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 **Example:** `p036502`

Feature	Description
`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 follow enumerated values: • `Grades PreK-2` • `Grades 3-5` • `Grades 6-8` • `Grades 9-12`
`project_subject_categories`	One or more (comma-separated) su categories for the project from the fo 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, Ma & Science`

Feature	Description
`school_state`	State where school is located (Two-l U.S. postal code). **Example:** `WY`
`project_subject_subcategories`	One or more (comma-separated) su subcategories for the project. **Examp** - `Literacy` - `Literature & Writing, Social Sciences`
`project_resource_summary`	An explanation of the resources nee the project. **Example:** - `My students need hands ( 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 w submitted. **Example:** `2016-04-28 12:43:56.245`
`teacher_id`	A unique identifier for the teacher of proposed project. **Example:** `bdf8baa8fedef6bfeec7ae4ff1c`

Feature	**Description**
`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 previ submitted by the same teacher. **Exa** 2

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

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

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

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

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

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

Notes on the Essay Data

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

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

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

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

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

In [6]:

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

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

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

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

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

from tqdm import tqdm
import os

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

```
IOPub data rate exceeded.
The notebook server will temporarily stop sending output
to the client in order to avoid crashing it.
To change this limit, set the config variable
`--NotebookApp.iopub_data_rate_limit`.
```

1.1 Reading Data

In [7]:

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

In [8]:

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

```
Number of data points in train data (32414, 17)
--------------------------------------------------
The attributes of data : ['Unnamed: 0' 'id' 'teacher_id' 'teacher_prefi
x' 'school_state'
 'project_submitted_datetime' 'project_grade_category'
 'project_subject_categories' 'project_subject_subcategories'
 'project_title' 'project_essay_1' 'project_essay_2' 'project_essay_3'
 'project_essay_4' 'project_resource_summary'
 'teacher_number_of_previously_posted_projects' 'project_is_approved']
```

In [9]:

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

```
Number of data points in train data (32414, 4)
['id' 'description' 'quantity' 'price']
```

Out[9]:

	id	description	quantity	price
0	p253737	LC652 - Lakeshore Double-Space Mobile Drying Rack	1	149.00
1	p258326	Bouncy Bands for Desks (Blue support pipes)	3	14.95

1.2 preprocessing of `project_subject_categories`

In [10]:

```
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 & Hunger"]
        if 'The' in j.split(): # this will split each of the catogory based on space "Math & Science"=> "Math","&", "Science"
            j=j.replace('The','') # if we have the words "The" we are going to replace it with ''(i.e removing 'The')
        j = j.replace(' ','') # we are 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 [11]:

```
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 & Hunger"]
        if 'The' in j.split(): # this will split each of the catogory based on space "Math & Science"=> "Math","&", "Science"
            j=j.replace('The','') # if we have the words "The" we are going to replace it with ''(i.e removing 'The')
        j = j.replace(' ','') # we are 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]))
```

1.3 Text preprocessing

In [12]:

```
# 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 [13]:

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

Out[13]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL

In [14]:
```
#### 1.4.2.3 Using Pretrained Models: TFIDF weighted W2V
```

In [15]:
```
# printing some random reviews
print(project_data['essay'].values[0])
print("="*50)
print(project_data['essay'].values[150])
print("="*50)
print(project_data['essay'].values[1000])
```

```
My students are English learners that are working on English as their s
econd or third languages. We are a melting pot of refugees, immigrants,
and native-born Americans bringing the gift of language to our school.
\r\n\r\n We have over 24 languages represented in our English Learner p
rogram with students at every level of mastery.  We also have over 40 c
ountries represented with the families within our school.  Each student
brings a wealth of knowledge and experiences to us that open our eyes t
o new cultures, beliefs, and respect.\"The limits of your language are
the limits of your world.\"-Ludwig Wittgenstein  Our English learner's
have a strong support system at home that begs for more resources.  Man
y times our parents are learning to read and speak English along side o
f their children.  Sometimes this creates barriers for parents to be ab
le to help their child learn phonetics, letter recognition, and other r
eading skills.\r\n\r\nBy providing these dvd's and players, students ar
e able to continue their mastery of the English language even if no one

at home is able to assist.  All families with students within the Level
```

```
1 proficiency status, will be a offered to be a part of this program.
These educational videos will be specially chosen by the English Learne
r Teacher and will be sent home regularly to watch.  The videos are to
help the child develop early reading skills.\r\n\r\nParents that do not
have access to a dvd player will have the opportunity to check out a dv
d player to use for the year.  The plan is to use these videos and educ
ational dvd's for the years to come for other EL students.\r\nnannan
==================================================
The 51 fifth grade students that will cycle through my classroom this y
ear all love learning, at least most of the time. At our school, 97.3%
of the students receive free or reduced price lunch. Of the 560 student
s, 97.3% are minority students. \r\nThe school has a vibrant community
that loves to get together and celebrate. Around Halloween there is a w
hole school parade to show off the beautiful costumes that students wea
r. On Cinco de Mayo we put on a big festival with crafts made by the st
udents, dances, and games. At the end of the year the school hosts a ca
rnival to celebrate the hard work put in during the school year, with a
dunk tank being the most popular activity.My students will use these fi
ve brightly colored Hokki stools in place of regular, stationary, 4-leg
ged chairs. As I will only have a total of ten in the classroom and not
enough for each student to have an individual one, they will be used in
a variety of ways. During independent reading time they will be used as
special chairs students will each use on occasion. I will utilize them
in place of chairs at my small group tables during math and reading tim
es. The rest of the day they will be used by the students who need the
highest amount of movement in their life in order to stay focused on sc
hool.\r\n\r\nWhenever asked what the classroom is missing, my students
always say more Hokki Stools. They can't get their fill of the 5 stools
we already have. When the students are sitting in group with me on the
Hokki Stools, they are always moving, but at the same time doing their
work. Anytime the students get to pick where they can sit, the Hokki St
ools are the first to be taken. There are always students who head over
to the kidney table to get one of the stools who are disappointed as th
ere are not enough of them. \r\n\r\nWe ask a lot of students to sit for
7 hours a day. The Hokki stools will be a compromise that allow my stud
ents to do desk work and move at the same time. These stools will help
students to meet their 60 minutes a day of movement by allowing them to
activate their core muscles for balance while they sit. For many of my

students, these chairs will take away the barrier that exists in school
```

```
s for a child who can't sit still.nannan
==================================================
How do you remember your days of school? Was it in a sterile environmen
t with plain walls, rows of desks, and a teacher in front of the room?
A typical day in our room is nothing like that. I work hard to create a
warm inviting themed room for my students look forward to coming to eac
h day.\r\n\r\nMy class is made up of 28 wonderfully unique boys and gir
ls of mixed races in Arkansas.\r\nThey attend a Title I school, which m
eans there is a high enough percentage of free and reduced-price lunch
to qualify. Our school is an \"open classroom\" concept, which is very
unique as there are no walls separating the classrooms. These 9 and 10
year-old students are very eager learners; they are like sponges, absor
bing all the information and experiences and keep on wanting more.With
these resources such as the comfy red throw pillows and the whimsical n
autical hanging decor and the blue fish nets, I will be able to help cr
eate the mood in our classroom setting to be one of a themed nautical e
nvironment. Creating a classroom environment is very important in the s
uccess in each and every child's education. The nautical photo props wi
ll be used with each child as they step foot into our classroom for the
first time on Meet the Teacher evening. I'll take pictures of each chil
d with them, have them developed, and then hung in our classroom ready
for their first day of 4th grade.  This kind gesture will set the tone
before even the first day of school! The nautical thank you cards will
be used throughout the year by the students as they create thank you ca
rds to their team groups.\r\n\r\nYour generous donations will help me t
o help make our classroom a fun, inviting, learning environment from da
y one.\r\n\r\nIt costs lost of money out of my own pocket on resources
to get our classroom ready. Please consider helping with this project t
o make our new school year a very successful one. Thank you!nannan
```

In [16]:

```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 [17]:
```
sent = decontracted(project_data['essay'].values[20000])
print(sent)
print("="*50)
```

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

In [18]:
```
# \r \n \t remove from string python: http://texthandler.com/info/remov
e-line-breaks-python/
sent = sent.replace('\\r', ' ')
sent = sent.replace('\\"', ' ')
sent = sent.replace('\\n', ' ')
print(sent)
```

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

In [19]:

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

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

In [20]:
```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'no
t'
stopwords= ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves'
, 'you', "you're", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselve
s', 'he', 'him', 'his', 'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'it
s', 'itself', 'they', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'th
is', 'that', "that'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'h
ave', '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', 'h
ow', 'all', 'any', 'both', 'each', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 's
o', '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', "doesn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "is
n'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 [21]:
```
# Combining all the above stundents
from tqdm import tqdm
preprocessed_essays = []
# tqdm is for printing the status bar
for sentance in tqdm(project_data['essay'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e not in stopwords)
    preprocessed_essays.append(sent.lower().strip())
```

```
100%|█████████████████████████████████████████████████████████████
█████████| 32414/32414 [00:34<00:00, 940.15it/s]
```

In [22]:
```
# after preprocesing
preprocessed_essays[20000]
```

Out[22]:
```
'my kindergarten students varied disabilities ranging speech language d
elays cognitive delays gross fine motor delays autism they eager beaver
s always strive work hardest working past limitations the materials one
s i seek students i teach title i school students receive free reduced
price lunch despite disabilities limitations students love coming schoo
l come eager learn explore have ever felt like ants pants needed groove
move meeting this kids feel time the want able move learn say wobble ch
airs answer i love develop core enhances gross motor turn fine motor sk
ills they also want learn games kids not want sit worksheets they want
learn count jumping playing physical engagement key success the number
toss color shape mats make happen my students forget work fun 6 year ol
d deserves nannan'
```

1.4 Preprocessing of `project_title`

In [23]:
```
# printing some project titles.
for i in range (0,21):

    print(project_data['project_title'].values[i])
    print("="*50)
```

```
Educational Support for English Learners at Home
==================================================
Wanted: Projector for Hungry Learners
==================================================
Soccer Equipment for AWESOME Middle School Students
==================================================
Techie Kindergarteners
==================================================
Interactive Math Tools
==================================================
Flexible Seating for Mrs. Jarvis' Terrific Third Graders!!
==================================================
Chromebooks for Special Education Reading Program
==================================================
It's the 21st Century
==================================================
Targeting More Success in Class
==================================================
Just For the Love of Reading--\r\nPure Pleasure
==================================================
Reading Changes Lives
==================================================
Elevating Academics and Parent Rapports Through Technology
==================================================
Building Life Science Experiences
==================================================
Everyone deserves to be heard!
==================================================
TABLETS CAN SHOW US THE WORLD
==================================================
Making Recess Active
==================================================
Making Great LEAP's With Leapfrog!
==================================================
Technology Teaches Tomorrow's Talents Today
==================================================
Test Time
==================================================
Wiggling Our Way to Success
==================================================
Magic Carpet Ride in Our Library
==================================================
```

In [24]:

```
preprocessed_titles = []

for dataset in tqdm(project_data['project_title'].values):
    data = decontracted(dataset)  # Replacing some specific and general
 short form into proper word/stopword.
    data = re.sub(r"it's", "it is", data) # Replacing it's with it is a
s it is not part of function decontracted
    data = data.replace('\\r', ' ') # Replacing \r with space
    data = data.replace('\\"', ' ') # Replacing \ with space
    data = data.replace('\\n', ' ') # Replacing \n with space
    data = re.sub('[^A-Za-z0-9]+', ' ', data) # Replacing special chara
cters with space
    data = re.sub("\S*\d\S*", "", data).strip() # Trimming numbers cont
aining digits

    data = ' '.join(e for e in data.split() if e not in stopwords) # Re
moving stopwords
    preprocessed_titles.append(data.lower().strip()) # Creating array i
n all the lower cases.
```

```
100%|██████████████████████████████████████████████████████████████████
███████| 32414/32414 [00:02<00:00, 14616.25it/s]
```

In [25]:

```
for i in range (0,21):
    print(preprocessed_titles[i])
    print("="*50)
```

```
educational support english learners home
==================================================
wanted projector hungry learners
==================================================
soccer equipment awesome middle school students
==================================================
techie kindergarteners
==================================================
interactive math tools
```

```
==================================================
flexible seating mrs jarvis terrific third graders
==================================================
chromebooks special education reading program
==================================================
it century
==================================================
targeting more success class
==================================================
just for love reading pure pleasure
==================================================
reading changes lives
==================================================
elevating academics parent rapports through technology
==================================================
building life science experiences
==================================================
everyone deserves heard
==================================================
tablets can show us the world
==================================================
making recess active
==================================================
making great leap with leapfrog
==================================================
technology teaches tomorrow talents today
==================================================
test time
==================================================
wiggling our way success
==================================================
magic carpet ride our library
==================================================
```

1.5 Preparing data for models

In [26]:
```
project_data.columns
```

Out[26]:

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_stat
e',
       'project_submitted_datetime', 'project_grade_category', 'project
_title',
       'project_essay_1', 'project_essay_2', 'project_essay_3',
       'project_essay_4', 'project_resource_summary',
       'teacher_number_of_previously_posted_projects', 'project_is_appr
oved',
       'clean_categories', 'clean_subcategories', 'essay'],
      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/

In [27]:

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

```
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(vocabulary=list(sorted_cat_dict.keys()), l
owercase=False, binary=True)
categories_one_hot = vectorizer.fit_transform(project_data['clean_categ
ories'].values)
print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encodig ",categories_one_hot.shape
)
```

```
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearn
ing', 'SpecialNeeds', 'Health_Sports', 'Math_Science', 'Literacy_Langua
ge']
Shape of matrix after one hot encodig  (32414, 9)
```

In [28]:

```
# we use count vectorizer to convert the values into one
vectorizer = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys
()), lowercase=False, binary=True)
sub_categories_one_hot = vectorizer.fit_transform(project_data['clean_s
ubcategories'].values)
print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encodig ",sub_categories_one_hot.s
hape)
```

```
['Economics', 'CommunityService', 'FinancialLiteracy', 'ParentInvolveme
nt', 'Extracurricular', 'Civics_Government', 'ForeignLanguages', 'Nutri
tionEducation', 'Warmth', 'Care_Hunger', 'SocialSciences', 'PerformingA
rts', 'CharacterEducation', 'TeamSports', 'Other', 'College_CareerPre
p', 'Music', 'History_Geography', 'Health_LifeScience', 'EarlyDevelopme
nt', 'ESL', 'Gym_Fitness', 'EnvironmentalScience', 'VisualArts', 'Healt
h_Wellness', 'AppliedSciences', 'SpecialNeeds', 'Literature_Writing',
'Mathematics', 'Literacy']
Shape of matrix after one hot encodig  (32414, 30)
```

In [29]:

```
school_state_vectorizer = CountVectorizer(lowercase=False, binary=True)
school_state_vectorizer.fit(project_data['school_state'].values)
print(school_state_vectorizer.get_feature_names())

school_state_one_hot = school_state_vectorizer.transform(project_data[
'school_state'].values)
```

```
print("Shape of matrix after one hot encodig ",school_state_one_hot.shape)
print("the type of count vectorizer ",type(school_state_one_hot))
```

```
['AK', 'AL', 'AR', 'AZ', 'CA', 'CO', 'CT', 'DC', 'DE', 'FL', 'GA', 'H
I', 'IA', 'ID', 'IL', 'IN', 'KS', 'KY', 'LA', 'MA', 'MD', 'ME', 'MI',
'MN', 'MO', 'MS', 'MT', 'NC', 'ND', 'NE', 'NH', 'NJ', 'NM', 'NV', 'NY',
'OH', 'OK', 'OR', 'PA', 'RI', 'SC', 'SD', 'TN', 'TX', 'UT', 'VA', 'VT',
'WA', 'WI', 'WV', 'WY']
Shape of matrix after one hot encodig  (32414, 51)
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
```

In [30]:

```
# https://www.geeksforgeeks.org/python-pandas-dataframe-fillna-to-replace-null-values-in-dataframe/
project_data["teacher_prefix"].fillna("No_Prefix", inplace = True)

teacher_prefix_vectorizer = CountVectorizer(lowercase=False, binary=True)
teacher_prefix_vectorizer.fit(project_data['teacher_prefix'].values)
print(teacher_prefix_vectorizer.get_feature_names())

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

```
['Mr', 'Mrs', 'Ms', 'No_Prefix', 'Teacher']
Shape of matrix after one hot encodig  (32414, 5)
```

In [31]:

```
my_grade_counter = Counter()

for project_grade in project_data['project_grade_category'].values:

    if (' ' in project_grade):

        project_grade = project_grade.replace(" ", "~")


    my_grade_counter.update(project_grade.split())
```

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

grade_cat_vectorizer = CountVectorizer(vocabulary=list(sorted_project_grade_cat_dict.keys()), lowercase=False, binary=True)
grade_cat_vectorizer.fit(project_data['project_grade_category'].values)
print(grade_cat_vectorizer.get_feature_names())

grade_cat_one_hot = grade_cat_vectorizer.transform(project_data['project_grade_category'].values)
print("Shape of matrix after one hot encodig ",grade_cat_one_hot.shape)
```

```
['Grades~9-12', 'Grades~6-8', 'Grades~3-5', 'Grades~PreK-2']
Shape of matrix after one hot encodig  (32414, 4)
```

1.5.2 Vectorizing Text data

1.5.2.1 Bag of words

In [32]:

```
# We are considering only the words which appeared in at least 10 documents(rows or projects).
vectorizer = CountVectorizer(min_df=10)
text_bow = vectorizer.fit_transform(preprocessed_essays)
print("Shape of matrix after one hot encodig ",text_bow.shape)
```

```
Shape of matrix after one hot encodig  (32414, 10299)
```

In [33]:

```
titles_vectorizer = CountVectorizer(min_df=10)
titles_bow = titles_vectorizer.fit_transform(preprocessed_titles)
print("some sample features(unique words in the corpus)",titles_vectorizer.get_feature_names()[0:10])
print("Shape of matrix after one hot encodig ",titles_bow.shape)
print("the type of count vectorizer ",type(titles_bow))
print("the number of unique words ", titles_bow.get_shape()[1])
```

```
some sample features(unique words in the corpus) ['abc', 'about', 'acad
emic', 'access', 'accessible', 'accessing', 'accessories', 'achieve',
'achievement', 'achieving']
Shape of matrix after one hot encodig  (32414, 1601)
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the number of unique words  1601
```

1.5.2.2 TFIDF vectorizer

In [34]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10)
text_tfidf = vectorizer.fit_transform(preprocessed_essays)
print("Shape of matrix after one hot encodig ",text_tfidf.shape)
```

```
Shape of matrix after one hot encodig  (32414, 10299)
```

1.5.2.3 Using Pretrained Models: Avg W2V

In [35]:

```
'''
# Reading glove vectors in python: https://stackoverflow.com/a/3823034
9/4084039
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 o
ur 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-save-and-load-variables-in-python/

