Applying SVM on Donors Choose dataset

DonorsChoose

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

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

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

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

About the DonorsChoose Data Set

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

Feature	Description
`project_id`	A unique identifier for the proposed project. **Example:** `p036502`
`project_title`	Title of the project. **Examples:** • `Art Will Make You Happy!` • `First Grade Fun`
`project_grade_category`	Grade level of students for which the project is targeted. One of the following enumerated values: • `Grades PreK-2` • `Grades 3-5` • `Grades 6-8` • `Grades 9-12`
`project_subject_categories`	One or more (comma-separated) subject categories for the project from the following enumerated list of values: • `Applied Learning` • `Care & Hunger` • `Health & Sports` • `History & Civics` • `Literacy & Language` • `Math & Science` • `Music & The Arts` • `Special Needs` • `Warmth` **Examples:** • `Music & The Arts` • `Literacy & Language, Math & Science`
`school_state`	State where school is located (Two-letter U.S. postal code (https://en.wikipedia.org/wiki/List_of_U.S._state_abbreviations#Postal_codes)). **Example:** `WY`
`project_subject_subcategories`	One or more (comma-separated) subject subcategories for the project. **Examples:** • `Literacy` • `Literature & Writing, Social Sciences`
`project_resource_summary`	An explanation of the resources needed for the project. **Example:** • `My students need hands on literacy materials to manage sensory needs!`
`project_essay_1`	First application essay*
`project_essay_2`	Second application essay*
`project_essay_3`	Third application essay*
`project_essay_4`	Fourth application essay*
`project_submitted_datetime`	Datetime when project application was submitted. **Example:** `2016-04-28 12:43:56.245`
`teacher_id`	A unique identifier for the teacher of the proposed project. **Example:** `bdf8baa8fedef6bfeec7ae4ff1c15c56`

teacher_prefix	Teacher's title. One of the following enumerated values: • `nan` • `Dr.` • `Mr.` • `Mrs.` • `Ms.` • `Teacher.`
teacher_number_of_previously_posted_projects	Number of project applications previously submitted by the same teacher. **Example:** `2`

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

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

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

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

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

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

Notes on the Essay Data

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

- __project_essay_1:__ "Introduce us to your classroom"
- __project_essay_2:__ "Tell us more about your students"
- __project_essay_3:__ "Describe how your students will use the materials you're requesting"
- __project_essay_4:__ "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 [3]:

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

import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pandas as pd
import sqlite3
import string
import nltk
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import CountVectorizer

from sklearn import metrics
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_curve,auc
from nltk.stem.porter import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer

from gensim.models import word2vec
from gensim.models import keyedvectors

from tqdm import tqdm
import os
import pickle
import re
from nltk.corpus import stopwords

import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
from collections import Counter
```

Reading data

In [4]:

```
project_data = pd.read_csv('train_data.csv', nrows=15000)
resource_data = pd.read_csv('resources.csv', nrows=15000)
project_data.shape
```

Out[4]:

```
(15000, 17)
```

In [5]:

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

Out[5]:

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

In [6]:

```
resource_data.head(2)
```

Out[6]:

	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 [7]:

```
print(project_data.shape)
print(resource_data.shape)
```

```
(15000, 17)
(15000, 4)
```

In [8]:

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

Out[8]:

	id	price	quantity
0	p000157	3508.32	9
1	p000170	208.51	5

In [9]:

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

Out[9]:

```
(15000, 19)
```

preprocessing of project_subject_categories

In [10]:

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

# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-string
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-python
cat_list = []
for i in catogories:
    temp = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
    for j in i.split(','): # it will split it in three parts ["Math & Science", "Warmth", "Care & Hunger"]
        if 'The' in j.split(): # this will split each of the catogory based on space "Math & Science"=> "Math","&
", "Science"
            j=j.replace('The','') # if we have the words "The" we are going to replace it with ''(i.e removing 'T
he')
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"Math & Science"=>"Math&Scie
nce"
        temp+=j.strip()+" " #" abc ".strip() will return "abc", remove the trailing spaces
        temp = temp.replace('&','_') # we are replacing the & value into
    cat_list.append(temp.strip())

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

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

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

preprocessing of project_subject_subcategories

In [11]:

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

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

sub_cat_list = []
for i in sub_catogories:
    temp = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
    for j in i.split(','): # it will split it in three parts ["Math & Science", "Warmth", "Care & Hunger"]
        if 'The' in j.split(): # this will split each of the catogory based on space "Math & Science"=> "Math","&
", "Science"
            j=j.replace('The','') # if we have the words "The" we are going to replace it with ''(i.e removing 'T
he')
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"Math & Science"=>"Math&Scie
nce"
        temp +=j.strip()+" "#" abc ".strip() will return "abc", remove the trailing spaces
        temp = temp.replace('&','_')
    sub_cat_list.append(temp.strip())

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

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

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

Text preprocessing (Project_essay)

In [12]:

```
# 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 [13]:

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

Out[13]:

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

In [14]:

```
# printing some random essays.
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[2000])
print("="*50)
print(project_data['essay'].values[4999])
print("="*50)
```

My students are English learners that are working on English as their second or third languages. We are a melting pot of refugees, immigrants, and native-born Americans bringing the gift of language to our school. \r\n\r\n We have over 24 languages represented in our English Learner program with students at every level of mastery. We also have over 40 countries represented with the families within our school. Each student brings a wealth of knowledge and experiences to us that open our eyes to new cultures, beliefs, and respect.\"The limits of your language are the limits of your world.\"-Ludwig Wittgenstein Our English learner's have a strong support system at home that begs for more resources. Many times our parents are learning to read and speak English along side of their children. Sometimes this creates barriers for parents to be able to help their child learn phonetics, letter recognition, and other reading skills.\r\n\r\nBy providing these dvd's and players, students are able to continue their mastery of the English language even if no one at home is able to assist. All families with students within the Level 1 proficiency status, will be a offered to be a part of this program. These educational videos will be specially chosen by the English Learner Teacher and will be sent home regularly to watch. The videos are to help the child develop early reading skills.\r\n\r\nParents that do not have access to a dvd player will have the opportunity to check out a dvd player to use for the year. The plan is to use these videos and educational dvd's for the years to come for other EL students.\r\nnannan

==

The 51 fifth grade students that will cycle through my classroom this year all love learning, at least most of the time. At our school, 97.3% of the students receive free or reduced price lunch. Of the 560 students, 97.3% are minority students. \r\nThe school has a vibrant community that loves to get together and celebrate. Around Halloween there is a whole school parade to show off the beautiful costumes that students wear. On Cinco de Mayo we put on a big festival with crafts made by the students, dances, and games. At the end of the year the school hosts a carnival to celebrate the hard work put in during the school year, with a dunk tank being the most popular activity.My students will use these five brightly colored Hokki stools in place of regular, stationary, 4-legged chairs. As I will only have a total of ten in the classroom and not enough for each student to have an individual one, they will be used in a variety of ways. During independent reading time they will be used as special chairs students will each use on occasion. I will utilize them in place of chairs at my small group tables during math and reading times. The rest of the day they will be used by the students who need the highest amount of movement in their life in order to stay focused on school.\r\n\r\nWhenever asked what the classroom is missing, my students always say more Hokki Stools. They can't get their fill of the 5 stools we already have. When the students are sitting in group with me on the Hokki Stools, they are always moving, but at the same time doing their work. Anytime the students get to pick where they can sit, the Hokki Stools are the first to be taken. There are always students who head over to the kidney table to get one of the stools who are disappointed as there are not enough of them. \r\n\r\nWe ask a lot of students to sit for 7 hours a day. The Hokki stools will be a compromise that allow my students to do desk work and move at the same time. These stools will help students to meet their 60 minutes a day of movement by allowing them to activate their core muscles for balance while they sit. For many of my students, these chairs will take away the barrier that exists in schools for a child who can't sit still.nannan

==

How do you remember your days of school? Was it in a sterile environment with plain walls, rows of desks, and a teacher in front of the room? A typical day in our room is nothing like that. I work hard to create a warm inviting themed room for my students look forward to coming to each day.\r\n\r\nMy class is made up of 28 wonderfully unique boys and girls of mixed races in Arkansas.\r\nThey attend a Title I school, which means there is a high enough percentage of free and reduced-price lunch to qualify. Our school is an \"open classroom\" concept, which is very unique as there are no walls separating the classrooms. These 9 and 10 year-old students are very eager learners; they are like sponges, absorbing all the information and experiences and keep on wanting more.With these resources such as the comfy red throw pillows and the whimsical nautical hanging decor and the blue fish nets, I will be able to help create the mood in our classroom setting to be one of a themed nautical environment. Creating a classroom environment is very important in the success in each and every child's education. The nautical photo props will be used with each child as they step foot into our classroom for the first time on Meet the Teacher evening. I'll take pictures of each child with them, have them developed, and then hung in our classroom ready for their first day of 4th grade. This kind gesture will set the tone before even the first day of school! The nautical thank you cards will be used throughout the year by the students as they create thank you cards to their team groups.\r\n\r\nYour generous donations will help me to help make our classroom a fun, inviting, learning environment from day one.\r\n\r\nIt costs lost of money out of my own pocket on resources to get our classroom ready. Please consider helping with this project to make our new school year a very successful one. Thank you!nannan

==

Describing my students isn't an easy task. Many would say that they are inspirational, creative, and hard-working. They are all unique - unique in their interests, their learning, their abilities, and so much more. What they all have in common is their desire to learn each day, despite difficulties that they encounter. \r\nOur classroom is amazing - because we understand that everyone learns at their own pace. As the teacher, I pride myself in making sure my students are always engaged, motivated, and inspired to create their own learning! \r\nThis project is to help my students choose seating that is more appropriate for them, developmentally. Many students tire of sitting in chairs during lessons, and having different seats available helps to keep them engaged and learning.\r\nFlexible seating is important in our classroom, as many of our students struggle with attention, focus, and engagement. We currently have stability balls for seating, as well as regular chairs, but these stools will help students who have trouble with balance, or find it difficult to sit on a stability ball for a long period of time. We are excited to try these stools as a part of our engaging classroom community!nannan

==

Loud and proud are who we are. We are a special basketball family like no other. Our school is in a great community with vast diverseness. We are surrounded by colleges and low income housing. We pride ourselves in preparing our athletes to be great on and off the court.\r\n\r\nOur students recite every day that, \"We are destined for greatness.\" I believe this wholeheartedly. I am forming winners in life and in basketball. A great of kids is coming your way!We need socks to add to our two uniforms. Every basketball season our girls basketball team strives to play their best. Not only do I push them to give it all on the court I also to teach them to take pride in how they look on th

```
e team.  We want to look like a team from head to toe.\r\n\r\nGirls should feel good about themselve
s as they play ball and look good on and off the court.  I have seen lime green socks, purple socks,
and all the crazy mismatched socks there is.  We need uniformity all the way around.nannan
==================================================
```

In [15]:

```python
import re

def decontracted(phrase):
    # specific
    phrase = re.sub(r"won't", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)

    # general
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)
    return phrase
```

