DonorsChoose

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

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

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

In [0]:
```
# root_path="gdrive/My Drive/"
```

In [0]:
```
pip install chart_studio
```

```
Collecting chart_studio
  Downloading https://files.pythonhosted.org/packages/b9/3f/d2f3f506ba1
aaf109f549f8b01d1483cd3e324c5ebe6b206acee66efdf46/chart_studio-1.0.0-py
3-none-any.whl (76kB)
     |████████████████████████████████| 81kB 3.2MB/s
Requirement already satisfied: requests in /usr/local/lib/python3.6/dis
t-packages (from chart_studio) (2.21.0)
Requirement already satisfied: plotly in /usr/local/lib/python3.6/dist-
packages (from chart_studio) (4.1.1)
Requirement already satisfied: retrying>=1.3.3 in /usr/local/lib/python
3.6/dist-packages (from chart_studio) (1.3.3)
Requirement already satisfied: six in /usr/local/lib/python3.6/dist-pac
kages (from chart_studio) (1.12.0)
```

```
Requirement already satisfied: urllib3<1.25,>=1.21.1 in /usr/local/lib/
python3.6/dist-packages (from requests->chart_studio) (1.24.3)
Requirement already satisfied: idna<2.9,>=2.5 in /usr/local/lib/python
3.6/dist-packages (from requests->chart_studio) (2.8)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/pyt
hon3.6/dist-packages (from requests->chart_studio) (2019.6.16)
Requirement already satisfied: chardet<3.1.0,>=3.0.2 in /usr/local/lib/
python3.6/dist-packages (from requests->chart_studio) (3.0.4)
Installing collected packages: chart-studio
Successfully installed chart-studio-1.0.0
```

In [0]:
```
pip install plotly
```

```
Requirement already satisfied: plotly in /usr/local/lib/python3.6/dist-
packages (4.1.1)
Requirement already satisfied: six in /usr/local/lib/python3.6/dist-pac
kages (from plotly) (1.12.0)
Requirement already satisfied: retrying>=1.3.3 in /usr/local/lib/python
3.6/dist-packages (from plotly) (1.3.3)
```

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

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

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

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

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

from tqdm import tqdm
import os
from collections import Counter
import chart_studio.plotly as py
from collections import Counter
```

In [0]:
```
#from plotly import plotly
#import plotly.offline as offline
#import plotly.graph_objs as go
#offline.init_notebook_mode()
```

In [0]:
```
donors_data = pd.read_csv('gdrive/My Drive/applied ai/assignment3/train_data.csv')
resource_data = pd.read_csv('gdrive/My Drive/applied ai/assignment3/resources.csv')
```

In [0]:
```
data=pd.merge(donors_data,resource_data, on='id' ,how='left')
```

In [0]:
```
#sort dataframe based on time pandas python: https://stackoverflow.com/a/49702492/4084039
data.drop(['project_submitted_datetime','id','teacher_id','description','Unnamed: 0'], axis=1, inplace=True)
```

In [0]:
```
# !wget http://nlp.stanford.edu/data/glove.42B.300d.zip
```

In [0]:
```
# import zipfile
# zip_ref = zipfile.ZipFile("glove.42B.300d.zip", 'r')
# zip_ref.extractall("glove")
# zip_ref.close()
```

In [0]:
```
data=data.head(50000)
data.shape
```

Out[0]:
```
(50000, 15)
```

Data cleaning and preprocessing

In [0]:
```
#check null values if any
data.isnull().any()
```

Out[0]:
```
teacher_prefix                                 True
school_state                                  False
project_grade_category                        False
project_subject_categories                    False
project_subject_subcategories                 False
project_title                                 False
project_essay_1                               False
project_essay_2                               False
project_essay_3                                True
project_essay_4                                True
project_resource_summary                      False
teacher_number_of_previously_posted_projects  False
project_is_approved                           False
quantity                                      False
price                                         False
dtype: bool
```

In [0]:
```
null_columns=data.columns[data.isnull().any()]
data[null_columns].isnull().sum()
```

Out[0]:
```
teacher_prefix        2
project_essay_3    48344
```

```
project_essay_4    48344
dtype: int64
```

In [0]:
```
data.dropna(subset=['teacher_prefix'],inplace=True)
data.shape
```

Out[0]:
```
(49998, 15)
```

preprocessing project_subject_category

In [0]:
```
print(data['project_subject_categories'].head(10))
```

```
0                  Literacy & Language
1                  Literacy & Language
2                  Literacy & Language
3                  Literacy & Language
4                  Literacy & Language
5                  Literacy & Language
6                  Literacy & Language
7    History & Civics, Health & Sports
8                       Health & Sports
9                       Health & Sports
Name: project_subject_categories, dtype: object
```

In [0]:
```
catogories = list(data['project_subject_categories'].values)
# remove special characters from list of strings python: https://stacko
verflow.com/a/47301924/4084039

# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-
word-from-a-string
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-
a-string-in-python
cat_list = []
for i in catogories:
    temp = ""
    # consider we have text like this "Math & Science, Warmth, Care & H
```

```
unger"
    for j in i.split(','): # it will split it in three parts ["Math & S
cience", "Warmth", "Care & Hunger"]
        if 'The' in j.split(): # this will split each of the catogory b
ased on space "Math & Science"=> "Math","&", "Science"
            j=j.replace('The','') # if we have the words "The" we are g
oing 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 int
o
    cat_list.append(temp.strip())

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

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

In [0]:
```
print(data['clean_categories'].head(10))
```

```
0               Literacy_Language
1               Literacy_Language
2               Literacy_Language
3               Literacy_Language
4               Literacy_Language
5               Literacy_Language
6               Literacy_Language
7    History_Civics Health_Sports
8                   Health_Sports
9                   Health_Sports
Name: clean_categories, dtype: object
```

preprocessing of project_subject_subcategories

In [0]:
```
print(data['project_subject_subcategories'].head(10))
```

```
0                       ESL, Literacy
1                       ESL, Literacy
2                       ESL, Literacy
3                       ESL, Literacy
4                       ESL, Literacy
5                       ESL, Literacy
6                       ESL, Literacy
7    Civics & Government, Team Sports
8      Health & Wellness, Team Sports
9      Health & Wellness, Team Sports
Name: project_subject_subcategories, dtype: object
```

In [0]:
```
sub_catogories = list(data['project_subject_subcategories'].values)
# remove special characters from list of strings python: https://stacko
verflow.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 & H
unger"
    for j in i.split(','): # it will split it in three parts ["Math & S
cience", "Warmth", "Care & Hunger"]
        if 'The' in j.split(): # this will split each of the catogory b
ased on space "Math & Science"=> "Math","&", "Science"
            j=j.replace('The','') # if we have the words "The" we are g
oing 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())

data['clean_subcategories'] = sub_cat_list
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 data['clean_subcategories'].values:
    my_counter.update(word.split())

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

In [0]:
```
print(data['clean_subcategories'].head(10))
```

```
0                    ESL Literacy
1                    ESL Literacy
2                    ESL Literacy
3                    ESL Literacy
4                    ESL Literacy
5                    ESL Literacy
6                    ESL Literacy
7    Civics_Government TeamSports
8      Health_Wellness TeamSports
9      Health_Wellness TeamSports
Name: clean_subcategories, dtype: object
```

Preprocessing project_grade_category

In [0]:
```
print(data['project_grade_category'].head(10))
```

```
0    Grades PreK-2
1    Grades PreK-2
2    Grades PreK-2
3    Grades PreK-2
4    Grades PreK-2
5    Grades PreK-2
6    Grades PreK-2
7       Grades 6-8
8       Grades 6-8
9       Grades 6-8
Name: project_grade_category, dtype: object
```

In [0]:
```
data["project_grade_category"] = data["project_grade_category"].str.rep
lace(" ","_")
data["project_grade_category"] = data["project_grade_category"].str.rep
lace("-","_to_")
print(data['project_grade_category'].head(10))
```

```
0    Grades_PreK_to_2
1    Grades_PreK_to_2
2    Grades_PreK_to_2
3    Grades_PreK_to_2
4    Grades_PreK_to_2
5    Grades_PreK_to_2
6    Grades_PreK_to_2
7       Grades_6_to_8
8       Grades_6_to_8
9       Grades_6_to_8
Name: project_grade_category, dtype: object
```

In [0]:
```
#how to count words in corpus python
#https://stackoverflow.com/questions/8139239/how-to-count-words-in-a-co
rpus-document
my_counter = Counter()
for word in data['project_grade_category'].values:
  #print(word)
  my_counter.update({word:1})
```

In [0]:
```
print(my_counter)
```

```
Counter({'Grades_PreK_to_2': 21098, 'Grades_3_to_5': 16870, 'Grades_6_t
o_8': 7354, 'Grades_9_to_12': 4676})
```

In [0]:
```
#how to sort dictionary in python by value
#https://stackoverflow.com/questions/613183/how-do-i-sort-a-dictionary-by-value
project_grade_category_dict = dict(my_counter)
sorted_project_grade_category_dict = dict(sorted(project_grade_category_dict.items(), key=lambda kv: kv[1]))
```

preprocessing teacher prefix

In [0]:
```
print(data['teacher_prefix'].head(10))
```

```
0    Mrs.
1    Mrs.
2    Mrs.
3    Mrs.
4    Mrs.
5    Mrs.
6    Mrs.
7     Mr.
8     Ms.
9     Ms.
Name: teacher_prefix, dtype: object
```

In [0]:
```
data['teacher_prefix']=data['teacher_prefix'].str.replace('.','')
print(data['teacher_prefix'].head(10))
```

```
0    Mrs
1    Mrs
2    Mrs
3    Mrs
4    Mrs
5    Mrs
```

```
6    Mrs
7     Mr
8     Ms
9     Ms
Name: teacher_prefix, dtype: object
```

In [0]:
```
my_counter = Counter()
for word in data['teacher_prefix'].values:
  #print(word)
  my_counter.update({word:1})
```

In [0]:
```
print(my_counter)
```

```
Counter({'Mrs': 24681, 'Ms': 20179, 'Mr': 4139, 'Teacher': 999})
```

In [0]:
```
teacher_prefix_dict = dict(my_counter)
sorted_teacher_prefix_dict = dict(sorted(teacher_prefix_dict.items(), k
ey=lambda kv: kv[1]))
```

Preprocessing_school_state

In [0]:
```
print(data['school_state'].head(10))
```

```
0    IN
1    IN
2    IN
3    IN
4    IN
5    IN
6    IN
7    FL
8    AZ
9    AZ
Name: school_state, dtype: object
```

In [0]:
```
data["school_state"] = data["school_state"].str.strip()
```

```
data["school_state"] = data["school_state"].str.replace(' ', '')
```

In [0]:
```
from collections import Counter
my_counter = Counter()
for word in data['school_state'].values:
    my_counter.update(word.split())

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

text_preprocessing-essay

In [0]:
```
#merge four columns into one
data['essay']=data['project_essay_1'].map(str) +\
              data['project_essay_2'].map(str) +\
              data['project_essay_3'].map(str) +\
              data['project_essay_4'].map(str)

data.drop(['project_essay_1'], axis=1, inplace=True)
data.drop(['project_essay_2'], axis=1, inplace=True)
data.drop(['project_essay_3'], axis=1, inplace=True)
data.drop(['project_essay_4'], axis=1, inplace=True)
```

In [0]:
```
data.head(2)
```

Out[0]:

	teacher_prefix	school_state	project_grade_category	project_title	project_resource_summary	t
0	Mrs	IN	Grades_PreK_to_2	Educational Support for English Learners at Home	My students need opportunities to practice beg...	

	teacher_prefix	school_state	project_grade_category	project_title	project_resource_summary	t
1	Mrs	IN	Grades_PreK_to_2	Educational Support for English Learners at Home	My students need opportunities to practice beg...	

In [0]:
```
print(data['essay'].values[10])
print('='*50)
print(data['essay'].values[15])
```

```
\r\n\"True champions aren't always the ones that win, but those with th
e most guts.\" By Mia Hamm This quote best describes how the students a
t Cholla Middle School approach playing sports, especially for the girl
s and boys soccer teams. The teams are made up of 7th and 8th grade stu
dents, and most of them have not had the opportunity to play in an orga
nized sport due to family financial difficulties. \r\nI teach at a Titl
e One middle school in an urban neighborhood. 74% of our students quali
fy for free and reduced lunch and many come from very activity/ sport o
pportunity-poor homes. My students love to participate in sports to lea
rn new skills and be apart of team atmosphere. My school lacks the fund
ing to meet my students' needs and I am concerned that their lack of ex
posure will not prepare them for the participating in sports and teams
in high school. By the end of the school year, the goal is to provide o
ur students with an opportunity to learn a variety of soccer skills, an
d positive qualities of a person who actively participates on a team.Th
e students on the campus come to school knowing they face an uphill bat
tle when it comes to participating in organized sports. The players wou
ld thrive on the field, with the confidence from having the appropriate
soccer equipment to play soccer to the best of their abilities. The stu
dents will experience how to be a helpful person by being part of team
that teaches them to be positive, supportive, and encouraging to other
s. \r\nMy students will be using the soccer equipment during practice a
nd games on a daily basis to learn and practice the necessary skills to
develop a strong soccer team. This experience will create the opportuni
ty for students to learn about being part of a team, and how to be a po
```

```
sitive contribution for their teammates. The students will get the oppo
rtunity to learn and practice a variety of soccer skills, and how to us
e those skills during a game.  Access to this type of experience is nea
rly impossible without soccer equipment for the students/ players to ut
ilize during practice and games .nannan
==================================================
\r\n\"True champions aren't always the ones that win, but those with th
e most guts.\" By Mia Hamm This quote best describes how the students a
t Cholla Middle School approach playing sports, especially for the girl
s and boys soccer teams. The teams are made up of 7th and 8th grade stu
dents, and most of them have not had the opportunity to play in an orga
nized sport due to family financial difficulties. \r\nI teach at a Titl
e One middle school in an urban neighborhood. 74% of our students quali
fy for free and reduced lunch and many come from very activity/ sport o
pportunity-poor homes. My students love to participate in sports to lea
rn new skills and be apart of team atmosphere. My school lacks the fund
ing to meet my students' needs and I am concerned that their lack of ex
posure will not prepare them for the participating in sports and teams
in high school. By the end of the school year, the goal is to provide o
ur students with an opportunity to learn a variety of soccer skills, an
d positive qualities of a person who actively participates on a team.Th
e students on the campus come to school knowing they face an uphill bat
tle when it comes to participating in organized sports. The players wou
ld thrive on the field, with the confidence from having the appropriate
soccer equipment to play soccer to the best of their abilities. The stu
dents will experience how to be a helpful person by being part of team
that teaches them to be positive, supportive, and encouraging to other
s. \r\nMy students will be using the soccer equipment during practice a
nd games on a daily basis to learn and practice the necessary skills to
develop a strong soccer team. This experience will create the opportuni
ty for students to learn about being part of a team, and how to be a po
sitive contribution for their teammates. The students will get the oppo
rtunity to learn and practice a variety of soccer skills, and how to us
e those skills during a game.  Access to this type of experience is nea
rly impossible without soccer equipment for the students/ players to ut
ilize during practice and games .nannan
```

In [0]:
```
# https://stackoverflow.com/a/47091490/4084039
import re
```

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

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

In [0]:
```
sent=decontracted(data['essay'].values[10])
print(sent)
```

```
\r\n\"True champions are not always the ones that win, but those with t
he most guts.\" By Mia Hamm This quote best describes how the students
at Cholla Middle School approach playing sports, especially for the gir
ls and boys soccer teams. The teams are made up of 7th and 8th grade st
udents, and most of them have not had the opportunity to play in an org
anized sport due to family financial difficulties. \r\nI teach at a Tit
le One middle school in an urban neighborhood. 74% of our students qual
ify for free and reduced lunch and many come from very activity/ sport
opportunity-poor homes. My students love to participate in sports to le
arn new skills and be apart of team atmosphere. My school lacks the fun
ding to meet my students' needs and I am concerned that their lack of e
xposure will not prepare them for the participating in sports and teams
in high school. By the end of the school year, the goal is to provide o
ur students with an opportunity to learn a variety of soccer skills, an
d positive qualities of a person who actively participates on a team.Th
e students on the campus come to school knowing they face an uphill bat
tle when it comes to participating in organized sports. The players wou
ld thrive on the field, with the confidence from having the appropriate
soccer equipment to play soccer to the best of their abilities. The stu
```

```
dents will experience how to be a helpful person by being part of team
that teaches them to be positive, supportive, and encouraging to other
s. \r\nMy students will be using the soccer equipment during practice a
nd games on a daily basis to learn and practice the necessary skills to
develop a strong soccer team. This experience will create the opportuni
ty for students to learn about being part of a team, and how to be a po
sitive contribution for their teammates. The students will get the oppo
rtunity to learn and practice a variety of soccer skills, and how to us
e those skills during a game.  Access to this type of experience is nea
rly impossible without soccer equipment for the students/ players to ut
ilize during practice and games .nannan
```

In [0]:
```
# \r \n \t remove from string python: http://texthandler.com/info/remov
e-line-breaks-python/
sent = sent.replace('\\r', ' ')
sent = sent.replace('\\"', ' ')
sent = sent.replace('\\n', ' ')
print(sent)
```

```
   True champions are not always the ones that win, but those with the
most guts.  By Mia Hamm This quote best describes how the students at C
holla Middle School approach playing sports, especially for the girls a
nd boys soccer teams. The teams are made up of 7th and 8th grade studen
ts, and most of them have not had the opportunity to play in an organiz
ed sport due to family financial difficulties.   I teach at a Title One
middle school in an urban neighborhood. 74% of our students qualify for
free and reduced lunch and many come from very activity/ sport opportun
ity-poor homes. My students love to participate in sports to learn new
skills and be apart of team atmosphere. My school lacks the funding to
meet my students' needs and I am concerned that their lack of exposure
will not prepare them for the participating in sports and teams in high
school. By the end of the school year, the goal is to provide our stude
nts with an opportunity to learn a variety of soccer skills, and positi
ve qualities of a person who actively participates on a team.The studen
ts on the campus come to school knowing they face an uphill battle when
it comes to participating in organized sports. The players would thrive
on the field, with the confidence from having the appropriate soccer eq
uipment to play soccer to the best of their abilities. The students wil
l experience how to be a helpful person by being part of team that teac
```

```
hes them to be positive, supportive, and encouraging to others.   My st
udents will be using the soccer equipment during practice and games on
a daily basis to learn and practice the necessary skills to develop a s
trong soccer team. This experience will create the opportunity for stud
ents to learn about being part of a team, and how to be a positive cont
ribution for their teammates. The students will get the opportunity to
learn and practice a variety of soccer skills, and how to use those ski
lls during a game.  Access to this type of experience is nearly impossi
ble without soccer equipment for the students/ players to utilize durin
g practice and games .nannan
```

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

```
 True champions are not always the ones that win but those with the mos
t guts By Mia Hamm This quote best describes how the students at Cholla
Middle School approach playing sports especially for the girls and boys
soccer teams The teams are made up of 7th and 8th grade students and mo
st of them have not had the opportunity to play in an organized sport d
ue to family financial difficulties I teach at a Title One middle schoo
l in an urban neighborhood 74 of our students qualify for free and redu
ced lunch and many come from very activity sport opportunity poor homes
My students love to participate in sports to learn new skills and be ap
art of team atmosphere My school lacks the funding to meet my students
needs and I am concerned that their lack of exposure will not prepare t
hem for the participating in sports and teams in high school By the end
of the school year the goal is to provide our students with an opportun
ity to learn a variety of soccer skills and positive qualities of a per
son who actively participates on a team The students on the campus come
to school knowing they face an uphill battle when it comes to participa
ting in organized sports The players would thrive on the field with the
confidence from having the appropriate soccer equipment to play soccer
to the best of their abilities The students will experience how to be a
helpful person by being part of team that teaches them to be positive s
upportive and encouraging to others My students will be using the socce
r equipment during practice and games on a daily basis to learn and pra
ctice the necessary skills to develop a strong soccer team This experie
nce will create the opportunity for students to learn about being part
```

```
of a team and how to be a positive contribution for their teammates The
students will get the opportunity to learn and practice a variety of so
ccer skills and how to use those skills during a game Access to this ty
pe of experience is nearly impossible without soccer equipment for the
students players to utilize during practice and games nannan
```

In [0]:
```
# stopwords
nltk.download('stopwords')
stop = set(stopwords.words('english'))
print(stop)
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
{'few', 'most', 'same', "doesn't", 'below', 'what', 'doesn', 'the', 'm
a', 'her', "you'd", 'further', 'such', 'shan', "you'll", 'all', 'll',
'won', 'wasn', 'been', 'were', 'some', 'hasn', 'themselves', 'did',
'y', 'that', 'they', "don't", "isn't", 'there', 'd', 'its', 'didn', 'm
y', 'weren', 'ourselves', 'aren', 'herself', 'having', 'am', 'of', 'wit
h', 'now', 'does', 'before', 'them', 'yours', 'should', 'to', 'why', 'c
ouldn', 'against', 'had', "aren't", 'a', 'from', 'do', 'about', "could
n't", "you've", 'at', 'are', 'wouldn', 'here', 'him', 'in', "you're",
'me', 'have', 'which', 'no', 'mustn', "that'll", 'but', 'very', "should
n't", "mightn't", 'will', 'up', 'hers', 'once', 'this', 'being', "had
n't", 'your', "weren't", 'how', 'doing', 'an', 'after', "shan't", 'othe
r', 'over', "hasn't", 'can', 'don', 'yourselves', 'we', 'or', "wasn't",
'during', 't', 'i', 'isn', 'out', 'so', 'theirs', 'is', "should've", 't
oo', 'nor', 'whom', 'through', 'off', "haven't", 'haven', 'own', 're',
'when', 've', 'where', 'each', 'itself', 'any', "won't", 'on', 'only',
'hadn', 'above', 'myself', 'you', 'into', 'who', 'our', 'their', 'jus
t', "wouldn't", 'then', "needn't", 'm', "didn't", 'if', 'as', 'has', 'a
gain', 'his', 'it', 'he', 'by', 'both', 'be', 'down', 'while', 'should
n', 'yourself', 'was', 'and', 'mightn', 'needn', 'than', "it's", 'betwe
en', 'himself', "she's", 'ours', 'o', 'these', 'she', 'not', 'those',
'for', 'more', 'under', 's', "mustn't", 'until', 'because', 'ain'}
```

In [0]:
```
# remove stop words from essay https://stackoverflow.com/questions/1956
0498/faster-way-to-remove-stop-words-in-python
sent = ' '.join([word for word in sent.split() if word not in (stopword
```

```
s.words('english'))])
print(sent)
```

```
True champions always ones win guts By Mia Hamm This quote best describ
es students Cholla Middle School approach playing sports especially gir
ls boys soccer teams The teams made 7th 8th grade students opportunity
play organized sport due family financial difficulties I teach Title On
e middle school urban neighborhood 74 students qualify free reduced lun
ch many come activity sport opportunity poor homes My students love par
ticipate sports learn new skills apart team atmosphere My school lacks
funding meet students needs I concerned lack exposure prepare participa
ting sports teams high school By end school year goal provide students
opportunity learn variety soccer skills positive qualities person activ
ely participates team The students campus come school knowing face uphi
ll battle comes participating organized sports The players would thrive
field confidence appropriate soccer equipment play soccer best abilitie
s The students experience helpful person part team teaches positive sup
portive encouraging others My students using soccer equipment practice
games daily basis learn practice necessary skills develop strong soccer
team This experience create opportunity students learn part team positi
ve contribution teammates The students get opportunity learn practice v
ariety soccer skills use skills game Access type experience nearly impo
ssible without soccer equipment students players utilize practice games
nannan
```

In [0]:
```
# Combining all the above statemennts
from tqdm import tqdm
from tqdm.auto import tqdm, trange
preprocessed_essays = []
# tqdm is for printing the status bar
for sentance in tqdm(data['essay'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    sent = ' '.join([word for word in sent.split() if word not in (stop
words.words('english'))])
    preprocessed_essays.append(sent.lower().strip())
```

In [0]:
```
#after preprocessing
preprocessed_essays[10]
```

Out[0]:
```
'true champions always ones win guts by mia hamm this quote best descri
bes students cholla middle school approach playing sports especially gi
rls boys soccer teams the teams made 7th 8th grade students opportunity
play organized sport due family financial difficulties i teach title on
e middle school urban neighborhood 74 students qualify free reduced lun
ch many come activity sport opportunity poor homes my students love par
ticipate sports learn new skills apart team atmosphere my school lacks
funding meet students needs i concerned lack exposure prepare participa
ting sports teams high school by end school year goal provide students
opportunity learn variety soccer skills positive qualities person activ
ely participates team the students campus come school knowing face uphi
ll battle comes participating organized sports the players would thrive
field confidence appropriate soccer equipment play soccer best abilitie
s the students experience helpful person part team teaches positive sup
portive encouraging others my students using soccer equipment practice
games daily basis learn practice necessary skills develop strong soccer
team this experience create opportunity students learn part team positi
ve contribution teammates the students get opportunity learn practice v
ariety soccer skills use skills game access type experience nearly impo
ssible without soccer equipment students players utilize practice games
nannan'
```

In [0]:
```
data['preprocessed_essays']=preprocessed_essays
data.drop(['essay'],axis=1, inplace=True)
```

In [0]:
```
print("\n".join(list(data.columns.values)))
```

```
teacher_prefix
school_state
project_grade_category
project_title
project_resource_summary
teacher_number_of_previously_posted_projects
project_is_approved
```

```
quantity
price
clean_categories
clean_subcategories
preprocessed_essays
```

text peprocessing resource_summary

In [0]:
```
sent=decontracted(data['project_resource_summary'].values[5])
print(sent)
```

```
My students need opportunities to practice beginning reading skills in
English at home.
```

In [0]:
```
# \r \n \t remove from string python: http://texthandler.com/info/remov
e-line-breaks-python/
sent = sent.replace('\\r', ' ')
sent = sent.replace('\\"', ' ')
sent = sent.replace('\\n', ' ')
print(sent)
```

```
My students need opportunities to practice beginning reading skills in
English at home.
```

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

```
My students need opportunities to practice beginning reading skills in
English at home
```

In [0]:
```
# remove stop words from essay https://stackoverflow.com/questions/1956
0498/faster-way-to-remove-stop-words-in-python
sent = ' '.join([word for word in sent.split() if word not in (stopword
s.words('english'))])
print(sent)
```

```
My students need opportunities practice beginning reading skills Englis
h home
```

In [0]:
```
# Combining all the above statemennts
from tqdm import tqdm
from tqdm.auto import tqdm, trange
preprocessed_project_resource_summary = []
# tqdm is for printing the status bar
for sentance in tqdm(data['project_resource_summary'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    sent = ' '.join([word for word in sent.split() if word not in (stop
words.words('english'))])
    preprocessed_project_resource_summary.append(sent.lower().strip())
```

In [0]:
```
preprocessed_project_resource_summary[10]
```

Out[0]:
```
'my students need shine guards athletic socks soccer balls goalie glove
s training materials upcoming soccer season'
```

In [0]:
```
data['preprocessed_project_resource_summary']=preprocessed_project_reso
urce_summary
data.drop('project_resource_summary',axis=1,inplace=True)
```

In [0]:
```
print("\n".join(list(data.columns.values)))
```

```
teacher_prefix
school_state
project_grade_category
project_title
teacher_number_of_previously_posted_projects
project_is_approved
quantity
price
```

```
clean_categories
clean_subcategories
preprocessed_essays
preprocessed_project_resource_summary
```

In [0]:
```
data.head(1)
```

Out[0]:

	teacher_prefix	school_state	project_grade_category	project_title	teacher_number_of_previously
0	Mrs	IN	Grades_PreK_to_2	Educational Support for English Learners at Home	

project_title_preprocessing

In [0]:
```
sent=decontracted(data['project_title'].values[35])
print(sent)
```

```
Just For the Love of Reading--\r\nPure Pleasure
```

In [0]:
```
# \r \n \t remove from string python: http://texthandler.com/info/remove-line-breaks-python/
sent = sent.replace('\\r', ' ')
sent = sent.replace('\\"', ' ')
sent = sent.replace('\\n', ' ')
print(sent)
```

```
Just For the Love of Reading--  Pure Pleasure
```

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

```
Just For the Love of Reading Pure Pleasure
```

In [0]:
```
# remove stop words from essay https://stackoverflow.com/questions/19560498/faster-way-to-remove-stop-words-in-python
sent = ' '.join([word for word in sent.split() if word not in (stopwords.words('english'))])
print(sent)
```

```
Just For Love Reading Pure Pleasure
```

In [0]:
```
# Combining all the above statemennts
from tqdm import tqdm
from tqdm.auto import tqdm, trange
preprocessed_project_title = []
# tqdm is for printing the status bar
for sentance in tqdm(data['project_title'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    sent = ' '.join([word for word in sent.split() if word not in (stopwords.words('english'))])
    preprocessed_project_title.append(sent.lower().strip())
```

In [0]:
```
preprocessed_project_title[35]
```

Out[0]:
```
'just for love reading pure pleasure'
```

In [0]:
```
data["preprocessed_project_title"] = preprocessed_project_title
data.drop('project_title',axis=1,inplace=True)
```

In [0]:
```
print("\n".join(list(data.columns.values)))
```

```
teacher_prefix
school_state
project_grade_category
teacher_number_of_previously_posted_projects
```

```
project_is_approved
quantity
price
clean_categories
clean_subcategories
preprocessed_essays
preprocessed_project_resource_summary
preprocessed_project_title
```

In [0]:
```
#check if data is imbalance
data['project_is_approved'].value_counts()
```

Out[0]:
```
1    39965
0    10033
Name: project_is_approved, dtype: int64
```

In [0]:
```
y = data['project_is_approved'].values
X = data.drop(['project_is_approved'], axis=1)
X.head(1)
```

Out[0]:

	teacher_prefix	school_state	project_grade_category	teacher_number_of_previously_posted_proj
0	Mrs	IN	Grades_PreK_to_2	

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

In [0]:
```
# train test split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, stratify=y)
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33, stratify=y_train)
```

In [0]:
```
print(X_train.shape, y_train.shape)
print(X_cv.shape, y_cv.shape)
print(X_test.shape, y_test.shape)

print("="*100)
```

```
(22443, 11) (22443,)
(11055, 11) (11055,)
(16500, 11) (16500,)
====================================================================================================
```

Preparing data for models

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

Out[0]:
```
Index(['teacher_prefix', 'school_state', 'project_grade_category',
       'teacher_number_of_previously_posted_projects', 'quantity', 'pri
ce',
       'clean_categories', 'clean_subcategories', 'preprocessed_essay
s',
       'preprocessed_project_resource_summary', 'preprocessed_project_t
itle'],
      dtype='object')
```

we are going to consider

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

- project_title : text data
- text : text data

```
- project_resource_summary: text data (optinal)

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

1.4.1 Vectorizing Categorical data/one hot encoding

school_state

In [0]:
```
#school state
vectorizer = CountVectorizer()
vectorizer = CountVectorizer(vocabulary=list(sorted_state_dict.keys()),
 lowercase=False, binary=True)
vectorizer.fit(X_train['school_state'].values) # fit has to happen only
 on train data

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

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

```
After vectorizations
(22443, 51) (22443,)
(11055, 51) (11055,)
(16500, 51) (16500,)
['VT', 'ND', 'MT', 'RI', 'DE', 'WY', 'NH', 'SD', 'ME', 'NE', 'KS', 'N
M', 'HI', 'AK', 'WV', 'IA', 'DC', 'ID', 'MS', 'AR', 'CO', 'AL', 'MN',
```

```
'KY', 'NV', 'TN', 'MD', 'OR', 'WI', 'CT', 'AZ', 'VA', 'UT', 'NJ', 'MA',
'MO', 'OH', 'IN', 'LA', 'OK', 'WA', 'PA', 'GA', 'MI', 'SC', 'IL', 'NC',
'FL', 'NY', 'TX', 'CA']
==========================================================================
==============================
```

project_grade_category

In [0]:

```
#project_grade_category
vectorizer = CountVectorizer()
vectorizer = CountVectorizer(vocabulary=list(sorted_project_grade_categ
ory_dict.keys()), lowercase=False, binary=True)
vectorizer.fit(X_train['project_grade_category'].values) # fit has to h
appen only on train data

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

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

```
After vectorizations
(22443, 4) (22443,)
(11055, 4) (11055,)
(16500, 4) (16500,)
['Grades_9_to_12', 'Grades_6_to_8', 'Grades_3_to_5', 'Grades_PreK_to_
2']
==========================================================================
```

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

In [0]:
```
X_cv_grade_ohe[:2]
```

Out[0]:
```
<2x4 sparse matrix of type '<class 'numpy.int64'>'
        with 2 stored elements in Compressed Sparse Row format>
```

clean category

In [0]:
```
#clean_categories
vectorizer = CountVectorizer()
vectorizer = CountVectorizer(vocabulary=list(sorted_cat_dict.keys()), l
owercase=False, binary=True)
vectorizer.fit(X_train['clean_categories'].values) # fit has to happen
 only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_clean_ohe = vectorizer.transform(X_train['clean_categories'].va
lues)
X_cv_clean_ohe = vectorizer.transform(X_cv['clean_categories'].values)
X_test_clean_ohe = vectorizer.transform(X_test['clean_categories'].valu
es)

print("After vectorizations")
print(X_train_clean_ohe.shape, y_train.shape)
print(X_cv_clean_ohe.shape, y_cv.shape)
print(X_test_clean_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(22443, 9) (22443,)
(11055, 9) (11055,)
(16500, 9) (16500,)
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearn
ing', 'Health_Sports', 'SpecialNeeds', 'Math_Science', 'Literacy_Langua
ge']
====================================================================
```

==========

clean sub *category*

In [0]:
```
#clean_sub_categories
vectorizer = CountVectorizer()
vectorizer = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys
()), lowercase=False, binary=True)
vectorizer.fit(X_train['clean_subcategories'].values) # fit has to happ
en only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_sub_ohe = vectorizer.transform(X_train['clean_subcategories'].v
alues)
X_cv_sub_ohe = vectorizer.transform(X_cv['clean_subcategories'].values)
X_test_sub_ohe = vectorizer.transform(X_test['clean_subcategories'].val
ues)

print("After vectorizations")
print(X_train_sub_ohe.shape, y_train.shape)
print(X_cv_sub_ohe.shape, y_cv.shape)
print(X_test_sub_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(22443, 30) (22443,)
(11055, 30) (11055,)
(16500, 30) (16500,)
['Economics', 'FinancialLiteracy', 'Civics_Government', 'CommunityServi
ce', 'Extracurricular', 'ForeignLanguages', 'ParentInvolvement', 'Socia
lSciences', 'PerformingArts', 'NutritionEducation', 'College_CareerPre
p', 'Warmth', 'Care_Hunger', 'TeamSports', 'Music', 'Other', 'History_G
eography', 'CharacterEducation', 'Health_LifeScience', 'EarlyDevelopmen
t', 'Gym_Fitness', 'ESL', 'EnvironmentalScience', 'VisualArts', 'Applie
dSciences', 'Health_Wellness', 'SpecialNeeds', 'Literature_Writing', 'M
athematics', 'Literacy']
====================================================================
```

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

teacher prefix

In [0]:
```
#teacher prefix
vectorizer = CountVectorizer()
vectorizer = CountVectorizer(vocabulary=list(sorted_teacher_prefix_dict
.keys()), lowercase=False, binary=True)
vectorizer.fit(X_train['teacher_prefix'].values) # fit has to happen on
ly on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_prefix_ohe = vectorizer.transform(X_train['teacher_prefix'].val
ues.astype(str))
X_cv_prefix_ohe = vectorizer.transform(X_cv['teacher_prefix'].values.as
type(str))
X_test_prefix_ohe = vectorizer.transform(X_test['teacher_prefix'].value
s.astype(str))

print("After vectorizations")
print(X_train_prefix_ohe.shape, y_train.shape)
print(X_cv_prefix_ohe.shape, y_cv.shape)
print(X_test_prefix_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(22443, 4) (22443,)
(11055, 4) (11055,)
(16500, 4) (16500,)
['Teacher', 'Mr', 'Ms', 'Mrs']
============================================================================
============================
```

Vectorizing numerical data

In [0]:
```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
# normalizer.fit(X_train['price'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.
normalizer.fit(X_train['price'].values.reshape(-1,1))

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

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

```
After vectorizations
(22443, 1) (22443,)
(11055, 1) (11055,)
(16500, 1) (16500,)
====================================================================
============================
```

In [0]:
```
X_train['preprocessed_essays'][:5]
```

Out[0]:
```
2733     many students come low income homes basic supp...
10616    as teacher language immersion school i 46 stud...
31195    i 24 amazing kindergarteners live low income h...
30804    my thirty 3rd grade students amazingly bright ...
3986     my students creative active absolutely amazing...
Name: preprocessed_essays, dtype: object
```

Vectorizing Text data

BOW-eassy

In [0]:
```
##essay
from sklearn.feature_extraction.text import CountVectorizer
vectorizer=CountVectorizer(min_df=10,max_features=5000)
vectorizer.fit(X_train['preprocessed_essays'].values)

# we use the fitted CountVectorizer to convert the text to vector
X_train_essays_bow=vectorizer.transform(X_train['preprocessed_essays'].
values)
X_test_essays_bow=vectorizer.transform(X_test['preprocessed_essays'].va
lues)
X_cv_essays_bow=vectorizer.transform(X_cv['preprocessed_essays'].values
)

print("After vectorizations")
print(X_train_essays_bow.shape, y_train.shape)
print(X_cv_essays_bow.shape, y_cv.shape)
print(X_test_essays_bow.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(22443, 5000) (22443,)
(11055, 5000) (11055,)
(16500, 5000) (16500,)
====================================================================
============================
```

In [0]:
```
#project title
print(X_train.shape, y_train.shape)
print(X_cv.shape, y_cv.shape)
print(X_test.shape, y_test.shape)

print("="*100)
from sklearn.feature_extraction.text import CountVectorizer
vectorizer=CountVectorizer(min_df=10)
```

```
vectorizer.fit(X_train['preprocessed_project_title'].values)

# we use the fitted CountVectorizer to convert the text to vector
X_train_title_bow=vectorizer.transform(X_train['preprocessed_project_ti
tle'].values)
X_test_title_bow=vectorizer.transform(X_test['preprocessed_project_titl
e'].values)
X_cv_title_bow=vectorizer.transform(X_cv['preprocessed_project_title'].
values)

print("After vectorizations")
print(X_train_title_bow.shape, y_train.shape)
print(X_cv_title_bow.shape, y_cv.shape)
print(X_test_title_bow.shape, y_test.shape)
print("="*100)
```

```
(22443, 11) (22443,)
(11055, 11) (11055,)
(16500, 11) (16500,)
====================================================================
==============================
After vectorizations
(22443, 1382) (22443,)
(11055, 1382) (11055,)
(16500, 1382) (16500,)
====================================================================
==============================
```

apply knn set 1

In [0]:

```
print(X_train_state_ohe.shape)
print(X_train_grade_ohe.shape)
print(X_train_clean_ohe.shape)
print(X_train_sub_ohe.shape)
print(X_train_prefix_ohe.shape)
print(X_train_price_norm.shape)
print(X_train_title_bow.shape)
print(X_train_essays_bow.shape)
```

```
print(X_cv_state_ohe.shape)
print(X_cv_grade_ohe.shape)
print(X_cv_clean_ohe.shape)
print(X_cv_sub_ohe.shape)
print(X_cv_prefix_ohe.shape)
print(X_cv_price_norm.shape)
print(X_cv_title_bow.shape)
print(X_cv_essays_bow.shape)

print(X_test_state_ohe.shape)
print(X_test_grade_ohe.shape)
print(X_test_clean_ohe.shape)
print(X_test_sub_ohe.shape)
print(X_test_prefix_ohe.shape)
print(X_test_price_norm.shape)
print(X_test_title_bow.shape)
print(X_test_essays_bow.shape)
```

```
(22443, 51)
(22443, 4)
(22443, 9)
(22443, 30)
(22443, 4)
(22443, 1)
(22443, 1382)
(22443, 5000)
(11055, 51)
(11055, 4)
(11055, 9)
(11055, 30)
(11055, 4)
(11055, 1)
(11055, 1382)
(11055, 5000)
(16500, 51)
(16500, 4)
(16500, 9)
(16500, 30)
(16500, 4)
```

```
(16500, 1)
(16500, 1382)
(16500, 5000)
```

In [0]:
```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr_set1=hstack((X_train_state_ohe,X_train_grade_ohe,X_train_clean_ohe,X_train_sub_ohe,X_train_prefix_ohe,X_train_price_norm,X_train_title_bow,X_train_essays_bow)).tocsr()

X_te_set1=hstack((X_test_state_ohe,X_test_grade_ohe,X_test_clean_ohe,X_test_sub_ohe,X_test_prefix_ohe, X_test_price_norm,X_test_title_bow,X_test_essays_bow)).tocsr()

X_cr_set1=hstack((X_cv_state_ohe,X_cv_grade_ohe,X_cv_clean_ohe,X_cv_sub_ohe,X_cv_prefix_ohe,X_cv_price_norm,X_cv_essays_bow,X_cv_title_bow)).tocsr()

print("Final Data matrix")
print(X_tr_set1.shape)
print(X_te_set1.shape)
print(X_cr_set1.shape)
print("="*100)
```

```
Final Data matrix
(22443, 6481)
(16500, 6481)
(11055, 6481)
====================================================================================================
```

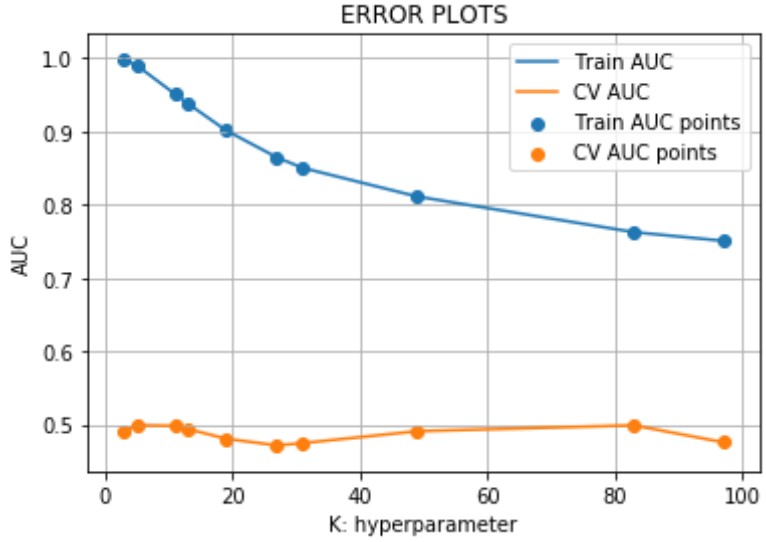
In [0]:
```
def batch_predict(clf, data):
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
    # not the predicted outputs
   # print(data.shape)
    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000
```

```
    # consider you X_tr shape is 49041, then your tr_loop will be 49041
 - 49041%1000 = 49000
    # in this for loop we will iterate unti the last 1000 multiplier
    #print ('pythoon')
    for i in range(0, tr_loop, 1000):

        y_data_pred.extend(clf.predict_proba(data[i:i+1000])[:,1])

    # we will be predicting for the last data points

    if data.shape[0]%1000 !=0:

        y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

    return y_data_pred
```

In [0]:

```
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
"""
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 clas
s, 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 wit
h greater label.

"""
train_auc = []
cv_auc = []
K = [3,5,11,13,19,27,31,49,83,97]
for i in tqdm(K):
    neigh = KNeighborsClassifier(n_neighbors=i,algorithm='brute',n_jobs
=-1)
    neigh.fit(X_tr_set1, y_train)
```

```
    y_train_pred = batch_predict(neigh, X_tr_set1)
    y_cv_pred = batch_predict(neigh, X_cr_set1)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probab
ility estimates of the positive 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()
```



In [0]:
```
X_tr_set1.shape, y_train.shape,X_te_set1.shape,y_cv.shape
```

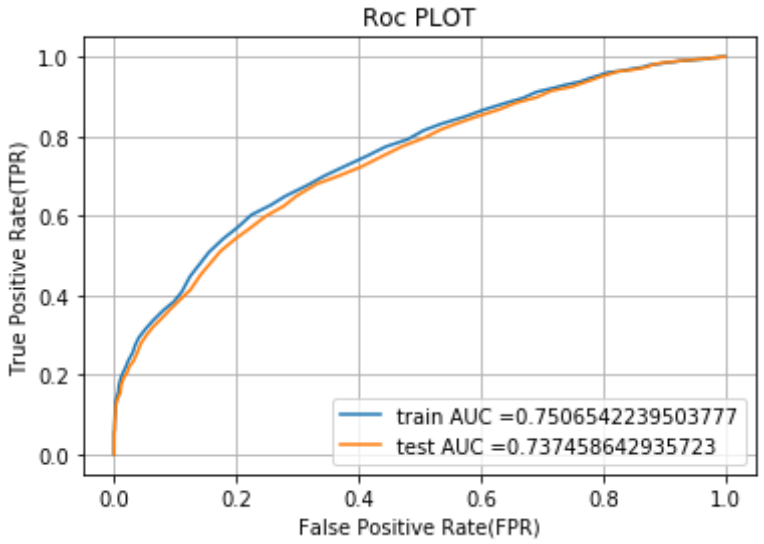
Out[0]:
```
((22443, 6481), (22443,), (16500, 6481), (11055,))
```

In [0]:
```
# from the error plot we choose K such that, we will have maximum AUC o
n cv data and gap between the train and cv is less
# Note: based on the method you use you might get different hyperparame
ter values as best one
# so, you choose according to the method you choose, you use gridsearch
 if you are having more computing power and note it will take more time
# if you increase the cv values in the GridSearchCV you will get more r
obust results.

#here we are choosing the best_k based on forloop results
best_k = 97
```

In [0]:
```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc
_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc
print(X_tr_set1.shape)
neigh = KNeighborsClassifier(n_neighbors=best_k, n_jobs=-1)
neigh.fit(X_tr_set1, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probabilit
y estimates of the positive class
# not the predicted outputs

y_train_pred = batch_predict(neigh, X_tr_set1)
y_test_pred = batch_predict(neigh, X_te_set1)

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

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, t
rain_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_
tpr)))
plt.legend()
plt.xlabel("False Positive Rate(FPR)")
```

```
plt.ylabel("True Positive Rate(TPR)")
plt.title("Roc PLOT")
plt.grid()
plt.show()
```

```
(22443, 6481)
```



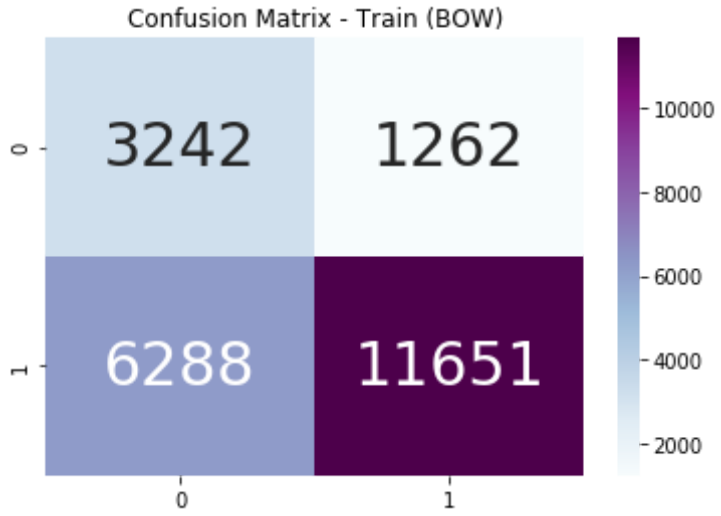
In [0]:
```
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def find_best_threshold(threshould, fpr, tpr):
    t = threshould[np.argmax(tpr*(1-fpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is
 very high
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for th
reshold", np.round(t,3))
    return t

def predict_with_best_t(proba, threshould):
    predictions = []
    for i in proba:
        if i>=threshould:
            predictions.append(1)
```

```
        else:
            predictions.append(0)
    return predictions
```

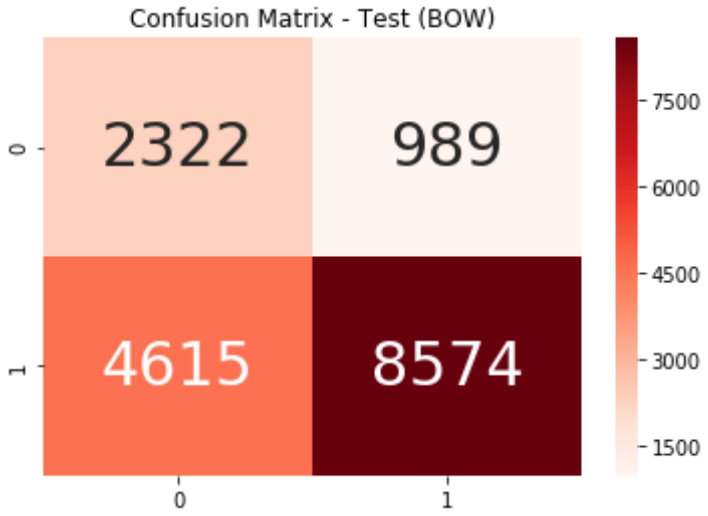
In [0]:
```
print("="*100)
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_
t)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t
)))
```

```
====================================================================
==============================
the maximum value of tpr*(1-fpr) 0.467497831857062 for threshold 0.711
Train confusion matrix
[[ 3242  1262]
 [ 6288 11651]]
Test confusion matrix
[[2322  989]
 [4615 8574]]
```

In [0]:
```
ax = plt.axes()
df_cm = confusion_matrix(y_train, predict_with_best_t(y_train_pred, bes
t_t))
sns.heatmap(df_cm, annot=True, annot_kws={"size": 30}, fmt='d',cmap="Bu
Pu", ax = ax )

ax.set_title('Confusion Matrix - Train (BOW)')
plt.show()
```



In [0]:
```
ax = plt.axes()
df_cm = confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_
t))
sns.heatmap(df_cm, annot=True, annot_kws={"size": 30}, fmt='d',cmap="Re
ds", ax = ax )

ax.set_title('Confusion Matrix - Test (BOW)')
plt.show()
```



2.TFIDF

In [0]:
```
#project title
print(X_train.shape, y_train.shape)
print(X_cv.shape, y_cv.shape)
print(X_test.shape, y_test.shape)

print("="*100)

from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer=TfidfVectorizer(min_df=10)
vectorizer.fit(X_train['preprocessed_project_title'].values)

# we use the fitted TfidfVectorizer to convert the text to vector
X_train_title_tfidf=vectorizer.transform(X_train['preprocessed_project_
title'].values)
X_test_title_tfidf=vectorizer.transform(X_test['preprocessed_project_ti
tle'].values)
X_cv_title_tfidf=vectorizer.transform(X_cv['preprocessed_project_title'
```

```
].values)

print("After vectorizations")
print(X_train_title_tfidf.shape, y_train.shape)
print(X_cv_title_tfidf.shape, y_cv.shape)
print(X_test_title_tfidf.shape, y_test.shape)
print("="*100)
```

```
(22443, 11) (22443,)
(11055, 11) (11055,)
(16500, 11) (16500,)
====================================================================================================
After vectorizations
(22443, 1382) (22443,)
(11055, 1382) (11055,)
(16500, 1382) (16500,)
====================================================================================================
```

In [0]:

```
##essay
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer=TfidfVectorizer(min_df=10,max_features=5000)
vectorizer.fit(X_train['preprocessed_essays'].values)

# we use the fitted CountVectorizer to convert the text to vector
X_train_essays_tfidf=vectorizer.transform(X_train['preprocessed_essays'
].values)
X_test_essays_tfidf=vectorizer.transform(X_test['preprocessed_essays'].
values)
X_cv_essays_tfidf=vectorizer.transform(X_cv['preprocessed_essays'].valu
es)

print("After vectorizations")
print(X_train_essays_tfidf.shape, y_train.shape)
print(X_cv_essays_tfidf.shape, y_cv.shape)
print(X_test_essays_tfidf.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
```

```
(22443, 5000) (22443,)
(11055, 5000) (11055,)

(16500, 5000) (16500,)
==========================================================================
============================
```

set 2

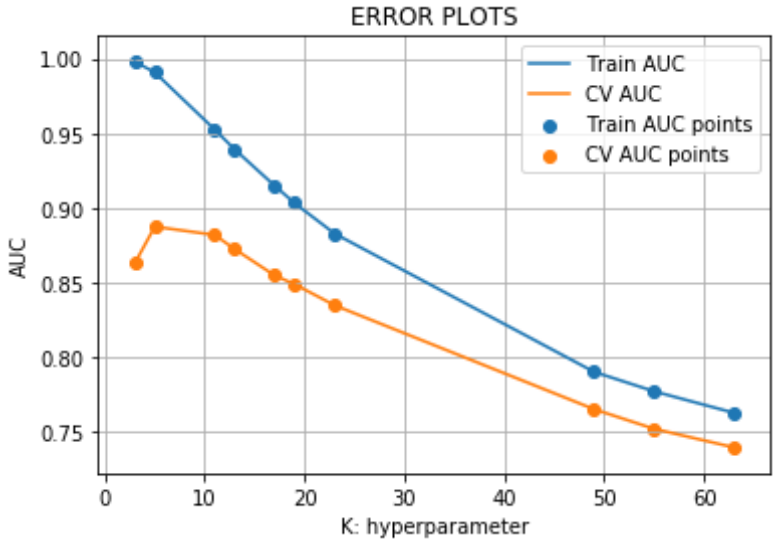
In [0]:
```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr_set2=hstack((X_train_state_ohe,X_train_grade_ohe,X_train_clean_ohe,X_train_sub_ohe,X_train_prefix_ohe,X_train_price_norm,X_train_title_tfidf,X_train_essays_tfidf)).tocsr()

X_te_set2=hstack((X_test_state_ohe,X_test_grade_ohe,X_test_clean_ohe,X_test_sub_ohe,X_test_prefix_ohe, X_test_price_norm,X_test_title_tfidf,X_test_essays_tfidf)).tocsr()

X_cr_set2=hstack((X_cv_state_ohe,X_cv_grade_ohe,X_cv_clean_ohe,X_cv_sub_ohe,X_cv_prefix_ohe,X_cv_price_norm,X_cv_essays_tfidf,X_cv_title_tfidf)).tocsr()

print("Final Data matrix")
print(X_tr_set2.shape)
print(X_te_set2.shape)
print(X_cr_set2.shape)
print("="*100)
```

```
Final Data matrix
(22443, 6481)
(16500, 6481)
(11055, 6481)
==========================================================================
============================
```

In [0]:
```
def batch_predict(clf, data):
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probab
ility estimates of the positive class
    # not the predicted outputs
   # print(data.shape)
    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000
    # consider you X_tr shape is 49041, then your tr_loop will be 49041
 - 49041%1000 = 49000
    # in this for loop we will iterate unti the last 1000 multiplier

    for i in range(0, tr_loop, 1000):

        y_data_pred.extend(clf.predict_proba(data[i:i+1000])[:,1])

    # we will be predicting for the last data points

    if data.shape[0]%1000 !=0:

        y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])
    return y_data_pred
```

In [97]:
```
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
"""
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 clas
s, 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 wit
h greater label.

"""

train_auc = []
```

```
cv_auc = []
K = [3,5,11,13,17,19,23,49,55,63]
for i in tqdm(K):
    neigh = KNeighborsClassifier(n_neighbors=i, n_jobs=-1)
    neigh.fit(X_tr_set2, y_train)

    y_train_pred = batch_predict(neigh, X_tr_set2)
    y_cv_pred = batch_predict(neigh, X_cr_set2)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probab
ility estimates of the positive 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()
```



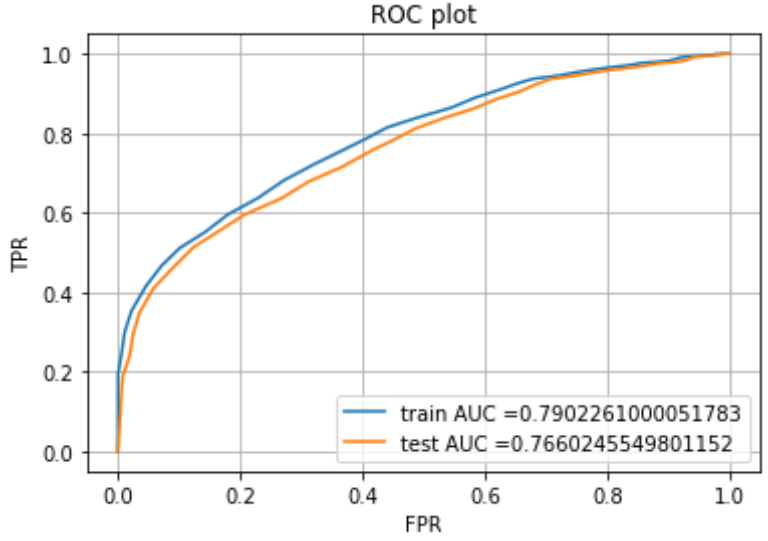
In [99]:
```
X_tr_set2.shape,X_te_set2.shape,y_train.shape
```

Out[99]:
```
((22443, 6481), (16500, 6481), (22443,))
```

In [0]:
```
# from the error plot we choose K such that, we will have maximum AUC o
n cv data and gap between the train and cv is less
# Note: based on the method you use you might get different hyperparame
ter values as best one
# so, you choose according to the method you choose, you use gridsearch
 if you are having more computing power and note it will take more time
# if you increase the cv values in the GridSearchCV you will get more r
ebust results.

#here we are choosing the best_k based on forloop results
best_k =49
```

In [101]:
```
# 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, n_jobs=-1)
neigh.fit(X_tr_set2, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probabilit
y estimates of the positive class
# not the predicted outputs

y_train_pred = batch_predict(neigh, X_tr_set2)
y_test_pred = batch_predict(neigh, X_te_set2)

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

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, t
rain_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_
tpr)))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC plot")
plt.grid()
plt.show()
```

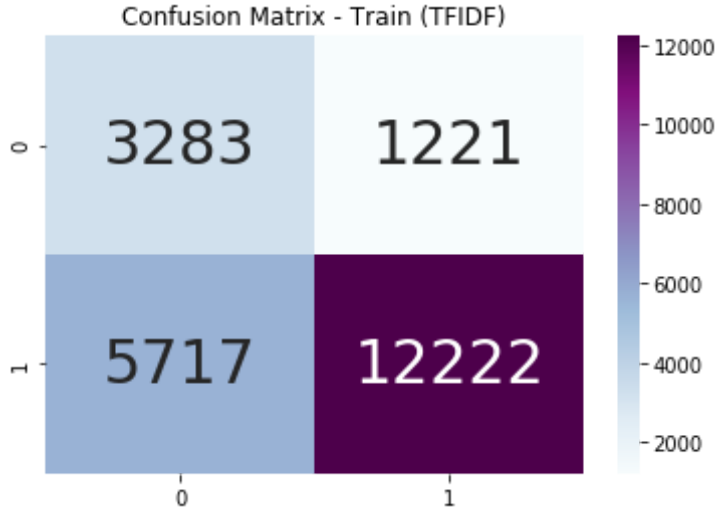


In [0]:
```
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def find_best_threshold(threshould, fpr, tpr):
    t = threshould[np.argmax(tpr*(1-fpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is
 very high
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for th
reshold", np.round(t,3))
    return t

def predict_with_best_t(proba, threshould):
    predictions = []
    for i in proba:
        if i>=threshould:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

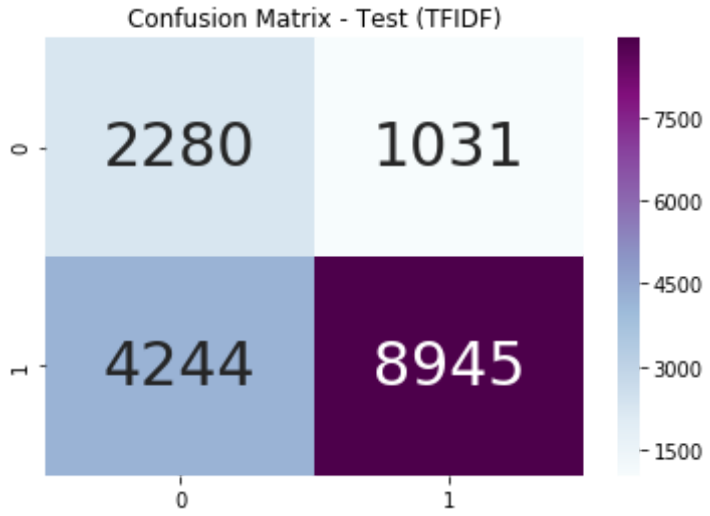
In [103]:
```
print("="*100)
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_
t)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t
)))
```

```
====================================================================
============================
the maximum value of tpr*(1-fpr) 0.49661124630272097 for threshold 0.79
6
Train confusion matrix
[[ 3283  1221]
 [ 5717 12222]]
Test confusion matrix
[[2280 1031]
 [4244 8945]]
```

In [104]:
```
ax = plt.axes()
df_cm = confusion_matrix(y_train, predict_with_best_t(y_train_pred, bes
t_t))
sns.heatmap(df_cm, annot=True, annot_kws={"size": 30}, fmt='d',cmap="Bu
Pu", ax = ax )

ax.set_title('Confusion Matrix - Train (TFIDF)')
plt.show()
```



In [105]:
```
ax = plt.axes()
df_cm = confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_
t))
sns.heatmap(df_cm, annot=True, annot_kws={"size": 30}, fmt='d',cmap="Bu
Pu", ax = ax )

ax.set_title('Confusion Matrix - Test (TFIDF)')
plt.show()
```



set 3

Using Pretrained Models: Avg W2V

In [0]:
```
# Reading glove vectors in python: https://stackoverflow.com/a/3823034
9/4084039
def loadGloveModel(gloveFile):
    print ("Loading Glove Model")
    f = open(gloveFile,'r', encoding="utf8")
    model = {}
    for line in tqdm(f):
        splitLine = line.split()
        word = splitLine[0]
        embedding = np.array([float(val) for val in splitLine[1:]])
        model[word] = embedding
    print ("Done.",len(model)," words loaded!")
```

```
    return model
# model = loadGloveModel('glove/glove.42B.300d.txt')
```

In [107]:
```
from tqdm.auto import tqdm ,trange
model_1 = loadGloveModel("gdrive/My Drive/applied ai/assignment3/glove.
42B.300d.txt")
```

```
Loading Glove Model


Done. 1917495  words loaded!
```

In [108]:
```
words = []
for i in preprocessed_essays:
    words.extend(i.split(' '))

for i in preprocessed_project_title:
    words.extend(i.split(' '))

for i in preprocessed_project_resource_summary:
    words.extend(i.split(' '))

print("all the words in the coupus", len(words))
words = set(words)
print("the unique words in the coupus", len(words))

inter_words = set(model_1.keys()).intersection(words)
print("The number of words that are present in both glove vectors and o
ur coupus", \
      len(inter_words),"(",np.round(len(inter_words)/len(words)*100,3),
"%)")

words_courpus = {}
words_glove = set(model_1.keys())
for i in words:
    if i in words_glove:
        words_courpus[i] = model_1[i]
print("word 2 vec length", len(words_courpus))
```

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

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

```
all the words in the coupus 8674778
the unique words in the coupus 22473
The number of words that are present in both glove vectors and our coup
us 21597 ( 96.102 %)
word 2 vec length 21597
```

In [109]:
```
avg_w2v_vectors_train_essays= []; # the avg-w2v for each sentence/revie
w is stored in this list
for sentence in tqdm(X_train['preprocessed_essays'].values): # for each
 review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/re
view
    for word in sentence.split(): # for each word in a review/sentence
        if word in words_glove:
            vector += model_1[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_train_essays.append(vector)
```

In [110]:
```
avg_w2v_vectors_test_essays= []; # the avg-w2v for each sentence/review
 is stored in this list
for sentence in tqdm(X_test['preprocessed_essays'].values): # for each
 review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/re
view
    for word in sentence.split(): # for each word in a review/sentence
```

```
        if word in words_glove:
            vector += model_1[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_test_essays.append(vector)
```

In [111]:
```
avg_w2v_vectors_cv_essays = []; # the avg-w2v for each sentence/review
 is stored in this list
for sentence in tqdm(X_cv['preprocessed_essays'].values): # for each re
view/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/re
view
    for word in sentence.split(): # for each word in a review/sentence
        if word in words_glove:
            vector += model_1[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_cv_essays.append(vector)
```

In [113]:
```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_vectors_train_summary = []; # the avg-w2v for each sentence/rev
iew is stored in this list
for sentence in tqdm(X_train['preprocessed_project_resource_summary'].v
alues): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/re
view
    for word in sentence.split(): # for each word in a review/sentence
        if word in words_glove:
            vector += model_1[word]
            cnt_words += 1
    if cnt_words != 0:
```

```
        vector /= cnt_words
    avg_w2v_vectors_train_summary.append(vector)

print(len(avg_w2v_vectors_train_summary))
```

```
22443
```

In [114]:
```
avg_w2v_vectors_test_summary= []; # the avg-w2v for each sentence/revie
w is stored in this list
for sentence in tqdm(X_test['preprocessed_project_resource_summary'].va
lues): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/re
view
    for word in sentence.split(): # for each word in a review/sentence
        if word in words_glove:
            vector += model_1[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_test_summary.append(vector)
```

In [115]:
```
avg_w2v_vectors_cv_summary= []; # the avg-w2v for each sentence/review
 is stored in this list
for sentence in tqdm(X_cv['preprocessed_project_resource_summary'].valu
es): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/re
view
    for word in sentence.split(): # for each word in a review/sentence
        if word in words_glove:
            vector += model_1[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_cv_summary.append(vector)
```

In [116]:
```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_vectors_train_title= []; # the avg-w2v for each sentence/review
 is stored in this list
for sentence in tqdm(X_train['preprocessed_project_title'].values): # f
or each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/re
view
    for word in sentence.split(): # for each word in a review/sentence
        if word in words_glove:
            vector += model_1[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_train_title.append(vector)

print(len(avg_w2v_vectors_train_title))
```

```
22443
```

In [117]:
```
avg_w2v_vectors_test_title= []; # the avg-w2v for each sentence/review
 is stored in this list
for sentence in tqdm(X_test['preprocessed_project_title'].values): # fo
r each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/re
view
    for word in sentence.split(): # for each word in a review/sentence
        if word in words_glove:
            vector += model_1[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_test_title.append(vector)
```

In [118]:
```
avg_w2v_vectors_cv_title= []; # the avg-w2v for each sentence/review is
 stored in this list
for sentence in tqdm(X_cv['preprocessed_project_title'].values): # for
 each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/re
view
    for word in sentence.split(): # for each word in a review/sentence
        if word in words_glove:
            vector += model_1[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_cv_title.append(vector)
```

In [119]:
```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/40840
39
from scipy.sparse import hstack
X_tr_set3=hstack((avg_w2v_vectors_train_essays,avg_w2v_vectors_train_ti
tle,X_train_state_ohe,X_train_grade_ohe,X_train_clean_ohe,X_train_sub_o
he,X_train_prefix_ohe,X_train_price_norm,)).tocsr()

X_te_set3=hstack((avg_w2v_vectors_test_essays,avg_w2v_vectors_test_titl
e,X_test_state_ohe,X_test_grade_ohe,X_test_clean_ohe,X_test_sub_ohe,X_t
est_prefix_ohe, X_test_price_norm)).tocsr()

X_cr_set3=hstack((avg_w2v_vectors_cv_essays,avg_w2v_vectors_cv_title,X_
cv_state_ohe,X_cv_grade_ohe,X_cv_clean_ohe,X_cv_sub_ohe,X_cv_prefix_ohe
,X_cv_price_norm)).tocsr()

print("Final Data matrix")
print(X_tr_set3.shape)
print(X_te_set3.shape)
print(X_cr_set3.shape)
print("="*100)
```

```
Final Data matrix
(22443, 699)
```

```
(22443, 699)
(16500, 699)
(11055, 699)
==========================================================================
============================
```

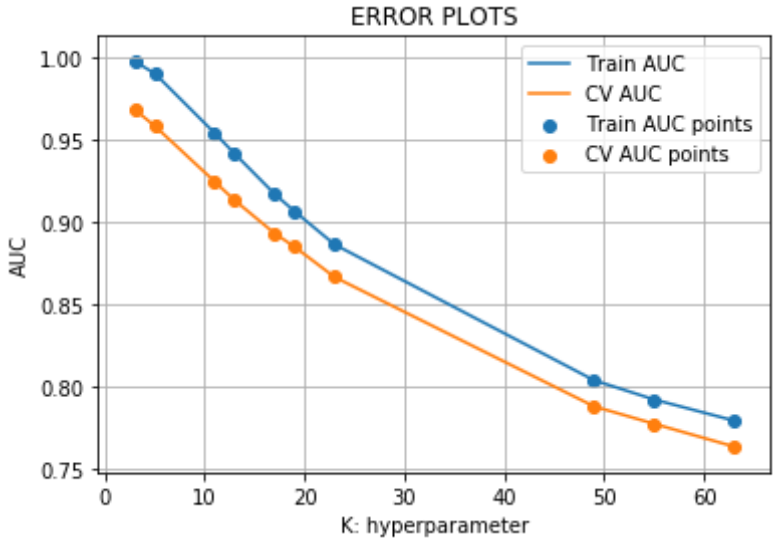
In [120]:
```
y_train.shape,X_tr_set3.shape
```

Out[120]:
```
((22443,), (22443, 699))
```

In [0]:
```
def batch_predict(clf, data):
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probab
ility estimates of the positive class
    # not the predicted outputs
   # print(data.shape)
    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000
    # consider you X_tr shape is 49041, then your tr_loop will be 49041
 - 49041%1000 = 49000
    # in this for loop we will iterate unti the last 1000 multiplier
    #print ('pythoon')
    for i in range(0, tr_loop, 1000):

        y_data_pred.extend(clf.predict_proba(data[i:i+1000])[:,1])

    # we will be predicting for the last data points

    if data.shape[0]%1000 !=0:

        y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])
    return y_data_pred
```

In [122]:
```
#simple cv
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
"""
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 clas
s, 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 wit
h greater label.

"""
train_auc = []
cv_auc = []
K = [3,5,11,13,17,19,23,49,55,63]
for i in tqdm(K):
    neigh = KNeighborsClassifier(n_neighbors=i,algorithm='brute',n_jobs
=-1,)
    neigh.fit(X_tr_set3, y_train)

    y_train_pred = batch_predict(neigh, X_tr_set3)
    y_cv_pred = batch_predict(neigh, X_cr_set3)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probab
ility estimates of the positive 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()
```



In [123]:
```
X_tr_set3.shape,X_te_set3.shape,y_train.shape
```

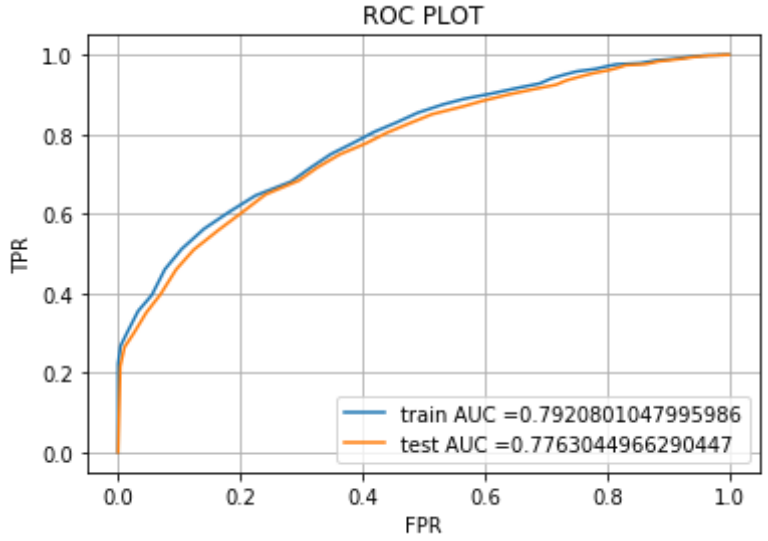
Out[123]:
```
((22443, 699), (16500, 699), (22443,))
```

In [0]:
```
#data['y'].values_counts()
```

In [0]:
```
best_k=55
```

In [125]:
```
# 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, n_jobs=-1)
neigh.fit(X_tr_set3, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probabilit
y estimates of the positive class
# not the predicted outputs

y_train_pred = batch_predict(neigh, X_tr_set3)
```

```
y_test_pred = batch_predict(neigh, X_te_set3)

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

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, t
rain_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_
tpr)))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC PLOT")
plt.grid()
plt.show()
```



In [0]:
```
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def find_best_threshold(threshould, fpr, tpr):
    t = threshould[np.argmax(tpr*(1-fpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is
 very high
```

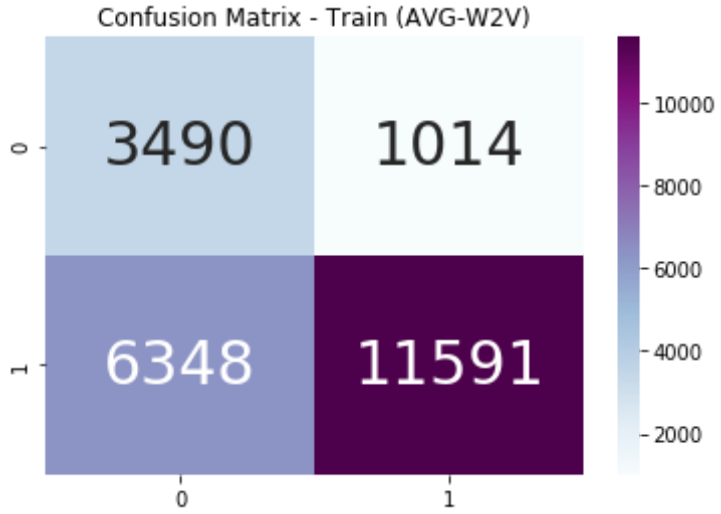
```
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for th
reshold", np.round(t,3))
    return t

def predict_with_best_t(proba, threshould):
    predictions = []
    for i in proba:
        if i>=threshould:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

In [128]:
```
print("="*100)
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_
t)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t
)))
```

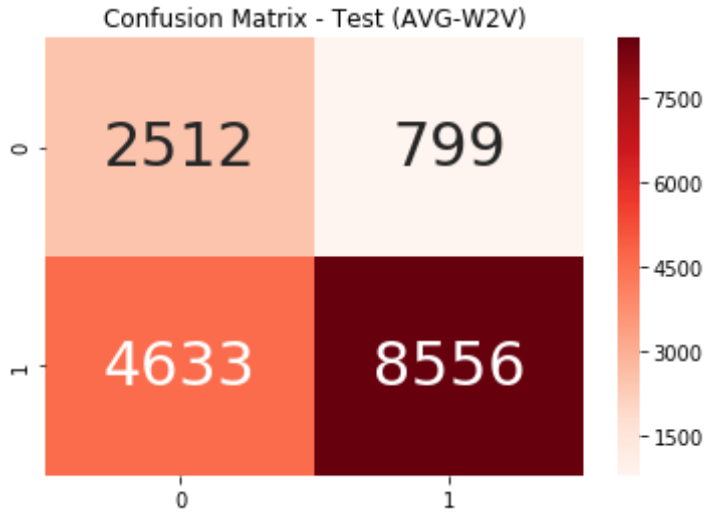
```
====================================================================
============================
the maximum value of tpr*(1-fpr) 0.5006678692157565 for threshold 0.836
Train confusion matrix
[[ 3490  1014]
 [ 6348 11591]]
Test confusion matrix
[[2512  799]
 [4633 8556]]
```

In [129]:
```
ax = plt.axes()
df_cm = confusion_matrix(y_train, predict_with_best_t(y_train_pred, bes
t_t))
sns.heatmap(df_cm, annot=True, annot_kws={"size": 30}, fmt='d',cmap="Bu
Pu", ax = ax )
```

```
ax.set_title('Confusion Matrix - Train (AVG-W2V)')
plt.show()
```



In [130]:
```
ax = plt.axes()
df_cm = confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_
t))
sns.heatmap(df_cm, annot=True, annot_kws={"size": 30}, fmt='d',cmap="Re
ds", ax = ax )

ax.set_title('Confusion Matrix - Test (AVG-W2V)')
plt.show()
```



Using Pretrained Models: TFIDF weighted W2V

set 4

For Preprocessed Essays

In [0]:
```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model_train = TfidfVectorizer()
tfidf_model_train.fit(X_train['preprocessed_essays'].values)
# we are converting a dictionary with word as a key, and the idf as a v
alue
dictionary = dict(zip(tfidf_model_train.get_feature_names(), list(tfidf
_model_train.idf_)))
tfidf_words = set(tfidf_model_train.get_feature_names())
```

In [136]:
```
# average Word2Vec
```

```
# compute average word2vec for each review.
tfidf_w2v_vectors_train_essays= []; # the avg-w2v for each sentence/rev
iew is stored in this list
for sentence in tqdm(X_train['preprocessed_essays'].values):  # for eac
h 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 sentenc
e/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in words_glove) and (word in tfidf_words):
            vec = model_1[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and t
he tf value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentenc
e.split())) # getting the tfidf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_train_essays.append(vector)

print(len(tfidf_w2v_vectors_train_essays))
```

```
22443
```

In [0]:
```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model_test = TfidfVectorizer()
tfidf_model_test.fit(X_test['preprocessed_essays'].values)
# we are converting a dictionary with word as a key, and the idf as a v
alue
dictionary = dict(zip(tfidf_model_test.get_feature_names(), list(tfidf_
model_test.idf_)))
tfidf_words = set(tfidf_model_test.get_feature_names())
```

In [138]:
```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_test_essays = []; # the avg-w2v for each sentence/rev
iew is stored in this list
```

```
for sentence in tqdm(X_test['preprocessed_essays'].values):  # for each
 review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentenc
e/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in words_glove) and (word in tfidf_words):
            vec = model_1[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and t
he tf value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentenc
e.split())) # getting the tfidf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_test_essays.append(vector)

print(len(tfidf_w2v_vectors_test_essays))
```

```
16500
```

In [0]:
```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model_cv = TfidfVectorizer()
tfidf_model_cv.fit(X_cv['preprocessed_essays'].values)
# we are converting a dictionary with word as a key, and the idf as a v
alue
dictionary = dict(zip(tfidf_model_cv.get_feature_names(), list(tfidf_mo
del_cv.idf_)))
tfidf_words = set(tfidf_model_cv.get_feature_names())
```

In [140]:
```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_cv_essays = []; # the avg-w2v for each sentence/revie
w is stored in this list
for sentence in tqdm(X_cv['preprocessed_essays'].values):  # for each r
eview/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 sentenc
e/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in words_glove) and (word in tfidf_words):
            vec = model_1[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and t
he tf value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentenc
e.split())) # getting the tfidf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_cv_essays.append(vector)

print(len(tfidf_w2v_vectors_cv_essays))
```

```
11055
```

for project title

In [0]:
```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model_train = TfidfVectorizer()
tfidf_model_train.fit(X_train['preprocessed_project_title'].values)
# we are converting a dictionary with word as a key, and the idf as a v
alue
dictionary = dict(zip(tfidf_model_train.get_feature_names(), list(tfidf
_model_train.idf_)))
tfidf_words = set(tfidf_model_train.get_feature_names())
```

In [142]:
```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_train_title= []; # the avg-w2v for each sentence/revi
ew is stored in this list
for sentence in tqdm(X_train['preprocessed_project_title'].values):  #
 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 sentenc
e/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in words_glove) and (word in tfidf_words):
            vec = model_1[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and t
he tf value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentenc
e.split())) # getting the tfidf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_train_title.append(vector)

print(len(tfidf_w2v_vectors_train_title))
```

```
22443
```

In [0]:
```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model_test = TfidfVectorizer()
tfidf_model_test.fit(X_test['preprocessed_project_title'].values)
# we are converting a dictionary with word as a key, and the idf as a v
alue
dictionary = dict(zip(tfidf_model_test.get_feature_names(), list(tfidf_
model_test.idf_)))
tfidf_words = set(tfidf_model_test.get_feature_names())
```

In [144]:
```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_test_title= []; # the avg-w2v for each sentence/revie
w is stored in this list
for sentence in tqdm(X_test['preprocessed_project_title'].values):  # f
or 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 sentenc
e/review
```

```
    for word in sentence.split(): # for each word in a review/sentence
        if (word in words_glove) and (word in tfidf_words):
            vec = model_1[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and t
he tf value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentenc
e.split())) # getting the tfidf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_test_title.append(vector)

print(len(tfidf_w2v_vectors_test_title))
```

```
16500
```

In [0]:
```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model_cv = TfidfVectorizer()
tfidf_model_cv.fit(X_cv['preprocessed_project_title'].values)
# we are converting a dictionary with word as a key, and the idf as a v
alue
dictionary = dict(zip(tfidf_model_cv.get_feature_names(), list(tfidf_mo
del_cv.idf_)))
tfidf_words = set(tfidf_model_cv.get_feature_names())
```

In [146]:
```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_cv_title = []; # the avg-w2v for each sentence/review
 is stored in this list
for sentence in tqdm(X_cv['preprocessed_project_title'].values):  # 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 sentenc
e/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in words_glove) and (word in tfidf_words):
            vec = model_1[word] # getting the vector for each word
```

```
            # here we are multiplying idf value(dictionary[word]) and t
he tf value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentenc
e.split())) # getting the tfidf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_cv_title.append(vector)

print(len(tfidf_w2v_vectors_cv_title))
```

```
11055
```

Merging all the above features

In [149]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/40840
39
from scipy.sparse import hstack
X_tr_set4=hstack((tfidf_w2v_vectors_train_essays,tfidf_w2v_vectors_trai
n_title,X_train_grade_ohe,X_train_clean_ohe,X_train_sub_ohe,X_train_pre
fix_ohe,X_train_state_ohe,X_train_price_norm,)).tocsr()

X_te_set4=hstack((tfidf_w2v_vectors_test_essays,tfidf_w2v_vectors_test_
title,X_test_state_ohe,X_test_grade_ohe,X_test_clean_ohe,X_test_sub_ohe
,X_test_prefix_ohe, X_test_price_norm)).tocsr()

X_cr_set4=hstack((tfidf_w2v_vectors_cv_essays,tfidf_w2v_vectors_cv_titl
e,X_cv_state_ohe,X_cv_grade_ohe,X_cv_clean_ohe,X_cv_sub_ohe,X_cv_prefix
_ohe,X_cv_price_norm)).tocsr()

print("Final Data matrix")
print(X_cr_set4.shape)
print(X_tr_set4.shape)
print(X_te_set4.shape)
print("="*100)
```

```
Final Data matrix
```

```
Final Data matrix
(11055, 699)

(22443, 699)
(16500, 699)
================================================================================
==============================
```

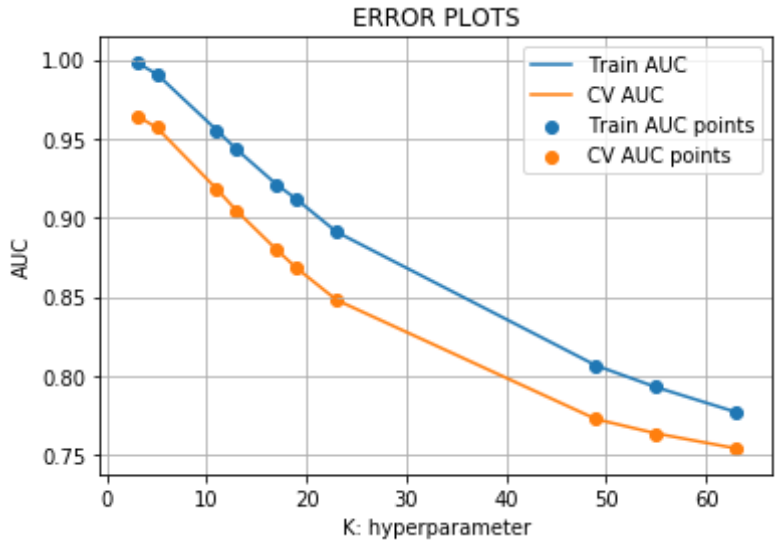
In [0]:

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

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

    return y_data_pred
```

In [151]:

```
train_auc = []
cv_auc = []
K = [3,5,11,13,17,19,23,49,55,63]
for i in tqdm(K):
    neigh = KNeighborsClassifier(n_neighbors=i,algorithm='brute', n_job
s=-1)
    neigh.fit(X_tr_set4, y_train)

    y_train_pred = batch_predict(neigh, X_tr_set4)
    y_cv_pred = batch_predict(neigh, X_cr_set4)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probab
ility estimates of the positive 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()
```

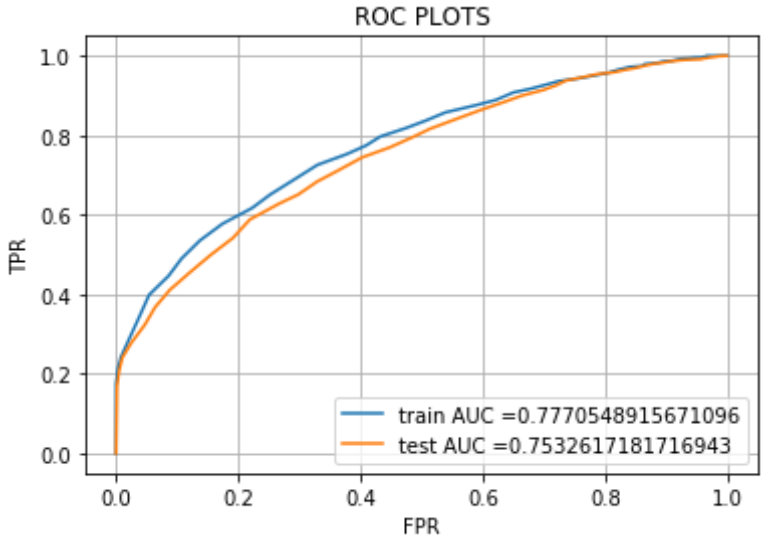


In [0]:
```
# from the error plot we choose K such that, we will have maximum AUC o
n cv data and gap between the train and cv is less
# Note: based on the method you use you might get different hyperparame
ter values as best one
```

```
# so, you choose according to the method you choose, you use gridsearch
 if you are having more computing power and note it will take more time
# if you increase the cv values in the GridSearchCV you will get more r
ebust results.

#here we are choosing the best_k based on forloop results
best_k = 63
```

In [153]:

```
# 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, n_jobs=-1)
neigh.fit(X_tr, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probabilit
y estimates of the positive class
# not the predicted outputs

y_train_pred = batch_predict(neigh, X_tr)
y_test_pred = batch_predict(neigh, X_te)

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

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, t
rain_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_
tpr)))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC PLOTS")
plt.grid()
plt.show()
```

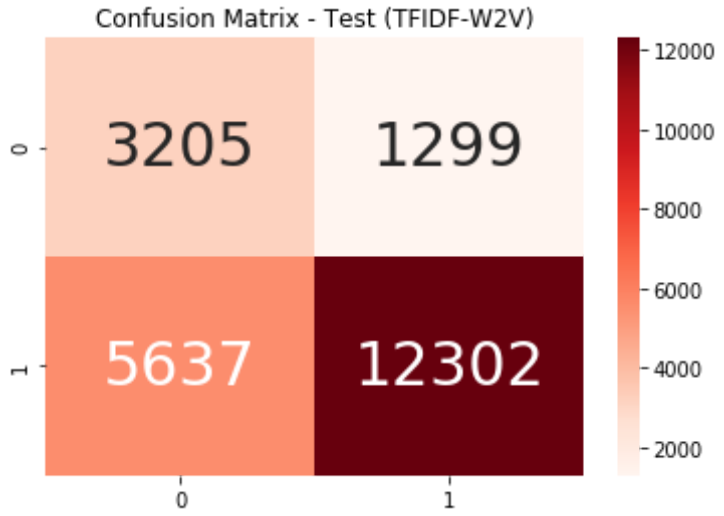


In [0]:
```
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def find_best_threshold(threshould, fpr, tpr):
    t = threshould[np.argmax(tpr*(1-fpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is
 very high
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for th
reshold", np.round(t,3))
    return t

def predict_with_best_t(proba, threshould):
    predictions = []
    for i in proba:
        if i>=threshould:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

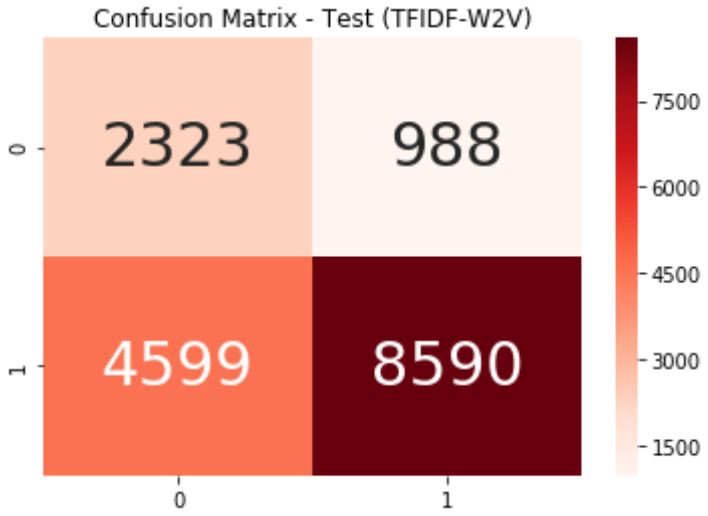
In [155]:
```
print("="*100)
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_
t)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t
)))
```

```
====================================================================
==============================
the maximum value of tpr*(1-fpr) 0.48798575535783045 for threshold 0.81
Train confusion matrix
[[ 3205  1299]
 [ 5637 12302]]
Test confusion matrix
[[2323  988]
 [4599 8590]]
```

In [156]:
```
ax = plt.axes()
df_cm = confusion_matrix(y_train, predict_with_best_t(y_train_pred, bes
t_t))
sns.heatmap(df_cm, annot=True, annot_kws={"size": 30}, fmt='d',cmap="Re
ds", ax = ax )

ax.set_title('Confusion Matrix - Test (TFIDF-W2V)')
plt.show()
```



In [157]:
```
ax = plt.axes()
df_cm = confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_
t))
sns.heatmap(df_cm, annot=True, annot_kws={"size": 30}, fmt='d',cmap="Re
ds", ax = ax )

ax.set_title('Confusion Matrix - Test (TFIDF-W2V)')
plt.show()
```



Task 2

In [158]:
```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
X_tr_task2=hstack((X_train_state_ohe,X_train_grade_ohe,X_train_clean_ohe,X_train_sub_ohe,X_train_prefix_ohe,X_train_price_norm,X_train_title_tfidf,X_train_essays_tfidf)).tocsr()

X_te_task2=hstack((X_test_state_ohe,X_test_grade_ohe,X_test_clean_ohe,X_test_sub_ohe,X_test_prefix_ohe, X_test_price_norm,X_test_title_tfidf,X_test_essays_tfidf)).tocsr()

X_cr_task2=hstack((X_cv_state_ohe,X_cv_grade_ohe,X_cv_clean_ohe,X_cv_sub_ohe,X_cv_prefix_ohe,X_cv_price_norm,X_cv_essays_tfidf,X_cv_title_tfidf)).tocsr()

print("Final Data matrix")
print(X_tr_task2.shape)
print(X_te_task2.shape)
```

```
print(X_cr_task2.shape)
print("="*100)
```

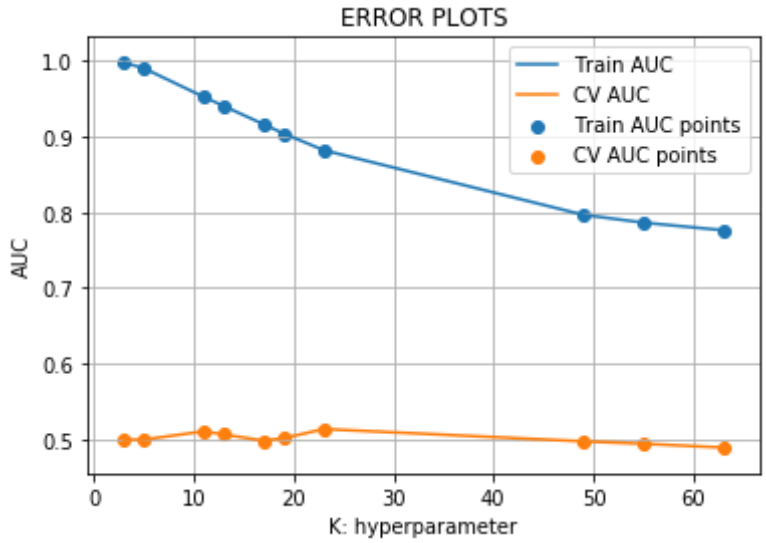
```
Final Data matrix

(22443, 6481)
(16500, 6481)
(11055, 6481)
====================================================================================
============================
```

In [160]:
```
#from sklearn.datasets import load_digits
from sklearn.feature_selection import SelectKBest, chi2
 #X, y = load_digits(return_X_y=True)
  #X.shape
X_tr_new = SelectKBest(chi2, k=2000).fit_transform(X_tr_task2,y_train)
X_te_new = SelectKBest(chi2, k=2000).fit_transform(X_te_task2,y_test)
X_cr_new = SelectKBest(chi2, k=2000).fit_transform(X_cr_task2,y_cv)
X_tr_new.shape,X_te_new.shape,X_cr_new.shape
```

Out[160]:
```
((22443, 2000), (16500, 2000), (11055, 2000))
```

In [161]:
```
train_auc = []
cv_auc = []
K = [3,5,11,13,17,19,23,49,55,63]
for i in tqdm(K):
    neigh = KNeighborsClassifier(n_neighbors=i, n_jobs=-1)
    neigh.fit(X_tr_new, y_train)

    y_train_pred = batch_predict(neigh, X_tr_new)
    y_cv_pred = batch_predict(neigh, X_cr_new)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probab
ility estimates of the positive 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')
```

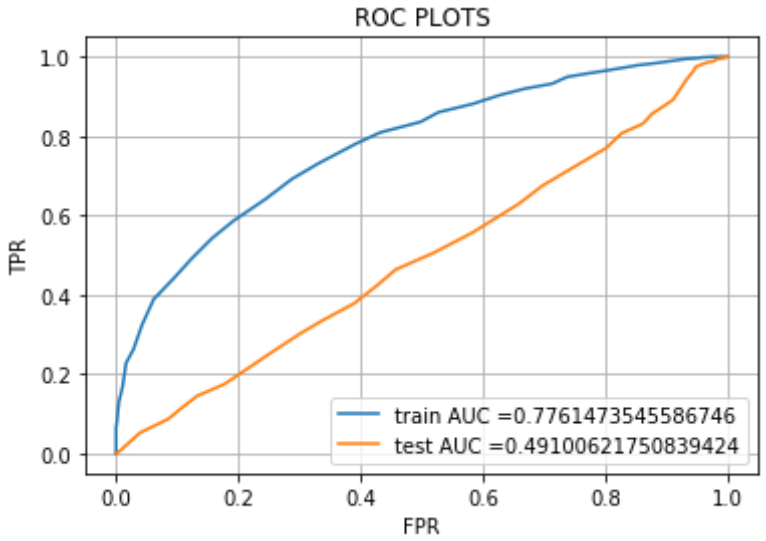
```
plt.plot(K, cv_auc, label='CV AUC')

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()
```



In [0]:
```
besk_k=55
```

In [165]:
```
# 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, n_jobs=-1)
```

```
neigh.fit(X_tr_new, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probabilit
y estimates of the positive class
# not the predicted outputs

y_train_pred = batch_predict(neigh, X_tr_new)
y_test_pred = batch_predict(neigh, X_te_new)

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

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, t
rain_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_
tpr)))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC PLOTS")
plt.grid()
plt.show()
```

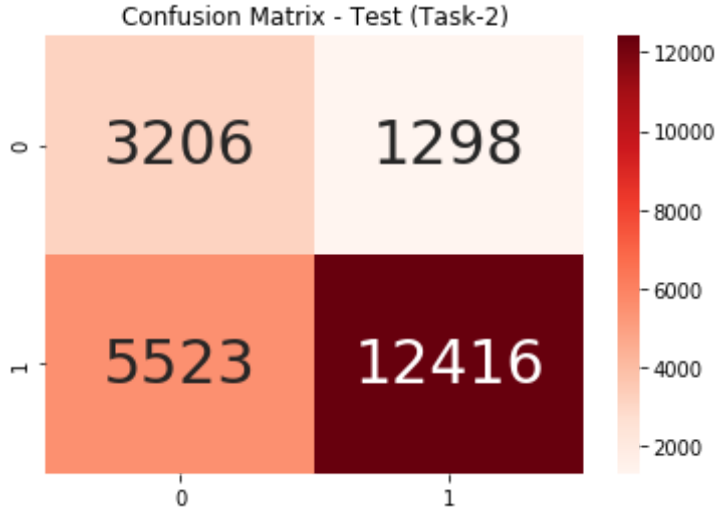


In [0]:
```
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def find_best_threshold(threshould, fpr, tpr):
    t = threshould[np.argmax(tpr*(1-fpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is
 very high
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for th
reshold", np.round(t,3))
    return t

def predict_with_best_t(proba, threshould):
    predictions = []
    for i in proba:
        if i>=threshould:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

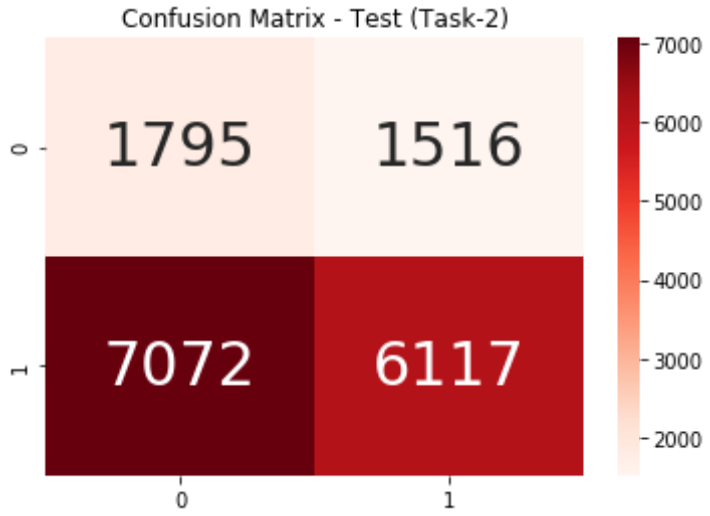
In [167]:
```
print("="*100)
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_
t)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t
)))
```

```
====================================================================
============================
the maximum value of tpr*(1-fpr) 0.4926614834543391 for threshold 0.794
Train confusion matrix
[[ 3206  1298]
 [ 5523 12416]]
Test confusion matrix
[[1795 1516]
 [7072 6117]]
```

In [168]:
```
ax = plt.axes()
df_cm = confusion_matrix(y_train, predict_with_best_t(y_train_pred, bes
t_t))
sns.heatmap(df_cm, annot=True, annot_kws={"size": 30}, fmt='d',cmap="Re
ds", ax = ax )

ax.set_title('Confusion Matrix - Test (Task-2)')
plt.show()
```



In [169]:
```
ax = plt.axes()
df_cm = confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_
t))
sns.heatmap(df_cm, annot=True, annot_kws={"size": 30}, fmt='d',cmap="Re
ds", ax = ax )

ax.set_title('Confusion Matrix - Test (Task-2)')
plt.show()
```



preety Table

In [172]:
```
from prettytable import PrettyTable

x = PrettyTable()
x.field_names = ["Vectorizer","model", "Hyperparameter(k)","Train AUC",
"Test AUC"]
x.add_row(["set 1","brute", 97,0.75, 0.73])
x.add_row(["set 2","brute",99,0.79, 0.76])
x.add_row(["set 3","brute", 55,0.79,0.77])
x.add_row(["set 4","brute", 63,0.77,0.75])
x.add_row(["task 2","brute",55,0.77,0.49])

print(x)
```

```
+------------+-------+------------------+-----------+----------+
| Vectorizer | model | Hyperparameter(k) | Train AUC | Test AUC |
+------------+-------+------------------+-----------+----------+
|   set 1    | brute |         97        |    0.75   |   0.73   |
```

```
|   set 2    | brute |         99         |    0.79    |   0.76   |
|   set 3    | brute |         55         |    0.79    |   0.77   |
|   set 4    | brute |         63         |    0.77    |   0.75   |
|   task 2   | brute |         55         |    0.77    |   0.49   |
+------------+-------+--------------------+------------+----------+
```

In [0]: