# DonorsChoose

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

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

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

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

# About the DonorsChoose Data Set

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

| Feature | Description |
|---:|---:|
| **`project_id`** | A unique identifier for the proposed project. **Example:** `p036502` |
| **`project_title`** | Title of the project. **Examples:**<br>• `Art Will Make You Happy!`<br>• `First Grade Fun` |
| **`project_grade_category`** | Grade level of students for which the project is targeted. One of the following enumerated values:<br>• `Grades PreK-2`<br>• `Grades 3-5`<br>• `Grades 6-8`<br>• `Grades 9-12` |
| **`project_subject_categories`** | One or more (comma-separated) subject categories for the project from the following enumerated list of values:<br>• `Applied Learning`<br>• `Care & Hunger`<br>• `Health & Sports`<br>• `History & Civics`<br>• `Literacy & Language`<br>• `Math & Science`<br>• `Music & The Arts`<br>• `Special Needs`<br>• `Warmth`<br>**Examples:**<br>• `Music & The Arts`<br>• `Literacy & Language, Math & Science` |
| **`school_state`** | State where school is located (Two-letter U.S. postal code (https://en.wikipedia.org/wiki/List_of_U.S._state_abbreviations#Postal_codes)). **Example:** `WY` |
| **`project_subject_subcategories`** | One or more (comma-separated) subject subcategories for the project. **Examples:**<br>• `Literacy`<br>• `Literature & Writing, Social Sciences` |
| **`project_resource_summary`** | An explanation of the resources needed for the project. **Example:**<br>• `My students need hands on literacy materials to manage sensory needs!` |
| **`project_essay_1`** | First application essay* |
| **`project_essay_2`** | Second application essay* |
| **`project_essay_3`** | Third application essay* |
| **`project_essay_4`** | Fourth application essay* |
| **`project_submitted_datetime`** | Datetime when project application was submitted. **Example:** `2016-04-28 12:43:56.245` |

| Feature | Description |
|---|---|
| **`teacher_id`** | A unique identifier for the teacher of the proposed project. **Example:** `bdf8baa8fedef6bfeec7ae4ff1c15c56` |
| **`teacher_prefix`** | Teacher's title. One of the following enumerated values:<br>• `nan`<br>• `Dr.`<br>• `Mr.`<br>• `Mrs.`<br>• `Ms.`<br>• `Teacher.` |
| **`teacher_number_of_previously_posted_projects`** | Number of project applications previously submitted by the same teacher. **Example:** `2` |

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

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

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

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

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

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

## Notes on the Essay Data

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

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

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

- __project_essay_1:__ "Describe your students: What makes your students special? Specific details about their background, your neighborhood, and your school are all helpful."
- __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 [1]:

```
%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
```

```
C:\Users\wwang26\AppData\Local\Continuum\anaconda3\lib\site-packages\smart_open
\ssh.py:34: UserWarning: paramiko missing, opening SSH/SCP/SFTP paths will be di
sabled.  `pip install paramiko` to suppress
  warnings.warn('paramiko missing, opening SSH/SCP/SFTP paths will be disabled.
`pip install paramiko` to suppress')
C:\Users\wwang26\AppData\Local\Continuum\anaconda3\lib\site-packages\gensim\util
s.py:1197: UserWarning: detected Windows; aliasing chunkize to chunkize_serial
  warnings.warn("detected Windows; aliasing chunkize to chunkize_serial")
```

## 1.1 Reading Data

In [2]:

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

In [3]:
```
print("Number of data points in train data", project_data.shape)
print('-'*50)
print("The attributes of data :", project_data.columns.values)
```

```
Number of data points in train data (109248, 17)
--------------------------------------------------
The attributes of data : ['Unnamed: 0' 'id' 'teacher_id' 'teacher_prefix' 'schoo
l_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 [4]:
```
# how to replace elements in list python: https://stackoverflow.com/a/2582163/4084039
cols = ['Date' if x=='project_submitted_datetime' else x for x in list(project_data.columns)]


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


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


project_data.head(2)
```

Out[4]:

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

In [5]:

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

```
Number of data points in train data (1541272, 4)
['id' 'description' 'quantity' 'price']
```

Out[5]:

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

## 1.2 preprocessing of `project_subject_categories`

In [6]:

```
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 [7]:

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

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

sub_cat_list = []
for i in sub_catogories:
    temp = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
    for j in i.split(','): # it will split it in three parts ["Math & Science", "Warmth", "Care & 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 [8]:

```python
# 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 [9]:
```
project_data.head(2)
```

Out[9]:

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

In [10]:
```
#### 1.4.2.3 Using Pretrained Models: TFIDF weighted W2V
```

In [11]:

```
# 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])
print("="*50)
print(project_data['essay'].values[20000])
print("="*50)
print(project_data['essay'].values[99999])
print("="*50)
```

I have been fortunate enough to use the Fairy Tale STEM kits in my classroom as well as the STEM journals, which my students really enjoyed. I would love to implement more of the Lakeshore STEM kits in my classroom for the next school year as they provide excellent and engaging STEM lessons.My students come from a variety of backgrounds, including language and socioeconomic status. Many of them don't have a lot of experience in science and engineering and these kits give me the materials to provide these exciting opportunities for my students.Each month I try to do several science or STEM/STEAM projects. I would use the kits and robot to help guide my science instruction in engaging and meaningful ways. I can adapt the kits to my current language arts pacing guide where we already teach some of the material in the kits like tall tales (Paul Bunyan) or Johnny Appleseed. The following units will be taught in the next school year where I will implement these kits: magnets, motion, sink vs. float, robots. I often get to these units and don't know If I am teaching the right way or using the right materials. The kits will give me additional ideas, strategies, and lessons to prepare my students in science.It is challenging to develop high quality science activities. These kits give me the materials I need to provide my students with science activities that will go along with the curriculum in my classroom. Although I have some things (like magnets) in my classroom, I don't know how to use them effectively. The kits will provide me with the right amount of materials and show me how to use them in an appropriate way.

==================================================

I teach high school English to students with learning and behavioral disabilities. My students all vary in their ability level. However, the ultimate goal is to increase all students literacy levels. This includes their reading, writing, and communication levels.I teach a really dynamic group of students. However, my students face a lot of challenges. My students all live in poverty and in a dangerous neighborhood. Despite these challenges, I have students who have the the desire to defeat these challenges. My students all have learning disabilities and currently all are performing below grade level. My students are visual learners and will benefit from a classroom that fulfills their preferred learning style.The materials I am requesting will allow my students to be prepared for the classroom with the necessary supplies. Too often I am challenged with students who come to school unprepared for class due to economic challenges. I want my students to be able to focus on learning and not how they will be able to get school supplies. The supplies will last all year. Students will be able to complete written assignments and maintain a classroom journal. The chart paper will be used to make learning more visual in class and to create posters to aid students in their learning. The students have access to a classroom printer. The toner will be used to print student work that is completed on the classroom Chromebooks.I want to try and remove all barriers for the students learning and create opportunities for learning. One of the biggest barriers is the students not having the resources to get pens, paper, and folders. My students will be able to increase their literacy skills because of this project.

==================================================

\"Life moves pretty fast. If you don't stop and look around once in awhile, you could miss it.\" from the movie, Ferris Bueller's Day Off. Think back...what do you remember about your grandparents? How amazing would it be to be able to flip through a book to see a day in their lives?My second graders are voracious readers! They love to read both fiction and nonfiction books. Their favorite characters include Pete the Cat, Fly Guy, Piggie and Elephant, and Mercy Watson. They also love to read about insects, space and plants. My students are hungry bookworms! My students are eager to learn and read about the world around them. My kids love to be at school and are like little sponges absorbing everything around them. Their parents work long hours and usually do not see their children. My students are usually cared for by their grandparents or a family friend. Most of my students do not have someone who speaks English at home. Thus it is difficult for my students to acquire language.Now think forward... wouldn't it mean a lot to your kids, nieces or nephews or grandchildren, to be able to see a day in you r life today 30 years from now? Memories are so precious to us and being able to share these memories with future generations will be a rewarding experience.  As part of our social studies curriculum, students will be learning about changes over time.  Students will be studying photos to learn about how their community has changed over time.  In particular, we will look at photos to study how the land, buildings, clothing, and schools have changed over time.  As a culminating activity, my students will capture a slice of their history and preserve it through scrap booking. Key important events in their young lives will be documented with the date, location, and names.   Students will be using photos from home and from school to create their second grade memories.   Their scrap books will preserve their unique stories for future generations to enjoy.Your donation to this project will provide my second graders with an opportunity to learn about social studies in a fun and creative manner.  Through their scrapbooks, children will share their story with others and have a historical document for the rest of their lives.

==================================================

\"A person's a person, no matter how small.\" (Dr.Seuss) I teach the smallest students with the biggest enthusiasm for learning. My students learn in many different ways using all of our senses and multiple intelligences. I use a wide range of techniques to help all my students succeed. \r\nStudents in my class come from a variety of different backgrounds which makes for wonderful sharing of experiences and cultures, including Native Americans.\r\nOur school is a caring community of successful learners which can be seen through collaborative student project based learning in and out of the classroom. Kindergarteners in my class love to work with hands-on materials and have many different opportunities to practice a skill before it is mastered. Having the social skills to work cooperatively with friends is a crucial aspect of the kindergarten curriculum.Montana is the perfect place to learn about agriculture and nutrition. My students love to role play in our pretend kitchen in the early childhood classroom. I have had several kids ask me, \"Can we try cooking with REAL food?\" I will take their idea and create \"Common Core Cooking Lessons\" where we learn important math and writing concepts while cooking delicious healthy food for snack time. My students will have a grounded appreciation for the work that went into making the food and knowledge of where the ingredients came from as well as how it's healthy for their bodies. This project would expand our learning of nutrition and agricultural cooking recipes by having us peel our own apples to make homemade applesauce, make our own bread, and mix up healthy plants from our classroom garden in the spring. We will also create our own cookbooks to be printed and shared with families. \r\nStudents will gain math and literature skills as well as a life long enjoyment for healthy cooking.nannan

==================================================

My classroom consists of twenty-two amazing sixth graders from different cultures and backgrounds. They are a social bunch who enjoy working in partners and working with groups. They are hard-working and eager to head to middle school next year. My job is to get them ready to make this transition and make it as smooth as possible. In order to do this, my students need to come to school every day and feel safe and ready to learn. Because they are getting ready to head to middle school, I give them lots of choice- choice on where to sit and work, the order to complete assignments, choice of projects, etc. Part of the students feeling safe is the ability for them to come into a welcoming, encouraging environment. My room is colorful and the atmosphere is casual. I want them to take ownership of the classroom because we ALL share it together. Because my time with them is limited, I want to ensure they get the most of this time and enjoy it to the best of their abilities.Currently, we have twenty-two desks of differing sizes, yet the desks are similar to the ones the students will use in middle school. We also have a kidney table with crates for seating. I allow my students to choose their own spots while they are working independently or in groups. More often than not, most of them move out of their desks and onto the crates. Believe it or not, this has proven to be more successful than making them stay at their desks! It i

