Copy_of_3_DonorsChoose_KNN

March 11, 2019

1 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 p

How to scale current manual processes and resources to screen 500,000 projects so that they can be posted as quickly a
<li>How to increase the consistency of project vetting across different volunteers to improve the experience for teach
<li>How to focus volunteer time on the applications that need the most assistance</li>
</ul>

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.

1.1 About the DonorsChoose Data Set

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

Feature	Description
project_id	A unique identifier for the proposed project. **Example:** p036502

project_title | Title of the project. **Examples:**
Art Will Make You Happy!
First Grade Fun
project_grade_category | Grade level of students for which the project is targeted. One of the following enumerated values:
Grades PreK-2
Grades 3-5
Grades 6-8
Grades 9-12
project_subject_categories | One or more (comma-separated) subject categories for the project from the following enumerated list of values:

Applied Learning
Care & Hunger
Health & Sports
History & Civics
Literacy & Language
Math & Science
Music & The Arts
Special Needs
Warmth
Examples:
Music & The Arts
Literacy & Language, Math & Science

school_state | State where school is located (Two-letter U.S. postal code). **Example:** WY
project_subject_subcategories | One or more (comma-separated) subject subcategories for the project. **Examples:**
Literacy
Literature & Writing, Social Sciences

project_resource_summary | An explanation of the resources needed for the project. **Example:** My students need hands on literacy materials to manage sensory needs!

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

Feature	Description
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.

1.1.1 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

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

1.2 1.1 Reading Data

```
In [0]: #since im using google_colab, i have to mount the gdrive folder for accessing the files

        from google.colab import drive
        drive.mount('/content/gdrive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf

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

```
In [0]: #reading the datasets, i have taken only 5000 datapoints into consideration for avoiding mermory issues
```

```
project_data = pd.read_csv('/content/gdrive/My Drive/Colab Notebooks/Assignments_DonorsChoose_2
resource_data = pd.read_csv('/content/gdrive/My Drive/Colab Notebooks/Assignments_DonorsChoose_
```

```
In [0]: print("Number of data points in train data", project_data.shape)
        print('-'*50)
        print("The attributes of data :", project_data.columns.values)
```

```
Number of data points in train data (50000, 17)
--------------------------------------------------
The attributes of data : ['Unnamed: 0' 'id' 'teacher_id' 'teacher_prefix' 'school_state'
 'project_submitted_datetime' 'project_grade_category'
 'project_subject_categories' 'project_subject_subcategories'
 'project_title' 'project_essay_1' 'project_essay_2' 'project_essay_3'
 'project_essay_4' 'project_resource_summary'
 'teacher_number_of_previously_posted_projects' 'project_is_approved']
```

```
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[0]:        Unnamed: 0       id                        teacher_id teacher_prefix  \
        473        100660  p234804  cbc0e38f522143b86d372f8b43d4cff3           Mrs.
        41558       33679  p137682  06f6e62e17de34fcf81020c77549e1d5           Mrs.

              school_state                Date project_grade_category  \
        473             GA 2016-04-27 00:53:00          Grades PreK-2
        41558           WA 2016-04-27 01:05:25             Grades 3-5

              project_subject_categories project_subject_subcategories  \
        473             Applied Learning             Early Development
        41558        Literacy & Language                      Literacy

                                         project_title  \
        473    Flexible Seating for Flexible Learning
        41558  Going Deep: The Art of Inner Thinking!
```

```
                                        project_essay_1  \
473    I recently read an article about giving studen...
41558  My students crave challenge, they eat obstacle...

                                        project_essay_2  \
473    I teach at a low-income (Title 1) school. Ever...
41558  We are an urban, public k-5 elementary school...

                                        project_essay_3  \
473    We need a classroom rug that we can use as a c...
41558  With the new common core standards that have b...

                                        project_essay_4  \
473    Benjamin Franklin once said, \"Tell me and I f...
41558  These remarkable gifts will provide students w...

                                project_resource_summary  \
473    My students need flexible seating in the class...
41558  My students need copies of the New York Times ...

       teacher_number_of_previously_posted_projects  project_is_approved
473                                               2                    1
41558                                             2                    1
```

```
In [0]: print("Number of data points in train data", resource_data.shape)
        print(resource_data.columns.values)
        resource_data.head(2)
```

```
Number of data points in train data (1541272, 4)
['id' 'description' 'quantity' 'price']
```

```
Out[0]:         id                                       description  quantity  \
        0  p233245  LC652 - Lakeshore Double-Space Mobile Drying Rack         1
        1  p069063        Bouncy Bands for Desks (Blue support pipes)         3

            price
        0  149.00
        1   14.95
```

1.3 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
            j=j.replace('The','') # if we have the words "The" we are going to replace it with ''(i.e removing '
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"Math & Science"=>"Math&
        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]))
```

1.4 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
                    j=j.replace('The','') # if we have the words "The" we are going to replace it with ''(i.e removing '
                j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"Math & Science"=>"Math&
                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]))
```

1.5 1.3 Text preprocessing

In [0]:
```
# merge two column text dataframe:
project_data["essay"] = project_data["project_essay_1"].map(str) +\
                        project_data["project_essay_2"].map(str) + \
                        project_data["project_essay_3"].map(str) + \
                        project_data["project_essay_4"].map(str)
```

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

Out[0]:
```
       Unnamed: 0       id                        teacher_id teacher_prefix  \
473        100660  p234804  cbc0e38f522143b86d372f8b43d4cff3           Mrs.
41558       33679  p137682  06f6e62e17de34fcf81020c77549e1d5           Mrs.

      school_state                 Date project_grade_category  \
473             GA  2016-04-27 00:53:00          Grades PreK-2
41558           WA  2016-04-27 01:05:25             Grades 3-5

                                  project_title  \
473    Flexible Seating for Flexible Learning
41558  Going Deep: The Art of Inner Thinking!

                                         project_essay_1  \
473    I recently read an article about giving studen...
41558  My students crave challenge, they eat obstacle...

                                         project_essay_2  \
473    I teach at a low-income (Title 1) school. Ever...
41558  We are an urban, public k-5 elementary school...

                                         project_essay_3  \
473    We need a classroom rug that we can use as a c...
41558  With the new common core standards that have b...

                                         project_essay_4  \
473    Benjamin Franklin once said, \"Tell me and I f...
41558  These remarkable gifts will provide students w...

                                project_resource_summary  \
473    My students need flexible seating in the class...
41558  My students need copies of the New York Times ...
```

```
       teacher_number_of_previously_posted_projects  project_is_approved  \
473                                               2                    1
41558                                             2                    1

        clean_categories clean_subcategories  \
473      AppliedLearning    EarlyDevelopment
41558  Literacy_Language            Literacy

                                                   essay
473    I recently read an article about giving studen...
41558  My students crave challenge, they eat obstacle...
```

In [0]:
```
#### 1.4.2.3 Using Pretrained Models: TFIDF weighted W2V
```

In [0]:
```
# printing some random reviews
print(project_data['essay'].values[0])
print("="*50)
print(project_data['essay'].values[150])
print("="*50)
print(project_data['essay'].values[1000])
print("="*50)
print(project_data['essay'].values[20000])
print("="*50)
#print(project_data['essay'].values[99999])
#print("="*50)
```

```
I recently read an article about giving students a choice about how they learn. We already set goals; why not let t
==================================================
At the beginning of every class we start out with a Math Application problem to help students see the relevance o
==================================================
My students love coming to school and they love learning. I strive daily to make our classroom a relaxed, comfort
==================================================
I teach at a Title 1 school, with 73% of my students who receive free/reduced lunch. Our school provides free brea
==================================================
```

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

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

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

```
In [0]: sent = decontracted(project_data['essay'].values[20000])
        print(sent)
        print("="*50)
```

```
I teach at a Title 1 school, with 73% of my students who receive free/reduced lunch. Our school provides free brea
==================================================
```

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

```
I teach at a Title 1 school, with 73% of my students who receive free/reduced lunch. Our school provides free brea
```

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

```
I teach at a Title 1 school with 73 of my students who receive free reduced lunch Our school provides free breakfa
```

```
In [0]: # https://gist.github.com/sebleier/554280
        # we are removing the words from the stop words list: 'no', 'nor', 'not'
        stopwords= ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've",\
                    "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', \
                    'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their',\
                    'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', \
                    'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', \
                    'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', \
                    'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after',\
                    'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further',\
                    'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', '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', "
                    'won', "won't", 'wouldn', "wouldn't"]
```

```
In [0]: # Combining all the above stundents
        from tqdm import tqdm
        preprocessed_essays = []
        # tqdm is for printing the status bar
        for sentance in tqdm(project_data['essay'].values):
            sent = decontracted(sentance)
            sent = sent.replace('\\r', ' ')
            sent = sent.replace('\\"', ' ')
            sent = sent.replace('\\n', ' ')
            sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
            # https://gist.github.com/sebleier/554280
            sent = ' '.join(e for e in sent.split() if e.lower() not in stopwords)
            preprocessed_essays.append(sent.lower().strip())
```

```
100%|| 50000/50000 [00:29<00:00, 1681.15it/s]
```

```
In [0]: # after preprocesing
        preprocessed_essays[20000]
```

```
Out[0]: 'teach title 1 school 73 students receive free reduced lunch school provides free breakfast students special
```

1.4 Preprocessing of project_title

```
In [0]: # similarly you can preprocess the titles also

        from tqdm import tqdm
        preprocessed_project_title = []
        # tqdm is for printing the status bar
        for sentance in tqdm(project_data['project_title'].values):
            sent = decontracted(sentance)
            sent = sent.replace('\\r', ' ')
            sent = sent.replace('\\"', ' ')
            sent = sent.replace('\\n', ' ')
            sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
            # https://gist.github.com/sebleier/554280
            sent = ' '.join(e for e in sent.split() if e not in stopwords)
            preprocessed_project_title.append(sent.lower().strip())
```

```
100%|| 50000/50000 [00:01<00:00, 36446.36it/s]
```

1.6 1.5 Preparing data for models

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

```
Out[0]: Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
               'Date', 'project_grade_category', 'project_title', 'project_essay_1',
               'project_essay_2', 'project_essay_3', 'project_essay_4',
        'project_resource_summary',
        'teacher_number_of_previously_posted_projects', 'project_is_approved',
        'clean_categories', 'clean_subcategories', 'essay'],
       dtype='object')
```

we are going to consider

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

- project_title : text data
- text : text data
- project_resource_summary: text data (optinal)

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

1.6.1 Modifying DataSet (essay & project_title)

```
In [0]: project_data['clean_essay'] = preprocessed_essays
        project_data['clean_project_title'] = preprocessed_project_title
        project_data.drop(['essay'], axis=1, inplace=True)
        project_data.drop(['project_title'], axis=1, inplace=True)
```

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

```
Out[0]:      Unnamed: 0       id                        teacher_id teacher_prefix  \
        473      100660  p234804  cbc0e38f522143b86d372f8b43d4cff3           Mrs.

            school_state                 Date project_grade_category  \
        473           GA  2016-04-27 00:53:00          Grades PreK-2

                                             project_essay_1  \
        473  I recently read an article about giving studen...

                                             project_essay_2  \
        473  I teach at a low-income (Title 1) school. Ever...

                                             project_essay_3  \
        473  We need a classroom rug that we can use as a c...

                                             project_essay_4  \
        473  Benjamin Franklin once said, \"Tell me and I f...

                                    project_resource_summary  \
473  My students need flexible seating in the class...

     teacher_number_of_previously_posted_projects  project_is_approved  \
473                                             2                    1

    clean_categories clean_subcategories  \
473  AppliedLearning    EarlyDevelopment

                                           clean_essay  \
473  recently read article giving students choice l...

                      clean_project_title
473  flexible seating flexible learning
```

1.7 Spliiting DataSet

```
In [0]: y = project_data['project_is_approved'].values
        project_data.drop(['project_is_approved'], axis=1, inplace=True)
        project_data.head(1)
```

```
Out[0]:      Unnamed: 0       id                        teacher_id teacher_prefix  \
        473      100660  p234804  cbc0e38f522143b86d372f8b43d4cff3           Mrs.

    school_state                 Date project_grade_category  \
473           GA 2016-04-27 00:53:00          Grades PreK-2

                                       project_essay_1  \
473  I recently read an article about giving studen...

                                       project_essay_2  \
473  I teach at a low-income (Title 1) school. Ever...

                                       project_essay_3  \
473  We need a classroom rug that we can use as a c...

                                       project_essay_4  \
473  Benjamin Franklin once said, \"Tell me and I f...

                              project_resource_summary  \
473  My students need flexible seating in the class...

     teacher_number_of_previously_posted_projects clean_categories  \
473                                             2  AppliedLearning

    clean_subcategories                                        clean_essay  \
473    EarlyDevelopment  recently read article giving students choice l...

                      clean_project_title
```

```
473  flexible seating flexible learning
```

```
In [0]: X = project_data
```

```
In [0]: # please write all the code with proper documentation, and proper titles for each subsection
        # go through documentations and blogs before you start coding
        # first figure out what to do, and then think about how to do.
        # reading and understanding error messages will be very much helpfull in debugging your code
        # when you plot any graph make sure you use
            # a. Title, that describes your plot, this will be very helpful to the reader
            # b. Legends if needed
            # c. X-axis label
            # d. Y-axis label


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

1.7.1 1.5.1 Vectorizing Categorical data

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

1.7.2 clean_categories

```
In [0]: vectorizer = CountVectorizer()
        vectorizer.fit(X_train['clean_categories'].values) # fit has to happen only on train data

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

        print("After vectorizations")
        print(X_train_cc_ohe.shape, y_train.shape)
        print(X_cv_cc_ohe.shape, y_cv.shape)
        print(X_test_cc_ohe.shape, y_test.shape)
        print(vectorizer.get_feature_names())
        print("="*100)
```

```
After vectorizations
(22445, 9) (22445,)
(11055, 9) (11055,)
(16500, 9) (16500,)
['appliedlearning', 'care_hunger', 'health_sports', 'history_civics', 'literacy_language', 'math_science', 'music_a
==========================================================================
```

1.7.3 clean_subcategories

```
In [0]: vectorizer = CountVectorizer()
        vectorizer.fit(X_train['clean_subcategories'].values) # fit has to happen only on train data

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

        print("After vectorizations")
        print(X_train_csc_ohe.shape, y_train.shape)
        print(X_cv_csc_ohe.shape, y_cv.shape)
        print(X_test_csc_ohe.shape, y_test.shape)
        print(vectorizer.get_feature_names())
        print("="*100)
```

```
After vectorizations
(22445, 30) (22445,)
(11055, 30) (11055,)
(16500, 30) (16500,)
['appliedsciences', 'care_hunger', 'charactereducation', 'civics_government', 'college_careerprep', 'communityserv
==========================================================================
```

1.7.4 school_state

```
In [0]: vectorizer = CountVectorizer()
        vectorizer.fit(X_train['school_state'].values) # fit has to happen only on train data

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

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

```
After vectorizations
(22445, 51) (22445,)
(11055, 51) (11055,)
(16500, 51) (16500,)
['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl', 'ga', 'hi', 'ia', 'id', 'il', 'in', 'ks', 'ky', 'la', 'ma', 'md', 'me', 'mi', 'm
==========================================================================
```

1.7.5 teacher_prefix

```
In [0]: vectorizer = CountVectorizer()
        vectorizer.fit(X_train['teacher_prefix'].values.astype('U')) # fit has to happen only on train data

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

        print("After vectorizations")
        print(X_train_teacher_ohe.shape, y_train.shape)
        print(X_cv_teacher_ohe.shape, y_cv.shape)
        print(X_test_teacher_ohe.shape, y_test.shape)
        print(vectorizer.get_feature_names())
        print("="*100)
```

```
After vectorizations
(22445, 6) (22445,)
(11055, 6) (11055,)
(16500, 6) (16500,)
['dr', 'mr', 'mrs', 'ms', 'nan', 'teacher']
==============================================================================
```

1.7.6 project_grade_category

```
In [0]: vectorizer = CountVectorizer()
        vectorizer.fit(X_train['project_grade_category'].values) # fit has to happen only on train data

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

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

```
After vectorizations
(22445, 3) (22445,)
(11055, 3) (11055,)
(16500, 3) (16500,)
['12', 'grades', 'prek']
==============================================================================
```

1.7.7 1.5.2 Vectorizing Text data

1.5.2.1 Bag of words

1.7.8 essays

```
In [0]: from sklearn.feature_extraction.text import CountVectorizer
        vectorizer = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=50000)
        vectorizer.fit(X_train['clean_essay'].values) # fit has to happen only on train data

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

        print("After vectorizations")
        print(X_train_essay_bow.shape, y_train.shape)
        print(X_cv_essay_bow.shape, y_cv.shape)
        print(X_test_essay_bow.shape, y_test.shape)
        print("="*100)
```

```
After vectorizations
(22445, 50000) (22445,)
(11055, 50000) (11055,)
(16500, 50000) (16500,)
====================================================================
```

1.7.9 project_title

```
In [0]: from sklearn.feature_extraction.text import CountVectorizer
        vectorizer = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=50000)
        vectorizer.fit(X_train['clean_project_title'].values) # fit has to happen only on train data

        # we use the fitted CountVectorizer to convert the text to vector
        X_train_pt_bow = vectorizer.transform(X_train['clean_project_title'].values)
        X_cv_pt_bow = vectorizer.transform(X_cv['clean_project_title'].values)
        X_test_pt_bow = vectorizer.transform(X_test['clean_project_title'].values)

        print("After vectorizations")
        print(X_train_pt_bow.shape, y_train.shape)
        print(X_cv_pt_bow.shape, y_cv.shape)
        print(X_test_pt_bow.shape, y_test.shape)
        print("="*100)
```

```
After vectorizations
(22445, 1967) (22445,)
(11055, 1967) (11055,)
(16500, 1967) (16500,)
```

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

1.5.2.2 TFIDF vectorizer

1.7.10 essays

```
In [0]: from sklearn.feature_extraction.text import TfidfVectorizer
        vectorizer = TfidfVectorizer(min_df=10)
        X_train_essay_tfidf = vectorizer.fit_transform(X_train['clean_essay'].values)
        X_cv_essay_tfidf = vectorizer.transform(X_cv['clean_essay'].values)
        X_test_essay_tfidf = vectorizer.transform(X_test['clean_essay'].values)




        print("Shape of matrix after one hot encodig ",X_train_essay_tfidf.shape)
        print("Shape of matrix after one hot encodig ",X_cv_essay_tfidf.shape)
        print("Shape of matrix after one hot encodig ",X_test_essay_tfidf.shape)
```

```
Shape of matrix after one hot encodig  (22445, 8754)
Shape of matrix after one hot encodig  (11055, 8754)
Shape of matrix after one hot encodig  (16500, 8754)
```

1.7.11 project_title

```
In [0]: from sklearn.feature_extraction.text import TfidfVectorizer
        vectorizer = TfidfVectorizer(min_df=10)
        X_train_pt_tfidf = vectorizer.fit_transform(X_train['clean_project_title'].values)
        X_cv_pt_tfidf = vectorizer.transform(X_cv['clean_project_title'].values)
        X_test_pt_tfidf = vectorizer.transform(X_test['clean_project_title'].values)




        print("Shape of matrix after one hot encodig ",X_train_pt_tfidf.shape)
        print("Shape of matrix after one hot encodig ",X_cv_pt_tfidf.shape)
        print("Shape of matrix after one hot encodig ",X_test_pt_tfidf.shape)
```

```
Shape of matrix after one hot encodig  (22445, 1222)
Shape of matrix after one hot encodig  (11055, 1222)
Shape of matrix after one hot encodig  (16500, 1222)
```

1.5.2.3 Using Pretrained Models: Avg W2V

1.7.12 essays

Train

```
In [0]: i=0
        list_of_sentanceTrain=[]
        for sentance in X_train['clean_essay']:
            list_of_sentanceTrain.append(sentance.split())
```

```
In [0]: is_your_ram_gt_16g=False
        want_to_use_google_w2v = False
        want_to_train_w2v = True

        if want_to_train_w2v:
            # min_count = 5 considers only words that occured atleast 5 times
            w2v_model=Word2Vec(list_of_sentanceTrain,min_count=5,size=50, workers=4)
            print(w2v_model.wv.most_similar('great'))
            print('='*50)
            #print(w2v_model.wv.most_similar('worst'))

        elif want_to_use_google_w2v and is_your_ram_gt_16g:
            if os.path.isfile('GoogleNews-vectors-negative300.bin'):
                w2v_model=KeyedVectors.load_word2vec_format('GoogleNews-vectors-negative300.bin', binary=Tr
                #print(w2v_model.wv.most_similar('great'))
                #print(w2v_model.wv.most_similar('worst'))
            else:
                print("you don't have gogole's word2vec file, keep want_to_train_w2v = True, to train your own w2
```

```
[('wonderful', 0.7205535173416138), ('amazing', 0.697180449962616), ('awesome', 0.6534806489944458), ('incredibl
==================================================
```

```
In [0]: w2v_words = list(w2v_model.wv.vocab)
        print("number of words that occured minimum 5 times ",len(w2v_words))
        print("sample words ", w2v_words[0:50])
```

```
number of words that occured minimum 5 times  12511
sample words  ['team', 'fabulous', 'call', 'class', 'students', 'eager', 'learn', 'soak', 'information', 'like', 'sponges', 'e
```

```
In [0]: sent_vectorsPPE_train = []; # the avg-w2v for each sentence/review is stored in this list
        for sent in tqdm(list_of_sentanceTrain): # for each review/sentence
            sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might need to change this to 300 i
            cnt_words =0; # num of words with a valid vector in the sentence/review
            for word in sent: # for each word in a review/sentence
                if word in w2v_words:
                    vec = w2v_model.wv[word]
                    sent_vec += vec
                    cnt_words += 1
            if cnt_words != 0:
                sent_vec /= cnt_words
            sent_vectorsPPE_train.append(sent_vec)
        print(len(sent_vectorsPPE_train))
        print(len(sent_vectorsPPE_train[0]))
```

```
100%|| 22445/22445 [02:17<00:00, 163.52it/s]

22445
50
```

CV

```
In [0]: i=0
        list_of_sentanceCV=[]
        for sentance in X_cv['clean_essay']:
            list_of_sentanceCV.append(sentance.split())
```

```
In [0]: is_your_ram_gt_16g=False
        want_to_use_google_w2v = False
        want_to_train_w2v = True

        if want_to_train_w2v:
            # min_count = 5 considers only words that occured atleast 5 times
            w2v_model=Word2Vec(list_of_sentanceCV,min_count=5,size=50, workers=4)
            print(w2v_model.wv.most_similar('great'))
            print('='*50)
            print(w2v_model.wv.most_similar('worst'))

        elif want_to_use_google_w2v and is_your_ram_gt_16g:
            if os.path.isfile('GoogleNews-vectors-negative300.bin'):
                w2v_model=KeyedVectors.load_word2vec_format('GoogleNews-vectors-negative300.bin', binary=Tr
                print(w2v_model.wv.most_similar('great'))
                print(w2v_model.wv.most_similar('worst'))
            else:
                print("you don't have gogole's word2vec file, keep want_to_train_w2v = True, to train your own w2
```

```
[('amazing', 0.6580399870872498), ('wonderful', 0.6261288523674011), ('excellent', 0.6008976101875305), ('awesom
==================================================
[('1500', 0.8865181803703308), ('forests', 0.874229907989502), ('upheaval', 0.8732448220252991), ('shops', 0.87287
```

```
In [0]: w2v_words = list(w2v_model.wv.vocab)
        print("number of words that occured minimum 5 times ",len(w2v_words))
        print("sample words ", w2v_words[0:50])
```

```
number of words that occured minimum 5 times  9409
sample words  ['work', 'hard', 'play', 'favorite', 'saying', 'students', 'come', 'vast', 'backgrounds', 'walk', 'day', 'sm
```

```
In [0]: sent_vectorsPPE_cv = []; # the avg-w2v for each sentence/review is stored in this list
        for sent in tqdm(list_of_sentanceCV): # for each review/sentence
            sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might need to change this to 300 i
            cnt_words =0; # num of words with a valid vector in the sentence/review
            for word in sent: # for each word in a review/sentence
                if word in w2v_words:
                    vec = w2v_model.wv[word]
                    sent_vec += vec
                    cnt_words += 1
            if cnt_words != 0:
                sent_vec /= cnt_words
            sent_vectorsPPE_cv.append(sent_vec)
        print(len(sent_vectorsPPE_cv))
        print(len(sent_vectorsPPE_cv[0]))
```

```
100%|| 11055/11055 [00:45<00:00, 241.78it/s]

11055
50
```

Test

```
In [0]: i=0
        list_of_sentanceTest=[]
        for sentance in X_test['clean_essay']:
            list_of_sentanceTest.append(sentance.split())
```

```
In [0]: is_your_ram_gt_16g=False
        want_to_use_google_w2v = False
        want_to_train_w2v = True

        if want_to_train_w2v:
            # min_count = 5 considers only words that occured atleast 5 times
            w2v_model=Word2Vec(list_of_sentanceTest,min_count=5,size=50, workers=4)
            print(w2v_model.wv.most_similar('great'))
            print('='*50)
            print(w2v_model.wv.most_similar('worst'))

        elif want_to_use_google_w2v and is_your_ram_gt_16g:
            if os.path.isfile('GoogleNews-vectors-negative300.bin'):
                w2v_model=KeyedVectors.load_word2vec_format('GoogleNews-vectors-negative300.bin', binary=Tr
            print(w2v_model.wv.most_similar('great'))
            print(w2v_model.wv.most_similar('worst'))
        else:
            print("you don't have gogole's word2vec file, keep want_to_train_w2v = True, to train your own w2
```

```
[('excellent', 0.6836485266685486), ('amazing', 0.67774897813797), ('wonderful', 0.6578258275985718), ('incredible
==================================================
[('tn', 0.8603484034538269), ('maine', 0.8542816638946533), ('floods', 0.84330153465271), ('ridden', 0.83880484104
```

```
In [0]: w2v_words = list(w2v_model.wv.vocab)
        print("number of words that occured minimum 5 times ",len(w2v_words))
        print("sample words ", w2v_words[0:50])
```

```
number of words that occured minimum 5 times  11077
sample words  ['students', 'school', 'come', 'extremely', 'high', 'poverty', 'area', 'many', 'problems', 'even', 'challen
```

```
In [0]: sent_vectorsPPE_test = []; # the avg-w2v for each sentence/review is stored in this list
        for sent in tqdm(list_of_sentanceTest): # for each review/sentence
            sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might need to change this to 300 i
            cnt_words =0; # num of words with a valid vector in the sentence/review
            for word in sent: # for each word in a review/sentence
                if word in w2v_words:
                    vec = w2v_model.wv[word]
                    sent_vec += vec
                    cnt_words += 1
            if cnt_words != 0:
                sent_vec /= cnt_words
            sent_vectorsPPE_test.append(sent_vec)
        print(len(sent_vectorsPPE_test))
        print(len(sent_vectorsPPE_test[0]))
```

```
100%|| 16500/16500 [01:14<00:00, 222.05it/s]

16500
50
```

1.7.13 project_title

Train

```
In [0]: # Similarly you can vectorize for title also
```

```
        # Train your own Word2Vec model using your own text corpus
        i=0
        list_of_sentancePTtrain=[]
        for sentance in X_train['clean_project_title']:
            list_of_sentancePTtrain.append(sentance.split())
```

```
In [0]: is_your_ram_gt_16g=False
        want_to_use_google_w2v = False
        want_to_train_w2v = True

        try :


          if want_to_train_w2v:
            # min_count = 5 considers only words that occured atleast 5 times
            w2v_model=Word2Vec(list_of_sentancePTtrain,min_count=5,size=50, workers=4)
            print(w2v_model.wv.most_similar('great'))
            print('='*50)
            print(w2v_model.wv.most_similar('worst'))

          elif want_to_use_google_w2v and is_your_ram_gt_16g:
            if os.path.isfile('GoogleNews-vectors-negative300.bin'):
                w2v_model=KeyedVectors.load_word2vec_format('GoogleNews-vectors-negative300.bin', binary=Tr
                print(w2v_model.wv.most_similar('great'))
                print(w2v_model.wv.most_similar('worst'))
            else:
                print("you don't have gogole's word2vec file, keep want_to_train_w2v = True, to train your own w2

        except KeyError :

          pass

        finally :

          print("Execution Done")
```

```
[('day', 0.998775839805603), ('off', 0.9985997676849365), ('right', 0.9985504746437073), ('sense', 0.99850177764892
==================================================
Execution Done
```

```
In [0]: w2v_words = list(w2v_model.wv.vocab)
        print("number of words that occured minimum 5 times ",len(w2v_words))
        print("sample words ", w2v_words[0:50])
```

```
number of words that occured minimum 5 times  2168
sample words  ['chevron', 'fuel', 'my', 'school', 'paperless', 'classroom', 'ms', 'first', 'grade', 'class', 'buzzing', 'techn
```

```
In [0]: sent_vectorsPT_train = []; # the avg-w2v for each sentence/review is stored in this list
        for sent in tqdm(list_of_sentancePTtrain): # for each review/sentence
            sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might need to change this to 300 i
            cnt_words =0; # num of words with a valid vector in the sentence/review
            for word in sent: # for each word in a review/sentence
                if word in w2v_words:
                    vec = w2v_model.wv[word]
                    sent_vec += vec
                    cnt_words += 1
            if cnt_words != 0:
                sent_vec /= cnt_words
            sent_vectorsPT_train.append(sent_vec)
        print(len(sent_vectorsPT_train))
        print(len(sent_vectorsPT_train[0]))
```

```
100%|| 22445/22445 [00:01<00:00, 11781.52it/s]

22445
50
```

CV

```
In [0]: i=0
        list_of_sentancePT_cv=[]
        for sentance in X_cv['clean_project_title']:
            list_of_sentancePT_cv.append(sentance.split())
```

```
In [0]: is_your_ram_gt_16g=False
        want_to_use_google_w2v = False
        want_to_train_w2v = True

        try :

          if want_to_train_w2v:
            # min_count = 5 considers only words that occured atleast 5 times
            w2v_model=Word2Vec(list_of_sentancePT_cv,min_count=5,size=50, workers=4)
            print(w2v_model.wv.most_similar('great'))
            print('='*50)
            print(w2v_model.wv.most_similar('worst'))

          elif want_to_use_google_w2v and is_your_ram_gt_16g:
            if os.path.isfile('GoogleNews-vectors-negative300.bin'):
                w2v_model=KeyedVectors.load_word2vec_format('GoogleNews-vectors-negative300.bin', binary=Tr
                print(w2v_model.wv.most_similar('great'))
                print(w2v_model.wv.most_similar('worst'))
            else:
                print("you don't have gogole's word2vec file, keep want_to_train_w2v = True, to train your own w2

        except KeyError :

          pass


        finally :

          print("Execution Done")
```

```
[('create', 0.9996914267539978), ('brains', 0.9996170401573181), ('project', 0.9996063113212585), ('your', 0.999589
==========================================================
Execution Done
```

```
In [0]: w2v_words = list(w2v_model.wv.vocab)
        print("number of words that occured minimum 5 times ",len(w2v_words))
        print("sample words ", w2v_words[0:50])
```

```
number of words that occured minimum 5 times  1312
sample words  ['technology', 'techies', 'future', 'flexible', 'seating', 'helps', 'us', 'reading', 'blended', 'learning', 'firs
```

```
In [0]: sent_vectorsPT_cv = []; # the avg-w2v for each sentence/review is stored in this list
        for sent in tqdm(list_of_sentancePT_cv): # for each review/sentence
            sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might need to change this to 300 i
            cnt_words =0; # num of words with a valid vector in the sentence/review
            for word in sent: # for each word in a review/sentence
                if word in w2v_words:
                    vec = w2v_model.wv[word]
                    sent_vec += vec
                    cnt_words += 1
            if cnt_words != 0:
                sent_vec /= cnt_words
            sent_vectorsPT_cv.append(sent_vec)
        print(len(sent_vectorsPT_cv))
        print(len(sent_vectorsPT_cv[0]))
```

```
100%|| 11055/11055 [00:00<00:00, 14934.56it/s]

11055
50
```

Test

```
In [0]: i=0
        list_of_sentancePT_test=[]
        for sentance in X_test['clean_project_title']:
            list_of_sentancePT_test.append(sentance.split())
```

```
In [0]: # Using Google News Word2Vectors

        # in this project we are using a pretrained model by google
        # its 3.3G file, once you load this into your memory
        # it occupies ~9Gb, so please do this step only if you have >12G of ram
        # we will provide a pickle file wich contains a dict ,
        # and it contains all our courpus words as keys and  model[word] as values
        # To use this code-snippet, download "GoogleNews-vectors-negative300.bin"
        # from https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edit
        # it's 1.9GB in size.


        # http://kavita-ganesan.com/gensim-word2vec-tutorial-starter-code/#.W17SRFAzZPY
        # you can comment this whole cell
        # or change these varible according to your need

        is_your_ram_gt_16g=False
        want_to_use_google_w2v = False
        want_to_train_w2v = True

        if want_to_train_w2v:
            # min_count = 5 considers only words that occured atleast 5 times
            w2v_model=Word2Vec(list_of_sentancePT_test,min_count=5,size=50, workers=4)
            print(w2v_model.wv.most_similar('great'))
            print('='*50)
            #print(w2v_model.wv.most_similar('worst'))

        elif want_to_use_google_w2v and is_your_ram_gt_16g:
            if os.path.isfile('GoogleNews-vectors-negative300.bin'):
                w2v_model=KeyedVectors.load_word2vec_format('GoogleNews-vectors-negative300.bin', binary=Tr
                print(w2v_model.wv.most_similar('great'))
                print(w2v_model.wv.most_similar('worst'))
            else:
                print("you don't have gogole's word2vec file, keep want_to_train_w2v = True, to train your own w2
```

```
[('table', 0.9995646476745605), ('club', 0.9995392560958862), ('computer', 0.9995108246803284), ('literature', 0.99
==================================================
```

```
In [0]: w2v_words = list(w2v_model.wv.vocab)
        print("number of words that occured minimum 5 times ",len(w2v_words))
        print("sample words ", w2v_words[0:50])
```

```
number of words that occured minimum 5 times  1784
sample words  ['creating', 'having', 'fun', 'makerspace', 'classroom', 'seating', '2017', 'with', 'mobile', 'shelves', 'sp
```

```
In [0]: # average Word2Vec
        # compute average word2vec for each review.
        sent_vectorsPT_test = []; # the avg-w2v for each sentence/review is stored in this list
        for sent in tqdm(list_of_sentancePT_test): # for each review/sentence
            sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might need to change this to 300 i
            cnt_words =0; # num of words with a valid vector in the sentence/review
            for word in sent: # for each word in a review/sentence
                if word in w2v_words:
                    vec = w2v_model.wv[word]
                    sent_vec += vec
                    cnt_words += 1
            if cnt_words != 0:
                sent_vec /= cnt_words
            sent_vectorsPT_test.append(sent_vec)
        print(len(sent_vectorsPT_test))
        print(len(sent_vectorsPT_test[0]))
```

```
100%|| 16500/16500 [00:01<00:00, 12495.29it/s]

16500
50
```

1.5.2.3 Using Pretrained Models: TFIDF weighted W2V

1.7.14 essays

Train

```
In [0]: # S = ["abc def pqr", "def def def abc", "pqr pqr def"]
        model = TfidfVectorizer()
        model.fit(X_train['clean_essay'])
        # we are converting a dictionary with word as a key, and the idf as a value
        dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
```

```
In [0]: # TF-IDF weighted Word2Vec
        tfidf_feat = model.get_feature_names() # tfidf words/col-names
        # final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val = tfidf

        tfidf_sent_vectors_essay_train = []; # the tfidf-w2v for each sentence/review is stored in this list
        row=0;
        for sent in tqdm(list_of_sentanceTrain): # for each review/sentence
```

```
    sent_vec = np.zeros(50) # as word vectors are of zero length
    weight_sum =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]
#              tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
            # to reduce the computation we are
            # dictionary[word] = idf value of word in whole courpus
            # sent.count(word) = tf valeus of word in this review
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors_essay_train.append(sent_vec)
    row += 1
```

```
100%|| 22445/22445 [17:08<00:00, 24.11it/s]
```

CV

```
In [0]: # S = ["abc def pqr", "def def def abc", "pqr pqr def"]
        model = TfidfVectorizer()
        model.fit(X_cv['clean_essay'])
        # we are converting a dictionary with word as a key, and the idf as a value
        dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
```

```
In [0]: # TF-IDF weighted Word2Vec
        tfidf_feat = model.get_feature_names() # tfidf words/col-names
        # final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val = tfidf

        tfidf_sent_vectors_essay_cv = []; # the tfidf-w2v for each sentence/review is stored in this list
        row=0;
        for sent in tqdm(list_of_sentanceCV): # for each review/sentence
            sent_vec = np.zeros(50) # as word vectors are of zero length
            weight_sum =0; # num of words with a valid vector in the sentence/review
            for word in sent: # for each word in a review/sentence
                if word in w2v_words and word in tfidf_feat:
                    vec = w2v_model.wv[word]
        #              tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
                    # to reduce the computation we are
                    # dictionary[word] = idf value of word in whole courpus
                    # sent.count(word) = tf valeus of word in this review
                    tf_idf = dictionary[word]*(sent.count(word)/len(sent))
                    sent_vec += (vec * tf_idf)
                    weight_sum += tf_idf
            if weight_sum != 0:
```

```
            sent_vec /= weight_sum
        tfidf_sent_vectors_essay_cv.append(sent_vec)
        row += 1
```

```
100%|| 11055/11055 [06:16<00:00, 29.39it/s]
```

Test

```
In [0]: # S = ["abc def pqr", "def def def abc", "pqr pqr def"]
        model = TfidfVectorizer()
        model.fit(X_test['clean_essay'])
        # we are converting a dictionary with word as a key, and the idf as a value
        dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
```

```
In [0]: # TF-IDF weighted Word2Vec
        tfidf_feat = model.get_feature_names() # tfidf words/col-names
        # final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val = tfidf

        tfidf_sent_vectors_essay_test = []; # the tfidf-w2v for each sentence/review is stored in this list
        row=0;
        for sent in tqdm(list_of_sentanceTest): # for each review/sentence
            sent_vec = np.zeros(50) # as word vectors are of zero length
            weight_sum =0; # num of words with a valid vector in the sentence/review
            for word in sent: # for each word in a review/sentence
                if word in w2v_words and word in tfidf_feat:
                    vec = w2v_model.wv[word]
        #               tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
                    # to reduce the computation we are
                    # dictionary[word] = idf value of word in whole courpus
                    # sent.count(word) = tf valeus of word in this review
                    tf_idf = dictionary[word]*(sent.count(word)/len(sent))
                    sent_vec += (vec * tf_idf)
                    weight_sum += tf_idf
            if weight_sum != 0:
                sent_vec /= weight_sum
            tfidf_sent_vectors_essay_test.append(sent_vec)
            row += 1
```

```
100%|| 16500/16500 [11:32<00:00, 23.83it/s]
```

1.7.15 project_title

Train

```
In [0]: # S = ["abc def pqr", "def def def abc", "pqr pqr def"]
        model = TfidfVectorizer()
        model.fit(X_train['clean_project_title'])
        # we are converting a dictionary with word as a key, and the idf as a value
        dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
```

```
In [0]: # TF-IDF weighted Word2Vec
        tfidf_feat = model.get_feature_names() # tfidf words/col-names
        # final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val = tfidf

        tfidf_sent_vectorsPT_train = []; # the tfidf-w2v for each sentence/review is stored in this list
        row=0;
        for sent in tqdm(list_of_sentancePTtrain): # for each review/sentence
            sent_vec = np.zeros(50) # as word vectors are of zero length
            weight_sum =0; # num of words with a valid vector in the sentence/review
            for word in sent: # for each word in a review/sentence
                if word in w2v_words and word in tfidf_feat:
                    vec = w2v_model.wv[word]
        #             tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
                    # to reduce the computation we are
                    # dictionary[word] = idf value of word in whole courpus
                    # sent.count(word) = tf valeus of word in this review
                    tf_idf = dictionary[word]*(sent.count(word)/len(sent))
                    sent_vec += (vec * tf_idf)
                    weight_sum += tf_idf
            if weight_sum != 0:
                sent_vec /= weight_sum
            tfidf_sent_vectorsPT_train.append(sent_vec)
            row += 1
```

```
100%|| 22445/22445 [00:12<00:00, 1778.82it/s]
```

CV

```
In [0]: # S = ["abc def pqr", "def def def abc", "pqr pqr def"]
        model = TfidfVectorizer()
        model.fit(X_cv['clean_project_title'])
        # we are converting a dictionary with word as a key, and the idf as a value
        dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
```

```
In [0]: # TF-IDF weighted Word2Vec
        tfidf_feat = model.get_feature_names() # tfidf words/col-names
        # final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val = tfidf

        tfidf_sent_vectorsPT_cv = []; # the tfidf-w2v for each sentence/review is stored in this list
        row=0;
        for sent in tqdm(list_of_sentancePT_cv): # for each review/sentence
            sent_vec = np.zeros(50) # as word vectors are of zero length
            weight_sum =0; # num of words with a valid vector in the sentence/review
            for word in sent: # for each word in a review/sentence
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]
#               tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
            # to reduce the computation we are
            # dictionary[word] = idf value of word in whole courpus
            # sent.count(word) = tf valeus of word in this review
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectorsPT_cv.append(sent_vec)
    row += 1
```

```
100%|| 11055/11055 [00:04<00:00, 2211.02it/s]
```

Test

```
In [0]: # S = ["abc def pqr", "def def def abc", "pqr pqr def"]
        model = TfidfVectorizer()
        model.fit(X_test['clean_project_title'])
        # we are converting a dictionary with word as a key, and the idf as a value
        dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
```

```
In [0]: # TF-IDF weighted Word2Vec
        tfidf_feat = model.get_feature_names() # tfidf words/col-names
        # final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val = tfidf

        tfidf_sent_vectorsPT_test = []; # the tfidf-w2v for each sentence/review is stored in this list
        row=0;
        for sent in tqdm(list_of_sentancePT_test): # for each review/sentence
            sent_vec = np.zeros(50) # as word vectors are of zero length
            weight_sum =0; # num of words with a valid vector in the sentence/review
            for word in sent: # for each word in a review/sentence
                if word in w2v_words and word in tfidf_feat:
                    vec = w2v_model.wv[word]
        #               tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
                    # to reduce the computation we are
                    # dictionary[word] = idf value of word in whole courpus
                    # sent.count(word) = tf valeus of word in this review
                    tf_idf = dictionary[word]*(sent.count(word)/len(sent))
                    sent_vec += (vec * tf_idf)
                    weight_sum += tf_idf
            if weight_sum != 0:
                sent_vec /= weight_sum
            tfidf_sent_vectorsPT_test.append(sent_vec)
            row += 1
```

```
100%|| 16500/16500 [00:08<00:00, 1892.90it/s]
```

1.7.16 1.5.3 Vectorizing Numerical features

1.7.17 price

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

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

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

```
After vectorizations
(22445, 1) (22445,)
(11055, 1) (11055,)
(16500, 1) (16500,)
============================================================================
```

1.7.18 tnppp

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

```
          normalizerT.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))

          X_train_tnppp_norm = normalizerT.transform(X_train['teacher_number_of_previously_posted_projec
          X_cv_tnppp_norm = normalizerT.transform(X_cv['teacher_number_of_previously_posted_projects'].va
          X_test_tnppp_norm = normalizerT.transform(X_test['teacher_number_of_previously_posted_projects'

          print("After vectorizations")
          print(X_train_tnppp_norm.shape, y_train.shape)
          print(X_cv_tnppp_norm.shape, y_cv.shape)
          print(X_test_tnppp_norm.shape, y_test.shape)
          print("="*100)
```

```
After vectorizations
(22445, 1) (22445,)
(11055, 1) (11055,)
(16500, 1) (16500,)
=====================================================================================
```

1.7.19 1.5.4 Merging all the above features

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

SET1

```
In [0]: # merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
          from scipy.sparse import hstack
          X_tr1 = hstack((X_train_cc_ohe, X_train_csc_ohe, X_train_grade_ohe, X_train_price_norm, X_tra
          X_cr1 = hstack((X_cv_cc_ohe, X_cv_csc_ohe, X_cv_grade_ohe, X_cv_price_norm, X_cv_tnppp_n
          X_te1 = hstack((X_test_cc_ohe, X_test_csc_ohe, X_test_grade_ohe, X_test_price_norm, X_test_tr

          print("Final Data matrix")
          print(X_tr1.shape, y_train.shape)
          print(X_cr1.shape, y_cv.shape)
          print(X_te1.shape, y_test.shape)
          print("="*100)
```

```
Final Data matrix
(22445, 52011) (22445,)
(11055, 52011) (11055,)
(16500, 52011) (16500,)
=====================================================================================
```

SET2

```
In [0]: # merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
          from scipy.sparse import hstack
          X_tr2 = hstack((X_train_cc_ohe, X_train_csc_ohe, X_train_grade_ohe, X_train_price_norm, X_tra
        X_cr2 = hstack((X_cv_cc_ohe, X_cv_csc_ohe, X_cv_grade_ohe, X_cv_price_norm, X_cv_tnppp_n
        X_te2 = hstack((X_test_cc_ohe, X_test_csc_ohe, X_test_grade_ohe, X_test_price_norm, X_test_tr

        print("Final Data matrix")
        print(X_tr2.shape, y_train.shape)
        print(X_cr2.shape, y_cv.shape)
        print(X_te2.shape, y_test.shape)
        print("="*100)
```

```
Final Data matrix
(22445, 10020) (22445,)
(11055, 10020) (11055,)
(16500, 10020) (16500,)
============================================================================================
```

SET3

```
In [0]: # merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
        from scipy.sparse import hstack
        X_tr3 = hstack((X_train_cc_ohe, X_train_csc_ohe, X_train_grade_ohe, X_train_price_norm, X_tra
        X_cr3 = hstack((X_cv_cc_ohe, X_cv_csc_ohe, X_cv_grade_ohe, X_cv_price_norm, X_cv_tnppp_n
        X_te3 = hstack((X_test_cc_ohe, X_test_csc_ohe, X_test_grade_ohe, X_test_price_norm, X_test_tr

        print("Final Data matrix")
        print(X_tr3.shape, y_train.shape)
        print(X_cr3.shape, y_cv.shape)
        print(X_te3.shape, y_test.shape)
        print("="*100)
```

```
Final Data matrix
(22445, 144) (22445,)
(11055, 144) (11055,)
(16500, 144) (16500,)
============================================================================================
```

SET4

```
In [0]: # merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
        from scipy.sparse import hstack
        X_tr4 = hstack((X_train_cc_ohe, X_train_csc_ohe, X_train_grade_ohe, X_train_price_norm, X_tra
        X_cr4 = hstack((X_cv_cc_ohe, X_cv_csc_ohe, X_cv_grade_ohe, X_cv_price_norm, X_cv_tnppp_n
        X_te4 = hstack((X_test_cc_ohe, X_test_csc_ohe, X_test_grade_ohe, X_test_price_norm, X_test_tr

        print("Final Data matrix")
        print(X_tr4.shape, y_train.shape)
        print(X_cr4.shape, y_cv.shape)
```

```
    print(X_te4.shape, y_test.shape)
    print("="*100)
```

```
Final Data matrix
(22445, 144) (22445,)
(11055, 144) (11055,)
(16500, 144) (16500,)
==========================================================================
```

2 Assignment 3: Apply KNN

```
<li><strong>[Task-1] Apply KNN(brute force version) on these feature sets</strong>
    <ul>
      <li><font color='red'>Set 1</font>: categorical, numerical features + project_title(BOW) + preprocessed_e
      <li><font color='red'>Set 2</font>: categorical, numerical features + project_title(TFIDF)+  preprocessed_
      <li><font color='red'>Set 3</font>: categorical, numerical features + project_title(AVG W2V)+  preprocesse
      <li><font color='red'>Set 4</font>: categorical, numerical features + project_title(TFIDF W2V)+  preproce
    </ul>
</li>
<br>
<li><strong>Hyper paramter tuning to find best K</strong>
    <ul>
<li>Find the best hyper parameter which results in the maximum <a href='https://www.appliedaicourse.com/cour
ai-course-online/lessons/receiver-operating-characteristic-curve-roc-curve-and-auc-
1/'>AUC</a> value</li>
<li>Find the best hyper paramter using k-fold cross validation (or) simple cross validation data</li>
<li>Use gridsearch-cv or randomsearch-cv or   write your own for loops to do this task</li>
    </ul>
</li>
<br>
<li>
<strong>Representation of results</strong>
    <ul>
<li>You need to plot the performance of model both on train data and cross validation data for each hyper parameter
<img src='train_cv_auc.JPG' width=300px></li>
<li>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.
<img src='train_test_auc.JPG' width=300px></li>
<li>Along with plotting ROC curve, you need to print the <a href='https://www.appliedaicourse.com/course/appli
ai-course-online/lessons/confusion-matrix-tpr-fpr-fnr-tnr-1/'>confusion matrix</a> with predicted and original lab
<img src='confusion_matrix.png' width=300px></li>
    </ul>
</li>
<li><strong> [Task-2] </strong>
    <ul>
```

<li>Select top 2000 features from feature <font color='red'>Set 2</font> using <a href='https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html'>`SelectKBest`</a>

and then apply KNN on top of these features

```
        <li>
            <pre>
            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)
            </pre>
        </li>
        <li>Repeat the steps 2 and 3 on the data matrix after feature selection</li>
    </ul>
</li>
<br>
<li><strong>Conclusion</strong>
    <ul>
<li>You need to summarize the results at the end of the notebook, summarize it in the table format. To print out a ta
    <img src='summary.JPG' width=400px>
</li>
    </ul>
```

Note: Data Leakage

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

2. K Nearest Neighbor

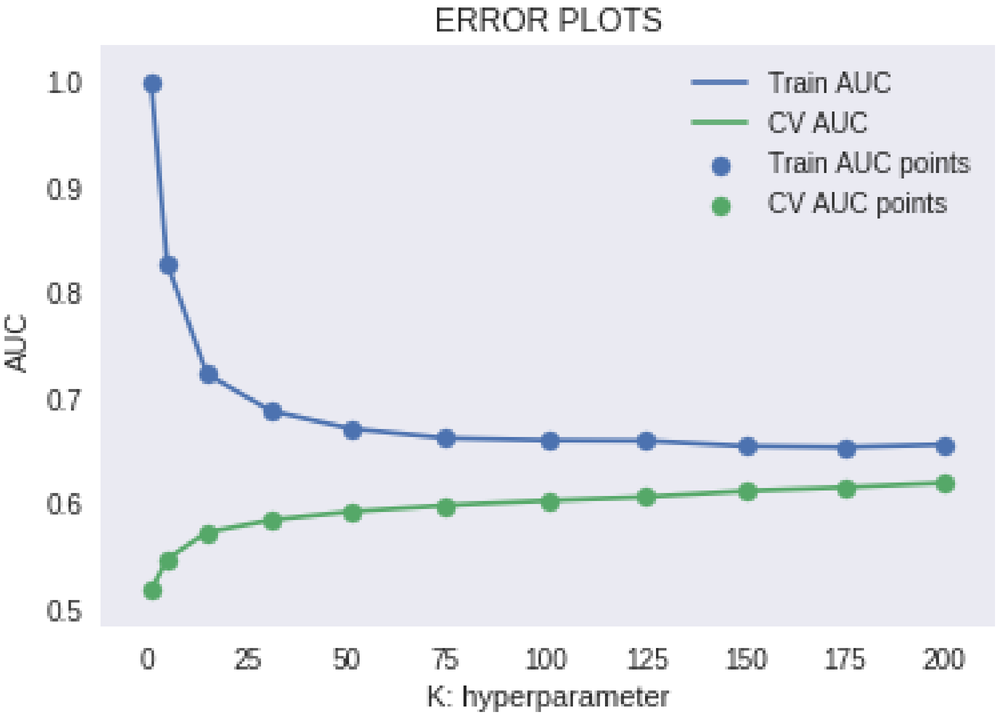
2.1 HyperParameter Tuning

2.1.1 SET1

```
In [0]: def batch_predict(clf, data):
            # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive cl
            # not the predicted outputs
```

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

    return y_data_pred
```

```
In [84]: import matplotlib.pyplot as plt
         from sklearn.neighbors import KNeighborsClassifier
         from sklearn.metrics import roc_auc_score
         """
         y_true : array, shape = [n_samples] or [n_samples, n_classes]
         True binary labels or binary label indicators.

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

         """

         train_auc1 = []
         cv_auc1 = []
         #K1 = [1, 5, 10, 15, 21, 31, 41, 51, 75, 101, 121, 151, 171]
         K1 = [1, 5, 15, 31, 51, 75, 101, 125, 150, 175, 200]

         for i in K1:
             neigh = KNeighborsClassifier(algorithm='brute', n_neighbors=i)
             neigh.fit(X_tr1, y_train)

             y_train_pred = batch_predict(neigh, X_tr1)
             y_cv_pred = batch_predict(neigh, X_cr1)

             # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive c
             # not the predicted outputs
             train_auc1.append(roc_auc_score(y_train,y_train_pred))
             cv_auc1.append(roc_auc_score(y_cv, y_cv_pred))


         plt.plot(K1, train_auc1, label='Train AUC')
         plt.plot(K1, cv_auc1, label='CV AUC')

         plt.scatter(K1, train_auc1, label='Train AUC points')
         plt.scatter(K1, cv_auc1, label='CV AUC points')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()


print(train_auc1)
print(cv_auc1)          #k_best = 150
```



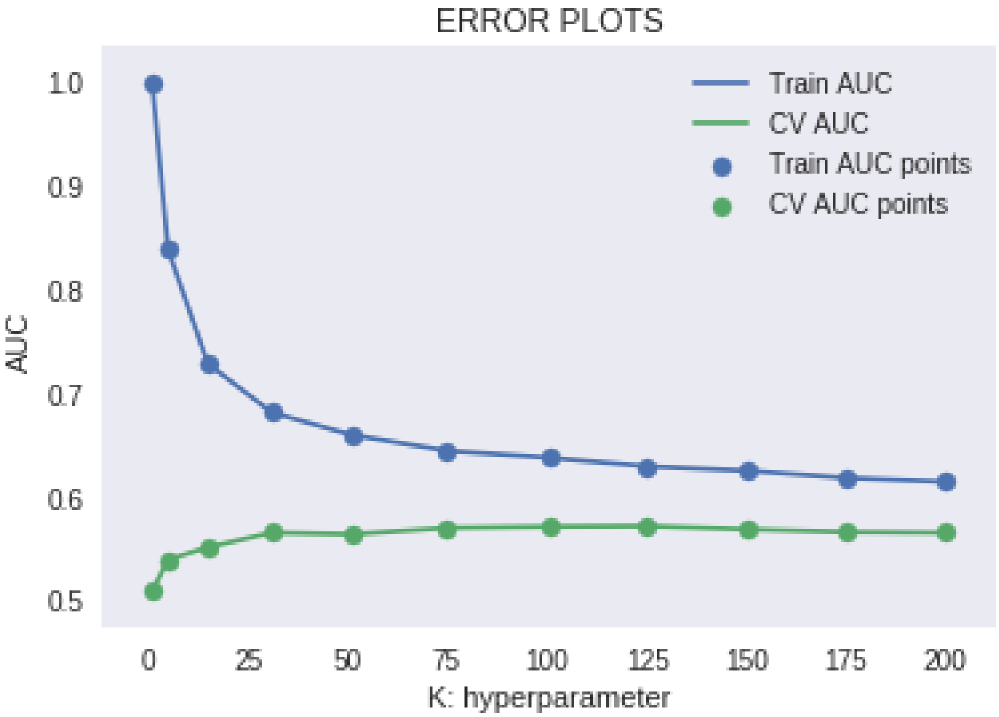
```
[0.9998556165174705, 0.8274489901568831, 0.7222711301218142, 0.6871828404817636, 0.670382032518428, 0.66128
[0.5167431430050027, 0.5467120610974329, 0.5719048410619914, 0.5836100177207646, 0.5915359041510498, 0.5976
```

2.1.2 SET2

```
In [0]: def batch_predict(clf, data):
            # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive cl
            # not the predicted outputs

            y_data_pred = []
            tr_loop = data.shape[0] - data.shape[0]%1000
```

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

    return y_data_pred
```

```
In [86]: import matplotlib.pyplot as plt
         from sklearn.neighbors import KNeighborsClassifier
         from sklearn.metrics import roc_auc_score
         """
         y_true : array, shape = [n_samples] or [n_samples, n_classes]
         True binary labels or binary label indicators.

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

         """

         train_auc2 = []
         cv_auc2 = []
         K2 = [1, 5, 15, 31, 51, 75, 101, 125, 150, 175, 200]
         for i in K2:
             neigh = KNeighborsClassifier(algorithm='brute', n_neighbors=i)
             neigh.fit(X_tr2, y_train)

             y_train_pred = batch_predict(neigh, X_tr2)
             y_cv_pred = batch_predict(neigh, X_cr2)

             # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive c
             # not the predicted outputs
             train_auc2.append(roc_auc_score(y_train,y_train_pred))
             cv_auc2.append(roc_auc_score(y_cv, y_cv_pred))


         plt.plot(K2, train_auc2, label='Train AUC')
         plt.plot(K2, cv_auc2, label='CV AUC')

         plt.scatter(K2, train_auc2, label='Train AUC points')
         plt.scatter(K2, cv_auc2, label='CV AUC points')

         plt.legend()
         plt.xlabel("K: hyperparameter")
         plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()

print(train_auc2)
print(cv_auc2)          #k_best = 150
```



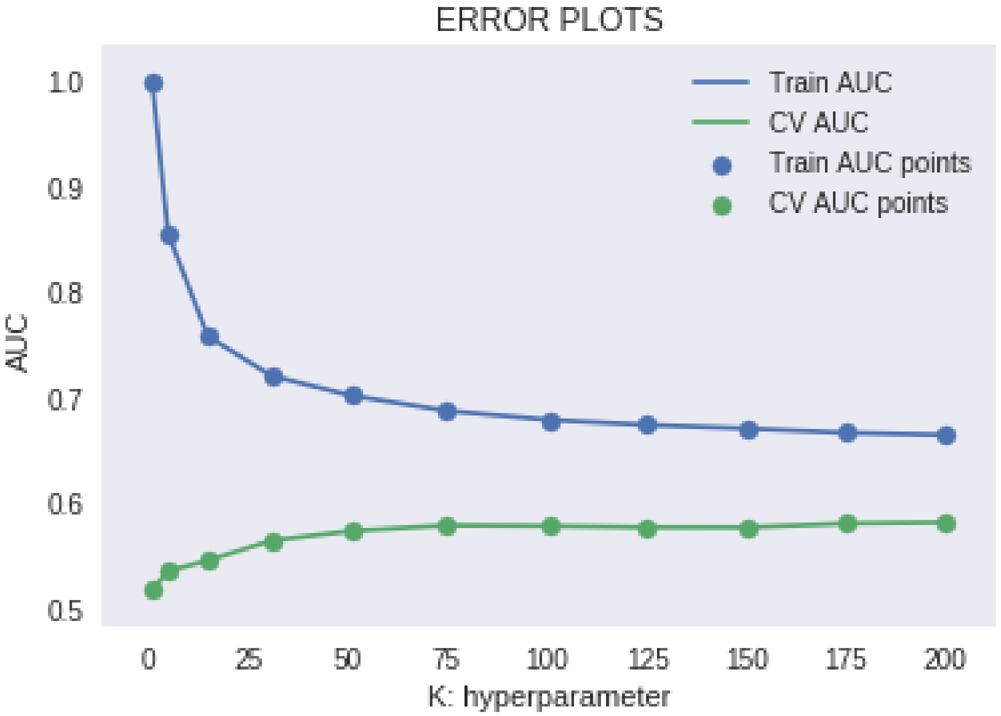
```
[0.9998556165174705, 0.8381776139244399, 0.7279712959977616, 0.6808104798159315, 0.659179419881741, 0.64385
[0.5080006900120753, 0.5388195775244248, 0.5506083083726692, 0.56478517101322, 0.5632044160772813, 0.569342
```

2.1.3 SET3

In [0]:
```
def batch_predict(clf, data):
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive cl
    # not the predicted outputs

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

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

        return y_data_pred
```

```
In [88]: import matplotlib.pyplot as plt
         from sklearn.neighbors import KNeighborsClassifier
         from sklearn.metrics import roc_auc_score
         """
         y_true : array, shape = [n_samples] or [n_samples, n_classes]
         True binary labels or binary label indicators.

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

         """

         train_auc3 = []
         cv_auc3 = []
         K3 = [1, 5, 15, 31, 51, 75, 101, 125, 150, 175, 200]
         for i in K3:
             neigh = KNeighborsClassifier(algorithm='brute', n_neighbors=i)
             neigh.fit(X_tr3, y_train)

             y_train_pred = batch_predict(neigh, X_tr3)
             y_cv_pred = batch_predict(neigh, X_cr3)

             # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive c
             # not the predicted outputs
             train_auc3.append(roc_auc_score(y_train,y_train_pred))
             cv_auc3.append(roc_auc_score(y_cv, y_cv_pred))

         plt.plot(K3, train_auc3, label='Train AUC')
         plt.plot(K3, cv_auc3, label='CV AUC')

         plt.scatter(K3, train_auc3, label='Train AUC points')
         plt.scatter(K3, cv_auc3, label='CV AUC points')

         plt.legend()
         plt.xlabel("K: hyperparameter")
         plt.ylabel("AUC")
         plt.title("ERROR PLOTS")
         plt.grid()
         plt.show()
         print(train_auc3)          #k_best = 200
         print(cv_auc3)
```



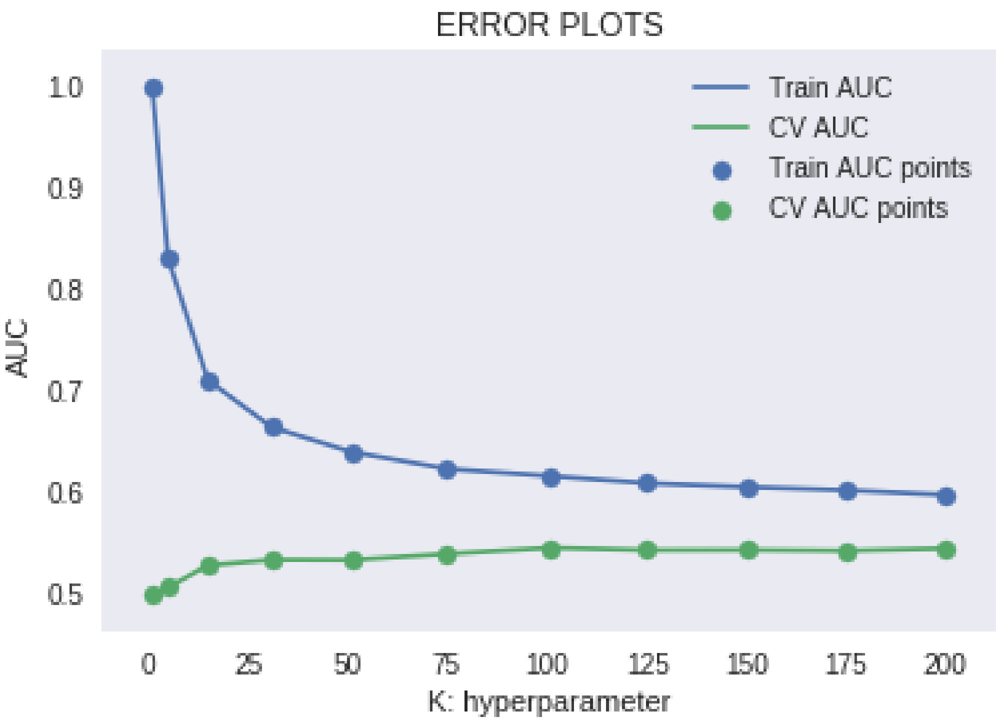
```
[0.9998556165174705, 0.8554670377423077, 0.7577025720340619, 0.720194273140446, 0.7016263230119706, 0.68711
[0.5168638951181646, 0.5353022723352203, 0.5453037778161118, 0.5642919064719997, 0.57323800711967, 0.578659
```

2.1.4 SET4

```
In [0]: def batch_predict(clf, data):
            # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive cl
            # not the predicted outputs

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

            return y_data_pred
```

```
In [90]: import matplotlib.pyplot as plt
         from sklearn.neighbors import KNeighborsClassifier
         from sklearn.metrics import roc_auc_score
```

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

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

"""

train_auc4 = []
cv_auc4 = []
K4 = [1, 5, 15, 31, 51, 75, 101, 125, 150, 175, 200]
for i in K4:
    neigh = KNeighborsClassifier(algorithm='brute', n_neighbors=i)
    neigh.fit(X_tr4, y_train)

    y_train_pred = batch_predict(neigh, X_tr4)
    y_cv_pred = batch_predict(neigh, X_cr4)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive c
    # not the predicted outputs
    train_auc4.append(roc_auc_score(y_train,y_train_pred))
    cv_auc4.append(roc_auc_score(y_cv, y_cv_pred))



plt.plot(K4, train_auc4, label='Train AUC')
plt.plot(K4, cv_auc4, label='CV AUC')

plt.scatter(K4, train_auc4, label='Train AUC points')
plt.scatter(K4, cv_auc4, label='CV AUC points')

plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
print(train_auc4)                    #k_best = 175
print(cv_auc4)
```



```
[0.9998556165174705, 0.8291961656882838, 0.709051902690127, 0.662247983430843, 0.6378715075543245, 0.621067
[0.4959375539071934, 0.50385434472376, 0.5257013815923597, 0.53128370473756, 0.5310602662819327, 0.53712860
```

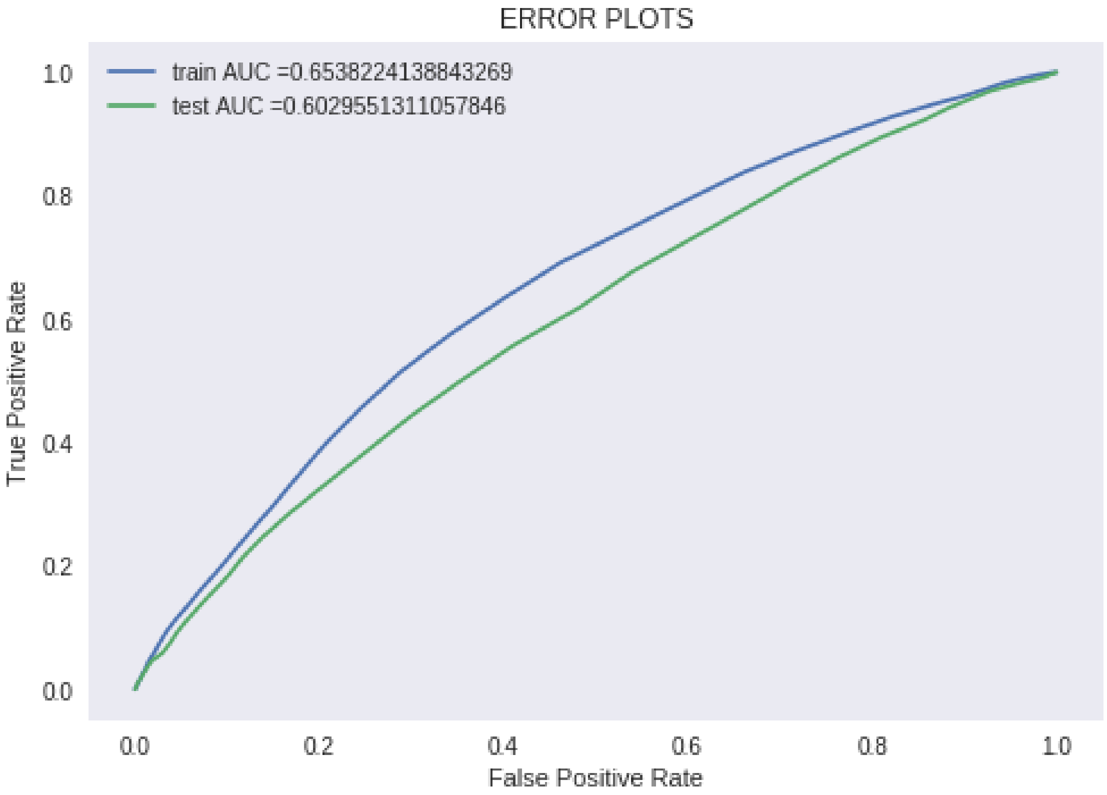
In [0]: csdcscssdds

2.1.5 Plot of AUC on Test and Train

SET1

In [104]:
```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc

best_k1 = 150
from sklearn.metrics import roc_curve, auc


neigh = KNeighborsClassifier(n_neighbors=best_k1)
neigh.fit(X_tr1, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive clas
# not the predicted outputs

y_train_pred1 = batch_predict(neigh, X_tr1)
y_test_pred1 = batch_predict(neigh, X_te1)
```

```
train_fpr1, train_tpr1, tr_thresholds1 = roc_curve(y_train, y_train_pred1)
test_fpr1, test_tpr1, te_thresholds1 = roc_curve(y_test, y_test_pred1)

plt.plot(train_fpr1, train_tpr1, label="train AUC ="+str(auc(train_fpr1, train_tpr1)))
plt.plot(test_fpr1, test_tpr1, label="test AUC ="+str(auc(test_fpr1, test_tpr1)))
plt.legend()
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



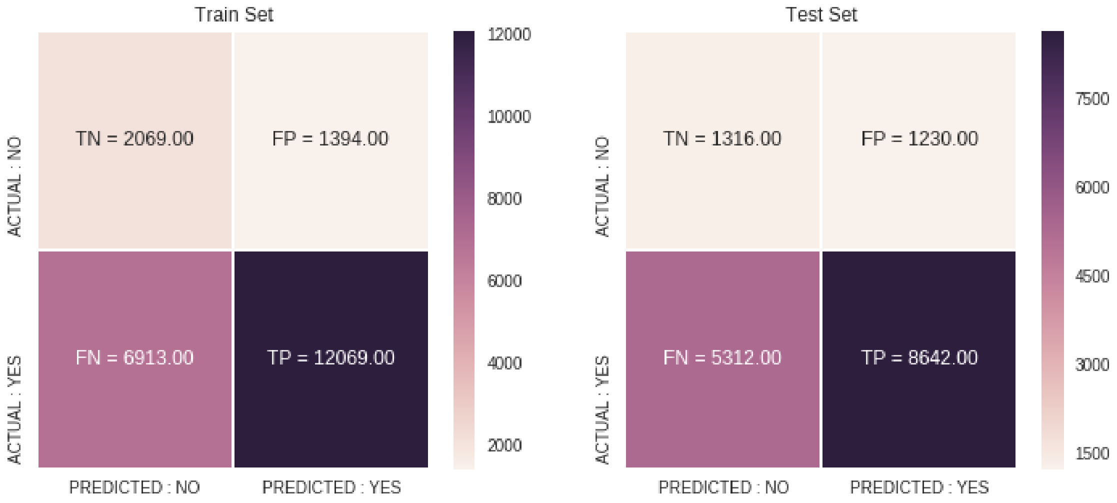
Confusion Matrix

```
In [0]: def predict(proba, threshould, fpr, tpr):

            t = threshould[np.argmax(fpr*(1-tpr))]

            # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high

            print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
            predictions = []
```

```
    for i in proba:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

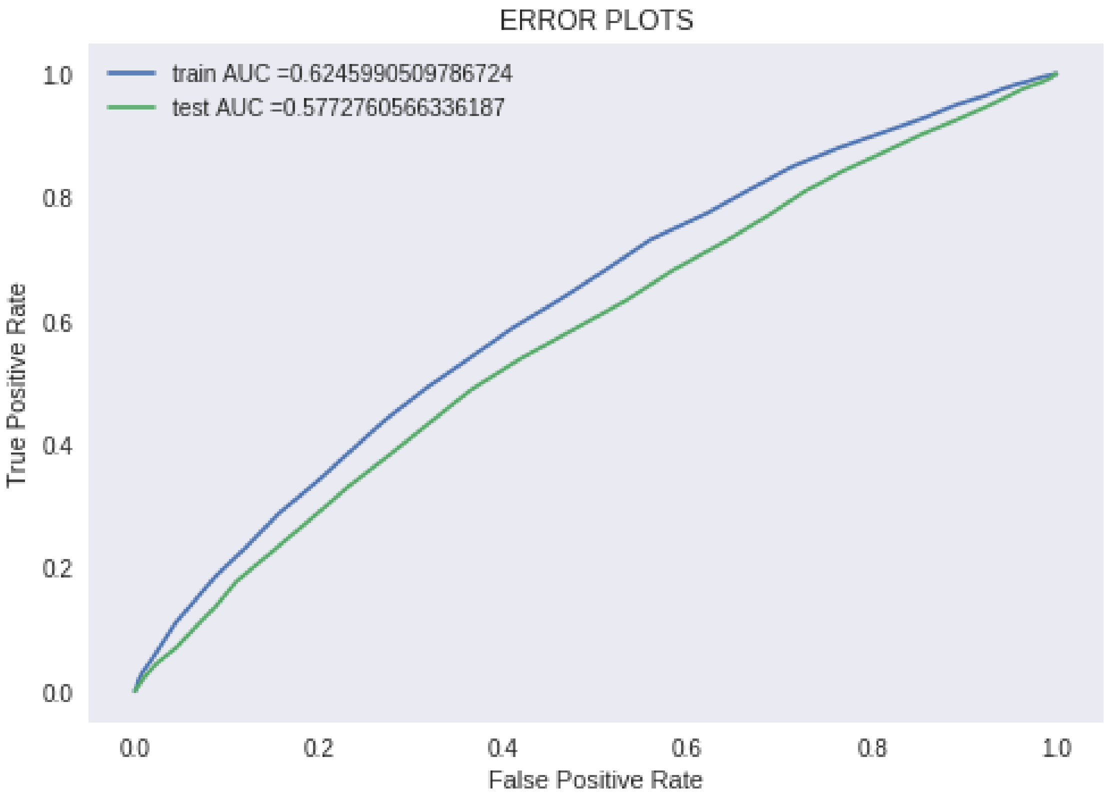
```
In [106]: #https://www.quantinsti.com/blog/creating-heatmap-using-python-seaborn
          import seaborn as sns; sns.set()

          con_m_train = confusion_matrix(y_train, predict(y_train_pred1, tr_thresholds1, train_fpr1, train_tp
          con_m_test = confusion_matrix(y_test, predict(y_test_pred1, te_thresholds1, test_fpr1, test_tpr1))

          key = (np.asarray([['TN','FP'], ['FN', 'TP']]))

          fig, ax = plt.subplots(1,2, figsize=(12,5))

          labels_train = (np.asarray(["{0} = {1:.2f}" .format(key, value) for key, value in zip(key.flatten(), con_m_
          labels_test = (np.asarray(["{0} = {1:.2f}" .format(key, value) for key, value in zip(key.flatten(), con_m_

          sns.heatmap(con_m_train, linewidths=.5, xticklabels=['PREDICTED : NO', 'PREDICTED : YES'], ytic
          sns.heatmap(con_m_test, linewidths=.5, xticklabels=['PREDICTED : NO', 'PREDICTED : YES'], ytick

          ax[0].set_title('Train Set')
          ax[1].set_title('Test Set')

          plt.show()
```

```
the maximum value of tpr*(1-fpr) 0.3798720297749744 for threshold 0.78
the maximum value of tpr*(1-fpr) 0.32852487147479636 for threshold 0.78
```



SET2

```
In [107]: # https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc

          best_k2 = 150
          from sklearn.metrics import roc_curve, auc


          neigh = KNeighborsClassifier(n_neighbors=best_k2)
          neigh.fit(X_tr2, y_train)
          # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive clas
          # not the predicted outputs

          y_train_pred2 = batch_predict(neigh, X_tr2)
          y_test_pred2 = batch_predict(neigh, X_te2)

          train_fpr2, train_tpr2, tr_thresholds2 = roc_curve(y_train, y_train_pred2)
          test_fpr2, test_tpr2, te_thresholds2 = roc_curve(y_test, y_test_pred2)

          plt.plot(train_fpr2, train_tpr2, label="train AUC ="+str(auc(train_fpr2, train_tpr2)))
          plt.plot(test_fpr2, test_tpr2, label="test AUC ="+str(auc(test_fpr2, test_tpr2)))
          plt.legend()
          plt.xlabel("False Positive Rate")
          plt.ylabel("True Positive Rate")
          plt.title("ERROR PLOTS")
          plt.grid()
          plt.show()
```



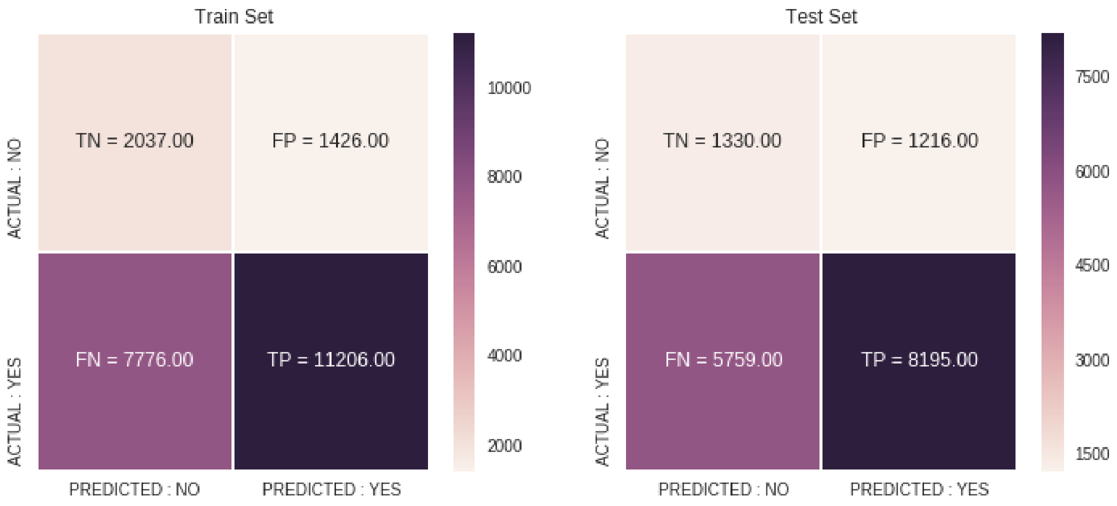
Confusion Matrix

```
In [0]: def predict(proba, threshould, fpr, tpr):

            t = threshould[np.argmax(fpr*(1-tpr))]

            # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high

            print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
            predictions = []
            for i in proba:
                if i>=t:
                    predictions.append(1)
                else:
                    predictions.append(0)
            return predictions
```

```
In [109]: #https://www.quantinsti.com/blog/creating-heatmap-using-python-seaborn
          import seaborn as sns; sns.set()

          con_m_train = confusion_matrix(y_train, predict(y_train_pred2, tr_thresholds2, train_fpr2, train_tp
          con_m_test = confusion_matrix(y_test, predict(y_test_pred2, te_thresholds2, test_fpr2, test_tpr2))
```

```
    key = (np.asarray([['TN','FP'], ['FN', 'TP']]))

    fig, ax = plt.subplots(1,2, figsize=(12,5))

    labels_train = (np.asarray(["{0} = {1:.2f}" .format(key, value) for key, value in zip(key.flatten(), con_m
    labels_test = (np.asarray(["{0} = {1:.2f}" .format(key, value) for key, value in zip(key.flatten(), con_m_

    sns.heatmap(con_m_train, linewidths=.5, xticklabels=['PREDICTED : NO', 'PREDICTED : YES'], ytic
    sns.heatmap(con_m_test, linewidths=.5, xticklabels=['PREDICTED : NO', 'PREDICTED : YES'], ytick

    ax[0].set_title('Train Set')
    ax[1].set_title('Test Set')

    plt.show()
```

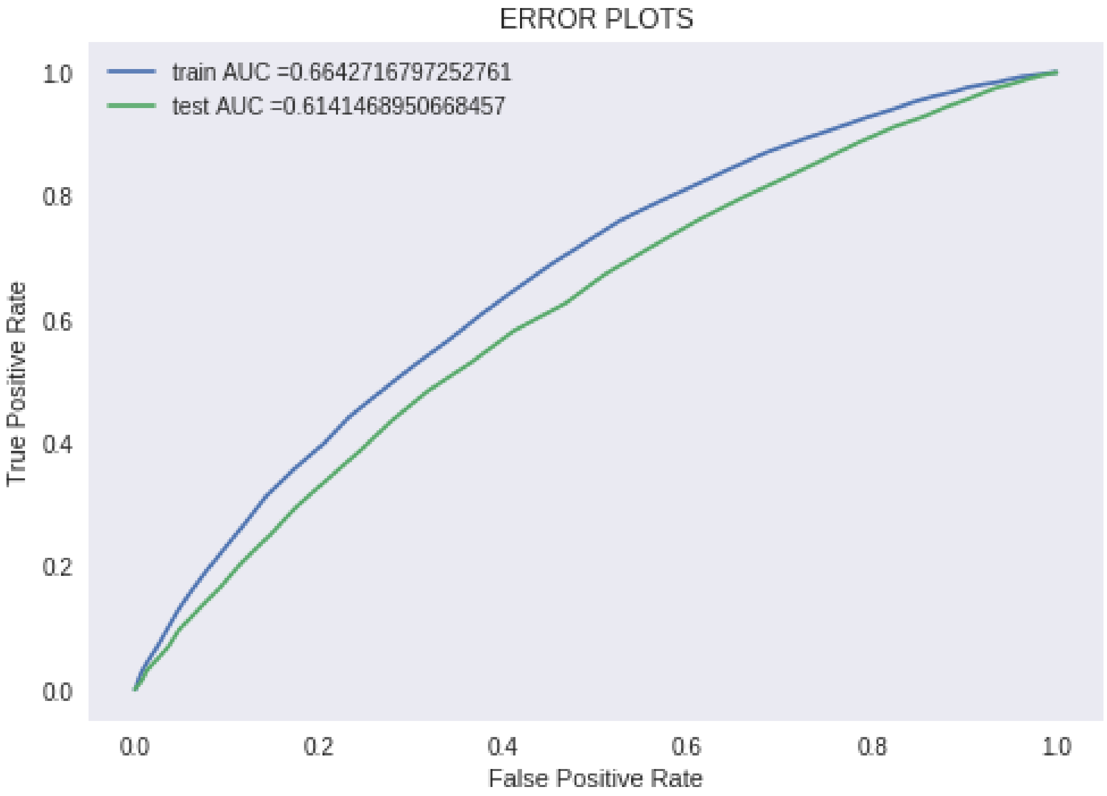
```
the maximum value of tpr*(1-fpr) 0.34725394360412504 for threshold 0.847
the maximum value of tpr*(1-fpr) 0.3133649717211338 for threshold 0.847
```



SET3

```
In [111]: # https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc

    best_k3 = 200
    from sklearn.metrics import roc_curve, auc


    neigh = KNeighborsClassifier(n_neighbors=best_k3)
    neigh.fit(X_tr3, y_train)
```

```
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive clas
# not the predicted outputs

y_train_pred3 = batch_predict(neigh, X_tr3)
y_test_pred3 = batch_predict(neigh, X_te3)

train_fpr3, train_tpr3, tr_thresholds3 = roc_curve(y_train, y_train_pred3)
test_fpr3, test_tpr3, te_thresholds3 = roc_curve(y_test, y_test_pred3)

plt.plot(train_fpr3, train_tpr3, label="train AUC ="+str(auc(train_fpr3, train_tpr3)))
plt.plot(test_fpr3, test_tpr3, label="test AUC ="+str(auc(test_fpr3, test_tpr3)))
plt.legend()
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



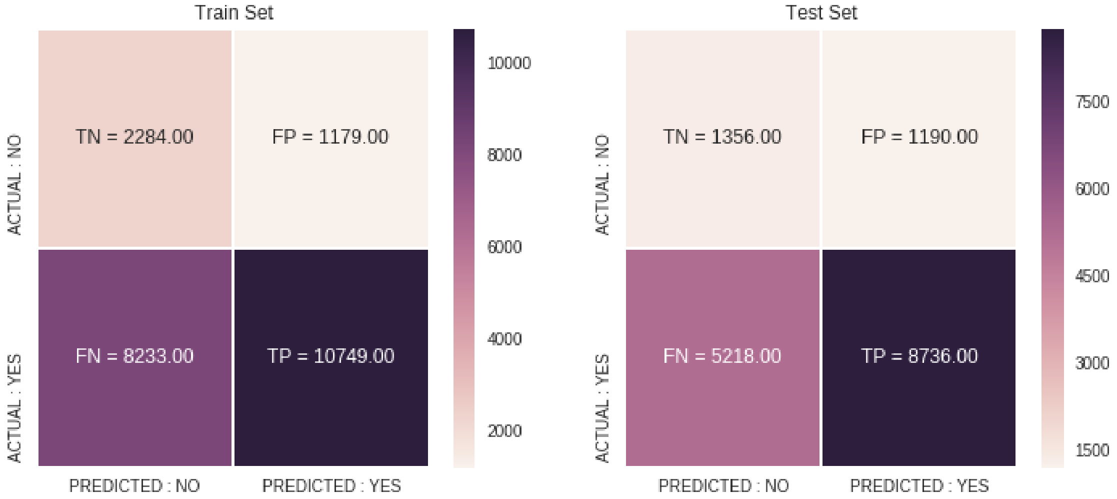
Confusion Matrix

```
In [0]: def predict(proba, threshould, fpr, tpr):
    t = threshould[np.argmax(fpr*(1-tpr))]

    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high

    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
    predictions = []
    for i in proba:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

In [113]:

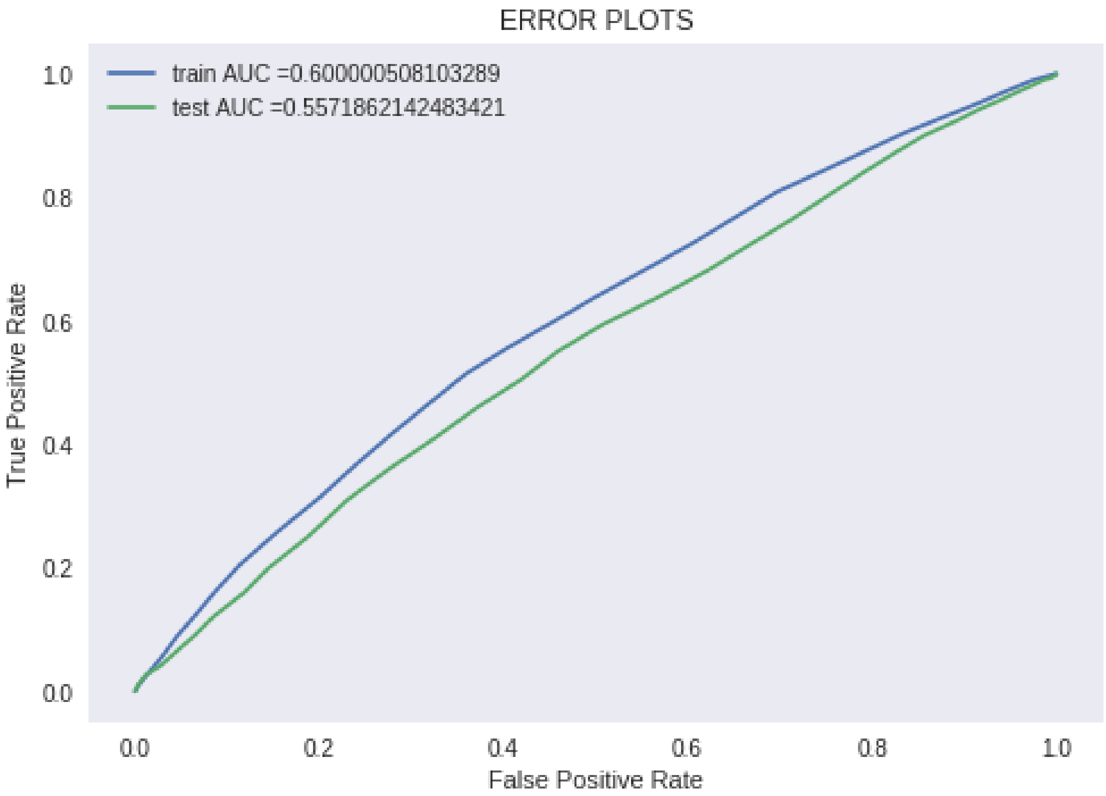
```
#https://www.quantinsti.com/blog/creating-heatmap-using-python-seaborn
import seaborn as sns; sns.set()

con_m_train = confusion_matrix(y_train, predict(y_train_pred3, tr_thresholds3, train_fpr3, train_tp
con_m_test = confusion_matrix(y_test, predict(y_test_pred3, te_thresholds3, test_fpr3, test_tpr3))

key = (np.asarray([['TN','FP'], ['FN', 'TP']]))

fig, ax = plt.subplots(1,2, figsize=(12,5))

labels_train = (np.asarray(["{0} = {1:.2f}" .format(key, value) for key, value in zip(key.flatten(), con_m_
labels_test = (np.asarray(["{0} = {1:.2f}" .format(key, value) for key, value in zip(key.flatten(), con_m_

sns.heatmap(con_m_train, linewidths=.5, xticklabels=['PREDICTED : NO', 'PREDICTED : YES'], ytic
sns.heatmap(con_m_test, linewidths=.5, xticklabels=['PREDICTED : NO', 'PREDICTED : YES'], ytick

ax[0].set_title('Train Set')
ax[1].set_title('Test Set')

plt.show()
```

```
the maximum value of tpr*(1-fpr) 0.3806715166089077 for threshold 0.865
the maximum value of tpr*(1-fpr) 0.3423572413499591 for threshold 0.865
```



SET4

```
In [115]: # https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc

          best_k4 = 175
          from sklearn.metrics import roc_curve, auc


          neigh = KNeighborsClassifier(n_neighbors=best_k4)
          neigh.fit(X_tr4, y_train)
          # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive clas
          # not the predicted outputs

          y_train_pred4 = batch_predict(neigh, X_tr4)
          y_test_pred4 = batch_predict(neigh, X_te4)

          train_fpr4, train_tpr4, tr_thresholds4 = roc_curve(y_train, y_train_pred4)
          test_fpr4, test_tpr4, te_thresholds4 = roc_curve(y_test, y_test_pred4)

          plt.plot(train_fpr4, train_tpr4, label="train AUC ="+str(auc(train_fpr4, train_tpr4)))
          plt.plot(test_fpr4, test_tpr4, label="test AUC ="+str(auc(test_fpr4, test_tpr4)))
          plt.legend()
          plt.xlabel("False Positive Rate")
          plt.ylabel("True Positive Rate")
          plt.title("ERROR PLOTS")
          plt.grid()
          plt.show()
```

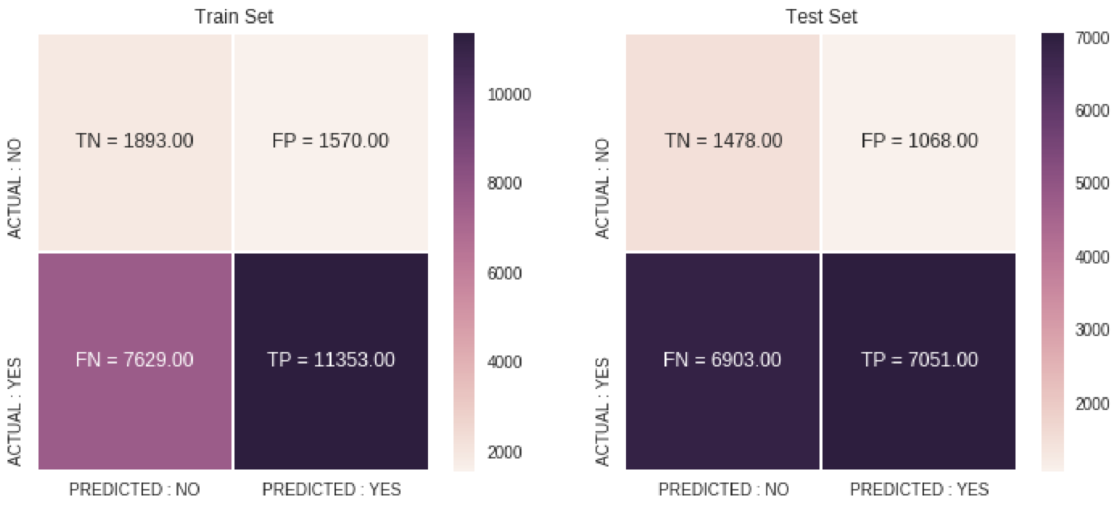


Confusion Matrix

```
In [0]: def predict(proba, threshould, fpr, tpr):

            t = threshould[np.argmax(fpr*(1-tpr))]

            # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high

            print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
            predictions = []
            for i in proba:
                if i>=t:
                    predictions.append(1)
                else:
                    predictions.append(0)
            return predictions
```

```
In [117]: #https://www.quantinsti.com/blog/creating-heatmap-using-python-seaborn
          import seaborn as sns; sns.set()

          con_m_train = confusion_matrix(y_train, predict(y_train_pred4, tr_thresholds4, train_fpr4, train_tp
          con_m_test = confusion_matrix(y_test, predict(y_test_pred4, te_thresholds4, test_fpr4, test_tpr4))
          key = (np.asarray([['TN','FP'], ['FN', 'TP']]))

          fig, ax = plt.subplots(1,2, figsize=(12,5))

          labels_train = (np.asarray(["{0} = {1:.2f}" .format(key, value) for key, value in zip(key.flatten(), con_m
          labels_test = (np.asarray(["{0} = {1:.2f}" .format(key, value) for key, value in zip(key.flatten(), con_m_

          sns.heatmap(con_m_train, linewidths=.5, xticklabels=['PREDICTED : NO', 'PREDICTED : YES'], ytic
          sns.heatmap(con_m_test, linewidths=.5, xticklabels=['PREDICTED : NO', 'PREDICTED : YES'], ytick

          ax[0].set_title('Train Set')
          ax[1].set_title('Test Set')

          plt.show()
```

```
the maximum value of tpr*(1-fpr) 0.3313022538214464 for threshold 0.846
the maximum value of tpr*(1-fpr) 0.2979433546719155 for threshold 0.857
```



2.2 K-BEST

```
In [123]: #from sklearn.datasets import load_digits
          from sklearn.feature_selection import SelectKBest, chi2
          #X, y = load_digits(return_X_y=True)

          X_new_tr = SelectKBest(chi2, k=2000).fit_transform(X_tr2, y_train)
          X_new_cv = SelectKBest(chi2, k=2000).fit_transform(X_cr2, y_cv)
          X_new_te = SelectKBest(chi2, k=2000).fit_transform(X_te2, y_test)
```

```
print(X_new_tr.shape)
print(X_new_cv.shape)
print(X_new_te.shape)

#X_new_cv.shape
#X_new_te.shape
```

```
(22445, 2000)
(11055, 2000)
(16500, 2000)
```

In [124]:

```
X_new_tr
```

Out[124]:

```
<22445x2000 sparse matrix of type '<class 'numpy.float64'>'
        with 709198 stored elements in Compressed Sparse Row format>
```

2.2.1 AUC Curve on K_BEST_2000

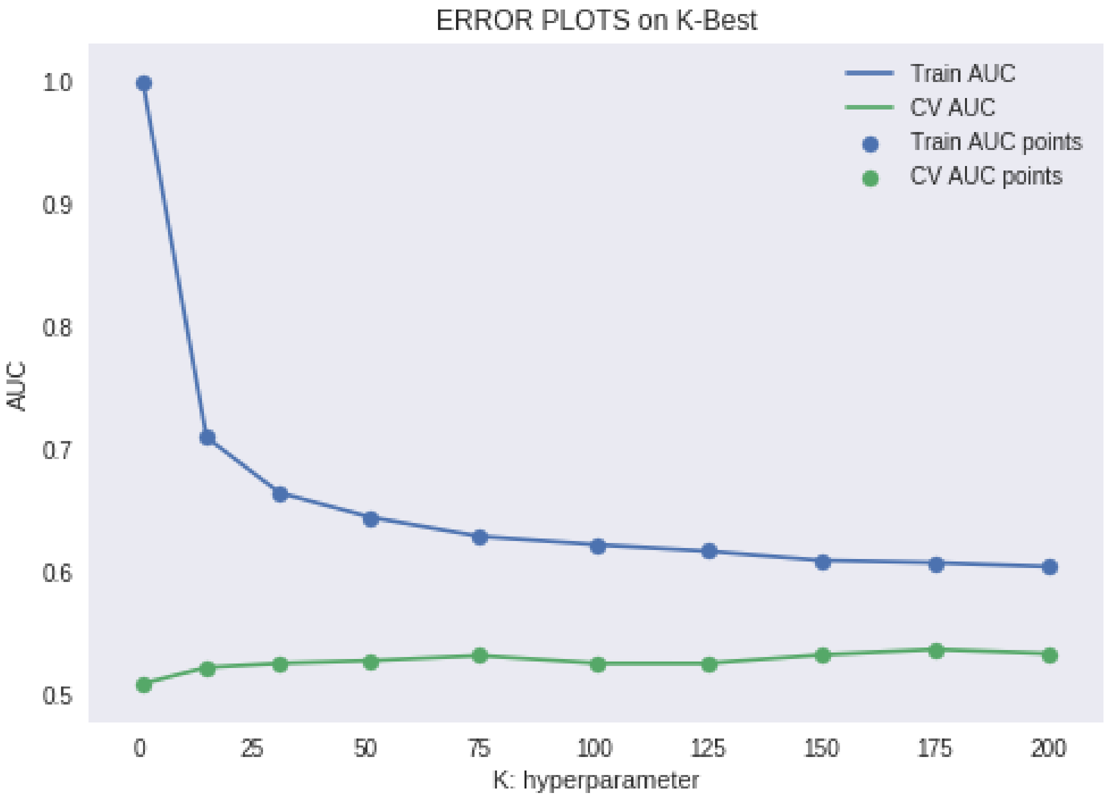
In [0]:

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

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

    return y_data_pred
```

In [126]:

```
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
"""
y_true : array, shape = [n_samples] or [n_samples, n_classes]
True binary labels or binary label indicators.

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

"""
```

```
train_auc_new = []
cv_auc_new = []
#K1 = [1, 5, 10, 15, 21, 31, 41, 51, 75, 101, 121, 151, 171]
K_new = [1, 15, 31, 51, 75, 101, 125, 150, 175, 200]

for i in K_new:
    neigh = KNeighborsClassifier(algorithm='brute', n_neighbors=i)
    neigh.fit(X_new_tr, y_train)

    y_train_pred_new = batch_predict(neigh, X_new_tr)
    y_cv_pred_new = batch_predict(neigh, X_new_cv)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
    # not the predicted outputs
    train_auc_new.append(roc_auc_score(y_train,y_train_pred_new))
    cv_auc_new.append(roc_auc_score(y_cv, y_cv_pred_new))


plt.plot(K_new, train_auc_new, label='Train AUC')
plt.plot(K_new, cv_auc_new, label='CV AUC')

plt.scatter(K_new, train_auc_new, label='Train AUC points')
plt.scatter(K_new, cv_auc_new, label='CV AUC points')

plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS on K-Best")
plt.grid()
plt.show()


print(train_auc_new)
print(cv_auc_new)
```



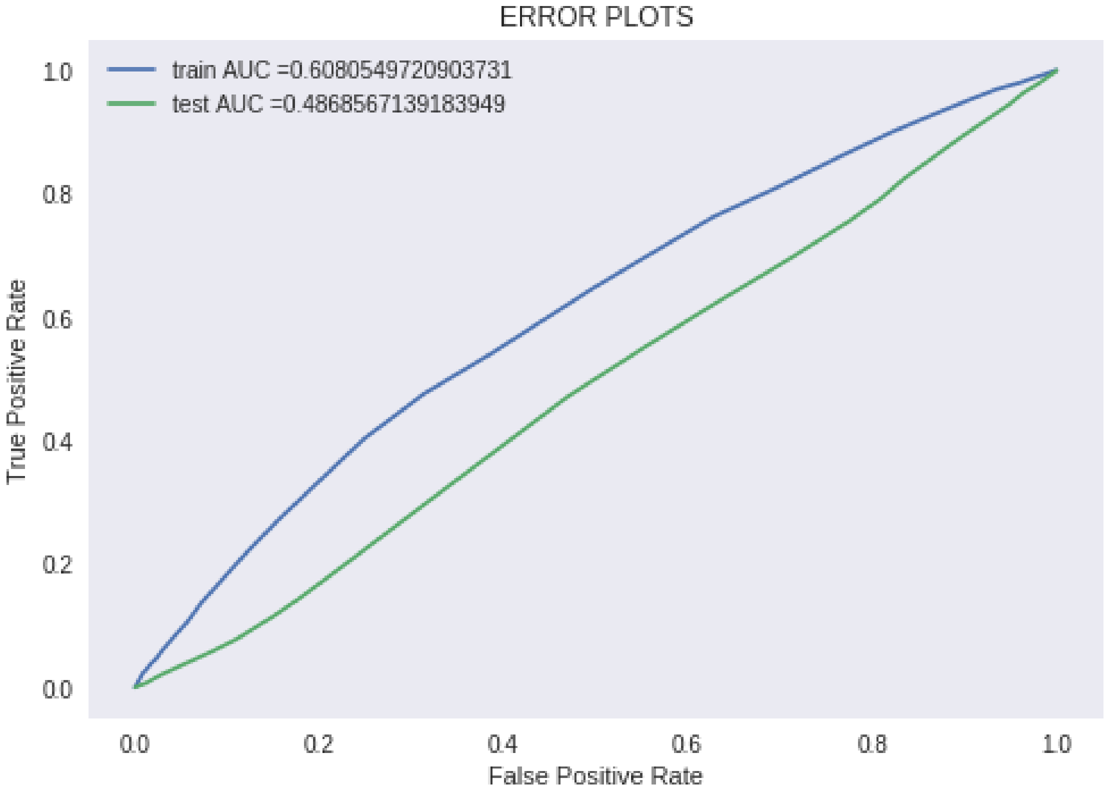
```
[0.9998556165174705, 0.7101442867299272, 0.6650296055356849, 0.6451055824943266, 0.6294805164142767, 0.6226
[0.5096049680869416, 0.5226695939906222, 0.5260762776984961, 0.5282413160412125, 0.5324186177803567, 0.5260
```

```
In [0]: sarfascsadcsadas
```

2.2.2 ROC Curve

```
In [128]: # https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc

          best_k_new = 150
          from sklearn.metrics import roc_curve, auc


          neigh = KNeighborsClassifier(n_neighbors=best_k4)
          neigh.fit(X_new_tr, y_train)
          # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive clas
          # not the predicted outputs

          y_train_pred_new = batch_predict(neigh, X_new_tr)
          y_test_pred_new = batch_predict(neigh, X_new_te)
```

```
train_fprn, train_tprn, tr_thresholdsn = roc_curve(y_train, y_train_pred_new)
test_fprn, test_tprn, te_thresholdsn = roc_curve(y_test, y_test_pred_new)

plt.plot(train_fprn, train_tprn, label="train AUC ="+str(auc(train_fprn, train_tprn)))
plt.plot(test_fprn, test_tprn, label="test AUC ="+str(auc(test_fprn, test_tprn)))
plt.legend()
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



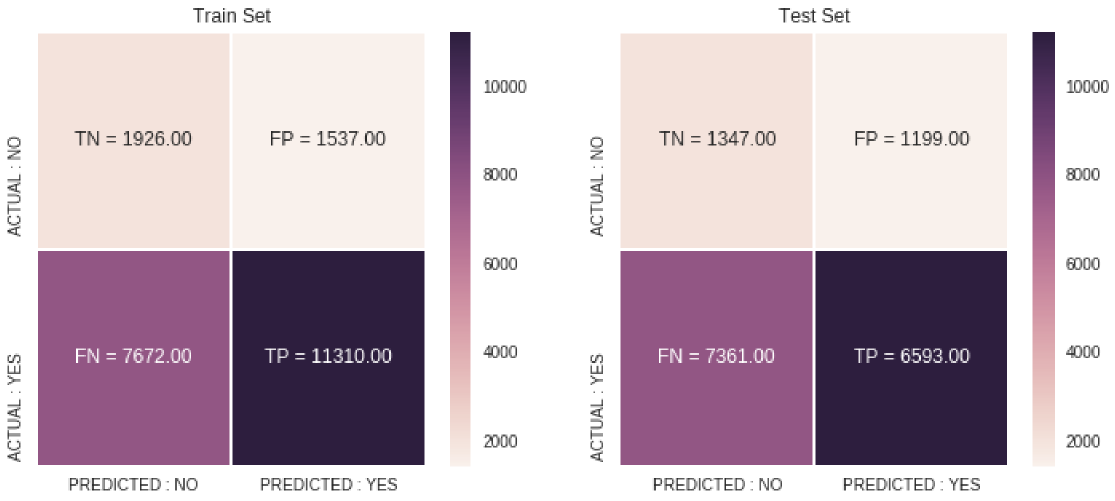
Confusion Matrix

```
In [0]: def predict(proba, threshould, fpr, tpr):

            t = threshould[np.argmax(fpr*(1-tpr))]

            # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high

            print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
            predictions = []
```

```
    for i in proba:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

```
In [132]: #https://www.quantinsti.com/blog/creating-heatmap-using-python-seaborn
          import seaborn as sns; sns.set()

          con_m_train = confusion_matrix(y_train, predict(y_train_pred_new, tr_thresholdsn, train_fprn, trai
          con_m_test = confusion_matrix(y_test, predict(y_test_pred_new, te_thresholdsn, test_fprn, test_tpr

          key = (np.asarray([['TN','FP'], ['FN', 'TP']]))

          fig, ax = plt.subplots(1,2, figsize=(12,5))

          labels_train = (np.asarray(["{0} = {1:.2f}" .format(key, value) for key, value in zip(key.flatten(), con_m_
          labels_test = (np.asarray(["{0} = {1:.2f}" .format(key, value) for key, value in zip(key.flatten(), con_m_

          sns.heatmap(con_m, linewidths=.5, xticklabels=['PREDICTED : NO', 'PREDICTED : YES'], yticklabels
          sns.heatmap(con_m, linewidths=.5, xticklabels=['PREDICTED : NO', 'PREDICTED : YES'], yticklabels

          ax[0].set_title('Train Set')
          ax[1].set_title('Test Set')

          plt.show()
```

```
the maximum value of tpr*(1-fpr) 0.3317921019025182 for threshold 0.829
the maximum value of tpr*(1-fpr) 0.2499732596869458 for threshold 0.817
```



3. Conclusions

In [133]:
```
# Please compare all your models using Prettytable library

# Please compare all your models using Prettytable library

from prettytable import PrettyTable

x = PrettyTable()
x.field_names = ["Vectorizer", "Model", "HyperParameter", "AUC"]

x.add_row(["BOW",        "KNN_Brute", 150, 0.602])
x.add_row(["TFIDF",      "KNN_Brute", 150, 0.577])
x.add_row(["Avg W2V",    "KNN_Brute", 200, 0.614])
x.add_row(["TFIDF-W2v",  "KNN_Brute", 175, 0.557])


print(x)
```

```
+------------+-----------+----------------+-------+
| Vectorizer |   Model   | HyperParameter |  AUC  |
+------------+-----------+----------------+-------+
|    BOW     | KNN_Brute |      150       | 0.602 |
|   TFIDF    | KNN_Brute |      150       | 0.577 |
|  Avg W2V   | KNN_Brute |      200       | 0.614 |
| TFIDF-W2v  | KNN_Brute |      175       | 0.557 |
+------------+-----------+----------------+-------+
```