DonorsChoose

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

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

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

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

About the DonorsChoose Data Set

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

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

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

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

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

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

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

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

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

Notes on the Essay Data

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

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

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

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

your neighborhood, and your school are all helpful."

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

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

In [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_new_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]:

```
catogories = list(project_data['project_subject_categories'].values)
# remove special characters from list of strings python:
https://stackoverflow.com/a/47301924/4084039
# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-string
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-python
cat_list = []
for i in catogories:
    temp = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
    for j in i.split(','): # it will split it in three parts ["Math & Science", "Warmth", "Care & H
unger"]
        if 'The' in j.split(): # this will split each of the catogory based on space "Math & Scienc
e"=> "Math","&", "Science"
            j=j.replace('The','') # if we have the words "The" we are going to replace it with ''(i
.e removing 'The')
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"Math &
Science"=>"Math&Science"
        temp+=j.strip()+" " #" abc ".strip() will return "abc", remove the trailing spaces
        temp = temp.replace('&','_') # we are replacing the & value into
    cat_list.append(temp.strip())

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

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

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

1.3 preprocessing of `project_subject_subcategories`

In [7]:

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

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

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

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

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

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

preprocessing school state

In [8]:

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

preprocessing teacher prefix

In [9]:

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

preprocessing project grade category

In [10]:

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

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

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

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

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

1.3 Text preprocessing

In [11]:

```
# 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 [12]:

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

Out[12]:

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

Decontracting function for sentence

In [13]:

```
# 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 [14]:

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

In [15]:

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

```
100%|██████████| 109248/109248 [01:33<00:00, 1172.26it/s]
```

In [16]:

```
# after preprocesing
preprocessed_essays[2000]
```

Out[16]:

```
'bilingual first grade students full joy eager learn classroom place daily growth constant
challenge discovery students spend year learning foundations reading writing math order succeed li
ves quickly becoming independent learners taking information learned apply multiple activities all
ow use imagination high level thinking skills teacher low income high poverty school district stud
ents faced several challenges classroom personal folders used every day reading writing math
classes provide students personal space using folders help students focus work not neighbor studen
ts able use dividers whole group independent small group time instruction generous donation projec
t improve students self confidence independence donating project not help improve increase student
attention focus ultimately help increase academic achievementnannan'
```

In [17]:

```
project_data["clean_essays"] = preprocessed_essays

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

1.4 Preprocessing of `project_title`

In [18]:

```
preprocessed_pt = []
for titles in tqdm(project_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)
    preprocessed_pt.append(title.lower().strip())
```

```
100%|██████████| 109248/109248 [00:04<00:00, 25792.39it/s]
```

In [19]:

```
project_data["clean_pt"] = preprocessed_pt
project_data.drop(['project_title'], axis=1, inplace=True)
```

number of words in title

In [20]:

```
title_word_count = []
for i in project_data["clean_pt"] :
    j = len(i.split())
    title_word_count.append(j)
project_data["title_word_count"] = title_word_count
project_data.head(5)
```

Out[20]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime	pro
0	0	p036502	484aaf11257089a66cfedc9461c6bd0a	Ms.	NV	18-11-2016 14:45	Mo kin stu fro
1	3	p185307	525fdbb6ec7f538a48beebaa0a51b24f	Mr.	NC	12-08-2016 15:42	My the stu ...
2	4	p013780	a63b5547a7239eae4c1872670848e61a	Mr.	CA	06-08-2016 09:09	My ath stu ...
3	5	p063374	403c6783e9286e51ab318fba40f8d729	Mrs.	DE	05-11-2016 10:01	My eac the ma
4	6	p103285	4e156c5fb3eea2531601c8736f3751a7	Mrs.	MO	31-08-2016 00:30	Kin the gra stu

number of words in essay

In [21]:

```
essay_word_count = []
for i in project_data["clean_essays"] :
    j = len(i.split())
    essay_word_count.append(j)
project_data["essay_word_count"] = essay_word_count
project_data.head(5)
```

Out[21]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime	pro
0	0	p036502	484aaf11257089a66cfedc9461c6bd0a	Ms.	NV	18-11-2016 14:45	Mo kin stu fro
1	3	p185307	525fdbb6ec7f538a48beebaa0a51b24f	Mr.	NC	12-08-2016 15:42	My the stu ...
2	4	p013780	a63b5547a7239eae4c1872670848e61a	Mr.	CA	06-08-2016 09:09	My ath stu ...
3	5	p063374	403c6783e9286e51ab318fba40f8d729	Mrs.	DE	05-11-2016 10:01	My eac the ma
4	6	p103285	4e156c5fb3eea2531601c8736f3751a7	Mrs.	MO	31-08-2016 00:30	Kir the gra stu

Calculate Sentiment Scores for the essays

In [22]:

```
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer

analyser = SentimentIntensityAnalyzer()
```

In [23]:

```
neg = []
pos = []
neu = []
compound = []
for i in tqdm(project_data["clean_essays"]) :
    j = analyser.polarity_scores(i)['neg']
    k = analyser.polarity_scores(i)['pos']
    l = analyser.polarity_scores(i)['neu']
    m = analyser.polarity_scores(i)['compound']
    neg.append(j)
    pos.append(k)
    neu.append(l)
    neu.append(l)
    compound.append(m)
```

```
100%|██████████| 109248/109248 [19:59<00:00, 91.06it/s]
```

In [24]:

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

In [25]:

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

Out[25]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime	proj
0	0	p036502	484aaf11257089a66cfedc9461c6bd0a	Ms.	NV	18-11-2016 14:45	Mos kind stud from
1	3	p185307	525fdbb6ec7f538a48beebaa0a51b24f	Mr.	NC	12-08-2016 15:42	My s the g stud ...

2 rows × 24 columns

Splitting data as train ,test and CV

In [26]:

```
from sklearn.model_selection import train_test_split
S_train, S_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'
])
S_train, S_cv, y_train, y_cv = train_test_split(S_train, y_train, test_size=0.30, stratify=y_train)
```

In [27]:

```
S_train.drop(['project_is_approved'], axis=1, inplace=True)
S_test.drop(['project_is_approved'], axis=1, inplace=True)
S_cv.drop(['project_is_approved'], axis=1, inplace=True)
```

1.5 Preparing data for models

In [28]:

```
project_data.columns
```

Out[28]:

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

we are going to consider

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

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

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

1.5.1 Vectorizing Categorical data

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

VECTORIZING CLEAN CATEGORIES USING ONE HOT ENCODING

In [29]:

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

```
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'SpecialNeeds',
'Health_Sports', 'Math_Science', 'Literacy_Language']
Shape of matrix of Train data after one hot encoding  (51237, 9)
Shape of matrix of Test data after one hot encoding  (36052, 9)
Shape of matrix of CV data after one hot encoding  (21959, 9)
```

VECTORIZING CLEAN SUBCATEGORIES USING ONE HOT ENCODING

In [30]:

```python
vectorizer_clean_subcat = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()), lowercase=False, binary=
True)
vectorizer_clean_subcat.fit(S_train['clean_subcategories'].values)
sub_categories_one_hot_train = vectorizer_clean_subcat.transform(S_train['clean_subcategories'].values)
sub_categories_one_hot_test =
vectorizer_clean_subcat.transform(S_test['clean_subcategories'].values)
sub_categories_one_hot_cv = vectorizer_clean_subcat.transform(S_cv['clean_subcategories'].values)
print(vectorizer_clean_subcat.get_feature_names())
print("Shape of matrix of Train data after one hot encoding ",sub_categories_one_hot_train.shape)
print("Shape of matrix of Test data after one hot encoding ",sub_categories_one_hot_test.shape)
print("Shape of matrix of Cross Validation data after one hot encoding ",sub_categories_one_hot_cv.shape)
```

```
['Economics', 'CommunityService', 'FinancialLiteracy', 'ParentInvolvement', 'Extracurricular',
['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  (51237, 30)
Shape of matrix of Test data after one hot encoding  (36052, 30)
Shape of matrix of Cross Validation data after one hot encoding  (21959, 30)
```

VECTORIZING SCHOOL STATE USING ONE HOT ENCODING

In [31]:

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

```
['VT', 'WY', 'ND', 'MT', 'RI', 'SD', 'NE', 'DE', 'AK', 'NH', 'WV', 'ME', 'HI', 'DC', 'NM', 'KS', 'I
A', 'ID', 'AR', 'CO', 'MN', 'OR', 'KY', 'MS', 'NV', 'MD', 'CT', 'TN', 'UT', 'AL', 'WI', 'VA', 'AZ',
'NJ', 'OK', 'WA', 'MA', 'LA', 'OH', 'MO', 'IN', 'PA', 'MI', 'SC', 'GA', 'IL', 'NC', 'FL', 'NY', 'TX
', 'CA']
Shape of matrix of Train data after one hot encoding  (51237, 51)
Shape of matrix of Test data after one hot encoding  (36052, 51)
Shape of matrix of Cross Validation data after one hot encoding  (21959, 51)
```

VECTORIZING TEACHER PREFIX USING ONE HOT ENCODING

In [32]:

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

```
['Dr.', 'Teacher', 'Mr.', 'Ms.', 'Mrs.']
Shape of matrix of Train data after one hot encoding  (51237, 5)
Shape of matrix of Test data after one hot encoding  (36052, 5)
Shape of matrix of Cross Validation data after one hot encoding  (21959, 5)
```

VECTORIZING PROJECT GRADE CATEGORY USING ONE HOT ENCODING

In [33]:

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

```
['Grades9-12', 'Grades6-8', 'Grades3-5', 'GradesPreK-2']
Shape of matrix of Train data after one hot encoding  (51237, 4)
Shape of matrix of Test data after one hot encoding  (36052, 4)
Shape of matrix of Cross Validation data after one hot encoding  (21959, 4)
```

1.5.2 Vectorizing Text data

1.5.2.1 Bag of words

In [34]:

```
# We are considering only the words which appeared in at least 10 documents(rows or projects).
vectorizer_bow = CountVectorizer(min_df=10)
text_bow = vectorizer_bow.fit_transform(S_train["clean_essays"])
print("Shape of matrix after one hot encoding ",text_bow.shape)
```

```
Shape of matrix after one hot encoding  (51237, 12268)
```

In [35]:

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

```
Shape of matrix after one hot encoding  (36052, 12268)
```

In [36]:

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

```
Shape of matrix after one hot encoding  (21959, 12268)
```

In [37]:

```
vectorizer_title_bow = CountVectorizer( min_df=10)
title_bow_train= vectorizer_title_bow.fit_transform(S_train["clean_pt"])
print("Shape of matrix after one hot encoding ",title_bow_train.shape)
```

```
Shape of matrix after one hot encoding  (51237, 2127)
```

In [38]:

```
title_bow_test = vectorizer_title_bow.transform(S_test["clean_pt"])
print("Shape of matrix after one hot encoding ",title_bow_test.shape)
```

```
Shape of matrix after one hot encoding  (36052, 2127)
```

In [39]:

```
title_bow_cv = vectorizer_title_bow.transform(S_cv["clean_pt"])
print("Shape of matrix after one hot encoding ",title_bow_cv.shape)
```

```
Shape of matrix after one hot encoding  (21959, 2127)
```

1.5.2.2 TFIDF vectorizer

In [40]:

In [40]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer_tfidf_essay = TfidfVectorizer( min_df=10)
vectorizer_tfidf_essay.fit(S_train["clean_essays"])
text_tfidf_train = vectorizer_tfidf_essay.transform(S_train["clean_essays"])
print("Shape of matrix after one hot encoding ",text_tfidf_train.shape)
```

```
Shape of matrix after one hot encoding  (51237, 12268)
```

In [41]:

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

```
Shape of matrix after one hot encoding  (36052, 12268)
```

In [42]:

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

```
Shape of matrix after one hot encoding  (21959, 12268)
```

In [43]:

```
vectorizer_tfidf_title = TfidfVectorizer( min_df=10)
vectorizer_tfidf_title.fit(S_train["clean_pt"])
title_tfidf_train = vectorizer_tfidf_title.transform(S_train["clean_pt"])
print("Shape of matrix after one hot encoding ",title_tfidf_train.shape)
```

```
Shape of matrix after one hot encoding  (51237, 2127)
```

In [44]:

```
title_tfidf_test = vectorizer_tfidf_title.transform(S_test["clean_pt"])
print("Shape of matrix after one hot encoding ",title_tfidf_test.shape)
```

```
Shape of matrix after one hot encoding  (36052, 2127)
```

In [45]:

```
title_tfidf_cv = vectorizer_tfidf_title.transform(S_cv["clean_pt"])
print("Shape of matrix after one hot encoding ",title_tfidf_cv.shape)
```

```
Shape of matrix after one hot encoding  (21959, 2127)
```

1.5.2.3 Using Pretrained Models: Avg W2V

In [46]:

```
# 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')



words = []
words = []
for i in preprocessed_essays:
    words.extend(i.split(' '))

for i in preprocessed_pt:
    words.extend(i.split(' '))
print("all the words in the coupus", len(words))
words = set(words)
print("the unique words in the coupus", len(words))

inter_words = set(model.keys()).intersection(words)
print("The number of words that are present in both glove vectors and our coupus", \
      len(inter_words),"(",np.round(len(inter_words)/len(words)*100,3),"%)")

words_courpus = {}
words_glove = set(model.keys())
for i in words:
    if i in words_glove:
        words_courpus[i] = model[i]
print("word 2 vec length", len(words_courpus))


# stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-pickle-to-sa
ve-and-load-variables-in-python/

import pickle
with open('glove_vectors', 'wb') as f:
    pickle.dump(words_courpus, f)
```

```
Loading Glove Model
```

```
279727it [01:13, 3785.86it/s]
```

```
Done. 279727  words loaded!
all the words in the coupus 15565024
the unique words in the coupus 58960
The number of words that are present in both glove vectors and our coupus 44760 ( 75.916 %)
word 2 vec length 44760
```

In [47]:

```
# stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-pickle-to-sa
ve-and-load-variables-in-python/
# make sure you have the glove_vectors file
with open('glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())
```

In [48]:

```
# 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(S_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%|██████████| 51237/51237 [00:39<00:00, 1283.52it/s]
```

```
51237
300
```

In [49]:

```
avg_w2v_vectors_test = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(S_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, 2252.52it/s]
```

```
36052
300
```

In [50]:

```
avg_w2v_vectors_cv = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(S_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%|██████████| 21959/21959 [00:09<00:00, 2205.08it/s]
```

```
21959
300
```

In [51]:

```
avg_w2v_title_train = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(S_train["clean_pt"]): # 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_title_train.append(vector)

print(len(avg_w2v_title_train))
print(len(avg_w2v_title_train[0]))
```

```
100%|██████████| 51237/51237 [00:01<00:00, 40136.13it/s]
```

```
51237
300
```

In [52]:

```
avg_w2v_title_test = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(S_test["clean_pt"]): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    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_title_test.append(vector)

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

```
100%|██████████| 36052/36052 [00:01<00:00, 33785.09it/s]
```

```
36052
300
```

In [53]:

```
avg_w2v_title_cv = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(S_cv["clean_pt"]): # 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_title_cv.append(vector)

print(len(avg_w2v_title_cv))
print(len(avg_w2v_title_cv[0]))
```

```
100%|██████████| 21959/21959 [00:00<00:00, 25602.11it/s]
```

```
21959
300
```

1.5.2.3 Using Pretrained Models: TFIDF weighted W2V

In [54]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(S_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 [55]:

```
tfidf_w2v_vectors_train = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(S_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]))
print(len(tfidf_w2v_vectors_train[0]))
```

```
100%|██████████| 51237/51237 [02:29<00:00, 343.78it/s]
```

```
51237
300
```

In [56]:

```
tfidf_w2v_vectors_test= []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(S_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 [01:43<00:00, 348.42it/s]
```

```
36052
300
```

In [57]:

```
tfidf_w2v_vectors_cv= []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(S_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%|██████████| 21959/21959 [01:03<00:00, 344.01it/s]
```

```
21959
300
```

In [58]:

```
# Similarly you can vectorize for title also
# average Word2Vec
# compute average word2vec for each review.
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(S_train["clean_pt"])
# 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())
tfidf_w2v_ppt_train= []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(S_train["clean_pt"]): # 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_ppt_train.append(vector)

print(len(tfidf_w2v_ppt_train))
print(len(tfidf_w2v_ppt_train[0]))
```

```
100%|██████████| 51237/51237 [00:03<00:00, 16247.35it/s]
```

```
51237
300
```

In [59]:

```
# Similarly you can vectorize for title also
# average Word2Vec
# compute average word2vec for each review.
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_w2v_ppt_test= []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(S_test["clean_pt"]): # 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_ppt_test.append(vector)

print(len(tfidf_w2v_ppt_test))
print(len(tfidf_w2v_ppt_test[0]))
```

```
100%|██████████| 36052/36052 [00:02<00:00, 14153.34it/s]
```

```
36052
300
```

In [60]:

```
tfidf_w2v_ppt_cv= []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(S_cv["clean_pt"]): # 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_ppt_cv.append(vector)

print(len(tfidf_w2v_ppt_cv))
print(len(tfidf_w2v_ppt_cv[0]))
```

```
100%|██████████| 21959/21959 [00:01<00:00, 11985.71it/s]
```

```
21959
300
```

1.5.3 Vectorizing Numerical features

In [61]:

```
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 [62]:

```
S_train = pd.merge(S_train, price_data, on='id', how='left')
S_test = pd.merge(S_test, price_data, on='id', how='left')
S_cv = pd.merge(S_cv, price_data, on='id', how='left')
```

Normalizing Price

In [63]:

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

price_scalar = Normalizer()
price_scalar.fit(S_train['price'].values.reshape(-1,1)) # finding the mean and standard deviation
of this data
price_standardized_train = price_scalar.transform(S_train['price'].values.reshape(-1, 1))
price_standardized_test = price_scalar.transform(S_test['price'].values.reshape(-1, 1))
price_standardized_cv = price_scalar.transform(S_cv['price'].values.reshape(-1, 1))
```

In [64]:

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

```
(51237, 1)
(36052, 1)
(21959, 1)
```

Normalizing number of previously posted projects

In [65]:

```
price_scalar.fit(S_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))
prev_project_standardized_train =
price_scalar.transform(S_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,
1))
prev_project_standardized_test =
price_scalar.transform(S_test['teacher_number_of_previously_posted_projects'].values.reshape(-1, 1)
)
prev_project_standardized_cv =
price_scalar.transform(S_cv['teacher_number_of_previously_posted_projects'].values.reshape(-1, 1))
```

In [66]:

```
print(prev_project_standardized_train.shape)
print(prev_project_standardized_train.shape)
print(prev_project_standardized_test.shape)
print(prev_project_standardized_cv.shape)
```

```
(51237, 1)
(36052, 1)
(21959, 1)
```

Normalizing Quantity

In [67]:

```
price_scalar.fit(S_train['quantity'].values.reshape(-1,1)) # finding the mean and standard
deviation of this data
quantity_standardized_train = price_scalar.transform(S_train['quantity'].values.reshape(-1, 1))
quantity_standardized_test = price_scalar.transform(S_test['quantity'].values.reshape(-1, 1))
quantity_standardized_cv = price_scalar.transform(S_cv['quantity'].values.reshape(-1, 1))
```

In [68]:

```
print(quantity_standardized_train.shape)
print(quantity_standardized_test.shape)
print(quantity_standardized_cv.shape)
```

```
(51237, 1)
(36052, 1)
(21959, 1)
```

normalizing title word count

In [69]:

```
normalizer = Normalizer()
normalizer.fit(S_train['title_word_count'].values.reshape(-1,1))
title_word_count_train = normalizer.transform(S_train['title_word_count'].values.reshape(-1,1))
title_word_count_cv = normalizer.transform(S_cv['title_word_count'].values.reshape(-1,1))
title_word_count_test = normalizer.transform(S_test['title_word_count'].values.reshape(-1,1))
print("After vectorizations")
print(title_word_count_train.shape, y_train.shape)
print(title_word_count_cv.shape, y_cv.shape)
print(title_word_count_test.shape, y_test.shape)
```

```
After vectorizations
(51237, 1) (51237,)
(21959, 1) (21959,)
(36052, 1) (36052,)
```

NORMALIZING ESSAY WORD COUNT

In [70]:

```
normalizer = Normalizer()
normalizer.fit(S_train['essay_word_count'].values.reshape(-1,1))
essay_word_count_train = normalizer.transform(S_train['essay_word_count'].values.reshape(-1,1))
essay_word_count_cv = normalizer.transform(S_cv['essay_word_count'].values.reshape(-1,1))
essay_word_count_test = normalizer.transform(S_test['essay_word_count'].values.reshape(-1,1))
print("After vectorizations")
print(essay_word_count_train.shape, y_train.shape)
print(essay_word_count_cv.shape, y_cv.shape)
print(essay_word_count_test.shape, y_test.shape)
```

```
After vectorizations
(51237, 1) (51237,)
(21959, 1) (21959,)
(36052, 1) (36052,)
```

In [71]:

```
normalizer = Normalizer()
normalizer.fit(S_train['essay_word_count'].values.reshape(-1,1))
essay_word_count_train = normalizer.transform(S_train['essay_word_count'].values.reshape(-1,1))
essay_word_count_cv = normalizer.transform(S_cv['essay_word_count'].values.reshape(-1,1))
essay_word_count_test = normalizer.transform(S_test['essay_word_count'].values.reshape(-1,1))
print("After vectorizations")
print(essay_word_count_train.shape, y_train.shape)
print(essay_word_count_cv.shape, y_cv.shape)
print(essay_word_count_test.shape, y_test.shape)
```

```
After vectorizations
(51237, 1) (51237,)
(21959, 1) (21959,)
(36052, 1) (36052,)
```

NORMALIZING ESSAY SENTIMENT-POS

In [72]:

```
normalizer = Normalizer()
normalizer.fit(S_train['pos'].values.reshape(-1,1))
essay_sent_pos_train = normalizer.transform(S_train['pos'].values.reshape(-1,1))
essay_sent_pos_cv = normalizer.transform(S_cv['pos'].values.reshape(-1,1))
essay_sent_pos_test = normalizer.transform(S_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)
```

```
After vectorizations
(51237, 1) (51237,)
(21959, 1) (21959,)
(36052, 1) (36052,)
```

NORMALIZING ESSAY SENTIMEN-NEG

In [73]:

```
normalizer = Normalizer()
normalizer.fit(S_train['neg'].values.reshape(-1,1))
essay_sent_neg_train = normalizer.transform(S_train['neg'].values.reshape(-1,1))
essay_sent_neg_cv = normalizer.transform(S_cv['neg'].values.reshape(-1,1))
essay_sent_neg_test = normalizer.transform(S_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)
```

```
After vectorizations
(51237, 1) (51237,)
(21959, 1) (21959,)
(36052, 1) (36052,)
```

NORMALIZING ESSAY SENTIMEN-NEU

In [74]:

```
normalizer = Normalizer()
normalizer.fit(S_train['neu'].values.reshape(-1,1))
essay_sent_neu_train = normalizer.transform(S_train['neu'].values.reshape(-1,1))
essay_sent_neu_cv = normalizer.transform(S_cv['neu'].values.reshape(-1,1))
essay_sent_neu_test = normalizer.transform(S_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)
```

```
After vectorizations
(51237, 1) (51237,)
(21959, 1) (21959,)
```

```
(36052, 1) (36052,)
```

NORMALIZING ESSAY SENTIMEN-COMPOUND

In [75]:

```
normalizer = Normalizer()
normalizer.fit(S_train['compound'].values.reshape(-1,1))
essay_sent_comp_train = normalizer.transform(S_train['compound'].values.reshape(-1,1))
essay_sent_comp_cv = normalizer.transform(S_cv['compound'].values.reshape(-1,1))
essay_sent_comp_test = normalizer.transform(S_test['compound'].values.reshape(-1,1))
print("After vectorizations")
print(essay_sent_comp_train.shape, y_train.shape)
print(essay_sent_comp_cv.shape, y_cv.shape)
print(essay_sent_comp_test.shape, y_test.shape)
```

```
After vectorizations
(51237, 1) (51237,)
(21959, 1) (21959,)
(36052, 1) (36052,)
```

1.5.4 Merging all the above features

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

Assignment 7: SVM

[Task-1] Apply Support Vector Machines(SGDClassifier with hinge loss: Linear SVM) on these feature sets Set 1: categorical, numerical features + project_title(BOW) + preprocessed_eassay (BOW) Set 2: categorical, numerical features + project_title(TFIDF)+ preprocessed_eassay (TFIDF) Set 3: categorical, numerical features + project_title(AVG W2V)+ preprocessed_eassay (AVG W2V) Set 4: categorical, numerical features + project_title(TFIDF W2V)+ preprocessed_eassay (TFIDF W2V)

The hyper paramter tuning (best alpha in range [10^-4 to 10^4], and the best penalty among 'l1', 'l2') Find the best hyper parameter which will give the maximum AUC 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

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 with predicted and original labels of test data points. Please visualize your confusion matrices using seaborn heatmaps.

[Task-2] Apply the Support Vector Machines on these features by finding the best hyper paramter as suggested in step 2 and step 3 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 Apply TruncatedSVD on TfidfVectorizer of essay text, choose the number of components (n_components) using elbow method : numerical data

Conclusion You need to summarize the results at the end of the notebook, summarize it in the table format. To print out a table please refer to this prettytable library link

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.

2. Support Vector Machines

2.4 Appling 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 instrucations

Feature set 1 using BOW

In [75]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
S_BOW_train=
hstack((categories_one_hot_train,sub_categories_one_hot_train,school_state_one_hot_train,teacher_pr
efix_one_hot_train,clean_project_grade_category_one_hot_train,text_bow,title_bow_train,price_standa
rdized_train,prev_project_standardized_train,quantity_standardized_train,title_word_count_train,es
say_word_count_train,essay_sent_pos_train,essay_sent_neg_train,essay_sent_neu_train,essay_sent_comp
_train)).tocsr()
print(S_BOW_train.shape)
```

```
(51237, 14497)
```

In [76]:

```
S_BOW_test= hstack((categories_one_hot_test,sub_categories_one_hot_test,school_state_one_hot_test,
teacher_prefix_one_hot_test,clean_project_grade_category_one_hot_test,text_bow_test,title_bow_test
,price_standardized_test,prev_project_standardized_test,quantity_standardized_test,title_word_count
_test,essay_word_count_test,essay_sent_pos_test,essay_sent_neg_test,essay_sent_neu_test,essay_sent_
comp_test)).tocsr()
print(S_BOW_test.shape)
```

```
(36052, 14497)
```

In [81]:

```
S_BOW_cv=
hstack((categories_one_hot_cv,sub_categories_one_hot_cv,school_state_one_hot_cv,teacher_prefix_one_
hot_cv,clean_project_grade_category_one_hot_cv,text_bow_cv,title_bow_cv,price_standardized_cv,prev_
project_standardized_cv,quantity_standardized_cv,title_word_count_cv,essay_word_count_cv,essay_sent
_pos_cv,essay_sent_neg_cv,essay_sent_neu_cv,essay_sent_comp_cv)).tocsr()
print(S_BOW_cv.shape)
```

```
(21959, 14497)
```

In [76]:

```
def batch_predict(clf, data):
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive class
    # not the predicted outputs
    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000
    # consider you X_tr shape is 49041, then your tr_loop will be 49041 - 49041%1000 = 49000
    # in this for loop we will iterate unti the last 1000 multiplier
    for i in range(0, tr_loop, 1000):
        y_data_pred.extend(clf.predict_proba(data[i:i+1000])[:,1])
    # we will be predicting for the last data points
    if data.shape[0]%1000 !=0:
        y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

    return y_data_pred
```

finding best hyperparameter using CV

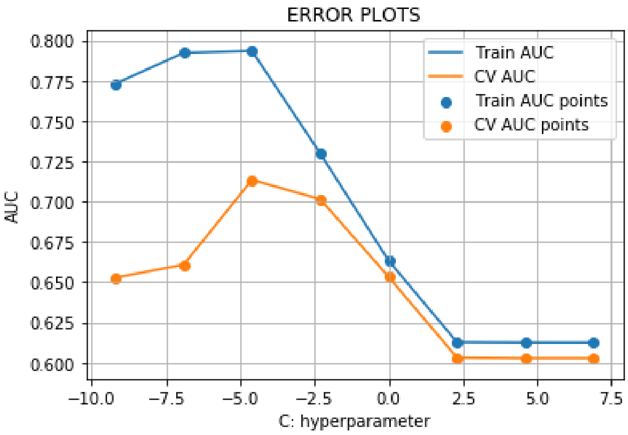
In [84]:

```
from sklearn.linear_model import SGDClassifier
from sklearn.calibration import CalibratedClassifierCV
import matplotlib.pyplot as plt
from sklearn.metrics import roc_auc_score
train_auc = []
cv_auc = []
a = []
b = []
import math
alpha=[10**x for x in range(-4,4)]

for i in tqdm(alpha):
    svm= SGDClassifier(alpha=i,loss='hinge', class_weight='balanced' )
    s=svm.fit(S_BOW_train, y_train)
    clfcalibrated=CalibratedClassifierCV(svm,cv='prefit',method='isotonic')
    clfcalibrated.fit(S_BOW_cv,y_cv)
    y_train_pred = batch_predict(clfcalibrated,S_BOW_train)
    y_cv_pred = batch_predict(clfcalibrated, S_BOW_cv)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
    a.append(y_train_pred)
    b.append(y_cv_pred)

plt.plot([math.log(i) for i in alpha],train_auc, label='Train AUC')
plt.plot([math.log(i) for i in alpha],cv_auc, label='CV AUC')
plt.scatter([math.log(i) for i in alpha],train_auc, label='Train AUC points')
plt.scatter([math.log(i) for i in alpha],cv_auc, label='CV AUC points')
plt.legend()
plt.xlabel("C: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```

```
100%|██████████| 8/8 [00:06<00:00,  1.14it/s]
```



using Gridsearch CV for finding best hyperparameter

In [123]:

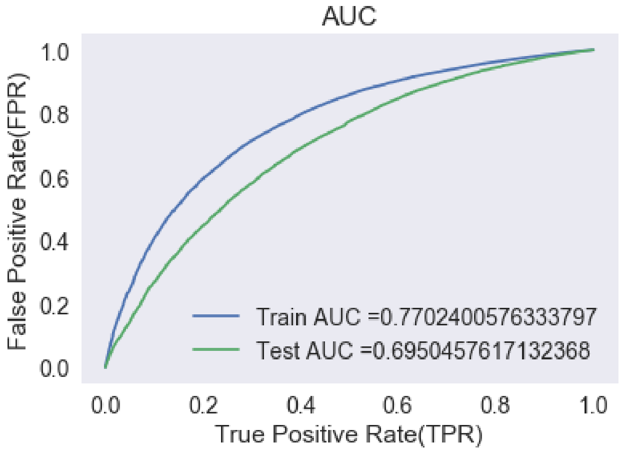
```
from sklearn.linear_model import SGDClassifier
from sklearn.model_selection import GridSearchCV
svm = SGDClassifier(loss='hinge',class_weight='balanced')
alpha_vals=[10**x for x in range(-4,4)]
penalty = ['l1', 'l2']
parameters = {'alpha':alpha_vals,'penalty':penalty}
clf = GridSearchCV(svm, parameters, cv= 10, scoring='roc_auc')
best_model=clf.fit(S_BOW_train, y_train)
print('Best alpha:', best_model.best_estimator_.get_params()['alpha'])
print('Best penalty:', best_model.best_estimator_.get_params()['penalty'])
```

```
Best alpha: 0.01
Best penalty: l2
Best penalty: l2
```

we can take alpha=0.01 and penalty=l2

In [97]:

```
#https://scikitlearn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.ro
rve
from sklearn.metrics import roc_curve, auc
svm= SGDClassifier(alpha=0.01,loss='hinge', penalty='l2', class_weight='balanced', )
s=svm.fit(S_BOW_train[0:26237], y_train[0:26237])
clfcalibrated=CalibratedClassifierCV(svm,method='isotonic')
clfcalibrated.fit(S_BOW_train[26237:51237],y_train[26237:51237])
y_train_pred = batch_predict(clfcalibrated,S_BOW_train)
y_test_pred = batch_predict(clfcalibrated, S_BOW_test)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs
train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)
plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
```



In [77]:

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

confusion matrix for train data

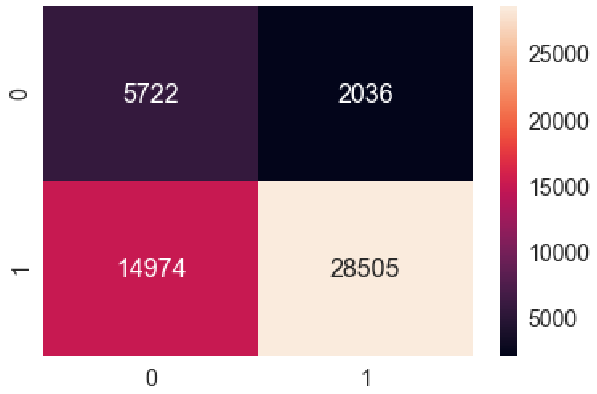
In [94]:

```
conf_matr_df_train = pd.DataFrame(confusion_matrix(y_train, prediction(y_train_pred, tr_thresholds
,
train_fpr, train_tpr)), range(2),range(2))
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_train, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
the maximum value of tpr*(1-fpr) 0.48975698271598056 for threshold 0.857
```

Out[94]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x220fff302e8>
```



Confuision matrix for test data

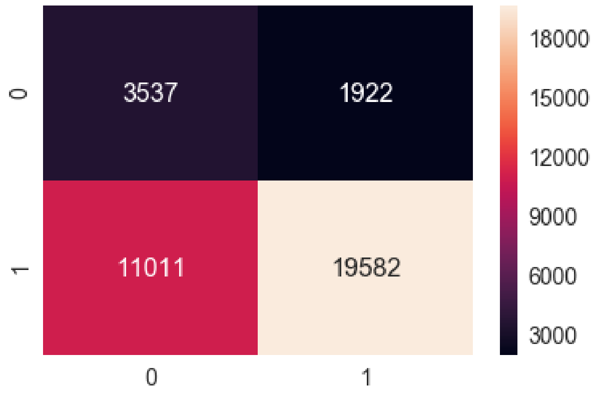
In [96]:

```
conf_matr_df_test = pd.DataFrame(confusion_matrix(y_test, prediction(y_test_pred, tr_thresholds,
train_fpr, train_tpr)), range(2),range(2))
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_test, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
the maximum value of tpr*(1-fpr) 0.48975698271598056 for threshold 0.857
```

Out[96]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x220851842e8>
```



Feature set 2 USING TFIDF_Train

In [98]:

```
# Please write all the code with proper documentation
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
S_TFIDF_train=
hstack((categories_one_hot_train,sub_categories_one_hot_train,school_state_one_hot_train,teacher_pr
efix_one_hot_train,clean_project_grade_category_one_hot_train,text_tfidf_train,title_tfidf_train,p
rice_standardized_train,prev_project_standardized_train,quantity_standardized_train,title_word_cour
t_train,essay_word_count_train,essay_sent_pos_train,essay_sent_neg_train,essay_sent_neu_train,essa
y_sent_comp_train)).tocsr()
S_TFIDF_train.shape
```

Out[98]:

```
(51237, 14497)
```

In [99]:

```
S_TFIDF_test=
hstack((categories_one_hot_test,sub_categories_one_hot_test,school_state_one_hot_test,teacher_prefi
x_one_hot_test,clean_project_grade_category_one_hot_test,text_tfidf_test,title_tfidf_test,price_sta
ndardized_test,prev_project_standardized_test,quantity_standardized_test,title_word_count_test,ess
ay_word_count_test,essay_sent_pos_test,essay_sent_neg_test,essay_sent_neu_test,essay_sent_comp_test
)).tocsr()
S_TFIDF_test.shape
```

Out[99]:

```
(36052, 14497)
```

In [100]:

```
S_TFIDF_cv=
hstack((categories_one_hot_cv,sub_categories_one_hot_cv,school_state_one_hot_cv,teacher_prefix_one_
hot_cv,clean_project_grade_category_one_hot_cv,text_tfidf_cv,title_tfidf_cv,price_standardized_cv,
prev_project_standardized_cv,quantity_standardized_cv,title_word_count_cv,essay_word_count_cv,essay
_sent_pos_cv,essay_sent_neg_cv,essay_sent_neu_cv,essay_sent_comp_cv)).tocsr()
S_TFIDF_cv.shape
```

Out[100]:

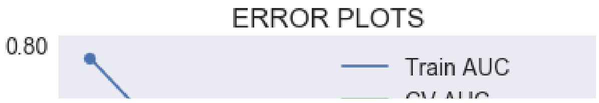
```
(21959, 14497)
```

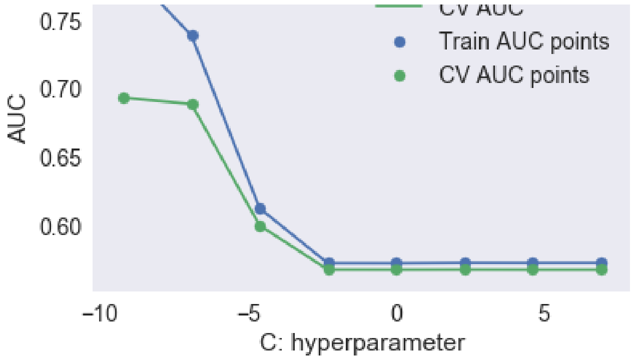
Finding best parameter using CV

In [101]:

```
from sklearn.linear_model import SGDClassifier
from sklearn.calibration import CalibratedClassifierCV
import matplotlib.pyplot as plt
from sklearn.metrics import roc_auc_score
train_auc = []
cv_auc = []
a = []
b = []
import math
alpha=[10**x for x in range(-4,4)]
for i in tqdm(alpha):
    svm= SGDClassifier(alpha=i,loss='hinge', class_weight='balanced' )
    s=svm.fit(S_TFIDF_train, y_train)
    clfcalibrated=CalibratedClassifierCV(svm,cv='prefit',method='isotonic')
    clfcalibrated.fit(S_TFIDF_cv,y_cv)
    y_train_pred = batch_predict(clfcalibrated,S_TFIDF_train)
    y_cv_pred = batch_predict(clfcalibrated, S_TFIDF_cv)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
    a.append(y_train_pred)
    b.append(y_cv_pred)
plt.plot([math.log(i) for i in alpha],train_auc, label='Train AUC')
plt.plot([math.log(i) for i in alpha],cv_auc, label='CV AUC')
plt.scatter([math.log(i) for i in alpha],train_auc, label='Train AUC points')
plt.scatter([math.log(i) for i in alpha],cv_auc, label='CV AUC points')
plt.legend()
plt.xlabel("C: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```

```
100%|██████████| 8/8 [00:04<00:00,  1.87it/s]
```





Finding best hyperparameter using GridSearchCV

In [125]:

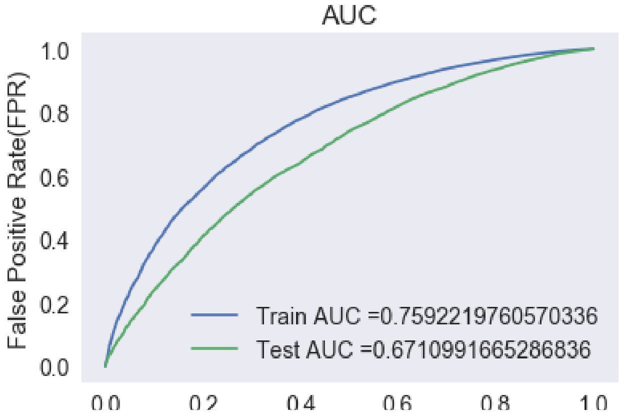
```
from sklearn.linear_model import SGDClassifier
from sklearn.model_selection import GridSearchCV
svm = SGDClassifier(loss='hinge',class_weight='balanced')
alpha_vals=[10**x for x in range(-4,4)]
penalty=['l1','l2']
parameters = {'alpha':alpha_vals,'penalty':penalty}
clf = GridSearchCV(svm, parameters, cv= 10, scoring='roc_auc')
best_model=clf.fit(S_TFIDF_train, y_train)
print('Best alpha:', best_model.best_estimator_.get_params()['alpha'])
print('Best penalty:', best_model.best_estimator_.get_params()['penalty'])
```

```
Best alpha: 0.001
Best penalty: l2
```

so we can take alpha=0.001 and penalty=l2

In [106]:

```
from sklearn.metrics import roc_curve, auc
svm= SGDClassifier(alpha=0.001,loss='hinge', penalty='l2', class_weight='balanced', )
s=svm.fit(S_TFIDF_train[0:26237], y_train[0:26237])
clfcalibrated=CalibratedClassifierCV(svm,method='isotonic')
clfcalibrated.fit(S_TFIDF_train[26237:51237],y_train[26237:51237])
y_train_pred = batch_predict(clfcalibrated,S_TFIDF_train)
y_test_pred = batch_predict(clfcalibrated, S_TFIDF_test)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs
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()
```



confusion matrix for train data

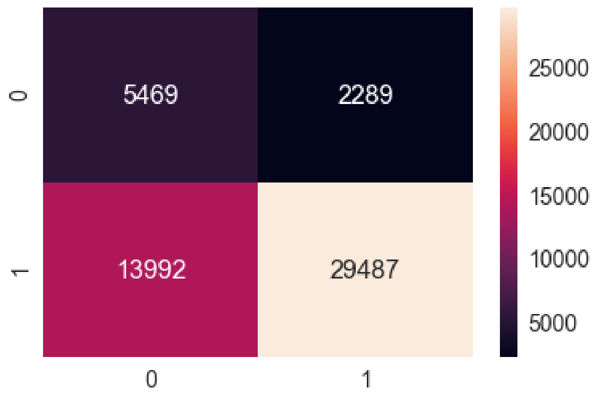
In [107]:

```
conf_matr_df_train = pd.DataFrame(confusion_matrix(y_train, prediction(y_train_pred, tr_thresholds
,
train_fpr, train_tpr)), range(2),range(2))
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_train, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
the maximum value of tpr*(1-fpr) 0.482216241612369 for threshold 0.845
```

Out[107]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x220fff6b128>
```



Confusion matrix for test data

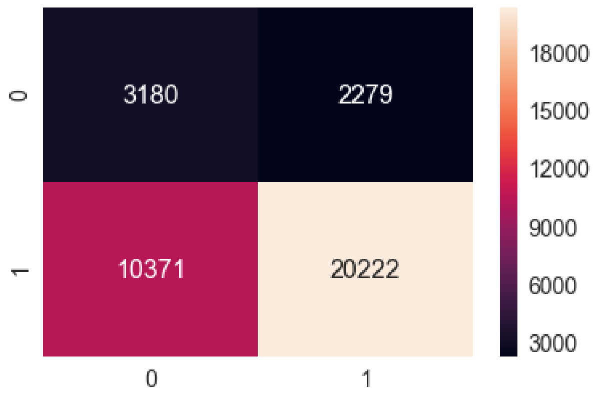
In [108]:

```
conf_matr_df_test = pd.DataFrame(confusion_matrix(y_test, prediction(y_test_pred, tr_thresholds,
train_fpr, train_tpr)), range(2),range(2))
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_test, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
the maximum value of tpr*(1-fpr) 0.482216241612369 for threshold 0.845
```

Out[108]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x220ffe55e80>
```



Feature set 3 USING AVG_W2V

In [109]:

```
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
S_avgw2v_train=
hstack((categories_one_hot_train,sub_categories_one_hot_train,school_state_one_hot_train,teacher_pr
efix_one_hot_train,clean_project_grade_category_one_hot_train,avg_w2v_vectors_train,avg_w2v_title_t
rain,price_standardized_train,prev_project_standardized_train,quantity_standardized_train,title_wor
d_count_train,essay_word_count_train,essay_sent_pos_train,essay_sent_neg_train,essay_sent_neu_train
,essay_sent_comp_train)).tocsr()
print(S_avgw2v_train.shape)

S_avgw2v_test=
hstack((categories_one_hot_test,sub_categories_one_hot_test,school_state_one_hot_test,teacher_prefi
x_one_hot_test,clean_project_grade_category_one_hot_test,avg_w2v_vectors_test,avg_w2v_title_test,p
rice_standardized_test,prev_project_standardized_test,quantity_standardized_test,title_word_count_t
est,essay_word_count_test,essay_sent_pos_test,essay_sent_neg_test,essay_sent_neu_test,essay_sent_co
mp_test)).tocsr()
print(S_avgw2v_test.shape)

S_avgw2v_cv=
hstack((categories_one_hot_cv,sub_categories_one_hot_cv,school_state_one_hot_cv,teacher_prefix_one_
hot_cv,clean_project_grade_category_one_hot_cv,avg_w2v_vectors_cv,avg_w2v_title_cv,price_standardiz
ed_cv,prev_project_standardized_cv,quantity_standardized_cv,title_word_count_cv,essay_word_count_cv
,essay_sent_pos_cv,essay_sent_neg_cv,essay_sent_neu_cv,essay_sent_comp_cv)).tocsr()
print(S_avgw2v_cv.shape)
```

```
(51237, 708)
(36052, 708)
(21959, 708)
```

FINDING BEST HYPERPARAMETER USING CV

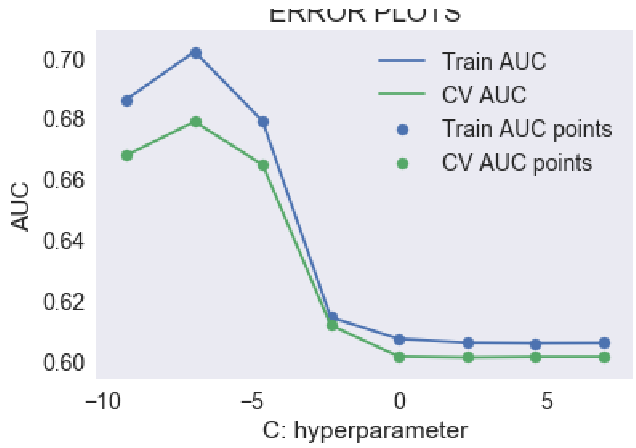
In [110]:

```
from sklearn.linear_model import SGDClassifier
from sklearn.calibration import CalibratedClassifierCV
import matplotlib.pyplot as plt
from sklearn.metrics import roc_auc_score
train_auc = []
cv_auc = []
a = []
b = []
import math
alpha=[10**x for x in range(-4,4)]

for i in tqdm(alpha):
    svm= SGDClassifier(alpha=i,loss='hinge', class_weight='balanced' )
    s=svm.fit(S_avgw2v_train, y_train)
    clfcalibrated=CalibratedClassifierCV(svm,cv='prefit',method='isotonic')
    clfcalibrated.fit(S_avgw2v_cv,y_cv)
    y_train_pred = batch_predict(clfcalibrated,S_avgw2v_train)
    y_cv_pred = batch_predict(clfcalibrated, S_avgw2v_cv)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
    a.append(y_train_pred)
    b.append(y_cv_pred)

plt.plot([math.log(i) for i in alpha],train_auc, label='Train AUC')
plt.plot([math.log(i) for i in alpha],cv_auc, label='CV AUC')
plt.scatter([math.log(i) for i in alpha],train_auc, label='Train AUC points')
plt.scatter([math.log(i) for i in alpha],cv_auc, label='CV AUC points')
plt.legend()
plt.xlabel("C: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```

```
100%|██████████| 8/8 [00:23<00:00,  2.80s/it]
```



FINDING BEST HYPERPARAMETER USING GRIDSEARCHCV

In [126]:

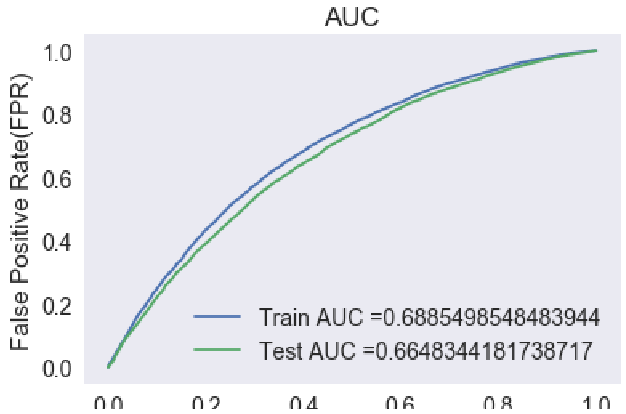
```
from sklearn.linear_model import SGDClassifier
from sklearn.model_selection import GridSearchCV
svm = SGDClassifier(loss='hinge',class_weight='balanced')
penalty=['l1','l2']
alpha_vals=[10**x for x in range(-4,4)]
parameters = {'alpha':alpha_vals,'penalty':penalty}
clf = GridSearchCV(svm, parameters, cv= 10, scoring='roc_auc')
best_model=clf.fit(S_avgw2v_train, y_train)
print('Best penalty:', best_model.best_estimator_.get_params()['penalty'])
print('Best alpha:', best_model.best_estimator_.get_params()['alpha'])
```

```
Best penalty: l2
Best alpha: 0.001
```

lets alpha=0.001 and penalty=l2

In [112]:

```
from sklearn.metrics import roc_curve, auc
svm= SGDClassifier(alpha=0.001,loss='hinge', penalty='l2', class_weight='balanced', )
s=svm.fit(S_avgw2v_train[0:26237], y_train[0:26237])
clfcalibrated=CalibratedClassifierCV(svm,method='isotonic')
clfcalibrated.fit(S_avgw2v_train[26237:51237],y_train[26237:51237])
y_train_pred = batch_predict(clfcalibrated,S_avgw2v_train)
y_test_pred = batch_predict(clfcalibrated,S_avgw2v_test)
train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)
plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
```



CONFUSION MATRIX FOR TRAIN DATA

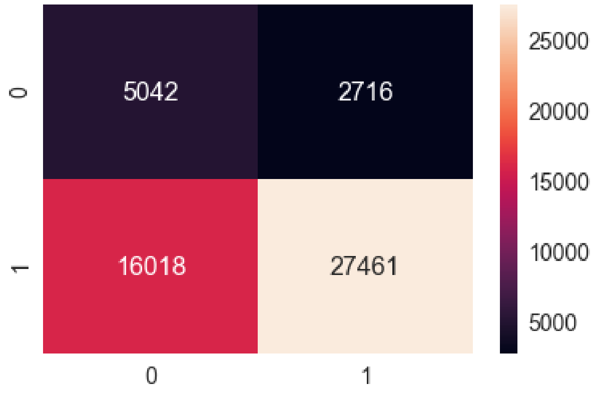
In [113]:

```
conf_matr_df_train = pd.DataFrame(confusion_matrix(y_train, prediction(y_train_pred, tr_thresholds
,
train_fpr, train_tpr)), range(2),range(2))
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_train, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
the maximum value of tpr*(1-fpr) 0.41248060293673644 for threshold 0.854
```

Out[113]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x22085564ef0>
```



CONFUSION MATRIX FOR TEST DATA

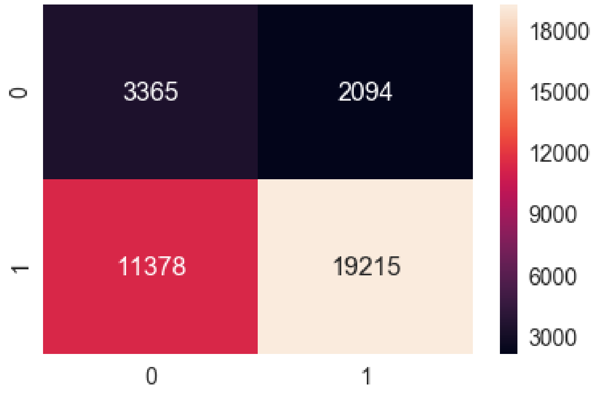
In [114]:

```
conf_matr_df_test = pd.DataFrame(confusion_matrix(y_test, prediction(y_test_pred, tr_thresholds,
train_fpr, train_tpr)), range(2),range(2))
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_test, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
the maximum value of tpr*(1-fpr) 0.41248060293673644 for threshold 0.854
```

Out[114]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x220ffe59c88>
```



FEATURE SET 4:TFIDF_W2V

In [115]:

```
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
S_tfidf_w2v_train= hstack((categories_one_hot_train,sub_categories_one_hot_train,school_state_one_hot_train,teacher_prefix_one_hot_train,clean_project_grade_category_one_hot_train,tfidf_w2v_vectors_train,tfidf_w2v_ppt_train,price_standardized_train,prev_project_standardized_train,quantity_standardized_train,title_word_count_train,essay_word_count_train,essay_sent_pos_train,essay_sent_neg_train,essay_sent_neu_train,essay_sent_comp_train)).tocsr()
print(S_tfidf_w2v_train.shape)

S_tfidf_w2v_test= hstack((categories_one_hot_test,sub_categories_one_hot_test,school_state_one_hot_test,teacher_prefix_one_hot_test,clean_project_grade_category_one_hot_test,tfidf_w2v_vectors_test,tfidf_w2v_ppt_test,price_standardized_test,prev_project_standardized_test,quantity_standardized_test,title_word_count_test,essay_word_count_test,essay_sent_pos_test,essay_sent_neg_test,essay_sent_neu_test,essay_sent_comp_test)).tocsr()
print(S_tfidf_w2v_test.shape)

S_tfidf_w2v_cv= hstack((categories_one_hot_cv,sub_categories_one_hot_cv,school_state_one_hot_cv,teacher_prefix_one_hot_cv,clean_project_grade_category_one_hot_cv,tfidf_w2v_vectors_cv,tfidf_w2v_ppt_cv,price_standardized_cv,prev_project_standardized_cv,quantity_standardized_cv,title_word_count_cv,essay_word_count_cv,essay_sent_pos_cv,essay_sent_neg_cv,essay_sent_neu_cv,essay_sent_comp_cv)).tocsr()
print(S_tfidf_w2v_cv.shape)
```

```
(51237, 708)
(36052, 708)
(21959, 708)
```

Using CV to find best hyperparameter

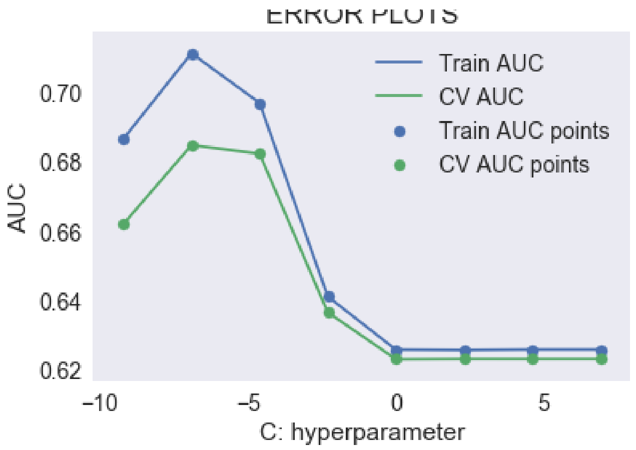
In [116]:

```
from sklearn.linear_model import SGDClassifier
from sklearn.calibration import CalibratedClassifierCV
import matplotlib.pyplot as plt
from sklearn.metrics import roc_auc_score
train_auc = []
cv_auc = []
a = []
b = []
import math
alpha=[10**x for x in range(-4,4)]

for i in tqdm(alpha):
    svm= SGDClassifier(alpha=i,loss='hinge', class_weight='balanced' )
    s=svm.fit(S_tfidf_w2v_train, y_train)
    clfcalibrated=CalibratedClassifierCV(svm,cv='prefit',method='isotonic')
    clfcalibrated.fit(S_tfidf_w2v_cv,y_cv)
    y_train_pred = batch_predict(clfcalibrated,S_tfidf_w2v_train)
    y_cv_pred = batch_predict(clfcalibrated, S_tfidf_w2v_cv)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
# not the predicted outputs
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
    a.append(y_train_pred)
    b.append(y_cv_pred)

plt.plot([math.log(i) for i in alpha],train_auc, label='Train AUC')
plt.plot([math.log(i) for i in alpha],cv_auc, label='CV AUC')
plt.scatter([math.log(i) for i in alpha],train_auc, label='Train AUC points')
plt.scatter([math.log(i) for i in alpha],cv_auc, label='CV AUC points')
plt.legend()
plt.xlabel("C: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```

```
100%|██████████| 8/8 [00:28<00:00,  3.24s/it]
```



Using GridsearchCV to find best hyperparameter

In [128]:

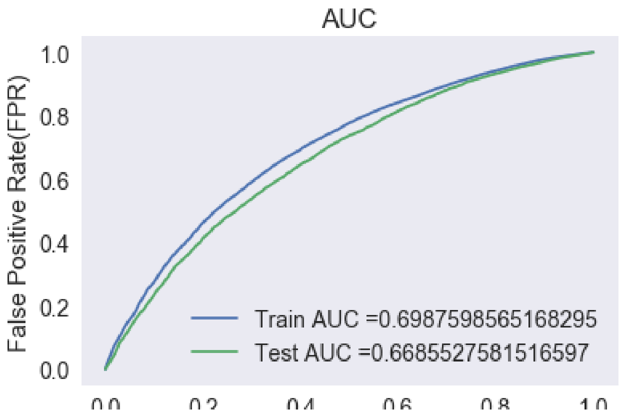
```
from sklearn.linear_model import SGDClassifier
from sklearn.model_selection import GridSearchCV
svm = SGDClassifier(loss='hinge',class_weight='balanced')
alpha_vals=[10**x for x in range(-4,4)]
penalty=['l1','l2']
parameters = {'alpha':alpha_vals,'penalty':penalty}
clf = GridSearchCV(svm, parameters, cv= 10, scoring='roc_auc')
best_model=clf.fit(S_tfidf_w2v_train, y_train)
print('Best alpha:', best_model.best_estimator_.get_params()['alpha'])
print('Best penalty:', best_model.best_estimator_.get_params()['penalty'])
```

```
Best alpha: 0.001
Best penalty: l2
```

lets take c=0.001 and penalty=l2

In [129]:

```
from sklearn.metrics import roc_curve, auc
svm= SGDClassifier(alpha=0.001,loss='hinge', penalty='l2', class_weight='balanced', )
s=svm.fit(S_tfidf_w2v_train[0:26237], y_train[0:26237])
clfcalibrated=CalibratedClassifierCV(svm,method='isotonic')
clfcalibrated.fit(S_tfidf_w2v_train[26237:51237],y_train[26237:51237])
y_train_pred = batch_predict(clfcalibrated,S_tfidf_w2v_train)
y_test_pred = batch_predict(clfcalibrated,S_tfidf_w2v_test)
train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)
plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
```



confusion matrix for train data

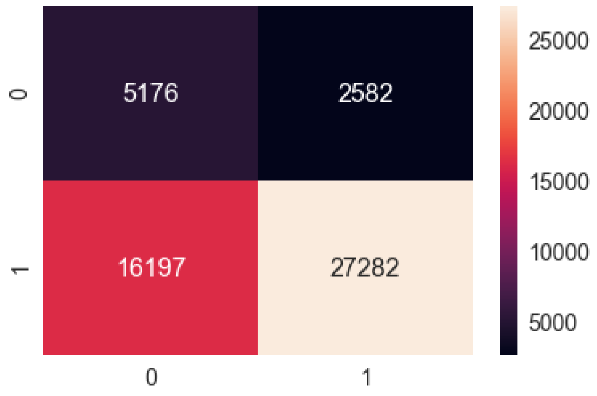
In [130]:

```
conf_matr_df_train = pd.DataFrame(confusion_matrix(y_train, prediction(y_train_pred, tr_thresholds
,
train_fpr, train_tpr)), range(2),range(2))
sns.set(font_scale=1.4)#for label
sns.heatmap(conf_matr_df_train, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
the maximum value of tpr*(1-fpr) 0.42129864650769616 for threshold 0.853
```

Out[130]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x220ffe11748>
```



Confusion matrix on test data

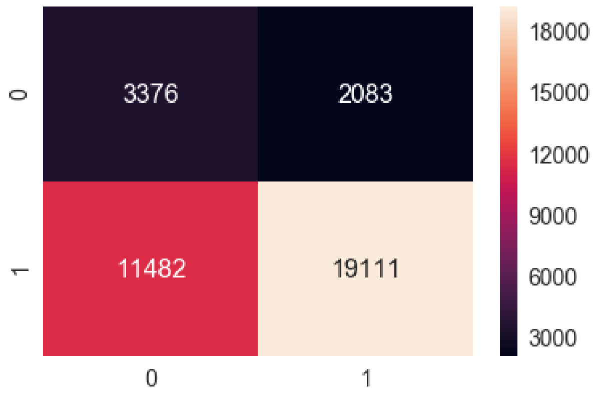
In [131]:

```
conf_matr_df_test = pd.DataFrame(confusion_matrix(y_test, prediction(y_test_pred, tr_thresholds,
train_fpr, train_tpr)), range(2),range(2))
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_test, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
the maximum value of tpr*(1-fpr) 0.42129864650769616 for threshold 0.853
```

Out[131]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x220851e9c50>
```



2.5 Logistic Regression with added Features `Set 5`

Generating new set of features without text

Generating new set of features without text

In [78]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer_tfidf_essay = TfidfVectorizer( min_df=10,max_features=2000)
vectorizer_tfidf_essay.fit(S_train["clean_essays"])
text_tfidf_train = vectorizer_tfidf_essay.transform(S_train["clean_essays"])
print("Shape of matrix after one hot encoding ",text_tfidf_train.shape)
```

```
Shape of matrix after one hot encoding  (51237, 2000)
```

In [79]:

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

```
Shape of matrix after one hot encoding  (36052, 2000)
```

In [86]:

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

```
Shape of matrix after one hot encoding  (21959, 2000)
```

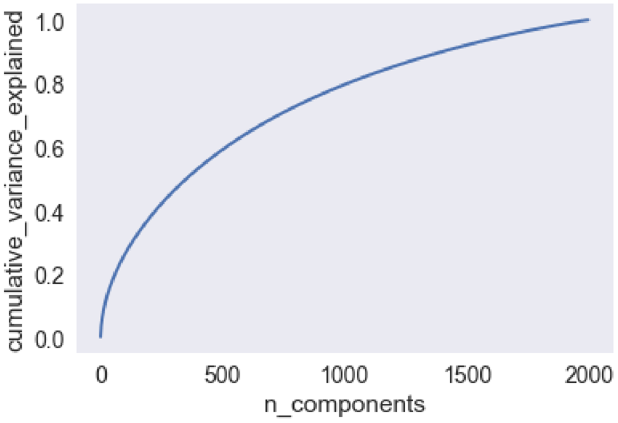
In [125]:

```
from sklearn.decomposition import TruncatedSVD
from scipy.sparse import csr_matrix
X_sparse = csr_matrix(text_tfidf_train)
tsvd = TruncatedSVD(n_components=X_sparse.shape[1]-1)
X_tsvd = tsvd.fit(text_tfidf_train)
percentage_var_explained=tsvd.explained_variance_/np.sum(tsvd.explained_variance_)
cum_var_explained=np.cumsum(percentage_var_explained)
```

ELBOW METHOD: TO FIND COMPONENTS THAT PRESERVE MOST VARIANCE

In [126]:

```
plt.figure(1,figsize=(6,4))
plt.clf()
plt.plot(cum_var_explained,linewidth=2)
plt.axis('tight')
plt.grid()
plt.xlabel('n_components')
plt.ylabel('cumulative_variance_explained')
plt.show()
```



we will preserve 90% of variance at n_componenets=1500

In [113]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer_tfidf_essay = TfidfVectorizer( min_df=10,max_features=1500)
vectorizer_tfidf_essay.fit(S_train["clean_essays"])
text_tfidf_train_new = vectorizer_tfidf_essay.transform(S_train["clean_essays"])
print("Shape of matrix after one hot encoding ",text_tfidf_train_new.shape)
```

```
Shape of matrix after one hot encoding  (51237, 1500)
```

In [114]:

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

```
Shape of matrix after one hot encoding  (36052, 1500)
```

In [115]:

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

```
Shape of matrix after one hot encoding  (21959, 1500)
```

In [130]:

```
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
with_tfidf_text_train=
hstack((categories_one_hot_train,sub_categories_one_hot_train,school_state_one_hot_train,teacher_pr
efix_one_hot_train,clean_project_grade_category_one_hot_train,price_standardized_train,prev_project
_standardized_train,quantity_standardized_train,title_word_count_train,essay_word_count_train,essa
y_sent_pos_train,essay_sent_neg_train,essay_sent_neu_train,essay_sent_comp_train,text_tfidf_train_r
ew)).tocsr()
print(with_tfidf_text_train.shape)
with_tfidf_text_test=
hstack((categories_one_hot_test,sub_categories_one_hot_test,school_state_one_hot_test,teacher_prefi
x_one_hot_test,clean_project_grade_category_one_hot_test,price_standardized_test,prev_project_stand
ardized_test,quantity_standardized_test,title_word_count_test,essay_word_count_test,essay_sent_pos_
test,essay_sent_neg_test,essay_sent_neu_test,essay_sent_comp_test,text_tfidf_test_new)).tocsr()
print(with_tfidf_text_test.shape)
with_tfidf_text_cv=
hstack((categories_one_hot_cv,sub_categories_one_hot_cv,school_state_one_hot_cv,teacher_prefix_one_
hot_cv,clean_project_grade_category_one_hot_cv,price_standardized_cv,prev_project_standardized_cv,
quantity_standardized_cv,title_word_count_cv,essay_word_count_cv,essay_sent_pos_cv,essay_sent_neg_c
v,essay_sent_neu_cv,essay_sent_comp_cv,text_tfidf_cv_new)).tocsr()
print(with_tfidf_text_cv.shape)
```

```
(51237, 1608)
(36052, 1608)
(21959, 1608)
```

Finding best hyperparameter using CV

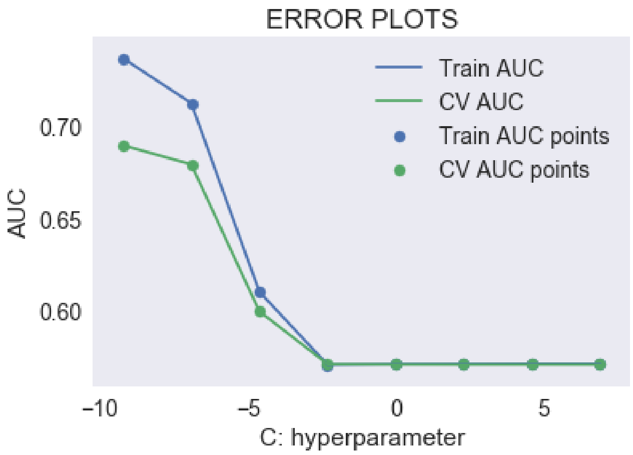
In [127]:

```
from sklearn.linear_model import SGDClassifier
from sklearn.calibration import CalibratedClassifierCV
import matplotlib.pyplot as plt
from sklearn.metrics import roc_auc_score
train_auc = []
cv_auc = []
a = []
b = []
import math
alpha=[10**x for x in range(-4,4)]

for i in tqdm(alpha):
    svm= SGDClassifier(alpha=i,loss='hinge', class_weight='balanced' )
    s=svm.fit(with_tfidf_text_train, y_train)
    clfcalibrated=CalibratedClassifierCV(svm,cv='prefit',method='isotonic')
    clfcalibrated.fit(with_tfidf_text_cv,y_cv)
    y_train_pred = batch_predict(clfcalibrated,with_tfidf_text_train)
    y_cv_pred = batch_predict(clfcalibrated, with_tfidf_text_cv)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
# not the predicted outputs
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
    a.append(y_train_pred)
    b.append(y_cv_pred)

plt.plot([math.log(i) for i in alpha],train_auc, label='Train AUC')
plt.plot([math.log(i) for i in alpha],cv_auc, label='CV AUC')
plt.scatter([math.log(i) for i in alpha],train_auc, label='Train AUC points')
plt.scatter([math.log(i) for i in alpha],cv_auc, label='CV AUC points')
plt.legend()
plt.xlabel("C: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```

```
100%|██████████| 8/8 [00:16<00:00,  1.54s/it]
```



Finding best hyperparameter using GridSearchCV

In [129]:

```
from sklearn.linear_model import SGDClassifier
from sklearn.model_selection import GridSearchCV
svm = SGDClassifier(loss='hinge',class_weight='balanced')
alpha_vals=[10**x for x in range(-4,4)]
penalty=['l1','l2']
parameters = {'alpha':alpha_vals,'penalty':penalty}
clf = GridSearchCV(svm, parameters, cv= 10, scoring='roc_auc')
best_model=clf.fit(with_tfidf_text_train,y_train)
print('Best alpha:',best_model.best_estimator_.get_params()['alpha'])
print('Best penalty:',best_model.best_estimator_.get_params()['penalty'])
```

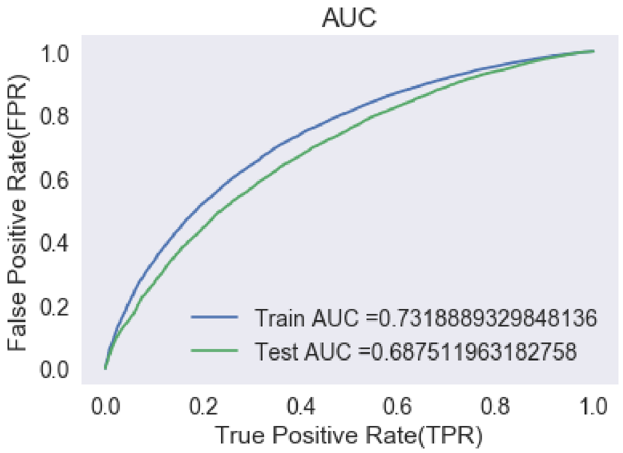
```
Best alpha: 0.0001
Best penalty: l1
```

lets take C=0.0001 from graph

In [134]:

```
from sklearn.metrics import roc_curve, auc
svm= SGDClassifier(alpha=0.0001,loss='hinge', penalty='l1', class_weight='balanced', )
s=svm.fit(with_tfidf_text_train[0:25000,:], y_train[0:25000])
clfcalibrated=CalibratedClassifierCV(svm,method='isotonic')
clfcalibrated.fit(with_tfidf_text_train[25000:51237,:],y_train[25000:51237])
y_train_pred = batch_predict(clfcalibrated,with_tfidf_text_train)
y_test_pred = batch_predict(clfcalibrated,with_tfidf_text_test)
train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
```

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



Confusion matix for train data:

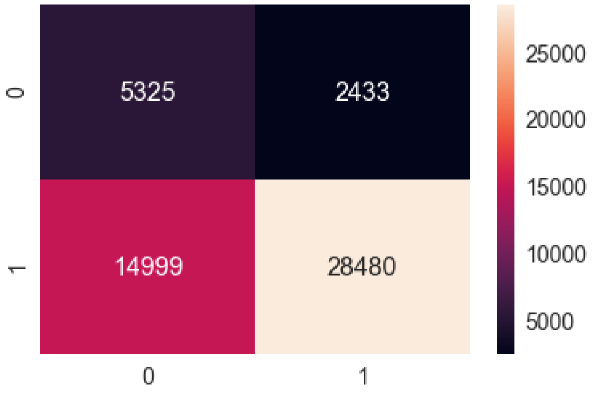
In [135]:

```
conf_matr_df_train = pd.DataFrame(confusion_matrix(y_train, prediction(y_train_pred, tr_thresholds
,
train_fpr, train_tpr)), range(2),range(2))
sns.set(font_scale=1.4)#for label
sns.heatmap(conf_matr_df_train, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
the maximum value of tpr*(1-fpr) 0.45356347812930176 for threshold 0.844
```

Out[135]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1f6f363eb00>
```



Confusion matrix for test data:

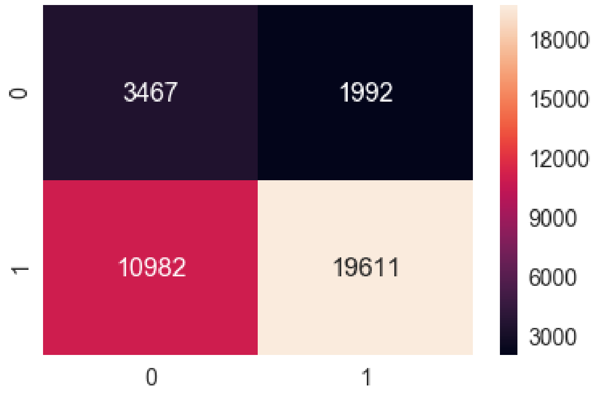
In [136]:

```
conf_matr_df_test = pd.DataFrame(confusion_matrix(y_test, prediction(y_test_pred, tr_thresholds,
train_fpr, train_tpr)), range(2),range(2))
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_test, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
the maximum value of tpr*(1-fpr) 0.45356347812930176 for threshold 0.844
```

```
Out[136]:
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1f6d63beb70>
```



3. Conclusion

```
In [124]:
```

```
# 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","Alpha:HyperParameter","AUC","Best-Penalty"]
x.add_row(["BOW","SGD-HINGE LOSS",0.01, 0.6,'l2'])
x.add_row(["TFIDF","SGD-HINGE LOSS",0.001, 0.58,'l2'])
x.add_row(["AVG W2V","SGD-HINGE LOSS",0.001, 0.6,'l2'])
x.add_row(["TFIDF W2V","SGD-HINGE LOSS",0.001, 0.62,'l2'])
x.add_row(["With tfidf TEXT","SGD-HINGE LOSS",0.0001, 0.58,'l1'])
print(x)
```

```
+-----------------+----------------+----------------------+------+--------------+
|    Vectorizer   |     Model      | Alpha:HyperParameter | AUC  | Best-Penalty |
+-----------------+----------------+----------------------+------+--------------+
|       BOW       | SGD-HINGE LOSS |         0.01         | 0.6  |      l2      |
|      TFIDF      | SGD-HINGE LOSS |        0.001         | 0.58 |      l2      |
|     AVG W2V     | SGD-HINGE LOSS |        0.001         | 0.6  |      l2      |
|    TFIDF W2V    | SGD-HINGE LOSS |        0.001         | 0.62 |      l2      |
| With tfidf TEXT | SGD-HINGE LOSS |        0.0001        | 0.58 |      l1      |
+-----------------+----------------+----------------------+------+--------------+
```

Conclusion: 1.TFIDF W2V has highest AUC Value. 2.Most of the times or 80% of times 'l2' was best regularization. 3.The computation time taken or time complexity is very less as compared to KNN.