```
s because of this that I am looking toward the “Flexible Seating” option for my
classroom.\r\n The students look forward to their work time so they can move aro
und the room. I would like to get rid of the constricting desks and move toward
more “fun” seating options. I am requesting various seating so my students have
more options to sit. Currently, I have a stool and a papasan chair I inherited f
rom the previous sixth-grade teacher as well as five milk crate seats I made, bu
t I would like to give them more options and reduce the competition for the “goo
d seats”. I am also requesting two rugs as not only more seating options but to
make the classroom more welcoming and appealing. In order for my students to be
able to write and complete work without desks, I am requesting a class set of cl
ipboards. Finally, due to curriculum that requires groups to work together, I am
requesting tables that we can fold up when we are not using them to leave more r
oom for our flexible seating options.\r\nI know that with more seating options,
they will be that much more excited about coming to school! Thank you for your s
upport in making my classroom one students will remember forever!nannan
==================================================
```

In [12]:

```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 [13]:
```
sent = decontracted(project_data['essay'].values[20000])
print(sent)
print("="*50)
```

```
\"A person is a person, no matter how small.\" (Dr.Seuss) I teach the smallest s
tudents with the biggest enthusiasm for learning. My students learn in many diff
erent ways using all of our senses and multiple intelligences. I use a wide rang
e of techniques to help all my students succeed. \r\nStudents in my class come f
rom a variety of different backgrounds which makes for wonderful sharing of expe
riences and cultures, including Native Americans.\r\nOur school is a caring comm
unity of successful learners which can be seen through collaborative student pro
ject based learning in and out of the classroom. Kindergarteners in my class lov
e to work with hands-on materials and have many different opportunities to pract
ice a skill before it is mastered. Having the social skills to work cooperativel
y with friends is a crucial aspect of the kindergarten curriculum.Montana is the
perfect place to learn about agriculture and nutrition. My students love to role
play in our pretend kitchen in the early childhood classroom. I have had several
kids ask me, \"Can we try cooking with REAL food?\" I will take their idea and c
reate \"Common Core Cooking Lessons\" where we learn important math and writing 
concepts while cooking delicious healthy food for snack time. My students will h
ave a grounded appreciation for the work that went into making the food and know
ledge of where the ingredients came from as well as how it is healthy for their 
bodies. This project would expand our learning of nutrition and agricultural coo
king recipes by having us peel our own apples to make homemade applesauce, make 
our own bread, and mix up healthy plants from our classroom garden in the sprin
g. We will also create our own cookbooks to be printed and shared with families.
\r\nStudents will gain math and literature skills as well as a life long enjoyme
nt for healthy cooking.nannan
==================================================
```

In [14]:

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

```
 A person is a person, no matter how small.  (Dr.Seuss) I teach the smallest stu
dents with the biggest enthusiasm for learning. My students learn in many differ
ent ways using all of our senses and multiple intelligences. I use a wide range
of techniques to help all my students succeed.   Students in my class come from
a variety of different backgrounds which makes for wonderful sharing of experien
ces and cultures, including Native Americans.  Our school is a caring community
of successful learners which can be seen through collaborative student project b
ased learning in and out of the classroom. Kindergarteners in my class love to w
ork with hands-on materials and have many different opportunities to practice a
skill before it is mastered. Having the social skills to work cooperatively with
friends is a crucial aspect of the kindergarten curriculum.Montana is the perfec
t place to learn about agriculture and nutrition. My students love to role play
in our pretend kitchen in the early childhood classroom. I have had several kids
ask me,  Can we try cooking with REAL food?  I will take their idea and create
Common Core Cooking Lessons  where we learn important math and writing concepts
while cooking delicious healthy food for snack time. My students will have a gro
unded appreciation for the work that went into making the food and knowledge of
where the ingredients came from as well as how it is healthy for their bodies. T
his project would expand our learning of nutrition and agricultural cooking reci
pes by having us peel our own apples to make homemade applesauce, make our own b
read, and mix up healthy plants from our classroom garden in the spring. We will
also create our own cookbooks to be printed and shared with families.   Students
will gain math and literature skills as well as a life long enjoyment for health
y cooking.nannan
```

In [15]:

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

```
 A person is a person no matter how small Dr Seuss I teach the smallest students
with the biggest enthusiasm for learning My students learn in many different way
s using all of our senses and multiple intelligences I use a wide range of techn
iques to help all my students succeed Students in my class come from a variety o
f different backgrounds which makes for wonderful sharing of experiences and cul
tures including Native Americans Our school is a caring community of successful
learners which can be seen through collaborative student project based learning
in and out of the classroom Kindergarteners in my class love to work with hands
on materials and have many different opportunities to practice a skill before it
is mastered Having the social skills to work cooperatively with friends is a cru
cial aspect of the kindergarten curriculum Montana is the perfect place to learn
about agriculture and nutrition My students love to role play in our pretend kit
chen in the early childhood classroom I have had several kids ask me Can we try
cooking with REAL food I will take their idea and create Common Core Cooking Les
sons where we learn important math and writing concepts while cooking delicious
healthy food for snack time My students will have a grounded appreciation for th
e work that went into making the food and knowledge of where the ingredients cam
e from as well as how it is healthy for their bodies This project would expand o
ur learning of nutrition and agricultural cooking recipes by having us peel our
own apples to make homemade applesauce make our own bread and mix up healthy pla
nts from our classroom garden in the spring We will also create our own cookbook
s to be printed and shared with families Students will gain math and literature
skills as well as a life long enjoyment for healthy cooking nannan
```

In [16]:

```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
stopwords= ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you',
"you're", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he',
'him', 'his', 'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itsel
f', 'they', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'tha
t', "that'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'ha
s', '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', 'th
rough', 'during', 'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'of
f', 'over', 'under', 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all'
, 'any', 'both', 'each', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than',
'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should'v
e", 'now', 'd', 'll', 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "di
dn't", 'doesn', "doesn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma',
'mightn', "mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "should
n't", 'wasn', "wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"]
```

In [17]:

```
# 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.lower() not in stopwords)
    preprocessed_essays.append(sent.lower().strip())
```

```
100%|██████████████████████████████████████████████████████████████████████████|
109248/109248 [02:19<00:00, 783.51it/s]
```

In [121]:

```
# after preprocesing

preprocessed_essays = pd.DataFrame({'preprocessed_essays': preprocessed_essays})
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-121-bb3cb8b52168> in <module>
      1 # after preprocesing
      2 
----> 3 preprocessed_essays = pd.DataFrame({'preprocessed_essays': preprocessed_
essays})
      4 

~\AppData\Local\Continuum\anaconda3\lib\site-packages\pandas\core\frame.py in __
init__(self, data, index, columns, dtype, copy)
    390                                  dtype=dtype, copy=copy)
    391         elif isinstance(data, dict):
--> 392             mgr = init_dict(data, index, columns, dtype=dtype)
    393         elif isinstance(data, ma.MaskedArray):
    394             import numpy.ma.mrecords as mrecords

~\AppData\Local\Continuum\anaconda3\lib\site-packages\pandas\core\internals\cons
truction.py in init_dict(data, index, columns, dtype)
    210         arrays = [data[k] for k in keys]
    211 
--> 212     return arrays_to_mgr(arrays, data_names, index, columns, dtype=dtype
)
    213 
    214 

~\AppData\Local\Continuum\anaconda3\lib\site-packages\pandas\core\internals\cons
truction.py in arrays_to_mgr(arrays, arr_names, index, columns, dtype)
     49     # figure out the index, if necessary
     50     if index is None:
---> 51         index = extract_index(arrays)
     52     else:
     53         index = ensure_index(index)

~\AppData\Local\Continuum\anaconda3\lib\site-packages\pandas\core\internals\cons
truction.py in extract_index(data)
    306 
    307         if not indexes and not raw_lengths:
--> 308             raise ValueError('If using all scalar values, you must pass'
    309                              ' an index')
    310 

ValueError: If using all scalar values, you must pass an index
```

In [19]:

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

project_data = pd.concat([project_data, preprocessed_essays], axis=1)
project_data.head(1)
```

Out[19]:

| | Unnamed: 0 | id | teacher_id | teacher_prefix | school_state | Date | project_grac |
|---|---|---|---|---|---|---|---|
| **0** | 160221 | p253737 | c90749f5d961ff158d4b4d1e7dc665fc | Mrs. | IN | 2016-12-05 13:43:57 | Gr |

## 1.4 Preprocessing of `project_title`

In [20]:

```
# similarly you can preprocess the titles also


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

```
100%|████████████████████████████████████████████████████████████| 10
9248/109248 [00:06<00:00, 15662.69it/s]
```

In [21]:

```
preprocessed_titles = pd.DataFrame({'preprocessed_titles': preprocessed_titles})


project_data = pd.concat([project_data, preprocessed_titles], axis=1)
project_data.head(1)
```

Out[21]:

| | Unnamed: 0 | id | teacher_id | teacher_prefix | school_state | Date | project_grac |
|---|---|---|---|---|---|---|---|
| **0** | 160221 | p253737 | c90749f5d961ff158d4b4d1e7dc665fc | Mrs. | IN | 2016-12-05 13:43:57 | Gr |

