exercise-3-robert-siipola

February 18, 2026

1 TKO_7092 Evaluation of Machine Learning Methods 2026

1.1 Exercise 3

Student name: Robert Siipola

Student number: 2508568

Student email: roamsi@utu.fi

1.2 IMPORTANT

This exercise involves using AI. Use only the *Study Chat* service provided by the University of Turku at `https://ai.utu.fi/en`. **Do not use any other AI service!** Before starting, remember to carefully read the guidelines for using AI (`https://intranet.utu.fi/en/sites/ai/guidelines/Pages/default.aspx`) and the terms of use. **Do not share any personal information or copyrighted material with AI.**

Save all your discussions (including your prompts and AI's output) as well as the name of the model you used.

1.3 Instructions

The deadline of this exercise is **Wednesday 18 February 2026 at 11:59 PM**. Please contact Juho Heimonen (juaheim@utu.fi) if you have any questions about the exercise. Remember to follow all the general exercise guidelines that are stated in Moodle.

The exercise has several parts, all of which concern the letter below. You will take the role of a data scientist who has been assigned to solve the problem described in the letter. You have an AI tool to assist you, but you alone are responsible for the quality of the solution.

1 Ask AI to write code to solve the task. Analyse which parts of the AI-generated code are correct and which are incorrect. Pay particular attention to the key parts of the cross-validation. You may ask AI to improve the code as many times as you like, as long as you keep analysing its output.

2 Implement the required leave-one-out cross-validations and run your code to get the estimates you were asked to obtain. Here it is okay to use any amount of the AI-generated code you produced above. You can use a complete, fully correct AI-written solution, you can write the implementation from scratch by yourself, or you can take some AI-generated code and complete the implementation manually.

3 Write a report in which you discuss the following:

- Why did the cross-validation described in the letter fail? What is the correct way to do cross-validation here?
- Which parts of the task was AI able to code correctly and which not? Focus particularly on the core of the cross-validation.
- Which parts of the AI-generated code did you use in your implementation? Why? Explain why the selected pieces of code work correctly in your implementation.
- What results did you get with your implementation? Report the estimates and interpret the results in terms of how well the model will work in the situations described in the letter. Explain in detail why your cross-validation is the correct way to estimate the generalisation performance.

Write the report in your own words and explain everything clearly, precisely, and comprehensively. **You are not allowed to use AI to write the report for you** because this is where you show that you have understood the theory and are able to apply it. If you use AI as a teacher (i.e. to explain things to you for learning purposes), you must attach the discussions and clearly state what and how you learnt from the AI. **If there is uncertainty about how the text was produced, you may be required to explain the content of your report in a face-to-face meeting.**

4 Submit the following documents to Moodle:

- The discussions with AI (including your prompts and AI's output), as PDF.
- The implementation of your cross-validation, as PDF and as a Jupyter notebook.
- The report, as PDF. (It is okay to integrate the report to the Jupyter notebook.)

Note that it is not enough to just implement the cross-validation correctly to pass this exercise. You must also explain in plain words what you have done and demonstrate that you understand how cross-validation should be performed on pair-input data. Small errors are acceptable, but you will fail this exercise if there are significant error(s) or omission(s) in the report or in the implementation.

1.4 Letter from your client

Dear Data Scientist,

I have a long-term research project regarding a specific set of proteins. I am attempting to discover small organic compounds that can bind strongly to these proteins and thus act as drugs. I have already made laboratory experiments to measure the affinities between some proteins and drug molecules.

My colleague is working on another set of proteins, and the objectives of his project are similar to mine. He has recently discovered thousands of new potential drug molecules. He asked me if I could find the pairs that have the strongest affinities among his proteins and drug molecules. Obviously I do not have the resources to measure all the possible pairs in my laboratory, so I need to prioritise. I decided to do this with the help 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 already made in the laboratory with my proteins and drug molecules. They comprise of 77 target proteins and 59 drug molecules. To estimate the generalisation performance of the model, I then performed a leave-one-out cross-validation. I used C-index and got a stellar score above 90%. Finally I used the model to predict the affinities of my colleague's proteins and drug molecules. The problem is that when I selected the highest predicted affinities and my colleague tried to verify them in the lab, we found that many of them are much lower in reality. My model clearly does not work despite the high cross-validation score. We also tested the model with my proteins against my colleague's drugs (which is another task I would like to use my model for), but the model did not work there either.

Please explain why my estimation failed and how leave-one-out cross-validation should be performed to get reliable estimates. Also, please implement the leave-one-out cross-validation correctly and report the numbers I need. I want to know whether it would be a waste of my colleague's and my resources if we were to use my model any further.

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 *ith* row in each file is about the same pair.

Looking forward to hearing from you soon.

Yours sincerely,
Bio Scientist

[63]:
```
import os

import pandas as pd
import numpy as np

from sklearn.neighbors import KNeighborsRegressor
from sklearn.model_selection import LeaveOneOut, LeaveOneGroupOut
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
```

[64]:
```
input_df = pd.read_csv("../data/excercise-3/input.data", sep=r"\s+",␣
  ↪dtype=float, header=None)
X = input_df.values

output_df = pd.read_csv("../data/excercise-3/output.data", sep="\t",␣
  ↪header=None)
y = output_df.values.ravel()

pairs_df = pd.read_csv("../data/excercise-3/pairs.data",
    sep=r"\s+",
    header=None,
    names=["drug", "target"],
    dtype=str,
    engine="c"
    )

drugs = pairs_df["drug"].values
targets = pairs_df["target"].values
```

[65]:
```
def c_index(y_true, y_pred):
    n = len(y_true)
    if n <= 1: return 0.0

    concordant = 0
    permissible = 0
    ties = 0

    for i in range(n):
        for j in range(i + 1, n):
            if y_true[i] != y_true[j]:
                permissible += 1
                diff_prod = (y_true[i] - y_true[j]) * (y_pred[i] - y_pred[j])
                if diff_prod > 0:
                    concordant += 1
                elif y_pred[i] == y_pred[j]:
                    ties += 1

    if permissible == 0:
        return 0.0
    return (concordant + 0.5 * ties) / permissible
```

[67]:
```
if X is not None:
    # 1. Standard LOO
    pipeline = make_pipeline(StandardScaler(),␣
 ↪KNeighborsRegressor(n_neighbors=10, weights="distance"))

    loo = LeaveOneOut()
    y_pred_loo = np.zeros_like(y)
    for train_idx, test_idx in loo.split(X):
        pipeline.fit(X[train_idx], y[train_idx])
        y_pred_loo[test_idx] = pipeline.predict(X[test_idx])
    print(f"Standard Leave-One-Out C-index: {c_index(y, y_pred_loo):.4f}")

    # 2. Leave-One-Drug-Out (LDO)
    logo = LeaveOneGroupOut()
    y_pred_ldo = np.zeros_like(y)
    for train_idx, test_idx in logo.split(X, y, groups=drugs): # grouped by␣
↪DRUGS
        pipeline.fit(X[train_idx], y[train_idx])
        y_pred_ldo[test_idx] = pipeline.predict(X[test_idx])
    print(f"Leave-One-Drug-Out C-index:     {c_index(y, y_pred_ldo):.4f}")

    # 3. Leave-One-Target-Out (LTO)
    y_pred_lto = np.zeros_like(y)
    for train_idx, test_idx in logo.split(X, y, groups=targets): # grouped by␣
↪TARGETS
        pipeline.fit(X[train_idx], y[train_idx])
        y_pred_lto[test_idx] = pipeline.predict(X[test_idx])
    print(f"Leave-One-Target-Out C-index:   {c_index(y, y_pred_lto):.4f}")

    y_pred_lbo = np.zeros_like(y)
    n_samples = len(y)

    for i in range(n_samples):
        test_drug = drugs[i]
        test_target = targets[i]

        # Identify training samples:
        # Keep ONLY rows where the drug is DIFFERENT AND the target is DIFFERENT
        # train_mask = (drugs != test_drug) & (targets != test_target)
        train_idx = np.where((drugs != test_drug) & (targets != test_target))[0]

        # Fit on this restricted training set
        pipeline.fit(X[train_idx], y[train_idx])

        # Predict on the single test sample
        prediction = pipeline.predict(X[i].reshape(1, -1))
        y_pred_lbo[i] = prediction[0]

    print(f"Leave-Both-Out C-index:         {c_index(y, y_pred_lbo):.4f}")
```

```
Standard Leave-One-Out C-index: 0.9273
Leave-One-Drug-Out C-index:     0.5120
Leave-One-Target-Out C-index:   0.9276
Leave-Both-Out C-index:         0.5131
```

[72]:
```
if X is not None:
    # Define Parameter Grid for K
    k_grid = np.arange(1, 16)

    # Outer Loop: Leave-One-Drug-Out (The Test)
    outer_cv = LeaveOneGroupOut()
    y_pred_nested = np.zeros_like(y)
```

```
print("Starting Nested Cross-Validation (LDO)...")

# Iterate through each drug as the "Test Drug"
for train_idx_outer, test_idx_outer in outer_cv.split(X, y, groups=drugs):

    # Outer Training Data (All other drugs)
    X_train_out = X[train_idx_outer]
    y_train_out = y[train_idx_outer]
    drugs_train_out = drugs[train_idx_outer]

    # --- INNER LOOP: Hyperparameter Tuning ---
    best_k = k_grid[0]
    best_score = -1.0

    # Inner CV: Leave-One-Drug-Out on the Outer Training Set
    inner_cv = LeaveOneGroupOut()

    for k in k_grid:
        # We need predictions for all samples in the inner loop to␣
↪calculate C-index
        # We can use cross_val_predict, or manual loop
        y_pred_inner = np.zeros_like(y_train_out)

        for train_idx_in, val_idx_in in inner_cv.split(X_train_out,␣
↪y_train_out, groups=drugs_train_out):
            # Pipeline for the fold
            pipeline = make_pipeline(StandardScaler(),␣
↪KNeighborsRegressor(n_neighbors=k, weights='distance'))
            pipeline.fit(X_train_out[train_idx_in],␣
↪y_train_out[train_idx_in])
            y_pred_inner[val_idx_in] = pipeline.
↪predict(X_train_out[val_idx_in])

        # Calculate score for this K
        score = c_index(y_train_out, y_pred_inner)
        if score > best_score:
            best_score = score
            best_k = k

    # --- END INNER LOOP ---

    # Refit on the FULL Outer Training Set using the Best K
    final_model = make_pipeline(StandardScaler(),␣
↪KNeighborsRegressor(n_neighbors=best_k, weights='distance'))
    final_model.fit(X_train_out, y_train_out)
```

```
        # Predict on the Left-Out Drug (Outer Test Set)
        y_pred_nested[test_idx_outer] = final_model.predict(X[test_idx_outer])

    # Final Evaluation
    print(f"Nested CV (LDO) C-index: {c_index(y, y_pred_nested):.4f}")
    print(f"Best K: {best_k}")
```

```
Starting Nested Cross-Validation (LDO)…
Nested CV (LDO) C-index: 0.5340
Best K: 1
```

2 Analysis

2.1 Why did standard Leave-One-Out Cross-Validation (LOOCV) produce overtly optimistic results?

LOOCV produced overtly optimistic results because while it holds out one measured drug-target pair at a time, it still leaves in the training set other measurements that share the same drug and/or the same target. Fundamentally drug-target affinity is a paired-input problem and in such problems there are different generalisation settings, which were discussed in Pahikkala et al.:

```
* S1: Both the drug and the target are encountered in the training set
* S2: Target is seen in the training set, but not the drug.
* S3: Drug is seen in the training set, but not the target.
* S4: Neither one is seen in the training set.
```

Each setting requires its own approach and the scientist implemented LOOCV, which is suitable to S1 to a setting which is best described either by settings S2 or S4.

2.2 How to correctly do the Cross-Validation (CV)

Depending on the setting you either need to do Leave-Drug-Out (LDOCV) which corresponds to S2 or Leave-Both-OUT (LBOCV) which corresponds to S4. Also, if one want's to experiment with the value for K then one needs to implement nested CV, where the inner loop loops through the K-values and the outer one does the target and drug CVs.

2.3 Gemini performance

I used Google's Gemini 3 Pro for the task (with explicit permission from the person responsible for the excercise). **The link to the discussion can be found here** The model got the gist of the task fairly well, but did not automatically suggest an implementation for S4. Still its basic CV code for S1-S3 seemed to be correct. For S2 and S3 the model uses ScikitLearn's `LeaveOneGroupOut` method for the drug/target grouping:

```
    logo = LeaveOneGroupOut()
    y_pred_ldo = np.zeros_like(y)
    for train_idx, test_idx in logo.split(X, y, groups=drugs): # grouped by DRUGS
        pipeline.fit(X[train_idx], y[train_idx])
        y_pred_ldo[test_idx] = pipeline.predict(X[test_idx])
    print(f"Leave-One-Drug-Out C-index:     {c_index(y, y_pred_ldo):.4f}")
```

Interestingly, the model was not initially able to reproduce the over 90% accuracy of the LOOCV result. Only after noting about the mismatch did it decide to try to change the "weighting" parameter to "distance", with which we got >90% in the LOOCV.

For the LBOCV Gemini implemented a mask to perform the CV:

```
    y_pred_lbo = np.zeros_like(y)
    n_samples = len(y)

    for i in range(n_samples):
        test_drug = drugs[i]
        test_target = targets[i]

        # Identify training samples:
        # Keep ONLY rows where the drug is DIFFERENT AND the target is DIFFERENT
        # train_mask = (drugs != test_drug) & (targets != test_target)
        train_idx = np.where((drugs != test_drug) & (targets != test_target))[0]

        # Fit on this restricted training set
        pipeline.fit(X[train_idx], y[train_idx])

        # Predict on the single test sample
        prediction = pipeline.predict(X[i].reshape(1, -1))
        y_pred_lbo[i] = prediction[0]

    print(f"Leave-Both-Out C-index:         {c_index(y, y_pred_lbo):.4f}")
```

Interestingly, Gemini 3 Pro was able to produce an implementation for the nested CV for LDOCV but not for LBOCV (see discussion PDF).

2.4 Results

For K=10, we got the following results:

```
* Standard Leave-One-Out C-index: 0.9273
* Leave-One-Drug-Out C-index:     0.5120
* Leave-One-Target-Out C-index:   0.9276
* Leave-Both-Out C-index:         0.5131
```

With a C-index of ~0.51 the model is only about 1 percentage point better than random at ordering affinities in the Leave-Both-Out setting (new drug and new target).