In [16]:

```python
sent = decontracted(project_data['essay'].values[2000])
print(sent)# \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)
print("="*50)
```

```
Describing my students is not an easy task.  Many would say that they are inspirational, creative, a
nd hard-working.  They are all unique - unique in their interests, their learning, their abilities,
and so much more.  What they all have in common is their desire to learn each day, despite difficult
ies that they encounter.  \r\nOur classroom is amazing - because we understand that everyone learns
at their own pace.  As the teacher, I pride myself in making sure my students are always engaged, mo
tivated, and inspired to create their own learning! \r\nThis project is to help my students choose s
eating that is more appropriate for them, developmentally.  Many students tire of sitting in chairs
during lessons, and having different seats available helps to keep them engaged and learning.\r\nFle
xible seating is important in our classroom, as many of our students struggle with attention, focus,
and engagement.  We currently have stability balls for seating, as well as regular chairs, but these
stools will help students who have trouble with balance, or find it difficult to sit on a stability
ball for a long period of time.  We are excited to try these stools as a part of our engaging classr
oom community!nannan
Describing my students is not an easy task.  Many would say that they are inspirational, creative, a
nd hard-working.  They are all unique - unique in their interests, their learning, their abilities,
and so much more.  What they all have in common is their desire to learn each day, despite difficult
ies that they encounter.    Our classroom is amazing - because we understand that everyone learns at
their own pace.  As the teacher, I pride myself in making sure my students are always engaged, motiv
ated, and inspired to create their own learning!   This project is to help my students choose seatin
g that is more appropriate for them, developmentally.  Many students tire of sitting in chairs durin
g lessons, and having different seats available helps to keep them engaged and learning.  Flexible s
eating is important in our classroom, as many of our students struggle with attention, focus, and en
gagement.  We currently have stability balls for seating, as well as regular chairs, but these stool
s will help students who have trouble with balance, or find it difficult to sit on a stability ball
for a long period of time.  We are excited to try these stools as a part of our engaging classroom c
ommunity!nannan
==================================================
```

In [17]:

```
# \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)
```

```
Describing my students is not an easy task.  Many would say that they are inspirational, creative, a
nd hard-working.  They are all unique - unique in their interests, their learning, their abilities,
and so much more.  What they all have in common is their desire to learn each day, despite difficult
ies that they encounter.    Our classroom is amazing - because we understand that everyone learns at
their own pace.  As the teacher, I pride myself in making sure my students are always engaged, motiv
ated, and inspired to create their own learning!   This project is to help my students choose seatin
g that is more appropriate for them, developmentally.  Many students tire of sitting in chairs durin
g lessons, and having different seats available helps to keep them engaged and learning.  Flexible s
eating is important in our classroom, as many of our students struggle with attention, focus, and en
gagement.  We currently have stability balls for seating, as well as regular chairs, but these stool
s will help students who have trouble with balance, or find it difficult to sit on a stability ball
for a long period of time.  We are excited to try these stools as a part of our engaging classroom c
ommunity!nannan
```

In [18]:

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

```
Describing my students is not an easy task Many would say that they are inspirational creative and h
ard working They are all unique unique in their interests their learning their abilities and so much
more What they all have in common is their desire to learn each day despite difficulties that they e
ncounter Our classroom is amazing because we understand that everyone learns at their own pace As th
e teacher I pride myself in making sure my students are always engaged motivated and inspired to cre
ate their own learning This project is to help my students choose seating that is more appropriate f
or them developmentally Many students tire of sitting in chairs during lessons and having different
seats available helps to keep them engaged and learning Flexible seating is important in our classro
om as many of our students struggle with attention focus and engagement We currently have stability
balls for seating as well as regular chairs but these stools will help students who have trouble wit
h balance or find it difficult to sit on a stability ball for a long period of time We are excited t
o try these stools as a part of our engaging classroom community nannan
```

In [19]:

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

In [20]:

```
# 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.lower() not in stopwords)
    preprocessed_essays.append(sent.lower().strip())
```

```
100%|███████████████████████████████████| 15000/15000 [00:15<00:00, 939.80it/s]
```

In [21]:

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

Out[21]:

```
'describing students not easy task many would say inspirational creative hard working unique unique
interests learning abilities much common desire learn day despite difficulties encounter classroom a
mazing understand everyone learns pace teacher pride making sure students always engaged motivated i
nspired create learning project help students choose seating appropriate developmentally many studen
ts tire sitting chairs lessons different seats available helps keep engaged learning flexible seatin
g important classroom many students struggle attention focus engagement currently stability balls se
ating well regular chairs stools help students trouble balance find difficult sit stability ball lon
g period time excited try stools part engaging classroom community nannan'
```

Preprocessing of project_title

In [22]:

```
sent_0=project_data["project_title"].values[11]
print(sent_0)
print("="*50)

sent_1000=project_data["project_title"].values[34]
print(sent_1000)
print("="*50)

sent_1500=project_data["project_title"].values[147]
print(sent_1500)
print("="*50)

sent_1500=project_data["project_title"].values[1277]
print(sent_1500)
print("="*50)
```

```
Elevating Academics and Parent Rapports Through Technology
==================================================
\"Have A Ball!!!\"
==================================================
Who needs a Chromebook?\r\nWE DO!!
==================================================
Time Keeper=Empathy Builder
==================================================
```

In [23]:

```
# 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 [24]:

```
sent = decontracted(project_data['project_title'].values[34])
print(sent)
print("="*50)
```

```
\"Have A Ball!!!\"
==================================================
```

In [25]:

```
sent = sent.replace('\\r', ' ')
sent = sent.replace('\\"', ' ')
sent = sent.replace('\\n', ' ')
print(sent)
```

```
 Have A Ball!!!
```

In [26]:

```
sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
print(sent)
```

```
 Have A Ball
```

In [27]:

```
from tqdm import tqdm
preprocessed_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_title.append(sent.lower().strip())
```

```
100%|█████████████████████████████████| 15000/15000 [00:00<00:00, 20160.13it/s]
```

In [28]:

```
preprocessed_title[34]
```

Out[28]:

```
'have a ball'
```

In [29]:

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

Out[29]:

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

Adding a new feature Number of words in title

In [30]:

```
project_data['preprocessed_title']=preprocessed_title
```

In [31]:

```
title_word_count = []
```

In [32]:

```
for a in project_data["preprocessed_title"]:
    b = len(a.split())
    title_word_count.append(b)
```

In [33]:

```
project_data["title_word_count"] = title_word_count
```

In [34]:

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

Out[34]:

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

2 rows × 22 columns

Adding a new feature Number of words in essay

In [35]:

```
project_data['preprocessed_essays']=preprocessed_essays
```

In [36]:

```
essay_word_count=[]
```

In [37]:

```
for ess in project_data["preprocessed_essays"] :
    c = len(ess.split())
    essay_word_count.append(c)
```

In [38]:

```
project_data["essay_word_count"] = essay_word_count
```

In [39]:

```
print(project_data.shape)
project_data.head(2)
```

```
(15000, 24)
```

Out[39]:

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

2 rows × 24 columns

Calculating sentiment scores of essay

In [40]:

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

analyser = SentimentIntensityAnalyzer()


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

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

```
100%|██████████████████████████████████████████| 15000/15000 [02:59<00:00, 83.56it/s]
```

In [41]:

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

In [42]:

```
y = project_data['project_is_approved'].values
X = project_data.drop(['project_is_approved'], axis=1)
print(X.shape)
print(y.shape)
X.head(1)
```

```
(15000, 27)
(15000,)
```

Out[42]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime	project_grade_catego
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	2016-12-05 13:43:57	Grades Prek

1 rows × 27 columns

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

In [43]:

```
# train test split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, stratify=y)
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33, stratify=y_train)
```

In [44]:

```
print(X_train.shape, y_train.shape)
print(X_cv.shape, y_cv.shape)
print(X_test.shape, y_test.shape)
```

```
(6733, 27) (6733,)
(3317, 27) (3317,)
(4950, 27) (4950,)
```

Preparing Data For Models

Make Data Model Ready: encoding numerical, categorical features

Vectorizing categorical data

In [45]:

```
vectorizer = CountVectorizer()
vectorizer.fit(X_train['school_state'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_state_ohe = vectorizer.transform(X_train['school_state'].values)
X_cv_state_ohe = vectorizer.transform(X_cv['school_state'].values)
X_test_state_ohe = vectorizer.transform(X_test['school_state'].values)

print("After vectorizations")
print(X_train_state_ohe.shape, y_train.shape)
print(X_cv_state_ohe.shape, y_cv.shape)
print(X_test_state_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(6733, 51) (6733,)
(3317, 51) (3317,)
(4950, 51) (4950,)
['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl', 'ga', 'hi', 'ia', 'id', 'il', 'in', 'ks
', 'ky', 'la', 'ma', 'md', 'me', 'mi', 'mn', 'mo', 'ms', 'mt', 'nc', 'nd', 'ne', 'nh', 'nj', 'nm', '
nv', 'ny', 'oh', 'ok', 'or', 'pa', 'ri', 'sc', 'sd', 'tn', 'tx', 'ut', 'va', 'vt', 'wa', 'wi', 'wv',
'wy']
====================================================================================================
```

In [46]:

```
vectorizer = CountVectorizer()
vectorizer.fit(X_train['teacher_prefix'].values.astype('U')) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
#https://stackoverflow.com/questions/39303912/tfidfvectorizer-in-scikit-learn-valueerror-np-nan-is-an-invalid-doc
ument
X_train_teacher_ohe = vectorizer.transform(X_train['teacher_prefix'].values.astype('U'))
X_cv_teacher_ohe = vectorizer.transform(X_cv['teacher_prefix'].values.astype('U'))
X_test_teacher_ohe = vectorizer.transform(X_test['teacher_prefix'].values.astype('U'))

print("After vectorizations")
print(X_train_teacher_ohe.shape, y_train.shape)
print(X_cv_teacher_ohe.shape, y_cv.shape)
print(X_test_teacher_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(6733, 4) (6733,)
(3317, 4) (3317,)
(4950, 4) (4950,)
['mr', 'mrs', 'ms', 'teacher']
====================================================================================================
```

In [47]:

```
vectorizer = CountVectorizer()
vectorizer.fit(X_train['project_grade_category'].values.astype('U')) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_grade_ohe = vectorizer.transform(X_train['project_grade_category'].values)
X_cv_grade_ohe = vectorizer.transform(X_cv['project_grade_category'].values)
X_test_grade_ohe = vectorizer.transform(X_test['project_grade_category'].values)

print("After vectorizations")
print(X_train_grade_ohe.shape, y_train.shape)
print(X_cv_grade_ohe.shape, y_cv.shape)
print(X_test_grade_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(6733, 3) (6733,)
(3317, 3) (3317,)
(4950, 3) (4950,)
['12', 'grades', 'prek']
====================================================================================================
```

In [48]:

```
vectorizer = CountVectorizer()
vectorizer.fit(X_train['clean_categories'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_clean_categories_ohe = vectorizer.transform(X_train['clean_categories'].values)
X_cv_clean_categories_ohe = vectorizer.transform(X_cv['clean_categories'].values)
X_test_clean_categories_ohe = vectorizer.transform(X_test['clean_categories'].values)

print("After vectorizations")
print(X_train_clean_categories_ohe.shape , y_train.shape)
print(X_cv_clean_categories_ohe.shape , y_cv.shape)
print(X_test_clean_categories_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(6733, 9) (6733,)
(3317, 9) (3317,)
(4950, 9) (4950,)
['appliedlearning', 'care_hunger', 'health_sports', 'history_civics', 'literacy_language', 'math_sci
ence', 'music_arts', 'specialneeds', 'warmth']
====================================================================================================
```

In [49]:

