DonorsChoose

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

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

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

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

About the DonorsChoose Data Set

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

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

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

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

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

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

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

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

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

Notes on the Essay Data

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

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

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

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

your neighborhood, and your school are all helpful.

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

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

In [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
print('done')

!pip install -U -q PyDrive
```

```
paramiko missing, opening SSH/SCP/SFTP paths will be disabled.  `pip install paramiko` to suppress
```

```
done
```

In [0]:

```
# (get links for data)   from goolge_drive

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.
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)#4/JwE0irBAjQkylFmncCCCvQqkYTQdvhhN06KJRdK4Koq_ic22bSILcXA

link='https://drive.google.com/open?id=18VAiuw3vfETGcuJOdicvkgQT0pSxF7Wy'

link3='https://drive.google.com/open?id=1Z6bjXmyCaoEzXYo_tRDwLTsfeA2F3K3j'
```

```
flufff, id2 = link3.split('=')
print (id2) # Verify that you have everything after '='
downloaded = drive.CreateFile({'id':id2})
downloaded.GetContentFile('glove_vectors')
```

- **NOTE:**I willl take only 40k datapoints. As i called to ur team sir, i talked abou the isssue of 4 gb ram and laptop is slow. So u told to take 40k*

1.1 Reading Data

In [69]:

```
fluff, id = link.split('=')
print (id) # Verify that you have everything after '='

downloaded = drive.CreateFile({'id':id})
downloaded.GetContentFile('train_data.csv')
project_data = pd.read_csv('train_data.csv',nrows=40000)

print(project_data.shape)
link1='https://drive.google.com/open?id=11uHEj9KOgWD9SU-CPgKyb6VrWqVos4uV'
```

```
18VAiuw3vfETGcuJOdicvkgQT0pSxF7Wy
(40000, 17)
```

In [70]:

```
fluff1, idi = link1.split('=')
print (idi) # Verify that you have everything after '='
downloaded = drive.CreateFile({'id':idi})
downloaded.GetContentFile('resources .csv')
resource_data = pd.read_csv('resources .csv')
print(resource_data .head(3))
```

```
11uHEj9KOgWD9SU-CPgKyb6VrWqVos4uV
        id                                      description  quantity  \
0  p233245  LC652 - Lakeshore Double-Space Mobile Drying Rack         1   
1  p069063        Bouncy Bands for Desks (Blue support pipes)         3   
2  p069063  Cory Stories: A Kid's Book About Living With Adhd         1   

    price  
0  149.00  
1   14.95  
2    8.45  
```

In [72]:

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

print(resource_data.shape)
print(resource_data.columns.values)
```

```
Number of data points in train data (40000, 17)
--------------------------------------------------
The attributes of data : ['Unnamed: 0' 'id' 'teacher_id' 'teacher_prefix' 'school_state' 'Date'
 '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']
(1541272, 4)
['id' 'description' 'quantity' 'price']
```

In [0]:

```
# how to replace elements in list python: https://stackoverflow.com/a/2582163/4084039
cols = ['Date' if x=='project_submitted_datetime' else x for x in list(project_data.columns)]


#sort dataframe based on time pandas python: https://stackoverflow.com/a/49702492/4084039
project_data['Date'] = pd.to_datetime(project_data['project_submitted_datetime'])
project_data.drop('project_submitted_datetime', axis=1, inplace=True)# we drop the col
project_data.sort_values(by=['Date'], inplace=True)# sort the values y date


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


#project_data.head(2)
```

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 [75]:

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

Out[75]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_cate
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	GA	2016-04-27 00:53:00	Grades PreK-2
29891	146723	p099708	c0a28c79fe8ad5810da49de47b3fb491	Mrs.	CA	2016-04-27 01:10:09	Grades 3-5

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"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)
    return phrase
```

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', 'e
ach', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll'
, 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "d
esn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn',
"mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn',
"wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"]
```

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
```

Note: Firstly i clean the project_subject_categories and project_subject_subcategories or preprocess it. then after cleaing i convert to train,test and cv.

Preprocessing of project_subject_categories

In [0]:

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

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

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

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

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

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

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

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

Preprocessing of project_grade_category

In [80]:

```
print(project_data['project_grade_category'][:20])#   we have to remove the graddes from every row
```

```
473      Grades PreK-2
29891       Grades 3-5
23374    Grades PreK-2
7176     Grades PreK-2
35006       Grades 3-5
5145        Grades 3-5
36468    Grades PreK-2
36358    Grades PreK-2
39438    Grades PreK-2
2521     Grades PreK-2
25460       Grades 6-8
34399       Grades 3-5
5364        Grades 6-8
29183       Grades 3-5
33043       Grades 3-5
37160       Grades 6-8
27157      Grades 9-12
38830       Grades 3-5
10985    Grades PreK-2
15560    Grades PreK-2
Name: project_grade_category, dtype: object
```

In [0]:

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

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

grade_cat_list = []
for i in d:
    # consider we have text like this:
    for j in i.split(' '): #     # split by spae
        j=j.replace('Grades','')# clean grades from the row
    grade_cat_list.append(j.strip())



project_data['clean_grade'] = grade_cat_list
project_data.drop(['project_grade_category'], 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_grade'].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]))
```

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 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 with predicted and original labels of test data points

4. **[Task-2]**

   - Select top 2000 features from feature Set 2 using `SelectKBest` 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

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.

2. Preparing our data for the models

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

In [0]:

```
#Splitting Data into train and Test sklearn https://scikit-
learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(project_data,
                                                    project_data['project_is_approved'],
                                                      stratify=  project_data['project_is_approved'],
                                                      test_size = 0.33
                                                     )
```

In [0]:

```
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train,  stratify= y_train,
                                                 test_size = 0.33)
```

In [87]:

```
print(y_train.value_counts())
print(y_test.value_counts())
print(y_cv.value_counts())
```

```
# huge imbalance
```

```
1    15184
0     2772
Name: project_is_approved, dtype: int64
1    11163
0     2037
Name: project_is_approved, dtype: int64
1    7479
0    1365
Name: project_is_approved, dtype: int64
```

In [0]:

```
#droping the y labels
#https://stackoverflow.com/questions/13411544/delete-column-from-pandas-dataframe-by-column-name
#x_train =
X_train.drop(["project_is_approved"], axis = 1, inplace = True)
#x_test =
X_test.drop(["project_is_approved"], axis = 1, inplace = True)
#x_cv =
X_cv.drop(["project_is_approved"], axis = 1, inplace = True)
```

Preprocess train,test and cv data

In [89]:

```
# Preprocessing Train Data of Project Essays

from tqdm import tqdm
train_preprocessed_essays = []
# tqdm is for printing the status bar
for sentance in tqdm(X_train['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)
    train_preprocessed_essays.append(sent.lower().strip())
```

```
100%|██████████| 17956/17956 [00:10<00:00, 1635.96it/s]
```

In [90]:

```
#Preprocessing Test Data of Project Essays

# Combining all the above stundents
from tqdm import tqdm
test_preprocessed_essays = []
# tqdm is for printing the status bar
for sentance in tqdm(X_test['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)
    test_preprocessed_essays.append(sent.lower().strip())
```

```
100%|██████████| 13200/13200 [00:08<00:00, 1567.91it/s]
```

In [91]:

```
#Preprocessing Cross Validation Data  of Project Essays

# Combining all the above stundents
from tqdm import tqdm
from tqdm import tqdm
cv_preprocessed_essays = []
# tqdm is for printing the status bar
for sentance in tqdm(X_cv['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)
    cv_preprocessed_essays.append(sent.lower().strip())
```

```
100%|██████████| 8844/8844 [00:05<00:00, 1625.40it/s]
```

In [92]:

```
#Preprocessing Train Data for Project Titles
from tqdm import tqdm
train_preprocessed_titles = []
# tqdm is for printing the status bar
for sentance in tqdm(X_train['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.lower() not in stopwords)
    train_preprocessed_titles.append(sent.lower().strip())
```

```
100%|██████████| 17956/17956 [00:00<00:00, 33625.80it/s]
```

In [93]:

```
#Preprocessing Test Data for Project Titles
from tqdm import tqdm
test_preprocessed_titles = []
# tqdm is for printing the status bar
for sentance in tqdm(X_test['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.lower() not in stopwords)
    test_preprocessed_titles.append(sent.lower().strip())
```

```
100%|██████████| 13200/13200 [00:00<00:00, 33545.14it/s]
```

In [94]:

```
#Preprocessing CV Data for Project Titles
from tqdm import tqdm
cv_preprocessed_titles = []
# tqdm is for printing the status bar
for sentance in tqdm(X_cv['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.lower() not in stopwords)
    cv_preprocessed_titles.append(sent.lower().strip())
```

```
100%|██████████| 8844/8844 [00:00<00:00, 33462.69it/s]
```

In [95]:

```
cv_preprocessed_titles[1]
```

```
cv_preprocessed_titles[1]
```

Out[95]:

```
'yogata move'
```

2.2 Make Data Model Ready: encoding numerical, categorical features

1. vectorize categorical data

1.*project*_subject_categories convert categorical to vectors*

In [96]:

```
# convert train,cv and test data of clean_categories into vectors


# 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)

# firstly convert fit the train data into the vectoriaer then it learn hte vocablery

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

print(vectorizer.get_feature_names())
```

```
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'SpecialNeeds',
'Health_Sports', 'Math_Science', 'Literacy_Language']
```

In [97]:

```
print("After vectorizations")
print(X_train_cat.shape, y_train.shape)
print(X_cv_cat.shape, y_cv.shape)
print(X_test_cat.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(17956, 9) (17956,)
(8844, 9) (8844,)
(13200, 9) (13200,)
====================================================================================================
```

2.*project*_subject_subcategories convert categorical to vectors*

In [63]:

```
# convert train,cv and test data of clean_categories into vectors


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

# firstly convert fit the train data into the vectoriaer then it learn hte vocablery
# we use the fitted CountVectorizer to convert the text to vector
X_train_subcat = vectorizer.transform(X_train['clean_subcategories'].values)
X_cv_subcat = vectorizer.transform(X_cv['clean_subcategories'].values)
X_test_subcat = vectorizer.transform(X_test['clean_subcategories'].values)

print(vectorizer.get_feature_names())
```

```
['Economics', 'CommunityService', 'FinancialLiteracy', 'ParentInvolvement', 'Extracurricular',
'ForeignLanguages', 'Civics_Government', 'NutritionEducation', 'Warmth', 'Care_Hunger',
'SocialSciences', 'PerformingArts', 'CharacterEducation', 'TeamSports', 'Other',
'College_CareerPrep', 'Music', 'History_Geography', 'Health_LifeScience', 'EarlyDevelopment', 'ESL
', 'Gym_Fitness', 'EnvironmentalScience', 'VisualArts', 'Health_Wellness', 'AppliedSciences',
'SpecialNeeds', 'Literature_Writing', 'Mathematics', 'Literacy']
```

In [64]:

```
print("After vectorizations")
print(X_train_subcat.shape, y_train.shape)
print(X_cv_subcat.shape, y_cv.shape)
print(X_test_subcat.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(17956, 30) (17956,)
(8844, 30) (8844,)
(13200, 30) (13200,)
====================================================================================================
```

*3 school_state convert categorical to vectors**

In [65]:

```
# now time to cont the each words
from collections import Counter
my_counter = Counter()
for word in project_data['school_state'].values:
    my_counter.update(word.split())# count the words

school_state_dict = dict(my_counter)# store in dicionary
sorted_school_state_dict = dict(sorted(school_state_dict.items(), key=lambda kv: kv[1]))# sor it
print(sorted_school_state_dict)
```

```
{'VT': 22, 'WY': 39, 'ND': 54, 'MT': 85, 'RI': 107, 'NH': 107, 'SD': 115, 'AK': 116, 'NE': 121,
'DE': 130, 'WV': 181, 'HI': 183, 'ME': 184, 'NM': 187, 'DC': 204, 'KS': 228, 'ID': 238, 'IA': 241,
'AR': 344, 'CO': 422, 'MN': 443, 'MS': 461, 'OR': 461, 'KY': 492, 'MD': 526, 'NV': 539, 'AL': 620,
'CT': 630, 'UT': 631, 'TN': 632, 'WI': 663, 'VA': 739, 'NJ': 813, 'AZ': 816, 'OK': 836, 'MA': 858,
'LA': 872, 'WA': 891, 'MO': 924, 'IN': 936, 'OH': 960, 'PA': 1139, 'MI': 1185, 'SC': 1449, 'GA': 14
53, 'IL': 1598, 'NC': 1872, 'FL': 2238, 'TX': 2673, 'NY': 2730, 'CA': 5612}
```

In [66]:

```
# convert train,cv and test data of clean_categories into vectors


# we use count vectorizer to convert the values into one
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(vocabulary=list(sorted_school_state_dict.keys()), lowercase=False, bi
nary=True)
vectorizer.fit(project_data['school_state'].values)

# firstly convert fit the train data into the vectoriaer then it learn hte vocablery

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

print(vectorizer.get_feature_names())
```

```
['VT', 'WY', 'ND', 'MT', 'RI', 'NH', 'SD', 'AK', 'NE', 'DE', 'WV', 'HI', 'ME', 'NM', 'DC', 'KS', 'I
D', 'IA', 'AR', 'CO', 'MN', 'MS', 'OR', 'KY', 'MD', 'NV', 'AL', 'CT', 'UT', 'TN', 'WI', 'VA', 'NJ',
'AZ', 'OK', 'MA', 'LA', 'WA', 'MO', 'IN', 'OH', 'PA', 'MI', 'SC', 'GA', 'IL', 'NC', 'FL', 'TX', 'NY
', 'CA']
```

In [67]:

```
print("After vectorizations")
print(X_train_school_state .shape, y_train.shape)
print(X_cv_school_state .shape, y_cv.shape)
print(X_test_school_state .shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(17956, 51) (17956,)
(8844, 51) (8844,)
(13200, 51) (13200,)
====================================================================================================
```

*4. project_grade_category **categorical** to vectors**

In [98]:

```
# convert train,cv and test data of clean_categories into vectors


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

# firstly convert fit the train data into the vectoriaer then it learn hte vocablery

# we use the fitted CountVectorizer to convert the text to vector
X_train_project_grade_category = vectorizer.transform(X_train['clean_grade'].values)
X_cv_project_grade_category = vectorizer.transform(X_cv['clean_grade'].values)
X_test_project_grade_category = vectorizer.transform(X_test['clean_grade'].values)

print(vectorizer.get_feature_names())
```

```
['9-12', '6-8', '3-5', 'PreK-2']
```

In [99]:

```
print("After vectorizations")
print(X_train_project_grade_category  .shape, y_train.shape)
print(X_cv_project_grade_category  .shape, y_cv.shape)
print(X_test_project_grade_category  .shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(17956, 4) (17956,)
(8844, 4) (8844,)
(13200, 4) (13200,)
====================================================================================================
```

_5. teacher_prefix categorical to vectors**_

In [0]:

```
#https://stackoverflow.com/questions/42224700/attributeerror-float-object-has-no-attribute-split
project_data['teacher_prefix']=project_data['teacher_prefix'].fillna(" ")# filll the null values
```

```
project_data['teacher_prefix']=project_data['teacher_prefix'].fillna(' ')# filll the null values
with space


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


# dict sort by value python: https://stackoverflow.com/a/613218/4084039
teacher_cat_dict = dict(my_counter)
sorted_teacher_prefix_dict = dict(sorted(teacher_cat_dict.items(), key=lambda kv: kv[1]))
```

In [103]:

```
# convert train,cv and test data of clean_categories into vectors


# we use count vectorizer to convert the values into one
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(vocabulary=list(sorted_teacher_prefix_dict.keys()), lowercase=False,
binary=True)
vectorizer.fit(project_data['teacher_prefix'].values.astype('U'))

# firstly convert fit the train data into the vectoriaer then it learn hte vocablery

# we use the fitted CountVectorizer to convert the text to vector
X_train_teacher_prefix = vectorizer.transform(X_train['teacher_prefix'].values.astype('U'))
X_cv_teacher_prefix= vectorizer.transform(X_cv['teacher_prefix'].values.astype('U'))
X_test_teacher_prefix = vectorizer.transform(X_test['teacher_prefix'].values.astype('U'))

print(vectorizer.get_feature_names())


# when i executeed this error comes
#np.nan is an invalid document, expected byte or unicode string.
# then iconvert to unicode     just writ .astype('U') after the .values in fit and trainform
#https://stackoverflow.com/questions/39303912/tfidfvectorizer-in-scikit-learn-valueerror-np-nan-is
-an-invalid-document
```

```
['Dr.', 'Teacher', 'Mr.', 'Ms.', 'Mrs.']
```

In [104]:

```
print("After vectorizations")
print(X_train_teacher_prefix   .shape, y_train.shape)
print(X_cv_teacher_prefix   .shape, y_cv.shape)
print(X_test_teacher_prefix   .shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(17956, 5) (17956,)
(8844, 5) (8844,)
(13200, 5) (13200,)
====================================================================================================
```

2.3 Make Data Model Ready: encoding eassay, and project_title

-> preprocess essay and title for gts ready for apply featureization

Apply Baw featurezation *essay*

In [105]:

```
# We are considering only the words which appeared in at least 10 documents(rows or projects).
```

```
vectorizer = CountVectorizer(min_df=10)# its a countvectors used for convert text to vectors
vectorizer.fit(train_preprocessed_essays)# that is learned from trainned  data



# we use the fitted CountVectorizer to convert the text to vector
X_train_bow = vectorizer.transform(train_preprocessed_essays)
X_cv_bow = vectorizer.transform(cv_preprocessed_essays)
X_test_bow = vectorizer.transform(test_preprocessed_essays)



print("After vectorizations")
print(X_train_bow.shape, y_train.shape)
print(X_cv_bow.shape, y_cv.shape)
print(X_test_bow.shape, y_test.shape)
print("="*100)
# so the dimension of alll are the same by using first fit and then transform
```

```
After vectorizations
(17956, 8021) (17956,)
(8844, 8021) (8844,)
(13200, 8021) (13200,)
====================================================================================================
```

Apply Baw featurezation *Title*

In [106]:

```
vectorizer.fit(train_preprocessed_titles)# that is learned from trainned  data



# we use the fitted CountVectorizer to convert the text to vector
X_train_bow_title = vectorizer.transform(train_preprocessed_titles)
X_cv_bow_title= vectorizer.transform(cv_preprocessed_titles)
X_test_bow_title = vectorizer.transform(test_preprocessed_titles)



print("After vectorizations")
print(X_train_bow_title.shape, y_train.shape)
print(X_cv_bow_title.shape, y_cv.shape)
print(X_test_bow_title.shape, y_test.shape)
print("="*100)
# so the dimension of alll are the same by using first fit and then transform
```

```
After vectorizations
(17956, 996) (17956,)
(8844, 996) (8844,)
(13200, 996) (13200,)
====================================================================================================
```

Applly tf-idf featureization *titles*

In [107]:

```
#for titles
from sklearn.feature_extraction.text import TfidfVectorizer
# We are considering only the words which appeared in at least 10 documents(rows or projects).
vectorizer = TfidfVectorizer(min_df=10)# its a countvectors used for convert text to vectors
vectorizer.fit(train_preprocessed_titles)# that is learned from trainned  data
```

```
# we use the fitted CountVectorizer to convert the text to vector
X_train_tf_title = vectorizer.transform(train_preprocessed_titles)
X_cv_tf_title= vectorizer.transform(cv_preprocessed_titles)
X_test_tf_title = vectorizer.transform(test_preprocessed_titles)



print("After vectorizations")
print(X_train_tf_title.shape, y_train.shape)
print(X_cv_tf_title.shape, y_cv.shape)
print(X_test_tf_title.shape, y_test.shape)
print("="*100)
# so the dimension of alll are the same by using first fit and then transform
```

```
After vectorizations
(17956, 996) (17956,)
(8844, 996) (8844,)
(13200, 996) (13200,)
====================================================================================================
```

Applly tf-idf featureization *Essays*

In [108]:

```
#for essay
from sklearn.feature_extraction.text import TfidfVectorizer
# We are considering only the words which appeared in at least 10 documents(rows or projects).
vectorizer = TfidfVectorizer(min_df=10)# its a countvectors used for convert text to vectors
vectorizer.fit(train_preprocessed_essays)# that is learned from trainned  data



# we use the fitted CountVectorizer to convert the text to vector
X_train_tf_essay = vectorizer.transform(train_preprocessed_essays)
X_cv_tf_essay= vectorizer.transform(cv_preprocessed_essays)
X_test_tf_essay = vectorizer.transform(test_preprocessed_essays)



print("After vectorizations")
print(X_train_tf_essay.shape, y_train.shape)
print(X_cv_tf_essay.shape, y_cv.shape)
print(X_test_tf_essay.shape, y_test.shape)
print("="*100)
# so the dimension of alll are the same by using first fit and then transform
```

```
After vectorizations
(17956, 8021) (17956,)
(8844, 8021) (8844,)
(13200, 8021) (13200,)
====================================================================================================
```

1.5.2.3 **Using Pretrained Models: Avg W2V**

In [0]:

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

For Project_essays

In [0]:

```
# average Word2Vec
# compute average word2vec for each review.
def func(wordlist):


  train_avg_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
  for sentence in tqdm(wordlist): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length    # we are taking the 300
dimensions  very large
    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
    train_avg_w2v_vectors.append(vector)

  print(len(train_avg_w2v_vectors))
  print(len(train_avg_w2v_vectors[0]))
  return train_avg_w2v_vectors
```

In [112]:

```
train_avg_w2v_vectors=func(train_preprocessed_essays)
test_avg_w2v_vectors=func(test_preprocessed_essays)
cv_avg_w2v_vectors=func(cv_preprocessed_essays)
```

```
100%|██████████| 17956/17956 [00:05<00:00, 3103.71it/s]
  3%|▎         | 346/13200 [00:00<00:03, 3459.25it/s]
```

```
17956
300
```

```
100%|██████████| 13200/13200 [00:04<00:00, 3286.93it/s]
  4%|▎         | 312/8844 [00:00<00:02, 3119.63it/s]
```

```
13200
300
```

```
100%|██████████| 8844/8844 [00:02<00:00, 3270.78it/s]
```

```
8844
300
```

For titles

In [114]:

```
cv_avg_w2v_vectors_title=func(cv_preprocessed_titles)


test_avg_w2v_vectors_title=func(test_preprocessed_titles)


train_avg_w2v_vectors_title=func(train_preprocessed_titles)
```

```
100%|██████████| 8844/8844 [00:00<00:00, 48595.97it/s]
 46%|████▌     | 6044/13200 [00:00<00:00, 60424.50it/s]
```

```
8844
300
```

```
100%|██████████| 13200/13200 [00:00<00:00, 56588.54it/s]
 32%|███       | 5676/17956 [00:00<00:00, 56755.45it/s]
```

```
13200
300
```

```
100%|██████████| 17956/17956 [00:00<00:00, 56904.66it/s]
```

```
17956
300
```

1.5.2.3 Using Pretrained Models: TFIDF weighted W2V

In [0]:

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

In [0]:

```
# average Word2Vec
# compute average word2vec for each review.
def tf_idf_done(word_list):

  train_title_tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
  for sentence in tqdm(word_list): # 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():#.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    train_title_tfidf_w2v_vectors.append(vector)

  print(len(train_title_tfidf_w2v_vectors))
  print(len(train_title_tfidf_w2v_vectors[0]))
  return train_title_tfidf_w2v_vectors
```

In [118]:

```
#For essays

#train_title_tfidf_w2v_vectors=tf_idf_done(tf_idf_train_title)
#train_title_tfidf_w2v_vector
train_tfidf_w2v_vectors=tf_idf_done(train_preprocessed_essays)
test_tfidf_w2v_vectors=tf_idf_done(test_preprocessed_essays)
cv_tfidf_w2v_vectors=tf_idf_done(cv_preprocessed_essays)
```

```
100%|██████████| 17956/17956 [00:33<00:00, 539.89it/s]
  0%|          | 58/13200 [00:00<00:22, 578.19it/s]
```

```
17956
300
```

```
100%|██████████| 13200/13200 [00:24<00:00, 535.04it/s]
  1%|          | 45/8844 [00:00<00:19, 445.11it/s]
```

```
13200
300
```

```
100%|██████████| 8844/8844 [00:16<00:00, 524.19it/s]
```

```
8844
300
```

In [119]:

```
#train_title_tfidf_w2v_vectors=tf_idf_done(tf_idf_train_title)
#train_title_tfidf_w2v_vector
train_title_tfidf_w2v_vectors=tf_idf_done(train_preprocessed_titles)
test_title_tfidf_w2v_vectors=tf_idf_done(test_preprocessed_titles)
cv_title_tfidf_w2v_vectors=tf_idf_done(cv_preprocessed_titles)
```

```
100%|██████████| 17956/17956 [00:00<00:00, 23478.38it/s]
 15%|█         | 1975/13200 [00:00<00:00, 19746.82it/s]
```

```
17956
300
```

```
100%|██████████| 13200/13200 [00:00<00:00, 26579.91it/s]
 32%|███       | 2847/8844 [00:00<00:00, 28460.05it/s]
```

```
13200
300
```

```
100%|██████████| 8844/8844 [00:00<00:00, 26974.84it/s]
```

```
8844
300
```

1.5.3 Vectorizing Numerical features¶

In [120]:

```
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')
print(price_data.head(2))


#merging
# we also have to do this in tran,test and cv
# so also merge the resource data with the trian,cv and test

X_train = pd.merge(X_train, price_data, on = "id", how = "left")
#print(x_train.columns)
X_test = pd.merge(X_test, price_data, on = "id", how = "left")
X_cv = pd.merge(X_cv, price_data, on = "id", how = "left")
```

```
        id   price  quantity
0  p000001  459.56         7
1  p000002  515.89        21
```

Standadized price for the train,test and cv

For train

In [121]:

```
# 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)

price_scalar = StandardScaler()

price_scalar.fit(X_train['price'].values.reshape(-1,1)) # finding the mean and standard deviation of this data
print(f"Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(price_scalar.var_[0])}")

# Now standardize the data with above maen and variance.
train_price_standar = price_scalar.transform(X_train['price'].values.reshape(-1, 1))
train_price_standar
```

```
Mean : 297.69971207395855, Standard deviation : 377.51114848662314
```

Out[121]:

```
array([[-0.65364881],
       [-0.1820336 ],
       [-0.46970722],
       ...,
       [ 2.58048085],
       [-0.37453122],
       [ 1.69939428]])
```

For test

In [122]:

```
# Now standardize the data with above maen and variance.
test_price_standar = price_scalar.transform(X_test['price'].values.reshape(-1, 1))
test_price_standar
```

Out[122]:

```
array([[-0.77467835],
       [-0.06685819],
       [ 0.34841431],
       ...,
       [-0.45749566],
       [-0.21628424],
       [-0.51786474]])
```

For cv

In [123]:

```
# Now standardize the data with above maen and variance.
cv_price_standar = price_scalar.transform(X_cv['price'].values.reshape(-1, 1))
test_price_standar
```

Out[123]:

```
array([[-0.77467835],
       [-0.06685819],
       [ 0.34841431],
       ...,
       [-0.45749566],
       [-0.21628424],
       [-0.51786474]])
```

Shapes"

In [124]:

```
print(train_price_standar.shape, y_train.shape)
print(test_price_standar.shape, y_test.shape)
print(cv_price_standar.shape, y_cv.shape)
```

```
(17956, 1) (17956,)
(13200, 1) (13200,)
(8844, 1) (8844,)
```

Stadadized Previous_year_tecaher_projects train,test and cv

For train

In [125]:

```
# previous_year_projects
price_scalar.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1)) # fi
nding the mean and standard deviation of this data
print(f"Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(price_scalar.var_[0])}")

# Now standardize the data with above maen and variance.
train_prev_proj_standar =
price_scalar.transform(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,
1))
train_prev_proj_standar
```

```
Mean : 11.534640231677434, Standard deviation : 28.93578306988893
```

Out[125]:

```
array([[-0.39862893],
       [-0.36406964],
       [-0.12215464],
       ...,
       [-0.19127321],
       [-0.36406964],
       [-0.32951036]])
```

For test

In [126]:

```
# Now standardize the data with above maen and variance.
test_prev_proj_standar =
price_scalar.transform(X_test['teacher_number_of_previously_posted_projects'].values.reshape(-1, 1)
)
test_prev_proj_standar
```

Out[126]:

```
array([[-0.36406964],
       [-0.26039179],
       [-0.39862893],
       ...,
       [-0.39862893],
       [-0.39862893],
       [ 1.01830179]])
```

For cv

In [127]:

```
# Now standardize the data with above maen and variance.
cv_prev_proj_standar = price_scalar.transform(X_cv['teacher_number_of_previously_posted_projects']
.values.reshape(-1, 1))
cv_prev_proj_standar
```

Out[127]:

```
array([[-0.32951036],
       [-0.26039179],
       [-0.32951036],
       ...,
       [-0.36406964],
       [-0.15671393],
       [-0.36406964]])
```

Shapes of all

In [128]:

```
print(train_prev_proj_standar.shape, y_train.shape)
print(test_prev_proj_standar.shape, y_test.shape)
print(cv_prev_proj_standar.shape, y_cv.shape)
```

```
(17956, 1) (17956,)
(13200, 1) (13200,)
(8844, 1) (8844,)
```

Standaized the Quantity column of the train,test and cv

For train

In [129]:

```
price_scalar.fit(X_train['quantity'].values.reshape(-1,1)) # finding the mean and standard
deviation of this data
print(f"Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(price_scalar.var_[0])}")

# Now standardize the data with above maen and variance.
train_qnty_standar = price_scalar.transform(X_train['quantity'].values.reshape(-1, 1))
train_qnty_standar
```

```
Mean : 17.024337268879485, Standard deviation : 26.994430632551296
```

Out[129]:

```
array([[ 0.85112604],
       [-0.51952706],
       [-0.33430367],
       ...,
       [-0.03794624],
       [-0.33430367],
       [-0.33430367]])
```

For cv

In [130]:

```
# Now standardize the data with above maen and variance.
cv_qnty_standar = price_scalar.transform(X_cv['quantity'].values.reshape(-1, 1))
cv_qnty_standar
```

Out[130]:

```
array([[-0.51952706],
       [ 1.14748346],
       [-0.44543771],
       ...,
       [-0.44543771],
       [-0.26021431],
       [-0.14908028]])
```

For test

For test

In [132]:

```
# Now standardize the data with above maen and variance.
test_qnty_standar = price_scalar.transform(X_test['quantity'].values.reshape(-1, 1))
test_qnty_standar
```

Out[132]:

```
array([[ 0.66590264],
       [-0.48248238],
       [-0.33430367],
       ...,
       [-0.18612496],
       [-0.26021431],
       [ 0.25841118]])
```

Shapes

In [133]:

```
print(train_qnty_standar.shape, y_train.shape)
print(test_qnty_standar.shape, y_test.shape)
print(cv_qnty_standar.shape, y_cv.shape)
```

```
(17956, 1) (17956,)
(13200, 1) (13200,)
(8844, 1) (8844,)
```

Merge all features whchh we clean till now

All categorical:

In [134]:

```
#project_categories
print("--------------------------------------------------------------------------------")
print("Shape of Train  ->",X_train_cat.shape)
print("Shape of test  ->",X_test_cat.shape)
print("Shape of cv ->",X_cv_cat.shape)
print("--------------------------------------------------------------------------------")
```

```
--------------------------------------------------------------------------------
Shape of Train  -> (17956, 9)
Shape of test  -> (13200, 9)
Shape of cv -> (8844, 9)
--------------------------------------------------------------------------------
```

In [135]:

```
#project_subcategories
print("--------------------------------------------------------------------------------")
print("Shape of Train  ->",X_train_subcat.shape)
print("Shape of test  ->",X_test_subcat.shape)
print("Shape of cv ->",X_cv_subcat.shape)
print("--------------------------------------------------------------------------------")
```

```
--------------------------------------------------------------------------------
Shape of Train  -> (17956, 30)
Shape of test  -> (13200, 30)
Shape of cv -> (8844, 30)
--------------------------------------------------------------------------------
```

In [136]:

```
#project_school_state
```

```
print("---------------------------------------------------------------------------------")
print("Shape of Train  ->",X_train_school_state.shape)
print("Shape of test  ->",X_test_school_state.shape)
print("Shape of cv ->",X_cv_school_state.shape)
print("---------------------------------------------------------------------------------")
```

```
---------------------------------------------------------------------------------
Shape of Train  -> (17956, 51)
Shape of test  -> (13200, 51)
Shape of cv -> (8844, 51)
---------------------------------------------------------------------------------
```

In [137]:

```
#project_grade_category
print("---------------------------------------------------------------------------------")
print("Shape of Train  ->",X_train_project_grade_category.shape)
print("Shape of test  ->",X_test_project_grade_category.shape)
print("Shape of cv ->",X_cv_project_grade_category.shape)
print("---------------------------------------------------------------------------------")
```

```
---------------------------------------------------------------------------------
Shape of Train  -> (17956, 4)
Shape of test  -> (13200, 4)
Shape of cv -> (8844, 4)
---------------------------------------------------------------------------------
```

In [138]:

```
#project_teacher_prefix
print("---------------------------------------------------------------------------------")
print("Shape of Train  ->",X_train_teacher_prefix.shape)
print("Shape of test  ->",X_test_teacher_prefix.shape)
print("Shape of cv ->",X_cv_teacher_prefix.shape)
print("---------------------------------------------------------------------------------")
```

```
---------------------------------------------------------------------------------
Shape of Train  -> (17956, 5)
Shape of test  -> (13200, 5)
Shape of cv -> (8844, 5)
---------------------------------------------------------------------------------
```

All numerical:

In [139]:

```
#project_quantity
print("---------------------------------------------------------------------------------")
print("Shape of Train  ->",train_qnty_standar.shape)
print("Shape of test  ->",test_qnty_standar.shape)
print("Shape of cv ->",cv_qnty_standar.shape)
print("---------------------------------------------------------------------------------")
```

```
---------------------------------------------------------------------------------
Shape of Train  -> (17956, 1)
Shape of test  -> (13200, 1)
Shape of cv -> (8844, 1)
---------------------------------------------------------------------------------
```

In [140]:

```
#project_price
print("---------------------------------------------------------------------------------")
print("Shape of Train  ->",train_price_standar.shape)
print("Shape of test  ->",test_price_standar.shape)
print("Shape of cv ->",cv_price_standar.shape)
print("---------------------------------------------------------------------------------")
```

```
---------------------------------------------------------------------------------
Shape of Train  -> (17956, 1)
Shape of Train  -> (17956, 1)
Shape of test  -> (13200, 1)
Shape of cv -> (8844, 1)
--------------------------------------------------------------------------------
```

In [141]:

```
##project_previous_year_teacher_projects
print("--------------------------------------------------------------------------------")
print("Shape of Train  ->",train_prev_proj_standar.shape)
print("Shape of test  ->",test_prev_proj_standar.shape)
print("Shape of cv ->",cv_prev_proj_standar.shape)
print("--------------------------------------------------------------------------------")
```

```
--------------------------------------------------------------------------------
Shape of Train  -> (17956, 1)
Shape of test  -> (13200, 1)
Shape of cv -> (8844, 1)
--------------------------------------------------------------------------------
```

All featurization Bow,tf-idf etc ESSAY AND TITLES:

For BOW:

In [142]:

```
#BOW Project_Essays
print("- "*50)
print("Shape of train ",X_train_bow.shape)
print("Shape of test  ",X_test_bow.shape)
print("Shape of cv  ",X_cv_bow.shape)
print("- "*50)
#BOW Project_Titles
print("Shape of train  ",X_train_bow_title.shape)
print("Shape of test  ",X_test_bow_title.shape)
print("Shape of cv  ",X_cv_bow_title.shape)
print("- "*50)
```

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
-
Shape of train  (17956, 8021)
Shape of test   (13200, 8021)
Shape of cv   (8844, 8021)
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
-
Shape of train   (17956, 996)
Shape of test   (13200, 996)
Shape of cv   (8844, 996)
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
-
```

For tf-idf:

In [143]:

```
#TFIDF Project_Essays
print("- "*50)
print("Shape of train ",X_train_tf_essay.shape)
print("Shape of test  ",X_test_tf_essay.shape)
print("Shape of cv  ",X_cv_tf_essay.shape)
print("- "*50)
#TFIDF Project_Title

print("Shape of train  ",X_train_tf_title.shape)
print("Shape of test  ",X_test_tf_title.shape)
print("Shape of  cv ",X_cv_tf_title.shape)
print("- "*50)
```

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
-
Shape of train  (17956, 8021)
Shape of test   (13200, 8021)
Shape of cv   (8844, 8021)
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
-
Shape of train   (17956, 996)
Shape of test   (13200, 996)
Shape of  cv  (8844, 996)
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
-
```

For avg_w2v:

In [0]:

```
# list to np.array
train_avg_w2v_vectors_title=np.array(train_avg_w2v_vectors_title)
test_avg_w2v_vectors_title=np.array(test_avg_w2v_vectors_title)
cv_avg_w2v_vectors_title=np.array(cv_avg_w2v_vectors_title)



train_avg_w2v_vectors=np.array(train_avg_w2v_vectors)
test_avg_w2v_vectors=np.array(test_avg_w2v_vectors)
cv_avg_w2v_vectors=np.array(cv_avg_w2v_vectors)
```

In [145]:

```
#TFIDF Project_Essays
print("- "*50)
print("Shape of train ",train_avg_w2v_vectors.shape)#train_avg_w2v_vectors_title
print("Shape of test  ",test_avg_w2v_vectors.shape)
print("Shape of cv  ",cv_avg_w2v_vectors.shape)
print("- "*50)
#TFIDF Project_Title

print("Shape of train  ",train_avg_w2v_vectors_title.shape)
print("Shape of test  ",test_avg_w2v_vectors_title.shape)
print("Shape of  cv ",cv_avg_w2v_vectors_title.shape)
print("- "*50)
```

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
-
Shape of train  (17956, 300)
Shape of test   (13200, 300)
Shape of cv   (8844, 300)
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
-
Shape of train   (17956, 300)
Shape of test   (13200, 300)
Shape of  cv  (8844, 300)
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
-
```

For tf-idf word2vec:

In [0]:

```
# list to np.array
train_title_tfidf_w2v_vectors=np.array(train_title_tfidf_w2v_vectors)
test_title_tfidf_w2v_vectors=np.array(test_title_tfidf_w2v_vectors)
cv_title_tfidf_w2v_vectors=np.array(cv_title_tfidf_w2v_vectors)



train_essay_tfidf_w2v_vectors=np.array(train_tfidf_w2v_vectors)
test_essay_tfidf_w2v_vectors=np.array(test_tfidf_w2v_vectors)
cv_essay_tfidf_w2v_vectors=np.array(cv_tfidf_w2v_vectors)
```

In [149]:

```
#TFIDF Project_Essays
```

```
print("- "*50)
print("Shape of train ",train_essay_tfidf_w2v_vectors.shape)#train_avg_w2v_vectors_title
print("Shape of test  ",test_essay_tfidf_w2v_vectors.shape)
print("Shape of cv  ",cv_essay_tfidf_w2v_vectors.shape)
print("- "*50)
#TFIDF Project_Title

print("Shape of train  ",train_title_tfidf_w2v_vectors.shape)
print("Shape of test  ",test_title_tfidf_w2v_vectors.shape)
print("Shape of  cv ",cv_title_tfidf_w2v_vectors.shape)
print("- "*50)
```

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
-
Shape of train  (17956, 300)
Shape of test   (13200, 300)
Shape of cv   (8844, 300)
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
-
Shape of train   (17956, 300)
Shape of test   (13200, 300)
Shape of  cv  (8844, 300)
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
-
```

Prepare for set 1:

In [150]:

```
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X_set1_train = hstack((X_train_bow_title,X_train_bow,train_prev_proj_standar,train_price_standar,t
rain_qnty_standar,
                      X_train_teacher_prefix,X_train_cat,X_train_subcat,
                      X_train_project_grade_category,X_train_school_state))


print(X_set1_train.shape, y_train.shape)
```

```
(17956, 9119) (17956,)
```

In [151]:

```
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X_set1_cv = hstack((X_cv_bow_title,X_cv_bow,cv_prev_proj_standar,cv_price_standar,cv_qnty_standar,
                      X_cv_teacher_prefix,X_cv_cat,X_cv_subcat,
                      X_cv_project_grade_category,X_cv_school_state))


print(X_set1_cv.shape, y_cv.shape)
```

```
(8844, 9119) (8844,)
```

In [152]:

```
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X_set1_test =
hstack((X_test_bow_title,X_test_bow,test_prev_proj_standar,test_price_standar,test_qnty_standar,
                      X_test_teacher_prefix,X_test_cat,X_test_subcat,
                      X_test_project_grade_category,X_test_school_state))


print(X_set1_test.shape, y_test.shape)
```

```
(13200, 9119) (13200,)
```

Prepare for set 2:

In [153]:

```
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X_set2_train =
hstack((X_train_tf_essay,X_train_tf_title,train_prev_proj_standar,train_price_standar,train_qnty_st
andar,
                      X_train_teacher_prefix,X_train_cat,X_train_subcat,
                      X_train_project_grade_category,X_train_school_state))


print(X_set2_train.shape, y_train.shape)
```

```
(17956, 9119) (17956,)
```

In [154]:

```
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X_set2_cv =
hstack((X_cv_tf_essay,X_cv_tf_title,cv_prev_proj_standar,cv_price_standar,cv_qnty_standar,
                      X_cv_teacher_prefix,X_cv_cat,X_cv_subcat,
                      X_cv_project_grade_category,X_cv_school_state))


print(X_set2_cv.shape, y_cv.shape)
```

```
(8844, 9119) (8844,)
```

In [155]:

```
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X_set2_test = hstack((X_test_tf_essay,X_test_tf_title,test_prev_proj_standar,test_price_standar,te
st_qnty_standar,
                      X_test_teacher_prefix,X_test_cat,X_test_subcat,
                      X_test_project_grade_category,X_test_school_state))


print(X_set2_test.shape, y_test.shape)
```

```
(13200, 9119) (13200,)
```

NOTE: For the wordtovec it restarting so many times takes so much time, restarting again and again, so u said to take less ppoints for average-wordtovec and tf-idf wordtovec. so i take 10k points and split train,test and cv for apply knn. i took 30% test,42% train,28%cv

Prepare for set 3:

In [0]:

```
y_train1=y_train[:4200]
y_test1=y_test[:3000]
y_cv1=y_cv[:2800]
```

In [157]:

```
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X_set3_train = hstack((train_avg_w2v_vectors,train_avg_w2v_vectors_title,train_prev_proj_standar,t
rain_price_standar,train_qnty_standar,
                      X_train_teacher_prefix,X_train_cat,X_train_subcat,
                      X_train_project_grade_category,X_train_school_state))


print(X_set3_train.shape, y_train.shape)
```

```
(17956, 702) (17956,)
```

In [158]:

```
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X_set3_cv =
hstack((cv_avg_w2v_vectors,cv_avg_w2v_vectors_title,cv_prev_proj_standar,cv_price_standar,cv_qnty_s
tandar,
                      X_cv_teacher_prefix,X_cv_cat,X_cv_subcat,
                      X_cv_project_grade_category,X_cv_school_state))


print(X_set3_cv.shape, y_cv.shape)
```

```
(8844, 702) (8844,)
```

In [159]:

```
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X_set3_test =
hstack((test_avg_w2v_vectors,test_avg_w2v_vectors_title,test_prev_proj_standar,test_price_standar,
test_qnty_standar,
                      X_test_teacher_prefix,X_test_cat,X_test_subcat,
                      X_test_project_grade_category,X_test_school_state))


print(X_set3_test.shape, y_test.shape)
```

```
(13200, 702) (13200,)
```

In [160]:

```
# convert to dataframe
#https://stackoverflow.com/questions/20763012/creating-a-pandas-dataframe-from-a-numpy-array-how-d
o-i-specify-the-index-colum
X_set3_test=pd.DataFrame(X_set3_test.toarray())
#print(X_set4_test[0:10])
X_set3_cv=pd.DataFrame(X_set3_cv.toarray())

X_set3_train=pd.DataFrame(X_set3_train.toarray())


# train take 7000 ,test take 3000
X_set3_test=X_set3_test[:3000]
X_set3_train=X_set3_train[:4200]
X_set3_cv=X_set3_cv[:2800]# take 4200

print(X_set3_test.shape, y_test1.shape)
print(X_set3_cv.shape, y_cv1.shape)
print(X_set3_train.shape, y_train1.shape)
```

```
(3000, 702) (3000,)
(2800, 702) (2800,)
(4200, 702) (4200,)
```

Prepare for set 4:

In [161]:

```
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X_set4_train =
X_set4_train =
hstack((train_tfidf_w2v_vectors,train_title_tfidf_w2v_vectors,train_prev_proj_standar,train_price_s
tandar,train_qnty_standar,
                      X_train_teacher_prefix,X_train_cat,X_train_subcat,
                      X_train_project_grade_category,X_train_school_state))


print(X_set4_train.shape, y_train.shape)
```

```
(17956, 702) (17956,)
```

In [162]:

```
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X_set4_cv =
hstack((cv_tfidf_w2v_vectors,cv_title_tfidf_w2v_vectors,cv_prev_proj_standar,cv_price_standar,cv_q
nty_standar,
                      X_cv_teacher_prefix,X_cv_cat,X_cv_subcat,
                      X_cv_project_grade_category,X_cv_school_state))


print(X_set4_cv.shape, y_cv.shape)
```

```
(8844, 702) (8844,)
```

In [163]:

```
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X_set4_test = hstack((test_title_tfidf_w2v_vectors,test_tfidf_w2v_vectors,test_prev_proj_standar,t
est_price_standar,test_qnty_standar,
                      X_test_teacher_prefix,X_test_cat,X_test_subcat,
                      X_test_project_grade_category,X_test_school_state))


print(X_set4_test.shape, y_test.shape)
```

```
(13200, 702) (13200,)
```

In [0]:

```
X_set4_test=pd.DataFrame(X_set4_test.toarray())
#print(X_set4_test[0:10])
X_set4_cv=pd.DataFrame(X_set4_cv.toarray())

X_set4_train=pd.DataFrame(X_set4_train.toarray())
```

In [0]:

```
# train take 7000 ,test take 3000
X_set4_test=X_set4_test[:3000]
X_set4_train=X_set4_train[:4200]
X_set4_cv=X_set4_cv[:2800]# take 4200
```

In [166]:

```
print(X_set4_test.shape, y_test1.shape)
print(X_set4_cv.shape, y_cv1.shape)
print(X_set4_train.shape, y_train1.shape)
```

```
(3000, 702) (3000,)
(2800, 702) (2800,)
(4200, 702) (4200,)
```

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

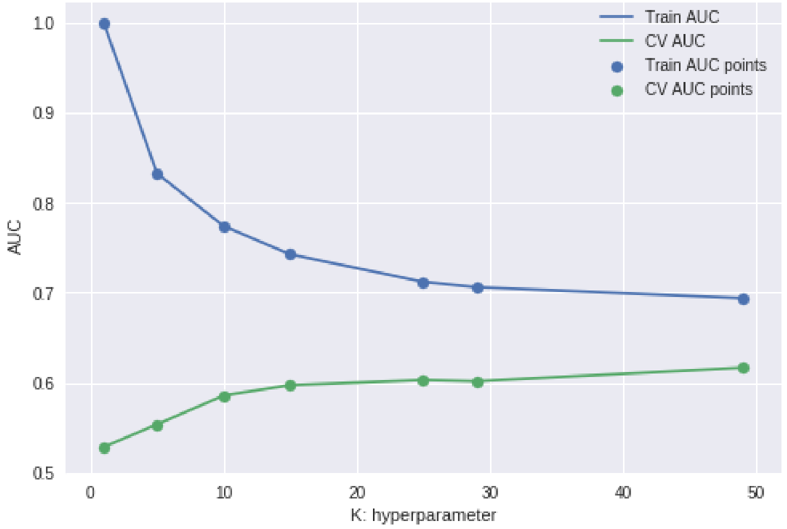
Applying knn section

2.4.1 Applying KNN brute force on BOW, SET 1

In [167]:

```
#http://localhost:8888/notebooks/Assignment_SAMPLE_SOLUTION%20(1).ipynb  (for reference) Which you
provided

from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score

import matplotlib.pyplot as plt

"""
y_true : array, shape = [n_samples] or [n_samples, n_classes]
True binary labels or binary label indicators.

y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the positive class, confidence values, or no
n-thresholded measure of
decisions (as returned by "decision_function" on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with greater label.

"""

train_auc = []
cv_auc = []
K = [1, 5, 10, 15,25,29,49]# min k causes overfitting, max k causes underfitting
for i in  tqdm(K):

    neigh = KNeighborsClassifier(n_neighbors=i,algorithm='brute')# takes the k from the  i th list
value
    neigh.fit(X_set1_train, y_train)# fit the model


    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive class
    # not the predicted outputs
    y_train_pred =  neigh.predict_proba(X_set1_train)[:,1]#Return probability estimates for the
set1x ,for the class label 1 or +ve.
    y_cv_pred =  neigh.predict_proba(X_set1_cv)[:,1]#Return probability estimates for the
setcvx,for the class label 1 or +ve .


    # roc curve
    #Compute Area Under the Receiver Operating Characteristic Curve (ROC AUC) from prediction scor
es.
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

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


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("ERROR PLOTS")
plt.show()
```

```
100%|██████████| 7/7 [07:54<00:00, 67.78s/it]
```



In [168]:

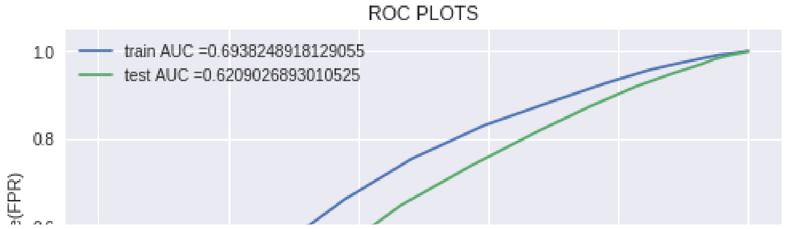
```
score_t_cv = [x for x in cv_auc]
opt_t_cv = K[score_t_cv.index(max(score_t_cv))]
print("Maximum AUC score of cv is:" + ' ' + str(max(score_t_cv)))
print("Corresponding k value of cv is:",opt_t_cv, '\n')
best_k=opt_t_cv
print(best_k)
```

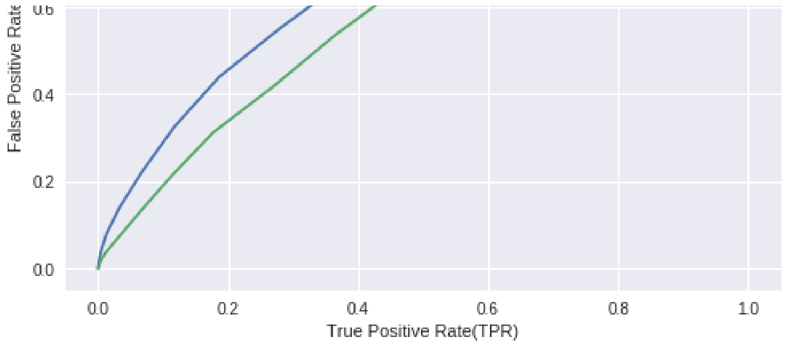
```
Maximum AUC score of cv is: 0.6164812635330085
Corresponding k value of cv is: 49

49
```

Fitting Model to Hyper-Parameter Curve (Using bruteforce KNN)

In [169]:

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


neigh = KNeighborsClassifier(n_neighbors=best_k,algorithm='brute')
neigh.fit(X_set1_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, neigh.predict_proba(X_set1_train)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, neigh.predict_proba(X_set1_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("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("ROC PLOTS")
plt.show()

print("="*100)
```





```
====================================================================================================
```

OBSERVATIONS: As we seen form the roc plot ,as we increase the k value this roc curve improve little bit , not more because this is the imbalanced dataset,so lets see in further plots.

Confusion matrix :

In [170]:

```
print("="*100)
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, neigh.predict(X_set1_train )))
```

```
====================================================================================================

Train confusion matrix
[[    4  2768]
 [    2 15182]]
```

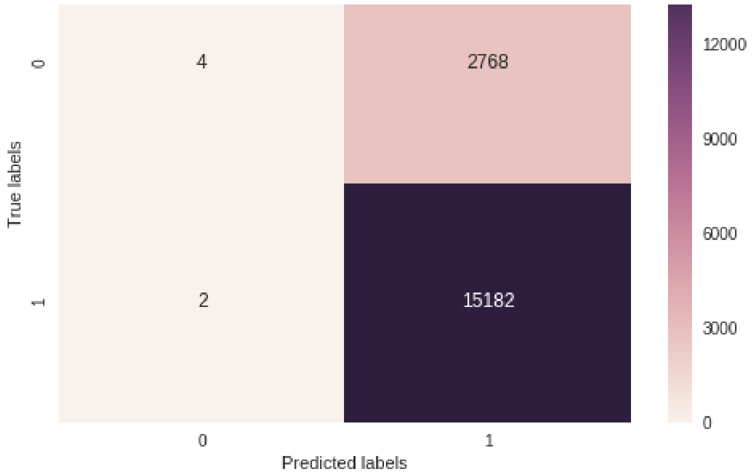
In [171]:

```
from sklearn.metrics import classification_report

print(classification_report(y_train,neigh.predict(X_set1_train) ))
```

```
              precision    recall  f1-score   support

           0       0.67      0.00      0.00      2772
           1       0.85      1.00      0.92     15184

   micro avg       0.85      0.85      0.85     17956
   macro avg       0.76      0.50      0.46     17956
weighted avg       0.82      0.85      0.78     17956
```

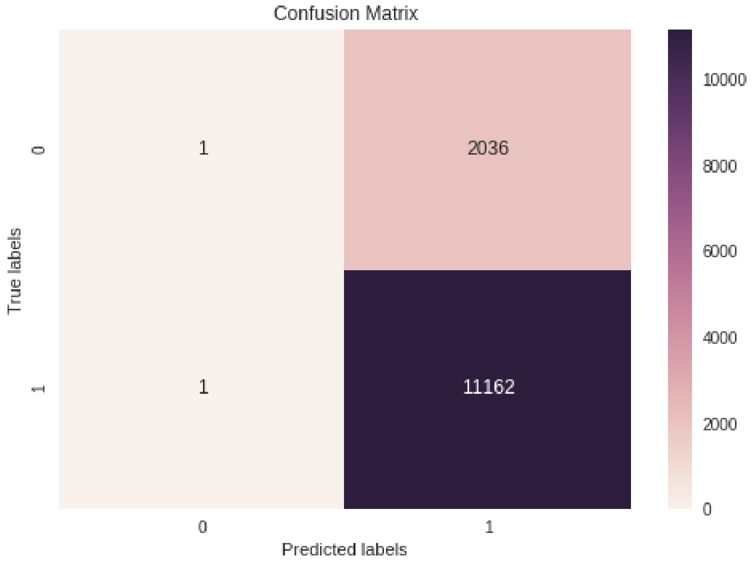
In [172]:

```
#https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix
import seaborn as sns
import matplotlib.pyplot as plt

ax= plt.subplot()
sns.heatmap(confusion_matrix(y_train, neigh.predict(X_set1_train )), annot=True, ax = ax,fmt='g');
#annot=True to annotate cells

# labels, title and ticks
ax.set_xlabel('Predicted labels');
ax.set_ylabel('True labels');
ax.set_title('Confusion Matrix');
#ax.xaxis.set_ticklabels(['business', 'health']); ax.yaxis.set_ticklabels(['health', 'business']);
```



In [173]:

```
#https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix
import seaborn as sns
import matplotlib.pyplot as plt

ax= plt.subplot()
sns.heatmap(confusion_matrix(y_test, neigh.predict(X_set1_test )), annot=True, ax = ax,fmt='g'); #a
nnot=True to annotate cells

# labels, title and ticks
ax.set_xlabel('Predicted labels');
ax.set_ylabel('True labels');
ax.set_title('Confusion Matrix');
#ax.xaxis.set_ticklabels(['business', 'health']); ax.yaxis.set_ticklabels(['health', 'business']);
```



OBSERVATOINS: As we see from this confusion matrix ,In our prediction true positives is of greater weitage,beacuse of high k value all the negatives are dominating so that true negaties arezero,all are predictee wrong, but for the better prediction we want tp and tn both to be more,but if we choose k to be low then our roc cure,auc value less than ,50 or 50 worst value, if we increasee k then it will dominating the posities values,so lets see in further plots ,what inference we make from this plots, and what is auc and confusion matrix, but from now i am clear that , This imbalancing is not good for our model, and also if our best k to be big then, cause of underfitting, so simply means we have to take more data for overcome underfitting,but more data can;t be handled by my laptop.

Also their a reason why this auc is not so good,knn is a basic algo,means not so good as compared to some advanced ml algos, so may be that is the reason for our not so good prediction like roc and confusion matri is not good.

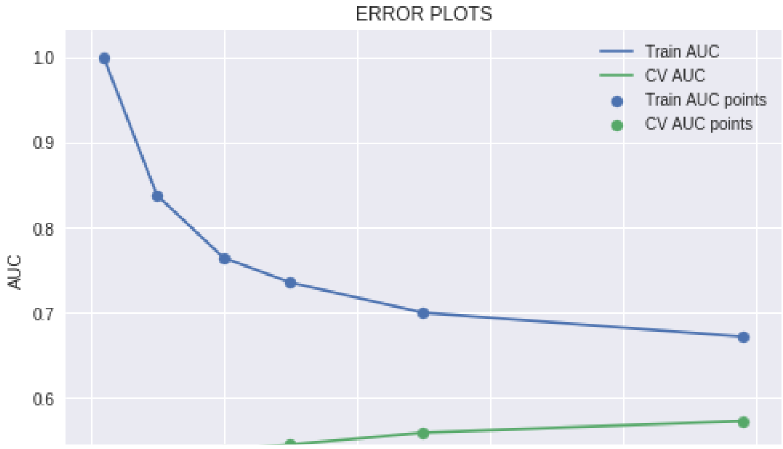
2.4.2 Applying KNN brute force on TFIDF, SET 2

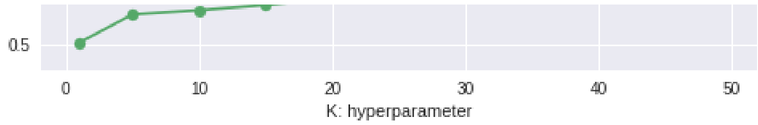
In [180]:

```
#http://localhost:8888/notebooks/Assignment_SAMPLE_SOLUTION%20(1).ipynb  (for reference) Which you
```

```
#http://localhost:8888/notebooks/Assignment_SAMPLE_SOLUTION%20(1).ipynb  (for reference) which you
provided

from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score

import matplotlib.pyplot as plt

"""
y_true : array, shape = [n_samples] or [n_samples, n_classes]
True binary labels or binary label indicators.

y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the positive class, confidence values, or no
n-thresholded measure of
decisions (as returned by “decision_function” on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with greater label.

"""

train_auc = []
cv_auc = []
K = [1, 5, 10, 15,25,49]# min k causes overfitting, max k causes underfitting
for i in  tqdm(K):

    neigh = KNeighborsClassifier(n_neighbors=i,algorithm='brute')# takes the k from the  i th list
value
    neigh.fit(X_set2_train, y_train)# fit the model


    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive class
    # not the predicted outputs
    y_train_pred =  neigh.predict_proba(X_set2_train)[:,1]#Return probability estimates for the
set1x ,for the class label 1 or +ve.
    y_cv_pred =  neigh.predict_proba(X_set2_cv)[:,1]#Return probability estimates for the
setcvx,for the class label 1 or +ve .


    # roc curve
    #Compute Area Under the Receiver Operating Characteristic Curve (ROC AUC) from prediction scor
es.
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

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


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("ERROR PLOTS")
plt.show()
```

```
100%|██████████| 6/6 [06:44<00:00, 67.52s/it]
```





In [181]:

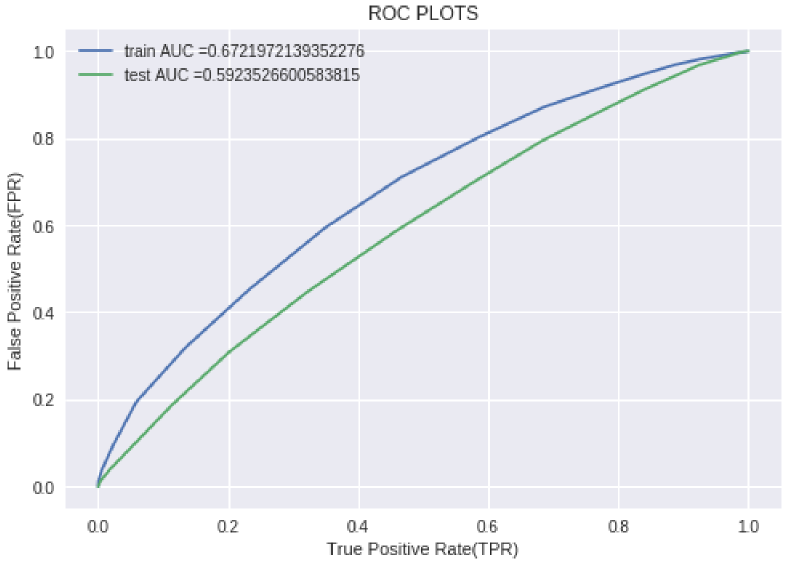
```
score_t_cv_3 = [x for x in cv_auc]
opt_t_cv_3 = K[score_t_cv.index(max(score_t_cv))-1]
print("Maximum AUC score of cv is:" + ' ' + str(max(score_t_cv_3)))
print("Corresponding k value of cv is:",opt_t_cv_3, '\n')
```

```
Maximum AUC score of cv is: 0.5731908684977277
Corresponding k value of cv is: 49
```

Fitting Model to Hyper-Parameter Curve (using brute force KNN):

In [182]:

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


neigh = KNeighborsClassifier(n_neighbors=49,algorithm='brute')
neigh.fit(X_set2_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, neigh.predict_proba(X_set2_train)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, neigh.predict_proba(X_set2_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("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("ROC PLOTS")
plt.show()

print("="*100)
```



```
====================================================================================================
```

OBSERVATONS: We can see in tf-idf,roc curve improve when we increaes the k value , as i already said this is underfitting ,because of imbalancing, so our imference is not so good in real word scenarios.And confusing matrix also has domating class.

COnfusion matrix

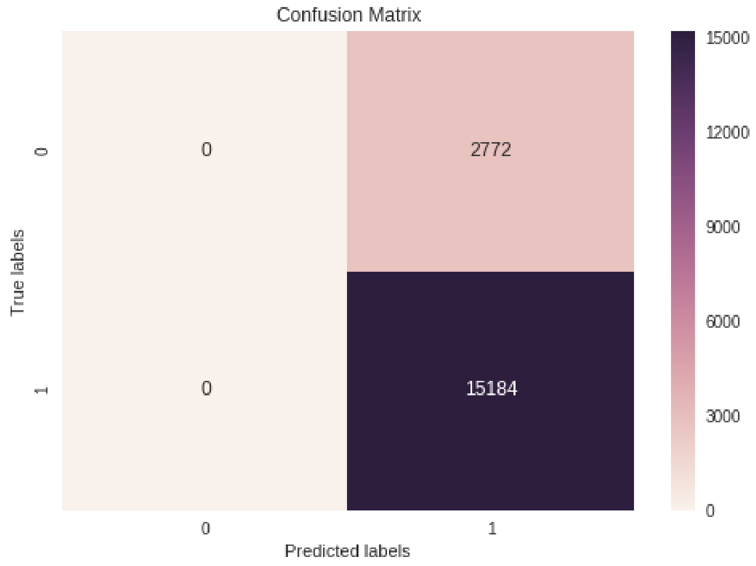
In [183]:

```
from sklearn.metrics import classification_report

print(classification_report(y_train,neigh.predict(X_set2_train) ))
```

```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00      2772
           1       0.85      1.00      0.92     15184

   micro avg       0.85      0.85      0.85     17956
   macro avg       0.42      0.50      0.46     17956
weighted avg       0.72      0.85      0.77     17956
```

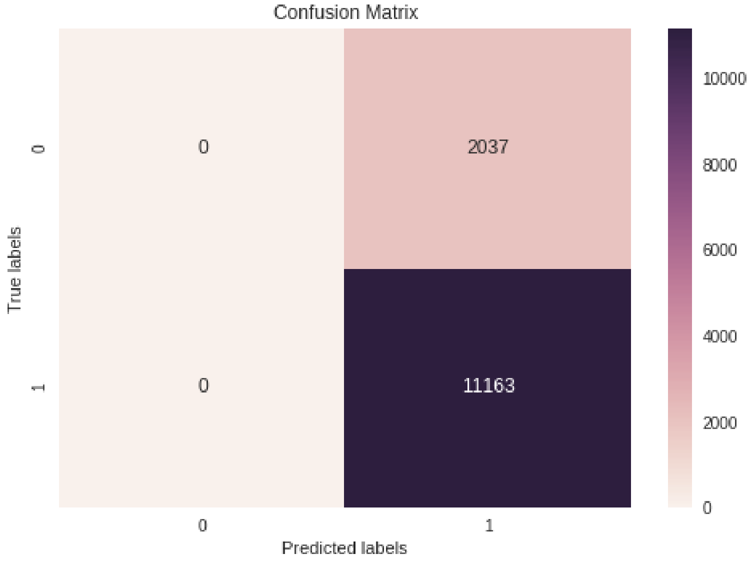
In [184]:

```
#https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix
import seaborn as sns
import matplotlib.pyplot as plt

ax= plt.subplot()
sns.heatmap(confusion_matrix(y_train, neigh.predict(X_set2_train )), annot=True, ax = ax,fmt='g');
#annot=True to annotate cells

# labels, title and ticks
ax.set_xlabel('Predicted labels');
ax.set_ylabel('True labels');
ax.set_title('Confusion Matrix');
#ax.xaxis.set_ticklabels(['business', 'health']); ax.yaxis.set_ticklabels(['health', 'business']);
```



In [185]:

```
#https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix
import seaborn as sns
import matplotlib.pyplot as plt

ax= plt.subplot()
sns.heatmap(confusion_matrix(y_test, neigh.predict(X_set2_test )), annot=True, ax = ax,fmt='g'); #a
nnot=True to annotate cells

# labels, title and ticks
ax.set_xlabel('Predicted labels');
ax.set_ylabel('True labels');
ax.set_title('Confusion Matrix');
#ax.xaxis.set_ticklabels(['business', 'health']); ax.yaxis.set_ticklabels(['health', 'business']);
```



Observation: Due to highly imbalan in the data set or due to high k vlaue this is totaly dominating the negative class

Apply the wordtovec for set3

2.4.3 Applying KNN brute force on AVG W2V, SET 3

In [186]:

```
print(X_set3_train.shape,y_train1.shape)
```

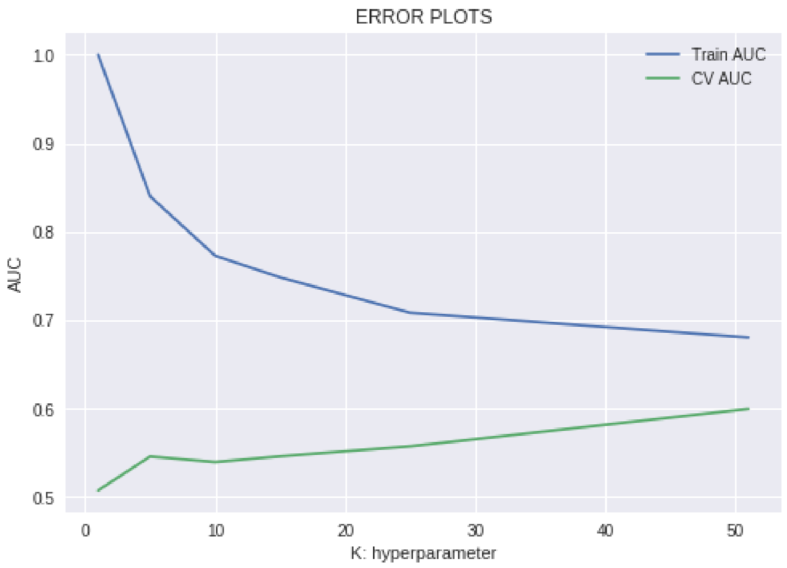
```
(4200, 702) (4200,)
```

In [187]:

```
#http://localhost:8888/notebooks/Assignment_SAMPLE_SOLUTION%20(1).ipynb  (for reference) Which you
provided

from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt

"""
y_true : array, shape = [n_samples] or [n_samples, n_classes]
True binary labels or binary label indicators.

y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the positive class, confidence values, or no
n-thresholded measure of
decisions (as returned by "decision_function" on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with greater label.

"""

train_auc = []
cv_auc = []
K = [1, 5, 10, 15,25,51]# min k causes overfitting, max k causes underfitting
for i in tqdm(K):

    neigh = KNeighborsClassifier(n_neighbors=i,algorithm='brute')# takes the k from the  i th list
value
    neigh.fit(X_set3_train, y_train1)# for the model


    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive class
    # not the predicted outputs
    y_train_pred =  neigh.predict_proba(X_set3_train)[:,1]#Return probability estimates for the
```

```
set3x ,for the class label 1 or +ve.
    y_cv_pred =  neigh.predict_proba(X_set3_cv)[:,1]#Return probability estimates for the
set3cvx,for the class label 1 or +ve .


    # roc curve
    #Compute Area Under the Receiver Operating Characteristic Curve (ROC AUC) from prediction scor
es.
    train_auc.append(roc_auc_score(y_train1,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv1, y_cv_pred))

plt.plot(K, train_auc, label='Train AUC')
plt.plot(K, cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

```
100%|██████████| 6/6 [00:10<00:00,  1.79s/it]
```



In [188]:

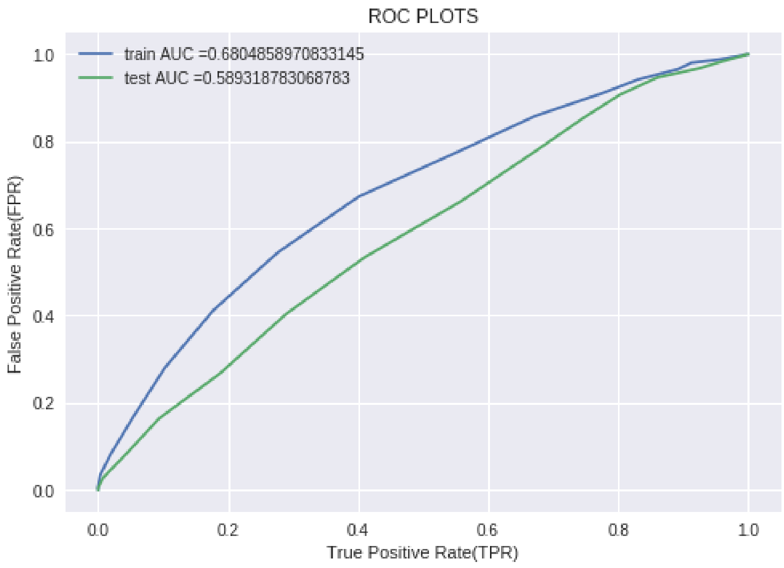
```
scor = [x for x in cv_auc]
opt_t_cv_3 = K[scor.index(max(scor))]
print("Maximum AUC score of cv is:" + ' ' + str(max(scor)))
print("Corresponding k value of cv is:",opt_t_cv_3, '\n')
```

```
Maximum AUC score of cv is: 0.5996691780615399
Corresponding k value of cv is: 51
```

Fitting Model to Hyper-Parameter Curve (using Bruteforce KNN):

In [189]:

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


neigh = KNeighborsClassifier(n_neighbors=opt_t_cv_3,algorithm='brute')
neigh.fit(X_set3_train ,y_train1)
# 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_train1, neigh.predict_proba(X_set3_train)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test1, neigh.predict_proba(X_set3_test)[:,1])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("ROC PLOTS")
plt.show()

print("="*100)
```



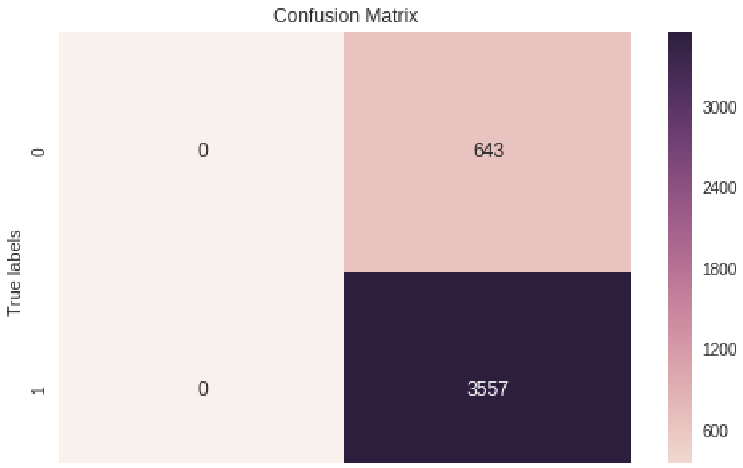
```
====================================================================================================
```

Observations: SO in this word2vece we take only 10k points, so that why ,not so good interpretation, so that why roc curve is worst like random.

COnfusion matrix

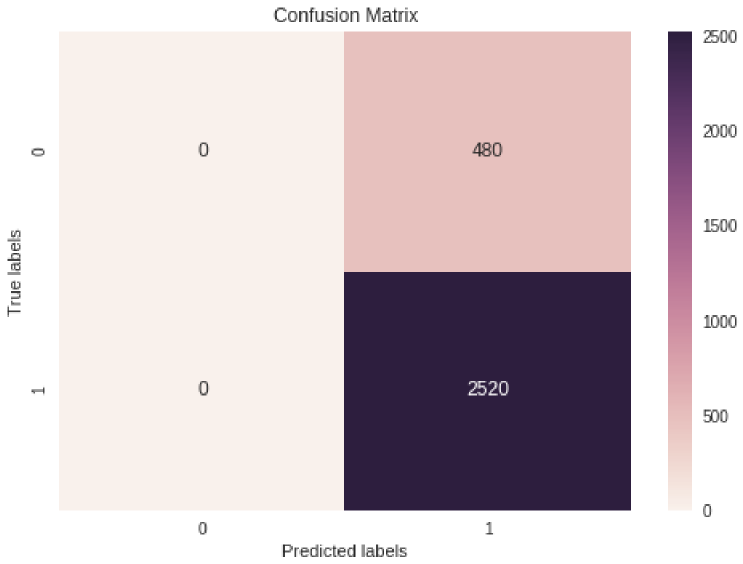
In [190]:

```
#https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix
import seaborn as sns
import matplotlib.pyplot as plt

ax= plt.subplot()
sns.heatmap(confusion_matrix(y_train1, neigh.predict(X_set3_train )), annot=True, ax = ax,fmt='g');
#annot=True to annotate cells

# labels, title and ticks
ax.set_xlabel('Predicted labels');
ax.set_ylabel('True labels');
ax.set_title('Confusion Matrix');
#ax.xaxis.set_ticklabels(['business', 'health']); ax.yaxis.set_ticklabels(['health', 'business']);
```



In [191]:

```
#https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix
import seaborn as sns
import matplotlib.pyplot as plt

ax= plt.subplot()
sns.heatmap(confusion_matrix(y_test1, neigh.predict(X_set3_test )), annot=True, ax = ax,fmt='g'); # annot=True to annotate cells

# labels, title and ticks
ax.set_xlabel('Predicted labels');
ax.set_ylabel('True labels');
ax.set_title('Confusion Matrix');
#ax.xaxis.set_ticklabels(['business', 'health']); ax.yaxis.set_ticklabels(['health', 'business']);
```



Observations: We can't make some correct inferences from this confusion matrix also its so bad confusion matrix.'Totaly worst confusion matrix, just because of imbalanced data

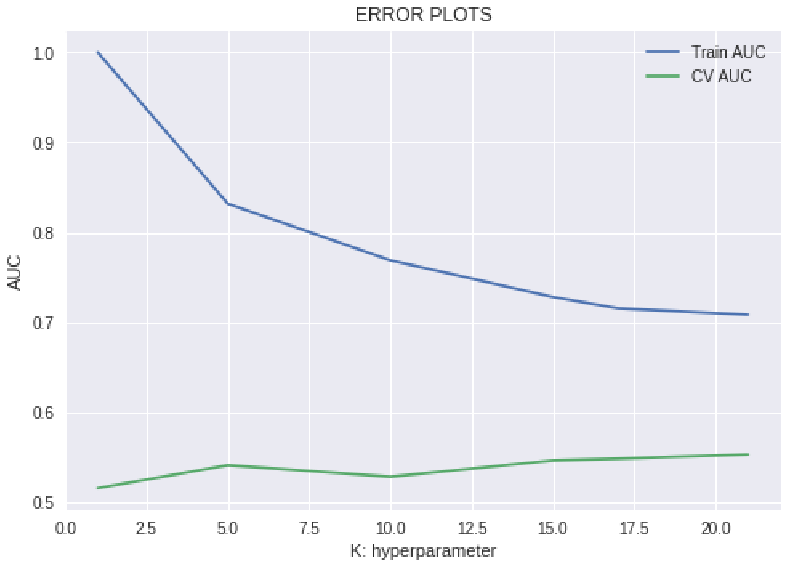
2.4.4 Applying KNN brute force on TFIDF W2V, SET 4

In [192]:

```
#http://localhost:8888/notebooks/Assignment_SAMPLE_SOLUTION%20(1).ipynb  (for reference) Which you provided

from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt

"""
y_true : array, shape = [n_samples] or [n_samples, n_classes]
True binary labels or binary label indicators.

y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the positive class, confidence values, or non-thresholded measure of
decisions (as returned by “decision_function” on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with greater label.

"""

train_auc = []
cv_auc = []
K = [1, 5, 10, 15, 17,21]# min k causes overfitting, max k causes underfitting
```

```
for i in tqdm(K):

    neigh = KNeighborsClassifier(n_neighbors=i,algorithm='brute')# takes the k from the  i th list
value
    neigh.fit(X_set4_train, y_train1)# for the model


    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive class
    # not the predicted outputs
    y_train_pred =  neigh.predict_proba(X_set4_train)[:,1]#Return probability estimates for the
set3x ,for the class label 1 or +ve.
    y_cv_pred =  neigh.predict_proba(X_set4_cv)[:,1]#Return probability estimates for the
set3cvx,for the class label 1 or +ve .


    # roc curve
    #Compute Area Under the Receiver Operating Characteristic Curve (ROC AUC) from prediction scor
es.
    train_auc.append(roc_auc_score(y_train1,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv1, y_cv_pred))

plt.plot(K, train_auc, label='Train AUC')
plt.plot(K, cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

```
100%|██████████| 6/6 [00:11<00:00,  1.85s/it]
```



In [193]:

```
sc = [x for x in cv_auc]
opt_t_cv_4 = K[sc.index(max(sc ))]
print("Maximum AUC score of cv is:" + ' ' + str(max(sc )))
print("Corresponding k value of cv is:",opt_t_cv_4, '\n')
```

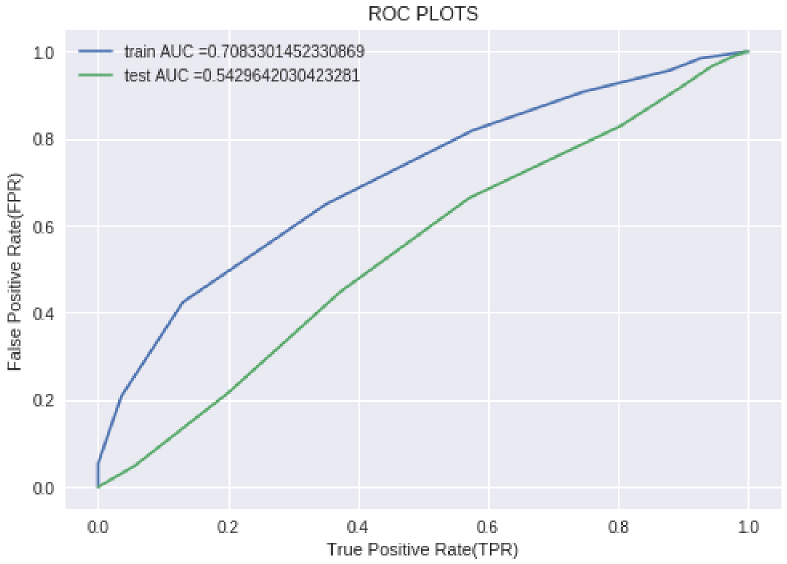
```
Maximum AUC score of cv is: 0.5527143228656284
Corresponding k value of cv is: 21
```

Fitting Model to Hyper-Parameter Curve: (using brute force KNN)

In [194]:

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

```
neigh = KNeighborsClassifier(n_neighbors=21,algorithm='brute')
neigh.fit(X_set4_train ,y_train1)
# 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_train1, neigh.predict_proba(X_set4_train)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test1, neigh.predict_proba(X_set4_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("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("ROC PLOTS")
plt.show()

print("="*100)
```



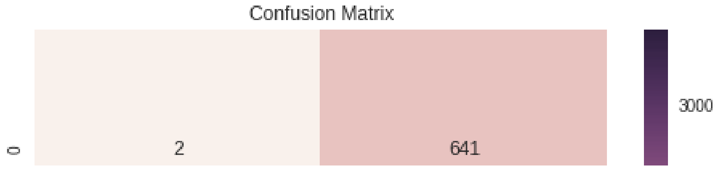
```
====================================================================================================
```

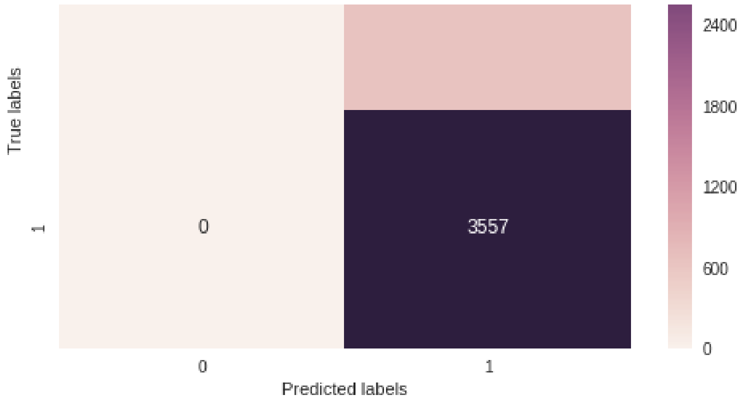
Observations: SO in this word2vece we take only 10k points, so that why ,not so good interpretation, so that why roc curve is worst like random.

COnfusion matrix

In [195]:

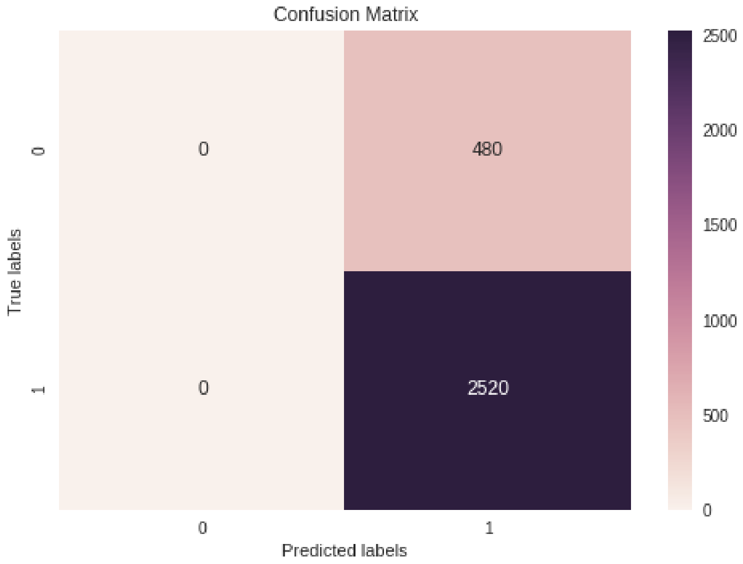
```
#https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix
import seaborn as sns
import matplotlib.pyplot as plt

ax= plt.subplot()
sns.heatmap(confusion_matrix(y_train1, neigh.predict(X_set4_train )), annot=True, ax = ax,fmt='g');
#annot=True to annotate cells

# labels, title and ticks
ax.set_xlabel('Predicted labels');
ax.set_ylabel('True labels');
ax.set_title('Confusion Matrix');
#ax.xaxis.set_ticklabels(['business', 'health']); ax.yaxis.set_ticklabels(['health', 'business']);
```





In [196]:

```
#https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix
import seaborn as sns
import matplotlib.pyplot as plt

ax= plt.subplot()
sns.heatmap(confusion_matrix(y_test1, neigh.predict(X_set4_test )), annot=True, ax = ax,fmt='g'); #
annot=True to annotate cells

# labels, title and ticks
ax.set_xlabel('Predicted labels');
ax.set_ylabel('True labels');
ax.set_title('Confusion Matrix');
#ax.xaxis.set_ticklabels(['business', 'health']); ax.yaxis.set_ticklabels(['health', 'business']);
```



Observations: We can't make some correct inferences from this confusion matrix also its so bad confusion matrix.'

2.5 Feature selection with `SelectKBest`: (Using Bruteforce KNN)

In [197]:

```
# apply this on tf-idf
print(X_set2_train.shape, y_train.shape)
print(X_set2_test.shape, y_test.shape)
print(X_set2_cv.shape, y_cv.shape)
```

```
(17956, 9119) (17956,)
(13200, 9119) (13200,)
(8844, 9119) (8844,)
```

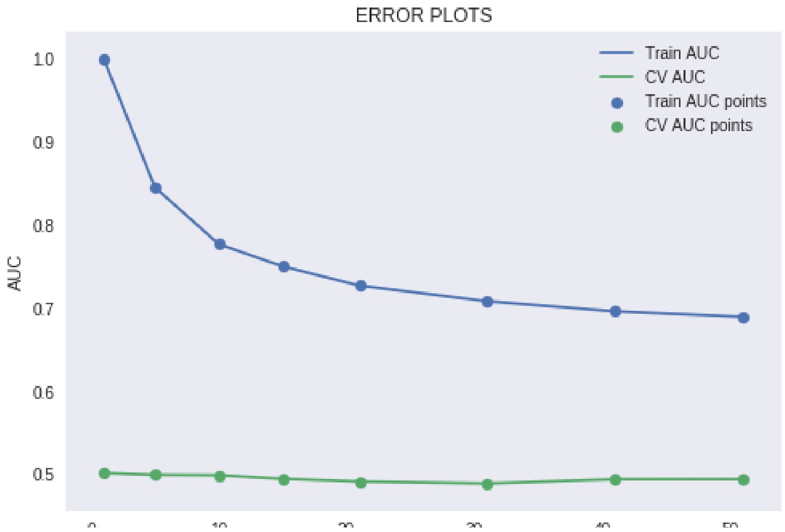
In [0]:

```
#https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_classif,chi2
#ValueError: Input X must be non-negative.

# not use chi because of error
##https://stackoverflow.com/questions/25792012/feature-selection-using-scikit-learn
X_train2_new = SelectKBest(f_classif, k=2000).fit_transform(X_set2_train, y_train)
X_test2_new = SelectKBest(f_classif, k=2000).fit_transform(X_set2_test, y_test)
X_cv2_new = SelectKBest(f_classif, k=2000).fit_transform(X_set2_cv, y_cv)
```

In [206]:

```
#train_essay_tfidf_w2v_vectors
#test_essay_tfidf_w2v_vectors
train_auc = []
cv_auc = []
K = [1, 5, 10, 15, 21, 31, 41, 51]
for i in tqdm(K):
    neigh = KNeighborsClassifier(n_neighbors=i,algorithm='brute')
    neigh.fit(X_train2_new, y_train)




    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive class
    # not the predicted outputs
    y_train_pred = neigh.predict_proba(X_train2_new)[:,1]#Return probability estimates for the
set3x ,for the class label 1 or +ve.
    y_cv_pred =  neigh.predict_proba(X_cv2_new)[:,1]#Return probability estimates for the
set3cvx,for the class label 1 or +ve .


    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive class
    # not the predicted outputs
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

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

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("ERROR PLOTS")
plt.grid()
plt.show()
```

```
100%|██████████| 8/8 [05:19<00:00, 40.30s/it]
```



In [207]:

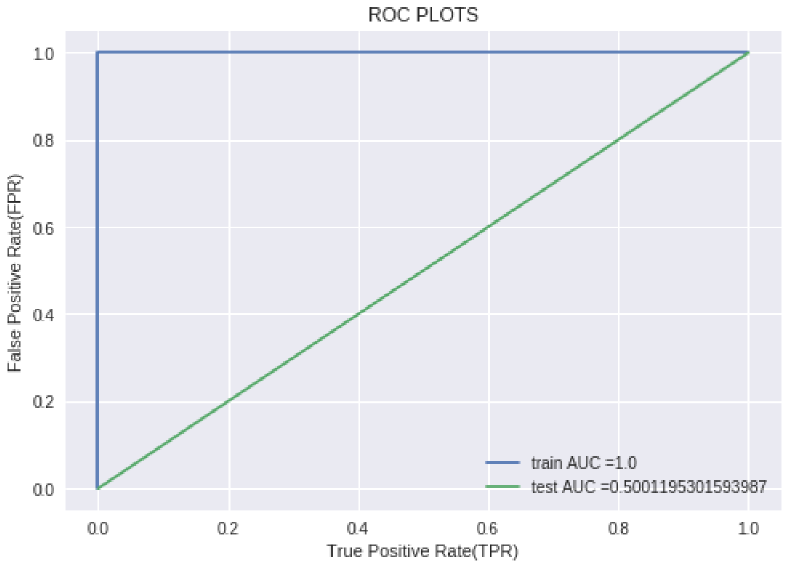
```
sc1 = [x for x in cv_auc]
opt_t_cv_4 = K[sc1.index(max(sc1 ))]
print("Maximum AUC score of cv is:" + ' ' + str(max(sc )))
print("Corresponding k value of cv is:",opt_t_cv_4, '\n')
```

```
Maximum AUC score of cv is: 0.5527143228656284
Corresponding k value of cv is: 1
```

Fitting Model to Hyper-Parameter Curve: (Using bruteforce KNN)

In [208]:

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


neigh = KNeighborsClassifier(n_neighbors=opt_t_cv_4,algorithm='brute')
neigh.fit(X_train2_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

train_fpr, train_tpr, thresholds = roc_curve(y_train, neigh.predict_proba(X_train2_new)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, neigh.predict_proba(X_test2_new)[:,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("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("ROC PLOTS")
plt.show()

print("="*100)
```



```
====================================================================================================
```

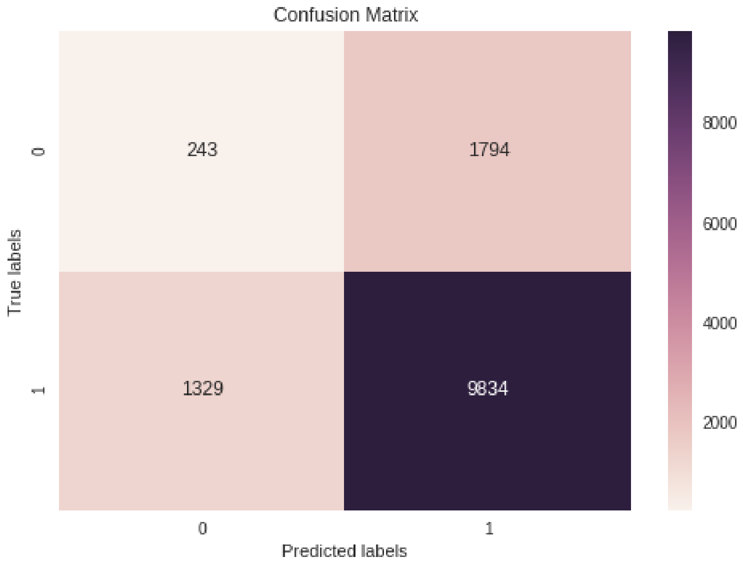
Observations: Finding the top 2000 featues not helpful,their are lots of reasons of it

1. In cv dta bcz of less data or highly imbalance their is underfitting so k=1 is best, meaing totlay random rooc curve

Confusion matrix

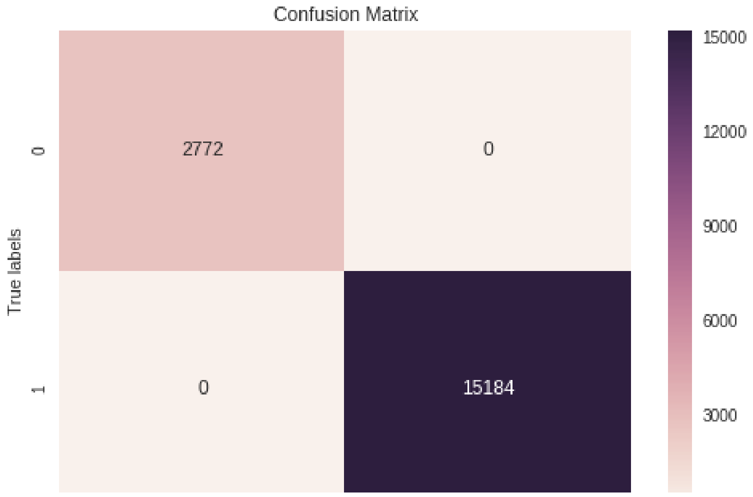
In [209]:

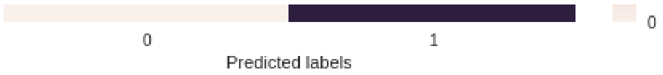
```
#https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix
import seaborn as sns
import matplotlib.pyplot as plt

ax= plt.subplot()
sns.heatmap(confusion_matrix(y_test, neigh.predict(X_test2_new )), annot=True, ax = ax,fmt='g'); #annot=True to annotate cells

# labels, title and ticks
ax.set_xlabel('Predicted labels');
ax.set_ylabel('True labels');
ax.set_title('Confusion Matrix');
#ax.xaxis.set_ticklabels(['business', 'health']); ax.yaxis.set_ticklabels(['health', 'business']);
```



In [210]:

```
#https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix
import seaborn as sns
import matplotlib.pyplot as plt

ax= plt.subplot()
sns.heatmap(confusion_matrix(y_train, neigh.predict(X_train2_new )), annot=True, ax = ax,fmt='g'); #annot=True to annotate cells

# labels, title and ticks
ax.set_xlabel('Predicted labels');
ax.set_ylabel('True labels');
ax.set_title('Confusion Matrix');
#ax.xaxis.set_ticklabels(['business', 'health']); ax.yaxis.set_ticklabels(['health', 'business']);
```





Observatoins: In train data as our best k iis one thats why fully pefcet train_data, but totaly overfitting this is.

3. Conclusions

In [212]:

```
# Please compare all your models using Prettytable library
# Please compare all your models using Prettytable library
#how to use pretty table http://zetcode.com/python/prettytable/
from prettytable import PrettyTable

tb = PrettyTable()
tb.field_names= ("Vectorizer", "Model", "HyperParameter", "AUC")
tb.add_row(["BOW", "Auto", 49, 62])
tb.add_row(["Tf-Idf", "Auto", 49, 59])
tb.add_row(["AVG-W2v", "Auto", 25, 58])
tb.add_row(["Tf-Idf W2v", "Auto", 21, 54])
tb.add_row(["Tf-Idf KBest", "Auto", 1, 50])
print(tb.get_string(titles = "KNN - Observations"))
#print(tb)
```

```
+--------------+-------+----------------+-----+
|  Vectorizer  | Model | HyperParameter | AUC |
+--------------+-------+----------------+-----+
|     BOW      |  Auto |       49       |  62 |
|    Tf-Idf    |  Auto |       49       |  59 |
|   AVG-W2v    |  Auto |       25       |  58 |
|  Tf-Idf W2v  |  Auto |       21       |  54 |
| Tf-Idf KBest |  Auto |       1        |  50 |
+--------------+-------+----------------+-----+
```

Performance of Model: So as we see from all our models, their are so much true positives, and true negaties is 0. simply say that because of the high k value or We can say it underfits the data.This is also a unbalanced data so thats why our result is not so much accurate.It dominates the zeros class labels because of the high k value.As we said it underfits the data ,so we have to improve or make some features which better determine the class label.And also our data is very less only 40k,may be thats why predictions is not so good.