

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
<code>project_id</code>		A unique identifier for the proposed project. Example: p036502
<code>project_title</code>	<ul style="list-style-type: none">••	Title of the project. Examples: Art Will Make You Happy! First Grade Fun
<code>project_grade_category</code>	<ul style="list-style-type: none">••••	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
<code>project_subject_categories</code>	<ul style="list-style-type: none">••••••••	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
<code>school_state</code>		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
<code>project_subject_subcategories</code>	<ul style="list-style-type: none">••	One or more (comma-separated) subject subcategories for the project. Examples: Literacy Literature & Writing, Social Sciences
<code>project_resource_summary</code>	<ul style="list-style-type: none">•	An explanation of the resources needed for the project. Example: My students need hands on literacy materials to manage sensory needs!
<code>project_essay_1</code>		First application essay*
<code>project_essay_2</code>		Second application essay*
<code>project_essay_3</code>		Third application essay*
<code>project_essay_4</code>		Fourth application essay*
<code>project_submitted_datetime</code>		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's title. One of the following enumerated values:
	•	nan
	•	Dr.
teacher_prefix	•	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 <code>project_id</code> value from the <code>train.csv</code> file. Example: p036502
description		Description 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
<code>project_is_approved</code>	A binary flag indicating whether DonorsChoose approved the project. A value of <code>0</code> indicates the project was not approved, and a value of <code>1</code> 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
```

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' 'school_state'
'project_submitted_datetime' 'project_grade_category'
'project_subject_categories' 'project_subject_subcategories'
'project_title' 'project_essay_1' 'project_essay_2' 'project_essay_3'
'project_essay_4' 'project_resource_summary'
'teacher_number_of_previously_posted_projects' 'project_is_approved']

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

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

sub_cat_list = []
for i in sub_categories:
    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 category based on space "Math & Science"=> "Math", "&", "Science"
            j=j.replace('The','') # if we have the words "The" we are going to replace it with ''(i.e removing 'The')
            j = j.replace(' ','') # we are placing all the ' '(space) with ''(empty)
            ex:"Math & Science"=>"Math&Science"
            temp +=j.strip()+" #" abc ".strip() will return "abc", remove the trailing spaces
        temp = temp.replace('&','_')
    sub_cat_list.append(temp.strip())

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

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

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

```

1.3 Text preprocessing

```

In [8]: # 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_date
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	2016-12-05 1
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL	2016-10-25 0

```
In [10]: ##### 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 language 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 in a Title I school where most of the students receive free or reduced price lunch. Despite their 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 them because they develop their core, 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 students 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-breaks-python/
sent = sent.replace('\r', ' ')
sent = sent.replace('\n', ' ')
sent = sent.replace('\t', ' ')
print(sent)
```

My kindergarten students have varied disabilities ranging from speech and language 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 Title I school where most of the students receive free or reduced price lunch. Despite their 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 them because they develop their core, which enhances gross motor and in turn fine motor skills. They also want to learn through games, my kids do not want to sit and do worksheets. 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 forget 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 language 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 Title I school where most of the students receive free or reduced price lunch Despite their 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 them because they develop their core which enhances gross motor and in Turn fine motor skills They also want to learn through games my kids do not want to sit and do worksheets 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 forget they are doing work 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', 'itself',
'they', 'them', 'their', \
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that',
"that'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has',
'had', 'having', 'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because',
'as', 'until', 'while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through',
'during', 'before', 'after', \
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off',
'over', 'under', 'again', 'further', \
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all',
'any', 'both', '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', "didn'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', "shouldn't",
'wasn', "wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"]
```



```
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_date
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	2016-12-05 13:

1 rows x 22 columns

Calculate sentiment score of each essay


```
In [35]: project_data = pd.concat([project_data,ss_neg, ss_neu, ss_pos, ss_com], axis=1)
project_data.head(1)
```

```
Out[35]:
```

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_date
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 [36]: 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 [37]: project_data.columns
```

```
Out[37]: 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 [47]: project_data.head(1)
```

```
Out[47]:
```

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_date
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 5: Logistic Regression

1. [Task-1] Logistic Regression(either SGDClassifier with log loss, or LogisticRegression) on these feature sets

- **Set 1:** categorical, numerical features + project_title(BOW) + preprocessed_eassay (BOW with bi-grams` with `min_df=10` and `max_features=5000`)
- **Set 2:** categorical, numerical features + project_title(TFIDF)+ preprocessed_eassay (TFIDF with bi-grams` with `min_df=10` and `max_features=5000`)
- **Set 3:** categorical, numerical features + project_title(AVG W2V)+ preprocessed_eassay (AVG W2V)
- **Set 4:** categorical, numerical features + project_title(TFIDF W2V)+ preprocessed_essay (TFIDF W2V)

2. Hyper paramter tuning (find best hyper parameters corresponding the algorithm that you choose)

- 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/) (<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/) (<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 Logistic Regression on the below feature set **Set 5** by finding the best hyper parameter as suggested in step 2 and step 3.

5. Consider these set of features **Set 5** :

- **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

And apply the Logistic regression 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>)

6. Conclusion (<https://seaborn.pydata.org/generated/seaborn.heatmap.html>)

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

- You need to summarize the results at the end of the notebook, summarize it in the table format. To print out a table please refer to this [prettytable](https://seaborn.pydata.org/generated/seaborn.heatmap.html) library (<https://seaborn.pydata.org/generated/seaborn.heatmap.html>) 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). (<https://soundcloud.com/applied-ai-course/leakage-bow-and-tfidf>)

2. Logistic Regression

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

In [176]:

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

from sklearn.model_selection import train_test_split

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

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

#X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33, stratify=y_train)

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 [178]: 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 [205]: # Encoding of Categorical Features:

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

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

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

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

#you can do the similar thing with state, teacher_prefix and project_grade_category 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 encoding ", state_one_hot_train.shape)

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

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

print("tp Shape of matrix after one hot encoding ", 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 [207]: 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, 2)      1
  (1, 2)      1
  (2, 2)      1
  (3, 2)      1
  (4, 3)      1
project grade matrix after one hot encoding  (73196, 4)
['Grades PreK-2' 'Grades 9-12' 'Grades 3-5' ... 'Grades 3-5' 'Grades 3-5'
 'Grades 3-5']
```

```
In [234]: # 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))
```

```
3.591737420504359e-18
1.0
73196
```

```
In [235]: previous_scalar = preprocessing.StandardScaler()
previous_scalar.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))

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

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

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

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

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

print(previous_standardized_train[100])
```

2.1356276554350243e-17
0.9999999999999999
[-0.07606696]

```
In [236]: 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(previous_scalar.var_[0])}")

# Now standardize the data with above mean and variance.
wc_standardized_train = wc_scalar.transform(X_train['word_count_essays'].values.reshape(-1, 1))
wc_standardized_cv = wc_scalar.transform(X_cv['word_count_essays'].values.reshape(-1, 1))
wc_standardized_test = wc_scalar.transform(X_test['word_count_essays'].values.reshape(-1, 1))

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

print(wc_standardized_train[100])
```

-2.0696173460852144e-16
1.0
[-1.13610293]


```
In [237]: 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(previous_scalar.var_[0])}")

# Now standardize the data with above mean and variance.
wc_title_standardized_train = wc_title_scalar.transform(X_train['word_count_titles'].values.reshape(-1, 1))
#wc_title_standardized_cv = wc_title_scalar.transform(X_cv['word_count_titles'].values.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])

-1.8094590680594934e-16
1.0
[-1.30880399]
```

```
In [238]: 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(previous_scalar.var_[0])}")

# Now standardize the data with above mean and variance.
quantity_standardized_train = quantity_scalar.transform(X_train['quantity'].values.reshape(-1, 1))
#quantity_standardized_cv = quantity_scalar.transform(X_cv['quantity'].values.reshape(-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)

-3.43641904556363e-17
1.0
[0.1880109]
```

```
In [239]: 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(previous_scalar.var_[0])}")

# Now standardize the data with above mean and variance.
ss_neg_standardized_train = ss_neg_scalar.transform(X_train['sentiment_score_essays_neg'].values.reshape(-1, 1))
#ss_neg_standardized_cv = ss_neg_scalar.transform(X_cv['sentiment_score_essays_neg'].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)

-4.060604764856685e-16
1.0
[-0.50421325]
```

```
In [240]: 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(previous_scalar.var_[0])}")

# Now standardize the data with above mean and variance.
ss_neu_standardized_train = ss_neu_scalar.transform(X_train['sentiment_score_essays_neu'].values.reshape(-1, 1))
#ss_neu_standardized_cv = ss_neu_scalar.transform(X_cv['sentiment_score_essays_neu'].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)

1.7211217423119537e-16
1.0000000000000002
[-0.47531255]
```

```
In [241]: 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(previous_scalar.var_[0])}")

# Now standardize the data with above mean and variance.
ss_pos_standardized_train = ss_pos_scalar.transform(X_train['sentiment_score_essays_pos'].values.reshape(-1, 1))
#ss_pos_standardized_cv = ss_pos_scalar.transform(X_cv['sentiment_score_essays_pos'].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)

5.805266948370593e-16
1.0
[0.69606184]
```

```
In [242]: 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(previous_scalar.var_[0])}")

# Now standardize the data with above mean and variance.
ss_com_standardized_train = ss_com_scalar.transform(X_train['sentiment_score_essays_com'].values.reshape(-1, 1))
#ss_com_standardized_cv = ss_com_scalar.transform(X_cv['sentiment_score_essays_com'].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)

2.186591497212451e-16
0.9999999999999998
[0.16235604]
```

2.3 Make Data Model Ready: encoding eassay, and project_title

2.3.1 Bag of words

```
In [216]: # 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)
Shape of matrix after one hot encodig (24155, 5000)
(0, 2708) 1
(0, 2007) 1
(0, 4466) 1
(0, 2276) 1
(0, 4629) 1
(0, 467) 1
(0, 1354) 1
(0, 4916) 1
(0, 4920) 1
(0, 726) 1
(0, 516) 1
(0, 2233) 2
(0, 909) 1
(0, 4072) 2
(0, 4528) 1
(0, 4593) 1
(0, 4046) 1
(0, 197) 1
(0, 3136) 1
(0, 4684) 2
(0, 476) 1
(0, 4039) 1
(0, 4289) 2
(0, 3143) 1
(0, 3791) 1
:
(0, 4401) 1
(0, 2704) 1
(0, 1207) 1
(0, 1432) 1
(0, 3827) 1
(0, 4321) 1
(0, 2029) 1
(0, 188) 1
(0, 2299) 1
(0, 3883) 1
(0, 3021) 1
(0, 1517) 1
(0, 673) 1
(0, 1058) 1
(0, 3386) 1
(0, 1708) 1
(0, 2571) 2
(0, 1151) 1
(0, 3553) 1
(0, 3870) 1
(0, 4771) 1
(0, 3596) 2
(0, 2908) 1
(0, 1898) 1
(0, 2588) 2

```

```
In [218]: # 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, 2699)
Shape of matrix after one hot encodig  (36052, 2699)
```

2.3.2 TFIDF

```
In [219]: 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 [220]: # 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, 2699)
Shape of matrix after one hot encodig  (36052, 2699)
```

2.3.3 AVG W2V

```

In [80]: 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, min_count=0)
#this line of code trains your w2v model on the give list of sentences. Instead of train on X_train.values, need to train individual words in it

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

'''

```


Done. 1917494 words loaded!

```
Out[80]: '\n\n\n# Step 1: Getting each word from the sentence\ndef list_of_words(Sentence):\n    return Sentence.split()\n\nlist_of_Sentence=list(X_train['processed_essays'].values)\nprint(len(list_of_Sentence))\nwords_list_of_each_Sentence=list(map(list_of_words,list_of_Sentence))\n\nStep 2: Apply word2vec\n\nfrom gensim.models import word2vec\nw2v_model = word2vec.Word2Vec(words_list_of_each_Sentence, size=100,workers=2, min_count=0)\n\n#this line of code trains your w2v model on the give list of sentences. Instead of train on X_train.values, \nneed to train individual words in it\n\nglove_words = list(w2v_model.wv.vocab)\nprint("number of words that occurred minimum 5 times ",len(glove_words))\nprint("sample words ", glove_words)\n\n'
```

```
In [81]: with open('glove_vectors', 'rb') as f:
          w2v_model = pickle.load(f)
          glove_words = set(model.keys())
```

```
In [221]: # average Word2Vec
# compute average word2vec for each review.
avg_w2v_vectors_train = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['preprocessed_essays'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length, 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 [01:05<00:00, 1120.84it/s]
```

73196

```
In [83]: #this line of code trains your w2v model on the give list of sentences
'''

w2v_model=Word2Vec(X_cv['preprocessed_essays'].values,min_count=5,size=50, worker
s=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[83]: '\n\nw2v_model=Word2Vec(X_cv[\'preprocessed_essays\'].values,min_count=5,size=5
0, workers=4)\n\n\nglove_words = list(w2v_model.wv.vocab)\n\n\nprint("number of
words that occurred minimum 5 times ",len(glove_words))\nprint("sample words ", g
love_vector[0:50])\n'
```

```
In [84]: '''
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/sente
nce
    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 f
unction
            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))
'''
```

```
100%|███████████████████████████████████████████████████████████████  
24155/24155 [00:17<00:00, 1406.34it/s]  
  
24155
```



```
In [87]: # 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 occurred 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))
'''
```

```
100% |██████████████████████████████████████████████████████████████████████████████|
24155/24155 [00:01<00:00, 15935.35it/s]

24155
```


In [228]:

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

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

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

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

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

```
100% |██████████████████████████████████████████████████████████████████████████████|
73196/73196 [00:09<00:00, 7947.79it/s]

73196
300
```

```

# 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 value(sentence.count(word)/len(sentence.split()))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_title_cv.append(vector)

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

'''

```

```
100% |██████████████████████████████████████████████████████████████████████|
24155/24155 [00:02<00:00, 11277.32it/s]

24155
300
```

In [229]:

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

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

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

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

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

```
100% |██████████████████████████████████████████████████████████████████████████████|
36052/36052 [00:05<00:00, 6303.60it/s]

36052
300
```

In []:

2.4 Applying Logistic Regression on different kind of featurization as mentioned in the instructions

Apply Logistic Regression 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 instructions

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

from scipy.sparse import hstack

X_train_bow = hstack((categories_one_hot_train, sub_categories_one_hot_train, state_one_hot_train, pg_one_hot_train, tp_one_hot_train, price_standardized_train, previous_standardized_train, \
                      quantity_standardized_train, wc_standardized_train, wc_title_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_one_hot_test, pg_one_hot_test, tp_one_hot_test, price_standardized_test, previous_standardized_test, \
                   quantity_standardized_test, wc_standardized_test, wc_title_standardized_test, \
                   ss_neg_standardized_test, ss_neu_standardized_test, ss_pos_standardized_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_hot_cv, pg_one_hot_cv, tp_one_hot_cv, price_standardized_cv, previous_standardized_cv, \
#                  quantity_standardized_cv, wc_standardized_cv, wc_title_standardized_cv, \
#                  ss_neg_standardized_cv, ss_neu_standardized_cv, ss_pos_standardized_cv, ss_com_standardized_cv, \
#                  title_cv_bow, text_cv_bow)).tocsr()
```

```
In [244]: 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, 7808)
(36052,)
(73196, 7808)
(73196,)
```

```
In [248]: from sklearn.model_selection import GridSearchCV
          from sklearn.linear_model import LogisticRegression
          from sklearn.metrics import roc_auc_score
          import matplotlib.pyplot as plt

          LR = LogisticRegression(class_weight= 'balanced')
          parameters = [{'C': [10**-4, 10**-3, 10**-2.5, 10**-2, 10**0, 10, 10**2, 10**4]}]
          clf = GridSearchCV(LR, parameters, cv=5, return_train_score = True, scoring='roc_
          auc')
          clf.fit(X_train_bow, y_train)
```

```
Out[248]: GridSearchCV(cv=5, error_score='raise-deprecating',
                      estimator=LogisticRegression(C=1.0, class_weight='balanced',
                                                    dual=False, fit_intercept=True,
                                                    intercept_scaling=1, l1_ratio=None,
                                                    max_iter=100, multi_class='warn',
                                                    n_jobs=None, penalty='l2',
                                                    random_state=None, solver='warn',
                                                    tol=0.0001, verbose=0,
                                                    warm_start=False),
                      iid='warn', n_jobs=None,
                      param_grid=[{'C': [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 [249]: 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 [250]: # import math
C = [10**-4,10**-3, 10**-2.5, 10**-2, 10**0, 10, 10**2, 10**4]

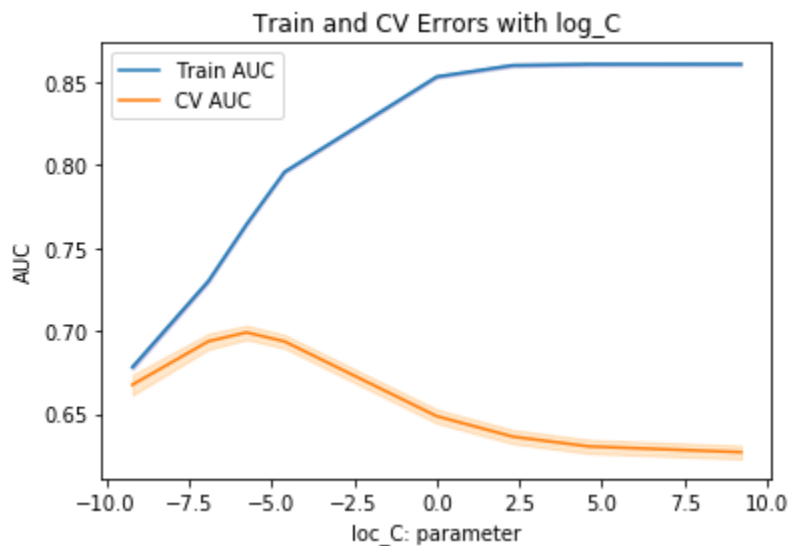
log_C = []

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,color='darkorange')
plt.legend()
plt.xlabel("log_C: parameter")
plt.ylabel("AUC")
plt.title("Train and CV Errors with log_C")
plt.show()

```



```

In [251]: best_c = 10** -2.5

```

```
In [269]: LR = LogisticRegression(class_weight= 'balanced',C = best_c, penalty= 'l2')
print(LR)
clf = LR.fit(X_train_bow, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates
# of the positive class
# not the predicted outputs

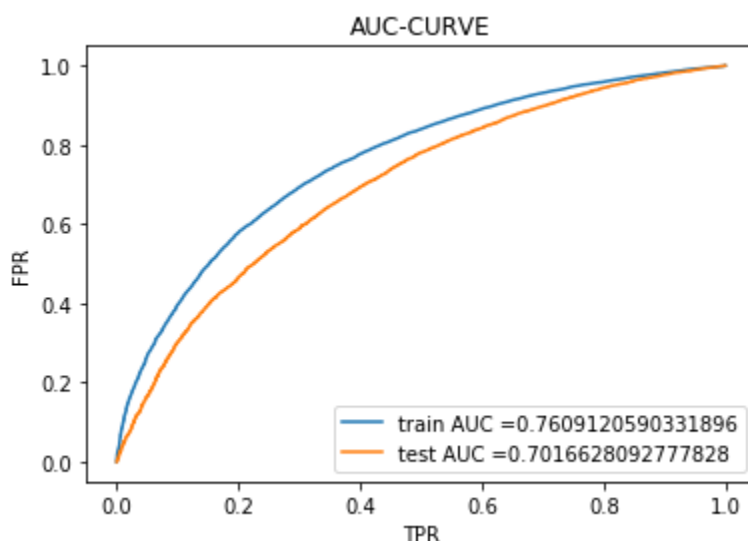
# DO not use predict for ROC curve
# https://discuss.analyticsvidhya.com/t/what-is-the-difference-between-predict-and-predict-proba/67376/2
y_train_pred = clf.predict_proba(X_train_bow)
y_test_pred = clf.predict_proba(X_test_bow)

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

plt.plot(train_fpr, train_tpr, label="train AUC =" +str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC =" +str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("TPR")
plt.ylabel("FPR")
plt.title("AUC-CURVE")
plt.show()

print("="*100)
```

```
LogisticRegression(C=0.0031622776601683794, class_weight='balanced', dual=False,
fit_intercept=True, intercept_scaling=1, l1_ratio=None,
max_iter=100, multi_class='warn', n_jobs=None, penalty='l2',
random_state=None, solver='warn', tol=0.0001, verbose=0,
warm_start=False)
```



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

```
In [270]: print(y_train_pred[:,1].shape)
print(y_test.shape)
print(X_train_bow.shape)
```

```
(73196,)
(36052,)
(73196, 7808)
```

```
In [261]: # how to choose threshold for Confusion matrix
#https://stackoverflow.com/questions/32627926/scikit-changing-the-threshold-to-cr
#eate-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_ma
#trix.html

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

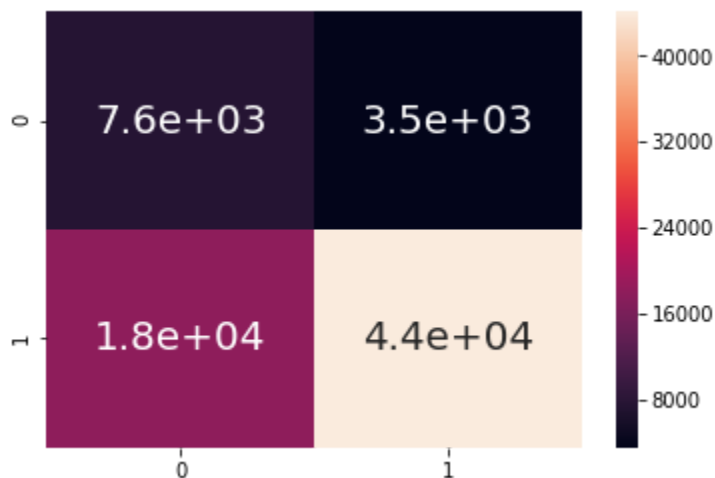


```
In [272]: print("Train confusion matrix")
cm_train = pd.DataFrame(confusion_matrix(y_train, predict(y_train_pred[:, 1], tr_
thresholds, train_fpr, train_tpr)))
print(cm_train)

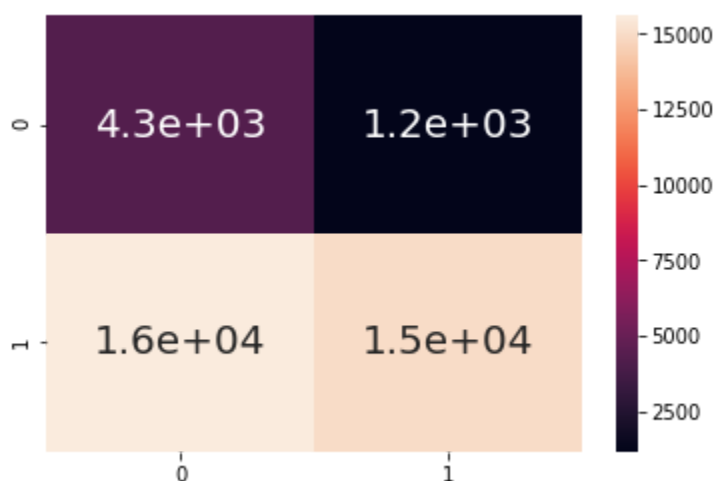
print("Test confusion matrix")
cm_test = pd.DataFrame(confusion_matrix(y_test, predict(y_test_pred[:, 1], tr_thr
esholds, test_fpr, test_tpr)))
print(cm_test)
```

```
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.48543389728115555 for threshold 0.487
      0      1
0   7550   3533
1  18042  44071
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.4198274413184385 for threshold 0.567
      0      1
0   4280   1179
1  15588  15005
```

```
In [273]: 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[273]: <matplotlib.axes._subplots.AxesSubplot at 0x2298e77b0f0>
```



2.4.2 Applying LR brute force on TFIDF

```
In [254]: # 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, state_one_hot_train, pg_one_hot_train, tp_one_hot_train, price_standardized_train, previous_standardized_train, \
                        quantity_standardized_train, wc_standardized_train, wc_title_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, previous_standardized_test, \
                      quantity_standardized_test, wc_standardized_test, wc_title_standardized_test, \
                      ss_neg_standardized_test, ss_neu_standardized_test, ss_pos_standardized_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_standardized_cv, \
#                    quantity_standardized_cv, wc_standardized_cv, wc_title_standardized_cv, \
#                    ss_neg_standardized_cv, ss_neu_standardized_cv, ss_pos_standardized_cv, ss_com_standardized_cv, \
#                    title_cv_tfidf, text_cv_tfidf)).tocsr()
```

```
In [255]: 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, 7808)
(36052,)
(73196, 7808)
(73196,)
```

```
In [274]: from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt

LR = LogisticRegression(class_weight= 'balanced')
parameters = [{'C': [10**-4, 10**-3, 10**-2.5, 10**-2, 10**-1, 10**0,10, 10**2, 1
0**4]]}]
clf = GridSearchCV(LR, parameters, cv=5, return_train_score = True, scoring='roc_
auc')
clf.fit(X_train_tfidf, y_train)
```

```
Out[274]: GridSearchCV(cv=5, error_score='raise-deprecating',
                        estimator=LogisticRegression(C=1.0, class_weight='balanced',
                                                    dual=False, fit_intercept=True,
                                                    intercept_scaling=1, l1_ratio=None,
                                                    max_iter=100, multi_class='warn',
                                                    n_jobs=None, penalty='l2',
                                                    random_state=None, solver='warn',
                                                    tol=0.0001, verbose=0,
                                                    warm_start=False),
                        iid='warn', n_jobs=None,
                        param_grid=[{'C': [0.0001, 0.001, 0.0031622776601683794, 0.01, 0.1,
                                           1, 10, 100, 10000]}],
                        pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
                        scoring='roc_auc', verbose=0)
```

```
In [275]: 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 [283]: import math
C= [10**-4, 10**-3, 10**-2.5, 10**-2, 10**-1, 10**0,10, 10**2, 10**4]

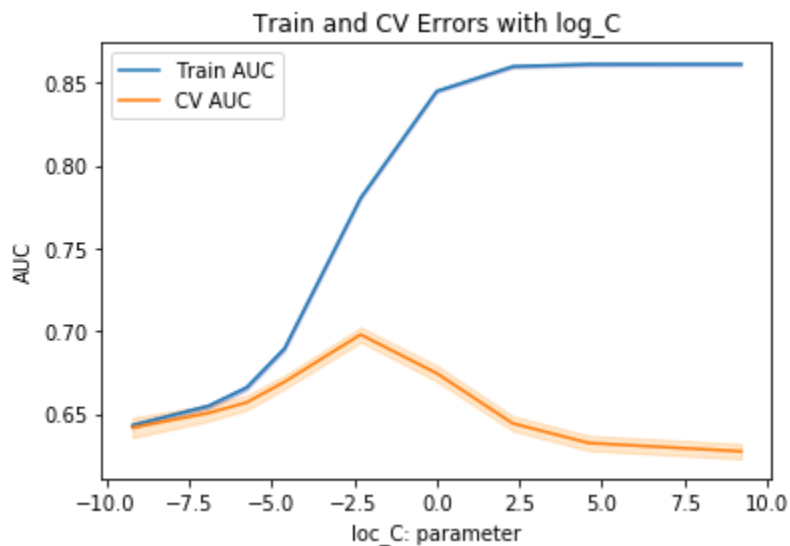
log_C = []

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,color='darkorange')
plt.legend()
plt.xlabel("log_C: parameter")
plt.ylabel("AUC")
plt.title("Train and CV Errors with log_C")
plt.show()

```



```

In [288]: best_c = 10**-2.3

```

```
In [289]: LR = LogisticRegression(class_weight = 'balanced',C = best_c, penalty= 'l2')
print(LR)
clf = LR.fit(X_train_tfidf, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates
# of the positive class
# not the predicted outputs

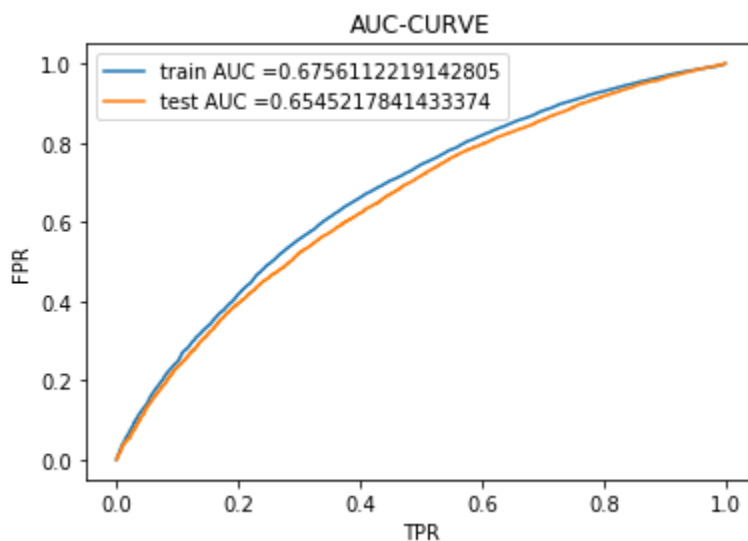
# DO not use predict for ROC curve
# https://discuss.analyticsvidhya.com/t/what-is-the-difference-between-predict-and-predict-proba/67376/2
y_train_pred = clf.predict_proba(X_train_tfidf)
y_test_pred = clf.predict_proba(X_test_tfidf)

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

plt.plot(train_fpr, train_tpr, label="train AUC =" +str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC =" +str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("TPR")
plt.ylabel("FPR")
plt.title("AUC-CURVE")
plt.show()

print("="*100)
```

```
LogisticRegression(C=0.005011872336272725, class_weight='balanced', dual=False,
fit_intercept=True, intercept_scaling=1, l1_ratio=None,
max_iter=100, multi_class='warn', n_jobs=None, penalty='l2',
random_state=None, solver='warn', tol=0.0001, verbose=0,
warm_start=False)
```



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

```
In [290]: print("Train confusion matrix")
cm_train = pd.DataFrame(confusion_matrix(y_train, predict(y_train_pred[:, 1], 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[:, 1], tr_thr
esholds, test_fpr, test_tpr)))
print(cm_test)
```

Train confusion matrix

the maximum value of $tpr \cdot (1 - fpr)$ 0.3996167835252849 for threshold 0.499

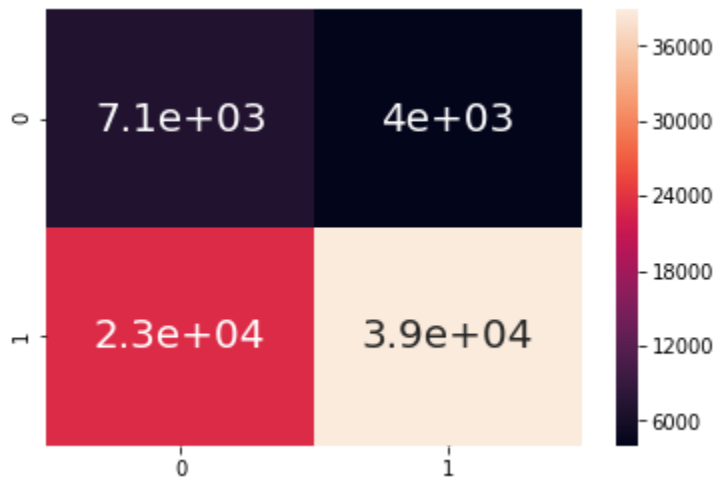
	0	1
0	7054	4029
1	23259	38854

Test confusion matrix

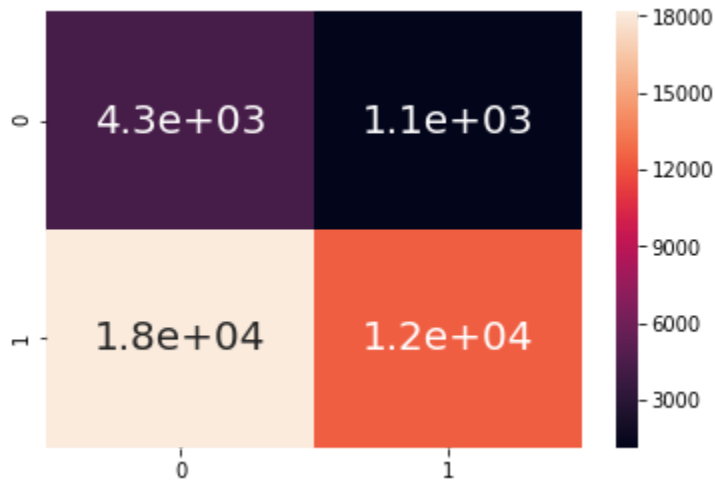
the maximum value of $tpr \cdot (1 - fpr)$ 0.37515990254958315 for threshold 0.558

	0	1
0	4312	1147
1	18161	12432

```
In [291]: 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[291]: <matplotlib.axes._subplots.AxesSubplot at 0x2298c3be3c8>
```



2.4.3 Applying LR brute force on AVG W2V

In [292]: *# 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, state_one_hot_train, pg_one_hot_train, tp_one_hot_train, price_standardized_train, previous_standardized_train, \
                      quantity_standardized_train, wc_standardized_train, wc_title_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_one_hot_test, pg_one_hot_test, tp_one_hot_test, price_standardized_test, previous_standardized_test, \
                    quantity_standardized_test, wc_standardized_test, wc_title_standardized_test, \
                    ss_neg_standardized_test, ss_neu_standardized_test, ss_pos_standardized_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_hot_cv, pg_one_hot_cv, tp_one_hot_cv, price_standardized_cv, previous_standardized_cv, \
#                  quantity_standardized_cv, wc_standardized_cv, wc_title_standardized_cv, \
#                  ss_neg_standardized_cv, ss_neu_standardized_cv, ss_pos_standardized_cv, ss_com_standardized_cv, \
#                  avg_w2v_vectors_titles_cv, avg_w2v_vectors_cv)).tocsr()
```

In [294]:

```
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,)
```



```
In [296]: from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt

LR = LogisticRegression(class_weight='balanced')
parameters = [{'C': [10**-4, 10**-3, 10**-2.5, 10**-2, 10**0, 10, 10**2, 10**4]}]
clf = GridSearchCV(LR, parameters, cv=5, return_train_score = True, scoring='roc_auc')
clf.fit(X_train_w2v, y_train)
```

```
Out[296]: GridSearchCV(cv=5, error_score='raise-deprecating',
                        estimator=LogisticRegression(C=1.0, class_weight='balanced',
                                                    dual=False, fit_intercept=True,
                                                    intercept_scaling=1, l1_ratio=None,
                                                    max_iter=100, multi_class='warn',
                                                    n_jobs=None, penalty='l2',
                                                    random_state=None, solver='warn',
                                                    tol=0.0001, verbose=0,
                                                    warm_start=False),
                        iid='warn', n_jobs=None,
                        param_grid=[{'C': [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 [297]: 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 [298]: import math
C = [10**-4, 10**-3, 10**-2.5, 10**-2, 10**0, 10, 10**2, 10**4]

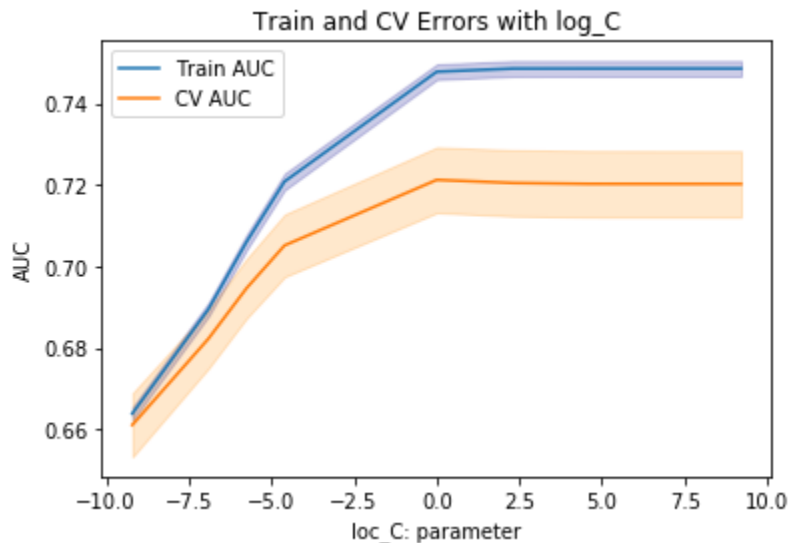
log_C = []

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, color='darkorange')
plt.legend()
plt.xlabel("log_C: parameter")
plt.ylabel("AUC")
plt.title("Train and CV Errors with log_C")
plt.show()

```



```

In [299]: best_c = 10** 0

```

```
In [300]: LR = LogisticRegression(class_weight = 'balanced', C = best_c, penalty= 'l2')
print(LR)
clf = LR.fit(X_train_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

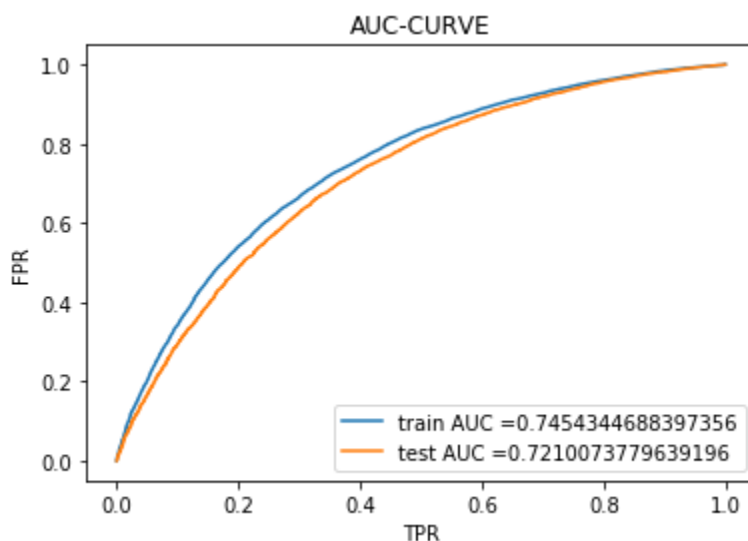
# DO not use predict for ROC curve
# https://discuss.analyticsvidhya.com/t/what-is-the-difference-between-predict-and-predict-proba/67376/2
y_train_pred = clf.predict_proba(X_train_w2v)
y_test_pred = clf.predict_proba(X_test_w2v)

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

plt.plot(train_fpr, train_tpr, label="train AUC =" + str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC =" + str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("TPR")
plt.ylabel("FPR")
plt.title("AUC-CURVE")
plt.show()

print("="*100)
```

```
LogisticRegression(C=1, class_weight='balanced', dual=False, fit_intercept=True,
intercept_scaling=1, l1_ratio=None, max_iter=100,
multi_class='warn', n_jobs=None, penalty='l2',
random_state=None, solver='warn', tol=0.0001, verbose=0,
warm_start=False)
```



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

```
In [301]: print("Train confusion matrix")
cm_train = pd.DataFrame(confusion_matrix(y_train, predict(y_train_pred[:, 1], 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[:, 1], tr_thr
esholds, test_fpr, test_tpr)))
print(cm_test)
```

Train confusion matrix

the maximum value of $tpr \cdot (1 - fpr)$ 0.4687953891884455 for threshold 0.509

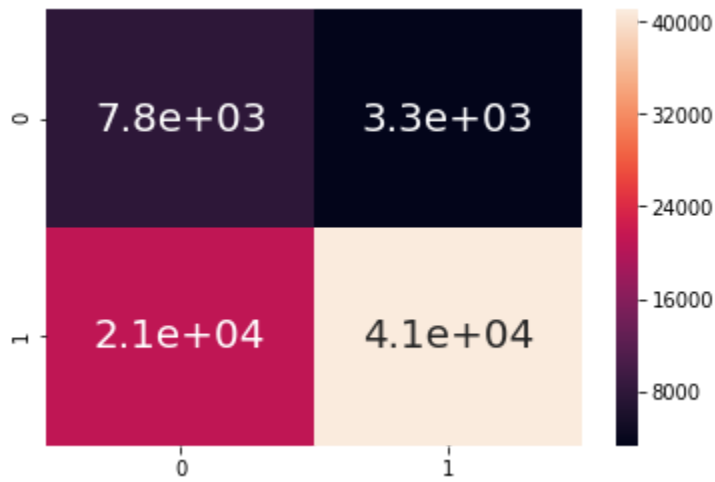
	0	1
0	7799	3284
1	21100	41013

Test confusion matrix

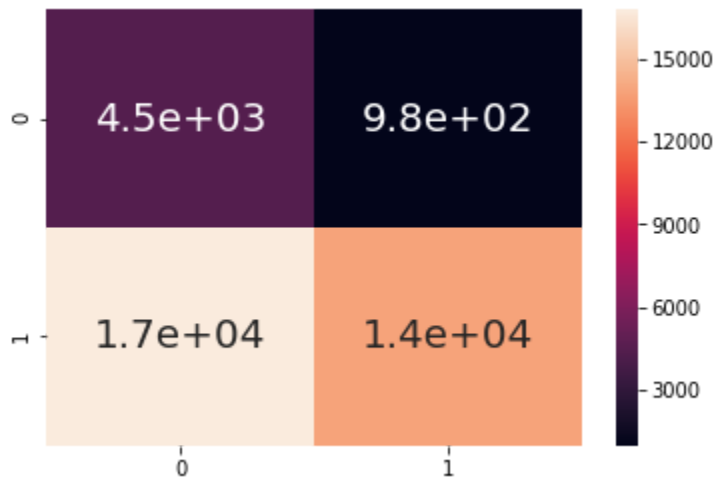
the maximum value of $tpr \cdot (1 - fpr)$ 0.44584083079011444 for threshold 0.619

	0	1
0	4479	980
1	16755	13838

```
In [302]: 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[302]: <matplotlib.axes._subplots.AxesSubplot at 0x2298e7c4710>
```



2.4.4 Applying KNN brute force on TFIDF WEIGHTED W2V

In [303]: *# 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_train,
state_one_hot_train, pg_one_hot_train, tp_one_hot_train, price_standardized_train,
previous_standardized_train, \
                            quantity_standardized_train, wc_standardized_train, wc_title_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)).tocsr()

X_test_tfidf_w2v= hstack((categories_one_hot_test, sub_categories_one_hot_test, state_one_hot_test, pg_one_hot_test, tp_one_hot_test, price_standardized_test, previous_standardized_test, \
                            quantity_standardized_test, wc_standardized_test, wc_title_standardized_test,\
                            ss_neg_standardized_test,ss_neu_standardized_test,ss_pos_standardized_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_standardized_cv, \
#                            quantity_standardized_cv, wc_standardized_cv, wc_title_standardized_cv,\
#                            ss_neg_standardized_cv,ss_neu_standardized_cv,ss_pos_standardized_cv, ss_com_standardized_cv,\
#                            tfidf_w2v_vectors_title_cv,tfidf_w2v_vectors_cv)).tocsr()
```

In [304]: `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 [308]: from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt

LR = LogisticRegression(class_weight= 'balanced')
parameters = [{'C': [10**-4, 10**-2, 10**0, 10, 10**2, 10**4]}]
clf = GridSearchCV(LR, parameters, cv=5, return_train_score = True, scoring='roc_
auc')
clf.fit(X_train_tfidf_w2v, y_train)
```

```
Out[308]: GridSearchCV(cv=5, error_score='raise-deprecating',
                        estimator=LogisticRegression(C=1.0, class_weight='balanced',
                                                    dual=False, fit_intercept=True,
                                                    intercept_scaling=1, l1_ratio=None,
                                                    max_iter=100, multi_class='warn',
                                                    n_jobs=None, penalty='l2',
                                                    random_state=None, solver='warn',
                                                    tol=0.0001, verbose=0,
                                                    warm_start=False),
                        iid='warn', n_jobs=None,
                        param_grid=[{'C': [0.0001, 0.01, 1, 10, 100, 10000]}],
                        pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
                        scoring='roc_auc', verbose=0)
```

```
In [310]: 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 [311]: import math
C = [10**-4, 10**-2, 10**0, 10, 10**2, 10**4]

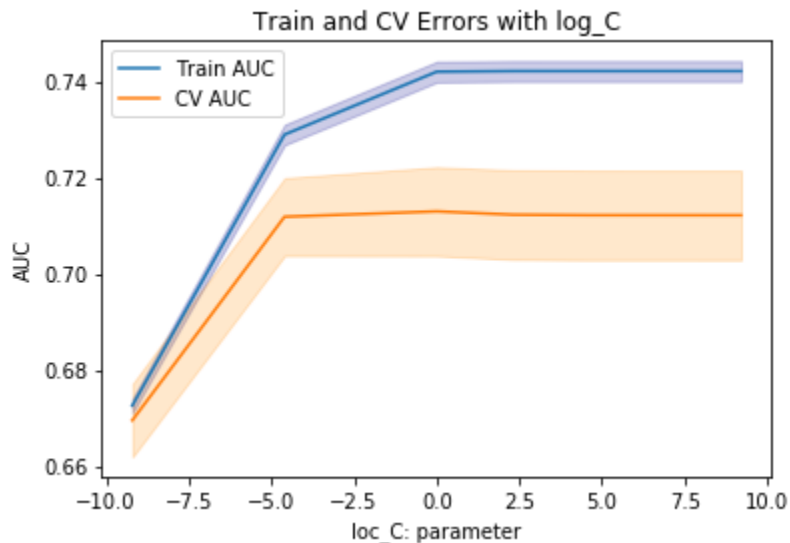
log_C = []

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, color='darkorange')
plt.legend()
plt.xlabel("log_C: parameter")
plt.ylabel("AUC")
plt.title("Train and CV Errors with log_C")
plt.show()

```



```

In [312]: best_c = 10** 0

```



```
In [315]: LR = LogisticRegression(class_weight = 'balanced',C = best_c, penalty= 'l2')
print(LR)
clf = LR.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

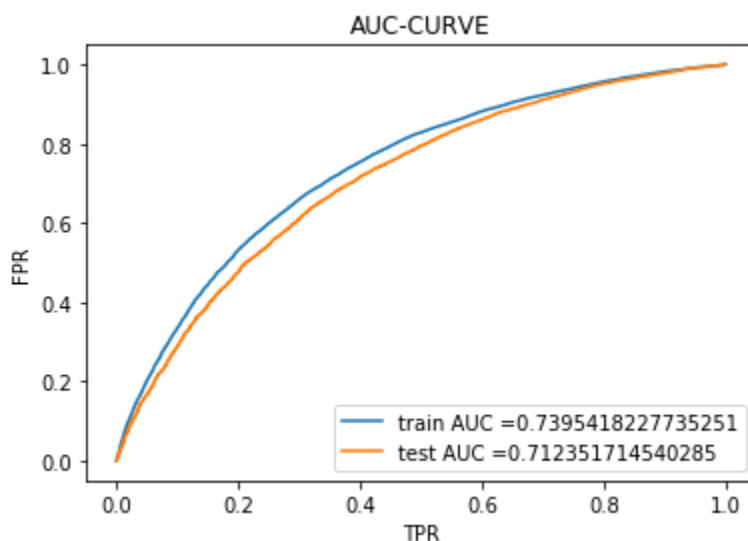
# DO not use predict for ROC curve
# https://discuss.analyticsvidhya.com/t/what-is-the-difference-between-predict-and-predict-proba/67376/2
y_train_pred = clf.predict_proba(X_train_tfidf_w2v)
y_test_pred = clf.predict_proba(X_test_tfidf_w2v)

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

plt.plot(train_fpr, train_tpr, label="train AUC =" +str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC =" +str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("TPR")
plt.ylabel("FPR")
plt.title("AUC-CURVE")
plt.show()

print("="*100)
```

```
LogisticRegression(C=1, class_weight='balanced', dual=False, fit_intercept=True,
intercept_scaling=1, l1_ratio=None, max_iter=100,
multi_class='warn', n_jobs=None, penalty='l2',
random_state=None, solver='warn', tol=0.0001, verbose=0,
warm_start=False)
```



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

```
In [316]: print("Train confusion matrix")
cm_train = pd.DataFrame(confusion_matrix(y_train, predict(y_train_pred[:, 1], 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[:, 1], tr_thr
esholds, test_fpr, test_tpr)))
print(cm_test)
```

Train confusion matrix

the maximum value of $tpr \cdot (1 - fpr)$ 0.46374637817094566 for threshold 0.481

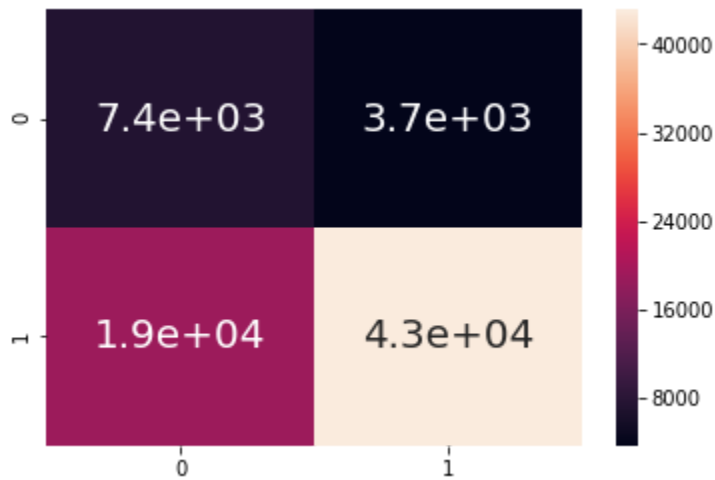
	0	1
0	7370	3713
1	19011	43102

Test confusion matrix

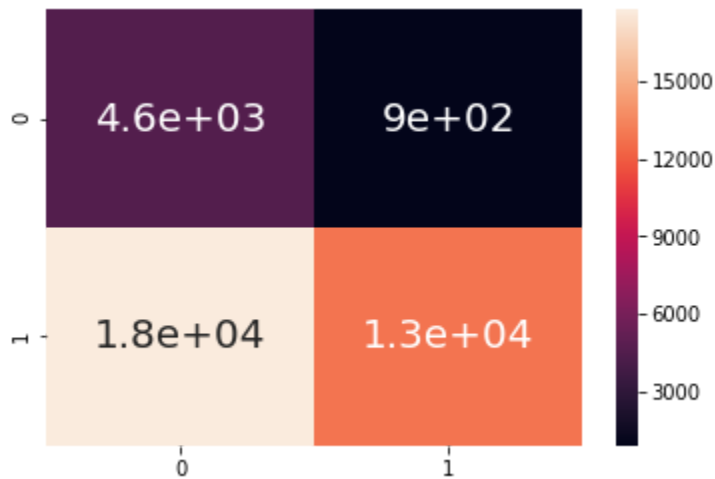
the maximum value of $tpr \cdot (1 - fpr)$ 0.43504318170450956 for threshold 0.623

	0	1
0	4561	898
1	17763	12830

```
In [317]: 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[317]: <matplotlib.axes._subplots.AxesSubplot at 0x2298e7ed828>
```



2.5 Logistic Regression 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
# 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 [318]: *# Please write all the code with proper documentation*

```
from scipy.sparse import hstack

X_train_new = hstack((categories_one_hot_train, sub_categories_one_hot_train, state_one_hot_train, pg_one_hot_train, tp_one_hot_train, price_standardized_train, previous_standardized_train, \
                      quantity_standardized_train, wc_standardized_train, wc_title_standardized_train, \
                      ss_neg_standardized_train, ss_neu_standardized_train, ss_pos_standardized_train, ss_com_standardized_train, \
                      )).tocsr()

X_test_new= hstack((categories_one_hot_test, sub_categories_one_hot_test, state_one_hot_test, pg_one_hot_test, tp_one_hot_test, price_standardized_test, previous_standardized_test, \
                   quantity_standardized_test, wc_standardized_test, wc_title_standardized_test, \
                   ss_neg_standardized_test, ss_neu_standardized_test, ss_pos_standardized_test, ss_com_standardized_test, \
                   )).tocsr()

#X_cv_new = hstack((categories_one_hot_cv, sub_categories_one_hot_cv, state_one_hot_cv, pg_one_hot_cv, tp_one_hot_cv, price_standardized_cv, previous_standardized_cv, \
#                  quantity_standardized_cv, wc_standardized_cv, wc_title_standardized_cv, \
#                  ss_neg_standardized_cv, ss_neu_standardized_cv, ss_pos_standardized_cv, ss_com_standardized_cv, \
#                  )).tocsr()
```

In [319]: `print(X_test_new.shape)`
`print(y_test.shape)`

```
print(X_train_new.shape)  
print(y_train.shape)
```

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

```
(36052, 109)
(36052,)
(73196, 109)
(73196,)
```

In []:

```
In [321]: from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt

LR = LogisticRegression(class_weight='balanced')
parameters = [{'C': [10**-4, 10**-2.5, 10**-2, 10**-1, 10**0, 10, 10**2, 10**4]}]
clf = GridSearchCV(LR, parameters, cv=5, return_train_score = True, scoring='roc_
auc')
clf.fit(X_train_new, y_train)
```

```
Out[321]: GridSearchCV(cv=5, error_score='raise-deprecating',
                        estimator=LogisticRegression(C=1.0, class_weight='balanced',
                                                    dual=False, fit_intercept=True,
                                                    intercept_scaling=1, l1_ratio=None,
                                                    max_iter=100, multi_class='warn',
                                                    n_jobs=None, penalty='l2',
                                                    random_state=None, solver='warn',
                                                    tol=0.0001, verbose=0,
                                                    warm_start=False),
                        iid='warn', n_jobs=None,
                        param_grid=[{'C': [0.0001, 0.0031622776601683794, 0.01, 0.1, 1, 10,
                                           100, 10000]}],
                        pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
                        scoring='roc_auc', verbose=0)
```

```
In [139]: 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 [140]: import math
C = [10**-4,10**-2.5, 10**-2,10**-1,10**0,10,10**2,10**4]

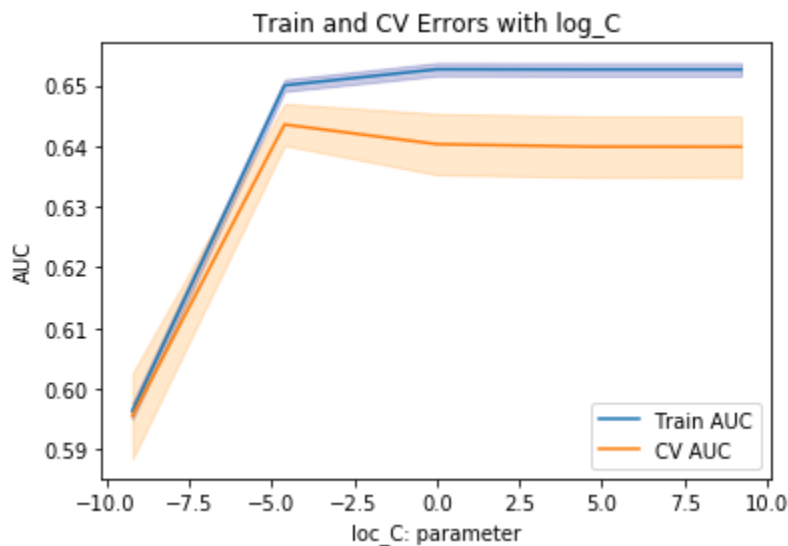
log_C = []

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,color='darkorange')
plt.legend()
plt.xlabel("log_C: parameter")
plt.ylabel("AUC")
plt.title("Train and CV Errors with log_C")
plt.show()

```



```

In [334]: best_c = 10** -3.5

```

```
In [335]: LR = LogisticRegression(class_weight = 'balanced',C = best_c, penalty= 'l2')
print(LR)
clf = LR.fit(X_train_new, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates
# of the positive class
# not the predicted outputs

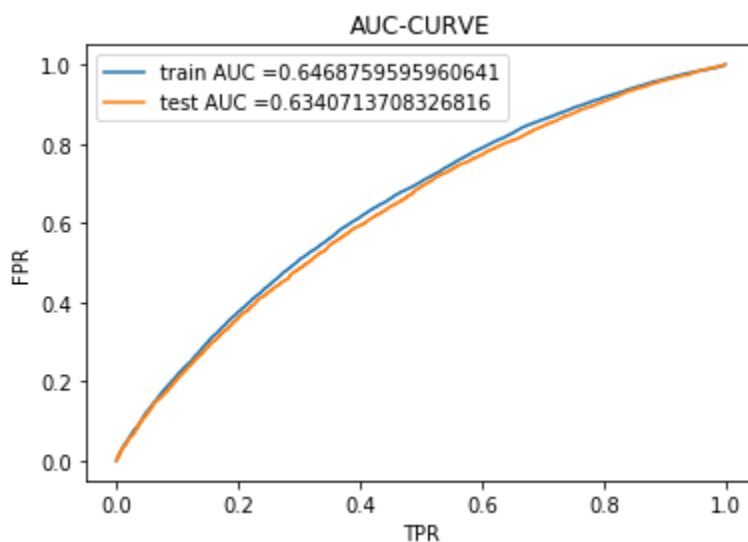
# DO not use predict for ROC curve
# https://discuss.analyticsvidhya.com/t/what-is-the-difference-between-predict-and-predict-proba/67376/2
y_train_pred = clf.predict_proba(X_train_new)
y_test_pred = clf.predict_proba(X_test_new)

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

plt.plot(train_fpr, train_tpr, label="train AUC =" +str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC =" +str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("TPR")
plt.ylabel("FPR")
plt.title("AUC-CURVE")
plt.show()

print(" "*100)
```

```
LogisticRegression(C=0.00031622776601683794, class_weight='balanced',
                    dual=False, fit_intercept=True, intercept_scaling=1,
                    l1_ratio=None, max_iter=100, multi_class='warn', n_jobs=None,
                    penalty='l2', random_state=None, solver='warn', tol=0.0001,
                    verbose=0, warm_start=False)
```



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

```
In [336]: print("Train confusion matrix")
cm_train = pd.DataFrame(confusion_matrix(y_train, predict(y_train_pred[:, 1], 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[:, 1], tr_thr
esholds, test_fpr, test_tpr)))
print(cm_test)
```

Train confusion matrix

the maximum value of $tpr \cdot (1 - fpr)$ 0.36977279111228123 for threshold 0.497

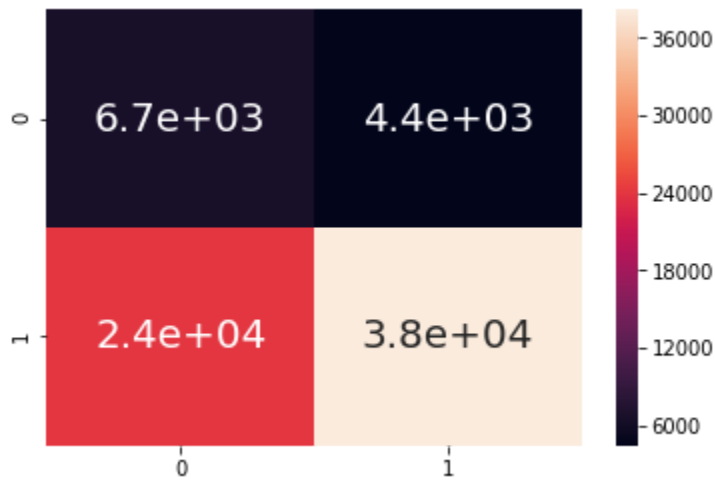
	0	1
0	6651	4432
1	23968	38145

Test confusion matrix

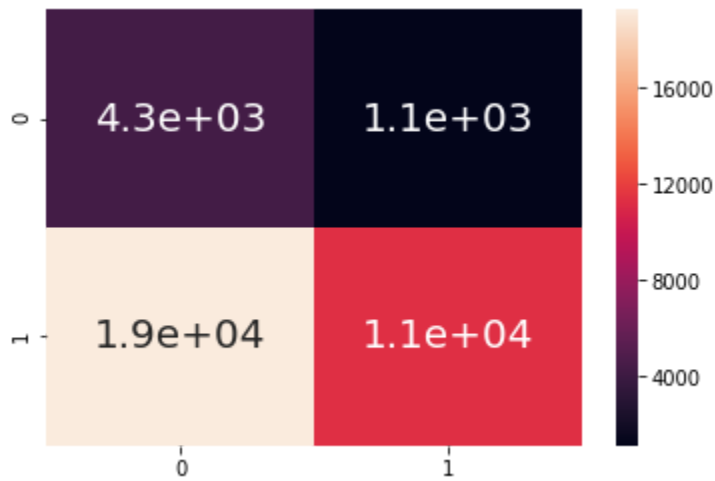
the maximum value of $tpr \cdot (1 - fpr)$ 0.35807588927295686 for threshold 0.547

	0	1
0	4330	1129
1	19255	11338


```
In [337]: 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[337]: <matplotlib.axes._subplots.AxesSubplot at 0x2298c2b6c18>
```



3. Conclusion

```

In [338]: # 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", "LR", 10**-2.5, 0.7017])
x.add_row(["TFIDF", "LR", 10**-2.5, 0.655])
x.add_row(["AVG W2V", "LR", 1, 0.721])
x.add_row(["TFIDF WEIGHTED W2V", "LR", 1, 0.7124 ])
x.add_row(["SET 5", "LR", 10**-3.5, 0.6341])

print(x)

```

Vectorizer	Model	Hyper Parameter	AUC
BOW	LR	0.0031622776601683794	0.7017
TFIDF	LR	0.0031622776601683794	0.655
AVG W2V	LR	1	0.721
TFIDF WEIGHTED W2V	LR	1	0.7124
SET 5	LR	0.00031622776601683794	0.6341

In []: