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 (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:** • `Literacy` • `Literature & Writing, Social Sciences`
`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`

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

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

In [2]:
```
%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 [3]:
```
project_data = pd.read_csv('train_data.csv')
resource_data = pd.read_csv('resources.csv')
```

In [4]:
```
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 [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.co
m/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-i
n-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 sp
ace "Math & Science"=> "Math","&", "Science"
            j=j.replace('The','') # if we have the words "The" we are going to re
place 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 traili
ng 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]:
```
sub_catogories = list(project_data['project_subject_subcategories'].values)
# remove special characters from list of strings python: https://stackoverflow.co
m/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-i
n-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 sp
ace "Math & Science"=> "Math","&", "Science"
            j=j.replace('The','') # if we have the words "The" we are going to re
place 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 traili
ng 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/22898595/4
084039
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]:
```
# 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	project_submitted_da
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	2016-12-05 1
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL	2016-10-25 0

1.4.2.3 Using Pretrained Models: TFIDF weighted W2V

In [11]:
```
# 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 [12]:
```
sent = decontracted(project_data['essay'].values[20000])
print(sent)
print("="*50)
```

```
My kindergarten students have varied disabilities ranging from speech and langua
ge delays, cognitive delays, gross/fine motor delays, to autism. They are eager
beavers and always strive to work their hardest working past their limitations.
\r\n\r\nThe materials we have are the ones I seek out for my students. I teach i
n a Title I school where most of the students receive free or reduced price lunc
h.  Despite their disabilities and limitations, my students love coming to schoo
l and come eager to learn and explore.Have you ever felt like you had ants in yo
ur pants and you needed to groove and move as you were in a meeting? This is how
my kids feel all the time. The want to be able to move as they learn or so they
say.Wobble chairs are the answer and I love then because they develop their cor
e, which enhances gross motor and in Turn fine motor skills. \r\nThey also want
to learn through games, my kids do not want to sit and do worksheets. They want
to learn to count by jumping and playing. Physical engagement is the key to our
success. The number toss and color and shape mats can make that happen. My stude
nts will forget they are doing work and just have the fun a 6 year old deserves.
nannan
==================================================
```

In [13]:
```
# \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)
```

```
My kindergarten students have varied disabilities ranging from speech and langua
ge delays, cognitive delays, gross/fine motor delays, to autism. They are eager
beavers and always strive to work their hardest working past their limitations.
The materials we have are the ones I seek out for my students. I teach in a Titl
e I school where most of the students receive free or reduced price lunch.  Desp
ite their disabilities and limitations, my students love coming to school and co
me eager to learn and explore.Have you ever felt like you had ants in your pants
and you needed to groove and move as you were in a meeting? This is how my kids
feel all the time. The want to be able to move as they learn or so they say.Wobb
le chairs are the answer and I love then because they develop their core, which
enhances gross motor and in Turn fine motor skills.  They also want to learn th
rough games, my kids do not want to sit and do worksheets. They want to learn to
count by jumping and playing. Physical engagement is the key to our success. The
number toss and color and shape mats can make that happen. My students will forg
et they are doing work and just have the fun a 6 year old deserves.nannan
```

In [14]:
```
#remove spacial character: https://stackoverflow.com/a/5843547/4084039
sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
print(sent)
```

```
My kindergarten students have varied disabilities ranging from speech and langua
ge delays cognitive delays gross fine motor delays to autism They are eager beav
ers and always strive to work their hardest working past their limitations The m
aterials we have are the ones I seek out for my students I teach in a Title I sc
hool where most of the students receive free or reduced price lunch Despite thei
r disabilities and limitations my students love coming to school and come eager
to learn and explore Have you ever felt like you had ants in your pants and you
needed to groove and move as you were in a meeting This is how my kids feel all
the time The want to be able to move as they learn or so they say Wobble chairs
are the answer and I love then because they develop their core which enhances gr
oss motor and in Turn fine motor skills They also want to learn through games my
kids do not want to sit and do worksheets They want to learn to count by jumping
and playing Physical engagement is the key to our success The number toss and co
lor and shape mats can make that happen My students will forget they are doing w
ork and just have the fun a 6 year old deserves 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', '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've
", '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 [16]:
```
# Combining all the above stundents
from tqdm import tqdm
preprocessed_essays = []
# tqdm is for printing the status bar
for sentance in tqdm(project_data['essay'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e not in stopwords)
    preprocessed_essays.append(sent.lower().strip())
```

```
100%|██████████████████████████████████████████████████████████████████████████████| 1
09248/109248 [01:34<00:00, 1150.35it/s]
```

In [17]:
```
# after preprocesing

print( len(preprocessed_essays[20000].split()))
```

```
126
```

In [18]:
```
# count words in each combined essay
word_count_essays = []
for sentance in tqdm(preprocessed_essays):
     count = len(sentance.split())
     word_count_essays.append(count)

print(word_count_essays[20000])
```

```
100%|██████████████████████████████████████████████████████████████████████████████| 10
9248/109248 [00:02<00:00, 53442.15it/s]

126
```

1.4 Preprocessing of `project_title`

In [19]:
```
# 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 [00:07<00:00, 13740.78it/s]
```

In [20]:
```
print(preprocessed_titles[20000])
```

```
we need to move it while we input it
```

In [21]:
```
#count words in title

word_count_titles = []
for sentance in tqdm(preprocessed_titles):
     count = len(sentance.split())
     word_count_titles.append(count)

print(word_count_titles[20000])
```

```
100%|██████████████████████████████████████████████████████████████████████████████| 109
248/109248 [00:00<00:00, 349547.20it/s]

9
```

1.4.1 Add preprocessed data to dataframe

In [22]:
```
preprocessed_title = pd.DataFrame({'preprocessed_titles': preprocessed_titles})
preprocessed_essay = pd.DataFrame({'preprocessed_essays': preprocessed_essays})
word_count_title = pd.DataFrame({'word_count_titles': word_count_titles})
word_count_essay = pd.DataFrame({'word_count_essays': word_count_essays})
```

In [23]:
```
project_data = pd.concat([project_data, preprocessed_title,preprocessed_essay, \
                          word_count_title, word_count_essay], axis=1)

project_data.head(1)
```

Out[23]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_dat
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	2016-12-05 13:

1 rows × 22 columns

Calculate sentiment score of each essay

In [24]:
```
import nltk
nltk.downloader.download('vader_lexicon')
from nltk.sentiment.vader import SentimentIntensityAnalyzer

# import nltk
# nltk.download('vader_lexicon')

sid = SentimentIntensityAnalyzer()

sentiment_score_essays_neg = []
sentiment_score_essays_neu = []
sentiment_score_essays_pos = []
sentiment_score_essays_com = []

for sentance in tqdm(preprocessed_essays):
    for_sentiment = sentance
    ss = sid.polarity_scores(for_sentiment)
    sentiment_score_essays_neg.append(ss['neg'])
    sentiment_score_essays_neu.append(ss['neu'])
    sentiment_score_essays_pos.append(ss['pos'])
    sentiment_score_essays_com.append(ss['compound'])



for k in ss:
    print('{0}: {1}, '.format(k, ss[k]), end='')

# we can use these 4 things as features/attributes (neg, neu, pos, compound)
# neg: 0.0, neu: 0.753, pos: 0.247, compound: 0.93
```

```
[nltk_data] Downloading package vader_lexicon to
[nltk_data]     C:\Users\wwang26\AppData\Roaming\nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!
100%|██████████████████████████████████████████████████████████████████████████████|
109248/109248 [10:54<00:00, 106.91it/s]

neg: 0.059, neu: 0.693, pos: 0.248, compound: 0.9868,
```

In [25]:
```
print(sentiment_score_essays_neg[10])
print(sentiment_score_essays_neu[10])
print(sentiment_score_essays_pos[10])
print(sentiment_score_essays_com[10])
```

```
0.111
0.619
0.269
0.9842
```

In [26]:
```
ss_neg = pd.DataFrame({'sentiment_score_essays_neg': sentiment_score_essays_neg})
ss_neu = pd.DataFrame({'sentiment_score_essays_neu': sentiment_score_essays_neu})
ss_pos = pd.DataFrame({'sentiment_score_essays_pos': sentiment_score_essays_pos})
ss_com = pd.DataFrame({'sentiment_score_essays_com': sentiment_score_essays_com})
```

In [27]:
```
project_data = pd.concat([project_data,ss_neg, ss_neu, ss_pos, ss_com], axis=1)

project_data.head(1)
```

Out[27]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_dat
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	2016-12-05 13:

1 rows × 26 columns

In [ ]:
```

```

1.5 Preparing data for models

1.5.1 Merge project data with resource data

In [28]:
```
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 [29]:
```
project_data.columns
```

Out[29]:
```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'project_submitted_datetime', 'project_grade_category', 'project_title',
       'project_essay_1', 'project_essay_2', 'project_essay_3',
       'project_essay_4', 'project_resource_summary',
       'teacher_number_of_previously_posted_projects', 'project_is_approved',
       'clean_categories', 'clean_subcategories', 'essay',
       'preprocessed_titles', 'preprocessed_essays', 'word_count_titles',
       'word_count_essays', 'sentiment_score_essays_neg',
       'sentiment_score_essays_neu', 'sentiment_score_essays_pos',
       'sentiment_score_essays_com', 'price', 'quantity'],
      dtype='object')
```

In [30]:
```
project_data.head(1)
```

Out[30]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_dat
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	2016-12-05 13:

1 rows × 28 columns

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

Assignment 7: SVM

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

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

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

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

3. **Representation of results**

   - You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure.

   - Once after you found the best hyper parameter, you need to train your model with it, and find the AUC on test data and plot the ROC curve on both train and test.

   - Along with plotting ROC curve, you need to print the confusion matrix (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/confusion-matrix-tpr-fpr-fnr-tnr-1/) with predicted and original labels of test data points. Please visualize your confusion matrices using seaborn heatmaps.

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

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

4. **[Task-2] Apply the Support Vector Machines on these features by finding the best hyper paramter as suggested in step 2 and step 3** (https://seaborn.pydata.org/generated/seaborn.heatmap.html)

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

   - Consider these set of features Set 5 : (https://seaborn.pydata.org/generated/seaborn.heatmap.html)
     - **school_state** : categorical data
     - **clean_categories** : categorical data
     - **clean_subcategories** : categorical data
     - **project_grade_category** :categorical data
     - **teacher_prefix** : categorical data
     - **quantity** : numerical data
     - **teacher_number_of_previously_posted_projects** : numerical data
     - **price** : numerical data
     - **sentiment score's of each of the essay** : numerical data
     - **number of words in the title** : numerical data
     - **number of words in the combine essays** : numerical data
       (https://seaborn.pydata.org/generated/seaborn.heatmap.html)

- **Apply** (https://seaborn.pydata.org/generated/seaborn.heatmap.html)**TruncatedSVD (http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.TruncatedSVD.html) on TfidfVectorizer (https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html) of essay text, choose the number of components (`n_components`) using elbow method (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/pca-code-example-using-non-visualization/)** : numerical data

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

Note: Data Leakage

1. There will be an issue of data-leakage if you vectorize the entire data and then split it into train/cv/test.
2. To avoid the issue of data-leakage, make sure to split your data first and then vectorize it.
3. While vectorizing your data, apply the method fit_transform() on you train data, and apply the method transform() on cv/test data.
4. For more details please go through this link. (https://soundcloud.com/applied-ai-course/leakage-bow-and-tfidf)

2. Support Vector Machines

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

In [31]:
```
#Stratify vs ramdom sampling. oversampling for imbalanced data
#https://stats.stackexchange.com/questions/250273/benefits-of-stratified-vs-rando
m-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)  # thi
s will drop in raw data so would not work

X_train, X_test, y_train, y_test = train_test_split(project_data, project_data['p
roject_is_approved'],
                                                    test_size=0.33, stratify = pr
oject_data['project_is_approved'])

#X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.3
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 [32]:
```
print(X_test.shape)
print(y_test.shape)
print(X_train.shape)
print(y_train.shape)
```

```
(36052, 27)
(36052,)
(73196, 27)
(73196,)
```

2.2 Make Data Model Ready: encoding numerical, categorical features

In [33]:

```
# Encoding of Categorical Features:

# Category:
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(vocabulary=list(sorted_cat_dict.keys()), lowercase=F
alse, binary=True)
categories_one_hot_train = vectorizer.fit_transform(X_train['clean_categories'].v
alues)
#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()), lowerca
se=False, binary=True)
sub_categories_one_hot_train = vectorizer.fit_transform(X_train['clean_subcategor
ies'].values)
#sub_categories_one_hot_cv = vectorizer.transform(X_cv['clean_subcategories'].val
ues)
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_trai
n.shape)


#you can do the similar thing with state, teacher_prefix and project_grade_catego
ry also

vectorizer = CountVectorizer(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)



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


tp_one_hot_train = vectorizer.fit_transform(X_train['teacher_prefix'].apply(lambd
a x: np.str_(x)))
#tp_one_hot_cv = vectorizer.transform(X_cv['teacher_prefix'].apply(lambda x: np.s
tr_(x)))
tp_one_hot_test = vectorizer.transform(X_test['teacher_prefix'].apply(lambda x: n
p.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)

sorted_grade_list = dict(sorted(grade_list.items(), key=lambda kv: kv[1]))
print(sorted_grade_list)

# If not generating the above list and put into vocabulary, the vector will some
 mess up results ['12', 'Grades', 'PreK']

# This is because of space and new lines. Otherwise no need for vocabulary

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

pg_one_hot_train = vectorizer.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  (73196, 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  (73196, 30)
state Shape of matrix after one hot encodig  (73196, 51)
tp Shape of matrix after one hot encodig  (73196, 6)
{'Grades PreK-2': 44225, 'Grades 6-8': 16923, 'Grades 3-5': 37137, 'Grades 9-1
2': 10963}
{'Grades 9-12': 10963, 'Grades 6-8': 16923, 'Grades 3-5': 37137, 'Grades PreK-
2': 44225}
['Grades 9-12', 'Grades 6-8', 'Grades 3-5', 'Grades PreK-2']
pg Shape of matrix after one hot encodig  (73196, 4)
```

