DonorsChoose

Adding explanations as suggested

Answers to question 1 and 2 were already documented in the code. However, it seems that the comments got burried under lot of analysis and were not spotted while evaluation. I answering everything in more detail here.

1. Can you please tell us the reason behind splitting the data into train, cross-validation and test, since you are considering k-fold cross validation?

Answer: Only after I explored the library I got to know that RandomSearchCV and GridSerachCV automatically handle splitting training data into cross validation sets and therefore, there was no need to do it. So, I merged them and gave them to the lib, this resulted in my machine dying. I tried the batch prediction code but it was not helping. So, I let them remain as is and have only used train data and passed it to the library, essentially the train and CV happens only on train_Data and **X_cv data is not used while searching for best_k** (infact I have not used it at all anywhere in the code). Please find below the comments that I had added during the first submission itself.

```
   * Even though the data has been split into train, test and cv. Only th
e train data has been use for CV    purpose as the library function Rando
mSearchCV is used.

   * This is because I was facing a lot of memory issues while training t
he data even on google colab. I tried stacking up train and CV data and g
ave it for hyperparameter tuning, my local machine died and google colab
 was behaving weird so decided to go only with training data(~40K point
s).
```

2. Can you please explain us, how you are going to plot ROC curve by using "K-hyperparameter" and "AUC" score.

Answer: That was a typo from my part. I had put in comments for the graphs which are incorrectly labelled during the first submission. Please note that the data used to plot the ROC curve is False Positive Rate (X-axis) and True positive rate (Y-axis). By the time I came back and checked on the output of my code, a few more cells had already run and since I am using same variable names the former ones got overridden. Plotting the ROC curve again with correct labels would have meant predict every point in test data again using KNN which is time consuming so I added comments for incorrect labels. Again only labels are incorrect, data used for plotting is correct train_fpr, train_tpr,test_fpr and test_tpr.

Following comments had been added previously:

#

The following labels are incorrect in the plot.

Title: ROC curve for train and test data for BOW rep of text.

X-axis is False Positive Rate (FPR)

Y-axis is True Positive Rate.

#

3. Please explain the difference between applying fit() function on data, "transform() function on data " and "fit_transform() function on data"

Answer: In context of text encodings and vectorization. The fit() function only learns the vocabulary from the data and does not convert the data into BOW/Tfidf/ohe etc. The transform() function when applied on the data gives back the actual encoding we are looking for based on the vocabulary learnt during training, that is

when fit() function was applied. The fit_transform() is basically a single shot method to do both at once.

To avoid data leakage: fit() is done only on X_train which returns a vectorizer, this contains the vocab learnt from train data; then, transform is applied to train and test data. Vocab of test data is not shown to the model apriori

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	
project_id	A unique identifier for the proposed project. **Examp**
project_title	Title of the proje • Art Will Make • First
project_grade_category	Grade level of students for which the project is targeted. One enum • Gra • • • (
project_subject_categories	One or more (comma-separated) subject categories for the p following enumerated • Applie • Ca • Healt • Histo • Literacy • Mat • Music • Spe • • • Music • Literacy & Language, Mat
school_state	State where school is located (Two-letter U (https://en.wikipedia.org/wiki/List_of_U.S._state_abbreviations#F
project_subject_subcategories	One or more (comma-separated) subject subcategories • • Literature & Writing, Socia
project_resource_summary	An explanation of the resources needed for the proj • My students need hands on literacy material sens
project_essay_1	First ap
project_essay_2	Second ap
project_essay_3	Third ap
project_essay_4	Fourth ap
project_submitted_datetime	Datetime when project application was submitted. **Example:** 12
teacher_id	A unique identifier for the teacher of the proposed pro bdf8baa8fedef6bfeec7ae

Feature	
teacher_prefix	Teacher's title. One of the following enum • • • • • •
teacher_number_of_previously_posted_projects	Number of project applications previously submitted by the

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

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

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

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

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

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

Notes on the Essay Data

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

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

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

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

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

In [1]:

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

import sqlite3
import pandas as pd
import numpy as np
import nltk
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
from scipy.sparse import hstack
from scipy.sparse import vstack
from scipy.stats import randint as sp_randint
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
```

1.1 Reading Data

In [2]:

```
project_data = pd.read_csv('train_data.csv',nrows=50000)
resource_data = pd.read_csv('resources.csv')
```

Adding price attribute to project_data dataframe from resources using merge function

In [3]:

```
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 [4]:

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

```
Number of data points in train data (50000, 19)
--------------------------------------------------
The attributes of data : ['Unnamed: 0' 'id' 'teacher_id' 'teacher_pr
efix' '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_approve
d'
 'price' 'quantity']
```

In [5]:

```
# how to replace elements in list python: https://stackoverflow.com/a/2582163/40
84039
cols = ['Date' if x=='project_submitted_datetime' else x for x in list(project_d
ata.columns)]


#sort dataframe based on time pandas python: https://stackoverflow.com/a/4970249
2/4084039
project_data['Date'] = pd.to_datetime(project_data['project_submitted_datetime'
])
project_data.drop('project_submitted_datetime', axis=1, inplace=True)
project_data.sort_values(by=['Date'], inplace=True)


# how to reorder columns pandas python: https://stackoverflow.com/a/13148611/408
4039
project_data = project_data[cols]
```

In [6]:

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

Out[6]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	GA
41558	33679	p137682	06f6e62e17de34fcf81020c77549e1d5	Mrs.	WA

In [7]:

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

```
Number of data points in train data (1541272, 4)
['id' 'description' 'quantity' 'price']
```

Out[7]:

	id	description	quantity	price
0	p233245	LC652 - Lakeshore Double-Space Mobile Drying Rack	1	149.00
1	p069063	Bouncy Bands for Desks (Blue support pipes)	3	14.95

1.2 preprocessing of `project_subject_categories`

Here we perform the following operations

- Clean the project_subject_categrories by converting Math & Science, Care & Hunger ==> Math_Science Care_Hunger and put them in the cat_list.
- Remove the actual column from the pandas dataframe and instead include another column called 'cleaned categories'.
- Then create a dictonary which holds the frequency of the unique subject categories, if a project falls under two different categories then this will will increase the count of each of those subject categories by one. After cleaning the data multiple categories for a project are separated by space, hence, it is easy to split them and then use Counter to create the dictionary with frequency of each category.
- Sort the dictionary based on the frequency.

In [8]:

```
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)
```

1.3 preprocessing of `project_subject_subcategories`

- We process the subject sub-categories in a manner similar to the subject categories.
- Then, create a dictionary with key as the sub-category and the value as the frequency .
- Sort the dictionary based on the frerquency.

In [9]:

```
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
```

In [10]:

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

1.3 Text preprocessing : Essays

- We combine all the eassay_1, essay_2, essay_3 and essay_4 into one single essay by concatinating all the columns as strings with each other.
- We will now work on a single 'essay' column in the dataframe rather than four different columns.

In [11]:

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

In [12]:

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

Out[12]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	GA
41558	33679	p137682	06f6e62e17de34fcf81020c77549e1d5	Mrs.	WA

In [13]:

```
# https://stackoverflow.com/a/47091490/4084039
import re

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

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

In [14]:

```
# https://gist.github.com/sebleier/554280
# 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', 'itse
lf', 'they', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'tha
t', "that'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'ha
s', 'had', 'having', 'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'becaus
e', 'as', 'until', 'while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 't
hrough', 'during', 'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'of
f', 'over', 'under', 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'al
l', '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'v
e", 'now', 'd', 'll', 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "d
idn'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', "should
n't", 'wasn', "wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"}
```

In [15]:

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

```
100%|██████████| 50000/50000 [00:10<00:00, 4899.40it/s]
```

In [16]:

```
# after preprocesing
preprocessed_essays[20000]
```

Out[16]:

```
'teach title 1 school 73 students receive free reduced lunch school
provides free breakfast students special education certified teacher
teach kindergarten general education setting class consists 52 stude
nts special needs disabilities include autism spectrum disorder spee
ch impaired language impaired health impaired adhd developmentally d
elayed also 42 students english language learners self motivated lea
rners synonym students love learn possess positive outlook attitude
school almost everyday students would ask ms perez going learn today
could not ask better greeting students project greatly impact studen
ts learning daily basis wobble chairs provide assistance students di
fficulties focusing attending lessons discussions despite fact stude
nts participate physical activities p e recess gonoodle dance videos
sessions classroom students still energy stand wiggle seats lessons
due special needs beyond students control lot distraction student le
arning not really achieved full potential lack appropriate stimulati
on hinders focus learn class students special needs able sit wobble
chairs whole group small group lessons enable little active bodies m
ove sitting still without disrupting students result students improv
e focus increase student attention learning content areas addition v
isual timer help students actually see allotted time activities bene
fit especially ell students students special needs whenever independ
ent classwork work centers students refer self monitor progress comp
leting assignments encourage use time wisely finish tasks time also
help students smoother transition one activity another donating proj
ect significantly help students special needs equal opportunity lear
n peers behavior issues greatly minimized classroom management optim
ized help set students success looking forward seeing students becom
e active listeners engaged learners always happy go school nannan'
```

In [17]:

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

In [18]:

```
project_data.drop(['project_essay_1','project_essay_2','project_essay_3','projec
t_essay_4'],axis=1,inplace=True)
```

1.4 Preprocessing of `project_title`

- Decontract project titles, remove line breaks and extra spaces, convert everything to lowercase and then remove all the stop words.

In [19]:

```
preprocessed_titles = []

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

100%|██████████| 50000/50000 [00:00<00:00, 59917.74it/s]

In [20]:

```
#after processing, printing random titiles
print(preprocessed_titles[20000])
```

```
wiggle waggle wobble hocus focus
```

In [21]:

```
project_data['clean_title'] = preprocessed_titles
project_data.drop(['project_title'],axis=1,inplace=True)
```

Pre-processing teacher_prefix

In [22]:

```
#remove nan from teacher prefix:
#https://stackoverflow.com/questions/21011777/how-can-i-remove-nan-from-list-python-numpy
def remove_nan(prefix):
    if str(prefix)!='nan':
        pr = str(prefix)
        pr = re.sub("\\.","",pr) #remove dot from the end of prefix
        return pr
    return "none"

cleaned_teacher_prefix = project_data['teacher_prefix'].map(remove_nan)
project_data['clean_teacher_prefix'] = cleaned_teacher_prefix
```

In [23]:

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

Pre-process project_grade_category

- Clean the project grade categories:
  - Convert `Grades 3-5 ==> Grades_3_5`

In [24]:

```
def clean_project_grades(grade):
    grade = re.sub("\-","_",grade)
    grade = re.sub(" ","_",grade)
    return grade.strip()

clean_grades = project_data['project_grade_category'].map(clean_project_grades)
project_data['clean_grade_category'] = clean_grades
```

In [25]:

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

Pre-process project_resource_summary

In [26]:

```
preprocessed_summary = []

for summary in tqdm(project_data['project_resource_summary'].values):
    summary = decontracted(summary)
    summary = summary.replace('\\r', ' ')
    summary = summary.replace('\\"', ' ')
    summary = summary.replace('\\n', ' ')
    summary = re.sub('[^A-Za-z0-9]+', ' ', summary)
    summary = ' '.join(e for e in summary.split() if e.lower() not in stopwords)
    preprocessed_summary.append(summary.lower().strip())
```

```
100%|██████████| 50000/50000 [00:01<00:00, 36396.92it/s]
```

In [27]:

```
print(preprocessed_summary[20000])
print(preprocessed_summary[0])
project_data['clean_resource_summary'] = preprocessed_summary
```

```
students need 6 kore patented wobble chairs time timer focus better
outlet get wiggles non disruptive way help manage time visually
students need flexible seating classroom choose comfortable learn be
st
```

In [28]:

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

In [29]:

```
# Dropping all features we won't need going forward
project_data.drop(['Unnamed: 0','teacher_id'],axis=1,inplace=True)
```

In [30]:

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

Out[30]:

	id	school_state	Date	teacher_number_of_previously_posted_projects	projec
473	p234804	GA	2016-04-27 00:53:00	2	
41558	p137682	WA	2016-04-27 01:05:25	2	

Splitting data into Train, CV and Test

In [31]:

```
from sklearn.model_selection import train_test_split
```

In [32]:

```
Y = project_data['project_is_approved']
X = project_data.drop(['project_is_approved','id'],axis=1)
```

In [33]:

```
print("Shape of X: ",X.shape)
print("Shape of Y: ",Y.shape)
```

```
Shape of X:  (50000, 13)
Shape of Y:  (50000,)
```

In [34]:

```
X.head(2)
```

Out[34]:

	school_state	Date	teacher_number_of_previously_posted_projects	price	quantity
473	GA	2016-04-27 00:53:00	2	481.04	9
41558	WA	2016-04-27 01:05:25	2	17.74	14

In [35]:

```
X_train, X_test, Y_train, Y_test = train_test_split(X,Y,test_size=0.30,stratify=Y)
X_train, X_cv, Y_train, Y_cv = train_test_split(X_train,Y_train,test_size=0.30,stratify=Y_train)
print("Shape of X_train: ", X_train.shape)
print("Shape of Y_train: ",Y_train.shape)
print("Shape of X_cv: ",X_cv.shape)
print("Shape of Y_cv: ",Y_cv.shape)
print("Shape of X_test: ",X_test.shape)
print("Shape of Y_test: ",Y_test.shape)
```

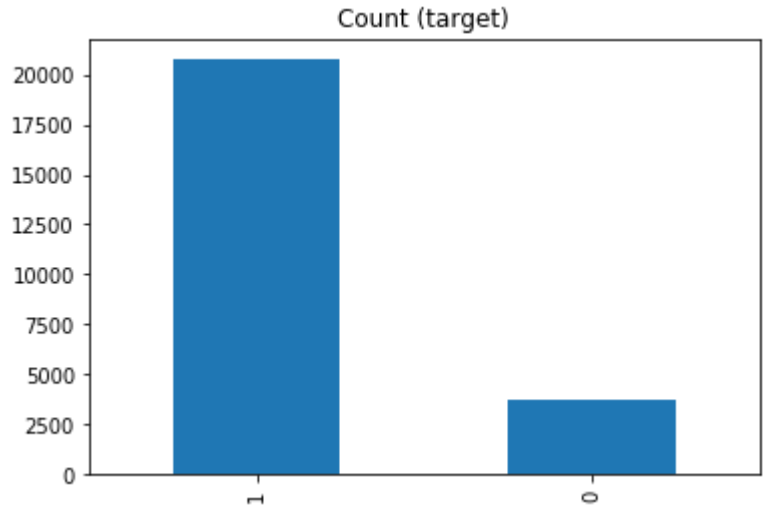
```
Shape of X_train:  (24500, 13)
Shape of Y_train:  (24500,)
Shape of X_cv:  (10500, 13)
Shape of Y_cv:  (10500,)
Shape of X_test:  (15000, 13)
Shape of Y_test:  (15000,)
```

In [36]:

```
X_train['project_is_approved'] = Y_train.values
```

In [37]:

```
X_train['project_is_approved'].value_counts().plot(kind='bar', title='Count (target)');
```



From the above result, it is clear that our training dataset is imbalance with 'project_is_approved'=0 being the minority class. In order to balance our dataset we will perform simple upsampling.

Performing Simple Upsampling on training dataset

Reference: https://elitedatascience.com/imbalanced-classes (https://elitedatascience.com/imbalanced-classes)

```
1. In the assignment video on classroom it was mentioned that we should d
o simple upsampling instead of using SMOTE. So, I did it.


2. I have later learnt that, we just had to train only on the given data
 without doing any kind of upsampling, I read it somewhere, maybe slack,
 can't remember exactly. Seems it still works pretty good if we take care
of the false rate of the minority class.


3. I got aware only after I had run the code and it would have been painf
ul to run it all over again. However, my results with simple upsampling a
re good, as can be seen from the plots.
```

In [38]:

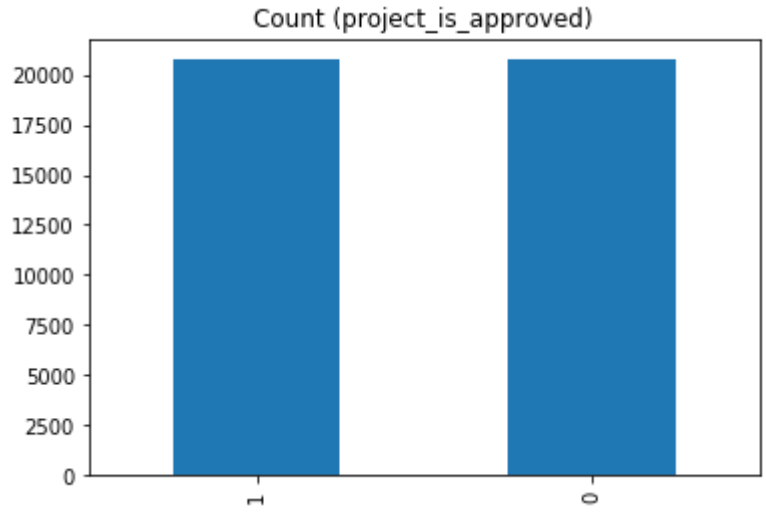
```
from sklearn.utils import resample

# Separate majority and minority classes
X_train_majority = X_train[X_train.project_is_approved==1]
X_train_minority = X_train[X_train.project_is_approved==0]

# Upsample minority class
X_train_minority_upsampled = resample(X_train_minority,
                                      replace=True,     # sample with replacement
                                      n_samples=20720,  # to match majority class
                                      random_state=123) # reproducible results

# Combine majority class with upsampled minority class
X_train_upsampled = pd.concat([X_train_majority, X_train_minority_upsampled])

# Display new class counts
X_train_upsampled.project_is_approved.value_counts().plot(kind='bar', title='Cou
nt (project_is_approved)');
```



Separating X_train and Y_train after upsampling

In [39]:

```
Y_train = X_train_upsampled['project_is_approved']
X_train = X_train_upsampled.drop(['project_is_approved'],axis=1)
print("Shape of X_train after updampling: ", X_train.shape, "Shape of Y_train af
ter upsampling: ",Y_train.shape)
```

```
Shape of X_train after updampling:  (41440, 13) Shape of Y_train aft
er upsampling:  (41440,)
```

1.5 Preparing data for models

In [40]:

```
X_train.columns
```

Out[40]:

```
Index(['school_state', 'Date', 'teacher_number_of_previously_posted_
projects',
       'price', 'quantity', 'clean_categories', 'clean_subcategorie
s', 'essay',
       'clean_essay', 'clean_title', 'clean_teacher_prefix',
       'clean_grade_category', 'clean_resource_summary'],
      dtype='object')
```

we are going to consider

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

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

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

1.5.1 Vectorizing Categorical data

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

One hot encoding: clean_categories

In [63]:

```
from collections import Counter
my_counter = Counter()
for word in X_train['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]))
```

In [42]:

```
# we use count vectorizer to convert the values into one
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(vocabulary=list(sorted_cat_dict.keys()), lowercase=
False, binary=True)
vectorizer.fit(X_train['clean_categories'].values)

X_train_category_ohe = vectorizer.transform(X_train['clean_categories'].values)
X_cv_category_ohe = vectorizer.transform(X_cv['clean_categories'].values)
X_test_category_ohe = vectorizer.transform(X_test['clean_categories'].values)
```

In [43]:

```
print(vectorizer.get_feature_names())
print("Shape of X_train after one hot encodig ",X_train_category_ohe.shape)
print("Shape of X_cv after one hot encodig ",X_cv_category_ohe.shape)
print("Shape of X_test after one hot encodig ",X_test_category_ohe.shape)
print("Print some random encoded categories: ")
print(X_train_category_ohe[0].toarray())
print(X_cv_category_ohe[15].toarray())
print(X_test_category_ohe[15].toarray())
```

```
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLe
arning', 'SpecialNeeds', 'Health_Sports', 'Math_Science', 'Literacy_
Language']
Shape of X_train after one hot encodig  (41440, 9)
Shape of X_cv after one hot encodig  (10500, 9)
Shape of X_test after one hot encodig  (15000, 9)
Print some random encoded categories:
[[0 0 0 0 1 0 1 0 0]]
[[0 0 0 0 1 0 0 0 0]]
[[0 0 0 0 0 0 0 0 1]]
```

One hot encoding: clean_subcategories

In [44]:

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

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

In [45]:

```
# we use count vectorizer to convert the values into one
vectorizer = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()), lowerc
ase=False, binary=True)
vectorizer.fit(X_train['clean_subcategories'].values)

X_train_subcategory_ohe = vectorizer.transform(X_train['clean_subcategories'].va
lues)
X_cv_subcategory_ohe = vectorizer.transform(X_cv['clean_subcategories'].values)
X_test_subcategory_ohe = vectorizer.transform(X_test['clean_subcategories'].valu
es)
```

In [46]:

```
print(vectorizer.get_feature_names())
print("Shape of X_train subcategory after one hot encodig ",X_train_subcategory_
ohe.shape)
print("Shape of X_cv subcategory after one hot encodig ",X_cv_subcategory_ohe.sh
ape)
print("Shape of X_test subcategory after one hot encodig ",X_test_subcategory_oh
e.shape)
print("Print some random encoded categories: ")
print(X_train_subcategory_ohe[0].toarray())
print(X_cv_subcategory_ohe[15].toarray())
print(X_test_subcategory_ohe[10].toarray())
```

```
['Economics', 'CommunityService', 'FinancialLiteracy', 'ParentInvolv
ement', 'Civics_Government', 'Extracurricular', 'ForeignLanguages',
'Warmth', 'Care_Hunger', 'NutritionEducation', 'SocialSciences', 'Pe
rformingArts', 'CharacterEducation', 'College_CareerPrep', 'TeamSpor
ts', 'Music', 'Other', 'History_Geography', 'Health_LifeScience', 'E
arlyDevelopment', 'ESL', 'Gym_Fitness', 'EnvironmentalScience', 'Vis
ualArts', 'Health_Wellness', 'AppliedSciences', 'SpecialNeeds', 'Lit
erature_Writing', 'Mathematics', 'Literacy']
Shape of X_train subcategory after one hot encodig  (41440, 30)
Shape of X_cv subcategory after one hot encodig  (10500, 30)
Shape of X_test subcategory after one hot encodig  (15000, 30)
Print some random encoded categories:
[[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0]]
[[0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]]
[[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1]]
```

One hot encoding: school_state

In [47]:

```
# create a vocabulary for states
unique_states = np.unique(X_train['school_state'].values)

vectorizer = CountVectorizer(vocabulary=unique_states,lowercase=False,binary=Tru
e)
vectorizer.fit(X_train['school_state'].values)

X_train_school_state_ohe = vectorizer.transform(X_train['school_state'].values)
X_cv_school_state_ohe = vectorizer.transform(X_cv['school_state'].values)
X_test_school_state_ohe = vectorizer.transform(X_test['school_state'].values)
```

In [48]:

```
print(vectorizer.get_feature_names())
print("Shape of X_train school_state after one hot encodig ",X_train_school_stat
e_ohe.shape)
print("Shape of X_cv school_state after one hot encodig ",X_cv_school_state_ohe.
shape)
print("Shape of X_test school_state after one hot encodig ",X_test_school_state_
ohe.shape)
print("Print some random encoded school_state: ")
print(X_train_school_state_ohe[0].toarray())
print(X_cv_school_state_ohe[15].toarray())
print(X_test_school_state_ohe[15].toarray())
```

```
['AK', 'AL', 'AR', 'AZ', 'CA', 'CO', 'CT', 'DC', 'DE', 'FL', 'GA',
'HI', 'IA', 'ID', 'IL', 'IN', 'KS', 'KY', 'LA', 'MA', 'MD', 'ME', 'M
I', 'MN', 'MO', 'MS', 'MT', 'NC', 'ND', 'NE', 'NH', 'NJ', 'NM', 'N
V', 'NY', 'OH', 'OK', 'OR', 'PA', 'RI', 'SC', 'SD', 'TN', 'TX', 'U
T', 'VA', 'VT', 'WA', 'WI', 'WV', 'WY']
Shape of X_train school_state after one hot encodig  (41440, 51)
Shape of X_cv school_state after one hot encodig  (10500, 51)
Shape of X_test school_state after one hot encodig  (15000, 51)
Print some random encoded school_state:
[[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0
  1 0 0 0 0 0 0 0 0 0 0 0 0 0]]
[[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0]]
[[0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0]]
```

One hot encoding: teacher_prefix

In [49]:

```
unique_teacher_prefix = np.unique(X_train['clean_teacher_prefix'])

vectorizer = CountVectorizer(vocabulary=unique_teacher_prefix,lowercase=False,bi
nary=True)
vectorizer.fit(X_train['clean_teacher_prefix'].values)

X_train_teacher_prefix_ohe = vectorizer.transform(X_train['clean_teacher_prefix'
].values)
X_cv_teacher_prefix_ohe = vectorizer.transform(X_cv['clean_teacher_prefix'].valu
es)
X_test_teacher_prefix_ohe = vectorizer.transform(X_test['clean_teacher_prefix'].
values)
```

In [50]:

```
print(vectorizer.get_feature_names())
print("Shape of X_train clean_teacher_prefix after one hot encodig ",X_train_tea
cher_prefix_ohe.shape)
print("Shape of X_cv clean_teacher_prefix after one hot encodig ",X_cv_teacher_p
refix_ohe.shape)
print("Shape of X_test clean_teacher_prefix after one hot encodig ",X_test_teach
er_prefix_ohe.shape)
print("Print some random encoded clean_teacher_prefix: ")
print(X_train_teacher_prefix_ohe[0].toarray())
print(X_cv_teacher_prefix_ohe[15].toarray())
print(X_test_teacher_prefix_ohe[15].toarray())
```

```
['Mr', 'Mrs', 'Ms', 'Teacher', 'none']
Shape of X_train clean_teacher_prefix after one hot encodig  (41440,
5)
Shape of X_cv clean_teacher_prefix after one hot encodig  (10500, 5)
Shape of X_test clean_teacher_prefix after one hot encodig  (15000,
5)
Print some random encoded clean_teacher_prefix:
[[0 1 0 0 0]]
[[1 0 0 0 0]]
[[1 0 0 0 0]]
```

One hot encoding: project_grade_category

In [51]:

```
unique_grades = np.unique(X_train['clean_grade_category'])

vectorizer = CountVectorizer(vocabulary=unique_grades,lowercase=False,binary=Tru
e)
vectorizer.fit(X_train['clean_grade_category'].values)


X_train_grade_category_ohe = vectorizer.transform(X_train['clean_grade_category'
].values)
X_cv_grade_category_ohe = vectorizer.transform(X_cv['clean_grade_category'].valu
es)
X_test_grade_category_ohe = vectorizer.transform(X_test['clean_grade_category'].
values)
```

In [52]:

```
print(vectorizer.get_feature_names())
print("Shape of X_train clean_grade_category after one hot encodig ",X_train_gra
de_category_ohe.shape)
print("Shape of X_cv clean_grade_category after one hot encodig ",X_cv_grade_cat
egory_ohe.shape)
print("Shape of X_test clean_grade_category after one hot encodig ",X_test_grade
_category_ohe.shape)
print("Print some random encoded clean_grade_category: ")
print(X_train_grade_category_ohe[0].toarray())
print(X_cv_grade_category_ohe[15].toarray())
print(X_test_grade_category_ohe[15].toarray())
```

```
['Grades_3_5', 'Grades_6_8', 'Grades_9_12', 'Grades_PreK_2']
Shape of X_train clean_grade_category after one hot encodig  (41440,
4)
Shape of X_cv clean_grade_category after one hot encodig  (10500, 4)
Shape of X_test clean_grade_category after one hot encodig  (15000,
4)
Print some random encoded clean_grade_category:
[[0 0 0 1]]
[[0 0 1 0]]
[[0 1 0 0]]
```

1.5.2 Vectorizing Text data

In [53]:

```
#Function to save trained model
def save_model(model,name):
    with open(name,'wb') as f:
        pickle.dump(model,f)
```

1.5.2.1 Bag of words : Essay

In [54]:

```
# We are considering only the words which appeared in at least 10 documents(rows
or projects).
vectorizer = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
vectorizer.fit(X_train['clean_essay'])
#save_model(vectorizer,"DC_essay_BOW.pk")
```

Out[54]:

```
CountVectorizer(analyzer='word', binary=False, decode_error='stric
t',
        dtype=<class 'numpy.int64'>, encoding='utf-8', input='conten
t',
        lowercase=True, max_df=1.0, max_features=5000, min_df=10,
        ngram_range=(1, 4), preprocessor=None, stop_words=None,
        strip_accents=None, token_pattern='(?u)\\b\\w\\w+\\b',
        tokenizer=None, vocabulary=None)
```

In [55]:

```
X_train_essay_bow = vectorizer.transform(X_train['clean_essay'])
X_test_essay_bow = vectorizer.transform(X_test['clean_essay'])
X_cv_essay_bow = vectorizer.transform(X_cv['clean_essay'])

print("Shape of X_train_essay_bow ",X_train_essay_bow.shape)
print("Shape of X_cv_essay_bow ",X_cv_essay_bow.shape)
print("Shape of X_test_essay_bow ",X_test_essay_bow.shape)
```

```
Shape of X_train_essay_bow  (41440, 5000)
Shape of X_cv_essay_bow  (10500, 5000)
Shape of X_test_essay_bow  (15000, 5000)
```

1.5.2.2 Bag of words : Project Title

In [56]:

```
# you can vectorize the title also
# before you vectorize the title make sure you preprocess it
vectorizer = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
vectorizer.fit(X_train['clean_title'])
```

Out[56]:

```
CountVectorizer(analyzer='word', binary=False, decode_error='stric
t',
        dtype=<class 'numpy.int64'>, encoding='utf-8', input='conten
t',
        lowercase=True, max_df=1.0, max_features=5000, min_df=10,
        ngram_range=(1, 4), preprocessor=None, stop_words=None,
        strip_accents=None, token_pattern='(?u)\\b\\w\\w+\\b',
        tokenizer=None, vocabulary=None)
```

In [57]:

```
X_train_title_bow = vectorizer.transform(X_train['clean_title'])
X_cv_title_bow = vectorizer.transform(X_cv['clean_title'])
X_test_title_bow = vectorizer.transform(X_test['clean_title'])

print("Shape of X_train_title_bow ",X_train_title_bow.shape)
print("Shape of X_cv_title_bow ",X_cv_title_bow.shape)
print("Shape of X_test_title_bow ",X_test_title_bow.shape)
```

```
Shape of X_train_title_bow  (41440, 4237)
Shape of X_cv_title_bow  (10500, 4237)
Shape of X_test_title_bow  (15000, 4237)
```

1.5.2.3 TFIDF vectorizer: Essay

In [58]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
vectorizer.fit(X_train['clean_essay'])
```

Out[58]:

```
TfidfVectorizer(analyzer='word', binary=False, decode_error='stric
t',
        dtype=<class 'numpy.float64'>, encoding='utf-8', input='cont
ent',
        lowercase=True, max_df=1.0, max_features=5000, min_df=10,
        ngram_range=(1, 4), norm='l2', preprocessor=None, smooth_idf
=True,
        stop_words=None, strip_accents=None, sublinear_tf=False,
        token_pattern='(?u)\\b\\w\\w+\\b', tokenizer=None, use_idf=T
rue,
        vocabulary=None)
```

In [59]:

```
X_train_essay_tfidf = vectorizer.transform(X_train['clean_essay'])
X_cv_essay_tfidf = vectorizer.transform(X_cv['clean_essay'])
X_test_essay_tfidf = vectorizer.transform(X_test['clean_essay'])

print("Shape of X_train_essay_tfidf ",X_train_essay_tfidf.shape)
print("Shape of X_cv_essay_tfidf ",X_cv_essay_tfidf.shape)
print("Shape of X_test_essay_tfidf ",X_test_essay_tfidf.shape)
```

```
Shape of X_train_essay_tfidf  (41440, 5000)
Shape of X_cv_essay_tfidf  (10500, 5000)
Shape of X_test_essay_tfidf  (15000, 5000)
```

1.5.2.4 TFIDF vectorizer: Project title

In [60]:

```
vectorizer = TfidfVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
vectorizer.fit(X_train['clean_title'])
```

Out[60]:

```
TfidfVectorizer(analyzer='word', binary=False, decode_error='stric
t',
        dtype=<class 'numpy.float64'>, encoding='utf-8', input='cont
ent',
        lowercase=True, max_df=1.0, max_features=5000, min_df=10,
        ngram_range=(1, 4), norm='l2', preprocessor=None, smooth_idf
=True,
        stop_words=None, strip_accents=None, sublinear_tf=False,
        token_pattern='(?u)\\b\\w\\w+\\b', tokenizer=None, use_idf=T
rue,
        vocabulary=None)
```

In [61]:

```
X_train_title_tfidf = vectorizer.transform(X_train['clean_title'])
X_cv_title_tfidf = vectorizer.transform(X_cv['clean_title'])
X_test_title_tfidf = vectorizer.transform(X_test['clean_title'])

print("Shape of X_train_title_tfidf ",X_train_title_tfidf.shape)
print("Shape of X_cv_title_tfidf",X_cv_title_tfidf.shape)
print("Shape of X_test_title_tfidf",X_test_title_tfidf.shape)
```

```
Shape of X_train_title_tfidf  (41440, 4237)
Shape of X_cv_title_tfidf (10500, 4237)
Shape of X_test_title_tfidf (15000, 4237)
```

1.5.2.5 Using Pretrained Models: Avg W2V : Essay

In [62]:

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

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

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

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

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

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

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

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


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

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


'''
```

Out[62]:

```
'\n# Reading glove vectors in python: https://stackoverflow.com/a/38
230349/4084039\ndef loadGloveModel(gloveFile):\n    print ("Loading
Glove Model")\n    f = open(gloveFile,\'r\', encoding="utf8")\n    m
odel = {}\n    for line in tqdm(f):\n        splitLine = line.split
()\n        word = splitLine[0]\n        embedding = np.array([float
(val) for val in splitLine[1:]])\n        model[word] = embedding\n
print ("Done.",len(model)," words loaded!")\n    return model\nmodel
= loadGloveModel(\'glove.42B.300d.txt\')\n\n# ====================
======\nOutput:\n    \nLoading Glove Model\n1917495it [06:32, 4879.6
9it/s]\nDone. 1917495  words loaded!\n\n# ========================
==\n\nwords = []\nfor i in preproced_texts:\n    words.extend(i.spli
t(\' \'))\n\nfor i in preproced_titles:\n    words.extend(i.split(\'
\'))\nprint("all the words in the coupus", len(words))\nwords = set
(words)\nprint("the unique words in the coupus", len(words))\n\ninte
r_words = set(model.keys()).intersection(words)\nprint("The number o
f words that are present in both glove vectors and our coupus",
len(inter_words),"(",np.round(len(inter_words)/len(words)*100,
3),"%)")\n\nwords_courpus = {}\nwords_glove = set(model.keys())\nfor
i in words:\n    if i in words_glove:\n        words_courpus[i] = mo
del[i]\nprint("word 2 vec length", len(words_courpus))\n\n\n# strong
ing variables into pickle files python: http://www.jessicayung.com/h
ow-to-use-pickle-to-save-and-load-variables-in-python/\n\nimport pic
kle\nwith open(\'glove_vectors\', \'wb\') as f:\n    pickle.dump(wor
ds_courpus, f)\n\n\n'
```

In [63]:

```
# stronging variables into pickle files python: http://www.jessicayung.com/how-t
o-use-pickle-to-save-and-load-variables-in-python/
# make sure you have the glove_vectors file
with open('glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())
```

In [64]:

```
# average Word2Vec
def get_avg_w2v(corpus):
    avg_w2v_vectors=[]
    for sentence in tqdm(corpus): # 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)
    return avg_w2v_vectors

X_train_essay_avg_w2v_vectors = get_avg_w2v(X_train['clean_essay'])
X_cv_essay_avg_w2v_vectors = get_avg_w2v(X_cv['clean_essay'])
X_test_essay_avg_w2v_vectors = get_avg_w2v(X_test['clean_essay'])
```

```
100%|██████████| 41440/41440 [00:13<00:00, 3070.86it/s]
100%|██████████| 10500/10500 [00:03<00:00, 3028.58it/s]
100%|██████████| 15000/15000 [00:05<00:00, 2979.02it/s]
```

In [65]:

```
print("Shape of X_train_essay_avg_w2v_vectors",len(X_train_essay_avg_w2v_vectors
),len(X_train_essay_avg_w2v_vectors[0]))
print("Shape of X_cv_essay_avg_w2v_vectors ",len(X_cv_essay_avg_w2v_vectors),len
(X_cv_essay_avg_w2v_vectors[0]))
print("Shape of X_test_essay_avg_w2v_vectors ",len(X_test_essay_avg_w2v_vectors
),len(X_test_essay_avg_w2v_vectors[0]))
```

```
Shape of X_train_essay_avg_w2v_vectors 41440 300
Shape of X_cv_essay_avg_w2v_vectors  10500 300
Shape of X_test_essay_avg_w2v_vectors  15000 300
```

1.5.2.6 Using Pretrained Models: Avg W2V : Project Title

In [66]:

```
X_train_title_avg_w2v_vectors = get_avg_w2v(X_train['clean_title'])
X_cv_title_avg_w2v_vectors = get_avg_w2v(X_cv['clean_title'])
X_test_title_avg_w2v_vectors = get_avg_w2v(X_test['clean_title'])
```

```
100%|██████████| 41440/41440 [00:00<00:00, 58283.52it/s]
100%|██████████| 10500/10500 [00:00<00:00, 59783.44it/s]
100%|██████████| 15000/15000 [00:00<00:00, 58033.75it/s]
```

1.5.2.3 Using Pretrained Models: TFIDF weighted W2V

In [67]:

```
tfidf_model = TfidfVectorizer(ngram_range=(1,4),min_df=10,max_features=5000)
tfidf_model.fit(X_train['clean_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 [68]:

```
# average Word2Vec
def get_tfidf_weighted_w2v(corpus,dictionary,tfidf_words):
    tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in
 this list
    for sentence in tqdm(corpus): # 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/rev
iew
        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.spl
it())) # 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.append(vector)
    return tfidf_w2v_vectors
```

In [69]:

```
X_train_essay_tfidf_w2v_vectors = get_tfidf_weighted_w2v(X_train['clean_essay'].
values,dictionary,tfidf_words)
X_cv_essay_tfidf_w2v_vectors = get_tfidf_weighted_w2v(X_cv['clean_essay'].values
,dictionary,tfidf_words)
X_test_essay_tfidf_w2v_vectors = get_tfidf_weighted_w2v(X_test['clean_essay'].va
lues,dictionary,tfidf_words)
```

```
100%|██████████| 41440/41440 [01:03<00:00, 649.76it/s]
100%|██████████| 10500/10500 [00:16<00:00, 629.31it/s]
100%|██████████| 15000/15000 [00:23<00:00, 626.78it/s]
```

In [70]:

```
print("Shape of X_train_essay_tfidf_w2v_vectors",len(X_train_essay_tfidf_w2v_vec
tors),len(X_train_essay_tfidf_w2v_vectors[0]))
print("Shape of X_cv_essay_tfidf_w2v_vectors ",len(X_cv_essay_tfidf_w2v_vectors
),len(X_cv_essay_tfidf_w2v_vectors[0]))
print("Shape of X_test_essay_tfidf_w2v_vectors ",len(X_test_essay_tfidf_w2v_vect
ors),len(X_test_essay_tfidf_w2v_vectors[0]))
```

```
Shape of X_train_essay_tfidf_w2v_vectors 41440 300
Shape of X_cv_essay_tfidf_w2v_vectors  10500 300
Shape of X_test_essay_tfidf_w2v_vectors  15000 300
```

In [71]:

```
# Similarly you can vectorize for title also
tfidf_model = TfidfVectorizer(ngram_range=(1,4),min_df=10,max_features=5000)
tfidf_model.fit(X_train['clean_title'])

#save_model(tfidf_model,"DC_title_tfidf.pk")

# 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 [72]:

```
X_train_title_tfidf_w2v_vectors = get_tfidf_weighted_w2v(X_train['clean_title'],
dictionary,tfidf_words)
X_cv_title_tfidf_w2v_vectors = get_tfidf_weighted_w2v(X_cv['clean_title'],dictio
nary,tfidf_words)
X_test_title_tfidf_w2v_vectors = get_tfidf_weighted_w2v(X_test['clean_title'],di
ctionary,tfidf_words)
```

```
100%|██████████| 41440/41440 [00:01<00:00, 34337.27it/s]
100%|██████████| 10500/10500 [00:00<00:00, 35958.54it/s]
100%|██████████| 15000/15000 [00:00<00:00, 34019.44it/s]
```

In [73]:

```
print("Shape of X_train_title_tfidf_w2v_vectors",len(X_train_title_tfidf_w2v_vec
tors),len(X_train_title_tfidf_w2v_vectors[0]))
print("Shape of X_cv_title_tfidf_w2v_vectors ",len(X_cv_title_tfidf_w2v_vectors
),len(X_cv_title_tfidf_w2v_vectors[0]))
print("Shape of X_title_title_tfidf_w2v_vectors ",len(X_test_title_tfidf_w2v_vec
tors),len(X_test_title_tfidf_w2v_vectors[0]))
```

```
Shape of X_train_title_tfidf_w2v_vectors 41440 300
Shape of X_cv_title_tfidf_w2v_vectors  10500 300
Shape of X_title_title_tfidf_w2v_vectors  15000 300
```

1.5.3 Vectorizing Numerical features

In [64]:

```
# 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 StandardScaler

# 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)

def replace_nan_with_zero(price):
    if np.isnan(price):
        return 0.0
    return price

def standardize_feature(feature):
    scalar = StandardScaler()
    scalar.fit(feature) # finding the mean and standard deviation of this data
    mean = mean=scalar.mean_[0]
    dev = dev=np.sqrt(scalar.var_[0])
    print(f"Mean : {mean}, Standard deviation : {dev}")
    # Now standardize the data with above maen and variance.
    standardized = scalar.transform(feature)
    return standardized
```

Standardize Price

In [65]:

```
X_train_price_standardized = standardize_feature(X_train['price'].values.reshape
(-1, 1))

X_cv_price_standardized = standardize_feature(X_cv['price'].values.reshape(-1, 1
))

X_test_price_standardized = standardize_feature(X_test['price'].values.reshape(-
1, 1))
```

```
Mean : 323.4271225868726, Standard deviation : 353.9673303743552
Mean : 299.15042666666665, Standard deviation : 394.90914914094793
Mean : 300.8634046666666, Standard deviation : 393.09525045396856
```

In [66]:

```
X_test_price_standardized
```

Out[66]:

```
array([[-0.39907225],
       [-0.67635873],
       [-0.57460222],
       ...,
       [ 0.37936504],
       [ 0.67733353],
       [ 0.45316395]])
```

Standardize teacher_number_of_previously_posted_projects

In [67]:

```
X_train_std_previous_project = standardize_feature(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1, 1))

X_cv_std_previous_project = standardize_feature(X_cv['teacher_number_of_previously_posted_projects'].values.reshape(-1, 1))

X_test_std_previous_project = standardize_feature(X_test['teacher_number_of_previously_posted_projects'].values.reshape(-1, 1))
```

```
Mean : 9.474372586872587, Standard deviation : 24.32024951117668
Mean : 11.645523809523809, Standard deviation : 29.271804473017646
Mean : 11.0674, Standard deviation : 27.895558617337873
```

Assignment 3: Apply KNN

1. **[Task-1] Apply KNN(brute force version) on these feature sets**

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

2. **Hyper paramter tuning to find best K**

   - Find the best hyper parameter which results in 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 write your own for loops to do this task

3. **Representation of results**

   - You need to plot the performance of model both on train data and cross validation data for each hyper parameter, as shown in the figure 
   - Once you find the best hyper parameter, you need to train your model-M using the best hyper-param. Now, find the AUC on test data and plot the ROC curve on both train and test using model-M.
   
   - 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

4. **[Task-2]**

   - Select top 2000 features from feature Set 2 using `SelectKBest` (https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html) and then apply KNN on top of these features
   - ```
     from sklearn.datasets import load_digits
     from sklearn.feature_selection import SelectKBest, chi2
     X, y = load_digits(return_X_y=True)
     X.shape
     X_new = SelectKBest(chi2, k=20).fit_transform(X, y)
     X_new.shape
     ========
     output:
     (1797, 64)
     (1797, 20)
     ```
   - Repeat the steps 2 and 3 on the data matrix after feature selection

5. **Conclusion**
     ```

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

Note: Data Leakage

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

2. K Nearest Neighbor

2.4 Appling KNN on different kind of featurization as mentioned in the instructions

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

Plese note the following points about the code

1. Even though the data has been split into train, test and cv. Only the train data has been use for CV purpose as the library function RandomSearchCV is used.

2. This is because I was facing a lot of memory issues while training the data even on google colab. I tried stacking up train and CV data and gave it for hyperparameter tuning, my local machine died and google colab was behaving weird so decided to go only with training data(~40K points).

3. Certain cells in the following code will be marked as 0, for their running sequence. These were the cells which were first run in google colab and the output had been printed. However, once the session dies in colab, the notebook resets all running counters of cells.

4. The notebook was dying out on colab and colab started behaving abnormally during mnultiple re-runs of code.

5. Later this notebook was moved to Google Cloud Platform in order to be able to execute the code. Even then KNN took a lot of time and resources to run.

6. Cells which were run in colab have not been run in GCP again the interest of time.

2.4.1 Applying KNN brute force on BOW, SET 1

Train Data

In [79]:

```
f1 = X_train_school_state_ohe
f2 = X_train_category_ohe
f3 = X_train_subcategory_ohe
f4 = X_train_grade_category_ohe
f5 = X_train_teacher_prefix_ohe
f6 = np.array(X_train_price_standardized)
f7 = np.array(X_train_std_previous_project)

X_train_knn = hstack((f1,f2,f3,f4,f5,f6,f7,X_train_essay_bow,X_train_title_bow))
```

In [80]:

```
X_train_knn.shape
```

Out[80]:

```
(41440, 9170)
```

Cross Validation Data

In [81]:

```
f1 = X_cv_school_state_ohe
f2 = X_cv_category_ohe
f3 = X_cv_subcategory_ohe
f4 = X_cv_grade_category_ohe
f5 = X_cv_teacher_prefix_ohe
f6 = X_cv_price_standardized
f7 = X_cv_std_previous_project

X_cv_knn = hstack((f1,f2,f3,f4,f5,f6,f7,X_cv_essay_bow,X_cv_title_bow))
```

Test Data

In [82]:

```
f1 = X_test_school_state_ohe
f2 = X_test_category_ohe
f3 = X_test_subcategory_ohe
f4 = X_test_grade_category_ohe
f5 = X_test_teacher_prefix_ohe
f6 = X_test_price_standardized
f7 = X_test_std_previous_project

X_test_knn = hstack((f1,f2,f3,f4,f5,f6,f7,X_test_essay_bow,X_test_title_bow))
X_test_knn.shape
```

Out[82]:

```
(15000, 9170)
```

In [83]:

```
X_tr_knn = vstack((X_train_knn,X_cv_knn))
Y_tr_knn = np.vstack((Y_train.values.reshape(-1,1),Y_cv.values.reshape(-1,1)))
print(X_tr_knn.shape)
print(Y_tr_knn.shape)
print(np.unique(Y_tr_knn,return_counts=True))
print(X_train_knn.shape, Y_train.shape)
```

```
(51940, 9170)
(51940, 1)
(array([0, 1]), array([22340, 29600]))
(41440, 9170) (41440,)
```

In [84]:

```
np.unique(Y_train, return_counts=True)
```

Out[84]:

```
(array([0, 1]), array([20720, 20720]))
```

In [0]:

```
#Intantiate the KNN classifier, n_jobs =-1 detects CPU cores automatically
neigh = KNeighborsClassifier(n_jobs=-1)
#Set parameters for random search
parameters = {'n_neighbors':sp_randint(50, 100)}
# Use randomizedCV to search for the optimal value of K
# Here, we are using roc_auc as our scoring metric since we have imbalanced data
set
clf = RandomizedSearchCV(neigh, parameters, cv=3, scoring='roc_auc', return_trai
n_score=True)
#pass X_train and Y_train as data to search K. Here randomized search will autom
atically split the data
#into stratified samples.
#NOTE: We have therefore, combined X_train_knn and X_cv_knn as X_tr_knn
clf.fit(X_train_knn, Y_train)
```

Out[0]:

```
RandomizedSearchCV(cv=3, error_score='raise-deprecating',
                   estimator=KNeighborsClassifier(algorithm='auto',
                                                  leaf_size=30,
                                                  metric='minkowsk
i',
                                                  metric_params=Non
e, n_jobs=-1,
                                                  n_neighbors=5, p=
2,
                                                  weights='unifor
m'),
                   iid='warn', n_iter=10, n_jobs=None,
                   param_distributions={'n_neighbors': <scipy.stats.
_distn_infrastructure.rv_frozen object at 0x7f27517db278>},
                   pre_dispatch='2*n_jobs', random_state=None, refit
=True,
                   return_train_score=True, scoring='roc_auc', verbo
se=0)
```

In [0]:

```
results = pd.DataFrame.from_dict(clf.cv_results_)
results = results.sort_values(['param_n_neighbors'])

train_auc= results['mean_train_score']
train_auc_std= results['std_train_score']
cv_auc = results['mean_test_score']
cv_auc_std= results['std_test_score']
K =  results['param_n_neighbors']

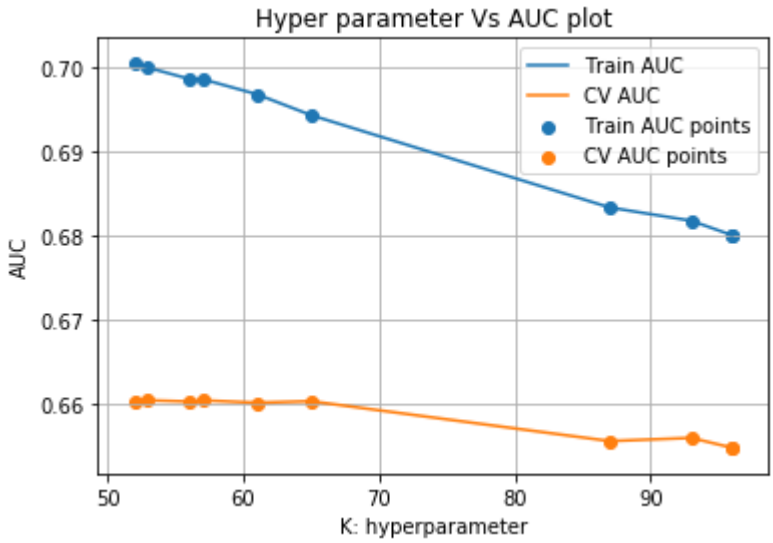
plt.plot(K, tra in_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
# plt.gca().fill_between(K, train_auc - train_auc_std,train_auc + train_auc_std,
alpha=0.2,color='darkblue')

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

plt.scatter(K, train_auc, label='Train AUC points')
plt.scatter(K, cv_auc, label='CV AUC points')


plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("Hyper parameter Vs AUC plot")
plt.grid()
plt.show()

results.head()
```



Out[0]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_n_neighbors	p
0	0.068463	0.012581	54.540463	0.468267	52	{'n_neig
1	0.058378	0.000509	54.506467	0.471826	53	{'n_neig
7	0.057186	0.000275	54.143410	0.388408	56	{'n_neig
3	0.057448	0.001444	54.009878	0.423547	57	{'n_neig
4	0.064553	0.009800	54.082129	0.545744	61	{'n_neig

In [0]:

```
clf.best_estimator_.get_params()
```

Out[0]:

```
{'algorithm': 'auto',
 'leaf_size': 30,
 'metric': 'minkowski',
 'metric_params': None,
 'n_jobs': -1,
 'n_neighbors': 53,
 'p': 2,
 'weights': 'uniform'}
```

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

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

best_k=53
neigh = KNeighborsClassifier(n_neighbors=best_k, n_jobs=-1)
neigh.fit(X_train_knn, Y_train)

y_train_pred = neigh.predict_proba(X_train_knn)
y_test_pred = neigh.predict_proba(X_test_knn)
```

In [0]:

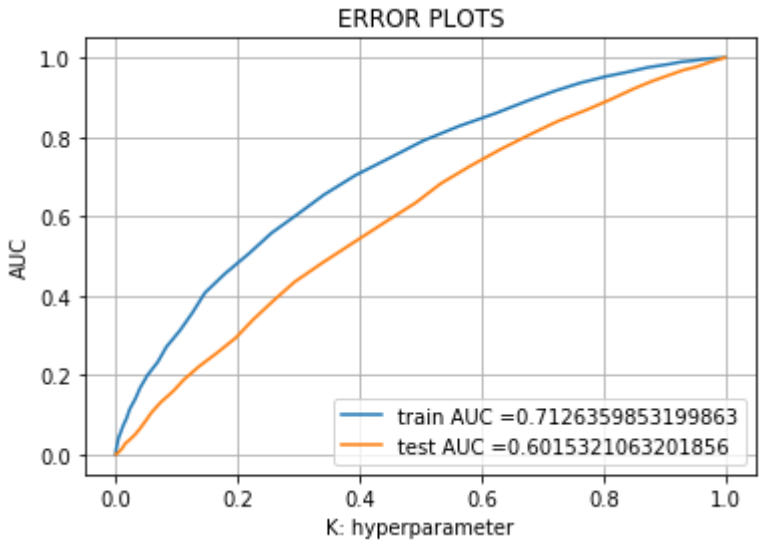
```
print(y_test_pred[:,1].shape)
```

```
(15000,)
```

In [0]:

```
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
# not the predicted outputs
train_fpr, train_tpr, tr_thresholds = roc_curve(Y_train, y_train_pred[:,1])
test_fpr, test_tpr, te_thresholds = roc_curve(Y_test, y_test_pred[:,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()
##############################
#The following labels are incorrect in the plot.
# Title: ROC curve for train and test data for BOW rep of text.
# X-axis is False Positive Rate (FPR)
# Y-axis is True Positive Rate.
##############################
plt.xlabel("K: hyperparameter")# please excuse this
plt.ylabel("AUC") # this
plt.title("ERROR PLOTS") # and this as well
plt.grid()
plt.show()
```



In [0]:

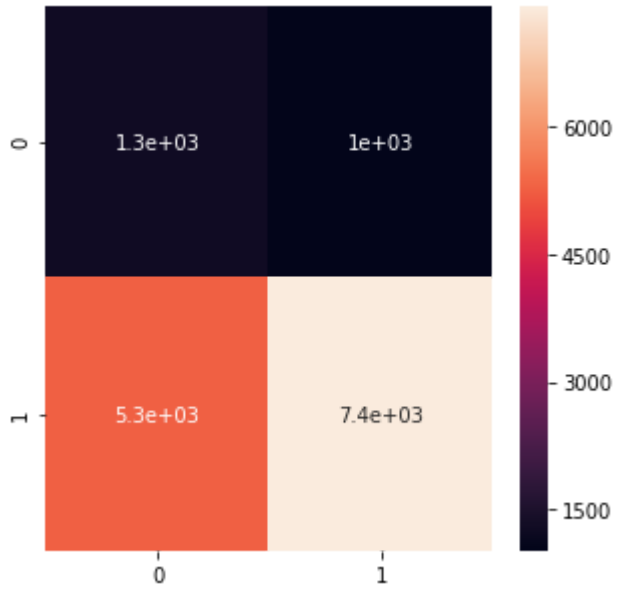
```
y_predicted = []
for prob in y_test_pred[:,1]:
    if prob>0.5:
        y_predicted.append(1)
    else:
        y_predicted.append(0)
```

In [0]:

```
from sklearn.metrics import confusion_matrix
results = confusion_matrix(Y_test, y_predicted)
plt.figure(figsize = (5,5))
sns.heatmap(results, annot=True)
# a better job is done for plotting the confusion matrix without the scientific
 notation.
# I did not know initially that sns would do this :(
```

Out[0]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2740456438>
```



2.4.2 Applying KNN brute force on TFIDF, SET 2

In [0]:

```
#Training Data
f1 = X_train_school_state_ohe
f2 = X_train_category_ohe
f3 = X_train_subcategory_ohe
f4 = X_train_grade_category_ohe
f5 = X_train_teacher_prefix_ohe
f6 = np.array(X_train_price_standardized)
f7 = np.array(X_train_std_previous_project)

X_train_knn = hstack((f1,f2,f3,f4,f5,f6,f7,X_train_essay_tfidf,X_train_title_tfi
df))
print(X_train_knn.shape)
#Testing Data
f1 = X_test_school_state_ohe
f2 = X_test_category_ohe
f3 = X_test_subcategory_ohe
f4 = X_test_grade_category_ohe
f5 = X_test_teacher_prefix_ohe
f6 = X_test_price_standardized
f7 = X_test_std_previous_project

X_test_knn = hstack((f1,f2,f3,f4,f5,f6,f7,X_test_essay_tfidf,X_test_title_tfidf
))
print(X_test_knn.shape)
```

```
(41440, 9301)
(15000, 9301)
```

In [0]:

```
#Intantiate the KNN classifier, n_jobs =-1 detects CPU cores automatically
neigh = KNeighborsClassifier(n_jobs=-1)
#Set parameters for random search
parameters = {'n_neighbors':sp_randint(50, 100)}
# Use randomizedCV to search for the optimal value of K
# Here, we are using roc_auc as our scoring metric since we have imbalanced data
set
clf = RandomizedSearchCV(neigh, parameters, cv=3, scoring='roc_auc', return_trai
n_score=True)
#pass X_train and Y_train as data to search K. Here randomized search will autom
atically split the data
#into stratified samples.
#NOTE: We have therefore, combined X_train_knn and X_cv_knn as X_tr_knn
clf.fit(X_train_knn, Y_train)
clf.best_estimator_.get_params()
```

Out[0]:

```
{'algorithm': 'auto',
 'leaf_size': 30,
 'metric': 'minkowski',
 'metric_params': None,
 'n_jobs': -1,
 'n_neighbors': 53,
 'p': 2,
 'weights': 'uniform'}
```

In [0]:

```
results = pd.DataFrame.from_dict(clf.cv_results_)
results = results.sort_values(['param_n_neighbors'])

train_auc= results['mean_train_score']
train_auc_std= results['std_train_score']
cv_auc = results['mean_test_score']
cv_auc_std= results['std_test_score']
K =  results['param_n_neighbors']

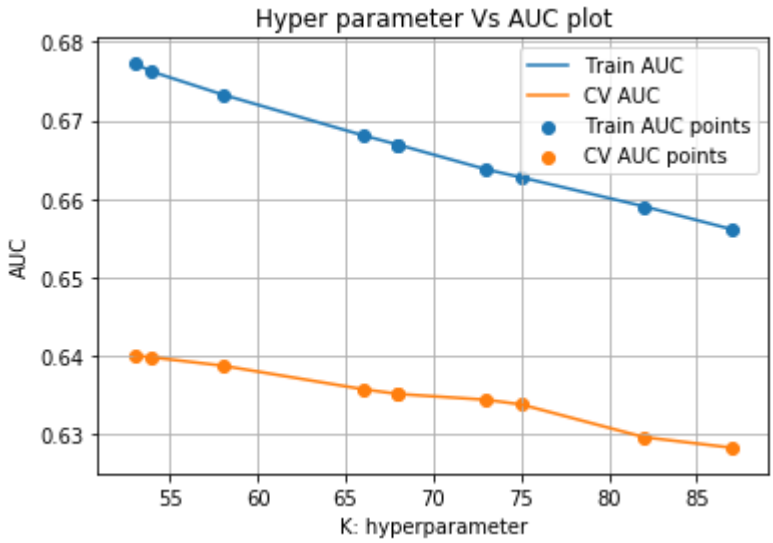
plt.plot(K, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
# plt.gca().fill_between(K, train_auc - train_auc_std,train_auc + train_auc_std,
alpha=0.2,color='darkblue')

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

plt.scatter(K, train_auc, label='Train AUC points')
plt.scatter(K, cv_auc, label='CV AUC points')


plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("Hyper parameter Vs AUC plot")
plt.grid()
plt.show()

results.head()
```



Out[0]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_n_neighbors	p
6	0.044970	0.000736	47.242383	0.312542	53	{'n_neig
2	0.043719	0.000488	46.921051	0.228771	54	{'n_neig
8	0.043391	0.000312	46.844047	0.391627	58	{'n_neig
0	0.049212	0.001544	48.375628	0.525753	66	{'n_neig
5	0.044112	0.000550	47.260100	0.659190	68	{'n_neig

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

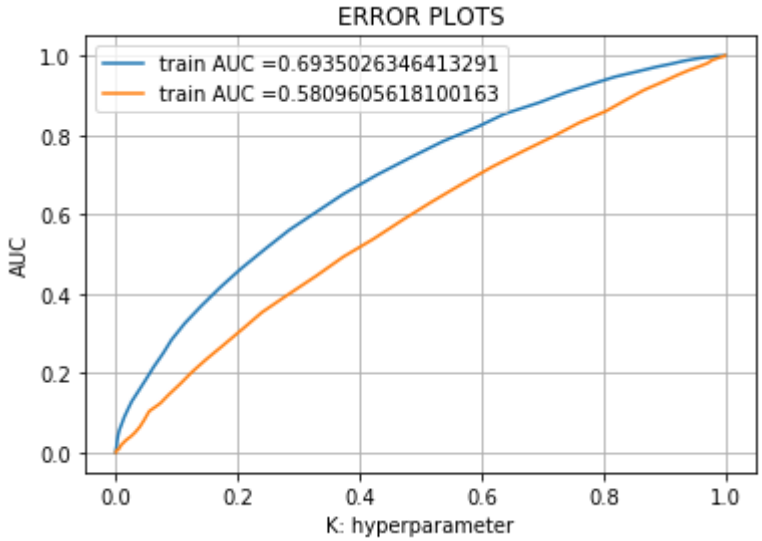
In [0]:

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

best_k=53
neigh = KNeighborsClassifier(n_neighbors=best_k, n_jobs=-1)
neigh.fit(X_train_knn, 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 = neigh.predict_proba(X_train_knn)#batch_predict(neigh, X_tr)
y_test_pred = neigh.predict_proba(X_test_knn)#batch_predict(neigh, X_te)

train_fpr, train_tpr, tr_thresholds = roc_curve(Y_train, y_train_pred[:,1])
test_fpr, test_tpr, te_thresholds = roc_curve(Y_test, y_test_pred[:,1])
```



In [0]:

```
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("FPR")
plt.ylabel("TPR")
plt.title("ROC PLOT for train and test data")
plt.grid()
plt.show()
```

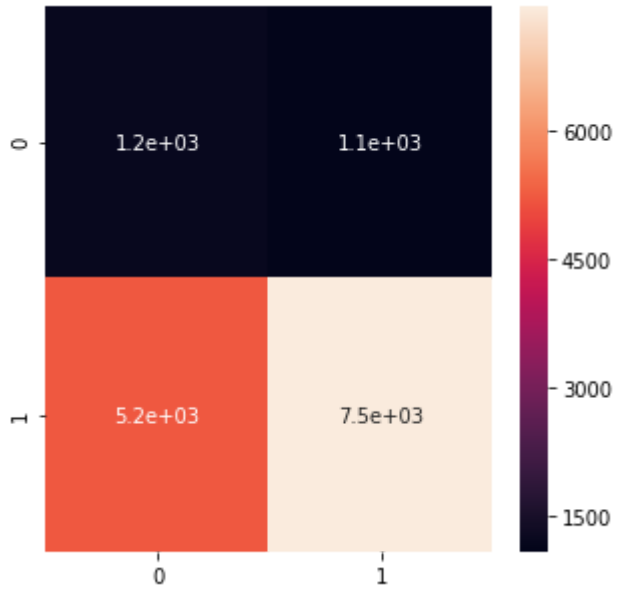
In [0]:

```
y_predicted = []
for prob in y_test_pred[:,1]:
  if prob>0.5:
    y_predicted.append(1)
  else:
    y_predicted.append(0)

results = confusion_matrix(Y_test, y_predicted)
plt.figure(figsize = (5,5))
sns.heatmap(results, annot=True)
```

Out[0]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4fbe18d080>
```



2.4.3 Applying KNN brute force on AVG W2V, SET 3

In [79]:

```
#Training Data
f1 = X_train_school_state_ohe
f2 = X_train_category_ohe
f3 = X_train_subcategory_ohe
f4 = X_train_grade_category_ohe
f5 = X_train_teacher_prefix_ohe
f6 = np.array(X_train_price_standardized)
f7 = np.array(X_train_std_previous_project)

X_train_knn = hstack((f1,f2,f3,f4,f5,f6,f7,X_train_essay_avg_w2v_vectors,X_train
_title_avg_w2v_vectors)) #X_train_title_avg_w2v_vectors

#Testing Data
f1 = X_test_school_state_ohe
f2 = X_test_category_ohe
f3 = X_test_subcategory_ohe
f4 = X_test_grade_category_ohe
f5 = X_test_teacher_prefix_ohe
f6 = X_test_price_standardized
f7 = X_test_std_previous_project

X_test_knn = hstack((f1,f2,f3,f4,f5,f6,f7,X_test_essay_avg_w2v_vectors,X_test_ti
tle_avg_w2v_vectors))
X_test_knn.shape
```

Out[79]:

```
(15000, 701)
```

In [80]:

```
#Intantiate the KNN classifier, n_jobs =-1 detects CPU cores automatically
neigh = KNeighborsClassifier(n_jobs=-1)
#Set parameters for random search
parameters = {'n_neighbors':sp_randint(1, 60)}
# Use randomizedCV to search for the optimal value of K
# Here, we are using roc_auc as our scoring metric since we have imbalanced data
set
clf = RandomizedSearchCV(neigh, parameters, cv=3, scoring='roc_auc', return_trai
n_score=True)
#pass X_train and Y_train as data to search K. Here randomized search will autom
atically split the data
#into stratified samples.
#NOTE: We have therefore, combined X_train_knn and X_cv_knn as X_tr_knn
clf.fit(X_train_knn, Y_train)
```

Out[80]:

```
RandomizedSearchCV(cv=3, error_score='raise-deprecating',
          estimator=KNeighborsClassifier(algorithm='auto', leaf_size
=30, metric='minkowski',
           metric_params=None, n_jobs=-1, n_neighbors=5, p=2,
           weights='uniform'),
          fit_params=None, iid='warn', n_iter=10, n_jobs=None,
          param_distributions={'n_neighbors': <scipy.stats._distn_in
frastructure.rv_frozen object at 0x7f60950b6908>},
          pre_dispatch='2*n_jobs', random_state=None, refit=True,
          return_train_score=True, scoring='roc_auc', verbose=0)
```

In [81]:

```
clf.best_estimator_.get_params()
```

Out[81]:

```
{'algorithm': 'auto',
 'leaf_size': 30,
 'metric': 'minkowski',
 'metric_params': None,
 'n_jobs': -1,
 'n_neighbors': 9,
 'p': 2,
 'weights': 'uniform'}
```

In [82]:

```
results = pd.DataFrame.from_dict(clf.cv_results_)
results = results.sort_values(['param_n_neighbors'])

train_auc= results['mean_train_score']
train_auc_std= results['std_train_score']
cv_auc = results['mean_test_score']
cv_auc_std= results['std_test_score']
K =  results['param_n_neighbors']

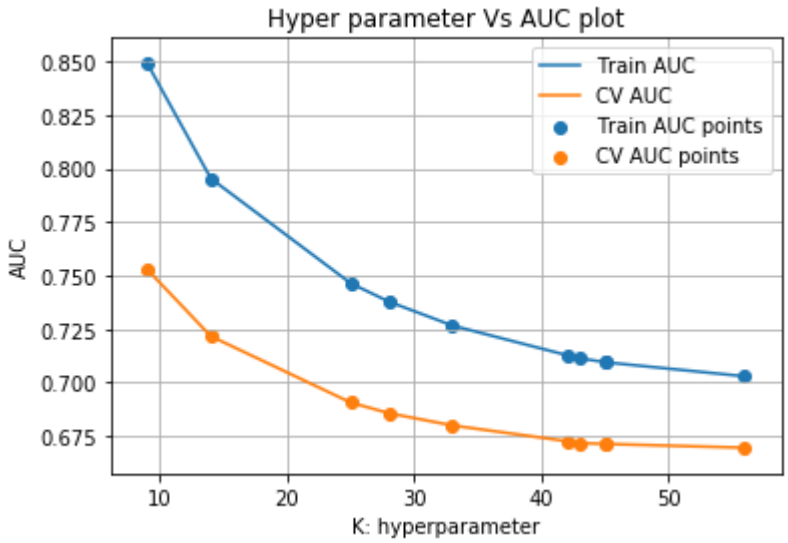
plt.plot(K, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
# plt.gca().fill_between(K, train_auc - train_auc_std,train_auc + train_auc_std,
alpha=0.2,color='darkblue')

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

plt.scatter(K, train_auc, label='Train AUC points')
plt.scatter(K, cv_auc, label='CV AUC points')


plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("Hyper parameter Vs AUC plot")
plt.grid()
plt.show()

results.head()
```



Out[82]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_n_neighbors	p
5	0.399579	0.088868	150.685497	0.325834	9	{'n_neig
9	0.350171	0.025143	151.521350	0.366818	14	{'n_neig
3	0.343742	0.015044	149.274210	0.252616	25	{'n_neig
2	0.379768	0.018767	149.816930	0.197381	28	{'n_neig
4	0.319726	0.005300	150.945687	1.146479	33	{'n_neig

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

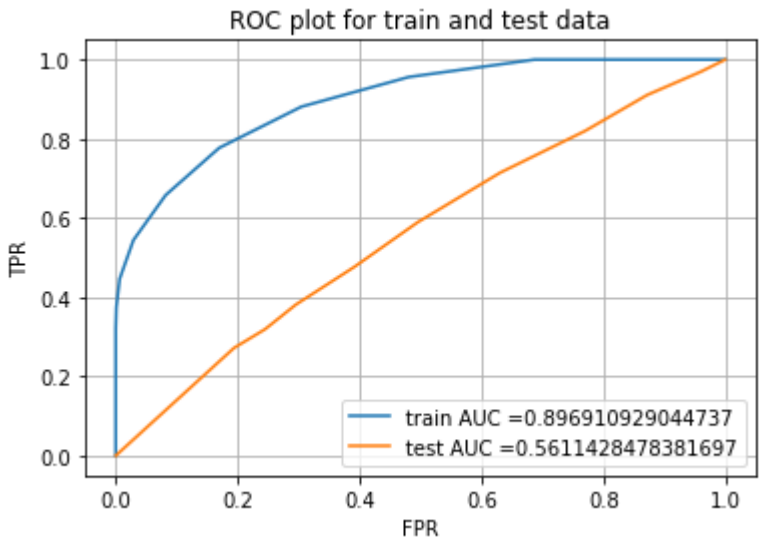
In [83]:

```
best_k = 9
neigh = KNeighborsClassifier(n_neighbors=best_k, n_jobs=-1)
neigh.fit(X_train_knn, Y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimat
es of the positive class
# not the predicted outputs

y_train_pred = neigh.predict_proba(X_train_knn)#batch_predict(neigh, X_tr)
y_test_pred = neigh.predict_proba(X_test_knn)#batch_predict(neigh, X_te)

train_fpr, train_tpr, tr_thresholds = roc_curve(Y_train, y_train_pred[:,1])
test_fpr, test_tpr, te_thresholds = roc_curve(Y_test, y_test_pred[:,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("FPR")
plt.ylabel("TPR")
plt.title("ROC plot for train and test data")
plt.grid()
plt.show()
```



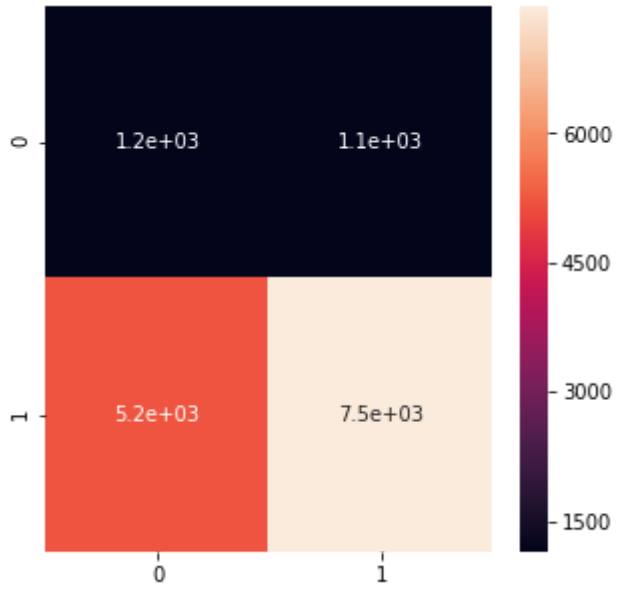
In [84]:

```
y_predicted = []
for prob in y_test_pred[:,1]:
    if prob>0.5:
        y_predicted.append(1)
    else:
        y_predicted.append(0)

results = confusion_matrix(Y_test, y_predicted)
plt.figure(figsize = (5,5))
sns.heatmap(results, annot=True)
```

Out[84]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f60603776a0>
```



2.4.4 Applying KNN brute force on TFIDF W2V, SET 4

In [85]:

```
#Training Data
f1 = X_train_school_state_ohe
f2 = X_train_category_ohe
f3 = X_train_subcategory_ohe
f4 = X_train_grade_category_ohe
f5 = X_train_teacher_prefix_ohe
f6 = np.array(X_train_price_standardized)
f7 = np.array(X_train_std_previous_project)

X_train_knn = hstack((f1,f2,f3,f4,f5,f6,f7,X_train_essay_tfidf_w2v_vectors,X_tra
in_title_tfidf_w2v_vectors))

#Testing Data
f1 = X_test_school_state_ohe
f2 = X_test_category_ohe
f3 = X_test_subcategory_ohe
f4 = X_test_grade_category_ohe
f5 = X_test_teacher_prefix_ohe
f6 = X_test_price_standardized
f7 = X_test_std_previous_project

X_test_knn = hstack((f1,f2,f3,f4,f5,f6,f7,X_test_essay_tfidf_w2v_vectors,X_test_
title_tfidf_w2v_vectors))
X_test_knn.shape
```

Out[85]:

```
(15000, 701)
```

In [86]:

```
#Intantiate the KNN classifier, n_jobs =-1 detects CPU cores automatically
neigh = KNeighborsClassifier(n_jobs=-1)
#Set parameters for random search
parameters = {'n_neighbors':sp_randint(1, 60)}
# Use randomizedCV to search for the optimal value of K
# Here, we are using roc_auc as our scoring metric since we have imbalanced data
set
clf = RandomizedSearchCV(neigh, parameters, cv=3, scoring='roc_auc', return_trai
n_score=True)
#pass X_train and Y_train as data to search K. Here randomized search will autom
atically split the data
#into stratified samples.
#NOTE: We have therefore, combined X_train_knn and X_cv_knn as X_tr_knn
clf.fit(X_train_knn, Y_train)
```

Out[86]:

```
RandomizedSearchCV(cv=3, error_score='raise-deprecating',
          estimator=KNeighborsClassifier(algorithm='auto', leaf_size
=30, metric='minkowski',
           metric_params=None, n_jobs=-1, n_neighbors=5, p=2,
           weights='uniform'),
          fit_params=None, iid='warn', n_iter=10, n_jobs=None,
          param_distributions={'n_neighbors': <scipy.stats._distn_in
frastructure.rv_frozen object at 0x7f604bf57630>},
          pre_dispatch='2*n_jobs', random_state=None, refit=True,
          return_train_score=True, scoring='roc_auc', verbose=0)
```

In [87]:

```
results = pd.DataFrame.from_dict(clf.cv_results_)
results = results.sort_values(['param_n_neighbors'])

train_auc= results['mean_train_score']
train_auc_std= results['std_train_score']
cv_auc = results['mean_test_score']
cv_auc_std= results['std_test_score']
K =  results['param_n_neighbors']

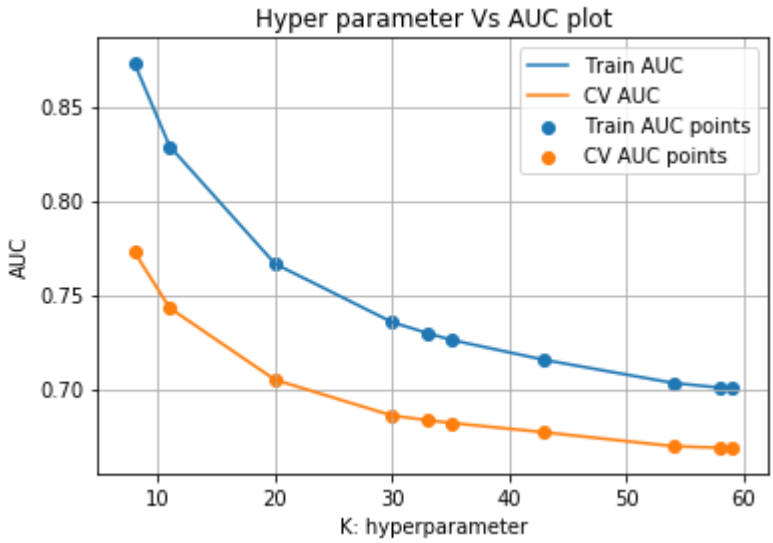
plt.plot(K, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
# plt.gca().fill_between(K, train_auc - train_auc_std,train_auc + train_auc_std,
alpha=0.2,color='darkblue')

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

plt.scatter(K, train_auc, label='Train AUC points')
plt.scatter(K, cv_auc, label='CV AUC points')


plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("Hyper parameter Vs AUC plot")
plt.grid()
plt.show()

results.head()
```



Out[87]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_n_neighbors	p
8	0.367551	0.001222	151.603236	1.258612	8	{'n_neig
3	0.351261	0.020104	150.782440	0.172366	11	{'n_neig
1	0.369437	0.007852	150.360562	0.647708	20	{'n_neig
7	0.340291	0.017774	151.056251	0.149819	30	{'n_neig
9	0.367738	0.002951	150.947676	0.442605	33	{'n_neig

In [88]:

```
clf.best_estimator_.get_params()
```

Out[88]:

```
{'algorithm': 'auto',
 'leaf_size': 30,
 'metric': 'minkowski',
 'metric_params': None,
 'n_jobs': -1,
 'n_neighbors': 8,
 'p': 2,
 'weights': 'uniform'}
```

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

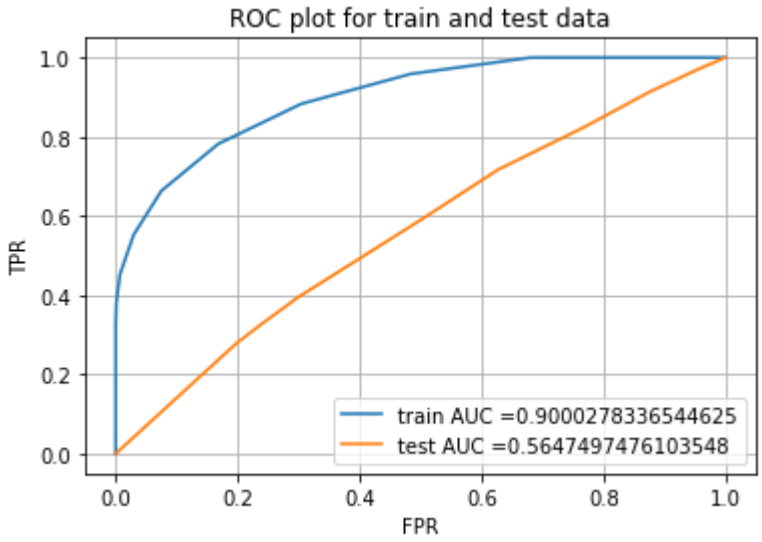
In [91]:

```
best_k = 9 # using odd k value even though the best returned is 8, to avoid ties
neigh = KNeighborsClassifier(n_neighbors=best_k, n_jobs=-1)
neigh.fit(X_train_knn, 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 = neigh.predict_proba(X_train_knn)#batch_predict(neigh, X_tr)
y_test_pred = neigh.predict_proba(X_test_knn)#batch_predict(neigh, X_te)

train_fpr, train_tpr, tr_thresholds = roc_curve(Y_train, y_train_pred[:,1])
test_fpr, test_tpr, te_thresholds = roc_curve(Y_test, y_test_pred[:,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("FPR")
plt.ylabel("TPR")
plt.title("ROC plot for train and test data")
plt.grid()
plt.show()
```



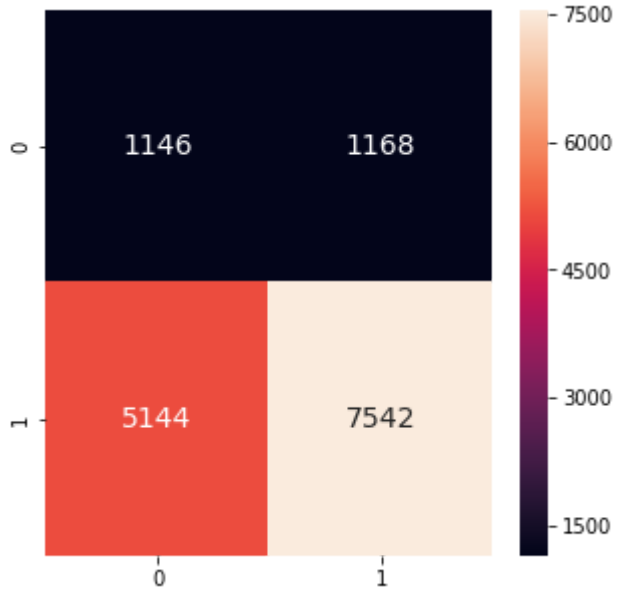
In [92]:

```
y_predicted = []
for prob in y_test_pred[:,1]:
    if prob>0.5:
        y_predicted.append(1)
    else:
        y_predicted.append(0)

results = confusion_matrix(Y_test, y_predicted)
plt.figure(figsize = (5,5))
sns.heatmap(results, annot=True,annot_kws={"size": 14}, fmt='g')
```

Out[92]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f60499203c8>
```



2.5 Feature selection with `SelectKBest`

NOTE: SelectKBest uses chi squared test which assumes a frequency distribution and frequency distribution cannot be negative

Hence, Normalizing price and number_of_previously_submitted_projects.

In [69]:

```
from sklearn import preprocessing
#Use minmax scalar
mm_scaler = preprocessing.MinMaxScaler()
#Apply transform, this is equivalent to X = X-min(X)/Max(X)-Min(X)
X_train_price_normalized = mm_scaler.fit_transform(X_train['price'].values.resha
pe(-1,1))
X_test_price_normalized = mm_scaler.fit_transform(X_test['price'].values.reshape
(-1,1))

X_train_previous_project_normalized = mm_scaler.fit_transform(X_train['teacher_n
umber_of_previously_posted_projects'].values.reshape(-1,1))
X_test_previous_project_normalized = mm_scaler.fit_transform(X_test['teacher_num
ber_of_previously_posted_projects'].values.reshape(-1,1))
```

In [70]:

```
#Training Data
f1 = X_train_school_state_ohe
f2 = X_train_category_ohe
f3 = X_train_subcategory_ohe
f4 = X_train_grade_category_ohe
f5 = X_train_teacher_prefix_ohe
f6 = np.array(X_train_price_normalized)
f7 = np.array(X_train_previous_project_normalized)

X_train_knn = hstack((f1,f2,f3,f4,f5,f6,f7,X_train_essay_tfidf,X_train_title_tfi
df))
print(X_train_knn.shape)
#Testing Data
f1 = X_test_school_state_ohe
f2 = X_test_category_ohe
f3 = X_test_subcategory_ohe
f4 = X_test_grade_category_ohe
f5 = X_test_teacher_prefix_ohe
f6 = X_test_price_normalized
f7 = X_test_previous_project_normalized

X_test_knn = hstack((f1,f2,f3,f4,f5,f6,f7,X_test_essay_tfidf,X_test_title_tfidf
))
print(X_test_knn.shape)
```

```
(41440, 9338)
(15000, 9338)
```

Selecting the best k=2000 features

In [71]:

```
from sklearn.feature_selection import SelectKBest, chi2

vectorizer = SelectKBest(chi2, k=2000).fit(X_train_knn, Y_train)
X_tr_new = vectorizer.transform(X_train_knn)
X_te_new = vectorizer.transform(X_test_knn)
```

In [72]:

```
neigh = KNeighborsClassifier()

parameters = {'n_neighbors':[1, 5, 10, 15, 21, 33, 41, 51, 65, 71, 85, 91]}

clf = GridSearchCV(neigh, parameters, cv=5, scoring='roc_auc')
clf.fit(X_tr_new, Y_train)

results = pd.DataFrame.from_dict(clf.cv_results_)
results = results.sort_values(['param_n_neighbors'])

train_auc= results['mean_train_score']
train_auc_std= results['std_train_score']
cv_auc = results['mean_test_score']
cv_auc_std= results['std_test_score']
K =  results['param_n_neighbors']

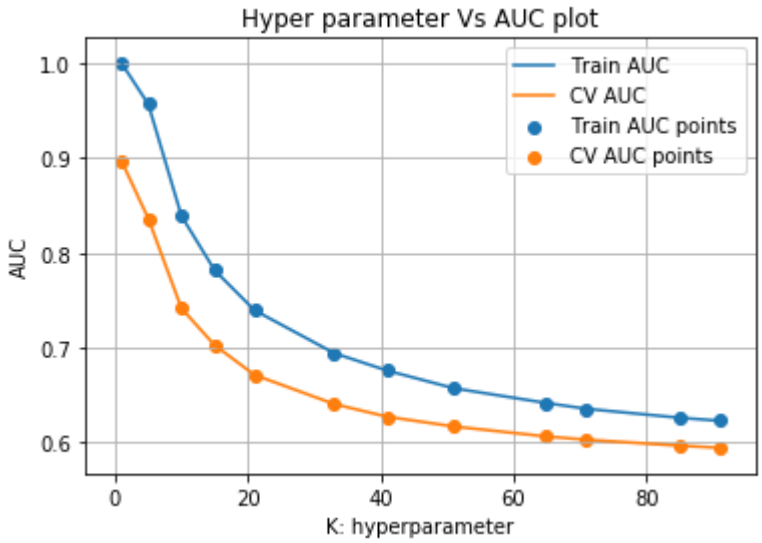
plt.plot(K, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
# plt.gca().fill_between(K, train_auc - train_auc_std,train_auc + train_auc_std,
alpha=0.2,color='darkblue')

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

plt.scatter(K, train_auc, label='Train AUC points')
plt.scatter(K, cv_auc, label='CV AUC points')


plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("Hyper parameter Vs AUC plot")
plt.grid()
plt.show()

results.head()
```



Out[72]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_n_neighbors	p
0	0.012539	0.000366	16.395512	0.211900	1	{'n_neig
1	0.012926	0.001143	18.918546	0.245428	5	{'n_neig
2	0.012445	0.000297	18.856829	0.241521	10	{'n_neig
3	0.012422	0.000375	18.779285	0.232806	15	{'n_neig
4	0.012289	0.000260	19.122557	0.063918	21	{'n_neig

5 rows × 21 columns

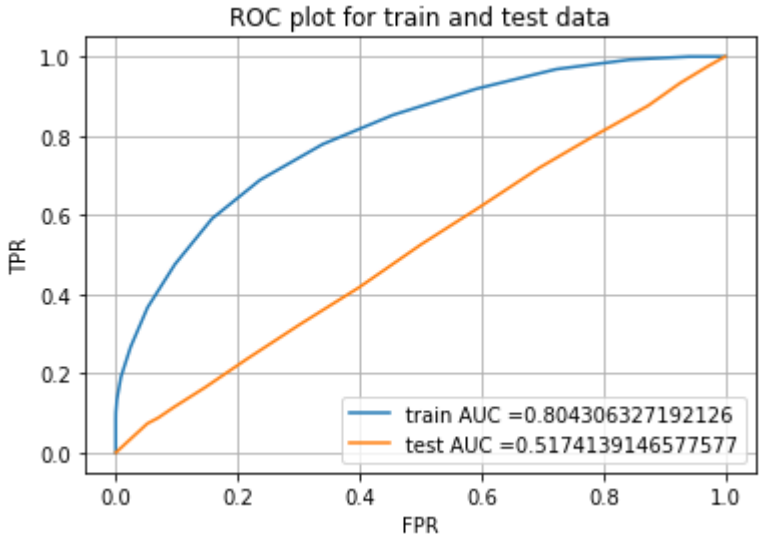
In [74]:

```
best_k = 15
neigh = KNeighborsClassifier(n_neighbors=best_k, n_jobs=-1)
neigh.fit(X_tr_new, 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 = neigh.predict_proba(X_tr_new)#batch_predict(neigh, X_tr)
y_test_pred = neigh.predict_proba(X_te_new)#batch_predict(neigh, X_te)

train_fpr, train_tpr, tr_thresholds = roc_curve(Y_train, y_train_pred[:,1])
test_fpr, test_tpr, te_thresholds = roc_curve(Y_test, y_test_pred[:,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("FPR")
plt.ylabel("TPR")
plt.title("ROC plot for train and test data")
plt.grid()
plt.show()
```



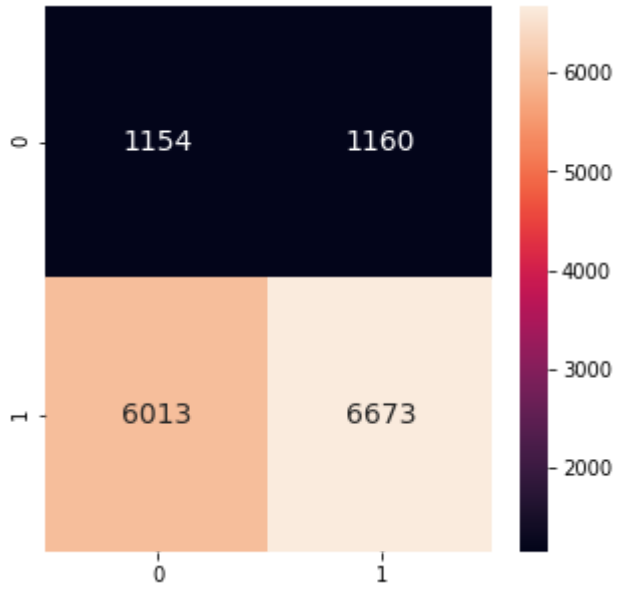
In [75]:

```
y_predicted = []
for prob in y_test_pred[:,1]:
    if prob>0.5:
        y_predicted.append(1)
    else:
        y_predicted.append(0)

results = confusion_matrix(Y_test, y_predicted)
plt.figure(figsize = (5,5))
sns.heatmap(results, annot=True,annot_kws={"size": 14}, fmt='g')
```

Out[75]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f74bf59fb38>
```



3. Conclusions

In [2]:

```
from prettytable import PrettyTable

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

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

x.add_row(["BOW", "Brute", 53, 0.60])
x.add_row(["TFIDF", "Brute", 53, 0.58])
x.add_row(["AVG W2V", "Brute", 9, 0.56])
x.add_row(["TFIDF W2V", "Brute", 9, 0.56])
x.add_row(["TFIDF", "Top 2000", 15, 0.51])

print(x)
```

```
+------------+----------+-----------------+------+
| Vectorizer |  Model   | Hyper Parameter | AUC  |
+------------+----------+-----------------+------+
|    BOW     |  Brute   |        53       | 0.6  |
|   TFIDF    |  Brute   |        53       | 0.58 |
|  AVG W2V   |  Brute   |        9        | 0.56 |
| TFIDF W2V  |  Brute   |        9        | 0.56 |
|   TFIDF    | Top 2000 |        15       | 0.51 |
+------------+----------+-----------------+------+
```