Jeremias Shadbolt
Student number 1900386
7 - 8.2.2023

Exercise 2 | TKO_7092 Evaluation of Machine Learning Methods 2023

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), for each sample using number of neighbors k = 3.

```
- You may use Nearest Neighbor Regression from https://scikit-
learn.org/stable/modules/neighbors.html
- The data should be standarized using z-score. (Using
sklearn.preprocessing.StandardScaler is allowed)
- Implement your own Leave-One-Out cross-validation and calculate the C-
index for each output (c_total, Cd, Pb).
- Implement your own 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.
- Submit to moodle your solution on ** Wednesday 8 of February ** at the
latest.
```

Import libraries

In [1]:
```
#In this cell import all libraries you need. For example:
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsRegressor
```

Read and visualize the dataset

In [2]:
```
#In this cell read the file Water_data.csv
#Print the dataset dimesions (i.e. number of rows and columns)
#Print the first 5 rows of the dataset
data = pd.read_csv('Water_data.csv')

display(data.shape)
display(data.head())
display(data.dtypes)
```

```
(225, 6)
```

	c_total	Cd	Pb	Mod1	Mod2	Mod3
0	0	0.0	0.0	9945	119	72335
1	0	0.0	0.0	10786	117	82977
2	0	0.0	0.0	10812	120	98594
3	14	0.0	14.0	9742	127	154323

	c_total	Cd	Pb	Mod1	Mod2	Mod3
4	14	0.0	14.0	8495	120	131672

```
c_total      int64
Cd         float64
Pb         float64
Mod1         int64
Mod2         int64
Mod3         int64
dtype: object
```

To show understanding of the data, answer the following questions:

- How many different mixtures of Cadmium (Cd) and Lead (Pb) were measured?
- How many total concentrations (c_total) were measured?
- How many mixtures have less than 4 replicas?
- Make plots of Lead (Pb) and Cadmium (Cd) mixtures for low and high concentrations.
  Where low concentrations are those with c_total <= 100, while in high concentration c_total > 100.
  Hint: plots are similar to the ones presented in the video lecture.

In [3]:

```
#In this cell write the code to answer the previous questions and print the answers.
print(f"Total different mixtures: {len(data[['Cd', 'Pb']].drop_duplicates())}")
print(f"Total concentrations measured: {len(data['c_total'].unique())}")
print(f"Amount of mixtures with less than 4 replicas: {len([mixture for mixture in data[['
```

```
Total different mixtures: 67
Total concentrations measured: 12
Amount of mixtures with less than 4 replicas: 0
```

In [4]:

```
high_concentration = data[data['c_total'] > 100]
low_concentration = data[data['c_total'] <= 100]
display(high_concentration.head())
display(low_concentration.head())
```

	c_total	Cd	Pb	Mod1	Mod2	Mod3
129	200	0.0	200.0	32540	8047	56799
130	200	0.0	200.0	32365	7653	52215
131	200	0.0	200.0	35378	7998	51276
132	200	0.0	200.0	31259	7282	54850
133	200	40.0	160.0	163432	16606	59335

	c_total	Cd	Pb	Mod1	Mod2	Mod3
0	0	0.0	0.0	9945	119	72335
1	0	0.0	0.0	10786	117	82977
2	0	0.0	0.0	10812	120	98594
3	14	0.0	14.0	9742	127	154323
4	14	0.0	14.0	8495	120	131672

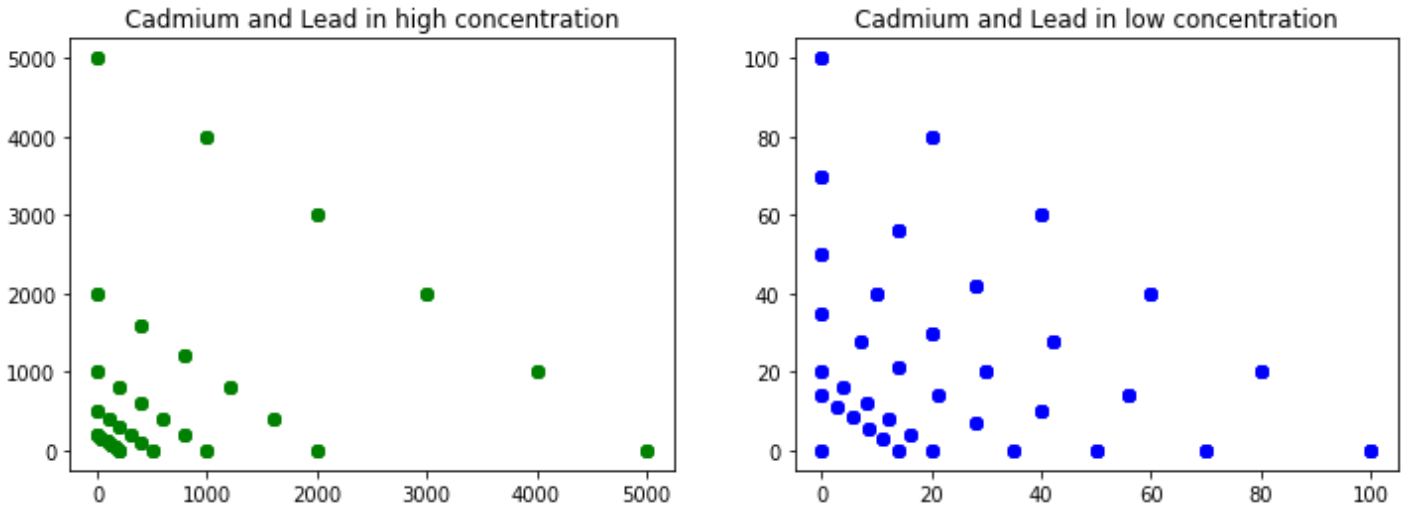
In [5]:

```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize = (12, 4))

ax1.scatter(high_concentration['Cd'], high_concentration['Pb'], color = 'Green')
```

```
ax1.set_title('Cadmium and Lead in high concentration')
ax2.scatter(low_concentration['Cd'], low_concentration['Pb'], color = 'Blue')
ax2.set_title('Cadmium and Lead in low concentration')

plt.show()
```



Seems about right

Standardization of the dataset

In [6]:
```
#In this cell standardize the dataset features by removing the mean and scaling to unit va
#In other words, use z-score to scale the dataset features (Mod1, Mod2, Mod3)
#Print the 5 first samples (i.e. rows) of the scaled dataset

# Transform the features
features = ['Mod1', 'Mod2', 'Mod3']
data[features] = StandardScaler().fit_transform(data[features])
```

In [7]:
```
data[features].head()
```

Out[7]:

	Mod1	Mod2	Mod3
0	-0.999216	-0.714208	-0.414911
1	-0.990800	-0.714373	-0.238335
2	-0.990539	-0.714125	0.020788
3	-1.001247	-0.713546	0.945465
4	-1.013727	-0.714125	0.569631

C-index code

In [8]:
```
# An honest try to implement this from scratch was done, but after numerous fails i had to
# the correct code from exc 1.
def cindex(true_labels, pred_labels):
    """Returns C-index between true labels and predicted labels"""
    n = 0
    h_num = 0
    for i in range(0, len(true_labels)):
        t = true_labels[i]
        p = pred_labels[i]
```

```
        for j in range(i+1, len(true_labels)):
            nt = true_labels[j]
            np = pred_labels[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
    cindex = h_num / n
    return cindex
```

In [9]:

```
#test cindex function with following values
true_labels = [-1, 1, 1, -1, 1]
predictions = [0.60, 0.80, 0.75, 0.75, 0.70]
cindx = cindex(true_labels, predictions)
print(cindx)
```

```
0.75
```

Functions

Include here all the functions that you need to run in the data analysis part.
Note: using a leave-one-out and leave-replicas-out cross-validation from an already made package (e.g. Scikit-learn) is not accepted.

Leave one out

In [10]:

```
def leave_one_out_cross_validation(model, X, y):
    n = X.shape[0]
    predictions = list()
    actuals = list()
    for i in range(n):
        X_train = np.concatenate([X[:i], X[i+1:]])
        y_train = np.concatenate([y[:i], y[i+1:]])
        X_test = X[i].reshape(1, -1)
        y_test = y[i].reshape(1, -1)

        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)
        predictions.append(y_pred)
        actuals.append(y_test)

    return cindex(actuals, predictions)
```

Leave Replica Out

Credit where credit is due note: Some help was received from a fellow student. Not much, just some 'osviitta' as one would say.

In [11]:

```
def leave_replica_out(data, el1, el2, X, y, model):

    # Lists for folds, training set and predictions and also an indexer
    folds = list()
    trains = list()
    predictions = list()
    i = 0
    # Loop through the given data
    while i < len(data):
        # Save the current row
```

```
        tmp1 = el1[i]
        tmp2 = el2[i]
        # Initialize replicate rows counter
        replicates = 0
        # Check for replicated rows, if none: break loop
        for j in range(i, len(data)):
            if el1[j] == tmp1 and el2[j] == tmp2:
                replicates += 1
            else:
                break
        # Add saved rows to folds
        folds.append(data.loc[i: i + replicates - 1])
        # Remove the rows from the training set
        rows_to_remove = list(range(i, i + replicates))
        trains.append(data.drop(rows_to_remove))
        # Skip the replica rows
        i += replicates

    for i in range(0, len(data)):
        # This is for edge cases when the test set is one of the last rows
        try:
            train_all = trains[i]
            fold_all = folds[i]
        except:
            pass
        # Parameters should be: X = Mod1, Mod2, Mod3; y = 'c_tota'l/'Cd'/'Pb'
        x_train = train_all[list(X)]
        y_train = train_all[(y)]
        x_test = fold_all[list(X)]


        model.fit(x_train, y_train)
        prediction = model.predict(x_test)
        predictions.extend(prediction)
    # Return C-index score for prections
    return(cindex(data[(y)], predictions))
```

Results for Leave-One-Out cross-validation

In [14]:

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

# Initialize knn for each target variable
knn_c_total = KNeighborsRegressor(n_neighbors=3)
knn_cd = KNeighborsRegressor(n_neighbors=3)
knn_pb = KNeighborsRegressor(n_neighbors=3)

# Split the data into feature variables and target variables
x_data = data.drop(["c_total", "Cd", "Pb"], axis=1)
x_data = x_data.values
y1 = data["c_total"]
y2 = data["Cd"]
y3 = data["Pb"]
y1 = y1.values
y2 = y2.values
y3 = y3.values

c_total_loocv_results = leave_one_out_cross_validation(knn_c_total, x_data, y1)
cd_loocv_results = leave_one_out_cross_validation(knn_cd, x_data, y2)
pb_loocv_results = leave_one_out_cross_validation(knn_pb, x_data, y3)

print(f"Results for Total Concentration: {c_total_loocv_results:.5f}")
print(f"Results for Cadmium: {cd_loocv_results:.5f}")
print(f"Results for Lead: {pb_loocv_results:.5f}")
```

```
Results for Total Concentration: 0.91419
Results for Cadmium: 0.89959
Results for Lead: 0.87445
```

Results for Leave-Replicas-Out cross-validation

In [13]:

```
#In this cell run your script for Leave-Replicas-Out cross-validation and print the corres
lro_data = data.sort_values(by = ['Pb', 'Cd', 'c_total'])
lro_data = lro_data.reset_index(drop = True)

lro_y1 = lro_data['Cd']
lro_y2 = lro_data['Pb']
mod_data = ['Mod1', 'Mod2', 'Mod3']

# Initialize knn for each target variable
knn_c_total = KNeighborsRegressor(n_neighbors=3)
knn_cd = KNeighborsRegressor(n_neighbors=3)
knn_pb = KNeighborsRegressor(n_neighbors=3)

print(f"C-index for total concentration using LROCV: {leave_replica_out(lro_data, lro_y1,
print(f"C-index for Cd concentration using LROCV: {leave_replica_out(lro_data, lro_y1, lro
print(f"C-index for Pb concentration using LROCV: {leave_replica_out(lro_data, lro_y1, lro
```

```
C-index for total concentration using LROCV: 0.81867
C-index for Cd concentration using LROCV: 0.76145
C-index for Pb concentration using LROCV: 0.76895
```

Interpretation of results

Answer the following questions based on the results obtained

- Which cross-validation approach had more optimistic results?
- Which cross-validation generalize better on unseen data? Why?

In this cell write your answers to the questions about Interpretation of Results.

C Index results would indicate that LOOCV was more optimistic. But since our data has multiple replicas, LROCV gives a more realistic estimate on model performance thus it's able to generalize better on unseen data.

In [ ]: