Arttu Kuitunen
Student number 1500155
February, 2, 2025

# Exercise 1 | TKO_7092 Evaluation of Machine Learning Methods 2025

Prediction of the metal ion content from multi-parameter data

**Use K-Nearest Neighbor Regression with euclidean distance to predict total metal concentration (c_total), concentration of Cadmium (Cd) and concentration of Lead (Pb), using number of neighbors k = 1, 3, 5, 7.**

```
- You may use Nearest Neighbor Regression from https://scikit-learn.org/stable/modules/neighbors.html
- Implement Leave-One-Out cross-validation and calculate the C-index for each output (c_total, Cd, Pb).
- Implement Leave-Replicas-Out cross-validation and calculate the C-index for each output (c_total, Cd, Pb).
- Return your solution as a Jupyter Notebook .ipynb notebook and as a PDF-file made from it. Please, add your full name to the file name.
- The exercise will be graded by a maximum of 2 points.
- Submit to moodle your solution on ** Wednesday 5 of February ** at the latest.
```

**Please be aware that you are required to submit your individual solution. Submissions with identical or similar code will result in a failure for the exercise.**

## Import libraries

In [199…

```python
# In this cell import all libraries you need. For example:
import numpy as np
import pandas as pd
from scipy.stats import somersd
from sklearn.neighbors import KNeighborsRegressor
from sklearn.model_selection import LeaveOneOut
import matplotlib.pyplot as plt
```

## Read and visualize the dataset

**Note:** This dataset differs slightly from the one used in the video lectures.
**In this dataset, some mixtures have 3 replicas, while others have 4 replicas.**

In the following cell:

- Read the file water_data.csv
- Print the dimesions of the dataset (i.e. number of rows and columns) and display the first 5 rows.
- Identify the inputs and the outputs columns.
- Provide the number of mixtures with 3 replicas and 4 replicas, respectively.

In [200…
```
#  Maybe I want average of the replicas, since some have 3 and some 4 replicas
data = pd.read_csv('water_data.csv')
input_columns = ['Mod1', 'Mod2', 'Mod3']
output_columns = ['c_total', 'Cd', 'Pb']

outputs = data[input_columns]
inputs = data[output_columns]

print("Shape (rows, columns):\n", data.shape),
print("\nFirst 5 rows:\n", data.head()),
print("Inputs:", inputs.columns),
print("Outputs:", outputs.columns)
```

```
Shape (rows, columns):
 (225, 6)

First 5 rows:
    c_total   Cd    Pb      Mod1      Mod2      Mod3
0      0.0  0.0   0.0 -0.999216 -0.714208 -0.414911
1      0.0  0.0   0.0 -0.990800 -0.714373 -0.238335
2      0.0  0.0   0.0 -0.990539 -0.714125  0.020788
3     14.0  0.0  14.0 -1.001247 -0.713546  0.945465
4     14.0  0.0  14.0 -1.013727 -0.714125  0.569631
Inputs: Index(['c_total', 'Cd', 'Pb'], dtype='object')
Outputs: Index(['Mod1', 'Mod2', 'Mod3'], dtype='object')
```

In [201…
```
# The amount of occurrences in dataset
four = (data.groupby(output_columns).size() == 4).sum()
three = (data.groupby(output_columns).size() == 3).sum()

if four * 4 + three * 3 == len(data):
    print("Matching amount of replicas to data")
else:
    print("Not matching amount of replicas to data")
```

```
Matching amount of replicas to data
```

# C-index code

In [202…
```
# In this cell is the fuction that computes the c-index value based on Somers'D sta
# Use this fuction as the evaluation metric in the Leave-One-Out (LOOCV) and Leave-

def cindex(true, pred):
    s_d = somersd(true, y=pred, alternative='two-sided')
```

```
    c_index = (s_d.statistic + 1.0)/2.0
    return c_index
```

# Functions

In [203…

```
# In this cell add the functions that you need for the data analysis part.

# Leave-one-out cross-validation
def leave_one_out_cv(df, k_values):
    loo = LeaveOneOut()
    results = {k: {col: [] for col in output_columns} for k in k_values}

    # Iterating every train-test split possible
    for train_idx, test_idx in loo.split(df):
        train_data, test_data = df.iloc[train_idx], df.iloc[test_idx]
        X_train, y_train = train_data[input_columns], train_data[output_columns]
        X_test, y_test = test_data[input_columns], test_data[output_columns]

        # Fit models for every k neighbors
        for k in k_values:
            model = KNeighborsRegressor(n_neighbors=k, metric='euclidean')
            model.fit(X_train, y_train)
            predictions = model.predict(X_test)

            # set results for outputs
            for i, col in enumerate(output_columns):
                results[k][col].append((y_test.iloc[0][col], predictions[0][i]))

    c_index_scores = {k: {col: cindex(*zip(*results[k][col])) for col in output_col
    return c_index_scores

# Leave-replicas-out cross-validation
def leave_replicas_out_cv(df, k_values):
    results = {k: {col: [] for col in output_columns} for k in k_values}

    # Get all unique groups
    groups = list(df.groupby(output_columns).groups.values())

    # For each sample in each group
    for test_indices in zip(*groups):  # Combines one replica from each group

        # Create combined train/test split
        test_data = df.loc[list(test_indices)]
        train_data = df.drop(list(test_indices), axis=0)

        X_train, y_train = train_data[input_columns], train_data[output_columns]
        X_test, y_test = test_data[input_columns], test_data[output_columns]

        # Fit models for every k neighbors
        for k in k_values:
            model = KNeighborsRegressor(n_neighbors=k, metric='euclidean')
            model.fit(X_train, y_train)
            predictions = model.predict(X_test)
```

```
            for i, col in enumerate(output_columns):
                for j in range(len(test_indices)):
                    results[k][col].append((y_test.iloc[j][col], predictions[j][i])

    c_index_scores = {k: {col: cindex(*zip(*results[k][col])) for col in output_col
    return c_index_scores
```

In [204…

```
# Some list comprehension help I found.
# https://stackoverflow.com/questions/40646458/list-comprehension-in-pandas

# Cross validation practical python example
# https://www.youtube.com/watch?v=-8s9KuNo5SA
```

# Results for Leave-One-Out cross-validation

In [205…

```
# Here run your script for Leave-One-Out cross-validation and print the correspondi

k_values = [1, 3, 5, 7]
loocv_results = leave_one_out_cv(data, k_values)
print("\nLeave-one-out Cross-validation results:")
for k, scores in loocv_results.items():
    print(f"k={k}: {scores}")
```

```
Leave-one-out Cross-validation results:
k=1: {'c_total': 0.9082833811137173, 'Cd': 0.921869127656909, 'Pb': 0.88054871173842
23}
k=3: {'c_total': 0.9141907740422205, 'Cd': 0.8995907629348143, 'Pb': 0.8744519146448
406}
k=5: {'c_total': 0.8941012944140387, 'Cd': 0.8619660082682591, 'Pb': 0.8542614941328
768}
k=7: {'c_total': 0.8737294761532447, 'Cd': 0.8141520858562659, 'Pb': 0.8355326345680
043}
```

# Results for Leave-Replicas-Out cross-validation

In [206…

```
# Here run your script for Leave-Replicas-Out cross-validation and print the corres

lrocv_results = leave_replicas_out_cv(data, k_values)
print("\nLeave-replicas-out cross-validation results:")
for k, scores in lrocv_results.items():
    print(f"k={k}: {scores}")
```

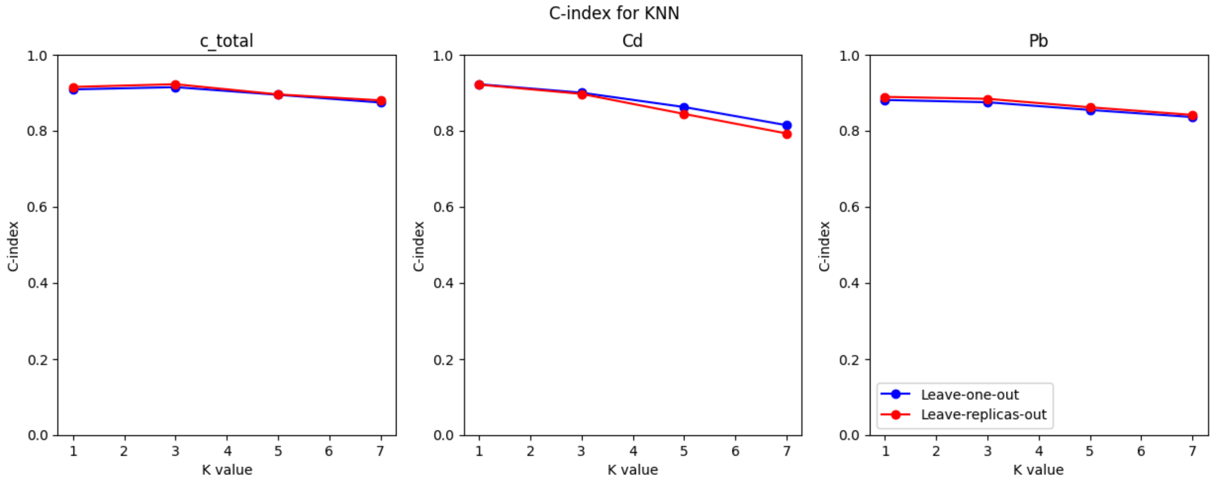
```
Leave-replicas-out cross-validation results:
k=1: {'c_total': 0.9147659389594873, 'Cd': 0.9208777987026575, 'Pb': 0.8886534839924
67}
k=3: {'c_total': 0.9218529379819702, 'Cd': 0.8964741577735929, 'Pb': 0.8835268884703
913}
k=5: {'c_total': 0.8951069838166612, 'Cd': 0.8438480853735091, 'Pb': 0.8612680477087
257}
k=7: {'c_total': 0.8794395568589117, 'Cd': 0.7924513496547395, 'Pb': 0.8411801632140
615}

k=1: {'c_total': 0.9147659389594873, 'Cd': 0.9208777987026575, 'Pb': 0.8886534839924
67}
k=3: {'c_total': 0.9218529379819702, 'Cd': 0.8964741577735929, 'Pb': 0.8835268884703
913}
k=5: {'c_total': 0.8951069838166612, 'Cd': 0.8438480853735091, 'Pb': 0.8612680477087
257}
k=7: {'c_total': 0.8794395568589117, 'Cd': 0.7924513496547395, 'Pb': 0.8411801632140
615}
```

## Plot Leave-One-Out and Leave-Replicas-Out Results

Note: You may plot the results as they were presented in the video lecture (refer to MOOC2-Module 2 .pptx slides).

In [207…

```python
# Create figure with 3 subplots

# Axes for each output
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(15, 5))
k_values = [1, 3, 5, 7]

loocv_plot = {col: [loocv_results[k][col] for k in k_values] for col in output_colu
lrocv_plot = {col: [lrocv_results[k][col] for k in k_values] for col in output_colu

# Plot with lines connecting points
ax1.plot(k_values, loocv_plot['c_total'], 'bo-')
ax1.plot(k_values, lrocv_plot['c_total'], 'ro-')

ax2.plot(k_values, loocv_plot['Cd'], 'bo-')
ax2.plot(k_values, lrocv_plot['Cd'], 'ro-')

ax3.plot(k_values, loocv_plot['Pb'], 'bo-')
ax3.plot(k_values, lrocv_plot['Pb'], 'ro-')

# Set titles and labels
titles = ['c_total', 'Cd', 'Pb']
fig.suptitle('C-index for KNN')
for ax, title in zip([ax1, ax2, ax3], titles):
    ax.set_title(title)
    ax.set_xlabel('K value')
    ax.set_ylabel('C-index')
    ax.set_ylim(0, 1)

plt.legend(['Leave-one-out', 'Leave-replicas-out'])
plt.show()
```




# Interpretation of results

## Answer the following questions based on the results obtained

1. Which cross-validation approach produced more optimistic results, and why?
2. Which cross-validation method provides a better estimate of the model's performance on unseen mixtures? Explain your answer.

### Answers:

1. At first, leave-one-out was highly more optimal, showing similiar difference than in the lecture slideshow. At first I performed leave-replicas-out in more of leave-group-out fashion, where every group was trained individually, which in my opinion was not the goal here. Leave-one-out has more training data, therefore it should perform better. It takes more computing.
2. I guess leave-replicas-out, since it is less likely to over fitting.