DonorsChoose

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

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

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

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

About the DonorsChoose Data Set

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

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

- Literature & Writing, Social Sciences

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

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

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

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

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

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

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

Notes on the Essay Data

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

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

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

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

your neighborhood, and your school are all helpful."

- __project_essay_2:__ "About your project: How will these materials make a difference in your students' learning and improve their school lives?"

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

In [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
```

1.1 Reading Data

In [2]:

```
project_data = pd.read_csv('train_new_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' 'school_state'
 'project_submitted_datetime' 'project_grade_category'
 'project_subject_categories' 'project_subject_subcategories'
 'project_title' 'project_essay_1' 'project_essay_2' 'project_essay_3'
 'project_essay_4' 'project_resource_summary'
 'teacher_number_of_previously_posted_projects' 'project_is_approved']
```

In [4]:

```
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[4]:

	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 [5]:

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

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

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

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

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

1.3 preprocessing of `project_subject_subcategories`

In [6]:

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

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

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

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

preprocessing of school states

In [7]:

```
from collections import Counter
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]))
```

preprocessing of teacher prefix

In [8]:

```
from collections import Counter
my_counter = Counter()
for word in project_data['teacher_prefix'].values:
    my_counter.update(word.split())
prefix_dict = dict(my_counter)
sorted_prefix_dict = dict(sorted(prefix_dict.items(), key=lambda kv: kv[1]))
```

preprocessing of project_grade_category

In [9]:

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

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

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

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

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

1.3 Text preprocessing

In [10]:

```
# 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 [11]:

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

Out[11]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime	proj
0	0	p036502	484aaf11257089a66cfedc9461c6bd0a	Ms.	NV	18-11-2016 14:45	Supe Word Cent
1	3	p185307	525fdbb6ec7f538a48beebaa0a51b24f	Mr.	NC	12-08-2016 15:42	\"Kid Insp Equi to In Activ

splitting dataset into training,testing and testing

In [12]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(project_data,
project_data['project_is_approved'], test_size=0.33, stratify = project_data['project_is_approved'
])
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33, stratify=y_train)
```

In [13]:

```
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 [14]:

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

```
My students are special to me because they demonstrate daily, their desire to learn. They are atte
ntive during lessons, which motivates me to plan more engaging lessons. They have high attendance
and most complete their nightly homework assignments! \r\nThe majority of our students are transie
nt and low-income, they value their time here at school.  These students often experience
challenges out of the classroom that could easily distract them, so while they are here with me, m
y job is not only to teach them, but to love on them. I work hard daily to ensure that our
classroom community is one that offers them a happy place.  Despite the many challenges they face,
I am looking to keep things simple and provide my students with creative and meaningful learning e
xperiences.\r\nI teach 4th grade and I want to encourage good health and fitness with my students.
These students have been active all summer and sitting in hard chairs all day would be counter pro
ductive. I am requesting Kore Wooble Chairs to be used to help my students remain active while foc
using on their assignments. They will be able to work and move at the same time. Using these seats
will activate various muscles because they will be sitting on an unbalanced seat. \r\n\r\nSome
kids need more movement than others. So, for my students with ADHD, being in motion allows their b
rains to be engaged. Movement actually allows for alertness and attention. \r\n\r\nI have students
who get up during independent work time and wander, and they stated that is because they don't lik
e the hard seats. They like flexible seating options.  We have discussed using the large stability
balls, but after looking online and viewing the videos on the Kore Wobble Chairs we decided that t
hese would be the best choice to use.  Having these in our class will allow the students to have a
t least 60 minutes of movement each day! A healthy body leads to a healthy mind!nannan
==================================================
My Students have not had the opportunity for a true art education, Because of the lack of supplies
and the funds to get art supplies in this low-income community, the previous teacher resorted to c
opy paper and pencil only assignments. There is so much more to creating art then two materials.
By giving students the opportunity to crate with more choice will open theirs harts to see that th
e world and their teacher really do hope the best for them in all that they do. \r\n \r\nAs a teac
her in a low-income/high poverty school district, my students are faced with several challenges bo
th in and out of the classroom. Despite the many challenges they face, I am looking to keep things
simple and provide my students with creative and meaningful learning experiences.\r\n\r\nWith your
help we can help close the achievement gap that my school is in when students do not have the nece
ssary materials to succeed? \r\n\r\nHelping students think out side the box is essential to critic
al thinking. This thinking helps in all areas of learning. My art class helps student in achieving
this ability of higher thinking skills witch are so desperately needed in today's society. \r\nYou
can give hope to kids that don't see a lot of positives in their lives.\r\nIn capturing my
students harts in art this will allow them the power to succeed in there future. One day become po
sitive and productive members of their community .\r\nWithout the help of generous donors, we can
not afford even the most basic supplies. Please help me to get my students the supplies we need fo
r a successful art program.  \r\nnannan
==================================================
I teach third grade in an elementary school in Lexington, SC. I am working hard this school year
to ensure opportunities for success for my awesome class. My students are an amazing group of chil
dren who come to school every day ready and eager to learn. \r\n\r\nMy classroom is always full of
lively children who love to learn. We know that having the opportunity to move while learning is v
ital. Our classroom thrives on flexible seating. We have stability balls, rocker seats, stools, ch
airs, and yoga mats. We need something else!Our classroom is full of flexible seating, but it is l
acking in opportunities to stand and learn. These standing desks will offer another option for my
third grade students who like to move. No one likes to sit still all day. Why should we when we ca
n stand? These 4 standing desks are adjustable so that the students can stand comfortably while wo
rking and learning. These sranding desks will help my students focus on their work and improve the
ir learning. There are also many health benefits to being able to stand while working instead of s
itting all of the time.nannan
==================================================
```

In [15]:

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

function to decontract sentances

In [16]:

```
# 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
```

preprocessing essays-train

In [17]:

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

```
100%|██████████| 49041/49041 [00:44<00:00, 1111.23it/s]
```

In [18]:

```
preprocessed_essays_train[2000]
```

Out[18]:

```
'brains head feet shoes steer direction choose dr seuss students wiggling children love learn rang
e five six years age case students need wiggle listen students come home lives sit boring chair la
y floor play video games day donation project make world difference students day day learning kind
ergarten use game centers like children work independently teacher small group setting centers
used help reinforce new skill topic children learning week cases students keep playing center week
s later make sure still good skill centers used whole year impact students year years come nannan'
```

preprocessing project titles-train

In [19]:

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

```
100%|██████████| 49041/49041 [00:01<00:00, 25881.47it/s]
```

preprocessing project essay-test

In [20]:

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

```
100%|██████████| 36052/36052 [00:32<00:00, 1120.67it/s]
```

In [21]:

```
preprocessed_essays_test[2000]
```

Out[21]:

```
'students enter first grade much excitement enthusiasm 75 english language learners poor families
receive free reduced price lunches parents highest expectations children excel school encourage we
ll many lack time skills personally help academically many students struggle school also learning
speak english others special needs make learning challenging backgrounds many struggle come school
unprepared teacher must find strategies motivate best thrive succeed first grade quite challenging
students must learn concepts subject areas important writing reading math try best ensure every on
e progresses ultimately masters first grade concepts remember child always looked forward school a
ugust would anxiously await weekly ads neighborhood stores see school supplies sale mother would s
hop things needed school even though could not afford everything wanted parents always scrimped sa
ved would basic school supplies included backpack every year project hope give students advantage
child something basic help stay organized focused school backpack hope provide good quality
backpack use not elementary school middle school backpacks allow students come school equally prep
ared school year students backpacks worn years use hand downs siblings cousins brand new backpack
excite students attending school encourage best students better focus learning not lacking student
s support donations low income inner city students access school backpack proudly wear carry resul
t motivated succeed school building important foundation skills help reach higher academic success
throughout educational years breaking cycle poverty instilling lifelong love learning nannan'
```

preprocessing project titles-test

In [22]:

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

```
100%|██████████| 36052/36052 [00:01<00:00, 23577.39it/s]
```

preprocessing project essays-CV

In [23]:

```
from tqdm import tqdm
preprocessed_essays_cv = []
# tqdm is for printing the status bar
for sentance in tqdm(X_cv['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=sent.lower()
    sent = ' '.join(e for e in sent.split() if e.lower() not in stopwords)
    preprocessed_essays_cv.append(sent.lower().strip())
```

```
100%|██████████| 24155/24155 [00:21<00:00, 1121.73it/s]
```

In [24]:

```
preprocessed_essays_cv[2000]
```

Out[24]:

```
'90 wonderful students live near poverty line 60 students learning english school sometimes school
struggles get resources need teach care incredible students makes job school even important many s
tudents teach experience poverty starting students love create art read write computer codes even
perform shakespeare big dreams need work hard help reach dreams kids love laugh love smile kids ne
ed healthy teeth smile bright many not always basic necessities sometimes means go without
toothpaste not clean toothbrushes let change give kids toothpaste toothbrushes show healthy
confident smile school sends home extra food friday 50 students need toothpaste toothbrushes send
us go home students believe kids need learn smile project nannan'
```

preprocessing project titles-CV

In [25]:

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

```
100%|██████████| 24155/24155 [00:00<00:00, 26761.34it/s]
```

1.5 Preparing data for models

In [26]:

```
project_data.columns
```

Out[26]:

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'project_submitted_datetime', 'project_title', 'project_essay_1',
       'project_essay_2', 'project_essay_3', 'project_essay_4',
       'project_resource_summary',
       'teacher_number_of_previously_posted_projects', 'project_is_approved',
       'clean_categories', 'clean_subcategories', 'clean_pgc', 'essay'],
      dtype='object')
```

we are going to consider

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

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

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

1.5.1 Vectorizing Categorical data

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

VECTORIZING CLEAN CATEGORIES USING ONE HOT ENCODING

In [27]:

```
from sklearn.feature_extraction.text import CountVectorizer
vectorizer_clean_cat = CountVectorizer(vocabulary=list(sorted_cat_dict.keys()), lowercase=False, b
inary=True)
vectorizer_clean_cat.fit(X_train['clean_categories'].values)
categories_one_hot_train = vectorizer_clean_cat.transform(X_train['clean_categories'].values)
categories_one_hot_test = vectorizer_clean_cat.transform(X_test['clean_categories'].values)
categories_one_hot_cv = vectorizer_clean_cat.transform(X_cv['clean_categories'].values)
print(vectorizer_clean_cat.get_feature_names())
print("Shape of matrix of Train data after one hot encoding ",categories_one_hot_train.shape)
print("Shape of matrix of Test data after one hot encoding ",categories_one_hot_test.shape)
print("Shape of matrix of CV data after one hot encoding ",categories_one_hot_cv.shape)
```

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

VECTORIZING CLEAN SUBCATEGORIES USING ONE HOT ENCODING

In [28]:

```
vectorizer_clean_subcat = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()), lowercase=F
alse, binary=
True)
vectorizer_clean_subcat.fit(X_train['clean_subcategories'].values)
sub_categories_one_hot_train = vectorizer_clean_subcat.transform(X_train['clean_subcategories'].va
lues)
sub_categories_one_hot_test =
vectorizer_clean_subcat.transform(X_test['clean_subcategories'].values)
sub_categories_one_hot_cv = vectorizer_clean_subcat.transform(X_cv['clean_subcategories'].values)
print(vectorizer_clean_subcat.get_feature_names())
print("Shape of matrix of Train data after one hot encoding ",sub_categories_one_hot_train.shape)
print("Shape of matrix of Test data after one hot encoding ",sub_categories_one_hot_test.shape)
print("Shape of matrix of Cross Validation data after one hot encoding ",sub_categories_one_hot_cv
.shape)
```

```
['Economics', 'CommunityService', 'FinancialLiteracy', 'ParentInvolvement', 'Extracurricular',
'Civics_Government', 'ForeignLanguages', 'NutritionEducation', 'Warmth', 'Care_Hunger',
'SocialSciences', 'PerformingArts', 'CharacterEducation', 'TeamSports', 'Other',
'College_CareerPrep', 'Music', 'History_Geography', 'Health_LifeScience', 'EarlyDevelopment', 'ESL
', 'Gym_Fitness', 'EnvironmentalScience', 'VisualArts', 'Health_Wellness', 'AppliedSciences',
'SpecialNeeds', 'Literature_Writing', 'Mathematics', 'Literacy']
Shape of matrix of Train data after one hot encoding  (49041, 30)
Shape of matrix of Test data after one hot encoding  (36052, 30)
Shape of matrix of Cross Validation data after one hot encoding  (24155, 30)
```

VECTORIZING SCHOOL STATE USING ONE HOT ENCODING

In [29]:

```
# you can do the similar thing with state, teacher_prefix and project_grade_category also
vectorizer_school_state= CountVectorizer(vocabulary=list(sorted_state_dict.keys()), lowercase=Fals
e, binary=
True)
vectorizer_school_state.fit(X_train['school_state'].values)
school_state_one_hot_train = vectorizer_school_state.transform(X_train['school_state'].values)
school_state_one_hot_test = vectorizer_school_state.transform(X_test['school_state'].values)
school_state_one_hot_cv = vectorizer_school_state.transform(X_cv['school_state'].values)
print(vectorizer_school_state.get_feature_names())
print("Shape of matrix of Train data after one hot encoding ",school_state_one_hot_train.shape)
print("Shape of matrix of Test data after one hot encoding ",school_state_one_hot_test.shape)
print("Shape of matrix of Cross Validation data after one hot encoding ",school_state_one_hot_cv
.shape)
```

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

VECTORIZING TEACHER PREFIX USING ONE HOT ENCODING

In [30]:

```
vectorizer_prefix = CountVectorizer(vocabulary=list(sorted_prefix_dict.keys()), lowercase=False, b
inary=
True)
vectorizer_prefix.fit(X_train['teacher_prefix'].values)
teacher_prefix_one_hot_train = vectorizer_prefix.transform(X_train['teacher_prefix'].values)
teacher_prefix_one_hot_test = vectorizer_prefix.transform(X_test['teacher_prefix'].values)
teacher_prefix_one_hot_cv = vectorizer_prefix.transform(X_cv['teacher_prefix'].values)
print(vectorizer_prefix.get_feature_names())
print("Shape of matrix of Train data after one hot encoding ",teacher_prefix_one_hot_train.shape)
print("Shape of matrix of Test data after one hot encoding ",teacher_prefix_one_hot_test.shape)
print("Shape of matrix of Cross Validation data after one hot encoding ",teacher_prefix_one_hot_cv
.shape)
```

```
['Dr.', 'Teacher', 'Mr.', 'Ms.', 'Mrs.']
Shape of matrix of Train data after one hot encoding  (49041, 5)
Shape of matrix of Test data after one hot encoding  (36052, 5)
Shape of matrix of Cross Validation data after one hot encoding  (24155, 5)
```

VECTORIZING PROJECT GRADE CATEGORY USING ONE HOT ENCODING

In [31]:

```
vectorizer_pgc= CountVectorizer(vocabulary=list(sorted_pgc_dict.keys()), lowercase=False, binary=
True)
vectorizer_pgc.fit(X_train['clean_pgc'].values)
clean_project_grade_category_one_hot_train = vectorizer_pgc.transform(X_train['clean_pgc'].values)
clean_project_grade_category_one_hot_test = vectorizer_pgc.transform(X_test['clean_pgc'].values)
clean_project_grade_category_one_hot_cv = vectorizer_pgc.transform(X_cv['clean_pgc'].values)
print(vectorizer_pgc.get_feature_names())
print("Shape of matrix of Train data after one hot encoding
",clean_project_grade_category_one_hot_train.shape)
print("Shape of matrix of Test data after one hot encoding
",clean_project_grade_category_one_hot_test.shape)
print("Shape of matrix of Cross Validation data after one hot encoding
",clean_project_grade_category_one_hot_cv
.shape)
```

```
['Grades9-12', 'Grades6-8', 'Grades3-5', 'GradesPreK-2']
Shape of matrix of Train data after one hot encoding  (49041, 4)
Shape of matrix of Test data after one hot encoding  (36052, 4)
Shape of matrix of Cross Validation data after one hot encoding  (24155, 4)
```

