Tiina Nokelainen

k-NN regressor and leave-p-out cross-validation

In [1]:
```
import pandas as pd
import numpy as np
import math
import operator
from sklearn.utils import shuffle
from scipy.stats import zscore
from itertools import permutations


df_values = pd.read_csv("Water_data.csv", usecols=["c_total", "Cd", "Pb"])
df_mods = pd.read_csv("Water_data.csv", usecols=["Mod1", "Mod2", "Mod3"])

df_mods=df_mods.apply(zscore)

df = pd.concat([df_values, df_mods], axis=1, sort=False)

df_shuffled = shuffle(df).reset_index(drop=True)
```

I shuffled the data at first, but I didn't get same c-index values as I heard they should be. I thought that shuffling the data would be good since I noticed that the data is in order from smallest values to biggest. Then I tried without shuffle and then I got "right values"

In [2]:
```
# Euclidean distance between train set points and test point
def distance(trainX, predX):

    distance = 0

    # first 3 elements are the c_total, Cd and Pb, let's not calculate the dist
ances of them
    for n in range(3,len(trainX)):
        distance += math.pow((trainX[n] - predX[n]), 2)

    return math.sqrt(distance)
```

In [3]:
```
# returns k number of nearest neighbors
def getNeighbors(data, testInstance, k):

    distances = []

    for i in range(len(data)):
        # adds all the distances to the distances-variable
        distances.append((data[i], distance(data[i], testInstance)))

    # sorts the distances by the distance-value (which is the second column)
    distances.sort(key=operator.itemgetter(1))

    neighbors = []

    # stores k nearest neighbors to neighbors
    for i in range(k):
        neighbors.append(distances[i][0])

    return neighbors
```

In [4]:
```
# from the slides
# returns c-index
def cindex(predictions, truevalues):

    predictions = list(predictions)

    n = 0
    n_sum = 0
    for i in range(len(truevalues)):
        t = truevalues[i]
        p = predictions[i]
        for j in range(i+1,len(truevalues)):
            nt = truevalues[j]
            np = predictions[j]
            if (t != nt):
                n += 1
                if (p < np and t < nt) or (p > np and t > nt):
                        n_sum += 1
                elif p == np:
                    n_sum += 0.5
    return n_sum/n
```

I tested the cindex-method with the data given in the slides. I got correct answer 0.75

In [5]:
```
# divides the data to n-sized groups
def divide(data,n):

    for i in range(0, len(data), n):
        yield data[i:i + n]
```

In [6]:
```
# predicts c_total, Cd and Pb for testinstance
# prediction is the mean of neighbors' values
# returns list of predictions [c_total, Cd, Pb]
def predict(neighbors):

    neighbors = np.array(neighbors)
    preds = []

    for i in range(3):
        preds.append(np.mean(neighbors[:,i]))

    return preds
```

In [7]:
```python
# Leave-p-out cross-validation using knn regression
def lpo(data, p, k):

    # splitted has groups of p-sized lists
    # each group is one test set
    splitted = list(divide(data.values, p))

    # list to store every instances' predictions of c_total, Cd and Pb values
    preds = []


    for i in range(len(splitted)):
        # trainset without the test set of current iteration
        trainset = list(data.drop(data.index[i*p:(i*p+p)]).values)

        # predicts test set's instances' c_total, Cd and Pd
        for j in range(len(splitted[i])):
            neighbors = getNeighbors(trainset, splitted[i][j], k)
            # all the predictions of the instance splitted[i][j]
            # [c_total, Pb, Cd]
            predicts = predict(neighbors)
            # adds instances' predictions to data's predictions list preds
            preds.append(predicts)

    preds = np.array(preds)

    C_total_c = round(cindex(preds[:,0], data.iloc[:,0]),2)
    Cd_c = round(cindex(preds[:,1], data.iloc[:,1]),2)
    Pb_c = round(cindex(preds[:,2], data.iloc[:,2]),2)

    print("k =", k, "C-index for c_total, Cd, Pb:", [C_total_c, Cd_c, Pb_c])
```

In [8]:
```python
print("Performing leave-1-out cross-validation:\n")

for i in range(1,6):
    lpo(df,1,i)

print("\nPerforming leave-3-out cross-validation:\n")

for i in range(1,6):
    lpo(df,3,i)
```

```
Performing leave-1-out cross-validation:

k = 1 C-index for c_total, Cd, Pb: [0.9, 0.9, 0.87]
k = 2 C-index for c_total, Cd, Pb: [0.91, 0.9, 0.87]
k = 3 C-index for c_total, Cd, Pb: [0.9, 0.88, 0.85]
k = 4 C-index for c_total, Cd, Pb: [0.89, 0.85, 0.85]
k = 5 C-index for c_total, Cd, Pb: [0.88, 0.83, 0.83]

Performing leave-3-out cross-validation:

k = 1 C-index for c_total, Cd, Pb: [0.82, 0.74, 0.74]
k = 2 C-index for c_total, Cd, Pb: [0.82, 0.75, 0.75]
k = 3 C-index for c_total, Cd, Pb: [0.82, 0.74, 0.75]
k = 4 C-index for c_total, Cd, Pb: [0.82, 0.72, 0.76]
k = 5 C-index for c_total, Cd, Pb: [0.82, 0.72, 0.76]
```

Values of C-index with leave-3-out are significally lower than with leave-1-out. This is due to the fact that the data is sorted. Every test set has instances that are very close to each other so the predictions are not good.

In [9]:
```python
print("Performing leave-1-out cross-validation:\n")

for i in range(1,6):
    lpo(df_shuffled,1,i)

print("\nPerforming leave-3-out cross-validation:\n")

for i in range(1,6):
    lpo(df_shuffled,3,i)
```

```
Performing leave-1-out cross-validation:

k = 1 C-index for c_total, Cd, Pb: [0.9, 0.9, 0.87]
k = 2 C-index for c_total, Cd, Pb: [0.91, 0.9, 0.87]
k = 3 C-index for c_total, Cd, Pb: [0.9, 0.88, 0.85]
k = 4 C-index for c_total, Cd, Pb: [0.89, 0.85, 0.85]
k = 5 C-index for c_total, Cd, Pb: [0.88, 0.83, 0.83]

Performing leave-3-out cross-validation:

k = 1 C-index for c_total, Cd, Pb: [0.9, 0.9, 0.87]
k = 2 C-index for c_total, Cd, Pb: [0.91, 0.9, 0.87]
k = 3 C-index for c_total, Cd, Pb: [0.9, 0.88, 0.85]
k = 4 C-index for c_total, Cd, Pb: [0.89, 0.85, 0.84]
k = 5 C-index for c_total, Cd, Pb: [0.88, 0.82, 0.83]
```

Here is the example of the c-index values of shuffled data. The leave-3-out cross-validation gets better results with shuffled data than unshuffled. The difference between leave-one-out and leave-three-out is not significant.

I googled that leave-p-out cross-validation should be done with all combinations of p-sized groups. I tried to implement that, but I struggled with it a lot. However, the leave-3-out from the slides didn't sound like that it should be done with all combinations of p-sized (3-sized) groups.