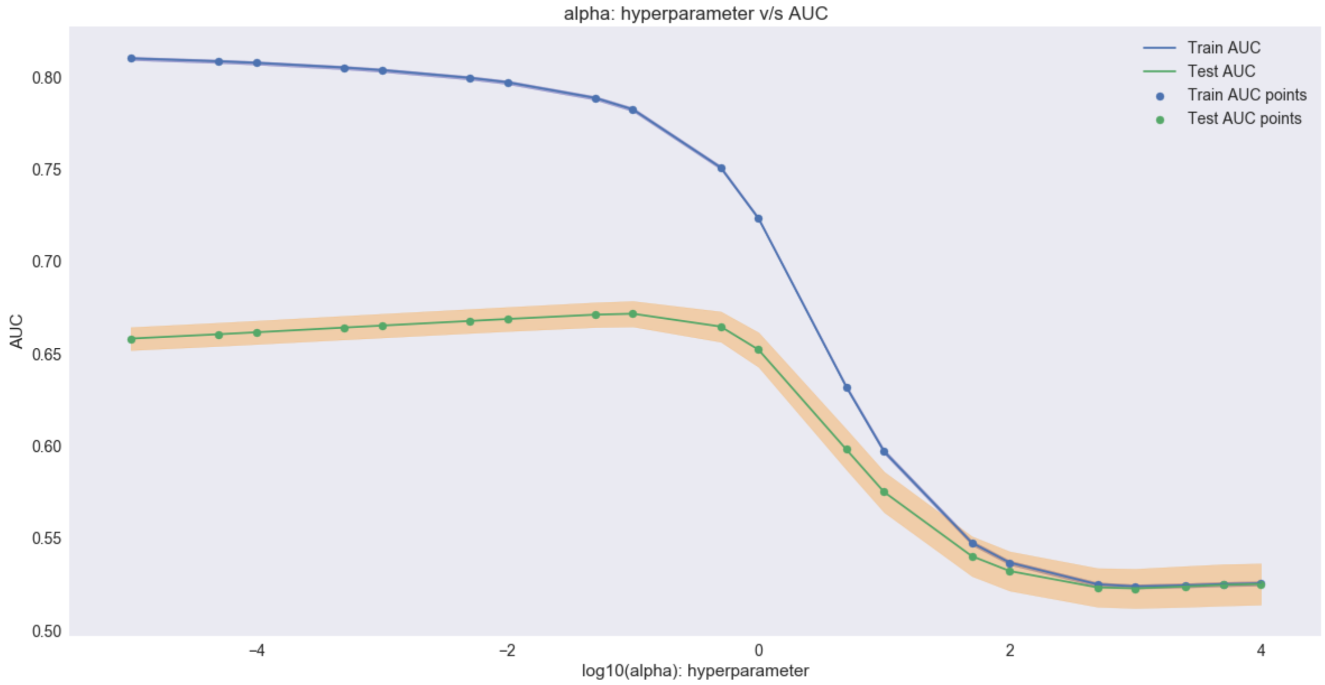
In [401]:

```
nb = MultinomialNB(class_prior=[0.5,0.5])

parameters = {'alpha':[0.00001, 0.00005, 0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 5,
10, 50, 100, 500, 1000, 2500, 5000, 10000]}

clf = GridSearchCV(nb, parameters,cv=10,  scoring='roc_auc')

clf.fit(X_tr, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
test_auc = clf.cv_results_['mean_test_score']
test_auc_std= clf.cv_results_['std_test_score']
```

In [403]:

```
alphas = [0.00001, 0.00005, 0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 5, 10, 50, 100, 5
00, 1000, 2500, 5000, 10000]
log_alphas =[]

for a in tqdm(alphas):
    b = np.log10(a)
    log_alphas.append(b)

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

plt.plot(log_alphas, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(log_alphas,train_auc - train_auc_std,train_auc + train_auc_std,alpha=0.3,col
or='darkblue')

plt.plot(log_alphas, test_auc, label='Test AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(log_alphas,test_auc - test_auc_std,test_auc + test_auc_std,alpha=0.3,color='
darkorange')

plt.scatter(log_alphas, train_auc, label='Train AUC points')
plt.scatter(log_alphas, test_auc, label='Test AUC points')


plt.legend()
plt.legend()
plt.xlabel("log10(alpha): hyperparameter")
plt.ylabel("AUC")
plt.title("alpha: hyperparameter v/s AUC")
plt.grid()
plt.show()
```

```
100%|██████████████████████████████████████████████████████████████████████| 20/20
[00:00<00:00, 17203.87it/s]
```



Summary

Alpha values ranging from 0.00001 to 10000.0 was taken and the following results were obtained :

1. 0.00001 as alpha values seemed to work very well on train data and the model seems to not work that efficiently on cross validation data.
2. Values closer to 0.01 works pretty well both on Train data and Cross Validation data.
3. Values more than 0.01 also doesnt seem to be effective on both Train and Cross Validation data

alpha value chosen is 0.01

Train model using the best hyper-parameter value

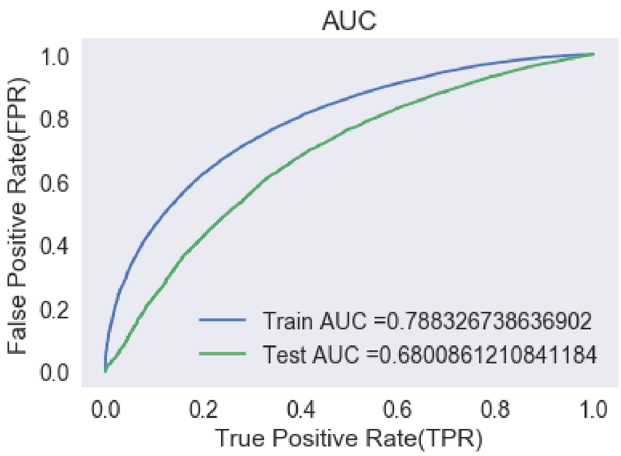
In [404]:

```
best_alpha_2=0.01
```

In [405]:

```
nb_tfidf = MultinomialNB(alpha = best_alpha_2,class_prior=[0.5,0.5])

nb_tfidf.fit(X_tr, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs
y_train_pred = nb_tfidf.predict_proba(X_tr)[:,1]
y_test_pred = nb_tfidf.predict_proba(X_te)[:,1]
#y_train_pred = batch_predict(nb_tfidf, X_tr)
#y_test_pred = batch_predict(nb_tfidf, 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 [406]:

```
print("="*100)
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) 0.24999999796471448 for threshold 0.362
[[ 5541  5542]
 [ 8536 53577]]
```

In [407]:

```
conf_matr_df_train_2 = pd.DataFrame(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds,
train_fpr, train_fpr)), range(2),range(2))
```

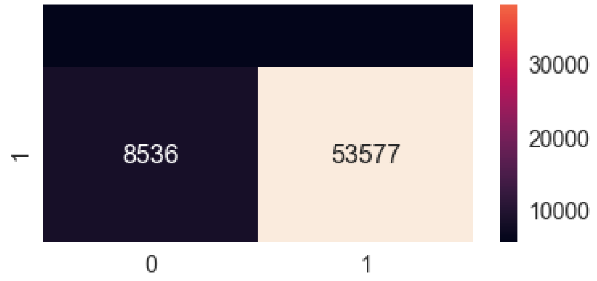
```
the maximum value of tpr*(1-fpr) 0.24999999796471448 for threshold 0.362
```

In [408]:

```
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_train_2, annot=True,annot_kws={"size": 16}, fmt='g')
```

Out[408]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x21fb66b1978>
```



Test Data

In [409]:

```
print("="*100)
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, train_fpr, train_fpr)))
```

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

Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24999999796471448 for threshold 0.362
[[ 1960  3499]
 [ 4508 26085]]
```

In [410]:

```
conf_matr_df_test_2 = pd.DataFrame(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, tra
in_fpr, train_fpr)), range(2),range(2))
```

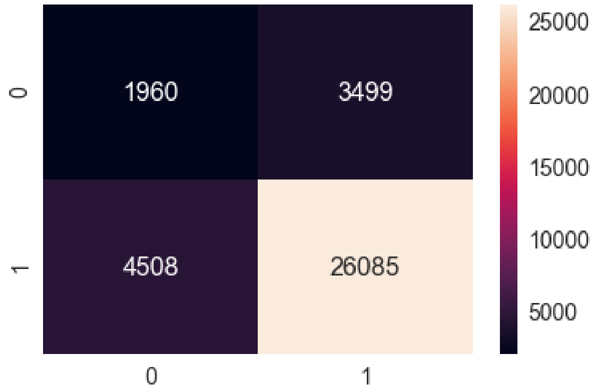
```
the maximum value of tpr*(1-fpr) 0.24999999796471448 for threshold 0.362
```

In [411]:

```
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[411]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x21fb29f8240>
```



Select best 10 features of both Positive and negative class for both the sets of data

SET1

In [412]:

```
X_tr = hstack((categories_one_hot_train, sub_categories_one_hot_train
X_tr = hstack((categories_one_hot_train, sub_categories_one_hot_train,
school_state_categories_one_hot_train, project_grade_categories_one_hot_train,
teacher_prefix_categories_one_hot_train, price_train, quantity_train, prev_projects_train, title_wo
rd_count_train, essay_word_count_train, title_bow_train, text_bow_train)).tocsr()
X_te = hstack((categories_one_hot_test, sub_categories_one_hot_test,
school_state_categories_one_hot_test, project_grade_categories_one_hot_test,
teacher_prefix_categories_one_hot_test, price_test, quantity_test, prev_projects_test,
title_word_count_test, essay_word_count_test, title_bow_test, text_bow_test)).tocsr()
#X_cr = hstack((categories_one_hot_cv, sub_categories_one_hot_cv,
school_state_categories_one_hot_cv, project_grade_categories_one_hot_cv,
teacher_prefix_categories_one_hot_cv, price_cv, quantity_cv, prev_projects_cv,
title_word_count_cv, essay_word_count_cv, title_bow_cv, text_bow_cv)).tocsr()
```

In [413]:

```
nb_bow = MultinomialNB(alpha = 1,class_prior=[0.5,0.5])

nb_bow.fit(X_tr, y_train)
```

Out[413]:

```
MultinomialNB(alpha=1, class_prior=[0.5, 0.5], fit_prior=True)
```

In [414]:

```
bow_features_names = []
```

In [415]:

```
for a in vectorizer_proj.get_feature_names() :
    bow_features_names.append(a)
```

In [416]:

```
for a in vectorizer_sub_proj.get_feature_names() :
    bow_features_names.append(a)
```

In [417]:

```
for a in vectorizer_states.get_feature_names() :
    bow_features_names.append(a)
```

In [418]:

```
for a in vectorizer_grade.get_feature_names() :
    bow_features_names.append(a)
```

In [419]:

```
for a in vectorizer_teacher.get_feature_names() :
    bow_features_names.append(a)
```

In [420]:

```
len(bow_features_names)
```

Out[420]:

```
100
```

In [421]:

```
bow_features_names.append("price")
```

In [422]:

```
bow_features_names.append("quantity")
bow_features_names.append("prev_proposed_projects")
bow_features_names.append("title_word_count")
bow_features_names.append("essay_word_count")
```

In [423]:

```
len(bow_features_names)
```

Out[423]:

```
105
```

In [424]:

```
for a in vectorizer_bow_title.get_feature_names() :
    bow_features_names.append(a)
```

In [425]:

```
len(bow_features_names)
```

Out[425]:

```
2638
```

In [426]:

```
for a in vectorizer_bow_essay.get_feature_names() :
    bow_features_names.append(a)
```

In [427]:

```
len(bow_features_names)
```

Out[427]:

```
16764
```

In [428]:

```
bow_features_prob=[]
```

In [429]:

```
for a in range(14133):
    bow_features_prob.append(nb_bow.feature_log_prob_[1,a])
print(len(bow_features_prob))
```

```
14133
```

10 Positive features from BOW model

In [430]:

```
pos_class_prob_sorted = nb_bow.feature_log_prob_[1, :].argsort()[::-1][:len(bow_features_prob)]
for i in pos_class_prob_sorted[:10]:
    print(bow_features_names[i])
```

```
students
school
learning
classroom
not
learn
help
many
```

```
nannan
need
```

10 Negative features from BOW model

In [431]:

```
neg_class_prob_sorted = nb_bow.feature_log_prob_[0, :].argsort()[::-1][:len(bow_features_prob)]
for i in neg_class_prob_sorted[:10]:
    print(bow_features_names[i])
```

```
students
school
learning
classroom
learn
not
help
nannan
many
need
```

SET 2

In [432]:

```
X_tr = hstack((categories_one_hot_train, sub_categories_one_hot_train,
school_state_categories_one_hot_train, project_grade_categories_one_hot_train,
teacher_prefix_categories_one_hot_train, price_train, quantity_train, prev_projects_train, title_wo
rd_count_train, essay_word_count_train, text_tfidf_train, title_tfidf_train)).tocsr()
X_te = hstack((categories_one_hot_test, sub_categories_one_hot_test,
school_state_categories_one_hot_test, project_grade_categories_one_hot_test,
teacher_prefix_categories_one_hot_test, price_test, quantity_test, prev_projects_test,
title_word_count_test, essay_word_count_test, text_tfidf_test, title_tfidf_test)).tocsr()
#X_cr = hstack((categories_one_hot_cv, sub_categories_one_hot_cv,
school_state_categories_one_hot_cv, project_grade_categories_one_hot_cv,
teacher_prefix_categories_one_hot_cv, price_cv, quantity_cv, prev_projects_cv,
title_word_count_cv, essay_word_count_cv, text_tfidf_cv, title_tfidf_cv)).tocsr()
```

In [433]:

```
nb_tfidf = MultinomialNB(alpha = 0.01,class_prior=[0.5,0.5])

nb_tfidf.fit(X_tr, y_train)
```

Out[433]:

```
MultinomialNB(alpha=0.01, class_prior=[0.5, 0.5], fit_prior=True)
```

In [434]:

```
tfidf_features_names = []
```

In [435]:

```
for a in vectorizer_proj.get_feature_names() :
    tfidf_features_names.append(a)
```

In [436]:

```
for a in vectorizer_sub_proj.get_feature_names() :
    tfidf_features_names.append(a)
```

In [437]:

```
for a in vectorizer_states.get_feature_names() :
```

```
for a in vectorizer_states.get_feature_names() :
    tfidf_features_names.append(a)
```

In [438]:

```
for a in vectorizer_grade.get_feature_names() :
    tfidf_features_names.append(a)
```

In [439]:

```
for a in vectorizer_teacher.get_feature_names() :
    tfidf_features_names.append(a)
```

In [440]:

```
len(tfidf_features_names)
```

Out[440]:

```
100
```

In [441]:

```
tfidf_features_names.append("price")
tfidf_features_names.append("quantity")
tfidf_features_names.append("prev_proposed_projects")
tfidf_features_names.append("title_word_count")
tfidf_features_names.append("essay_word_count")
```

In [442]:

```
for a in vectorizer_tfidf_titles.get_feature_names() :
    tfidf_features_names.append(a)
```

In [443]:

```
for a in vectorizer_tfidf_essay.get_feature_names() :
    tfidf_features_names.append(a)
```

In [444]:

```
len(tfidf_features_names)
```

Out[444]:

```
16764
```

10 Positive features from TFIDF model

In [445]:

```
pos_class_prob_sorted = nb_tfidf.feature_log_prob_[1, :].argsort()[::-1][:len(bow_features_prob)]
for i in pos_class_prob_sorted[:10]:
    print(tfidf_features_names[i])
```

```
Mrs
Literacy_Language
Grades_PreK_2
Math_Science
Ms
Grades_3_5
Literacy
Mathematics
Literature_Writing
Grades_6_8
```

10 Negative features from TFIDF model

In [446]:

```
neg_class_prob_sorted = nb_tfidf.feature_log_prob_[0,:].argsort()[::-1][:len(bow_features_prob)]
for i in neg_class_prob_sorted[:10]:
    print(tfidf_features_names[i])
```

```
Mrs
Literacy_Language
Math_Science
Grades_PreK_2
Ms
Grades_3_5
Mathematics
Literacy
Literature_Writing
Grades_6_8
```

Conclusions

In [447]:

```
# Please compare all your models using Prettytable library
# http://zetcode.com/python/prettytable/

from prettytable import PrettyTable

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

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

x.add_row(["BOW", "Naive Bayes", 1.0, 0.70])
x.add_row(["TFIDF", "Naive Bayes", 0.01, 0.68])

print(x)
```

```
+------------+-------------+-----------------------+------+
| Vectorizer |    Model    | Alpha:Hyper Parameter | AUC  |
+------------+-------------+-----------------------+------+
|    BOW     | Naive Bayes |          1.0          | 0.7  |
|   TFIDF    | Naive Bayes |          0.01         | 0.68 |
+------------+-------------+-----------------------+------+
```