## 1.5 Preparing data for models

In [22]:
```
project_data.columns
```

Out[22]:
```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'Date', 'project_grade_category', 'project_title', 'project_essay_1',
       'project_essay_2', 'project_essay_3', 'project_essay_4',
       'project_resource_summary',
       'teacher_number_of_previously_posted_projects', 'project_is_approved',
       'clean_categories', 'clean_subcategories', 'essay',
       'preprocessed_essays', 'preprocessed_titles'],
      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 Merge with resource data

In [23]:
```
price_data = resource_data.groupby('id').agg({'price':'sum', 'quantity':'sum'}).r
eset_index()
project_data = pd.merge(project_data, price_data, on='id', how='left')
```

# 2. NB Model Preparation

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

In [24]:

```
#Stratify vs ramdom sampling. oversampling for imbalanced data
#https://stats.stackexchange.com/questions/250273/benefits-of-stratified-vs-random-sampling-for-generating-training-data-in-classi




from sklearn.model_selection import train_test_split

# train = project_data.drop(['project_is_approved'], axis=1, inplace=True)  # this will drop in raw data so would not work

X_train, X_test, y_train, y_test = train_test_split(project_data, project_data['project_is_approved'],
                                                    test_size=0.3, stratify = project_data['project_is_approved'])

X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.3, stratify=y_train)


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

In [25]:

```
print(X_test.shape)
print(y_test.shape)
print(X_cv.shape)
print(y_cv.shape)
```

```
(32775, 21)
(32775,)
(22942, 21)
(22942,)
```

## 2.2 Make Data Model Ready: encoding numerical, categorical features

In [26]:

```
# Encoding of Categorical Features:

# Category:
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(vocabulary=list(sorted_cat_dict.keys()), lowercase=False, binary=True)
categories_one_hot_train = vectorizer.fit_transform(X_train['clean_categories'].values)
categories_one_hot_cv = vectorizer.transform(X_cv['clean_categories'].values)
categories_one_hot_test = vectorizer.transform(X_test['clean_categories'].values)

print(vectorizer.get_feature_names())
print("category Shape of matrix after one hot encodig ",categories_one_hot_train.shape)



# Subcategory
vectorizer = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()), lowercase=False, binary=True)
sub_categories_one_hot_train = vectorizer.fit_transform(X_train['clean_subcategories'].values)
sub_categories_one_hot_cv = vectorizer.transform(X_cv['clean_subcategories'].values)
sub_categories_one_hot_test = vectorizer.transform(X_test['clean_subcategories'].values)


print(vectorizer.get_feature_names())
print("subctg Shape of matrix after one hot encodig ",sub_categories_one_hot_train.shape)


#you can do the similar thing with state, teacher_prefix and project_grade_category also
# count of all the words in corpus python: https://stackoverflow.com/a/22898595/4084039
my_counter = Counter()
for word in project_data['school_state'].values:
    my_counter.update(word.split())


state_dict = dict(my_counter)
sorted_state_dict = dict(sorted(state_dict.items(), key=lambda kv: kv[1]))




vectorizer = CountVectorizer(vocabulary = list(sorted_state_dict.keys()), lowercase=False, binary=True)

state_one_hot_train = vectorizer.fit_transform(X_train['school_state'].values)
state_one_hot_cv = vectorizer.transform(X_cv['school_state'].values)
state_one_hot_test = vectorizer.transform(X_test['school_state'].values)


print("state Shape of matrix after one hot encodig ",state_one_hot_train.shape)
```

```
#teacher prefix

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

tp_dict = dict(my_counter)
sorted_tp_dict = dict(sorted(tp_dict.items(), key=lambda kv: kv[1]))
vectorizer = CountVectorizer(vocabulary = list(sorted_tp_dict.keys()),lowercase=False, binary=True)


tp_one_hot_train = vectorizer.fit_transform(X_train['teacher_prefix'].apply(lambda x: np.str_(x)))
tp_one_hot_cv = vectorizer.transform(X_cv['teacher_prefix'].apply(lambda x: np.str_(x)))
tp_one_hot_test = vectorizer.transform(X_test['teacher_prefix'].apply(lambda x: np.str_(x)))


print("tp Shape of matrix after one hot encodig ",tp_one_hot_train.shape)




# Project Grade List
from collections import Counter
my_counter = Counter()
for word in project_data['project_grade_category'].values:
    my_counter.update(word.splitlines())



grade_list = dict(my_counter)
print(grade_list)


# If not generating the above list and put into vocabulary, the vector will some messed up results ['12', 'Grades', 'PreK']
# This is because of space and new lines. Otherwise no need for vocabulary

vectorizer = CountVectorizer(vocabulary=list(grade_list.keys()),lowercase=False, binary=True)



pg_one_hot_train = vectorizer.fit_transform(X_train['project_grade_category'].values)
pg_one_hot_cv = vectorizer.transform(X_cv['project_grade_category'].values)
pg_one_hot_test = vectorizer.transform(X_test['project_grade_category'].values)


print("pg Shape of matrix after one hot encodig ",pg_one_hot_train.shape)
```

```
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'Sp
ecialNeeds', 'Health_Sports', 'Math_Science', 'Literacy_Language']
category Shape of matrix after one hot encodig  (53531, 9)
['Economics', 'CommunityService', 'FinancialLiteracy', 'ParentInvolvement', 'Ext
racurricular', 'Civics_Government', 'ForeignLanguages', 'NutritionEducation', 'W
armth', 'Care_Hunger', 'SocialSciences', 'PerformingArts', 'CharacterEducation',
'TeamSports', 'Other', 'College_CareerPrep', 'Music', 'History_Geography', 'Heal
th_LifeScience', 'EarlyDevelopment', 'ESL', 'Gym_Fitness', 'EnvironmentalScienc
e', 'VisualArts', 'Health_Wellness', 'AppliedSciences', 'SpecialNeeds', 'Literat
ure_Writing', 'Mathematics', 'Literacy']
subctg Shape of matrix after one hot encodig  (53531, 30)
state Shape of matrix after one hot encodig  (53531, 51)
tp Shape of matrix after one hot encodig  (53531, 6)
{'Grades PreK-2': 44225, 'Grades 6-8': 16923, 'Grades 3-5': 37137, 'Grades 9-1
2': 10963}
pg Shape of matrix after one hot encodig  (53531, 4)
```

In [27]:

```
print(pg_one_hot_test.shape)
print(pg_one_hot_train.shape)

print(state_one_hot_test.shape)
print(state_one_hot_train.shape)


print(tp_one_hot_test.shape)
print(tp_one_hot_train.shape)
```

```
(32775, 4)
(53531, 4)
(32775, 51)
(53531, 51)
(32775, 6)
(53531, 6)
```

In [28]:

```
# Numerical Data


from sklearn.preprocessing import Normalizer

# 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)


#instead of standardize, try normalization since chi2 requires non-negative


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

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

price_standardized_cv = price_scalar.transform(X_cv['price'].values.reshape(-1, 1
))
price_standardized_test = price_scalar.transform(X_test['price'].values.reshape(-
1, 1))


print(price_standardized_train.mean())
print(price_standardized_train.std())

print(price_standardized_train[100])
```

```
1.0
0.0
[1.]
```

In [30]:

```
previous_scalar = Normalizer()
previous_scalar.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))

# finding the mean and standard deviation of this data
#print(f"Mean : {previous_scalar.mean_[0]}, Standard deviation : {np.sqrt(previous_scalar.var_[0])}")

# Now standardize the data with above maen and variance.
previous_standardized_train = previous_scalar.transform(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1, 1))



previous_standardized_cv = previous_scalar.transform(X_cv['teacher_number_of_previously_posted_projects'].values.reshape(-1, 1))


previous_standardized_test = previous_scalar.transform(X_test['teacher_number_of_previously_posted_projects'].values.reshape(-1, 1))


print(previous_standardized_train.mean())
print(previous_standardized_train.std())

print(previous_standardized_train[100])
```

```
0.7255982514804505
0.446212313735248
[1.]
```

In [31]:

```
print(price_standardized_test.shape)
print(price_standardized_train.shape)


print(previous_standardized_test.shape)
print(previous_standardized_train.shape)
```

```
(32775, 1)
(53531, 1)
(32775, 1)
(53531, 1)
```

## 2.3 Make Data Model Ready: encoding essay, and project_title

#### 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_bow = CountVectorizer(min_df=10,max_features = 5000)
text_train_bow = vectorizer_bow.fit_transform(X_train['preprocessed_essays'].values)


# should fit_transferm only on train data . Transform on test data
text_cv_bow = vectorizer_bow.transform(X_cv['preprocessed_essays'].values)
text_test_bow = vectorizer_bow.transform(X_test['preprocessed_essays'].values)

print("Shape of matrix after one hot encodig ",text_train_bow.shape)
print("Shape of matrix after one hot encodig ",text_test_bow.shape)
print("Shape of matrix after one hot encodig ",text_cv_bow.shape)
print(text_train_bow[1])
```

```
Shape of matrix after one hot encodig  (53531, 5000)
Shape of matrix after one hot encodig  (32775, 5000)
Shape of matrix after one hot encodig  (22942, 5000)
  (0, 3008)	1
  (0, 2437)	1
  (0, 1704)	1
  (0, 3947)	1
  (0, 939)	1
  (0, 218)	1
  (0, 163)	1
  (0, 3240)	1
  (0, 2988)	1
  (0, 2823)	1
  (0, 3656)	1
  (0, 4707)	1
  (0, 3651)	1
  (0, 3910)	1
  (0, 3603)	1
  (0, 2197)	1
  (0, 2299)	1
  (0, 839)	1
  (0, 690)	3
  (0, 1357)	3
  (0, 1135)	1
  (0, 1818)	1
  (0, 4349)	1
  (0, 4869)	1
  (0, 3317)	2
  :	:
  (0, 902)	2
  (0, 2045)	1
  (0, 4532)	1
  (0, 4570)	1
  (0, 268)	1
  (0, 2262)	1
  (0, 4222)	1
  (0, 1766)	2
  (0, 3864)	1
  (0, 4266)	1
  (0, 2676)	1
  (0, 1163)	2
  (0, 3561)	2
  (0, 2185)	1
  (0, 4879)	1
  (0, 3084)	2
  (0, 248)	1
  (0, 236)	1
  (0, 4963)	3
  (0, 4489)	1
  (0, 4937)	3
  (0, 832)	1
  (0, 1147)	3
  (0, 4355)	5
  (0, 825)	1
```

In [33]:

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


vectorizer_bow_title = CountVectorizer(min_df=10,max_features = 5000)
title_train_bow = vectorizer_bow_title.fit_transform(X_train['preprocessed_titles'].values)
title_cv_bow = vectorizer_bow_title.transform(X_cv['preprocessed_titles'].values)
title_test_bow = vectorizer_bow_title.transform(X_test['preprocessed_titles'].values)

print("Shape of matrix after one hot encodig ",title_train_bow.shape)
print("Shape of matrix after one hot encodig ",title_cv_bow.shape)
print("Shape of matrix after one hot encodig ",title_test_bow.shape)
```

```
Shape of matrix after one hot encodig  (53531, 2218)
Shape of matrix after one hot encodig  (22942, 2218)
Shape of matrix after one hot encodig  (32775, 2218)
```

### 1.5.2.2 TFIDF vectorizer

In [34]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer_tfidf = TfidfVectorizer(min_df=10, max_features = 5000)
text_train_tfidf = vectorizer_tfidf.fit_transform(X_train['preprocessed_essays'].values)
text_cv_tfidf = vectorizer_tfidf.transform(X_cv['preprocessed_essays'].values)
text_test_tfidf = vectorizer_tfidf.transform(X_test['preprocessed_essays'].values)

print("Shape of matrix after one hot encodig ",text_train_tfidf.shape)
print("Shape of matrix after one hot encodig ",text_cv_tfidf.shape)
print("Shape of matrix after one hot encodig ",text_test_tfidf.shape)
```

```
Shape of matrix after one hot encodig  (53531, 5000)
Shape of matrix after one hot encodig  (22942, 5000)
Shape of matrix after one hot encodig  (32775, 5000)
```

In [35]:

```
# Similarly you can vectorize for title also
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer_tfidf_title = TfidfVectorizer(min_df=10)
title_train_tfidf = vectorizer_tfidf_title.fit_transform(X_train['preprocessed_titles'].values)
title_cv_tfidf = vectorizer_tfidf_title.transform(X_cv['preprocessed_titles'].values)
title_test_tfidf = vectorizer_tfidf_title.transform(X_test['preprocessed_titles'].values)

print("Shape of matrix after one hot encodig ",title_train_tfidf.shape)
print("Shape of matrix after one hot encodig ",title_cv_tfidf.shape)
print("Shape of matrix after one hot encodig ",title_test_tfidf.shape)
```

```
Shape of matrix after one hot encodig  (53531, 2218)
Shape of matrix after one hot encodig  (22942, 2218)
Shape of matrix after one hot encodig  (32775, 2218)
```

### 1.5.2.3 Using Pretrained Models: Avg W2V

In [49]:

```
from gensim.models import Word2Vec
from gensim.models import KeyedVectors


#this line of code trains your w2v model on the give list of sentances
w2v_model=Word2Vec(X_train['preprocessed_essays'].values, min_count=5,size=50, wo
rkers=4)


glove_words = list(w2v_model.wv.vocab)
print("number of words that occured minimum 5 times ",len(glove_words))
print("sample words ", glove_words)
```

```
number of words that occured minimum 5 times  37
sample words  ['s', 't', 'u', 'd', 'e', 'n', ' ', 'c', 'o', 'm', 'h', 'i', 'g',
'p', 'v', 'r', 'y', 'a', 'l', 'w', 'k', 'x', 'b', 'f', 'q', 'z', 'j', '1', '2',
'6', '8', '7', '9', '0', '5', '3', '4']
```

In [48]:

```
print(X_train['preprocessed_titles'].values[1])
```

```
group work made easy
```

In [57]:

```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_vectors_train = []; # the avg-w2v for each sentence/review is stored in t
his list
for sentence in tqdm(X_train['preprocessed_essays'].values): # for each review/se
ntence
    vector = np.zeros(50) # 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 += w2v_model.wv[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_train.append(vector)

print(len(avg_w2v_vectors_train))
```

```
100%|██████████████████████████████████████████████████|
53531/53531 [00:08<00:00, 5970.19it/s]

53531
```

In [58]:

```python
#this line of code trains your w2v model on the give list of sentances
w2v_model=Word2Vec(X_cv['preprocessed_essays'].values,min_count=5,size=50, workers=4)


glove_words = list(w2v_model.wv.vocab)
print("number of words that occured minimum 5 times ",len(glove_words))
print("sample words ", glove_words[0:50])
```

```
number of words that occured minimum 5 times  37
sample words  ['s', 't', 'u', 'd', 'e', 'n', ' ', 'a', 'c', 'i', 'v', 'r', 'l',
'f', 'g', 'o', 'm', 'y', 'w', 'h', 'p', 'b', 'k', 'x', '5', 'q', 'j', 'z', '1',
'6', '0', '8', '2', '3', '4', '7', '9']
```

In [59]:

```python
avg_w2v_vectors_cv = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['preprocessed_essays'].values): # for each review/sentence
    vector = np.zeros(50) # 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 += w2v_model.wv[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_cv.append(vector)

print(len(avg_w2v_vectors_cv))
```

```
100%|██████████████████████████████████████████████████|
22942/22942 [00:03<00:00, 6861.22it/s]

22942
```

In [60]:

```python
w2v_model=Word2Vec(X_test['preprocessed_essays'].values,min_count=5,size=50, workers=4)


glove_words = list(w2v_model.wv.vocab)
print("number of words that occured minimum 5 times ",len(glove_words))
print("sample words ", glove_words[0:50])
```

```
number of words that occured minimum 5 times  37
sample words  ['t', 'e', 'a', 'c', 'h', 'i', 'n', 'g', ' ', 'r', 'd', '1', '6',
'y', 's', 'f', 'o', 'u', 'x', 'l', 'k', 'w', '8', 'q', 'v', 'm', 'b', 'p', 'j',
'5', '3', '0', 'z', '2', '7', '4', '9']
```

```
In [61]: avg_w2v_vectors_test = []; # the avg-w2v for each sentence/review is stored in th
         is list
         for sentence in tqdm(X_test['preprocessed_essays'].values): # for each review/sen
         tence
             vector = np.zeros(50) # 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 += w2v_model.wv[word]
                     cnt_words += 1
             if cnt_words != 0:
                 vector /= cnt_words
             avg_w2v_vectors_test.append(vector)

         print(len(avg_w2v_vectors_test))
```

```
100%|████████████████████████████████████████████████████████████|
32775/32775 [00:08<00:00, 3859.05it/s]

32775
```

```
In [62]: # Similarly you can vectorize for title also

         w2v_model=Word2Vec(X_train['preprocessed_titles'].values,min_count=5,size=50, wor
         kers=4)


         glove_words = list(w2v_model.wv.vocab)
         print("number of words that occured minimum 5 times ",len(glove_words))
         print("sample words ", glove_words[0:50])

         avg_w2v_vectors_titles_train = []; # the avg-w2v for each sentence/review is stor
         ed in this list
         for sentence in tqdm(X_train['preprocessed_titles'].values): # for each review/se
         ntence
             vector = np.zeros(50) # 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 += w2v_model.wv[word]
                     cnt_words += 1
             if cnt_words != 0:
                 vector /= cnt_words
             avg_w2v_vectors_titles_train.append(vector)

         print(len(avg_w2v_vectors_titles_train))
```

```
number of words that occured minimum 5 times  37
sample words  ['p', 'l', 'a', 'y', ' ', 'w', 'i', 't', 'h', 'd', 'o', 'u', 's',
'e', 'g', 'r', 'k', 'm', 'n', 'c', 'j', 'f', 'q', 'b', 'v', '2', 'x', '3', '1',
'z', '6', '0', '4', '5', '8', '7', '9']

100%|███████████████████████████████████████████████████████████|
53531/53531 [00:00<00:00, 99791.23it/s]

53531
```

In [88]:

```
# Similarly you can vectorize for title also

w2v_model=Word2Vec(X_cv['preprocessed_titles'],min_count=5,size=50, workers=4)


glove_words = list(w2v_model.wv.vocab)
print("number of words that occured minimum 5 times ",len(glove_words))
print("sample words ", glove_words[0:50])

avg_w2v_vectors_titles_cv = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['preprocessed_titles']): # for each review/sentence
    vector = np.zeros(50) # 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 += w2v_model.wv[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_titles_cv.append(vector)

print(len(avg_w2v_vectors_titles_cv))
```

```
number of words that occured minimum 5 times  37
sample words  ['w', 'e', ' ', 'g', 'o', 't', 's', 'p', 'i', 'r', 'y', 'd', 'm',
'n', 'c', 'h', 'l', 'u', 'a', 'k', 'f', 'b', 'v', 'q', '4', 'x', 'z', '2', '1',
'j', '3', '5', '0', '6', '7', '9', '8']

100%|████████████████████████████████████████████████████████████|
22942/22942 [00:00<00:00, 74652.32it/s]

22942
```

In [63]:

```
# Similarly you can vectorize for title also

w2v_model=Word2Vec(X_test['preprocessed_titles'].values,min_count=5,size=50, workers=4)


glove_words = list(w2v_model.wv.vocab)
print("number of words that occured minimum 5 times ",len(glove_words))
print("sample words ", glove_words[0:50])

avg_w2v_vectors_titles_test = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['preprocessed_titles'].values): # for each review/sentence
    vector = np.zeros(50) # 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 += w2v_model.wv[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_titles_test.append(vector)

print(len(avg_w2v_vectors_titles_test))
```

```
number of words that occured minimum 5 times  37
sample words  ['w', 'e', ' ', 'l', 'o', 'v', 'a', 'r', 'n', 't', 'h', 'u', 'g',
's', 'c', 'i', 'd', 'y', 'm', 'k', 'b', 'p', 'f', 'j', 'z', 'x', 'q', '2', '0',
'1', '6', '3', '4', '5', '7', '8', '9']

100%|██████████████████████████████████████████████████|
32775/32775 [00:00<00:00, 65769.31it/s]

32775
```

### 1.5.2.3 Using Pretrained Models: TFIDF weighted W2V

In [64]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
w2v_model=Word2Vec(X_train['preprocessed_essays'].values,min_count=5,size=50, workers=4)
glove_words = list(w2v_model.wv.vocab)


tfidf_model_train= TfidfVectorizer()
tfidf_model_train.fit(X_train['preprocessed_essays'].values)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model_train.get_feature_names(), list(tfidf_model_train.idf_)))
tfidf_words_train = set(tfidf_model_train.get_feature_names())
```

In [65]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_train = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['preprocessed_essays'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words_train):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the 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_train.append(vector)

print(len(tfidf_w2v_vectors_train))
```

```
100%|██████████████████████████████████████████████████████████████████████|
53531/53531 [00:11<00:00, 4843.74it/s]

53531
```

In [67]:

```
w2v_model=Word2Vec(X_cv['preprocessed_essays'].values,min_count=5,size=50, worker
s=4)
glove_words = list(w2v_model.wv.vocab)


tfidf_model_cv= TfidfVectorizer()
tfidf_model_cv.fit(X_cv['preprocessed_essays'].values)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model_cv.get_feature_names(), list(tfidf_model_cv.idf
_)))
tfidf_words_cv = set(tfidf_model_cv.get_feature_names())



tfidf_w2v_vectors_cv = []; # the avg-w2v for each sentence/review is stored in th
is list

for sentence in tqdm(X_cv['preprocessed_essays'].values): # 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 sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words_cv):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf valu
e((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.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_cv.append(vector)

print(len(tfidf_w2v_vectors_cv))
```

```
100%|██████████████████████████████████████████████████████████████████████████|
22942/22942 [00:04<00:00, 4992.21it/s]

22942
```

In [68]:

```python
w2v_model=Word2Vec(X_test['preprocessed_essays'].values,min_count=5,size=50, workers=4)
glove_words = list(w2v_model.wv.vocab)


tfidf_model_test= TfidfVectorizer()
tfidf_model_test.fit(X_test['preprocessed_essays'].values)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model_test.get_feature_names(), list(tfidf_model_test.idf_)))
tfidf_words_test = set(tfidf_model_test.get_feature_names())


tfidf_w2v_vectors_test = []; # the avg-w2v for each sentence/review is stored in this list


for sentence in tqdm(X_test['preprocessed_essays'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words_test):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the 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_test.append(vector)

print(len(tfidf_w2v_vectors_test))
```

```
100%|██████████████████████████████████████████████████|
32775/32775 [00:04<00:00, 6742.75it/s]

32775
```

In [77]:

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

In [69]:

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

w2v_model=Word2Vec(X_train['preprocessed_titles'].values,min_count=5,size=50, workers=4)
glove_words = list(w2v_model.wv.vocab)




tfidf_model_title_train= TfidfVectorizer()
tfidf_model_title_train.fit(X_train['preprocessed_titles'].values)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model_title_train.get_feature_names(), list(tfidf_model_title_train.idf_)))
tfidf_words_title_train = set(tfidf_model_title_train.get_feature_names())




tfidf_w2v_vectors_title_train = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['preprocessed_titles'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words_title_train):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the 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_title_train.append(vector)

print(len(tfidf_w2v_vectors_title_train))
print(len(tfidf_w2v_vectors_title_train[0]))
```

```
100%|██████████████████████████████████████████████████| 53531/53531 [00:00<00:00, 107307.45it/s]

53531
300
```

In [99]:

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

w2v_model=Word2Vec(X_cv['preprocessed_titles'],min_count=5,size=50, workers=4)
glove_words = list(w2v_model.wv.vocab)




tfidf_model_title_cv= TfidfVectorizer()
tfidf_model_title_cv.fit(X_cv['preprocessed_titles'])
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model_title_cv.get_feature_names(), list(tfidf_model_
title_cv.idf_)))
tfidf_words_title_cv = set(tfidf_model_title_cv.get_feature_names())




tfidf_w2v_vectors_title_cv = []; # the avg-w2v for each sentence/review is stored
in this list
for sentence in tqdm(X_cv['preprocessed_titles']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words_title_cv):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf valu
e((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.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_title_cv.append(vector)

print(len(tfidf_w2v_vectors_title_cv))
print(len(tfidf_w2v_vectors_title_cv[0]))
```

```
100%|████████████████████████████████████████████████████████████|
22942/22942 [00:00<00:00, 81428.81it/s]

22942
300
```

In [70]:

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

w2v_model=Word2Vec(X_test['preprocessed_titles'].values,min_count=5,size=50, workers=4)
glove_words = list(w2v_model.wv.vocab)




tfidf_model_title_test= TfidfVectorizer()
tfidf_model_title_test.fit(X_test['preprocessed_titles'].values)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model_title_test.get_feature_names(), list(tfidf_model_title_test.idf_)))
tfidf_words_title_test = set(tfidf_model_title_test.get_feature_names())




tfidf_w2v_vectors_title_test = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['preprocessed_titles'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words_title_test):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the 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_title_test.append(vector)

print(len(tfidf_w2v_vectors_title_test))
print(len(tfidf_w2v_vectors_title_test[0]))
```

```
100%|████████████████████████████████████████████████████████████|
32775/32775 [00:00<00:00, 62203.22it/s]

32775
300
```

## 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

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 (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/receiver-operating-characteristic-curve-roc-curve-and-auc-1/) 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 (https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html) 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 (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/confusion-matrix-tpr-fpr-fnr-tnr-1/) with predicted and original labels of test data points. Please visualize your confusion matrices using seaborn heatmaps.
   - (https://seaborn.pydata.org/generated/seaborn.heatmap.html)

   (https://seaborn.pydata.org/generated/seaborn.heatmap.html)

   (https://seaborn.pydata.org/generated/seaborn.heatmap.html)

   (https://seaborn.pydata.org/generated/seaborn.heatmap.html)

5. **Conclusion** (https://seaborn.pydata.org/generated/seaborn.heatmap.html)

   (https://seaborn.pydata.org/generated/seaborn.heatmap.html)

   - 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 (https://seaborn.pydata.org/generated/seaborn.heatmap.html) link (http://zetcode.com/python/prettytable/)

In [71]:

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

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

    return y_data_pred

'''
```

### 2.4.1 Applying Naive Bayes on BOW, SET 1

In [36]:

```python
# Please write all the code with proper documentation


from scipy.sparse import hstack

X_train_bow = hstack((categories_one_hot_train, sub_categories_one_hot_train, state_one_hot_train, pg_one_hot_train, tp_one_hot_train, price_standardized_train, previous_standardized_train, title_train_bow, text_train_bow)).tocsr()
X_test_bow= hstack((categories_one_hot_test, sub_categories_one_hot_test, state_one_hot_test, pg_one_hot_test, tp_one_hot_test, price_standardized_test,  previous_standardized_test, title_test_bow, text_test_bow)).tocsr()
X_cv_bow = hstack((categories_one_hot_cv, sub_categories_one_hot_cv, state_one_hot_cv, pg_one_hot_cv, tp_one_hot_cv, price_standardized_cv, previous_standardized_cv,title_cv_bow, text_cv_bow)).tocsr()
```

In [37]:

```python
print(X_test_bow.shape)
print(y_test.shape)

print(X_train_bow.shape)
print(y_train.shape)

print(X_cv_bow.shape)
print(y_cv.shape)
```

```
(32775, 7320)
(32775,)
(53531, 7320)
(53531,)
(22942, 7320)
(22942,)
```

## a. Train model and return score for parameter tuninng

In [83]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
# https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html

# A good example of Gridsearch with NB: https://scikit-learn.org/stable/tutorial/text_analytics/working_with_text_data.html

from sklearn.model_selection import GridSearchCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt

NB = MultinomialNB()
parameters = {'alpha':[0.00001,0.0001,0.001, 0.01, 0.1,0.5,1, 5, 10,20, 50, 100]}
clf = GridSearchCV(NB, parameters, cv=10, return_train_score = True, scoring='roc_auc')
clf.fit(X_train_bow, y_train)
```

Out[83]:

```
GridSearchCV(cv=10, error_score='raise-deprecating',
             estimator=MultinomialNB(alpha=1.0, class_prior=None,
                                     fit_prior=True),
             iid='warn', n_jobs=None,
             param_grid={'alpha': [1e-05, 0.0001, 0.001, 0.01, 0.1, 0.5, 1, 5,
                                   10, 20, 50, 100]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='roc_auc', verbose=0)
```

In [84]:

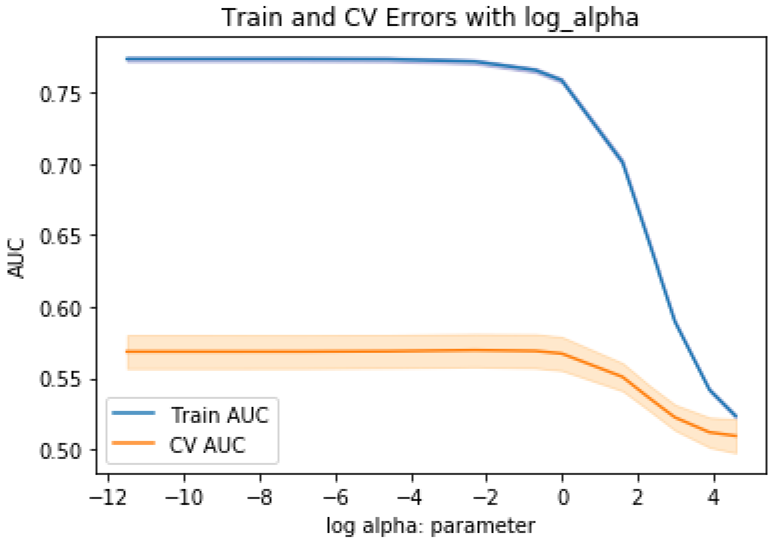
```
train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']
```

