# 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** |
| **`project_title`** | Title of the project. **Examples:**<br>• `Art Will Make You Happy!`<br>• `First Grade Fun` |
| **`project_grade_category`** | Grade level of students for which the project is targete enumerated values:<br>• `Grades PreK-2`<br>• `Grades 3-5`<br>• `Grades 6-8`<br>• `Grades 9-12` |
| **`project_subject_categories`** | One or more (comma-separated) subject categories f following enumerated list of values:<br>• `Applied Learning`<br>• `Care & Hunger`<br>• `Health & Sports`<br>• `History & Civics`<br>• `Literacy & Language`<br>• `Math & Science`<br>• `Music & The Arts`<br>• `Special Needs`<br>• `Warmth`<br>**Examples:**<br>• `Music & The Arts`<br>• `Literacy & Language, Math & Science` |
| **`school_state`** | State where school is located (Two-letter U.S. postal (https://en.wikipedia.org/wiki/List_of_U.S._state_abbr **Example:** `WY` |
| **`project_subject_subcategories`** | One or more (comma-separated) subject subcategori **Examples:**<br>• `Literacy`<br>• `Literature & Writing, Social Sciences` |
| **`project_resource_summary`** | An explanation of the resources needed for the projec<br>• `My students need hands on literacy mate sensory needs!` |

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

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

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

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

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

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

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

## Notes on the Essay Data

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

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

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

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

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

In [311]:

```
%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 [312]:

```
project_data = pd.read_csv('train_data.csv')
resource_data = pd.read_csv('resources.csv')
```

In [313]:

```
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 [314]:

```
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[314]:

| | 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 |

# 1.1 Sorted by time

In [315]:

```
#https://stats.stackexchange.com/questions/341312/train-test-split-with-time-and-person
-indexed-data
# how to replace elements in list python: https://stackoverflow.com/a/2582163/4084039
cols = ['Date' if x=='project_submitted_datetime' else x for x in list(project_data.col
umns)]


#sort dataframe based on time pandas python: https://stackoverflow.com/a/49702492/40840
39
project_data['Date'] = pd.to_datetime(project_data['project_submitted_datetime'])
project_data.drop('project_submitted_datetime', axis=1, inplace=True)
project_data.sort_values(by=['Date'], inplace=True)


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


project_data.head(2)
```

Out[315]:

| | Unnamed: 0 | id | teacher_id | teacher_prefix | scho |
|---|---|---|---|---|---|
| **55660** | 8393 | p205479 | 2bf07ba08945e5d8b2a3f269b2b3cfe5 | Mrs. | CA |
| **76127** | 37728 | p043609 | 3f60494c61921b3b43ab61bdde2904df | Ms. | UT |

## 1.2 Adding resource data in dataframe

In [316]:

```
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[316]:

| | 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 |

In [317]:

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

In [318]:

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

Out[318]:

| | Unnamed: 0 | id | teacher_id | teacher_prefix | school_st |
|---|---|---|---|---|---|
| **0** | 8393 | p205479 | 2bf07ba08945e5d8b2a3f269b2b3cfe5 | Mrs. | CA |
| **1** | 37728 | p043609 | 3f60494c61921b3b43ab61bdde2904df | Ms. | UT |

In [319]:

```
project_grade_category = []

for i in range(len(project_data)):
    a = project_data["project_grade_category"][i].replace(" ", "_")
    project_grade_category.append(a)


project_grade_category[0:5]
```

Out[319]:

```
['Grades_PreK-2', 'Grades_3-5', 'Grades_PreK-2', 'Grades_PreK-2', 'Grades_
3-5']
```

In [320]:

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

In [321]:

```
#project_data=project_data.tail(50000)
#project_data.shape
```

## 1.3 preprocessing of `project_subject_categories`

In [322]:

```
catogories = list(project_data['project_subject_categories'].values)
# remove special characters from list of strings python: https://stackoverflow.com/a/47
301924/4084039

# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-stri
ng
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-pyth
on
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", "Warmt
h", "Care & Hunger"]
        if 'The' in j.split(): # this will split each of the catogory based on space "M
ath & Science"=> "Math","&", "Science"
            j=j.replace('The','') # if we have the words "The" we are going to replace
 it with ''(i.e removing 'The')
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"M
ath & Science"=>"Math&Science"
        temp+=j.strip()+" " #" abc ".strip() will return "abc", remove the trailing spa
ces
        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]))
```

## 1.4 preprocessing of `project_subject_subcategories`

In [323]:

```
sub_catogories = list(project_data['project_subject_subcategories'].values)
# remove special characters from list of strings python: https://stackoverflow.com/a/47
301924/4084039

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

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

In [324]:

```
project_data.columns
```

Out[324]:

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'Date', '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_approve
d',
       'price', 'quantity', 'project_grade_category', 'clean_categories',
       'clean_subcategories'],
      dtype='object')
```

# 1.3 Text preprocessing

In [325]:

```
# 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 [326]:

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

Out[326]:

|  | Unnamed: 0 | id | teacher_id | teacher_prefix | school_s |
|---|---|---|---|---|---|
| **0** | 8393 | p205479 | 2bf07ba08945e5d8b2a3f269b2b3cfe5 | Mrs. | CA |
| **1** | 37728 | p043609 | 3f60494c61921b3b43ab61bdde2904df | Ms. | UT |

In [327]:

```
#### 1.4.2.3 Using Pretrained Models: TFIDF weighted W2V
```

In [328]:

```
'''# printing some random reviews
print(project_data['essay'].values[0])
print("="*50)
print(project_data['essay'].values[150])
print("="*50)
print(project_data['essay'].values[1000])
print("="*50)
print(project_data['essay'].values[20000])
print("="*50)
print(project_data['essay'].values[99999])
print("="*50)'''
```

Out[328]:

```
'# printing some random reviews\nprint(project_data[\'essay\'].values[0])
\nprint("="*50)\nprint(project_data[\'essay\'].values[150])\nprint("="*50)
\nprint(project_data[\'essay\'].values[1000])\nprint("="*50)\nprint(projec
t_data[\'essay\'].values[20000])\nprint("="*50)\nprint(project_data[\'essa
y\'].values[99999])\nprint("="*50)'
```

In [329]:

```
# 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 [330]:

```
'''sent = decontracted(project_data['essay'].values[20000])
print(sent)
print("="*50)'''
```

Out[330]:

```
'sent = decontracted(project_data[\'essay\'].values[20000])\nprint(sent)\n
print("="*50)'
```

In [331]:

```
'''# \r \n \t remove from string python: http://texthandler.com/info/remove-line-breaks
-python/
sent = sent.replace('\\r', ' ')
sent = sent.replace('\\"', ' ')
sent = sent.replace('\\n', ' ')
print(sent)'''
```

Out[331]:

```
'# \r \n \t remove from string python: http://texthandler.com/info/remove-
line-breaks-python/\nsent = sent.replace(\'\\r\', \' \')\nsent = sent.repl
ace(\'\\"\', \' \')\nsent = sent.replace(\'\\n\', \' \')\nprint(sent)'
```

In [332]:

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

Out[332]:

```
"#remove spacial character: https://stackoverflow.com/a/5843547/4084039\ns
ent = re.sub('[^A-Za-z0-9]+', ' ', sent)\nprint(sent)"
```

In [333]:

```
# 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'r
e", "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', 't
hey', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "th
at'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'ha
d', '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', 'ov
er', 'under', 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'an
y', 'both', 'each', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too'
, 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'no
w', 'd', 'll', 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't",
'doesn', "doesn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'migh
tn', "mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'w
asn', "wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"]
```

## 1.3.1Preprocess of Preprocessing of `essay`

In [334]:

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

```
100%|█████████████████████████████████████████████████████████████████████
███| 109248/109248 [00:46<00:00, 2334.62it/s]
```

In [335]:

```
# after preprocesing
#preprocessed_essays[10:]
```

In [336]:

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

## 1.3.2Preprocessing of `project_title`

In [337]:

```
# similarly you can preprocess the titles also
```

In [338]:

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

```
100%|███████████████████████████████████████████████████████████████████
██| 109248/109248 [00:02<00:00, 52186.92it/s]
```

In [339]:

```
# after preprocesing
#preprocessed_project_title[1000]
```

In [340]:

```
#https://stackoverflow.com/questions/26666919/add-column-in-dataframe-from-list/3849072
7
project_data['preprocessed_project_title'] = preprocessed_project_title
project_data.drop(['project_title'], axis=1, inplace=True)
```

In [341]:

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

Out[341]:

| | Unnamed: 0 | id | teacher_id | teacher_prefix | school_s |
|---|---|---|---|---|---|
| **0** | 8393 | p205479 | 2bf07ba08945e5d8b2a3f269b2b3cfe5 | Mrs. | CA |
| **1** | 37728 | p043609 | 3f60494c61921b3b43ab61bdde2904df | Ms. | UT |

In [342]:

```
project_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 109248 entries, 0 to 109247
Data columns (total 20 columns):
Unnamed: 0                                      109248 non-null int64
id                                              109248 non-null object
teacher_id                                      109248 non-null object
teacher_prefix                                  109245 non-null object
school_state                                    109248 non-null object
Date                                            109248 non-null datetime64
[ns]
project_essay_1                                 109248 non-null object
project_essay_2                                 109248 non-null object
project_essay_3                                 3758 non-null object
project_essay_4                                 3758 non-null object
project_resource_summary                        109248 non-null object
teacher_number_of_previously_posted_projects    109248 non-null int64
project_is_approved                             109248 non-null int64
price                                           109248 non-null float64
quantity                                        109248 non-null int64
project_grade_category                          109248 non-null object
clean_categories                                109248 non-null object
clean_subcategories                             109248 non-null object
preprocessed_essays                             109248 non-null object
preprocessed_project_title                      109248 non-null object
dtypes: datetime64[ns](1), float64(1), int64(4), object(14)
memory usage: 22.5+ MB
```

# 1.5 Preparing data for models

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
```

### 1.5.4 Merging all the above features

**Computing Sentiment Scores**

In [343]:

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

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

sid = SentimentIntensityAnalyzer()

for_sentiment = 'a person is a person no matter how small dr seuss i teach the smallest
students with the biggest enthusiasm \
for learning my students learn in many different ways using all of our senses and multi
ple intelligences i use a wide range\
of techniques to help all my students succeed students in my class come from a variety
 of different backgrounds which makes\
for wonderful sharing of experiences and cultures including native americans our school
is a caring community of successful \
learners which can be seen through collaborative student project based learning in and
 out of the classroom kindergarteners \
in my class love to work with hands on materials and have many different opportunities
 to practice a skill before it is\
mastered having the social skills to work cooperatively with friends is a crucial aspec
t of the kindergarten curriculum\
montana is the perfect place to learn about agriculture and nutrition my students love
 to role play in our pretend kitchen\
in the early childhood classroom i have had several kids ask me can we try cooking with
real food i will take their idea \
and create common core cooking lessons where we learn important math and writing concep
ts while cooking delicious healthy \
food for snack time my students will have a grounded appreciation for the work that wen
t into making the food and knowledge \
of where the ingredients came from as well as how it is healthy for their bodies this p
roject would expand our learning of \
nutrition and agricultural cooking recipes by having us peel our own apples to make hom
emade applesauce make our own bread \
and mix up healthy plants from our classroom garden in the spring we will also create o
ur own cookbooks to be printed and \
shared with families students will gain math and literature skills as well as a life lo
ng enjoyment for healthy cooking \
nannan'
ss = sid.polarity_scores(for_sentiment)

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

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

Out[343]:

```
"import nltk\nfrom nltk.sentiment.vader import SentimentIntensityAnalyzer
\n\n# import nltk\n# nltk.download('vader_lexicon')\n\nsid = SentimentInte
nsityAnalyzer()\n\nfor_sentiment = 'a person is a person no matter how sma
ll dr seuss i teach the smallest students with the biggest enthusiasm for
learning my students learn in many different ways using all of our senses
and multiple intelligences i use a wide rangeof techniques to help all my
students succeed students in my class come from a variety of different bac
kgrounds which makesfor wonderful sharing of experiences and cultures incl
uding native americans our school is a caring community of successful lear
ners which can be seen through collaborative student project based learnin
g in and out of the classroom kindergarteners in my class love to work wit
h hands on materials and have many different opportunities to practice a s
kill before it ismastered having the social skills to work cooperatively w
ith friends is a crucial aspect of the kindergarten curriculummontana is t
he perfect place to learn about agriculture and nutrition my students love
to role play in our pretend kitchenin the early childhood classroom i have
had several kids ask me can we try cooking with real food i will take thei
r idea and create common core cooking lessons where we learn important mat
h and writing concepts while cooking delicious healthy food for snack time
my students will have a grounded appreciation for the work that went into
making the food and knowledge of where the ingredients came from as well a
s how it is healthy for their bodies this project would expand our learnin
g of nutrition and agricultural cooking recipes by having us peel our own
apples to make homemade applesauce make our own bread and mix up healthy p
lants from our classroom garden in the spring we will also create our own
cookbooks to be printed and shared with families students will gain math a
nd literature skills as well as a life long enjoyment for healthy cooking
nannan'\nss = sid.polarity_scores(for_sentiment)\n\nfor k in ss:\n    prin
t('{0}: {1}, '.format(k, ss[k]), end='')\n\n# we can use these 4 things as
features/attributes (neg, neu, pos, compound)\n# neg: 0.0, neu: 0.753, po
s: 0.247, compound: 0.93"
```

In [344]:

```
!pip install vaderSentiment
#sentiment analysis of essay
#____________________________________________
#https://medium.com/analytics-vidhya/simplifying-social-media-sentiment-analysis-using-vader-in-python-f9e6ec6fc52f

from nltk.sentiment.vader import SentimentIntensityAnalyzer

import nltk
nltk.download('vader_lexicon')
catogories = list(project_data['preprocessed_essays'].values)
sentiment_positive=[]
sentiment_negative=[]
sentiment_neutral=[]
sentiment_compound=[]
#cat_list = []
for i in catogories:
  sid = SentimentIntensityAnalyzer()
  sentiment_val=sid.polarity_scores(i)
  #cat_list.append(sentiment_val)
  sentiment_positive.append(sentiment_val['pos'])
  sentiment_negative.append(sentiment_val['neg'])
  sentiment_neutral.append(sentiment_val['neu'])
  sentiment_compound.append(sentiment_val['compound'])

#project_data['sentiment_pos_value'] = cat_list
project_data['sentiment_pos_essay']=sentiment_positive
project_data['sentiment_neg_essay']=sentiment_negative
project_data['sentiment_neu_essay']=sentiment_neutral
project_data['sentiment_compound_essay']=sentiment_compound
```

```
Requirement already satisfied: vaderSentiment in c:\programdata\anaconda3
\lib\site-packages (3.2.1)
```

```
wxpython 4.0.3 requires PyPubSub, which is not installed.
distributed 1.21.8 requires msgpack, which is not installed.
You are using pip version 10.0.1, however version 19.1.1 is available.
You should consider upgrading via the 'python -m pip install --upgrade pi
p' command.
```

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

In [345]:

```
#!pip install vaderSentiment
```

# Assignment 7: SVMn

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

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

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

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

3. **Representation of results**

    - You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure.
    - Once after you found the best hyper parameter, you need to train your model with it, and find the AUC on test data and plot the ROC curve on both train and test.
    - Along with plotting ROC curve, you need to print the confusion matrix (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/confusion-matrix-tpr-fpr-fnr-tnr-1/) with predicted and original labels of test data points. Please visualize your confusion matrices using seaborn heatmaps.
      (https://seaborn.pydata.org/generated/seaborn.heatmap.html)
      (https://seaborn.pydata.org/generated/seaborn.heatmap.html)

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

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

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

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

    - Consider these set of features Set 5 :
      (https://seaborn.pydata.org/generated/seaborn.heatmap.html)
        - **school_state** : categorical data
        - **clean_categories** : categorical data
        - **clean_subcategories** : categorical data
        - **project_grade_category** :categorical data
        - **teacher_prefix** : categorical data
        - **quantity** : numerical data
        - **teacher_number_of_previously_posted_projects** : numerical data

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

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

**Note: Data Leakage**

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

# 2. Support Vector Machines

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

In [346]:

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

In [347]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import model_selection
from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score

from collections import Counter
from sklearn.metrics import accuracy_score

from sklearn.model_selection import cross_val_score
from sklearn.model_selection import cross_validate
```

In [348]:

```
y=project_data['project_is_approved']
y.shape
```

Out[348]:

```
(109248,)
```

In [349]:

```
#replace NAN to space https ://stackoverflow.com/questions/49259305/raise-valueerrornp-
nan-is-an-invalid-document-expected-byte-or?rq=1
project_data['teacher_prefix'] = project_data['teacher_prefix'].fillna(' ')
```

In [350]:

```
#https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_s
plit.html

#split the data into train and test fo bag of words

x_train,x_test,y_train,y_test=model_selection.train_test_split(project_data,y,test_size
=0.3,random_state=0)
#split train into cross val train and cross val test
#x_train,x_cv,y_train,y_cv=model_selection.train_test_split(x_t,y_t,test_size=0.3,rando
m_state=0)
```

In [351]:

```
project_data.columns
```

Out[351]:

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'Date', 'project_essay_1', 'project_essay_2', 'project_essay_3',
       'project_essay_4', 'project_resource_summary',
       'teacher_number_of_previously_posted_projects', 'project_is_approve
d',
       'price', 'quantity', 'project_grade_category', 'clean_categories',
       'clean_subcategories', 'preprocessed_essays',
       'preprocessed_project_title', 'sentiment_pos_essay',
       'sentiment_neg_essay', 'sentiment_neu_essay',
       'sentiment_compound_essay'],
      dtype='object')
```

## 2.2 Make Data Model Ready: encoding numerical, categorical features</h2>

### 2.2.1 encoding categorical features

In [352]:

```
# please write all the code with proper documentation, and proper titles for each subse
ction
# go through documentations and blogs before you start coding
# first figure out what to do, and then think about how to do.
# reading and understanding error messages will be very much helpfull in debugging your
code
# make sure you featurize train and test data separatly

# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the reader
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label
```

In [353]:

```
x_train.head(2)
```

Out[353]:

| | Unnamed: 0 | id | teacher_id | teacher_prefix | sch |
|---|---|---|---|---|---|
| **34921** | 114444 | p155197 | 72aca64240ff04da1395fb68ed8dd789 | Mrs. | WA |
| **101335** | 89212 | p202747 | f09b36021ccef9bbb32ffde05d9ae73a | Mrs. | TX |

2 rows × 24 columns

In [354]:

```
#one hot encoding for clean_categories
#_________________________
# we use count vectorizer to convert the values into one
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(vocabulary=list(sorted_cat_dict.keys()), lowercase=False,
binary=True)
x_train_categories_one_hot = vectorizer.fit_transform(x_train['clean_categories'].value
s)
#x_cv_categories_one_hot = vectorizer.fit_transform(x_cv['clean_categories'].values)
x_test_categories_one_hot = vectorizer.fit_transform(x_test['clean_categories'].values)
print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encodig ",x_train_categories_one_hot.shape)
#print("Shape of matrix after one hot encodig ",x_cv_categories_one_hot.shape)
print("Shape of matrix after one hot encodig ",x_test_categories_one_hot.shape)
```

```
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearnin
g', 'SpecialNeeds', 'Health_Sports', 'Math_Science', 'Literacy_Language']
Shape of matrix after one hot encodig  (76473, 9)
Shape of matrix after one hot encodig  (32775, 9)
```

In [355]:

```
#one hot encoding for clean_subcategories
#__________________________
# we use count vectorizer to convert the values into one
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()), lowercase=False, binary=True)
x_train_subcategories_one_hot = vectorizer.fit_transform(x_train['clean_subcategories'].values)
#x_cv_subcategories_one_hot = vectorizer.fit_transform(x_cv['clean_subcategories'].values)
x_test_subcategories_one_hot = vectorizer.fit_transform(x_test['clean_subcategories'].values)
print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encodig ",x_train_subcategories_one_hot.shape)
#print("Shape of matrix after one hot encodig ",x_cv_subcategories_one_hot.shape)
print("Shape of matrix after one hot encodig ",x_test_subcategories_one_hot.shape)
```

```
['Economics', 'CommunityService', 'FinancialLiteracy', 'ParentInvolvement', 'Extracurricular', 'Civics_Government', 'ForeignLanguages', 'NutritionEducation', 'Warmth', 'Care_Hunger', 'SocialSciences', 'PerformingArts', 'CharacterEducation', 'TeamSports', 'Other', 'College_CareerPrep', 'Music', 'History_Geography', 'Health_LifeScience', 'EarlyDevelopment', 'ESL', 'Gym_Fitness', 'EnvironmentalScience', 'VisualArts', 'Health_Wellness', 'AppliedSciences', 'SpecialNeeds', 'Literature_Writing', 'Mathematics', 'Literacy']
Shape of matrix after one hot encodig  (76473, 30)
Shape of matrix after one hot encodig  (32775, 30)
```

In [356]:

```
#one hot encoding for school_state

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

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

#_________________________
# we use count vectorizer to convert the values into one
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(vocabulary=list(sorted_school_state_cat_dict.keys()), lowe
rcase=False, binary=True)
x_train_school_state_one_hot = vectorizer.fit_transform(x_train['school_state'].values)
#x_cv_school_state_one_hot = vectorizer.fit_transform(x_cv['school_state'].values)
x_test_school_state_one_hot = vectorizer.fit_transform(x_test['school_state'].values)
print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encodig ",x_train_school_state_one_hot.shape)
#print("Shape of matrix after one hot encodig ",x_cv_school_state_one_hot.shape)
print("Shape of matrix after one hot encodig ",x_test_school_state_one_hot.shape)
```

```
['VT', 'WY', 'ND', 'MT', 'RI', 'SD', 'NE', 'DE', 'AK', 'NH', 'WV', 'ME',
'HI', 'DC', 'NM', 'KS', 'IA', 'ID', 'AR', 'CO', 'MN', 'OR', 'KY', 'MS', 'N
V', 'MD', 'CT', 'TN', 'UT', 'AL', 'WI', 'VA', 'AZ', 'NJ', 'OK', 'WA', 'M
A', 'LA', 'OH', 'MO', 'IN', 'PA', 'MI', 'SC', 'GA', 'IL', 'NC', 'FL', 'N
Y', 'TX', 'CA']
Shape of matrix after one hot encodig  (76473, 51)
Shape of matrix after one hot encodig  (32775, 51)
```

In [357]:

```
#one hot encoding for project_grade_category
my_counter = Counter()
for project_grade in project_data['project_grade_category'].values:
    my_counter.update(project_grade.split())



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

#_________________________
# we use count vectorizer to convert the values into one
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(vocabulary=list(sorted_project_grade_cat_dict.keys()), low
ercase=False, binary=True)
x_train_grade_category_one_hot = vectorizer.fit_transform(x_train['project_grade_catego
ry'].values)
#x_cv_grade_category_one_hot = vectorizer.fit_transform(x_cv['project_grade_category'].
values)
x_test_grade_category_one_hot = vectorizer.fit_transform(x_test['project_grade_categor
y'].values)
print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encodig ",x_train_grade_category_one_hot.shape)
#print("Shape of matrix after one hot encodig ",x_cv_grade_category_one_hot.shape)
print("Shape of matrix after one hot encodig ",x_test_grade_category_one_hot.shape)
```

```
['Grades_9-12', 'Grades_6-8', 'Grades_3-5', 'Grades_PreK-2']
Shape of matrix after one hot encodig  (76473, 4)
Shape of matrix after one hot encodig  (32775, 4)
```

In [358]:

```
#one hot encoding for prefix_category


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

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


#_________________________
# we use count vectorizer to convert the values into one
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(vocabulary=list(sorted_teacher_prefix_cat_dict.keys()), lowercase=False, binary=True)
x_train_prefix_one_hot = vectorizer.fit_transform(x_train['teacher_prefix'].values)
#x_cv_prefix_one_hot = vectorizer.fit_transform(x_cv['teacher_prefix'].values)
x_test_prefix_one_hot = vectorizer.fit_transform(x_test['teacher_prefix'].values)
print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encodig ",x_train_prefix_one_hot.shape)
#print("Shape of matrix after one hot encodig ",x_cv_prefix_one_hot.shape)
print("Shape of matrix after one hot encodig ",x_test_prefix_one_hot.shape)
```

```
['Dr.', 'Teacher', 'Mr.', 'Ms.', 'Mrs.']
Shape of matrix after one hot encodig  (76473, 5)
Shape of matrix after one hot encodig  (32775, 5)
```

## 2.2.2 encoding numerical features</font>

In [359]:

```
x_train.head(2)
```

Out[359]:

| | Unnamed: 0 | id | teacher_id | teacher_prefix | sch |
|---|---|---|---|---|---|
| **34921** | 114444 | p155197 | 72aca64240ff04da1395fb68ed8dd789 | Mrs. | WA |
| **101335** | 89212 | p202747 | f09b36021ccef9bbb32ffde05d9ae73a | Mrs. | TX |

2 rows × 24 columns

In [360]:

```
from sklearn.preprocessing import Normalizer

normalizer = Normalizer()

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

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

x_train_price_standardized = normalizer.transform(x_train['price'].values.reshape(-1,1
))
#x_cv_price_standardized = normalizer.transform(x_cv['price'].values.reshape(-1,1))
x_test_price_standardized = normalizer.transform(x_test['price'].values.reshape(-1,1))

print("After vectorizations")
print(x_train_price_standardized.shape, y_train.shape)
#print(x_cv_price_standardized.shape, y_cv.shape)
print(x_test_price_standardized.shape, y_test.shape)
```

```
After vectorizations
(76473, 1) (76473,)
(32775, 1) (32775,)
```

## 2.2.3 merge numerical and categorical data

In [361]:

```
# 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
:)
x_train_ohe = hstack((x_train_categories_one_hot, x_train_subcategories_one_hot, x_trai
n_school_state_one_hot, x_train_grade_category_one_hot, x_train_prefix_one_hot, x_train
_price_standardized))
#x_cv_ohe = hstack((x_cv_categories_one_hot, x_cv_subcategories_one_hot, x_cv_school_st
ate_one_hot, x_cv_grade_category_one_hot, x_cv_prefix_one_hot, x_cv_price_standardize
d))
x_test_ohe = hstack((x_test_categories_one_hot, x_test_subcategories_one_hot, x_test_sc
hool_state_one_hot, x_test_grade_category_one_hot, x_test_prefix_one_hot, x_test_price_
standardized))

print(x_train_ohe.shape)
#print(x_cv_ohe.shape)
print(x_test_ohe.shape)
```

```
(76473, 100)
(32775, 100)
```

In [362]:

```
print(x_train_categories_one_hot.shape)
print(x_train_subcategories_one_hot.shape)
print(x_train_school_state_one_hot.shape)
print(x_train_grade_category_one_hot.shape)
print(x_train_prefix_one_hot.shape)
print(x_train_price_standardized.shape)
```

```
(76473, 9)
(76473, 30)
(76473, 51)
(76473, 4)
(76473, 5)
(76473, 1)
```

# 2.3 Make Data Model Ready: encoding eassay, and project_title

In [363]:

```
# please write all the code with proper documentation, and proper titles for each subse
ction
# go through documentations and blogs before you start coding
# first figure out what to do, and then think about how to do.
# reading and understanding error messages will be very much helpfull in debugging your
code
# make sure you featurize train and test data separatly

# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the reader
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label
```

# 2.3 Apply Support Vector Machines on different kind of featurization as mentioned in the instructions

Apply Support Vector Machines on different kind of featurization as mentioned in the instructions
For Every model that you work on make sure you do the step 2 and step 3 of instrucations

## 2.3.1 Apply Support Vector Machines with BOW, SET 1

**vectorize the essay and title data, SET 1**

In [364]:

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

In [365]:

```
#vectorize the essay
#______________________________
#https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html

# We are considering only the words which appeared in at least 10 documents(rows or projects).
vectorizer = CountVectorizer(min_df=10, ngram_range=(1,2), max_features=5000)
vectorizer.fit(x_train['preprocessed_essays'].values)# fit has to apply only on train data
#z_bow1=vectorizer.fit(x_train['preprocessed_essays'].values)# fit has to apply only on train data

# we use fitted CountVectorizer to convert the text to vector
x_train_bow_essays = vectorizer.transform(x_train['preprocessed_essays'].values)
#x_cv_bow_essays = vectorizer.transform(x_cv['preprocessed_essays'].values)
x_test_bow_essays = vectorizer.transform(x_test['preprocessed_essays'].values)

print("Shape of matrix after one hot encodig ",x_train_bow_essays.shape, y_train.shape)
#print("Shape of matrix after one hot encodig ",x_cv_bow_essays.shape)
print("Shape of matrix after one hot encodig ",x_test_bow_essays.shape)
```

```
Shape of matrix after one hot encodig  (76473, 5000) (76473,)
Shape of matrix after one hot encodig  (32775, 5000)
```

In [366]:

```
#https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.Coun
tVectorizer.html
#you can vectorize the title
# We are considering only the words which appeared in at least 10 documents(rows or pro
jects).
vectorizer = CountVectorizer(min_df=10, ngram_range=(1,2), max_features=5000)
vectorizer.fit(x_train['preprocessed_project_title'].values)# fit has to apply only on
 train data
#z_bow2=vectorizer.fit(x_train['preprocessed_project_title'].values)# fit has to apply
 only on train data

# we use fitted CountVectorizer to convert the text to vector
x_train_bow_title = vectorizer.transform(x_train['preprocessed_project_title'].values)
#x_cv_bow_title = vectorizer.transform(x_cv['preprocessed_project_title'].values)
x_test_bow_title = vectorizer.transform(x_test['preprocessed_project_title'].values)

print("Shape of matrix after one hot encodig ",x_train_bow_title.shape)
#print("Shape of matrix after one hot encodig ",x_cv_bow_title.shape)
print("Shape of matrix after one hot encodig ",x_test_bow_title.shape)
```

```
Shape of matrix after one hot encodig  (76473, 5000)
Shape of matrix after one hot encodig  (32775, 5000)
```

### merge dataset, SET 1

In [367]:

```
# 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
:)
x_train_bow = hstack((x_train_ohe, x_train_bow_essays, x_train_bow_title)).tocsr()
#x_cv_bow = hstack((x_cv_ohe, x_cv_bow_essays, x_cv_bow_title)).tocsr()
x_test_bow = hstack((x_test_ohe, x_test_bow_essays, x_test_bow_title)).tocsr()


print(x_train_bow.shape)
#print(x_cv_bow.shape)
print(x_test_bow.shape)
```

```
(76473, 10100)
(32775, 10100)
```

In [368]:

```
type(x_train_bow)
```

Out[368]:

```
scipy.sparse.csr.csr_matrix
```

### GridsearchCV, SET 1

In [369]:

```
%time
#https://chrisalbon.com/machine_learning/model_selection/hyperparameter_tuning_using_grid_search/
#https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
#https://stackoverflow.com/questions/48220125/how-to-generate-roc-curve-with-cross-validation-using-sgd-classifier-loss-hinge
#https://www.kaggle.com/udaysa/svm-with-scikit-learn-svm-with-parameter-tuning
from sklearn.model_selection import GridSearchCV,TimeSeriesSplit
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import roc_curve, auc

print("***************Grid search cv perform on train dataset*****************")

svm_linear = SGDClassifier(loss='hinge')

parameters = {'alpha':[1000,500,100,50,10,1,0.1,0.05,0.01,0.005,0.001,0.0005,0.0001],
             'penalty':['l1','l2']}
ts_cv = TimeSeriesSplit(n_splits=10) #For time based splitting
clf = GridSearchCV(svm_linear, parameters, cv= ts_cv, scoring='roc_auc')

clf.fit(x_train_bow, y_train)

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

penalty_hyperparameter_bow=clf.best_params_['penalty']
alpha_hyperparameter_bow=clf.best_params_['alpha']
print("Best HyperParameter: ",clf.best_params_)
print("Best Accuracy: %.2f%%"%(clf.best_score_*100))
```

```
Wall time: 0 ns
***************Grid search cv perform on train dataset*****************
Best HyperParameter:  {'alpha': 0.01, 'penalty': 'l2'}
Best Accuracy: 67.23%
```

In [370]:

```
penalty_hyperparameter_bow
```

Out[370]:

```
'l2'
```

In [371]:

```
alpha_hyperparameter_bow
```

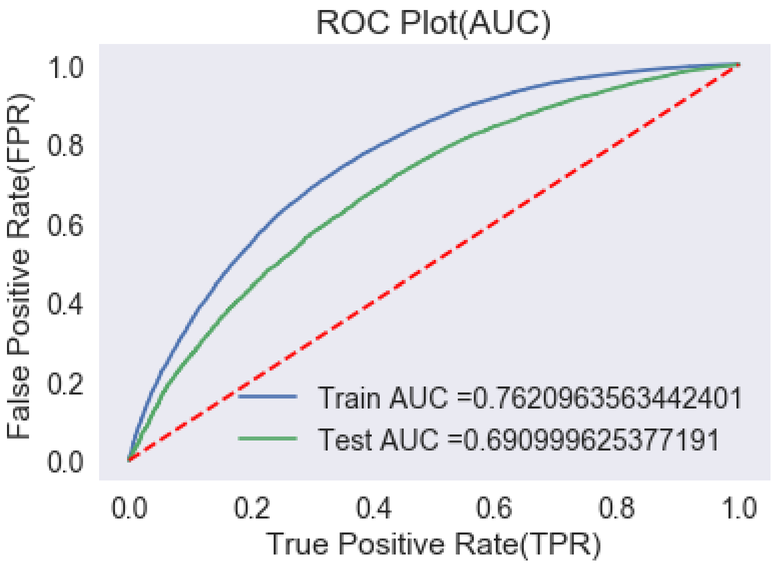
Out[371]:

```
0.01
```

### Implementing on test dataset, and ROC Plot SET 1

In [372]:

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


svm_linear = SGDClassifier(loss='hinge', penalty=penalty_hyperparameter_bow, alpha=alpha_hyperparameter_bow)

svm_linear.fit(x_train_bow, y_train)

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

y_train_pred = svm_linear.decision_function(x_train_bow)
y_test_pred = svm_linear.decision_function(x_test_bow)

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

BOW_auc_train = auc(train_fpr, train_tpr)
BOW_auc_test = auc(test_fpr, test_tpr)

plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.plot([0,1],[0,1],'r--')
plt.legend()
plt.xlabel("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("ROC Plot(AUC)")
plt.grid()
plt.show()
```



### Confusion matrix, SET 1

In [373]:

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

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

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

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

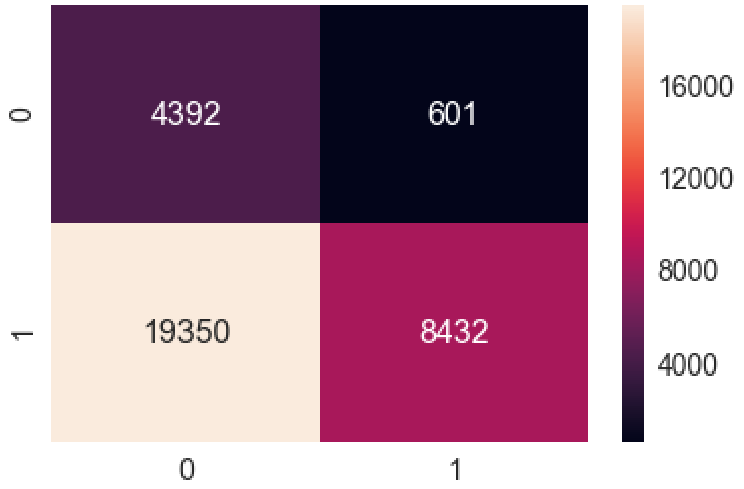
In [374]:

```
print("="*100)
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_tpr
)))
conf_matr_df_test_1 = pd.DataFrame(confusion_matrix(y_test, predict(y_test_pred, tr_thr
esholds, test_fpr, test_tpr)), range(2),range(2))

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

```
==========================================================================
==========================
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.4100615240430981 for threshold 1.236
[[ 4392   601]
 [19350  8432]]
the maximum value of tpr*(1-fpr) 0.4100615240430981 for threshold 1.236
```

Out[374]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1f7aa7e28d0>
```



## 2.3.2 Apply Support Vector Machines with TFIDF, SET 2

### vectorize the essay and title data, SET 2

In [375]:

```
#convert the essay text to vector
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer_tfidf = TfidfVectorizer()
vectorizer_tfidf.fit(x_train['preprocessed_essays'].values)# fit has to apply only on train data
z_tfidf1=vectorizer_tfidf.fit(x_train['preprocessed_essays'].values)# fit has to apply only on train data

# we use fitted CountVectorizer to convert the text to vector
x_train_tfidf_essays = vectorizer_tfidf.transform(x_train['preprocessed_essays'].values)
#x_cv_tfidf_essays = vectorizer_tfidf.transform(x_cv['preprocessed_essays'].values)
x_test_tfidf_essays = vectorizer_tfidf.transform(x_test['preprocessed_essays'].values)

print("Shape of matrix after one hot encodig ",x_train_tfidf_essays.shape, y_train.shape)
#print("Shape of matrix after one hot encodig ",x_cv_tfidf_essays.shape)
print("Shape of matrix after one hot encodig ",x_test_tfidf_essays.shape)
```

```
Shape of matrix after one hot encodig  (76473, 49033) (76473,)
Shape of matrix after one hot encodig  (32775, 49033)
```

In [376]:

```
#convert the title text to vector
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer_tfidf = TfidfVectorizer()
vectorizer_tfidf.fit(x_train['preprocessed_project_title'].values)# fit has to apply only on train data
z_tfidf2=vectorizer_tfidf.fit(x_train['preprocessed_project_title'].values)# fit has to apply only on train data

# we use fitted CountVectorizer to convert the text to vector
x_train_tfidf_title = vectorizer.transform(x_train['preprocessed_project_title'].values)
#x_cv_tfidf_title = vectorizer.transform(x_cv['preprocessed_project_title'].values)
x_test_tfidf_title = vectorizer.transform(x_test['preprocessed_project_title'].values)

print("Shape of matrix after one hot encodig ",x_train_tfidf_title.shape)
#print("Shape of matrix after one hot encodig ",x_cv_tfidf_title.shape)
print("Shape of matrix after one hot encodig ",x_test_tfidf_title.shape)
```

```
Shape of matrix after one hot encodig  (76473, 5000)
Shape of matrix after one hot encodig  (32775, 5000)
```

### merge all sparse data, SET 2

In [377]:

```
# 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
:)
x_train_tfidf = hstack((x_train_ohe, x_train_tfidf_essays, x_train_tfidf_title)).tocsr
()
#x_cv_tfidf = hstack((x_cv_ohe, x_cv_tfidf_essays, x_cv_tfidf_title)).tocsr()
x_test_tfidf = hstack((x_test_ohe, x_test_tfidf_essays, x_test_tfidf_title)).tocsr()

print(x_train_tfidf.shape)
#print(x_cv_tfidf.shape)
print(x_test_tfidf.shape)
```

```
(76473, 54133)
(32775, 54133)
```

### GridsearchCV, SET 2

In [378]:

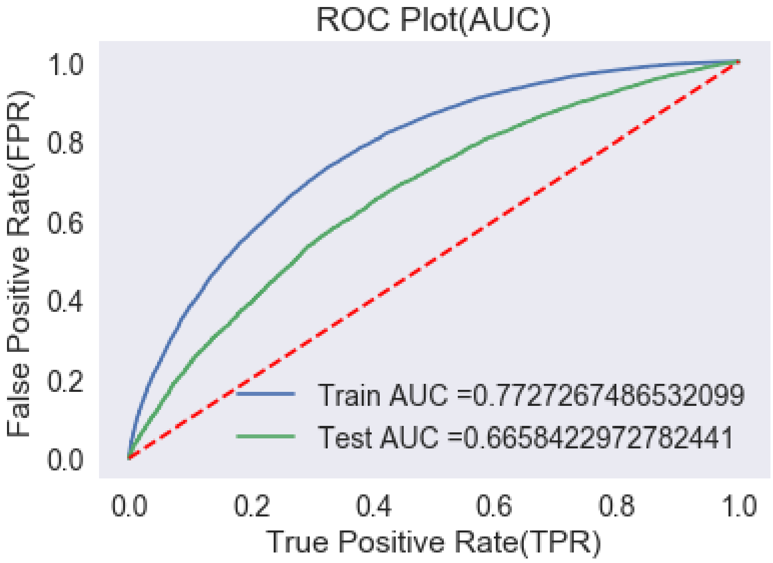
```
%time
#https://chrisalbon.com/machine_learning/model_selection/hyperparameter_tuning_using_gr
id_search/
#https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.h
tml
#https://stackoverflow.com/questions/48220125/how-to-generate-roc-curve-with-cross-vali
dation-using-sgd-classifier-loss-hinge
#https://www.kaggle.com/udaysa/svm-with-scikit-learn-svm-with-parameter-tuning
from sklearn.model_selection import GridSearchCV,TimeSeriesSplit
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import roc_curve, auc

print("***************Grid search cv perform on train dataset*****************")

svm_linear = SGDClassifier(loss='hinge')

parameters = {'alpha':[1000,500,100,50,10,1,0.1,0.05,0.01,0.005,0.001,0.0005,0.0001],
              'penalty':['l1','l2']}
ts_cv = TimeSeriesSplit(n_splits=10) #For time based splitting
clf = GridSearchCV(svm_linear, parameters, cv= ts_cv, scoring='roc_auc')

clf.fit(x_train_tfidf, y_train)

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

penalty_hyperparameter_tfidf=clf.best_params_['penalty']
alpha_hyperparameter_tfidf=clf.best_params_['alpha']
print("Best HyperParameter: ",clf.best_params_)
print("Best Accuracy: %.2f%%"%(clf.best_score_*100))
```

```
Wall time: 0 ns
***************Grid search cv perform on train dataset*****************
Best HyperParameter:  {'alpha': 0.005, 'penalty': 'l2'}
Best Accuracy: 64.79%
```

### Implementing on test dataset, and AUC Plot SET 2

In [379]:

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


svm_linear = SGDClassifier(loss='hinge', penalty=penalty_hyperparameter_tfidf, alpha=alpha_hyperparameter_tfidf)

svm_linear.fit(x_train_tfidf, y_train)

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

y_train_pred = svm_linear.decision_function(x_train_tfidf)
y_test_pred = svm_linear.decision_function(x_test_tfidf)

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

TFIDF_auc_train = auc(train_fpr, train_tpr)
TFIDF_auc_test = auc(test_fpr, test_tpr)

plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.plot([0,1],[0,1],'r--')
plt.legend()
plt.xlabel("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("ROC Plot(AUC)")
plt.grid()
plt.show()
```



### Confusion matrix, SET 2

In [380]:

```
x_test_tfidf.shape
```

Out[380]:

```
(32775, 54133)
```

In [381]:

```
#https://seaborn.pydata.org/generated/seaborn.heatmap.html
#------------------------------------------------
def predict(proba, threshould, fpr, tpr):

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

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

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

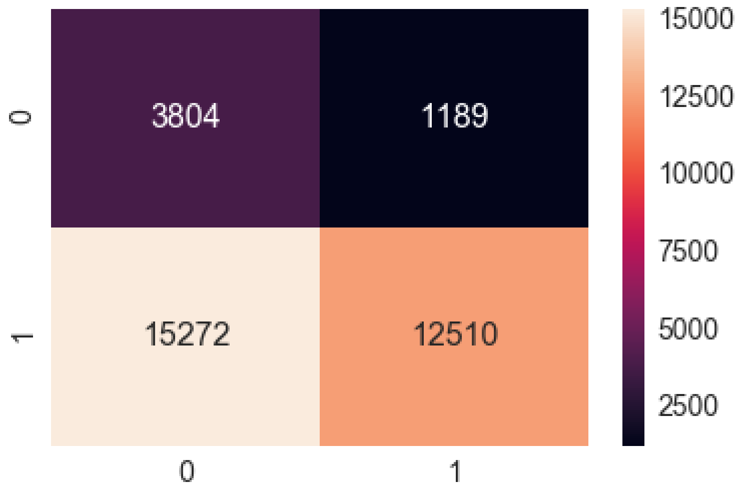
In [382]:

```
print("="*100)
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_tpr
)))
conf_matr_df_test_1 = pd.DataFrame(confusion_matrix(y_test, predict(y_test_pred, tr_thr
esholds, test_fpr, test_tpr)), range(2),range(2))

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

```
==========================================================================
==========================
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.3889660988633673 for threshold 1.004
[[ 3804  1189]
 [15272 12510]]
the maximum value of tpr*(1-fpr) 0.3889660988633673 for threshold 1.004
```

Out[382]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1f831b87ef0>
```



## 2.3.3 Apply Support Vector Machines with AVG W2V, SET 3

#### vectorize using AVG W2V, SET 3

In [383]:

```
'''
# 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')

# ============================
Output:

Loading Glove Model
1917495it [06:32, 4879.69it/s]
Done. 1917495  words loaded!

# ============================

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

for i in preproced_titles:
    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-p
ickle-to-save-and-load-variables-in-python/

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


'''
```

Out[383]:

```
'\n# Reading glove vectors in python: https://stackoverflow.com/a/3823034
9/4084039\ndef loadGloveModel(gloveFile):\n    print ("Loading Glove Mode
l")\n    f = open(gloveFile,\'r\', encoding="utf8")\n    model = {}\n    f
or line in tqdm(f):\n        splitLine = line.split()\n        word = spli
tLine[0]\n        embedding = np.array([float(val) for val in splitLine
[1:]])\n        model[word] = embedding\n    print ("Done.",len(model)," w
ords loaded!")\n    return model\nmodel = loadGloveModel(\'glove.42B.300d.
txt\')\n\n# ============================\nOutput:\n    \nLoading Glove Mod
el\n1917495it [06:32, 4879.69it/s]\nDone. 1917495  words loaded!\n\n# ====
========================\n\nwords = []\nfor i in preproced_texts:\n    wor
ds.extend(i.split(\' \'))\n\nfor i in preproced_titles:\n    words.extend
(i.split(\' \'))\nprint("all the words in the coupus", len(words))\nwords
= set(words)\nprint("the unique words in the coupus", len(words))\n\ninter
_words = set(model.keys()).intersection(words)\nprint("The number of words
that are present in both glove vectors and our coupus",       len(inter_wo
rds),"(",np.round(len(inter_words)/len(words)*100,3),"%)")\n\nwords_courpu
s = {}\nwords_glove = set(model.keys())\nfor i in words:\n    if i in word
s_glove:\n        words_courpus[i] = model[i]\nprint("word 2 vec length",
len(words_courpus))\n\n\n# stronging variables into pickle files python: h
ttp://www.jessicayung.com/how-to-use-pickle-to-save-and-load-variables-in-
python/\n\nimport pickle\nwith open(\'glove_vectors\', \'wb\') as f:\n
pickle.dump(words_courpus, f)\n\n\n'
```

In [384]:

```
# stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-p
ickle-to-save-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 [385]:

```
# Using Pretrained Models: AVG W2V on `essay`
#_______________________________________________

# -----average Word2Vec on train
# compute average word2vec for each review.
avg_w2v_vectors_essays_train = []; # the avg-w2v for each sentence/review is stored in
 this list
for sentence in tqdm(x_train['preprocessed_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_essays_train.append(vector)

print(len(avg_w2v_vectors_essays_train))
print(len(avg_w2v_vectors_essays_train[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████
█████| 76473/76473 [00:22<00:00, 3348.67it/s]

76473
300
```

In [386]:

```
'''# -----average Word2Vec on CV
# compute average word2vec for each review.
avg_w2v_vectors_essays_cv = []; # the avg-w2v for each sentence/review is stored in thi
s list
for sentence in tqdm(x_cv['preprocessed_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_essays_cv.append(vector)

print(len(avg_w2v_vectors_essays_cv))
print(len(avg_w2v_vectors_essays_cv[0]))'''
```

Out[386]:

```
"# -----average Word2Vec on CV\n# compute average word2vec for each revie
w.\navg_w2v_vectors_essays_cv = []; # the avg-w2v for each sentence/review
is stored in this list\nfor sentence in tqdm(x_cv['preprocessed_essays']):
# for each review/sentence\n    vector = np.zeros(300) # as word vectors a
re of zero length\n    cnt_words =0; # num of words with a valid vector in
the sentence/review\n    for word in sentence.split(): # for each word in
a review/sentence\n        if word in glove_words:\n            vector +=
model[word]\n            cnt_words += 1\n    if cnt_words != 0:\n        v
ector /= cnt_words\n    avg_w2v_vectors_essays_cv.append(vector)\n\nprint
(len(avg_w2v_vectors_essays_cv))\nprint(len(avg_w2v_vectors_essays_cv
[0]))"
```

In [387]:

```
# -----average Word2Vec on test
# compute average word2vec for each review.
avg_w2v_vectors_essays_test = []; # the avg-w2v for each sentence/review is stored in t
his list
for sentence in tqdm(x_test['preprocessed_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_essays_test.append(vector)

print(len(avg_w2v_vectors_essays_test))
print(len(avg_w2v_vectors_essays_test[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████
█████| 32775/32775 [00:09<00:00, 3306.59it/s]

32775
300
```

In [388]:

```
# Using Pretrained Models: AVG W2V on `project_title`
#_____________________________________________________

# ------average Word2Vec on train
# compute average word2vec for each review.
avg_w2v_vectors_project_title_train = []; # the avg-w2v for each sentence/review is sto
red in this list
for sentence in tqdm(x_train['preprocessed_project_title']): # 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_project_title_train.append(vector)

print(len(avg_w2v_vectors_project_title_train))
print(len(avg_w2v_vectors_project_title_train[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████
████| 76473/76473 [00:01<00:00, 60553.15it/s]

76473
300
```

In [389]:

```
'''# ------average Word2Vec on cv
# compute average word2vec for each review.
avg_w2v_vectors_project_title_cv = []; # the avg-w2v for each sentence/review is stored
in this list
for sentence in tqdm(x_cv['preprocessed_project_title']): # 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_project_title_cv.append(vector)

print(len(avg_w2v_vectors_project_title_cv))
print(len(avg_w2v_vectors_project_title_cv[0]))'''
```

Out[389]:

```
"# ------average Word2Vec on cv\n# compute average word2vec for each revie
w.\navg_w2v_vectors_project_title_cv = []; # the avg-w2v for each sentenc
e/review is stored in this list\nfor sentence in tqdm(x_cv['preprocessed_p
roject_title']): # for each review/sentence\n    vector = np.zeros(300) #
as word vectors are of zero length\n    cnt_words =0; # num of words with
a valid vector in the sentence/review\n    for word in sentence.split(): #
for each word in a review/sentence\n        if word in glove_words:\n
vector += model[word]\n            cnt_words += 1\n    if cnt_words !=
0:\n        vector /= cnt_words\n    avg_w2v_vectors_project_title_cv.appe
nd(vector)\n\nprint(len(avg_w2v_vectors_project_title_cv))\nprint(len(avg_
w2v_vectors_project_title_cv[0]))"
```

In [390]:

```
# ------average Word2Vec on test
# compute average word2vec for each review.
avg_w2v_vectors_project_title_test = []; # the avg-w2v for each sentence/review is stor
ed in this list
for sentence in tqdm(x_test['preprocessed_project_title']): # 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_project_title_test.append(vector)

print(len(avg_w2v_vectors_project_title_test))
print(len(avg_w2v_vectors_project_title_test[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████
████| 32775/32775 [00:00<00:00, 57249.22it/s]

32775
300
```

### merge all sparse data, SET 3

In [391]:

```
# 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
:)
x_train_AVGW2V = hstack((x_train_ohe, avg_w2v_vectors_essays_train, avg_w2v_vectors_pro
ject_title_train)).tocsr()
#x_cv_AVGW2V = hstack((x_cv_ohe, avg_w2v_vectors_essays_cv, avg_w2v_vectors_project_tit
le_cv)).tocsr()
x_test_AVGW2V = hstack((x_test_ohe, avg_w2v_vectors_essays_test, avg_w2v_vectors_projec
t_title_test)).tocsr()

print(x_train_AVGW2V.shape)
#print(x_cv_AVGW2V.shape)
print(x_test_AVGW2V.shape)
```

```
(76473, 700)
(32775, 700)
```

### GridsearchCV, SET 3

In [392]:

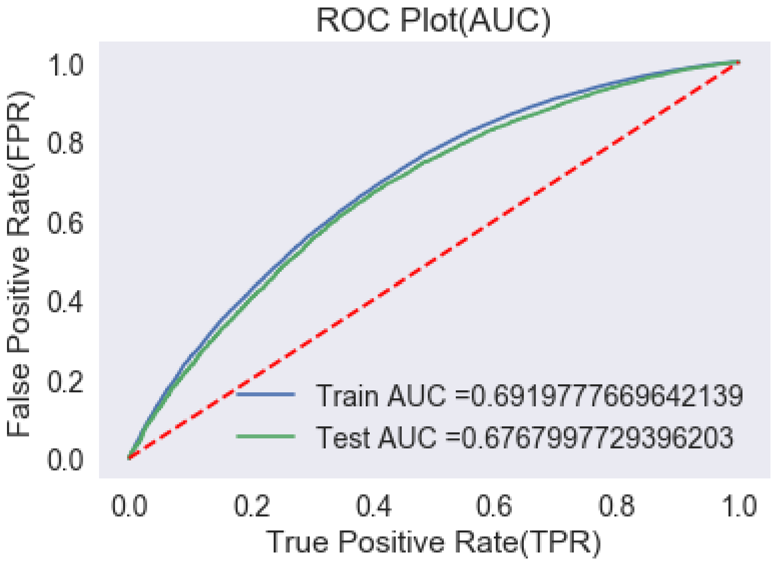
```
%time
#https://chrisalbon.com/machine_learning/model_selection/hyperparameter_tuning_using_gr
id_search/
#https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.h
tml
#https://stackoverflow.com/questions/48220125/how-to-generate-roc-curve-with-cross-vali
dation-using-sgd-classifier-loss-hinge
#https://www.kaggle.com/udaysa/svm-with-scikit-learn-svm-with-parameter-tuning
from sklearn.model_selection import GridSearchCV,TimeSeriesSplit
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import roc_curve, auc

print("***************Grid search cv perform on train dataset*****************")

svm_linear = SGDClassifier(loss='hinge')

parameters = {'alpha':[1000,500,100,50,10,1,0.1,0.05,0.01,0.005,0.001,0.0005,0.0001],
             'penalty':['l1','l2']}
ts_cv = TimeSeriesSplit(n_splits=10) #For time based splitting
clf = GridSearchCV(svm_linear, parameters, cv= ts_cv, scoring='roc_auc')

clf.fit(x_train_AVGW2V, y_train)

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

penalty_hyperparameter_AVGW2V=clf.best_params_['penalty']
alpha_hyperparameter_AVGW2V=clf.best_params_['alpha']
print("Best HyperParameter: ",clf.best_params_)
print("Best Accuracy: %.2f%%"%(clf.best_score_*100))
```

```
Wall time: 0 ns
***************Grid search cv perform on train dataset*****************
Best HyperParameter:  {'alpha': 0.0001, 'penalty': 'l1'}
Best Accuracy: 66.95%
```

### Implementing on test dataset, and ROC Plot SET 3

In [393]:

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


svm_linear = SGDClassifier(loss='hinge', penalty=penalty_hyperparameter_AVGW2V, alpha=alpha_hyperparameter_AVGW2V)

svm_linear.fit(x_train_AVGW2V, y_train)

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

y_train_pred = svm_linear.decision_function(x_train_AVGW2V)
y_test_pred = svm_linear.decision_function(x_test_AVGW2V)

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)

AVGW2V_auc_train = auc(train_fpr, train_tpr)
AVGW2V_auc_test = auc(test_fpr, test_tpr)

plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.plot([0,1],[0,1],'r--')
plt.legend()
plt.xlabel("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("ROC Plot(AUC)")
plt.grid()
plt.show()
```



**Confusion matrix, SET 3**

In [394]:

```
#https://seaborn.pydata.org/generated/seaborn.heatmap.html
#----------------------------------------------------

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

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

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

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

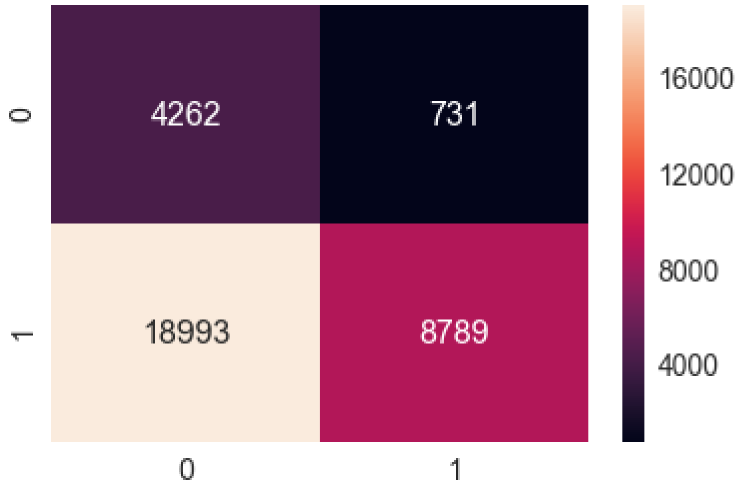
In [395]:

```
print("="*100)
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_tpr
)))
conf_matr_df_test_1 = pd.DataFrame(confusion_matrix(y_test, predict(y_test_pred, tr_thr
esholds, test_fpr, test_tpr)), range(2),range(2))

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

```
==========================================================================
==========================
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.40210199685938547 for threshold 1.544
[[ 4262   731]
 [18993  8789]]
the maximum value of tpr*(1-fpr) 0.40210199685938547 for threshold 1.544
```

Out[395]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1f82c736198>
```



## 2.3.4 Apply Support Vector Machines with TFIDF AVG W2V, SET 4

### Vectorize using TFIDF W2V, SET 4

In [396]:

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

In [397]:

```
#Using Pretrained Models: TFIDFW weighted W2V on `essay
#___________________________________

# average Word2Vec---train
# compute average word2vec for each review.
tfidf_w2v_vectors_essays_train = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(x_train['preprocessed_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_essay):
            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_essay[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_essays_train.append(vector)

print(len(tfidf_w2v_vectors_essays_train))
print(len(tfidf_w2v_vectors_essays_train[0]))
```

```
100%|██████████████████████████████████████████████████████████████████
█████| 76473/76473 [02:14<00:00, 569.97it/s]

76473
300
```

In [398]:

```
'''# average Word2Vec---cv
# compute average word2vec for each review.
tfidf_w2v_vectors_essays_cv = []; # the avg-w2v for each sentence/review is stored in t
his list
for sentence in tqdm(x_cv['preprocessed_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((sen
tence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # ge
tting the tfidf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_essays_cv.append(vector)

print(len(tfidf_w2v_vectors_essays_cv))
print(len(tfidf_w2v_vectors_essays_cv[0]))'''
```

Out[398]:

```
"# average Word2Vec---cv\n# compute average word2vec for each review.\ntfi
df_w2v_vectors_essays_cv = []; # the avg-w2v for each sentence/review is s
tored in this list\nfor sentence in tqdm(x_cv['preprocessed_essays']): # f
or each review/sentence\n    vector = np.zeros(300) # as word vectors are 
of zero length\n    tf_idf_weight =0; # num of words with a valid vector i
n the sentence/review\n    for word in sentence.split(): # for each word i
n a review/sentence\n        if (word in glove_words) and (word in tfidf_w
ords):\n            vec = model[word] # getting the vector for each word\n
# here we are multiplying idf value(dictionary[word]) and the tf value((se
ntence.count(word)/len(sentence.split())))\n            tf_idf = dictionar
y[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf v
alue for each word\n            vector += (vec * tf_idf) # calculating tfi
df weighted w2v\n            tf_idf_weight += tf_idf\n    if tf_idf_weight
!= 0:\n        vector /= tf_idf_weight\n    tfidf_w2v_vectors_essays_cv.ap
pend(vector)\n\nprint(len(tfidf_w2v_vectors_essays_cv))\nprint(len(tfidf_w
2v_vectors_essays_cv[0]))"
```

In [399]:

```
# average Word2Vec---test
# compute average word2vec for each review.
tfidf_w2v_vectors_essays_test = []; # the avg-w2v for each sentence/review is stored in
this list
for sentence in tqdm(x_test['preprocessed_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_essay):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sen
tence.count(word)/len(sentence.split())))
            tf_idf = dictionary_essay[word]*(sentence.count(word)/len(sentence.split
())) # getting the tfidf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_essays_test.append(vector)

print(len(tfidf_w2v_vectors_essays_test))
print(len(tfidf_w2v_vectors_essays_test[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████
██████| 32775/32775 [00:59<00:00, 543.60it/s]

32775
300
```

In [400]:

```
tfidf_model = TfidfVectorizer()
tfidf_model.fit(x_train['preprocessed_project_title'])
# we are converting a dictionary with word as a key, and the idf as a value
dictionary_title = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words_title = set(tfidf_model.get_feature_names())
```

In [401]:

```
#Using Pretrained Models: TFIDFW weighted W2V on "preprocessed_project_title"
#_________________________________________________
# average Word2Vec--train
# compute average word2vec for each review.
tfidf_w2v_vectors_project_title_train = []; # the avg-w2v for each sentence/review is s
tored in this list
for sentence in tqdm(x_train['preprocessed_project_title']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words_title):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sen
tence.count(word)/len(sentence.split())))
            tf_idf = dictionary_title[word]*(sentence.count(word)/len(sentence.split
())) # getting the tfidf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_project_title_train.append(vector)

print(len(tfidf_w2v_vectors_project_title_train))
print(len(tfidf_w2v_vectors_project_title_train[0]))
```

```
100%|█████████████████████████████████████████████████████████████████████
████| 76473/76473 [00:02<00:00, 33362.48it/s]

76473
300
```

In [402]:

```
'''# average Word2Vec--cv
# compute average word2vec for each review.
tfidf_w2v_vectors_project_title_cv = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(x_cv['preprocessed_project_title']): # 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 tfidf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_project_title_cv.append(vector)

print(len(tfidf_w2v_vectors_project_title_cv))
print(len(tfidf_w2v_vectors_project_title_cv[0]))'''
```

Out[402]:

```
"# average Word2Vec--cv\n# compute average word2vec for each review.\ntfid
f_w2v_vectors_project_title_cv = []; # the avg-w2v for each sentence/revie
w is stored in this list\nfor sentence in tqdm(x_cv['preprocessed_project_
title']): # for each review/sentence\n    vector = np.zeros(300) # as word
vectors are of zero length\n    tf_idf_weight =0; # num of words with a va
lid vector in the sentence/review\n    for word in sentence.split(): # for
each word in a review/sentence\n        if (word in glove_words) and (word
in tfidf_words):\n            vec = model[word] # getting the vector for e
ach word\n            # here we are multiplying idf value(dictionary[wor
d]) and the tf value((sentence.count(word)/len(sentence.split())))\n
tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # g
etting the tfidf value for each word\n            vector += (vec * tf_idf)
# calculating tfidf weighted w2v\n            tf_idf_weight += tf_idf\n
if tf_idf_weight != 0:\n        vector /= tf_idf_weight\n    tfidf_w2v_vec
tors_project_title_cv.append(vector)\n\nprint(len(tfidf_w2v_vectors_projec
t_title_cv))\nprint(len(tfidf_w2v_vectors_project_title_cv[0]))"
```

In [403]:

```
# average Word2Vec--test
# compute average word2vec for each review.
tfidf_w2v_vectors_project_title_test = []; # the avg-w2v for each sentence/review is st
ored in this list
for sentence in tqdm(x_test['preprocessed_project_title']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words_title):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sen
tence.count(word)/len(sentence.split())))
            tf_idf = dictionary_title[word]*(sentence.count(word)/len(sentence.split
())) # getting the tfidf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_project_title_test.append(vector)

print(len(tfidf_w2v_vectors_project_title_test))
print(len(tfidf_w2v_vectors_project_title_test[0]))
```

```
100%|█████████████████████████████████████████████████████████████████████
████| 32775/32775 [00:01<00:00, 29306.67it/s]

32775
300
```

**merge all aparse data, SET 4**

In [404]:

```
# 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
:)
x_train_TFIDFW2V = hstack((x_train_ohe, tfidf_w2v_vectors_essays_train, tfidf_w2v_vecto
rs_project_title_train)).tocsr()
#x_cv_TFIDFW2V = hstack((x_cv_ohe, tfidf_w2v_vectors_essays_cv, tfidf_w2v_vectors_proje
ct_title_cv)).tocsr()
x_test_TFIDFW2V = hstack((x_test_ohe, tfidf_w2v_vectors_essays_test, tfidf_w2v_vectors_
project_title_test)).tocsr()

print(x_train_TFIDFW2V.shape)
#print(x_cv_TFIDFW2V.shape)
print(x_test_TFIDFW2V.shape)
```

```
(76473, 700)
(32775, 700)
```

**GridsearchCV, SET 4**

In [405]:

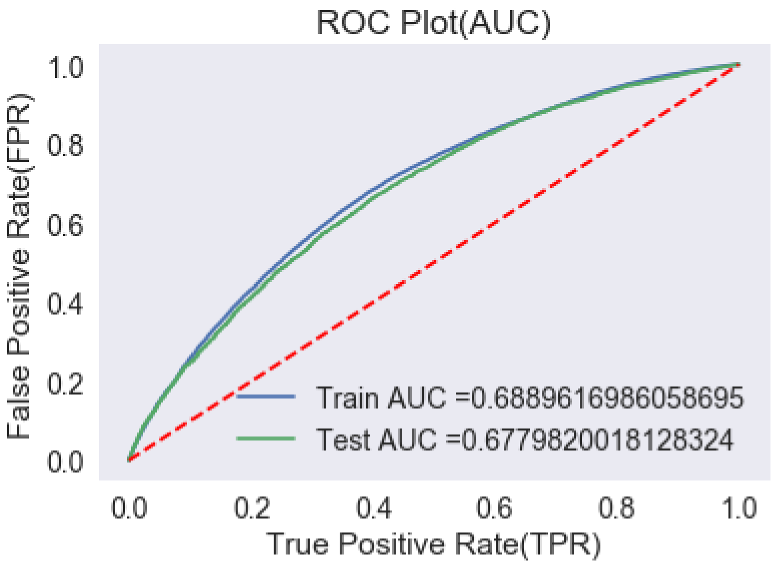
```
%time
#https://chrisalbon.com/machine_learning/model_selection/hyperparameter_tuning_using_gr
id_search/
#https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.h
tml
#https://stackoverflow.com/questions/48220125/how-to-generate-roc-curve-with-cross-vali
dation-using-sgd-classifier-loss-hinge
#https://www.kaggle.com/udaysa/svm-with-scikit-learn-svm-with-parameter-tuning
from sklearn.model_selection import GridSearchCV,TimeSeriesSplit
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import roc_curve, auc

print("***************Grid search cv perform on train dataset*****************")

svm_linear = SGDClassifier(loss='hinge')

parameters = {'alpha':[1000,500,100,50,10,1,0.1,0.05,0.01,0.005,0.001,0.0005,0.0001],
             'penalty':['l1','l2']}
ts_cv = TimeSeriesSplit(n_splits=10) #For time based splitting
clf = GridSearchCV(svm_linear, parameters, cv= ts_cv, scoring='roc_auc')

clf.fit(x_train_TFIDFW2V, y_train)

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

penalty_hyperparameter_TFIDFW2V=clf.best_params_['penalty']
alpha_hyperparameter_TFIDFW2V=clf.best_params_['alpha']
print("Best HyperParameter: ",clf.best_params_)
print("Best Accuracy: %.2f%%"%(clf.best_score_*100))
```

```
Wall time: 0 ns
***************Grid search cv perform on train dataset*****************
Best HyperParameter:  {'alpha': 0.0001, 'penalty': 'l1'}
Best Accuracy: 65.59%
```

### AUC plot for test and train dataset, SET 4

In [406]:

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


svm_linear = SGDClassifier(loss='hinge', penalty=penalty_hyperparameter_TFIDFW2V, alpha
=alpha_hyperparameter_TFIDFW2V)

svm_linear.fit(x_train_TFIDFW2V, y_train)

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

y_train_pred = svm_linear.decision_function(x_train_TFIDFW2V)
y_test_pred = svm_linear.decision_function(x_test_TFIDFW2V)

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)

TFIDFW2V_auc_train = auc(train_fpr, train_tpr)
TFIDFW2V_auc_test = auc(test_fpr, test_tpr)

plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.plot([0,1],[0,1],'r--')
plt.legend()
plt.xlabel("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("ROC Plot(AUC)")
plt.grid()
plt.show()
```



**Confusion matrix, SET 4**

In [407]:

```
#https://seaborn.pydata.org/generated/seaborn.heatmap.html
#-------------------------------------------------

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

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

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

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

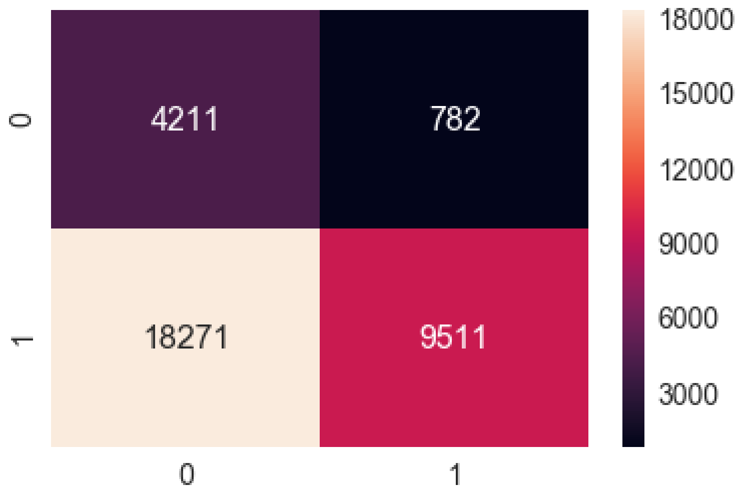
In [408]:

```
print("="*100)
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_tpr
)))
conf_matr_df_test_1 = pd.DataFrame(confusion_matrix(y_test, predict(y_test_pred, tr_thr
esholds, test_fpr, test_tpr)), range(2),range(2))

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

```
==========================================================================
==========================
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.3984699593036183 for threshold 1.654
[[ 4211   782]
 [18271  9511]]
the maximum value of tpr*(1-fpr) 0.3984699593036183 for threshold 1.654
```

Out[408]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1f83828ee80>
```



## 2.5 Apply Support Vector Machines with added Features `Set 5` SET 5

In [0]:

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

#### vectorize the essay SET 5

In [280]:

```
#convert the essay text to vector
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer_tfidf = TfidfVectorizer()
vectorizer_tfidf.fit(x_train['preprocessed_essays'].values)# fit has to apply only on train data
z_tfidf1=vectorizer_tfidf.fit(x_train['preprocessed_essays'].values)# fit has to apply only on train data

# we use fitted CountVectorizer to convert the text to vector
x_train_tfidf_essays = vectorizer_tfidf.transform(x_train['preprocessed_essays'].values)
#x_cv_tfidf_essays = vectorizer_tfidf.transform(x_cv['preprocessed_essays'].values)
x_test_tfidf_essays = vectorizer_tfidf.transform(x_test['preprocessed_essays'].values)

print("Shape of matrix after one hot encodig ",x_train_tfidf_essays.shape, y_train.shape)
#print("Shape of matrix after one hot encodig ",x_cv_tfidf_essays.shape)
print("Shape of matrix after one hot encodig ",x_test_tfidf_essays.shape)
```

```
Shape of matrix after one hot encodig  (35000, 35824) (35000,)
Shape of matrix after one hot encodig  (15000, 35824)
```

# apply "elbo method" on vectorize tfidf dataset

In [222]:

```
from sklearn.decomposition import TruncatedSVD

index = [5,50,500,1000,3000,5000,7500,10000]
variance_sum = []

for i in tqdm(index):
    svd = TruncatedSVD(n_components= i, n_iter=5, random_state=42)
    svd.fit(x_train_tfidf_essays)
    variance_sum.append(svd.explained_variance_ratio_.sum())
```

```
100%|█████████████████████████████████████████████████████████████████████
██████████████| 8/8 [37:49<00:00, 536.84s/it]
```

In [223]:

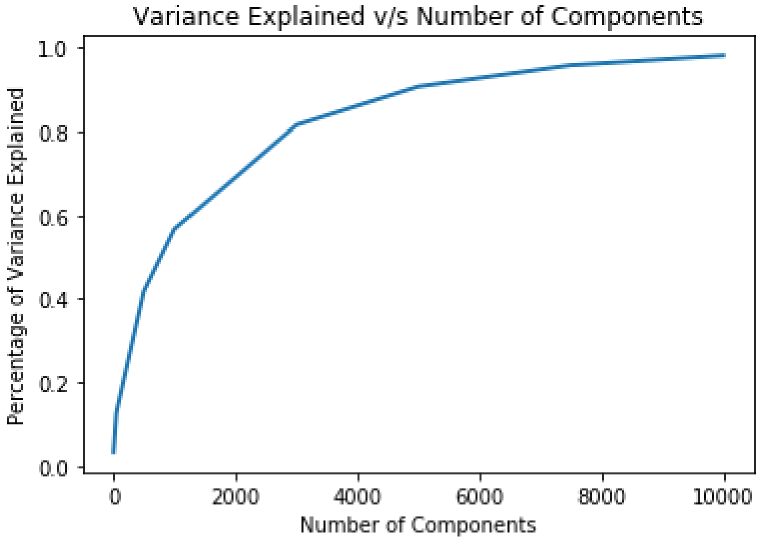
```
variance_sum
```

Out[223]:

```
[0.03262932268853709,
 0.12660119052214197,
 0.41746368678434914,
 0.5666683158823187,
 0.8153762053316091,
 0.9065883714941158,
 0.9576889930169241,
 0.9811006480115353]
```

In [224]:

```
plt.xlabel("Number of Components")
plt.ylabel("Percentage of Variance Explained")
plt.title("Variance Explained v/s Number of Components")
plt.plot(index,variance_sum,lw=2)
plt.show()
```



Here 5000 component explane almost 90% of variable so let take only 5000 component

In [281]:

```
#apply on train dataset
svd = TruncatedSVD(n_components= 5000, n_iter=5, random_state=42)
svd.fit(x_train_tfidf_essays)
svd_train = svd.transform(x_train_tfidf_essays)
```

In [282]:

```
print("Shape of matrix after Decomposition ",svd_train.shape)
```

```
Shape of matrix after Decomposition  (35000, 5000)
```

In [283]:

```
#apply on test dataset
svd_test = svd.transform(x_test_tfidf_essays)
print("Shape of matrix after Decomposition ",svd_train.shape)
```

```
Shape of matrix after Decomposition  (35000, 5000)
```

#### total number of word in each title and essay, SET 5

Counting number of word

In [284]:

```
#https://stackoverflow.com/questions/49984905/count-number-of-words-per-row
x_train['train_totalwords_title'] = x_train['preprocessed_project_title'].str.split().str.len()
x_test['test_totalwords_title'] = x_test['preprocessed_project_title'].str.split().str.len()

x_train['train_totalwords_essay'] = x_train['preprocessed_essays'].str.split().str.len()
x_test['test_totalwords_essay'] = x_test['preprocessed_essays'].str.split().str.len()
```

#### encoding categorical features, SET 5

In [285]:

```
#one hot encoding for clean_categories
#_________________________
# we use count vectorizer to convert the values into one
from sklearn.feature_extraction.text import CountVectorizer

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

categories_one_hot_train = vectorizer_proj.transform(x_train['clean_categories'].values)
categories_one_hot_test = vectorizer_proj.transform(x_test['clean_categories'].values)
#categories_one_hot_cv = vectorizer_proj.transform(X_cv['clean_categories'].values)

print(vectorizer_proj.get_feature_names())

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

```
['Warmth', 'Care_Hunger', 'History_Civics', 'Health_Sports', 'Music_Arts',
'AppliedLearning', 'SpecialNeeds', 'Math_Science', 'Literacy_Language']
Shape of matrix of Train data after one hot encoding  (35000, 9)
Shape of matrix of Test data after one hot encoding  (15000, 9)
```

In [286]:

```
#one hot encoding for clean_subcategories
#__________________________
# we use count vectorizer to convert the values into one
vectorizer_sub_proj = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()), lowe
rcase=False, binary=True)
vectorizer_sub_proj.fit(x_train['clean_subcategories'].values)

sub_categories_one_hot_train = vectorizer_sub_proj.transform(x_train['clean_subcategori
es'].values)
sub_categories_one_hot_test = vectorizer_sub_proj.transform(x_test['clean_subcategorie
s'].values)
#sub_categories_one_hot_cv = vectorizer_sub_proj.transform(X_cv['clean_subcategories'].
values)


print(vectorizer_sub_proj.get_feature_names())

print("Shape of matrix of Train data after one hot encoding ",sub_categories_one_hot_tr
ain.shape)
print("Shape of matrix of Test data after one hot encoding ",sub_categories_one_hot_tes
t.shape)
#print("Shape of matrix of Cross Validation data after one hot encoding ",sub_categorie
s_one_hot_cv.shape)
```

```
['Economics', 'NutritionEducation', 'CommunityService', 'Civics_Governmen
t', 'ForeignLanguages', 'Extracurricular', 'FinancialLiteracy', 'ParentInv
olvement', 'SocialSciences', 'CharacterEducation', 'Gym_Fitness', 'Perform
ingArts', 'TeamSports', 'Other', 'College_CareerPrep', 'Warmth', 'Care_Hun
ger', 'Music', 'History_Geography', 'Health_LifeScience', 'EarlyDevelopmen
t', 'Health_Wellness', 'ESL', 'EnvironmentalScience', 'VisualArts', 'Appli
edSciences', 'SpecialNeeds', 'Literature_Writing', 'Mathematics', 'Literac
y']
Shape of matrix of Train data after one hot encoding  (35000, 30)
Shape of matrix of Test data after one hot encoding  (15000, 30)
```

In [287]:

```
#one hot encoding for school_state

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

school_state_cat_dict = dict(my_counter)
sorted_school_state_cat_dict = dict(sorted(school_state_cat_dict.items(), key=lambda kv
: kv[1]))
#_________________________
# we use count vectorizer to convert the values into one
vectorizer_states = CountVectorizer(vocabulary=list(sorted_school_state_cat_dict.keys
()), lowercase=False, binary=True)
vectorizer_states.fit(x_train['school_state'].values)

school_state_categories_one_hot_train = vectorizer_states.transform(x_train['school_sta
te'].values)
school_state_categories_one_hot_test = vectorizer_states.transform(x_test['school_stat
e'].values)
#school_state_categories_one_hot_cv = vectorizer_states.transform(X_cv['school_state'].
values)

print(vectorizer_states.get_feature_names())

print("Shape of matrix of Train data after one hot encoding ",school_state_categories_o
ne_hot_train.shape)
print("Shape of matrix of Test data after one hot encoding ",school_state_categories_on
e_hot_test.shape)
#print("Shape of matrix of Cross Validation data after one hot encoding ",school_state_
categories_one_hot_cv.shape)
```

```
['VT', 'ND', 'WY', 'MT', 'DE', 'NE', 'AK', 'SD', 'RI', 'NH', 'WV', 'DC',
'ME', 'IA', 'NM', 'HI', 'KS', 'ID', 'AR', 'OR', 'CO', 'UT', 'KY', 'MS', 'M
N', 'NV', 'TN', 'MD', 'WI', 'AL', 'CT', 'OK', 'LA', 'VA', 'AZ', 'WA', 'O
H', 'MA', 'MO', 'IN', 'NJ', 'PA', 'MI', 'GA', 'SC', 'IL', 'NC', 'FL', 'T
X', 'NY', 'CA']
Shape of matrix of Train data after one hot encoding  (35000, 51)
Shape of matrix of Test data after one hot encoding  (15000, 51)
```

In [288]:

```
#one hot encoding for project_grade_category

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

project_grade_cat_dict = dict(my_counter)
sorted_project_grade_cat_dict = dict(sorted(project_grade_cat_dict.items(), key=lambda
kv: kv[1]))
#_________________________
# we use count vectorizer to convert the values into one
vectorizer_grade = CountVectorizer(vocabulary=list(sorted_project_grade_cat_dict.keys
()), lowercase=False, binary=True)
vectorizer_grade.fit(x_train['project_grade_category'].values)

project_grade_categories_one_hot_train = vectorizer_grade.transform(x_train['project_gr
ade_category'].values)
project_grade_categories_one_hot_test = vectorizer_grade.transform(x_test['project_grad
e_category'].values)
#project_grade_categories_one_hot_cv = vectorizer_grade.transform(X_cv['project_grade_c
ategory'].values)

print(vectorizer_grade.get_feature_names())

print("Shape of matrix of Train data after one hot encoding ",project_grade_categories_
one_hot_train.shape)
print("Shape of matrix of Test data after one hot encoding ",project_grade_categories_o
ne_hot_test.shape)
#print("Shape of matrix of Cross Validation data after one hot encoding ",project_grade
_categories_one_hot_cv.shape)
```

```
['Grades_9-12', 'Grades_6-8', 'Grades_3-5', 'Grades_PreK-2']
Shape of matrix of Train data after one hot encoding  (35000, 4)
Shape of matrix of Test data after one hot encoding  (15000, 4)
```

In [289]:

```
#replace NAN to space https ://stackoverflow.com/questions/49259305/raise-valueerrornp-
nan-is-an-invalid-document-expected-byte-or?rq=1
project_data['teacher_prefix'] = project_data['teacher_prefix'].fillna(' ')
```

In [290]:

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

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

#one hot encoding for teacher_prefix
#_________________________
# we use count vectorizer to convert the values into one
vectorizer_teacher = CountVectorizer(vocabulary=list(sorted_teacher_prefix_cat_dict.key
s()), lowercase=False, binary=True)
vectorizer_teacher.fit(x_train['teacher_prefix'].values.astype("U"))

teacher_prefix_categories_one_hot_train = vectorizer_teacher.transform(x_train['teacher
_prefix'].values.astype("U"))
teacher_prefix_categories_one_hot_test = vectorizer_teacher.transform(x_test['teacher_p
refix'].values.astype("U"))
#teacher_prefix_categories_one_hot_cv = vectorizer_teacher.transform(X_cv['teacher_pref
ix'].values.astype("U"))

print(vectorizer_teacher.get_feature_names())

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

```
['Dr.', 'Teacher', 'Mr.', 'Ms.', 'Mrs.']
Shape of matrix after one hot encoding  (35000, 5)
Shape of matrix after one hot encoding  (15000, 5)
```

### encoding numerical features</font>

In [291]:

```
project_data.columns
```

Out[291]:

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'Date', 'project_essay_1', 'project_essay_2', 'project_essay_3',
       'project_essay_4', 'project_resource_summary',
       'teacher_number_of_previously_posted_projects', 'project_is_approve
d',
       'price', 'quantity', 'project_grade_category', 'clean_categories',
       'clean_subcategories', 'preprocessed_essays',
       'preprocessed_project_title', 'sentiment_pos_essay',
       'sentiment_neg_essay', 'sentiment_neu_essay',
       'sentiment_compound_essay'],
      dtype='object')
```

In [292]:

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

Out[292]:

| | Unnamed: 0 | id | teacher_id | teacher_prefix | scho |
|---|---|---|---|---|---|
| **59248** | 9545 | p213897 | b2d1972f8b6d454389ca645602e5ebf5 | Mrs. | LA |
| **59249** | 8558 | p239505 | a5ac4dc34ee41521a1263b030f1bb7fd | Ms. | TX |

2 rows × 24 columns

In [293]:

```
#price standardization
#---------------------------------------
# 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

normalizer = Normalizer()

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

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

price_train = normalizer.transform(x_train['price'].values.reshape(-1,1))

price_test = normalizer.transform(x_test['price'].values.reshape(-1,1))

print("After vectorizations")
print(price_train.shape, y_train.shape)

print(price_test.shape, y_test.shape)
```

```
After vectorizations
(35000, 1) (35000,)
(15000, 1) (15000,)
```

In [294]:

```
#quantity standardization
#-------------------------------------
# check this one: https://www.youtube.com/watch?v=0HOqOcln3Z4&t=530s
# standardization sklearn: https://scikit-learn.org/stable/modules/generated/sklearn.pr
eprocessing.StandardScaler.html
normalizer = Normalizer()

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

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

quantity_train = normalizer.transform(x_train['quantity'].values.reshape(-1,1))

quantity_test = normalizer.transform(x_test['quantity'].values.reshape(-1,1))

print("After vectorizations")
print(quantity_train.shape, y_train.shape)

print(quantity_test.shape, y_test.shape)
```

```
After vectorizations
(35000, 1) (35000,)
(15000, 1) (15000,)
```

In [295]:

```
#Normalize the teacher_number_of_previously_posted_projects
#_____________________________________________
#https://www.w3cschool.cn/doc_scikit_learn/scikit_learn-modules-generated-sklearn-prepr
ocessing-normalize.html
#https://stackoverflow.com/questions/53723928/attributeerror-series-object-has-no-attri
bute-reshape

normalizer = Normalizer()


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

prev_projects_train = normalizer.transform(x_train['teacher_number_of_previously_posted
_projects'].values.reshape(-1,1))

prev_projects_test = normalizer.transform(x_test['teacher_number_of_previously_posted_p
rojects'].values.reshape(-1,1))

print("After vectorizations")
print(prev_projects_train.shape, y_train.shape)

print(prev_projects_test.shape, y_test.shape)
```

```
After vectorizations
(35000, 1) (35000,)
(15000, 1) (15000,)
```

In [296]:

```
#Normalize the totalwords_title
#_______________________________________________
#https://www.w3cschool.cn/doc_scikit_learn/scikit_learn-modules-generated-sklearn-prepr
ocessing-normalize.html
#https://stackoverflow.com/questions/53723928/attributeerror-series-object-has-no-attri
bute-reshape
normalizer = Normalizer()

normalizer.fit(x_train['train_totalwords_title'].values.reshape(-1,1))

title_word_count_train = normalizer.transform(x_train['train_totalwords_title'].values.
reshape(-1,1))

title_word_count_test = normalizer.transform(x_test['test_totalwords_title'].values.res
hape(-1,1))

print("After vectorizations")
print(title_word_count_train.shape, y_train.shape)

print(title_word_count_test.shape, y_test.shape)
```

```
After vectorizations
(35000, 1) (35000,)
(15000, 1) (15000,)
```

In [297]:

```
#Normalize the totalwords_essay
#_______________________________________________
#https://www.w3cschool.cn/doc_scikit_learn/scikit_learn-modules-generated-sklearn-prepr
ocessing-normalize.html
#https://stackoverflow.com/questions/53723928/attributeerror-series-object-has-no-attri
bute-reshape

normalizer = Normalizer()

normalizer.fit(x_train['train_totalwords_essay'].values.reshape(-1,1))

essay_word_count_train = normalizer.transform(x_train['train_totalwords_essay'].values.
reshape(-1,1))

essay_word_count_test = normalizer.transform(x_test['test_totalwords_essay'].values.res
hape(-1,1))

print("After vectorizations")
print(essay_word_count_train.shape, y_train.shape)

print(essay_word_count_test.shape, y_test.shape)
```

```
After vectorizations
(35000, 1) (35000,)
(15000, 1) (15000,)
```

# Sentiments

In [298]:

```
normalizer = Normalizer()

normalizer.fit(x_train['sentiment_pos_essay'].values.reshape(-1,1))

essay_sent_pos_train = normalizer.transform(x_train['sentiment_pos_essay'].values.resha
pe(-1,1))

essay_sent_pos_test = normalizer.transform(x_test['sentiment_pos_essay'].values.reshape
(-1,1))

print("After vectorizations")
print(essay_sent_pos_train.shape, y_train.shape)

print(essay_sent_pos_test.shape, y_test.shape)
```

```
After vectorizations
(35000, 1) (35000,)
(15000, 1) (15000,)
```

In [299]:

```
normalizer = Normalizer()

normalizer.fit(x_train['sentiment_neg_essay'].values.reshape(-1,1))

essay_sent_neg_train = normalizer.transform(x_train['sentiment_neg_essay'].values.resha
pe(-1,1))

essay_sent_neg_test = normalizer.transform(x_test['sentiment_neg_essay'].values.reshape
(-1,1))

print("After vectorizations")
print(essay_sent_neg_train.shape, y_train.shape)

print(essay_sent_neg_test.shape, y_test.shape)
```

```
After vectorizations
(35000, 1) (35000,)
(15000, 1) (15000,)
```

In [300]:

```
normalizer = Normalizer()

normalizer.fit(x_train['sentiment_neu_essay'].values.reshape(-1,1))

essay_sent_neu_train = normalizer.transform(x_train['sentiment_neu_essay'].values.reshape(-1,1))

essay_sent_neu_test = normalizer.transform(x_test['sentiment_neu_essay'].values.reshape(-1,1))

print("After vectorizations")
print(essay_sent_neu_train.shape, y_train.shape)

print(essay_sent_neu_test.shape, y_test.shape)
```

```
After vectorizations
(35000, 1) (35000,)
(15000, 1) (15000,)
```

In [301]:

```
normalizer = Normalizer()

normalizer.fit(x_train['sentiment_compound_essay'].values.reshape(-1,1))

essay_sent_comp_train = normalizer.transform(x_train['sentiment_compound_essay'].values.reshape(-1,1))

essay_sent_comp_test = normalizer.transform(x_test['sentiment_compound_essay'].values.reshape(-1,1))

print("After vectorizations")
print(essay_sent_comp_train.shape, y_train.shape)

print(essay_sent_comp_test.shape, y_test.shape)
```

```
After vectorizations
(35000, 1) (35000,)
(15000, 1) (15000,)
```

### merge numerical and categorical data and sentiment data</font>

In [302]:

```
# merge the 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
:)
x_train_svd = hstack((svd_train,categories_one_hot_train,sub_categories_one_hot_train,s
chool_state_categories_one_hot_train,project_grade_categories_one_hot_train,teacher_pre
fix_categories_one_hot_train,price_train,quantity_train,prev_projects_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()
x_test_svd = hstack((svd_test,categories_one_hot_test,sub_categories_one_hot_test,schoo
l_state_categories_one_hot_test,project_grade_categories_one_hot_test,teacher_prefix_ca
tegories_one_hot_test,price_test,quantity_test,prev_projects_test,title_word_count_test
,essay_word_count_test,essay_sent_pos_test,essay_sent_neg_test,essay_sent_neu_test,essa
y_sent_comp_test)).tocsr()

print(x_train_svd.shape)
print(x_test_svd.shape)
```

```
(35000, 5108)
(15000, 5108)
```

### GridSearchCV (K fold Cross Validation) using Penalty(regularization = l2) SET 5

In [306]:

```
#https://chrisalbon.com/machine_learning/model_selection/hyperparameter_tuning_using_gr
id_search/

#https://chrisalbon.com/machine_learning/model_selection/hyperparameter_tuning_using_gr
id_search/
#https://www.kaggle.com/udaysa/svm-with-scikit-learn-svm-with-parameter-tuning
%time
from sklearn.model_selection import GridSearchCV,TimeSeriesSplit
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import roc_curve, auc

print("**************Grid search cv perform on train dataset*****************")

svm_linear = SGDClassifier(loss='hinge')
#params we need to try on classifier
param_grid = {'alpha':[1000,500,100,50,10,5,1,0.5,0.1,0.05,0.01,0.005,0.001,0.0005,0.00
01],
             'penalty':['l1','l2']}
ts_cv = TimeSeriesSplit(n_splits=10) #For time based splitting
grid = GridSearchCV(svm_linear,param_grid,cv=ts_cv,verbose=1,scoring='roc_auc')
grid.fit(x_train_svd,y_train)

penalty_hyperparameter_svd=grid.best_params_['penalty']
C_hyperparameter_svd=grid.best_params_['alpha']

train_auc= grid.cv_results_['mean_train_score']
train_auc_std= grid.cv_results_['std_train_score']
cv_auc = grid.cv_results_['mean_test_score']
cv_auc_std= grid.cv_results_['std_test_score']
#savetofile(gsv,"Log Reg/gsv_uni")
print("Best HyperParameter: ",grid.best_params_)
print("Best Accuracy: %.2f%%"%(grid.best_score_*100))
```

```
Wall time: 0 ns
**************Grid search cv perform on train dataset*****************
Fitting 10 folds for each of 30 candidates, totalling 300 fits

[Parallel(n_jobs=1)]: Done 300 out of 300 | elapsed: 15.9min finished

Best HyperParameter:  {'alpha': 0.0001, 'penalty': 'l1'}
Best Accuracy: 66.23%
```

### AUC plot for test and train dataset, SET 5

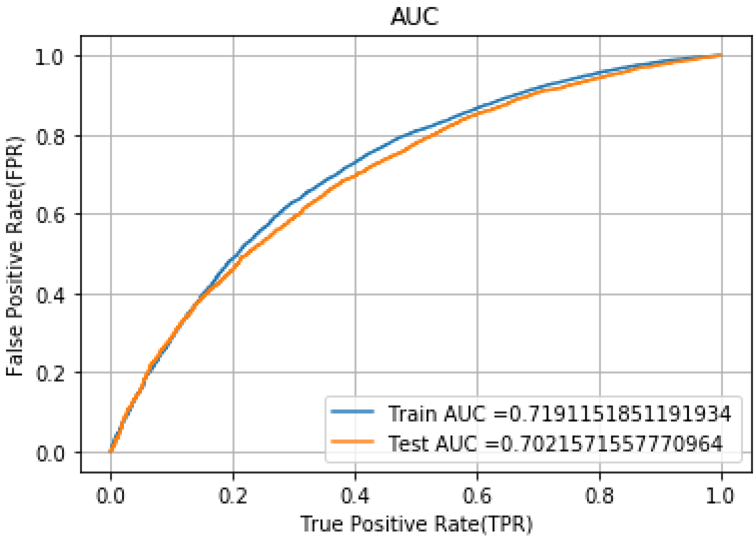
In [0]:

```
'''plt.figure(figsize=(20,20))

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

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

plt.scatter(param_grid['alpha'], train_auc, label='Train AUC points')
plt.scatter(param_grid['alpha'], cv_auc, label='CV AUC points')


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

### ROC of tain ,test and cv dataset, SET 5

In [307]:

```
#value taken from from GridsearchCV section
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#skle
arn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc


clf = SGDClassifier(loss='hinge', penalty=penalty_hyperparameter_svd, alpha= C_hyperpar
ameter_svd)

clf.fit(x_train_svd, y_train)

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

y_train_pred = clf.decision_function(x_train_svd)
y_test_pred = clf.decision_function(x_test_svd)

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)

svd_auc_train=auc(train_fpr, train_tpr)
svd_auc_test=auc(test_fpr, test_tpr)

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, SET 5

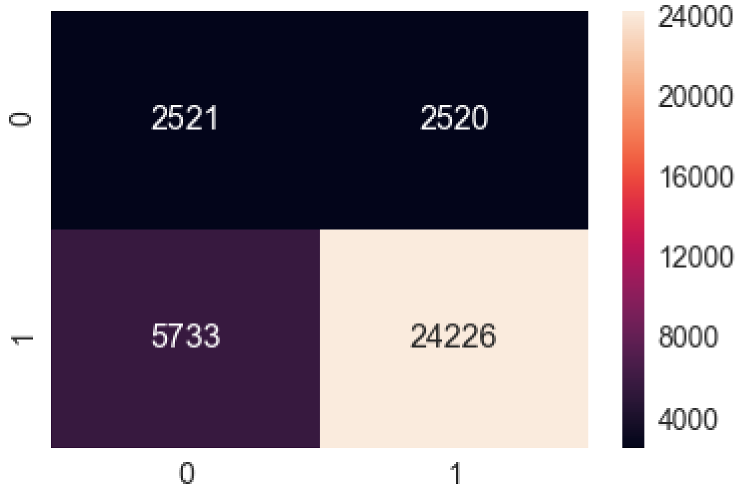
In [309]:

```
#https://seaborn.pydata.org/generated/seaborn.heatmap.html
#-----------------------------------------------

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

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

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

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

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


conf_matr_df_train_5_l2 = pd.DataFrame(confusion_matrix(y_train, predict(y_train_pred,
tr_thresholds, train_fpr, train_fpr)), range(2),range(2))

sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_train_5_l2, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
===========================================================================
==========================
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.24999999016200464 for threshold 0.347
[[ 2521  2520]
 [ 5733 24226]]
the maximum value of tpr*(1-fpr) 0.24999999016200464 for threshold 0.347
```

Out[309]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1f7a10326a0>
```



# 3. Conclusion

In [0]:

```
# Please compare all your models using Prettytable library
```

In [0]:

```
!pip install prettytable
```

In [417]:

```
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Vectorizer", "Model","Hyper parameter_penalty", "Hyper parameter_alph
a","AUC_Train", "AUC_test"]
x.add_row(["BOW", "SVM_Linear", penalty_hyperparameter_bow,alpha_hyperparameter_bow, BO
W_auc_train,BOW_auc_test])
x.add_row(["TFIDF", "SVM_Linear", penalty_hyperparameter_tfidf,alpha_hyperparameter_tfi
df, TFIDF_auc_train,TFIDF_auc_test])
x.add_row(["AVG W2V", "SVM_Linear", penalty_hyperparameter_AVGW2V,alpha_hyperparameter_
AVGW2V, AVGW2V_auc_train,AVGW2V_auc_test])
x.add_row(["TFIDF AVG W2V", "SVM_Linear", penalty_hyperparameter_TFIDFW2V,alpha_hyperpa
rameter_TFIDFW2V, TFIDFW2V_auc_train,TFIDFW2V_auc_test])
x.add_row(["SET 5", "SVM_Linear", penalty_hyperparameter_svd, C_hyperparameter_svd, svd
_auc_train,svd_auc_test])
print(x)

with open('Result_SVM.txt','w') as file:
    file.write(str(x))
```

```
+---------------+------------+-------------------------+------------------
-----+--------------------+--------------------+
|   Vectorizer  |   Model    | Hyper parameter_penalty | Hyper parameter_a
lpha |     AUC_Train      |      AUC_test      |
+---------------+------------+-------------------------+------------------
-----+--------------------+--------------------+
|      BOW      | SVM_Linear |            l2           |          0.01
| 0.7620963563442401 | 0.690999625377191  |
|     TFIDF     | SVM_Linear |            l2           |         0.005
| 0.7727267486532099 | 0.6658422972782441 |
|    AVG W2V    | SVM_Linear |            l1           |         0.0001
| 0.6919777669642139 | 0.6767997729396203 |
| TFIDF AVG W2V | SVM_Linear |            l1           |         0.0001
| 0.6889616986058695 | 0.6779820018128324 |
|     SET 5     | SVM_Linear |            l1           |         0.0001
| 0.7191151851191934 | 0.7021571557770964 |
+---------------+------------+-------------------------+------------------
-----+--------------------+--------------------+
```

Observation: Set 5 dont at 40000 dataset only because memort due to TruncatedSVD use. rest done at full dataset

1)BOW, TFIDF (combine all except vectorized dataset) give optimal result with L2 Regularizer while AVG W2W and TFIDF weighted AVG W2V shows optimality with L1 Regularization.

2) Hyper parameter prefer high value for model with AVG W2V vectorizer. and low value for last three model.

3)AUC value of trainin is higher than test data bet not much difference which show that model in all case is not underfit, nor overfit.