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

```
'''
```

Out[35]:

```
'\n# Reading glove vectors in python: https://stackoverflow.com/a/38230
349/4084039\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        w
ord = splitLine[0]\n        embedding = np.array([float(val) for val in
splitLine[1:]])\n        model[word] = embedding\n    print ("Done.",le
n(model)," words loaded!")\n    return model\nmodel = loadGloveModel
(\'glove.42B.300d.txt\')\n\n# ============================\nOutput:\n
  \nLoading Glove 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 preproce
d_titles:\n    words.extend(i.split(\' \'))\nprint("all the words in th
e coupus", len(words))\nwords = set(words)\nprint("the unique words in
the coupus", len(words))\n\ninter_words = set(model.keys()).intersectio
n(words)\nprint("The number of words that are present in both glove vec
tors and our coupus",       len(inter_words),"(",np.round(len(inter_wor
ds)/len(words)*100,3),"%)")\n\nwords_courpus = {}\nwords_glove = set(mo
del.keys())\nfor i in words:\n    if i in words_glove:\n        words_c
ourpus[i] = model[i]\nprint("word 2 vec length", len(words_courpus))\n
\n\n# stronging variables into pickle files python: http://www.jessicay
ung.com/how-to-use-pickle-to-save-and-load-variables-in-python/\n\nimpo
rt pickle\nwith open(\'glove_vectors\', \'wb\') as f:\n    pickle.dump
(words_courpus, f)\n\n\n'
```

In [36]:

```
# stronging variables into pickle files python: http://www.jessicayung.
com/how-to-use-pickle-to-save-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())
```

In [37]:

```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_vectors = []; # the avg-w2v for each sentence/review is stored
 in this list
```

```
for sentence in tqdm(preprocessed_essays): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/re
view
    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.append(vector)

print(len(avg_w2v_vectors))
print(len(avg_w2v_vectors[0]))
```

```
100%|██████████████████████████████████████████████████████████████████
████████| 32414/32414 [00:18<00:00, 1735.27it/s]
```

```
32414
300
```

1.5.2.3 Using Pretrained Models: TFIDF weighted W2V

In [38]:

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

In [39]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is store
d in this list
for sentence in tqdm(preprocessed_essays): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentenc
e/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 t
he tf value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentenc
e.split())) # getting the tfidf 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.append(vector)

print(len(tfidf_w2v_vectors))
print(len(tfidf_w2v_vectors[0]))
```

```
100%|██████████████████████████████████████████████████████████████████
█████████| 32414/32414 [02:27<00:00, 219.76it/s]
```

```
32414
300
```

In [40]:

```
# TFIDF on project titles
titles_tfidf_vectorizer = TfidfVectorizer(min_df=10)
titles_tfidf = titles_tfidf_vectorizer.fit_transform(preprocessed_title
s)
print("some sample features(unique words in the corpus)",titles_tfidf_v
ectorizer.get_feature_names()[10:21])
print("Shape of matrix after one hot encodig ",titles_tfidf.shape)
```

```
some sample features(unique words in the corpus) ['across', 'act', 'act
ion', 'active', 'activities', 'activity', 'add', 'adding', 'adventure',
'adventures', 'after']
Shape of matrix after one hot encodig  (32414, 1601)
```

In [41]:

```
# AVG W2V on project title
avg_w2v_titles_vectors = [];
```

```
for sentence in tqdm(preprocessed_titles):

    vector_titles = np.zeros(300)
    cnt_words_titles = 0;

    for word in sentence.split():

        if word in glove_words:

            vector += model[word]
            cnt_words_titles += 1

    if cnt_words_titles != 0:

        vector_titles /= cnt_words_titles

    avg_w2v_titles_vectors.append(vector_titles)

print(len(avg_w2v_titles_vectors))
print(len(avg_w2v_titles_vectors[0]))
```

```
100%|████████████████████████████████████████████████████████████████
███████| 32414/32414 [00:01<00:00, 31651.95it/s]
```

```
32414
300
```

In [42]:

```
# TFIDF weighted W2V on project_title
titles_tfidf_model = TfidfVectorizer()
titles_tfidf_model.fit(preprocessed_titles)
titles_dictionary = dict(zip(titles_tfidf_model.get_feature_names(), li
st(titles_tfidf_model.idf_)))
titles_tfidf_words = set(titles_tfidf_model.get_feature_names())
```

In [43]:

```
titles_tfidf_w2v_vectors = [];

for titles_sentence in tqdm(preprocessed_titles):

    titles_vector = np.zeros(300)
    titles_tfidf_weight = 0;

    for word in titles_sentence.split():

        if (word in glove_words) and (word in titles_tfidf_words):

            titles_vec = model[word]

            titles_tf_idf = titles_dictionary[word]*(titles_sentence.co
unt(word)/len(titles_sentence.split()))
            titles_vector += (titles_vec * titles_tf_idf)
            titles_tfidf_weight += titles_tf_idf

    if titles_tfidf_weight != 0:

        titles_vector /= titles_tfidf_weight

    titles_tfidf_w2v_vectors.append(titles_vector)

print(len(titles_tfidf_w2v_vectors))
print(len(titles_tfidf_w2v_vectors[0]))
```

```
100%|██████████████████████████████████████████████████████████████████
███████| 32414/32414 [00:02<00:00, 15288.99it/s]
```

```
32414
300
```

1.5.3 Vectorizing Numerical features

In [44]:

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

In [45]:

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

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

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

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

```
Mean : 51.22791725797495, Standard deviation : 149.72149781041014
```

In [46]:

```
price_standardized
```

Out[46]:

```
array([[ 0.65302635],
       [-0.24230266],
       [-0.2857166 ],
       ...,
       [-0.10978996],
       [-0.23014676],
       [-0.10250978]])
```

1.5.4 Merging all the above features

- we need to merge all the numerical vectors i.e catogorical, text, numerical vectors

In [47]:

```
print(categories_one_hot.shape)
print(sub_categories_one_hot.shape)
print(text_bow.shape)
print(price_standardized.shape)
```

```
(32414, 9)
(32414, 30)
(32414, 10299)
(32414, 1)
```

In [49]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X = hstack((categories_one_hot, sub_categories_one_hot, text_bow, price_standardized))
X.shape
```

Out[49]:

```
(32414, 10339)
```

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

2.1 Splitting data into Train and cross validation(or test): Stratified Sampling

In [50]:
```
approved_project = project_data['project_is_approved'].values
project_data.drop(['project_is_approved'], axis=1, inplace=True)
project_data.head(1)
```

Out[50]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN

In [51]:

```
# Data splitting

from sklearn.model_selection import train_test_split

# Splitting in train and test
X_train, X_test, y_train, y_test = train_test_split(project_data, appro
ved_project, test_size=0.33, stratify=approved_project)

# Splitting in Train Test and Cross Validation
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_
size=0.33, stratify=y_train)
```

2.2 Make Data Model Ready: encoding numerical, categorical features

In [52]:

```
# Vectorizing Categories on Train, Test and CV data
from sklearn.feature_extraction.text import CountVectorizer

vectorizer = CountVectorizer(vocabulary=list(sorted_cat_dict.keys()), l
owercase=False, binary=True)
```

```
# Fit only to train data
vectorizer.fit(X_train['clean_categories'].values)

# Transform to train,test and CV data
X_Train_categories_one_hot = vectorizer.transform(X_train['clean_catego
ries'].values)
X_Test_categories_one_hot = vectorizer.transform(X_test['clean_categori
es'].values)
X_CV_categories_one_hot = vectorizer.transform(X_cv['clean_categories']
.values)

print("Shape of train matrix after one hot encodig ",X_Train_categories
_one_hot.shape)
print("Shape of test matrix after one hot encodig ",X_Test_categories_o
ne_hot.shape)
print("Shape of cv matrix after one hot encodig ",X_CV_categories_one_h
ot.shape)
```

```
Shape of train matrix after one hot encodig  (14550, 9)
Shape of test matrix after one hot encodig  (10697, 9)
Shape of cv matrix after one hot encodig  (7167, 9)
```

In [53]:

```
# Vectorizing subcategories on train, test and cv

vectorizer = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys
()), lowercase=False, binary=True)
vectorizer.fit(X_train['clean_subcategories'].values)

X_Train_sub_categories_one_hot = vectorizer.transform(X_train['clean_su
bcategories'].values)
X_Test_sub_categories_one_hot = vectorizer.transform(X_test['clean_subc
ategories'].values)
X_CV_sub_categories_one_hot = vectorizer.transform(X_cv['clean_subcateg
ories'].values)

print("Shape of train matrix after one hot encodig ",X_Train_sub_catego
ries_one_hot.shape)
print("Shape of test matrix after one hot encodig ",X_Test_sub_categori
es_one_hot.shape)
```

```
print("Shape of cv matrix after one hot encodig ",X_CV_sub_categories_o
ne_hot.shape)
```

```
Shape of train matrix after one hot encodig  (14550, 30)
Shape of test matrix after one hot encodig  (10697, 30)
Shape of cv matrix after one hot encodig  (7167, 30)
```

In [54]:
```
# Vectorizing school state on train , test and cv

school_state_vectorizer = CountVectorizer(lowercase=False, binary=True)

school_state_vectorizer.fit(X_train['school_state'].values)
print(school_state_vectorizer.get_feature_names())

X_Train_school_state_one_hot = school_state_vectorizer.transform(X_trai
n['school_state'].values)
X_Test_school_state_one_hot = school_state_vectorizer.transform(X_test[
'school_state'].values)
X_CV_school_state_one_hot = school_state_vectorizer.transform(X_cv['sch
ool_state'].values)

print("Shape of train matrix after one hot encodig ",X_Train_school_sta
te_one_hot.shape)
print("Shape of test matrix after one hot encodig ",X_Test_school_state
_one_hot.shape)
print("Shape of cv matrix after one hot encodig ",X_CV_school_state_one
_hot.shape)

print("the type of count vectorizer ",type(X_Train_school_state_one_hot
))
print("the type of count vectorizer ",type(X_Test_school_state_one_hot
))
print("the type of count vectorizer ",type(X_CV_school_state_one_hot))
```

```
['AK', 'AL', 'AR', 'AZ', 'CA', 'CO', 'CT', 'DC', 'DE', 'FL', 'GA', 'H
I', 'IA', 'ID', 'IL', 'IN', 'KS', 'KY', 'LA', 'MA', 'MD', 'ME', 'MI',
'MN', 'MO', 'MS', 'MT', 'NC', 'ND', 'NE', 'NH', 'NJ', 'NM', 'NV', 'NY',
'OH', 'OK', 'OR', 'PA', 'RI', 'SC', 'SD', 'TN', 'TX', 'UT', 'VA', 'VT',
'WA', 'WI', 'WV', 'WY']
Shape of train matrix after one hot encodig  (14550, 51)
Shape of test matrix after one hot encodig  (10697, 51)
Shape of cv matrix after one hot encodig  (7167, 51)
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
```

In [55]:

```
# Vectorizing teacher prefix on train , test and cv

project_data["teacher_prefix"].fillna("No_Prefix", inplace = True)

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

print(teacher_prefix_vectorizer.get_feature_names())

X_Train_teacher_prefix_one_hot = teacher_prefix_vectorizer.transform(X_train['teacher_prefix'].values)
X_Test_teacher_prefix_one_hot = teacher_prefix_vectorizer.transform(X_test['teacher_prefix'].values)
X_CV_teacher_prefix_one_hot = teacher_prefix_vectorizer.transform(X_cv['teacher_prefix'].values)

print("Shape of train matrix after one hot encodig ",X_Train_teacher_prefix_one_hot.shape)
print("Shape of test matrix after one hot encodig ",X_Test_teacher_prefix_one_hot.shape)
print("Shape of cv matrix after one hot encodig ",X_CV_teacher_prefix_one_hot.shape)
```

```
['Mr', 'Mrs', 'Ms', 'No_Prefix', 'Teacher']
Shape of train matrix after one hot encodig  (14550, 5)
Shape of test matrix after one hot encodig  (10697, 5)
Shape of cv matrix after one hot encodig  (7167, 5)
```

In [56]:

```
# Vectorizing grade category on train , test and cv

my_grade_counter = Counter()
```

```
for project_grade in project_data['project_grade_category'].values:

    if (' ' in project_grade):

        project_grade = project_grade.replace(" ", "~")

    my_grade_counter.update(project_grade.split())

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

grade_cat_vectorizer = CountVectorizer(vocabulary=list(sorted_project_grade_cat_dict.keys()), lowercase=False, binary=True)
grade_cat_vectorizer.fit(X_train['project_grade_category'].values)
print(grade_cat_vectorizer.get_feature_names())

X_Train_grade_cat_one_hot = grade_cat_vectorizer.transform(X_train['project_grade_category'].values)
X_Test_grade_cat_one_hot = grade_cat_vectorizer.transform(X_test['project_grade_category'].values)
X_CV_grade_cat_one_hot = grade_cat_vectorizer.transform(X_cv['project_grade_category'].values)

print("Shape of train matrix after one hot encodig ",X_Train_grade_cat_one_hot.shape)
print("Shape of test matrix after one hot encodig ",X_Test_grade_cat_one_hot.shape)
print("Shape of cv matrix after one hot encodig ",X_CV_grade_cat_one_hot.shape)
```

```
['Grades~9-12', 'Grades~6-8', 'Grades~3-5', 'Grades~PreK-2']
Shape of train matrix after one hot encodig  (14550, 4)
Shape of test matrix after one hot encodig  (10697, 4)
Shape of cv matrix after one hot encodig  (7167, 4)
```

2.3 Make Data Model Ready: encoding eassay, and project_title

In [57]:

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

In [58]:

```
# preprocessing essay train data
from tqdm import tqdm
X_Train_preprocessed_essays = []
# tqdm is for printing the status bar
for sentence in tqdm(X_train['essay'].values):
    X_Train_essay_sent = decontracted(sentance)
    X_Train_essay_sent = X_Train_essay_sent.replace('\\r', ' ')
    X_Train_essay_sent = X_Train_essay_sent.replace('\\"', ' ')
    X_Train_essay_sent = X_Train_essay_sent.replace('\\n', ' ')
    X_Train_essay_sent = re.sub('[^A-Za-z0-9]+', ' ', X_Train_essay_sen
t)

    X_Train_essay_sent = ' '.join(e for e in X_Train_essay_sent.split()
 if e.lower() not in stopwords)
    X_Train_preprocessed_essays.append(X_Train_essay_sent.lower().strip
())
```

```
100%|████████████████████████████████████████████████████████████████████
█████████| 14550/14550 [00:25<00:00, 565.05it/s]
```

In [59]:

```
# preprocessing essay test data
from tqdm import tqdm
X_Test_preprocessed_essays = []
# tqdm is for printing the status bar
for sentence in tqdm(X_test['essay'].values):
    X_Test_essay_sent = decontracted(sentence)
    X_Test_essay_sent = X_Test_essay_sent.replace('\\r', ' ')
    X_Test_essay_sent = X_Test_essay_sent.replace('\\"', ' ')
    X_Test_essay_sent = X_Test_essay_sent.replace('\\n', ' ')
    X_Test_essay_sent = re.sub('[^A-Za-z0-9]+', ' ', X_Test_essay_sent)

    X_Test_essay_sent = ' '.join(e for e in X_Test_essay_sent.split() i
f e.lower() not in stopwords)
    X_Test_preprocessed_essays.append(X_Test_essay_sent.lower().strip
())
```

```
100%|██████████████████████████████████████████████████████████████████
█████████| 10697/10697 [00:12<00:00, 857.41it/s]
```

In [60]:

```
# preprocessing essay cv data
from tqdm import tqdm
X_CV_preprocessed_essays = []
# tqdm is for printing the status bar
for sentence in tqdm(X_cv['essay'].values):
    X_CV_essay_sent = decontracted(sentence)
    X_CV_essay_sent = X_CV_essay_sent.replace('\\r', ' ')
    X_CV_essay_sent = X_CV_essay_sent.replace('\\"', ' ')
    X_CV_essay_sent = X_CV_essay_sent.replace('\\n', ' ')
    X_CV_essay_sent = re.sub('[^A-Za-z0-9]+', ' ', X_CV_essay_sent)

    X_CV_essay_sent = ' '.join(e for e in X_CV_essay_sent.split() if e.
lower() not in stopwords)
    X_CV_preprocessed_essays.append(X_CV_essay_sent.lower().strip())
```

```
100%|██████████████████████████████████████████████████████████████████
███████████| 7167/7167 [00:08<00:00, 779.95it/s]
```

In [61]:

```
# preprocessing project title train data
X_Train_preprocessed_titles = []

for dataset in tqdm(X_train['project_title'].values):
    data = decontracted(dataset)  # Replacing some specific and general
 short form into proper word/stopword.
    data = re.sub(r"it's", "it is", data) # Replacing it's with it is a
s it is not part of function decontracted
    data = data.replace('\\r', ' ') # Replacing \r with space
    data = data.replace('\\"', ' ') # Replacing \ with space
    data = data.replace('\\n', ' ') # Replacing \n with space
    data = re.sub('[^A-Za-z0-9]+', ' ', data) # Replacing special chara
cters with space
    data = re.sub("\S*\d\S*", "", data).strip() # Trimming numbers cont
```

```
aining digits

    data = ' '.join(e for e in data.split() if e not in stopwords) # Re
moving stopwords
    X_Train_preprocessed_titles.append(data.lower().strip()) # Creating
 array in all the lower cases.
```

```
100%|█████████████████████████████████████████████████████████████████
███████| 14550/14550 [00:00<00:00, 15604.44it/s]
```

In [62]:

```
# preprocessing project title test data
X_Test_preprocessed_titles = []

for dataset in tqdm(X_test['project_title'].values):
    data = decontracted(dataset)  # Replacing some specific and general
 short form into proper word/stopword.
    data = re.sub(r"it's", "it is", data) # Replacing it's with it is a
s it is not part of function decontracted
    data = data.replace('\\r', ' ') # Replacing \r with space
    data = data.replace('\\"', ' ') # Replacing \ with space
    data = data.replace('\\n', ' ') # Replacing \n with space
    data = re.sub('[^A-Za-z0-9]+', ' ', data) # Replacing special chara
cters with space
    data = re.sub("\S*\d\S*", "", data).strip() # Trimming numbers cont
aining digits

    data = ' '.join(e for e in data.split() if e not in stopwords) # Re
moving stopwords
    X_Test_preprocessed_titles.append(data.lower().strip()) # Creating
 array in all the lower cases.
```

```
100%|█████████████████████████████████████████████████████████████████
███████| 10697/10697 [00:00<00:00, 15848.55it/s]
```

In [63]:

```
# preprocessing project title cv data
X_CV_preprocessed_titles = []

for dataset in tqdm(X_cv['project_title'].values):
    data = decontracted(dataset)  # Replacing some specific and general
```

```
 short form into proper word/stopword.
    data = re.sub(r"it's", "it is", data) # Replacing it's with it is as it is not part of function decontracted
    data = data.replace('\\r', ' ') # Replacing \r with space
    data = data.replace('\\"', ' ') # Replacing \ with space
    data = data.replace('\\n', ' ') # Replacing \n with space
    data = re.sub('[^A-Za-z0-9]+', ' ', data) # Replacing special characters with space
    data = re.sub("\S*\d\S*", "", data).strip() # Trimming numbers containing digits

    data = ' '.join(e for e in data.split() if e not in stopwords) # Removing stopwords
    X_CV_preprocessed_titles.append(data.lower().strip()) # Creating array in all the lower cases.
```

```
100%|██████████████████████████████████████████████████| 7167/7167 [00:00<00:00, 15223.94it/s]
```

In [64]:

```
# BOW Essay train, test and cv data
# We are considering only the words which appeared in at least 10 documents(rows or projects).
vectorizer = CountVectorizer(min_df=10)
vectorizer.fit(X_Train_preprocessed_essays)

X_Train_essay_bow = vectorizer.transform(X_Train_preprocessed_essays)
X_Test_essay_bow = vectorizer.transform(X_Test_preprocessed_essays)
X_CV_essay_bow = vectorizer.transform(X_CV_preprocessed_essays)

print("Shape of train matrix after one hot encodig ",X_Train_essay_bow.shape)
print("Shape of test matrix after one hot encodig ",X_Test_essay_bow.shape)
print("Shape of CV matrix after one hot encodig ",X_CV_essay_bow.shape)
```

```
Shape of train matrix after one hot encodig  (14550, 133)
Shape of test matrix after one hot encodig  (10697, 133)
Shape of CV matrix after one hot encodig  (7167, 133)
```

In [65]:
```
# BOW title train,test and cv data

titles_vectorizer = CountVectorizer(min_df=10)
titles_vectorizer.fit(X_Train_preprocessed_titles)

X_Train_titles_bow = titles_vectorizer.transform(X_Train_preprocessed_t
itles)
X_Test_titles_bow = titles_vectorizer.transform(X_Test_preprocessed_tit
les)
X_CV_titles_bow = titles_vectorizer.transform(X_CV_preprocessed_titles)

print("some sample features(unique words in the corpus)",titles_vectori
zer.get_feature_names()[0:10])
print("Shape of train matrix after one hot encodig ",X_Train_titles_bow
.shape)
print("Shape of test matrix after one hot encodig ",X_Test_titles_bow.s
hape)
print("Shape of CV matrix after one hot encodig ",X_CV_titles_bow.shape
)
```

```
some sample features(unique words in the corpus) ['abc', 'about', 'acad
emic', 'access', 'accessible', 'accessories', 'achieve', 'achievement',
'action', 'active']
Shape of train matrix after one hot encodig  (14550, 873)
Shape of test matrix after one hot encodig  (10697, 873)
Shape of CV matrix after one hot encodig  (7167, 873)
```

In [66]:
```
#TFIDF essay train,test and cv data

vectorizer = TfidfVectorizer(min_df=10)
vectorizer.fit(X_Train_preprocessed_essays)

X_Train_essay_tfidf = vectorizer.transform(X_Train_preprocessed_essays)
X_Test_essay_tfidf = vectorizer.transform(X_Test_preprocessed_essays)
X_CV_essay_tfidf = vectorizer.transform(X_CV_preprocessed_essays)

print("Shape of train matrix after one hot encodig ",X_Train_essay_tfid
f.shape)
print("Shape of test matrix after one hot encodig ",X_Test_essay_tfidf.
shape)
print("Shape of CV matrix after one hot encodig ",X_CV_essay_tfidf.shap
e)
```

```
Shape of train matrix after one hot encodig  (14550, 133)
Shape of test matrix after one hot encodig  (10697, 133)
Shape of CV matrix after one hot encodig  (7167, 133)
```

In [67]:

```
# TFIDF on project titles train,test and cv data

titles_tfidf_vectorizer = TfidfVectorizer(min_df=10)
titles_tfidf_vectorizer.fit(X_Train_preprocessed_titles)

X_Train_titles_tfidf = titles_vectorizer.transform(X_Train_preprocessed
_titles)
X_Test_titles_tfidf = titles_vectorizer.transform(X_Test_preprocessed_t
itles)
X_CV_titles_tfidf = titles_vectorizer.transform(X_CV_preprocessed_title
s)

print("Shape of train matrix after one hot encodig ",X_Train_titles_tfi
df.shape)
print("Shape of test matrix after one hot encodig ",X_Test_titles_tfidf
.shape)
print("Shape of CV matrix after one hot encodig ",X_CV_titles_tfidf.sha
pe)
```

```
Shape of train matrix after one hot encodig  (14550, 873)
Shape of test matrix after one hot encodig  (10697, 873)
Shape of CV matrix after one hot encodig  (7167, 873)
```

In [68]:

```
# average Word2Vec essay on train
# compute average word2vec for each review.
X_Train_avg_w2v_vectors = []; # the avg-w2v for each sentence/review is
 stored in this list
for sentence in tqdm(X_Train_preprocessed_essays): # for each review/se
ntence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/re
```

```
view
    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
    X_Train_avg_w2v_vectors.append(vector)

print(len(X_Train_avg_w2v_vectors))
print(len(X_Train_avg_w2v_vectors[0]))
```

```
100%|██████████████████████████████████████████████████
████████| 14550/14550 [00:10<00:00, 1342.02it/s]
```

```
14550
300
```

In [69]:

```
# average Word2Vec essay on test
# compute average word2vec for each review.
X_Test_avg_w2v_vectors = []; # the avg-w2v for each sentence/review is
 stored in this list
for sentence in tqdm(X_Test_preprocessed_essays): # for each review/sen
tence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/re
view
    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
    X_Test_avg_w2v_vectors.append(vector)

print(len(X_Test_avg_w2v_vectors))
print(len(X_Test_avg_w2v_vectors[0]))
```

```
100%|██████████████████████████████████████████████████
████████| 10697/10697 [00:05<00:00, 1859.66it/s]
```

```
10697
300
```

In [70]:

```python
# average Word2Vec essay on cv
# compute average word2vec for each review.
X_CV_avg_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_CV_preprocessed_essays): # for each review/sentence
    vector = np.zeros(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
    X_CV_avg_w2v_vectors.append(vector)

print(len(X_CV_avg_w2v_vectors))
print(len(X_CV_avg_w2v_vectors[0]))
```

```
100%|██████████████████████████████████████████████████████████████████
██████████| 7167/7167 [00:03<00:00, 1871.72it/s]
```

```
7167
300
```

In [71]:

```python
# AVG W2V on project title train
X_Train_avg_w2v_titles_vectors = [];

for sentence in tqdm(X_Train_preprocessed_titles):

    vector_titles = np.zeros(300)
    cnt_words_titles = 0;

    for word in sentence.split():
        if word in glove_words:

            vector += model[word]
            cnt_words_titles += 1

    if cnt_words_titles != 0:

        vector_titles /= cnt_words_titles

    X_Train_avg_w2v_titles_vectors.append(vector_titles)

print(len(X_Train_avg_w2v_titles_vectors))
print(len(X_Train_avg_w2v_titles_vectors[0]))
```

```
100%|██████████████████████████████████████████████████████████████████
███████| 14550/14550 [00:00<00:00, 38543.41it/s]
```

```
14550
300
```

In [72]:

```
# AVG W2V on project title test
X_Test_avg_w2v_titles_vectors = [];

for sentence in tqdm(X_Test_preprocessed_titles):

    vector_titles = np.zeros(300)
    cnt_words_titles = 0;

    for word in sentence.split():

        if word in glove_words:

            vector += model[word]
            cnt_words_titles += 1

    if cnt_words_titles != 0:

        vector_titles /= cnt_words_titles

    X_Test_avg_w2v_titles_vectors.append(vector_titles)
```

```
print(len(X_Test_avg_w2v_titles_vectors))
print(len(X_Test_avg_w2v_titles_vectors[0]))
```

```
100%|██████████████████████████████████████████████████████████████████
███████| 10697/10697 [00:00<00:00, 40100.70it/s]
```

```
10697
300
```

In [73]:

```
# AVG W2V on project title cv
X_CV_avg_w2v_titles_vectors = [];

for sentence in tqdm(X_CV_preprocessed_titles):

    vector_titles = np.zeros(300)
    cnt_words_titles = 0;

    for word in sentence.split():

        if word in glove_words:

            vector += model[word]
            cnt_words_titles += 1

    if cnt_words_titles != 0:

        vector_titles /= cnt_words_titles

    X_CV_avg_w2v_titles_vectors.append(vector_titles)

print(len(X_CV_avg_w2v_titles_vectors))
print(len(X_CV_avg_w2v_titles_vectors[0]))
```

```
100%|██████████████████████████████████████████████████████████████████
█████████| 7167/7167 [00:00<00:00, 40520.50it/s]
```

```
7167
300
```

In [74]:

```
# TFIDF W2V
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_Train_preprocessed_essays)
# we are converting a dictionary with word as a key, and the idf as a v
alue
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model
.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())
```

In [75]:

```
# TFIDF w2v essay train
# compute average word2vec for each review.
X_Train_tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review
 is stored in this list
for sentence in tqdm(X_Train_preprocessed_essays): # for each review/se
ntence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentenc
e/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 t
he tf value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentenc
e.split())) # getting the tfidf 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
    X_Train_tfidf_w2v_vectors.append(vector)

print(len(X_Train_tfidf_w2v_vectors))
print(len(X_Train_tfidf_w2v_vectors[0]))
```

```
100%|████████████████████████████████████████████████████████
████████| 14550/14550 [02:01<00:00, 120.21it/s]
```

```
14550
300
```

In [76]:

```
# TFIDF w2v essay test
# compute average word2vec for each review.
X_Test_tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review i
s stored in this list
for sentence in tqdm(X_Test_preprocessed_essays): # for each review/sen
tence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentenc
e/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 t
he tf value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentenc
e.split())) # getting the tfidf 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
    X_Test_tfidf_w2v_vectors.append(vector)

print(len(X_Test_tfidf_w2v_vectors))
print(len(X_Test_tfidf_w2v_vectors[0]))
```

```
100%|██████████████████████████████████████████████████████████████████
█████████| 10697/10697 [00:11<00:00, 924.68it/s]
```

```
10697
300
```

In [77]:

```
# TFIDF w2v essay cv
# compute average word2vec for each review.
X_CV_tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is
 stored in this list
for sentence in tqdm(X_CV_preprocessed_essays): # for each review/sente
nce
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentenc
e/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 t
he tf value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentenc
e.split())) # getting the tfidf 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
    X_CV_tfidf_w2v_vectors.append(vector)

print(len(X_CV_tfidf_w2v_vectors))
print(len(X_CV_tfidf_w2v_vectors[0]))
```

```
100%|██████████████████████████████████████████████████████████████████
███████████| 7167/7167 [00:08<00:00, 885.99it/s]
```

```
7167
300
```

In [78]:

```
# TFIDF weighted W2V on project_title
titles_tfidf_model = TfidfVectorizer()
titles_tfidf_model.fit(X_Train_preprocessed_titles)
titles_dictionary = dict(zip(titles_tfidf_model.get_feature_names(), li
st(titles_tfidf_model.idf_)))
titles_tfidf_words = set(titles_tfidf_model.get_feature_names())
```

In [79]:

```
# TFIDF w2v title train
X_Train_titles_tfidf_w2v_vectors = [];

for titles_sentence in tqdm(X_Train_preprocessed_titles):

    titles_vector = np.zeros(300)
    titles_tfidf_weight = 0;
    for word in titles_sentence.split():

        if (word in glove_words) and (word in titles_tfidf_words):

            titles_vec = model[word]

            titles_tf_idf = titles_dictionary[word]*(titles_sentence.co
unt(word)/len(titles_sentence.split()))
            titles_vector += (titles_vec * titles_tf_idf)
            titles_tfidf_weight += titles_tf_idf

    if titles_tfidf_weight != 0:

        titles_vector /= titles_tfidf_weight

    X_Train_titles_tfidf_w2v_vectors.append(titles_vector)

print(len(X_Train_titles_tfidf_w2v_vectors))
print(len(X_Train_titles_tfidf_w2v_vectors[0]))
```

```
100%|██████████████████████████████████████████████████████████████████
███████| 14550/14550 [00:00<00:00, 17129.18it/s]
```

```
14550
300
```

In [80]:

```
# TFIDF w2v title train
X_Test_titles_tfidf_w2v_vectors = [];

for titles_sentence in tqdm(X_Test_preprocessed_titles):

    titles_vector = np.zeros(300)
    titles_tfidf_weight = 0;

    for word in titles_sentence.split():

        if (word in glove_words) and (word in titles_tfidf_words):

            titles_vec = model[word]
            titles_tf_idf = titles_dictionary[word]*(titles_sentence.count(word)/len(titles_sentence.split()))
            titles_vector += (titles_vec * titles_tf_idf)
            titles_tfidf_weight += titles_tf_idf

    if titles_tfidf_weight != 0:

        titles_vector /= titles_tfidf_weight

    X_Test_titles_tfidf_w2v_vectors.append(titles_vector)

print(len(X_Test_titles_tfidf_w2v_vectors))
print(len(X_Test_titles_tfidf_w2v_vectors[0]))
```

```
100%|██████████████████████████████████████████████████| 10697/10697 [00:00<00:00, 18879.07it/s]
```

```
10697
300
```

In [81]:

```
# TFIDF w2v title cv
X_CV_titles_tfidf_w2v_vectors = [];

for titles_sentence in tqdm(X_CV_preprocessed_titles):

    titles_vector = np.zeros(300)
    titles_tfidf_weight = 0;

    for word in titles_sentence.split():

        if (word in glove_words) and (word in titles_tfidf_words):

            titles_vec = model[word]

            titles_tf_idf = titles_dictionary[word]*(titles_sentence.count(word)/len(titles_sentence.split()))
            titles_vector += (titles_vec * titles_tf_idf)
            titles_tfidf_weight += titles_tf_idf

    if titles_tfidf_weight != 0:
        titles_vector /= titles_tfidf_weight

    X_CV_titles_tfidf_w2v_vectors.append(titles_vector)

print(len(X_CV_titles_tfidf_w2v_vectors))
print(len(X_CV_titles_tfidf_w2v_vectors[0]))
```

```
100%|██████████████████████████████████████████████████████████████████
█████████| 7167/7167 [00:00<00:00, 16954.29it/s]
```

```
7167
300
```

In [84]:

```
# Vectorizing numerical feature

# Merging price data with train, test and cv
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 [85]:

```
# Standardizing price train test and cv data

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))

X_train_price_norm = normalizer.transform(X_train['price'].values.resha
pe(-1,1))
X_test_price_norm = normalizer.transform(X_test['price'].values.reshape
(-1,1))
X_cv_price_norm = normalizer.transform(X_cv['price'].values.reshape(-1,
1))
```

```
print("After vectorizations")
print(X_train_price_norm.shape, y_train.shape)
print(X_test_price_norm.shape, y_test.shape)
print(X_cv_price_norm.shape, y_cv.shape)
print("="*100)
```

```
After vectorizations
(14550, 1) (14550,)
(10697, 1) (10697,)
(7167, 1) (7167,)
=======================================================================
=============================
```

2.4 Appling NB() on different kind of featurization as mentioned in the instructions

Apply Naive Bayes on different kind of featurization as mentioned in the instructions
For Every model that you work on make sure you do the step 2 and step 3 of instrucations

2.4.1 Applying Naive Bayes on BOW, SET 1

In [86]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/40840
39
from scipy.sparse import hstack

# Train data stack
X_tr = hstack((X_Train_categories_one_hot,X_Train_sub_categories_one_ho
t,X_Train_school_state_one_hot,
               X_Train_teacher_prefix_one_hot,X_Train_grade_cat_one_hot
,X_Train_essay_bow,X_Train_titles_bow,
               X_train_price_norm)).tocsr()

# CV data Stack
X_cr = hstack((X_CV_categories_one_hot,X_CV_sub_categories_one_hot,X_CV
_school_state_one_hot,
               X_CV_teacher_prefix_one_hot,X_CV_grade_cat_one_hot,X_CV_
essay_bow,X_CV_titles_bow,
               X_cv_price_norm)).tocsr()

# Test Data Stack
X_te = hstack((X_Test_categories_one_hot,X_Test_sub_categories_one_hot,
X_Test_school_state_one_hot,
               X_Test_teacher_prefix_one_hot,X_Test_grade_cat_one_hot,X
_Test_essay_bow,X_Test_titles_bow,
               X_test_price_norm)).tocsr()

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

```
Final Data matrix
(14550, 1106) (14550,)
(7167, 1106) (7167,)
(10697, 1106) (10697,)
======================================================================
==============================
```

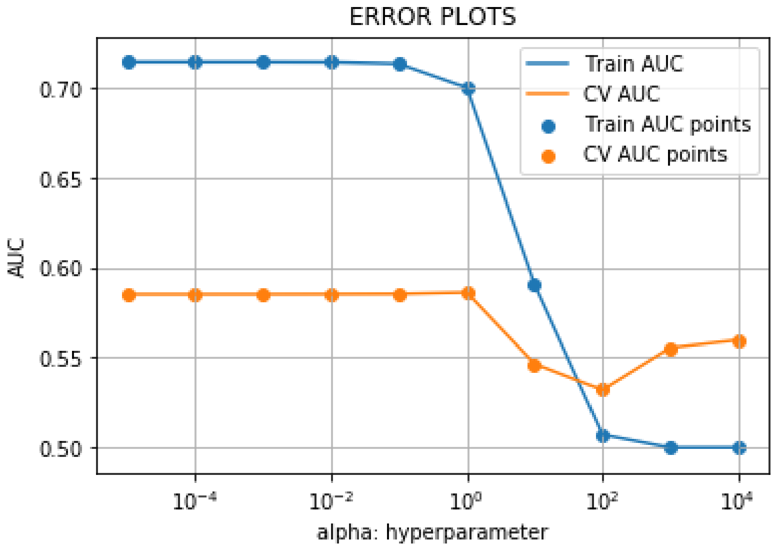
In [87]:

```
def batch_predict(clf, data):

    y_data_pred = []

    # Changing the shape of predicted data in the multiple of 1000
    tr_loop = data.shape[0] - data.shape[0]%1000

    # Running the loop for each 1000th data
    for i in range(0, tr_loop, 1000):

        # Predicting probability
        y_data_pred.extend(clf.predict_proba(data[i:i+1000])[:,1])

    y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])
    return y_data_pred
```

In [89]:

```
# Plotting error plot, AUC vs alpha plot to get best alpha

from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import roc_auc_score

train_auc = []
cv_auc = []

# Execute for different alpha values
alpha = [0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000]

for i in tqdm(alpha):

    neigh = MultinomialNB(alpha=i)
    neigh.fit(X_tr, y_train)

    y_train_pred = batch_predict(neigh, X_tr)
    y_cv_pred = batch_predict(neigh, X_cr)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
    # not the predicted outputs
    train_auc.append(roc_auc_score(y_train, y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

plt.plot(alpha, train_auc, label='Train AUC')
plt.plot(alpha, cv_auc, label='CV AUC')

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

plt.xscale('log')
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```

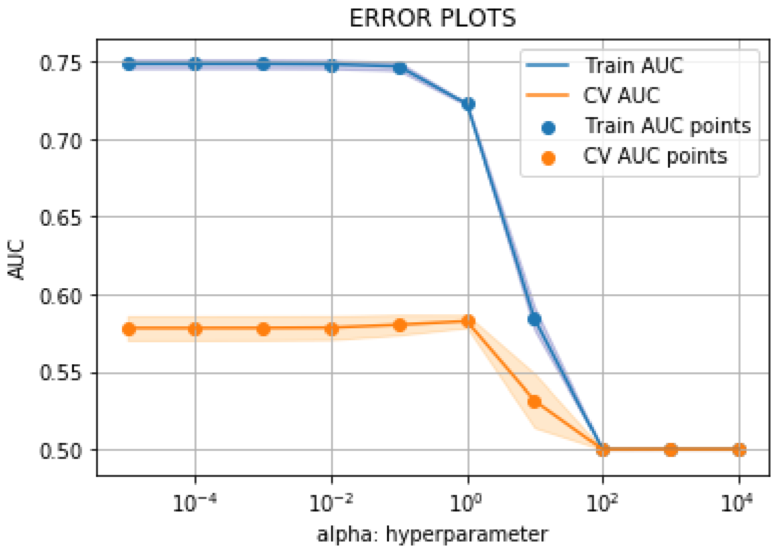
```
100%|████████████████████████████████████████
██████████| 10/10 [00:01<00:00,  8.44it/s]
```



In [90]:

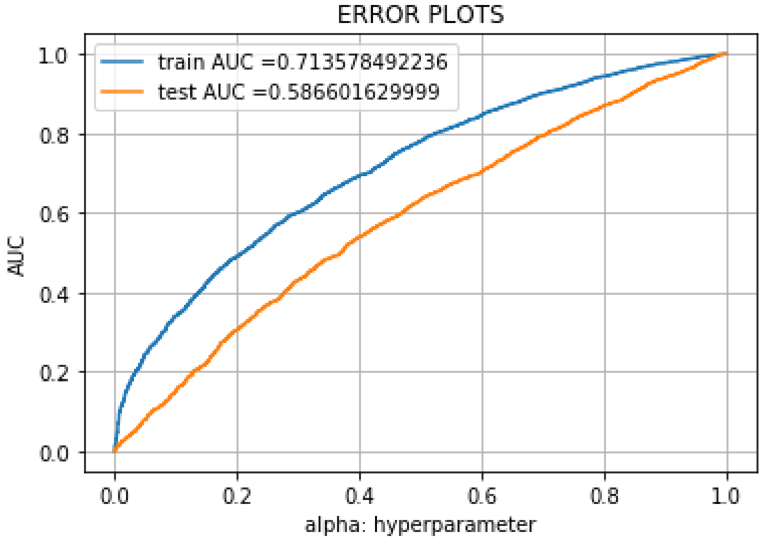
```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selec
tion.GridSearchCV.html
from sklearn.model_selection import GridSearchCV

neigh = MultinomialNB()
parameters = {'alpha':[0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1
000, 10000]}
clf = GridSearchCV(neigh, parameters, cv=3, 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']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']

plt.plot(parameters['alpha'], train_auc, label='Train AUC')
```

```
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['alpha'],train_auc - train_auc_std,train_auc + train_auc_std,alpha=0.2,color='darkblue')

plt.plot(parameters['alpha'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['alpha'],cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='darkorange')

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

plt.xscale("log")
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



In [100]:

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


neigh = MultinomialNB(alpha=0.1)
neigh.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 = batch_predict(neigh, X_tr)
y_test_pred = batch_predict(neigh, 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("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



In [101]:

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

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

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

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

In [102]:

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

```
====================================================================================================
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.777
[[1123 1123]
 [2721 9583]]
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.836
[[ 805  847]
 [3227 5818]]
```

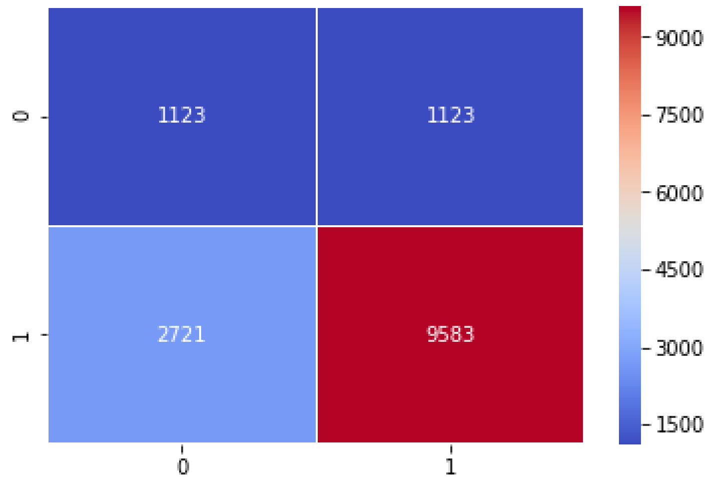
In [107]:

```python
# https://seaborn.pydata.org/generated/seaborn.heatmap.html
# https://stackoverflow.com/questions/29647749/seaborn-showing-scientific-notation-in-heatmap-for-3-digit-numbers

# Train Confusion Matrix Heatmap
train_confusion_matrix = confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr))

print("Train Confusion Matrix")
sns.heatmap(train_confusion_matrix,annot=True,linewidth = 0.1, cmap='coolwarm', fmt='g')
```

```
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.777
Train Confusion Matrix
```

Out[107]:

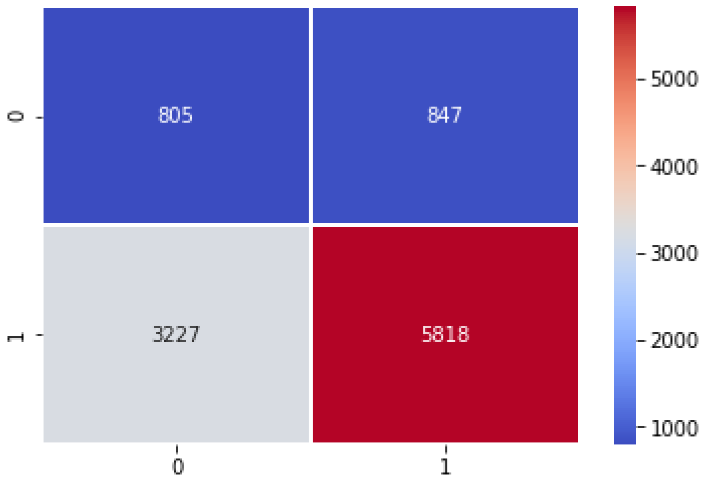
```
<matplotlib.axes._subplots.AxesSubplot at 0x23833bee208>
```



In [108]:
```
# https://seaborn.pydata.org/generated/seaborn.heatmap.html

# Test Confusion Matrix Heatmap

test_confusion_matrix = confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr))

print("Test Confusion Matrix")
sns.heatmap(test_confusion_matrix,annot=True,linewidth = 0.5, cmap='coolwarm', fmt='g')
```

```
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.836
Test Confusion Matrix
```

Out[108]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x23833f23198>
```



2.4.1.1 Top 10 important features of positive class from SET 1

In [128]:

```
# https://stackoverflow.com/questions/26976362/how-to-get-most-informat
ive-features-for-scikit-learn-classifier-for-different-c/26980472
# https://stackoverflow.com/questions/29867367/sklearn-multinomial-nb-m
ost-informative-features


def most_informative_feature(feature_names,classifier, val, n=10):
    class_labels = classifier.classes_
    feature_names =vectorizer.get_feature_names()
    topn_class = sorted(zip(classifier.feature_log_prob_[val], feature_
names), reverse = True)[:n]
    print(topn_class)
```

In [129]:

```
most_informative_feature(X_tr, neigh, 1)
```

```
[(-3.3645886423351214, 'talk'), (-4.0577348069668364, 'support'), (-4.7
508799556734154, 'urban'), (-4.7508799556734154, 'states'), (-4.7508799
556734154, 'st'), (-4.7508799556734154, 'science'), (-5.444023072539016
2, 'written'), (-5.4440230725390162, 'writing'), (-5.4440230725390162,
'world'), (-5.4440230725390162, 'working')]
```

2.4.1.2 Top 10 important features of negative class from SET 1

In [130]:
```
most_informative_feature(X_tr, neigh, 0)
```

```
[(-3.3643258233491462, 'talk'), (-4.0574674385058636, 'support'), (-4.7
506034883522759, 'urban'), (-4.7506034883522759, 'states'), (-4.7506034
883522759, 'st'), (-4.7506034883522759, 'science'), (-5.443728407856823
1, 'written'), (-5.4437284078568231, 'writing'), (-5.4437284078568231,
'world'), (-5.4437284078568231, 'working')]
```

2.4.2 Applying Naive Bayes on TFIDF, SET 2

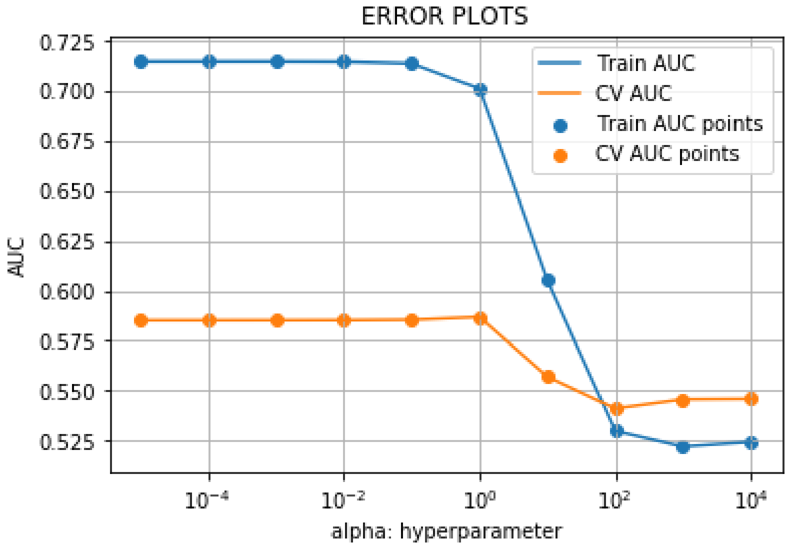
In [131]:
```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/40840
39
from scipy.sparse import hstack

# Train data stack
X_tr = hstack((X_Train_categories_one_hot,X_Train_sub_categories_one_ho
t,X_Train_school_state_one_hot,
               X_Train_teacher_prefix_one_hot,X_Train_grade_cat_one_hot
,X_Train_essay_tfidf,X_Train_titles_tfidf,
               X_train_price_norm)).tocsr()

# CV data Stack
X_cr = hstack((X_CV_categories_one_hot,X_CV_sub_categories_one_hot,X_CV
_school_state_one_hot,
               X_CV_teacher_prefix_one_hot,X_CV_grade_cat_one_hot,X_CV_
essay_tfidf,X_CV_titles_tfidf,
               X_cv_price_norm)).tocsr()

# Test Data Stack
X_te = hstack((X_Test_categories_one_hot,X_Test_sub_categories_one_hot,
X_Test_school_state_one_hot,
               X_Test_teacher_prefix_one_hot,X_Test_grade_cat_one_hot,X
_Test_essay_tfidf,X_Test_titles_tfidf,
               X_test_price_norm)).tocsr()

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

```
Final Data matrix
(14550, 1106) (14550,)
(7167, 1106) (7167,)
(10697, 1106) (10697,)
======================================================================
==============================
```

In [132]:

```
# Plotting error plot, AUC vs alpha plot to get best alpha

from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import roc_auc_score

train_auc = []
cv_auc = []

# Execute for different alpha values
alpha = [0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000]

for i in tqdm(alpha):

    neigh = MultinomialNB(alpha=i)
    neigh.fit(X_tr, y_train)

    y_train_pred = batch_predict(neigh, X_tr)
    y_cv_pred = batch_predict(neigh, X_cr)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probab
ility estimates of the positive class
    # not the predicted outputs
    train_auc.append(roc_auc_score(y_train, y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

plt.plot(alpha, train_auc, label='Train AUC')
plt.plot(alpha, cv_auc, label='CV AUC')

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

plt.xscale('log')
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```

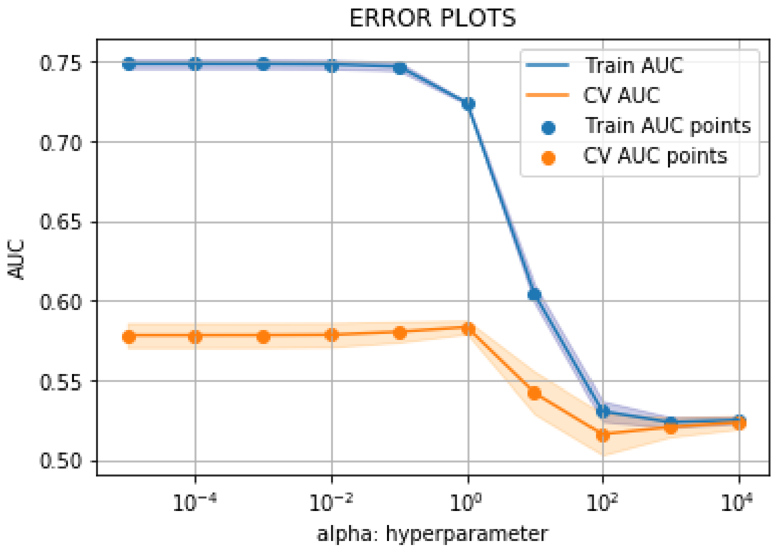
```
100%|██████████████████████████████████████████████████████████████
████████████████| 10/10 [00:01<00:00,  8.06it/s]
```



In [133]:

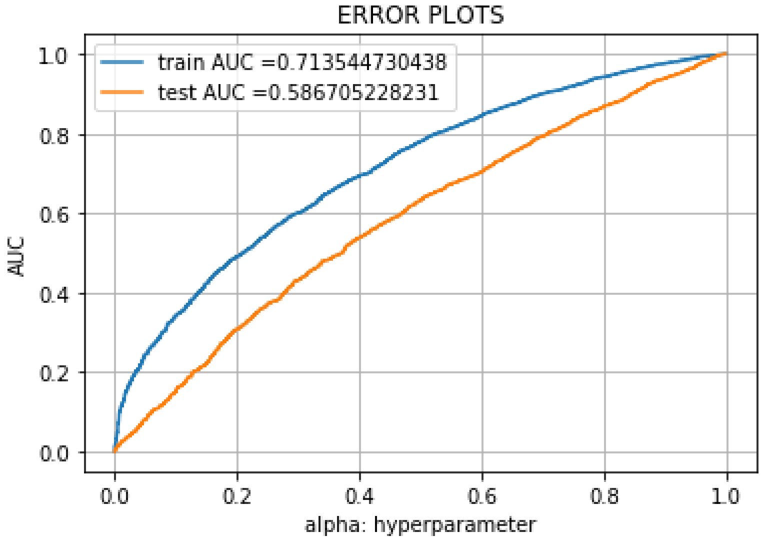
```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selec
tion.GridSearchCV.html
```

```
from sklearn.model_selection import GridSearchCV

neigh = MultinomialNB()
parameters = {'alpha':[0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000]}
clf = GridSearchCV(neigh, parameters, cv=3, 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']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']

plt.plot(parameters['alpha'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['alpha'],train_auc - train_auc_std,train_auc + train_auc_std,alpha=0.2,color='darkblue')

plt.plot(parameters['alpha'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['alpha'],cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='darkorange')

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

plt.xscale("log")
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



In [137]:

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


neigh = MultinomialNB(alpha=0.1)
neigh.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 = batch_predict(neigh, X_tr)
y_test_pred = batch_predict(neigh, 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, t
rain_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_
tpr)))
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



In [138]:

```
print("="*100)
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, tr
ain_fpr, train_fpr)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test
_fpr, test_fpr)))
```

```
========================================================================
============================
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.777
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.777
[[1123 1123]
 [2712 9592]]
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.84
[[ 842  810]
 [3421 5624]]
```

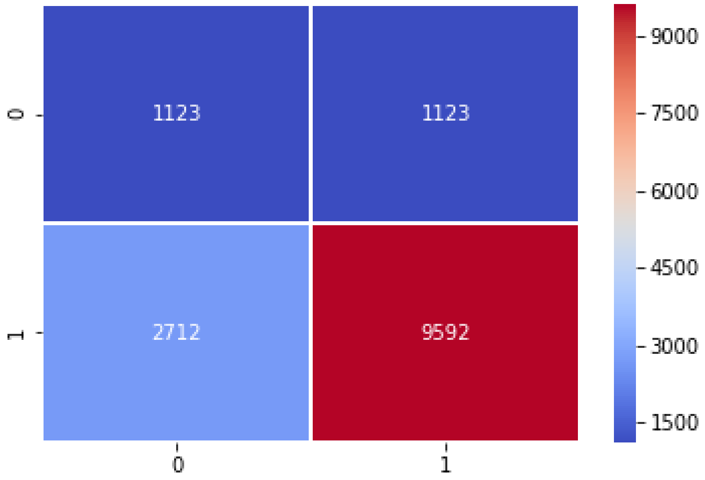
In [139]:

```
# https://seaborn.pydata.org/generated/seaborn.heatmap.html

# Train Confusion Matrix Heatmap
train_confusion_matrix = confusion_matrix(y_train, predict(y_train_pred
, tr_thresholds, train_fpr, train_fpr))

print("Train Confusion Matrix")
sns.heatmap(train_confusion_matrix,annot=True,linewidth = 0.5, cmap='co
olwarm', fmt='g')
```

```
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.777
Train Confusion Matrix
```

Out[139]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x23835c50ef0>
```

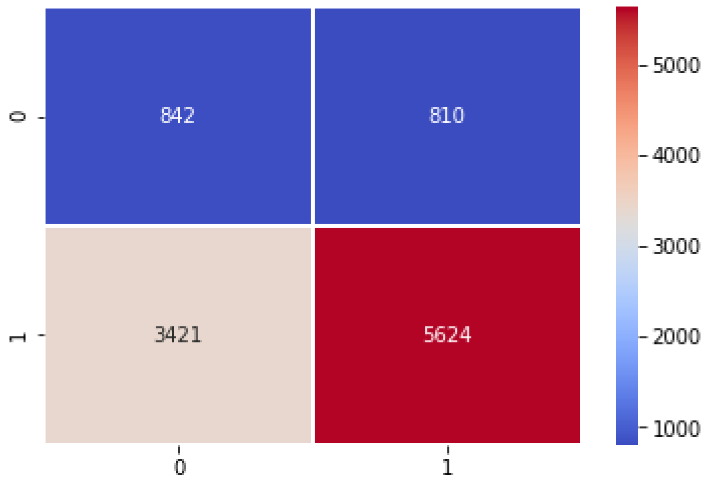


In [141]:

```
# https://seaborn.pydata.org/generated/seaborn.heatmap.html
# Test Confusion Matrix Heatmap

test_confusion_matrix = confusion_matrix(y_test, predict(y_test_pred, t
r_thresholds, test_fpr, test_fpr))

print("Test Confusion Matrix")
sns.heatmap(test_confusion_matrix,annot=True,linewidth = 0.5, cmap='coo
lwarm', fmt='g')
```

```
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.84
Test Confusion Matrix
```

Out[141]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x23833f92dd8>
```



2.4.2.1 Top 10 important features of positive class from SET 2

In [142]:
```
most_informative_feature(X_tr, neigh, 1)
```

```
[(-3.5538490010640977, 'questions'), (-3.6256801792598168, 'attend'),
(-3.891711544862968, 'asking'), (-3.9298189891016051, 'race'), (-4.0400
219001795481, 'talk'), (-4.0468373164315405, 'especially'), (-4.2684524
990924402, 'english'), (-4.5054667169393907, 'engineering'), (-4.733143
8669791206, 'support'), (-4.8546941460524886, 'fiction')]
```

2.4.2.2 Top 10 important features of negative class from SET 2

In [143]:

```
most_informative_feature(X_tr, neigh, 0)
```

```
[(-3.5773840055009209, 'questions'), (-3.7492724360904841, 'attend'),
(-3.8279259250838367, 'asking'), (-3.9181933539031624, 'race'), (-4.036
6418425026787, 'talk'), (-4.2344565036418791, 'especially'), (-4.249721
3862093846, 'english'), (-4.5413421571343058, 'engineering'), (-4.72965
09208480133, 'support'), (-4.8435422861731219, 'engaging')]
```

3. Conclusions

In [144]:

```
# http://zetcode.com/python/prettytable/

from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["Featurization", "Best alpha Value", "Train AUC", "Tes
t AUC"]

x.add_row(["Naive Bayes on BOW", 0.1, 0.71, 0.58])
x.add_row(["Naive Bayes on TFIDF", 0.1, 0.71, 0.58])

print(x)
```

```
+----------------------+------------------+-----------+----------+
|    Featurization     | Best alpha Value | Train AUC | Test AUC |
+----------------------+------------------+-----------+----------+
|  Naive Bayes on BOW  |       0.1        |    0.71   |   0.58   |
| Naive Bayes on TFIDF |       0.1        |    0.71   |   0.58   |
+----------------------+------------------+-----------+----------+
```

Observations

1. Very high or very low alpha value does not seems to be working fine on model.
2. Time taken by Naive Bayes model is extremely less as compared to other models.
3. Both set has alpha value works well in the range of 0 to 100.