## b. plot error and find best alpha.

## c. represemtation of results

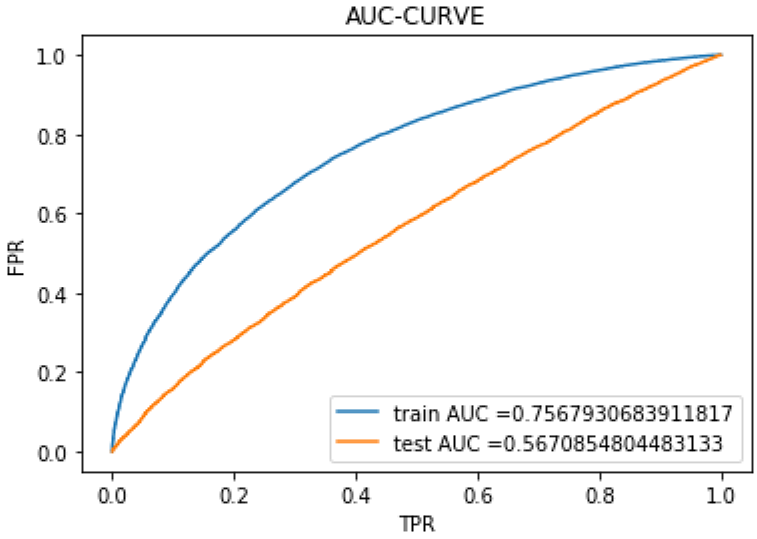
In [86]:

```
import math


alpha = [0.00001,0.0001,0.001, 0.01, 0.1,0.5,1, 5, 10,20, 50, 100]

log_alphas = []

for i in alpha:
    j= math.log(i)
    log_alphas.append(j)



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

plt.plot(log_alphas, cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(log_alphas,cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=
0.2,color='darkorange')
plt.legend()
plt.xlabel("log alpha: parameter")
plt.ylabel("AUC")
plt.title("Train and CV Errors with log_alpha")
plt.show()
```



```
In [124]: best_alpha = 0.5

          NB = MultinomialNB(alpha = best_alpha, fit_prior= True, class_prior = [0.5,0.5])
          print(NB)
          clf = NB.fit(X_train_bow, y_train)
          # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimate
          s of the positive class
          # not the predicted outputs



          # DO not use predict for ROC curve
          # https://discuss.analyticsvidhya.com/t/what-is-the-difference-between-predict-an
          d-predict-proba/67376/2
          y_train_pred = clf.predict_proba(X_train_bow)
          y_test_pred = clf.predict_proba(X_test_bow)

          train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred[:, 1])
          test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred[:, 1])

          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("TPR")
          plt.ylabel("FPR")
          plt.title("AUC-CURVE")
          plt.show()


          print("="*100)
```

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



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

## d. Top features of importance

In [41]:

```
print (clf. feature_log_prob_)
print (np.shape(clf.feature_log_prob_))


vectorizer_cat = CountVectorizer(vocabulary=list(sorted_cat_dict.keys()), lowerca
se=False, binary=True)
vectorizer_sub_cat = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()),
lowercase=False, binary=True)
vectorizer_state = CountVectorizer(vocabulary=list(sorted_state_dict.keys()), low
ercase=False, binary=True)
vectorizer_pg = CountVectorizer(vocabulary=list(grade_list.keys()),lowercase=Fals
e, binary=True)
vectorizer_tp = CountVectorizer(vocabulary=list(sorted_tp_dict.keys()),lowercase=
False, binary=True)
```

```
[[ -9.82674758  -9.82674758  -7.92319671 ... -10.39853391 -11.31482464
  -11.24583177]
 [ -9.24915523  -9.24915523  -7.86976164 ... -10.25510869 -11.39897757
  -11.22129639]]
(2, 7320)
```

In [46]:

```
string = ['price','previous']
bow_feature_names = vectorizer_cat.get_feature_names() + vectorizer_sub_cat.get_f
eature_names() + \
                    vectorizer_state.get_feature_names()+ vectorizer_pg.get_featu
re_names()+ \
                    vectorizer_tp.get_feature_names() + string +\
                    vectorizer_bow_title.get_feature_names() + vectorizer_bow.get
_feature_names()


print(len(bow_feature_names))
```

```
7320
```

In [56]:

```
#http://hiphoff.com/finding-words-and-phrases-most-associated-with-an-outcome/
likelihood_df = pd.DataFrame(clf.feature_log_prob_.transpose(),columns=['Negative', 'Positive'], \
                             index= bow_feature_names)
print(likelihood_df)
```

```
                     Negative   Positive
Warmth              -9.826748  -9.249155
Care_Hunger         -9.826748  -9.249155
History_Civics      -7.923197  -7.869762
Music_Arts          -7.269270  -7.317681
AppliedLearning     -6.988802  -7.168299
SpecialNeeds        -6.959032  -7.060235
Health_Sports       -6.958116  -6.982650
Math_Science        -5.881415  -5.924142
Literacy_Language   -5.819414  -5.676800
Economics          -11.063510 -10.839362
CommunityService   -10.041859 -10.510858
FinancialLiteracy  -10.103734 -10.186706
ParentInvolvement  -10.041859  -9.929440
Extracurricular     -9.749189  -9.837913
Civics_Government   -9.691202  -9.861513
ForeignLanguages    -9.535041  -9.829205
NutritionEducation  -9.019408  -9.349494
Warmth              -9.826748  -9.249155
Care_Hunger         -9.826748  -9.249155
SocialSciences      -9.078685  -9.006146
PerformingArts      -9.094070  -8.929232
CharacterEducation  -8.587906  -8.963903
TeamSports          -8.520160  -8.877703
Other               -8.728135  -8.805590
College_CareerPrep  -8.626006  -8.703975
Music               -8.744396  -8.468525
History_Geography   -8.564810  -8.498625
Health_LifeScience  -8.045799  -8.230810
EarlyDevelopment    -7.992877  -8.230810
ESL                 -8.160868  -8.165761
...                       ...        ...
writings           -11.245832 -11.293617
written             -8.829918  -8.664430
wrong              -10.552685 -10.471637
wrote              -10.621677 -10.712799
www                -11.245832 -11.358156
xylophones         -10.695785 -11.060523
yard               -10.621677 -11.070473
year                -5.488614  -5.485587
yearbook            -9.719775 -10.162215
yearly             -11.063510 -11.040915
yearn              -10.958150 -10.855362
yearning           -11.063510 -11.358156
years               -6.781457  -6.779603
yes                 -9.779495  -9.849644
yesterday          -11.181293 -11.385184
yet                 -7.688581  -7.689479
yoga                -8.107443  -8.207701
york                -9.253402  -9.309173
young               -7.081754  -7.062959
younger             -9.183197  -9.191008
youngest           -10.427521 -10.277481
youngsters         -11.468975 -11.153855
youth               -9.893439 -10.012683
youtube            -10.958150 -10.896520
zero               -11.314825 -11.344910
zest               -11.245832 -11.358156
zip                -10.552685 -10.913470
```

```
zone                    -10.398534 -10.255109
zones                   -11.314825 -11.398978
zoo                     -11.245832 -11.221296

[7320 rows x 2 columns]
```

In [133]:

```
positive = likelihood_df.sort_values(by =['Positive'],ascending = False)

print("Top 10 Positive Features")
positive.head(10)
```

```
Top 10 Positive Features
```

Out[133]:

| | Negative | Positive |
|---|---|---|
| **price** | -2.775766 | -2.776527 |
| **previous** | -3.162403 | -3.085973 |
| **Literacy_Language** | -3.641811 | -3.501567 |
| **Math_Science** | -3.703828 | -3.748920 |
| **Literacy** | -4.126274 | -3.938379 |
| **Mathematics** | -4.122472 | -4.130200 |
| **Literature_Writing** | -4.471431 | -4.358832 |
| **CA** | -4.774678 | -4.720335 |
| **students** | -4.750167 | -4.744904 |
| **Health_Sports** | -4.781073 | -4.807526 |

In [134]:

```
negative = likelihood_df.sort_values(by =['Negative'],ascending = False)
print("Top 10 Negative Features")
negative.head(10)
```

```
Top 10 Negative Features
```

Out[134]:

| | Negative | Positive |
|---|---|---|
| **price** | -2.775766 | -2.776527 |
| **previous** | -3.162403 | -3.085973 |
| **Literacy_Language** | -3.641811 | -3.501567 |
| **Math_Science** | -3.703828 | -3.748920 |
| **Mathematics** | -4.122472 | -4.130200 |
| **Literacy** | -4.126274 | -3.938379 |
| **Literature_Writing** | -4.471431 | -4.358832 |
| **students** | -4.750167 | -4.744904 |
| **CA** | -4.774678 | -4.720335 |
| **Health_Sports** | -4.781073 | -4.807526 |

### e. Confusion matrix

In [60]:

```
# how to choose threshold for Confusion matrix
#https://stackoverflow.com/questions/32627926/scikit-changing-the-threshold-to-create-multiple-confusion-matrixes


from sklearn.metrics import confusion_matrix


def predict(proba, threshold, fpr, tpr):
    t = threshold[np.argmax(fpr*(1-tpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
    predictions = []
    for i in proba:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions

# plot confusion matrix
# https://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html

#https://stackoverflow.com/questions/19984957/scikit-predict-default-threshold
# defaul threshold is 0.5
#print(confusion_matrix(y_train, neigh.predict(y_train_pred, tr_thresholds, train_fpr, train_fpr)))
```

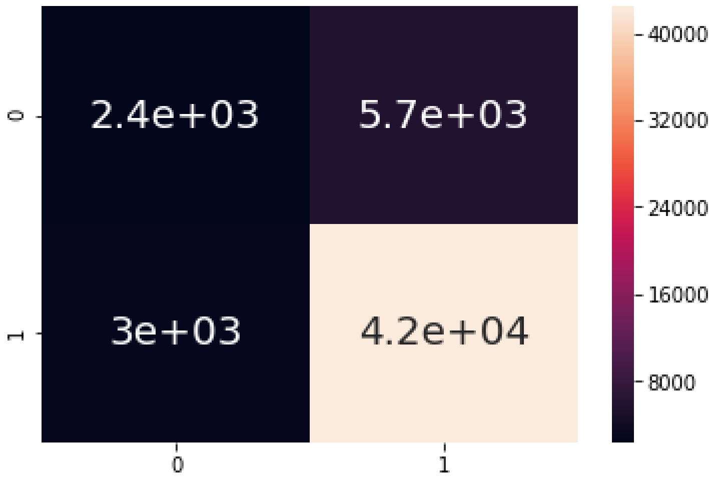
In [63]:

```
print("Train confusion matrix")
cm_train = pd.DataFrame(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_tpr)))
print(cm_train)

print("Test confusion matrix")
cm_test = pd.DataFrame(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_tpr)))
print(cm_test)
```

```
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.2736769277164048 for threshold 1
      0      1
0  2375   5730
1  3000  42426
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.10689802075992286 for threshold 1
      0      1
0   579   4384
1  2328  25484
```

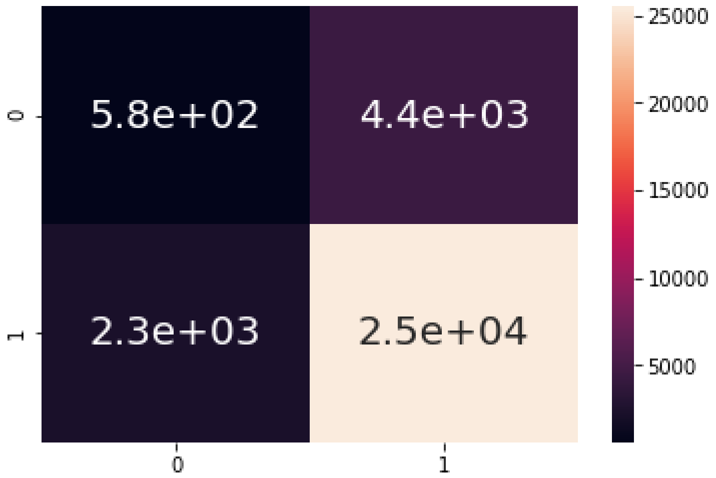
In [69]:

```
import seaborn as sns
sns.heatmap(cm_train, annot= True, annot_kws ={"size":20})
plt.show()
sns.heatmap(cm_test, annot= True, annot_kws ={"size":20})
```



Out[69]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x152065a0470>
```



## 2.4.2 Applying Naive Bayes on TFIDF, SET 2

In [69]:

```
from scipy.sparse import hstack

X_train_tfidf = hstack((categories_one_hot_train, sub_categories_one_hot_train, s
tate_one_hot_train, pg_one_hot_train, tp_one_hot_train, price_standardized_train,
previous_standardized_train, title_train_tfidf, text_train_tfidf)).tocsr()
X_test_tfidf = hstack((categories_one_hot_test, sub_categories_one_hot_test, stat
e_one_hot_test, pg_one_hot_test, tp_one_hot_test, price_standardized_test,  previ
ous_standardized_test, title_test_tfidf, text_test_tfidf)).tocsr()
X_cv_tfidf = hstack((categories_one_hot_cv, sub_categories_one_hot_cv, state_one_
hot_cv, pg_one_hot_cv, tp_one_hot_cv, price_standardized_cv, previous_standardize
d_cv,title_cv_tfidf, text_cv_tfidf)).tocsr()
```

In [70]:

```
print(X_test_tfidf.shape)
print(y_test.shape)

print(X_train_tfidf.shape)
print(y_train.shape)

print(X_cv_tfidf.shape)
print(y_cv.shape)
```

```
(32775, 7320)
(32775,)
(53531, 7320)
(53531,)
(22942, 7320)
(22942,)
```

## a. Train model and return score for parameter tuninng

In [71]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
# https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html

# A good example of Gridsearch with NB: https://scikit-learn.org/stable/tutorial/text_analytics/working_with_text_data.html

from sklearn.model_selection import GridSearchCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt

NB = MultinomialNB()
parameters = {'alpha':[0.00001,0.0001,0.001, 0.01, 0.1,0.5,1, 5, 10,20, 50, 100]}
clf = GridSearchCV(NB, parameters, cv=3, return_train_score = True, scoring='roc_auc')
clf.fit(X_train_tfidf, y_train)
```

Out[71]:

```
GridSearchCV(cv=3, error_score='raise-deprecating',
             estimator=MultinomialNB(alpha=1.0, class_prior=None,
                                     fit_prior=True),
             iid='warn', n_jobs=None,
             param_grid={'alpha': [1e-05, 0.0001, 0.001, 0.01, 0.1, 0.5, 1, 5,
                                   10, 20, 50, 100]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='roc_auc', verbose=0)
```

In [72]:

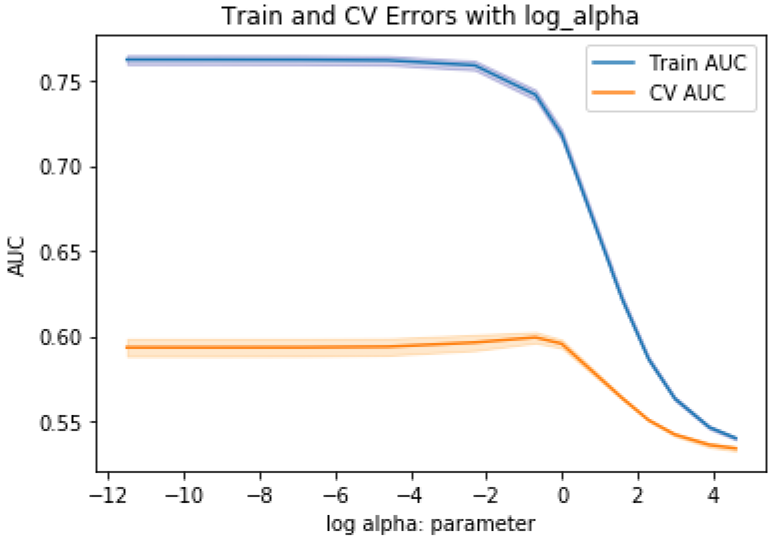
```
train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']
```

### b. plot error and find best alpha.

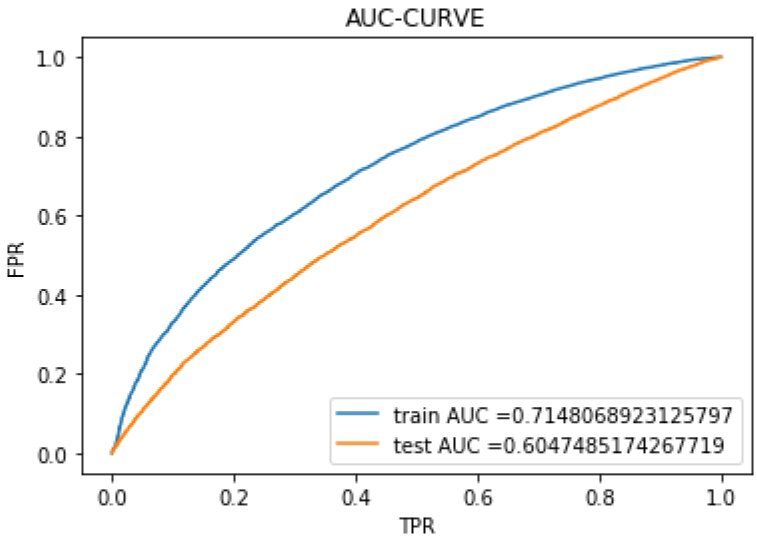
### c. represemtation of results

In [73]:

```
import math


alpha = [0.00001,0.0001,0.001, 0.01, 0.1,0.5,1, 5, 10,20, 50, 100]

log_alphas = []

for i in alpha:
    j= math.log(i)
    log_alphas.append(j)




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

plt.plot(log_alphas, cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(log_alphas,cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='darkorange')
plt.legend()
plt.xlabel("log alpha: parameter")
plt.ylabel("AUC")
plt.title("Train and CV Errors with log_alpha")
plt.show()
```



```
In [129]: best_alpha = 0.5

          NB = MultinomialNB(alpha = 0.5, class_prior = [0.5,0.5])
          clf = NB.fit(X_train_tfidf, y_train)
          # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimate
          s of the positive class
          # not the predicted outputs



          y_train_pred = clf.predict_proba(X_train_tfidf)
          y_test_pred = clf.predict_proba(X_test_tfidf)

          train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred[:, 1])
          test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred[:, 1])


          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("TPR")
          plt.ylabel("FPR")
          plt.title("AUC-CURVE")
          plt.show()


          print("="*100)
```



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

## d. Top features of importance

In [92]:

```
print (clf. feature_log_prob_)
print (np.shape(clf.feature_log_prob_))


vectorizer_cat = CountVectorizer(vocabulary=list(sorted_cat_dict.keys()), lowercase=False, binary=True)
vectorizer_sub_cat = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()), lowercase=False, binary=True)
vectorizer_state = CountVectorizer(vocabulary=list(sorted_state_dict.keys()), lowercase=False, binary=True)
vectorizer_pg = CountVectorizer(vocabulary=list(grade_list.keys()),lowercase=False, binary=True)
vectorizer_tp = CountVectorizer(vocabulary=list(sorted_tp_dict.keys()),lowercase=False, binary=True)
```

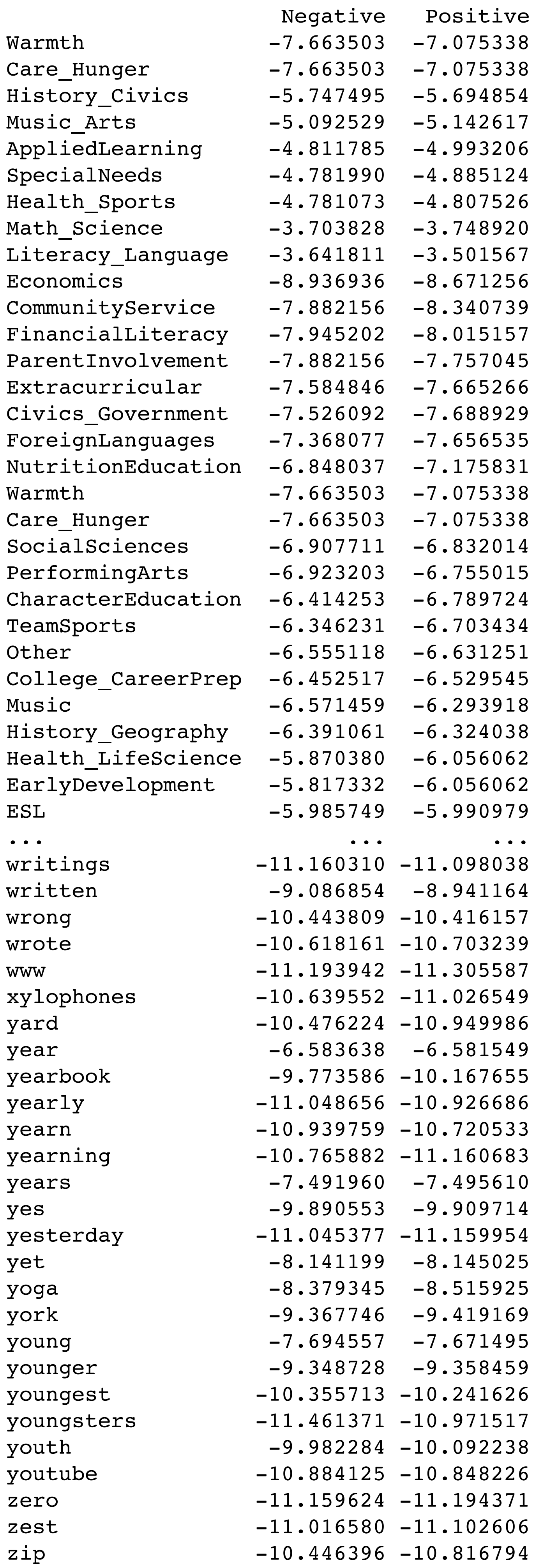
```
[[ -7.66350281  -7.66350281  -5.74749522 ... -10.3505508  -11.20695573
  -11.13770365]
 [ -7.07533794  -7.07533794  -5.69485366 ... -10.23634605 -11.29366931
  -11.079918  ]]
(2, 7320)
```

In [96]:

```
string = ['price','previous']
tfidf_feature_names = vectorizer_cat.get_feature_names() + vectorizer_sub_cat.get_feature_names() + \
                    vectorizer_state.get_feature_names()+ vectorizer_pg.get_feature_names()+ \
                    vectorizer_tp.get_feature_names() + string +\
                    vectorizer_tfidf_title.get_feature_names() + vectorizer_tfidf.get_feature_names()


print(len(tfidf_feature_names))
```

```
7320
```

```
In [97]: #http://hiphoff.com/finding-words-and-phrases-most-associated-with-an-outcome/
         likelihood_df = pd.DataFrame(clf.feature_log_prob_.transpose(),columns=['Negative', 'Positive'], \
                                      index= tfidf_feature_names)
         print(likelihood_df)
```



```
                     Negative   Positive
Warmth              -7.663503  -7.075338
Care_Hunger         -7.663503  -7.075338
History_Civics      -5.747495  -5.694854
Music_Arts          -5.092529  -5.142617
AppliedLearning     -4.811785  -4.993206
SpecialNeeds        -4.781990  -4.885124
Health_Sports       -4.781073  -4.807526
Math_Science        -3.703828  -3.748920
Literacy_Language   -3.641811  -3.501567
Economics           -8.936936  -8.671256
CommunityService    -7.882156  -8.340739
FinancialLiteracy   -7.945202  -8.015157
ParentInvolvement   -7.882156  -7.757045
Extracurricular     -7.584846  -7.665266
Civics_Government   -7.526092  -7.688929
ForeignLanguages    -7.368077  -7.656535
NutritionEducation  -6.848037  -7.175831
Warmth              -7.663503  -7.075338
Care_Hunger         -7.663503  -7.075338
SocialSciences      -6.907711  -6.832014
PerformingArts      -6.923203  -6.755015
CharacterEducation  -6.414253  -6.789724
TeamSports          -6.346231  -6.703434
Other               -6.555118  -6.631251
College_CareerPrep  -6.452517  -6.529545
Music               -6.571459  -6.293918
History_Geography   -6.391061  -6.324038
Health_LifeScience  -5.870380  -6.056062
EarlyDevelopment    -5.817332  -6.056062
ESL                 -5.985749  -5.990979
...                       ...        ...
writings           -11.160310 -11.098038
written             -9.086854  -8.941164
wrong              -10.443809 -10.416157
wrote              -10.618161 -10.703239
www                -11.193942 -11.305587
xylophones         -10.639552 -11.026549
yard               -10.476224 -10.949986
year                -6.583638  -6.581549
yearbook            -9.773586 -10.167655
yearly             -11.048656 -10.926686
yearn              -10.939759 -10.720533
yearning           -10.765882 -11.160683
years               -7.491960  -7.495610
yes                 -9.890553  -9.909714
yesterday          -11.045377 -11.159954
yet                 -8.141199  -8.145025
yoga                -8.379345  -8.515925
york                -9.367746  -9.419169
young               -7.694557  -7.671495
younger             -9.348728  -9.358459
youngest           -10.355713 -10.241626
youngsters         -11.461371 -10.971517
youth               -9.982284 -10.092238
youtube            -10.884125 -10.848226
zero               -11.159624 -11.194371
zest               -11.016580 -11.102606
zip                -10.446396 -10.816794
```

```
zone                 -10.350551 -10.236346
zones                -11.206956 -11.293669
zoo                  -11.137704 -11.079918

[7320 rows x 2 columns]
```

In [131]:
```
positive = likelihood_df.sort_values(by =['Positive'], ascending = False)
print("Top 10 Positive Features")
positive.head(10)
```

```
Top 10 Positive Features
```

Out[131]:

| | Negative | Positive |
|---|---|---|
| **price** | -2.775766 | -2.776527 |
| **previous** | -3.162403 | -3.085973 |
| **Literacy_Language** | -3.641811 | -3.501567 |
| **Math_Science** | -3.703828 | -3.748920 |
| **Literacy** | -4.126274 | -3.938379 |
| **Mathematics** | -4.122472 | -4.130200 |
| **Literature_Writing** | -4.471431 | -4.358832 |
| **CA** | -4.774678 | -4.720335 |
| **students** | -4.750167 | -4.744904 |
| **Health_Sports** | -4.781073 | -4.807526 |

In [132]:
```
negative = likelihood_df.sort_values(by =['Negative'],ascending = False)
print("Top 10 Negative Features")
negative.head(10)
```

```
Top 10 Negative Features
```

Out[132]:

| | Negative | Positive |
|---|---|---|
| **price** | -2.775766 | -2.776527 |
| **previous** | -3.162403 | -3.085973 |
| **Literacy_Language** | -3.641811 | -3.501567 |
| **Math_Science** | -3.703828 | -3.748920 |
| **Mathematics** | -4.122472 | -4.130200 |
| **Literacy** | -4.126274 | -3.938379 |
| **Literature_Writing** | -4.471431 | -4.358832 |
| **students** | -4.750167 | -4.744904 |
| **CA** | -4.774678 | -4.720335 |
| **Health_Sports** | -4.781073 | -4.807526 |

## e. Confusion matrix

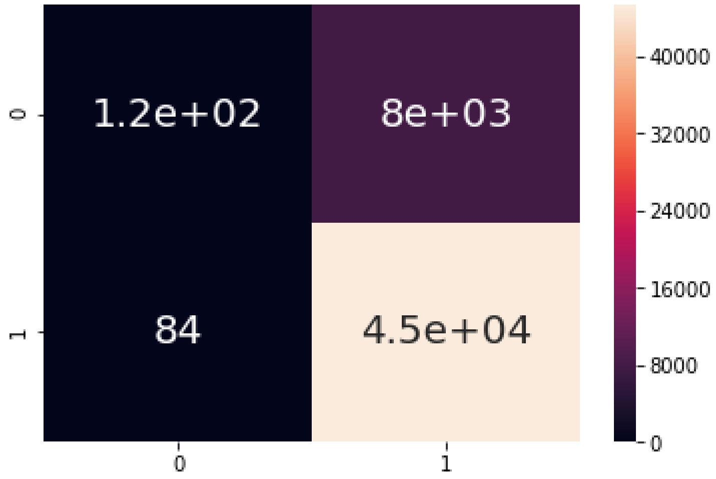
In [100]:
```
# how to choose threshold for Confusion matrix
#https://stackoverflow.com/questions/32627926/scikit-changing-the-threshold-to-create-multiple-confusion-matrixes


from sklearn.metrics import confusion_matrix


def predict(proba, threshold, fpr, tpr):
    t = threshold[np.argmax(fpr*(1-tpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
    predictions = []
    for i in proba:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions

# plot confusion matrix
# https://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html

#https://stackoverflow.com/questions/19984957/scikit-predict-default-threshold
# defaul threshold is 0.5
#print(confusion_matrix(y_train, neigh.predict(y_train_pred, tr_thresholds, train_fpr, train_fpr)))
```

In [104]:
```
print("Train confusion matrix")
cm_train = pd.DataFrame(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_tpr)))
print(cm_train)

print("Test confusion matrix")
cm_test = pd.DataFrame(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_tpr)))
print(cm_test)
```

```
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.015147754862848452 for threshold 1
     0      1
0  123   7982
1   84  45342
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.004826221735362038 for threshold 1
    0      1
0  24   4939
1  55  27757
```

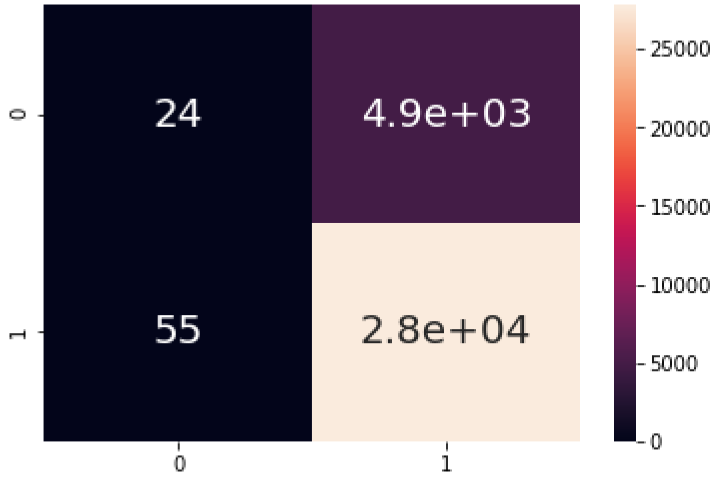
In [102]:

```
import seaborn as sns
sns.heatmap(cm_train, annot= True, annot_kws ={"size":20})
plt.show()
sns.heatmap(cm_test, annot= True, annot_kws ={"size":20})
```



Out[102]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x229a835eef0>
```



In [ ]:

# 3. Conclusions

In [130]:

```
# Please compare all your models using Prettytable library

from prettytable import PrettyTable

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

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

x.add_row(["BOW", "NB",0.5, 0.567])
x.add_row(["TFIDF", "NB", 0.1, 0.605])

print(x)
```

```
+------------+-------+-----------------+-------+
| Vectorizer | Model | Hyper Parameter |  AUC  |
+------------+-------+-----------------+-------+
|    BOW     |   NB  |       0.5       | 0.567 |
|   TFIDF    |   NB  |       0.1       | 0.605 |
+------------+-------+-----------------+-------+
```

In [ ]: