Assignment 5 - Logistic Regression on Donors choose dataset

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.

Problem Objective - The objective is to predict whether project proposal submitted by a teacher or not, by applying KNN algorithm and deciding the best Feature generation technique for given problem.

About the DonorsChoose Data Set

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

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

Feature	Description
`project_essay_1`	First application essay*
`project_essay_2`	Second application essay*
`project_essay_3`	Third application essay*
`project_essay_4`	Fourth application essay*
`project_submitted_datetime`	Datetime when project application was submitted. **Example:** `2016-04-28 12:43:56.245`
`teacher_id`	A unique identifier for the teacher of the proposed project. **Example:** `bdf8baa8fedef6bfeec7ae4ff1c15c56`
`teacher_prefix`	Teacher's title. One of the following enumerated values: • `nan` • `Dr.` • `Mr.` • `Mrs.` • `Ms.` • `Teacher.`
`teacher_number_of_previously_posted_projects`	Number of project applications previously submitted by the same teacher. **Example:** `2`

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

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

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

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

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

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

Notes on the Essay Data

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

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

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

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

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

In [1]:

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

```
import sqlite3
import pandas as pd
import numpy as np
import nltk
nltk.downloader.download('vader_lexicon')
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
```

```
[nltk_data] Downloading package vader_lexicon to
[nltk_data]     C:\Users\lenovo\AppData\Roaming\nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!
```

1.1) Reading the data

In [2]:

```
train_data = pd.read_csv(r"D:\Assignments of Applied AI\Donorschoose data set\train_data.csv")
resource_data = pd.read_csv(r"D:\Assignments of Applied AI\Donorschoose data set\resources.csv")
```

In [3]:

```
print('Number of data points in the train data', train_data.shape)
print('-'*127)
print('The attributes of the data points in the train data :', train_data.columns.values)
train_data.head(2)
```

```
Number of data points in the train data (109248, 17)
---------------------------------------------------------------------------------------------------------------
-----------------------
The attributes of the data points in the train 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']
```

Out[3]:

Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime	project_grade_cate

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime	project_grade_cate
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	2016-12-05 13:43:57	Grades P
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL	2016-10-25 09:22:10	Grade

In [4]:

```
# how to replace elements in list python: https://stackoverflow.com/a/2582163/4084039
cols = ['Date' if x=='project_submitted_datetime' else x for x in list(train_data.columns)]

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

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

#train_data.head(2)
```

In [5]:

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

```
Number of data points in resource 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 project_subject_categories

In [6]:

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

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

In [7]:

```
train_data['clean_categories'] = pro_sub_cat_list
train_data.drop(['project_subject_categories'], axis=1, inplace=True)
```

In [8]:

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

In [9]:

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

1.3) Preprocessing project_subject_subcategories

In [10]:

```
pro_sub_subcatogories = list(train_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

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

In [11]:

```
train_data['clean_subcategories'] = pro_sub_subcat_list
train_data.drop(['project_subject_subcategories'], axis=1, inplace=True)
```

In [12]:

```
from collections import Counter
my_counter = Counter()
for word in train_data['clean_subcategories'].values:
    my_counter.update(word.split())
```

In [13]:

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

1.4) Text Preprocessing the titles

In [14]:

```
# 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 [15]:

```
# 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 [16]:

```
clean_titles = []

for titles in tqdm(train_data["project_title"]):
    title = decontracted(titles)
    title = title.replace('\\r', ' ')
    title = title.replace('\\"', ' ')
    title = title.replace('\\n', ' ')
    title = re.sub('[^A-Za-z0-9]+', ' ', title)
    title = ' '.join(f for f in title.split() if f not in stopwords)
    clean_titles.append(title.lower().strip())
```

```
100%|██████████████████████████████| 109248/109248 [00:04<00:00, 24615.58it/s]
```

In [17]:

```
train_data["clean_titles"] = clean_titles
```

In [18]:

```
train_data.drop(['project_title'], axis=1, inplace=True)
```

1.5) Combine 4 project essay

In [19]:

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

1.6) Text preprocessing the essay

In [20]:

```
clean_essay = []

for ess in tqdm(train_data["essay"]):
    ess = decontracted(ess)
    ess = ess.replace('\\r', ' ')
    ess = ess.replace('\\"', ' ')
    ess = ess.replace('\\n', ' ')
    ess = re.sub('[^A-Za-z0-9]+', ' ', ess)
    ess = ' '.join(f for f in ess.split() if f not in stopwords)
    clean_essay.append(ess.lower().strip())
```

```
100%|██████████████████████████████████| 109248/109248 [01:32<00:00, 1184.38it/s]
```

In [21]:

```
train_data["clean_essays"] = clean_essay
```

In [22]:

```
train_data.drop(['essay'], axis=1, inplace=True)
```

1.7) Calculate sentiment score in essay

In [23]:

```
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer
analyser = SentimentIntensityAnalyzer()
```

In [24]:

```
neg = []
pos = []
neu = []
compound = []

for a in tqdm(train_data["clean_essays"]) :
    b = analyser.polarity_scores(a)['neg']
    c = analyser.polarity_scores(a)['pos']
    d = analyser.polarity_scores(a)['neu']
    e = analyser.polarity_scores(a)['compound']
    neg.append(b)
    pos.append(c)
    neu.append(d)
    compound.append(e)
```

```
100%|██████████████████████████████████| 109248/109248 [19:38<00:00, 92.72it/s]
```

In [25]:

```
train_data["pos"] = pos
```

In [26]:

```
train_data["neg"] = neg
```

In [27]:

```
train_data["neu"] = neu
```

In [28]:

```
train_data["compound"] = compound
```

In [29]:

```
train_data.head(2)
```

Out[29]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_category	project_
55660	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	CA	2016-04-27 00:27:36	Grades PreK-2	I ha fortunate to use t
76127	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	UT	2016-04-27 00:31:25	Grades 3-5	Imagine 9 y You'r

2 rows × 22 columns

In [30]:

```
train_data.project_grade_category = train_data.project_grade_category.str.replace('\s','_')
train_data['project_grade_category'].value_counts()
train_data.project_grade_category = train_data.project_grade_category.str.replace('-','_')
train_data['project_grade_category'].value_counts()
```

Out[30]:

```
Grades_PreK_2    44225
Grades_3_5       37137
Grades_6_8       16923
Grades_9_12      10963
Name: project_grade_category, dtype: int64
```

In [31]:

```
train_data.teacher_prefix = train_data.teacher_prefix.str.replace('.',' ')
train_data['teacher_prefix'].value_counts()
```

Out[31]:

```
Mrs        57269
Ms         38955
Mr         10648
Teacher     2360
Dr            13
Name: teacher_prefix, dtype: int64
```

In [32]:

```
train_data.teacher_prefix = train_data.teacher_prefix.str.replace('NaN','0')
```

1.8) Train-Test split

In [33]:

```
# train test split

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(train_data, train_data['project_is_approved'],\
                        test_size=0.33, stratify =train_data['project_is_approved'])
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33, stratify=y_train)
```

In [34]:

```
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)
```

1.9) Preparing data for model

In [35]:

```
train_data.columns
```

Out[35]:

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'Date', 'project_grade_category', 'project_essay_1', 'project_essay_2',
       'project_essay_3', 'project_essay_4', 'project_resource_summary',
       'teacher_number_of_previously_posted_projects', 'project_is_approved',
       'clean_categories', 'clean_subcategories', 'clean_titles',
       'clean_essays', 'pos', 'neg', 'neu', 'compound'],
      dtype='object')
```

Vectorizing the categorial features

1.9.1) One hot encode - Clean categories of project_subject_category

In [36]:

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

from sklearn.feature_extraction.text import CountVectorizer

vectorizer_proj = CountVectorizer(vocabulary=list(sorted_pro_sub_cat_dict.keys()), lowercase=False
, binary=True)
vectorizer_proj.fit(X_train['clean_categories'].values)

categories_one_hot_train = vectorizer_proj.transform(X_train['clean_categories'].values)
categories_one_hot_test = vectorizer_proj.transform(X_test['clean_categories'].values)
categories_one_hot_cv = vectorizer_proj.transform(X_cv['clean_categories'].values)

print(vectorizer_proj.get_feature_names())

print("Shape of matrix of Train data after one hot encoding ",categories_one_hot_train.shape)
print("Shape of matrix of Test data after one hot encoding ",categories_one_hot_test.shape)
print("Shape of matrix of CV data after one hot encoding ",categories_one_hot_cv.shape)
```

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

1.9.2) One hot encode - Clean categories of project_sub_subcategories

In [37]:

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

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

sub_categories_one_hot_train = vectorizer_sub_proj.transform(X_train['clean_subcategories'].values
)
sub_categories_one_hot_test = vectorizer_sub_proj.transform(X_test['clean_subcategories'].values)
sub_categories_one_hot_cv = vectorizer_sub_proj.transform(X_cv['clean_subcategories'].values)

print(vectorizer_sub_proj.get_feature_names())

print("Shape of matrix of Train data after one hot encoding ",sub_categories_one_hot_train.shape)
print("Shape of matrix of Test data after one hot encoding ",sub_categories_one_hot_test.shape)
print("Shape of matrix of Cross Validation data after one hot encoding ",sub_categories_one_hot_cv
.shape)
```

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

1.9.3) One hot encode - School states

In [38]:

```
my_counter = Counter()
for state in train_data['school_state'].values:
    my_counter.update(state.split())
```

In [39]:

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

In [40]:

```
## we use count vectorizer to convert the values into one hot encoded features
vectorizer_state = CountVectorizer(vocabulary=list(sorted_school_state_cat_dict.keys()), lowercase
=False, binary=True)
vectorizer_state.fit(X_train['school_state'].values)

school_state_categories_one_hot_train = vectorizer_state.transform(X_train['school_state'].values)
school_state_categories_one_hot_test = vectorizer_state.transform(X_test['school_state'].values)
school_state_categories_one_hot_cv = vectorizer_state.transform(X_cv['school_state'].values)

print(vectorizer_state.get_feature_names())

print("Shape of matrix of Train data after one hot encoding
",school_state_categories_one_hot_train.shape)
print("Shape of matrix of Test data after one hot encoding ",school_state_categories_one_hot_test.
shape)
print("Shape of matrix of Cross Validation data after one hot encoding
",school_state_categories_one_hot_cv.shape)
```

```
['VT', 'WY', 'ND', 'MT', 'RI', 'SD', 'NE', 'DE', 'AK', 'NH', 'WV', 'ME', 'HI', 'DC', 'NM', 'KS', 'I
A', 'ID', 'AR', 'CO', 'MN', 'OR', 'KY', 'MS', 'NV', 'MD', 'CT', 'TN', 'UT', 'AL', 'WI', 'VA', 'AZ',
'NJ', 'OK', 'WA', 'MA', 'LA', 'OH', 'MO', 'IN', 'PA', 'MI', 'SC', 'GA', 'IL', 'NC', 'FL', 'NY', 'TX
', 'CA']
```

```
Shape of matrix of Train data after one hot encoding  (49041, 51)
Shape of matrix of Test data after one hot encoding  (36052, 51)
Shape of matrix of Cross Validation data after one hot encoding  (24155, 51)
```

1.9.4) One hot encode - Teacher_prefix

In [41]:

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

In [42]:

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

In [43]:

```
## we use count vectorizer to convert the values into one hot encoded features
## Unlike the previous Categories this category returns a
## ValueError: np.nan is an invalid document, expected byte or unicode string.
## The link below explains h0w to tackle such discrepancies.
## https://stackoverflow.com/questions/39303912/tfidfvectorizer-in-scikit-learn-valueerror-np-nan-is-an-invalid-document/39308809#39308809
vectorizer_teacher = CountVectorizer(vocabulary=list(sorted_teacher_prefix_cat_dict.keys()), lower
case=False, binary=True)
vectorizer_teacher.fit(X_train['teacher_prefix'].values.astype("U"))

teacher_prefix_categories_one_hot_train = vectorizer_teacher.transform(X_train['teacher_prefix'].v
alues.astype("U"))
teacher_prefix_categories_one_hot_test =
vectorizer_teacher.transform(X_test['teacher_prefix'].values.astype("U"))
teacher_prefix_categories_one_hot_cv = vectorizer_teacher.transform(X_cv['teacher_prefix'].values.
astype("U"))

print(vectorizer_teacher.get_feature_names())

print("Shape of matrix after one hot encoding ",teacher_prefix_categories_one_hot_train.shape)
print("Shape of matrix after one hot encoding ",teacher_prefix_categories_one_hot_test.shape)
print("Shape of matrix after one hot encoding ",teacher_prefix_categories_one_hot_cv.shape)
```

```
['nan', 'Dr', 'Teacher', 'Mr', 'Ms', 'Mrs']
Shape of matrix after one hot encoding  (49041, 6)
Shape of matrix after one hot encoding  (36052, 6)
Shape of matrix after one hot encoding  (24155, 6)
```

1.9.5) One hot encode - project_grade_category

In [44]:

```
my_counter = Counter()
for project_grade in train_data['project_grade_category'].values:
    my_counter.update(project_grade.split())
```

In [45]:

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

In [46]:

```
## we use count vectorizer to convert the values into one hot encoded features
vectorizer_grade = CountVectorizer(vocabulary=list(sorted_project_grade_cat_dict.keys()),
lowercase=False, binary=True)
```

```
vectorizer_grade.fit(X_train['project_grade_category'].values)

project_grade_categories_one_hot_train =
vectorizer_grade.transform(X_train['project_grade_category'].values)
project_grade_categories_one_hot_test = vectorizer_grade.transform(X_test['project_grade_category'
].values)
project_grade_categories_one_hot_cv = vectorizer_grade.transform(X_cv['project_grade_category'].va
lues)

print(vectorizer_grade.get_feature_names())

print("Shape of matrix of Train data after one hot encoding
",project_grade_categories_one_hot_train.shape)
print("Shape of matrix of Test data after one hot encoding ",project_grade_categories_one_hot_test
.shape)
print("Shape of matrix of Cross Validation data after one hot encoding
",project_grade_categories_one_hot_cv.shape)
```

```
['Grades_9_12', 'Grades_6_8', 'Grades_3_5', 'Grades_PreK_2']
Shape of matrix of Train data after one hot encoding  (49041, 4)
Shape of matrix of Test data after one hot encoding  (36052, 4)
Shape of matrix of Cross Validation data after one hot encoding  (24155, 4)
```

Vectorizing the text data

I) Bag of words - with bi-grams with min_df=10 and max_features=5000

Bag of words - Train Data - Essays

In [47]:

```
# We are considering only the words which appeared in at least 10 documents(rows or projects).

vectorizer_bow_essay = CountVectorizer(ngram_range=(2,2), min_df=10, max_features = 5000)
vectorizer_bow_essay.fit(X_train["clean_essays"])
text_bow_train = vectorizer_bow_essay.transform(X_train["clean_essays"])
print("Shape of matrix after one hot encoding ",text_bow_train.shape)
```

```
Shape of matrix after one hot encoding  (49041, 5000)
```

Bag of words - Test Data - Essays

In [48]:

```
text_bow_test = vectorizer_bow_essay.transform(X_test["clean_essays"])
print("Shape of matrix after one hot encoding ",text_bow_test.shape)
```

```
Shape of matrix after one hot encoding  (36052, 5000)
```

Bag of words - CV Data - Essays

In [49]:

```
text_bow_cv = vectorizer_bow_essay.transform(X_cv["clean_essays"])
print("Shape of matrix after one hot encoding ",text_bow_cv.shape)
```

```
Shape of matrix after one hot encoding  (24155, 5000)
```

Bag of words - Train Data - Title

In [50]:

```
vectorizer_bow_title = CountVectorizer(ngram_range=(2,2), min_df=10, max_features = 5000)

vectorizer_bow_title.fit(X_train["clean_titles"])

title_bow_train = vectorizer_bow_title.transform(X_train["clean_titles"])
print("Shape of matrix after one hot encoding ",title_bow_train.shape)
```

```
Shape of matrix after one hot encoding  (49041, 1663)
```

Bag of words - Test Data - Title

In [51]:

```
title_bow_test = vectorizer_bow_title.transform(X_test["clean_titles"])
print("Shape of matrix after one hot encoding ",title_bow_test.shape)
```

```
Shape of matrix after one hot encoding  (36052, 1663)
```

Bag of words - CV Data - Title

In [52]:

```
title_bow_cv = vectorizer_bow_title.transform(X_cv["clean_titles"])
print("Shape of matrix after one hot encoding ",title_bow_cv.shape)
```

```
Shape of matrix after one hot encoding  (24155, 1663)
```

II) TFIDF vectorizer with bi-grams with min_df=10 and max_features=5000

TFIDF - Train Data - Essays

In [53]:

```
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer_tfidf_essay = TfidfVectorizer(ngram_range=(2,2), min_df=10, max_features = 5000)
vectorizer_tfidf_essay.fit(X_train["clean_essays"])

text_tfidf_train = vectorizer_tfidf_essay.transform(X_train["clean_essays"])
print("Shape of matrix after one hot encoding ",text_tfidf_train.shape)
```

```
Shape of matrix after one hot encoding  (49041, 5000)
```

TFIDF - Test Data - Essays

In [54]:

```
text_tfidf_test = vectorizer_tfidf_essay.transform(X_test["clean_essays"])
print("Shape of matrix after one hot encoding ",text_tfidf_test.shape)
```

```
Shape of matrix after one hot encoding  (36052, 5000)
```

TFIDF - CV Data - Essays

In [55]:

```
text_tfidf_cv = vectorizer_tfidf_essay.transform(X_cv["clean_essays"])
print("Shape of matrix after one hot encoding ",text_tfidf_cv.shape)
```

```
Shape of matrix after one hot encoding  (24155, 5000)
```

TFIDF - Train Data - Titles

In [56]:

```
vectorizer_tfidf_titles = TfidfVectorizer(ngram_range=(2,2), min_df=10, max_features = 5000)

vectorizer_tfidf_titles.fit(X_train["clean_titles"])
title_tfidf_train = vectorizer_tfidf_titles.transform(X_train["clean_titles"])
print("Shape of matrix after one hot encoding ",title_tfidf_train.shape)
```

```
Shape of matrix after one hot encoding  (49041, 1663)
```

TFIDF - Test Data - Titles

In [57]:

```
title_tfidf_test = vectorizer_tfidf_titles.transform(X_test["clean_titles"])
print("Shape of matrix after one hot encoding ",title_tfidf_test.shape)
```

```
Shape of matrix after one hot encoding  (36052, 1663)
```

TFIDF - CV Data - Titles

In [58]:

```
title_tfidf_cv = vectorizer_tfidf_titles.transform(X_cv["clean_titles"])
print("Shape of matrix after one hot encoding ",title_tfidf_cv.shape)
```

```
Shape of matrix after one hot encoding  (24155, 1663)
```

III) Using pretrained model - Avg W2V

In [59]:

```
with open (r'C:\Users\lenovo\Downloads\glove_vectors', "rb") as f:
    model = pickle.load(f)
    glove_words = set(model.keys())
```

Train - Essays

In [60]:

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

avg_w2v_vectors_train = [];

for sentence in tqdm(X_train["clean_essays"]): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_train.append(vector)

print(len(avg_w2v_vectors_train))
print(len(avg_w2v_vectors_train[0]))
```

```
100%|████████████████████████████████| 49041/49041 [00:22<00:00, 2164.32it/s]
```

```
49041
300
```

Test - Essays

In [61]:

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

avg_w2v_vectors_test = [];

for sentence in tqdm(X_test["clean_essays"]): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_test.append(vector)

print(len(avg_w2v_vectors_test))
print(len(avg_w2v_vectors_test[0]))
```

```
100%|████████████████████████████████| 36052/36052 [00:16<00:00, 2139.87it/s]
```

```
36052
300
```

CV - Essays

In [62]:

```
avg_w2v_vectors_cv = [];

for sentence in tqdm(X_cv["clean_essays"]): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_cv.append(vector)

print(len(avg_w2v_vectors_cv))
print(len(avg_w2v_vectors_cv[0]))
```

```
100%|████████████████████████████████| 24155/24155 [00:11<00:00, 2023.46it/s]
```

```
24155
300
```

Train - Titles

In [63]:

```
# Similarly you can vectorize for title also

avg_w2v_vectors_titles_train = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train["clean_titles"]): # for each title
```

```
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_titles_train.append(vector)

print(len(avg_w2v_vectors_titles_train))
print(len(avg_w2v_vectors_titles_train[0]))
```

```
100%|██████████████████████████████████| 49041/49041 [00:01<00:00, 38705.02it/s]
```

```
49041
300
```

Test - Titles

In [64]:

```
# Similarly you can vectorize for title also

avg_w2v_vectors_titles_test = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test["clean_titles"]): # for each title
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_titles_test.append(vector)

print(len(avg_w2v_vectors_titles_test))
print(len(avg_w2v_vectors_titles_test[0]))
```

```
100%|██████████████████████████████████| 36052/36052 [00:00<00:00, 36194.71it/s]
```

```
36052
300
```

CV - Titles

In [65]:

```
# Similarly you can vectorize for title also

avg_w2v_vectors_titles_cv = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv["clean_titles"]): # for each title
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_titles_cv.append(vector)

print(len(avg_w2v_vectors_titles_cv))
print(len(avg_w2v_vectors_titles_cv[0]))
```

```
100%|██████████████████████████████████| 24155/24155 [00:00<00:00, 36764.04it/s]
```

```
24155
300
```

IV) Using pretrained model - TFIDF weighted W2V

Train - Essays

In [66]:

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

In [67]:

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

print(len(tfidf_w2v_vectors_train))
print(len(tfidf_w2v_vectors_train[0]))
```

```
100%|██████████████████████████████████| 49041/49041 [02:49<00:00, 290.06it/s]
```

```
49041
300
```

Test - Essays

In [68]:

```
# compute average word2vec for each review.

tfidf_w2v_vectors_test = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test["clean_essays"]): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_test.append(vector)

print(len(tfidf_w2v_vectors_test))
print(len(tfidf_w2v_vectors_test[0]))
```

```
100%|████████████████████████████████| 36052/36052 [02:01<00:00, 296.63it/s]
```

```
36052
300
```

CV - Essays

In [69]:

```
# compute average word2vec for each review.

tfidf_w2v_vectors_cv = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv["clean_essays"]): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_cv.append(vector)

print(len(tfidf_w2v_vectors_cv))
print(len(tfidf_w2v_vectors_cv[0]))
```

```
100%|████████████████████████████████| 24155/24155 [01:23<00:00, 290.94it/s]
```

```
24155
300
```

Train - Titles

In [70]:

```
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train["clean_titles"])
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())
```

In [71]:

```
# compute average word2vec for each review.

tfidf_w2v_vectors_titles_train = [];

for sentence in tqdm(X_train["clean_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):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf 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_titles_train.append(vector)
```

```
print(len(tfidf_w2v_vectors_titles_train))
print(len(tfidf_w2v_vectors_titles_train[0]))
```

```
100%|████████████████████████████████| 49041/49041 [00:02<00:00, 16921.58it/s]
```

```
49041
300
```

Test - Titles

In [72]:

```
# compute average word2vec for each review.

tfidf_w2v_vectors_titles_test = [];

for sentence in tqdm(X_test["clean_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):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf 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_titles_test.append(vector)

print(len(tfidf_w2v_vectors_titles_test))
print(len(tfidf_w2v_vectors_titles_test[0]))
```

```
100%|████████████████████████████████| 36052/36052 [00:02<00:00, 17559.72it/s]
```

```
36052
300
```

CV - Titles

In [73]:

```
# compute average word2vec for each review.

tfidf_w2v_vectors_titles_cv = [];

for sentence in tqdm(X_cv["clean_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):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf 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_titles_cv.append(vector)

print(len(tfidf_w2v_vectors_titles_cv))
print(len(tfidf_w2v_vectors_titles_cv[0]))
```

```
100%|████████████████████████████████| 24155/24155 [00:01<00:00, 17592.41it/s]
```

```
24155
300
```

1.8) Vectorizing numerical features

In [74]:

```
# https://stackoverflow.com/questions/22407798/how-to-reset-a-dataframes-indexes-for-all-groups-in-one-step
price_data = resource_data.groupby('id').agg({'price':'sum', 'quantity':'sum'}).reset_index()
price_data.head(2)
```

Out[74]:

	id	price	quantity
0	p000001	459.56	7
1	p000002	515.89	21

In [75]:

```
# join two dataframes in python:
X_train = pd.merge(X_train, price_data, on='id', how='left')
X_test = pd.merge(X_test, price_data, on='id', how='left')
X_cv = pd.merge(X_cv, price_data, on='id', how='left')
```

1) Price

In [76]:

```
from sklearn.preprocessing import Normalizer

normalizer = Normalizer()

# normalizer.fit(X_train['price'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.

normalizer.fit(X_train['price'].values.reshape(-1,1))

price_train = normalizer.transform(X_train['price'].values.reshape(-1,1))
price_cv = normalizer.transform(X_cv['price'].values.reshape(-1,1))
price_test = normalizer.transform(X_test['price'].values.reshape(-1,1))

print("After vectorizations")
print(price_train.shape, y_train.shape)
print(price_cv.shape, y_cv.shape)
print(price_test.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(49041, 1) (49041,)
(24155, 1) (24155,)
(36052, 1) (36052,)
====================================================================================================
```

2) Quantity

In [77]:

```
normalizer = Normalizer()
```

```
# normalizer.fit(X_train['price'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.

normalizer.fit(X_train['quantity'].values.reshape(-1,1))

quantity_train = normalizer.transform(X_train['quantity'].values.reshape(-1,1))
quantity_cv = normalizer.transform(X_cv['quantity'].values.reshape(-1,1))
quantity_test = normalizer.transform(X_test['quantity'].values.reshape(-1,1))

print("After vectorizations")
print(quantity_train.shape, y_train.shape)
print(quantity_cv.shape, y_cv.shape)
print(quantity_test.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(49041, 1) (49041,)
(24155, 1) (24155,)
(36052, 1) (36052,)
====================================================================================================
```

3) Project proposal previously by Teacher

In [78]:

```
normalizer = Normalizer()

# normalizer.fit(X_train['price'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.

normalizer.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))

prev_projects_train = normalizer.transform(X_train['teacher_number_of_previously_posted_projects']
.values.reshape(-1,1))
prev_projects_cv =
normalizer.transform(X_cv['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))
prev_projects_test = normalizer.transform(X_test['teacher_number_of_previously_posted_projects'].v
alues.reshape(-1,1))

print("After vectorizations")
print(prev_projects_train.shape, y_train.shape)
print(prev_projects_cv.shape, y_cv.shape)
print(prev_projects_test.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(49041, 1) (49041,)
(24155, 1) (24155,)
(36052, 1) (36052,)
====================================================================================================
```

3) Essay sentiment - pos

In [79]:

```
normalizer = Normalizer()

normalizer.fit(X_train['pos'].values.reshape(-1,1))

essay_sent_pos_train = normalizer.transform(X_train['pos'].values.reshape(-1,1))
```

```
essay_sent_pos_cv = normalizer.transform(X_cv['pos'].values.reshape(-1,1))
essay_sent_pos_test = normalizer.transform(X_test['pos'].values.reshape(-1,1))

print("After vectorizations")
print(essay_sent_pos_train.shape, y_train.shape)
print(essay_sent_pos_cv.shape, y_cv.shape)
print(essay_sent_pos_test.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(49041, 1) (49041,)
(24155, 1) (24155,)
(36052, 1) (36052,)
====================================================================================================
```

3) Essay sentiment - neg

In [80]:

```
normalizer = Normalizer()

normalizer.fit(X_train['neg'].values.reshape(-1,1))

essay_sent_neg_train = normalizer.transform(X_train['neg'].values.reshape(-1,1))
essay_sent_neg_cv = normalizer.transform(X_cv['neg'].values.reshape(-1,1))
essay_sent_neg_test = normalizer.transform(X_test['neg'].values.reshape(-1,1))

print("After vectorizations")
print(essay_sent_neg_train.shape, y_train.shape)
print(essay_sent_neg_cv.shape, y_cv.shape)
print(essay_sent_neg_test.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(49041, 1) (49041,)
(24155, 1) (24155,)
(36052, 1) (36052,)
====================================================================================================
```

3) Essay sentiment - neu

In [81]:

```
normalizer = Normalizer()

normalizer.fit(X_train['neu'].values.reshape(-1,1))

essay_sent_neu_train = normalizer.transform(X_train['neu'].values.reshape(-1,1))
essay_sent_neu_cv = normalizer.transform(X_cv['neu'].values.reshape(-1,1))
essay_sent_neu_test = normalizer.transform(X_test['neu'].values.reshape(-1,1))

print("After vectorizations")
print(essay_sent_neu_train.shape, y_train.shape)
print(essay_sent_neu_cv.shape, y_cv.shape)
print(essay_sent_neu_test.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(49041, 1) (49041,)
(24155, 1) (24155,)
(36052, 1) (36052,)
====================================================================================================
```

3) Essay sentiment - compound

In [82]:

```
normalizer = Normalizer()

normalizer.fit(X_train['compound'].values.reshape(-1,1))

essay_sent_compound_train = normalizer.transform(X_train['compound'].values.reshape(-1,1))
essay_sent_compound_cv = normalizer.transform(X_cv['compound'].values.reshape(-1,1))
essay_sent_compound_test = normalizer.transform(X_test['compound'].values.reshape(-1,1))

print("After vectorizations")
print(essay_sent_compound_train.shape, y_train.shape)
print(essay_sent_compound_cv.shape, y_cv.shape)
print(essay_sent_compound_test.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(49041, 1) (49041,)
(24155, 1) (24155,)
(36052, 1) (36052,)
====================================================================================================
```

2) Logistic Regression

Set 1: Categorical, Numerical features + Project_title(BOW) + Preprocessed_essay (BOW with bi-grams with min_df=10 and max_features=5000)

In [83]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack

X_tr = hstack((categories_one_hot_train, sub_categories_one_hot_train,
school_state_categories_one_hot_train, project_grade_categories_one_hot_train,
teacher_prefix_categories_one_hot_train, price_train, quantity_train, prev_projects_train, title_bo
w_train, text_bow_train)).tocsr()
X_te = hstack((categories_one_hot_test, sub_categories_one_hot_test,
school_state_categories_one_hot_test, project_grade_categories_one_hot_test,
teacher_prefix_categories_one_hot_test, price_test, quantity_test, prev_projects_test,
title_bow_test, text_bow_test)).tocsr()
X_cv = hstack((categories_one_hot_cv, sub_categories_one_hot_cv,
school_state_categories_one_hot_cv, project_grade_categories_one_hot_cv,
teacher_prefix_categories_one_hot_cv, price_cv, quantity_cv, prev_projects_cv, title_bow_cv, text_b
ow_cv)).tocsr()
```

In [84]:

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

```
Final Data matrix
(49041, 6766) (49041,)
(24155, 6766) (24155,)
(36052, 6766) (36052,)
====================================================================================================
```

A) GridSearch CV

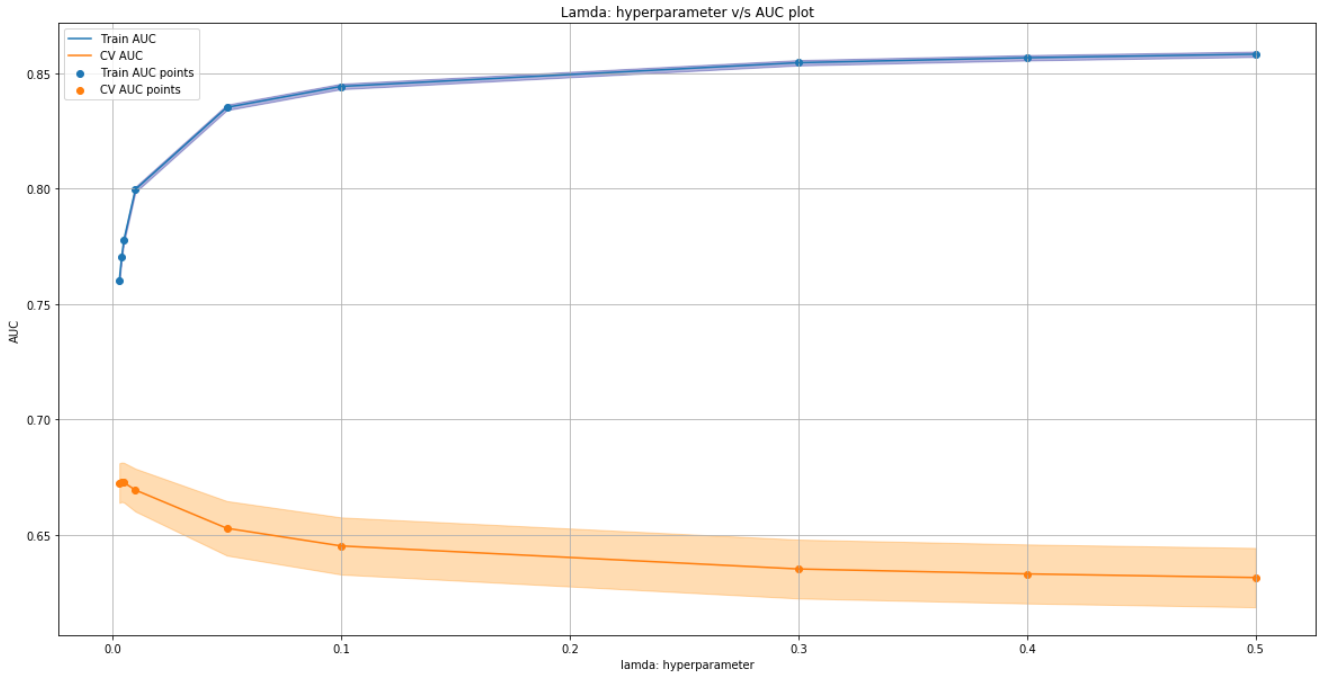
In [85]:

```
from sklearn.model_selection import GridSearchCV
```

```
from sklearn.linear_model import LogisticRegression
```

In [86]:

```
lr = LogisticRegression(random_state=4, class_weight='balanced')

C = [0.5, 0.4, 0.3, 0.1, 0.05, 0.01, 0.005, 0.004, 0.003]

lamda = {"C":[0.5, 0.4, 0.3, 0.1, 0.05, 0.01, 0.005, 0.004, 0.003]}

clf = GridSearchCV(lr, lamda, cv= 10, scoring='roc_auc')

clf.fit(X_tr, y_train)

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

plt.figure(figsize=(20,10))

plt.plot(lamda["C"], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(lamda["C"],train_auc - train_auc_std,train_auc + train_auc_std,alpha=0.3,col
or='darkblue')

plt.plot(lamda["C"], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(lamda["C"],cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.3,color='darkoran
ge')

plt.scatter(lamda["C"], train_auc, label='Train AUC points')
plt.scatter(lamda["C"], cv_auc, label='CV AUC points')


plt.legend()
plt.xlabel("lamda: hyperparameter")
plt.ylabel("AUC")
plt.title("Lamda: hyperparameter v/s AUC plot")
plt.grid()
plt.show()
```



Summary - 0.005 is chosen as the best hyperparameter value.

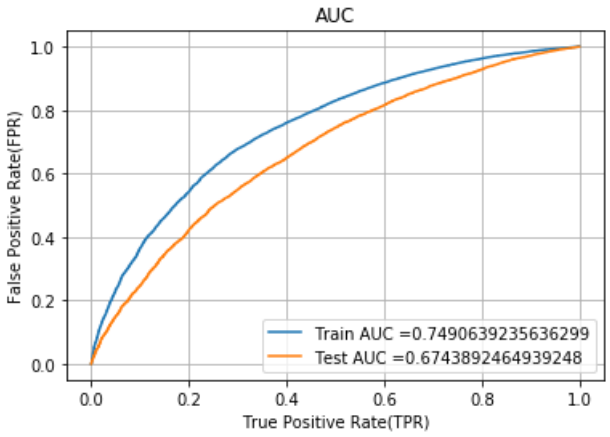
B) Train the model using the best hyper parameter value

In [87]:

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

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

    return y_data_pred
```

In [88]:

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


model = LogisticRegression(C = 0.005)

model.fit(X_tr, y_train)

# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs

y_train_pred = batch_predict(model, X_tr)
y_test_pred = batch_predict(model, X_te)

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)

plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
```



C) Confusion matrix

In [89]:

```
def predict(proba, threshould, fpr, tpr):

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

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

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

Train data

In [90]:

```
print("="*100)
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr)))
```

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

Train confusion matrix
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.804
[[ 3713  3713]
 [ 7088 34527]]
```

In [91]:

```
conf_matr_df_train_1 = pd.DataFrame(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds,
train_fpr, train_fpr)), range(2),range(2))
```

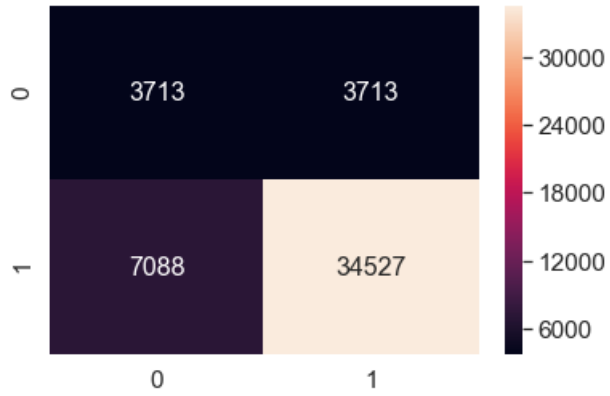
```
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.804
```

In [92]:

```
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_train_1, annot=True,annot_kws={"size": 16}, fmt='g')
```

Out[92]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x57207a20>
```



Test data

In [93]:

```
print("="*100)
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr)))
```

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

Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24999999161092998 for threshold 0.827
[[ 2958  2501]
 [ 9011 21582]]
```

In [94]:

```
conf_matr_df_test_1 = pd.DataFrame(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, tes
t_fpr, test_fpr)), range(2),range(2))
```

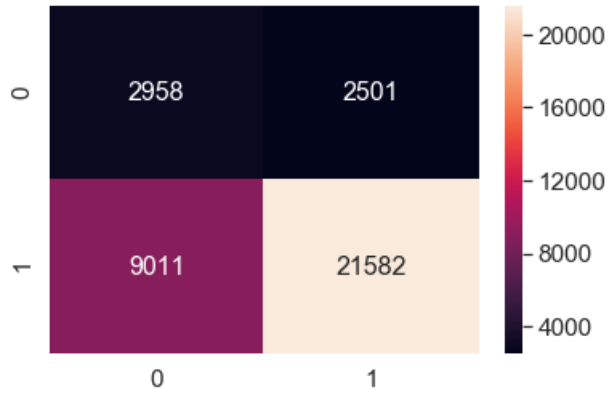
```
the maximum value of tpr*(1-fpr) 0.24999999161092998 for threshold 0.827
```

In [95]:

```
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_test_1, annot=True,annot_kws={"size": 16}, fmt='g')
```

Out[95]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x4dfd7668>
```



Set 2 : Categorical, Numerical features + Project_title(TFIDF) + Preprocessed_essay (TFIDF with bi-grams with min_df=10 and max_features=5000)

In [96]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack

X_tr = hstack((categories_one_hot_train, sub_categories_one_hot_train,
school_state_categories_one_hot_train, project_grade_categories_one_hot_train,
teacher_prefix_categories_one_hot_train, price_train, quantity_train, prev_projects_train, text_tfi
df_train, title_tfidf_train)).tocsr()
X_te = hstack((categories_one_hot_test, sub_categories_one_hot_test,
school_state_categories_one_hot_test, project_grade_categories_one_hot_test,
teacher_prefix_categories_one_hot_test, price_test, quantity_test, prev_projects_test,
text_tfidf_test, title_tfidf_test)).tocsr()
X_cv = hstack((categories_one_hot_cv, sub_categories_one_hot_cv,
school_state_categories_one_hot_cv, project_grade_categories_one_hot_cv,
teacher_prefix_categories_one_hot_cv, price_cv, quantity_cv, prev_projects_cv, text_tfidf_cv, title
_tfidf_cv)).tocsr()
```

In [97]:

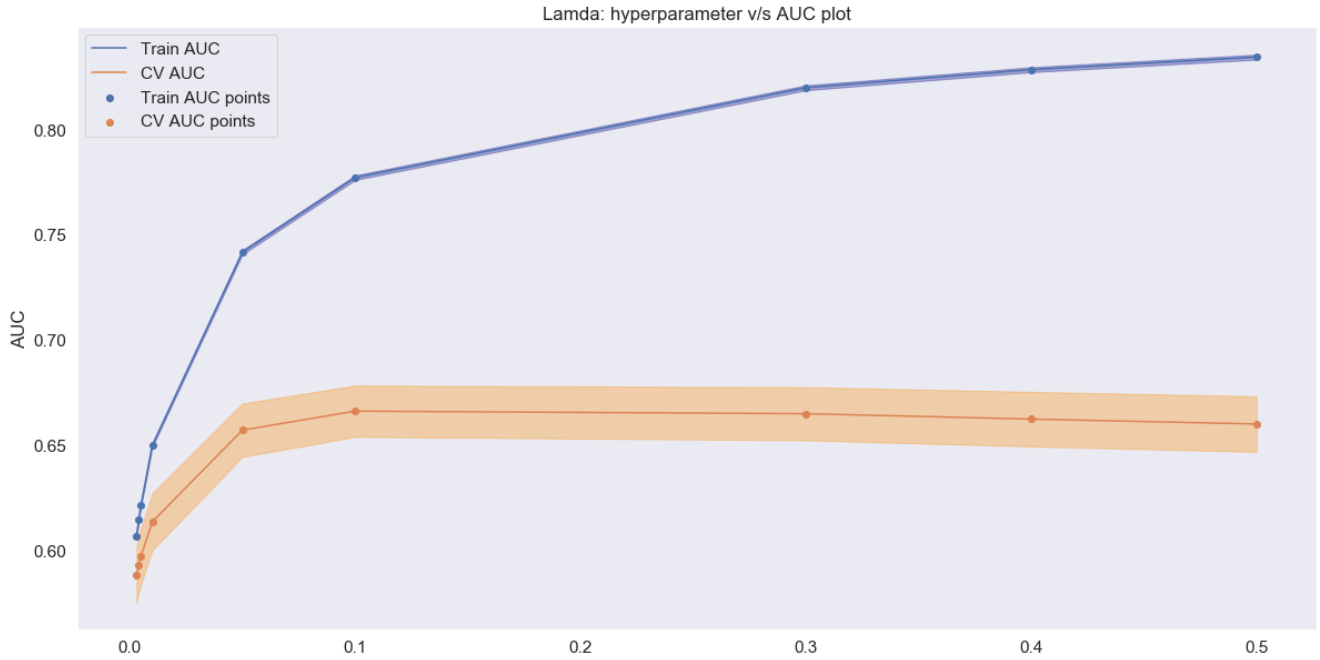
```
print("Final Data matrix")
print(X_tr.shape, y_train.shape)
print(X_cv.shape, y_cv.shape)
print(X_te.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(49041, 6766) (49041,)
(24155, 6766) (24155,)
(36052, 6766) (36052,)
==========================================================================================
```

A) GridSearch

In [98]:

```
lr = LogisticRegression(random_state=4, class_weight='balanced')

C = [0.5, 0.4, 0.3, 0.1, 0.05, 0.01, 0.005, 0.004, 0.003]

lamda = {"C":[0.5, 0.4, 0.3, 0.1, 0.05, 0.01, 0.005, 0.004, 0.003]}

clf = GridSearchCV(lr, lamda, cv= 10, scoring='roc_auc')

clf.fit(X_tr, y_train)

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

plt.figure(figsize=(20,10))

plt.plot(lamda["C"], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(lamda["C"],train_auc - train_auc_std,train_auc + train_auc_std,alpha=0.3,col
or='darkblue')

plt.plot(lamda["C"], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(lamda["C"],cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.3,color='darkoran
ge')

plt.scatter(lamda["C"], train_auc, label='Train AUC points')
plt.scatter(lamda["C"], cv_auc, label='CV AUC points')


plt.legend()
plt.xlabel("lamda: hyperparameter")
plt.ylabel("AUC")
plt.title("Lamda: hyperparameter v/s AUC plot")
plt.grid()
plt.show()
```

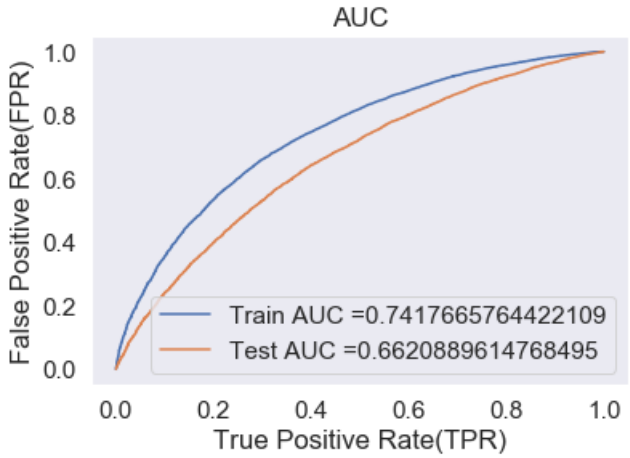


0.1 chosen for the best hyperparameter

B) Train the model using the best hyperparameter value

In [99]:

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


model = LogisticRegression(C = 0.1)

model.fit(X_tr, y_train)

# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs

y_train_pred = batch_predict(model, X_tr)
y_test_pred = batch_predict(model, X_te)

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)

plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
```



C) Confusion matrix

Train data

In [100]:

```
print("="*100)
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr)))
```

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

Train confusion matrix
```

```
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.815
[[ 3713  3713]
 [ 7492 34123]]
```

In [101]:

```
conf_matr_df_train_2 = pd.DataFrame(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds,
train_fpr, train_fpr)), range(2),range(2))
```

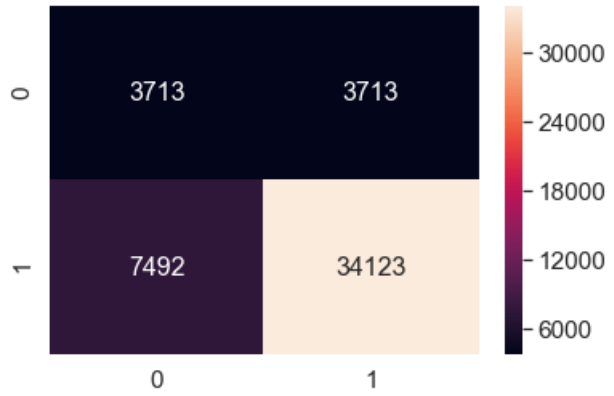
```
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.815
```

In [102]:

```
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_train_2, annot=True,annot_kws={"size": 16}, fmt='g')
```

Out[102]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x4d2c13c8>
```



Test data

In [103]:

```
print("="*100)
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr)))
```

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

Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24999999161092998 for threshold 0.835
[[ 2994  2465]
 [ 9623 20970]]
```

In [104]:

```
conf_matr_df_test_2 = pd.DataFrame(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, tes
t_fpr, test_fpr)), range(2),range(2))
```

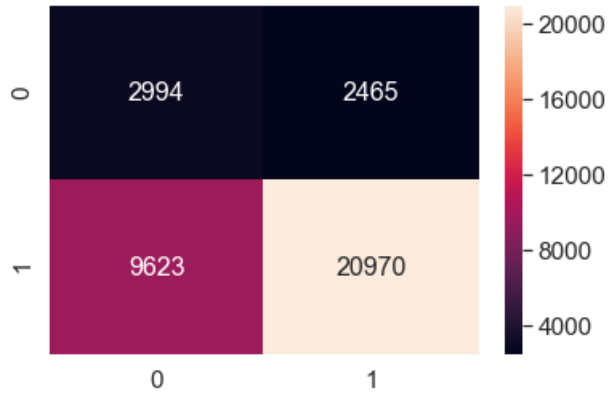
```
the maximum value of tpr*(1-fpr) 0.24999999161092998 for threshold 0.835
```

In [105]:

```
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_test_2, annot=True,annot_kws={"size": 16}, fmt='g')
```

Out[105]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x4c29b4e0>
```



Set 3 : Categorical, Numerical features + Project_title(AVG W2V) + Preprocessed_essay (AVG W2V)

In [106]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack

X_tr = hstack((categories_one_hot_train, sub_categories_one_hot_train,
school_state_categories_one_hot_train, project_grade_categories_one_hot_train,
teacher_prefix_categories_one_hot_train, price_train, quantity_train, prev_projects_train, avg_w2v_
vectors_train, avg_w2v_vectors_titles_train)).tocsr()
X_te = hstack((categories_one_hot_test, sub_categories_one_hot_test,
school_state_categories_one_hot_test, project_grade_categories_one_hot_test,
teacher_prefix_categories_one_hot_test, price_test, quantity_test, prev_projects_test,
avg_w2v_vectors_test, avg_w2v_vectors_titles_test)).tocsr()
X_cv = hstack((categories_one_hot_cv, sub_categories_one_hot_cv,
school_state_categories_one_hot_cv, project_grade_categories_one_hot_cv,
teacher_prefix_categories_one_hot_cv, price_cv, quantity_cv, prev_projects_cv, avg_w2v_vectors_cv,
avg_w2v_vectors_titles_cv)).tocsr()
```

In [107]:

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

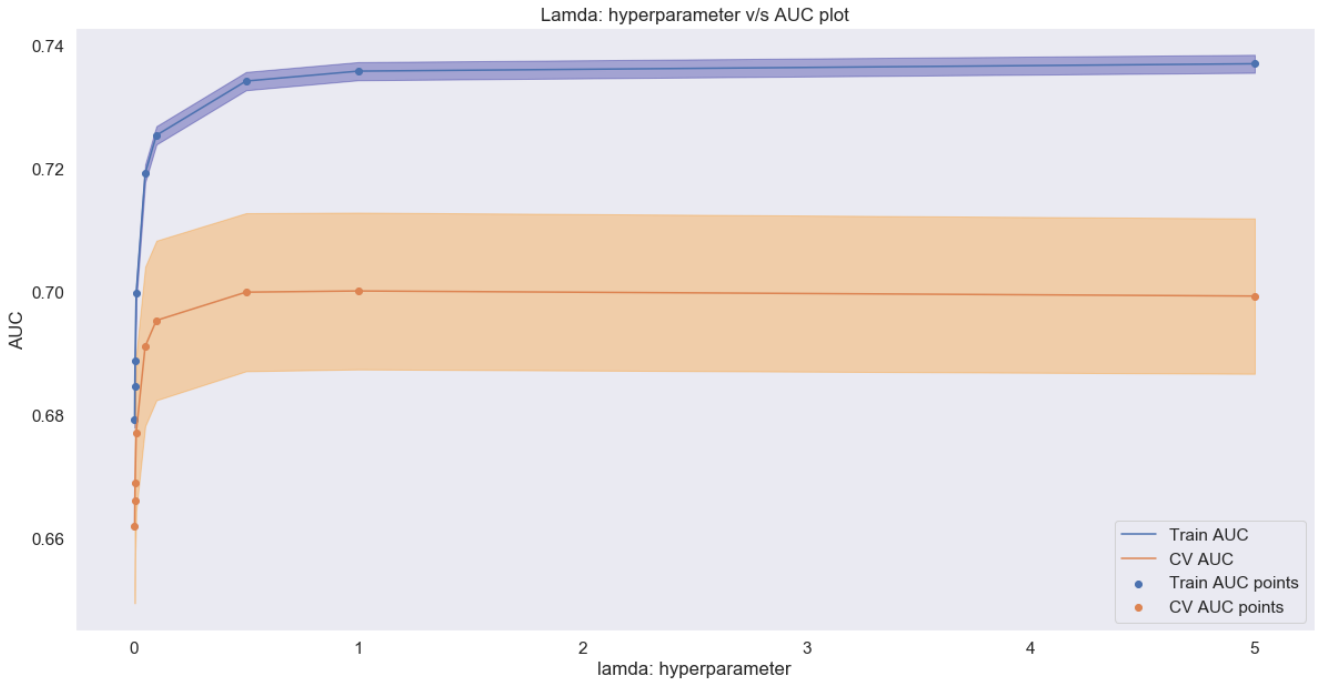
```
Final Data matrix
(49041, 703) (49041,)
(24155, 703) (24155,)
(36052, 703) (36052,)
====================================================================================================
```

A) GridSearch CV

In [108]:

```
lr = LogisticRegression(random_state=4, class_weight='balanced')

C = [5, 1, 0.5, 0.1, 0.05, 0.01, 0.005, 0.004, 0.003]

lamda = {"C":[5, 1, 0.5, 0.1, 0.05, 0.01, 0.005, 0.004, 0.003]}

clf = GridSearchCV(lr, lamda, cv= 10, scoring='roc_auc')

clf.fit(X_tr, y_train)

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

```
cv_auc_std= clf.cv_results_['std_test_score']

plt.figure(figsize=(20,10))

plt.plot(lamda["C"], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(lamda["C"],train_auc - train_auc_std,train_auc + train_auc_std,alpha=0.3,col
or='darkblue')

plt.plot(lamda["C"], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(lamda["C"],cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.3,color='darkoran
ge')

plt.scatter(lamda["C"], train_auc, label='Train AUC points')
plt.scatter(lamda["C"], cv_auc, label='CV AUC points')


plt.legend()
plt.xlabel("lamda: hyperparameter")
plt.ylabel("AUC")
plt.title("Lamda: hyperparameter v/s AUC plot")
plt.grid()
plt.show()
```



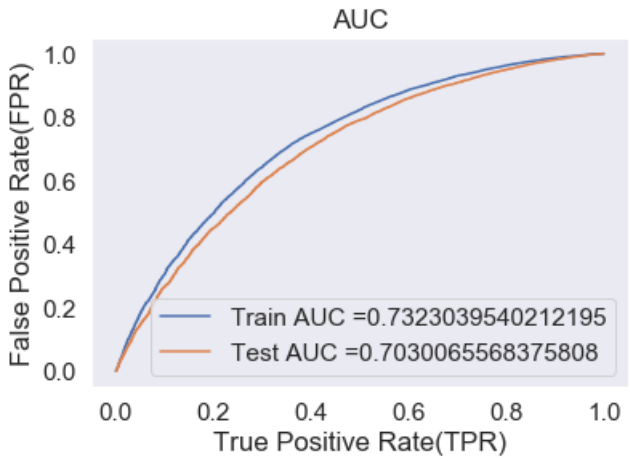
Summary - 1.0 chosen for the best hyperparameter value

B) Train the model using the best hyperparameter value

In [109]:

```
model = LogisticRegression(C = 1.0)

model.fit(X_tr, y_train)

# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs

y_train_pred = batch_predict(model, X_tr)
y_test_pred = batch_predict(model, X_te)

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)

plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
```

```
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
```



C) Confusion matrix

Train data

In [110]:

```
print("="*100)
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr)))
```

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

Train confusion matrix
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.787
[[ 3713  3713]
 [ 7364 34251]]
```

In [111]:

```
conf_matr_df_train_3 = pd.DataFrame(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds,
train_fpr, train_fpr)), range(2),range(2))
```

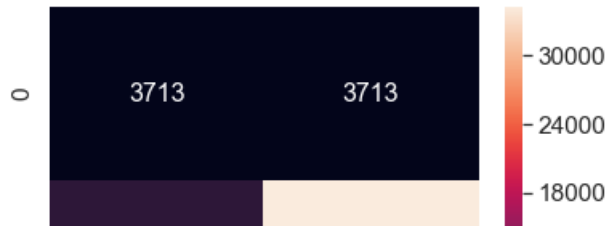
```
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.787
```

In [112]:

```
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_train_3, annot=True,annot_kws={"size": 16}, fmt='g')
```

Out[112]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x489e6080>
```



Test data

In [113]:

```
print("="*100)
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr)))
```

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

Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24999999161092998 for threshold 0.832
[[ 3317  2142]
 [ 9206 21387]]
```

In [114]:

```
conf_matr_df_test_3 = pd.DataFrame(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, tes
t_fpr, test_fpr)), range(2),range(2))
```

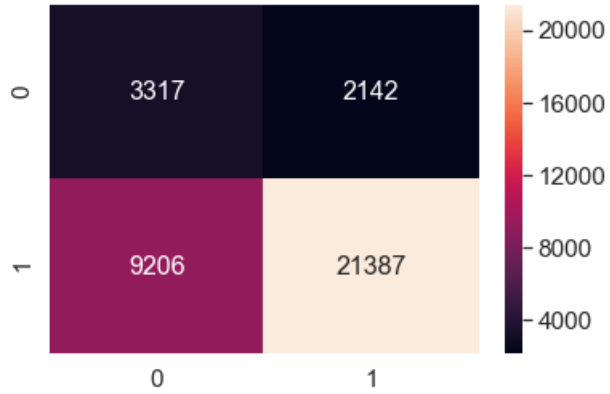
```
the maximum value of tpr*(1-fpr) 0.24999999161092998 for threshold 0.832
```

In [115]:

```
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_test_3, annot=True,annot_kws={"size": 16}, fmt='g')
```

Out[115]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x285ff320>
```



Set 4 : Categorical, Numerical features + Project_title(TFIDF W2V) + Preprocessed_essay (TFIDF W2V)

In [116]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack

X_tr = hstack((categories_one_hot_train, sub_categories_one_hot_train,
school_state_categories_one_hot_train, project_grade_categories_one_hot_train,
teacher_prefix_categories_one_hot_train, price_train, quantity_train, prev_projects_train, tfidf_w2
v_vectors_train, tfidf_w2v_vectors_titles_train)).tocsr()
X_te = hstack((categories_one_hot_test, sub_categories_one_hot_test
```

```
X_te = hstack((categories_one_hot_test, sub_categories_one_hot_test,
school_state_categories_one_hot_test, project_grade_categories_one_hot_test,
teacher_prefix_categories_one_hot_test, price_test, quantity_test, prev_projects_test,
tfidf_w2v_vectors_test, tfidf_w2v_vectors_titles_test)).tocsr()
X_cv = hstack((categories_one_hot_cv, sub_categories_one_hot_cv,
school_state_categories_one_hot_cv, project_grade_categories_one_hot_cv,
teacher_prefix_categories_one_hot_cv, price_cv, quantity_cv, prev_projects_cv,
tfidf_w2v_vectors_cv, tfidf_w2v_vectors_titles_cv)).tocsr()
```

In [117]:

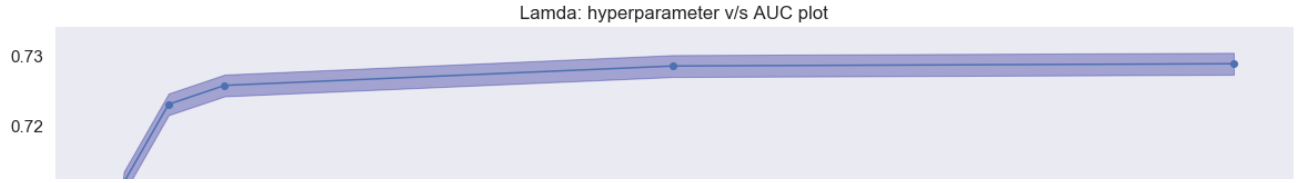
```
print("Final Data matrix")
print(X_tr.shape, y_train.shape)
print(X_cv.shape, y_cv.shape)
print(X_te.shape, y_test.shape)
print("="*100)
```

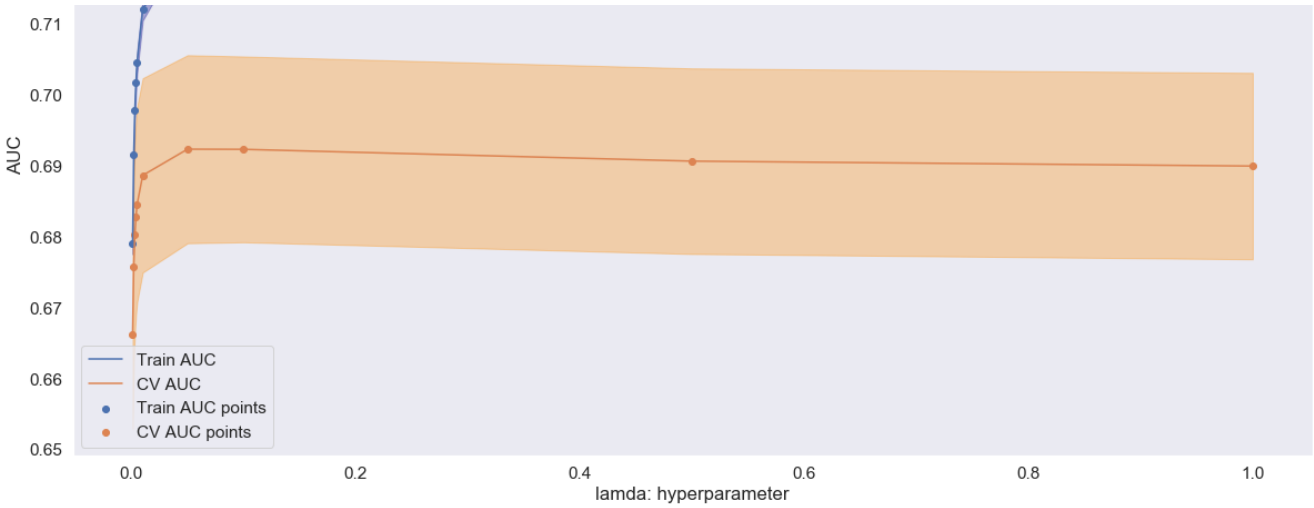
```
Final Data matrix
(49041, 703) (49041,)
(24155, 703) (24155,)
(36052, 703) (36052,)
====================================================================================================
```

A) GridSearch CV

In [118]:

```
lr = LogisticRegression(random_state=4, class_weight='balanced')

C = [1, 0.5, 0.1, 0.05, 0.01, 0.005, 0.004, 0.003, 0.002, 0.001]

lamda = {"C":[1, 0.5, 0.1, 0.05, 0.01, 0.005, 0.004, 0.003, 0.002, 0.001]}

clf = GridSearchCV(lr, lamda, cv= 10, scoring='roc_auc')

clf.fit(X_tr, y_train)

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

plt.figure(figsize=(20,10))

plt.plot(lamda["C"], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(lamda["C"],train_auc - train_auc_std,train_auc + train_auc_std,alpha=0.3,col
or='darkblue')

plt.plot(lamda["C"], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(lamda["C"],cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.3,color='darkoran
ge')

plt.scatter(lamda["C"], train_auc, label='Train AUC points')
plt.scatter(lamda["C"], cv_auc, label='CV AUC points')


plt.legend()
plt.xlabel("lamda: hyperparameter")
plt.ylabel("AUC")
plt.title("Lamda: hyperparameter v/s AUC plot")
plt.grid()
plt.show()
```



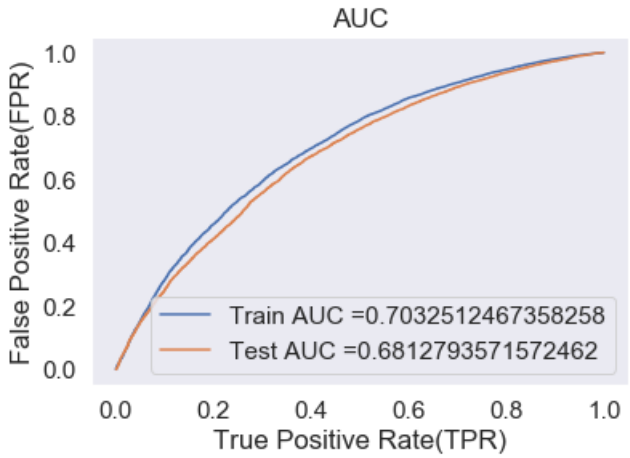


Summary - 0.01 chosen for the best hyperparameter value

B) Train the model using the best hyperparameter value

In [119]:

```
model = LogisticRegression(C = 0.01)

model.fit(X_tr, y_train)

# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs

y_train_pred = batch_predict(model, X_tr)
y_test_pred = batch_predict(model, X_te)

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)

plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
```



C) Confusion matrix

Train data

In [120]:

```
print("="*100)
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr)))
```

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

Train confusion matrix
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.809
[[ 3713  3713]
 [ 8887 32728]]
```

In [121]:

```
conf_matr_df_train_4 = pd.DataFrame(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds,
train_fpr, train_fpr)), range(2),range(2))
```

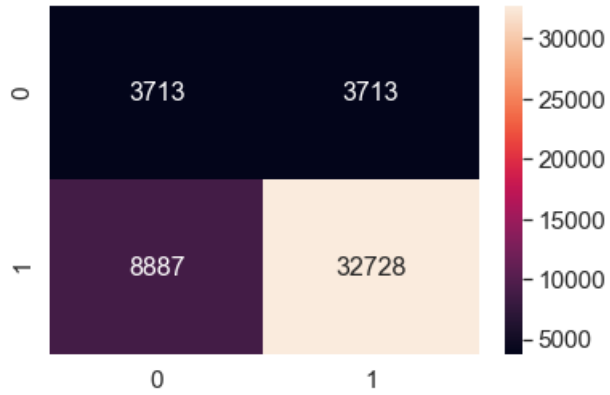
```
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.809
```

In [122]:

```
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_train_4, annot=True,annot_kws={"size": 16}, fmt='g')
```

Out[122]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x4a146eb8>
```



Test data

In [123]:

```
print("="*100)
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr)))
```

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

Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24999999161092998 for threshold 0.839
[[ 3439  2020]
 [10941 19652]]
```

In [124]:

```
conf_matr_df_test_4 = pd.DataFrame(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, tes
t_fpr, test_fpr)), range(2),range(2))
```

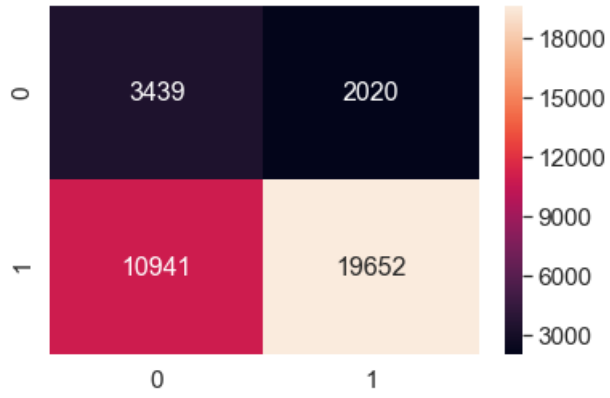
```
the maximum value of tpr*(1-fpr) 0.24999999161092998 for threshold 0.839
```

In [125]:

```
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_test_4, annot=True,annot_kws={"size": 16}, fmt='g')
```

Out[125]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x4af400f0>
```



Set 5 : Categorical features, Numerical features & Essay Sentiments

In [127]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack

X_tr = hstack((categories_one_hot_train, sub_categories_one_hot_train,
school_state_categories_one_hot_train, project_grade_categories_one_hot_train,
teacher_prefix_categories_one_hot_train, price_train, quantity_train, prev_projects_train, essay_se
nt_pos_train, essay_sent_neg_train, essay_sent_neu_train, essay_sent_compound_train)).tocsr()
X_te = hstack((categories_one_hot_test, sub_categories_one_hot_test,
school_state_categories_one_hot_test, project_grade_categories_one_hot_test,
teacher_prefix_categories_one_hot_test, price_test, quantity_test, prev_projects_test,
essay_sent_pos_test, essay_sent_neg_test, essay_sent_neu_test, essay_sent_compound_test)).tocsr()
X_cv = hstack((categories_one_hot_cv, sub_categories_one_hot_cv,
school_state_categories_one_hot_cv, project_grade_categories_one_hot_cv,
teacher_prefix_categories_one_hot_cv, price_cv, quantity_cv, prev_projects_cv, essay_sent_pos_cv, e
ssay_sent_neg_cv, essay_sent_neu_cv, essay_sent_compound_cv)).tocsr()
```

In [128]:

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

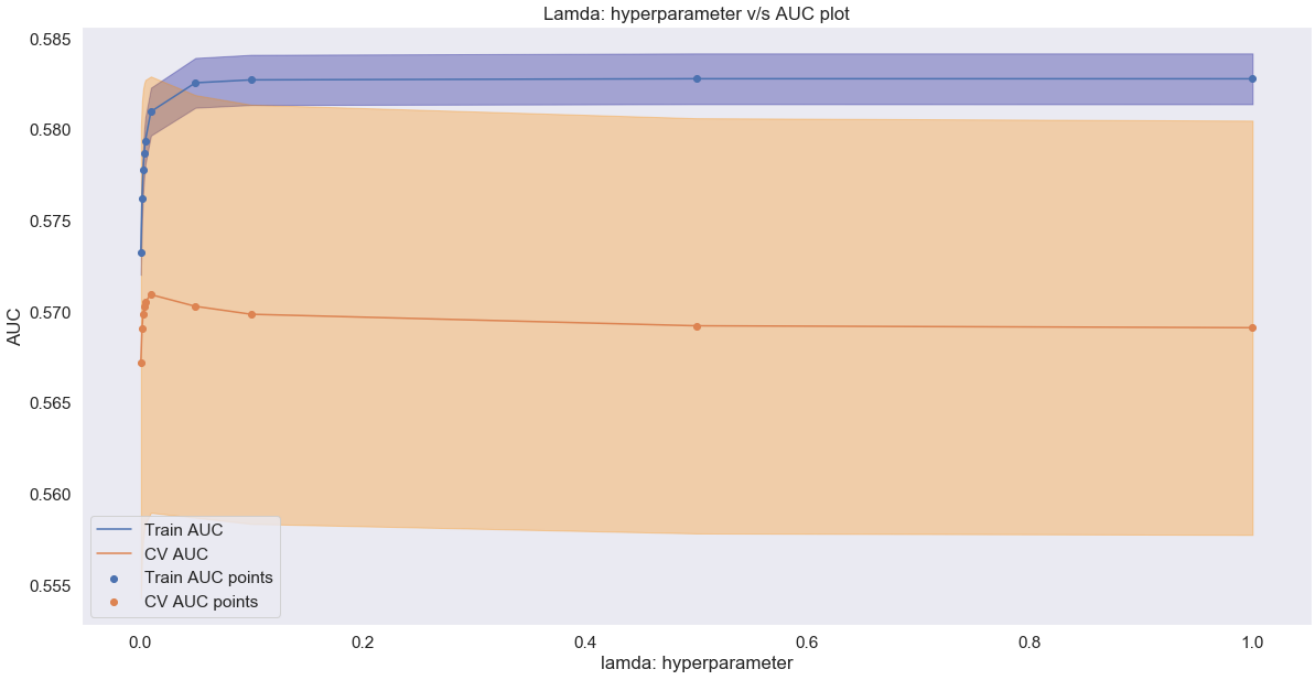
```
Final Data matrix
(49041, 107) (49041,)
(24155, 107) (24155,)
(36052, 107) (36052,)
====================================================================================================
```

GridSearch CV

In [129]:

```
lr = LogisticRegression(random_state=4, class_weight='balanced')
```

```
lr = LogisticRegression(random_state=4, class_weight='balanced')

C = [1, 0.5, 0.1, 0.05, 0.01, 0.005, 0.004, 0.003, 0.002, 0.001]

lamda = {"C":[1, 0.5, 0.1, 0.05, 0.01, 0.005, 0.004, 0.003, 0.002, 0.001]}

clf = GridSearchCV(lr, lamda, cv= 10, scoring='roc_auc')

clf.fit(X_tr, y_train)

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

plt.figure(figsize=(20,10))

plt.plot(lamda["C"], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(lamda["C"],train_auc - train_auc_std,train_auc + train_auc_std,alpha=0.3,col
or='darkblue')

plt.plot(lamda["C"], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(lamda["C"],cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.3,color='darkoran
ge')

plt.scatter(lamda["C"], train_auc, label='Train AUC points')
plt.scatter(lamda["C"], cv_auc, label='CV AUC points')


plt.legend()
plt.xlabel("lamda: hyperparameter")
plt.ylabel("AUC")
plt.title("Lamda: hyperparameter v/s AUC plot")
plt.grid()
plt.show()
```



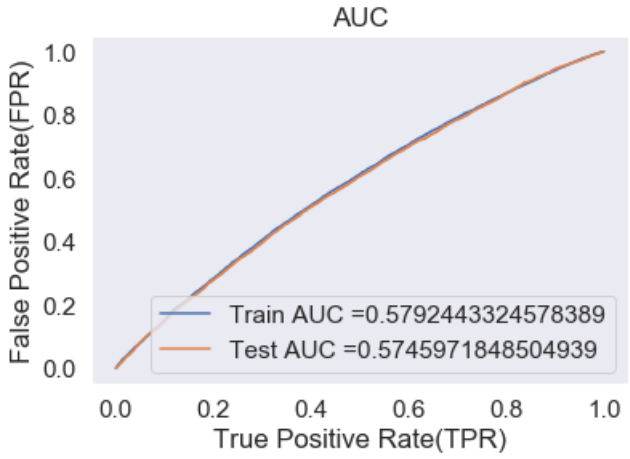
0.01 chosen the best hyperparameter value.

B) Train the model using the best hyperparameter value

In [130]:

```
model = LogisticRegression(C = 0.01)

model.fit(X_tr, y_train)
```

```
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs

y_train_pred = batch_predict(model, X_tr)
y_test_pred = batch_predict(model, X_te)

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)

plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
```



C) Confusion matrix

Train data

In [131]:

```
print("="*100)
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr)))
```

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

Train confusion matrix
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.843
[[ 3713  3713]
 [16143 25472]]
```

In [132]:

```
conf_matr_df_train_5 = pd.DataFrame(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds,
train_fpr, train_fpr)), range(2),range(2))
```

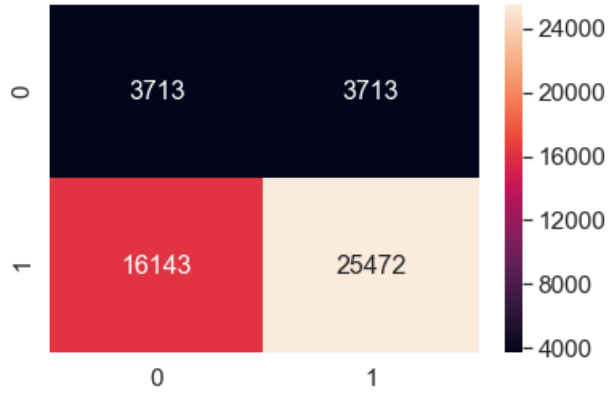
```
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.843
```

In [133]:

```
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_train_5, annot=True,annot_kws={"size": 16}, fmt='g')
```

Out[133]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x589d0518>
```



Test data

In [134]:

```
print("="*100)
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr)))
```

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

Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24999999161092998 for threshold 0.852
[[ 3325  2134]
 [15322 15271]]
```

In [135]:

```
conf_matr_df_test_5 = pd.DataFrame(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, tes
t_fpr, test_fpr)), range(2),range(2))
```

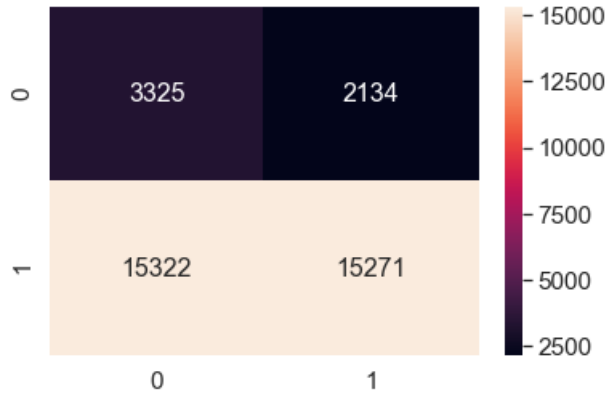
```
the maximum value of tpr*(1-fpr) 0.24999999161092998 for threshold 0.852
```

In [136]:

```
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_test_5, annot=True,annot_kws={"size": 16}, fmt='g')
```

Out[136]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x4aad8cc0>
```



3) Conclusion

In [137]:

```
# Please compare all your models using Prettytable library
# http://zetcode.com/python/prettytable/

from prettytable import PrettyTable

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

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

x.add_row(["BOW", "Logistic Regression", 0.005, 0.674])
x.add_row(["TFIDF", "Logistic Regression", 0.1, 0.659])
x.add_row(["AVG W2V", "Logistic Regression", 1.0, 0.703])
x.add_row(["TFIDF W2V", "Logistic Regression", 0.01, 0.684])
x.add_row(["WITHOUT TEXT", "Logistic Regression", 0.01, 0.574])


print(x)
```

```
+--------------+---------------------+-----------------------+-------+
|  Vectorizer  |        Model        | Alpha:Hyper Parameter |  AUC  |
+--------------+---------------------+-----------------------+-------+
|     BOW      | Logistic Regression |         0.005         | 0.674 |
|    TFIDF     | Logistic Regression |          0.1          | 0.659 |
|   AVG W2V    | Logistic Regression |          1.0          | 0.703 |
|  TFIDF W2V   | Logistic Regression |          0.01         | 0.684 |
| WITHOUT TEXT | Logistic Regression |          0.01         | 0.574 |
+--------------+---------------------+-----------------------+-------+
```

In [ ]:

```

```

In [ ]:

```

```