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`

Feature	Description
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 (https://en.wikipedia.org/wiki/List_of_U.S._state_abbreviations#Postal_codes)). **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!</code`
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 [0]:

```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from nltk.stem.porter import PorterStemmer
import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer
from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle
from tqdm import tqdm
import os
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.model_selection import GridSearchCV
from sklearn.feature_selection import SelectKBest,f_classif
from prettytable import PrettyTable
import pdb
```

1.1 Reading Data

In [0]:
```
Project_data = pd.read_csv('train_data25K.csv')
Resource_data = pd.read_csv('resources.csv')
print(Project_data.shape)
print(Resource_data.shape)
```

```
(25000, 17)
(1541272, 4)
```

In [0]:
```
# how to replace elements in list python: https://stackoverflow.com/a/2582163/4084039
cols = ['Date' if x=='project_submitted_datetime' else x for x in list(Project_data.columns)]
#sort dataframe based on time pandas python: https://stackoverflow.com/a/49702492/4084039
Project_data['Date'] = pd.to_datetime(Project_data['project_submitted_datetime'])
Project_data.drop('project_submitted_datetime', axis=1, inplace=True)
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[77]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_category	project_subject_categories	project
3287	159755	p147002	6ada7036aeb258d3653589d1f2a5b815	Mrs.	CA	2016-01-05 02:02:00	Grades 3-5	Literacy & Language, Special Needs	
19437	146532	p024903	55f60249d65840ee198285acdc455838	Mrs.	CA	2016-01-05 02:57:00	Grades 3-5	Math & Science, Literacy & Language	Health

1.2 preprocessing of `project_subject_categories`

In [0]:

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

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

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

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]))

```

1.3 preprocessing of `project_subject_subcategories`

In [0]:

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

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

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

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

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

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

In [0]:

```
y = Project_data['project_is_approved'].values
Project_data.drop(['project_is_approved'], axis=1, inplace=True)
X = Project_data
```

In [0]:
```
#Spliting the Dataset into three Train,CV and Test
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 the X_Train data is {0} and Y_Train data is: {1}'.format(X_Train.shape,Y_Train.shape[0]))
print('Shape of the X_CV data is    {0} and Y_CV data is   : {1}'.format(X_CV.shape,Y_CV.shape[0]))
print('Shape of the X_Test data is  {0} and Y_Test data is : {1}'.format(X_Test.shape,Y_Test.shape[0]))
```

```
Shape of the X_Train data is (11222, 16) and Y_Train data is: 11222
Shape of the X_CV data is    (5528, 16) and Y_CV data is   : 5528
Shape of the X_Test data is  (8250, 16) and Y_Test data is : 8250
```

1.3 Text preprocessing

In [0]:
```
# 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 [0]:

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

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

In [0]:

```python
# 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 [0]:

```
# Combining all the above stundents
# tqdm is for printing the status bar

#----------------------------------PreProcessing of Essays in Train data set-------------------------------------------------
preprocessed_essays_Train = []
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.append(sent.lower().strip())
# pdb.set_trace()

#----------------------------------PreProcessing of Essays in CV data set-------------------------------------------------
preprocessed_essays_CV = []
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.append(sent.lower().strip())
# pdb.set_trace()

#----------------------------------PreProcessing of Essays in Test data set-------------------------------------------------
preprocessed_essays_Test = []
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.append(sent.lower().strip())
# pdb.set_trace()
```

```
100%|██████████| 11222/11222 [00:05<00:00, 2018.50it/s]
100%|██████████| 5528/5528 [00:02<00:00, 2027.93it/s]
```

```
100%|██████████| 8250/8250 [00:04<00:00, 2043.56it/s]
```

1.4 Preprocessing of `project_title`

In [0]:

```
# Combining all the above stundents
# tqdm is for printing the status bar

#----------------------------------PreProcessing of Project Title in Train data set-------------------------------------
preprocessed_titles_Train = []
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.append(sent.lower().strip())
# pdb.set_trace()

#----------------------------------PreProcessing of Project Title in CV data set-------------------------------------
preprocessed_titles_CV = []
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.append(sent.lower().strip())
# pdb.set_trace()

#----------------------------------PreProcessing of Project Title in Test data set-------------------------------------
preprocessed_titles_Test = []
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.append(sent.lower().strip())
# pdb.set_trace()
```

```
100%|██████████| 11222/11222 [00:00<00:00, 44587.37it/s]
100%|██████████| 5528/5528 [00:00<00:00, 43345.50it/s]
100%|██████████| 8250/8250 [00:00<00:00, 43852.51it/s]
```

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

In [0]:

```
#------------------------Vectorizing categorical data for Train,CV and Test-----------------------------------------------

# we use count vectorizer to convert the values into one hot encoding
vectorizer_cat = CountVectorizer(vocabulary=list(sorted_cat_dict.keys()), lowercase=False, binary=True)
categories_one_hot_Train = vectorizer_cat.fit_transform(X_Train['clean_categories'].values)
categories_one_hot_CV = vectorizer_cat.transform(X_CV['clean_categories'].values)
categories_one_hot_Test = vectorizer_cat.transform(X_Test['clean_categories'].values)
print(vectorizer_cat.get_feature_names())
print("-"*120)
print('Shape of Train dataset matrix after one hot encoding is: {0}'.format(categories_one_hot_Train.shape))
print('Shape of CV dataset matrix after one hot encoding is: {0}'.format(categories_one_hot_CV.shape))
print('Shape of Test dataset matrix after one hot encoding is: {0}'.format(categories_one_hot_Test.shape))
```

```
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'SpecialNeeds', 'Health_Sports', 'Math_Science', 'Li
teracy_Language']
------------------------------------------------------------------------------------------------------------------------
Shape of Train dataset matrix after one hot encoding is: (11222, 9)
Shape of CV dataset matrix after one hot encoding is: (5528, 9)
Shape of Test dataset matrix after one hot encoding is: (8250, 9)
```

In [0]:

```
#-------------------------Vectorizing categorical data for Train,CV and Test-----------------------------------------------

# we use count vectorizer to convert the values into one hot encoding
vectorizer_sub_cat = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()), lowercase=False, binary=True)
sub_categories_one_hot_Train = vectorizer_sub_cat.fit_transform(X_Train['clean_subcategories'].values)
sub_categories_one_hot_CV = vectorizer_sub_cat.transform(X_CV['clean_subcategories'].values)
sub_categories_one_hot_Test = vectorizer_sub_cat.transform(X_Test['clean_subcategories'].values)
print(vectorizer_sub_cat.get_feature_names())
print("-"*120)
print('Shape of Train dataset matrix after one hot encoding is: {0}'.format(sub_categories_one_hot_Train.shape))
print('Shape of CV dataset matrix after one hot encoding is: {0}'.format(sub_categories_one_hot_CV.shape))
print('Shape of Test dataset matrix after one hot encoding is: {0}'.format(sub_categories_one_hot_Test.shape))
```

```
['Economics', 'CommunityService', 'FinancialLiteracy', 'ParentInvolvement', 'Extracurricular', 'Civics_Government', 'ForeignLangu
ages', 'NutritionEducation', 'Warmth', 'Care_Hunger', 'SocialSciences', 'PerformingArts', 'CharacterEducation', 'TeamSports', 'Ot
her', 'College_CareerPrep', 'Music', 'History_Geography', 'Health_LifeScience', 'ESL', 'EarlyDevelopment', 'Gym_Fitness', 'Enviro
nmentalScience', 'VisualArts', 'Health_Wellness', 'AppliedSciences', 'SpecialNeeds', 'Literature_Writing', 'Mathematics', 'Litera
cy']
------------------------------------------------------------------------------------------------------------------------
Shape of Train dataset matrix after one hot encoding is: (11222, 30)
Shape of CV dataset matrix after one hot encoding is: (5528, 30)
Shape of Test dataset matrix after one hot encoding is: (8250, 30)
```

School State

In [0]:

```
#------------------------Vectorizing categorical data of School state for Train dataset-----------------------------------

school_catogories_Train = list(X_Train['school_state'].values)
school_list_Train = []
for sent in school_catogories_Train:
    school_list_Train.append(sent.lower().strip())
X_Train['school_categories'] = school_list_Train
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_school_Train = Counter()
for word in X_Train['school_categories'].values:
    my_counter_school_Train.update(word.split())

# dict sort by value python: https://stackoverflow.com/a/613218/4084039
school_dict_Train = dict(my_counter_school_Train)
sorted_school_dict_Train = dict(sorted(school_dict_Train.items(), key=lambda kv: kv[1]))

vectorizer_school = CountVectorizer(vocabulary=list(sorted_school_dict_Train.keys()), lowercase=False, binary=True)
vectorizer_school.fit(X_Train['school_categories'].values)
#print(vectorizer.get_feature_names())

school_one_hot_Train = vectorizer_school.transform(X_Train['school_categories'].values)

#------------------------Vectorizing categorical data of School state for CV dataset-----------------------------------

school_catogories_CV = list(X_CV['school_state'].values)
school_list_CV = []
for sent in school_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_school_CV = Counter()
for word in X_CV['school_categories'].values:
    my_counter_school_CV.update(word.split())

# dict sort by value python: https://stackoverflow.com/a/613218/4084039
school_dict_CV = dict(my_counter_school_CV)
sorted_school_dict_CV = dict(sorted(school_dict_CV.items(), key=lambda kv: kv[1]))
school_one_hot_CV = vectorizer_school.transform(X_CV['school_categories'].values)

#------------------------Vectorizing categorical data of School state for Test dataset-----------------------------------

```

```
school_catogories_Test = list(X_Test['school_state'].values)
school_list_Test = []
for sent in school_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_school_Test = Counter()
for word in X_Test['school_categories'].values:
    my_counter_school_Test.update(word.split())

# dict sort by value python: https://stackoverflow.com/a/613218/4084039
school_dict_Test = dict(my_counter_school_Test)
sorted_school_dict_Test = dict(sorted(school_dict_Test.items(), key=lambda kv: kv[1]))
school_one_hot_Test = vectorizer_school.transform(X_Test['school_categories'].values)
print("-"*120)
print('Shape of Train dataset matrix after one hot encoding is: {0}'.format(school_one_hot_Train.shape))
print('Shape of CV dataset matrix after one hot encoding is: {0}'.format(school_one_hot_CV.shape))
print('Shape of Test dataset matrix after one hot encoding is: {0}'.format(school_one_hot_Test.shape))
```

```
------------------------------------------------------------------------------------------------------------------------
Shape of Train dataset matrix after one hot encoding is: (11222, 51)
Shape of CV dataset matrix after one hot encoding is: (5528, 51)
Shape of Test dataset matrix after one hot encoding is: (8250, 51)
```

Prefix

In [0]:

```
#------------------------Vectorizing categorical data of Teacher Prefix for Train dataset----------------------------------

# 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 = 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 = vectorizer_prefix.transform(X_Train['prefix_catogories'].values)
#print("Shape of matrix after one hot encodig ",prefix_one_hot.shape)

#------------------------Vectorizing categorical data of Teacher Prefix for CV dataset----------------------------------

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()
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 = dict(sorted(prefix_dict_CV.items(), key=lambda kv: kv[1]))
prefix_one_hot_CV = vectorizer_prefix.transform(X_CV['prefix_catogories'].values)

#------------------------Vectorizing categorical data of Teacher Prefix for Test dataset-----------------------------------

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 = dict(sorted(prefix_dict_Test.items(), key=lambda kv: kv[1]))
prefix_one_hot_Test = vectorizer_prefix.transform(X_Test['prefix_catogories'].values)
print("-"*120)
print('Shape of Train dataset matrix after one hot encoding is: {0}'.format(prefix_one_hot_Train.shape))
print('Shape of CV dataset matrix after one hot encoding is: {0}'.format(prefix_one_hot_CV.shape))
print('Shape of Test dataset matrix after one hot encoding is: {0}'.format(prefix_one_hot_Test.shape))
```

```
------------------------------------------------------------------------------------------------------------------------
Shape of Train dataset matrix after one hot encoding is: (11222, 4)
Shape of CV dataset matrix after one hot encoding is: (5528, 4)
Shape of Test dataset matrix after one hot encoding is: (8250, 4)
```

project_grade_category

In [0]:

```
#------------------------Vectorizing categorical data of Project Grade for Train dataset----------------------------------

# 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 = 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())

grade_one_hot_Train = vectorizer_grade.transform(X_Train['new_grade_category'].values)

#------------------------Vectorizing categorical data of Project Grade for CV dataset----------------------------------

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 = dict(sorted(grade_dict_CV.items(), key=lambda kv: kv[1]))

grade_one_hot_CV = vectorizer_grade.transform(X_CV['new_grade_category'].values)

#-------------------------Vectorizing categorical data of Project Grade for Train dataset----------------------------------

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 = dict(sorted(grade_dict_Test.items(), key=lambda kv: kv[1]))

grade_one_hot_Test = vectorizer_grade.transform(X_Test['new_grade_category'].values)
print("-"*120)
print('Shape of Train dataset matrix after one hot encoding is: {0}'.format(grade_one_hot_Train.shape))
print('Shape of CV dataset matrix after one hot encoding is: {0}'.format(grade_one_hot_CV.shape))
```

```
print('Shape of Test dataset matrix after one hot encoding is: {0}'.format(grade_one_hot_Test.shape))
```

```
----------------------------------------------------------------------------------------------------------------------------------
Shape of Train dataset matrix after one hot encoding is: (11222, 4)
Shape of CV dataset matrix after one hot encoding is: (5528, 4)
Shape of Test dataset matrix after one hot encoding is: (8250, 4)
```

1.5.2 Vectorizing Numerical features

In [0]:
```
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 [0]:
```
# 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 = price_scalar.transform(X_Train['price'].values.reshape(-1, 1))
price_standardized_CV = price_scalar.transform(X_CV['price'].values.reshape(-1, 1))
price_standardized_Test = price_scalar.transform(X_Test['price'].values.reshape(-1, 1))
```

1.5.3 Vectorizing Text data

1.5.3.1 Bag of words

In [0]:
```
# 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 = vectorizer_essays_bow.fit_transform(preprocessed_essays_Train)
text_bow_CV = vectorizer_essays_bow.transform(preprocessed_essays_CV)
text_bow_Test = vectorizer_essays_bow.transform(preprocessed_essays_Test)
print("-"*120)
print("Applying Bag Of Words for Text Data")
print("-"*120)
print('Shape of Train dataset matrix after one hot encoding is: {0}'.format(text_bow_Train.shape))
print('Shape of CV dataset matrix after one hot encoding is: {0}'.format(text_bow_CV.shape))
print('Shape of Test dataset matrix after one hot encoding is: {0}'.format(text_bow_Test.shape))
```

```
------------------------------------------------------------------------------------------------------------------------
Applying Bag Of Words for Text Data
------------------------------------------------------------------------------------------------------------------------
Shape of Train dataset matrix after one hot encoding is: (11222, 6478)
Shape of CV dataset matrix after one hot encoding is: (5528, 6478)
Shape of Test dataset matrix after one hot encoding is: (8250, 6478)
```

Bag of Words for Project Title

In [0]:
```
# 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 = vectorizer_titles_bow.fit_transform(preprocessed_titles_Train)
title_bow_CV = vectorizer_titles_bow.transform(preprocessed_titles_CV)
title_bow_Test = vectorizer_titles_bow.transform(preprocessed_titles_Test)
print("-"*120)
print("Applying Bag Of Words for Project Title Data")
print("-"*120)
print('Shape of Train dataset matrix after one hot encoding is: {0}'.format(title_bow_Train.shape))
print('Shape of CV dataset matrix after one hot encoding is: {0}'.format(title_bow_CV.shape))
print('Shape of Test dataset matrix after one hot encoding is: {0}'.format(title_bow_Test.shape))

```

```
------------------------------------------------------------------------------------------------------------------------
Applying Bag Of Words for Project Title Data
------------------------------------------------------------------------------------------------------------------------
Shape of Train dataset matrix after one hot encoding is: (11222, 731)
Shape of CV dataset matrix after one hot encoding is: (5528, 731)
Shape of Test dataset matrix after one hot encoding is: (8250, 731)
```

1.5.2.2 TFIDF vectorizer

In [0]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer_essays_tfidf = TfidfVectorizer(min_df=10)
text_tfidf_Train = vectorizer_essays_tfidf.fit_transform(preprocessed_essays_Train)
text_tfidf_CV = vectorizer_essays_tfidf.transform(preprocessed_essays_CV)
text_tfidf_Test = vectorizer_essays_tfidf.transform(preprocessed_essays_Test)
print("-"*120)
print("Applying TFIDF for Text Data")
print("-"*120)
print('Shape of Train dataset matrix after one hot encoding is: {0}'.format(text_tfidf_Train.shape))
print('Shape of CV dataset matrix after one hot encoding is: {0}'.format(text_tfidf_CV.shape))
print('Shape of Test dataset matrix after one hot encoding is: {0}'.format(text_tfidf_Test.shape))
```

```
------------------------------------------------------------------------------------------------------------------------
Applying TFIDF for Text Data
------------------------------------------------------------------------------------------------------------------------
Shape of Train dataset matrix after one hot encoding is: (11222, 6478)
Shape of CV dataset matrix after one hot encoding is: (5528, 6478)
Shape of Test dataset matrix after one hot encoding is: (8250, 6478)
```

TFIDF vectorizer for Project Title

In [0]:
```
vectorizer_titles_tfidf = TfidfVectorizer(min_df=10)
title_tfidf_Train = vectorizer_titles_tfidf.fit_transform(preprocessed_titles_Train)
title_tfidf_CV = vectorizer_titles_tfidf.transform(preprocessed_titles_CV)
title_tfidf_Test = vectorizer_titles_tfidf.transform(preprocessed_titles_Test)
print("-"*120)
print("Applying TFIDF for Project Title")
print("-"*120)
print('Shape of Train dataset matrix after one hot encoding is: {0}'.format(title_tfidf_Train.shape))
print('Shape of CV dataset matrix after one hot encoding is: {0}'.format(title_tfidf_CV.shape))
print('Shape of Test dataset matrix after one hot encoding is: {0}'.format(title_tfidf_Test.shape))
```

```
------------------------------------------------------------------------------------------------------------------------
Applying TFIDF for Project Title
------------------------------------------------------------------------------------------------------------------------
Shape of Train dataset matrix after one hot encoding is: (11222, 731)
Shape of CV dataset matrix after one hot encoding is: (5528, 731)
Shape of Test dataset matrix after one hot encoding is: (8250, 731)
```

1.5.2.3 Using Pretrained Models: Avg W2V

In [0]:

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

In [0]:

```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_vectors_Train = []; # 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.append(vector)
#-------------------------------------------------------------------------------------------

avg_w2v_vectors_CV = []; # 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.append(vector)
#-------------------------------------------------------------------------------------------

avg_w2v_vectors_Test = []; # 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.append(vector)

print(len(avg_w2v_vectors_Test))
print(len(avg_w2v_vectors_Test[1]))
```

```
100%|██████████| 11222/11222 [00:03<00:00, 3135.83it/s]
100%|██████████| 5528/5528 [00:01<00:00, 3163.21it/s]
100%|██████████| 8250/8250 [00:02<00:00, 3260.32it/s]
```

8250
300

AVG W2V on project_title

In [0]:

```
# Similarly you can vectorize for title also
# compute average word2vec for each title.
avg_w2v_vectors_title_Train = []; # 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.append(vector_title)


#-------------------------------------------------------------------------------------------
avg_w2v_vectors_title_CV = []; # 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.append(vector_title)

#-------------------------------------------------------------------------------------------
avg_w2v_vectors_title_Test = []; # 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.append(vector_title)

print(len(avg_w2v_vectors_title_Test))
print(len(avg_w2v_vectors_title_Test[0]))
```

```
100%|██████████| 11222/11222 [00:00<00:00, 57909.34it/s]
100%|██████████| 5528/5528 [00:00<00:00, 59084.94it/s]
```

```
100%|██████████| 8250/8250 [00:00<00:00, 58992.42it/s]

8250
300
```

1.5.2.3 Using Pretrained Models: TFIDF weighted W2V

In [0]:

```
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())
```

In [0]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_Train = []; # 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.append(vector)

#------------------------------------------------------------------------------------------
tfidf_w2v_vectors_CV = []; # 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.append(vector)
#------------------------------------------------------------------------------------------
tfidf_w2v_vectors_Test = []; # 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.append(vector)

print(len(tfidf_w2v_vectors_Test))
print(len(tfidf_w2v_vectors_Test[0]))

```

```
100%|██████████| 11222/11222 [00:20<00:00, 551.58it/s]
100%|██████████| 5528/5528 [00:10<00:00, 548.87it/s]
100%|██████████| 8250/8250 [00:15<00:00, 536.89it/s]

8250
300
```

Using Pretrained Models: TFIDF weighted W2V on project_title

In [0]:

```
# 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 = []; # 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
    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.append(vector_title)
#------------------------------------------------------------------------------------------------------------------------

tfidf_w2v_vectors_title_CV = []; # 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.append(vector_title)
#------------------------------------------------------------------------------------------------------------------------


tfidf_w2v_vectors_title_Test = []; # 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
    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.append(vector_title)

print(len(tfidf_w2v_vectors_title_Test))
print(len(tfidf_w2v_vectors_title_Test[0]))
```

```
100%|██████████| 11222/11222 [00:00<00:00, 28350.87it/s]
100%|██████████| 5528/5528 [00:00<00:00, 35369.22it/s]
100%|██████████| 8250/8250 [00:00<00:00, 35216.34it/s]