```
vectorizer = CountVectorizer()
vectorizer.fit(X_train['clean_subcategories'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_clean_subcategories_ohe = vectorizer.transform(X_train['clean_subcategories'].values)
X_cv_clean_subcategories_ohe = vectorizer.transform(X_cv['clean_subcategories'].values)
X_test_clean_subcategories_ohe = vectorizer.transform(X_test['clean_subcategories'].values)

print("After vectorizations")
print(X_train_clean_subcategories_ohe.shape , y_train.shape)
print(X_cv_clean_subcategories_ohe.shape , y_cv.shape)
print(X_test_clean_subcategories_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(6733, 30) (6733,)
(3317, 30) (3317,)
(4950, 30) (4950,)
['appliedsciences', 'care_hunger', 'charactereducation', 'civics_government', 'college_careerprep',
'communityservice', 'earlydevelopment', 'economics', 'environmentalscience', 'esl', 'extracurricular
', 'financialliteracy', 'foreignlanguages', 'gym_fitness', 'health_lifescience', 'health_wellness',
'history_geography', 'literacy', 'literature_writing', 'mathematics', 'music', 'nutritioneducation',
'other', 'parentinvolvement', 'performingarts', 'socialsciences', 'specialneeds', 'teamsports', 'vis
ualarts', 'warmth']
====================================================================================================
```

Bag of words on essay

In [50]:

```
vectorizer = CountVectorizer(min_df=10,ngram_range=(2,2), max_features=5000)
vectorizer.fit(X_train['essay'].values)# fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_essay_bow = vectorizer.transform(X_train['essay'].values)
X_cv_essay_bow = vectorizer.transform(X_cv['essay'].values)
X_test_essay_bow = vectorizer.transform(X_test['essay'].values)


print('Bow on essay')
print(X_train_essay_bow.shape, y_train.shape)
print(X_cv_essay_bow.shape, y_cv.shape)
print(X_test_essay_bow.shape, y_test.shape)

print('-'*50)
```

```
Bow on essay
(6733, 5000) (6733,)
(3317, 5000) (3317,)
(4950, 5000) (4950,)
--------------------------------------------------
```

TFIDF vectorizer on essays

In [51]:

```
vectorizer = TfidfVectorizer(min_df=10,ngram_range=(2,2),max_features=5000)
vectorizer.fit(X_train['essay'].values)# fit has to happen only on train data
# we use the fitted CountVectorizer to convert the text to vector
X_train_essay_tfidf = vectorizer.transform(X_train['essay'].values)
X_cv_essay_tfidf = vectorizer.transform(X_cv['essay'].values)
X_test_essay_tfidf = vectorizer.transform(X_test['essay'].values)

print('Tfidf vectrizer on essay')
print(X_train_essay_tfidf.shape, y_train.shape)
print(X_cv_essay_tfidf.shape, y_cv.shape)
print(X_test_essay_tfidf.shape, y_test.shape)

print('-'*50)
```

```
Tfidf vectrizer on essay
(6733, 5000) (6733,)
(3317, 5000) (3317,)
(4950, 5000) (4950,)
--------------------------------------------------
```

Bag of words on project title

In [52]:

```
vectorizer = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
vectorizer.fit(X_train['project_title'].values)
X_train_title_bow = vectorizer.transform(X_train['project_title'].values)
X_cv_title_bow = vectorizer.transform(X_cv['project_title'].values)
X_test_title_bow= vectorizer.transform(X_test['project_title'].values)

print('Bow on project title')
print(X_train_title_bow.shape, y_train.shape)
print(X_cv_title_bow.shape, y_cv.shape)
print(X_test_title_bow.shape, y_test.shape)
print('='*50)
```

```
Bow on project title
(6733, 781) (6733,)
(3317, 781) (3317,)
(4950, 781) (4950,)
==================================================
```

TFIDF vectorizer on project title

In [53]:

```
vectorizer = TfidfVectorizer(min_df=10,ngram_range=(1,4),max_features=5000)
vectorizer.fit(X_train['project_title'].values)
X_train_title_tfidf = vectorizer.transform(X_train['project_title'].values)
X_cv_title_tfidf = vectorizer.transform(X_cv['project_title'].values)
X_test_title_tfidf = vectorizer.transform(X_test['project_title'].values)

print('tfidf vectorizer on project title')
print(X_train_title_tfidf.shape, y_train.shape)
print(X_cv_title_tfidf.shape, y_cv.shape)
print(X_test_title_tfidf.shape, y_test.shape)
print('='*50)
```

```
tfidf vectorizer on project title
(6733, 781) (6733,)
(3317, 781) (3317,)
(4950, 781) (4950,)
==================================================
```

Vectorizing Numerical features

Vectorizing- teacher number of previously posted projects

In [54]:

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

X_train_tnopp_norm = normalizer.transform(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))
X_cv_tnopp_norm = normalizer.transform(X_cv['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))
X_test_tnopp_norm = normalizer.transform(X_test['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))

print("After vectorizations")
print(X_train_tnopp_norm.shape, y_train.shape)
print(X_cv_tnopp_norm.shape, y_cv.shape)
print(X_test_tnopp_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(6733, 1) (6733,)
(3317, 1) (3317,)
(4950, 1) (4950,)
====================================================================================================
```

Vectorizing - price

In [55]:

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()


#https://datascience.stackexchange.com/questions/11928/valueerror-input-contains-nan-infinity-or-a-value-too-large-for-dtypefloat32
X_train['price'].fillna(X_train['price'].mean(), inplace=True)
X_cv['price'].fillna(X_cv['price'].mean(), inplace=True)
X_test['price'].fillna(X_test['price'].mean(), inplace=True)
normalizer.fit(X_train['price'].values.reshape(1,-1))

X_train_price_norm = normalizer.transform(X_train['price'].values.reshape(-1,1))
X_cv_price_norm = normalizer.transform(X_cv['price'].values.reshape(-1,1))
X_test_price_norm = normalizer.transform(X_test['price'].values.reshape(-1,1))

print("After vectorizations")
print(X_train_price_norm.shape, y_train.shape)
print(X_cv_price_norm.shape, y_cv.shape)
print(X_test_price_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(6733, 1) (6733,)
(3317, 1) (3317,)
(4950, 1) (4950,)
====================================================================================================
```

Vectorizing quantity

In [56]:

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()

X_train['quantity'].fillna(X_train['quantity'].mean(), inplace=True)
X_cv['quantity'].fillna(X_cv['quantity'].mean(), inplace=True)
X_test['quantity'].fillna(X_test['quantity'].mean(), inplace=True)
normalizer.fit(X_train['quantity'].values.reshape(1,-1))



X_train_quantity_norm = normalizer.transform(X_train['quantity'].values.reshape(-1,1))
X_cv_quantity_norm = normalizer.transform(X_cv['quantity'].values.reshape(-1,1))
X_test_quantity_norm = normalizer.transform(X_test['quantity'].values.reshape(-1,1))

print("After vectorizations")
print(X_train_quantity_norm.shape, y_train.shape)
print(X_cv_quantity_norm.shape, y_cv.shape)
print(X_test_quantity_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(6733, 1) (6733,)
(3317, 1) (3317,)
(4950, 1) (4950,)
====================================================================================================
```

Vectorizeing essay_word_count

In [57]:

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()

X_train['essay_word_count'].fillna(X_train['essay_word_count'].mean())
normalizer.fit(X_train['essay_word_count'].values.reshape(1,-1))

X_train_ewcount_norm = normalizer.transform(X_train['essay_word_count'].values.reshape(-1,1))
X_cv_ewcount_norm = normalizer.transform(X_cv['essay_word_count'].values.reshape(-1,1))
X_test_ewcount_norm = normalizer.transform(X_test['essay_word_count'].values.reshape(-1,1))

print("After vectorizations")
print(X_train_ewcount_norm.shape, y_train.shape)
print(X_cv_ewcount_norm.shape, y_cv.shape)
print(X_test_ewcount_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(6733, 1) (6733,)
(3317, 1) (3317,)
(4950, 1) (4950,)
====================================================================================================
```

vectorizing title_word_count

In [58]:

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()

X_train['title_word_count'].fillna(X_train['title_word_count'].mean())
normalizer.fit(X_train['essay_word_count'].values.reshape(1,-1))

X_train_twcount_norm = normalizer.transform(X_train['title_word_count'].values.reshape(-1,1))
X_cv_twcount_norm = normalizer.transform(X_cv['title_word_count'].values.reshape(-1,1))
X_test_twcount_norm = normalizer.transform(X_test['title_word_count'].values.reshape(-1,1))

print("After vectorizations")
print(X_train_twcount_norm.shape, y_train.shape)
print(X_cv_twcount_norm.shape, y_cv.shape)
print(X_test_twcount_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(6733, 1) (6733,)
(3317, 1) (3317,)
(4950, 1) (4950,)
====================================================================================================
```

Vectorizing sentiment Scores essay of negative

In [59]:

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()

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

X_train_neg_norm = normalizer.transform(X_train['neg'].values.reshape(-1,1))
X_cv_neg_norm = normalizer.transform(X_cv['neg'].values.reshape(-1,1))
X_test_neg_norm = normalizer.transform(X_test['neg'].values.reshape(-1,1))

print("After vectorizations")
print(X_train_neg_norm.shape, y_train.shape)
print(X_cv_neg_norm.shape, y_cv.shape)
print(X_test_neg_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(6733, 1) (6733,)
(3317, 1) (3317,)
(4950, 1) (4950,)
====================================================================================================
```

Vectorizing sentiment Scores essay of positive

In [60]:

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()

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

X_train_pos_norm = normalizer.transform(X_train['pos'].values.reshape(-1,1))
X_cv_pos_norm = normalizer.transform(X_cv['pos'].values.reshape(-1,1))
X_test_pos_norm = normalizer.transform(X_test['pos'].values.reshape(-1,1))

print("After vectorizations")
print(X_train_pos_norm.shape, y_train.shape)
print(X_cv_pos_norm.shape, y_cv.shape)
print(X_test_pos_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(6733, 1) (6733,)
(3317, 1) (3317,)
(4950, 1) (4950,)
====================================================================================================
```

Vectorizing sentiment Scores essay of neutral

In [61]:

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()

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

X_train_neu_norm = normalizer.transform(X_train['neu'].values.reshape(-1,1))
X_cv_neu_norm = normalizer.transform(X_cv['neu'].values.reshape(-1,1))
X_test_neu_norm = normalizer.transform(X_test['neu'].values.reshape(-1,1))

print("After vectorizations")
print(X_train_neu_norm.shape, y_train.shape)
print(X_cv_neu_norm.shape, y_cv.shape)
print(X_test_neu_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(6733, 1) (6733,)
(3317, 1) (3317,)
(4950, 1) (4950,)
====================================================================================================
```

Vectorizing sentiment Scores essay of compound

In [62]:

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()

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

X_train_compound_norm = normalizer.transform(X_train['compound'].values.reshape(-1,1))
X_cv_compound_norm = normalizer.transform(X_cv['compound'].values.reshape(-1,1))
X_test_compound_norm = normalizer.transform(X_test['compound'].values.reshape(-1,1))

print("After vectorizations")
print(X_train_compound_norm.shape, y_train.shape)
print(X_cv_compound_norm.shape, y_cv.shape)
print(X_test_compound_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(6733, 1) (6733,)
(3317, 1) (3317,)
(4950, 1) (4950,)
====================================================================================================
```

Merging all the above

categorical features

In [63]:

```
# merging all the categorical features
from scipy.sparse import hstack
categorical_tr=hstack((X_train_state_ohe,X_train_teacher_ohe,X_train_grade_ohe,X_train_clean_categories_ohe,X_tra
in_clean_subcategories_ohe))
categorical_cv=hstack((X_cv_state_ohe,X_cv_teacher_ohe,X_cv_grade_ohe,X_cv_clean_categories_ohe,X_cv_clean_subcat
egories_ohe))
categorical_test=hstack((X_test_state_ohe,X_test_teacher_ohe,X_test_grade_ohe,X_test_clean_categories_ohe,X_test_
clean_subcategories_ohe))

print('='*50)

print('final datamatrix')
print(categorical_tr.shape,  y_train.shape)
print(categorical_cv.shape,  y_cv.shape)
print(categorical_test.shape, y_test.shape)
```

```
==================================================
final datamatrix
(6733, 97) (6733,)
(3317, 97) (3317,)
(4950, 97) (4950,)
```

numerical features

In [64]:

```
# merging all the numerical features
import scipy as sp
numerical_tr=sp.hstack((X_train_tnopp_norm,X_train_price_norm,X_train_quantity_norm,X_train_ewcount_norm,X_train_
twcount_norm,
                        X_train_neg_norm,X_train_neu_norm,X_train_pos_norm,X_train_compound_norm))

numerical_cv=sp.hstack((X_cv_tnopp_norm,X_cv_price_norm,X_cv_quantity_norm, X_cv_ewcount_norm, X_cv_twcount_norm,
X_cv_neg_norm,
                       X_cv_pos_norm, X_cv_neu_norm, X_cv_compound_norm))

numerical_test=sp.hstack((X_test_tnopp_norm,X_test_price_norm,X_test_quantity_norm, X_test_ewcount_norm, X_test_t
wcount_norm,
                         X_test_pos_norm,X_test_neg_norm, X_test_compound_norm,X_test_neu_norm))

print('='*100)

print('final matrix')
print(numerical_tr.shape, y_train.shape)
print(numerical_cv.shape, y_cv.shape)
print(numerical_test.shape, y_test.shape)
```

```
====================================================================================================
final matrix
(6733, 9) (6733,)
(3317, 9) (3317,)
(4950, 9) (4950,)
```

1.Applying SVM on categorical+numerical features + project_title(BOW) + preprocessed_eassay (BOW)

In [65]:

```
#  creating the matrix
x_tr_bow=hstack((categorical_tr,numerical_tr,X_train_essay_bow,X_train_title_bow)).tocsr()
x_cv_bow=hstack((categorical_cv,numerical_cv,X_cv_essay_bow,X_cv_title_bow)).tocsr()
x_test_bow=hstack((categorical_test,numerical_test,X_test_essay_bow,X_test_title_bow)).tocsr()

print('final matrix')
print(x_tr_bow.shape,   y_train.shape)
print(x_cv_bow.shape,   y_cv.shape)
print(x_test_bow.shape, y_test.shape)
```

```
final matrix
(6733, 5900) (6733,)
(3317, 5900) (3317,)
(4950, 5900) (4950,)
```

Hyper Parameter tuning

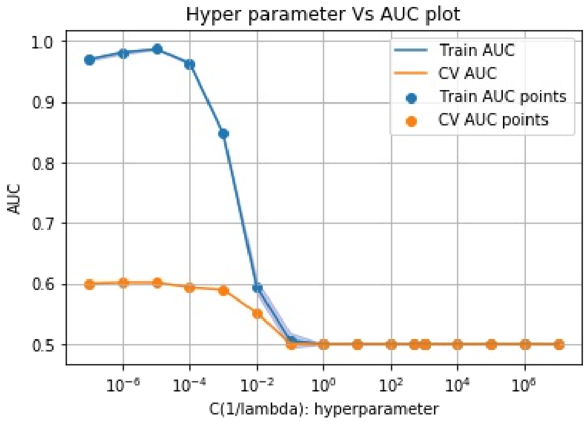
Grid search

In [77]:

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

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

    return y_data_pred
```

In [67]:

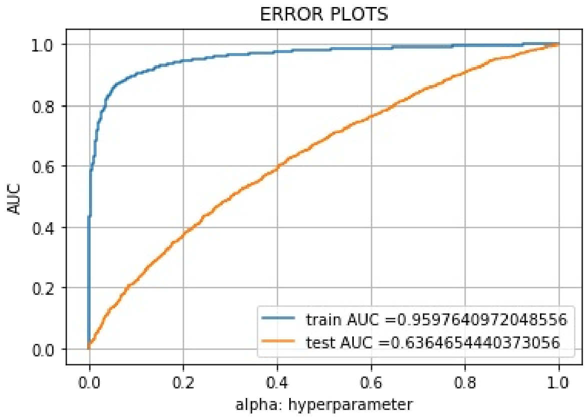
```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
import warnings
warnings.filterwarnings('ignore')
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
#from sklearn.grid_search import GridSearchCV
from sklearn.model_selection import learning_curve, GridSearchCV
from sklearn.model_selection import GridSearchCV
from sklearn import linear_model
from scipy.stats import randint as sp_randint
from sklearn.model_selection import RandomizedSearchCV


clf=linear_model.SGDClassifier(class_weight='balanced', penalty='l2' and 'l1')
parameters ={'alpha':[10**-7,10**-6,10**-5,10**-4, 10**-3,10**-2,10**-1,1,10,100,500,1000,1000,10000,10**5,10**6,
10**7]}
sd = GridSearchCV(clf, parameters, cv=5, scoring='roc_auc',return_train_score=True)
sd.fit(x_tr_bow, y_train)



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

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

plt.plot(parameters['alpha'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
# plt.gca().fill_between(K, cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='darkorange')

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



plt.legend()
plt.xlabel("C(1/lambda): hyperparameter")
plt.ylabel("AUC")
plt.title("Hyper parameter Vs AUC plot")
plt.grid()
plt.show()
```



Testing the performance of the model on test data, plotting ROC Curves

In [68]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc
##https://stackoverflow.com/questions/39200265/attributeerror-probability-estimates-are-not-available-for-loss-hi
nge
from sklearn.calibration import CalibratedClassifierCV


lr = linear_model.SGDClassifier(alpha=10**-2,class_weight='balanced', loss='hinge')
clf =lr.fit(x_tr_bow, y_train)
calibrator = CalibratedClassifierCV(clf, cv='prefit')
model=calibrator.fit(x_tr_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 = batch_predict(model, x_tr_bow)
y_test_pred = batch_predict(model, x_test_bow)

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

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



plotting confusion matrix

In [78]:

```
# we are writing our own function for predict, with defined threshold
# we will pick a threshold that will give the least fpr (False Positive Rate)
def predict(proba, threshold, fpr, tpr):

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

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

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

confusion matrix for train

In [70]:

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

```
====================================================================================================
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.24999906829057408 for threshold 0.308
[[ 517  519]
 [  99 5598]]
```

In [71]:

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

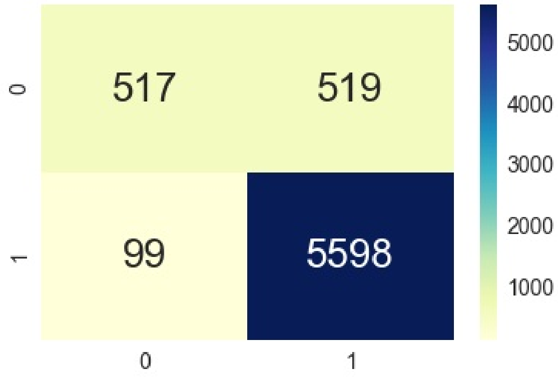
```
the maximum value of tpr*(1-fpr) 0.24999906829057408 for threshold 0.308
```

In [72]:

```
## Heatmaps -> https://likegeeks.com/seaborn-heatmap-tutorial/
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_train, annot=True,annot_kws={"size": 26}, fmt='g',cmap="YlGnBu")
```

Out[72]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x19a10940>
```



confusion matrix for test

In [73]:

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

```
====================================================================================================
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24999956831128556 for threshold 0.122
[[  47  714]
 [  93 4096]]
```

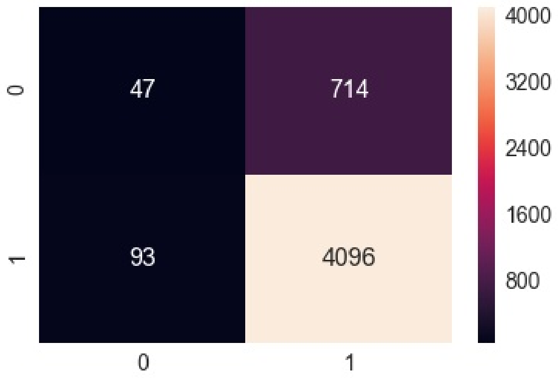
In [74]:

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

```
the maximum value of tpr*(1-fpr) 0.24999956831128556 for threshold 0.122
```

Out[74]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1e2e6c18>
```



2.Applying SVM on categorical+numerical features + project_title(TFIDF)+ preprocessed_eassay (TFIDF)

In [76]:

```
#  creating the matrix
x_tr_tfidf=hstack((categorical_tr,numerical_tr,X_train_essay_tfidf,X_train_title_tfidf)).tocsr()
x_cv_tfidf=hstack((categorical_cv,numerical_cv,X_cv_essay_tfidf,X_cv_title_tfidf)).tocsr()
x_test_tfidf=hstack((categorical_test,numerical_test,X_test_essay_tfidf,X_test_title_tfidf)).tocsr()

print('final matrix')
print(x_tr_tfidf.shape,   y_train.shape)
print(x_cv_tfidf.shape,   y_cv.shape)
print(x_test_tfidf.shape, y_test.shape)
```

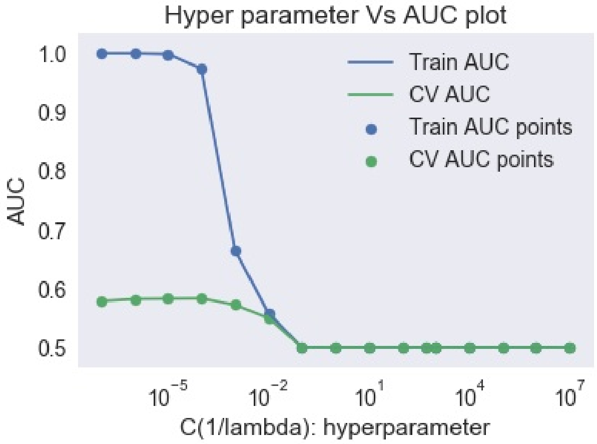
```
final matrix
(6733, 5895) (6733,)
(3317, 5895) (3317,)
(4950, 5895) (4950,)
```

Hyper parameter tuning

Grid search

In [77]:

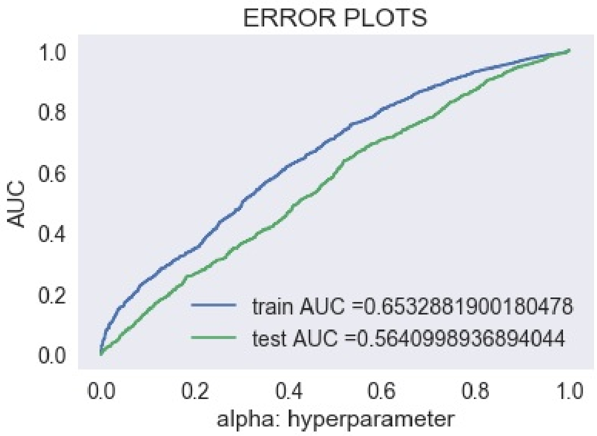
```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
import warnings
warnings.filterwarnings('ignore')
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
#from sklearn.grid_search import GridSearchCV
from sklearn.model_selection import learning_curve, GridSearchCV
from sklearn.model_selection import GridSearchCV
from sklearn import linear_model
from scipy.stats import randint as sp_randint
from sklearn.model_selection import RandomizedSearchCV


clf=linear_model.SGDClassifier(class_weight='balanced', penalty='l2'and'l1')
parameters ={'alpha':[10**-7,10**-6,10**-5,10**-4, 10**-3,10**-2,10**-1,1,10,100,500,1000,1000,10000,10**5,10**6,
10**7]}
sd = GridSearchCV(clf, parameters, cv=5, scoring='roc_auc',return_train_score=True)
sd.fit(x_tr_tfidf, y_train)



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

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

plt.plot(parameters['alpha'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
# plt.gca().fill_between(K, cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='darkorange')

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



plt.legend()
plt.xlabel("C(1/lambda): hyperparameter")
plt.ylabel("AUC")
plt.title("Hyper parameter Vs AUC plot")
plt.grid()
plt.show()
```



Testing the performance of the model on test data, plotting ROC Curves

In [79]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc
##https://stackoverflow.com/questions/39200265/attributeerror-probability-estimates-are-not-available-for-loss-hinge
from sklearn.calibration import CalibratedClassifierCV


lr = linear_model.SGDClassifier(alpha=10**-2,class_weight='balanced', loss='hinge')
clf =lr.fit(x_tr_tfidf, y_train)
calibrator = CalibratedClassifierCV(clf, cv='prefit')
model=calibrator.fit(x_tr_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 = batch_predict(model, x_tr_tfidf)
y_test_pred = batch_predict(model, x_test_tfidf)

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

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



confusion matrix for train

In [80]:

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

```
====================================================================================================
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.811
[[ 518  518]
 [1622 4075]]
```

In [81]:

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

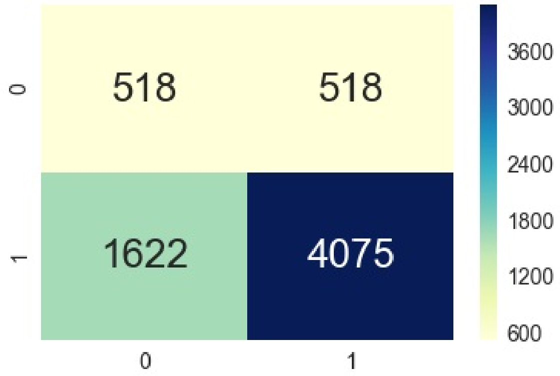
```
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.811
```

In [82]:

```
## Heatmaps -> https://likegeeks.com/seaborn-heatmap-tutorial/
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_train, annot=True,annot_kws={"size": 26}, fmt='g',cmap="YlGnBu")
```

Out[82]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x18bd0e48>
```



confusion matrix for test

In [83]:

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

```
====================================================================================================
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24999956831128556 for threshold 0.846
[[ 411  350]
 [1947 2242]]
```

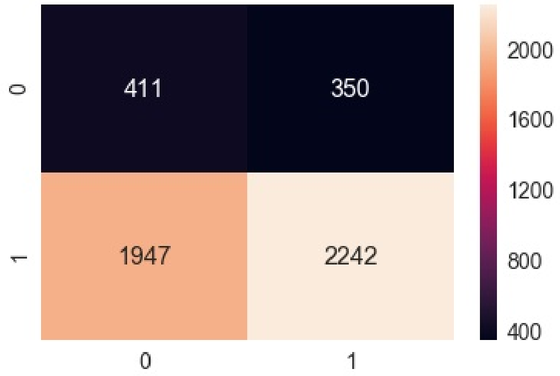
In [84]:

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

```
the maximum value of tpr*(1-fpr) 0.24999956831128556 for threshold 0.846
```

Out[84]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1d349208>
```



3.Applying SVM on categorical+numerical features + project_title(AVG W2V)+ preprocessed_eassay (AVG W2V)

AVG W2V featurization for essay

In [66]:

```
# stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-pickle-to-save-and-load-var
iables-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 [66]:

```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_vectors_train = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['preprocessed_essays'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_train.append(vector)

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

```
100%|████████████████████████████████████| 6733/6733 [00:03<00:00, 1876.95it/s]
```

```
6733
300
[ 3.34044599e-02  7.47895106e-03 -2.84866901e-03 -1.12112482e-01
  3.20094240e-02 -2.31917113e-02 -3.18244211e+00  2.69524296e-01
  7.81938775e-02  6.50705451e-02  2.89469465e-02  4.14666008e-02
  4.46101979e-02 -1.10078564e-01 -5.27422711e-02  2.23451674e-02
  2.44485538e-02 -8.41490211e-03  3.60374789e-02  2.95532007e-02
 -5.09585618e-02 -5.93073387e-02 -4.60782007e-02 -2.85732387e-02
 -7.04312324e-03 -2.41653155e-02  8.74828056e-02 -8.93268148e-02
 -5.96878099e-02 -3.76581930e-02 -2.57778603e-01 -3.81311831e-02
  4.58147762e-02  8.79267915e-02  4.53745704e-03 -1.10065001e-01
 -6.75504741e-02  4.13312408e-02 -2.92310056e-02 -2.72152396e-02
 -1.74138910e-02  5.64357303e-02  3.46134049e-02 -1.71740444e-01
  1.05775118e-01 -9.61490549e-02  3.63638894e-02  3.12050845e-02
  6.70239014e-03 -9.81891908e-02  2.29131063e-02  9.25448099e-03
 -7.89423197e-02 -4.77906479e-03  5.84926535e-02 -7.00010613e-02
  5.74876782e-02 -1.54653671e-02 -9.51435437e-02  9.11566979e-02
 -7.42487437e-02 -3.79449007e-02  2.08519765e-02  2.19803014e-02
 -3.79162775e-02  4.77148345e-02  1.89631871e-02 -5.17917831e-02
  1.45029535e-01 -1.60923527e-01 -1.22509611e-01 -6.00561282e-02
 -1.27450135e-02 -7.77205880e-02 -5.04890880e-02 -9.60622887e-02
 -9.99579225e-03  5.89434423e-03 -1.28683986e-02 -2.06090501e-02
  4.67861951e-02 -3.33577423e-01  2.32885394e-02  1.44626616e-02
 -1.31541294e-01  4.59369014e-02  1.11512018e-01 -1.29082063e-02
  1.27673098e-01 -2.34627746e-03  4.92748444e-02  3.35595218e-02
 -4.03831514e-02  8.93419472e-02  4.05974789e-03 -9.06464086e-02
 -2.28544408e+00  7.31850183e-02  1.29867172e-01  1.61595996e-01
 -1.00175422e-01  4.95162507e-02  2.05821369e-01 -5.56797662e-02
  1.05562202e-01 -4.77284472e-02 -5.75325331e-02 -1.35515394e-01
  9.12111409e-02  7.56667239e-02  6.85352817e-04 -5.28017063e-02
  3.80411810e-02  1.72324130e-01 -2.69972104e-02  5.37833169e-02
 -3.19156344e-01  2.09288246e-02  7.86083937e-02  2.32228046e-02
  3.74587127e-02  8.94276444e-02  1.11492724e-02 -7.90733683e-02
  8.98102704e-02 -1.45336451e-02  1.30675481e-01 -2.51825246e-02
 -7.07854345e-02  6.79568915e-02  9.65302796e-02 -1.54226408e-03
 -1.20659441e-02 -6.25855951e-02 -3.50958021e-02 -5.20276246e-02
  5.34826549e-02 -1.96853746e-02  2.17908563e-02  1.49123433e-01
  4.68952280e-02 -2.50410887e-02  2.99414669e-02 -4.73909746e-02
 -3.77286109e-02  1.25099109e-01  3.94272521e-02  5.60072535e-03
  1.21913225e-01 -1.18827243e-02 -3.70536641e-02 -1.18331639e-01
  3.27169318e-02 -1.68481148e-02  5.58811901e-03  1.25479697e-01
  5.56253803e-02 -2.50754394e-02 -4.40107394e-02 -9.67848645e-02
  1.19864711e-01 -3.29573479e-02 -2.89705937e-02 -1.82528592e-03
 -2.60042176e-02 -7.04178162e-02 -2.91844054e-02  6.02228429e-02
  4.81919268e-02 -3.30889000e-02 -1.00965282e-01 -1.70571521e-02
 -3.07338873e-03 -8.41552761e-02 -1.18052873e-03 -6.67774197e-02
  4.36864930e-02  6.14223169e-03 -1.45425013e-01 -4.96690563e-02
  3.89810915e-02  2.33009306e-01  3.28380021e-02 -2.66625134e-02
 -5.18648092e-02  1.85409676e-02 -3.19636946e-02  1.62550349e-02
 -2.05541549e-03  2.12273241e-02 -7.85004676e-02 -2.82907063e-02
 -1.26787514e-01 -5.08897556e-02  1.91789880e-02 -1.09966585e-01
 -6.70150831e-02  1.22426545e-01  4.54834085e-02  8.50794981e-02
  1.86434310e-01 -1.17256296e-02 -5.87890775e-03  2.05272810e-02
 -1.14325513e-01 -1.15840141e-03  1.02816606e-01 -1.14797184e-01
  6.88806406e-02 -1.92326854e-02 -1.60847535e-03 -3.51941908e-02
 -7.69139359e-02 -2.34827346e-01 -5.58373931e-02 -1.09452509e-02
 -8.21774972e-02 -7.81365315e-02 -3.25008425e-02  3.89394085e-03
 -1.61903917e-01 -3.33391766e-02 -1.26424304e-01 -1.23795162e-01
 -1.93120049e+00  1.07163796e-01  2.80501338e-03  4.43801756e-02
 -3.32830556e-02 -4.31186741e-02  3.18652127e-02 -7.93176599e-02
 -8.30265007e-02 -4.22516106e-02 -1.10745449e-01  4.23077887e-02
  3.91377768e-02 -7.25987923e-02  3.25440528e-02  1.22566296e-01
 -3.12363197e-02  2.75666683e-02 -2.28149038e-01 -1.53488521e-02
 -2.59690232e-02  3.90770373e-02 -2.99784859e-02 -7.70177520e-02
  1.95279380e-02  3.18714155e-03 -3.01962838e-02  1.08565958e-01
  6.85573514e-02 -1.00261315e-01  9.28285123e-02  5.72101451e-02
  9.30586094e-02 -4.42597113e-02  5.91464078e-02 -9.92403148e-02
  6.71392148e-02  3.62191951e-02  2.11434866e-03  2.54804930e-03
  5.53351627e-02 -7.13851887e-02 -6.93021094e-02 -2.80731952e-02
  6.43611908e-02  5.78538049e-02 -7.95336930e-02  9.75939930e-03
 -4.68014056e-02  8.43831831e-02  2.44354220e-02 -2.22171077e-02
 -7.47211197e-03  3.84262225e-02 -8.38632915e-02  1.04541442e-01
  1.77309965e-01 -8.20583113e-02 -2.19610239e-02 -3.14234599e-02
 -1.02945325e-02  1.56756001e-01 -4.29124007e-03  3.16629835e-02
  8.31802254e-03 -6.81043521e-02  4.81929215e-02 -3.78558254e-02
 -7.54411017e-02 -7.51194507e-02  9.92864648e-03  2.11655730e-02
 -8.55987845e-02  1.35649808e-01  1.31280673e-01 -3.63031339e-02]
```

In [68]:

```
avg_w2v_vectors_cv = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['preprocessed_essays'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_cv.append(vector)
```

```
100%|████████████████████████████████████| 3317/3317 [00:01<00:00, 1837.57it/s]
```

In [69]:

```
avg_w2v_vectors_test = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['preprocessed_essays'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_test.append(vector)
```

```
100%|████████████████████████████████████| 4950/4950 [00:02<00:00, 1785.88it/s]
```

AVG W2V featurization for title

In [70]:

```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_vectors_train_title = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['preprocessed_title'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_train_title.append(vector)

print(len(avg_w2v_vectors_train_title))
print(len(avg_w2v_vectors_train_title[0]))
print(avg_w2v_vectors_train_title[0])
```

```
100%|███████████████████████████████████| 6733/6733 [00:00<00:00, 24571.59it/s]
```

```
6733
300
[-3.12390000e-02 -1.26172429e-01 -3.99432857e-02 -1.20974857e-01
  3.47215100e-01 -1.13571857e-01 -2.28000429e+00  4.98011857e-01
  2.67578714e-01  1.23849571e-01  7.51697714e-02  8.14278571e-02
 -1.12765429e-01  6.27105714e-02 -8.46674286e-02 -1.84652571e-01
 -2.36832921e-01  1.80607714e-01 -1.06939714e-01  2.54296571e-01
 -2.67094571e-01 -2.34701429e-01  1.90609714e-02 -4.87562857e-02
  4.00775714e-01 -1.55872000e-01 -8.08205714e-02 -9.44108857e-02
  1.26308714e-01 -3.68094143e-01 -1.54739871e-01  1.30928571e-01
  2.25232857e-02  5.79412857e-02  1.76889571e-01 -1.91876371e-01
 -2.92881429e-02  1.19873000e-01 -5.33265714e-03  1.07480857e-01
 -3.86637143e-01 -6.10101429e-02 -7.07685714e-04 -9.73034286e-02
  3.65529000e-02 -2.00620000e-01  2.56227143e-02  4.42142857e-03
 -3.21041429e-01 -9.60914286e-02  1.81822857e-02 -3.68411429e-02
 -3.26777429e-01 -7.26274286e-02 -3.56571429e-03 -1.06658857e-01
  4.95948571e-02  7.57887143e-02 -8.30642857e-03  2.12598571e-02
 -3.46141429e-02  4.53208571e-02 -1.91804286e-01  1.44467857e-01
  1.42925714e-01 -2.41849429e-01  3.18611286e-02  1.03182857e-01
  9.05542857e-02 -2.10074714e-01  8.07092857e-02  6.88295714e-02
 -1.36152314e-01  1.35143857e-01 -2.16304286e-02 -1.73320414e-01
  5.10955714e-02 -4.07508571e-02  5.95558571e-02 -1.89846643e-01
 -2.74071429e-02  2.23141429e-01  1.25647571e-01 -1.55525714e-01
 -1.47247143e-01 -1.07534286e-01 -6.28948571e-02  3.36202429e-02
 -1.17735286e-01 -4.59994286e-02 -2.14337143e-02 -9.58972857e-02
 -2.95007143e-02  1.11701143e-01  6.55642857e-02  1.02166714e-01
 -1.97487857e+00  5.38465714e-02  1.75315000e-01  4.38727143e-01
 -1.29501429e-01  4.72531429e-02  4.65584000e-02 -1.86985571e-02
  1.52607286e-01 -1.67300857e-01 -2.34701857e-01 -1.60206000e-01
  1.53529714e-01 -1.52642857e-02  2.13638571e-01 -1.14040000e-01
  8.61645714e-02 -1.12963714e-01 -1.18860429e-01 -1.12758143e-01
 -6.84572857e-02 -1.39893857e-01  7.04387143e-02  3.71810000e-02
  2.85608000e-01  1.25825143e-01  1.70768714e-02 -1.62917143e-01
 -8.10715714e-02 -6.36045714e-02  9.22033714e-02  8.85357143e-03
 -2.78160000e-01 -1.24074943e-01 -1.57322429e-01  5.36190000e-02
 -1.98392143e-01 -5.88615714e-02 -3.23977143e-01 -2.63413000e-03
 -7.72832857e-02 -1.79601143e-01 -4.45081429e-02 -4.62200000e-02
 -1.37466386e-01 -2.79994857e-02  1.99523286e-01 -7.13839429e-02
 -1.54413143e-01  3.52019000e-01 -1.17265914e-01 -1.28002571e-01
  1.79490000e-01  1.42939000e-01  6.88601429e-02 -3.26553571e-01
  1.06130000e-01  1.45480000e-01  1.32083286e-01  7.33610000e-02
 -2.79998571e-02  4.51188571e-02 -8.96093143e-02  5.78698571e-02
 -3.43580000e-02 -7.05201429e-02 -7.02712857e-02  2.07551429e-02
  1.50618571e-01 -1.65349014e-01 -1.04503571e-01 -2.58681714e-01
 -2.02947143e-01 -2.07722857e-01 -8.23514286e-02 -3.04305714e-02
 -1.30346000e-01  6.97000000e-03 -1.62116529e-01 -3.49485714e-03
  2.19748571e-01 -1.94612714e-01  1.96077714e-01 -1.64788143e-01
 -1.50615429e-01  1.59530714e-01  9.56331429e-02  2.79260000e-01
  1.20257571e-01  1.46350857e-01 -1.74604571e-02  2.81060000e-01
 -2.09658286e-01 -1.24898571e-01 -6.04494286e-02  1.43448571e-01
 -3.31725714e-01  1.79344857e-01  1.20443000e-01 -3.60557143e-02
  5.09452043e-02  3.99431429e-02 -9.36331429e-02  1.91778571e-01
  5.69085714e-03 -1.27109600e-01  3.63852857e-02 -2.33376143e-01
  3.19118500e-01 -2.24571429e-04 -6.10461429e-02 -1.62709714e-01
  2.15164286e-02 -9.96285714e-04 -1.82283286e-01 -2.99951857e-01
 -9.46542857e-02 -3.89239143e-01 -1.89949571e-01  1.58415714e-02
 -2.25300543e-01  2.26298143e-01  2.23867857e-01 -1.81731429e-02
 -1.22566029e-01 -1.79151429e-01 -1.20378314e-01  2.22571429e-02
 -1.74209571e+00 -1.87992286e-01 -4.08922857e-02 -3.20063714e-01
 -1.62663286e-01 -1.07108857e-01 -7.88225714e-02  1.06185429e-01
  1.16381571e-01  3.01035714e-02 -2.31276714e-01  1.07167857e-01
  2.36373429e-02  6.06754286e-02  2.99655143e-01  9.20742857e-02
 -8.14694286e-02  1.69491543e-01  1.84785714e-02  1.30417143e-02
  4.79277143e-02 -6.99137143e-02  3.25000000e-03 -1.64521286e-01
  1.35285714e-01 -9.71940000e-02  1.68774857e-01  1.52960571e-01
 -8.21785714e-02 -6.38302857e-02  1.11470329e-01  1.35414714e-01
  1.52000571e-01  1.65743571e-01 -1.23138429e-01  7.59142857e-02
  1.27142714e-01  1.20602857e-01 -1.97462429e-01 -1.74465429e-02
  8.59244286e-02 -1.29447000e-01 -1.79813000e-01 -1.83992714e-01
  8.41072857e-02  1.24558286e-01 -2.09035571e-01 -2.70000000e-02
  6.76514286e-02  1.69486857e-01 -1.43360857e-01 -1.74068857e-01
 -2.45654286e-01  1.90546029e-01  1.40287714e-01  1.34360286e-01
  1.68684429e-01 -3.96942857e-03  2.65399429e-01 -2.79035714e-02
 -2.80245714e-02  1.11666843e-01  1.01870714e-01 -5.66105714e-02
 -8.09579143e-02 -4.09714286e-03  8.26141429e-02 -1.48474286e-01
 -1.95906714e-01 -2.87494286e-01 -1.64851429e-02  1.71265714e-02
 -9.94514286e-02 -1.25285714e-02  1.44624571e-01 -2.21658971e-01]
```

In [71]:

```
avg_w2v_vectors_cv_title = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['preprocessed_title'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_cv_title.append(vector)
```

```
100%|██████████████████████████████████████████| 3317/3317 [00:00<00:00, 25319.08it/s]
```

In [72]:

```
avg_w2v_vectors_test_title = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['preprocessed_title'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_test_title.append(vector)
```

```
100%|██████████████████████████████████████████| 4950/4950 [00:00<00:00, 26611.39it/s]
```

In [73]:

```
x_tr_avg_w2v=hstack((categorical_tr,numerical_tr,avg_w2v_vectors_train,avg_w2v_vectors_train_title)).tocsr()
x_cv_avg_w2v=hstack((categorical_cv,numerical_cv,avg_w2v_vectors_cv,avg_w2v_vectors_cv_title)).tocsr()
x_test_avg_w2v=hstack((categorical_test,numerical_test,avg_w2v_vectors_test,avg_w2v_vectors_test_title)).tocsr()


print('final matrix')
print(x_tr_avg_w2v.shape,  y_train.shape)
print(x_cv_avg_w2v.shape,  y_cv.shape)
print(x_test_avg_w2v.shape, y_test.shape)
```

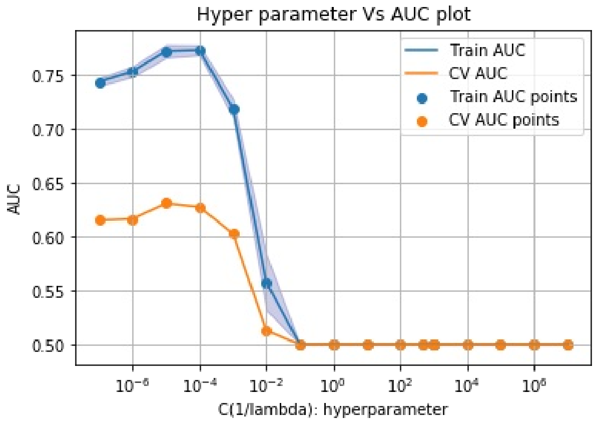
```
final matrix
(6733, 706) (6733,)
(3317, 706) (3317,)
(4950, 706) (4950,)
```

Hyper parameter tuning

Grid Search

In [77]:

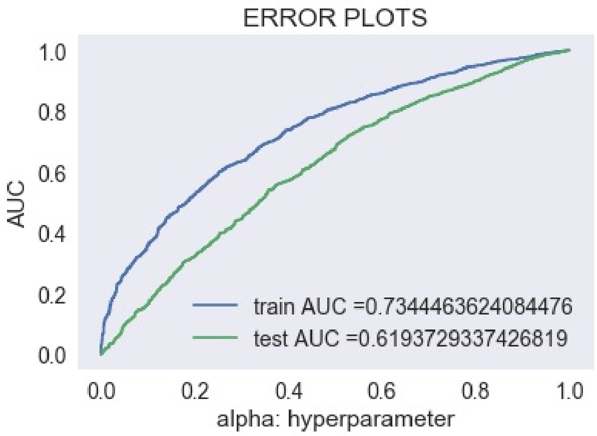
```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
import warnings
warnings.filterwarnings('ignore')
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
#from sklearn.grid_search import GridSearchCV
from sklearn.model_selection import learning_curve, GridSearchCV
from sklearn.model_selection import GridSearchCV
from sklearn import linear_model
from scipy.stats import randint as sp_randint
from sklearn.model_selection import RandomizedSearchCV


clf=linear_model.SGDClassifier(class_weight='balanced', penalty='l2'and'l1')
parameters ={'alpha':[10**-7,10**-6,10**-5,10**-4, 10**-3,10**-2,10**-1,1,10,100,500,1000,1000,10000,10**5,10**6,10**7]}
sd = GridSearchCV(clf, parameters, cv=5, scoring='roc_auc',return_train_score=True)
sd.fit(x_tr_avg_w2v, y_train)



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

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

plt.plot(parameters['alpha'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
# plt.gca().fill_between(K, cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='darkorange')

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



plt.legend()
plt.xlabel("C(1/lambda): hyperparameter")
plt.ylabel("AUC")
plt.title("Hyper parameter Vs AUC plot")
plt.grid()
plt.show()
```



Testing the performance of the model on test data, plotting ROC Curves

In [103]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc
##https://stackoverflow.com/questions/39200265/attributeerror-probability-estimates-are-not-available-for-loss-hi
nge
from sklearn.calibration import CalibratedClassifierCV


lr = linear_model.SGDClassifier(alpha=10**-3,class_weight='balanced', loss='hinge')
clf =lr.fit(x_tr_avg_w2v, y_train)
calibrator = CalibratedClassifierCV(clf, cv='prefit')
model=calibrator.fit(x_tr_avg_w2v, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
# not the predicted outputs

y_train_pred = batch_predict(model, x_tr_avg_w2v)
y_test_pred = batch_predict(model, x_test_avg_w2v)

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

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



confusion matrix for train

In [81]:

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

```
====================================================================================================
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.797
[[ 518  518]
 [1148 4549]]
```

In [82]:

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

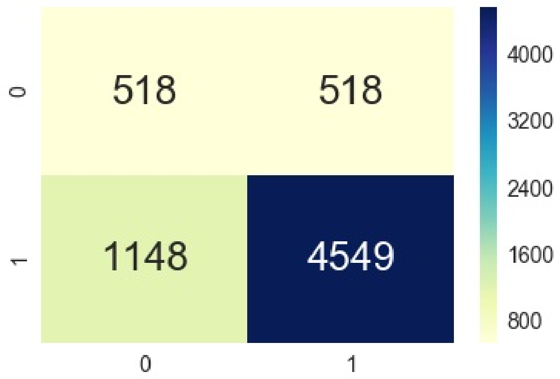
```
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.797
```

In [83]:

```
## Heatmaps -> https://likegeeks.com/seaborn-heatmap-tutorial/
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_train, annot=True,annot_kws={"size": 26}, fmt='g',cmap="YlGnBu")
```

Out[83]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x263e8d30>
```



confusion matrix for test

In [84]:

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

```
====================================================================================================
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24999956831128556 for threshold 0.838
[[ 391  370]
 [1371 2818]]
```

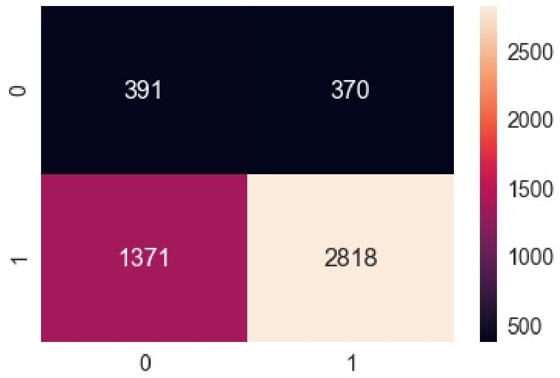
In [85]:

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

```
the maximum value of tpr*(1-fpr) 0.24999956831128556 for threshold 0.838
```

Out[85]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x167f32b0>
```



4.Applying Svm on categorical+numerical features + project_title(TFIDF W2V)+ preprocessed_eassay (TFIDF W2V)

TFIDF weighted W2V on essay

In [65]:

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

In [67]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_tr = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['essay']): # 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(sent
ence.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_tr.append(vector)

print(len(tfidf_w2v_vectors_tr))
print(len(tfidf_w2v_vectors_tr[0]))
```

```
100%|██████████████████████████████████████| 6733/6733 [01:05<00:00, 103.03it/s]

6733
300
```

In [68]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_cv = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['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((sentence.count(word)/len(sent
ence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf value for
each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_cv.append(vector)

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

```
100%|██████████████████████████████████████| 3317/3317 [00:15<00:00, 211.54it/s]

3317
300
```

In [69]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_test = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['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((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_test.append(vector)

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

```
100%|█████████████████████████████████████| 4950/4950 [00:21<00:00, 231.01it/s]

4950
300
```

TFIDF weighted W2V on title

In [70]:

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

In [71]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_tr_title = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['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_tr_title.append(vector)

print(len(tfidf_w2v_vectors_tr_title))
print(len(tfidf_w2v_vectors_tr_title[0]))
```

```
100%|███████████████████████████████████| 6733/6733 [00:00<00:00, 31025.88it/s]

6733
300
```

In [72]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_cv_title = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['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_cv_title.append(vector)

print(len(tfidf_w2v_vectors_cv_title))
print(len(tfidf_w2v_vectors_cv_title[0]))
```

```
100%|██████████████████████████████████████████████████| 3317/3317 [00:00<00:00, 30152.74it/s]

3317
300
```

In [73]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_test_title = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['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_test_title.append(vector)

print(len(tfidf_w2v_vectors_test_title))
print(len(tfidf_w2v_vectors_test_title[0]))
```

```
100%|██████████████████████████████████████████████████| 4950/4950 [00:00<00:00, 32563.96it/s]

4950
300
```

In [74]:

```
x_tr_tfidf_w2v=hstack((categorical_tr,numerical_tr,tfidf_w2v_vectors_tr,tfidf_w2v_vectors_tr_title)).tocsr()
x_cv_tfidf_w2v=hstack((categorical_cv,numerical_cv,tfidf_w2v_vectors_cv,tfidf_w2v_vectors_cv_title)).tocsr()
x_test_tfidf_w2v=hstack((categorical_test,numerical_test,tfidf_w2v_vectors_test,tfidf_w2v_vectors_test_title)).tocsr()

print("final matrix")
print(x_tr_tfidf_w2v.shape,   y_train.shape)
print(x_cv_tfidf_w2v.shape,   y_cv.shape)
print(x_test_tfidf_w2v.shape, y_test.shape)
```

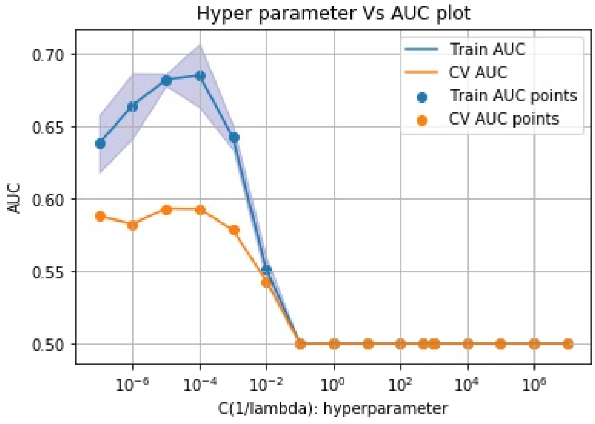
```
final matrix
(6733, 706) (6733,)
(3317, 706) (3317,)
(4950, 706) (4950,)
```

Hyper parameter tuning

Grid search

In [75]:

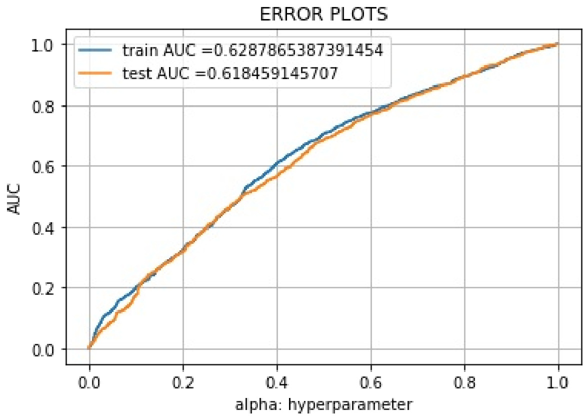
```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
import warnings
warnings.filterwarnings('ignore')
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
#from sklearn.grid_search import GridSearchCV
from sklearn.model_selection import learning_curve, GridSearchCV
from sklearn.model_selection import GridSearchCV
from sklearn import linear_model
from scipy.stats import randint as sp_randint
from sklearn.model_selection import RandomizedSearchCV


clf=linear_model.SGDClassifier(class_weight='balanced', penalty='l2'and'l1')
parameters ={'alpha':[10**-7,10**-6,10**-5,10**-4, 10**-3,10**-2,10**-1,1,10,100,500,1000,1000,10000,10**5,10**6,
10**7]}
sd = GridSearchCV(clf, parameters, cv=5, scoring='roc_auc',return_train_score=True)
sd.fit(x_tr_tfidf_w2v, y_train)



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

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

plt.plot(parameters['alpha'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
# plt.gca().fill_between(K, cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='darkorange')

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



plt.legend()
plt.xlabel("C(1/lambda): hyperparameter")
plt.ylabel("AUC")
plt.title("Hyper parameter Vs AUC plot")
plt.grid()
plt.show()
```



Testing the performance of the model on test data, plotting ROC Curves

In [85]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc
##https://stackoverflow.com/questions/39200265/attributeerror-probability-estimates-are-not-available-for-loss-hi
nge
from sklearn.calibration import CalibratedClassifierCV


lr = linear_model.SGDClassifier(alpha=10**-4,class_weight='balanced', loss='hinge')
clf =lr.fit(x_tr_tfidf_w2v, y_train)
calibrator = CalibratedClassifierCV(clf, cv='prefit')
model=calibrator.fit(x_tr_tfidf_w2v, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
# not the predicted outputs

y_train_pred = batch_predict(model, x_tr_tfidf_w2v)
y_test_pred = batch_predict(model, x_test_tfidf_w2v)

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

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



plotting confusion matrix

In [86]:

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

```
====================================================================================================
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.831
[[ 518  518]
 [1684 4013]]
```

In [87]:

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

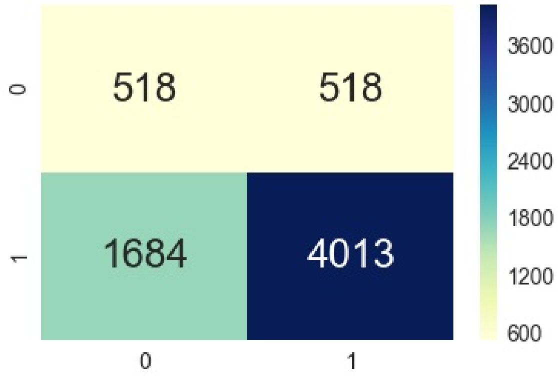
```
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.831
```

In [88]:

```
## Heatmaps -> https://likegeeks.com/seaborn-heatmap-tutorial/
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_train, annot=True,annot_kws={"size": 26}, fmt='g',cmap="YlGnBu")
```

Out[88]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x5749710>
```



confusion matrix for test

In [89]:

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

```
====================================================================================================
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24999956831128556 for threshold 0.849
[[ 556  205]
 [2421 1768]]
```

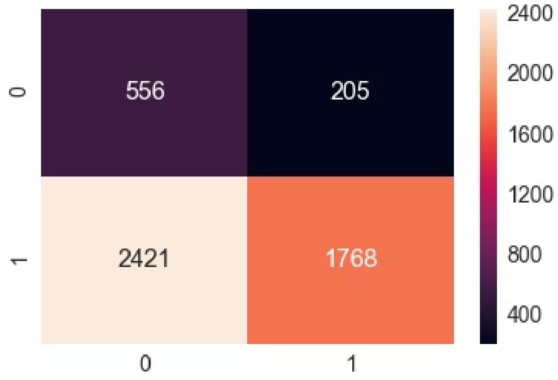
In [90]:

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

```
the maximum value of tpr*(1-fpr) 0.24999956831128556 for threshold 0.849
```

Out[90]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1daf42e8>
```



Applying SVM on categorical+numerical

applying elbow method on tfidfvectorizer of essays

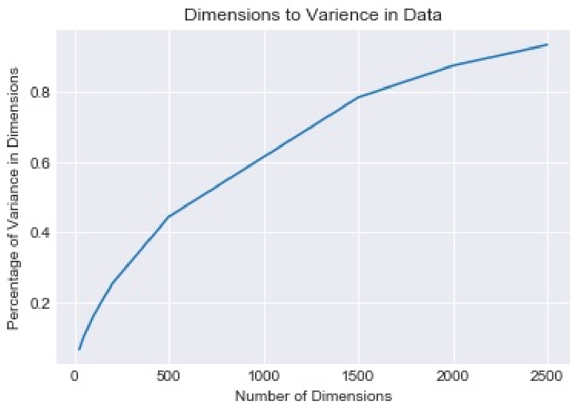
In [77]:

```
#Dimensions are very large so thats why i take less here.
X_train_tf_essay=X_train_essay_tfidf[:,0:4000]
X_cv_tf_essay=X_cv_essay_tfidf[:,0:4000]
X_test_tf_essay=X_test_essay_tfidf[:,0:4000]
from sklearn.decomposition import TruncatedSVD
#https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.TruncatedSVD.html
#declaring index as Dimensions in train_text_tfidf
Di = [25,50,100,200,500,1500,2000,2500]
Varience_sum = []
for i in tqdm(Di):
    svd = TruncatedSVD(n_components = i, random_state = 42)
    svd.fit(X_train_tf_essay)
    Varience_sum.append(svd.explained_variance_ratio_.sum())
```

```
100%|██████████████████████████████████████████████████████| 8/8 [05:20<00:00, 40.10s/it]
```

In [78]:

```
sns.set_style('darkgrid')
plt.xlabel("Number of Dimensions")
plt.ylabel("Percentage of Variance in Dimensions")
plt.title("Dimensions to Varience in Data")
plt.plot(Di,Varience_sum)
plt.show()
```



In [79]:

```
svd = TruncatedSVD(n_components= 2000)
svd.fit(X_train_tf_essay)
#Transforms:
#Train SVD
X_train_tf_essay= svd.transform(X_train_tf_essay )
#Test SVD
X_test_tf_essay = svd.transform(X_test_tf_essay )
#CV SVD
X_cv_tf_essay =  svd.transform(X_cv_tf_essay )
```

In [81]:

```
print(X_train_tf_essay.shape)
print(X_cv_tf_essay.shape)
print(X_test_tf_essay.shape)
```

```
(6733, 2000)
(3317, 2000)
(4950, 2000)
```

Merging all the features

In [86]:

```
x_tr_set5=hstack((categorical_tr,numerical_tr,X_train_tf_essay)).tocsr()
x_cv_set5=hstack((categorical_cv,numerical_cv,X_cv_tf_essay)).tocsr()
x_test_set5=hstack((categorical_test,numerical_test,X_test_tf_essay)).tocsr()

print("final matrix")
print(x_tr_set5.shape,   y_train.shape)
print(x_cv_set5.shape,   y_cv.shape)
print(x_test_set5.shape, y_test.shape)
```

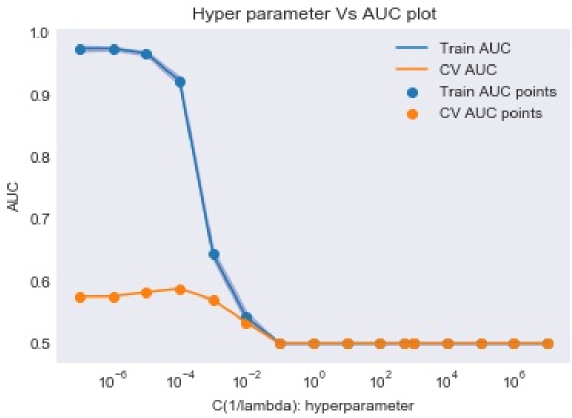
```
final matrix
(6733, 2106) (6733,)
(3317, 2106) (3317,)
(4950, 2106) (4950,)
```

Hyper parameter tuning

Grid search

In [87]:

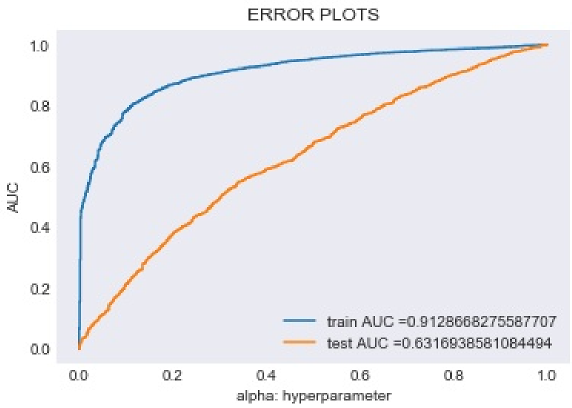
```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
import warnings
warnings.filterwarnings('ignore')
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
#from sklearn.grid_search import GridSearchCV
from sklearn.model_selection import learning_curve, GridSearchCV
from sklearn.model_selection import GridSearchCV
from sklearn import linear_model
from scipy.stats import randint as sp_randint
from sklearn.model_selection import RandomizedSearchCV


clf=linear_model.SGDClassifier(class_weight='balanced', penalty='l2'and'l1')
parameters ={'alpha':[10**-7,10**-6,10**-5,10**-4, 10**-3,10**-2,10**-1,1,10,100,500,1000,1000,10000,10**5,10**6,10**7]}
sd = GridSearchCV(clf, parameters, cv=5, scoring='roc_auc',return_train_score=True)
sd.fit(x_tr_set5, y_train)



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

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

plt.plot(parameters['alpha'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
# plt.gca().fill_between(K, cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='darkorange')

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



plt.legend()
plt.xlabel("C(1/lambda): hyperparameter")
plt.ylabel("AUC")
plt.title("Hyper parameter Vs AUC plot")
plt.grid()
plt.show()
```



Testing the performance of the model on test data, plotting ROC Curves

In [89]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc
##https://stackoverflow.com/questions/39200265/attributeerror-probability-estimates-are-not-available-for-loss-hinge
from sklearn.calibration import CalibratedClassifierCV


lr = linear_model.SGDClassifier(alpha=10**-4,class_weight='balanced', loss='hinge')
clf =lr.fit(x_tr_tfidf_w2v, y_train)
calibrator = CalibratedClassifierCV(clf, cv='prefit')
model=calibrator.fit(x_tr_set5, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
# not the predicted outputs

y_train_pred = batch_predict(model, x_tr_set5)
y_test_pred = batch_predict(model, x_test_set5)

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

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



Confusion matrix for train

In [90]:

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

```
====================================================================================================
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.24999906829057408 for threshold 0.523
[[ 517  519]
 [ 263 5434]]
```

In [91]:

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

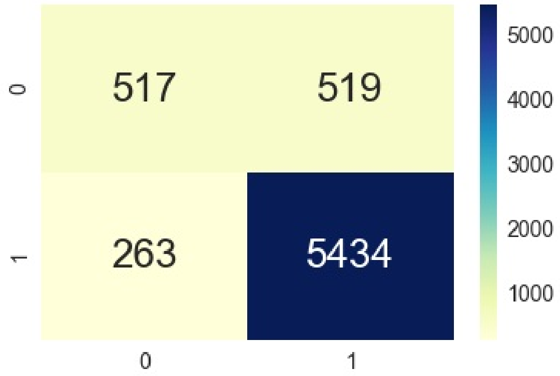
```
the maximum value of tpr*(1-fpr) 0.24999906829057408 for threshold 0.523
```

In [92]:

```
## Heatmaps -> https://likegeeks.com/seaborn-heatmap-tutorial/
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_train, annot=True,annot_kws={"size": 26}, fmt='g',cmap="YlGnBu")
```

Out[92]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x25522160>
```



Confysion matrix for text

In [93]:

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

```
====================================================================================================
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24999956831128556 for threshold 0.581
[[ 140  621]
 [ 371 3818]]
```

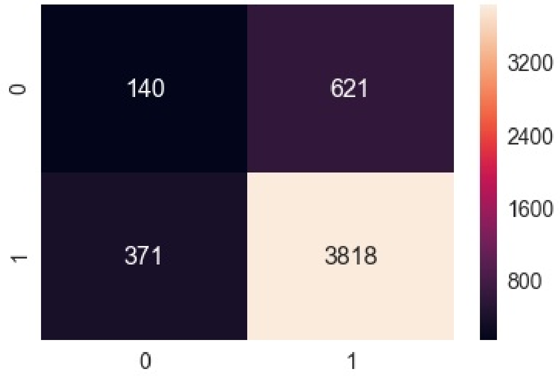
In [94]:

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

```
the maximum value of tpr*(1-fpr) 0.24999956831128556 for threshold 0.581
```

Out[94]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x252eae80>
```



Conclusion

In [87]:

```
#http://zetcode.com/python/prettytable/
from prettytable import PrettyTable

x = PrettyTable()
x.field_names = ["Vectorizer", "Hyperparameter", "trainAUC", "test AUC" ]
x.add_row(["Bag of Words",10**-2,0.9810,0.6092])
x.add_row(["TFIDF", 10**-2, 0.6525, 0.5640])
x.add_row(["Avgw2v",10**-3, 0.7344, 0.6193])
x.add_row(["Tfidfw2v", 10**-4, 0.6287, 0.6184])
x.add_row(["Truncated tfidf",10**-4, 0.9128, 0.6316 ])

print(x)
```

```
+-----------------+----------------+----------+----------+
|    Vectorizer   | Hyperparameter | trainAUC | test AUC |
+-----------------+----------------+----------+----------+
|   Bag of Words  |      0.01      |  0.981   |  0.6092  |
|      TFIDF      |      0.01      |  0.6525  |  0.564   |
|      Avgw2v     |     0.001      |  0.7074  |  0.6031  |
|     Tfidfw2v    |     0.001      |  0.7151  |  0.6098  |
| Truncated tfidf |     0.0001     |  0.9128  |  0.6316  |
+-----------------+----------------+----------+----------+
```

In [ ]: