In [1]:

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

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

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

import re
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

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

1.1 Reading Data

In [2]:

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

Out[2]:

```
"project_data = pd.read_csv('train_data.csv')\nresource_data =
pd.read_csv('resources.csv')"
```

In [3]:

```
'''print("Number of data points in project data", project_data.shape)
print("\nThe attributes of project data :\n\n", project_data.columns.values)
```

Out[3]:

```
'print("Number of data points in project data", project_data.s
hape)\nprint("\nThe attributes of project data :\n\n", project
_data.columns.values)'
```

In [4]:

```
'''# how to replace elements in list python: https://stackoverflow.com/a/2582
cols = ['Date' if x=='project_submitted_datetime' else x for x in list(projec


#sort dataframe based on time pandas python: https://stackoverflow.com/a/4970
project_data['Date'] = pd.to_datetime(project_data['project_submitted_datetim
project_data.drop('project_submitted_datetime', axis=1, inplace=True)
project_data.sort_values(by=['Date'], inplace=True)


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


project_data.head(5)'''
```

Out[4]:

```
"# how to replace elements in list python: https://stackoverfl
ow.com/a/2582163/4084039\ncols (https://stackoverflow.com/a/25
82163/4084039\ncols) = ['Date' if x=='project_submitted_dateti
me' else x for x in list(project_data.columns)]\n\n\n#sort dat
aframe based on time pandas python: https://stackoverflow.com/
a/49702492/4084039\nproject_data['Date'] (https://stackoverflo
w.com/a/49702492/4084039\nproject_data['Date']) = pd.to_dateti
me(project_data['project_submitted_datetime'])\nproject_data.d
rop('project_submitted_datetime', axis=1, inplace=True)\nproje
ct_data.sort_values(by=['Date'], inplace=True)\n\n\n# how to r
eorder columns pandas python: https://stackoverflow.com/a/1314
8611/4084039\nproject_data (https://stackoverflow.com/a/131486
11/4084039\nproject_data) = project_data[cols]\n\n\nproject_da
ta.head(5)"
```

In [5]:

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

Out[5]:

```
'print("Number of data points in resource data", resource_dat
a.shape)\nprint(resource_data.columns.values)\nresource_data.h
ead(2)'
```

1.2 preprocessing of project_subject_categories

In [6]:

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

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

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

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

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

Out[6]:

```
'catogories = list(project_data[\'project_subject_categorie
s\'].values)\n# remove special characters from list of stri
ngs python: https://stackoverflow.com/a/47301924/4084039\n
\n# (https://stackoverflow.com/a/47301924/4084039\n\n#) htt
ps://www.geeksforgeeks.org/removing-stop-words-nltk-python/
\n# (https://www.geeksforgeeks.org/removing-stop-words-nltk
-python/\n#) https://stackoverflow.com/questions/23669024/h
ow-to-strip-a-specific-word-from-a-string\n# (https://stack
overflow.com/questions/23669024/how-to-strip-a-specific-wor
d-from-a-string\n#) https://stackoverflow.com/questions/827
0092/remove-all-whitespace-in-a-string-in-python\ncat_list
 (https://stackoverflow.com/questions/8270092/remove-all-wh
itespace-in-a-string-in-python\ncat_list) = []\nfor i in ca
togories:\n    temp = ""\n    # consider we have text like
 this "Math & Science, Warmth, Care & Hunger"\n    for j in
i.split(\',\'): # it will split it in three parts ["Math &
 Science", "Warmth", "Care & Hunger"]\n        if \'The\' i
n j.split(): # this will split each of the catogory based o
```

```
n space "Math & Science"=> "Math","&", "Science"\n
j=j.replace(\'The\',\'\') # if we have the words "The" we a
re going to replace it with \'\'(i.e removing \'The\')\n
j = j.replace(\' \',\'\') # we are placeing all the \' \'(s
pace) with \'\'(empty) ex:"Math & Science"=>"Math&Scienc
e"\n        temp+=j.strip()+" " #" abc ".strip() will retur
n "abc", remove the trailing spaces\n        temp = temp.re
place(\'&\',\'_\') # we are replacing the & value into \n
cat_list.append(temp.strip())\n    \nproject_data[\'clean_c
ategories\'] = cat_list\nproject_data.drop([\'project_subje
ct_categories\'], axis=1, inplace=True)\n\nfrom collections
import Counter\nmy_counter = Counter()\nfor word in project
_data[\'clean_categories\'].values:\n    my_counter.update
(word.split())\n\ncat_dict = dict(my_counter)\nsorted_cat_d
ict = dict(sorted(cat_dict.items(), key=lambda kv: kv[1]))
\nprint(sorted_cat_dict)'
```

1.3 preprocessing of project_subject_subcategories

In [7]:

```
'''sub_catogories = list(project_data['project_subject_subcategories'].values
# remove special characters from list of strings python: https://stackoverflo

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

sub_cat_list = []
for i in sub_catogories:
    temp = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunger'
    for j in i.split(','): # it will split it in three parts ["Math & Science
        if 'The' in j.split(): # this will split each of the catogory based o
            j=j.replace('The','') # if we have the words "The" we are going t
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(em
        temp +=j.strip()+" "#" abc ".strip() will return "abc", remove the tr
        temp = temp.replace('&','_')
    sub_cat_list.append(temp.strip())

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

# count of all the words in corpus python: https://stackoverflow.com/a/228985
my_counter = Counter()
for word in project_data['clean_subcategories'].values:
    my_counter.update(word.split())

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

Out[7]:

```
'sub_catogories = list(project_data[\'project_subject_subca
tegories\'].values)\n# remove special characters from list
of strings python: https://stackoverflow.com/a/47301924/408
4039\n\n# (https://stackoverflow.com/a/47301924/4084039\n\n
#) https://www.geeksforgeeks.org/removing-stop-words-nltk-p
ython/\n# (https://www.geeksforgeeks.org/removing-stop-word
s-nltk-python/\n#) https://stackoverflow.com/questions/2366
9024/how-to-strip-a-specific-word-from-a-string\n# (http
s://stackoverflow.com/questions/23669024/how-to-strip-a-spe
cific-word-from-a-string\n#) https://stackoverflow.com/ques
tions/8270092/remove-all-whitespace-in-a-string-in-python\n
\nsub_cat_list (https://stackoverflow.com/questions/827009
2/remove-all-whitespace-in-a-string-in-python\n\nsub_cat_li
st) = []\nfor i in sub_catogories:\n    temp = ""\n    # co
nsider we have text like this "Math & Science, Warmth, Care
& Hunger"\n    for j in i.split(\',\'): # it will split it
 in three parts ["Math & Science", "Warmth", "Care & Hunge
r"]\n        if \'The\' in j.split(): # this will split eac
```

```
h of the catogory based on space "Math & Science"=> "Mat
h","&", "Science"\n            j=j.replace(\'The\',\'\') #
 if we have the words "The" we are going to replace it with
\'\'(i.e removing \'The\')\n        j = j.replace(\'
 \',\'\') # we are placeing all the \' \'(space) with
 \'\'(empty) ex:"Math & Science"=>"Math&Science"\n        t
emp +=j.strip()+" "#" abc ".strip() will return "abc", remo
ve the trailing spaces\n        temp = temp.replace(\'&
\',\'_\')\n    sub_cat_list.append(temp.strip())\n\nproject
_data[\'clean_subcategories\'] = sub_cat_list\nproject_dat
a.drop([\'project_subject_subcategories\'], axis=1, inplace
=True)\n\n# count of all the words in corpus python: http
s://stackoverflow.com/a/22898595/4084039\nmy_counter (http
s://stackoverflow.com/a/22898595/4084039\nmy_counter) = Cou
nter()\nfor word in project_data[\'clean_subcategories\'].v
alues:\n    my_counter.update(word.split())\n    \nsub_cat_
dict = dict(my_counter)\nsorted_sub_cat_dict = dict(sorted
(sub_cat_dict.items(), key=lambda kv: kv[1]))'
```

1.3 preprocessing teacher_prefix

In [8]:

```
'''project_data['teacher_prefix'] = project_data['teacher_prefix'].replace(np
project_data['teacher_prefix'] = project_data['teacher_prefix'].replace('Dr.
project_data['teacher_prefix'] = project_data['teacher_prefix'].replace('Teac
project_data['teacher_prefix'] = project_data['teacher_prefix'].replace('Mr.
project_data['teacher_prefix'] = project_data['teacher_prefix'].replace('Ms.
project_data['teacher_prefix'] = project_data['teacher_prefix'].replace('Mrs.
print(project_data['teacher_prefix'].head(5))
project_data.head(2)
'''
```

Out[8]:

```
"project_data['teacher_prefix'] = project_data['teacher_prefi
x'].replace(np.nan,'MRS')\nproject_data['teacher_prefix'] = pr
oject_data['teacher_prefix'].replace('Dr.','DR')\nproject_data
['teacher_prefix'] = project_data['teacher_prefix'].replace('T
eacher','TEACHER')\nproject_data['teacher_prefix'] = project_d
ata['teacher_prefix'].replace('Mr.','MR')\nproject_data['teach
er_prefix'] = project_data['teacher_prefix'].replace('Ms.','M
S')\nproject_data['teacher_prefix'] = project_data['teacher_pr
efix'].replace('Mrs.','MRS')\nprint(project_data['teacher_pref
ix'].head(5))\nproject_data.head(2)\n"
```

1.3 preprocessing project_grade_category

In [9]:

```
'''project_data['project_grade_category'] = project_data['project_grade_categ
project_data['project_grade_category'] = project_data['project_grade_category
project_data['project_grade_category'] = project_data['project_grade_category
project_data['project_grade_category'] = project_data['project_grade_category
print(project_data['project_grade_category'].head(5))
project_data.head(2)
'''
```

Out[9]:

```
"project_data['project_grade_category'] = project_data['projec
t_grade_category'].replace('Grades PreK-2', 'Grades_PreK_2')\n
project_data['project_grade_category'] = project_data['project
_grade_category'].replace('Grades 3-5', 'Grades_3_5')\nproject
_data['project_grade_category'] = project_data['project_grade_
category'].replace('Grades 6-8', 'Grades_6_8')\nproject_data
['project_grade_category'] = project_data['project_grade_categ
ory'].replace('Grades 9-12', 'Grades_9_12')\nprint(project_dat
a['project_grade_category'].head(5))\nproject_data.head(2)\n"
```

1.3 Text preprocessing

In [10]:

```
'''# merge two column text dataframe:
project_data["essay"] = project_data["project_essay_1"].map(str) +\
                        project_data["project_essay_2"].map(str) + \
                        project_data["project_essay_3"].map(str) + \
                        project_data["project_essay_4"].map(str)
project_data.head(2)'''
```

Out[10]:

```
'# merge two column text dataframe: \nproject_data["essay"] =
project_data["project_essay_1"].map(str) +
project_data["project_essay_2"].map(str) +
project_data["project_essay_3"].map(str) +
project_data["project_essay_4"].map(str)\nproject_data.head
(2)'
```

Preprocessing essays

In [11]:

```
'''# printing some random reviews
print(project_data['essay'].values[0])
print("\n\n")
print(project_data['essay'].values[150])
print("\n\n")
print(project_data['essay'].values[1000])
print("\n\n")
print(project_data['essay'].values[20000])
print("\n\n")
print(project_data['essay'].values[99999])'''
```

Out[11]:

```
'# printing some random reviews\nprint(project_data[\'essay
\'].values[0])\nprint("\n\n")\nprint(project_data[\'essay\'].v
alues[150])\nprint("\n\n")\nprint(project_data[\'essay\'].valu
es[1000])\nprint("\n\n")\nprint(project_data[\'essay\'].values
[20000])\nprint("\n\n")\nprint(project_data[\'essay\'].values
[99999])'
```

In [12]:

```
'''# 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'''
```

Out[12]:

```
'# https://stackoverflow.com/a/47091490/4084039\nimport (http
s://stackoverflow.com/a/47091490/4084039\nimport) re\n\ndef de
contracted(phrase):\n    # specific\n    phrase = re.sub(r"won
\'t", "will not", phrase)\n    phrase = re.sub(r"can\'t", "can
not", phrase)\n\n    # general\n    phrase = re.sub(r"n\'t", "
not", phrase)\n    phrase = re.sub(r"\'re", " are", phrase)\n
phrase = re.sub(r"\'s", " is", phrase)\n    phrase = re.sub
(r"\'d", " would", phrase)\n    phrase = re.sub(r"\'ll", " wil
l", phrase)\n    phrase = re.sub(r"\'t", " not", phrase)\n
 phrase = re.sub(r"\'ve", " have", phrase)\n    phrase = re.su
b(r"\'m", " am", phrase)\n    return phrase'
```

In [13]:

```
'''sent = decontracted(project_data['essay'].values[20000])
print(sent)'''
```

Out[13]:

```
"sent = decontracted(project_data['essay'].values[20000])\npri
nt(sent)"
```

In [14]:

```
'''# \r \n \t remove from string python: http://texthandler.com/info/remove-l
sent = sent.replace('\\r', ' ')
sent = sent.replace('\\"', ' ')
sent = sent.replace('\\n', ' ')
print(sent)'''
```

Out[14]:

```
'# \r \n \t remove from string python: http://texthandler.com/
info/remove-line-breaks-python/\nsent (http://texthandler.com/
info/remove-line-breaks-python/\nsent) = sent.replace(\'\\r\',
\' \')\nsent = sent.replace(\'\\"\', \' \')\nsent = sent.repla
ce(\'\\n\', \' \')\nprint(sent)'
```

In [15]:

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

Out[15]:

```
"#remove spacial character: https://stackoverflow.com/a/584354
7/4084039\nsent (https://stackoverflow.com/a/5843547/4084039\n
sent) = re.sub('[^A-Za-z0-9]+', ' ', sent)\nprint(sent)"
```

In [16]:

```
'''
# we are removing the words from the stop words list: 'no', 'nor', 'not'
stopwords= ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'i
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this',
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have',
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'beca
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into',
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on',
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how',
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'th
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "shoul
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn',
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't",
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shou
            'won', "won't", 'wouldn', "wouldn't"]'''
```

Out[16]:

```
'\n# we are removing the words from the stop words list:
\'no\', \'nor\', \'not\'\nstopwords= [\'i\', \'me\', \'my
\', \'myself\', \'we\', \'our\', \'ours\', \'ourselves\',
\'you\', "you\'re", "you\'ve",            "you\'ll", "you
\'d", \'your\', \'yours\', \'yourself\', \'yourselves\',
\'he\', \'him\', \'his\', \'himself\',             \'she\',
"she\'s", \'her\', \'hers\', \'herself\', \'it\', "it\'s",
\'its\', \'itself\', \'they\', \'them\', \'their\',
\'theirs\', \'themselves\', \'what\', \'which\', \'who\',
\'whom\', \'this\', \'that\', "that\'ll", \'these\', \'thos
e\',             \'am\', \'is\', \'are\', \'was\', \'were
\', \'be\', \'been\', \'being\', \'have\', \'has\', \'had
\', \'having\', \'do\', \'does\',             \'did\', \'do
ing\', \'a\', \'an\', \'the\', \'and\', \'but\', \'if\',
\'or\', \'because\', \'as\', \'until\', \'while\', \'of\',
\'at\', \'by\', \'for\', \'with\', \'about\', \'against\',
\'between\', \'into\', \'through\', \'during\', \'before\',
\'after\',           \'above\', \'below\', \'to\', \'from
\', \'up\', \'down\', \'in\', \'out\', \'on\', \'off\', \'o
ver\', \'under\', \'again\', \'further\',           \'then
\', \'once\', \'here\', \'there\', \'when\', \'where\', \'w
hy\', \'how\', \'all\', \'any\', \'both\', \'each\', \'few
\', \'more\',            \'most\', \'other\', \'some\', \'s
uch\', \'only\', \'own\', \'same\', \'so\', \'than\', \'too
\', \'very\',             \'s\', \'t\', \'can\', \'will\',
\'just\', \'don\', "don\'t", \'should\', "should\'ve", \'no
w\', \'d\', \'ll\', \'m\', \'o\', \'re\',             \'ve
\', \'y\', \'ain\', \'aren\', "aren\'t", \'couldn\', "could
n\'t", \'didn\', "didn\'t", \'doesn\', "doesn\'t", \'hadn
\',           "hadn\'t", \'hasn\', "hasn\'t", \'haven\',
```

```
"haven\'t", \'isn\', "isn\'t", \'ma\', \'mightn\', "mightn
\'t", \'mustn\',            "mustn\'t", \'needn\', "needn
\'t", \'shan\', "shan\'t", \'shouldn\', "shouldn\'t", \'was
n\', "wasn\'t", \'weren\', "weren\'t",             \'won\',
"won\'t", \'wouldn\', "wouldn\'t"]'
```

In [17]:

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

Out[17]:

```
'# Combining all the above stundents \nfrom tqdm import tqdm\n
preprocessed_essays = []\n# tqdm is for printing the status ba
r\nfor sentance in tqdm(project_data[\'essay\'].values):\n
sent = decontracted(sentance)\n    sent = sent.replace(\'\\r
\', \' \')\n    sent = sent.replace(\'\\"\', \' \')\n    sent
= sent.replace(\'\\n\', \' \')\n    sent = re.sub(\'[^A-Za-z0-
9]+\', \' \', sent)\n    sent = \' \'.join(e for e in sent.spl
it() if e.lower() not in stopwords)\n    preprocessed_essays.a
ppend(sent.lower().strip())'
```

In [18]:

```
'''# after preprocesing
preprocessed_essays[1000]'''
```

Out[18]:

```
'# after preprocesing\npreprocessed_essays[1000]'
```

In [19]:

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

Out[19]:

```
'# Combining all the above stundents \nfrom tqdm import tqdm\n
preprocessed_titles = []\n# tqdm is for printing the status ba
r\nfor sentance in tqdm(project_data[\'project_title\'].value
s):\n    sent = decontracted(sentance)\n    sent = sent.replac
e(\'\\r\', \' \')\n    sent = sent.replace(\'\\"\', \' \')\n
sent = sent.replace(\'\\n\', \' \')\n    sent = re.sub(\'[^A-Z
a-z0-9]+\', \' \', sent)\n    sent = \' \'.join(e for e in sen
t.split() if e.lower() not in stopwords)\n    preprocessed_tit
les.append(sent.lower().strip())'
```

In [20]:

```
'''# after preprocesing
preprocessed_titles[1000]'''
```

Out[20]:

```
'# after preprocesing\npreprocessed_titles[1000]'
```

In [21]:

```
'''# Combining all the above stundents
from tqdm import tqdm
preprocessed_resource_summary = []
# tqdm is for printing the status bar
for sentance in tqdm(project_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(e for e in sent.split() if e.lower() not in stopwords)
    preprocessed_resource_summary.append(sent.lower().strip())'''
```

Out[21]:

```
'# Combining all the above stundents \nfrom tqdm import tqdm\n
preprocessed_resource_summary = []\n# tqdm is for printing the
status bar\nfor sentance in tqdm(project_data[\'project_resour
ce_summary\'].values):\n    sent = decontracted(sentance)\n
sent = sent.replace(\'\\r\', \' \')\n    sent = sent.replace
(\'\\"\', \' \')\n    sent = sent.replace(\'\\n\', \' \')\n
sent = re.sub(\'[^A-Za-z0-9]+\', \' \', sent)\n    sent = \'
\'.join(e for e in sent.split() if e.lower() not in stopwords)
\n    preprocessed_resource_summary.append(sent.lower().strip
())'
```

In [22]:

```
'''# after preprocesing
preprocessed_resource_summary[1000]'''
```

Out[22]:

```
'# after preprocesing\npreprocessed_resource_summary[1000]'
```

In [23]:

```
'''price_data = resource_data.groupby('id').agg({'price':'sum', 'quantity':'s
project_data = pd.merge(project_data, price_data, on='id', how='left')'''
```

Out[23]:

```
"price_data = resource_data.groupby('id').agg({'price':'sum',
'quantity':'sum'}).reset_index()\nproject_data = pd.merge(proj
ect_data, price_data, on='id', how='left')"
```

Preprocessed data before splitting

In [24]:

```
'''project_data.head()'''
```

Out[24]:

```
'project_data.head()'
```

Saving preprocessed data to csv & reading data

In [25]:

```
'''project_data.to_csv('preprocessed_data.csv')'''
```

Out[25]:

```
"project_data.to_csv('preprocessed_data.csv')"
```

Splitting data into Train, Cross Validation and Test

In [26]:

```
prepeocessed_data = pd.read_csv('preprocessed_data.csv', nrows=50000)
prepeocessed_data.head(2)
```

Out[26]:

	Unnamed: 0	Unnamed: 0.1	id	teacher_id	teacher
0	0	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	
1	1	37728	p043609	3f60494c61921b3b43ab61bdde2904df	

2 rows × 21 columns

In [27]:

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

Out[27]:

```
(50000, 20)
```

In [28]:

```
# 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, str
X_test.shape
```

Out[28]:

```
(16500, 20)
```

1.4 Encoding Categorical and Numerical features

1.4.1 encoding categorical features: clean_categories

In [29]:

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

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

print("After vectorizations")
print(X_train_cc_ohe.shape, y_train.shape)
#print(X_cv_cc_ohe.shape, y_cv.shape)
print(X_test_cc_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
```

```
After vectorizations
(33500, 9) (33500,)
(16500, 9) (16500,)
['appliedlearning', 'care_hunger', 'health_sports', 'history_c
ivics', 'literacy_language', 'math_science', 'music_arts', 'sp
ecialneeds', 'warmth']
```

1.4.2 encoding categorical features: clean_subcategories

In [30]:

```
vectorizer = CountVectorizer()
vectorizer.fit(X_train['clean_subcategories'].values) # fit has to happen onl

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

print("After vectorizations")
print(X_train_csc_ohe.shape, y_train.shape)
#print(X_cv_csc_ohe.shape, y_cv.shape)
print(X_test_csc_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
```

```
After vectorizations
(33500, 30) (33500,)
(16500, 30) (16500,)
['appliedsciences', 'care_hunger', 'charactereducation', 'civi
cs_government', 'college_careerprep', 'communityservice', 'ear
lydevelopment', 'economics', 'environmentalscience', 'esl', 'e
xtracurricular', 'financialliteracy', 'foreignlanguages', 'gym
_fitness', 'health_lifescience', 'health_wellness', 'history_g
eography', 'literacy', 'literature_writing', 'mathematics', 'm
usic', 'nutritioneducation', 'other', 'parentinvolvement', 'pe
rformingarts', 'socialsciences', 'specialneeds', 'teamsports',
'visualarts', 'warmth']
```

1.4.3 encoding categorical features: school_state

In [31]:

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

# 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())
```

```
After vectorizations
(33500, 51) (33500,)
(16500, 51) (16500,)
['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl',
'ga', 'hi', 'ia', 'id', 'il', 'in', 'ks', 'ky', 'la', 'ma', 'm
d', 'me', 'mi', 'mn', 'mo', 'ms', 'mt', 'nc', 'nd', 'ne', 'n
h', 'nj', 'nm', 'nv', 'ny', 'oh', 'ok', 'or', 'pa', 'ri', 's
c', 'sd', 'tn', 'tx', 'ut', 'va', 'vt', 'wa', 'wi', 'wv', 'w
y']
```

1.4.4 encoding categorical features: teacher_prefix

In [32]:

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

# we use the fitted CountVectorizer to convert the text to vector
X_train_teacher_ohe = vectorizer.transform(X_train['teacher_prefix'].values)
#X_cv_teacher_ohe = vectorizer.transform(X_cv['teacher_prefix'].values)
X_test_teacher_ohe = vectorizer.transform(X_test['teacher_prefix'].values)

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

```
After vectorizations
(33500, 5) (33500,)
(16500, 5) (16500,)
['dr', 'mr', 'mrs', 'ms', 'teacher']
```

1.4.5 encoding categorical features: project_grade_category

In [33]:

```
vectorizer = CountVectorizer()
vectorizer.fit(X_train['project_grade_category'].values) # fit has to happen

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

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())
```

```
After vectorizations
(33500, 4) (33500,)
(16500, 4) (16500,)
['grades_3_5', 'grades_6_8', 'grades_9_12', 'grades_prek_2']
```

1.4.6 encoding numerical features: price

In [34]:

```
'''
# standardization sklearn: https://scikit-learn.org/stable/modules/generated/
from sklearn.preprocessing import StandardScaler

# price_standardized = standardScalar.fit(project_data['price'].values)
# this will rise the error
# ValueError: Expected 2D array, got 1D array instead: array=[725.05 213.03 3
# Reshape your data either using array.reshape(-1, 1)

price_scalar = StandardScaler()
price_scalar.fit(X_train['price'].values.reshape(-1,1))

# Now standardize the data with above maen and variance.
X_train_price_std = price_scalar.transform(X_train['price'].values.reshape(-1
#X_cv_price_std = price_scalar.transform(X_cv['price'].values.reshape(-1, 1))
X_test_price_std = price_scalar.transform(X_test['price'].values.reshape(-1,

print("After vectorizations")
print(X_train_price_std.shape, y_train.shape)
#print(X_cv_price_std.shape, y_cv.shape)
print(X_test_price_std.shape, y_test.shape)

After vectorizations
(335, 1) (335,)
(165, 1) (165,)
'''

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.reshape(-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)
```

```
After vectorizations
(33500, 1) (33500,)
(16500, 1) (16500,)
```

1.4.7 encoding numerical features: teacher_number_of_previously_posted_projects

In [35]:

```
'''
# standardization sklearn: https://scikit-learn.org/stable/modules/generated/
from sklearn.preprocessing import StandardScaler

# price_standardized = standardScalar.fit(project_data['price'].values)
# this will rise the error
# ValueError: Expected 2D array, got 1D array instead: array=[725.05 213.03 3
# Reshape your data either using array.reshape(-1, 1)

ppp_scalar = StandardScaler()
ppp_scalar.fit(X_train['teacher_number_of_previously_posted_projects'].values

# Now standardize the data with above maen and variance.
X_train_ppp_std = price_scalar.transform(X_train['teacher_number_of_previousl
#X_cv_ppp_std = price_scalar.transform(X_cv['teacher_number_of_previously_pos
X_test_ppp_std = price_scalar.transform(X_test['teacher_number_of_previously_

print("After vectorizations")
print(X_train_ppp_std.shape, y_train.shape)
#print(X_cv_ppp_std.shape, y_cv.shape)
print(X_test_ppp_std.shape, y_test.shape)

After vectorizations
(335, 1) (335,)
(165, 1) (165,)

'''


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['teacher_number_of_previously_posted_projects'].values

X_train_ppp_norm = normalizer.transform(X_train['teacher_number_of_previously
#X_cv_price_norm = normalizer.transform(X_cv['price'].values.reshape(1,-1))
X_test_ppp_norm = normalizer.transform(X_test['teacher_number_of_previously_p

print("After vectorizations")
print(X_train_ppp_norm.shape, y_train.shape)
#print(X_cv_price_norm.shape, y_cv.shape)
print(X_test_ppp_norm.shape, y_test.shape)
```

```
After vectorizations
(33500, 1) (33500,)
(16500, 1) (16500,)
```

1.5 Vectorizing Text features

1.5.1 Vectorizing using BOW

Essay

In [36]:

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

print("\n\n")


vectorizer = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
vectorizer.fit(X_train['essay'].values) # fit has to happen only on train dat

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

print("After vectorizations")
print(X_train_essay_bow.shape, y_train.shape)
#print(X_cv_essay_bow.shape, y_cv.shape)
print(X_test_essay_bow.shape, y_test.shape)
```

```
(33500, 20) (33500,)
(16500, 20) (16500,)



After vectorizations
(33500, 5000) (33500,)
(16500, 5000) (16500,)
```

project_title

In [37]:

```
vectorizer = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
vectorizer.fit(X_train['project_title'].values) # fit has to happen only on t

# we use the fitted CountVectorizer to convert the text to vector
X_train_titles_bow = vectorizer.transform(X_train['project_title'].values)
#X_cv_titles_bow = vectorizer.transform(X_cv['project_title'].values)
X_test_titles_bow = vectorizer.transform(X_test['project_title'].values)

print("After vectorizations")
print(X_train_titles_bow.shape, y_train.shape)
#print(X_cv_titles_bow.shape, y_cv.shape)
print(X_test_titles_bow.shape, y_test.shape)
```

```
After vectorizations
(33500, 4007) (33500,)
(16500, 4007) (16500,)
```

project_resource_summary

In [38]:

```
vectorizer = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
vectorizer.fit(X_train['project_resource_summary'].values) # fit has to happe

# we use the fitted CountVectorizer to convert the text to vector
X_train_psr_bow = vectorizer.transform(X_train['project_resource_summary'].va
#X_cv_psr_bow = vectorizer.transform(X_cv['project_resource_summary'].values)
X_test_psr_bow = vectorizer.transform(X_test['project_resource_summary'].valu

print("After vectorizations")
print(X_train_psr_bow.shape, y_train.shape)
#print(X_cv_psr_bow.shape, y_cv.shape)
print(X_test_psr_bow.shape, y_test.shape)
```

```
After vectorizations
(33500, 5000) (33500,)
(16500, 5000) (16500,)
```

1.5.2 Vectorizing using TFIDF

essay

In [39]:

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

X_train_essay_tfidf = vectorizer.transform(X_train['essay'].values)
#X_cv_essay_tfidf = vectorizer.transform(X_cv['essay'].values)
X_test_essay_tfidf = vectorizer.transform(X_test['essay'].values)

print("After vectorizations")
print(X_train_essay_tfidf.shape, y_train.shape)
#print(X_cv_essay_tfidf.shape, y_cv.shape)
print(X_test_essay_tfidf.shape, y_test.shape)
```

```
After vectorizations
(33500, 5000) (33500,)
(16500, 5000) (16500,)
```

project_title

In [40]:

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

X_train_titles_tfidf = vectorizer.transform(X_train['project_title'].values)
#X_cv_titles_tfidf = vectorizer.transform(X_cv['project_title'].values)
X_test_titles_tfidf = vectorizer.transform(X_test['project_title'].values)

print("After vectorizations")
print(X_train_titles_tfidf.shape, y_train.shape)
#print(X_cv_titles_tfidf.shape, y_cv.shape)
print(X_test_titles_tfidf.shape, y_test.shape)
```

```
After vectorizations
(33500, 4007) (33500,)
(16500, 4007) (16500,)
```

project_resource_summary

In [41]:

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

X_train_prs_tfidf = vectorizer.transform(X_train['project_resource_summary'].
#X_cv_prs_tfidf = vectorizer.transform(X_cv['project_resource_summary'].value
X_test_prs_tfidf = vectorizer.transform(X_test['project_resource_summary'].va

print("After vectorizations")
print(X_train_prs_tfidf.shape, y_train.shape)
#print(X_cv_prs_tfidf.shape, y_cv.shape)
print(X_test_prs_tfidf.shape, y_test.shape)
```

```
After vectorizations
(33500, 5000) (33500,)
(16500, 5000) (16500,)
```

1.5.3 Vectorizing using AVG W2V

In [42]:

```
# stronging variables into pickle files python: http://www.jessicayung.com/ho
# make sure you have the glove_vectors file
with open('glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())
```

essay

In [43]:

```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_essay_train = []; # the avg-w2v for each sentence/review is stored ir
for sentence in tqdm(X_train['essay'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_essay_train.append(vector)

print(len(avg_w2v_essay_train))
print(len(avg_w2v_essay_train[0]))
```

```
100%|████████████████████████████████████████████████████████████
████████████████| 33500/33500 [00:11<00:00, 2944.52it/s]

33500
300
```

In [44]:

```
'''avg_w2v_essay_cv = []; # the avg-w2v for each sentence/review is stored ir
for sentence in tqdm(X_cv['essay'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_essay_cv.append(vector)'''
```

Out[44]:

```
"avg_w2v_essay_cv = []; # the avg-w2v for each sentence/review
is stored in this list\nfor sentence in tqdm(X_cv['essay'].val
ues): # for each review/sentence\n    vector = np.zeros(300) #
as word vectors are of zero length\n    cnt_words =0; # num of
words with a valid vector in the sentence/review\n    for word
in sentence.split(): # for each word in a review/sentence\n
if word in glove_words:\n            vector += model[word]\n
cnt_words += 1\n    if cnt_words != 0:\n        vector /= cnt_
words\n    avg_w2v_essay_cv.append(vector)"
```

In [45]:

```
avg_w2v_essay_test = []; # the avg-w2v for each sentence/review is stored in
for sentence in tqdm(X_test['essay'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_essay_test.append(vector)
```

```
100%|████████████████████████████████████████████████████████
████████████████| 16500/16500 [00:05<00:00, 2947.55it/s]
```

project_title

In [46]:

```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_titles_train = []; # the avg-w2v for each sentence/review is stored i
for sentence in tqdm(X_train['project_title'].values): # for each review/sent
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_titles_train.append(vector)

print(len(avg_w2v_titles_train))
print(len(avg_w2v_titles_train[0]))
```

```
100%|████████████████████████████████████████████████████████████████
████████████████| 33500/33500 [00:00<00:00, 147323.95it/s]

33500
300
```

In [47]:

```
'''
avg_w2v_titles_cv = []; # the avg-w2v for each sentence/review is stored in t
for sentence in tqdm(X_cv['project_title'].values): # for each review/sentenc
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_titles_cv.append(vector)

'''
```

Out[47]:

```
"\navg_w2v_titles_cv = []; # the avg-w2v for each sentence/rev
iew is stored in this list\nfor sentence in tqdm(X_cv['project
_title'].values): # for each review/sentence\n    vector = np.
zeros(300) # as word vectors are of zero length\n    cnt_words
=0; # num of words with a valid vector in the sentence/review
\n    for word in sentence.split(): # for each word in a revie
w/sentence\n        if word in glove_words:\n            vecto
r += model[word]\n            cnt_words += 1\n    if cnt_words
!= 0:\n        vector /= cnt_words\n    avg_w2v_titles_cv.appe
nd(vector)\n    \n"
```

In [48]:

```
avg_w2v_titles_test = []; # the avg-w2v for each sentence/review is stored ir
for sentence in tqdm(X_test['project_title'].values): # for each review/sente
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_titles_test.append(vector)
```

```
100%|████████████████████████████████████████████████████████████
████████████████| 16500/16500 [00:00<00:00, 151782.22it/s]
```

project_resource_summary

In [49]:

```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_prs_train = []; # the avg-w2v for each sentence/review is stored in t
for sentence in tqdm(X_train['project_resource_summary'].values): # for each
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_prs_train.append(vector)

print(len(avg_w2v_prs_train))
print(len(avg_w2v_prs_train[0]))
```

```
100%|████████████████████████████████████████████████████████████████
████████████████| 33500/33500 [00:01<00:00, 31340.56it/s]

33500
300
```

In [50]:

```
'''
avg_w2v_prs_cv = []; # the avg-w2v for each sentence/review is stored in this
for sentence in tqdm(X_cv['project_resource_summary'].values): # for each rev
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_prs_cv.append(vector)

'''
```

Out[50]:

```
"\navg_w2v_prs_cv = []; # the avg-w2v for each sentence/review
is stored in this list\nfor sentence in tqdm(X_cv['project_res
ource_summary'].values): # for each review/sentence\n    vecto
r = np.zeros(300) # as word vectors are of zero length\n    cn
t_words =0; # num of words with a valid vector in the sentenc
e/review\n    for word in sentence.split(): # for each word in
a review/sentence\n        if word in glove_words:\n
vector += model[word]\n            cnt_words += 1\n    if cnt_
words != 0:\n        vector /= cnt_words\n    avg_w2v_prs_cv.a
ppend(vector)\n    \n"
```

In [51]:

```
avg_w2v_prs_test = []; # the avg-w2v for each sentence/review is stored in th
for sentence in tqdm(X_test['project_resource_summary'].values): # for each r
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_prs_test.append(vector)
```

```
100%|████████████████████████████████████████████████████████████
████████████████| 16500/16500 [00:00<00:00, 30808.41it/s]
```

1.5.4 Vectorizing using TFIDF W2V

project_title

In [52]:

```
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train['project_title'])
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)
tfidf_words = set(tfidf_model.get_feature_names())
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_title_train = []; # the avg-w2v for each sentence/review is stored
for sentence in tqdm(X_train['project_title']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/revi
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.spli
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_title_train.append(vector)

print(len(tfidf_w2v_title_train))
print(len(tfidf_w2v_title_train[0]))
```

```
100%|██████████████████████████████████████████████████████████
██████████████| 33500/33500 [00:00<00:00, 99514.91it/s]

33500
300
```

In [53]:

```
'''tfidf_w2v_title_cv = []; # the avg-w2v for each sentence/review is stored
for sentence in tqdm(X_cv['project_title']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/revi
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.spli
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_title_cv.append(vector)

print(len(tfidf_w2v_title_cv))
print(len(tfidf_w2v_title_cv[0]))'''
```

Out[53]:

```
"tfidf_w2v_title_cv = []; # the avg-w2v for each sentence/revi
ew is stored in this list\nfor sentence in tqdm(X_cv['project_
title']): # for each review/sentence\n    vector = np.zeros(30
0) # as word vectors are of zero length\n    tf_idf_weight =0;
# num of words with a valid vector in the sentence/review\n
for word in sentence.split(): # for each word in a review/sent
ence\n        if (word in glove_words) and (word in tfidf_word
s):\n            vec = model[word] # getting the vector for ea
ch word\n            # here we are multiplying idf value(dicti
onary[word]) and the tf value((sentence.count(word)/len(senten
ce.split())))\n            tf_idf = dictionary[word]*(sentenc
e.count(word)/len(sentence.split())) # getting the tfidf value
for each word\n            vector += (vec * tf_idf) # calculat
ing tfidf weighted w2v\n            tf_idf_weight += tf_idf\n
if tf_idf_weight != 0:\n        vector /= tf_idf_weight\n    t
fidf_w2v_title_cv.append(vector)\n\nprint(len(tfidf_w2v_title_
cv))\nprint(len(tfidf_w2v_title_cv[0]))"
```

In [54]:

```
tfidf_w2v_title_test = []; # the avg-w2v for each sentence/review is stored i
for sentence in tqdm(X_test['project_title']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/revi
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.spli
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_title_test.append(vector)

print(len(tfidf_w2v_title_test))
print(len(tfidf_w2v_title_test[0]))
```

```
100%|██████████████████████████████████████████████████████████
██████████████| 16500/16500 [00:00<00:00, 97877.87it/s]

16500
300
```

In [55]:

```
#### essay
```

In [56]:

```
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train['essay'])
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)
tfidf_words = set(tfidf_model.get_feature_names())
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_essay_train = []; # the avg-w2v for each sentence/review is stored
for sentence in tqdm(X_train['essay']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/revi
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.spli
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_essay_train.append(vector)

print(len(tfidf_w2v_essay_train))
print(len(tfidf_w2v_essay_train[0]))
```

```
100%|██████████████████████████████████████████████████████████
██████████████████| 33500/33500 [02:13<00:00, 250.16it/s]

33500
300
```

In [57]:

```
'''tfidf_w2v_essay_cv = []; # the avg-w2v for each sentence/review is stored
for sentence in tqdm(X_cv['essay']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/revi
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.spli
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_essay_cv.append(vector)

print(len(tfidf_w2v_essay_cv))
print(len(tfidf_w2v_essay_cv[0]))'''
```

Out[57]:

```
"tfidf_w2v_essay_cv = []; # the avg-w2v for each sentence/revi
ew is stored in this list\nfor sentence in tqdm(X_cv['essa
y']): # for each review/sentence\n    vector = np.zeros(300) #
as word vectors are of zero length\n    tf_idf_weight =0; # nu
m of words with a valid vector in the sentence/review\n    for
word in sentence.split(): # for each word in a review/sentence
\n        if (word in glove_words) and (word in tfidf_word
s):\n            vec = model[word] # getting the vector for ea
ch word\n            # here we are multiplying idf value(dicti
onary[word]) and the tf value((sentence.count(word)/len(senten
ce.split())))\n            tf_idf = dictionary[word]*(sentenc
e.count(word)/len(sentence.split())) # getting the tfidf value
for each word\n            vector += (vec * tf_idf) # calculat
ing tfidf weighted w2v\n            tf_idf_weight += tf_idf\n
if tf_idf_weight != 0:\n        vector /= tf_idf_weight\n    t
fidf_w2v_essay_cv.append(vector)\n\nprint(len(tfidf_w2v_essay_
cv))\nprint(len(tfidf_w2v_essay_cv[0]))"
```

In [58]:

```
tfidf_w2v_essay_test = []; # the avg-w2v for each sentence/review is stored i
for sentence in tqdm(X_test['essay']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/revi
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.spli
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_essay_test.append(vector)

print(len(tfidf_w2v_essay_test))
print(len(tfidf_w2v_essay_test[0]))
```

```
100%|████████████████████████████████████████████████████████████
████████████████| 16500/16500 [01:06<00:00, 249.58it/s]

16500
300
```

project_resource_summary

In [59]:

```
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train['project_resource_summary'])
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)
tfidf_words = set(tfidf_model.get_feature_names())
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_prs_train = []; # the avg-w2v for each sentence/review is stored in
for sentence in tqdm(X_train['project_resource_summary']): # for each review/
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/revi
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.spli
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_prs_train.append(vector)

print(len(tfidf_w2v_prs_train))
print(len(tfidf_w2v_prs_train[0]))
```

```
100%|██████████████████████████████████████████████████████████
██████████████| 33500/33500 [00:03<00:00, 10720.03it/s]

33500
300
```

In [60]:

```
'''tfidf_w2v_prs_cv = []; # the avg-w2v for each sentence/review is stored ir
for sentence in tqdm(X_cv['project_resource_summary']): # for each review/ser
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/revi
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.spli
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_prs_cv.append(vector)

print(len(tfidf_w2v_prs_cv))
print(len(tfidf_w2v_prs_cv[0]))'''
```

Out[60]:

```
"tfidf_w2v_prs_cv = []; # the avg-w2v for each sentence/review
is stored in this list\nfor sentence in tqdm(X_cv['project_res
ource_summary']): # for each review/sentence\n    vector = np.
zeros(300) # as word vectors are of zero length\n    tf_idf_we
ight =0; # num of words with a valid vector in the sentence/re
view\n    for word in sentence.split(): # for each word in a r
eview/sentence\n        if (word in glove_words) and (word in
tfidf_words):\n            vec = model[word] # getting the vec
tor for each word\n            # here we are multiplying idf v
alue(dictionary[word]) and the tf value((sentence.count(word)/
len(sentence.split())))\n            tf_idf = dictionary[word]
*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word\n            vector += (vec * tf_idf)
# calculating tfidf weighted w2v\n            tf_idf_weight +=
tf_idf\n    if tf_idf_weight != 0:\n        vector /= tf_idf_w
eight\n    tfidf_w2v_prs_cv.append(vector)\n\nprint(len(tfidf_
w2v_prs_cv))\nprint(len(tfidf_w2v_prs_cv[0]))"
```

In [61]:

```
tfidf_w2v_prs_test = []; # the avg-w2v for each sentence/review is stored in
for sentence in tqdm(X_test['project_resource_summary']): # for each review/s
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/revi
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.spli
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_prs_test.append(vector)

print(len(tfidf_w2v_prs_test))
print(len(tfidf_w2v_prs_test[0]))
```

```
100%|██████████████████████████████████████████████████████████
██████████████| 16500/16500 [00:01<00:00, 10590.01it/s]

16500
300
```

Merging all the categorical and numerical features with variations of text features

In [62]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack

X_train_bow_matrix = hstack((X_train_cc_ohe, X_train_csc_ohe, X_train_grade_c
                             X_train_teacher_ohe, X_train_price_norm, X_train
                             X_train_titles_bow, X_train_psr_bow)).tocsr()

#X_cv_bow_matrix = hstack((X_cv_cc_ohe, X_cv_csc_ohe, X_cv_grade_ohe, X_cv_st
#                          X_cv_price_std, X_cv_ppp_std, X_cv_essay_bow, X_cv

X_test_bow_matrix = hstack((X_test_cc_ohe, X_test_csc_ohe, X_test_grade_ohe,
                            X_test_price_norm, X_test_ppp_norm, X_test_essay_

print("Final Data matrix")
print(X_train_bow_matrix.shape, y_train.shape)
#print(X_cv_bow_matrix.shape, y_cv.shape)
print(X_test_bow_matrix.shape, y_test.shape)
```

```
Final Data matrix
(33500, 14108) (33500,)
(16500, 14108) (16500,)
```

In [63]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack

X_train_tfidf_matrix = hstack((X_train_cc_ohe, X_train_csc_ohe, X_train_grade
                             X_train_teacher_ohe, X_train_price_norm, X_train
                             X_train_titles_tfidf, X_train_essay_tfidf, X_tra

#X_cv_tfidf_matrix = hstack((X_cv_cc_ohe, X_cv_csc_ohe, X_cv_grade_ohe, X_cv_
#                          X_cv_price_std, X_cv_ppp_std, X_cv_titles_tfidf,
#                          X_cv_essay_tfidf, X_cv_prs_tfidf)).tocsr()

X_test_tfidf_matrix = hstack((X_test_cc_ohe, X_test_csc_ohe, X_test_grade_ohe
                            X_test_price_norm, X_test_ppp_norm, X_test_titles
                            X_test_essay_tfidf, X_test_prs_tfidf)).tocsr()

print("Final Data matrix")
print(X_train_tfidf_matrix.shape, y_train.shape)
#print(X_cv_tfidf_matrix.shape, y_cv.shape)
print(X_test_tfidf_matrix.shape, y_test.shape)
```

```
Final Data matrix
(33500, 14108) (33500,)
(16500, 14108) (16500,)
```

In [64]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack

X_train_aw2v_matrix = hstack((X_train_cc_ohe, X_train_csc_ohe, X_train_grade_
                              X_train_teacher_ohe, X_train_price_norm, X_trair
                              avg_w2v_essay_train, avg_w2v_titles_train, avg_v

#X_cv_aw2v_matrix = hstack((X_cv_cc_ohe, X_cv_csc_ohe, X_cv_grade_ohe, X_cv_s
#                           X_cv_price_std, X_cv_ppp_std, avg_w2v_essay_cv,
#                            avg_w2v_titles_cv, avg_w2v_prs_cv)).tocsr()

X_test_aw2v_matrix = hstack((X_test_cc_ohe, X_test_csc_ohe, X_test_grade_ohe,
                             X_test_price_norm, X_test_ppp_norm, avg_w2v_essay
                              avg_w2v_titles_test, avg_w2v_prs_test)).tocsr()

print("Final Data matrix")
print(X_train_aw2v_matrix.shape, y_train.shape)
#print(X_cv_aw2v_matrix.shape, y_cv.shape)
print(X_test_aw2v_matrix.shape, y_test.shape)
```

```
Final Data matrix
(33500, 1001) (33500,)
(16500, 1001) (16500,)
```

In [65]:

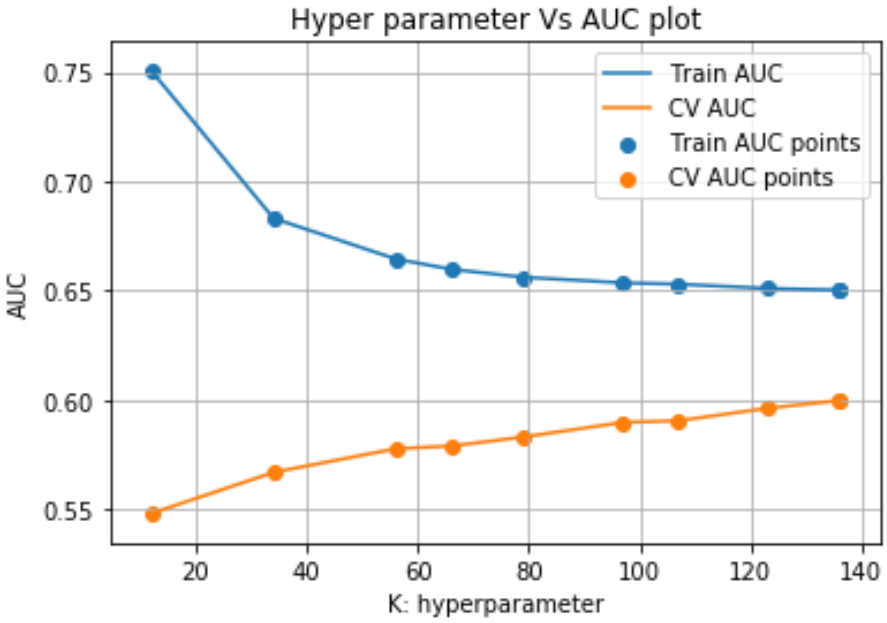
```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack

X_train_tw2v_matrix = hstack((X_train_cc_ohe, X_train_csc_ohe, X_train_grade_
                              X_train_teacher_ohe, X_train_price_norm, X_trair
                              tfidf_w2v_essay_train, tfidf_w2v_title_train, tf

#X_cv_tw2v_matrix = hstack((X_cv_cc_ohe, X_cv_csc_ohe, X_cv_grade_ohe, X_cv_s
#                           X_cv_price_std, X_cv_ppp_std, tfidf_w2v_essay_cv,
#                           tfidf_w2v_prs_cv)).tocsr()

X_test_tw2v_matrix = hstack((X_test_cc_ohe, X_test_csc_ohe, X_test_grade_ohe,
                              X_test_price_norm, X_test_ppp_norm, tfidf_w2v_es
                              tfidf_w2v_prs_test)).tocsr()

print("Final Data matrix")
print(X_train_tw2v_matrix.shape, y_train.shape)
#print(X_cv_tw2v_matrix.shape, y_cv.shape)
print(X_test_tw2v_matrix.shape, y_test.shape)
```

```
Final Data matrix
(33500, 1001) (33500,)
(16500, 1001) (16500,)
```

Finding Best Hyper parameter using K-Fold CV on BOW representation of text features

In [66]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.G
from sklearn.model_selection import GridSearchCV
from scipy.stats import randint as sp_randint
from sklearn.model_selection import RandomizedSearchCV
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score

neigh = KNeighborsClassifier(algorithm = 'brute',n_jobs=-1)
parameters = {'n_neighbors':sp_randint(1, 150)}
clf = RandomizedSearchCV(neigh, parameters, cv=5, scoring='roc_auc', return_t
clf.fit(X_train_bow_matrix, y_train)

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

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

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

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

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


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

results
```



Out[66]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_n_nei
1	0.108239	0.003040	10.755839	0.130516	
3	0.107204	0.003153	10.818862	0.237470	
9	0.104707	0.003287	10.465709	0.094702	
5	0.105123	0.002653	10.676665	0.097192	
0	0.103574	0.003447	10.725148	0.111154	
8	0.104330	0.002966	10.706611	0.195723	
7	0.103469	0.003869	10.643807	0.188224	
4	0.107164	0.004043	11.018056	0.267723	
2	0.105557	0.003835	10.737267	0.120059	
6	0.105180	0.003237	10.915866	0.132987	

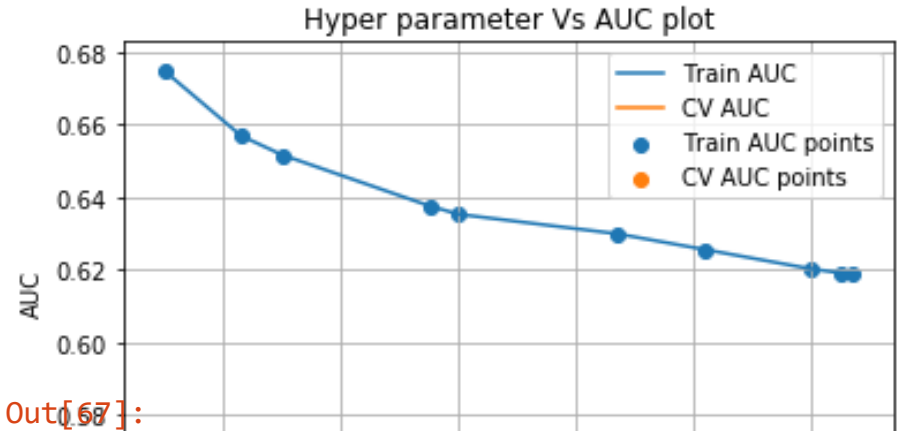
10 rows × 21 columns

Finding Best Hyper parameter using K-Fold CV on TFIDF representation of text features

In [67]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.G
from sklearn.model_selection import GridSearchCV
from scipy.stats import randint as sp_randint
from sklearn.model_selection import RandomizedSearchCV
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score

neigh = KNeighborsClassifier(algorithm = 'brute', n_jobs=-1)
parameters = {'n_neighbors':sp_randint(1, 150)}
clf = RandomizedSearchCV(neigh, parameters, cv=5, scoring='roc_auc', return_t
clf.fit(X_train_tfidf_matrix, y_train)

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

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


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

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

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


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

results
```

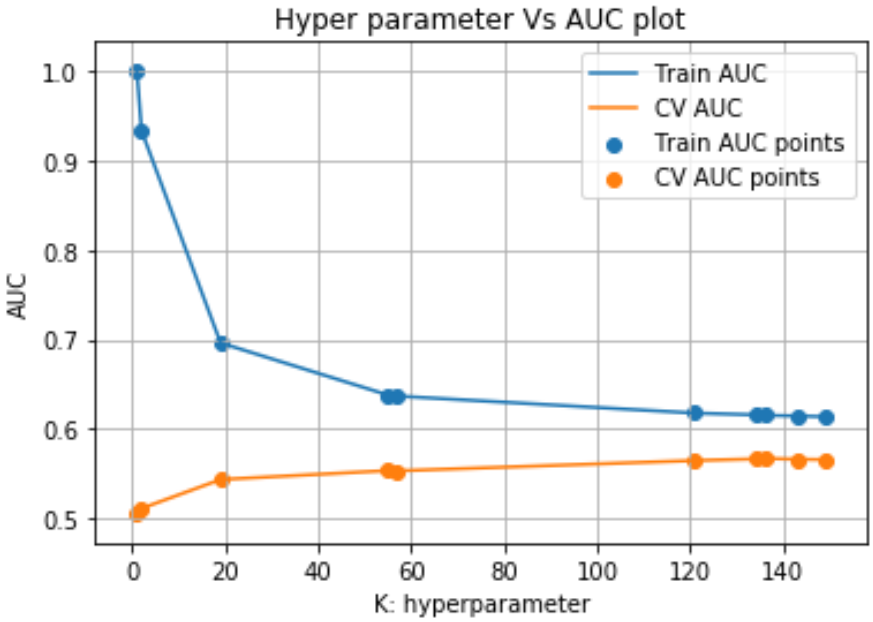
Out[67]:



Finding Best Hyper parameter using K-Fold CV on AVG W2V representation of text features

In [68]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.G
from sklearn.model_selection import GridSearchCV
from scipy.stats import randint as sp_randint
from sklearn.model_selection import RandomizedSearchCV
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score

neigh = KNeighborsClassifier(algorithm = 'brute', n_jobs=-1)
parameters = {'n_neighbors':sp_randint(1, 150)}
clf = RandomizedSearchCV(neigh, parameters, cv=5, scoring='roc_auc', return_t
clf.fit(X_train_aw2v_matrix, y_train)

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

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

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

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

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


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

results
```



Out[68]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_n_nei
7	0.299715	0.000837	75.778532	1.337191	
6	0.292595	0.008234	76.567012	3.127135	
8	0.300781	0.002156	76.563891	1.293774	
4	0.292823	0.006943	74.753818	0.418583	
5	0.287064	0.010353	82.982892	1.477475	
1	0.299073	0.002870	76.338219	0.994498	
0	0.301320	0.005962	76.284343	0.725688	
2	0.298657	0.002021	77.413077	1.271158	
9	0.297157	0.003262	76.301160	1.491027	
3	0.293747	0.007294	77.359023	1.140284	

10 rows × 21 columns

In [69]:

```
results
```

Out[69]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_n_nei
7	0.299715	0.000837	75.778532	1.337191	
6	0.292595	0.008234	76.567012	3.127135	
8	0.300781	0.002156	76.563891	1.293774	
4	0.292823	0.006943	74.753818	0.418583	
5	0.287064	0.010353	82.982892	1.477475	
1	0.299073	0.002870	76.338219	0.994498	
0	0.301320	0.005962	76.284343	0.725688	
2	0.298657	0.002021	77.413077	1.271158	
9	0.297157	0.003262	76.301160	1.491027	
3	0.293747	0.007294	77.359023	1.140284	

10 rows × 21 columns

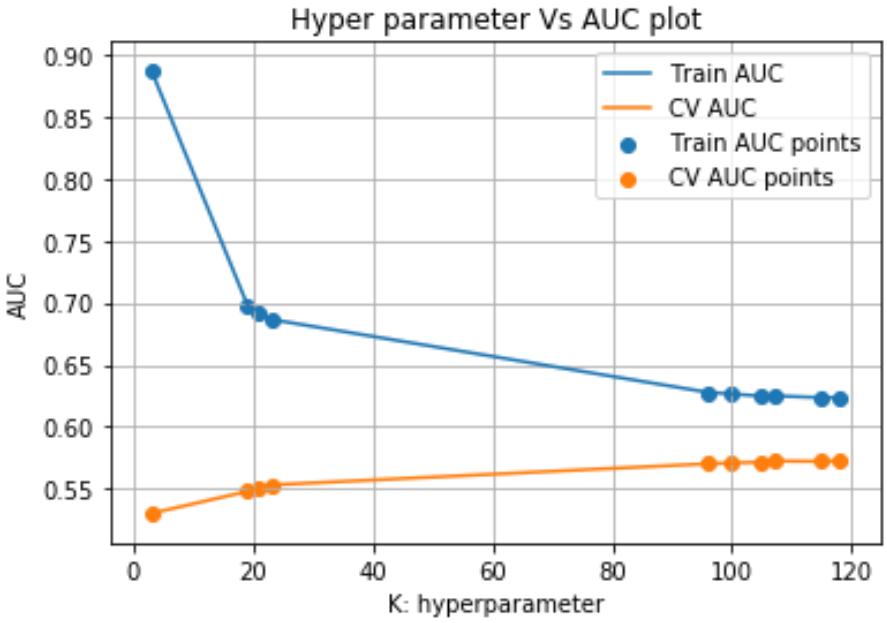
In [ ]:

```

```

Finding Best Hyper parameter using K-Fold CV on TFIDF W2V representation of text features

In [70]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.G
from sklearn.model_selection import GridSearchCV
from scipy.stats import randint as sp_randint
from sklearn.model_selection import RandomizedSearchCV
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score

neigh = KNeighborsClassifier(algorithm = 'brute',n_jobs=-1)
parameters = {'n_neighbors':sp_randint(1, 150)}
clf = RandomizedSearchCV(neigh, parameters, cv=5, scoring='roc_auc', return_t
clf.fit(X_train_tw2v_matrix, y_train)

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

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



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

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

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


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


Hyper parameter Vs AUC plot
Train AUC
CV AUC
Train AUC points
CV AUC points
AUC
K: hyperparameter
0.90
0.85
0.80
0.75
0.70
0.65
0.60
0.55
0
20
40
60
80
100
120

In [71]:

```
results
```

Out[71]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_n_nei
4	0.298974	0.003029	75.125938	1.174721	
1	0.297695	0.001645	75.025026	0.278172	
5	0.293615	0.009131	76.055030	0.458790	
6	0.298841	0.004585	75.178935	1.096542	
9	0.295378	0.001896	75.626227	0.911958	
0	0.310679	0.031671	75.872392	0.955168	
2	0.289153	0.010302	74.135857	0.914746	
8	0.295839	0.001652	75.268108	1.128078	
7	0.281392	0.011751	74.083354	0.927111	
3	0.294873	0.002759	76.253292	2.540745	

10 rows × 21 columns

In [ ]:

```

```

In [73]:

```
def batch_predict(clf, data):
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability
    # 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 - 496
    # 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
```

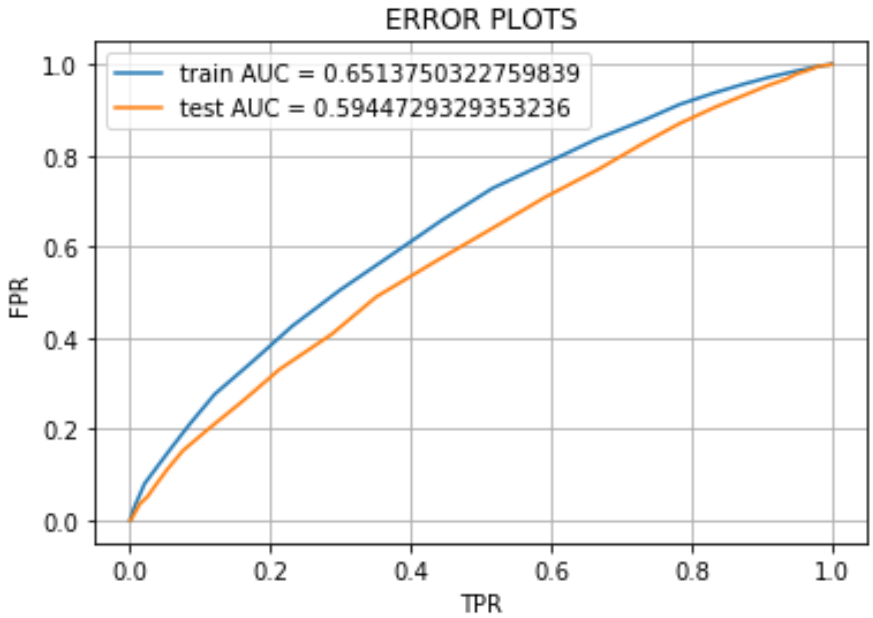
In [74]:

```
# 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 h
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for thresho
    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
```

Applying KNN with obtained best K (Hyper parameter) on BOW

In [75]:

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


neigh = KNeighborsClassifier(algorithm='brute', n_neighbors=137, n_jobs=-1)
neigh.fit(X_train_bow_matrix, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability esti
# not the predicted outputs

y_train_pred = batch_predict(neigh, X_train_bow_matrix)
y_test_pred = batch_predict(neigh, X_test_bow_matrix)

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

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

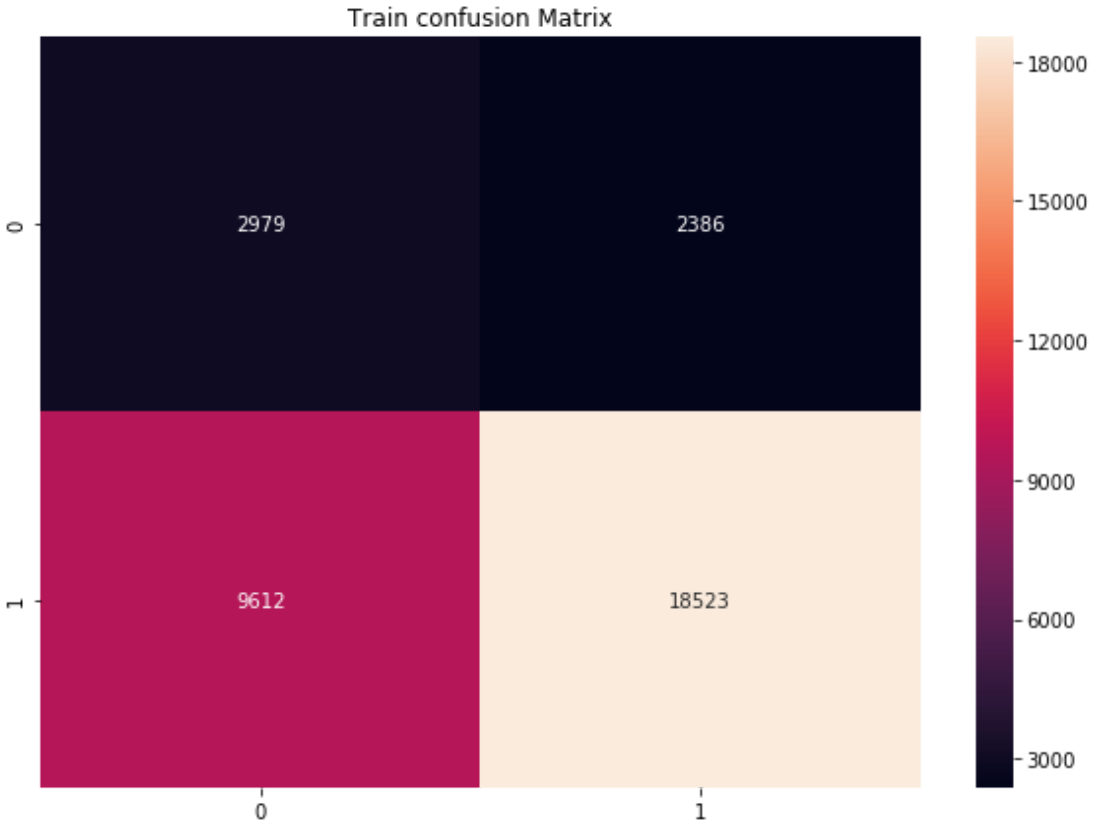


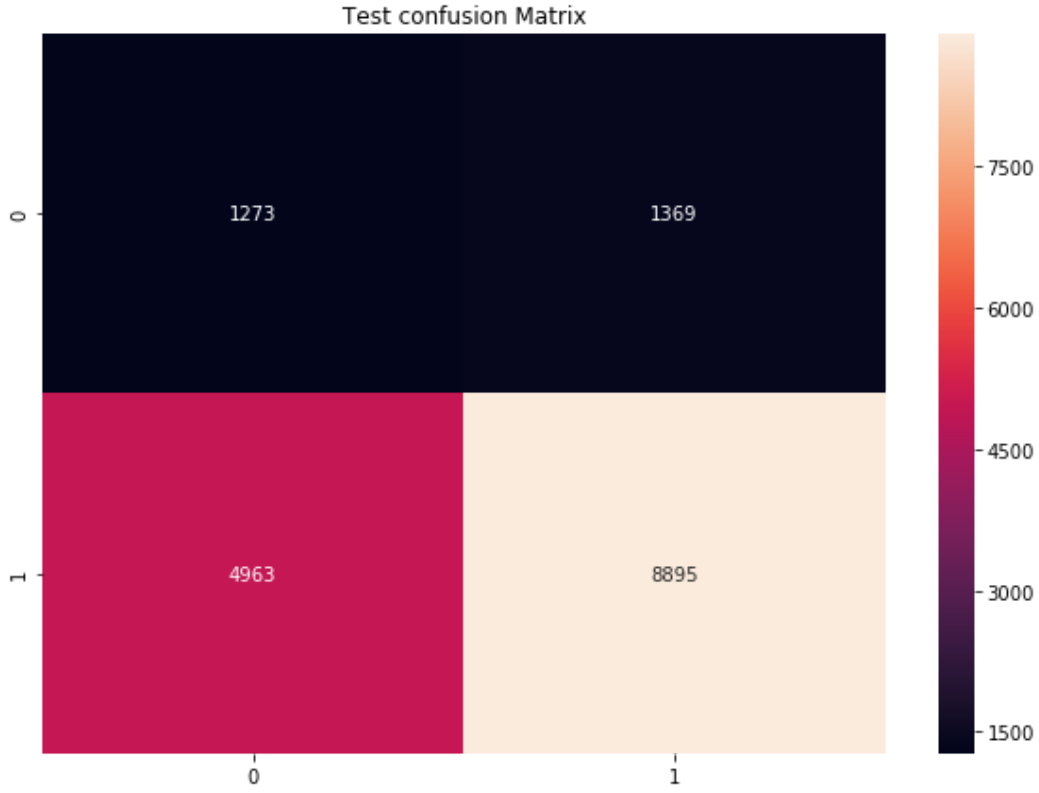
Confusion Matrix with predicted and original labels for BOW

In [77]:

```
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
train = confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t))
test = confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t))

#https://stackoverflow.com/a/35572247

df_cm = pd.DataFrame(train, index = [i for i in range(2)], columns = [i for i
plt.figure(figsize = (10,7))
plt.title('Train confusion Matrix')
sns.heatmap(train, annot=True, fmt="d")
plt.show()

df_cm = pd.DataFrame(test, index = [i for i in range(2)], columns = [i for i
plt.figure(figsize = (10,7))
plt.title('Test confusion Matrix')
sns.heatmap(test, annot=True, fmt="d")
plt.show()
```

```
the maximum value of tpr*(1-fpr) 0.3655654843484458 for thresh
old 0.832
```

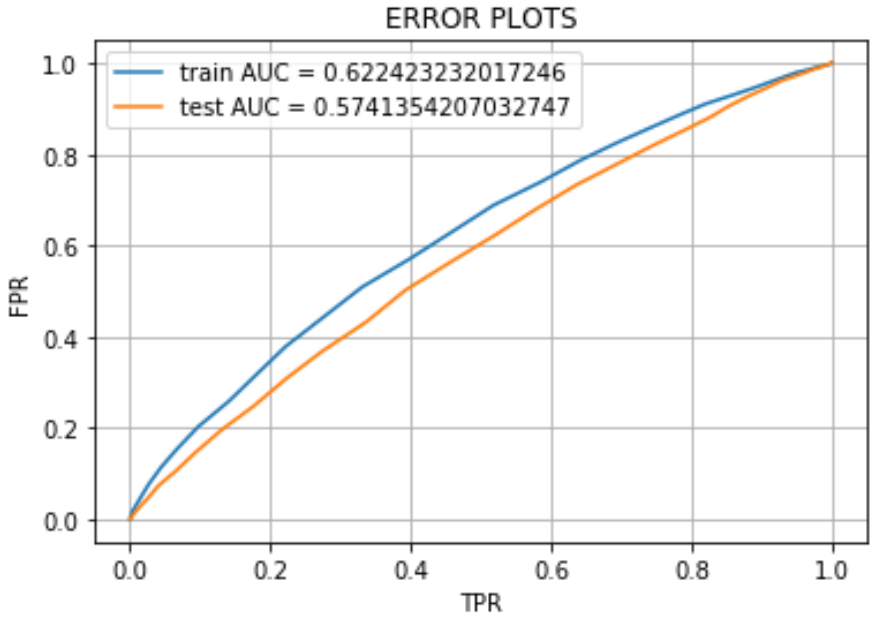




Applying KNN with obtained best K (Hyper parameter) on TFIDF

In [78]:

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


neigh = KNeighborsClassifier(algorithm='brute', n_neighbors=145, n_jobs=-1)
neigh.fit(X_train_tfidf_matrix, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability esti
# not the predicted outputs

y_train_pred = batch_predict(neigh, X_train_tfidf_matrix)
y_test_pred = batch_predict(neigh, X_test_tfidf_matrix)

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

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

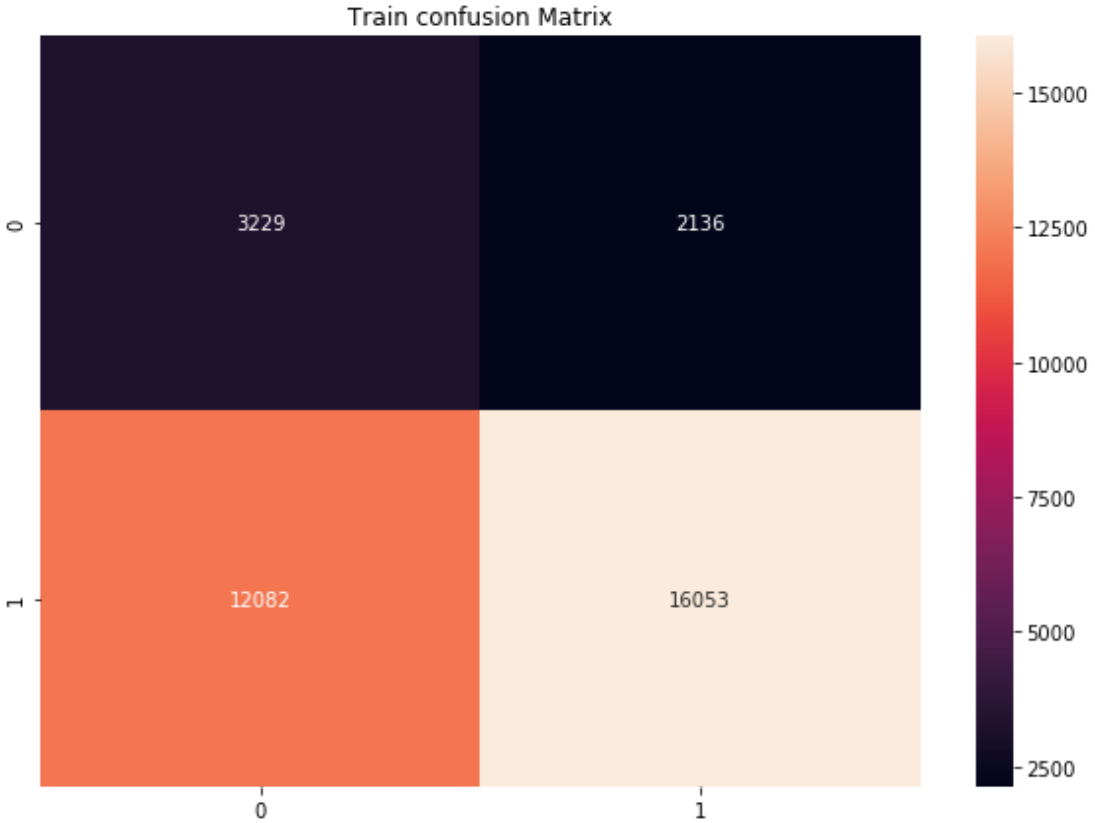


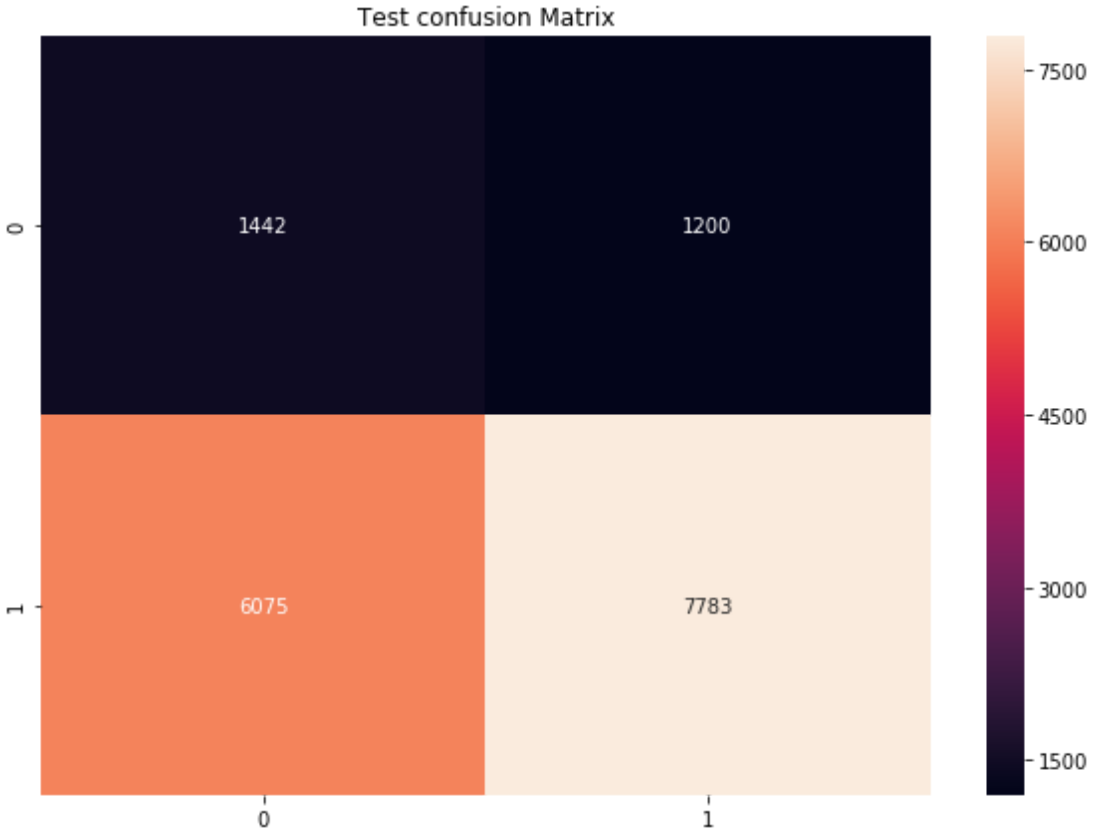
Confusion Matrix with predicted and original labels for TFIDF

In [79]:

```
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
train = confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t))
test = confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t))

#https://stackoverflow.com/a/35572247

df_cm = pd.DataFrame(train, index = [i for i in range(2)], columns = [i for i
plt.figure(figsize = (10,7))
plt.title('Train confusion Matrix')
sns.heatmap(train, annot=True, fmt="d")
plt.show()

df_cm = pd.DataFrame(test, index = [i for i in range(2)], columns = [i for i
plt.figure(figsize = (10,7))
plt.title('Test confusion Matrix')
sns.heatmap(test, annot=True, fmt="d")
plt.show()
```

```
the maximum value of tpr*(1-fpr) 0.343405783359455 for thresho
ld 0.848
```

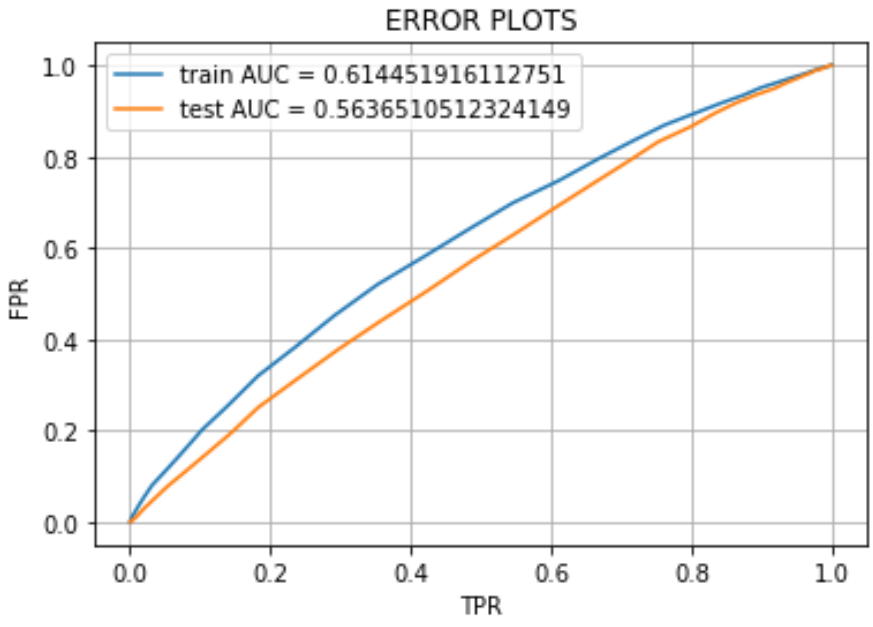




Applying KNN with obtained best K (Hyper parameter) on AVG W2V representation

In [80]:

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


neigh = KNeighborsClassifier(algorithm='brute', n_neighbors=149, n_jobs=-1)
neigh.fit(X_train_aw2v_matrix, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability esti
# not the predicted outputs

y_train_pred = batch_predict(neigh, X_train_aw2v_matrix)
y_test_pred = batch_predict(neigh, X_test_aw2v_matrix)

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

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

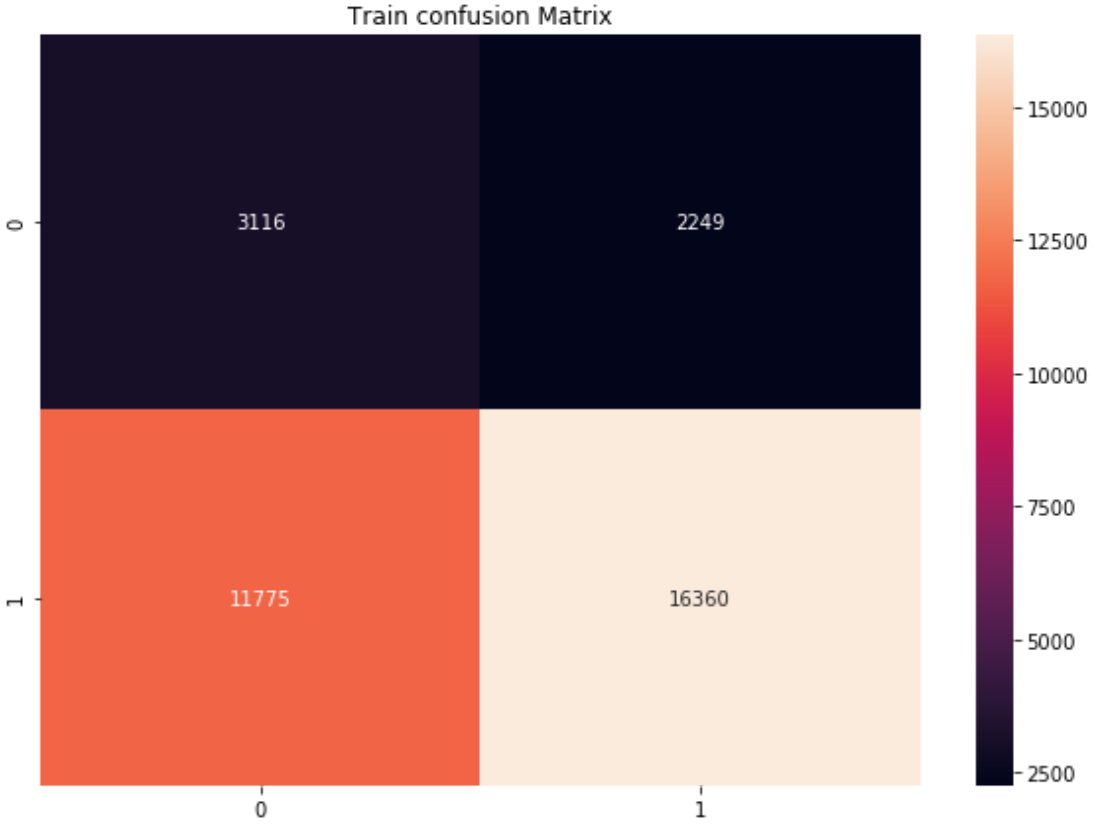


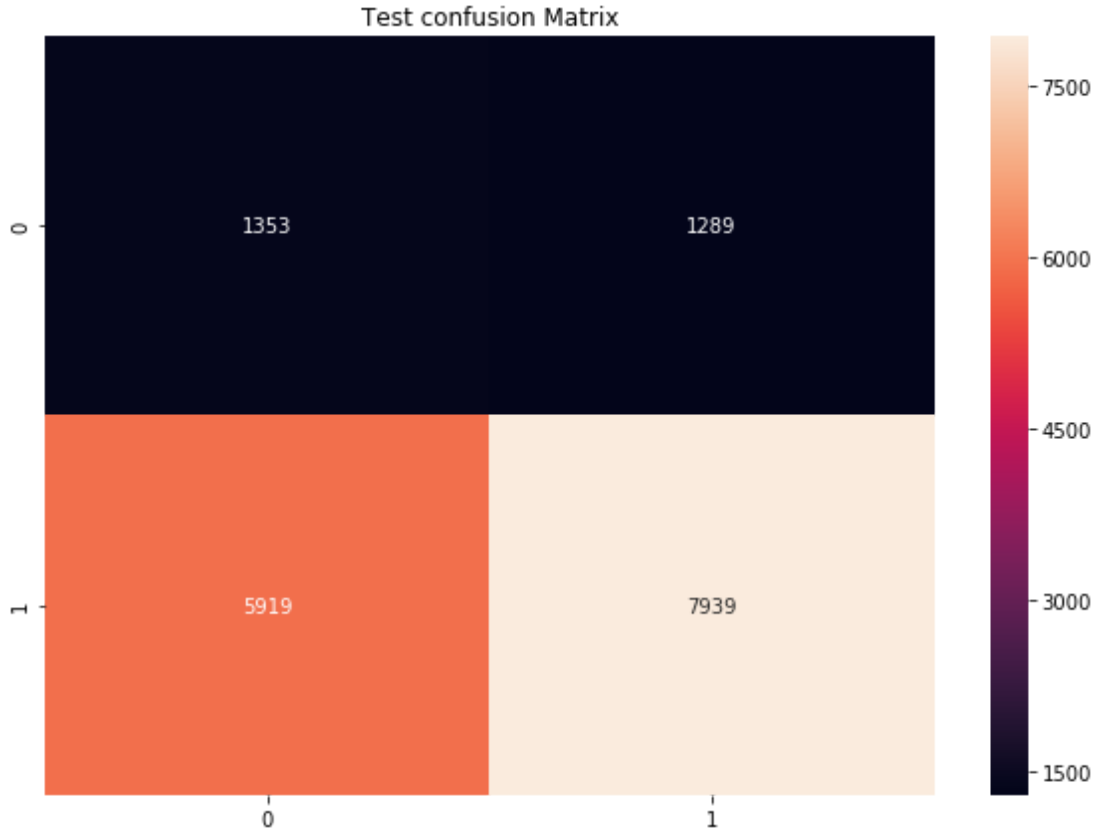
Confusion Matrix with predicted and original labels for AVG W2V

In [81]:

```
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
train = confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t))
test = confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t))

#https://stackoverflow.com/a/35572247

df_cm = pd.DataFrame(train, index = [i for i in range(2)], columns = [i for i
plt.figure(figsize = (10,7))
plt.title('Train confusion Matrix')
sns.heatmap(train, annot=True, fmt="d")
plt.show()

df_cm = pd.DataFrame(test, index = [i for i in range(2)], columns = [i for i
plt.figure(figsize = (10,7))
plt.title('Test confusion Matrix')
sns.heatmap(test, annot=True, fmt="d")
plt.show()
```

```
the maximum value of tpr*(1-fpr) 0.33772569380322637 for thres
hold 0.839
```

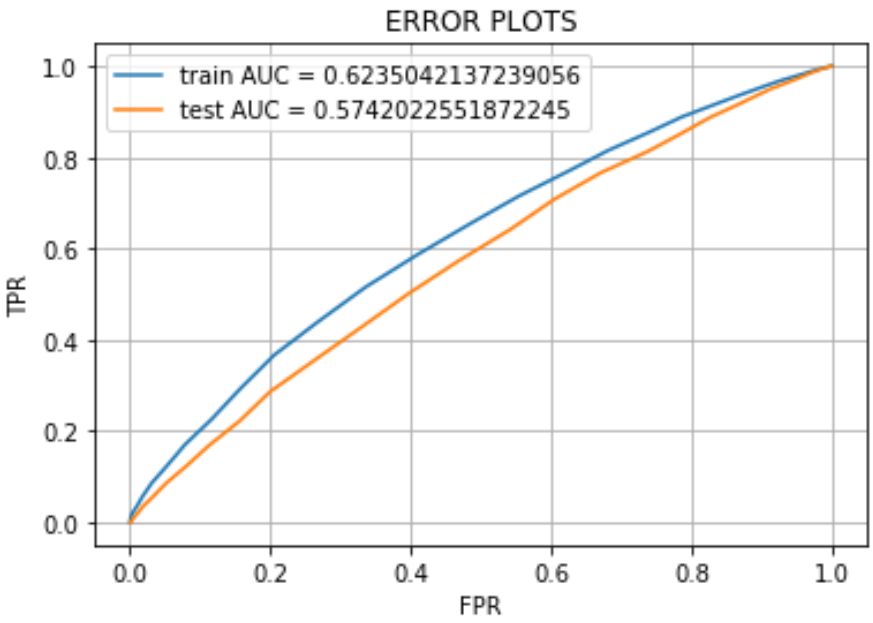




Applying KNN with obtained best K (Hyper parameter) on TFIDF W2V representation

In [82]:

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


neigh = KNeighborsClassifier(algorithm='brute', n_neighbors=119, n_jobs=-1)
neigh.fit(X_train_tw2v_matrix, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability esti
# not the predicted outputs

y_train_pred = batch_predict(neigh, X_train_tw2v_matrix)
y_test_pred = batch_predict(neigh, X_test_tw2v_matrix)

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

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

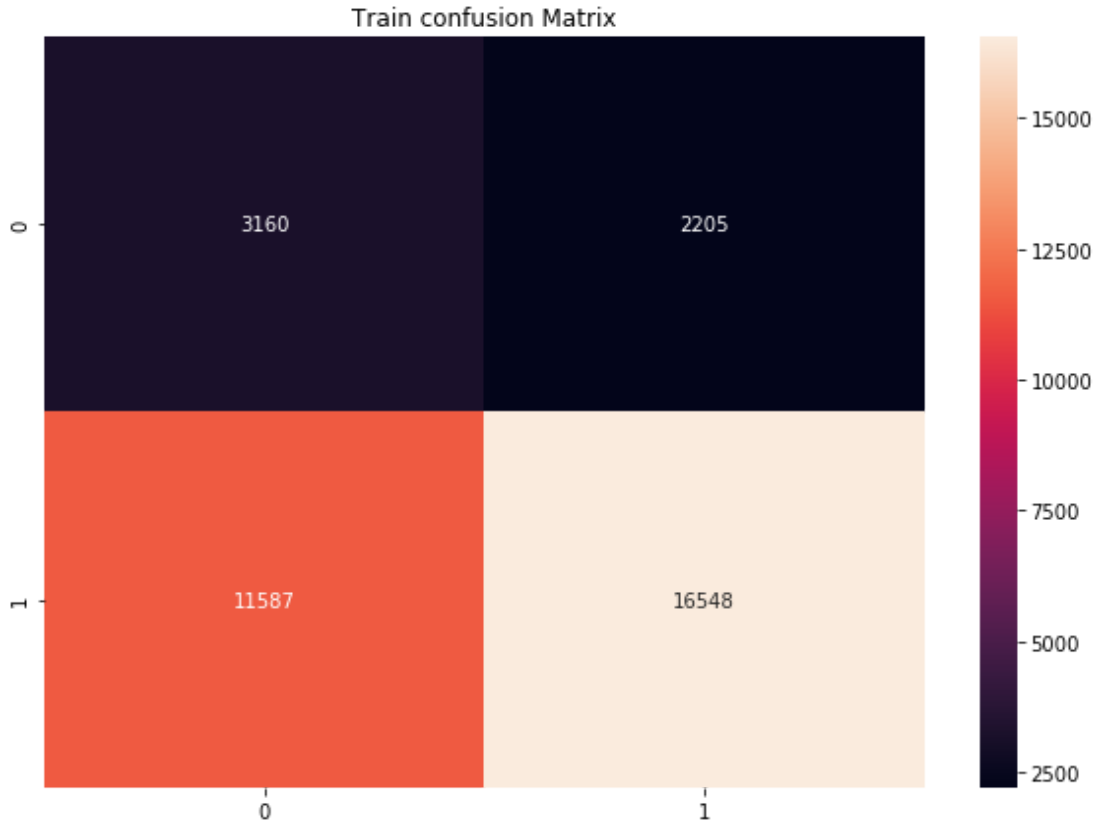


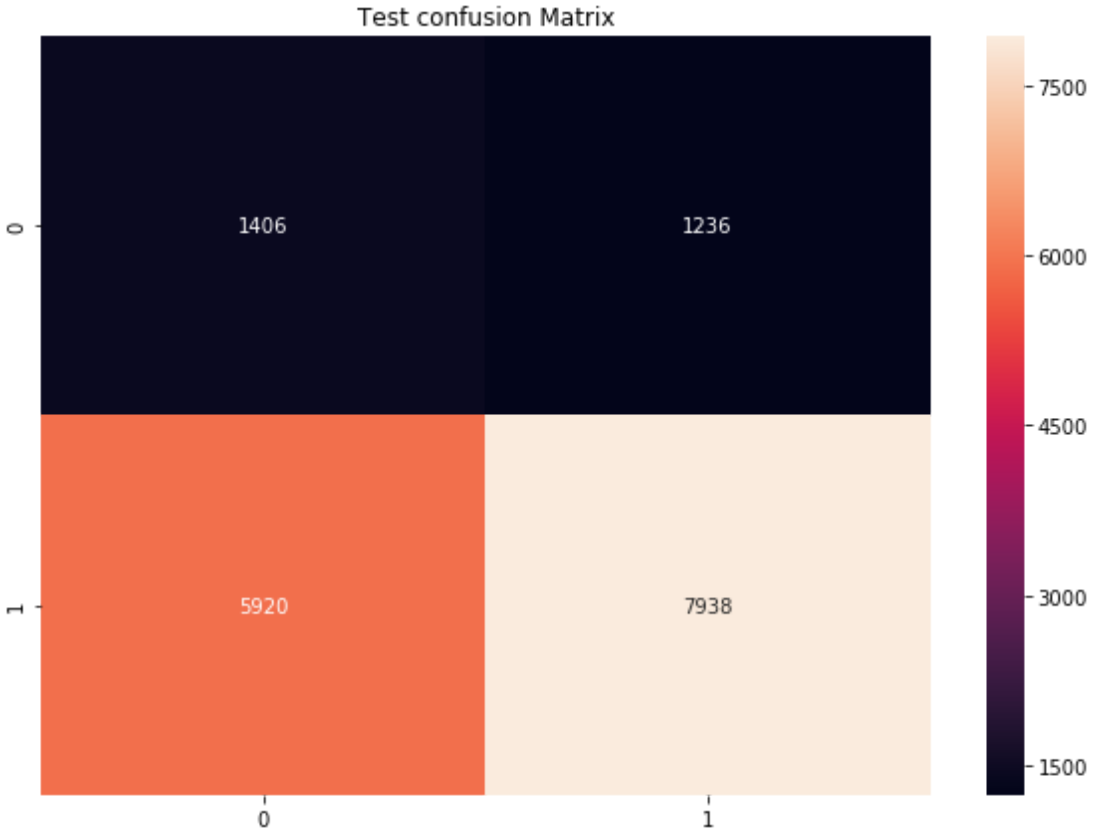
Confusion Matrix with predicted and original labels for TFIDF W2V

In [83]:

```
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
train = confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t))
test = confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t))

#https://stackoverflow.com/a/35572247

df_cm = pd.DataFrame(train, index = [i for i in range(2)], columns = [i for i
plt.figure(figsize = (10,7))
plt.title('Train confusion Matrix')
sns.heatmap(train, annot=True, fmt="d")
plt.show()

df_cm = pd.DataFrame(test, index = [i for i in range(2)], columns = [i for i
plt.figure(figsize = (10,7))
plt.title('Test confusion Matrix')
sns.heatmap(test, annot=True, fmt="d")
plt.show()
```

```
the maximum value of tpr*(1-fpr) 0.34643036312572967 for thres
hold 0.832
```





Selecting top 2000 features from TFIDF Train and Test data

In [84]:

```
from sklearn.feature_selection import SelectKBest, chi2
select = SelectKBest(chi2, k = 2000)
X_train_tfidf_matrix_2_k_features = select.fit_transform(X_train_tfidf_matrix
X_test_tfidf_matrix_2_k_features = select.transform(X_test_tfidf_matrix)
X_test_tfidf_matrix_2_k_features.shape
```

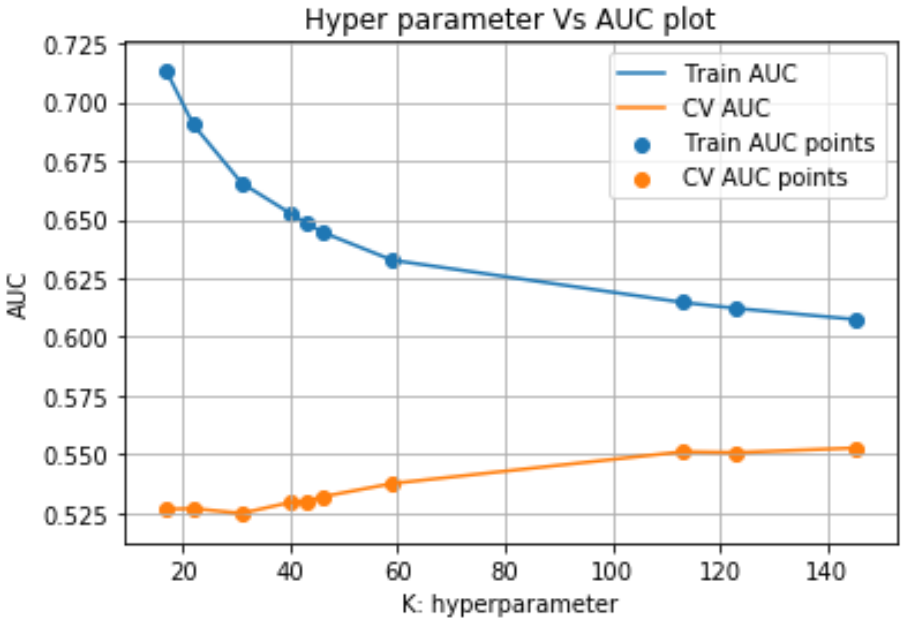
Out[84]:

```
(16500, 2000)
```

Finding Best Hyper parameter using K-Fold CV on TFIDF representation of 2000 text features

In [85]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.G
from sklearn.model_selection import GridSearchCV
from scipy.stats import randint as sp_randint
from sklearn.model_selection import RandomizedSearchCV
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score

neigh = KNeighborsClassifier(algorithm = 'brute', n_jobs=-1)
parameters = {'n_neighbors':sp_randint(1, 150)}
clf = RandomizedSearchCV(neigh, parameters, cv=5, scoring='roc_auc', return_t
clf.fit(X_train_tfidf_matrix_2_k_features, y_train)

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

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


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

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

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


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

results
```



Out[85]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_n_nei
6	0.012570	0.000795	5.000723	0.090404	
8	0.012572	0.000481	5.024391	0.155149	
1	0.012240	0.000441	5.022897	0.121563	
7	0.013159	0.000753	5.023350	0.098184	
2	0.013234	0.000381	5.055036	0.126651	
4	0.012969	0.000640	5.074456	0.169345	
9	0.013280	0.001177	5.109208	0.090077	
0	0.011973	0.000623	5.102439	0.106480	
5	0.012362	0.000801	5.132169	0.109037	
3	0.012222	0.000356	5.162560	0.070407	

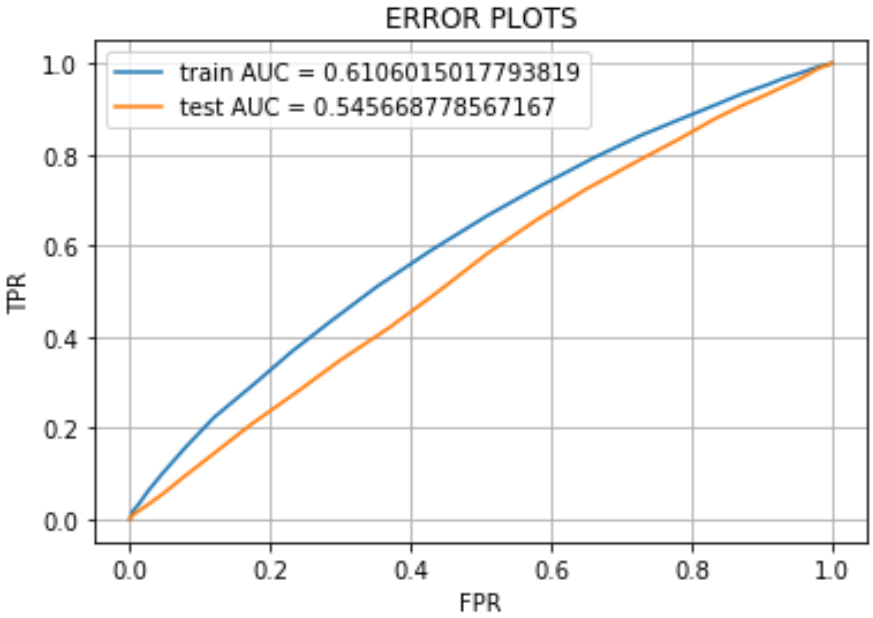
10 rows × 21 columns

Applying KNN with obtained best K (Hyper parameter)

on TFIDF representation with 2000 features

In [86]:

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


neigh = KNeighborsClassifier(algorithm='brute', n_neighbors=145, n_jobs=-1)
neigh.fit(X_train_tfidf_matrix_2_k_features, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability esti
# not the predicted outputs

y_train_pred = batch_predict(neigh, X_train_tfidf_matrix_2_k_features)
y_test_pred = batch_predict(neigh, X_test_tfidf_matrix_2_k_features)

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

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

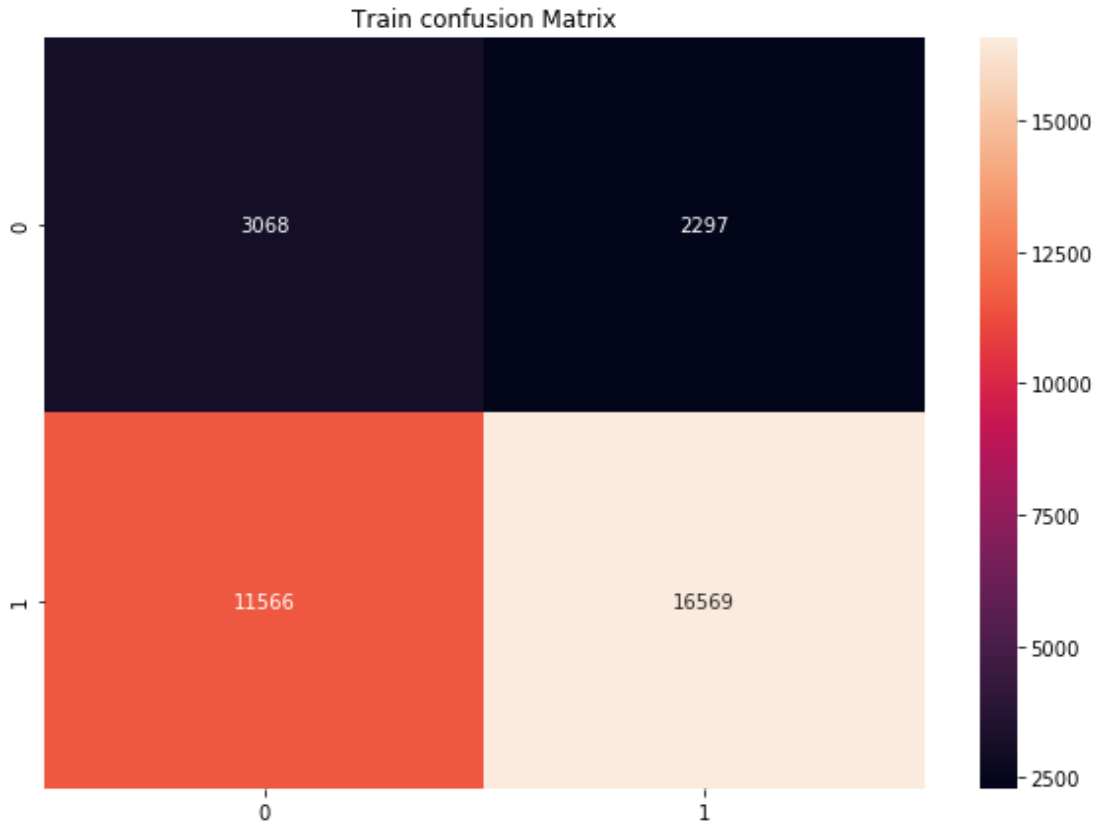


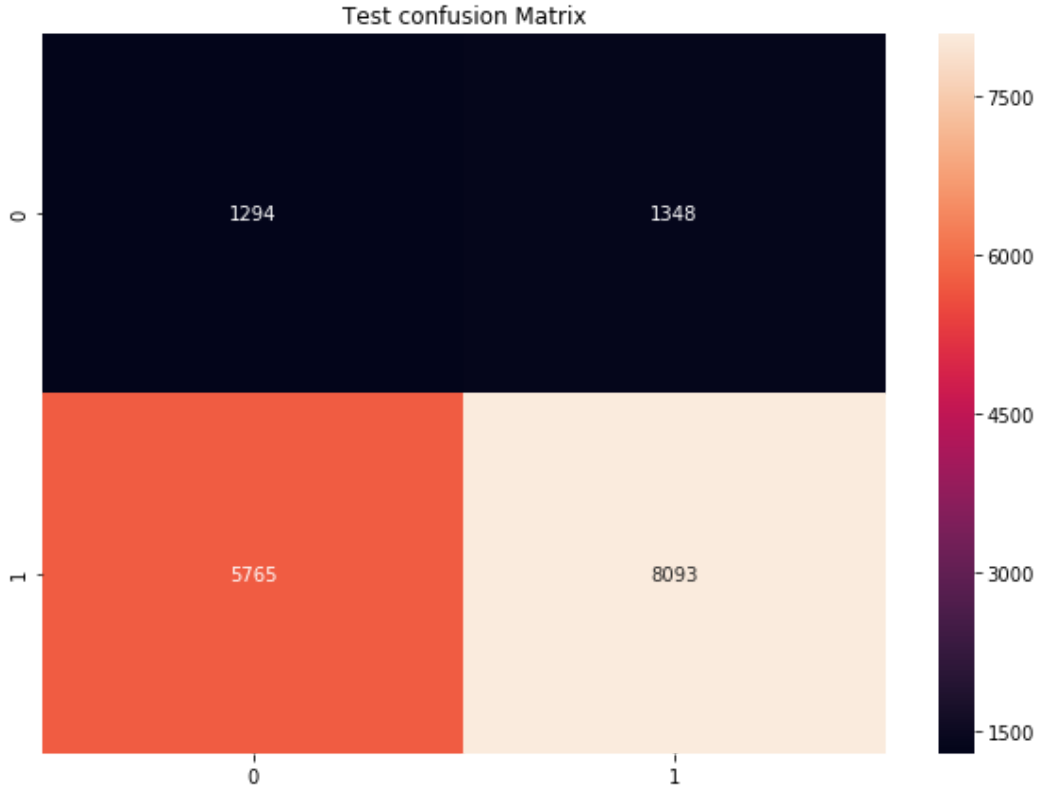
Confusion Matrix with predicted and original labels for 2000 TFIDF Train and Test data features

In [87]:

```
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
train = confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t))
test = confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t))

#https://stackoverflow.com/a/35572247

df_cm = pd.DataFrame(train, index = [i for i in range(2)], columns = [i for i
plt.figure(figsize = (10,7))
plt.title('Train confusion Matrix')
sns.heatmap(train, annot=True, fmt="d")
plt.show()

df_cm = pd.DataFrame(test, index = [i for i in range(2)], columns = [i for i
plt.figure(figsize = (10,7))
plt.title('Test confusion Matrix')
sns.heatmap(test, annot=True, fmt="d")
plt.show()
```

```
the maximum value of tpr*(1-fpr) 0.33677124885988546 for thres
hold 0.848
```





Conclusion

In [88]:

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

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

table.add_row(['BOW', 'Brute', 137, 0.594])
table.add_row(['TFIDF', 'Brute', 145, 0.574])
table.add_row(['AVG W2V', 'Brute', 149, 0.563])
table.add_row(['TFIDF W2V', 'Brute', 119, 0.574])
table.add_row(['TFIDF with 2K Features', 'Brute', 145, 0.545])
print(table)
```

```
+------------------------+-------+-----------------+-------+
|       Vectorizer       | Model | Hyper Parameter |  AUC  |
+------------------------+-------+-----------------+-------+
|          BOW           | Brute |       137       | 0.594 |
|         TFIDF          | Brute |       145       | 0.574 |
|        AVG W2V         | Brute |       149       | 0.563 |
|       TFIDF W2V        | Brute |       119       | 0.574 |
| TFIDF with 2K Features | Brute |       145       | 0.545 |
+------------------------+-------+-----------------+-------+
```

Summary

- BOW vectorizer gave AUC 0.594 with the best hyper parameter 137
- TFIDF vectorizer gave AUC 0.574 with the best hyper parameter 145
- AVG W2V vectorizer gave AUC 0.563 with the best hyper parameter 149
- TFIDF W2V vectorizer gave AUC 0.574 with the best hyper parameter 119
- TFIDF with 2K Features vectorizer gave AUC 0.545 with the best hyper parameter 145
- BOW vectorizer has the best AUC
- TFIDF, TFIDF W2V and AVG W2V has the next best AUC respectively
- TFIDF's AUC dropped from 0.574 to 0.545 when top 2000 features were selected
- TFIDF's hyper parameter number remained the same i. e. 145 when top 2000 features were selected