8250
300
```

Assignment 3: Apply KNN

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

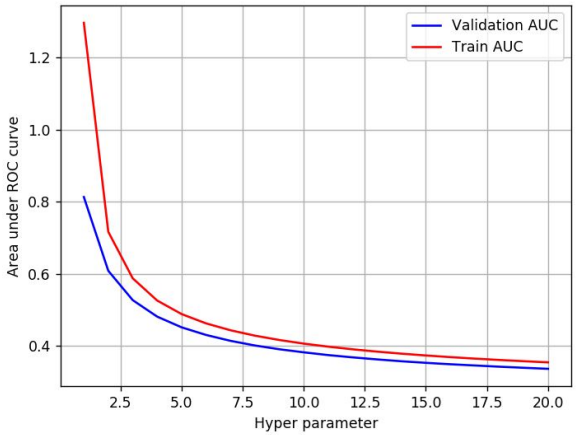
   - Set 1: categorical, numerical features + project_title(BOW) + preprocessed_essay (BOW)
   - Set 2: categorical, numerical features + project_title(TFIDF)+ preprocessed_essay (TFIDF)
   - Set 3: categorical, numerical features + project_title(AVG W2V)+ preprocessed_essay (AVG W2V)
   - Set 4: categorical,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

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

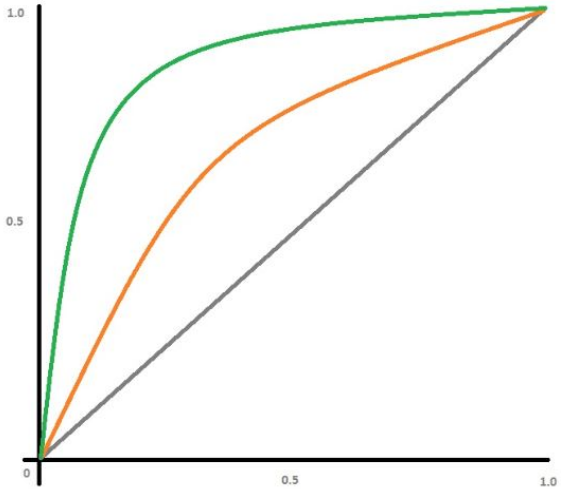
   - Find the best hyper parameter which results in the maximum AUC (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/receiver-operating-characteristic-curve-roc-curve-and-auc-1/) value
   - Find the best hyper paramter using k-fold cross validation (or) simple cross validation data
   - Use gridsearch-cv or randomsearch-cv or write your own for loops to do this task

3. **Representation of results**

- You need to plot the performance of model both on train data and cross validation data for each hyper parameter, as shown in the figure



- Once you find the best hyper parameter, you need to train your model-M using the best hyper-param. Now, find the AUC on test data and plot the ROC curve on both train and test using model-M.



- Along with plotting ROC curve, you need to print the [confusion matrix (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/confusion-matrix-tpr-fpr-fnr-tnr-1/)](https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/confusion-matrix-tpr-fpr-fnr-tnr-1/) with predicted and original labels of test data points

	Predicted: NO	Predicted: YES
Actual: NO	TN = ??	FP = ??
Actual: YES	FN = ??	TP = ??

4. **[Task-2]**

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

5. **Conclusion**

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

Vectorizer	Model	Hyper parameter	AUC
BOW	Brute	7	0.78
TFIDF	Brute	12	0.79
W2V	Brute	10	0.78
TFIDFW2V	Brute	6	0.78

Note: Data Leakage

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

2. K Nearest Neighbor

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

2.2 Make Data Model Ready: encoding numerical, categorical features

2.3 Make Data Model Ready: encoding eassay, and project_title

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

1.5.4 Merging all the above features

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

In [0]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
BOW_Train = hstack((categories_one_hot_Train,sub_categories_one_hot_Train,school_one_hot_Train,grade_one_hot_Train,prefix_one_
print(BOW_Train.shape)
TFIDF_Train = hstack((categories_one_hot_Train,sub_categories_one_hot_Train,school_one_hot_Train,grade_one_hot_Train,prefix_on
print(TFIDF_Train.shape)
AVG_W2V_Train = hstack((categories_one_hot_Train,sub_categories_one_hot_Train,school_one_hot_Train,grade_one_hot_Train,prefix_
print(AVG_W2V_Train.shape)
TFIDF_W2V_Train = hstack((categories_one_hot_Train,sub_categories_one_hot_Train,school_one_hot_Train,grade_one_hot_Train,prefi
print(TFIDF_W2V_Train.shape)
TFIDF_TrainKB = hstack((categories_one_hot_Train,sub_categories_one_hot_Train,school_one_hot_Train,grade_one_hot_Train,prefix_
print(TFIDF_TrainKB.shape)
```

```
(11222, 7308)
(11222, 7308)
(11222, 699)
(11222, 699)
(11222, 7308)
```

In [0]:

```
BOW_CV = hstack((categories_one_hot_CV,sub_categories_one_hot_CV,school_one_hot_CV,grade_one_hot_CV,prefix_one_hot_CV,price_sta
print(BOW_CV.shape)
TFIDF_CV = hstack((categories_one_hot_CV,sub_categories_one_hot_CV,school_one_hot_CV,grade_one_hot_CV,prefix_one_hot_CV,price_
print(TFIDF_CV.shape)
AVG_W2V_CV = hstack((categories_one_hot_CV,sub_categories_one_hot_CV,school_one_hot_CV,grade_one_hot_CV,prefix_one_hot_CV,pric
print(AVG_W2V_CV.shape)
TFIDF_W2V_CV = hstack((categories_one_hot_CV,sub_categories_one_hot_CV,school_one_hot_CV,grade_one_hot_CV,prefix_one_hot_CV,pr
print(TFIDF_W2V_CV.shape)
TFIDF_CVKB = hstack((categories_one_hot_CV,sub_categories_one_hot_CV,school_one_hot_CV,grade_one_hot_CV,prefix_one_hot_CV,pric
print(TFIDF_CVKB.shape)
```

```
(5528, 7308)
(5528, 7308)
(5528, 699)
(5528, 699)
(5528, 7308)
```

In [0]:

```
BOW_Test = hstack((categories_one_hot_Test,sub_categories_one_hot_Test,school_one_hot_Test,grade_one_hot_Test,prefix_one_hot_T
print(BOW_Test.shape)
TFIDF_Test = hstack((categories_one_hot_Test,sub_categories_one_hot_Test,school_one_hot_Test,grade_one_hot_Test,prefix_one_hot_
print(TFIDF_Test.shape)
AVG_W2V_Test = hstack((categories_one_hot_Test,sub_categories_one_hot_Test,school_one_hot_Test,grade_one_hot_Test,prefix_one_h
print(AVG_W2V_Test.shape)
TFIDF_W2V_Test = hstack((categories_one_hot_Test,sub_categories_one_hot_Test,school_one_hot_Test,grade_one_hot_Test,prefix_one_
print(TFIDF_W2V_Test.shape)
TFIDF_Test_KB = hstack((categories_one_hot_Test,sub_categories_one_hot_Test,school_one_hot_Test,grade_one_hot_Test,prefix_one_
print(TFIDF_Test_KB.shape)
```

```
(8250, 7308)
(8250, 7308)
(8250, 699)
(8250, 699)
(8250, 7308)
```

Loading Test Pickle files

In [0]:

```
pck = open('BOW_Test', 'wb')
pickle.dump(BOW_Test, pck)
pck = open('BOW_Test', 'rb')
BOW_Test = pickle.load(pck)
pck.close()

pck = open('TFIDF_Test', 'wb')
pickle.dump(TFIDF_Test, pck)
pck = open('TFIDF_Test', 'rb')
TFIDF_Test = pickle.load(pck)
pck.close()

pck = open('AVG_W2V_Test', 'wb')
pickle.dump(AVG_W2V_Test, pck)
pck = open('AVG_W2V_Test', 'rb')
AVG_W2V_Test = pickle.load(pck)
pck.close()

pck = open('TFIDF_W2V_Test', 'wb')
pickle.dump(TFIDF_W2V_Test, pck)
pck = open('TFIDF_W2V_Test', 'rb')
TFIDF_W2V_Test = pickle.load(pck)
pck.close()

pck = open('TFIDF_Test_KB', 'wb')
pickle.dump(TFIDF_Test_KB, pck)
pck = open('TFIDF_Test_KB', 'rb')
TFIDF_Test_KB = pickle.load(pck)
pck.close()
```

In [0]:

```
BOW_TCV = vstack((BOW_Train,BOW_CV))
print(BOW_TCV.shape)
TFIDF_TCV = vstack((TFIDF_Train,TFIDF_CV))
print(TFIDF_TCV.shape)
AVG_W2V_TCV = vstack((AVG_W2V_Train,AVG_W2V_CV))
print(AVG_W2V_TCV.shape)
TFIDF_W2V_TCV = vstack((TFIDF_W2V_Train,TFIDF_W2V_CV))
print(TFIDF_W2V_TCV.shape)
TFIDF_KB_TCV = vstack((TFIDF_TrainKB,TFIDF_CVKB))
print(TFIDF_KB_TCV.shape)
```

```
(16750, 7308)
(16750, 7308)
(16750, 699)
(16750, 699)
(16750, 7308)
```

Loading Train Pickle files

In [0]:

```
pck = open('BOW_TCV', 'wb')
pickle.dump(BOW_TCV, pck)
pck = open('BOW_TCV', 'rb')
BOW_TCV = pickle.load(pck)
pck.close()

pck = open('TFIDF_TCV', 'wb')
pickle.dump(TFIDF_TCV, pck)
pck = open('TFIDF_TCV', 'rb')
TFIDF_TCV = pickle.load(pck)
pck.close()

pck = open('AVG_W2V_TCV', 'wb')
pickle.dump(AVG_W2V_TCV, pck)
pck = open('AVG_W2V_TCV', 'rb')
AVG_W2V_TCV = pickle.load(pck)
pck.close()

pck = open('TFIDF_W2V_TCV', 'wb')
pickle.dump(TFIDF_W2V_TCV, pck)
pck = open('TFIDF_W2V_TCV', 'rb')
TFIDF_W2V_TCV = pickle.load(pck)
pck.close()

pck = open('TFIDF_KB_TCV', 'wb')
pickle.dump(TFIDF_KB_TCV, pck)
pck = open('TFIDF_KB_TCV', 'rb')
TFIDF_KB_TCV = pickle.load(pck)
pck.close()
```

In [0]:

```
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
```

2.4.1 Applying KNN brute force on BOW, SET 1

Find the best hyper parameter which results in the maximum AUC value

In [0]:

```
%%time
BOW_TCV_CSR = BOW_TCV.tocsr()
BOW_TR_CSR = BOW_Train.tocsr()
BOW_CV_CSR = BOW_CV.tocsr()
BOW_Test_CSR = BOW_Test.tocsr()

k_range = [5,13,17,21,25,33,41,57,65,71,79,89,99,109]

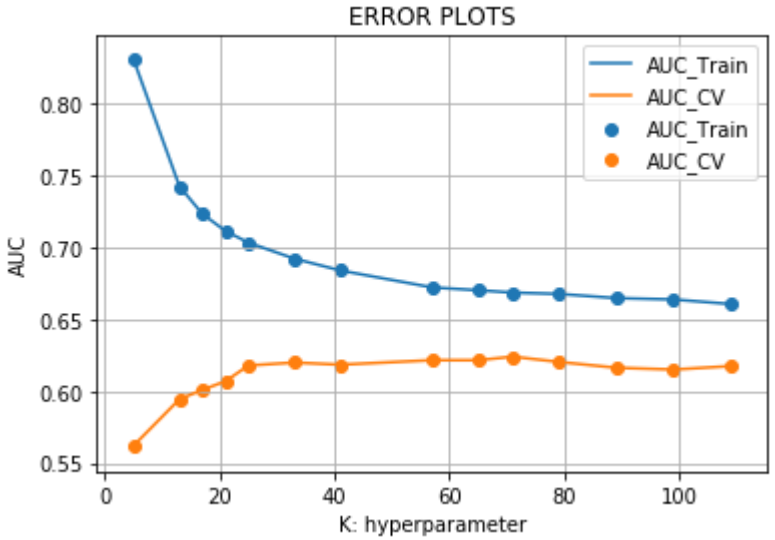
ACCV = []
AUC_TR = []
AUC_TS = []

for i in tqdm(k_range):
    knn = KNeighborsClassifier(n_neighbors=i, n_jobs=-1,algorithm='brute')
    knn.fit(BOW_Train, Y_Train)
    pred = knn.predict(BOW_CV)
    acc = accuracy_score(Y_CV, pred, normalize=True) * float(100)
    ACCV.append(acc)
    Train_pred = batch_predict(knn, BOW_TR_CSR)
    a_fpr_train,a_tpr_train,c = roc_curve(Y_Train, Train_pred)
    AUC_TR.append(auc(a_fpr_train, a_tpr_train))

    Test_pred = batch_predict(knn, BOW_CV_CSR)
    a_fpr_Test,a_tpr_Test,c = roc_curve(Y_CV, Test_pred)
    AUC_TS.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_TR, label='AUC_Train')
plt.scatter(k_range, AUC_TR, label='AUC_Train')
plt.gca()
plt.plot(k_range, AUC_TS, label='AUC_CV')
plt.scatter(k_range, AUC_TS, label='AUC_CV')
plt.gca()
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```

```
100%|██████████| 14/14 [06:05<00:00, 25.98s/it]
```




```
CPU times: user 10min 3s, sys: 1.8 s, total: 10min 5s
Wall time: 6min 6s
```

OBSERVATION: From the above plot the diffrence between AUC train and AUC cv started to reduce from the K value 71. From the plot we can see that K value 71 and 79 have same difference. Hence i have choose the optimal K value as 71.

In [0]:

```
k_opt=71
knn_opt = KNeighborsClassifier(n_neighbors = k_opt, n_jobs=-1,algorithm='brute')
knn_opt.fit(BOW_Train, Y_Train)
pred = knn.predict(BOW_Test)
acc = accuracy_score(Y_Test, pred, normalize=True) * float(100)
print('\nTest accuracy for k = {0} is {1}%'.format(k_opt,acc))

Y_Train_pred = batch_predict(knn_opt, BOW_TR_CSR)
Y_Test_pred = batch_predict(knn_opt, BOW_Test_CSR)

fpr_Train, tpr_Train, thresholds = roc_curve(Y_Train, Y_Train_pred)
fpr_Test, tpr_Test, thresholds = roc_curve(Y_Test, Y_Test_pred)
```

```
Test accuracy for k = 71 is 84.87272727272727%
```

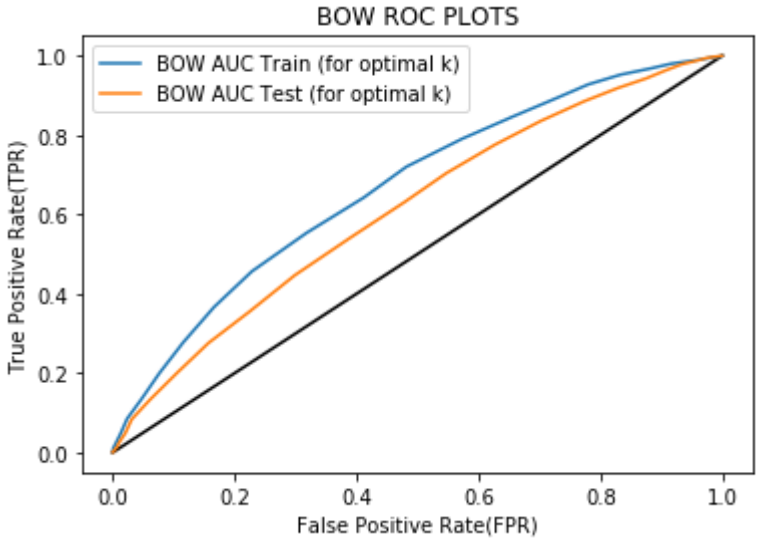
BOW ROC PLOT

In [0]:

```
%%time

#https://stackoverflow.com/questions/52910061/implementing-roc-curves-for-k-nn-machine-learning-algorithm-using-python-and-sci

plt.plot([0,1],[0,1],'k-')
plt.plot(fpr_Train, tpr_Train, label="BOW AUC Train (for optimal k)")
plt.plot(fpr_Test, tpr_Test, label="BOW AUC Test (for optimal k)")
plt.legend()
plt.ylabel("True Positive Rate(TPR)")
plt.xlabel("False Positive Rate(FPR)")
plt.title("BOW ROC PLOTS")
plt.show()
print("-"*120)
print("AUC Train (for optimal k) =", auc(fpr_Train, tpr_Train))
print("AUC Test (for optimal k) =", auc(fpr_Test, tpr_Test))
BOW_AUC=round(auc(fpr_Test, tpr_Test)*100)
BOW_K=k_opt
```




```
------------------------------------------------------------------------------------------------------------------------
AUC Train (for optimal k) = 0.6686594927594207
AUC Test (for optimal k) = 0.6138982039929983
```

```
CPU times: user 188 ms, sys: 3 ms, total: 191 ms
Wall time: 201 ms
```

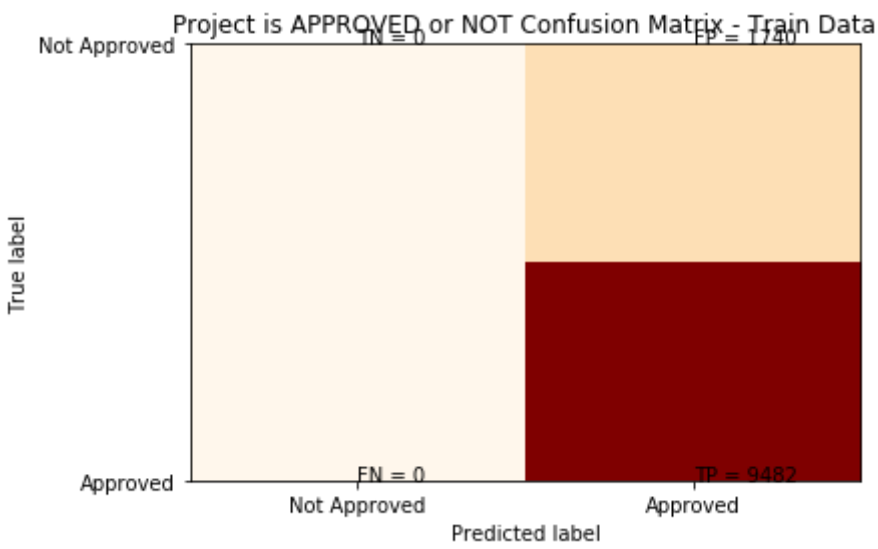
BOW CONFUSION MATRIX

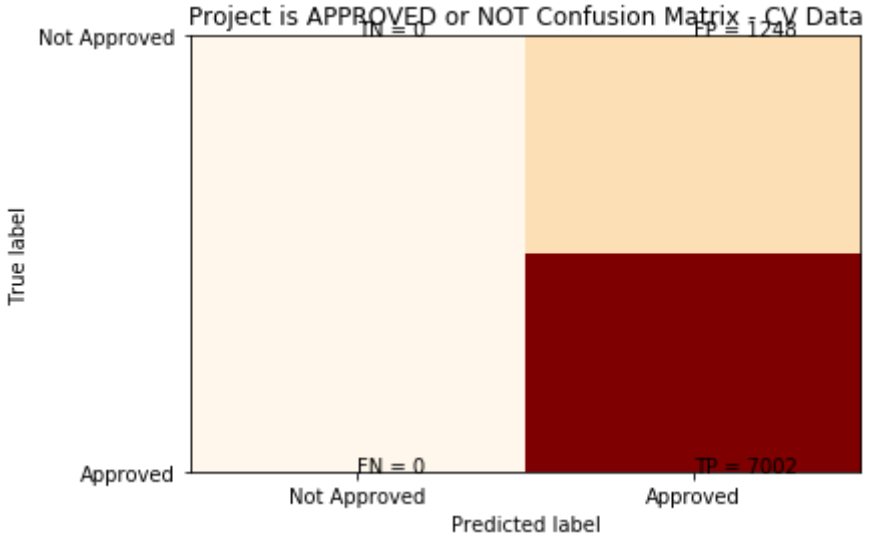
In [0]:

```
%%time
#https://tatwan.github.io/How-To-Plot-A-Confusion-Matrix-In-Python/
#https://matplotlib.org/3.1.1/gallery/color/colormap_reference.html

PR1= knn_opt.predict(BOW_Train)
PR2= knn_opt.predict(BOW_Test)
#-----------------------Confusion matrix for BOW Train Data----------------------------------------------------
plt.clf()
CM1 = confusion_matrix(Y_Train,PR1)
plt.imshow(CM1, interpolation='nearest', cmap='OrRd', aspect='auto')
classNames = ['Not Approved', 'Approved']
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(CM1[i][j]))
plt.show()

#-----------------------Confusion matrix for BOW Test Data----------------------------------------------------
plt.clf()
CM2 = confusion_matrix(Y_Test,PR2)
plt.imshow(CM2, interpolation='nearest', cmap='OrRd', aspect='auto')
classNames = ['Not Approved', 'Approved']
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()
```






```
CPU times: user 44.5 s, sys: 86.9 ms, total: 44.6 s
Wall time: 27.5 s
```

2.4.2 Applying KNN brute force on TFIDF, SET 2

In [0]:

```
%%time

TFIDF_TCV_CSR=TFIDF_TCV.tocsr()
TFIDF_TR_CSR = TFIDF_Train.tocsr()
TFIDF_CV_CSR = TFIDF_CV.tocsr()
TFIDF_Test_CSR = TFIDF_Test.tocsr()


k_range = [5,13,17,21,25,33,41,57,65,71,79,89,99,109]

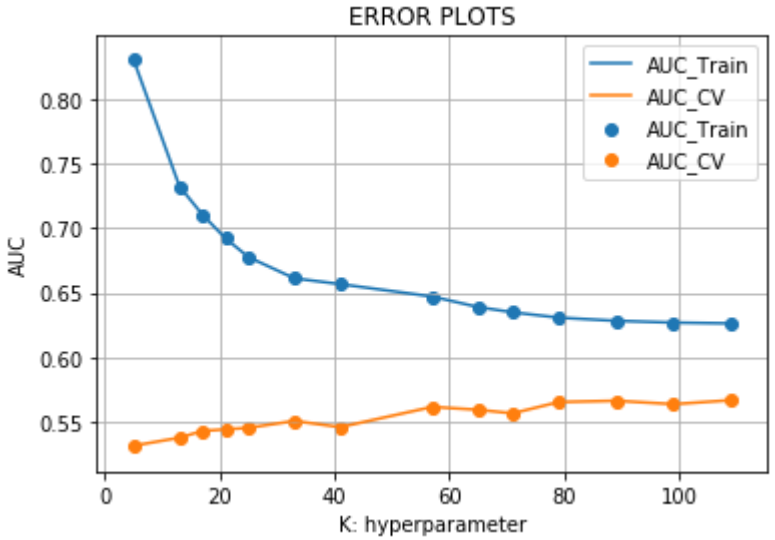
ACCV = []
AUC_TR = []
AUC_TS = []

for i in tqdm(k_range):
    knn = KNeighborsClassifier(n_neighbors=i, n_jobs=-1,algorithm='brute')
    knn.fit(TFIDF_TR_CSR, Y_Train)
    pred = knn.predict(TFIDF_CV)
    acc = accuracy_score(Y_CV, pred, normalize=True) * float(100)
    ACCV.append(acc)
    Train_pred = batch_predict(knn, TFIDF_TR_CSR)
    a_fpr_train,a_tpr_train,c = roc_curve(Y_Train, Train_pred)
    AUC_TR.append(auc(a_fpr_train, a_tpr_train))

    Test_pred = batch_predict(knn, TFIDF_CV_CSR)
    a_fpr_Test,a_tpr_Test,c = roc_curve(Y_CV, Test_pred)
    AUC_TS.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_TR, label='AUC_Train')
plt.scatter(k_range, AUC_TR, label='AUC_Train')
plt.gca()
plt.plot(k_range, AUC_TS, label='AUC_CV')
plt.scatter(k_range, AUC_TS, label='AUC_CV')
plt.gca()
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```

```
100%|██████████| 14/14 [06:05<00:00, 26.37s/it]
```



OBSERVATION: From the above plot the diffrence between AUC train and AUC cv started to reduce from the K value 89. After that differnce seems to be same. Hence i have choose Optimal K value as 89.

In [0]:

```
k_opt=89

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

Y_Train_pred = batch_predict(knn_opt, TFIDF_TR_CSR)
Y_Test_pred = batch_predict(knn_opt, TFIDF_Test_CSR)

fpr_Train, tpr_Train, thresholds = roc_curve(Y_Train, Y_Train_pred)
fpr_Test, tpr_Test, thresholds = roc_curve(Y_Test, Y_Test_pred)
```

```
Test accuracy for k = 89 is 84.87272727272727%
```

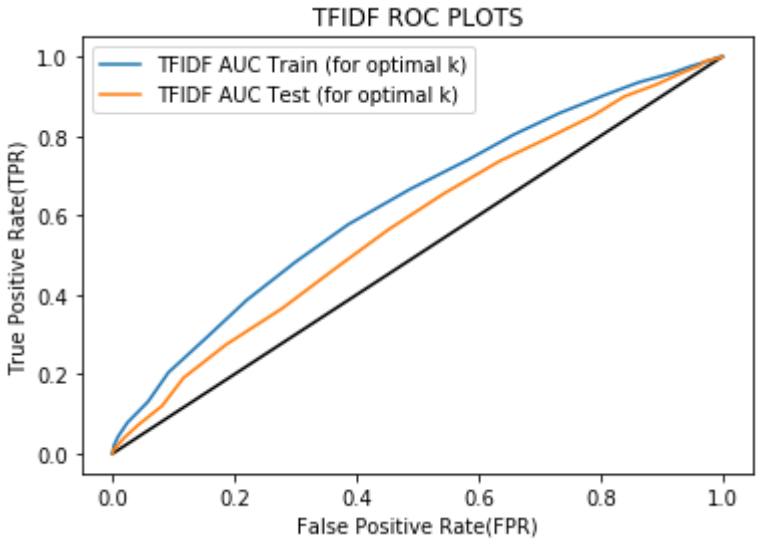
TFIDF ROC PLOT

In [0]:

```
%%time

#https://stackoverflow.com/questions/52910061/implementing-roc-curves-for-k-nn-machine-learning-algorithm-using-python-and-sci

plt.plot([0,1],[0,1],'k-')
plt.plot(fpr_Train, tpr_Train, label="TFIDF AUC Train (for optimal k)")
plt.plot(fpr_Test, tpr_Test, label="TFIDF AUC Test (for optimal k)")
plt.legend()
plt.ylabel("True Positive Rate(TPR)")
plt.xlabel("False Positive Rate(FPR)")
plt.title("TFIDF ROC PLOTS")
plt.show()
print("-"*120)
print("AUC Train (for optimal k) =", auc(fpr_Train, tpr_Train))
print("AUC Test (for optimal k) =", auc(fpr_Test, tpr_Test))
TFIDF_AUC=round(auc(fpr_Test, tpr_Test)*100)
TFIDF_K=k_opt
```




```
------------------------------------------------------------------------------------------------------------------------
AUC Train (for optimal k) = 0.6281867397876679
AUC Test (for optimal k) = 0.5761106945634581
CPU times: user 202 ms, sys: 4.97 ms, total: 207 ms
Wall time: 210 ms
```

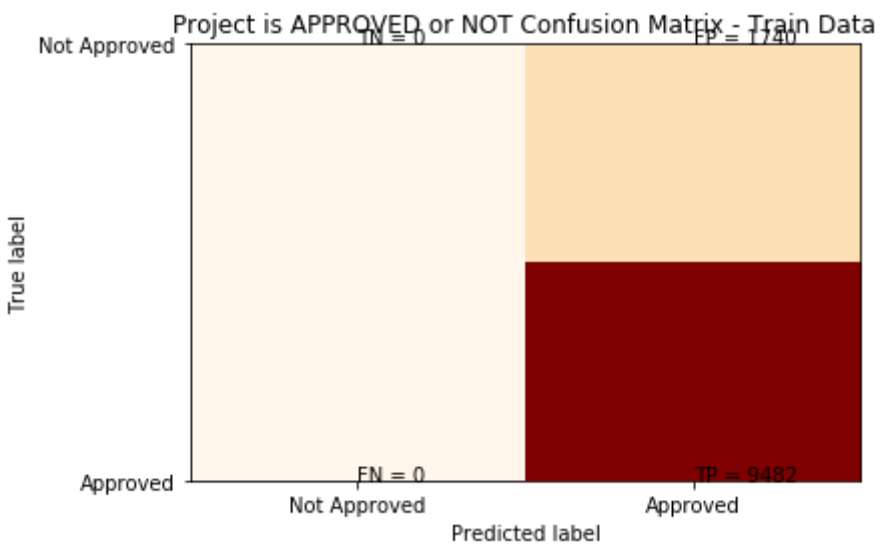
TFIDF CONFUSION MATRIX

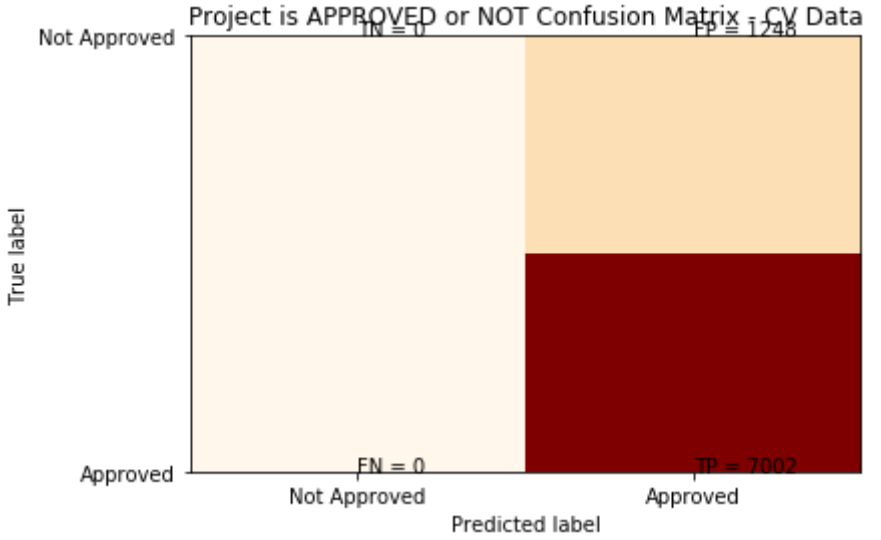
In [0]:

```
%%time
#https://tatwan.github.io/How-To-Plot-A-Confusion-Matrix-In-Python/
#https://matplotlib.org/3.1.1/gallery/color/colormap_reference.html

PR3= knn_opt.predict(TFIDF_Train)
PR4= knn_opt.predict(TFIDF_Test)
#-----------------------Confusion matrix for TFIDF Train Data----------------------------------------------------
plt.clf()
CM3 = confusion_matrix(Y_Train,PR3)
plt.imshow(CM3, interpolation='nearest', cmap='OrRd', aspect='auto')
classNames = ['Not Approved', 'Approved']
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(CM3[i][j]))
plt.show()

#-----------------------Confusion matrix for TFIDF Test Data----------------------------------------------------
plt.clf()
CM4 = confusion_matrix(Y_Test,PR4)
plt.imshow(CM4, interpolation='nearest', cmap='OrRd', aspect='auto')
classNames = ['Not Approved', 'Approved']
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(CM4[i][j]))
plt.show()
```






```
CPU times: user 45.1 s, sys: 89.8 ms, total: 45.2 s
Wall time: 27.9 s
```

2.4.3 Applying KNN brute force on AVG W2V, SET 3

In [0]:

```
%%time
AVG_W2V_TCV_CSR = AVG_W2V_TCV.tocsr()
AVG_W2V_TR_CSR = AVG_W2V_Train.tocsr()
AVG_W2V_CV_CSR = AVG_W2V_CV.tocsr()
AVG_W2V_Test_CSR = AVG_W2V_Test.tocsr()

k_range = [5,13,17,21,25,33,41,57,65,71,79,89,99,109]

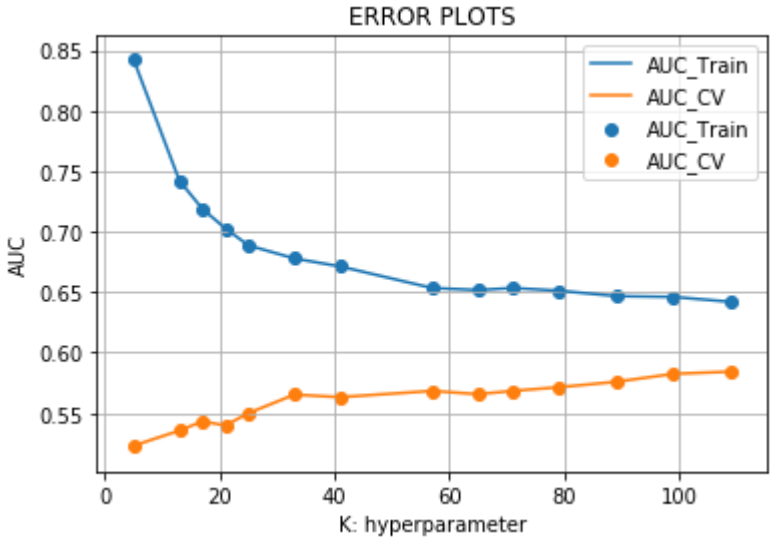
ACCV = []
AUC_TR = []
AUC_TS = []

for i in tqdm(k_range):
    knn = KNeighborsClassifier(n_neighbors=i, n_jobs=-1,algorithm='brute')
    knn.fit(AVG_W2V_TR_CSR, Y_Train)
    pred = knn.predict(AVG_W2V_CV)
    acc = accuracy_score(Y_CV, pred, normalize=True) * float(100)
    ACCV.append(acc)
    Train_pred = batch_predict(knn, AVG_W2V_TR_CSR)
    a_fpr_train,a_tpr_train,c = roc_curve(Y_Train, Train_pred)
    AUC_TR.append(auc(a_fpr_train, a_tpr_train))

    Test_pred = batch_predict(knn, AVG_W2V_CV_CSR)
    a_fpr_Test,a_tpr_Test,c = roc_curve(Y_CV, Test_pred)
    AUC_TS.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_TR, label='AUC_Train')
plt.scatter(k_range, AUC_TR, label='AUC_Train')
plt.gca()
plt.plot(k_range, AUC_TS, label='AUC_CV')
plt.scatter(k_range, AUC_TS, label='AUC_CV')
plt.gca()
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```

```
100%|██████████| 14/14 [1:22:53<00:00, 361.47s/it]
```




```
CPU times: user 2h 41min 54s, sys: 8.2 s, total: 2h 42min 2s
Wall time: 1h 22min 53s
```

OBSERVATION: From the above plot the diffrence between AUC train and AUC cv have very less value for the K=109. Hence i have choose that as an Optimal K value.

In [0]:

```
k_opt=109

knn_opt = KNeighborsClassifier(n_neighbors = k_opt, n_jobs=-1,algorithm='brute')
knn_opt.fit(AVG_W2V_Train, Y_Train)
pred = knn.predict(AVG_W2V_Test)
acc = accuracy_score(Y_Test, pred, normalize=True) * float(100)
print('\nTest accuracy for k = {0} is {1}%'.format(k_opt,acc))

Y_Train_pred = batch_predict(knn_opt, AVG_W2V_TR_CSR)
Y_Test_pred = batch_predict(knn_opt, AVG_W2V_Test_CSR)

fpr_Train, tpr_Train, thresholds = roc_curve(Y_Train, Y_Train_pred)
fpr_Test, tpr_Test, thresholds = roc_curve(Y_Test, Y_Test_pred)
```

```
Test accuracy for k = 109 is 84.87272727272727%
```

AVG W2V ROC PLOT

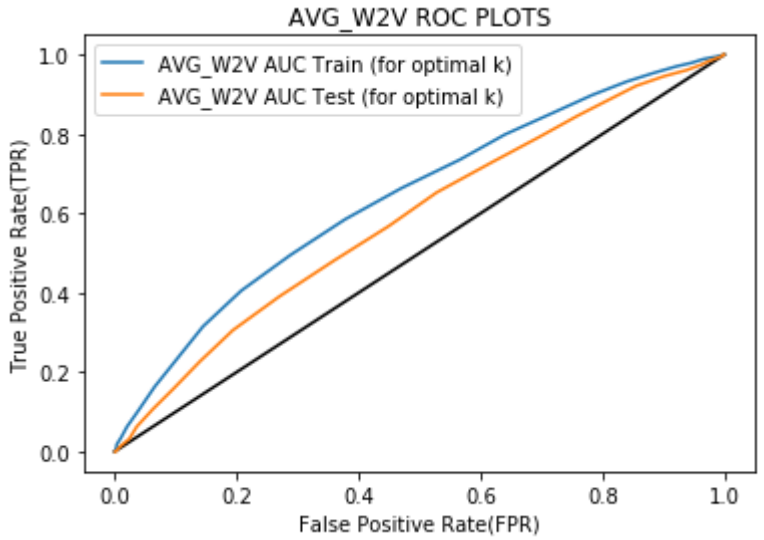
In [0]:

```
%%time

#https://stackoverflow.com/questions/52910061/implementing-roc-curves-for-k-nn-machine-learning-algorithm-using-python-and-sci

plt.plot([0,1],[0,1],'k-')
plt.plot(fpr_Train, tpr_Train, label="AVG_W2V AUC Train (for optimal k)")
plt.plot(fpr_Test, tpr_Test, label="AVG_W2V AUC Test (for optimal k)")
plt.legend()
plt.ylabel("True Positive Rate(TPR)")
plt.xlabel("False Positive Rate(FPR)")
plt.title("AVG_W2V ROC PLOTS")
plt.show()
print("-"*120)

print("AUC Train (for optimal k) =", auc(fpr_Train, tpr_Train))
print("AUC Test (for optimal k) =", auc(fpr_Test, tpr_Test))
AVG_W2V_AUC=round(auc(fpr_Test, tpr_Test)*100)
AVG_W2V_K=k_opt
```




```
------------------------------------------------------------------------------------------------------------------------
AUC Train (for optimal k) = 0.6419674483049553
AUC Test (for optimal k) = 0.5878782801983315
CPU times: user 200 ms, sys: 5 ms, total: 205 ms
Wall time: 208 ms
```

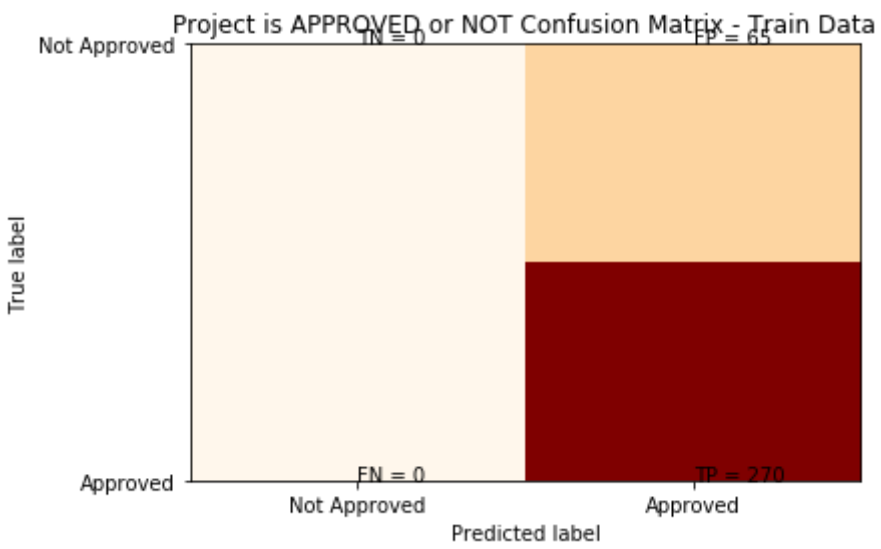
AVG W2V CONFUSION MATRIX

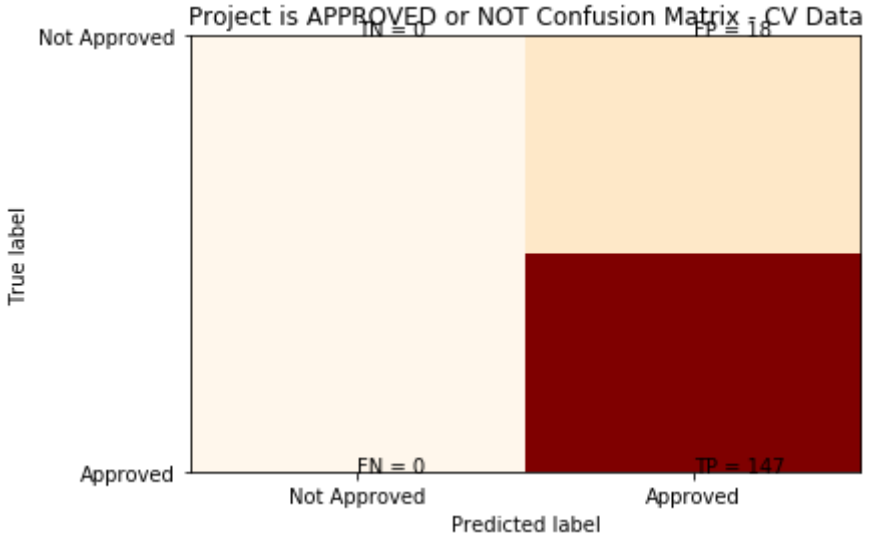
In [0]:

```
%%time
#https://tatwan.github.io/How-To-Plot-A-Confusion-Matrix-In-Python/
#https://matplotlib.org/3.1.1/gallery/color/colormap_reference.html

PR5= knn_opt.predict(AVG_W2V_Train)
PR6= knn_opt.predict(AVG_W2V_Test)
#-----------------------Confusion matrix for AVG_W2V Train Data-----------------------------------------------------
plt.clf()
CM5 = confusion_matrix(Y_Train,PR5)
plt.imshow(CM5, interpolation='nearest', cmap='OrRd', aspect='auto')
classNames = ['Not Approved', 'Approved']
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(CM5[i][j]))
plt.show()

#-----------------------Confusion matrix for AVG_W2V Test Data-----------------------------------------------------
plt.clf()
CM6 = confusion_matrix(Y_Test,PR6)
plt.imshow(CM6, interpolation='nearest', cmap='OrRd', aspect='auto')
classNames = ['Not Approved', 'Approved']
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(CM6[i][j]))
plt.show()
```






```
Wall time: 612 ms
```

2.4.4 Applying KNN brute force on TFIDF W2V, SET 4

In [0]:

```
%%time
TFIDF_W2V_TCV_CSR = TFIDF_W2V_TCV.tocsr()
TFIDF_W2V_TR_CSR = TFIDF_W2V_Train.tocsr()
TFIDF_W2V_CV_CSR = TFIDF_W2V_CV.tocsr()
TFIDF_W2V_Test_CSR = TFIDF_W2V_Test.tocsr()

k_range = [5,13,17,21,25,33,41,57,65,71,79,89,99,109]

ACCV = []
AUC_TR = []
AUC_TS = []

for i in tqdm(k_range):
    knn = KNeighborsClassifier(n_neighbors=i, n_jobs=-1,algorithm='brute')
    knn.fit(TFIDF_W2V_TR_CSR, Y_Train)
    pred = knn.predict(TFIDF_W2V_CV)
    acc = accuracy_score(Y_CV, pred, normalize=True) * float(100)
    ACCV.append(acc)
    Train_pred = batch_predict(knn, TFIDF_W2V_TR_CSR)
    a_fpr_train,a_tpr_train,c = roc_curve(Y_Train, Train_pred)
    AUC_TR.append(auc(a_fpr_train, a_tpr_train))

    Test_pred = batch_predict(knn, TFIDF_W2V_CV_CSR)
    a_fpr_Test,a_tpr_Test,c = roc_curve(Y_CV, Test_pred)
    AUC_TS.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_TR, label='AUC_Train')
plt.scatter(k_range, AUC_TR, label='AUC_Train')
plt.gca()
plt.plot(k_range, AUC_TS, label='AUC_CV')
plt.scatter(k_range, AUC_TS, label='AUC_CV')
plt.gca()
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```

```
100%|██████████| 14/14 [42:32<00:00, 183.43s/it]
```

OBSERVATION: From the above plot the diffrence between AUC train and AUC cv have very less value for the K=109. Hence i have choose that as an Optimal K value.

In [0]:

```
k_opt=109

knn_opt = KNeighborsClassifier(n_neighbors = k_opt, n_jobs=-1,algorithm='brute')
knn_opt.fit(TFIDF_W2V_Train, Y_Train)
pred = knn.predict(TFIDF_W2V_Test)
acc = accuracy_score(Y_Test, pred, normalize=True) * float(100)
print('\nTest accuracy for k = {0} is {1}%'.format(k_opt,acc))

Y_Train_pred = batch_predict(knn_opt, TFIDF_W2V_TR_CSR)
Y_Test_pred = batch_predict(knn_opt, TFIDF_W2V_Test_CSR)

fpr_Train, tpr_Train, thresholds = roc_curve(Y_Train, Y_Train_pred)
fpr_Test, tpr_Test, thresholds = roc_curve(Y_Test, Y_Test_pred)

```

```
Test accuracy for k = 109 is 84.87272727272727%
```

TFIDF W2V ROC PLOT

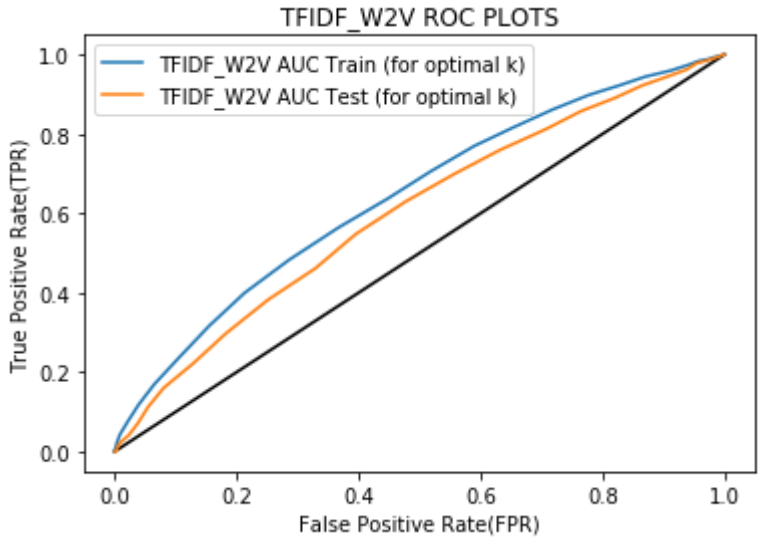
In [0]:

```
%%time

#https://stackoverflow.com/questions/52910061/implementing-roc-curves-for-k-nn-machine-learning-algorithm-using-python-and-sci

plt.plot([0,1],[0,1],'k-')
plt.plot(fpr_Train, tpr_Train, label="TFIDF_W2V AUC Train (for optimal k)")
plt.plot(fpr_Test, tpr_Test, label="TFIDF_W2V AUC Test (for optimal k)")
plt.legend()
plt.ylabel("True Positive Rate(TPR)")
plt.xlabel("False Positive Rate(FPR)")
plt.title("TFIDF_W2V ROC PLOTS")
plt.show()
print("-"*120)
print("AUC Train (for optimal k) =", auc(fpr_Train, tpr_Train))
print("AUC Test (for optimal k) =", auc(fpr_Test, tpr_Test))
TFIDF_W2V_AUC=round(auc(fpr_Test, tpr_Test)*100)
TFIDF_W2V_K=k_opt

```




```
------------------------------------------------------------------------------------------------------------------------
AUC Train (for optimal k) = 0.6408993931635742
```

```
AUC Test (for optimal k) = 0.6011983641120852
CPU times: user 199 ms, sys: 4.99 ms, total: 204 ms
Wall time: 207 ms
```

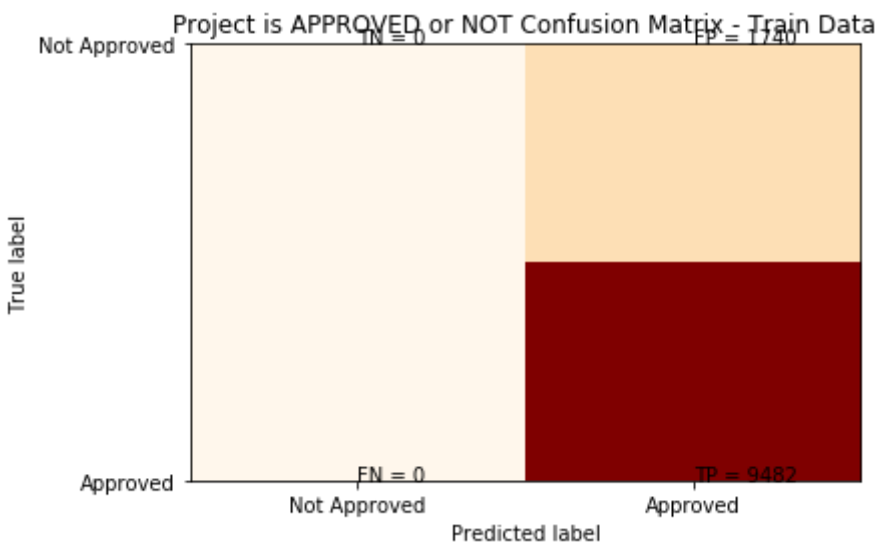
TFIDF W2V CONFUSION MATRIX

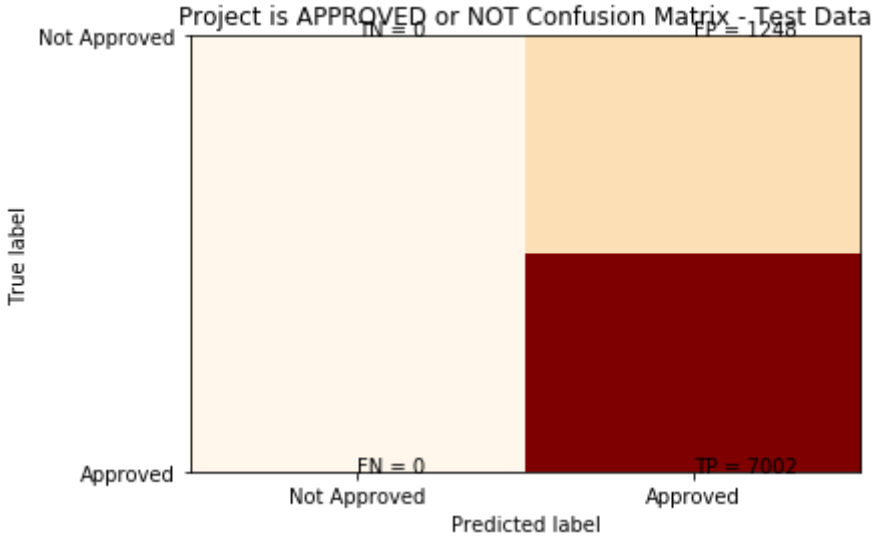
In [0]:

```
%%time
#https://tatwan.github.io/How-To-Plot-A-Confusion-Matrix-In-Python/
#https://matplotlib.org/3.1.1/gallery/color/colormap_reference.html

PR7= knn_opt.predict(TFIDF_W2V_Train)
PR8= knn_opt.predict(TFIDF_W2V_Test)
#-----------------------Confusion matrix for TFIDF_W2V Train Data----------------------------------------------------
plt.clf()
CM7 = confusion_matrix(Y_Train,PR7)
plt.imshow(CM7, interpolation='nearest', cmap='OrRd', aspect='auto')
classNames = ['Not Approved', 'Approved']
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(CM7[i][j]))
plt.show()

#-----------------------Confusion matrix for TFIDF_W2V Test Data----------------------------------------------------
plt.clf()
CM8 = confusion_matrix(Y_Test,PR8)
plt.imshow(CM8, interpolation='nearest', cmap='OrRd', aspect='auto')
classNames = ['Not Approved', 'Approved']
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(CM8[i][j]))
plt.show()
```






```
CPU times: user 5min 10s, sys: 572 ms, total: 5min 10s
Wall time: 2min 41s
```

2.5 Feature selection with `SelectKBest`

In [0]:

```
%%time
#https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.f_classif.html#sklearn.feature_selection.f_classi
warnings.filterwarnings("ignore")

TFIDF_Train_KB1 = SelectKBest(f_classif, k=2000).fit(TFIDF_TrainKB, Y_Train)
TFIDF_Train_KBST = TFIDF_Train_KB1.transform(TFIDF_TrainKB)
TFIDF_CV_KBST = TFIDF_Train_KB1.transform(TFIDF_CVKB)
TFIDF_Test_KBST = TFIDF_Train_KB1.transform(TFIDF_Test_KB)

print("-"*120)
print("Selecting the K Best")
print("-"*120)
print('Shape of Train dataset matrix after one hot encoding is: {0}'.format(TFIDF_Train_KBST.shape))
print('Shape of CV dataset matrix after one hot encoding is: {0}'.format(TFIDF_CV_KBST.shape))
print('Shape of Test dataset matrix after one hot encoding is: {0}'.format(TFIDF_Test_KBST.shape))
```

```
------------------------------------------------------------------------------------------------------------------------
Selecting the K Best
------------------------------------------------------------------------------------------------------------------------
Shape of Train dataset matrix after one hot encoding is: (11222, 2000)
Shape of CV dataset matrix after one hot encoding is: (5528, 2000)
Shape of Test dataset matrix after one hot encoding is: (8250, 2000)
CPU times: user 212 ms, sys: 1 ms, total: 213 ms
Wall time: 214 ms
```

In [0]:

```
%%time
TFIDF_TCV_CSR_KBST = TFIDF_KB_TCV.tocsr()
TFIDF_TR_CSR_KBST = TFIDF_Train_KBST.tocsr()
TFIDF_CV_CSR_KBST = TFIDF_CV_KBST.tocsr()
TFIDF_Test_CSR_KBST = TFIDF_Test_KB.tocsr()

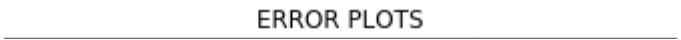
k_range = [5,13,17,21,25,33,41,57,65,71,79,89,99,109]

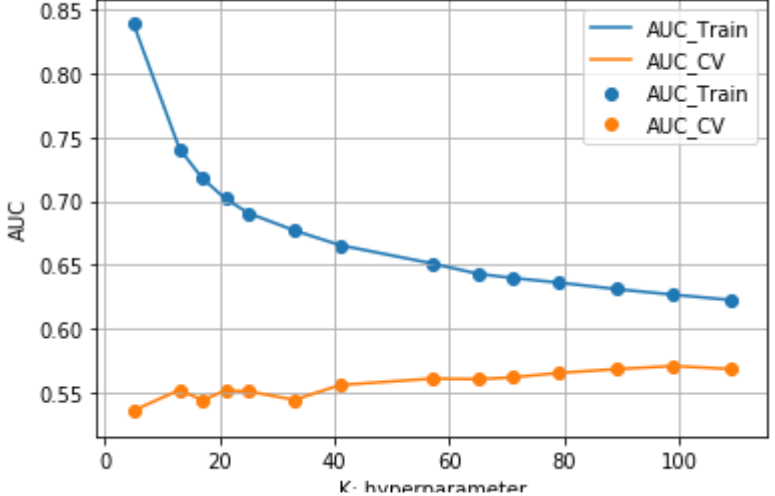

ACCV = []
AUC_TR = []
AUC_TS = []

for i in tqdm(k_range):
    knn = KNeighborsClassifier(n_neighbors=i, n_jobs=-1,algorithm='brute')
    knn.fit(TFIDF_Train_KBST, Y_Train)
    pred = knn.predict(TFIDF_CV_KBST)
    acc = accuracy_score(Y_CV, pred, normalize=True) * float(100)
    ACCV.append(acc)
    Train_pred = batch_predict(knn, TFIDF_TR_CSR_KBST)
    a_fpr_train,a_tpr_train,c = roc_curve(Y_Train, Train_pred)
    AUC_TR.append(auc(a_fpr_train, a_tpr_train))

    Test_pred = batch_predict(knn, TFIDF_CV_CSR_KBST)
    a_fpr_Test,a_tpr_Test,c = roc_curve(Y_CV, Test_pred)
    AUC_TS.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_TR, label='AUC_Train')
plt.scatter(k_range, AUC_TR, label='AUC_Train')
plt.gca()
plt.plot(k_range, AUC_TS, label='AUC_CV')
plt.scatter(k_range, AUC_TS, label='AUC_CV')
plt.gca()
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```





OBSERVATION: From the above plot the diffrence between AUC train and AUC cv started to reduce from the K value 99 and difference remain same for the K values 99 and 109. Hence i have choose the optimal K value as 99.

In [0]:

```
k_opt=99

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

Y_Train_pred = batch_predict(knn_opt, TFIDF_TR_CSR_KBST)
Y_Test_pred = batch_predict(knn_opt, TFIDF_Test_KBST.tocsr())

fpr_Train, tpr_Train, thresholds = roc_curve(Y_Train, Y_Train_pred)
fpr_Test, tpr_Test, thresholds = roc_curve(Y_Test, Y_Test_pred)

```

```
Test accuracy for k = 99 is 84.87272727272727%
```

SELECT K BEST TFIDF ROC PLOT

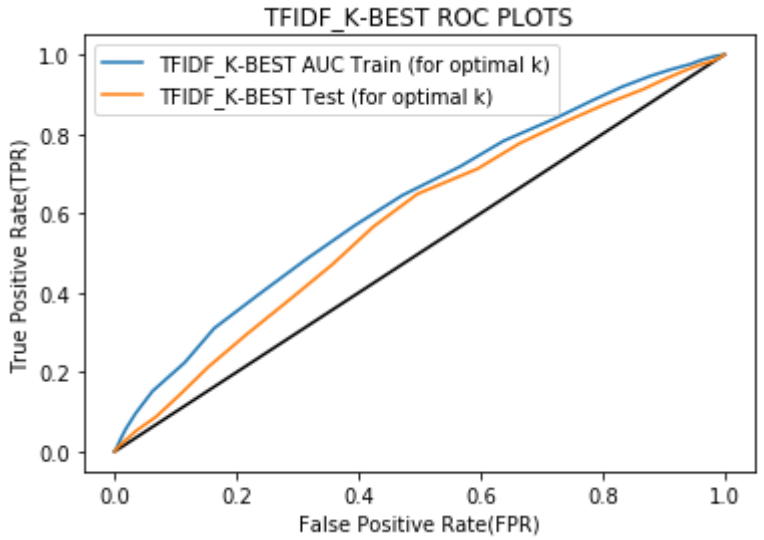
In [0]:

```
%%time

#https://stackoverflow.com/questions/52910061/implementing-roc-curves-for-k-nn-machine-learning-algorithm-using-python-and-sci

plt.plot([0,1],[0,1],'k-')
plt.plot(fpr_Train, tpr_Train, label="TFIDF_K-BEST AUC Train (for optimal k)")
plt.plot(fpr_Test, tpr_Test, label="TFIDF_K-BEST Test (for optimal k)")
plt.legend()
plt.ylabel("True Positive Rate(TPR)")
plt.xlabel("False Positive Rate(FPR)")
plt.title("TFIDF_K-BEST ROC PLOTS")
plt.show()
print("-"*120)
print("AUC Train (for optimal k) =", auc(fpr_Train, tpr_Train))
print("AUC Test (for optimal k) =", auc(fpr_Test, tpr_Test))
KBST_AUC=round(auc(fpr_Test, tpr_Test)*100)
KBST_K=k_opt

```




```
------------------------------------------------------------------------------------------------------------------------
AUC Train (for optimal k) = 0.6223923671469476
AUC Test (for optimal k) = 0.5824019945766411
```

```
CPU times: user 183 ms, sys: 3.99 ms, total: 187 ms
Wall time: 189 ms
```

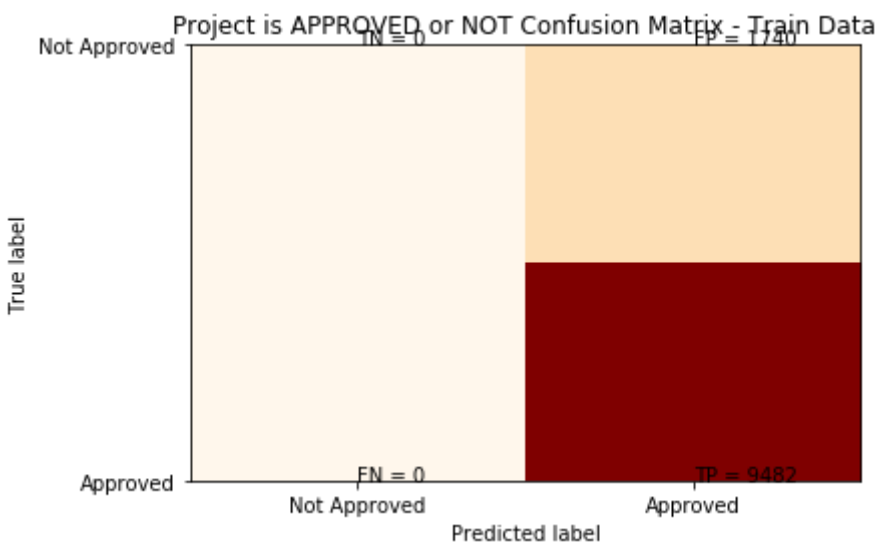
SELECT K BEST TFIDF CONFUSION MATRIX

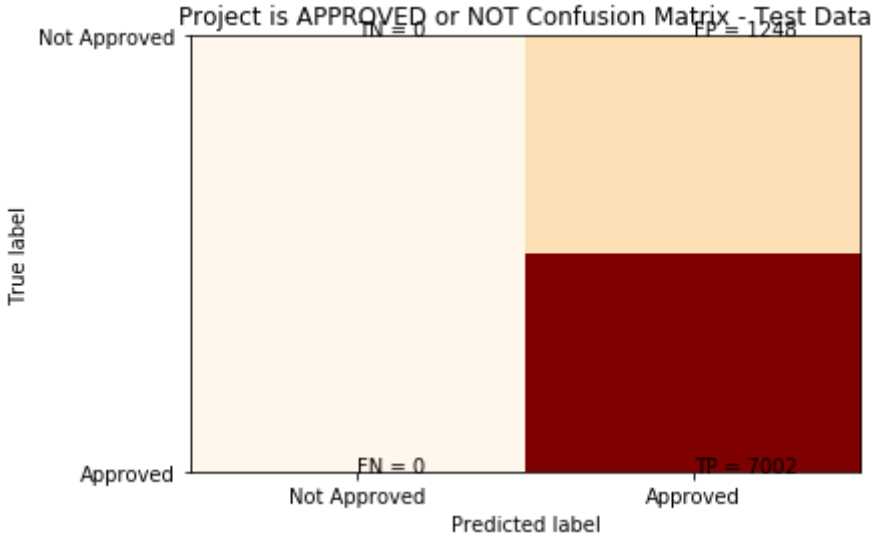
In [0]:

```
%%time
#https://tatwan.github.io/How-To-Plot-A-Confusion-Matrix-In-Python/
#https://matplotlib.org/3.1.1/gallery/color/colormap_reference.html

PR9= knn_opt.predict(TFIDF_Train_KBST)
PR10= knn_opt.predict(TFIDF_Test_KBST)
#-----------------------Confusion matrix for K-BEST TFIDF_W2V Train Data-----------------------------------------------
plt.clf()
CM9 = confusion_matrix(Y_Train,PR9)
plt.imshow(CM9, interpolation='nearest', cmap='OrRd', aspect='auto')
classNames = ['Not Approved', 'Approved']
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(CM9[i][j]))
plt.show()

#-----------------------Confusion matrix for K-BEST TFIDF_W2V Test Data-----------------------------------------------
plt.clf()
CM10 = confusion_matrix(Y_Test,PR10)
plt.imshow(CM8, interpolation='nearest', cmap='OrRd', aspect='auto')
classNames = ['Not Approved', 'Approved']
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(CM10[i][j]))
plt.show()
```






```
CPU times: user 34.2 s, sys: 1.38 s, total: 35.6 s
Wall time: 22.5 s
```

3. Conclusions

In [0]:

```
# Please compare all your models using Prettytable library

pt = PrettyTable()
pt.field_names= ("Vectorizer", "Model", "HyperParameter", "AUC")
pt.add_row(["BOW", "Brute",BOW_K, BOW_AUC])
pt.add_row(["Tf-Idf", "Brute", TFIDF_K, TFIDF_AUC])
pt.add_row(["AVG W2V", "Brute",AVG_W2V_K, AVG_W2V_AUC])
pt.add_row(["TFIDF W2V", "Brute", TFIDF_W2V_K, TFIDF_W2V_AUC])
pt.add_row(["Top 2000 features of Tf-Idf", "Brute", KBST_K, KBST_AUC])
print(pt)
```

```
+-----------------------------+-------+----------------+------+
|          Vectorizer         | Model | HyperParameter | AUC  |
+-----------------------------+-------+----------------+------+
|             BOW             | Brute |       71       | 61.0 |
|            Tf-Idf           | Brute |       89       | 58.0 |
|           AVG W2V           | Brute |      109       | 59.0 |
|          TFIDF W2V          | Brute |      109       | 60.0 |
| Top 2000 features of Tf-Idf | Brute |       99       | 58.0 |
+-----------------------------+-------+----------------+------+
```