In [34]:
```
print("teacher prefix of matrix after one hot encoding ",tp_one_hot_train[0:5])
print("project grade matrix after one hot encoding ", pg_one_hot_train.shape)
print(X_train['project_grade_category'].values)
```

```
teacher prefix of matrix after one hot encoding    (0, 3)        1
  (1, 3)        1
  (2, 3)        1
  (3, 2)        1
  (4, 2)        1
project grade matrix after one hot encoding  (73196, 4)
['Grades 3-5' 'Grades 6-8' 'Grades PreK-2' ... 'Grades PreK-2'
 'Grades 6-8' 'Grades 6-8']
```

In [35]:
```
# Numerical Data


from sklearn import preprocessing

# 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 = preprocessing.StandardScaler()
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(len(price_standardized_train))
```

```
1.912842861379416e-16
1.0
73196
```

In [36]:
```
previous_scalar = preprocessing.StandardScaler()
previous_scalar.fit(X_train['teacher_number_of_previously_posted_projects'].value
s.reshape(-1,1))

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

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



#previous_standardized_cv = previous_scalar.transform(X_cv['teacher_number_of_pre
viously_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])
```

```
-1.917211190674624e-17
1.0
[-0.32585051]
```

In [37]:
```
wc_scalar = preprocessing.StandardScaler()
wc_scalar.fit(X_train['word_count_essays'].values.reshape(-1,1))

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

# Now standardize the data with above maen and variance.
wc_standardized_train = wc_scalar.transform(X_train['word_count_essays'].values.r
eshape(-1, 1))
#wc_standardized_cv = wc_scalar.transform(X_cv['word_count_essays'].values.reshap
e(-1, 1))
wc_standardized_test = wc_scalar.transform(X_test['word_count_essays'].values.res
hape(-1, 1))


print(wc_standardized_train.mean())
print(wc_standardized_train.std())

print(wc_standardized_train[100])
```

```
1.7982955598606286e-16
0.9999999999999999
[0.01774369]
```

In [38]:
```
wc_title_scalar = preprocessing.StandardScaler()
wc_title_scalar.fit(X_train['word_count_titles'].values.reshape(-1,1))

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

# Now standardize the data with above maen and variance.
wc_title_standardized_train = wc_title_scalar.transform(X_train['word_count_title
s'].values.reshape(-1, 1))
#wc_title_standardized_cv = wc_title_scalar.transform(X_cv['word_count_titles'].v
alues.reshape(-1, 1))
wc_title_standardized_test = wc_title_scalar.transform(X_test['word_count_titles'
].values.reshape(-1, 1))


print(wc_title_standardized_train.mean())
print(wc_title_standardized_train.std())

print(wc_title_standardized_train[100])
```

```
-1.9356552476988358e-16
0.9999999999999999
[1.49647395]
```

In [39]:
```
quantity_scalar = preprocessing.StandardScaler()
quantity_scalar.fit(X_train['quantity'].values.reshape(-1,1))

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

# Now standardize the data with above maen and variance.
quantity_standardized_train = quantity_scalar.transform(X_train['quantity'].value
s.reshape(-1, 1))
#quantity_standardized_cv = quantity_scalar.transform(X_cv['quantity'].values.res
hape(-1, 1))
quantity_standardized_test = quantity_scalar.transform(X_test['quantity'].values.
reshape(-1, 1))


print(quantity_standardized_train.mean())
print(quantity_standardized_train.std())

print(quantity_standardized_train[100])
#print(quantity_standardized_train)
```

```
-1.5531837494072904e-18
0.9999999999999999
[-0.41968553]
```

In [40]:
```
ss_neg_scalar = preprocessing.StandardScaler()
ss_neg_scalar.fit(X_train['sentiment_score_essays_neg'].values.reshape(-1,1))

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

# Now standardize the data with above maen and variance.
ss_neg_standardized_train = ss_neg_scalar.transform(X_train['sentiment_score_essa
ys_neg'].values.reshape(-1, 1))
#ss_neg_standardized_cv = ss_neg_scalar.transform(X_cv['sentiment_score_essays_ne
g'].values.reshape(-1, 1))
ss_neg_standardized_test = ss_neg_scalar.transform(X_test['sentiment_score_essays
_neg'].values.reshape(-1, 1))


print(ss_neg_standardized_train.mean())
print(ss_neg_standardized_train.std())

print(ss_neg_standardized_train[100])
#print(quantity_standardized_train)
```

```
-1.0639308683439939e-16
1.0
[0.09136407]
```

In [41]:
```
ss_neu_scalar = preprocessing.StandardScaler()
ss_neu_scalar.fit(X_train['sentiment_score_essays_neu'].values.reshape(-1,1))

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

# Now standardize the data with above maen and variance.
ss_neu_standardized_train = ss_neu_scalar.transform(X_train['sentiment_score_essa
ys_neu'].values.reshape(-1, 1))
#ss_neu_standardized_cv = ss_neu_scalar.transform(X_cv['sentiment_score_essays_ne
u'].values.reshape(-1, 1))
ss_neu_standardized_test = ss_neu_scalar.transform(X_test['sentiment_score_essays
_neu'].values.reshape(-1, 1))


print(ss_neu_standardized_train.mean())
print(ss_neu_standardized_train.std())

print(ss_neu_standardized_train[100])
#print(quantity_standardized_train)
```

```
8.774517444307811e-16
1.0
[0.6822057]
```

In [42]:
```
ss_pos_scalar = preprocessing.StandardScaler()
ss_pos_scalar.fit(X_train['sentiment_score_essays_pos'].values.reshape(-1,1))

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

# Now standardize the data with above maen and variance.
ss_pos_standardized_train = ss_pos_scalar.transform(X_train['sentiment_score_essa
ys_pos'].values.reshape(-1, 1))
#ss_pos_standardized_cv = ss_pos_scalar.transform(X_cv['sentiment_score_essays_po
s'].values.reshape(-1, 1))
ss_pos_standardized_test = ss_pos_scalar.transform(X_test['sentiment_score_essays
_pos'].values.reshape(-1, 1))


print(ss_pos_standardized_train.mean())
print(ss_pos_standardized_train.std())

print(ss_pos_standardized_train[100])
#print(quantity_standardized_train)
```

```
-1.039468224290829e-15
0.9999999999999999
[-0.70914914]
```

In [43]:
```
ss_com_scalar = preprocessing.StandardScaler()
ss_com_scalar.fit(X_train['sentiment_score_essays_com'].values.reshape(-1,1))

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

# Now standardize the data with above maen and variance.
ss_com_standardized_train = ss_com_scalar.transform(X_train['sentiment_score_essa
ys_com'].values.reshape(-1, 1))
#ss_com_standardized_cv = ss_com_scalar.transform(X_cv['sentiment_score_essays_co
m'].values.reshape(-1, 1))
ss_com_standardized_test = ss_com_scalar.transform(X_test['sentiment_score_essays
_com'].values.reshape(-1, 1))


print(ss_com_standardized_train.mean())
print(ss_com_standardized_train.std())

print(ss_com_standardized_train[100])
#print(quantity_standardized_train)
```

```
7.765918747036452e-19
1.0
[0.09227185]
```

2.3 Make Data Model Ready: encoding eassay, and project_title

2.3.1 Bag of words

In [47]:
```
# We are considering only the words which appeared in at least 10 documents(rows
 or projects).
vectorizer = CountVectorizer(ngram_range = (2,2),min_df=10,max_features = 5000)
text_train_bow = vectorizer.fit_transform(X_train['preprocessed_essays'].values)


# should fit_transferm only on train data . Transform on test data
#text_cv_bow = vectorizer.transform(X_cv['preprocessed_essays'].values)
text_test_bow = vectorizer.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  (73196, 5000)
Shape of matrix after one hot encodig  (36052, 5000)
  (0, 3648)     1
  (0, 2439)     1
  (0, 3051)     1
  (0, 1024)     1
  (0, 4996)     1
  (0, 2440)     1
  (0, 3394)     1
  (0, 2582)     1
  (0, 4067)     1
  (0, 4992)     1
  (0, 2915)     1
  (0, 2643)     1
  (0, 1601)     1
  (0, 2403)     1
  (0, 3084)     1
  (0, 863)      1
  (0, 4979)     1
  (0, 4535)     1
  (0, 3143)     1
  (0, 3956)     1
  (0, 14)       2
  (0, 1043)     1
  (0, 3247)     1
  (0, 1937)     1
  (0, 359)      1
  (0, 2598)     1
  (0, 3914)     1
  (0, 2626)     1
  (0, 3897)     2
  (0, 3092)     2
  (0, 90)       1
  (0, 3606)     1
  (0, 387)      1
  (0, 2166)     1
  (0, 2891)     1
  (0, 220)      1
  (0, 3929)     1
  (0, 4964)     1
  (0, 3480)     1
  (0, 4006)     1
  (0, 1016)     1
  (0, 593)      1
  (0, 2566)     1
```

In [64]:
```
# you can vectorize the title also
# before you vectorize the title make sure you preprocess it


vectorizer = CountVectorizer(ngram_range = (2,2),min_df=10,max_features = 5000)
title_train_bow = vectorizer.fit_transform(X_train['preprocessed_titles'].values)
#title_cv_bow = vectorizer.transform(X_cv['preprocessed_titles'].values)
title_test_bow = vectorizer.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  (73196, 2686)
Shape of matrix after one hot encodig  (36052, 2686)
```

2.3.2 TFIDF

In [44]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(ngram_range=(2,2), min_df=10, max_features = 5000)
text_train_tfidf = vectorizer.fit_transform(X_train['preprocessed_essays'].values
)
#text_cv_tfidf = vectorizer.transform(X_cv['preprocessed_essays'].values)
text_test_tfidf = vectorizer.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  (73196, 5000)
Shape of matrix after one hot encodig  (36052, 5000)
```

In [45]:
```
# Similarly you can vectorize for title also
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(ngram_range = (2,2), min_df=10)
title_train_tfidf = vectorizer.fit_transform(X_train['preprocessed_titles'].value
s)
#title_cv_tfidf = vectorizer.transform(X_cv['preprocessed_titles'].values)
title_test_tfidf = vectorizer.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  (73196, 2690)
Shape of matrix after one hot encodig  (36052, 2690)
```

2.3.3 AVG W2V

In [45]:
```
from gensim.models import Word2Vec
from gensim.models import KeyedVectors

# Reading glove vectors in python: https://stackoverflow.com/a/38230349/4084039
def loadGloveModel(gloveFile):
    print ("Loading Glove Model")
    f = open(gloveFile,'r', encoding="utf8")
    model = {}
    for line in tqdm(f):
        splitLine = line.split()
        word = splitLine[0]
        embedding = np.array([float(val) for val in splitLine[1:]])
        model[word] = embedding
    print ("Done.",len(model)," words loaded!")
    return model
model = loadGloveModel('glove.42B.300d.txt')



# Word2Vec does not provide good result if only vectorize by letter, not words
# Need to split training to words first


'''


# Step 1: Getting each word from the sentence
def list_of_words(Sentence):
    return Sentence.split()

list_of_Sentence=list(X_train['preprocessed_essays'].values)
print(len(list_of_Sentence))
words_list_of_each_Sentence=list(map(list_of_words,list_of_Sentence))

Step 2: Apply word2vec

from gensim.models import word2vec
w2v_model = word2vec.Word2Vec(words_list_of_each_Sentence, size=100,workers=2, mi
n_count=0)
#this line of code trains your w2v model on the give list of sentances. Instead o
f train on X_train.values,
need to train individual words in it


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

'''
```

```
Loading Glove Model

1917494it [13:09, 2427.56it/s]

Done. 1917494  words loaded!
```

Out[45]:
```
'\n\n\n# Step 1: Getting each word from the sentence\ndef list_of_words(Sentenc
e):\n    return Sentence.split()\n    \nlist_of_Sentence=list(X_train[\'preproce
ssed_essays\'].values)\nprint(len(list_of_Sentence))\nwords_list_of_each_Sentenc
e=list(map(list_of_words,list_of_Sentence))\n\nStep 2: Apply word2vec\n\nfrom ge
nsim.models import word2vec\nw2v_model = word2vec.Word2Vec(words_list_of_each_Se
ntence, size=100,workers=2, min_count=0)\n#this line of code trains your w2v mod
el on the give list of sentances. Instead of train on X_train.values, \nneed to
train individual words in it\n\n\nglove_words = list(w2v_model.wv.vocab)\nprint
("number of words that occured minimum 5 times ",len(glove_words))\nprint("sampl
e words ", glove_words)\n\n'
```

In [46]:
```
with open('glove_vectors', 'rb') as f:
    w2v_model = pickle.load(f)
    glove_words =  set(model.keys())
```

In [53]:
```
# 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(300) # as word vectors are of zero length, if word2vec then
use 50
    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] # this is for w2v_model from Word2Vec
function
            vector += w2v_model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_train.append(vector)

print(len(avg_w2v_vectors_train))
```

```
100%|██████████████████████████████████████████████████████████████████████████|
73196/73196 [00:56<00:00, 1300.38it/s]