1.5.2 Vectorizing Text data

1.5.2.1 Bag of words

In [32]:

```
vectorizer_bow = CountVectorizer(min_df=10)
vectorizer_bow.fit(preprocessed_essays_train)
text_bow_train = vectorizer_bow.transform(preprocessed_essays_train)
print("Shape of matrix after one hot encoding ",text_bow_train.shape)
```

```
Shape of matrix after one hot encoding  (49041, 12006)
```

In [33]:

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

```
Shape of matrix after one hot encoding  (36052, 12006)
```

In [34]:

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

```
Shape of matrix after one hot encoding  (24155, 12006)
```

In [35]:

```
vectorizer_bow_ppt = CountVectorizer(min_df=10)
vectorizer_bow_ppt.fit(preprocessed_project_titles_train)
title_bow_train = vectorizer_bow_ppt.transform(preprocessed_project_titles_train)
print("Shape of matrix after one hot encoding ",title_bow_train.shape)
```

```
Shape of matrix after one hot encoding  (49041, 2004)
```

In [36]:

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

```
Shape of matrix after one hot encoding  (36052, 2004)
```

In [37]:

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

```
Shape of matrix after one hot encoding  (24155, 2004)
```

1.5.2.2 TFIDF vectorizer

In [38]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer_tfidf_text = TfidfVectorizer(min_df=10)
vectorizer_tfidf_text.fit(preprocessed_essays_train)
text_tfidf_train = vectorizer_tfidf_text.transform(preprocessed_essays_train)
print("Shape of matrix after one hot encoding ",text_tfidf_train.shape)
```

```
Shape of matrix after one hot encoding  (49041, 12006)
```

In [39]:

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

```
Shape of matrix after one hot encoding  (36052, 12006)
```

In [40]:

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

```
Shape of matrix after one hot encoding  (24155, 12006)
```

In [41]:

```
vectorizer_tfidf_ppt = TfidfVectorizer(min_df=10)
vectorizer_tfidf_ppt.fit(preprocessed_project_titles_train)
title_tfidf_train = vectorizer_tfidf_ppt.transform(preprocessed_project_titles_train)
print("Shape of matrix after one hot encoding ",title_tfidf_train.shape)
```

```
Shape of matrix after one hot encoding  (49041, 2004)
```

In [42]:

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

```
Shape of matrix after one hot encoding  (36052, 2004)
```

In [43]:

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

```
Shape of matrix after one hot encoding  (24155, 2004)
```

GROUPNG DATA AND PERFORMING SUM OPERATION

In [44]:

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

In [45]:

```
X_train = pd.merge(X_train, price_data, on='id', how='left')
X_test = pd.merge(X_test, price_data, on='id', how='left')
X_cv = pd.merge(X_cv, price_data, on='id', how='left')
```

NORMALIZING PRICE

In [50]:

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

scalar = Normalizer()
scalar.fit(X_train['price'].values.reshape(1,-1))# finding the mean and standard deviation of this
data
price_normalized_train = scalar.transform(X_train['price'].values.reshape( 1,-1))
price_normalized_test = scalar.transform(X_test['price'].values.reshape(1, -1))
price_normalized_cv = scalar.transform(X_cv['price'].values.reshape(1, -1))
```

In [51]:

```
print(price_normalized_train.shape)
print(price_normalized_test.shape)
print(price_normalized_cv.shape)
```

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

NORMALIZING PREVIOUSLY POSTED PROJECTS

In [52]:

```
scalar = Normalizer()
scalar.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(1,-1))
prev_project_normalized_train =
scalar.transform(X_train['teacher_number_of_previously_posted_projects'].values.reshape(1,-1))
prev_project_normalized_test =
scalar.transform(X_test['teacher_number_of_previously_posted_projects'].values.reshape(1,-1))
prev_project_normalized_cv = scalar.transform(X_cv['teacher_number_of_previously_posted_projects']
.values.reshape(1,-1))
```

In [53]:

```
print(prev_project_normalized_train.shape)
print(prev_project_normalized_test.shape)
print(prev_project_normalized_cv.shape)
```

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

Normalizing quantity

In [54]:

```
price_scalar.fit(X_train['quantity'].values.reshape(1,-1)) # finding the mean and standard
deviation of this data
quantity_normalized_train = price_scalar.transform(X_train['quantity'].values.reshape(1,- 1))
quantity_normalized_test = price_scalar.transform(X_test['quantity'].values.reshape(1,-1))
```

```
quantity_normalized_cv = price_scalar.transform(X_cv['quantity'].values.reshape(-1,-1))
```

In [55]:

```
print(quantity_normalized_train.shape)
print(quantity_normalized_test.shape)
print(quantity_normalized_cv.shape)
```

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

1.5.4 Merging all the above features

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

Assignment 4: Naive Bayes

1. **Apply Multinomial NaiveBayes on these feature sets**
   - Set 1: categorical, numerical features + project_title(BOW) + preprocessed_eassay (BOW)
   - Set 2: categorical, numerical features + project_title(TFIDF)+ preprocessed_eassay (TFIDF)
2. **The hyper paramter tuning(find best Alpha)**
   - Find the best hyper parameter which will give the maximum AUC value
   - Consider a wide range of alpha values for hyperparameter tuning, start as low as 0.00001
   - Find the best hyper paramter using k-fold cross validation or simple cross validation data
   - Use gridsearch cv or randomsearch cv or you can also write your own for loops to do this task of hyperparameter tuning
3. **Feature importance**
   - Find the top 10 features of positive class and top 10 features of negative class for both feature sets Set 1 and Set 2 using values of `feature_log_prob_` parameter of MultinomialNB and print their corresponding feature names
4. **Representation of results**
   - You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure. Here on X-axis you will have alpha values, since they have a wide range, just to represent those alpha values on the graph, apply log function on those alpha values.
   - Once after you found the best hyper parameter, you need to train your model with it, and find the AUC on test data and plot the ROC curve on both train and test.
   - Along with plotting ROC curve, you need to print the confusion matrix with predicted and original labels of test data points. Please visualize your confusion matrices using seaborn heatmaps.
5. **Conclusion**
   - You need to summarize the results at the end of the notebook, summarize it in the table format. To print out a table please refer to this prettytable library link

2. Naive Bayes

PERFORMING HORIZONTAL STACK ON VECTORSFOR TRAIN TEST AND CV

In [56]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
S_BOW_train=
hstack((categories_one_hot_train,sub_categories_one_hot_train,school_state_one_hot_train,teacher_pr
efix_one_hot_train,clean_project_grade_category_one_hot_train,text_bow_train,title_bow_train,price_
normalized_train.T,prev_project_normalized_train.T,quantity_normalized_train.T)).tocsr()
S_BOW_train.shape
```

Out[56]:

```
(49041, 14112)
```

In [57]:

```
S_BOW_test= hstack((categories_one_hot_test,sub_categories_one_hot_test,school_state_one_hot_test,
teacher_prefix_one_hot_test,clean_project_grade_category_one_hot_test,text_bow_test,title_bow_test
,price_normalized_test.T,prev_project_normalized_test.T,quantity_normalized_test.T)).tocsr()
S_BOW_test.shape
```

Out[57]:

```
(36052, 14112)
```

In [58]:

```
S_BOW_cv=
hstack((categories_one_hot_cv,sub_categories_one_hot_cv,school_state_one_hot_cv,teacher_prefix_one_
hot_cv,clean_project_grade_category_one_hot_cv,text_bow_cv,title_bow_cv,price_normalized_cv.T,prev_
project_normalized_cv.T,quantity_normalized_cv.T)).tocsr()
S_BOW_cv.shape
```

Out[58]:

```
(24155, 14112)
```

In [59]:

```
print("BOW with other features Data matrix")
print(S_BOW_train.shape, y_train.shape)
print(S_BOW_cv.shape, y_cv.shape)
print(S_BOW_test.shape, y_test.shape)
print("*"*50)
```

```
BOW with other features Data matrix
(49041, 14112) (49041,)
(24155, 14112) (24155,)
(36052, 14112) (36052,)
**************************************************
```

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

BATCHWISE PREDICTION:

In [60]:

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

    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000
    # consider you X_tr shape is 49041, then your tr_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
    if data.shape[0]%1000 !=0:
        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

FINDING BEST HYPERPARAMETER

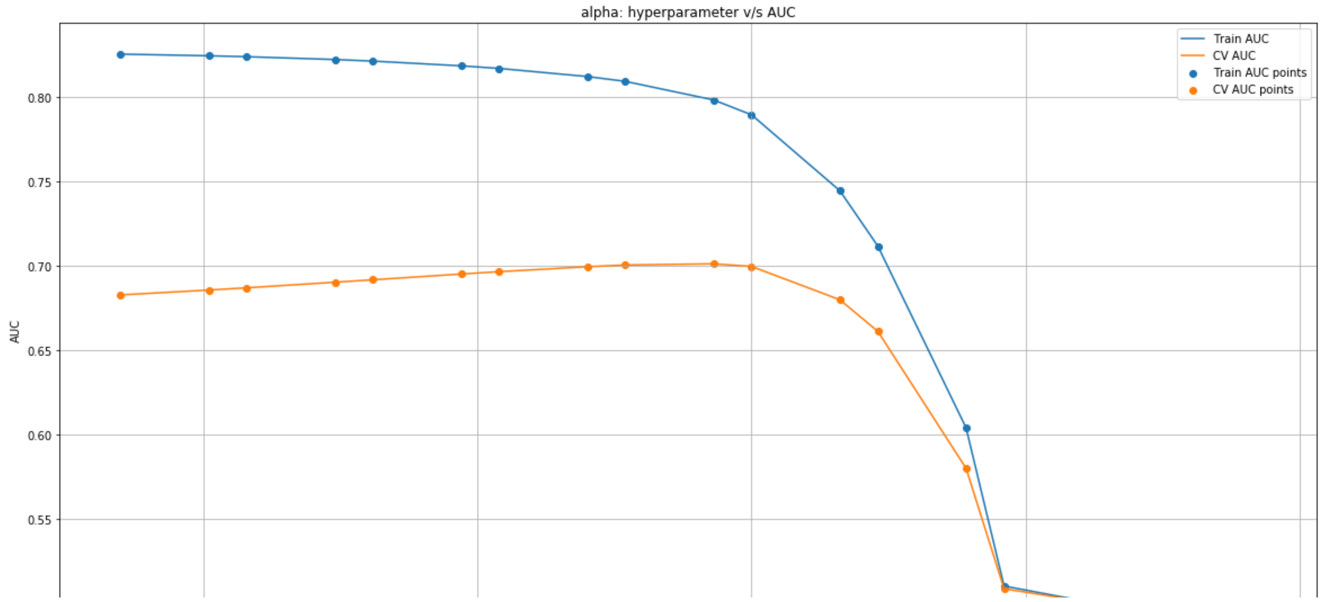
In [61]:

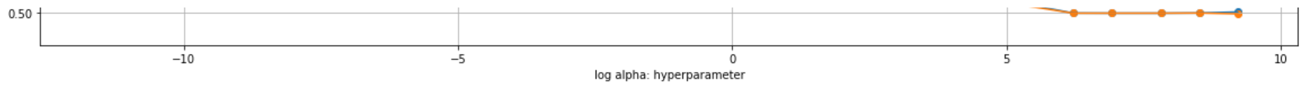
```
# Please write all the code with proper documentation
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import roc_auc_score
import math
train_auc = []
cv_auc = []
log_alphas = []
alphas = [0.00001, 0.00005, 0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 5, 10, 50, 100, 500, 1000, 2500, 5000, 10000]
for i in tqdm(alphas):
    nb = MultinomialNB(alpha = i,class_prior=[0.5,0.5])
    nb.fit(abs(S_BOW_train), y_train)
    y_train_pred = batch_predict(nb, abs(S_BOW_train))
    y_cv_pred = batch_predict(nb,abs(S_BOW_cv))
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
# not the predicted outputs
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
for a in tqdm(alphas):
    b = math.log(a)
    log_alphas.append(b)
```

```
100%|██████████| 20/20 [00:09<00:00,  2.40it/s]
100%|██████████| 20/20 [00:00<00:00, 15609.62it/s]
```

In [62]:

```
plt.figure(figsize=(20,10))
plt.plot(log_alphas, train_auc, label='Train AUC')
plt.plot(log_alphas, cv_auc, label='CV AUC')
plt.scatter(log_alphas, train_auc, label='Train AUC points')
plt.scatter(log_alphas, cv_auc, label='CV AUC points')
plt.legend()
plt.xlabel("log alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("alpha: hyperparameter v/s AUC")
plt.grid()
plt.show()
```





Using Grid Search to find alpha:

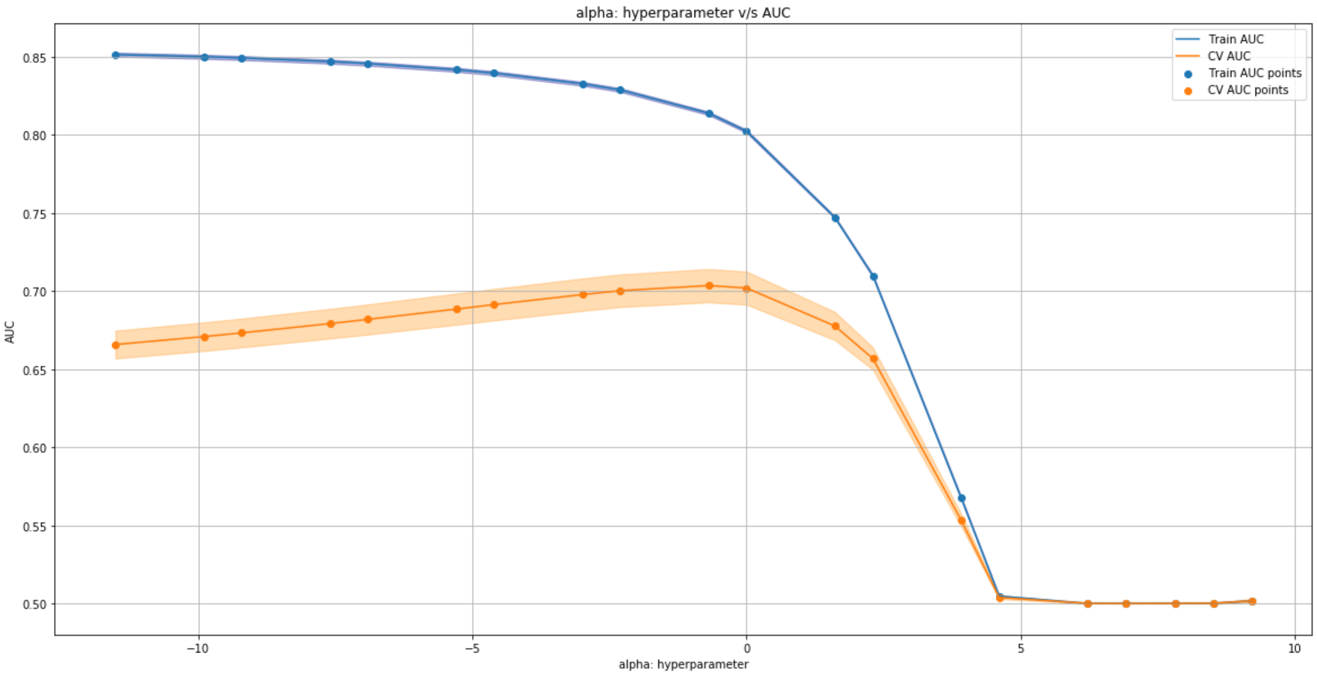
In [63]:

```
from sklearn.model_selection import GridSearchCV
nb = MultinomialNB(class_prior=[0.5,0.5])
parameters = {'alpha':[0.00001, 0.00005, 0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 5,
10, 50, 100, 500, 1000, 2500, 5000, 10000]}
clf = GridSearchCV(nb, parameters, cv= 5, scoring='roc_auc')
best_model=clf.fit(abs(S_BOW_train), y_train)
train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']
```

In [64]:

```
alphas = [0.00001, 0.00005, 0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 5, 10, 50, 100, 5
00, 1000, 2500, 5000, 10000]
log_alphas =[]
for a in tqdm(alphas):
    b = math.log(a)
    log_alphas.append(b)
plt.figure(figsize=(20,10))
plt.plot(log_alphas, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(log_alphas,train_auc - train_auc_std,train_auc + train_auc_std,alpha=0.3,col
or='darkblue')
plt.plot(log_alphas, 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=0.3,color='darkoran
ge')
plt.scatter(log_alphas, train_auc, label='Train AUC points')
plt.scatter(log_alphas, cv_auc, label='CV AUC points')
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("alpha: hyperparameter v/s AUC")
plt.grid()
plt.show()
```

```
100%|██████████| 20/20 [00:00<00:00, 22745.68it/s]
```



In [65]:

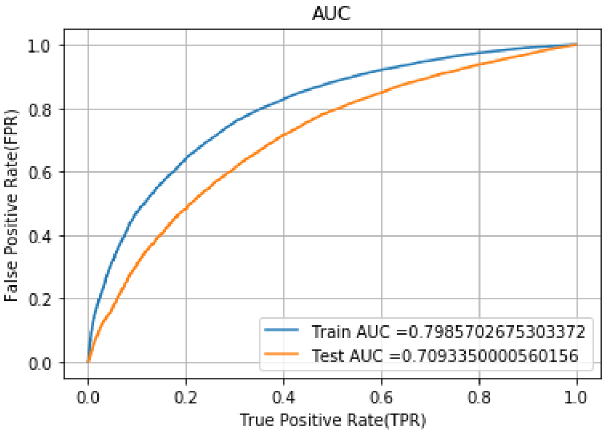
```
print('Best alpha:', best_model.best_estimator_.get_params()['alpha'])
```

```
Best alpha: 0.5
```

Applying Multinomial Naive bayes

In [66]:

```
nb_BOW = MultinomialNB(alpha = 0.5,class_prior=[0.5,0.5])
nb_model=nb_BOW.fit(abs(S_BOW_train), y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs
y_train_pred = batch_predict(nb_BOW,abs(S_BOW_train))
y_test_pred = batch_predict(nb_BOW, abs(S_BOW_test))
train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)
plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
```



In [67]:

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

confusion matrix for train using heatmap

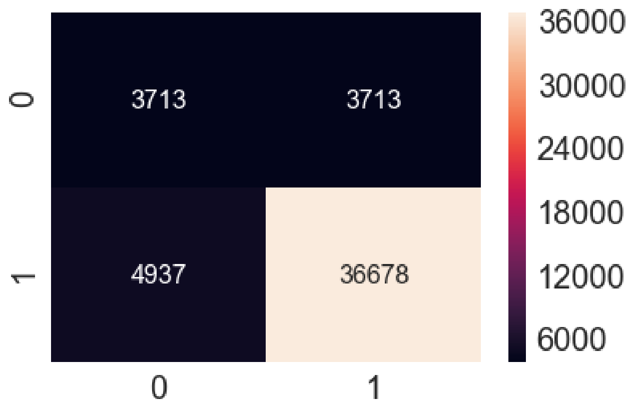
In [68]:

```
conf_matr_df_train = pd.DataFrame(confusion_matrix(y_train, prediction(y_train_pred, tr_thresholds
,
train_fpr, train_fpr)), range(2),range(2))
sns.set(font_scale=2)#for label size
sns.heatmap(conf_matr_df_train, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.025
```

Out[68]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1bf2e49db38>
```



confusion matrix for test data using heatmap

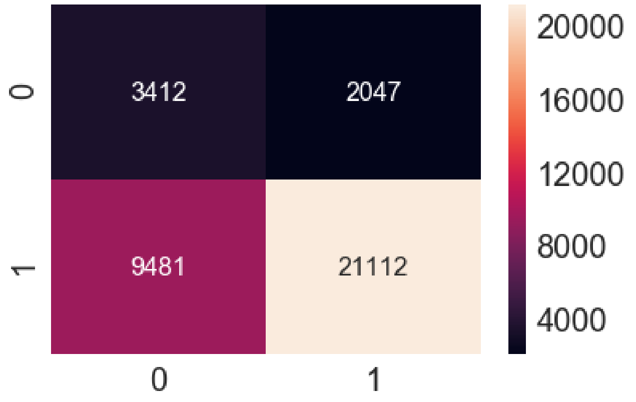
In [69]:

```
conf_matr_df_test = pd.DataFrame(confusion_matrix(y_test, prediction(y_test_pred, tr_thresholds, train_fpr, train_tpr)), range(2),range(2))
sns.set(font_scale=2)#for label size
sns.heatmap(conf_matr_df_test, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
the maximum value of tpr*(1-fpr) 0.5297744004612582 for threshold 0.551
```

Out[69]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1bf10507240>
```



getting feature probablities and words stored in vectorizer

In [83]:

```
features_prob_BOW = {}
for a in range(14112) :
    features_prob_BOW[a] = nb_model.feature_log_prob_[0,a]
len(features_prob_BOW.values())
```

Out[83]:

```
14112
```

In [84]:

```
features_name_BOW= []
for a in vectorizer_clean_cat.get_feature_names() :
    features_name_BOW.append(a)
```

In [85]:

```
for a in vectorizer_clean_subcat.get_feature_names() :
    features_name_BOW.append(a)
```

In [86]:

```
for a in vectorizer_school_state.get_feature_names() :
    features_name_BOW.append(a)
```

In [87]:

```
for a in vectorizer_pgc.get_feature_names() :
    features_name_BOW.append(a)
```

In [88]:

```
for a in vectorizer_prefix.get_feature_names() :
    features_name_BOW.append(a)
```

In [89]:

```
features_name_BOW.append("price")
```

In [90]:

```
features_name_BOW.append("prev_proposed_projects")
```

In [91]:

```
features_name_BOW.append("quantity")
```

In [92]:

```
for a in vectorizer_bow.get_feature_names() :
    features_name_BOW.append(a)
```

In [93]:

```
for a in vectorizer_bow_ppt.get_feature_names() :
    features_name_BOW.append(a)
```

In [94]:

```
len(features_name_BOW)
```

Out[94]:

```
14112
```

In [95]:

```
bow_features = pd.DataFrame({'feature_prob_estimates' : list(features_prob_BOW.values()),
'feature_names' : features_name_BOW})
```

In [96]:

```
bow_features.sort_values(by='feature_prob_estimates', ascending=True)
```

Out[96]:

	feature_names	feature_prob_estimates

	feature_names	feature_prob_estimates
1211	beanbags	-14.529042
10808	tender	-14.529042
12086	yummy	-14.529042
707	amongst	-14.529042
9287	roll	-14.529042
1793	caregivers	-14.529042
7568	orlando	-14.529042
714	amplifier	-14.529042
12091	zenergy	-14.529042
12093	zest	-14.529042
12094	zillion	-14.529042
10806	tendencies	-14.529042
12095	zip	-14.529042
13462	place	-14.529042
3538	dry	-14.529042
10817	tent	-14.529042
5881	intimidated	-14.529042
2859	cycle	-14.529042
12096	ziploc	-14.529042
13112	kinders	-14.529042
2868	dads	-14.529042
9252	risk	-14.529042
740	anger	-14.529042
3540	drying	-14.529042
11413	universe	-14.529042
695	ambitions	-14.529042
694	ambition	-14.529042
4782	generally	-14.529042
13812	stepping	-14.529042
7495	opens	-14.529042
...	...	...
10962	till	-5.690200
7480	olympic	-5.673807
10445	struggling	-5.590379
10771	technologically	-5.581236
669	alphabetic	-5.565754
7258	nevada	-5.556578
6608	major	-5.521920
12017	worth	-5.503226
11735	wander	-5.462688
12054	yarn	-5.458999
2095	clarinets	-5.444832
11502	usable	-5.410269
2901	daunting	-5.407205
9874	sketching	-5.398720

	feature_names	feature_prob_estimates
6721	matching	-5.383881
8738	readers	-5.375801
310	abdominal	-5.375590
6528	lounge	-5.343507
2242	combines	-5.301943
11976	wordless	-5.135298
7214	necklace	-5.102704
6663	mantra	-5.001340
7165	names	-4.968397
5203	helen	-4.795572
6295	leapfrog	-4.762922
7332	northwest	-4.751175
2107	classifying	-4.570357
6299	learned	-4.415740
9478	scholarship	-4.081691
10446	strumming	-3.004910

14112 rows × 2 columns

2.4.1.2 Top 10 important features of negative class from SET 1

In [102]:

```
neg_class_prob_sorted=nb_model.feature_log_prob_[0,:].argsort()
```

Printing top 10 features of negative class using BOW data

In [103]:

```
print(np.take(features_name_BOW,neg_class_prob_sorted[-10:]))
```

```
['necklace' 'mantra' 'names' 'helen' 'leapfrog' 'northwest' 'classifying'
 'learned' 'scholarship' 'strumming']
```

2.4.1.1 Top 10 important features of positive class from SET 1

In [104]:

```
pos_class_prob_sorted=nb_model.feature_log_prob_[1,:].argsort()
```

Printing top 10 features of negative class using BOW data

In [105]:

```
print(np.take(features_name_BOW,pos_class_prob_sorted[-10:]))
```

```
['wordless' 'names' 'mantra' 'helen' 'leapfrog' 'northwest' 'classifying'
 'learned' 'scholarship' 'strumming']
```

Horizontal stack of tfidf data-train CV and test(set 2)

In [106]:

```
# Please write all the code with proper documentation
# Please write all the code with proper documentation
# Please write all the code with proper documentation
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
S_TFIDF_train=
hstack((categories_one_hot_train,sub_categories_one_hot_train,school_state_one_hot_train,teacher_pr
efix_one_hot_train,clean_project_grade_category_one_hot_train,text_tfidf_train,title_tfidf_train,p
rice_normalized_train.T,prev_project_normalized_train.T,quantity_normalized_train.T)).tocsr()
S_TFIDF_train.shape
```

Out[106]:

```
(49041, 14112)
```

In [107]:

```
S_TFIDF_test=
hstack((categories_one_hot_test,sub_categories_one_hot_test,school_state_one_hot_test,teacher_prefi
x_one_hot_test,clean_project_grade_category_one_hot_test,text_tfidf_test,title_tfidf_test,price_nor
malized_test.T,prev_project_normalized_test.T,quantity_normalized_test.T)).tocsr()
S_TFIDF_test.shape
```

Out[107]:

```
(36052, 14112)
```

In [108]:

```
S_TFIDF_cv=
hstack((categories_one_hot_cv,sub_categories_one_hot_cv,school_state_one_hot_cv,teacher_prefix_one_
hot_cv,clean_project_grade_category_one_hot_cv,text_tfidf_cv,title_tfidf_cv,price_normalized_cv.T,p
rev_project_normalized_cv.T,quantity_normalized_cv.T)).tocsr()
S_TFIDF_cv.shape
```

Out[108]:

```
(24155, 14112)
```

Finding best alpha using AUC

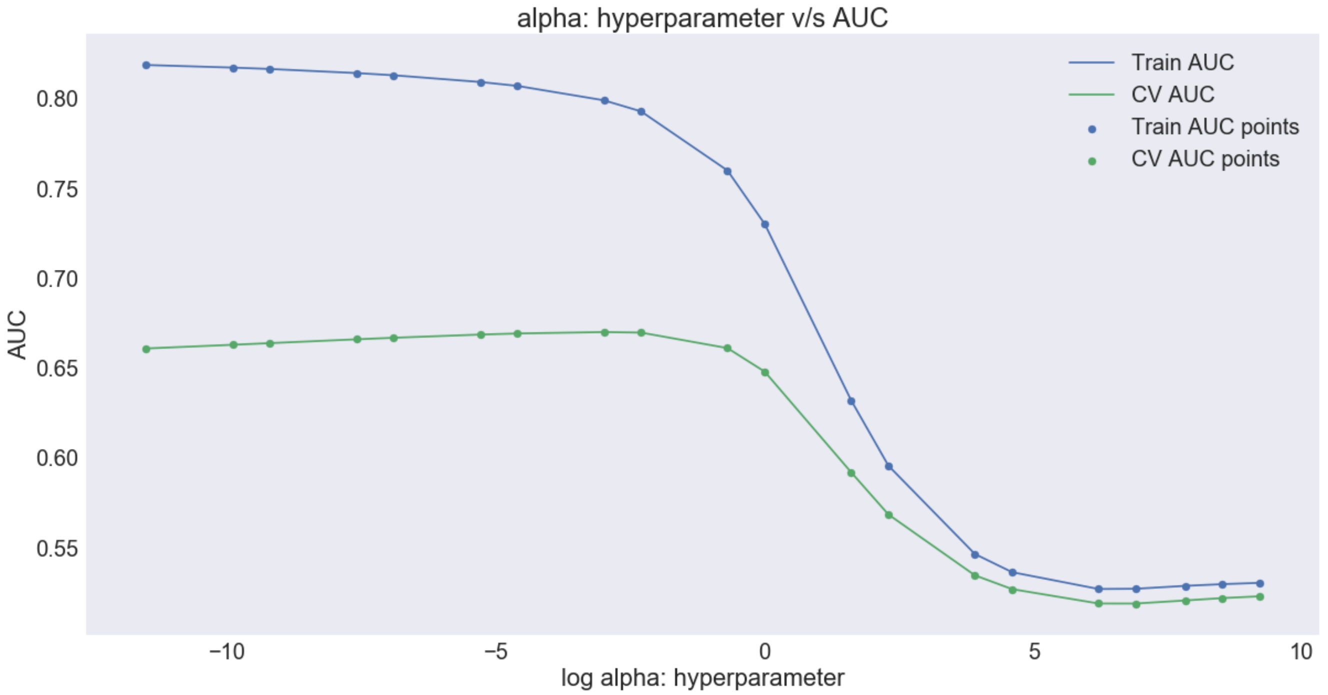
In [109]:

```
train_auc = []
cv_auc = []
log_alphas = []
alphas = [0.00001, 0.00005, 0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 5, 10, 50, 100, 5
00, 1000, 2500, 5000, 10000]
for i in tqdm(alphas):
    nb = MultinomialNB(alpha = i,class_prior=[0.5,0.5])
    nb.fit(abs(S_TFIDF_train), y_train)
    y_train_pred = batch_predict(nb,abs(S_TFIDF_train))
    y_cv_pred = batch_predict(nb,abs(S_TFIDF_cv))
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
for a in tqdm(alphas):
    b = math.log(a)
    log_alphas.append(b)
```

```
100%|██████████| 20/20 [00:08<00:00,  2.23it/s]
100%|██████████| 20/20 [00:00<00:00, 20106.92it/s]
```

In [110]:

```
plt.figure(figsize=(20,10))
plt.plot(log_alphas, train_auc, label='Train AUC')
plt.plot(log_alphas, cv_auc, label='CV AUC')
plt.scatter(log_alphas, train_auc, label='Train AUC points')
plt.scatter(log_alphas, cv_auc, label='CV AUC points')
plt.legend()
plt.xlabel("log alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("alpha: hyperparameter v/s AUC")
plt.grid()
plt.show()
```



Using GridsearchCV to find best alpha

In [111]:

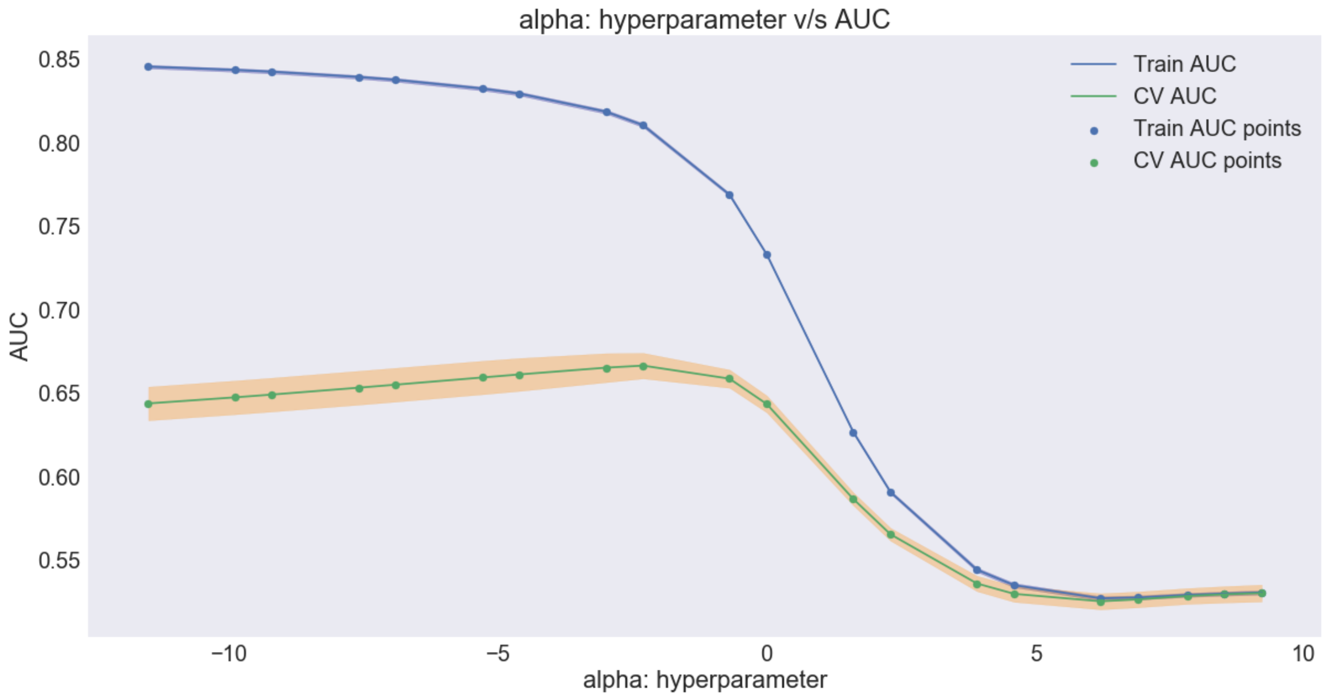
```
from sklearn.model_selection import GridSearchCV
nb = MultinomialNB(class_prior=[0.5,0.5])
parameters = {'alpha':[0.00001, 0.00005, 0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 5, 
10, 50, 100, 500, 1000, 2500, 5000, 10000]}
clf = GridSearchCV(nb, parameters, cv= 5, scoring='roc_auc')
best_model=clf.fit(abs(S_TFIDF_train), y_train)
train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']
```

In [112]:

```
alphas = [0.00001, 0.00005, 0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 5, 10, 50, 100, 5
00, 1000, 2500, 5000, 10000]
log_alphas =[]
for a in tqdm(alphas):
    b = math.log(a)
    log_alphas.append(b)
plt.figure(figsize=(20,10))
plt.plot(log_alphas, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(log_alphas,train_auc - train_auc_std,train_auc + train_auc_std,alpha=0.3,col
or='darkblue')
plt.plot(log_alphas, 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=0.3,color='darkoran
ge')
plt.scatter(log_alphas, train_auc, label='Train AUC points')
plt.scatter(log_alphas, cv_auc, label='CV AUC points')
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("alpha: hyperparameter v/s AUC")
plt.grid()
plt.show()
```

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

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



In [113]:

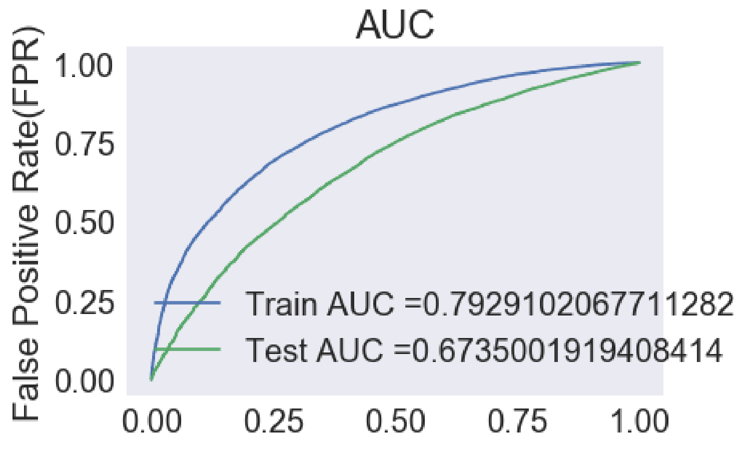
```
print('Best alpha:', best_model.best_estimator_.get_params()['alpha'])
```

```
Best alpha: 0.1
```

Applying Multinomial Naive bayes on TFIDF data

In [114]:

```
nb_TFIDF = MultinomialNB(alpha = 0.1,class_prior=[0.5,0.5])
nb_model=nb_TFIDF.fit(abs(S_TFIDF_train), y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive 
class
# not the predicted outputs
y_train_pred = batch_predict(nb_TFIDF,abs(S_TFIDF_train))
y_test_pred = batch_predict(nb_TFIDF, abs(S_TFIDF_test))
train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)
plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
```



True Positive Rate(TPR)

confusion matrix for train tfifd data

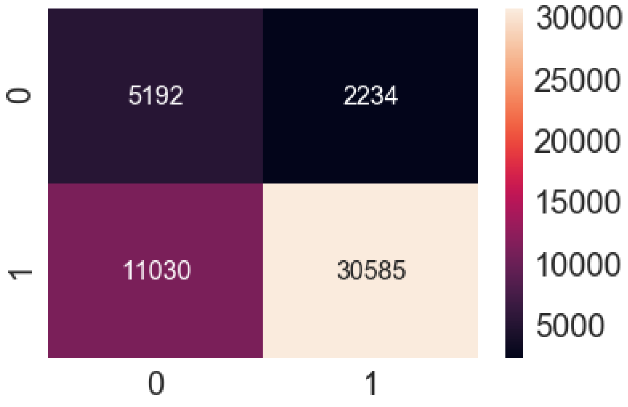
In [115]:

```
conf_matr_df_train = pd.DataFrame(confusion_matrix(y_train, prediction(y_train_pred, tr_thresholds
,
train_fpr, train_tpr)), range(2),range(2))
sns.set(font_scale=2)
sns.heatmap(conf_matr_df_train, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
the maximum value of tpr*(1-fpr) 0.5184757135476054 for threshold 0.48
```

Out[115]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1bf2d29a1d0>
```



confusion matrix for test tfifd data

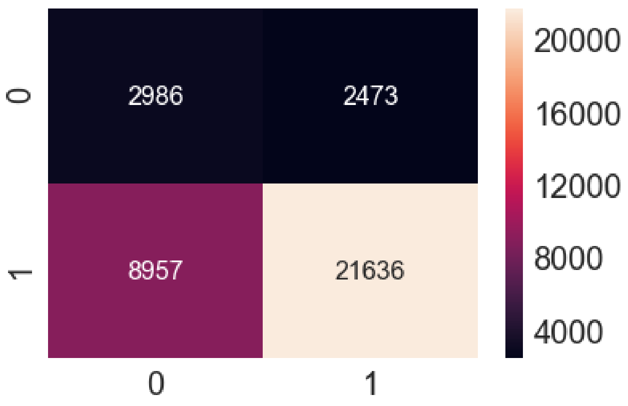
In [116]:

```
conf_matr_df_test = pd.DataFrame(confusion_matrix(y_test, prediction(y_test_pred, tr_thresholds,
train_fpr, train_tpr)), range(2),range(2))
sns.set(font_scale=2)#for label size
sns.heatmap(conf_matr_df_test, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
the maximum value of tpr*(1-fpr) 0.5184757135476054 for threshold 0.48
```

Out[116]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1bf0ffa9518>
```



In [119]:

```
features_prob_TFIDF = {}
for a in range(14112) :
    features_prob_TFIDF[a] = nb_TFIDF.feature_log_prob_[0,a]
len(features_prob_TFIDF.values())
```

Out[119]:

```
14112
```

In [120]:

```
features_name_TFIDF= []
for a in vectorizer_clean_cat.get_feature_names() :
    features_name_TFIDF.append(a)
```

In [121]:

```
for a in vectorizer_clean_subcat.get_feature_names() :
    features_name_TFIDF.append(a)
```

In [122]:

```
for a in vectorizer_school_state.get_feature_names() :
    features_name_TFIDF.append(a)
```

In [123]:

```
for a in vectorizer_pgc.get_feature_names() :
    features_name_TFIDF.append(a)
```

In [124]:

```
for a in vectorizer_prefix.get_feature_names() :
    features_name_TFIDF.append(a)
```

In [125]:

```
features_name_TFIDF.append("price")
```

In [126]:

```
features_name_TFIDF.append("prev_proposed_projects")
```

In [127]:

```
features_name_TFIDF.append("quantity")
```

In [128]:

```
for a in vectorizer_tfidf_text.get_feature_names() :
    features_name_TFIDF.append(a)
```

In [129]:

```
for a in vectorizer_tfidf_ppt.get_feature_names() :
    features_name_TFIDF.append(a)
```

In [130]:

```
len(features_name_TFIDF)
```

Out[130]:

```
14112
```

In [131]:

```
TFIDF_features = pd.DataFrame({'feature_prob_estimates' : list(features_prob_TFIDF.values()),
'feature_names' : features_name_TFIDF})
```

In [132]:

```
TFIDF_features.sort_values(by='feature_prob_estimates', ascending=True)
```

Out[132]:

	feature_names	feature_prob_estimates
13883	tales	-13.878585
9427	saturdays	-13.878585
9431	save	-13.878585
6013	joe	-13.878585
9432	saved	-13.878585
9439	savy	-13.878585
11804	webcam	-13.878585
11803	web	-13.878585
3886	entertaining	-13.878585
925	ashamed	-13.878585
6020	joining	-13.878585
7087	mounting	-13.878585
9425	satisfying	-13.878585
3855	enlarge	-13.878585
9512	scrapbook	-13.878585
9515	scratched	-13.878585
13843	studies	-13.878585
11771	watercolor	-13.878585
9518	scream	-13.878585
13845	study	-13.878585
11132	transience	-13.878585
11754	warriors	-13.878585
9533	sdc	-13.878585
3801	encouragement	-13.878585
974	assortment	-13.878585
3853	enjoys	-13.878585
6011	job	-13.878585
7663	pad	-13.878585
13708	sharing	-13.878585
4115	expeditionary	-13.878585
...	...	...
2107	classifying	-5.921364
29	ESL	-5.892080
83	GA	-5.865242
84	IL	-5.861937
30	Gym_Fitness	-5.835885
6299	learned	-5.786507
27	Health_LifeScience	-5.785734

	feature_names	feature_prob_estimates
85	NC	-5.776604
28	EarlyDevelopment	-5.767557
9478	scholarship	-5.644167
2	History_Civics	-5.571866
31	EnvironmentalScience	-5.488998
86	FL	-5.433747
87	NY	-5.418809
32	VisualArts	-5.337480
33	Health_Wellness	-5.157472
88	TX	-5.155842
3	Music_Arts	-4.977354
34	AppliedSciences	-4.861708
4	AppliedLearning	-4.715231
6	Health_Sports	-4.701698
10446	strumming	-4.701410
89	CA	-4.693460
35	SpecialNeeds	-4.651289
5	SpecialNeeds	-4.651289
36	Literature_Writing	-4.311200
38	Literacy	-4.039049
37	Mathematics	-4.028445
7	Math_Science	-3.605916
8	Literacy_Language	-3.523353

14112 rows × 2 columns

2.4.2.1 Top 10 important features of positive class from SET 2

In [133]:

```
neg_class_prob_sorted=nb_TFIDF.feature_log_prob_[1,:].argsort()
```

In [134]:

```
print(np.take(features_name_TFIDF,pos_class_prob_sorted[-10:]))
```

```
['wordless' 'names' 'mantra' 'helen' 'leapfrog' 'northwest' 'classifying'
 'learned' 'scholarship' 'strumming']
```

2.4.2.2 Top 10 important features of negative class from SET 2

In [135]:

```
neg_class_prob_sorted=nb_model.feature_log_prob_[0,:].argsort()
```

In [136]:

```
print(np.take(features_name_TFIDF,neg_class_prob_sorted[-10:]))
```

```
['Health_Sports' 'strumming' 'CA' 'SpecialNeeds' 'SpecialNeeds'
 'Literature_Writing' 'Literacy' 'Mathematics' 'Math_Science'
 'Literacy_Language']
```

3. Conclusions

In [137]:

```
# Please compare all your models using Prettytable library
from prettytable import PrettyTable
#If you get a ModuleNotFoundError error , install prettytable using: pip3 install prettytable
p = PrettyTable()
p.field_names = ["Vectorizer", "Model", "Alpha:Hyper Parameter", "AUC"]
p.add_row(["BOW", "Naive Bayes", 0.5, 0.7])
p.add_row(["TFIDF", "Naive Bayes", 0.1, 0.65])
print(p)
```

```
+------------+-------------+-----------------------+------+
| Vectorizer |    Model    | Alpha:Hyper Parameter | AUC  |
+------------+-------------+-----------------------+------+
|    BOW     | Naive Bayes |          0.5          | 0.7  |
|   TFIDF    | Naive Bayes |          0.1          | 0.65 |
+------------+-------------+-----------------------+------+
```

Conclusion: 1.Bow vectorization has better AUC value compared to TFIDF vectorization 2.Naive Bayes has lesser timecomplexity than KNN classifier. 3.Naive bayes is memory efficient when compared to KNN.