Metal Ion Analysis

Introduction

The main motivation of this project is to do the multiple regression analysis using the Water Dataset provided by University of Turku. The sample contains six column features. Three of these are measured from independent devices and they are independent data features. We have to predict the other dependent features i.e. Total, Cadmium and Lead content.

Data set contains 201 data samples. Here is an example :

```
c_total,Cd,Pb,Mod1,Mod2,Mod3
0,0,0,9945,119,72335
0,0,0,10786,117,82977
0,0,0,10812,120,98594
14,0,14,9742,127,154323
14,0,14,10566,108,136416
14,0,14,8495,120,131672
14,2.8,11.2,10400,134,96528
14,2.8,11.2,8298,113,99239
14,2.8,11.2,8563,130,113979
14,5.6,8.4,9879,130,87882
14,5.6,8.4,10412,101,95515
14,5.6,8.4,12605,130,125010
14,8.4,5.6,19491,133,186314
14,8.4,5.6,18774,118,120807
14,8.4,5.6,26959,143,157330
14,11.2,2.8,23158,123,145455
14,11.2,2.8,16340,129,112713
14,11.2,2.8,16957,138,153859
```

Our data sample is not shuffled as three data held together are so related.

Implementation

Leave-Three-CrossValidation is used to separate training and testing samples. C-Index is calculated for various number of K i.e. neighbours. The best C-Index among these C-Index will be used for regression in future samples.

The entire code is as follows:

```python
import pandas as pd

import math
import operator
import sys
import time
from scipy.stats import zscore
import matplotlib.pyplot as plt
datapath = "Water_data.csv"
# 1 .Loading Data, Shuffling the indices and resetting the indices
def load_data(datapath,shuffle=False):
    pd_data = pd.read_csv(datapath)
    if shuffle==False:
        return pd_data
    else:
        return pd_data.sample(frac=1).reset_index(drop=True)
# 2. Leave-one-out Cross-validation method
def leave_N_CV(data,N=1):
    totallength = len(data)
    for i in range(totallength):
        if N > 1 :
            final_indices = i + N-1
            if final_indices < totallength:
                test_list_indices = list(range(i,final_indices+1))# First N rows
                test_data = data.ix[i:final_indices,:].reset_index()
                train_data = data.drop(test_list_indices,axis=0).reset_index()
                yield train_data, test_data
def return_two_columns(data,column1,column2):
    leng = len(data.columns.values)
    data[leng] = data[column1]
    data[leng+1] = data[column2]
    return data.ix[:,leng:]
# 2. This definition returns euclidean distance between two instances
# This should be completely integer based
def euclidean_distance(data1, data2):
     leng = len(data1)-1 # This avoids including the class of the dataset, i.e. only have
four attributes
    distance = 0.0
    data = data1 - data2
     data = data.ix[[3,4,5]] # This is done to only compare 3,4,5 column of the dataset, so
that it would not include the target i.e. 0,1 and 2
    for i in range(leng):
        distance += pow((data.iloc[i]),2)
    return math.sqrt(distance)
# 2. This definition returns manhattan_distance between two instances
# This should be completely integer based
def manhattan_distance(data1,data2):
    leng =len(data1)-1
    man = 0.0
    man_item = 0.0
    for i in range(leng):
        x = float(data1.iloc[i])
        y = float(data2.iloc[i])
        z = x-y
        man_item = math.fabs(z)
        man += man_item
    return man
#. Get Neighbour
def getNeighbour(k,data,mydata_instance,classProperties=[3,4,5]):
    distances = [] # An array object with key value pair ??
    for i in range(len(data)):
        dist = euclidean_distance(data.ix[i,classProperties],mydata_instance)
        distances.append((data.ix[i], dist))
    distances.sort(key=operator.itemgetter(1))
    neighbors = []
    for x in range(k):
        neighbors.append(distances[x][0])
        #neighbors.append(distances[x])
    return neighbors
def getregressionValue(neighbours, columnIndex = [0,1,2]):#columnIndex = 0 , 1,2, suggest
prediction in columnIndex = 0,1,2, i.e. Total, Lead , Cadmium Content
    predictions = []
    N = len(neighbours)
    for i in range(len(columnIndex)):
        predict_value = 0.0
        for x in range (N):
            predict_value += neighbours[x][i]
        predictions.append(predict_value/len(neighbours))
    return predictions
def cindex(true_labels, pred_labels):
    """Returns C-index between true labels and predicted labels."""
    count = true_labels.shape[0]
    n = 0.0
    h_sum = 0.0
    for i in range(count):
        t = true_labels[i]
        p = pred_labels[i]
        for j in range(i + 1, count):
            nt = true_labels[j]
            np = pred_labels[j]
            if (t != nt):
                n += 1
                if (p < np and t < nt) or (p > np and t > nt):
                    h_sum += 1.0
                elif (p == np):
                    h_sum += .5
    return h_sum / n
def main():
    print("*************************************************************")
    print("         Metal Ion Concentration Analysis                    ")
    print("*************************************************************")
    # This loads data from csv files
    loaddata = load_data(datapath=datapath)
    dataproperties = loaddata.columns.values
    # This data is normalized using defined scikit zscore library
    normalizedata = zscore(loaddata)
    # This returns normalizeddata to proceed
    mydata = pd.DataFrame(normalizedata)
    dataproperties = [3,4,5,0,1,2]
    mydata = mydata[dataproperties]
    # Perform 1-N Cross Validation
    columnlist = [3, 4, 5, 0,1,2]
    cross_validate = leave_N_CV(mydata,N=3)
    mys = [5,6,7,8,9]
    for numberofneighbour in range(2,10):
        cindex_forallpredictions = []
        total_predictions = 0
        for train_data, test_data in cross_validate:
            traindata = train_data.ix[:, columnlist]  # Four columns 3,4,5, and 0 , 1, 2
            testdata = test_data.ix[:, columnlist]  # Four Columns 3,4,5 and 0, 1, 2
            for i in range(len(testdata)):
                                                                            neighbours     =
getNeighbour(k=numberofneighbour,data=traindata,mydata_instance=testdata.ix[i])
                regression_value = getregressionValue(neighbours)
                real_value = testdata.ix[i,[0,1,2]]
                c_index = cindex(real_value,regression_value)
                cindex_forallpredictions.append(c_index)
                total_predictions += 1
                print("C_Index : ",c_index)
        average_prediction = sum(cindex_forallpredictions) / total_predictions
         print(" C Index Average for all predictions with K =", numberofneighbour ,"is :",
average_prediction )
if __name__=="__main__":
    main()
```

These codes are in my opinion well commented.

Result:

```
 C Index Average for all predictions with K = 2 is : 0.7839195979899509

 C Index Average for all predictions with K = 3 is : 0.76493579006142

 C Index Average for all predictions with K = 4 is : 0.7554438860971538

 C Index Average for all predictions with K = 5 is : 0.7604690117252948

 C Index Average for all predictions with K = 6 is : 0.7375767727526538

 C Index Average for all predictions with K = 7 is : 0.7364600781686225

 C Index Average for all predictions with K = 8 is : 0.713288665549973

 C Index Average for all predictions with K = 9 is : 0.6912339475153557

 C Index Average for all predictions with K = 10 is : 0.6850921273031838
```

The best seems to be K = 2, but it is quite optimistic. I would choose K = 5 for better results.