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_3:__ "Close by sharing why your project will make a difference"

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

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

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

In [0]:
```
# Install the PyDrive wrapper & import libraries.
# This only needs to be done once per notebook.
!pip install -U -q PyDrive
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials

# Authenticate and create the PyDrive client.
# This only needs to be done once per notebook.
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)
```

```
  100% |████████████████████████████████| 993kB 17.8MB/s
Building wheel for PyDrive (setup.py) ... done
```

In [0]:
```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")

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

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from nltk.stem.porter import PorterStemmer

import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer

from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle

from tqdm import tqdm
import os

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

1.1 Reading Data

In [0]:
```
# Download a file based on its file ID.
#https://drive.google.com/file/d/1T48h84GLW3dpy9F6ble5nF_1gQxBO8rx/view?usp=sharing
file_id = '1T48h84GLW3dpy9F6ble5nF_1gQxBO8rx'
downloaded = drive.CreateFile({'id': file_id})
#print('Downloaded content "{}"'.format(downloaded.GetContentString()))
```

In [0]:
```
downloaded.GetContentFile('train_data.csv')
```

In [0]:
```
project_data = pd.read_csv('train_data.csv')
```

In [0]:
```
project_data.shape
project_data = project_data.sample(frac = 0.5)
```

In [0]:
```
print("Number of data points in train data", project_data.shape)
print('-'*50)
print("The attributes of data :", project_data.columns.values)
```

```
Number of data points in train data (54624, 17)
--------------------------------------------------
The attributes of data : ['Unnamed: 0' 'id' 'teacher_id' 'teacher_prefix' 'school_state'
 'project_submitted_datetime' 'project_grade_category'
 'project_subject_categories' 'project_subject_subcategories'
 'project_title' 'project_essay_1' 'project_essay_2' 'project_essay_3'
 'project_essay_4' 'project_resource_summary'
 'teacher_number_of_previously_posted_projects' 'project_is_approved']
```

In [0]:
```
# Download a file based on its file ID.
#https://drive.google.com/file/d/14OVXWu_SJU-lJD-jKMOCld14EZ21lYYe/view?usp=sharing
# A file ID looks like: laggVyWshwcyP6kEI-y_W3P8D26sz
#https://drive.google.com/file/d/14OVXWu_SJU-lJD-jKMOCld14EZ21lYYe/view?usp=sharing
file_id = '14OVXWu_SJU-lJD-jKMOCld14EZ21lYYe'
downloaded = drive.CreateFile({'id': file_id})
#print('Downloaded content "{}"'.format(downloaded.GetContentString()))
```

In [0]:
```
downloaded.GetContentFile('resources.csv')
```

In [0]:
```
resource_data = pd.read_csv('resources.csv')
resource_data = resource_data.sample(frac = 0.5)
```

In [0]:
```
print("Number of data points in train data", resource_data.shape)
print(resource_data.columns.values)
resource_data.head(2)
```

```
Number of data points in train data (770636, 4)
['id' 'description' 'quantity' 'price']
```

Out[0]:

	id	description	quantity	price
51061	p174857	MONOPOLY Property Trading Game	1	19.51
655971	p017895	Sony HD Video Recording HDRCX405 Handycam Camc...	1	219.90

1.2 preprocessing of project_subject_categories

In [0]:
```
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 'The')
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"Math & Science"=>"Math&Science"
        temp+=j.strip()+" " #" abc ".strip() will return "abc", remove the trailing spaces
        temp = temp.replace('&','_') # we are replacing the & value into
    cat_list.append(temp.strip())

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


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

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

1.3 preprocessing of project_subject_subcategories

In [0]:
```
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 'The')
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"Math & Science"=>"Math&Science"
        temp +=j.strip()+" "#" abc ".strip() will return "abc", remove the trailing spaces
        temp = temp.replace('&','_')
    sub_cat_list.append(temp.strip())

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

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

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

1.3 Text preprocessing

In [0]:
```
# 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 [0]:
```
project_data.head(2)
```

Out[0]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime	project_grade_category	project_title	project
15187	155785	p164955	5b753ad3c62afaa0e123a739366beede	Ms.	LA	2016-05-13 19:28:02	Grades 9-12	Creating a love of learning	Stude has and
67133	84109	p037386	fc4bde4b5265aff022e2019a89e59c8a	Ms.	WI	2016-08-10 15:52:43	Grades 3-5	We've Got to Move It, Move It!	M co varie

In [0]:
```
#### 1.4.2.3 Using Pretrained Models: TFIDF weighted W2V
```

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

```
Student learning has changed, and the way I teach must target these changes. While our students are Digital Natives - they have grow
n up with technology - schools have not been able to meet the growing demand of providing teachers with more technology to facilitat
e more engaged learning.My students are 11th graders from a variety of ethnic and economic backgrounds, with over 30% being free or
reduced lunch. While the school is new - only 8 years old - funding constraints have not allowed me to meet all of their needs. Many
of my students do not have computers at home, and I do not have student computers in the classroom. Our school is also limited to on
ly 2 computer labs which makes incorporating technology into my curriculum quite difficult. I have to share the 2 labs with the enti
re school, and I am lucky to get into the computer lab 3 or 4 times in the entire year. Because my students are more tech-savvy than
those of 20 years ago, I would like to incorporate interactive websites and other activities to enhance learning. I also teach stude
nts of various skill levels in one room, and being able to differentiate my curriculum to meet their specific needs is vital to allo
w each student to get what they need to be successful which means using any means available.In my master's program that I am current
ly taking, I have learned about many interactive websites for students to practice critical reading and thinking; brain-based games
to teach specific skills including research, writing, and grammar; supplemental reading outside of the textbook; proper research ski
lls; the ability to listen to speeches of presidents, civil rights leaders, and others; and the ability to corroborate information u
sing many sources not just one textbook. The use of technology will also facilitate the ease of cross-curricular learning in all sub
jects where one topic like reading the Great Gatsby can be used to teach history, science, and math. It will also afford me the abil
ity to access virtual tours of places many of my students may never get to visit without building this mental picture that could ins
pire them to keep studying and maybe get there one day. Basically, students live and breathe technology every day, and school should
capitalize on this.I have felt for quite some time that teaching methods have not kept up with the changing world. Many teachers use
antiquated methods, and we are losing kids to boredom or dropouts. I want to change that by sparking interest in my students using t
echnology and a fresh approach to create a love of learning so that students become life-long learners, not just forced learners. Ed
ucation is the greatest gift we give to ourselves, but I need to make learning desirable once again to all of my students.
--------------------------------------------------
\r\nMy students are a diverse group of middle-schoolers with a range of abilities from English Learners, students with special needs
and learning disabilities, to honors students. Specifically, my school serves over 717 students, of which 46.2% reside in poverty. I
also teach 50% of our 7th grade English Learners in my English Language Arts classes.\r\n\r\nOne thing my students all have in commo
n is that they need to have access to high quality technology to successfully navigate our new online curriculum.  Many of my studen
ts do not have computers at home; and so naturally, my students rely heavily on the technology provided at school. \r\n\r\nUnfortuna
tely, there just are not enough computers available for all students to have daily access to technology.  In order for my students t
o successfully navigate our new online curriculum, they need daily access to the computers in my classroom.My classroom needs four n
ew Chromebooks to allow for students to access our online curriculum.  Currently, my school shares four Chromebook carts among 38 te
achers.  Often times, it is the struggling students who need more time to complete tasks online, and this project will give my stude
nts the gift of time and technology to successfully complete their online learning.\r\n\r\nThis project will make a difference becau
se I will have four new Chromebooks that will permanently stay in my classroom to serve my students.  This will allow kids who need
to finish projects the opportunity to do so without feeling rushed.  They will have an opportunity to put forth their best work with
the help of these new Chromebooks.\r\n\r\nnannan
==================================================
Our elementary school has a high poverty status and is home to over 400 students. Twenty-five of these students are in my kindergart
en classroom.  Regardless of our socioeconomic status, these students are young, ambitious, and excited to learn. Kindergarten stude
nts are overjoyed with new experiences and enjoy new classroom tools. I want to provide my students with the educational experience
that they deserve. When I give them 100%, they always give 110%! It is an honor to lead them and to learn alongside them.It's time f
or teachers to start empowering students through healthy living. Last year, our school received a grant to teach our students about
healthy eating and the positive outcomes of physical movement. I asked my students to think about how we could continue this healthy
initiative into the next school year. Their response? \"Just keeping doing it!\" This kid-inspired project will accomplish their goa
l!\r\n\r\nMy students want more movement breaks throughout the school day. In fact, they requested the items in this project to make
it happen. They also expressed the need for having a healthy breakfast and snacks to keep our minds powered all day long. \r\n\r\nPe
r my student's request, we will incorporate more movement breaks by adding an Xbox One with Kinect to our classroom. My students ask
ed for games where they can simulate sports activities and dance moves. According to one student, Just Dance is \"the funniest game
in the world.\" That enthusiasm will get my kids physically active to meet the recommendations for sixty minutes of daily activity.
My students also understand the importance of fueling our bodies with a daily breakfast and healthy snacks. My students are requesti
ng cereal bars, oatmeal, and cereal bowls to ensure we have the energy we need to move and groove. All of the items in this project
will help us establish a healthy lifestyle for many years to come.\r\n\r\nnannan
==================================================
Hello and thanks for taking the time to read about our project. \r\n\r\n\r\nMy students are inquisitive and are in need of technolog
y that will support student to student discourse in the science classroom.\r\n\r\n\r\nOur school has over 75% free or reduced lunch
and breakfast.  In addition, the district has qualified for universal breakfast program.  \r\n\r\n\r\nIn spite of the challenges tha
t my students face, they come to school with the expectation of learning and improving their lives through advancing their educatio
n.  Maintaining their buy-in and interest in school is vital to ensure their success in making good choices that will lead to divers
e career and college ready opportunities that will shape their future.  In addition, it is our intention to increase student engagem
ent through discovery, inquiry and phenomenon based learning.  \r\nStudent discourse in the science classroom is extremely importan
t.   The iPads, when used in conjunction with the SWIVL C bot, will be used to record video and audio of students having scientific
conversations within cooperative learning opportunities and the inquiry process.  Once the video and audio is recorded it will be us
ed to analyze the level of scientific discourse.  The data collected will be used to enhance the career and college readiness of our
students as the develop the skills needed to improve the level of scientific discourse within the small groups. \r\n\r\n\r\nThese fi
les will then be posted to our district's online platform for students; turning science experimentation and discovery into an anytim
e anywhere experience. \r\n\r\n\r\nThe SWIVL C bot's markers serve as microphones for the iPad mini and will be strategically locate
d at each of the lab benches to record all of the scientific conversations that are taking place within the small groups.  Audio can
then be incorporated into the video to produce a powerful learning tool that can help guide instruction.   \r\n\r\n\r\nWith these to
ols, we will create sample videos that illustrate the progression of student scientific discourse throughout the school year.  It is
our intent to see an improvement in the level of scientific discourse as the year progresses. \r\nnannan
==================================================
```

In [0]:
```
# 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 [0]:
```
sent = decontracted(project_data['essay'].values[20000])
print(sent)
print("="*50)
```

```
Hello and thanks for taking the time to read about our project. \r\n\r\n\r\nMy students are inquisitive and are in need of technolog
y that will support student to student discourse in the science classroom.\r\n\r\n\r\nOur school has over 75% free or reduced lunch
and breakfast.  In addition, the district has qualified for universal breakfast program.  \r\n\r\n\r\nIn spite of the challenges tha
t my students face, they come to school with the expectation of learning and improving their lives through advancing their educatio
n.  Maintaining their buy-in and interest in school is vital to ensure their success in making good choices that will lead to divers
e career and college ready opportunities that will shape their future.  In addition, it is our intention to increase student engagem
ent through discovery, inquiry and phenomenon based learning.  \r\nStudent discourse in the science classroom is extremely importan
t.   The iPads, when used in conjunction with the SWIVL C bot, will be used to record video and audio of students having scientific
conversations within cooperative learning opportunities and the inquiry process.  Once the video and audio is recorded it will be us
ed to analyze the level of scientific discourse.  The data collected will be used to enhance the career and college readiness of our
students as the develop the skills needed to improve the level of scientific discourse within the small groups. \r\n\r\n\r\nThese fi
les will then be posted to our district is online platform for students; turning science experimentation and discovery into an anyti
me anywhere experience. \r\n\r\n\r\nThe SWIVL C bot is markers serve as microphones for the iPad mini and will be strategically loca
ted at each of the lab benches to record all of the scientific conversations that are taking place within the small groups.  Audio c
an then be incorporated into the video to produce a powerful learning tool that can help guide instruction.   \r\n\r\n\r\nWith these
tools, we will create sample videos that illustrate the progression of student scientific discourse throughout the school year.  It
is our intent to see an improvement in the level of scientific discourse as the year progresses. \r\nnannan
==================================================
```

In [0]:
```
# \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)
```

```
Hello and thanks for taking the time to read about our project.       My students are inquisitive and are in need of technology that
will support student to student discourse in the science classroom.      Our school has over 75% free or reduced lunch and breakfas
t.  In addition, the district has qualified for universal breakfast program.        In spite of the challenges that my students fac
e, they come to school with the expectation of learning and improving their lives through advancing their education.  Maintaining th
eir buy-in and interest in school is vital to ensure their success in making good choices that will lead to diverse career and colle
ge ready opportunities that will shape their future.  In addition, it is our intention to increase student engagement through discov
ery, inquiry and phenomenon based learning.    Student discourse in the science classroom is extremely important.   The iPads, when
used in conjunction with the SWIVL C bot, will be used to record video and audio of students having scientific conversations within
cooperative learning opportunities and the inquiry process.  Once the video and audio is recorded it will be used to analyze the lev
el of scientific discourse.  The data collected will be used to enhance the career and college readiness of our students as the deve
lop the skills needed to improve the level of scientific discourse within the small groups.       These files will then be posted to
our district is online platform for students; turning science experimentation and discovery into an anytime anywhere experience.
The SWIVL C bot is markers serve as microphones for the iPad mini and will be strategically located at each of the lab benches to re
cord all of the scientific conversations that are taking place within the small groups.  Audio can then be incorporated into the vid
eo to produce a powerful learning tool that can help guide instruction.        With these tools, we will create sample videos that
illustrate the progression of student scientific discourse throughout the school year.  It is our intent to see an improvement in th
e level of scientific discourse as the year progresses.   nannan
```

In [0]:
```
#remove spacial character: https://stackoverflow.com/a/5843547/4084039
sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
print(sent)
```

```
Hello and thanks for taking the time to read about our project My students are inquisitive and are in need of technology that will s
upport student to student discourse in the science classroom Our school has over 75 free or reduced lunch and breakfast In addition
the district has qualified for universal breakfast program In spite of the challenges that my students face they come to school with
the expectation of learning and improving their lives through advancing their education Maintaining their buy in and interest in sch
ool is vital to ensure their success in making good choices that will lead to diverse career and college ready opportunities that wi
ll shape their future In addition it is our intention to increase student engagement through discovery inquiry and phenomenon based
learning Student discourse in the science classroom is extremely important The iPads when used in conjunction with the SWIVL C bot w
ill be used to record video and audio of students having scientific conversations within cooperative learning opportunities and the
inquiry process Once the video and audio is recorded it will be used to analyze the level of scientific discourse The data collected
will be used to enhance the career and college readiness of our students as the develop the skills needed to improve the level of sc
ientific discourse within the small groups These files will then be posted to our district is online platform for students turning s
cience experimentation and discovery into an anytime anywhere experience The SWIVL C bot is markers serve as microphones for the iPa
d mini and will be strategically located at each of the lab benches to record all of the scientific conversations that are taking pl
ace within the small groups Audio can then be incorporated into the video to produce a powerful learning tool that can help guide in
struction With these tools we will create sample videos that illustrate the progression of student scientific discourse throughout t
he school year It is our intent to see an improvement in the level of scientific discourse as the year progresses nannan
```

In [0]:
```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
stopwords= ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"]
```

In [0]:
```
# Combining all the above stundents
from tqdm import tqdm
preprocessed_essays = []
# tqdm is for printing the status bar
for sentance in tqdm(project_data['essay'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e not in stopwords)
    preprocessed_essays.append(sent.lower().strip())
```

```
100%|██████████| 54624/54624 [00:35<00:00, 1539.01it/s]
```

In [0]:
```
# after preprocesing
preprocessed_essays[20000]
```

Out[0]:
```
'hello thanks taking time read project my students inquisitive need technology support student student discourse science classroom o
ur school 75 free reduced lunch breakfast in addition district qualified universal breakfast program in spite challenges students fa
ce come school expectation learning improving lives advancing education maintaining buy interest school vital ensure success making
good choices lead diverse career college ready opportunities shape future in addition intention increase student engagement discover
y inquiry phenomenon based learning student discourse science classroom extremely important the ipads used conjunction swivl c bot u
sed record video audio students scientific conversations within cooperative learning opportunities inquiry process once video audio
recorded used analyze level scientific discourse the data collected used enhance career college readiness students develop skills ne
eded improve level scientific discourse within small groups these files posted district online platform students turning science exp
erimentation discovery anytime anywhere experience the swivl c bot markers serve microphones ipad mini strategically located lab ben
ches record scientific conversations taking place within small groups audio incorporated video produce powerful learning tool help g
uide instruction with tools create sample videos illustrate progression student scientific discourse throughout school year it inten
t see improvement level scientific discourse year progresses nannan'
```

1.4 Preprocessing of `project_title`

In [0]:
```
# similarly you can preprocess the titles also
from tqdm import tqdm
preprocessed_titles = []
# 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_titles.append(sent.lower().strip())
```

```
100%|██████████| 54624/54624 [00:01<00:00, 30445.63it/s]
```

1.5 Preparing data for models

In [0]:
```
project_data.columns
```

Out[0]:
```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'project_submitted_datetime', 'project_grade_category', 'project_title',
       'project_essay_1', 'project_essay_2', 'project_essay_3',
       'project_essay_4', 'project_resource_summary',
       'teacher_number_of_previously_posted_projects', 'project_is_approved',
       'clean_categories', 'clean_subcategories', 'essay'],
      dtype='object')
```

we are going to consider

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

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

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

Assignment 5: Logistic Regression

1. **[Task-1] Logistic Regression(either SGDClassifier with log loss, or LogisticRegression) on these feature sets**
   - Set 1: categorical, numerical features + project_title(BOW) + preprocessed_eassay (`BOW with bi-grams` with `min_df=10` and `max_features=5000`)
   - Set 2: categorical, numerical features + project_title(TFIDF)+ preprocessed_eassay (`TFIDF with bi-grams` with `min_df=10` and `max_features=5000`)
   - Set 3: categorical, numerical features + project_title(AVG W2V)+ preprocessed_eassay (AVG W2V)
   - Set 4: categorical, numerical features + project_title(TFIDF W2V)+ preprocessed_essay (TFIDF W2V)
2. **Hyper paramter tuning (find best hyper parameters corresponding the algorithm that you choose)**
   - Find the best hyper parameter which will give the maximum AUC (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/receiver-operating-characteristic-curve-roc-curve-and-auc-1/) value
   - Find the best hyper paramter using k-fold cross validation or simple cross validation data
   - Use gridsearch cv or randomsearch cv or you can also write your own for loops to do this task of hyperparameter tuning
3. **Representation of results**
   - You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure.
   - Once after you found the best hyper parameter, you need to train your model with it, and find the AUC on test data and plot the ROC curve on both train and test.
   - Along with plotting ROC curve, you need to print the confusion matrix (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/confusion-matrix-tpr-fpr-fnr-tnr-1/) with predicted and original labels of test data points. Please visualize your confusion matrices using seaborn heatmaps.
   - (https://seaborn.pydata.org/generated/seaborn.heatmap.html)
   - (https://seaborn.pydata.org/generated/seaborn.heatmap.html)
   - (https://seaborn.pydata.org/generated/seaborn.heatmap.html)
4. **[Task-2] Apply Logistic Regression on the below feature set Set 5 by finding the best hyper parameter as suggested in step 2 and step 3.**
5. Consider these set of features Set 5 :
   - **school_state** : categorical data
   - **clean_categories** : categorical data
   - **clean_subcategories** : categorical data
   - **project_grade_category** :categorical data
   - **teacher_prefix** : categorical data
   - **quantity** : numerical data
   - **teacher_number_of_previously_posted_projects** : numerical data
   - **price** : numerical data
   - **sentiment score's of each of the essay** : numerical data
   - **number of words in the title** : numerical data
   - **number of words in the combine essays** : numerical data

   And apply the Logistic regression on these features by finding the best hyper paramter as suggested in step 2 and step 3

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

   (https://seaborn.pydata.org/generated/seaborn.heatmap.html)
   - You need to summarize the results at the end of the notebook, summarize it in the table format. To print out a table please refer to this prettytable library (https://seaborn.pydata.org/generated/seaborn.heatmap.html) link (http://zetcode.com/python/prettytable/)

Note: Data Leakage

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

In [0]:

2. Logistic Regression

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

In [0]:
```
# please write all the code with proper documentation, and proper titles for each subsection
# go through documentations and blogs before you start coding
# first figure out what to do, and then think about how to do.
# reading and understanding error messages will be very much helpfull in debugging your code
# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the reader
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label
```

In [0]:
```
from sklearn.model_selection import train_test_split
#splitting categorical data
# clean_categories
X = project_data
Y = project_data['project_is_approved']
X_train, X_test,Y_train, Y_test = train_test_split(X,Y,test_size = 0.25,random_state = 0 )
X_train_cv, X_test_cv,Y_train_cv, Y_test_cv  = train_test_split(X_train,Y_train,test_size = 0.25,random_state = 0)
```

In [0]:
```
#splitting Project_is_approved data
Y_train, Y_test = train_test_split(project_data['project_is_approved'].values,test_size = 0.25,random_state = 0,shuffle = False)
Y_train_cv, Y_test_cv = train_test_split(Y_train,test_size = 0.25,random_state = 0,shuffle = False)
```

2.2 Make Data Model Ready: encoding numerical, categorical features

In [0]:
```
# please write all the code with proper documentation, and proper titles for each subsection
# go through documentations and blogs before you start coding
# first figure out what to do, and then think about how to do.
# reading and understanding error messages will be very much helpfull in debugging your code
# make sure you featurize train and test data separatly

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

1.5.1 Vectorizing Categorical data

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

In [0]:
```
#categories
# we use count vectorizer to convert the values into one
from sklearn.feature_extraction.text import CountVectorizer
#vectorizer = CountVectorizer(vocabulary=list(sorted_cat_dict), lowercase=False, binary=True)
vectorizer = CountVectorizer(vocabulary=list(X_train), lowercase=False, binary=True)
categories_one_hot = vectorizer.fit_transform(X_train['clean_categories'])
print("Shape of matrix after one hot encodig ",categories_one_hot.shape)

categories_one_hot_te = vectorizer.transform(X_test['clean_categories'])
print("Shape of matrix after one hot encodig ",categories_one_hot_te.shape)


categories_one_hot_tecv = vectorizer.transform(X_test_cv['clean_categories'])
print("Shape of matrix after one hot encodig ",categories_one_hot_tecv.shape)
```

```
Shape of matrix after one hot encodig  (40968, 20)
Shape of matrix after one hot encodig  (30726, 20)
Shape of matrix after one hot encodig  (13656, 20)
Shape of matrix after one hot encodig  (10242, 20)
```

In [0]:

In [0]:
```
#subcategories
# we use count vectorizer to convert the values into one
vectorizer = CountVectorizer(vocabulary=list(X_train), lowercase=False, binary=True)
sub_categories_one_hot = vectorizer.fit_transform(X_train['clean_subcategories'])
print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encodig ",sub_categories_one_hot.shape)

sub_categories_one_hot_te = vectorizer.transform(X_test['clean_subcategories'])
print("Shape of matrix after one hot encodig ",sub_categories_one_hot_te.shape)

sub_categories_one_hot_tecv = vectorizer.transform(X_test_cv['clean_subcategories'])
print("Shape of matrix after one hot encodig ",sub_categories_one_hot_tecv.shape)
```

```
['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state', 'project_submitted_datetime', 'project_grade_category', 'projec
t_title', 'project_essay_1', 'project_essay_2', 'project_essay_3', 'project_essay_4', 'project_resource_summary', 'teacher_number_of
_previously_posted_projects', 'project_is_approved', 'clean_categories', 'clean_subcategories', 'essay', 'price', 'quantity']
Shape of matrix after one hot encodig  (40968, 20)
Shape of matrix after one hot encodig  (13656, 20)
Shape of matrix after one hot encodig  (10242, 20)
```

In [0]:
```
from collections import Counter
my_counter = Counter()
for word in X_train:
    if not isinstance(word, float):
      word = word.replace('.',' ')
      my_counter.update(word.split())

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

In [0]:
```
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(vocabulary=list(sorted_school_state_dict.keys()), lowercase=False, binary=True)


school_state_one_hot = vectorizer.transform(X_train['school_state'])
print("Shape of matrix after one hot encodig ",school_state_one_hot.shape)


school_state_one_hot_te = vectorizer.transform(X_test['school_state'])
print("Shape of matrix after one hot encodig ",school_state_one_hot_te.shape)

school_state_one_hot_tecv = vectorizer.transform(X_test_cv['school_state'])
print("Shape of matrix after one hot encodig ",school_state_one_hot_tecv.shape)
```

```
Shape of matrix after one hot encodig  (40968, 21)
Shape of matrix after one hot encodig  (13656, 21)
Shape of matrix after one hot encodig  (10242, 21)
```

In [0]:
```
print(project_data["teacher_prefix"])
```

```
15187         Ms.
67133         Ms.
28514        Mrs.
58874        Mrs.
7287         Mrs.
61997        Mrs.
97929         Ms.
20158         Ms.
27801        Mrs.
107101       Mrs.
20683        Mrs.
16949         Ms.
28210         Ms.
79027        Mrs.
18291        Mrs.
27946        Mrs.
99666        Mrs.
83360        Mrs.
96975         Mr.
57052         Ms.
105207        Ms.
38240        Mrs.
100752        Ms.
26947         Ms.
42467         Ms.
34229         Mr.
67171         Ms.
12578         Ms.
70928        Mrs.
90073         Ms.
            ...
42092        Mrs.
107963        Ms.
16674         Ms.
97265        Mrs.
74071        Mrs.
25553         Ms.
32414        Mrs.
43705        Mrs.
49633         Ms.
97923         Ms.
10146         Mr.
85533         Ms.
51870        Mrs.
70983         Ms.
14767         Mr.
101997       Mrs.
36450         Ms.
18472        Mrs.
102656        Mr.
93686         Ms.
9415          Mr.
67027         Mr.
63971         Ms.
58560        Mrs.
52480        Mrs.
14045     Teacher
62488        Mrs.
47193         Ms.
45463        Mrs.
59503         Ms.
Name: teacher_prefix, Length: 54624, dtype: object
```

In [0]:
```
from collections import Counter
my_counter = Counter()
for word in X_train:
    if not isinstance(word, float):
      word = word.replace('.',' ')
      my_counter.update(word.split())

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

In [0]:
```
##Vectorizing teacher_prefix
# we use count vectorizer to convert the values into one hot encoded features
#https://blog.csdn.net/ningzhimeng/article/details/80953916
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(vocabulary=list(sorted_teacher_prefix_dict.keys()), lowercase=False, binary=True)
vectorizer.fit(X_train['teacher_prefix'].astype('U'))
teacher_prefix_one_hot = vectorizer.transform(X_train['teacher_prefix'].astype("U"))
print("Shape of matrix after one hot encodig ",teacher_prefix_one_hot.shape)

teacher_prefix_one_hot_te = vectorizer.transform(X_test['teacher_prefix'].astype("U"))
print("Shape of matrix after one hot encodig ",teacher_prefix_one_hot_te.shape)

teacher_prefix_one_hot_tecv = vectorizer.transform(X_test_cv['teacher_prefix'].astype("U"))
print("Shape of matrix after one hot encodig ",teacher_prefix_one_hot_tecv.shape)
```

```
Shape of matrix after one hot encodig  (40968, 5)
Shape of matrix after one hot encodig  (13656, 5)
Shape of matrix after one hot encodig  (10242, 5)
```

In [0]:
```
from collections import Counter
my_counter = Counter()
for word in X_train['project_grade_category'].values:
    my_counter.update(word.split())
project_grade_category_dict = dict(my_counter)
sorted_project_grade_category_dict = dict(sorted(project_grade_category_dict.items(), key=lambda kv: kv[1]))
```

In [0]:
```
##Vectorizing project_grade_category
# we use count vectorizer to convert the values into one hot encoded features
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(vocabulary=list(sorted_project_grade_category_dict.keys()), lowercase=False, binary=True)
vectorizer.fit(X_train_pc)
project_grade_category_one_hot = vectorizer.transform(X_train['project_grade_category'])
print("Shape of matrix after one hot encodig ",project_grade_category_one_hot.shape)

project_grade_category_one_hot_te = vectorizer.transform(X_test['project_grade_category'])
print("Shape of matrix after one hot encodig ",project_grade_category_one_hot_te.shape)

project_grade_category_one_hot_tecv = vectorizer.transform(X_test_cv['project_grade_category'])
print("Shape of matrix after one hot encodig ",project_grade_category_one_hot_tecv.shape)
```

```
Shape of matrix after one hot encodig  (40968, 5)
Shape of matrix after one hot encodig  (13656, 5)
Shape of matrix after one hot encodig  (10242, 5)
```

In [0]:
```
```

1.5.3 Vectorizing Numerical features

In [0]:
```
price_data = resource_data.groupby('id').agg({'price':'sum', 'quantity':'sum'}).reset_index()
project_data = pd.merge(project_data, price_data, on='id', how='left')
```

In [0]:
```
```

In [0]:
```
# check this one: https://www.youtube.com/watch?v=0HOqOcln3Z4&t=530s
# standardization sklearn: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
from sklearn.preprocessing import Normalizer
# price_standardized = standardScalar.fit(project_data['price'].values)
# this will rise the error
# ValueError: Expected 2D array, got 1D array instead: array=[725.05 213.03 329.   ... 399.   287.73   5.5 ].
# Reshape your data either using array.reshape(-1, 1)
#normalized_X = preprocessing.normalize(X)
X_train['price'][np.isnan(X_train['price'])] = np.median(X_train['price'][~np.isnan(X_train['price'])])
Normalizer().fit(X_train['price'].reshape(-1,1))
price_normalized = Normalizer().transform(X_train['price'].reshape(-1,1))

X_test_cv['price'][np.isnan(X_test_cv['price'])] = np.median(X_test_cv['price'][~np.isnan(X_test_cv['price'])])
price_normalized_tecv= Normalizer().transform(X_test_cv['price'].reshape(-1,1))

X_test['price'][np.isnan(X_test['price'])] = np.median(X_test['price'][~np.isnan(X_test['price'])])
price_normalized_te= Normalizer().transform(X_test['price'].reshape(-1,1))

print(price_normalized.shape)

print(price_normalized_tecv.shape)
print(price_normalized_te.shape)
```

```
(40968, 1)
(10242, 1)
(13656, 1)
```

In [0]:
```
```

In [0]:
```
# check this one: https://www.youtube.com/watch?v=0HOqOcln3Z4&t=530s
# standardization sklearn: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
from sklearn.preprocessing import Normalizer
# price_standardized = standardScalar.fit(project_data['price'].values)
# this will rise the error
# ValueError: Expected 2D array, got 1D array instead: array=[725.05 213.03 329.   ... 399.   287.73   5.5 ].
# Reshape your data either using array.reshape(-1, 1)
#normalized_X = preprocessing.normalize(X)
X_train['quantity'][np.isnan(X_train['quantity'])] = np.median(X_train['quantity'][~np.isnan(X_train['quantity'])])
Normalizer().fit(X_train['quantity'].reshape(-1,1))
quantity_normalized = Normalizer().transform(X_train['quantity'].reshape(-1,1))


X_test_cv['quantity'][np.isnan(X_test_cv['quantity'])] = np.median(X_test_cv['quantity'][~np.isnan(X_test_cv['quantity'])])
quantity_normalized_tecv= Normalizer().transform(X_test_cv['quantity'].reshape(-1,1))

X_test['quantity'][np.isnan(X_test['quantity'])] = np.median(X_test['quantity'][~np.isnan(X_test['quantity'])])
quantity_normalized_te= Normalizer().transform(X_test['quantity'].reshape(-1,1))

print(quantity_normalized.shape)
print(quantity_normalized_tecv.shape)
print(quantity_normalized_te.shape)
```

```
(40968, 1)
(10242, 1)
(13656, 1)
```

2.3 Make Data Model Ready: encoding eassay, and project_title

In [0]:
```
# please write all the code with proper documentation, and proper titles for each subsection
# go through documentations and blogs before you start coding
# first figure out what to do, and then think about how to do.
# reading and understanding error messages will be very much helpfull in debugging your code
# make sure you featurize train and test data separatly

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

Bag of words

Bag of words

In [0]:
```
# We are considering only the words which appeared in at least 10 documents(rows or projects).
vectorizer_b = CountVectorizer()
text_bow = vectorizer_b.fit(X_train['essay'])
text_bow = vectorizer_b.transform(X_train['essay'])
print("Shape of matrix after one hot encodig ",text_bow.shape)

text_bow_te = vectorizer_b.transform(X_test['essay'])
print("Shape of matrix after one hot encodig ",text_bow_te.shape)

text_bow_tecv = vectorizer_b.transform(X_test_cv['essay'])
print("Shape of matrix after one hot encodig ",text_bow_tecv.shape)
```

```
Shape of matrix after one hot encodig  (40968, 41240)
Shape of matrix after one hot encodig  (13656, 41240)
Shape of matrix after one hot encodig  (10242, 41240)
```

In [0]:
```
#bow of Project_titles
```

In [0]:
```
vectorizer_t = CountVectorizer()
titles_bow = vectorizer_t.fit_transform(X_train['project_title'])
print("Shape of matrix after one hot encodig ",titles_bow.shape)

titles_bow_te = vectorizer_t.transform(X_test['project_title'])
print("Shape of matrix after one hot encodig ",titles_bow_te.shape)

titles_bow_tecv = vectorizer_t.transform(X_test_cv['project_title'])
print("Shape of matrix after one hot encodig ",titles_bow_tecv.shape)
```

```
Shape of matrix after one hot encodig  (40968, 11052)
Shape of matrix after one hot encodig  (13656, 11052)
Shape of matrix after one hot encodig  (10242, 11052)
```

combining data

In [0]:
```
%time
from scipy.sparse import hstack
#with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
x_train= hstack(( categories_one_hot,sub_categories_one_hot,teacher_prefix_one_hot,school_state_one_hot,quantity_normalized,project_grade_category_one_hot,text_bow,titles_bow,price_normalized)).tocsr()
#x_train = x_train.toarray()
#x_train[np.isnan(x_train)] = np.median(x_train[~np.isnan(x_train)])
x_train.shape
```

```
CPU times: user 10 µs, sys: 0 ns, total: 10 µs
Wall time: 14.3 µs
```

Out[0]: (40968, 52365)

In [0]:
```
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
x_test= hstack((categories_one_hot_te, sub_categories_one_hot_te,teacher_prefix_one_hot_te,school_state_one_hot_te,quantity_normalized_te,project_grade_category_one_hot_te,text_bow_te,titles_bow_te,price_normalized_te)).tocsr()
#x_test = x_test.toarray()
#x_test[np.isnan(x_test)] = np.median(x_test[~np.isnan(x_test)])
x_test.shape
```

Out[0]: (13656, 52365)

In [0]:
```
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
x_test_cv= hstack((categories_one_hot_tecv, sub_categories_one_hot_tecv,teacher_prefix_one_hot_tecv,quantity_normalized_tecv,school_state_one_hot_tecv,project_grade_category_one_hot_tecv,text_bow_tecv,titles_bow_tecv,price_normalized_tecv)).tocsr()
#x_test_cv= x_test_cv.toarray()
#x_test_cv[np.isnan(x_test_cv)] = np.median(x_test_cv[~np.isnan(x_test_cv)])
x_test_cv.shape
```

Out[0]: (10242, 52365)

In [0]:
```
print("Final Data matrix")
print(x_train.shape, Y_train.shape)
print(x_test_cv.shape, Y_test_cv.shape)
print(x_test.shape, Y_test.shape)
```

```
Final Data matrix
(40968, 52365) (40968,)
(10242, 52365) (10242,)
(13656, 52365) (13656,)
```

Set 1: categorical, numerical features + project_title(BOW) + preprocessed_eassay (BOW with bi-grams with min_df=10 and max_features=5000)

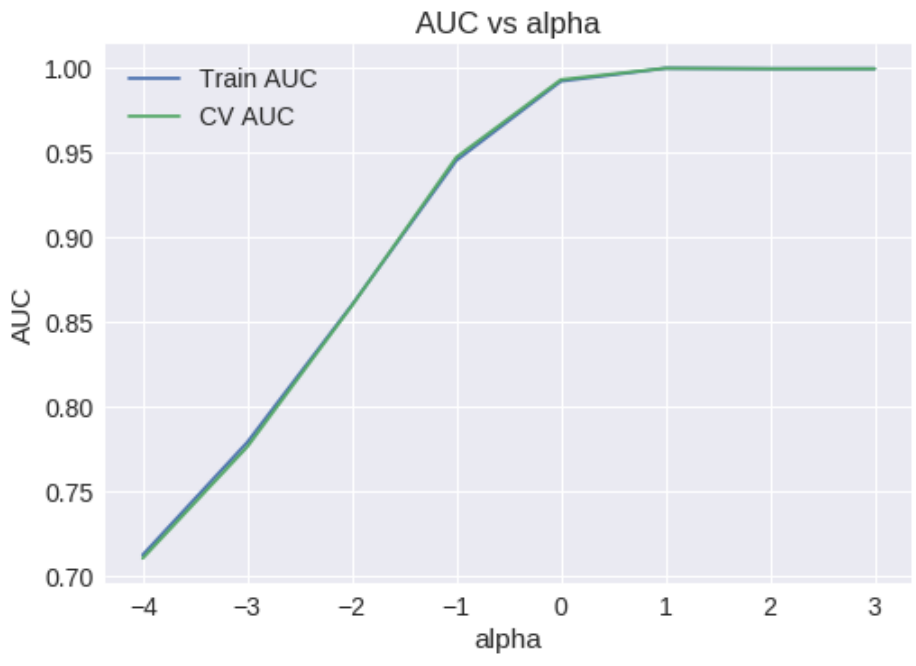
In [0]:
```
x_train_cv.shape
```

Out[0]:
```
(30726, 49356)
```

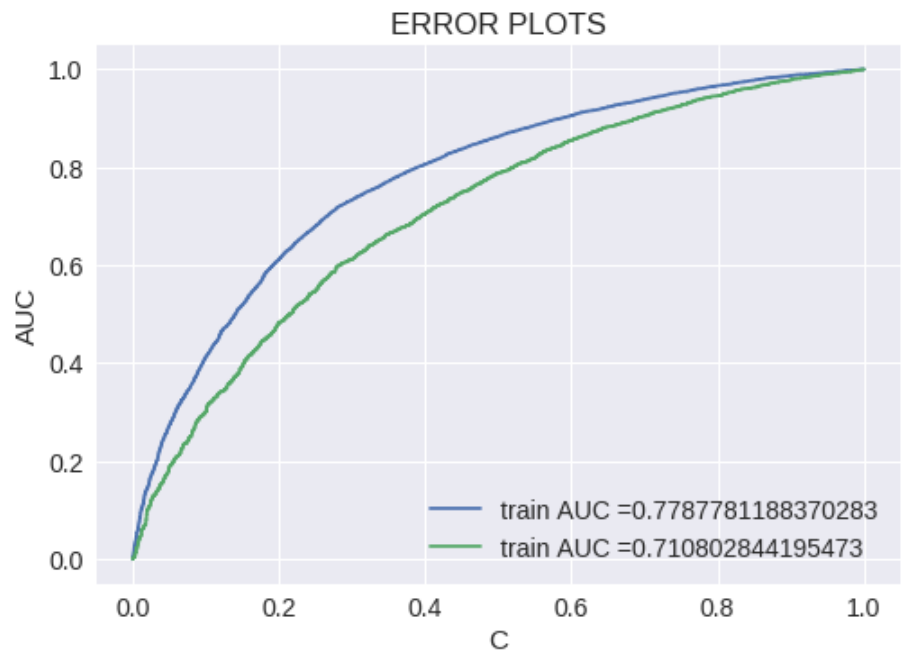
In [0]:
```
#https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html
#https://scikit-learn.org/stable/auto_examples/model_selection/plot_roc.html
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
from sklearn.metrics import roc_auc_score
import numpy as np
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score
from collections import Counter
from sklearn.metrics import accuracy_score
import random
import math
from sklearn import metrics
from sklearn.metrics import roc_auc_score
from sklearn.metrics import accuracy_score,confusion_matrix,f1_score,precision_score,recall_score

train_auc = []
cv_auc = []
C = [0.0001,0.001,0.01,0.1,1,10,100,1000 ]
for i in C:
    clf = LogisticRegression(C=i , class_weight = 'balanced')
    clf.fit(x_train, Y_train)
    y_train_pred = clf.predict_proba(x_train)[:,1]
    y_cv_pred = clf.predict_proba(x_test_cv)[:,1]
    train_auc_score = roc_auc_score(Y_train,y_train_pred)
    train_auc.append((train_auc_score))
    cv_auc.append(roc_auc_score(Y_test_cv, y_cv_pred))
    cv_auc_score=roc_auc_score(Y_test_cv, y_cv_pred)
    print("C",i,"cv:",cv_auc_score,"train:",train_auc_score)
```

```
C 0.0001 cv: 0.710245586822156 train: 0.712221350083654
C 0.001 cv: 0.7761155285729538 train: 0.7787781188370283
C 0.01 cv: 0.8594442154152776 train: 0.8598972581316002
C 0.1 cv: 0.9472744622922424 train: 0.9454031135299924
C 1 cv: 0.9931282105275645 train: 0.9922171354517144
C 10 cv: 0.9998901154307002 train: 0.9998802464982016
C 100 cv: 0.9996002040501584 train: 0.9996097191648058
C 1000 cv: 0.9995738821728969 train: 0.9996168249572338
```

In [0]:
```
a = [0.0001,0.001,0.01,0.1,1,10,100,1000]
log_a = [math.log10(num) for num in C]
plt.plot(log_a, train_auc, label='Train AUC')
plt.plot(log_a, cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("alpha")
plt.ylabel("AUC")
plt.title("AUC vs alpha")
plt.show()
```



In [0]:
```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc
clf = LogisticRegression(C =0.001,class_weight = 'balanced')
clf.fit(x_train, Y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of thepositive class
# not the predicted outputs
train_fpr, train_tpr, thresholds = roc_curve(Y_train, clf.predict_proba(x_train)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(Y_test, clf.predict_proba(x_test)[:,1])
plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="train AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("C")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
print("="*100)
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(Y_train, clf.predict(x_train)))
print("Test confusion matrix")
print(confusion_matrix(Y_test, clf.predict(x_test)))
```

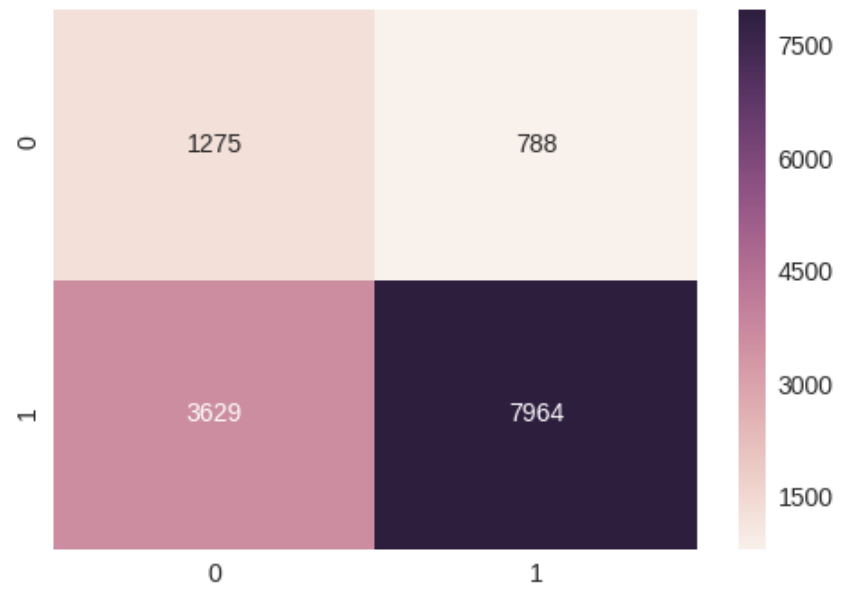



```
====================================================================================================
Train confusion matrix
[[ 4484  1624]
 [10455 24405]]
Test confusion matrix
[[1275  788]
 [3629 7964]]
```

In [0]:
```
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
from sklearn.metrics import recall_score
print('confusion matrix on test data')
y_new_pred  = clf.predict(x_test)
df_cm_bow = pd.DataFrame(confusion_matrix(Y_test, y_new_pred))
sns.set(font_scale=1.4)#for label size
sns.heatmap(df_cm_bow, annot=True,annot_kws={"size": 14}, fmt='g')
```

```
confusion matrix on test data
```

Out[0]: <matplotlib.axes._subplots.AxesSubplot at 0x7faffdad6358>



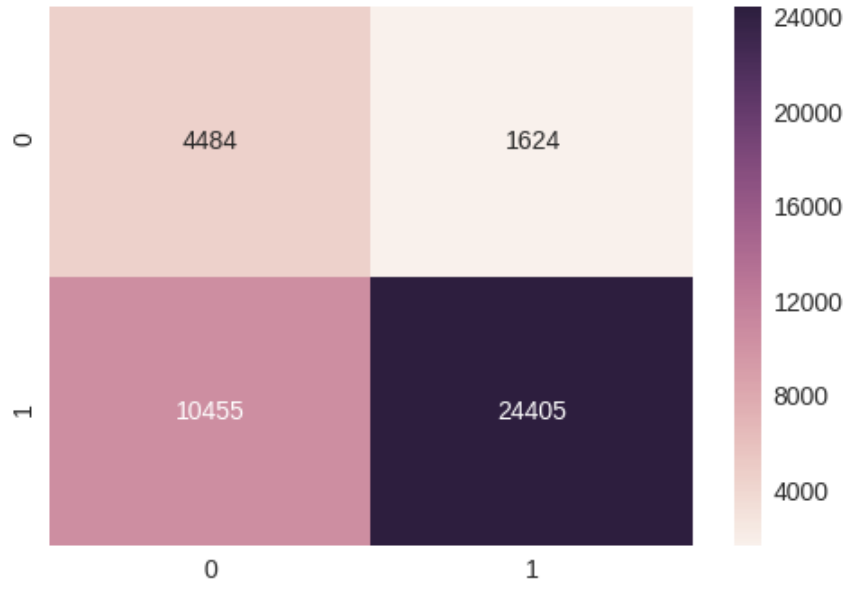
- **From the confusion matrix for test data we can say that,**

  7964+1275 = 9239 pouns are correctly classified and 3629+788 = 4417 points are wrongly classified

In [0]:
```
print("confusion matrix on train data")
y_new_pred_tr  = clf.predict(x_train)
df_cm_bow_tr = pd.DataFrame(confusion_matrix(Y_train, y_new_pred_tr))
sns.set(font_scale=1.4)#for label size
sns.heatmap(df_cm_bow_tr, annot=True,annot_kws={"size": 14}, fmt='g')
```

```
confusion matrix on train data
```

Out[0]: <matplotlib.axes._subplots.AxesSubplot at 0x7faffdcadf28>



- **From the confusion matrix for train data we can say that,**

  24405+4484 = 28889 pouns are correctly classified and 10455+1624 = 12079 points are wrongly classified

In [0]:
```
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
from sklearn.metrics import recall_score



print("Accuracy: %f%%"%(accuracy_score(Y_test, y_new_pred)*100))
print("Precision: %f"%(precision_score(Y_test, y_new_pred)))
print("Recall: %f"%(recall_score(Y_test, y_new_pred)))
print("F1-Score: %f"%(f1_score(Y_test, y_new_pred)))
```

```
Accuracy: 67.655243%
Precision: 0.909963
Recall: 0.686966
F1-Score: 0.782895
```

Encoding Project_essay and project_titles using TFIDF vectorizer and applying Multinomial Naive Bayes

In [0]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer_tfidf_b = TfidfVectorizer(min_df=10,max_features = 5000)
text_tfidf = vectorizer_tfidf_b.fit_transform(X_train['essay'])
print("Shape of matrix after one hot encodig ",text_tfidf.shape)
text_tfidf_te = vectorizer_tfidf_b.transform(X_test['essay'])
print("Shape of matrix after one hot encodig ",text_tfidf_te.shape)
text_tfidf_tecv = vectorizer_tfidf_b.transform(X_test_cv['essay'])
print("Shape of matrix after one hot encodig ",text_tfidf_tecv.shape)
```

```
Shape of matrix after one hot encodig  (40968, 5000)
Shape of matrix after one hot encodig  (13656, 5000)
Shape of matrix after one hot encodig  (10242, 5000)
```

In [0]:
```
# Similarly you can vectorize for title also
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer_tfidf_t = TfidfVectorizer(min_df=10, max_features =5000)
titles_tfidf = vectorizer_tfidf_t.fit_transform(X_train['project_title'])
print("Shape of matrix after one hot encodig ",titles_tfidf.shape)

titles_tfidf_te = vectorizer_tfidf_t.transform(X_test['project_title'])
print("Shape of matrix after one hot encodig ",titles_tfidf_te.shape)

titles_tfidf_tecv = vectorizer_tfidf_t.transform(X_test_cv['project_title'])
print("Shape of matrix after one hot encodig ",titles_tfidf_tecv.shape)
```

```
Shape of matrix after one hot encodig  (40968, 1873)
Shape of matrix after one hot encodig  (13656, 1873)
Shape of matrix after one hot encodig  (10242, 1873)
```

In [0]:
```

```

2.4.1 Combining all features,TFIDF SET 2

In [0]:
```
from scipy.sparse import hstack
#with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
x_train_tfidf= hstack(( categories_one_hot,sub_categories_one_hot,teacher_prefix_one_hot,school_state_one_hot,project_grade_category_one_hot,text_tfidf,titles_tfidf,price_normalized)).tocsr()
#x_train = x_train.toarray()
#x_train[np.isnan(x_train)] = np.median(x_train[~np.isnan(x_train)])
x_train_tfidf.shape
```

Out[0]: (40968, 6945)

In [0]:
```
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
x_test_tfidf= hstack((categories_one_hot_te, sub_categories_one_hot_te,teacher_prefix_one_hot_te,school_state_one_hot_te,project_grade_category_one_hot_te,text_tfidf_te,titles_tfidf_te,price_normalized_te)).tocsr()
#x_test = x_test.toarray()
#x_test[np.isnan(x_test)] = np.median(x_test[~np.isnan(x_test)])
x_test_tfidf.shape
```

Out[0]: (13656, 6945)

In [0]:
```
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
x_test_tfidf_cv= hstack((categories_one_hot_tecv, sub_categories_one_hot_tecv,teacher_prefix_one_hot_tecv,school_state_one_hot_tecv,project_grade_category_one_hot_tecv,text_tfidf_tecv,titles_tfidf_tecv,price_normalized_tecv)).tocsr()
#x_test_cv= x_test_cv.toarray()
#x_test_cv[np.isnan(x_test_cv)] = np.median(x_test_cv[~np.isnan(x_test_cv)])
x_test_tfidf_cv.shape
```

Out[0]: (10242, 6945)

In [0]:
```
print("Final Data matrix")
print(x_train_tfidf.shape, Y_train.shape)
print(x_test_tfidf_cv.shape, Y_test_cv.shape)
print(x_test_tfidf.shape, Y_test.shape)
```

```
Final Data matrix
(40968, 6945) (40968,)
(10242, 6945) (10242,)
(13656, 6945) (13656,)
```

In [0]:

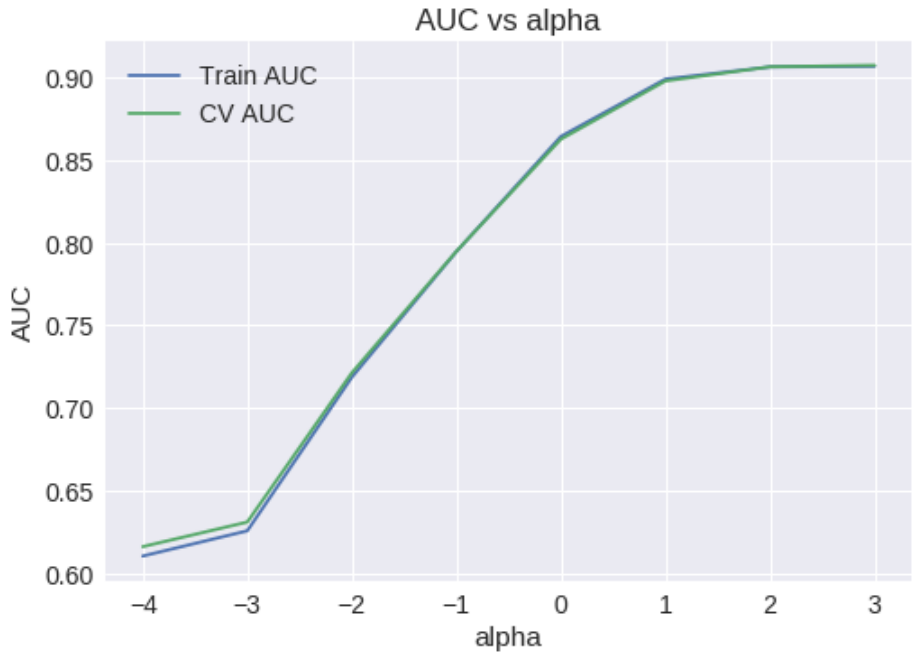
Set 2: categorical, numerical features + project_title(TFIDF)+ preprocessed_eassay (`TFIDF with bi-grams with min_df=10 and max_features=5000`)

In [0]:
```
#https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html
#https://scikit-learn.org/stable/auto_examples/model_selection/plot_roc.html
from sklearn.linear_model  import LogisticRegression
from sklearn import metrics
from sklearn.metrics import roc_auc_score
import numpy as np
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score
from collections import Counter
from sklearn.metrics import accuracy_score
import random
import math
from sklearn import metrics
from sklearn.metrics import roc_auc_score
from sklearn.metrics import accuracy_score,confusion_matrix,f1_score,precision_score,recall_score
train_auc = []
cv_auc = []
C = [0.0001,0.001,0.01,0.1,1,10,100,1000 ]
for i in C:
    clf = LogisticRegression(C=i , class_weight = 'balanced',penalty = 'l2')
    clf.fit(x_train_tfidf, Y_train)
    y_train_pred =  clf.predict_proba(x_train_tfidf)[:,1]
    y_cv_pred =  clf.predict_proba(x_test_tfidf_cv)[:,1]
    train_auc_score = roc_auc_score(Y_train,y_train_pred)
    train_auc.append((train_auc_score))
    cv_auc.append(roc_auc_score(Y_test_cv, y_cv_pred))
    cv_auc_score=roc_auc_score(Y_test_cv, y_cv_pred)
    print("C",i,"cv:",cv_auc_score,"train:",train_auc_score)
```

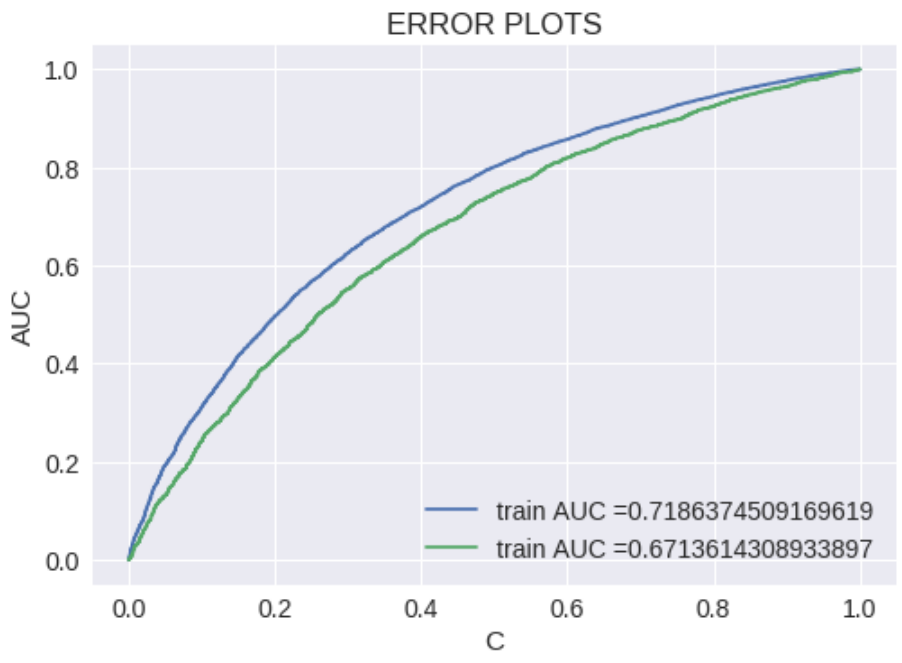
```
C 0.0001 cv: 0.6161491760881687 train: 0.6104608160399104
C 0.001 cv: 0.6310271167238086 train: 0.6257297644126886
C 0.01 cv: 0.7213181254671207 train: 0.7186374509169619
C 0.1 cv: 0.7948018370445944 train: 0.7946343846712512
C 1 cv: 0.8626526668881165 train: 0.8643105211565694
C 10 cv: 0.8978621445434377 train: 0.8990301743976562
C 100 cv: 0.9064524189434473 train: 0.9063657849660406
C 1000 cv: 0.9073304574519829 train: 0.9067927195732128
```

- **Here we took alpha values in the range of 0.0001 to 1000**

In [0]:
```
a = [0.0001,0.001,0.01,0.1,1,10,100,1000]
log_a = [math.log10(num) for num in C]
plt.plot(log_a, train_auc, label='Train AUC')
plt.plot(log_a, cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("alpha")
plt.ylabel("AUC")
plt.title("AUC vs alpha")
plt.show()
```



In [0]:
```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc

clf = LogisticRegression(C =0.01,class_weight = 'balanced')
clf.fit(x_train_tfidf, Y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
# not the predicted outputs
train_fpr, train_tpr, thresholds = roc_curve(Y_train, clf.predict_proba(x_train_tfidf)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(Y_test, clf.predict_proba(x_test_tfidf)[:,1])
plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="train AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("C")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
print("="*100)
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(Y_train, clf.predict(x_train_tfidf)))
print("Test confusion matrix")
print(confusion_matrix(Y_test, clf.predict(x_test_tfidf)))
```



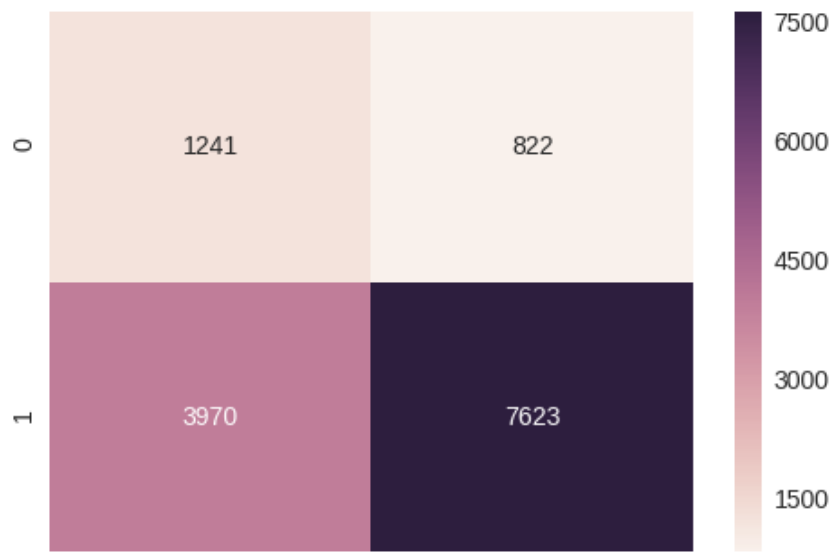

```
====================================================================================================
Train confusion matrix
[[ 4069  2039]
 [11798 23062]]
Test confusion matrix
[[1241  822]
 [3970 7623]]
```

- **here we got train accuracy as 71% and test accuracy as 67%**

In [0]:
```
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
from sklearn.metrics import recall_score
print("confusion matrix on test data")
y_tfidf_pred  = clf.predict(x_test_tfidf)
df_cm_bow = pd.DataFrame(confusion_matrix(Y_test, y_tfidf_pred))
sns.set(font_scale=1.4)#for label size
sns.heatmap(df_cm_bow, annot=True,annot_kws={"size": 14}, fmt='g')
```

```
confusion matrix on test data
```

Out[0]: <matplotlib.axes._subplots.AxesSubplot at 0x7faffe14b9e8>



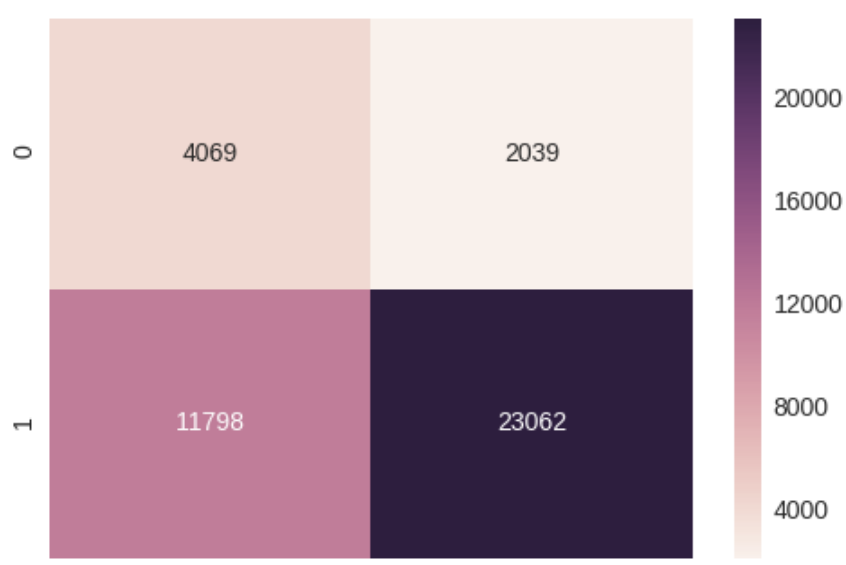
- **From the confusion matrix for test data we can say that,**

1241 + 7623 = 8864pouns are correctly classified and 3970+822 = 4792 points are wrongly classified

In [0]:
```
print("confusion matrix on train data")
y_new_pred_tr  = clf.predict(x_train_tfidf)
df_cm_bow_tr = pd.DataFrame(confusion_matrix(Y_train, y_new_pred_tr))
sns.set(font_scale=1.4)#for label size
sns.heatmap(df_cm_bow_tr, annot=True,annot_kws={"size": 14}, fmt='g')
```

```
confusion matrix on train data
```

Out[0]: <matplotlib.axes._subplots.AxesSubplot at 0x7faffdaf9d30>



- **From the confusion matrix for train data we can say that,**

  4069+23062 = 26431 pouns are correctly classified and 11798+2039 = 13837 points are wrongly classified

In [0]:
```
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
from sklearn.metrics import recall_score
print("Accuracy: %f%%"%(accuracy_score(Y_test, y_tfidf_pred)*100))
print("Precision: %f"%(precision_score(Y_test, y_tfidf_pred)))
print("Recall: %f"%(recall_score(Y_test, y_tfidf_pred)))
print("F1-Score: %f"%(f1_score(Y_test, y_tfidf_pred)))
```

```
Accuracy: 64.909197%
Precision: 0.902664
Recall: 0.657552
F1-Score: 0.760854
```

1.5.2.3 Using Pretrained Models: Avg W2V

In [0]:
```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3Aietf%3Awg%3Aoauth%3A2.0%3Aoob&scope=email%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdocs.test%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive.photos.readonly%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fpeopleapi.readonly&response_type=code

Enter your authorization code:
··········
Mounted at /content/gdrive
```

In [0]:
```
!cp "/content/gdrive/My Drive/glove.42B.300d.txt" "glove.42B.300d.txt"
```

In [0]:
```
# Reading glove vecors in python: https://stackoverflow.com/a/38230349/4084039
def loadGloveModel(gloveFile):
    print ("Loading Glove Model")
    f = open(gloveFile,'r', encoding="utf8")
    model = {}
    for line in tqdm(f):
        splitLine = line.split()
        word = splitLine[0]
        embedding = np.array([float(val) for val in splitLine[1:]])
        model[word] = embedding
    print ("Done.",len(model)," words loaded!")
    return model
model = loadGloveModel('glove.42B.300d.txt')
```

```
997it [00:00, 9964.12it/s]

Loading Glove Model
329739it [00:35, 9399.54it/s]

Done. 329739  words loaded!
```

In [0]:
```
words = []
for i in preprocessed_titles:
    words.extend(i.split(' '))
print("all the words in the coupus", len(words))
words = set(words)
print("the unique words in the coupus", len(words))

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

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

```
all the words in the coupus 236221
the unique words in the coupus 12369
The number of words that are present in both glove vectors and our coupus 11285 ( 91.236 %)
word 2 vec length 11285
```

In [0]:
```
# stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-pickle-to-save-and-load-variables-in-python/

import pickle
with open('glove.42B.300d.txt', 'wb') as f:
    pickle.dump(words_courpus, f)
```

In [0]:
```
# stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-pickle-to-save-and-load-variables-in-python/
# make sure you have the glove_vectors file
with open('glove.42B.300d.txt', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())
```

In [0]:
```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train_pe): # 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.append(vector)

print(len(avg_w2v_vectors))
print(len(avg_w2v_vectors[0]))
```

```
100%|██████████| 40968/40968 [00:12<00:00, 3210.65it/s]

40968
300
```

In [0]:
```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_vectors_trcv = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train_pecv): # for each review/sentence
    vector_trcv = np.zeros(300) # as word vectors are of zero length
    cnt_words_trcv =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_trcv += model[word]
            cnt_words_trcv += 1
    if cnt_words_trcv != 0:
        vector_trcv /= cnt_words_trcv
    avg_w2v_vectors_trcv.append(vector_trcv)

print(len(avg_w2v_vectors_trcv))
print(len(avg_w2v_vectors_trcv[0]))
```

```
100%|██████████| 30726/30726 [00:09<00:00, 3296.56it/s]

30726
300
```

In [0]:
```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_vectors_tecv = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test_pecv): # for each review/sentence
    vector_tecv = np.zeros(300) # as word vectors are of zero length
    cnt_words_tecv =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_tecv += model[word]
            cnt_words_tecv += 1
    if cnt_words_tecv != 0:
        vector_tecv /= cnt_words_tecv
    avg_w2v_vectors_tecv.append(vector_tecv)

print(len(avg_w2v_vectors_tecv))
print(len(avg_w2v_vectors_tecv[0]))
```

```
100%|██████████| 10242/10242 [00:03<00:00, 3271.20it/s]

10242
300
```

In [0]:
```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_vectors_te = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test_pe): # for each review/sentence
    vector_te = np.zeros(300) # as word vectors are of zero length
    cnt_words_te =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_te += model[word]
            cnt_words_te += 1
    if cnt_words_te != 0:
        vector_te /= cnt_words_te
    avg_w2v_vectors_te.append(vector_te)

print(len(avg_w2v_vectors_te))
print(len(avg_w2v_vectors_te[0]))
```

```
100%|██████████| 13656/13656 [00:04<00:00, 3253.22it/s]

13656
300
```

In [0]:
```
# average Word2Vec
# compute average word2vec for preprocessed_titles.
avg_w2v_vectors_titles = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train_pt): # for each review/sentence
    vector_titles = np.zeros(300) # as word vectors are of zero length
    cnt_words_titles =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_titles += model[word]
            cnt_words_titles += 1
    if cnt_words_titles != 0:
        vector_titles /= cnt_words_titles
    avg_w2v_vectors_titles.append(vector_titles)

print(len(avg_w2v_vectors_titles))
print(len(avg_w2v_vectors_titles[0]))
```

```
100%|██████████| 40968/40968 [00:00<00:00, 63703.44it/s]

40968
300
```

In [0]:
```
# average Word2Vec
# compute average word2vec for preprocessed_titles.
avg_w2v_vectors_titles_trcv = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train_ptcv): # for each review/sentence
    vector_titles_trcv = np.zeros(300) # as word vectors are of zero length
    cnt_words_titles_trcv =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_titles_trcv += model[word]
            cnt_words_titles_trcv += 1
    if cnt_words_titles_trcv != 0:
        vector_titles_trcv /= cnt_words_titles_trcv
    avg_w2v_vectors_titles_trcv.append(vector_titles_trcv)

print(len(avg_w2v_vectors_titles_trcv))
print(len(avg_w2v_vectors_titles_trcv[0]))
```

```
100%|██████████| 30726/30726 [00:00<00:00, 62885.03it/s]

30726
300
```

In [0]:
```
# average Word2Vec
# compute average word2vec for preprocessed_titles.
avg_w2v_vectors_titles_tecv = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test_ptcv): # for each review/sentence
    vector_titles_tecv = np.zeros(300) # as word vectors are of zero length
    cnt_words_titles_tecv =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_titles_tecv += model[word]
            cnt_words_titles_tecv += 1
    if cnt_words_titles_tecv != 0:
        vector_titles_tecv /= cnt_words_titles_tecv
    avg_w2v_vectors_titles_tecv.append(vector_titles_tecv)

print(len(avg_w2v_vectors_titles_tecv))
print(len(avg_w2v_vectors_titles_tecv[0]))
```

```
100%|██████████| 10242/10242 [00:00<00:00, 59405.40it/s]

10242
300
```

In [0]:
```
# average Word2Vec
# compute average word2vec for preprocessed_titles.
avg_w2v_vectors_titles_te = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test_pt): # for each review/sentence
    vector_titles_te = np.zeros(300) # as word vectors are of zero length
    cnt_words_titles_te =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_titles_te += model[word]
            cnt_words_titles_te += 1
    if cnt_words_titles_te != 0:
        vector_titles_te /= cnt_words_titles_te
    avg_w2v_vectors_titles_te.append(vector_titles_te)

print(len(avg_w2v_vectors_titles_te))
print(len(avg_w2v_vectors_titles_te[0]))
```

```
100%|██████████| 13656/13656 [00:00<00:00, 63432.46it/s]

13656
300
```

2.4.1 Combining all features,word 2 vec

In [0]:
```
from scipy.sparse import hstack
#with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
x_train_w2v= hstack(( categories_one_hot,sub_categories_one_hot,teacher_prefix_one_hot,school_state_one_hot,project_grade_category_one_hot,avg_w2v_vectors,avg_w2v_vectors_titles,price_normalized,normalized_tnpp)).tocsr()
#x_train = x_train.toarray()
#x_train[np.isnan(x_train)] = np.median(x_train[~np.isnan(x_train)])
x_train_w2v.shape
```

Out[0]: (40968, 702)

In [0]:
```
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
x_train_w2v_cv= hstack((categories_one_hot_cv, sub_categories_one_hot_cv,teacher_prefix_one_hot_cv,school_state_one_hot_cv,project_grade_category_one_hot_cv,avg_w2v_vectors_trcv,avg_w2v_vectors_titles_trcv,price_normalized_cv,normalized_tnppcv)).tocsr()
#x_train_cv = x_train_cv.toarray()
#x_train_cv[np.isnan(x_train_cv)] = np.median(x_train_cv[~np.isnan(x_train_cv)])
x_train_w2v_cv.shape
```

Out[0]: (30726, 702)

In [0]:
```
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
x_test_w2v= hstack((categories_one_hot_te, sub_categories_one_hot_te,teacher_prefix_one_hot_te,school_state_one_hot_te,project_grade_
category_one_hot_te,avg_w2v_vectors_te,avg_w2v_vectors_titles_te,price_normalized_te,normalized_tnppte)).tocsr()
#x_test = x_test.toarray()
#x_test[np.isnan(x_test)] = np.median(x_test[~np.isnan(x_test)])
x_test_w2v.shape
```

Out[0]: (13656, 702)

In [0]:
```
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
x_test_w2v_cv= hstack((categories_one_hot_tecv, sub_categories_one_hot_tecv,teacher_prefix_one_hot_tecv,school_state_one_hot_tecv,pro
ject_grade_category_one_hot_tecv,avg_w2v_vectors_tecv,avg_w2v_vectors_titles_tecv,price_normalized_tecv,normalized_tnpptecv)).tocsr()
#x_test_cv= x_test_cv.toarray()
#x_test_cv[np.isnan(x_test_cv)] = np.median(x_test_cv[~np.isnan(x_test_cv)])
x_test_w2v_cv.shape
```

Out[0]: (10242, 702)

In [0]:
```
print("Final Data matrix")
print(x_train_w2v.shape, Y_train.shape)
print(x_test_w2v_cv.shape, Y_test_cv.shape)
print(x_test_w2v.shape, Y_test.shape)
```

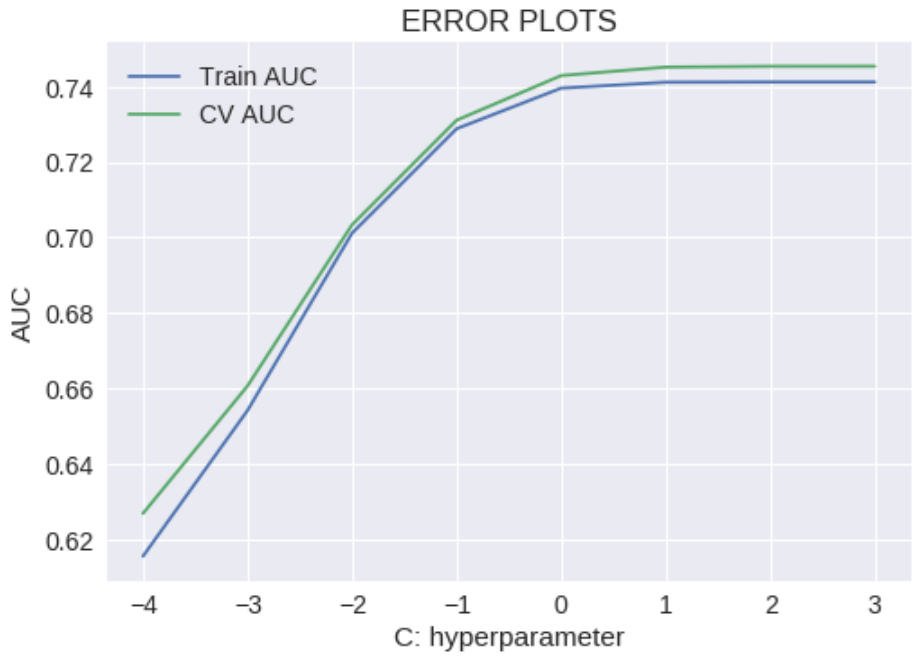
```
Final Data matrix
(40968, 702) (40968,)
(10242, 702) (10242,)
(13656, 702) (13656,)
```

Set 3: categorical, numerical features + project_title(AVG W2V)+ preprocessed_eassay (AVG W2V)

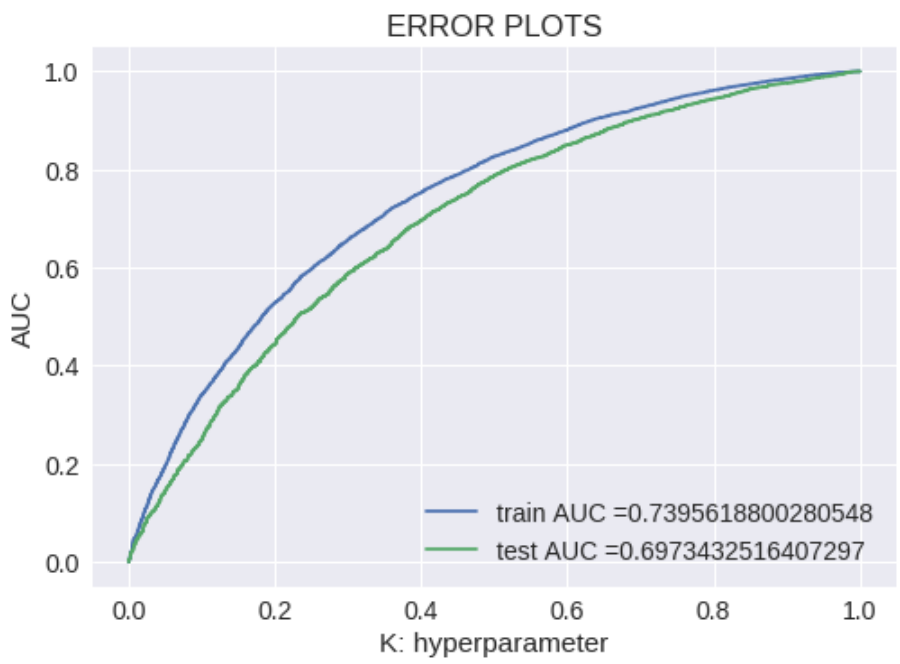
In [0]:
```
# Please write all the code with proper documentation
#finding the best hypermeter with bow train and cv data
#loading library's
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score
from collections import Counter
from sklearn.metrics import accuracy_score
import random
from sklearn import metrics
from sklearn.metrics import roc_auc_score

train_auc = []
cv_auc = []
C = [0.0001,0.001,0.01,0.1,1,10,100,1000]
for i in C:
    clss = LogisticRegression(C =i,class_weight = 'balanced')
    clss.fit(x_train_w2v, Y_train)
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
    # not the predicted outputs
    y_train_pred =  clss.predict_proba(x_train_w2v)[:,1]
    y_cv_pred =  clss.predict_proba(x_test_w2v_cv)[:,1]

    train_auc.append(roc_auc_score(Y_train,y_train_pred))
    cv_auc.append(roc_auc_score(Y_test_cv, y_cv_pred))
    cv_auc_score=roc_auc_score(Y_test_cv, y_cv_pred)
    print("C=",i,"cv:",cv_auc_score,"train:",train_auc)
```

```
C= 0.0001 cv: 0.6270216470783775 train: [0.6156980428993728]
C= 0.001 cv: 0.6607636663837845 train: [0.6156980428993728, 0.654267958602709]
C= 0.01 cv: 0.7034021124006518 train: [0.6156980428993728, 0.654267958602709, 0.7011937825022152]
C= 0.1 cv: 0.7310774531938351 train: [0.6156980428993728, 0.654267958602709, 0.7011937825022152, 0.7288231723435205]
C= 1 cv: 0.7428850217635383 train: [0.6156980428993728, 0.654267958602709, 0.7011937825022152, 0.7288231723435205, 0.739561880028054
8]
C= 10 cv: 0.7451390986985068 train: [0.6156980428993728, 0.654267958602709, 0.7011937825022152, 0.7288231723435205, 0.73956188002805
48, 0.7411253663473202]
C= 100 cv: 0.7453490686804969 train: [0.6156980428993728, 0.654267958602709, 0.7011937825022152, 0.7288231723435205, 0.7395618800280
548, 0.7411253663473202, 0.7411808012843549]
C= 1000 cv: 0.7453604010726002 train: [0.6156980428993728, 0.654267958602709, 0.7011937825022152, 0.7288231723435205, 0.739561880028
0548, 0.7411253663473202, 0.7411808012843549, 0.7411805018850274]
```

In [0]:
```
log_a = [math.log10(num) for num in C]
plt.plot(log_a, train_auc, label='Train AUC')
plt.plot(log_a, cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("C: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



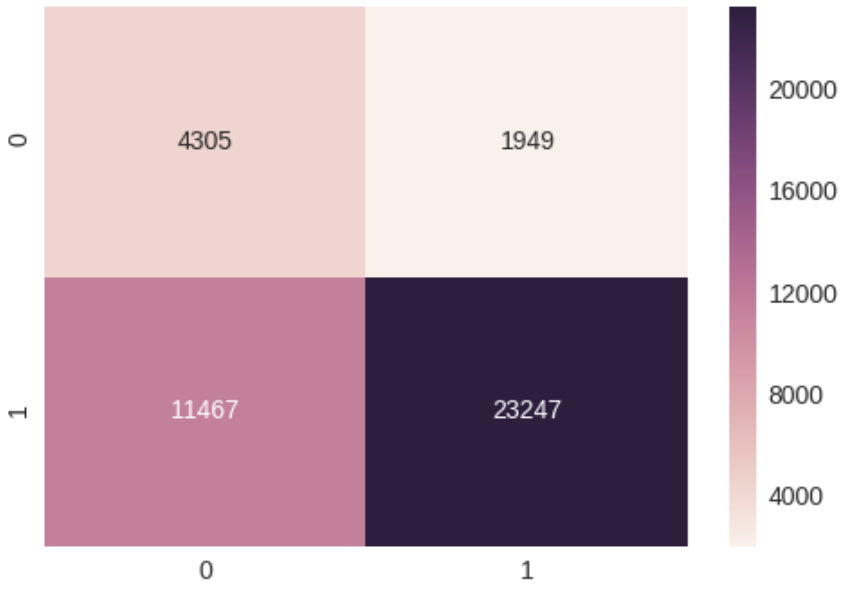
In [0]:
```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc

clss = LogisticRegression(C = 1,class_weight = 'balanced',penalty = 'l2')
clss.fit(x_train_w2v, Y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
# not the predicted outputs

train_fpr, train_tpr, thresholds = roc_curve(Y_train, clss.predict_proba(x_train_w2v)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(Y_test, clss.predict_proba(x_test_w2v)[:,1])

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



In [0]:
```
print("confusion matrix on train data")
y_new_pred_tr  = clss.predict(x_train_w2v)
df_cm_bow_tr = pd.DataFrame(confusion_matrix(Y_train, y_new_pred_tr))
sns.set(font_scale=1.4)#for label size
sns.heatmap(df_cm_bow_tr, annot=True,annot_kws={"size": 14}, fmt='g')
```

```
confusion matrix on train data
```

Out[0]: <matplotlib.axes._subplots.AxesSubplot at 0x7fc74b4e9438>

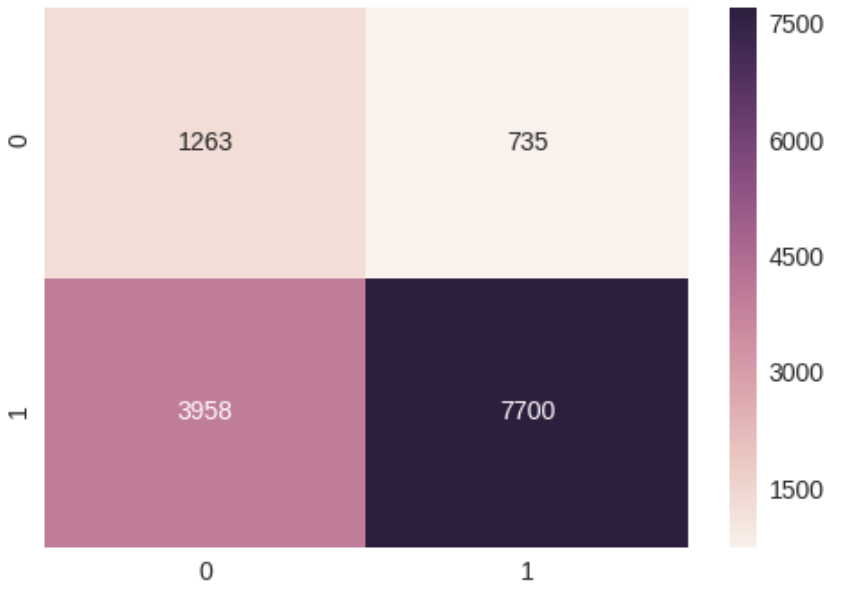


- **From the confusion matrix for train data we can say that,**

4305+23247 = 27652 pouns are correctly classified and 1949+11467 = 13116 points are wrongly classified

In [0]:
```
y_new_pred  = clss.predict(x_test_w2v)
df_cm_bow = pd.DataFrame(confusion_matrix(Y_test, y_new_pred))
sns.set(font_scale=1.4)#for label size
sns.heatmap(df_cm_bow, annot=True,annot_kws={"size": 14}, fmt='g')
```

Out[0]: <matplotlib.axes._subplots.AxesSubplot at 0x7fc74b4fe1d0>



- **From the confusion matrix for test data we can say that,**

1263+7700 = 8963 pouns are correctly classified and 3958 +735 =4693 points are wrongly classified

In [0]:
```
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
from sklearn.metrics import recall_score



print("Accuracy: %f%%"%(accuracy_score(Y_test, y_new_pred)*100))
print("Precision: %f"%(precision_score(Y_test, y_new_pred)))
print("Recall: %f"%(recall_score(Y_test, y_new_pred)))
print("F1-Score: %f"%(f1_score(Y_test, y_new_pred)))
```

```
Accuracy: 65.634153%
Precision: 0.912863
Recall: 0.660491
F1-Score: 0.766436
```

lr on tfidf avg w2v

In [0]:
```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
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 [0]:
```
# 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(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.append(vector)

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

```
100%|██████████| 40968/40968 [02:40<00:00, 254.49it/s]

40968
300
```

In [0]:
```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_test['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 [0]:
```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_te = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['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(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_te.append(vector)

print(len(tfidf_w2v_vectors_te))
print(len(tfidf_w2v_vectors_te[0]))
```

```
100%|██████████| 13656/13656 [00:53<00:00, 257.31it/s]

13656
300
```

In [0]:
```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_test_cv['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 [0]:
```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_tecv = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test_cv['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(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_tecv.append(vector)

print(len(tfidf_w2v_vectors_tecv))
print(len(tfidf_w2v_vectors_tecv[0]))
```

```
100%|██████████| 10242/10242 [00:40<00:00, 250.94it/s]

10242
300
```

project titles

In [0]:
```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train['project_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 [0]:
```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_ttr = []; # 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_ttr.append(vector)

print(len(tfidf_w2v_vectors_ttr))
print(len(tfidf_w2v_vectors_ttr[0]))
```

```
100%|██████████| 40968/40968 [00:00<00:00, 85032.00it/s]

40968
300
```

In [0]:
```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_test['project_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 [0]:
```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_tte = []; # 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_tte.append(vector)

print(len(tfidf_w2v_vectors_tte))
print(len(tfidf_w2v_vectors_tte[0]))
```

```
100%|██████████| 13656/13656 [00:00<00:00, 82265.12it/s]

13656
300
```

In [0]:
```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_test_cv['project_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 [0]:
```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_ttecv = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test_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_ttecv.append(vector)

print(len(tfidf_w2v_vectors_ttecv))
print(len(tfidf_w2v_vectors_ttecv[0]))
```

```
100%|██████████| 10242/10242 [00:00<00:00, 78637.73it/s]

10242
300
```

In [0]:
```
```

2.4.1 Combining all features,tfidf word 2 vec

In [0]:
```
from scipy.sparse import hstack
#with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
x_train_tfidf_w2v= hstack(( categories_one_hot,sub_categories_one_hot,teacher_prefix_one_hot,school_state_one_hot,project_grade_categ
ory_one_hot,tfidf_w2v_vectors_tr,tfidf_w2v_vectors_ttr,price_normalized)).tocsr()
#x_train = x_train.toarray()
#x_train[np.isnan(x_train)] = np.median(x_train[~np.isnan(x_train)])
x_train_tfidf_w2v.shape
```

Out[0]: (40968, 672)

In [0]:
```
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
x_test_tfidf_w2v= hstack((categories_one_hot_te, sub_categories_one_hot_te,teacher_prefix_one_hot_te,school_state_one_hot_te,project_
grade_category_one_hot_te,tfidf_w2v_vectors_te,tfidf_w2v_vectors_tte,price_normalized_te)).tocsr()
#x_test = x_test.toarray()
#x_test[np.isnan(x_test)] = np.median(x_test[~np.isnan(x_test)])
x_test_tfidf_w2v.shape
```

Out[0]: (13656, 672)

In [0]:
```
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
x_test_tfidf_w2v_cv= hstack((categories_one_hot_tecv, sub_categories_one_hot_tecv,teacher_prefix_one_hot_tecv,school_state_one_hot_te
cv,project_grade_category_one_hot_tecv,tfidf_w2v_vectors_tecv,tfidf_w2v_vectors_ttecv,price_normalized_tecv)).tocsr()
#x_test_cv= x_test_cv.toarray()
#x_test_cv[np.isnan(x_test_cv)] = np.median(x_test_cv[~np.isnan(x_test_cv)])
x_test_tfidf_w2v_cv.shape
```

Out[0]: (10242, 672)

In [0]:
```
print("Final Data matrix")
print(x_train_tfidf_w2v.shape, Y_train.shape)
print(x_test_tfidf_w2v_cv.shape, Y_test_cv.shape)
print(x_test_tfidf_w2v.shape, Y_test.shape)
```

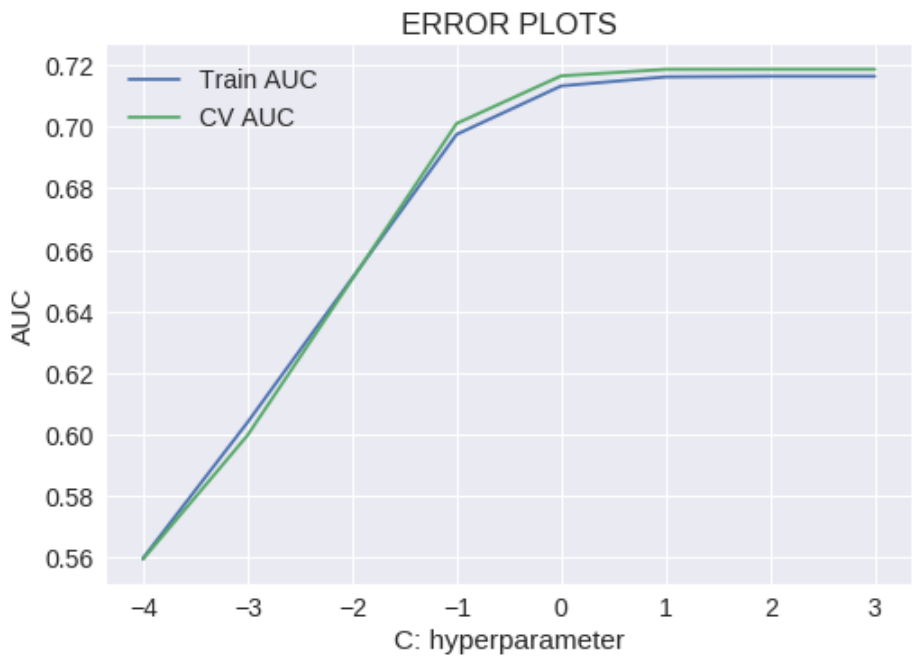
```
Final Data matrix
(40968, 672) (40968,)
(10242, 672) (10242,)
(13656, 672) (13656,)
```

In [0]:
```
###Set 4: categorical, numerical features + project_title(TFIDF W2V)+ preprocessed_essay (TFIDF W2V)
```

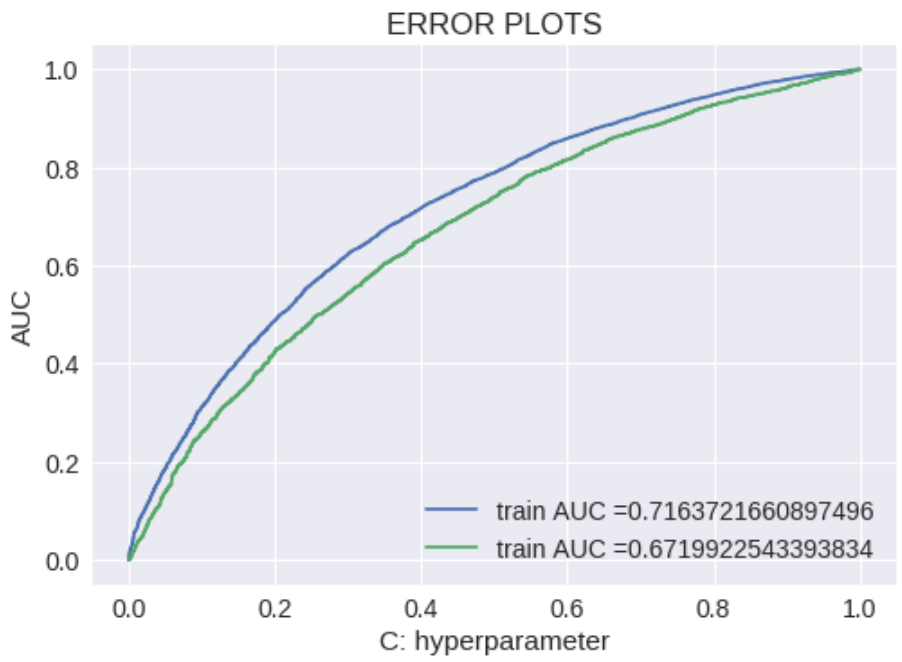
In [0]:
```
# Please write all the code with proper documentation
#finding the best hypermeter with bow train and cv data
#loading library's
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score
from collections import Counter
from sklearn.metrics import accuracy_score
import random
from sklearn import metrics
from sklearn.metrics import roc_auc_score

train_auc = []
cv_auc = []
C = [0.0001,0.001,.1,1,10,100,1000]
for i in C:
    clf = LogisticRegression(C=i,class_weight = 'balanced',penalty = 'l2')
    clf.fit(x_train_tfidf_w2v, Y_train)
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
    # not the predicted outputs
    y_train_pred =  clf.predict_proba(x_train_tfidf_w2v)[:,1]
    y_cv_pred =  clf.predict_proba(x_test_tfidf_w2v_cv)[:,1]
    train_auc_score = roc_auc_score(Y_train,y_train_pred)
    train_auc.append((train_auc_score))
    cv_auc.append(roc_auc_score(Y_test_cv, y_cv_pred))
    cv_auc_score=roc_auc_score(Y_test_cv, y_cv_pred)
    print("C",i,"cv:",cv_auc_score,"train:",train_auc_score)
```

```
C 0.0001 cv: 0.5595542482768439 train: 0.5598677406792479
C 0.001 cv: 0.599801140070943 train: 0.6040032029136285
C 0.1 cv: 0.7010507989987307 train: 0.6975095395145932
C 1 cv: 0.7165236882066126 train: 0.7132522371270094
C 10 cv: 0.7186453798061525 train: 0.7161959631021044
C 100 cv: 0.7186782265431293 train: 0.7163721660897496
C 1000 cv: 0.7186756314284697 train: 0.7163801712603995
```

In [0]:
```
log_a = [math.log10(num) for num in C]
plt.plot(log_a, train_auc, label='Train AUC')
plt.plot(log_a, cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("C: hyperparameter")
#set_xlim = (1e3,1000)
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

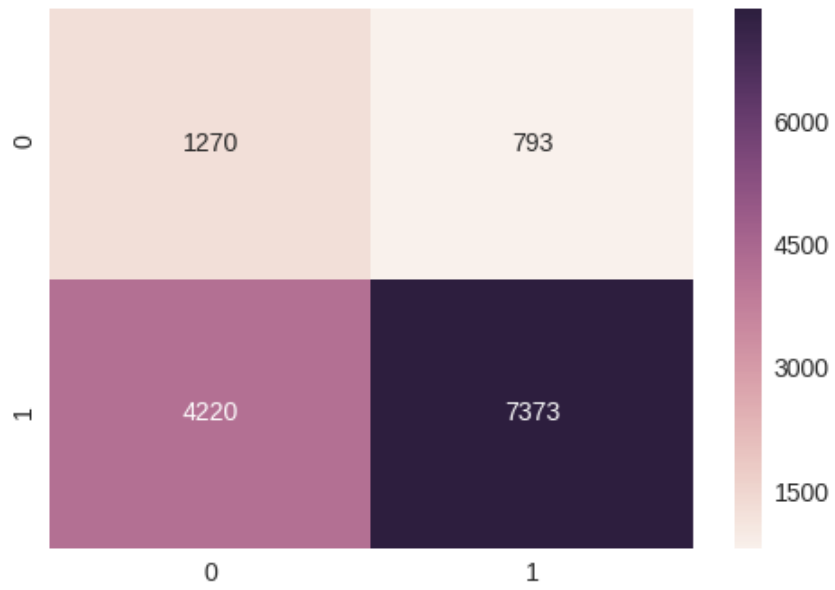


In [0]:
```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score
from collections import Counter
from sklearn.metrics import accuracy_score
import random
from sklearn import metrics
from sklearn.metrics import roc_auc_score

clf = LogisticRegression(C= 100,class_weight = 'balanced',penalty = 'l2',n_jobs = -1)
clf.fit(x_train_tfidf_w2v, Y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
# not the predicted outputs

train_fpr, train_tpr, thresholds = roc_curve(Y_train, clf.predict_proba(x_train_tfidf_w2v)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(Y_test, clf.predict_proba(x_test_tfidf_w2v)[:,1])

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



In [0]:
```
y_new_pred  = clf.predict(x_test_tfidf_w2v)
df_cm_bow = pd.DataFrame(confusion_matrix(Y_test, y_new_pred))
sns.set(font_scale=1.4)#for label size
sns.heatmap(df_cm_bow, annot=True,annot_kws={"size": 14}, fmt='g')
```

Out[0]: <matplotlib.axes._subplots.AxesSubplot at 0x7faff4ba65c0>



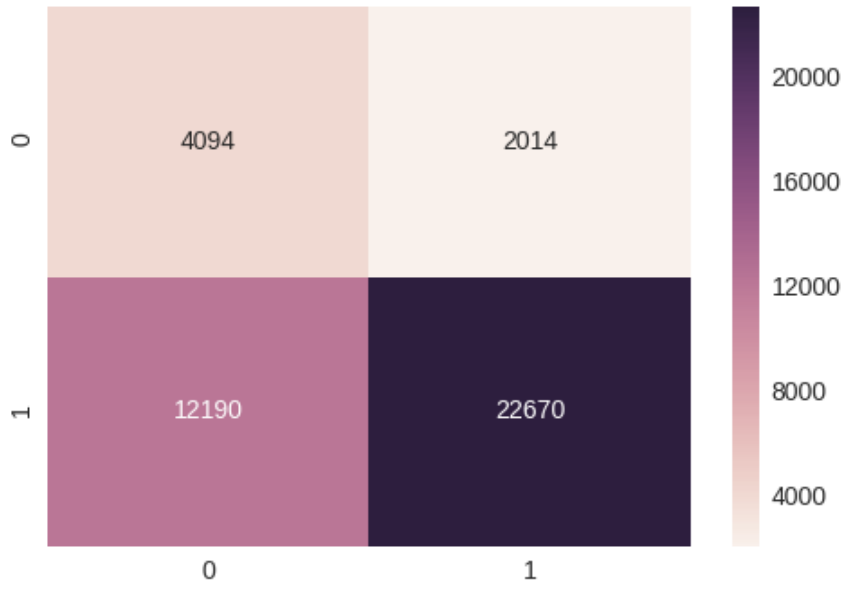
- **From the confusion matrix for test data we can say that,**

1270+7373 = 8653 pouns are correctly classified and 4220+793 = 5013 points are wrongly classified

In [0]:
```
print("confusion matrix on train data")
y_new_pred_tr  = clf.predict(x_train_tfidf_w2v)
df_cm_bow_tr = pd.DataFrame(confusion_matrix(Y_train, y_new_pred_tr))
sns.set(font_scale=1.4)#for label size
sns.heatmap(df_cm_bow_tr, annot=True,annot_kws={"size": 14}, fmt='g')
```

```
confusion matrix on train data
```

Out[0]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7faff1dfb6a0>
```



- **From the confusion matrix for train data we can say that,**

  4094+22670 = 26764 pouns are correctly classified and 12190+2014 = 14204 points are wrongly classified

In [0]:
```
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
from sklearn.metrics import recall_score



print("Accuracy: %f%%"%(accuracy_score(Y_test, y_new_pred)*100))
print("Precision: %f"%(precision_score(Y_test, y_new_pred)))
print("Recall: %f"%(recall_score(Y_test, y_new_pred)))
print("F1-Score: %f"%(f1_score(Y_test, y_new_pred)))
```

```
Accuracy: 63.290861%
Precision: 0.902890
Recall: 0.635987
F1-Score: 0.746293
```

set 5 (without text)

sentimental_scores calculation

In [0]:
```
import pandas as pd
from nltk.sentiment.vader import SentimentIntensityAnalyzer
```

In [0]:
```
import nltk
nltk.download('vader_lexicon')
```

calculation for essay and vectorisation

In [0]:
```
#https://github.com/llSourcell/Sentiment_Analysis/blob/master/Sentiment_Analysis.ipynb
sip = SentimentIntensityAnalyzer()
listn = []
data = pd.DataFrame(project_data["essay"])
for index, row in data.iterrows():
  sn = sip.polarity_scores(row["essay"]) ['neg']
  listn.append(sn)
ne = pd.Series(listn)
data['neagtive'] = ne.values
n = pd.DataFrame(data['neagtive'])
display(n.head(10))
```

In [0]:
```
#https://github.com/llSourcell/Sentiment_Analysis/blob/master/Sentiment_Analysis.ipynb
sipp = SentimentIntensityAnalyzer()
listp = []
data = pd.DataFrame(project_data["essay"])
for index, row in data.iterrows():
  snp = sipp.polarity_scores(row["essay"]) ['pos']
  listp.append(snp)
po = pd.Series(listp)
data['positive'] = po.values
```

In [0]:
```
p = pd.DataFrame(data['positive'])
display(p.head(10))
```

In [0]:
```
#https://github.com/llSourcell/Sentiment_Analysis/blob/master/Sentiment_Analysis.ipynb
sip = SentimentIntensityAnalyzer()
listneu = []
data = pd.DataFrame(project_data["essay"])
for index, row in data.iterrows():
  sn = sip.polarity_scores(row["essay"]) ['neu']
  listneu.append(sn)
neu = pd.Series(listneu)
data['neutral'] = neu.values
ne = pd.DataFrame(data['neutral'])
display(ne.head(10))
```

In [0]:
```
#splitting numerical features
from sklearn.model_selection import train_test_split
X_train_p, X_test_p = train_test_split(n.values,test_size = 0.25,shuffle = False , random_state = 0)
X_train_pcv, X_test_pcv = train_test_split(X_train_p,test_size = 0.25,shuffle = False , random_state = 0)
```

In [0]:
```
# check this one: https://www.youtube.com/watch?v=0HOqOcln3Z4&t=530s
# standardization sklearn: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
from sklearn.preprocessing import Normalizer
# price_standardized = standardScalar.fit(project_data['price'].values)
# this will rise the error
# ValueError: Expected 2D array, got 1D array instead: array=[725.05 213.03 329.   ... 399.   287.73   5.5 ].
# Reshape your data either using array.reshape(-1, 1)
#normalized_X = preprocessing.normalize(X)
X_train_p[np.isnan(X_train_p)] = np.median(X_train_p[~np.isnan(X_train_p)])
Normalizer().fit(X_train_p.reshape(-1,1))
essay_normalized = Normalizer().transform(X_train_p.reshape(-1,1))


X_train_pcv[np.isnan(X_train_pcv)] = np.median(X_train_pcv[~np.isnan(X_train_pcv)])
essay_normalized_cv= Normalizer().transform(X_train_pcv.reshape(-1,1))

X_test_pcv[np.isnan(X_test_pcv)] = np.median(X_test_pcv[~np.isnan(X_test_pcv)])
essay_normalized_tecv= Normalizer().transform(X_test_pcv.reshape(-1,1))

X_test_p[np.isnan(X_test_p)] = np.median(X_test_p[~np.isnan(X_test_p)])
essay_normalized_te= Normalizer().transform(X_test_p.reshape(-1,1))

print(essay_normalized.shape)
print(essay_normalized_cv.shape)
print(essay_normalized_tecv.shape)
print(essay_normalized_te.shape)
```

In [0]:
```
X_train_tnpp, X_test_tnpp = train_test_split(p.values,test_size = 0.25,shuffle = False , random_state = 0)
X_train_tnppcv, X_test_tnppcv = train_test_split(X_train_tnpp,test_size = 0.25,shuffle = False , random_state = 0)
```

In [0]:
```
#teacher_number_of_previously_posted_projects feature
from sklearn.preprocessing import Normalizer
# price_standardized = standardScalar.fit(project_data['price'].values)
# this will rise the error
# ValueError: Expected 2D array, got 1D array instead: array=[725.05 213.03 329.   ... 399.   287.73   5.5 ].
# Reshape your data either using array.reshape(-1, 1)
#normalized_X = preprocessing.normalize(X)
X_train_tnpp[np.isnan(X_train_tnpp)] = np.median(X_train_tnpp[~np.isnan(X_train_tnpp)])
Normalizer().fit(X_train_tnpp.reshape(-1,1))
p_normalized_tnpp = Normalizer().transform(X_train_tnpp.reshape(-1,1))


X_train_tnppcv[np.isnan(X_train_tnppcv)] = np.median(X_train_tnppcv[~np.isnan(X_train_tnppcv)])
p_normalized_tnppcv= Normalizer().transform(X_train_tnppcv.reshape(-1,1))

X_test_tnppcv[np.isnan(X_test_tnppcv)] = np.median(X_test_tnppcv[~np.isnan(X_test_tnppcv)])
p_normalized_tnpptecv= Normalizer().transform(X_test_tnppcv.reshape(-1,1))

X_test_p[np.isnan(X_test_tnpp)] = np.median(X_test_tnpp[~np.isnan(X_test_tnpp)])
p_normalized_tnppte= Normalizer().transform(X_test_tnpp.reshape(-1,1))

print(p_normalized_tnpp.shape)
print(p_normalized_tnppcv.shape)
print(p_normalized_tnpptecv.shape)
print(p_normalized_tnppte.shape)
```

In [0]:
```
#splitting numerical features
X_train_t, X_test_t = train_test_split(ne.values,test_size = 0.25,shuffle = False , random_state = 0)
X_train_tcv, X_test_tcv = train_test_split(X_train_p,test_size = 0.25,shuffle = False , random_state = 0)

# check this one: https://www.youtube.com/watch?v=0HOqOcln3Z4&t=530s
# standardization sklearn: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
from sklearn.preprocessing import Normalizer
# price_standardized = standardScalar.fit(project_data['price'].values)
# this will rise the error
# ValueError: Expected 2D array, got 1D array instead: array=[725.05 213.03 329.   ... 399.   287.73   5.5 ].
# Reshape your data either using array.reshape(-1, 1)
#normalized_X = preprocessing.normalize(X)
X_train_t[np.isnan(X_train_t)] = np.median(X_train_t[~np.isnan(X_train_t)])
Normalizer().fit(X_train_t.reshape(-1,1))
ne_normalized = Normalizer().transform(X_train_t.reshape(-1,1))


X_train_tcv[np.isnan(X_train_tcv)] = np.median(X_train_tcv[~np.isnan(X_train_tcv)])
ne_normalized_cv= Normalizer().transform(X_train_tcv.reshape(-1,1))

X_test_tcv[np.isnan(X_test_tcv)] = np.median(X_test_tcv[~np.isnan(X_test_tcv)])
ne_normalized_tecv= Normalizer().transform(X_test_tcv.reshape(-1,1))

X_test_t[np.isnan(X_test_t)] = np.median(X_test_t[~np.isnan(X_test_t)])
ne_normalized_te= Normalizer().transform(X_test_t.reshape(-1,1))

print(ne_normalized.shape)
print(ne_normalized_cv.shape)
print(ne_normalized_tecv.shape)
print(ne_normalized_te.shape)
```

calculation for the title

In [0]:
```
#https://github.com/llSourcell/Sentiment_Analysis/blob/master/Sentiment_Analysis.ipynb
sip = SentimentIntensityAnalyzer()
listn = []
data = pd.DataFrame(project_data["project_title"])
for index, row in data.iterrows():
  sn = sip.polarity_scores(row["project_title"]) ['pos']
  listn.append(sn)
ne = pd.Series(listn)
data['neagtive'] = ne.values
r = pd.DataFrame(data['neagtive'])
display(r.head(10))
```

In [0]:
```
#https://github.com/llSourcell/Sentiment_Analysis/blob/master/Sentiment_Analysis.ipynb
sip = SentimentIntensityAnalyzer()
listp = []
data = pd.DataFrame(project_data["project_title"])
for index, row in data.iterrows():
  sn = sip.polarity_scores(row["project_title"]) ['pos']
  listp.append(sn)
po = pd.Series(listp)
data['positive'] = po.values
s = pd.DataFrame(data['positive'])
display(s.head(10))
```

In [0]:
```
#https://github.com/llSourcell/Sentiment_Analysis/blob/master/Sentiment_Analysis.ipynb
sip = SentimentIntensityAnalyzer()
listneu = []
data = pd.DataFrame(project_data["project_title"])
for index, row in data.iterrows():
  sn = sip.polarity_scores(row["project_title"]) ['neu']
  listneu.append(sn)
neu = pd.Series(listneu)
data['neutral'] = neu.values
y = pd.DataFrame(data['neutral'])
display(y.head(10))
```

In [0]:
```
#splitting numerical features
X_train_r, X_test_r = train_test_split(r.values,test_size = 0.25,shuffle = False , random_state = 0)
X_train_rcv, X_test_rcv = train_test_split(X_train_r,test_size = 0.25,shuffle = False , random_state = 0)
```

In [0]:
```
# check this one: https://www.youtube.com/watch?v=0HOqOcln3Z4&t=530s
# standardization sklearn: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
from sklearn.preprocessing import Normalizer
# price_standardized = standardScalar.fit(project_data['price'].values)
# this will rise the error
# ValueError: Expected 2D array, got 1D array instead: array=[725.05 213.03 329.   ... 399.   287.73   5.5 ].
# Reshape your data either using array.reshape(-1, 1)
#normalized_X = preprocessing.normalize(X)
X_train_r[np.isnan(X_train_r)] = np.median(X_train_r[~np.isnan(X_train_r)])
Normalizer().fit(X_train_r.reshape(-1,1))
title_normalized = Normalizer().transform(X_train_r.reshape(-1,1))


X_train_rcv[np.isnan(X_train_rcv)] = np.median(X_train_rcv[~np.isnan(X_train_rcv)])
title_normalized_cv= Normalizer().transform(X_train_rcv.reshape(-1,1))

X_test_rcv[np.isnan(X_test_rcv)] = np.median(X_test_rcv[~np.isnan(X_test_rcv)])
title_normalized_tecv= Normalizer().transform(X_test_rcv.reshape(-1,1))

X_test_r[np.isnan(X_test_r)] = np.median(X_test_r[~np.isnan(X_test_r)])
title_normalized_te= Normalizer().transform(X_test_r.reshape(-1,1))

print(title_normalized.shape)
print(title_normalized_cv.shape)
print(title_normalized_tecv.shape)
print(title_normalized_te.shape)
```

In [0]:
```
X_train_rnpp, X_test_rnpp = train_test_split(s.values,test_size = 0.25,shuffle = False , random_state = 0)
X_train_rnppcv, X_test_rnppcv = train_test_split(X_train_rnpp,test_size = 0.25,shuffle = False , random_state = 0)
```

In [0]:
```
#teacher_number_of_previously_posted_projects feature
from sklearn.preprocessing import Normalizer
# price_standardized = standardScalar.fit(project_data['price'].values)
# this will rise the error
# ValueError: Expected 2D array, got 1D array instead: array=[725.05 213.03 329.   ... 399.   287.73   5.5 ].
# Reshape your data either using array.reshape(-1, 1)
#normalized_X = preprocessing.normalize(X)
X_train_rnpp[np.isnan(X_train_rnpp)] = np.median(X_train_rnpp[~np.isnan(X_train_rnpp)])
Normalizer().fit(X_train_rnpp.reshape(-1,1))
p_normalized_rnpp = Normalizer().transform(X_train_rnpp.reshape(-1,1))


X_train_rnppcv[np.isnan(X_train_rnppcv)] = np.median(X_train_rnppcv[~np.isnan(X_train_rnppcv)])
p_normalized_rnppcv= Normalizer().transform(X_train_rnppcv.reshape(-1,1))

X_test_rnppcv[np.isnan(X_test_rnppcv)] = np.median(X_test_rnppcv[~np.isnan(X_test_rnppcv)])
p_normalized_rnpptecv= Normalizer().transform(X_test_rnppcv.reshape(-1,1))

X_test_p[np.isnan(X_test_rnpp)] = np.median(X_test_rnpp[~np.isnan(X_test_rnpp)])
p_normalized_rnppte= Normalizer().transform(X_test_rnpp.reshape(-1,1))

print(p_normalized_rnpp.shape)
print(p_normalized_rnppcv.shape)
print(p_normalized_rnpptecv.shape)
print(p_normalized_rnppte.shape)
```

In [0]:

In [0]:
```
#splitting numerical features
X_train_rt, X_test_rt = train_test_split(y.values,test_size = 0.25,shuffle = False , random_state = 0)
X_train_rtcv, X_test_rtcv = train_test_split(X_train_rt,test_size = 0.25,shuffle = False , random_state = 0)
```

In [0]:
```
# check this one: https://www.youtube.com/watch?v=0HOqOcln3Z4&t=530s
# standardization sklearn: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
from sklearn.preprocessing import Normalizer
# price_standardized = standardScalar.fit(project_data['price'].values)
# this will rise the error
# ValueError: Expected 2D array, got 1D array instead: array=[725.05 213.03 329.   ... 399.   287.73   5.5 ].
# Reshape your data either using array.reshape(-1, 1)
#normalized_X = preprocessing.normalize(X)
X_train_rt[np.isnan(X_train_rt)] = np.median(X_train_rt[~np.isnan(X_train_rt)])
Normalizer().fit(X_train_rt.reshape(-1,1))
rne_normalized = Normalizer().transform(X_train_t.reshape(-1,1))


X_train_rtcv[np.isnan(X_train_rtcv)] = np.median(X_train_rtcv[~np.isnan(X_train_rtcv)])
rne_normalized_cv= Normalizer().transform(X_train_rtcv.reshape(-1,1))

X_test_rtcv[np.isnan(X_test_rtcv)] = np.median(X_test_rtcv[~np.isnan(X_test_rtcv)])
rne_normalized_tecv= Normalizer().transform(X_test_rtcv.reshape(-1,1))

X_test_rt[np.isnan(X_test_rt)] = np.median(X_test_rt[~np.isnan(X_test_rt)])
rne_normalized_te= Normalizer().transform(X_test_rt.reshape(-1,1))

print(rne_normalized.shape)
print(rne_normalized_cv.shape)
print(rne_normalized_tecv.shape)
print(rne_normalized_te.shape)
```

combining all the features

In [0]:
```
%time
from scipy.sparse import hstack
#with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
x_train= hstack(( categories_one_hot,sub_categories_one_hot,teacher_prefix_one_hot,rne_normalized,p_normalized_rnpp,
                 title_normalized,essay_normalized,ne_normalized,p_normalized_tnpp,school_state_one_hot,
                 project_grade_category_one_hot,teacher_prefix_one_hot,quantity_normalized,price_normalized,normalized_tnpp)).tocsr()
#x_train = x_train.toarray()
#x_train[np.isnan(x_train)] = np.median(x_train[~np.isnan(x_train)])
x_train.shape
```

```
CPU times: user 3 µs, sys: 0 ns, total: 3 µs
Wall time: 6.44 µs
```

Out[0]: (40968, 114)

In [0]:
```
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
x_train_cv= hstack((categories_one_hot_cv, sub_categories_one_hot_cv,rne_normalized_cv,
                    p_normalized_rnppcv,title_normalized_cv,teacher_prefix_one_hot_cv,essay_normalized_cv,
                    ne_normalized_cv,p_normalized_tnppcv,school_state_one_hot_cv,project_grade_category_one_hot_cv,
                    teacher_prefix_one_hot_cv,quantity_normalized_cv,price_normalized_cv,normalized_tnppcv)).tocsr()
#x_train_cv = x_train_cv.toarray()
#x_train_cv[np.isnan(x_train_cv)] = np.median(x_train_cv[~np.isnan(x_train_cv)])
x_train_cv.shape
```

Out[0]: (30726, 114)

In [0]:
```
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
x_test= hstack((categories_one_hot_te, sub_categories_one_hot_te,teacher_prefix_one_hot_te,
                rne_normalized_te,p_normalized_rnppte,title_normalized_te,essay_normalized_te,ne_normalized_te,
                p_normalized_tnppte,school_state_one_hot_te,teacher_prefix_one_hot_te,quantity_normalized_te,
                project_grade_category_one_hot_te,price_normalized_te,normalized_tnppte)).tocsr()
#x_test = x_test.toarray()
#x_test[np.isnan(x_test)] = np.median(x_test[~np.isnan(x_test)])
x_test.shape
```

Out[0]: (13656, 114)

In [0]:
```
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
x_test_cv= hstack((categories_one_hot_tecv, sub_categories_one_hot_tecv,teacher_prefix_one_hot_tecv,school_state_one_hot_tecv,
                   rne_normalized_tecv,p_normalized_rnpptecv,title_normalized_tecv,essay_normalized_tecv,ne_normalized_tecv,
                   p_normalized_tnpptecv,teacher_prefix_one_hot_tecv,quantity_normalized_tecv,project_grade_category_one_hot_tecv,
                   price_normalized_tecv,normalized_tnpptecv)).tocsr()
#x_test_cv= x_test_cv.toarray()
#x_test_cv[np.isnan(x_test_cv)] = np.median(x_test_cv[~np.isnan(x_test_cv)])
x_test_cv.shape
```

Out[0]: (10242, 114)

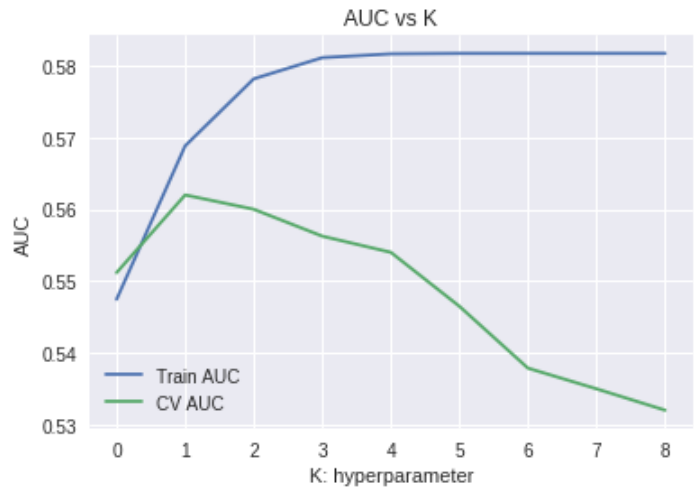
In [0]:
```
print("Final Data matrix")
print(x_train.shape, Y_train.shape)
print(x_test_cv.shape, Y_test_cv.shape)
print(x_test.shape, Y_test.shape)
```

```
Final Data matrix
(40968, 114) (36598,)
(10242, 114) (12078,)
(13656, 114) (18026,)
```

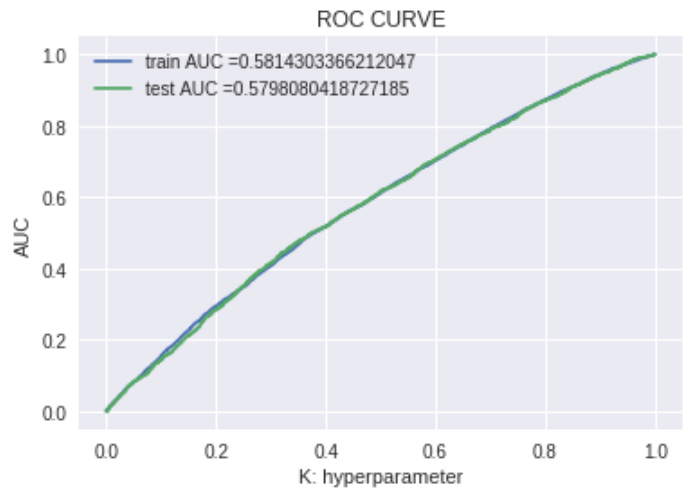
In [0]:
```
from sklearn import metrics
from sklearn.metrics import roc_auc_score
import numpy as np
from sklearn.linear_model import LogisticRegression

train_auc = []
cv_auc = []
C = [0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
for i in C:
    lg1_bow = LogisticRegression(C = i)# The “balanced” mode uses the values of y to automatically adjust weights inversely proportio
nal to class frequencies
    lg1_bow.fit(x_train, 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 =  lg1_bow.predict_proba(x_train)[:,1]
    y_cv_pred =  lg1_bow.predict_proba(x_test_cv)[:,1]
    train_auc_score=roc_auc_score(Y_train ,y_train_pred)
    train_auc.append(train_auc_score)
    cv_auc_score=roc_auc_score(Y_test_cv, y_cv_pred)
    cv_auc.append(cv_auc_score)
    print("C = ",i ,"\t","cv_auc_score\t:",cv_auc_score, "\t","train_auc_score\t:",train_auc_score)
```

```
C =  0.0001      cv_auc_score    : 0.5511896624066137     train_auc_score        : 0.5474880290475939
C =  0.001       cv_auc_score    : 0.5619937797283747     train_auc_score        : 0.5688219215913143
C =  0.01        cv_auc_score    : 0.5600066911004444     train_auc_score        : 0.5781379896125568
C =  0.1         cv_auc_score    : 0.5562642220629985     train_auc_score        : 0.5810874368744154
C =  1   cv_auc_score    : 0.5540167022354106     train_auc_score        : 0.5816126615994053
C =  10          cv_auc_score    : 0.5465116652009393     train_auc_score        : 0.5816759085558176
C =  100         cv_auc_score    : 0.5379105221403225     train_auc_score        : 0.581677889197523
C =  1000        cv_auc_score    : 0.5350355156455789     train_auc_score        : 0.5816773871278814
C =  10000       cv_auc_score    : 0.5320642907273165     train_auc_score        : 0.581678073443263
```

In [0]:
```
import math
C= [0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
log_a = [math.log10(num) for num in C]
plt.plot( train_auc, label='Train AUC')
plt.plot( cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("AUC vs K")
plt.show()
```



In [0]:
```
clf = LogisticRegression(C=0.4 ,penalty = 'l1',class_weight = 'balanced')
clf.fit(x_train, Y_train)

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

train_fpr, train_tpr, thresholds = roc_curve(Y_train, clf.predict_proba(x_train)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(Y_test, clf.predict_proba(x_test)[:,1])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ROC CURVE")
plt.show()

print("="*100)

from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(Y_train, clf.predict(x_train)))
print("Test confusion matrix")
print(confusion_matrix(Y_test, clf.predict(x_test)))
```

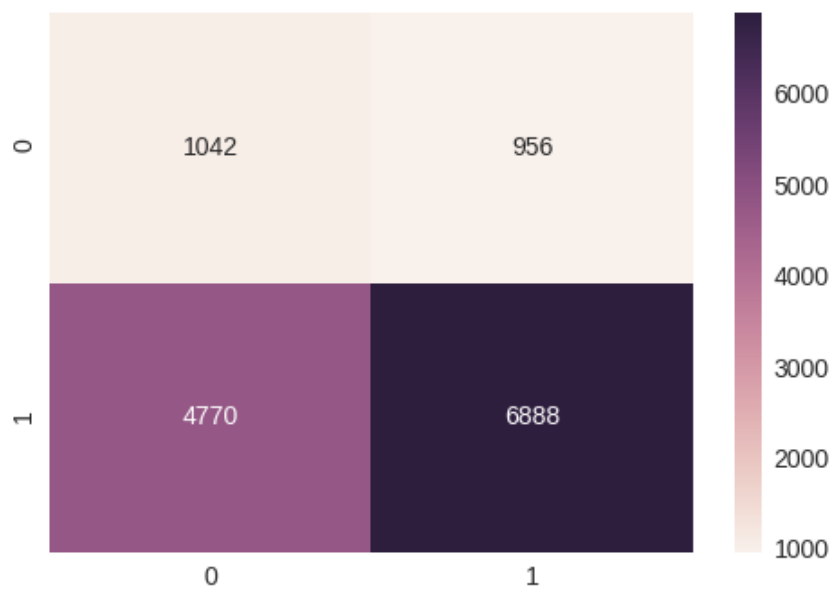



```
====================================================================================================
Train confusion matrix
[[ 3413  2841]
 [14940 19774]]
Test confusion matrix
[[1042  956]
 [4770 6888]]
```

In [0]:
```
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
from sklearn.metrics import recall_score
print('confusion matrix on test data')
y_new_pred  = clf.predict(x_test)
df_cm_bow = pd.DataFrame(confusion_matrix(Y_test, y_new_pred))
sns.set(font_scale=1.4)#for label size
sns.heatmap(df_cm_bow, annot=True,annot_kws={"size": 14}, fmt='g')
```

```
confusion matrix on test data
```

Out[0]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc754e09828>
```



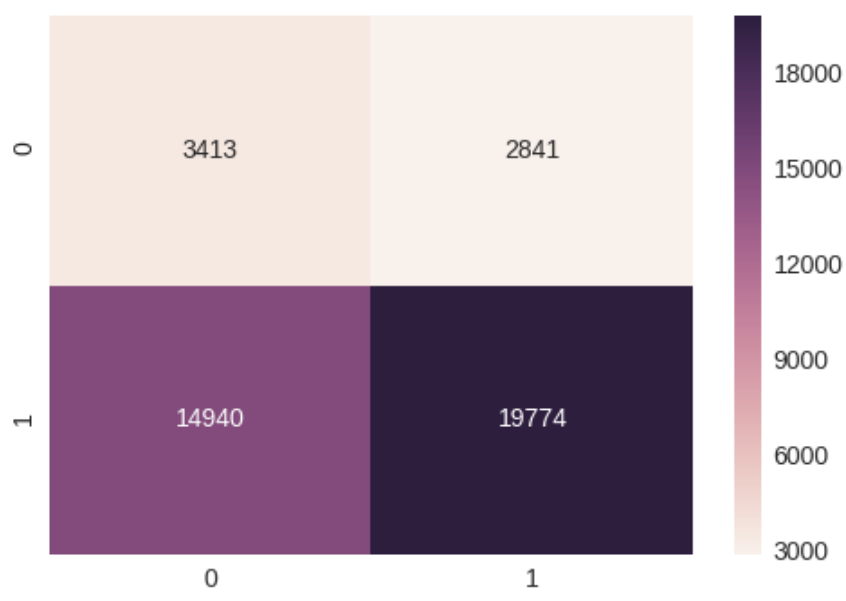
- **From the confusion matrix for test data we can say that,**

  6888+1042 = 7030 pouns are correctly classified and 4770+956 = 5714 points are wrongly classified

In [0]:
```
print("confusion matrix on train data")
y_new_pred_tr  = clf.predict(x_train)
df_cm_bow_tr = pd.DataFrame(confusion_matrix(Y_train, y_new_pred_tr))
sns.set(font_scale=1.4)#for label size
sns.heatmap(df_cm_bow_tr, annot=True,annot_kws={"size": 14}, fmt='g')
```

```
confusion matrix on train data
```

Out[0]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc754e0ab70>
```



- **From the confusion matrix for train data we can say that,**

  19774+3413 = 23187 pouns are correctly classified and 14940+2841 = 17781 points are wrongly classified

In [0]:
```
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
from sklearn.metrics import recall_score



print("Accuracy: %f%%"%(accuracy_score(Y_test, y_new_pred)*100))
print("Precision: %f"%(precision_score(Y_test, y_new_pred)))
print("Recall: %f"%(recall_score(Y_test, y_new_pred)))
print("F1-Score: %f"%(f1_score(Y_test, y_new_pred)))
```

```
Accuracy: 58.069713%
Precision: 0.878123
Recall: 0.590839
F1-Score: 0.706389
```

In [2]:
```
# compare all your models using Prettytable library
#ref : http://zetcode.com/python/prettytable/
from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["Vectorizer", "Model", "Best K", "Train_Auc","Test_Auc","Accuracy"]

x.add_row(["BOW", "LogisticRegression",0.01,77,71,67.75])
x.add_row(["TF-IDf","LogisticRegression",0.01,71,67,64.90])
x.add_row(["AVGW2V", "LogisticRegression",1,73,69,65.63])
x.add_row(["TFIDFW2V", "LogisticRegression",100,71,67,63.29])
x.add_row(["no text features set(set 5)","LogisticRegression",0.4,58,57,58.069])
print(x)
print("From the above petty table, we can observe that we got the better accuracy when we used tfidf vectorizer.The accuracy is 67.6
5")
print("In the case where we took only numerical features the accuracy dropped to 58.06 which tells us text data impact the model a lo
t")
```

```
+-----------------------------+--------------------+--------+-----------+----------+----------+
|          Vectorizer         |       Model        | Best K | Train_Auc | Test_Auc | Accuracy |
+-----------------------------+--------------------+--------+-----------+----------+----------+
|             BOW             | LogisticRegression |  0.01  |     77    |    71    |  67.75   |
|            TF-IDf           | LogisticRegression |  0.01  |     71    |    67    |   64.9   |
|            AVGW2V           | LogisticRegression |   1    |     73    |    69    |  65.63   |
|           TFIDFW2V          | LogisticRegression |  100   |     71    |    67    |  63.29   |
| no text features set(set 5) | LogisticRegression |  0.4   |     58    |    57    |  58.069  |
+-----------------------------+--------------------+--------+-----------+----------+----------+
From the above petty table, we can observe that we got the better accuracy when we used tfidf vectorizer.The accuracy is 67.65
In the case where we took only numerical features the accuracy dropped to 58.06 which tells us text data impact the model a lot
```

Observations

- The data set considered has more than 50k points(considered 50 k points from entire data set)
- The data was splitted into train and test in the ratio of 3:1
- The traindata is again splitted into train cross valiadted and test cross validated data in the ratio of 3:1

Bag Of words

- Tha optimal alpha value is 0.1
- From the confusion matrix,
- For Train Data,
  - 24405+4484 = 28889 are correctly classified
  - 10455+1624 = 12079 are wrongly classified
- For Test Data,
  - 7964+1275 = 9239 are correctly classified
  - 3629+788 = 4417 are wrongly classified
- The perfomance parameters:
- Accuracy using BOW is 67.75%
- Precision using BOW is 0.90
- Recall using BOW is 0.68
- F1 score using BOW is 0.78

TFIDF

- The optimal alpha value is 0.01
- From the confusion matrix,
- For Train Data,
  - 4069+23062 = 26431 are correctly classified
  - 11798+2039 = 13837 are wrongly classified
- For Test Data,
  - 1241 + 7623 = 8864 are correctly classified
  - 3970+822 = 4792 are wrongly classified
- The perfomance parameters:
- Accuracy using TFIDF is 64.90%
- Precision using TFIDF is 0.90
- Recall using TFIDFis 0.71
- F1 score using TFIDF is 0.79

Weighted W2V

- Tha optimal alpha value is 1
- From the confusion matrix,
- For Train Data,
  - 4305+23247 = 27652 are correctly classified
  - 1949+11467 = 13116 are wrongly classified
- For Test Data,
  - 1263+7700 = 8963 are correctly classified
  - 3958 +735 =4693 are wrongly classified
- The perfomance parameters:
- Accuracy using w2v is 65.63%
- Precision using w2v is 0.90
- Recall using w2v is 0.65
- F1 score using w2v is 076

TFIDF Weighted W2V

- The optimal alpha value is 100
- From the confusion matrix,
- For Train Data,
  - 4094+22670 = 26764 are correctly classified
  - 12190+2014 = 14204 are wrongly classified
- For Test Data,
  - 1270+7373 = 8653 are correctly classified
  - 4220+793 = 5013 are wrongly classified
- The perfomance parameters:
- Accuracy using TFIDFw2v is 65.63%
- Precision using TFIDF w2v is 0.90
- Recall using TFIDF w2vis 0.63
- F1 score using TFIDF w2v is 0.74

No text Features

- Tha optimal alpha value is 0.4
- From the confusion matrix,
- For Train Data,
  - 19774+3413 = 23187 are correctly classified
  - 14940+2841 = 17781re wrongly classified
- For Test Data,
  - 6888+1042 = 7030 are correctly classified
  - 4770+956 = 5714 are wrongly classified
- The perfomance parameters:

- Accuracy using set5 is 58.06%
- Precision using set5 is 0.87
- Recall using set5 is 0.59
- F1 score using set5 is 0.70

The No text set has numerical features and categorical features and for the text features,we have calculated the sentiment scores. so this set doesnt contain any text features.From the logistic regression on this set we got accuracy of 58.06% which is low .So we can infer that when text features are used the model performance will be much better.

In [0]:

In [0]:

In [0]:

In [0]:

In [0]:

In [0]:

In [0]: