# TKO_7092 Evaluation of Machine Learning Methods 2024

---

Student name: Ruslan Hasanov

Student number: 2310614

Student email: ruhasa@utu.fi

---

## Exercise 4

Complete the tasks given to you in the letter below. In your submission, explain clearly, precisely, and comprehensively why the cross-validation described in the letter failed, how cross-validation should be performed in the given scenario and why your cross-validation will give a reliable estimate of the generalisation performance. Then implement the correct cross-validation for the scenario and report its results.

Remember to follow all the general exercise guidelines that are stated in Moodle. Full points (2p) will be given for a submission that demonstrates a deep understanding of cross-validation on pair-input data and implements the requested cross-validation correctly (incl. reporting the results). Partial points (1p) will be given if there are small error(s) but the overall approach is correct. No points will be given if there are significant error(s).

The deadline of this exercise is **Wednesday 21 February 2024 at 11:59 PM**. Please contact Juho Heimonen (juaheim@utu.fi) if you have any questions about this exercise.

---

Dear Data Scientist,

I have a long-term research project regarding a specific set of proteins. Currently I am attempting to discover small organic compounds that can bind strongly to these proteins and thus act as drugs. I have a list of over 100.000 potential drug molecules, but their affinities still need to be verified in the lab. Obviously I do not have the resources to measure all the possible drug-target pairs, so I need to prioritise. I have decided to do this with the use of machine learning, but I have encountered a problem.

Here is what I have done so far: First I trained a K-nearest neighbours regressor with the parameter value K=10 using all the 400 measurements I had made in the lab, which comprise of all the 77 target proteins of interest but only 59 different drug molecules. Then I performed a leave-one-out cross-validation with this same data to estimate the generalisation performance of the model. I used C-index and got a stellar score above 90%. Finally I used the model to predict the affinities of the remaining drug molecules. The problem is: when I selected the highest predicted affinities and tried to verify them in the lab, I found that many of them are much lower in reality. My model clearly does not work despite the high cross-validation score.

Please explain why my estimation failed and how leave-one-out cross-validation should be performed to get a reliable estimate. Also, implement the correct leave-one-out cross-validation and report its results. I need to know whether I am wasting my lab resources by using my model.

The data I used to create my model is available in the files `input.data`, `output.data` and `pairs.data` for you to use. The first file contains the features of the pairs, whereas the second contains their affinities. The third file contains the identifiers of the drug and target molecules of which the pairs are composed. The files are paired, i.e. the i$^{*th*}$ row in each file is about the same pair.

Looking forward to hearing from you soon.

Yours sincerely, \ Bio Scientist

---

## Answer the questions about cross-validation on pair-input data

In [1]:
```
# Why did the estimation described in the letter fail?
# How should leave-one-out cross-validation be performed in the given scenario and why?
# Remember to provide comprehensive and precise arguments.
```

In the leave-one-out cross validation, there is one test observation and the rest is used as training in each fold. There is also an assumption that the test set is independent of the training set, which is not the case for your situation, since you are using pair-input data. The out-of-sample observations may share similar pair members in the sample. These dependencies within pair-input data can affect the estimation of out-of-sample prediction performance. Based on the number of pair members, the out-of-sample observations are categorized into four groups. These four types of out-of-sample observations differ by the nature and extent of dependencies. For example, the out-of-sample prediction of a hypothesis is expected to be the best on type A and the worst on type D observations, as type A observations share more pair members with sample observations. To achieve a succesful performance evaluation, performance evaluation method should be adapted to the nature of pair-input data. It is necessary to estimate the out-of-sample prediction performance for each type seperately.

In order to sucessfully implement the cross-validation, some changes should be made for training sets. Each type of out-of-sample observations has different rules. For the type A out-of-sample observation, there is no restriction on training dataset. For the type B, the first pair members of the test observations are not allowed to appear in the training set. For the type C, the second pair members of the test observations are not allowed to appear in the training set. For the type D, none of the pair members of the test observations are allowed to be in the training set.

Judging your dataset, it can be said that your situation only involves type A and type B out-of-sample observations, as you used all the target proteins. In your case the first pair members are the drug molecules and the second pair members are target proteins. The type B out-of-sample observations include all the pairings for any other drug molecules than the used 59 drug molecules, as your measurement includes all the 77 target proteins. By following the above mentioned rules, we can implement a succesful cros-validation.

Also never forget about standardization of the data.

## Import libraries

In [2]:
```
# Import the libraries you need.
import numpy as np
import pandas as pd
from sklearn.neighbors import KNeighborsRegressor
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import LeaveOneOut
```

### Write utility functions

```
In [3]:
# Write the utility functions you need in your analysis

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

### Load datasets

```
In [4]:
# Read the data files (input.data, output.data, pairs.data).
# Read the .data file without a header row

inputdata = pd.read_csv('C:\\Users\\Comp-servis\\Desktop\\Evaluation of Machine Learning
print('The dimensions of the input data are:', inputdata.shape)

output = pd.read_csv('C:\\Users\\Comp-servis\\Desktop\\Evaluation of Machine Learning Me
print('The dimensions of the output data are:', output.shape)

pairs = pd.read_csv('C:\\Users\\Comp-servis\\Desktop\\Evaluation of Machine Learning Met
print('The dimensions of the pairs data are:', pairs.shape)
```

```
The dimensions of the input data are: (400, 67)
The dimensions of the output data are: (400, 1)
The dimensions of the pairs data are: (400, 2)
```

### Implement and run cross-validation

```
In [5]:
# Cross-validation for A-type out-of-sample observations

# Lists to store predicted and true values
predicted_values = []
true_values = []

# Leave-one-out cross-validator
loo = LeaveOneOut()

# An instance of StandardScaler
scaler = StandardScaler()

# Performing the leave-one-out cross-validation for the A-type out-of-sample observation
for train_index, test_index in loo.split(inputdata):

    # Splitting data into training and test sets
    X_train, X_test = inputdata.iloc[train_index], inputdata.iloc[test_index]
    y_train, y_test = output.iloc[train_index], output.iloc[test_index]

    # Standardizing the features
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)

    # k-nearest neighbors regressor is trained
    knn_regressor = KNeighborsRegressor(n_neighbors=10)
    knn_regressor.fit(X_train_scaled, y_train)

    # Making predictions
    y_pred = knn_regressor.predict(X_test_scaled)

    # Storing predicted and true values for calculation of C-index value
    predicted_values.append(y_pred[0])  # Appending the predicted value
    true_values.append(y_test.values[0])   # Appending the true value

# Calculating and printing the c-index result
c_index = cindex(predicted_values, true_values)
print("The C-index value for the A-type out-of-sample observations:", c_index)

# Cross-validation for B-type out-of-sample observations

# Lists to store predicted and true values
predicted_values = []
true_values = []


# Performing the leave-one-out cross-validation for the B-type out-of-sample observation
for train_index, test_index in loo.split(inputdata):

    # Splitting data into training and test sets
    X_train, X_test = inputdata.iloc[train_index], inputdata.iloc[test_index]
    y_train, y_test = output.iloc[train_index], output.iloc[test_index]

    # The first member of the pair from the test data is extracted
    first_pair_member_test = pairs.iloc[test_index[0], 0]

    # A boolean mask to filter out rows in the training data where
    # the first member of the pair is not equal to the first member of the pair in the t
    mask = pairs.iloc[train_index, 0] != first_pair_member_test

    # The mask is applied to filter rows in the features (X_train) and labels (y_train)
    X_train = X_train[mask]
    y_train = y_train[mask]

    # Standardizing the features
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)

    # k-nearest neighbors regressor is trained
    knn_regressor = KNeighborsRegressor(n_neighbors=10)
    knn_regressor.fit(X_train_scaled, y_train)

    # Making predictions
    y_pred = knn_regressor.predict(X_test_scaled)

    # Storing predicted and true values for calculation of C-index value
    predicted_values.append(y_pred[0])  # Appending the predicted value
    true_values.append(y_test.values[0])   # Appending the true value

# Calculating and printing the c-index result
c_index = cindex(predicted_values, true_values)
print("The C-index value for the B-type out-of-sample observations:", c_index)
```

```
The C-index value for the A-type out-of-sample observations: 0.8293691571217324
The C-index value for the B-type out-of-sample observations: 0.5117952065887434
```

According to our results, it can be said that the machine learning model exhibits different performance values for the different types of out-of-sample observations. The c-index for the A type out-of-sample observations is 0.83 which can be considered as satisfactory, however the performance metric for the B-type out-of-sample observations is 0.51, which can be regarded as random. I would not recommend to use this machine learning model to make predictions on the affinities of the drug molecules other than the used 59 drug molecules, which categorizes into the B type out-of-sample observations.