73196
```

In [54]:
```
#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_vector[0:50])
'''
```

Out[54]:
```
'\n\nw2v_model=Word2Vec(X_cv[\'preprocessed_essays\'].values,min_count=5,size=50, workers=4)\n\n\nglove_words = list(w2v_model.wv.vocab)\n\n\nprint("number of words that occured minimum 5 times ",len(glove_words))\nprint("sample words ", glove_vector[0:50])\n'
```

In [55]:
```
'''
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(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            #vector += w2v_model.wv[word] # this is for w2v_model from Word2Vec function
            vector += w2v_model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_cv.append(vector)

print(len(avg_w2v_vectors_cv))
'''
```

Out[55]:
```
"\navg_w2v_vectors_cv = []; # the avg-w2v for each sentence/review is stored in this list\nfor sentence in tqdm(X_cv['preprocessed_essays'].values): # for each review/sentence\n    vector = np.zeros(300) # as word vectors are of zero length\n    cnt_words =0; # num of words with a valid vector in the sentence/review\n    for word in sentence.split(): # for each word in a review/sentence\n        if word in glove_words:\n            #vector += w2v_model.wv[word] # this is for w2v_model from Word2Vec function\n            vector += w2v_model[word]\n            cnt_words += 1\n    if cnt_words != 0:\n        vector /= cnt_words\n    avg_w2v_vectors_cv.append(vector)\n\nprint(len(avg_w2v_vectors_cv))\n"
```

In [56]:
```
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(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            #vector += w2v_model.wv[word] # this is for w2v_model from Word2Vec
 function
            vector += w2v_model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_test.append(vector)

print(len(avg_w2v_vectors_test))
```

```
100%|██████████████████████████████████████████████████████████████████████████████|
36052/36052 [00:24<00:00, 1494.55it/s]

36052
```

In [57]:
```
# Similarly you can vectorize for title also

#w2v_model=Word2Vec(X_train['preprocessed_titles'].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[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(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            #vector += w2v_model.wv[word] # this is for w2v_model from Word2Vec
 function
            vector += w2v_model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_titles_train.append(vector)

print(len(avg_w2v_vectors_titles_train))
```

```
100%|██████████████████████████████████████████████████████████████████████████████|
73196/73196 [00:02<00:00, 24635.45it/s]

73196
```

In [58]:
```
# 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(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
             #vector += w2v_model.wv[word] # this is for w2v_model from Word2Vec
 function
            vector += w2v_model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_titles_cv.append(vector)

print(len(avg_w2v_vectors_titles_cv))
'''
```

Out[58]:
```
"\navg_w2v_vectors_titles_cv = []; # the avg-w2v for each sentence/review is sto
red in this list\nfor sentence in tqdm(X_cv['preprocessed_titles']): # for each
review/sentence\n    vector = np.zeros(300) # as word vectors are of zero length
\n    cnt_words =0; # num of words with a valid vector in the sentence/review\n
for word in sentence.split(): # for each word in a review/sentence\n        if w
ord in glove_words:\n             #vector += w2v_model.wv[word] # this is for w2
v_model from Word2Vec function\n            vector += w2v_model[word]\n
cnt_words += 1\n    if cnt_words != 0:\n        vector /= cnt_words\n    avg_w2v
_vectors_titles_cv.append(vector)\n\nprint(len(avg_w2v_vectors_titles_cv))\n"
```

In [59]:
```
# Similarly you can vectorize for title also

#w2v_model=Word2Vec(X_test['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_test = []; # the avg-w2v for each sentence/review is store
d in this list
for sentence in tqdm(X_test['preprocessed_titles'].values): # for each review/sen
tence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
             #vector += w2v_model.wv[word] # this is for w2v_model from Word2Vec
 function
            vector += w2v_model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_titles_test.append(vector)

print(len(avg_w2v_vectors_titles_test))
```

```
100%|██████████████████████████████████████████████████████████████████████████|
36052/36052 [00:01<00:00, 26521.50it/s]

36052
```

2.3.4 TFIDF WEIGHTED W2V

In [47]:
```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
#w2v_model=Word2Vec(X_train['preprocessed_essays'].values,min_count=5,size=50, wo
rkers=4)
#glove_words = list(w2v_model.wv.vocab)

# Test and train should use same tfidf model
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_tra
in.idf_)))
tfidf_words_train = set(tfidf_model_train.get_feature_names())
```

In [48]:
```
# 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/se
ntence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the 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 = w2v_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_train.append(vector)

print(len(tfidf_w2v_vectors_train))
```

```
100%|██████████████████████████████████████████████████████████████████████████████
| 73196/73196 [05:07<00:00, 237.81it/s]

73196
```

In [62]:
```
#w2v_model=Word2Vec(X_cv['preprocessed_essays'].values,min_count=5,size=50, workers=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 this list

for sentence in tqdm(X_cv['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_cv):
            vec = w2v_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_cv.append(vector)

print(len(tfidf_w2v_vectors_cv))

'''
```

Out[62]:
```
"\ntfidf_model_cv= TfidfVectorizer()\ntfidf_model_cv.fit(X_cv['preprocessed_essays'].values)\n# we are converting a dictionary with word as a key, and the idf as a value\ndictionary = dict(zip(tfidf_model_cv.get_feature_names(), list(tfidf_model_cv.idf_)))\ntfidf_words_cv = set(tfidf_model_cv.get_feature_names())\n\n\n\ntfidf_w2v_vectors_cv = []; # the avg-w2v for each sentence/review is stored in this list\n\nfor sentence in tqdm(X_cv['preprocessed_essays'].values): # for each review/sentence\n    vector = np.zeros(300) # as word vectors are of zero length\n    tf_idf_weight =0; # num of words with a valid vector in the sentence/review\n    for word in sentence.split(): # for each word in a review/sentence\n        if (word in glove_words) and (word in tfidf_words_cv):\n            vec = w2v_model[word] # getting the vector for each word\n            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word)/len(sentence.split())))\n            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf value for each word\n            vector += (vec * tf_idf) # calculating tfidf weighted w2v\n            tf_idf_weight += tf_idf\n    if tf_idf_weight != 0:\n        vector /= tf_idf_weight\n    tfidf_w2v_vectors_cv.append(vector)\n\nprint(len(tfidf_w2v_vectors_cv))\n\n"
```

In [49]:
```
#w2v_model=Word2Vec(X_test['preprocessed_essays'].values,min_count=5,size=50, workers=4)
#glove_words = list(w2v_model.wv.vocab)

# test and train should use same tfidf w2v model
#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_train.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 = w2v_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%|██████████████████████████████████████████████████████████████████████████████████████████████████████████| 36052/36052 [02:20<00:00, 257.19it/s]

36052
```

similarly convert title

In [50]:
```
    # average Word2Vec
# compute average word2vec for each review.

#w2v_model=Word2Vec(X_train['preprocessed_titles'].values,min_count=5,size=50, wo
rkers=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_mod
el_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 sto
red in this list
for sentence in tqdm(X_train['preprocessed_titles'].values): # for each review/se
ntence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the 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 = w2v_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_train.append(vector)

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

```
100%|██████████████████████████████████████████████████████████████████████████|
73196/73196 [00:04<00:00, 18269.60it/s]

73196
300
```

In [51]:

```
    # 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 = w2v_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]))


'''
```

Out[51]:
```
"\n\ntfidf_model_title_cv= TfidfVectorizer()\ntfidf_model_title_cv.fit(X_cv['pre
processed_titles'])\n# we are converting a dictionary with word as a key, and th
e idf as a value\ndictionary = dict(zip(tfidf_model_title_cv.get_feature_names
(), list(tfidf_model_title_cv.idf_)))\ntfidf_words_title_cv = set(tfidf_model_ti
tle_cv.get_feature_names())\n\n\n\n\ntfidf_w2v_vectors_title_cv = []; # the avg-
w2v for each sentence/review is stored in this list\nfor sentence in tqdm(X_cv
['preprocessed_titles']): # for each review/sentence\n    vector = np.zeros(300)
# as word vectors are of zero length\n    tf_idf_weight =0; # num of words with
a valid vector in the sentence/review\n    for word in sentence.split(): # for e
ach word in a review/sentence\n        if (word in glove_words) and (word in tfi
df_words_title_cv):\n            vec = w2v_model[word] # getting the vector for
each word\n            # here we are multiplying idf value(dictionary[word]) and
the tf value((sentence.count(word)/len(sentence.split())))\n            tf_idf =
dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfid
f value for each word\n            vector += (vec * tf_idf) # calculating tfidf
weighted w2v\n            tf_idf_weight += tf_idf\n    if tf_idf_weight != 0:\n
vector /= tf_idf_weight\n    tfidf_w2v_vectors_title_cv.append(vector)\n\nprint
(len(tfidf_w2v_vectors_title_cv))\nprint(len(tfidf_w2v_vectors_title_cv[0]))\n
\n    \n"
```

In [52]:
```
    # average Word2Vec
# compute average word2vec for each review.

#w2v_model=Word2Vec(X_test['preprocessed_titles'].values,min_count=5,size=50, wor
kers=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_mod
el_title_test.idf_)))
tfidf_words_title_test = set(tfidf_model_title_train.get_feature_names())



tfidf_w2v_vectors_title_test = []; # the avg-w2v for each sentence/review is stor
ed in this list
for sentence in tqdm(X_test['preprocessed_titles'].values): # for each review/sen
tence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the 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 = w2v_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_test.append(vector)

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

```
100%|██████████████████████████████████████████████████████████████████████████|
36052/36052 [00:02<00:00, 17947.46it/s]

36052
300
```

In [ ]:

2.4 Appling Support Vector Machines on different kind of featurization as mentioned in the instructions

Apply Support Vector Machines 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

In [0]:
```
# please write all the code with proper documentation, and proper titles for each
subsection
# go through documentations and blogs before you start coding
# first figure out what to do, and then think about how to do.
# reading and understanding error messages will be very much helpfull in debuggin
g your code
# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the reader
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label
```

2.4.1 Applying LR brute force on BOW

In [65]:
```
# 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, sta
te_one_hot_train, pg_one_hot_train, tp_one_hot_train, price_standardized_train, p
revious_standardized_train, \
                      quantity_standardized_train, wc_standardized_train, wc_titl
e_standardized_train,\
                      ss_neg_standardized_train,ss_neu_standardized_train,ss_pos_
standardized_train, ss_com_standardized_train,\
                      title_train_bow, text_train_bow)).tocsr()

X_test_bow= hstack((categories_one_hot_test, sub_categories_one_hot_test, state_o
ne_hot_test, pg_one_hot_test, tp_one_hot_test, price_standardized_test,  previous
_standardized_test, \
                    quantity_standardized_test, wc_standardized_test, wc_title_st
andardized_test,\
                    ss_neg_standardized_test,ss_neu_standardized_test,ss_pos_stan
dardized_test, ss_com_standardized_test,\
                    title_test_bow, text_test_bow)).tocsr()

#X_cv_bow = hstack((categories_one_hot_cv, sub_categories_one_hot_cv, state_one_h
ot_cv, pg_one_hot_cv, tp_one_hot_cv, price_standardized_cv, previous_standardized
_cv,\
#                   quantity_standardized_cv, wc_standardized_cv, wc_title_standa
rdized_cv,\
 #                  ss_neg_standardized_cv,ss_neu_standardized_cv,ss_pos_standard
ized_cv, ss_com_standardized_cv,\
 #                  title_cv_bow, text_cv_bow)).tocsr()
```

In [66]:
```
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)
```

```
(36052, 7795)
(36052,)
(73196, 7795)
(73196,)
```

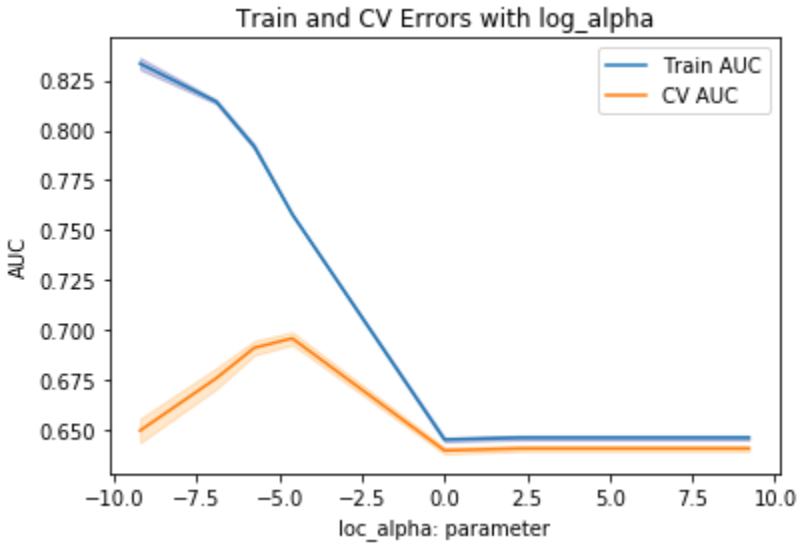
penalty = l2

In [70]:
```
from sklearn.model_selection import GridSearchCV
from sklearn import linear_model
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt

SGD = linear_model.SGDClassifier(loss = 'hinge', penalty = 'l2', learning_rate=
'optimal',class_weight= 'balanced')
parameters = [{'alpha': [10**-4,10**-3, 10**-2.5, 10**-2, 10**0, 10, 10**2, 10**4
]}]
clf = GridSearchCV(SGD, parameters, cv=5, return_train_score = True, scoring='roc
_auc')
clf.fit(X_train_bow, y_train)
```

Out[70]:
```
GridSearchCV(cv=5, error_score='raise-deprecating',
             estimator=SGDClassifier(alpha=0.0001, average=False,
                                     class_weight='balanced',
                                     early_stopping=False, epsilon=0.1,
                                     eta0=0.0, fit_intercept=True,
                                     l1_ratio=0.15, learning_rate='optimal',
                                     loss='hinge', max_iter=1000,
                                     n_iter_no_change=5, n_jobs=None,
                                     penalty='l2', power_t=0.5,
                                     random_state=None, shuffle=True, tol=0.001,
                                     validation_fraction=0.1, verbose=0,
                                     warm_start=False),
             iid='warn', n_jobs=None,
             param_grid=[{'alpha': [0.0001, 0.001, 0.003162277660168379, 0.01,
                                    1, 10, 100, 10000]}],
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='roc_auc', verbose=0)
```

In [71]:
```
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 [74]:
```
import math
alpha = [10**-4,10**-3, 10**-2.5, 10**-2, 10**0, 10, 10**2, 10**4]



log_alpha = []

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


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

plt.plot(log_alpha, cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(log_alpha,cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.
2,color='darkorange')
plt.legend()
plt.xlabel("loc_alpha: parameter")
plt.ylabel("AUC")
plt.title("Train and CV Errors with log_alpha")
plt.show()
```



In [77]:
```
print(log_alpha)
```

```
[-9.210340371976182, -6.907755278982137, -5.756462732485114, -4.605170185988091,
0.0, 2.302585092994046, 4.605170185988092, 9.210340371976184]
```

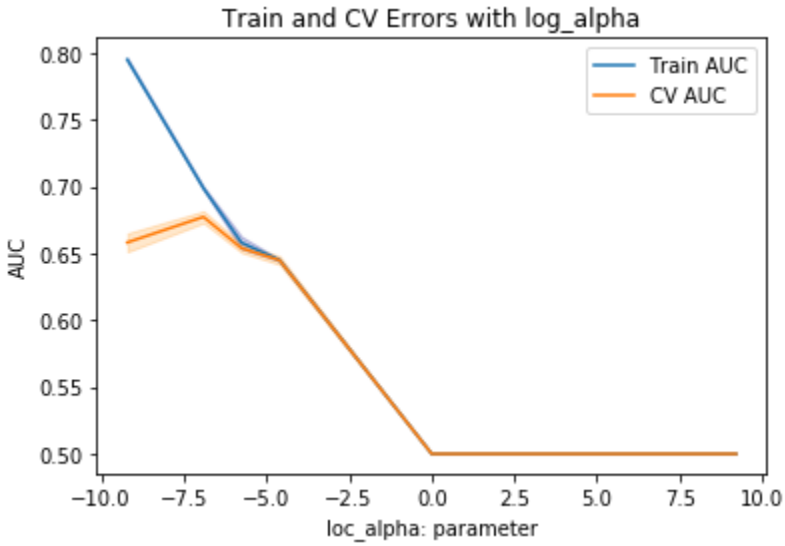
penalty = l1

In [75]:
```
from sklearn.model_selection import GridSearchCV
from sklearn import linear_model
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt

SGD = linear_model.SGDClassifier(loss = 'hinge', penalty = 'l1', learning_rate=
'optimal',class_weight= 'balanced')
parameters = [{'alpha': [10**-4,10**-3, 10**-2.5, 10**-2, 10**0, 10, 10**2, 10**4
]}]
clf = GridSearchCV(SGD, parameters, cv=5, return_train_score = True, scoring='roc
_auc')
clf.fit(X_train_bow, y_train)
```

Out[75]:
```
GridSearchCV(cv=5, error_score='raise-deprecating',
             estimator=SGDClassifier(alpha=0.0001, average=False,
                                     class_weight='balanced',
                                     early_stopping=False, epsilon=0.1,
                                     eta0=0.0, fit_intercept=True,
                                     l1_ratio=0.15, learning_rate='optimal',
                                     loss='hinge', max_iter=1000,
                                     n_iter_no_change=5, n_jobs=None,
                                     penalty='l1', power_t=0.5,
                                     random_state=None, shuffle=True, tol=0.001,
                                     validation_fraction=0.1, verbose=0,
                                     warm_start=False),
             iid='warn', n_jobs=None,
             param_grid=[{'alpha': [0.0001, 0.001, 0.003162277660168379, 0.01,
                                    1, 10, 100, 10000]}],
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='roc_auc', verbose=0)
```

In [76]:
```
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']
```

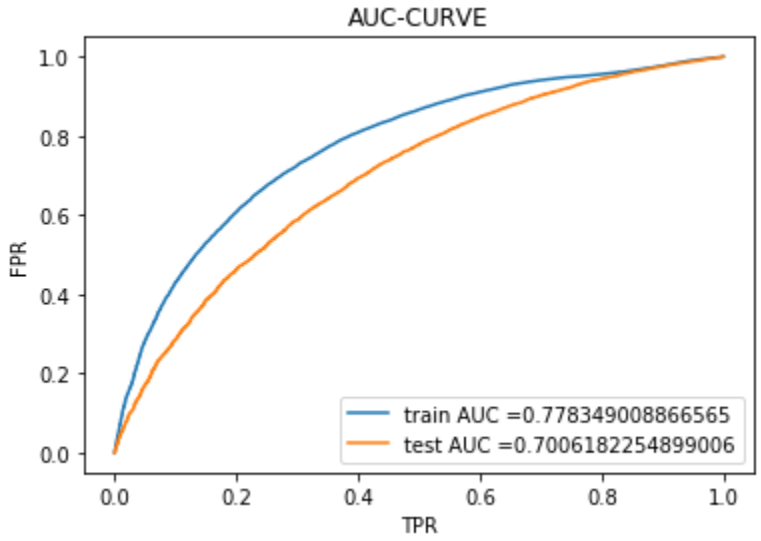
In [78]:
```
import math
alpha = [10**-4,10**-3, 10**-2.5, 10**-2, 10**0, 10, 10**2, 10**4]



log_alpha = []

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


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

plt.plot(log_alpha, cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(log_alpha,cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.
2,color='darkorange')
plt.legend()
plt.xlabel("loc_alpha: parameter")
plt.ylabel("AUC")
plt.title("Train and CV Errors with log_alpha")
plt.show()
```



In [83]:
```
best_c = 10** -2.5
# l2 loss is better
```

In [88]:
```
SGD = linear_model.SGDClassifier(class_weight= 'balanced', alpha = best_c, penalt
y= 'l2')
print(SGD)
clf = SGD.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

#hinge loss can not use predict_proba
y_train_pred = clf.decision_function(X_train_bow)
y_test_pred = clf.decision_function(X_test_bow)

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


print("="*100)
```

```
SGDClassifier(alpha=0.0031622776601683794, average=False,
              class_weight='balanced', early_stopping=False, epsilon=0.1,
              eta0=0.0, fit_intercept=True, l1_ratio=0.15,
              learning_rate='optimal', loss='hinge', max_iter=1000,
              n_iter_no_change=5, n_jobs=None, penalty='l2', power_t=0.5,
              random_state=None, shuffle=True, tol=0.001,
              validation_fraction=0.1, verbose=0, warm_start=False)
```




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

In [90]:
```
print(y_train_pred.shape)
print(y_test.shape)
print(X_train_bow.shape)
```

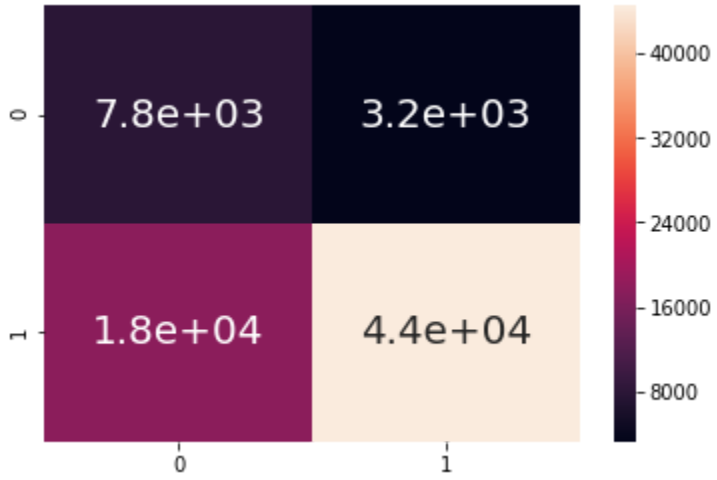
```
(73196,)
(36052,)
(73196, 7795)
```

In [64]:
```
# 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)))
```

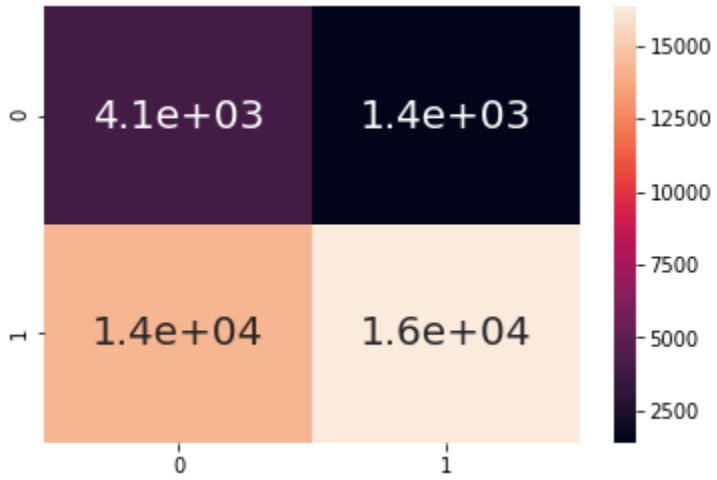
In [93]:
```
print("Train confusion matrix")
cm_train = pd.DataFrame(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_tpr)))
print(cm_train)

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) 0.5090601615144129 for threshold -0.118
       0      1
0   7846   3237
1  17614  44499
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.4173593798690831 for threshold 0.23
       0      1
0   4060   1399
1  14262  16331
```

In [94]:
```
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[94]: <matplotlib.axes._subplots.AxesSubplot at 0x26e8a46de10>



2.4.2 Applying LR brute force on TFIDF

In [95]:
```
# Please write all the code with proper documentation


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, \
                        quantity_standardized_train, wc_standardized_train, wc_titl
e_standardized_train,\
                        ss_neg_standardized_train,ss_neu_standardized_train,ss_pos_
standardized_train, ss_com_standardized_train,\
                        title_train_tfidf, text_train_tfidf)).tocsr()

X_test_tfidf= 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,  previo
us_standardized_test, \
                      quantity_standardized_test, wc_standardized_test, wc_title_st
andardized_test,\
                      ss_neg_standardized_test,ss_neu_standardized_test,ss_pos_stan
dardized_test, ss_com_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_standardiz
ed_cv,\
 #                     quantity_standardized_cv, wc_standardized_cv, wc_title_standa
rdized_cv,\
  #                    ss_neg_standardized_cv,ss_neu_standardized_cv,ss_pos_standard
ized_cv, ss_com_standardized_cv,\
   #                   title_cv_tfidf, text_cv_tfidf)).tocsr()
```

In [96]:
```
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)
```

```
(36052, 7795)
(36052,)
(73196, 7795)
(73196,)
```

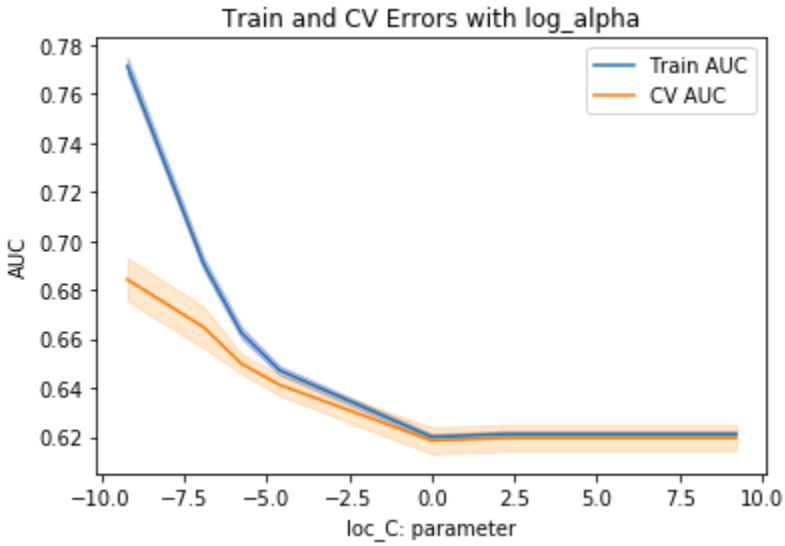
penalty = l2

In [72]:
```
from sklearn.model_selection import GridSearchCV
from sklearn import linear_model
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt

SGD = linear_model.SGDClassifier(loss = 'hinge', penalty = 'l2', learning_rate=
'optimal',class_weight= 'balanced')
parameters = [{'alpha': [10**-4,10**-3, 10**-2.5, 10**-2, 10**0, 10, 10**2, 10**4
]}]
clf = GridSearchCV(SGD, parameters, cv=5, return_train_score = True, scoring='roc
_auc')
clf.fit(X_train_tfidf, y_train)
```

Out[72]:
```
GridSearchCV(cv=5, error_score='raise-deprecating',
             estimator=SGDClassifier(alpha=0.0001, average=False,
                                     class_weight='balanced',
                                     early_stopping=False, epsilon=0.1,
                                     eta0=0.0, fit_intercept=True,
                                     l1_ratio=0.15, learning_rate='optimal',
                                     loss='hinge', max_iter=1000,
                                     n_iter_no_change=5, n_jobs=None,
                                     penalty='l2', power_t=0.5,
                                     random_state=None, shuffle=True, tol=0.001,
                                     validation_fraction=0.1, verbose=0,
                                     warm_start=False),
             iid='warn', n_jobs=None,
             param_grid=[{'alpha': [0.0001, 0.001, 0.0031622776601683794, 0.01,
                                    1, 10, 100, 10000]}],
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='roc_auc', verbose=0)
```

In [73]:
```
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 [74]:
```
import math
alpha = [10**-4, 10**-3, 10**-2.5, 10**-2, 10**0,10, 10**2, 10**4]



log_alpha = []

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


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

plt.plot(log_alpha, cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(log_alpha,cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.
2,color='darkorange')
plt.legend()
plt.xlabel("loc_C: parameter")
plt.ylabel("AUC")
plt.title("Train and CV Errors with log_alpha")
plt.show()
```



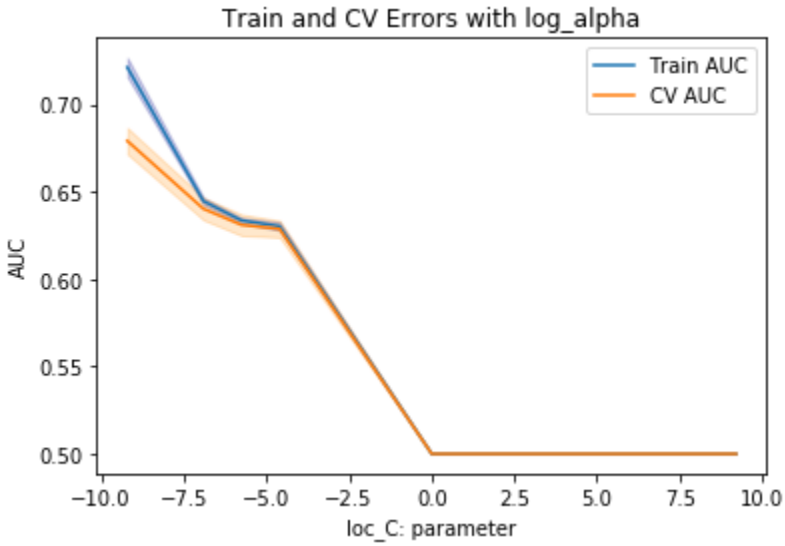
penalty = l1

In [75]:
```
from sklearn.model_selection import GridSearchCV
from sklearn import linear_model
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt

SGD = linear_model.SGDClassifier(loss = 'hinge', penalty = 'l1', learning_rate=
'optimal',class_weight= 'balanced')
parameters = [{'alpha': [10**-4,10**-3, 10**-2.5, 10**-2, 10**0, 10, 10**2, 10**4
]}]
clf = GridSearchCV(SGD, parameters, cv=5, return_train_score = True, scoring='roc
_auc')
clf.fit(X_train_tfidf, y_train)
```

Out[75]:
```
GridSearchCV(cv=5, error_score='raise-deprecating',
             estimator=SGDClassifier(alpha=0.0001, average=False,
                                     class_weight='balanced',
                                     early_stopping=False, epsilon=0.1,
                                     eta0=0.0, fit_intercept=True,
                                     l1_ratio=0.15, learning_rate='optimal',
                                     loss='hinge', max_iter=1000,
                                     n_iter_no_change=5, n_jobs=None,
                                     penalty='l1', power_t=0.5,
                                     random_state=None, shuffle=True, tol=0.001,
                                     validation_fraction=0.1, verbose=0,
                                     warm_start=False),
             iid='warn', n_jobs=None,
             param_grid=[{'alpha': [0.0001, 0.001, 0.003162277660168379, 0.01,
                                    1, 10, 100, 10000]}],
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='roc_auc', verbose=0)
```

In [76]:
```
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']
```

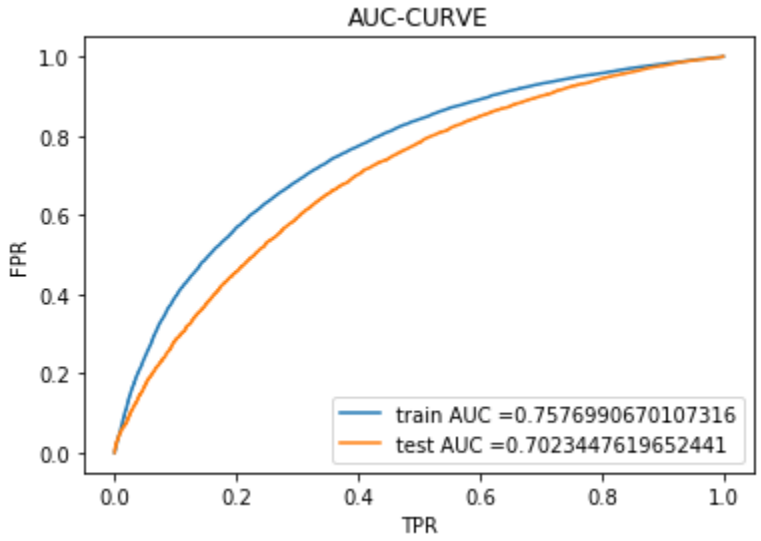
In [77]:
```
import math
alpha = [10**-4, 10**-3, 10**-2.5, 10**-2, 10**0,10, 10**2, 10**4]



log_alpha = []

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


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

plt.plot(log_alpha, cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(log_alpha,cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.
2,color='darkorange')
plt.legend()
plt.xlabel("loc_C: parameter")
plt.ylabel("AUC")
plt.title("Train and CV Errors with log_alpha")
plt.show()
```



In [78]:
```
best_c = 10**-4
```

In [79]:
```
SGD = linear_model.SGDClassifier(class_weight= 'balanced', alpha = best_c, penalt
y= 'l2')
print(SGD)
clf = SGD.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



# DO not use predict for ROC curve
# https://discuss.analyticsvidhya.com/t/what-is-the-difference-between-predict-an
d-predict-proba/67376/2

#hinge loss can not use predict_proba
y_train_pred = clf.decision_function(X_train_tfidf)
y_test_pred = clf.decision_function(X_test_tfidf)

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


print("="*100)
```

```
SGDClassifier(alpha=0.0001, average=False, class_weight='balanced',
              early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
              l1_ratio=0.15, learning_rate='optimal', loss='hinge',
              max_iter=1000, n_iter_no_change=5, n_jobs=None, penalty='l2',
              power_t=0.5, random_state=None, shuffle=True, tol=0.001,
              validation_fraction=0.1, verbose=0, warm_start=False)
```

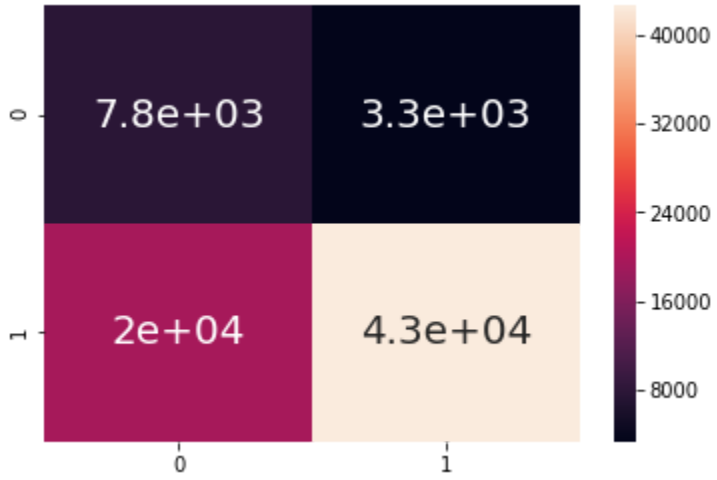



```
==========================================================================================
====================
```

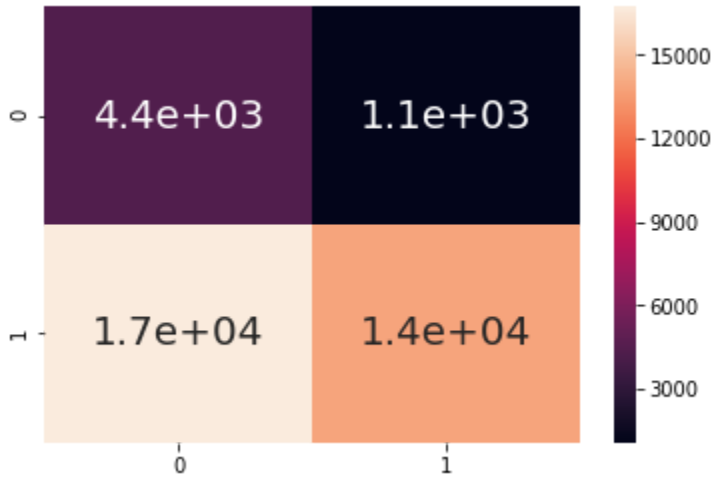
In [80]:
```
print("Train confusion matrix")
cm_train = pd.DataFrame(confusion_matrix(y_train, predict(y_train_pred, tr_thresh
olds, train_fpr, train_tpr)))
print(cm_train)

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

```
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.4812230099687669 for threshold 0.086
       0      1
0   7755   3328
1  19518  42595
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.4251218961013936 for threshold 0.562
       0      1
0   4383   1076
1  16731  13862
```

In [81]:
```
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[81]: <matplotlib.axes._subplots.AxesSubplot at 0x13306c9dfd0>



2.4.3 Applying LR brute force on AVG W2V

In [133]:
```
# Please write all the code with proper documentation


from scipy.sparse import hstack

X_train_w2v = hstack((categories_one_hot_train, sub_categories_one_hot_train, sta
te_one_hot_train, pg_one_hot_train, tp_one_hot_train, price_standardized_train, p
revious_standardized_train, \
                      quantity_standardized_train, wc_standardized_train, wc_titl
e_standardized_train,\
                      ss_neg_standardized_train,ss_neu_standardized_train,ss_pos_
standardized_train, ss_com_standardized_train,\
                      avg_w2v_vectors_titles_train, avg_w2v_vectors_train)).tocsr
()

X_test_w2v= hstack((categories_one_hot_test, sub_categories_one_hot_test, state_o
ne_hot_test, pg_one_hot_test, tp_one_hot_test, price_standardized_test,  previous
_standardized_test, \
                    quantity_standardized_test, wc_standardized_test, wc_title_st
andardized_test,\
                    ss_neg_standardized_test,ss_neu_standardized_test,ss_pos_stan
dardized_test, ss_com_standardized_test,\
                    avg_w2v_vectors_titles_test, avg_w2v_vectors_test)).tocsr()

#X_cv_w2v = hstack((categories_one_hot_cv, sub_categories_one_hot_cv, state_one_h
ot_cv, pg_one_hot_cv, tp_one_hot_cv, price_standardized_cv, previous_standardized
_cv,\
 #                  quantity_standardized_cv, wc_standardized_cv, wc_title_standa
rdized_cv,\
  #                 ss_neg_standardized_cv,ss_neu_standardized_cv,ss_pos_standard
ized_cv, ss_com_standardized_cv,\
   #                avg_w2v_vectors_titles_cv, avg_w2v_vectors_cv)).tocsr()
```

In [134]:
```
print(X_test_w2v.shape)
print(y_test.shape)

print(X_train_w2v.shape)
print(y_train.shape)

#print(X_cv_w2v.shape)
#print(y_cv.shape)
```

```
(36052, 709)
(36052,)
(73196, 709)
(73196,)
```

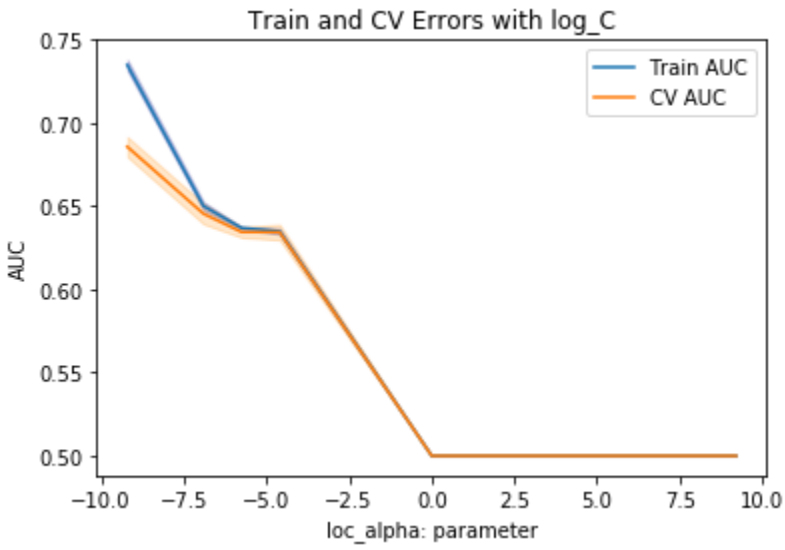
penalty = l2

In [135]:
```
from sklearn.model_selection import GridSearchCV
from sklearn import linear_model
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt

SGD = linear_model.SGDClassifier(loss = 'hinge', penalty = 'l2', learning_rate=
'optimal',class_weight= 'balanced')
parameters = [{'alpha': [10**-4,10**-3, 10**-2.5, 10**-2, 10**0, 10, 10**2, 10**4
]}]
clf = GridSearchCV(SGD, parameters, cv=5, return_train_score = True, scoring='roc
_auc')
clf.fit(X_train_w2v, y_train)
```

Out[135]:
```
GridSearchCV(cv=5, error_score='raise-deprecating',
             estimator=SGDClassifier(alpha=0.0001, average=False,
                                     class_weight='balanced',
                                     early_stopping=False, epsilon=0.1,
                                     eta0=0.0, fit_intercept=True,
                                     l1_ratio=0.15, learning_rate='optimal',
                                     loss='hinge', max_iter=1000,
                                     n_iter_no_change=5, n_jobs=None,
                                     penalty='l2', power_t=0.5,
                                     random_state=None, shuffle=True, tol=0.001,
                                     validation_fraction=0.1, verbose=0,
                                     warm_start=False),
             iid='warn', n_jobs=None,
             param_grid=[{'alpha': [0.0001, 0.001, 0.0031622776601683794, 0.01,
                                    1, 10, 100, 10000]}],
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='roc_auc', verbose=0)
```

In [113]:
```
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 [137]:
```
import math
alpha = [10**-4, 10**-3, 10**-2.5, 10**-2, 10**0,10, 10**2, 10**4]



log_alpha = []

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


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

plt.plot(log_alpha, cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(log_alpha,cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.
2,color='darkorange')
plt.legend()
plt.xlabel("loc_alpha: parameter")
plt.ylabel("AUC")
plt.title("Train and CV Errors with log_C")
plt.show()
```



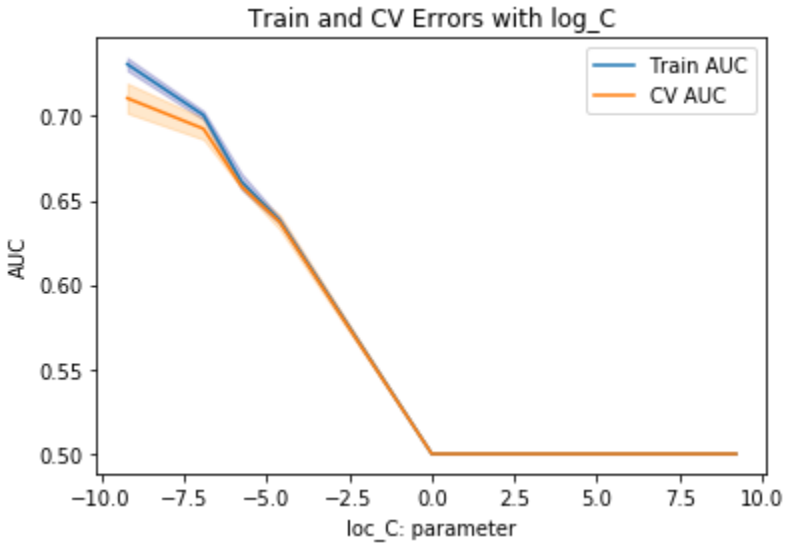
penalty = l1

In [138]:
```
from sklearn.model_selection import GridSearchCV
from sklearn import linear_model
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt

SGD = linear_model.SGDClassifier(loss = 'hinge', penalty = 'l1', learning_rate=
'optimal',class_weight= 'balanced')
parameters = [{'alpha': [10**-4,10**-3, 10**-2.5, 10**-2, 10**0, 10, 10**2, 10**4
]}]
clf = GridSearchCV(SGD, parameters, cv=5, return_train_score = True, scoring='roc
_auc')
clf.fit(X_train_w2v, y_train)
```

Out[138]:
```
GridSearchCV(cv=5, error_score='raise-deprecating',
             estimator=SGDClassifier(alpha=0.0001, average=False,
                                     class_weight='balanced',
                                     early_stopping=False, epsilon=0.1,
                                     eta0=0.0, fit_intercept=True,
                                     l1_ratio=0.15, learning_rate='optimal',
                                     loss='hinge', max_iter=1000,
                                     n_iter_no_change=5, n_jobs=None,
                                     penalty='l1', power_t=0.5,
                                     random_state=None, shuffle=True, tol=0.001,
                                     validation_fraction=0.1, verbose=0,
                                     warm_start=False),
             iid='warn', n_jobs=None,
             param_grid=[{'alpha': [0.0001, 0.001, 0.003162277660168379, 0.01,
                                    1, 10, 100, 10000]}],
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='roc_auc', verbose=0)
```

In [140]:
```
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 [141]:
```
import math
C = [10**-4, 10**-3, 10**-2.5, 10**-2, 10**0,10, 10**2, 10**4]



log_C = []

for i in C:
    j= math.log(i)
    log_C.append(j)


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

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

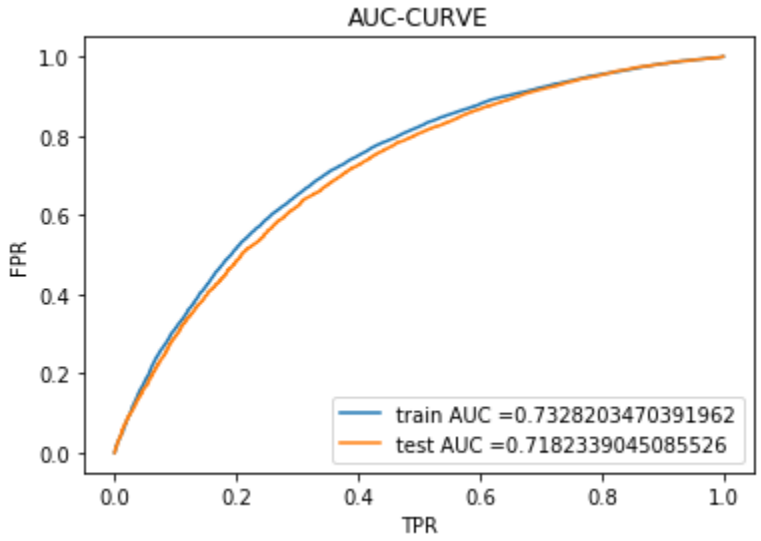


In [144]:
```
best_c = 10** -4
```

In [146]:

```
SGD = linear_model.SGDClassifier(class_weight= 'balanced', alpha = best_c, penalt
y= 'l2')
print(SGD)
clf = SGD.fit(X_train_w2v, 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

#hinge loss can not use predict_proba
y_train_pred = clf.decision_function(X_train_w2v)
y_test_pred = clf.decision_function(X_test_w2v)

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


print("="*100)
```

```
SGDClassifier(alpha=0.0001, average=False, class_weight='balanced',
              early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
              l1_ratio=0.15, learning_rate='optimal', loss='hinge',
              max_iter=1000, n_iter_no_change=5, n_jobs=None, penalty='l2',
              power_t=0.5, random_state=None, shuffle=True, tol=0.001,
              validation_fraction=0.1, verbose=0, warm_start=False)
```

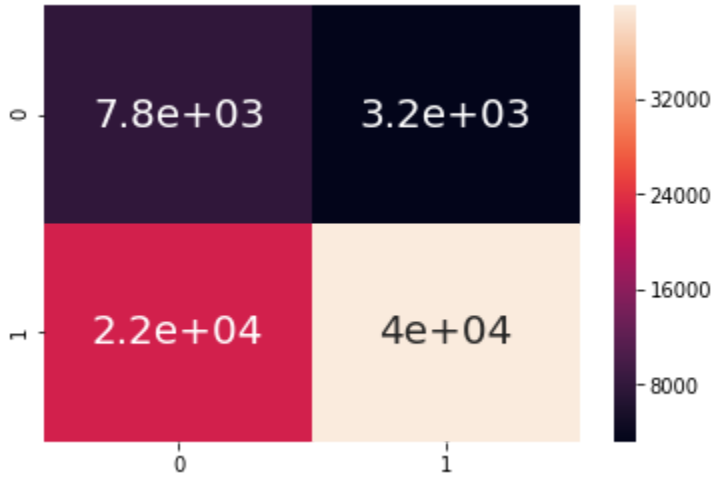



```
==========================================================================================
====================
```

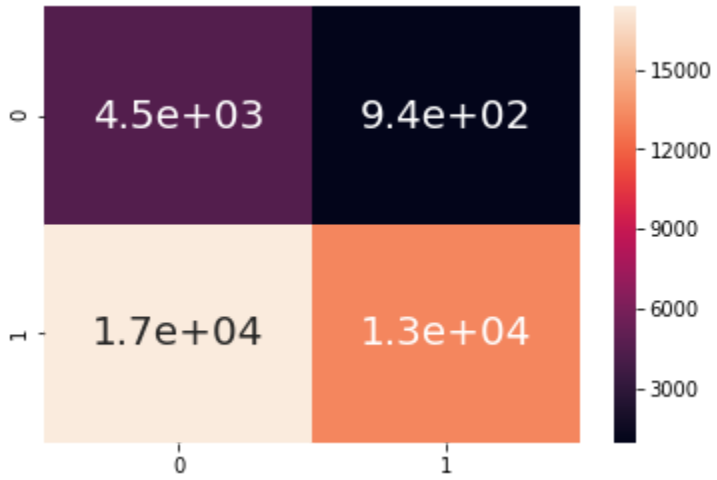
In [148]:
```
print("Train confusion matrix")
cm_train = pd.DataFrame(confusion_matrix(y_train, predict(y_train_pred, tr_thresh
olds, train_fpr, train_tpr)))
print(cm_train)

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

```
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.459791954274779 for threshold 0.24
       0      1
0   7841   3242
1  22318  39795
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.4408227054324315 for threshold 0.74
       0      1
0   4520    939
1  17363  13230
```

In [149]:
```
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[149]: <matplotlib.axes._subplots.AxesSubplot at 0x26e8b673a20>



2.4.4 Applying KNN brute force on TFIDF WEIGHTED W2V

In [53]:
```
# Please write all the code with proper documentation


from scipy.sparse import hstack

X_train_tfidf_w2v = hstack((categories_one_hot_train, sub_categories_one_hot_trai
n, state_one_hot_train, pg_one_hot_train, tp_one_hot_train, price_standardized_tr
ain, previous_standardized_train, \
                      quantity_standardized_train, wc_standardized_train, wc_titl
e_standardized_train,\
                      ss_neg_standardized_train,ss_neu_standardized_train,ss_pos_
standardized_train, ss_com_standardized_train,\
                      tfidf_w2v_vectors_title_train,tfidf_w2v_vectors_train)).toc
sr()

X_test_tfidf_w2v= hstack((categories_one_hot_test, sub_categories_one_hot_test, s
tate_one_hot_test, pg_one_hot_test, tp_one_hot_test, price_standardized_test,  pr
evious_standardized_test, \
                     quantity_standardized_test, wc_standardized_test, wc_title_st
andardized_test,\
                     ss_neg_standardized_test,ss_neu_standardized_test,ss_pos_stan
dardized_test, ss_com_standardized_test,\
                     tfidf_w2v_vectors_title_test,tfidf_w2v_vectors_test)).tocsr()

#X_cv_tfidf_w2v = 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_standa
rdized_cv,\
 #                     quantity_standardized_cv, wc_standardized_cv, wc_title_standa
rdized_cv,\
  #                    ss_neg_standardized_cv,ss_neu_standardized_cv,ss_pos_standard
ized_cv, ss_com_standardized_cv,\
   #                   tfidf_w2v_vectors_title_cv,tfidf_w2v_vectors_cv)).tocsr()
```

In [54]:
```
print(X_test_tfidf_w2v.shape)
print(y_test.shape)

print(X_train_tfidf_w2v.shape)
print(y_train.shape)

#print(X_cv_tfidf_w2v.shape)
#print(y_cv.shape)
```

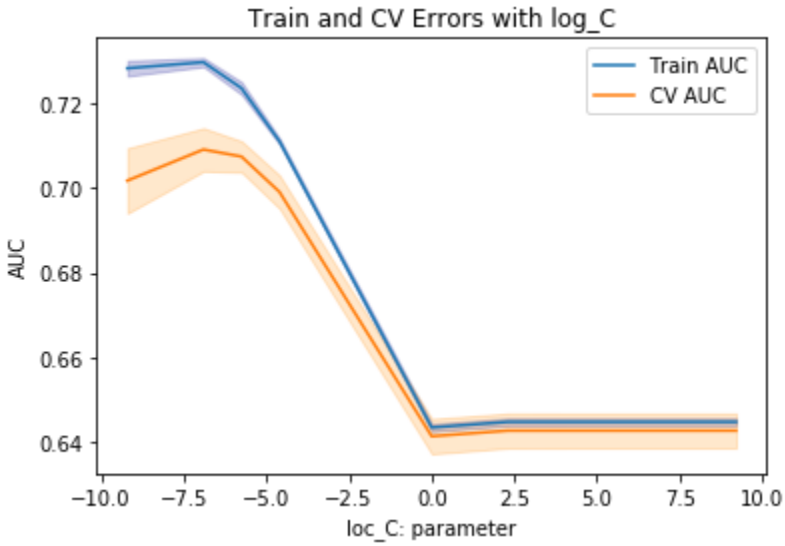
```
(36052, 709)
(36052,)
(73196, 709)
(73196,)
```

In [55]:
```
from sklearn.model_selection import GridSearchCV
from sklearn import linear_model
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt

SGD = linear_model.SGDClassifier(loss = 'hinge', penalty = 'l2', learning_rate=
'optimal',class_weight= 'balanced')
parameters = [{'alpha': [10**-4,10**-3, 10**-2.5, 10**-2, 10**0, 10, 10**2, 10**4
]}]
clf = GridSearchCV(SGD, parameters, cv=5, return_train_score = True, scoring='roc
_auc')
clf.fit(X_train_tfidf_w2v, y_train)
```

Out[55]:
```
GridSearchCV(cv=5, error_score='raise-deprecating',
             estimator=SGDClassifier(alpha=0.0001, average=False,
                                     class_weight='balanced',
                                     early_stopping=False, epsilon=0.1,
                                     eta0=0.0, fit_intercept=True,
                                     l1_ratio=0.15, learning_rate='optimal',
                                     loss='hinge', max_iter=1000,
                                     n_iter_no_change=5, n_jobs=None,
                                     penalty='l2', power_t=0.5,
                                     random_state=None, shuffle=True, tol=0.001,
                                     validation_fraction=0.1, verbose=0,
                                     warm_start=False),
             iid='warn', n_jobs=None,
             param_grid=[{'alpha': [0.0001, 0.001, 0.003162277660168379, 0.01,
                                    1, 10, 100, 10000]}],
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='roc_auc', verbose=0)
```

In [56]:
```
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 [57]:
```
import math
C = [10**-4,10**-3, 10**-2.5, 10**-2, 10**0, 10, 10**2, 10**4]



log_C = []

for i in C:
    j= math.log(i)
    log_C.append(j)


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

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



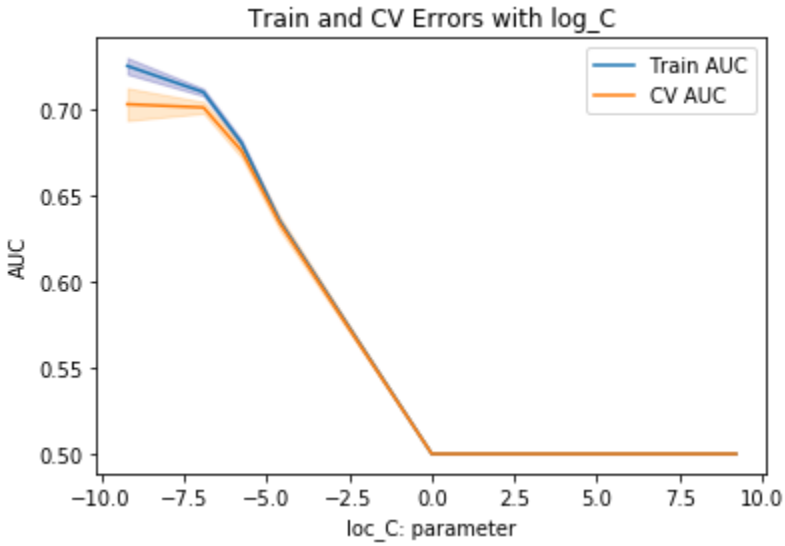
penalty = l1

In [58]:
```
from sklearn.model_selection import GridSearchCV
from sklearn import linear_model
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt

SGD = linear_model.SGDClassifier(loss = 'hinge', penalty = 'l1', learning_rate=
'optimal',class_weight= 'balanced')
parameters = [{'alpha': [10**-4,10**-3, 10**-2.5, 10**-2, 10**0, 10, 10**2, 10**4
]}]
clf = GridSearchCV(SGD, parameters, cv=5, return_train_score = True, scoring='roc
_auc')
clf.fit(X_train_tfidf_w2v, y_train)
```

Out[58]:
```
GridSearchCV(cv=5, error_score='raise-deprecating',
             estimator=SGDClassifier(alpha=0.0001, average=False,
                                     class_weight='balanced',
                                     early_stopping=False, epsilon=0.1,
                                     eta0=0.0, fit_intercept=True,
                                     l1_ratio=0.15, learning_rate='optimal',
                                     loss='hinge', max_iter=1000,
                                     n_iter_no_change=5, n_jobs=None,
                                     penalty='l1', power_t=0.5,
                                     random_state=None, shuffle=True, tol=0.001,
                                     validation_fraction=0.1, verbose=0,
                                     warm_start=False),
             iid='warn', n_jobs=None,
             param_grid=[{'alpha': [0.0001, 0.001, 0.003162277660168379, 0.01,
                                    1, 10, 100, 10000]}],
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='roc_auc', verbose=0)
```

In [59]:
```
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']
```

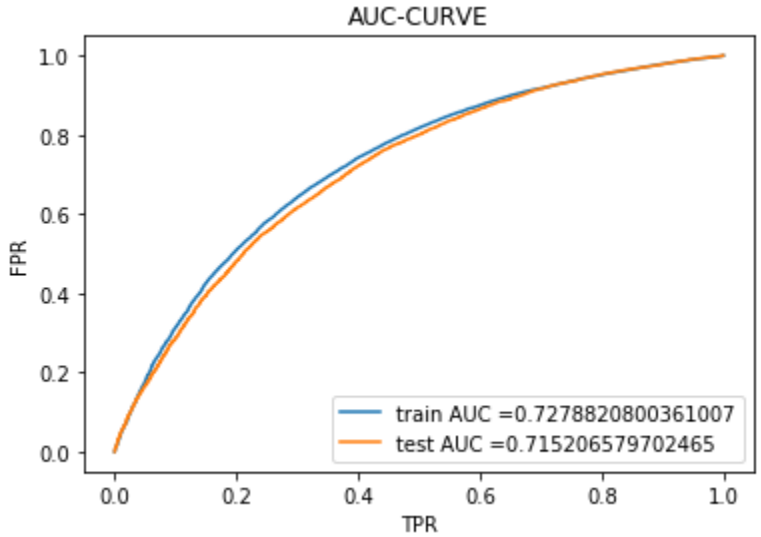
In [60]:
```
import math
C = [10**-4,10**-3, 10**-2.5, 10**-2, 10**0, 10, 10**2, 10**4]



log_C = []

for i in C:
    j= math.log(i)
    log_C.append(j)


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

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



In [61]:
```
best_c = 10** -3
```

In [62]:
```
SGD = linear_model.SGDClassifier(class_weight= 'balanced', alpha = best_c, penalty= 'l2')
print(SGD)
clf = SGD.fit(X_train_tfidf_w2v, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates 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-and-predict-proba/67376/2
y_train_pred = clf.decision_function(X_train_tfidf_w2v)
y_test_pred = clf.decision_function(X_test_tfidf_w2v)

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


print("="*100)
```

```
SGDClassifier(alpha=0.001, average=False, class_weight='balanced',
              early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
              l1_ratio=0.15, learning_rate='optimal', loss='hinge',
              max_iter=1000, n_iter_no_change=5, n_jobs=None, penalty='l2',
              power_t=0.5, random_state=None, shuffle=True, tol=0.001,
              validation_fraction=0.1, verbose=0, warm_start=False)
```

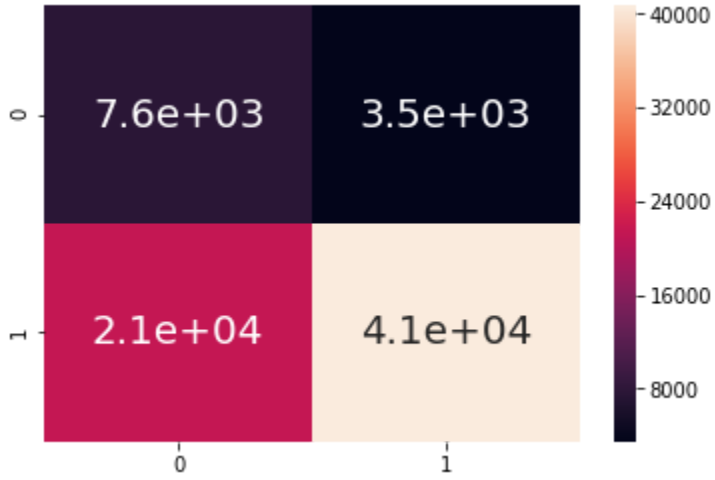



```
====================================================================================================
```

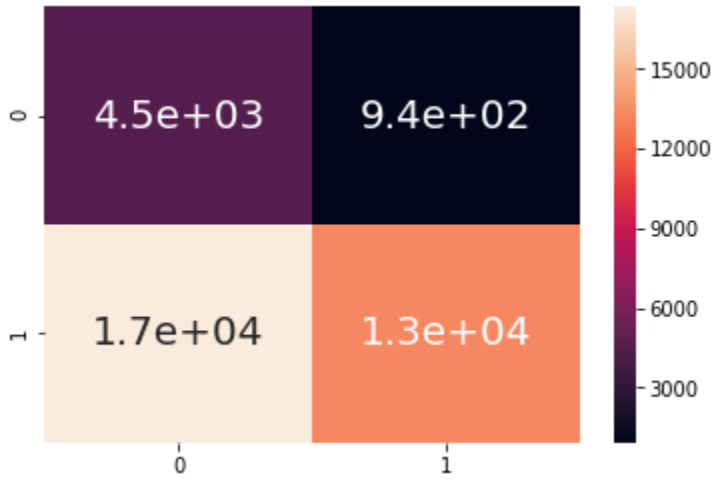
In [65]:
```
print("Train confusion matrix")
cm_train = pd.DataFrame(confusion_matrix(y_train, predict(y_train_pred, tr_thresh
olds, train_fpr, train_tpr)))
print(cm_train)

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

```
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.452418475261982 for threshold 0.071
       0      1
0   7624   3459
1  21436  40677
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.4353688563115551 for threshold 0.546
       0      1
0   4516    943
1  17321  13272
```

In [66]:
```
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[66]: <matplotlib.axes._subplots.AxesSubplot at 0x24d01ee9208>



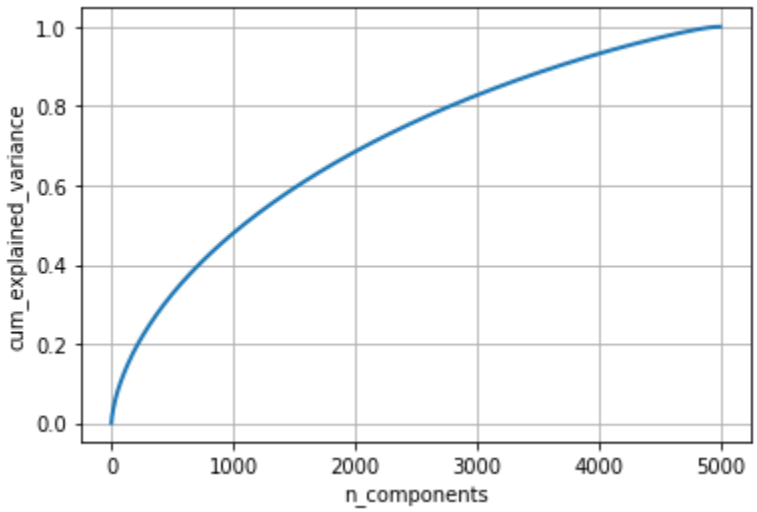
2.5 Support Vector Machines with added Features `Set 5`

In [0]:
```
# please write all the code with proper documentation, and proper titles for each subsection
# go through documentations and blogs before you start coding
# first figure out what to do, and then think about how to do.
# reading and understanding error messages will be very much helpfull in debuggin g your code
# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the reader
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label
```

In [46]:
```
# Apply Trunacated SVD on essay
from sklearn.decomposition import TruncatedSVD

com = [4999]


for i in tqdm(com):
    svd = TruncatedSVD(n_components=i, n_iter=5, random_state=42)
    svd.fit(text_train_tfidf)
    percentage_var_explained = svd.explained_variance_ / np.sum(svd.explained_var
iance_)
    cum_var_explained = np.cumsum(percentage_var_explained)
    #cum_var.append(cum_var_explained)
```

```
  0%|
| 0/1 [00:00<?, ?it/s]

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-46-269ffe3a2742> in <module>
     10     percentage_var_explained = svd.explained_variance_ / np.sum(svd.expl
ained_variance_)
     11     cum_var_explained = np.cumsum(percentage_var_explained)
---> 12     cum_var.append(cum_var_explained)
     13

NameError: name 'cum_var' is not defined
```

In [47]:
```
plt.figure(1,figsize=(6,4))

plt.clf()
plt.plot(cum_var_explained, linewidth =2)
plt.axis('tight')
plt.grid()
plt.xlabel('n_components')
plt.ylabel('cum_explained_variance')
plt.show()
```



Based on the output, we will use all 5109 features in tfidf_train which are necessary to explain 100% variance in the data

In [100]:
```
svd_text_train = svd.transform(text_train_tfidf)
svd_text_test = svd.transform(text_test_tfidf)

print(svd_text_train.shape)
```

```
(73196, 4999)
```

In [111]:
```
print(svd_text_train[1])
```

```
[ 1.47936298e-01 -9.84259294e-03 -3.95473664e-02 ... -1.80945398e-04
  3.12509717e-04  7.10810937e-05]
```

In [ ]:
```
# Please write all the code with proper documentation


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, \
                        quantity_standardized_train, wc_standardized_train, wc_titl
e_standardized_train,\
                        ss_neg_standardized_train,ss_neu_standardized_train,ss_pos_
standardized_train, ss_com_standardized_train,\
                       svd_text_train)).tocsr()

X_test_tfidf= 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,  previo
us_standardized_test, \
                      quantity_standardized_test, wc_standardized_test, wc_title_st
andardized_test,\
                      ss_neg_standardized_test,ss_neu_standardized_test,ss_pos_stan
dardized_test, ss_com_standardized_test,\
                      svd_text_test)).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_standardiz
ed_cv,\
 #                     quantity_standardized_cv, wc_standardized_cv, wc_title_standa
rdized_cv,\
  #                    ss_neg_standardized_cv,ss_neu_standardized_cv,ss_pos_standard
ized_cv, ss_com_standardized_cv,\
   #                   title_cv_tfidf, text_cv_tfidf)).tocsr()
```

In [108]:
```
print(X_test_tfidf.shape)
print(y_test.shape)

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

#print(X_cv_new.shape)
#print(y_cv.shape)
```

```
(36052, 5109)
(36052,)
(73196, 5109)
(73196,)
```

penalty = l2

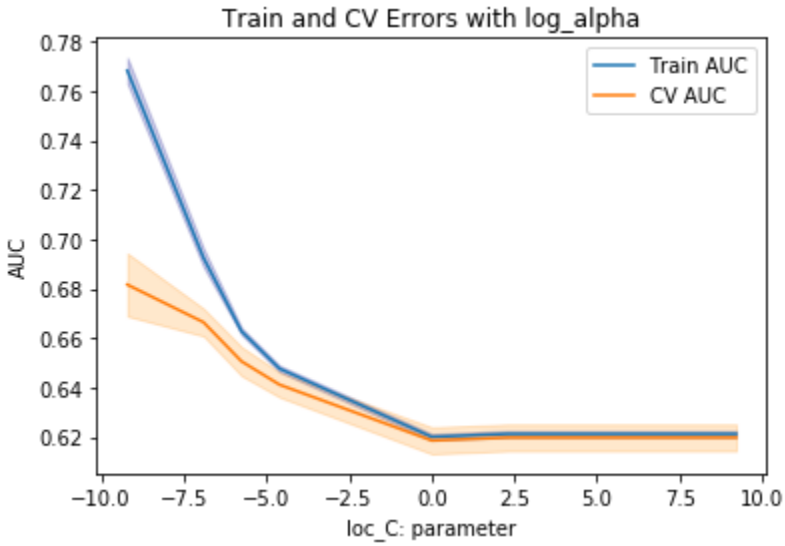
In [82]:
```
from sklearn.model_selection import GridSearchCV
from sklearn import linear_model
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt

SGD = linear_model.SGDClassifier(loss = 'hinge', penalty = 'l2', learning_rate=
'optimal',class_weight= 'balanced')
parameters = [{'alpha': [10**-4,10**-3, 10**-2.5, 10**-2, 10**0, 10, 10**2, 10**4
]}]
clf = GridSearchCV(SGD, parameters, cv=5, return_train_score = True, scoring='roc
_auc')
clf.fit(X_train_tfidf, y_train)
```

Out[82]:
```
GridSearchCV(cv=5, error_score='raise-deprecating',
             estimator=SGDClassifier(alpha=0.0001, average=False,
                                     class_weight='balanced',
                                     early_stopping=False, epsilon=0.1,
                                     eta0=0.0, fit_intercept=True,
                                     l1_ratio=0.15, learning_rate='optimal',
                                     loss='hinge', max_iter=1000,
                                     n_iter_no_change=5, n_jobs=None,
                                     penalty='l2', power_t=0.5,
                                     random_state=None, shuffle=True, tol=0.001,
                                     validation_fraction=0.1, verbose=0,
                                     warm_start=False),
             iid='warn', n_jobs=None,
             param_grid=[{'alpha': [0.0001, 0.001, 0.003162277660168379, 0.01,
                                    1, 10, 100, 10000]}],
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='roc_auc', verbose=0)
```

In [83]:
```
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 [ ]:
```
```

In [84]:
```
import math
alpha = [10**-4, 10**-3, 10**-2.5, 10**-2, 10**0,10, 10**2, 10**4]



log_alpha = []

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


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

plt.plot(log_alpha, cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(log_alpha,cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.
2,color='darkorange')
plt.legend()
plt.xlabel("loc_C: parameter")
plt.ylabel("AUC")
plt.title("Train and CV Errors with log_alpha")
plt.show()
```



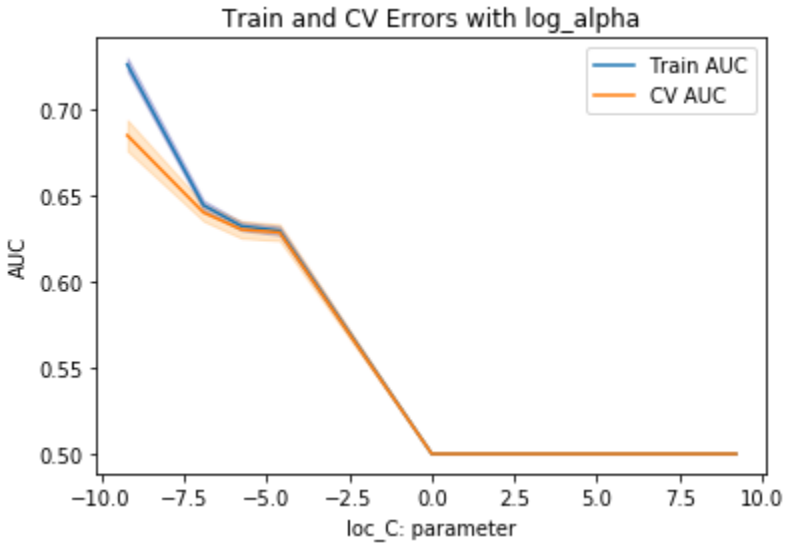
penalty = l1

In [85]:
```
from sklearn.model_selection import GridSearchCV
from sklearn import linear_model
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt

SGD = linear_model.SGDClassifier(loss = 'hinge', penalty = 'l1', learning_rate=
'optimal',class_weight= 'balanced')
parameters = [{'alpha': [10**-4,10**-3, 10**-2.5, 10**-2, 10**0, 10, 10**2, 10**4
]}]
clf = GridSearchCV(SGD, parameters, cv=5, return_train_score = True, scoring='roc
_auc')
clf.fit(X_train_tfidf, y_train)
```

Out[85]:
```
GridSearchCV(cv=5, error_score='raise-deprecating',
             estimator=SGDClassifier(alpha=0.0001, average=False,
                                     class_weight='balanced',
                                     early_stopping=False, epsilon=0.1,
                                     eta0=0.0, fit_intercept=True,
                                     l1_ratio=0.15, learning_rate='optimal',
                                     loss='hinge', max_iter=1000,
                                     n_iter_no_change=5, n_jobs=None,
                                     penalty='l1', power_t=0.5,
                                     random_state=None, shuffle=True, tol=0.001,
                                     validation_fraction=0.1, verbose=0,
                                     warm_start=False),
             iid='warn', n_jobs=None,
             param_grid=[{'alpha': [0.0001, 0.001, 0.003162277660168379, 0.01,
                                    1, 10, 100, 10000]}],
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='roc_auc', verbose=0)
```

In [86]:
```
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']
```

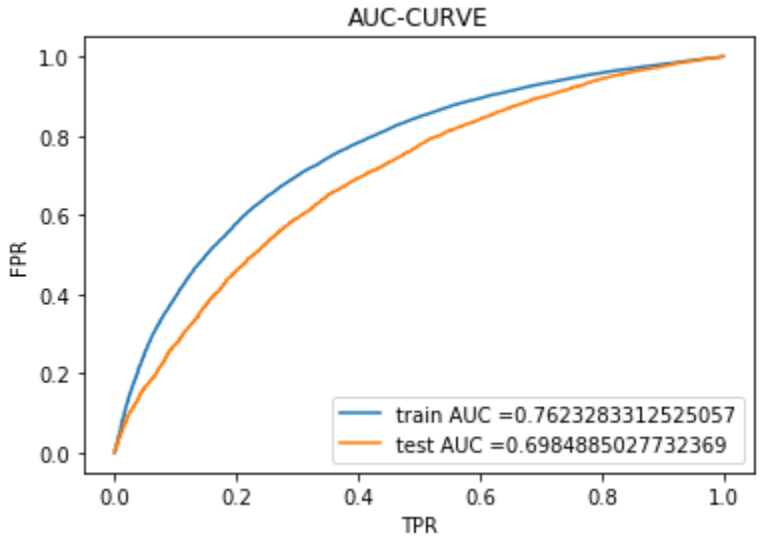
In [87]:
```
import math
alpha = [10**-4, 10**-3, 10**-2.5, 10**-2, 10**0,10, 10**2, 10**4]



log_alpha = []

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


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

plt.plot(log_alpha, cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(log_alpha,cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.
2,color='darkorange')
plt.legend()
plt.xlabel("loc_C: parameter")
plt.ylabel("AUC")
plt.title("Train and CV Errors with log_alpha")
plt.show()
```



In [88]:
```
best_c = 10**-4
```

In [89]:
```
SGD = linear_model.SGDClassifier(class_weight= 'balanced', alpha = best_c, penalt
y= 'l2')
print(SGD)
clf = SGD.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



# DO not use predict for ROC curve
# https://discuss.analyticsvidhya.com/t/what-is-the-difference-between-predict-an
d-predict-proba/67376/2

#hinge loss can not use predict_proba
y_train_pred = clf.decision_function(X_train_tfidf)
y_test_pred = clf.decision_function(X_test_tfidf)

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


print("="*100)
```

```
SGDClassifier(alpha=0.0001, average=False, class_weight='balanced',
              early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
              l1_ratio=0.15, learning_rate='optimal', loss='hinge',
              max_iter=1000, n_iter_no_change=5, n_jobs=None, penalty='l2',
              power_t=0.5, random_state=None, shuffle=True, tol=0.001,
              validation_fraction=0.1, verbose=0, warm_start=False)
```

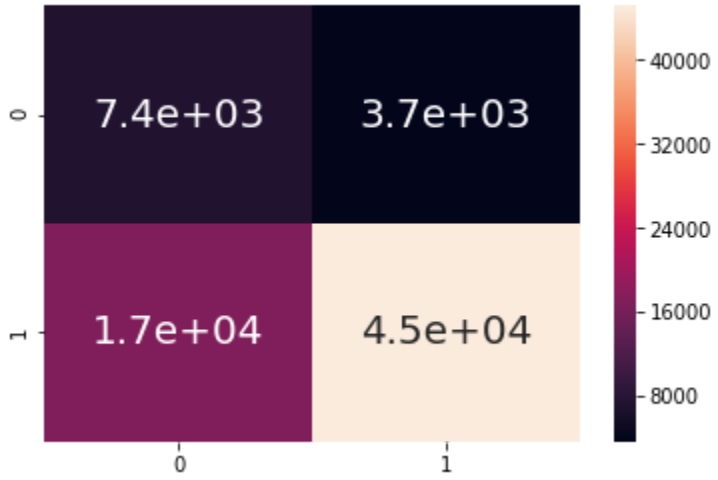



```
==========================================================================================
====================
```

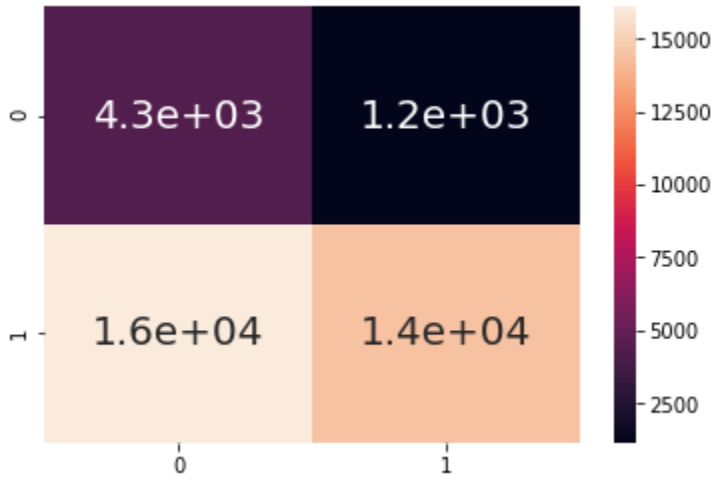
In [90]:
```
print("Train confusion matrix")
cm_train = pd.DataFrame(confusion_matrix(y_train, predict(y_train_pred, tr_thresh
olds, train_fpr, train_tpr)))
print(cm_train)

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

```
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.4899222460255096 for threshold -0.204
       0      1
0   7402   3681
1  16996  45117
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.4227212389368608 for threshold 0.284
       0      1
0   4309   1150
1  16108  14485
```

In [91]:
```
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[91]: <matplotlib.axes._subplots.AxesSubplot at 0x1330fd4b5f8>



3. Conclusion

In [67]:
```
# Please compare all your models using Prettytable library
# 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", "SGDClassifier",10**-2.5, 0.7017])
x.add_row(["TFIDF", "SGDClassifier", 10**-2.5, 0.655])
x.add_row(["AVG W2V", "SGDClassifier", 1, 0.7210])
x.add_row(["TFIDF WEIGHTED WV", "SGDClassifier", 1,0.7152 ])
x.add_row(["SET 5", "SGDClassifier", 10**-3.5, 0.6341])

print(x)
```

```
+-------------------+---------------+-------------------------+--------+
|     Vectorizer    |     Model     |    Hyper Parameter      |  AUC   |
+-------------------+---------------+-------------------------+--------+
|        BOW        | SGDClassifier | 0.0031622776601683794   | 0.7017 |
|       TFIDF       | SGDClassifier | 0.0031622776601683794   | 0.655  |
|      AVG W2V      | SGDClassifier |           1             | 0.721  |
| TFIDF WEIGHTED WV | SGDClassifier |           1             | 0.7152 |
|       SET 5       | SGDClassifier | 0.00031622776601683794  | 0.6341 |
+-------------------+---------------+-------------------------+--------+
```

In [ ]: