1 lakh data points from train_data.csv and 10,48,574 data points from resources.csv has been used in this assignment.

DonorsChoose

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

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

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

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

About the DonorsChoose Data Set

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

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

Feature	Description
`project_essay_1`	First application essay*
`project_essay_2`	Second application essay*
`project_essay_3`	Third application essay*
`project_essay_4`	Fourth application essay*
`project_submitted_datetime`	Datetime when project application was submitted. **Example:** `2016-04-28 12:43:56.245`
`teacher_id`	A unique identifier for the teacher of the proposed project. **Example:** `bdf8baa8fedef6bfeec7ae4ff1c15c56`
`teacher_prefix`	Teacher's title. One of the following enumerated values: • `nan` • `Dr.` • `Mr.` • `Mrs.` • `Ms.` • `Teacher.`
`teacher_number_of_previously_posted_projects`	Number of project applications previously submitted by the same teacher. **Example:** `2`

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

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

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

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

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

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

Notes on the Essay Data

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

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

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

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

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

In [1]:

```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
import sqlite3
import pandas as pd
```

```
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from nltk.stem.porter import PorterStemmer
import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer
from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle
from tqdm import tqdm
import os
import chart_studio.plotly
# from plotly import plotly
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()

from collections import Counter
from scipy.sparse import hstack,vstack
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score
from sklearn import model_selection
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import RandomizedSearchCV
#from sklearn.impute import SimpleImputer
from sklearn.datasets import load_digits
#from sklearn.feature_selection import SelectKBest, chi2
from sklearn.feature_selection import SelectKBest,f_classif
from sklearn.metrics import roc_auc_score
from prettytable import PrettyTable
import pdb
```

1.1 Reading Data

In [2]:

```
Project_data = pd.read_csv('1L_train_data.csv')
Resource_data = pd.read_csv('10L_resources.csv')
```

In [3]:

```
# how to replace elements in list python: https://stackoverflow.com/a/2582163/4084039
cols = ['Date' if x=='project_submitted_datetime' else x for x in list(Project_data.columns)]
#sort dataframe based on time pandas python: https://stackoverflow.com/a/49702492/4084039
Project_data['Date'] = pd.to_datetime(Project_data['project_submitted_datetime'])
Project_data.drop('project_submitted_datetime', axis=1, inplace=True)
Project_data.sort_values(by=['Date'], inplace=True)
# how to reorder columns pandas python: https://stackoverflow.com/a/13148611/4084039
Project_data = Project_data[cols]
Project_data.head(2)
```

Out[3]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_category	project_
70257	98044	p001225	9568c8968f974c1fc34def91394cb005	Ms	CA	2016-01-05	Grades PreK-2	Math &

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_category	project_
				Mrs.	CA	01:05:00	Grades PreK-2	
31477	47750	p185738	3afe10b996b7646d8641985a4b4b570d	Mrs.	UT	2016-01-05 01:05:00	Grades PreK-2	

1.2 preprocessing of `project_subject_categories`

In [4]:

```
#catogories = list(Project_data['project_subject_categories'].values)
## remove special characters from list of strings python:
https://stackoverflow.com/a/47301924/4084039
## https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
## https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-string
## https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-python
#cat_list = []
#for i in catogories:
#    temp = ""
#    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
#    for j in i.split(','): # it will split it in three parts ["Math & Science", "Warmth", "Care &
Hunger"]
#        if 'The' in j.split(): # this will split each of the catogory based on space "Math & Scie
ce"=> "Math","&", "Science"
#            j=j.replace('The','') # if we have the words "The" we are going to replace it with ''
i.e removing 'The')
#        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"Math & Scie
nce"=>"Math&Science"
#        temp+=j.strip()+" " #" abc ".strip() will return "abc", remove the trailing spaces
#        temp = temp.replace('&','_') # we are replacing the & value into
#    cat_list.append(temp.strip())
#
#Project_data['clean_categories'] = cat_list
#Project_data.drop(['project_subject_categories'], axis=1, inplace=True)
#
#from collections import Counter
#my_counter = Counter()
#for word in Project_data['clean_categories'].values:
#    my_counter.update(word.split())
#
#cat_dict = dict(my_counter)
#sorted_cat_dict_o = dict(sorted(cat_dict.items(), key=lambda kv: kv[1]))
#
```

1.3 preprocessing of `project_subject_subcategories`

In [5]:

```
#sub_catogories = list(Project_data['project_subject_subcategories'].values)
## remove special characters from list of strings python:
https://stackoverflow.com/a/47301924/4084039
#
## https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
## https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-string
## https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-python
#
#sub_cat_list = []
#for i in sub_catogories:
#    temp = ""
#    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
#    for j in i.split(','): # it will split it in three parts ["Math & Science", "Warmth", "Care &
Hunger"]
#        if 'The' in j.split(): # this will split each of the catogory based on space "Math & Scie
ce"=> "Math","&", "Science"
#            j=j.replace('The','') # if we have the words "The" we are going to replace it with ''
i.e removing 'The')
#        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"Math & Scie
nce"=>"Math&Science"
#        temp +=j.strip()+" "#" abc ".strip() will return "abc", remove the trailing spaces
#        temp = temp.replace('&','_')
```

```
#    sub_cat_list.append(temp.strip())
#
#Project_data['clean_subcategories'] = sub_cat_list
#Project_data.drop(['project_subject_subcategories'], axis=1, inplace=True)
#
## count of all the words in corpus python: https://stackoverflow.com/a/22898595/4084039
#my_counter = Counter()
#for word in Project_data['clean_subcategories'].values:
#    my_counter.update(word.split())
#
#sub_cat_dict = dict(my_counter)
#sorted_sub_cat_dict_o = dict(sorted(sub_cat_dict.items(), key=lambda kv: kv[1]))
```

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

In [6]:

```
y = Project_data['project_is_approved'].values
Project_data.drop(['project_is_approved'], axis=1, inplace=True)
X = Project_data
# train test split
X1, X_Test, y1, y_Test = train_test_split(X, y, test_size=0.33, random_state=0)
X_train, X_cv, y_train, y_cv = train_test_split(X1, y1, test_size=0.33, random_state=0)
print('Shape of X_train: ',X_train.shape)
print('Shape of y_train: ',y_train.shape)
print('Shape of X_cv: ',X_cv.shape)
print('Shape of y_cv: ',y_cv.shape)
print('Shape of X_Test: ',X_Test.shape)
print('Shape of y_Test: ',y_Test.shape)
```

```
Shape of X_train:  (44894, 16)
Shape of y_train:  (44894,)
Shape of X_cv:  (22112, 16)
Shape of y_cv:  (22112,)
Shape of X_Test:  (33004, 16)
Shape of y_Test:  (33004,)
```

1.3 Text preprocessing

In [7]:

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

In [8]:

```
## https://stackoverflow.com/a/47091490/4084039
#import re
#
#def decontracted(phrase):
#    # specific
#    phrase = re.sub(r"won't", "will not", phrase)
#    phrase = re.sub(r"can\'t", "can not", phrase)
#    # general
#    phrase = re.sub(r"n\'t", " not", phrase)
#    phrase = re.sub(r"\'re", " are", phrase)
#    phrase = re.sub(r"\'s", " is", phrase)
#    phrase = re.sub(r"\'d", " would", phrase)
```

```
#     phrase = re.sub(r"\'ll", " will", phrase)
#     phrase = re.sub(r"\'t", " not", phrase)
#     phrase = re.sub(r"\'ve", " have", phrase)
#     phrase = re.sub(r"\'m", " am", phrase)
#     return phrase
```

In [9]:

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

In [10]:

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

```
#     sent = ' '.join(e for e in sent.split() if e.lower() not in stopwords)
#     preprocessed_essays_Test_o.append(sent.lower().strip())
#
#print("Shape of preprocessed_essays_train_o after
preprocessing",len(preprocessed_essays_train_o))
#print("Shape of preprocessed_essays_cv_o after preprocessing",len(preprocessed_essays_cv_o))
#print("Shape of preprocessed_essays_Test_o after preprocessing",len(preprocessed_essays_Test_o))
## pdb.set_trace()
```

1.4 Preprocessing of `project_title`

In [11]:

```
#preprocessed_titles_train_o = []
## tqdm is for printing the status bar
#for sentance in tqdm(X_train['project_title'].values):
#     sent = decontracted(sentance)
#     sent = sent.replace('\\r', ' ')
#     sent = sent.replace('\\"', ' ')
#     sent = sent.replace('\\n', ' ')
#     sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
#     # https://gist.github.com/sebleier/554280
#     sent = ' '.join(e for e in sent.split() if e not in stopwords)
#     preprocessed_titles_train_o.append(sent.lower().strip())
#
#
#
#preprocessed_titles_cv_o = []
#for sentance in tqdm(X_cv['project_title'].values):
#     sent = decontracted(sentance)
#     sent = sent.replace('\\r', ' ')
#     sent = sent.replace('\\"', ' ')
#     sent = sent.replace('\\n', ' ')
#     sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
#     # https://gist.github.com/sebleier/554280
#     sent = ' '.join(e for e in sent.split() if e not in stopwords)
#     preprocessed_titles_cv_o.append(sent.lower().strip())
#
#preprocessed_titles_Test_o = []
#for sentance in tqdm(X_Test['project_title'].values):
#     sent = decontracted(sentance)
#     sent = sent.replace('\\r', ' ')
#     sent = sent.replace('\\"', ' ')
#     sent = sent.replace('\\n', ' ')
#     sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
#     # https://gist.github.com/sebleier/554280
#     sent = ' '.join(e for e in sent.split() if e not in stopwords)
#     preprocessed_titles_Test_o.append(sent.lower().strip())
#
#print("Shape of preprocessed_titles_train_o after
preprocessing",len(preprocessed_titles_train_o))
#print("Shape of preprocessed_titles_cv_o after preprocessing",len(preprocessed_titles_cv_o))
#print("Shape of preprocessed_titles_Test_o after preprocessing",len(preprocessed_titles_Test_o))
```

Make Data Model Ready: encoding numerical, categorical features

1.5 Preparing data for models

1.5.1 Vectorizing Categorical data

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

In [12]:

```
## we use count vectorizer to convert the values into one
#vectorizer_cat = CountVectorizer(vocabulary=list(sorted_cat_dict.keys()), lowercase=False, binary
=True)
#categories_one_hot_train_o = vectorizer_cat.fit_transform(X_train['clean_categories'].values)
```

```
#categories_one_hot_cv_o = vectorizer_cat.transform(X_cv['clean_categories'].values)
#categories_one_hot_Test_o = vectorizer_cat.transform(X_Test['clean_categories'].values)
#print(vectorizer_cat.get_feature_names())
#print("Shape of categories_one_hot_train matrix after one hot encodig
",categories_one_hot_train_o.shape)
#print("Shape of categories_one_hot_cv matrix after one hot encodig
",categories_one_hot_cv_o.shape)
#print("Shape of categories_one_hot_Test matrix after one hot encodig
",categories_one_hot_Test_o.shape)
```

In [13]:

```
# we use count vectorizer to convert the values into one
#vectorizer_sub_cat = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()),
lowercase=False, binary=True)
#sub_categories_one_hot_train_o =
vectorizer_sub_cat.fit_transform(X_train['clean_subcategories'].values)
#sub_categories_one_hot_cv_o = vectorizer_sub_cat.transform(X_cv['clean_subcategories'].values)
#sub_categories_one_hot_Test_o =
vectorizer_sub_cat.transform(X_Test['clean_subcategories'].values)
#print(vectorizer_sub_cat.get_feature_names())
#print("Shape of sub_categories_one_hot_train matrix after one hot encodig
",sub_categories_one_hot_train_o.shape)
#print("Shape of sub_categories_one_hot_cv matrix after one hot encodig
",sub_categories_one_hot_cv_o.shape)
#print("Shape of sub_categories_one_hot_Test matrix after one hot encodig
",sub_categories_one_hot_Test_o.shape)
```

School State

In [14]:

```
#sch1_catogories = list(X_train['school_state'].values)
#school_list = []
#for sent in sch1_catogories:
#    school_list.append(sent.lower().strip())
#X_train['school_categories'] = school_list
#X_train.drop(['school_state'], axis=1, inplace=True)
#
## count of all the words in corpus python: https://stackoverflow.com/a/22898595/4084039
#my_counter_sch = Counter()
#for word in X_train['school_categories'].values:
#    my_counter_sch.update(word.split())
#
## dict sort by value python: https://stackoverflow.com/a/613218/4084039
#sch_dict = dict(my_counter_sch)
#sorted_sch_dict_o = dict(sorted(sch_dict.items(), key=lambda kv: kv[1]))
#

#vectorizer_sch = CountVectorizer(vocabulary=list(sorted_sch_dict.keys()), lowercase=False, binary
=True)
#vectorizer_sch.fit(X_train['school_categories'].values)
##print(vectorizer.get_feature_names())
#
#sch_one_hot_train_o = vectorizer_sch.transform(X_train['school_categories'].values)
#print("Shape of sch_one_hot_train matrix after one hot encodig ",sch_one_hot_train_o.shape)
##------------------------------------------------------------------------------------------
#
#sch1_catogories_cv = list(X_cv['school_state'].values)
#school_list_cv = []
#for sent in sch1_catogories_cv:
#    school_list_cv.append(sent.lower().strip())
#X_cv['school_categories'] = school_list_cv
#X_cv.drop(['school_state'], axis=1, inplace=True)
#
## count of all the words in corpus python: https://stackoverflow.com/a/22898595/4084039
#my_counter_sch_cv = Counter()
#for word in X_cv['school_categories'].values:
#    my_counter_sch_cv.update(word.split())
#
## dict sort by value python: https://stackoverflow.com/a/613218/4084039
#sch_dict_cv = dict(my_counter_sch_cv)
#sorted_sch_dict_cv_o = dict(sorted(sch_dict_cv.items(), key=lambda kv: kv[1]))
#
```

```
#sch_one_hot_cv_o = vectorizer_sch.transform(X_cv['school_categories'].values)
#print("Shape of sch_one_hot_cv matrix after one hot encodig ",sch_one_hot_cv_o.shape)
#
##-------------------------------------------------------------------------------------------
#sch1_catogories_Test = list(X_Test['school_state'].values)
#school_list_Test = []
#for sent in sch1_catogories_Test:
#    school_list_Test.append(sent.lower().strip())
#X_Test['school_categories'] = school_list_Test
#X_Test.drop(['school_state'], axis=1, inplace=True)
#
## count of all the words in corpus python: https://stackoverflow.com/a/22898595/4084039
#my_counter_sch_Test = Counter()
#for word in X_Test['school_categories'].values:
#    my_counter_sch_Test.update(word.split())
#
## dict sort by value python: https://stackoverflow.com/a/613218/4084039
#sch_dict_Test = dict(my_counter_sch_Test)
#sorted_sch_dict_Test_o = dict(sorted(sch_dict_Test.items(), key=lambda kv: kv[1]))
#

#sch_one_hot_Test_o = vectorizer_sch.transform(X_Test['school_categories'].values)
#
#print("Shape of sch_one_hot_Test matrix after one hot encodig ",sch_one_hot_Test.shape)
```

Prefix

In [15]:

```
## remove special characters from list of strings python:
https://stackoverflow.com/a/47301924/4084039
## https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
## https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-string
## https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-python
#prefix_catogories_train = list(X_train['teacher_prefix'].values)
#prefix_list_train = []
#for sent in prefix_catogories_train:
#    sent = re.sub('[^A-Za-z0-9]+', ' ', str(sent))
#    # https://gist.github.com/sebleier/554280
#    sent = ' '.join(e for e in sent.split())
#    prefix_list_train.append(sent.lower().strip())
#X_train['prefix_catogories'] = prefix_list_train
#X_train.drop(['teacher_prefix'], axis=1, inplace=True)
#
## count of all the words in corpus python: https://stackoverflow.com/a/22898595/4084039
#my_counter_prefix_train = Counter()
#for word in X_train['prefix_catogories'].values:
#    my_counter_prefix_train.update(word.split())
#
## dict sort by value python: https://stackoverflow.com/a/613218/4084039
#prefix_dict_train = dict(my_counter_prefix_train)
#sorted_prefix_dict_train_o = dict(sorted(prefix_dict_train.items(), key=lambda kv: kv[1]))

#
#vectorizer_prefix = CountVectorizer(vocabulary=list(sorted_prefix_dict_train.keys()),
lowercase=False, binary=True)
#vectorizer_prefix.fit(X_train['prefix_catogories'].values)
##print(vectorizer.get_feature_names())
#
#prefix_one_hot_train_o = vectorizer_prefix.transform(X_train['prefix_catogories'].values)
#print("Shape of prefix_one_hot_train matrix after one hot encodig ",prefix_one_hot_train_o.shape)
##-------------------------------------------------------------------------------------------
#
#prefix_catogories_cv = list(X_cv['teacher_prefix'].values)
#prefix_list_cv = []
#for sent in prefix_catogories_cv:
#    sent = re.sub('[^A-Za-z0-9]+', ' ', str(sent))
#    # https://gist.github.com/sebleier/554280
#    sent = ' '.join(e for e in sent.split())
#    prefix_list_cv.append(sent.lower().strip())
#X_cv['prefix_catogories'] = prefix_list_cv
#X_cv.drop(['teacher_prefix'], axis=1, inplace=True)
#
## count of all the words in corpus python: https://stackoverflow.com/a/22898595/4084039
#my_counter_prefix_cv = Counter()
```

```
#my_counter_prefix_cv = Counter()
#for word in X_cv['prefix_catogories'].values:
#    my_counter_prefix_cv.update(word.split())
#
## dict sort by value python: https://stackoverflow.com/a/613218/4084039
#prefix_dict_cv = dict(my_counter_prefix_cv)
#sorted_prefix_dict_cv_o = dict(sorted(prefix_dict_cv.items(), key=lambda kv: kv[1]))
#

#prefix_one_hot_cv_o = vectorizer_prefix.transform(X_cv['prefix_catogories'].values)
#
#print("Shape of prefix_one_hot_cv matrix after one hot encodig ",prefix_one_hot_cv_o.shape)
#
##-------------------------------------------------------------------------------------------
#
#prefix_catogories_Test = list(X_Test['teacher_prefix'].values)
#prefix_list_Test = []
#for sent in prefix_catogories_Test:
#    sent = re.sub('[^A-Za-z0-9]+', ' ', str(sent))
#    # https://gist.github.com/sebleier/554280
#    sent = ' '.join(e for e in sent.split())
#    prefix_list_Test.append(sent.lower().strip())
#X_Test['prefix_catogories'] = prefix_list_Test
#X_Test.drop(['teacher_prefix'], axis=1, inplace=True)
#
## count of all the words in corpus python: https://stackoverflow.com/a/22898595/4084039
#my_counter_prefix_Test = Counter()
#for word in X_Test['prefix_catogories'].values:
#    my_counter_prefix_Test.update(word.split())
#
## dict sort by value python: https://stackoverflow.com/a/613218/4084039
#prefix_dict_Test = dict(my_counter_prefix_Test)
#sorted_prefix_dict_Test_o = dict(sorted(prefix_dict_Test.items(), key=lambda kv: kv[1]))
#

#prefix_one_hot_Test_o = vectorizer_prefix.transform(X_Test['prefix_catogories'].values)
#
#print("Shape of prefix_one_hot_Test matrix after one hot encodig ",prefix_one_hot_Test_o.shape)
```

project_grade_category

In [16]:

```
## remove special characters from list of strings python:
https://stackoverflow.com/a/47301924/4084039
## https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
## https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-string
## https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-python
#grade_catogories_train = list(X_train['project_grade_category'].values)
#grade_list_train = []
#for sent in grade_catogories_train:
#    sent = sent.replace('-','_')
#    sent = sent.replace(' ','_')
#    # sent = re.sub('[^A-Za-z0-9]+', ' ', str(sent))
#    # https://gist.github.com/sebleier/554280
#    sent = ' '.join(e for e in sent.split())
#    grade_list_train.append(sent.lower().strip())
#
## temp = temp.replace('-','_')
#X_train['new_grade_category'] = grade_list_train
#X_train.drop(['project_grade_category'], axis=1, inplace=True)
#
## count of all the words in corpus python: https://stackoverflow.com/a/22898595/4084039
#my_counter_grade_train = Counter()
#for word in X_train['new_grade_category'].values:
#    my_counter_grade_train.update(word.split())
#
## dict sort by value python: https://stackoverflow.com/a/613218/4084039
#grade_dict_train = dict(my_counter_grade_train)
#sorted_grade_dict_train_o = dict(sorted(grade_dict_train.items(), key=lambda kv: kv[1]))
#

#vectorizer_grade = CountVectorizer(vocabulary=list(sorted_grade_dict_train.keys()),
lowercase=False, binary=True)
#vectorizer_grade.fit(X_train['new_grade_category'].values)
##print(vectorizer.get_feature_names())
```

```
##print(vectorizer.get_feature_names())
#
#grade_one_hot_train_o = vectorizer_grade.transform(X_train['new_grade_category'].values)
#print("Shape of grade_one_hot_train matrix after one hot encodig ",grade_one_hot_train_o.shape)
##-------------------------------------------------------------------------------------------
#
#grade_catogories_cv = list(X_cv['project_grade_category'].values)
#grade_list_cv = []
#for sent in grade_catogories_cv:
#    sent = sent.replace('-','_')
#    sent = sent.replace(' ','_')
#    # sent = re.sub('[^A-Za-z0-9]+', ' ', str(sent))
#    # https://gist.github.com/sebleier/554280
#    sent = ' '.join(e for e in sent.split())
#    grade_list_cv.append(sent.lower().strip())
#
## temp = temp.replace('-','_')
#X_cv['new_grade_category'] = grade_list_cv
#X_cv.drop(['project_grade_category'], axis=1, inplace=True)
#
## count of all the words in corpus python: https://stackoverflow.com/a/22898595/4084039
#my_counter_grade_cv = Counter()
#for word in X_cv['new_grade_category'].values:
#    my_counter_grade_cv.update(word.split())
#
## dict sort by value python: https://stackoverflow.com/a/613218/4084039
#grade_dict_cv = dict(my_counter_grade_cv)
#sorted_grade_dict_cv_o = dict(sorted(grade_dict_cv.items(), key=lambda kv: kv[1]))
#
#


#grade_one_hot_cv_o = vectorizer_grade.transform(X_cv['new_grade_category'].values)
#print("Shape of grade_one_hot_cv matrix after one hot encodig ",grade_one_hot_cv_o.shape)
#
##-------------------------------------------------------------------------------------------
#
#grade_catogories_Test = list(X_Test['project_grade_category'].values)
#grade_list_Test = []
#for sent in grade_catogories_Test:
#    sent = sent.replace('-','_')
#    sent = sent.replace(' ','_')
#    # sent = re.sub('[^A-Za-z0-9]+', ' ', str(sent))
#    # https://gist.github.com/sebleier/554280
#    sent = ' '.join(e for e in sent.split())
#    grade_list_Test.append(sent.lower().strip())
#
## temp = temp.replace('-','_')
#X_Test['new_grade_category'] = grade_list_Test
#X_Test.drop(['project_grade_category'], axis=1, inplace=True)
#
## count of all the words in corpus python: https://stackoverflow.com/a/22898595/4084039
#my_counter_grade_Test = Counter()
#for word in X_Test['new_grade_category'].values:
#    my_counter_grade_Test.update(word.split())
#
## dict sort by value python: https://stackoverflow.com/a/613218/4084039
#grade_dict_Test = dict(my_counter_grade_Test)
#sorted_grade_dict_Test_o = dict(sorted(grade_dict_Test.items(), key=lambda kv: kv[1]))
#
#grade_one_hot_Test_o = vectorizer_grade.transform(X_Test['new_grade_category'].values)
#
#print("Shape of grade_one_hot_Test matrix after one hot encodig ",grade_one_hot_Test_o.shape)
```

1.5.2 Vectorizing Numerical features

In [17]:

```
#price_data = Resource_data.groupby('id').agg({'price':'sum', 'quantity':'sum'}).reset_index()
#X_train = pd.merge(X_train, price_data, on='id', how='left')
#X_cv = pd.merge(X_cv, price_data, on='id', how='left')
#X_Test = pd.merge(X_Test, price_data, on='id', how='left')
```

In [18]:

```
## check this one: https://www.youtube.com/watch?v=0HOqOcln3Z4&t=530s
## standardization sklearn: https://scikit-
learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
## price_standardized = standardScalar.fit(project_data['price'].values)
## this will rise the error
## ValueError: Expected 2D array, got 1D array instead: array=[725.05 213.03 329.   ... 399.   287
.73   5.5 ].
## Reshape your data either using array.reshape(-1, 1)
#
#price_scalar = StandardScaler()
#median_price = Resource_data['price'].median()
#X_train['price'] = X_train['price'].fillna(median_price)
#X_cv['price'] = X_cv['price'].fillna(median_price)
#X_Test['price'] = X_Test['price'].fillna(median_price)
#price_scalar.fit(X_train['price'].values.reshape(-1,1)) # finding the mean and standard deviation
of this data
#
#
## Now standardize the data with above maen and variance.
#price_standardized_train_o = price_scalar.transform(X_train['price'].values.reshape(-1, 1))
#price_standardized_cv_o = price_scalar.transform(X_cv['price'].values.reshape(-1, 1))
#price_standardized_Test_o = price_scalar.transform(X_Test['price'].values.reshape(-1, 1))
#print("Shape of price_standardized_train matrix after one hot encodig
",price_standardized_train_o.shape)
#print("Shape of price_standardized_cv matrix after one hot encodig
",price_standardized_cv_o.shape)
#print("Shape of price_standardized_Test matrix after one hot encodig
",price_standardized_Test_o.shape)
```

Make Data Model Ready: encoding eassay, and project_title

1.5.3 Vectorizing Text data

1.5.3.1 Bag of words

In [19]:

```
## We are considering only the words which appeared in at least 10 documents(rows or projects).
#vectorizer_essays_bow = CountVectorizer(min_df=10)
#text_bow_train_o = vectorizer_essays_bow.fit_transform(preprocessed_essays_train)
#text_bow_cv_o = vectorizer_essays_bow.transform(preprocessed_essays_cv)
#text_bow_Test_o = vectorizer_essays_bow.transform(preprocessed_essays_Test)
#print("Shape of matrix after one hot encodig ",text_bow_train_o.shape)
#print("Shape of text_bow_cv ",text_bow_cv_o.shape)
#print("Shape of text_bow_Test ",text_bow_Test_o.shape)
```

Bag of Words for Project Title

In [20]:

```
## you can vectorize the title also
## before you vectorize the title make sure you preprocess it
#vectorizer_titles_bow = CountVectorizer(min_df=10)
#title_bow_train_o = vectorizer_titles_bow.fit_transform(preprocessed_titles_train)
#title_bow_cv_o = vectorizer_titles_bow.transform(preprocessed_titles_cv)
#title_bow_Test_o = vectorizer_titles_bow.transform(preprocessed_titles_Test)
#print("Shape of matrix (title) after one hot encoding ",title_bow_train_o.shape)
#print("Shape of title_bow_cv ",title_bow_cv_o.shape)
#print("Shape of title_bow_test ",title_bow_Test_o.shape)
```

1.5.2.2 TFIDF vectorizer

In [21]:

```
#from sklearn.feature_extraction.text import TfidfVectorizer
#vectorizer_essays_tfidf = TfidfVectorizer(min_df=10)
#text_tfidf_train_o = vectorizer_essays_tfidf.fit_transform(preprocessed_essays_train)
#text_tfidf_cv_o = vectorizer_essays_tfidf.transform(preprocessed_essays_cv)
```

```
#text_tfidf_Test_o = vectorizer_essays_tfidf.transform(preprocessed_essays_Test)
#print("Shape of matrix after one hot encodig ",text_tfidf_train_o.shape)
#print("Shape of text_tfidf_cv ",text_tfidf_cv_o.shape)
#print("Shape of text_tfidf_test ",text_tfidf_Test_o.shape)
```

TFIDF vectorizer for Project Title

In [22]:

```
#vectorizer_titles_tfidf = TfidfVectorizer(min_df=10)
#title_tfidf_train_o = vectorizer_titles_tfidf.fit_transform(preprocessed_titles_train)
#title_tfidf_cv_o = vectorizer_titles_tfidf.transform(preprocessed_titles_cv)
#title_tfidf_Test_o = vectorizer_titles_tfidf.transform(preprocessed_titles_Test)
#print("Shape of matrix(title) after one hot encoding ",title_tfidf_train_o.shape)
#print("Shape of title_tfidf_cv ",title_tfidf_cv_o.shape)
#print("Shape of title_tfidf_test ",title_tfidf_Test_o.shape)
```

1.5.2.3 Using Pretrained Models: Avg W2V

In [23]:

```
# # Reading glove vectors in python: https://stackoverflow.com/a/38230349/4084039
# from tqdm import tqdm
# def loadGloveModel(gloveFile):
#     print ("Loading Glove Model")
#     f = open(gloveFile,'r', encoding="utf8")
#     model = {}
#     for line in tqdm(f):
#         splitLine = line.split()
#         word = splitLine[0]
#         embedding = np.array([float(val) for val in splitLine[1:]])
#         model[word] = embedding
#     print ("Done.",len(model)," words loaded!")
#     return model
# model = loadGloveModel('glove.42B.300d.txt')
# '''
# # ============================
# Output:
#
# Loading Glove Model
# 1917495it [06:32, 4879.69it/s]
# Done. 1917495  words loaded!
# # ============================
# '''
# words = []
# for i in preprocessed_essays:
#     words.extend(i.split(' '))
# for i in preprocessed_titles:
#     words.extend(i.split(' '))
# print("all the words in the corpus", len(words))
# words = set(words)
# print("the unique words in the corpus", len(words))
# inter_words = set(model.keys()).intersection(words)
# print("The number of words that are present in both glove vectors and our corpus", \
#       len(inter_words),"(",np.round(len(inter_words)/len(words)*100,3),"%)")
# words_corpus = {}
# words_glove = set(model.keys())
# for i in words:
#     if i in words_glove:
#         words_corpus[i] = model[i]
# print("word 2 vec length", len(words_corpus))
# # stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-pickle-to-
save-and-load-variables-in-python/
# import pickle
# with open('glove_vectors', 'wb') as f:
#     pickle.dump(words_corpus, f)
```

In [24]:

```
## stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-pickle-to-s
ave-and-load-variables-in-python/
## make sure you have the glove_vectors file
```

```
#with open('glove_vectors', 'rb') as f:
#    model = pickle.load(f)
#    glove_words =  set(model.keys())
```

Average Word2Vec for Project_Essays

In [25]:

```
#avg_w2v_vectors_train_o = []; # the avg-w2v for each sentence/review is stored in this list
#for sentence in tqdm(preprocessed_essays_train): # for each review/sentence
#    vector = np.zeros(300) # as word vectors are of zero length
#    cnt_words =0; # num of words with a valid vector in the sentence/review
#    for word in sentence.split(): # for each word in a review/sentence
#        if word in glove_words:
#            vector += model[word]
#            cnt_words += 1
#    if cnt_words != 0:
#        vector /= cnt_words
#    avg_w2v_vectors_train_o.append(vector)
##-------------------------------------------------------------------------------------------------
#
#avg_w2v_vectors_cv_o = []; # the avg-w2v for each sentence/review is stored in this list
#for sentence in tqdm(preprocessed_essays_cv): # for each review/sentence
#    vector = np.zeros(300) # as word vectors are of zero length
#    cnt_words =0; # num of words with a valid vector in the sentence/review
#    for word in sentence.split(): # for each word in a review/sentence
#        if word in glove_words:
#            vector += model[word]
#            cnt_words += 1
#    if cnt_words != 0:
#        vector /= cnt_words
#    avg_w2v_vectors_cv_o.append(vector)
#
##-------------------------------------------------------------------------------------------------
#
#avg_w2v_vectors_Test_o = []; # the avg-w2v for each sentence/review is stored in this list
#for sentence in tqdm(preprocessed_essays_Test): # for each review/sentence
#    vector = np.zeros(300) # as word vectors are of zero length
#    cnt_words =0; # num of words with a valid vector in the sentence/review
#    for word in sentence.split(): # for each word in a review/sentence
#        if word in glove_words:
#            vector += model[word]
#            cnt_words += 1
#    if cnt_words != 0:
#        vector /= cnt_words
#    avg_w2v_vectors_Test_o.append(vector)
#
#print(len(avg_w2v_vectors_cv_o))
#print(len(avg_w2v_vectors_cv_o[1]))
```

In [26]:

```
#ptt = open('avg_w2v_vectors_train', 'wb')
#pickle.dump(avg_w2v_vectors_train_o, ptt)
#ptt = open('avg_w2v_vectors_train', 'rb')
#avg_w2v_vectors_train = pickle.load(ptt)
#ptt.close()
#
#ptt = open('avg_w2v_vectors_cv', 'wb')
#pickle.dump(avg_w2v_vectors_cv_o, ptt)
#ptt = open('avg_w2v_vectors_cv', 'rb')
#avg_w2v_vectors_cv = pickle.load(ptt)
#ptt.close()
#
#ptt = open('avg_w2v_vectors_Test', 'wb')
#pickle.dump(avg_w2v_vectors_Test_o, ptt)
#ptt = open('avg_w2v_vectors_Test', 'rb')
#avg_w2v_vectors_Test = pickle.load(ptt)
#ptt.close()
```

AVG W2V on project_title

In [27]:

In [27]:

```
## Similarly you can vectorize for title also
## compute average word2vec for each title.
#avg_w2v_vectors_title_train_o = []; # the avg-w2v for each sentence/review is stored in this list
#for sentence in tqdm(preprocessed_titles_train): # for each review/sentence
#    vector_title = np.zeros(300) # as word vectors are of zero length
#    cnt_title_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_title += model[word]
#            cnt_title_words += 1
#    if cnt_title_words != 0:
#        vector_title /= cnt_title_words
#    avg_w2v_vectors_title_train_o.append(vector_title)
#
#
##-------------------------------------------------------------------------------------------
#avg_w2v_vectors_title_cv_o = []; # the avg-w2v for each sentence/review is stored in this list
#for sentence in tqdm(preprocessed_titles_cv): # for each review/sentence
#    vector_title = np.zeros(300) # as word vectors are of zero length
#    cnt_title_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_title += model[word]
#            cnt_title_words += 1
#    if cnt_title_words != 0:
#        vector_title /= cnt_title_words
#    avg_w2v_vectors_title_cv_o.append(vector_title)
#
##-------------------------------------------------------------------------------------------
#avg_w2v_vectors_title_Test_o = []; # the avg-w2v for each sentence/review is stored in this list
#for sentence in tqdm(preprocessed_titles_Test): # for each review/sentence
#    vector_title = np.zeros(300) # as word vectors are of zero length
#    cnt_title_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_title += model[word]
#            cnt_title_words += 1
#    if cnt_title_words != 0:
#        vector_title /= cnt_title_words
#    avg_w2v_vectors_title_Test_o.append(vector_title)
#
#print(len(avg_w2v_vectors_title_cv_o))
#print(len(avg_w2v_vectors_title_cv_o[0]))
```

1.5.2.3 Using Pretrained Models: TFIDF weighted W2V

In [28]:

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

TFIDF weighted W2V for Project_Essays

In [29]:

```
#tfidf_w2v_vectors_train_o = []; # the avg-w2v for each sentence/review is stored in this list
#for sentence in tqdm(preprocessed_essays_train): # for each review/sentence
#    vector = np.zeros(300) # as word vectors are of zero length
#    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
#    for word in sentence.split(): # for each word in a review/sentence
#        if (word in glove_words) and (word in tfidf_words_essays):
#            vec = model[word] # getting the vector for each word
#            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
#            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the
tfidf value for each word
#            vector += (vec * tf_idf) # calculating tfidf weighted w2v
#            tf_idf_weight += tf_idf
```

```
#    if tf_idf_weight != 0:
#        vector /= tf_idf_weight
#    tfidf_w2v_vectors_train_o.append(vector)
#
##-------------------------------------------------------------------------------------------
#tfidf_w2v_vectors_cv_o = []; # the avg-w2v for each sentence/review is stored in this list
#for sentence in tqdm(preprocessed_essays_cv): # for each review/sentence
#    vector = np.zeros(300) # as word vectors are of zero length
#    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
#    for word in sentence.split(): # for each word in a review/sentence
#        if (word in glove_words) and (word in tfidf_words_essays):
#            vec = model[word] # getting the vector for each word
#            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
#            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the
tfidf value for each word
#            vector += (vec * tf_idf) # calculating tfidf weighted w2v
#            tf_idf_weight += tf_idf
#    if tf_idf_weight != 0:
#        vector /= tf_idf_weight
#    tfidf_w2v_vectors_cv_o.append(vector)
#
##-------------------------------------------------------------------------------------------
#tfidf_w2v_vectors_Test_o = []; # the avg-w2v for each sentence/review is stored in this list
#for sentence in tqdm(preprocessed_essays_Test): # for each review/sentence
#    vector = np.zeros(300) # as word vectors are of zero length
#    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
#    for word in sentence.split(): # for each word in a review/sentence
#        if (word in glove_words) and (word in tfidf_words_essays):
#            vec = model[word] # getting the vector for each word
#            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
#            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the
tfidf value for each word
#            vector += (vec * tf_idf) # calculating tfidf weighted w2v
#            tf_idf_weight += tf_idf
#    if tf_idf_weight != 0:
#        vector /= tf_idf_weight
#    tfidf_w2v_vectors_Test_o.append(vector)
#
#print(len(tfidf_w2v_vectors_cv_o))
#print(len(tfidf_w2v_vectors_cv_o[0]))
#
```

TFIDF weighted W2V on project_title

In [30]:

```
## Similarly you can vectorize for title also
#tfidf_model_title = TfidfVectorizer()
#tfidf_model_title.fit(preprocessed_titles_train)
## we are converting a dictionary with word as a key, and the idf as a value
#dictionary = dict(zip(tfidf_model_title.get_feature_names(), list(tfidf_model_title.idf_)))
#tfidf_words_title = set(tfidf_model_title.get_feature_names())
#
## compute tfidf word2vec for each title.
#tfidf_w2v_vectors_title_train_o = []; # the avg-w2v for each sentence/review is stored in this li
st
#for sentence in tqdm(preprocessed_titles_train): # for each review/sentence
#    vector_title = np.zeros(300) # as word vectors are of zero length
#    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
#    for word in sentence.split(): # for each word in a review/sentence
#        if (word in glove_words) and (word in tfidf_words_title):
#            vec = model[word] # getting the vector for each word
#            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
#            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the
tfidf value for each word
#            vector_title += (vector_title * tf_idf) # calculating tfidf weighted w2v
#            tf_idf_weight += tf_idf
#    if tf_idf_weight != 0:
#        vector_title /= tf_idf_weight
#    tfidf_w2v_vectors_title_train_o.append(vector_title)
##-------------------------------------------------------------------------------------------
-----------------------------------
```

```
#
#tfidf_w2v_vectors_title_cv_o = []; # the avg-w2v for each sentence/review is stored in this list
#for sentence in tqdm(preprocessed_titles_cv): # for each review/sentence
#    vector_title = np.zeros(300) # as word vectors are of zero length
#    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
#    for word in sentence.split(): # for each word in a review/sentence
#        if (word in glove_words) and (word in tfidf_words_title):
#            vec = model[word] # getting the vector for each word
#            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
#            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the
tfidf value for each word
#            vector_title += (vector_title * tf_idf) # calculating tfidf weighted w2v
#            tf_idf_weight += tf_idf
#    if tf_idf_weight != 0:
#        vector_title /= tf_idf_weight
#    tfidf_w2v_vectors_title_cv_o.append(vector_title)
#
##-------------------------------------------------------------------------------------------------------
-----------------------------------
#
#
#tfidf_w2v_vectors_title_Test_o = []; # the avg-w2v for each sentence/review is stored in this lis
t
#for sentence in tqdm(preprocessed_titles_Test): # for each review/sentence
#    vector_title = np.zeros(300) # as word vectors are of zero length
#    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
#    for word in sentence.split(): # for each word in a review/sentence
#        if (word in glove_words) and (word in tfidf_words_title):
#            vec = model[word] # getting the vector for each word
#            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
#            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the
tfidf value for each word
#            vector_title += (vector_title * tf_idf) # calculating tfidf weighted w2v
#            tf_idf_weight += tf_idf
#    if tf_idf_weight != 0:
#        vector_title /= tf_idf_weight
#    tfidf_w2v_vectors_title_Test_o.append(vector_title)
#
#print(len(tfidf_w2v_vectors_title_cv_o))
#print(len(tfidf_w2v_vectors_title_cv_o[0]))
#
```

1.5.4 Merging all the above features

- we need to merge all the numerical vectors i.e catogorical, text, numerical vectors

Merging Vectorised Train data

In [31]:

```
## merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
#X1_train =
hstack((categories_one_hot_train,sub_categories_one_hot_train,sch_one_hot_train,grade_one_hot_trai
fix_one_hot_train,price_standardized_train,text_bow_train,title_bow_train))
#print(X1_train.shape)
#X3_train =
hstack((categories_one_hot_train,sub_categories_one_hot_train,sch_one_hot_train,grade_one_hot_trai
fix_one_hot_train,price_standardized_train,avg_w2v_vectors_train,avg_w2v_vectors_title_train))
#print(X3_train.shape)
#X4_train =
hstack((categories_one_hot_train,sub_categories_one_hot_train,sch_one_hot_train,grade_one_hot_trai
fix_one_hot_train,price_standardized_train,text_tfidf_train,title_tfidf_train))
#print(X4_train.shape)
#X2_train =
hstack((categories_one_hot_train,sub_categories_one_hot_train,sch_one_hot_train,grade_one_hot_trai
fix_one_hot_train,price_standardized_train,tfidf_w2v_vectors_train,tfidf_w2v_vectors_title_train))
#print(X2_train.shape)
#X5_train =
hstack((categories_one_hot_train,sub_categories_one_hot_train,sch_one_hot_train,grade_one_hot_trai
fix_one_hot_train,price_standardized_train,text_tfidf_train,title_tfidf_train))
#print(X5_train.shape)
```

Merging vectorised cv data

In [32]:

```
#X1_cv =
hstack((categories_one_hot_cv,sub_categories_one_hot_cv,sch_one_hot_cv,grade_one_hot_cv,prefix_one_
cv,price_standardized_cv,text_bow_cv,title_bow_cv))
#print(X1_cv.shape)
#X3_cv =
hstack((categories_one_hot_cv,sub_categories_one_hot_cv,sch_one_hot_cv,grade_one_hot_cv,prefix_one_
cv,price_standardized_cv,avg_w2v_vectors_cv,avg_w2v_vectors_title_cv))
#print(X3_cv.shape)
#X4_cv =
hstack((categories_one_hot_cv,sub_categories_one_hot_cv,sch_one_hot_cv,grade_one_hot_cv,prefix_one_
cv,price_standardized_cv,text_tfidf_cv,title_tfidf_cv))
#print(X4_cv.shape)
#X2_cv =
hstack((categories_one_hot_cv,sub_categories_one_hot_cv,sch_one_hot_cv,grade_one_hot_cv,prefix_one_
cv,price_standardized_cv,tfidf_w2v_vectors_cv,tfidf_w2v_vectors_title_cv))
#print(X2_cv.shape)
#X5_cv =
hstack((categories_one_hot_cv,sub_categories_one_hot_cv,sch_one_hot_cv,grade_one_hot_cv,prefix_one_
cv,price_standardized_cv,text_tfidf_cv,title_tfidf_cv))
#print(X5_cv.shape)
```

Merging vectorised Test data

In [33]:

```
#X1_Test =
hstack((categories_one_hot_Test,sub_categories_one_hot_Test,sch_one_hot_Test,grade_one_hot_Test,pre
one_hot_Test,price_standardized_Test,text_bow_Test,title_bow_Test))
#print(X1_Test.shape)
#X3_Test =
hstack((categories_one_hot_Test,sub_categories_one_hot_Test,sch_one_hot_Test,grade_one_hot_Test,pre
one_hot_Test,price_standardized_Test,avg_w2v_vectors_Test,avg_w2v_vectors_title_Test))
#print(X3_Test.shape)
#X4_Test =
hstack((categories_one_hot_Test,sub_categories_one_hot_Test,sch_one_hot_Test,grade_one_hot_Test,pre
one_hot_Test,price_standardized_Test,text_tfidf_Test,title_tfidf_Test))
#print(X4_Test.shape)
#X2_Test =
hstack((categories_one_hot_Test,sub_categories_one_hot_Test,sch_one_hot_Test,grade_one_hot_Test,pre
one_hot_Test,price_standardized_Test,tfidf_w2v_vectors_Test,tfidf_w2v_vectors_title_Test))
#print(X2_Test.shape)
#X5_Test =
hstack((categories_one_hot_Test,sub_categories_one_hot_Test,sch_one_hot_Test,grade_one_hot_Test,pre
one_hot_Test,price_standardized_Test,text_tfidf_Test,title_tfidf_Test))
#print(X5_Test.shape)
```

In [34]:

```
#ptt = open('X1_Test', 'wb')
#pickle.dump(X1_Test, ptt)
ptt = open('X1_Test', 'rb')
X1_Test = pickle.load(ptt)
ptt.close()

#ptt = open('X3_Test', 'wb')
#pickle.dump(X3_Test, ptt)
ptt = open('X3_Test', 'rb')
X3_Test = pickle.load(ptt)
ptt.close()

#ptt = open('X4_Test', 'wb')
#pickle.dump(X4_Test, ptt)
ptt = open('X4_Test', 'rb')
X4_Test = pickle.load(ptt)
ptt.close()

#ptt = open('X2_Test', 'wb')
```

```
#pickle.dump(X2_Test, ptt)
ptt = open('X2_Test', 'rb')
X2_Test = pickle.load(ptt)
ptt.close()

#ptt = open('X5_Test', 'wb')
#pickle.dump(X5_Test, ptt)
ptt = open('X5_Test', 'rb')
X5_Test = pickle.load(ptt)
ptt.close()
```

In [35]:

```
#X1_combo_o = vstack((X1_train,X1_cv))
#print(X1_combo_o.shape)
#X4_combo_o = vstack((X4_train,X4_cv))
#print(X4_combo_o.shape)
#X3_combo_o = vstack((X3_train,X3_cv))
#print(X3_combo_o.shape)
#X2_combo_o = vstack((X2_train,X2_cv))
#print(X2_combo_o.shape)
#X5_combo_o = vstack((X5_train,X5_cv))
#print(X5_combo_o.shape)
#
#print(y1.shape)
```

In [36]:

```
#ptt = open('X1_combo', 'wb')
#pickle.dump(X1_combo_o, ptt)
ptt = open('X1_combo', 'rb')
X1_combo = pickle.load(ptt)
ptt.close()

#ptt = open('X4_combo', 'wb')
#pickle.dump(X4_combo_o, ptt)
ptt = open('X4_combo', 'rb')
X4_combo = pickle.load(ptt)
ptt.close()

#ptt = open('X3_combo', 'wb')
#pickle.dump(X3_combo_o, ptt)
ptt = open('X3_combo', 'rb')
X3_combo = pickle.load(ptt)
ptt.close()

#ptt = open('X2_combo', 'wb')
#pickle.dump(X2_combo_o, ptt)
ptt = open('X2_combo', 'rb')
X2_combo = pickle.load(ptt)
ptt.close()

#ptt = open('X5_combo', 'wb')
#pickle.dump(X5_combo_o, ptt)
ptt = open('X5_combo', 'rb')
X5_combo = pickle.load(ptt)
ptt.close()
```

Assignment 3: Apply KNN

1. **[Task-1] Apply KNN(brute force version) on these feature sets**
   - Set 1: categorical, numerical features + project_title(BOW) + preprocessed_essay (BOW)
   - Set 2: categorical, numerical features + project_title(TFIDF)+ preprocessed_essay (TFIDF)
   - Set 3: categorical, numerical features + project_title(AVG W2V)+ preprocessed_essay (AVG W2V)
   - Set 4: categorical, numerical features + project_title(TFIDF W2V)+ preprocessed_essay (TFIDF W2V)

2. **Hyper paramter tuning to find best K**
   - Find the best hyper parameter which results in the maximum AUC value
   - Find the best hyper paramter using k-fold cross validation (or) simple cross validation data

- Use gridsearch-cv or randomsearch-cv or write your own for loops to do this task

3. **Representation of results**

   - You need to plot the performance of model both on train data and cross validation data for each hyper parameter, as shown in the figure
   - Once you find the best hyper parameter, you need to train your model-M using the best hyper-param. Now, find the AUC on test data and plot the ROC curve on both train and test using model-M.
   - Along with plotting ROC curve, you need to print the confusion matrix with predicted and original labels of test data points

4. **[Task-2]**

   - Select top 2000 features from feature Set 2 using `SelectKBest` and then apply KNN on top of these features

     ```
     from sklearn.datasets import load_digits
     from sklearn.feature_selection import SelectKBest, chi2
     X, y = load_digits(return_X_y=True)
     X.shape
     X_new = SelectKBest(chi2, k=20).fit_transform(X, y)
     X_new.shape
     ========
     output:
     (1797, 64)
     (1797, 20)
     ```

   - Repeat the steps 2 and 3 on the data matrix after feature selection

5. **Conclusion**

   - You need to summarize the results at the end of the notebook, summarize it in the table format. To print out a table please refer to this prettytable library link

Note: Data Leakage

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

2. K Nearest Neighbor

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

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

2.4.1 Applying KNN brute force on BOW, SET 1

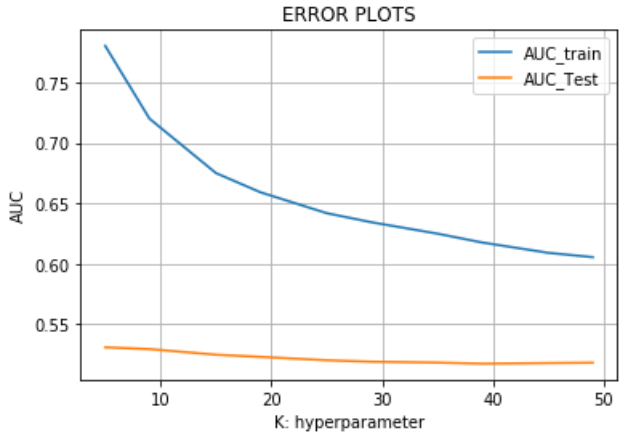
In [37]:

```
def batch_predict(clf, data):
    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000
    for i in range(0, tr_loop, 1000):
        y_data_pred.extend(clf.predict_proba(data[i:i+1000])[:,1])
    y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])
    return y_data_pred
```

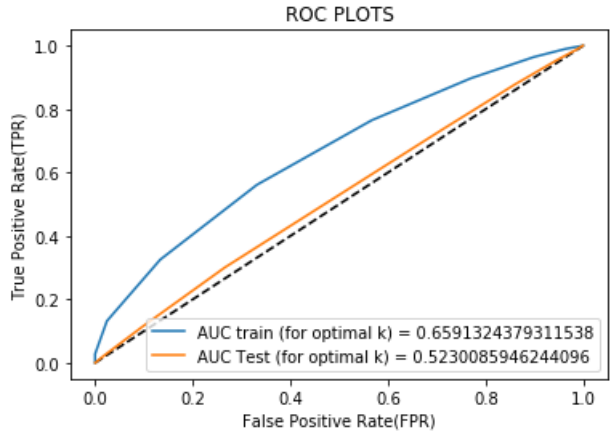
In [38]:

```
k_range = [5,9,15,19,25,29,35,39,45,49]
acc_val = []
auc_scores_train = []
auc_scores_Test = []

for i in k_range:
    knn = KNeighborsClassifier(n_neighbors=i, n_jobs=-1,algorithm='brute')
    knn.fit(X1_combo, y1)
    pred = knn.predict(X1_Test)
    acc = accuracy_score(y_Test, pred, normalize=True) * float(100)
    acc_val.append(acc)
    #print('\nCV accuracy for k = %d is %d%%' % (i, acc))
    train_pred = batch_predict(knn, X1_combo.tocsr())
    a_fpr_train,a_tpr_train,c = roc_curve(y1, train_pred)
    auc_scores_train.append(auc(a_fpr_train, a_tpr_train))

    Test_pred = batch_predict(knn, X1_Test.tocsr())
    a_fpr_Test,a_tpr_Test,c = roc_curve(y_Test, Test_pred)
    auc_scores_Test.append(auc(a_fpr_Test, a_tpr_Test))

# Performance of model on Train data and Test data for each hyper parameter.
plt.plot(k_range, auc_scores_train, label='AUC_train')
plt.gca()
plt.plot(k_range, auc_scores_Test, label='AUC_Test')
plt.gca()
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()


ma=max(acc_val)
ma_ind = np.where(acc_val == ma)
ma_i = int(ma_ind[0][0])
k_opt=k_range[ma_i-1]



knn_opt = KNeighborsClassifier(n_neighbors = k_opt, n_jobs=-1,algorithm='brute')
knn_opt.fit(X1_combo, y1)
pred = knn.predict(X1_Test)
acc = accuracy_score(y_Test, pred, normalize=True) * float(100)
print('\nTest accuracy for k = {0} is {1}%'.format(k_opt,ma))

y_train_pred = batch_predict(knn_opt, X1_combo.tocsr())
y_Test_pred = batch_predict(knn_opt, X1_Test.tocsr())

# https://qiita.com/bmj0114/items/460424c110a8ce22d945

fpr_train, tpr_train, thresholds = roc_curve(y1, y_train_pred)
fpr_Test, tpr_Test, thresholds = roc_curve(y_Test, y_Test_pred)


#ROC plot
plt.plot([0,1],[0,1],'k--')
plt.plot(fpr_train, tpr_train, label="AUC train (for optimal k) = "+str(auc(fpr_train, tpr_train))
)
plt.plot(fpr_Test, tpr_Test, label="AUC Test (for optimal k) = "+str(auc(fpr_Test, tpr_Test)))
plt.legend()
plt.xlabel("False Positive Rate(FPR)")
plt.ylabel("True Positive Rate(TPR)")
plt.title("ROC PLOTS")
plt.show()
print("AUC train (for optimal k) =", auc(fpr_train, tpr_train))
print("AUC Test (for optimal k) =", auc(fpr_Test, tpr_Test))
print("="*115)

a1 = auc(fpr_Test, tpr_Test)
p1 = k_opt
pred0 = knn_opt.predict(X1_combo)
pred2 = knn_opt.predict(X1_Test)

# http://www.tarekatwan.com/index.php/2017/12/how-to-plot-a-confusion-matrix-in-python/
```

```
#----------------------Confusion matrix for BOW Train Data-------------------------------------
---------
plt.clf()
cm0 = confusion_matrix(y1, pred0)
plt.imshow(cm0, interpolation='nearest', cmap=plt.cm.Wistia)
classNames = ['0', '1']
plt.title('Project is APPROVED or NOT Confusion Matrix - Train Data')
plt.ylabel('True label')
plt.xlabel('Predicted label')
tick_marks = np.arange(len(classNames))
plt.xticks(tick_marks, classNames, rotation=0)
plt.yticks(tick_marks, classNames)
s = [['TN','FP'], ['FN', 'TP']]
for i in range(2):
    for j in range(2):
        plt.text(j,i, str(s[i][j])+" = "+str(cm0[i][j]))
plt.show()

#----------------------Confusion matrix for BOW Test Data-------------------------------------
--------
plt.clf()
cm2 = confusion_matrix(y_Test, pred2)
plt.imshow(cm2, interpolation='nearest', cmap=plt.cm.Wistia)
classNames = ['0', '1']
plt.title('Project is APPROVED or NOT Confusion Matrix - Test Data')
plt.ylabel('True label')
plt.xlabel('Predicted label')
tick_marks = np.arange(len(classNames))
plt.xticks(tick_marks, classNames, rotation=0)
plt.yticks(tick_marks, classNames)
s = [['TN','FP'], ['FN', 'TP']]
for i in range(2):
    for j in range(2):
        plt.text(j,i, str(s[i][j])+" = "+str(cm2[i][j]))
plt.show()
```

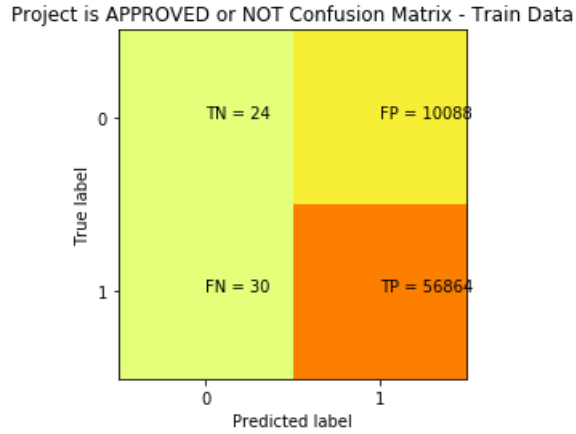


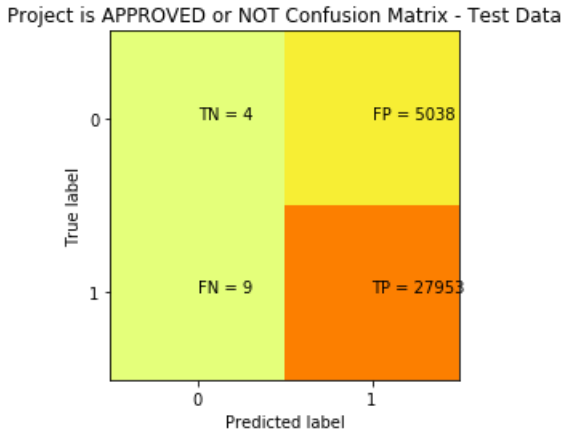

```
Test accuracy for k = 19 is 84.72306387104594%
```




```
AUC train (for optimal k) = 0.6591324379311538
AUC Test (for optimal k) = 0.5230085946244096
```

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





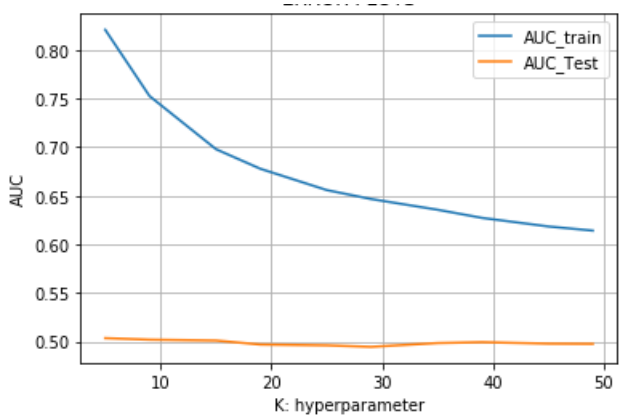
2.4.2 Applying KNN brute force on TFIDF, SET 2

In [39]:

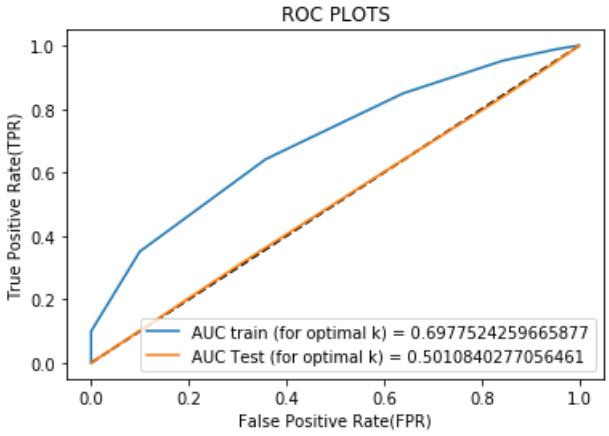
```
k_range = [5,9,15,19,25,29,35,39,45,49]
acc_val = []
auc_scores_train = []
auc_scores_Test = []

for i in k_range:
    knn = KNeighborsClassifier(n_neighbors=i, n_jobs=-1,algorithm='brute')
    knn.fit(X4_combo, y1)
    pred = knn.predict(X4_Test)
    acc = accuracy_score(y_Test, pred, normalize=True) * float(100)
    acc_val.append(acc)
    #print('\nCV accuracy for k = %d is %d%%' % (i, acc))
    train_pred = batch_predict(knn, X4_combo.tocsr())
    a_fpr_train,a_tpr_train,c = roc_curve(y1, train_pred)
    auc_scores_train.append(auc(a_fpr_train, a_tpr_train))

    Test_pred = batch_predict(knn, X4_Test.tocsr())
    a_fpr_Test,a_tpr_Test,c = roc_curve(y_Test, Test_pred)
    auc_scores_Test.append(auc(a_fpr_Test, a_tpr_Test))

# Performance of model on Train data and Test data for each hyper parameter.
plt.plot(k_range, auc_scores_train, label='AUC_train')
plt.gca()
plt.plot(k_range, auc_scores_Test, label='AUC_Test')
plt.gca()
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```

```
ma=max(acc_val)
ma_ind = np.where(acc_val == ma)
ma_i = int(ma_ind[0][0])
k_opt=k_range[ma_i-1]

knn_opt = KNeighborsClassifier(n_neighbors = k_opt, n_jobs=-1,algorithm='brute')
knn_opt.fit(X4_combo, y1)
pred = knn.predict(X4_Test)
acc = accuracy_score(y_Test, pred, normalize=True) * float(100)
print('\nTest accuracy for k = {0} is {1}%'.format(k_opt,ma))

y_train_pred = batch_predict(knn_opt, X4_combo.tocsr())
y_Test_pred = batch_predict(knn_opt, X4_Test.tocsr())

# https://qiita.com/bmj0114/items/460424c110a8ce22d945
fpr_train, tpr_train, thresholds = roc_curve(y1, y_train_pred)
fpr_Test, tpr_Test, thresholds = roc_curve(y_Test, y_Test_pred)

#ROC plot
plt.plot([0,1],[0,1],'k--')
plt.plot(fpr_train, tpr_train, label="AUC train (for optimal k) = "+str(auc(fpr_train, tpr_train))
)
plt.plot(fpr_Test, tpr_Test, label="AUC Test (for optimal k) = "+str(auc(fpr_Test, tpr_Test)))
plt.legend()
plt.xlabel("False Positive Rate(FPR)")
plt.ylabel("True Positive Rate(TPR)")
plt.title("ROC PLOTS")
plt.show()
print("AUC train (for optimal k) =", auc(fpr_train, tpr_train))
print("AUC Test (for optimal k) =", auc(fpr_Test, tpr_Test))
print("="*115)

a2 = auc(fpr_Test, tpr_Test)
p2 = k_opt
pred0 = knn_opt.predict(X4_combo)
pred2 = knn_opt.predict(X4_Test)

# http://www.tarekatwan.com/index.php/2017/12/how-to-plot-a-confusion-matrix-in-python/
#------------------------Confusion matrix for TFIDF Train Data-------------------------------------
-----------
plt.clf()
cm0 = confusion_matrix(y1, pred0)
plt.imshow(cm0, interpolation='nearest', cmap=plt.cm.Wistia)
classNames = ['0', '1']
plt.title('Project is APPROVED or NOT Confusion Matrix - Train Data')
plt.ylabel('True label')
plt.xlabel('Predicted label')
tick_marks = np.arange(len(classNames))
plt.xticks(tick_marks, classNames, rotation=0)
plt.yticks(tick_marks, classNames)
s = [['TN','FP'], ['FN', 'TP']]
for i in range(2):
    for j in range(2):
        plt.text(j,i, str(s[i][j])+" = "+str(cm0[i][j]))
plt.show()

#------------------------Confusion matrix for TFIDF Test Data-------------------------------------
----------
plt.clf()
cm2 = confusion_matrix(y_Test, pred2)
plt.imshow(cm2, interpolation='nearest', cmap=plt.cm.Wistia)
classNames = ['0', '1']
plt.title('Project is APPROVED or NOT Confusion Matrix - Test Data')
plt.ylabel('True label')
plt.xlabel('Predicted label')
tick_marks = np.arange(len(classNames))
plt.xticks(tick_marks, classNames, rotation=0)
plt.yticks(tick_marks, classNames)
s = [['TN','FP'], ['FN', 'TP']]
for i in range(2):
    for j in range(2):
        plt.text(j,i, str(s[i][j])+" = "+str(cm2[i][j]))
plt.show()
```

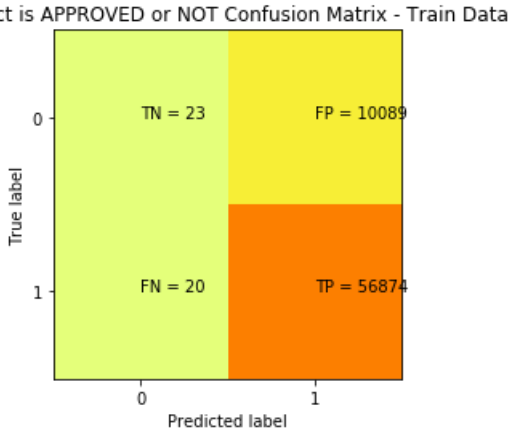
ERROR PLOTS

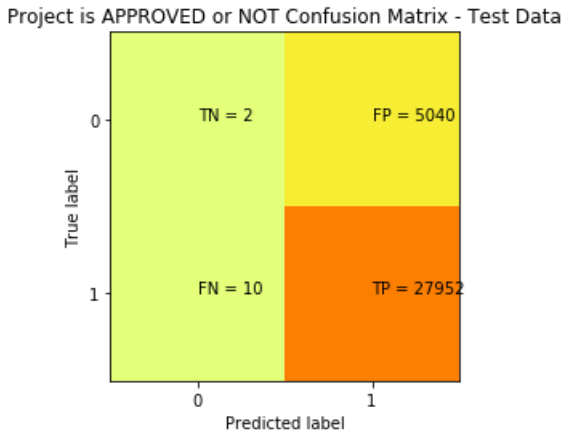



```
Test accuracy for k = 15 is 84.72306387104594%
```




```
AUC train (for optimal k) = 0.6977524259665877
AUC Test (for optimal k) = 0.5010840277056461
================================================================================================================
```



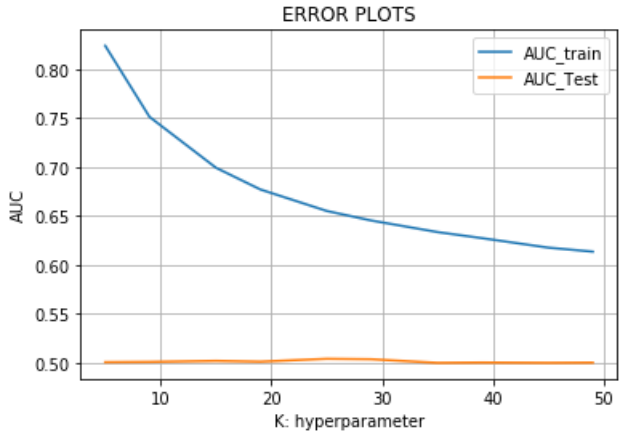


2.4.3 Applying KNN brute force on AVG W2V, SET 3

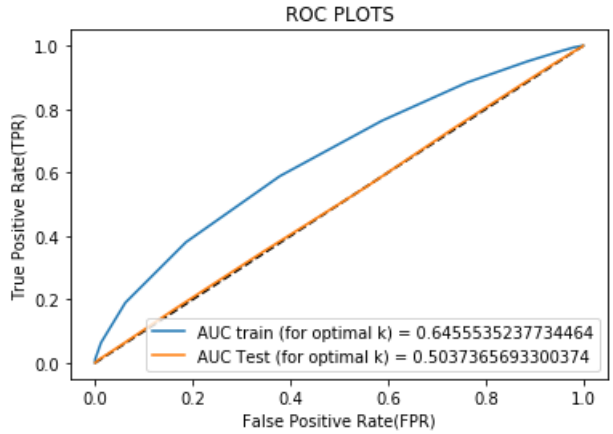
In [40]:

```
k_range = [5,9,15,19,25,29,35,39,45,49]
acc_val = []
auc_scores_train = []
auc_scores_Test = []

for i in k_range:
    knn = KNeighborsClassifier(n_neighbors=i, n_jobs=-1,algorithm='brute')
    knn.fit(X3_combo, y1)
    pred = knn.predict(X3_Test)
    acc = accuracy_score(y_Test, pred, normalize=True) * float(100)
    acc_val.append(acc)
    #print('\nCV accuracy for k = %d is %d%%' % (i, acc))
    train_pred = batch_predict(knn, X3_combo.tocsr())
    a_fpr_train,a_tpr_train,c = roc_curve(y1, train_pred)
    auc_scores_train.append(auc(a_fpr_train, a_tpr_train))

    Test_pred = batch_predict(knn, X3_Test.tocsr())
    a_fpr_Test,a_tpr_Test,c = roc_curve(y_Test, Test_pred)
    auc_scores_Test.append(auc(a_fpr_Test, a_tpr_Test))

# Performance of model on Train data and Test data for each hyper parameter.
plt.plot(k_range, auc_scores_train, label='AUC_train')
plt.gca()
plt.plot(k_range, auc_scores_Test, label='AUC_Test')
plt.gca()
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()


ma=max(acc_val)
ma_ind = np.where(acc_val == ma)
ma_i = int(ma_ind[0][0])
k_opt=k_range[ma_i-1]

knn_opt = KNeighborsClassifier(n_neighbors = k_opt, n_jobs=-1,algorithm='brute')
knn_opt.fit(X3_combo, y1)
pred = knn.predict(X3_Test)
acc = accuracy_score(y_Test, pred, normalize=True) * float(100)
print('\nTest accuracy for k = {0} is {1}%'.format(k_opt,ma))

y_train_pred = batch_predict(knn_opt, X3_combo.tocsr())
y_Test_pred = batch_predict(knn_opt, X3_Test.tocsr())

# https://qiita.com/bmj0114/items/460424c110a8ce22d945
fpr_train, tpr_train, thresholds = roc_curve(y1, y_train_pred)
fpr_Test, tpr_Test, thresholds = roc_curve(y_Test, y_Test_pred)

#ROC plot
plt.plot([0,1],[0,1],'k--')
plt.plot(fpr_train, tpr_train, label="AUC train (for optimal k) = "+str(auc(fpr_train, tpr_train))
)
plt.plot(fpr_Test, tpr_Test, label="AUC Test (for optimal k) = "+str(auc(fpr_Test, tpr_Test)))
plt.legend()
plt.xlabel("False Positive Rate(FPR)")
plt.ylabel("True Positive Rate(TPR)")
plt.title("ROC PLOTS")
plt.show()
print("AUC train (for optimal k) =", auc(fpr_train, tpr_train))
print("AUC Test (for optimal k) =", auc(fpr_Test, tpr_Test))
print("="*115)

a3 = auc(fpr_Test, tpr_Test)
p3 = k_opt
pred0 = knn_opt.predict(X3_combo)
pred2 = knn_opt.predict(X3_Test)
```

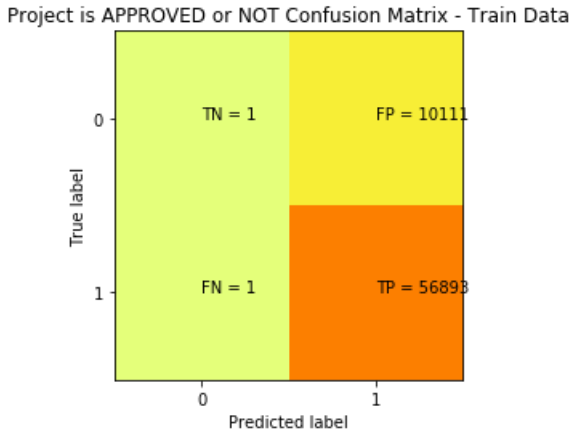
```
# http://www.tarekatwan.com/index.php/2017/12/how-to-plot-a-confusion-matrix-in-python/
#------------------------Confusion matrix for AVG W2V Train Data-----------------------------------
-------------
plt.clf()
cm0 = confusion_matrix(y1, pred0)
plt.imshow(cm0, interpolation='nearest', cmap=plt.cm.Wistia)
classNames = ['0', '1']
plt.title('Project is APPROVED or NOT Confusion Matrix - Train Data')
plt.ylabel('True label')
plt.xlabel('Predicted label')
tick_marks = np.arange(len(classNames))
plt.xticks(tick_marks, classNames, rotation=0)
plt.yticks(tick_marks, classNames)
s = [['TN','FP'], ['FN', 'TP']]
for i in range(2):
    for j in range(2):
        plt.text(j,i, str(s[i][j])+" = "+str(cm0[i][j]))
plt.show()

#------------------------Confusion matrix for AVG W2V Test Data-----------------------------------
------------
plt.clf()
cm2 = confusion_matrix(y_Test, pred2)
plt.imshow(cm2, interpolation='nearest', cmap=plt.cm.Wistia)
classNames = ['0', '1']
plt.title('Project is APPROVED or NOT Confusion Matrix - Test Data')
plt.ylabel('True label')
plt.xlabel('Predicted label')
tick_marks = np.arange(len(classNames))
plt.xticks(tick_marks, classNames, rotation=0)
plt.yticks(tick_marks, classNames)
s = [['TN','FP'], ['FN', 'TP']]
for i in range(2):
    for j in range(2):
        plt.text(j,i, str(s[i][j])+" = "+str(cm2[i][j]))
plt.show()
```

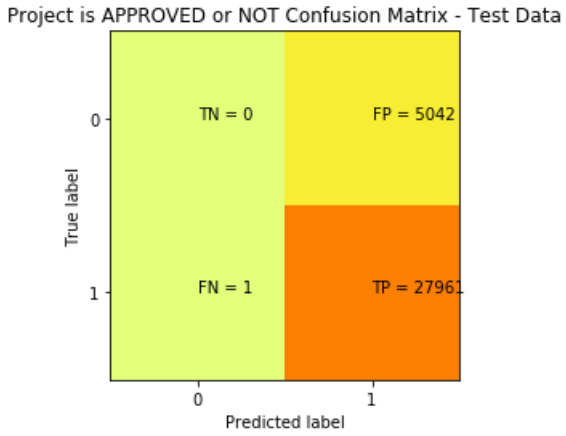



```
Test accuracy for k = 29 is 84.72306387104594%
```



```
AUC train (for optimal k) = 0.6455535237734464
AUC Test (for optimal k) = 0.5037365693300374
================================================================================================================
============
```



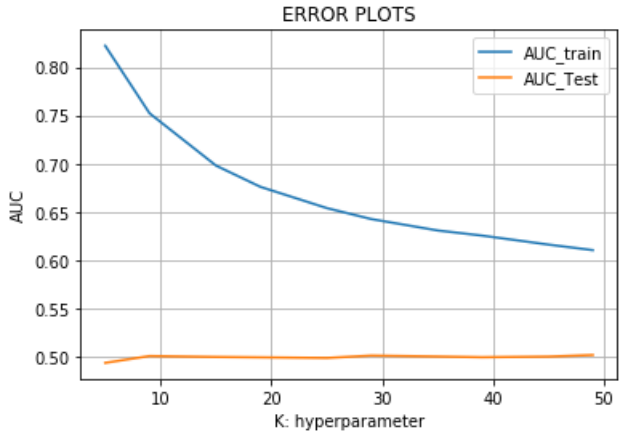


2.4.4 Applying KNN brute force on TFIDF W2V, SET 4

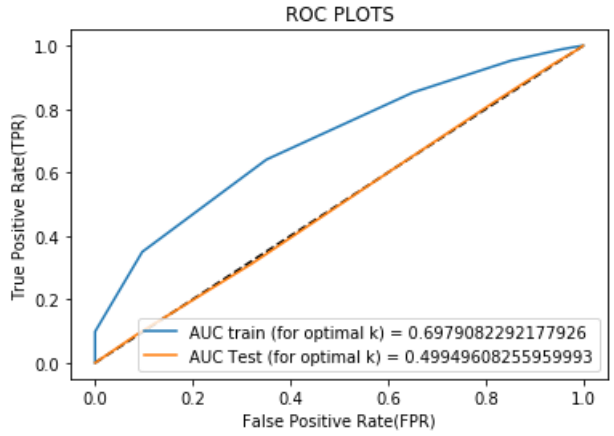
In [41]:

```
k_range = [5,9,15,19,25,29,35,39,45,49]
acc_val = []
auc_scores_train = []
auc_scores_Test = []

for i in k_range:
    knn = KNeighborsClassifier(n_neighbors=i, n_jobs=-1,algorithm='brute')
    knn.fit(X2_combo, y1)
    pred = knn.predict(X2_Test)
    acc = accuracy_score(y_Test, pred, normalize=True) * float(100)
    acc_val.append(acc)
    #print('\nCV accuracy for k = %d is %d%%' % (i, acc))
    train_pred = batch_predict(knn, X2_combo.tocsr())
    a_fpr_train,a_tpr_train,c = roc_curve(y1, train_pred)
    auc_scores_train.append(auc(a_fpr_train, a_tpr_train))

    Test_pred = batch_predict(knn, X2_Test.tocsr())
    a_fpr_Test,a_tpr_Test,c = roc_curve(y_Test, Test_pred)
    auc_scores_Test.append(auc(a_fpr_Test, a_tpr_Test))

# Performance of model on Train data and Test data for each hyper parameter.
plt.plot(k_range, auc_scores_train, label='AUC_train')
plt.gca()
plt.plot(k_range, auc_scores_Test, label='AUC_Test')
plt.gca()
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
```

```
plt.show()


ma=max(acc_val)
ma_ind = np.where(acc_val == ma)
ma_i = int(ma_ind[0][0])
k_opt=k_range[ma_i-1]


knn_opt = KNeighborsClassifier(n_neighbors = k_opt, n_jobs=-1,algorithm='brute')
knn_opt.fit(X2_combo, y1)
pred = knn.predict(X2_Test)
acc = accuracy_score(y_Test, pred, normalize=True) * float(100)
print('\nTest accuracy for k = {0} is {1}%'.format(k_opt,ma))

y_train_pred = batch_predict(knn_opt, X2_combo.tocsr())
y_Test_pred = batch_predict(knn_opt, X2_Test.tocsr())

# https://qiita.com/bmj0114/items/460424c110a8ce22d945
fpr_train, tpr_train, thresholds = roc_curve(y1, y_train_pred)
fpr_Test, tpr_Test, thresholds = roc_curve(y_Test, y_Test_pred)

#ROC plot
plt.plot([0,1],[0,1],'k--')
plt.plot(fpr_train, tpr_train, label="AUC train (for optimal k) = "+str(auc(fpr_train, tpr_train))
)
plt.plot(fpr_Test, tpr_Test, label="AUC Test (for optimal k) = "+str(auc(fpr_Test, tpr_Test)))
plt.legend()
plt.xlabel("False Positive Rate(FPR)")
plt.ylabel("True Positive Rate(TPR)")
plt.title("ROC PLOTS")
plt.show()
print("AUC train (for optimal k) =", auc(fpr_train, tpr_train))
print("AUC Test (for optimal k) =", auc(fpr_Test, tpr_Test))
print("="*115)

a4 = auc(fpr_Test, tpr_Test)
p4 = k_opt
pred0 = knn_opt.predict(X2_combo)
pred2 = knn_opt.predict(X2_Test)

# http://www.tarekatwan.com/index.php/2017/12/how-to-plot-a-confusion-matrix-in-python/
#------------------------Confusion matrix for TFIDF W2V Train Data--------------------------------
---------------
plt.clf()
cm0 = confusion_matrix(y1, pred0)
plt.imshow(cm0, interpolation='nearest', cmap=plt.cm.Wistia)
classNames = ['0', '1']
plt.title('Project is APPROVED or NOT Confusion Matrix - Train Data')
plt.ylabel('True label')
plt.xlabel('Predicted label')
tick_marks = np.arange(len(classNames))
plt.xticks(tick_marks, classNames, rotation=0)
plt.yticks(tick_marks, classNames)
s = [['TN','FP'], ['FN', 'TP']]
for i in range(2):
    for j in range(2):
        plt.text(j,i, str(s[i][j])+" = "+str(cm0[i][j]))
plt.show()

#------------------------Confusion matrix for TFIDF W2V Test Data--------------------------------
--------------
plt.clf()
cm2 = confusion_matrix(y_Test, pred2)
plt.imshow(cm2, interpolation='nearest', cmap=plt.cm.Wistia)
classNames = ['0', '1']
plt.title('Project is APPROVED or NOT Confusion Matrix - Test Data')
plt.ylabel('True label')
plt.xlabel('Predicted label')
tick_marks = np.arange(len(classNames))
plt.xticks(tick_marks, classNames, rotation=0)
plt.yticks(tick_marks, classNames)
s = [['TN','FP'], ['FN', 'TP']]
for i in range(2):
    for j in range(2):
        plt.text(j,i, str(s[i][j])+" = "+str(cm2[i][j]))
plt.show()
```

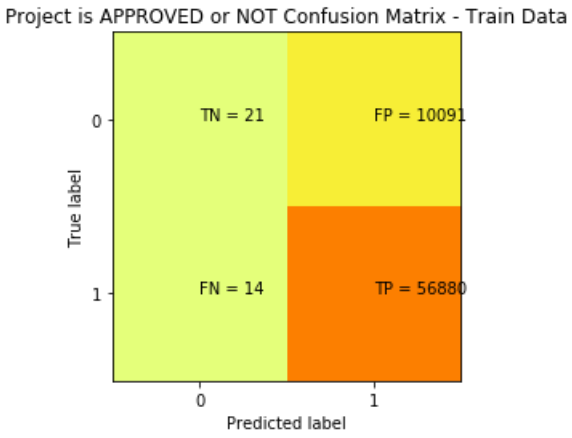


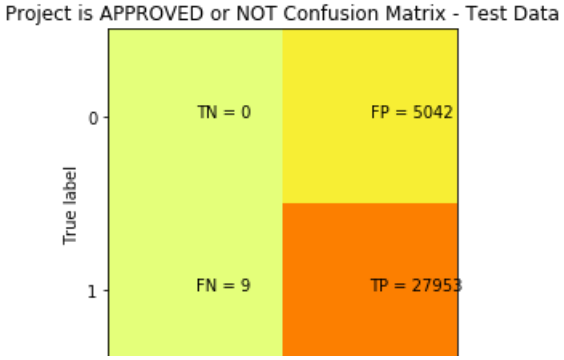

```
Test accuracy for k = 15 is 84.72306387104594%
```




```
AUC train (for optimal k) = 0.6979082292177926
AUC Test (for optimal k) = 0.49949608255959993
```

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





2.5 Feature selection with `SelectKBest`

Selecting the top 2000 useful/relevant features

In [42]:

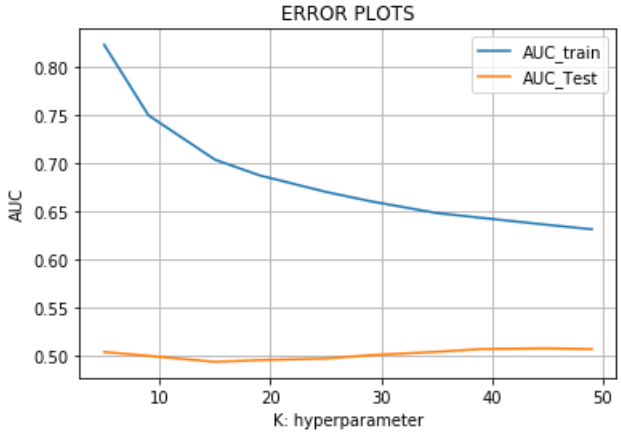
```
# https://stats.stackexchange.com/questions/341332/how-to-scale-for-selectkbest-for-feature-selection
X5_train_K = SelectKBest(f_classif, k=2000).fit(X5_combo, y1)
X5_train_new = X5_train_K.transform(X5_combo)
X5_Test_new = X5_train_K.transform(X5_Test)
print(X5_train_new.shape)
print(X5_Test_new.shape)
```

```
(67006, 2000)
(33004, 2000)
```

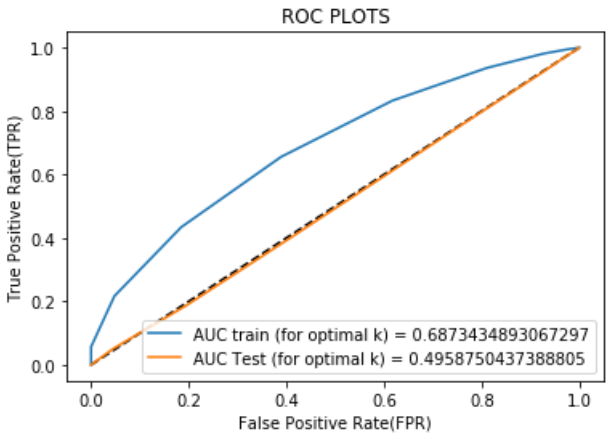
In [43]:

```
k_range = [5,9,15,19,25,29,35,39,45,49]
acc_val = []
auc_scores_train = []
auc_scores_Test = []

for i in k_range:
    knn = KNeighborsClassifier(n_neighbors=i, n_jobs=-1,algorithm='brute')
    knn.fit(X5_train_new, y1)
    pred = knn.predict(X5_Test_new)
    acc = accuracy_score(y_Test, pred, normalize=True) * float(100)
    acc_val.append(acc)
    #print('\nCV accuracy for k = %d is %d%%' % (i, acc))
    train_pred = batch_predict(knn, X5_train_new.tocsr())
    a_fpr_train,a_tpr_train,c = roc_curve(y1, train_pred)
    auc_scores_train.append(auc(a_fpr_train, a_tpr_train))

    Test_pred = batch_predict(knn, X5_Test_new.tocsr())
    a_fpr_Test,a_tpr_Test,c = roc_curve(y_Test, Test_pred)
    auc_scores_Test.append(auc(a_fpr_Test, a_tpr_Test))

# Performance of model on Train data and Test data for each hyper parameter.
plt.plot(k_range, auc_scores_train, label='AUC_train')
plt.gca()
plt.plot(k_range, auc_scores_Test, label='AUC_Test')
plt.gca()
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()


ma=max(acc_val)
ma_ind = np.where(acc_val == ma)
ma_i = int(ma_ind[0][0])
k_opt=k_range[ma_i-1]


knn_opt = KNeighborsClassifier(n_neighbors = k_opt, n_jobs=-1,algorithm='brute')
knn_opt.fit(X5_train_new, y1)
pred = knn.predict(X5_Test_new)
acc = accuracy_score(y_Test, pred, normalize=True) * float(100)
print('\nTest accuracy for k = {0} is {1}%'.format(k_opt,ma))

y_train_pred = batch_predict(knn_opt, X5_train_new.tocsr())
y_Test_pred = batch_predict(knn_opt, X5_Test_new.tocsr())

# https://qiita.com/bmj0114/items/460424c110a8ce22d945
```

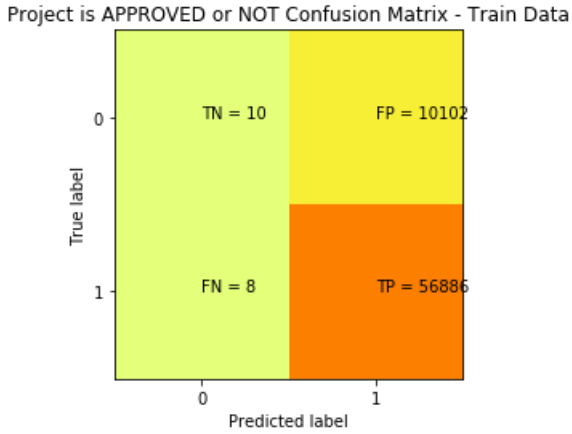
```
fpr_train, tpr_train, thresholds = roc_curve(y1, y_train_pred)
fpr_Test, tpr_Test, thresholds = roc_curve(y_Test, y_Test_pred)

#ROC plot
plt.plot([0,1],[0,1],'k--')
plt.plot(fpr_train, tpr_train, label="AUC train (for optimal k) = "+str(auc(fpr_train, tpr_train))
)
plt.plot(fpr_Test, tpr_Test, label="AUC Test (for optimal k) = "+str(auc(fpr_Test, tpr_Test)))
plt.legend()
plt.xlabel("False Positive Rate(FPR)")
plt.ylabel("True Positive Rate(TPR)")
plt.title("ROC PLOTS")
plt.show()
print("AUC train (for optimal k) =", auc(fpr_train, tpr_train))
print("AUC Test (for optimal k) =", auc(fpr_Test, tpr_Test))
print("="*115)

a5 = auc(fpr_Test, tpr_Test)
p5 = k_opt
pred0 = knn_opt.predict(X5_train_new)
pred2 = knn_opt.predict(X5_Test_new)

# http://www.tarekatwan.com/index.php/2017/12/how-to-plot-a-confusion-matrix-in-python/
#------------------------Confusion matrix for TFIDF Train Data-------------------------------------
-----------
plt.clf()
cm0 = confusion_matrix(y1, pred0)
plt.imshow(cm0, interpolation='nearest', cmap=plt.cm.Wistia)
classNames = ['0', '1']
plt.title('Project is APPROVED or NOT Confusion Matrix - Train Data')
plt.ylabel('True label')
plt.xlabel('Predicted label')
tick_marks = np.arange(len(classNames))
plt.xticks(tick_marks, classNames, rotation=0)
plt.yticks(tick_marks, classNames)
s = [['TN','FP'], ['FN', 'TP']]
for i in range(2):
    for j in range(2):
        plt.text(j,i, str(s[i][j])+" = "+str(cm0[i][j]))
plt.show()

#------------------------Confusion matrix for TFIDF Test Data-------------------------------------
----------
plt.clf()
cm2 = confusion_matrix(y_Test, pred2)
plt.imshow(cm2, interpolation='nearest', cmap=plt.cm.Wistia)
classNames = ['0', '1']
plt.title('Project is APPROVED or NOT Confusion Matrix - Test Data')
plt.ylabel('True label')
plt.xlabel('Predicted label')
tick_marks = np.arange(len(classNames))
plt.xticks(tick_marks, classNames, rotation=0)
plt.yticks(tick_marks, classNames)
s = [['TN','FP'], ['FN', 'TP']]
for i in range(2):
    for j in range(2):
        plt.text(j,i, str(s[i][j])+" = "+str(cm2[i][j]))
plt.show()
```

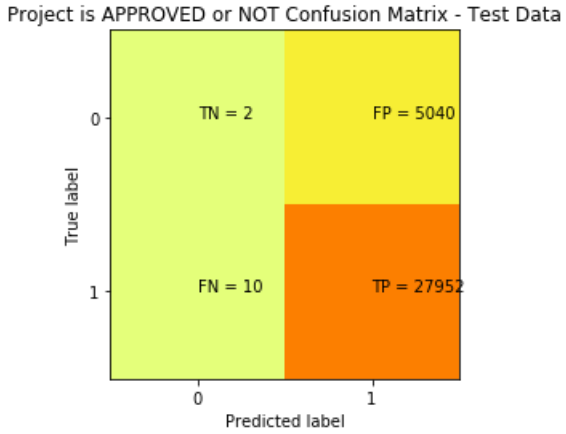


```
Test accuracy for k = 19 is 84.72609380681129%
```




```
AUC train (for optimal k) = 0.6873434893067297
AUC Test (for optimal k) = 0.4958750437388805
================================================================================================================
============
```





3. Conclusions

In [44]:

```
# Please compare all your models using Prettytable library
pt = PrettyTable()
pt.field_names= ("Vectorizer", "Model", "HyperParameter", "AUC(cv)")
pt.add_row(["BOW", "brute",p1, a1])
pt.add_row(["Tf-Idf", "brute", p2, a2])
pt.add_row(["AVG W2V", "brute",p3, a3])
pt.add_row(["TFIDF W2V", "brute", p4, a4])
pt.add_row(["Top 2000 features of Tf-Idf", "brute", p5, a5])
```

```
print(pt)
```

```
+-----------------------------+-------+----------------+---------------------+
|          Vectorizer         | Model | HyperParameter |       AUC(cv)       |
+-----------------------------+-------+----------------+---------------------+
|             BOW             | brute |       19       |  0.5230085946244096 |
|            Tf-Idf           | brute |       15       |  0.5010840277056461 |
|           AVG W2V           | brute |       29       |  0.5037365693300374 |
|          TFIDF W2V          | brute |       15       | 0.49949608255959993 |
| Top 2000 features of Tf-Idf | brute |       19       |  0.4958750437388805 |
+-----------------------------+-------+----------------+---------------------+
```