# Introduction to Data Science 2023
# Assignment 2

Thomas Hamelryck, Stella Frank, Daniel Hershcovich, François Lauze, Antonia Karamolegkou

Your solution to Assignment 2 must be uploaded to Absalon no later than Monday, February 27, 2023, 23:59.

Guidelines for the assignment:

- **The assignments in IDS must be completed and written individually.** This means that your code and report must be written completely by yourself.
- Upload your report as a single PDF file (no Word .docx file) named `firstname_lastname.pdf`. The report should include all the points of the deliverables stated at the end of each exercise. Adding code snippets to the report is optional, but nice to have especially when you implement things from scratch.
- Upload your Python code. Upload it in a zip archive, even if there is only one file. Expected is either one or several "`.py`" files or a "`.ipynb`" notebook.
- We will grade using Python 3.10 and the specific package versions specified in requirements.txt. Other versions may work, or may break; using these versions is the safest. You can create an appropriate environment with all the requirements using Anaconda. You can use the same environment you had in the previous assignment and install the new requirements file. To do so, open a terminal or command prompt and navigate to the directory containing the requirements.txt using `cd` (e.g. `cd Desktop/courses/ids/as2`). Then activate your environment (make sure to replace `name_of_env` with the name you specified in the previous assignment):

  ```
  conda activate name_of_env
  ```

  Install the requirements running:

  ```
  pip install -r requirements.txt
  ```

  If you want to create a new environment you first need to run (and then follow the previous steps):

  ```
  conda create --name my_env python=3.10
  ```

Each exercise is worth 10 points, the total number of points for this assignment is 40.

## Classification with nearest neighbors

In the following, we will work with three fundamental data analysis techniques. We will perform classification with the *nearest neighbor classifier*, a non-linear, non-parametric method for classification. Then we will apply *cross-validation* for model selection and standard *data normalization* for preprocessing.

### The data

The data for the following tasks are taken from a research project from the University of Mons, Belgium (Candanedo and Feldheim [2016]) on the detection of occupancy in buildings for the purpose of potential energy saving.

**Introduction to the problem (not relevant for solving the exercises).** Have you ever thought about how some buildings are able to adjust their energy usage based on whether or not people are inside? That's occupancy detection! It typically involves the use of sensors or other types of technology to collect data on movement, heat, or other indicators that suggest the presence of people. The data is then analyzed to determine occupancy patterns and inform decisions on building operations such as lighting, heating, and cooling. Building occupancy detection can help improve indoor air quality, enhance occupant comfort, and most importantly, reduce energy consumption. Different studies for different types of occupancy report savings ranging from 29% to 80%. Also, being able to detect occupancy without resorting to a camera can become necessary in order to avoid privacy concerns and GDPR violations.

**Description of the data.** Each record contains the following data

- temperature, in Celcius,
- relative humidity, %,
- Light in lux,
- CO2 in ppm,
- humidity ratio, derived quantity from temperature and relative humidity, in kg-water-vapor/kg-air,
- occupancy, 0=not occupied, 1=occupied – the class label.

The original data have been reduced in size to limit computations. We are going to use the data for training and testing a model, and you can find the corresponding training and test data in the files `OccupancyTrain.csv` and `OccupancyTest.csv`.

We have already seen how to read a dataset with the Pandas library. In this case we have to use `.iloc` function to select a subset of the columns and `.values` to convert the selected columns to a NumPy array. This is useful for many reasons, such as being able to perform mathematical operations on the column. If we don't use `.values` at the end, the resulting object would be a Pandas Series instead of a NumPy array.

```
import pandas as pd
# read in the data
dataTrain = pd.read_csv('OccupancyTrain.csv', header=None)
dataTest = pd.read_csv('OccupancyTest.csv', header=None)
# split input variables and labels
XTrain = dataTrain.iloc[:, :-1].values # use all rows and all but the last column
YTrain = dataTrain.iloc[:, -1].values # use all rows, only the last column
XTest = dataTest.iloc[:, :-1].values
YTest = dataTest.iloc[:, -1].values
```

Since we want the data to be in an array format, another way is to load the data directly using Numpy (this is the same as above, so do not do both):

```
import numpy as np
# read in the data
dataTrain = np.loadtxt('OccupancyTrain.csv', delimiter=',')
dataTest = np.loadtxt('OccupancyTest.csv', delimiter=',')
# split input variables and labels
XTrain = dataTrain[:, :-1] # use all rows and all but the last column
YTrain = dataTrain[:, -1] # use all rows, only the last column
XTest = dataTest[:, :-1]
YTest = dataTest[:, -1]
```

### Neartest neighbor classification

**Exercise 1** (Neartest neighbor classification)**.** Apply a nearest neighbor classifier (1-NN) to the data. You are encouraged to implement it on your own. However, you can also use scikit-learn:

```
from sklearn.neighbors import KNeighborsClassifier
```

After training the model, you are supposed to determine the classification accuracy of your model on the training and test data. One way to do this in Python is the following:

```
from sklearn.metrics import accuracy_score
# given classifier called knn, compute the accuracy on the test set
accTest = accuracy_score(YTest, knn.predict(XTest))
```

*Deliverables.* Training and test results of your 1-NN classifier; discussion of the results.

| As a general rule, you must not use the test data in the model building process at all (neither for training, data normalization nor hyperparameter selection - see below), because otherwise, you may get a biased estimate of the generalization performance of the model. |
|---|

### Hyperparameter selection using cross-validation

Hyperparameter selection using cross-validation is a technique used in machine learning to choose the best hyperparameters for a model and improve their accuracy/performance.

Hyperparameters are settings that are not learned by the model during training but instead are set by the user before training begins. The performance of the $k$ nearest neighbor classifier ($k$-NN) depends on the choice of the parameter $k$ determining the number of neighbors. Such a parameter of the learning algorithm (i.e., not a parameter of the resulting model) is called a *hyperparameter*.

Cross-validation involves splitting the available data into several subsets, or "folds", and using each fold as a validation set while training the model on the remaining folds. The process is repeated multiple times, with each fold taking a turn as the validation set. By testing the model's performance on different folds with different hyperparameter settings, cross-validation provides a way to evaluate the model's ability to *generalize to new data* and to *select the hyperparameters* that perform best on average.

Cross-validation is supported in scikit-learn, for example:

```
from sklearn.model_selection import KFold
# create indices for Cross Validation (CV)
cv = KFold(n_splits=5)
# loop over CV folds
for train, test in cv.split(XTrain):
        XTrainCV, XTestCV, YTrainCV, YTestCV = XTrain[train],
        XTrain[test], YTrain[train], YTrain[test]
```

**Exercise 2** (Cross-validation)**.** You are supposed to find a good value for $k$ from $\{1, 3, 5, 7, 9, 11\}$. For every choice of $k$, estimate the performance of the $k$-NN classifier using 5-fold cross-validation and visualize the 0-1 loss (classification error) for different values of k. Pick the $k$ with the lowest average 0-1 loss, which we will call $k_{\text{best}}$ in the following. Only use the training data in the cross-validation process to generate the folds.

*Deliverables.* Short description on how you proceeded; found parameter $k_{\text{best}}$; a plot of the losses for the different values of $k$.

**Exercise 3** (Evaluation of classification performance)**.** To estimate the generalization performance, build a $k_{\text{best}}$-NN classifier using the complete training data set `OccupancyTrain.csv` and evaluate it on the independent test set `OccupancyTest.csv`.

*Deliverables.* Training and test accuracy of $k_{\text{best}}$.

### Data normalization

Data normalization is an important preprocessing step that helps transform the data on a similar scale. A basic normalization method is to generate zero-mean, unit variance input data. This ensures that the distribution of each feature is centered at zero with a standard deviation of one, making it easier for the model to learn from the data and reducing the impact of outliers or features with large variances. If you want to learn more about input normalization see the first three pages of Section 9.1 from the chapter available in Absalon Modules Lecture 5 [Abu-Mostafa et al., 2012].

**Exercise 4** (Data normalization)**.** Center and normalize the data and repeat the model selection and classification process from Exercise 2 and Exercise 3. However, keep the general rule from above in mind.

You can implement the normalization yourself. First, compute the mean and the variance of every input feature (i.e. of every component of the input vector). Then, find the affine linear mapping that transforms the training input data such that the mean and the variance of every feature are zero and one, respectively, after the transformation.

You may also use scikit-learn. Here are three different ways how one could apply the preprocessing from scikit-learn, only one of which is correct:

```
from sklearn import preprocessing
# version 1
scaler = preprocessing.StandardScaler().fit(XTrain)
XTrainN = scaler.transform(XTrain)
XTestN = scaler.transform(XTest)
# version 2
scaler = preprocessing.StandardScaler().fit(XTrain)
XTrainN = scaler.transform(XTrain)
scaler = preprocessing.StandardScaler().fit(XTest)
XTestN = scaler.transform(XTest)
# version 3
XTotal = np.concatenate((XTrain, XTest))
scaler = preprocessing.StandardScaler().fit(XTotal)
XTrainN = scaler.transform(XTrain)
XTestN = scaler.transform(XTest)
```

Discuss which version is correct and why the other two are not (even if you implemented the normalization without scikit-learn). You can briefly discuss what each normalization approach does, and argue for the best way to normalize this specific dataset.

*Deliverables.* Discussion of the three normalization variants including conceptual arguments why two of them are flawed; parameter $k_{\text{best}}$ found in the cross-validation procedure; training and test accuracy of $k_{\text{best}}$; short discussion comparing results with and without normalization

## References


Y. S. Abu-Mostafa, M. Magdon-Ismail, and H.-T. Lin. *Learning from data.* AMLbook, 2012.

L. M. Candanedo and V. Feldheim. Accurate occupancy detection of an office room from light, temperature, humidity and CO2 measurements using statistical learning models. *Energy and Buidlings*, pages 28–39, 2016.