Machine Learning A

2025-2026

Home Assignment 1

Christian Igel and Yevgeny Seldin
Department of Computer Science
University of Copenhagen

The deadline for this assignment is **11 September 2025, 22:00**. You must submit your *individual* solution electronically via the Absalon home page.

A solution consists of:

- A PDF file with detailed answers to the questions, which may include graphs and tables if needed. Do *not* include your full source code in the PDF file, only selected lines if you are asked to do so.
- A .zip file with all your solution source code with comments about the major steps involved in each question (see below). Source code must be submitted in the original file format, not as PDF. The programming language of the course is Python.
- IMPORTANT: Do NOT zip the PDF file, since zipped files cannot be opened in *SpeedGrader*. Zipped PDF submissions will not be graded.
- Your PDF report should be self-sufficient. I.e., it should be possible to grade it without opening the .zip file. We do not guarantee opening the .zip file when grading.
- Your code should be structured such that there is one main file (or one main file per question) that we can run to reproduce all the results presented in your report. This main file can, if you like, call other files with functions, classes, etc.
- Handwritten solutions will not be accepted. Please use the provided LaTeX template to write your report.

1 Make Your Own (10 points)

Imagine that you would like to write a learning algorithm that would predict the final grade of a student in the Machine Learning course based on their profile, for example, their grades in prior courses, their study program, etc. Such an algorithm would have been extremely useful: we could save significant time on grading and predict the final grade when the student just signs up for the course. We expect that the students would also appreciate such service and avoid all the worries about their grades. Anyhow, if you were to make such an algorithm,

1. What profile information would you collect and what would be the sample space $\mathcal{X}$?
2. What would be the label space $\mathcal{Y}$?
3. How would you define the loss function $\ell(y, \hat{y})$?
4. Assuming that you want to apply K-Nearest-Neighbors, how would you define the distance measure $d(x, x')$?
5. How would you evaluate the performance of your algorithm? (In terms of the loss function you have defined earlier.)
6. Assuming that you have achieved excellent performance and decided to deploy the algorithm, would you expect any issues coming up? How could you alleviate them?

There is no single right answer to the question. The main purpose is to help you digest the definitions we are working with. Your answer should be short, no more than 2-3 sentences for each bullet point. For example, it is sufficient to mention 2-3 items for the profile information, you should not make a page-long list.

2 Digits Classification with K Nearest Neighbors (40 points)

In this question you will implement and apply the K Nearest Neighbors learning algorithm to classify handwritten digits. You should make your own implementation (rather than use libraries), but it is allowed to use library functions for vector and matrix operations. Apart from implementation of the K-NN algorithm, the question aims to improve your skills of working with vector operations in Python.

Preparation

- Download `MNIST-5-6-Subset.zip` file from Absalon.
  The file contains:
  - `MNIST-5-6-Subset.txt`
  - `MNIST-5-6-Subset-Labels.txt`
  - `MNIST-5-6-Subset-Light-Corruption.txt`
  - `MNIST-5-6-Subset-Moderate-Corruption.txt`
  - `MNIST-5-6-Subset-Heavy-Corruption.txt`
- `MNIST-5-6-Subset.txt` is a space-separated file of real numbers (written as text).[1] It contains a 784×1877 matrix, written column-by-column (the first 784 numbers in the file correspond to the first column; the next 784 numbers are the second column, and so on).
  - Each column in the matrix is a 28×28 grayscale image of a digit, stored column-by-column (the first 28 out of 784 values correspond to the first column of the 28×28 image, the next 28 values correspond to the second column, and so on). In the appendix you can find a Python script that serves as an illustration of one way to load and visualize the data.
- `MNIST-5-6-Subset-Labels.txt` is a space-separated file of 1877 integers. The numbers label the images in `MNIST-5-6-Subset.txt` file: the first number ("5") is the number drawn in the image corresponding to the first column; the second number corresponds to the second column, and so on.
- `Light-Corruption`, `Moderate-Corruption`, and `Heavy-Corruption` are corrupted versions of the digits in `MNIST-5-6-Subset.txt`, the order is preserved. It is a good idea to visualize the corrupted images to get a feeling of the corruption magnitude.

Detailed Instructions We pursue several goals in this question:

- Get your hands on implementation of K-NN and practice vector operations in Python.
- Explore fluctuations of the validation error as a function of the size of a validation set. (**Task#1**)

[1] It is a subset of digits '5' and '6' from the famous MNIST dataset (LeCun et al., 1994).

- Explore the impact of data corruption on the optimal value of K. (**Task#2**, optional)

IMPORTANT: Please, remember to include axis labels, legends and appropriate titles in your plots!

Task #1 In order to explore fluctuations of the validation error as a function of the size of the validation set, we use the following construction:

- Implement a Python function `knn(training_points, training_labels, test_points, test_labels)` that takes as input a $d \times m$ matrix of training points `training_points`, where m is the number of training points and d is the dimension of each point ($d = 784$ in the case of digits), a vector `training_labels` of the corresponding m training labels, a $d \times n$ matrix `test_points` of n test points, and their labels `test_labels` (you will need to convert the labels from $\{5, 6\}$ to $\{-1, 1\}$). The function should return a vector of length m, where each element represents the average error of K-NN on the test points for the corresponding value of K for $K \in \{1, \ldots, m\}$. **Include a printout of your implementation of the function in the report.** (Only this function, not all of your code, the complete code should be included in the `.zip` file.) Ideally, the function should have no for-loops, check the practical advice at the end of the question.

- Use the first m digits for training the K-NN model. Take $m = 50$.

- Consider five validation sets, where for $i \in \{1, \ldots, 5\}$ the set i consists of digits $m + (i-1) \times n + 1, \ldots, m + i \times n$, and where n is the size of each of the five validation sets (we will specify n in a moment). The data split is visualized below.

Training data (m points)	Validation set #1 (n points)	Validation set #2 (n points)	Validation set #3 (n points)	Validation set #4 (n points)	Validation set #5 (n points)

- Calculate the validation error for each of the sets as a function of K, for $K \in \{1, \ldots, m\}$. Plot the validation error for each of the five validation sets as a function of K in the same figure (you will get five lines in the figure).

- Execute the experiment above with $n \in \{10, 20, 40, 80\}$. You will get four figures for the four values of n, with five lines in each figure. **Include these four figures in your report.**

- Create a figure where for each $n \in \{10, 20, 40, 80\}$ you plot the variance of the validation error over the five validation sets, as a function of K. You will get four lines in this figure, one for each n. **Include this figure in your report.** (Clarification, in case you got confused: fix n and K, then you

have five numbers corresponding to validation errors on the five validation sets. You should compute the variance of these five values. Now keep n fixed, take $K \in \{1, \ldots, m\}$, and compute the variance as a function of K, i.e., compute it for each K separately. This gives you one line. And then each $n \in \{10, 20, 40, 80\}$ gives you a line, so you get four lines.)

- What can you say about fluctuations of the validation error as a function of n? **Answer in the report.**
- What can you say about the prediction accuracy of K-NN as a function of K? **Answer in the report.**
- A high-level comment: a more common way of visualizing variation of outcomes of experiment repetitions is to plot the mean and error bars, but this form of visualization makes it too easy for humans to ignore the error bars and concentrate just on the mean, see the excellent book of Kahneman (2011). The visualization you are asked to provide in this question makes it hard to ignore the variation.

Task #2 (optional, not for submission) In order to explore the influence of corruptions on the performance of K-NN and on the optimal value of K, we use this construction:

- Take the uncorrupted set, take m as before and $n = 80$, and construct training and validation sets as above. Plot five lines for the five validation sets, as a function of K, for $K \in \{1, \ldots, m\}$. **Include this figure in your report.**
- Repeat the experiment with the `Light-Corruption` set (both training and test images should be taken from the lightly corrupted set), then with the `Moderate-Corruption` set, and then with the `Heavy-Corruption` set. **Include one figure for each of the corrupted sets in your report.**
- Discuss how corruption magnitude influences the prediction accuracy of K-NN and the optimal value of K. **Answer in the report.**

Optional, not for submission: You are very welcome to experiment further with the data.

Practical Advice

We remind a few basic facts from linear algebra and advise on how to use them to write efficient Python code based on vector operations.

Basic Linear Algebra

- We use $\mathbf{x} = \begin{pmatrix} x_1 \\ \vdots \\ x_d \end{pmatrix}$ to denote a *vector* in $\mathbb{R}^d$. By default, vectors are column vectors.
- For two vectors $\mathbf{x} = \begin{pmatrix} x_1 \\ \vdots \\ x_d \end{pmatrix}$ and $\mathbf{z} = \begin{pmatrix} z_1 \\ \vdots \\ z_d \end{pmatrix}$ in $\mathbb{R}^d$, the *inner product* is defined by $\mathbf{x}^T\mathbf{z} = \sum_{i=1}^d x_i z_i$. The same quantity is also known as the *scalar product*, and the *dot product*. An alternative notation is $\langle \mathbf{x}, \mathbf{z} \rangle = \mathbf{x}^T\mathbf{z}$. Note that $\mathbf{x}^T\mathbf{z} = \mathbf{z}^T\mathbf{x}$ and that the result is a scalar (a number).
- The *outer product* is defined as $\mathbf{x}\mathbf{z}^T = \begin{pmatrix} x_1z_1 & \dots & x_1z_d \\ \vdots & \ddots & \vdots \\ x_dz_1 & \dots & x_dz_d \end{pmatrix}$. Note that the outer product is a matrix in $\mathbb{R}^{d\times d}$, and that $\mathbf{x}^T\mathbf{z} \neq \mathbf{x}\mathbf{z}^T$. Also note that $\mathbf{x}^T\mathbf{z}$ is only defined when $\mathbf{x}$ and $\mathbf{z}$ have the same dimension, whereas $\mathbf{x}\mathbf{z}^T$ is defined also when the dimensions are not the same.
- If you consider $\mathbf{x}$ and $\mathbf{z}$ as matrices in $\mathbb{R}^{d\times 1}$, then it is easy to see that the definition of the inner and the outer product follow directly from the rules of matrix multiplication.
- Python, as well as other interpreter programming languages, provide built-in precompiled functions for vector and matrix operations, making it more efficient to work with vector operations rather than explicitly loop through vectors in the code.
- The *Euclidean norm* of a vector $\mathbf{x}$, which corresponds to the length of $\mathbf{x}$, is denoted by $\|\mathbf{x}\| = \sqrt{\sum_{i=1}^d x_i^2}$. Note that the square norm satisfies $\|\mathbf{x}\|^2 = \mathbf{x}^T\mathbf{x}$.
- For two points $\mathbf{x}$ and $\mathbf{z}$ in $\mathbb{R}^d$, the *Euclidean distance* between $\mathbf{x}$ and $\mathbf{z}$ is given by the norm of the vector from $\mathbf{x}$ to $\mathbf{z}$, $\mathrm{dist}(\mathbf{x}, \mathbf{z}) = \|\mathbf{z} - \mathbf{x}\|$. The square distance can then be written as

$$\mathrm{dist}(\mathbf{x}, \mathbf{z})^2 = \|\mathbf{x} - \mathbf{z}\|^2 = (\mathbf{x} - \mathbf{z})^T(\mathbf{x} - \mathbf{z}) = \mathbf{x}^T\mathbf{x} - 2\mathbf{x}^T\mathbf{z} + \mathbf{z}^T\mathbf{z}. \quad (1)$$

Efficient Computation of Distances In order to label a target point $\mathbf{x}$, the K-NN algorithm requires sorting training points by their distance to the target point. Note that sorting the points by distance is equivalent to sorting them by the square distance, which allows to save the computation of the square root in the definition of the Euclidean norm. We show how to use the square distance representation in Equation (1) for efficient vectorized computation of distances from all training points to all target points without using any for-loops. Let $\mathbf{X}^{\texttt{train}} = \left(\begin{pmatrix} | \\ \mathbf{x}_1^{\texttt{train}} \\ | \end{pmatrix}, \cdots, \begin{pmatrix} | \\ \mathbf{x}_m^{\texttt{train}} \\ | \end{pmatrix} \right) \in \mathbb{R}^{d \times m}$ be a set of m training points and $\mathbf{X}^{\texttt{targ}} = \left(\begin{pmatrix} | \\ \mathbf{x}_1^{\texttt{targ}} \\ | \end{pmatrix}, \cdots, \begin{pmatrix} | \\ \mathbf{x}_n^{\texttt{targ}} \\ | \end{pmatrix} \right) \in \mathbb{R}^{d \times n}$ be a set of n target points, written as column vectors in the corresponding matrices. Let $\mathbf{x}_i^{\texttt{train}}$ be the i-th training point and $\mathbf{x}_j^{\texttt{targ}}$ be the j-th target point, then

$$\text{dist}(\mathbf{x}_i^{\texttt{train}}, \mathbf{x}_j^{\texttt{targ}}) = (\mathbf{x}_i^{\texttt{train}})^T \mathbf{x}_i^{\texttt{train}} - 2(\mathbf{x}_i^{\texttt{train}})^T \mathbf{x}_j^{\texttt{targ}} + (\mathbf{x}_j^{\texttt{targ}})^T \mathbf{x}_j^{\texttt{targ}}.$$

Our aim is to compute the matrix

$$D = \begin{pmatrix} \text{dist}(\mathbf{x}_1^{\texttt{train}}, \mathbf{x}_1^{\texttt{targ}}) & \dots & \text{dist}(\mathbf{x}_1^{\texttt{train}} \mathbf{x}_n^{\texttt{targ}}) \\ \vdots & \ddots & \vdots \\ \text{dist}(\mathbf{x}_m^{\texttt{train}}, \mathbf{x}_1^{\texttt{targ}}) & \dots & \text{dist}(\mathbf{x}_m^{\texttt{train}} \mathbf{x}_n^{\texttt{targ}}) \end{pmatrix} \in \mathbb{R}^{m \times n}$$

of all pairwise distances.

- For a square matrix $M = \begin{pmatrix} m_{1,1} & \dots & m_{1,d} \\ \vdots & \ddots & \vdots \\ m_{d,1} & \dots & m_{d,d} \end{pmatrix} \in \mathbb{R}^{d \times d}$ let $\text{diag}(M) = \begin{pmatrix} m_{1,1} \\ \vdots \\ m_{d,d} \end{pmatrix}$ be a column vector of the diagonal elements of M. (You can find a built-in Python function for extracting the diagonal.)
- Let $\mathbf{1}^d = \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} \in \mathbb{R}^d$ be a vector of d ones.
- Verify (convince yourself) that

  $$D = \text{diag}\left((\mathbf{X}^{\texttt{train}})^T \mathbf{X}^{\texttt{train}}\right) (\mathbf{1}^n)^T - 2(\mathbf{X}^{\texttt{train}})^T \mathbf{X}^{\texttt{targ}} + \mathbf{1}^m \left(\text{diag}\left((\mathbf{X}^{\texttt{targ}})^T \mathbf{X}^{\texttt{targ}}\right)\right)^T.$$

  It should be a good idea to visualize the relevant vectors and matrices and their products to convince yourself. Appreciate the power of linear algebra — the above expression provides distances from all the training points to all the target points in just one line without any for-loops!

Additional Guidance

- Note that for a single data point you can compute the output of K-NN for all K in one shot using vector operations. No need in for-loops! And with a bit extra effort you should be able to do it without for-loops for the whole dataset.
- You may find the following functions useful:
  - Built-in sorting functions for sorting the distances.
  - Built-in functions for computing a cumulative sum of elements of a vector `v` (for computing the predictions of K-NN for all K at once).
- It may be a good idea to debug your code with a small subset of the data.

3 Regression (50 points)

Consider the data $S = \{(x_1, y_1), \ldots, (x_N, y_n)\}$ in the file `PCB.dt`[2], which contains the concentrations of polychlorinated biphenyl (PCB) residues in lake trouts from the Cayuga Lake, NY, as reported by Bache et al. (1972). "The ages of the fish were accurately known, because the fish are annually stocked as yearlings and distinctly marked as to year class. Each whole fish was mechanically chopped, ground, and thoroughly mixed, and 5-gram samples taken. The samples were treated and PCB residues in parts per million (ppm) were estimated using column chromatography" (Bates and Watts, 1988).

Each line in `PCB.dt` is one training pattern. The first number is the input (x), the age of the fish in years, and the second is the corresponding output / target / label (y), the PCB concentration (in ppm).

Tasks

1. (5 points) Implement the linear regression algorithm as described in the lecture. For vector and matrix operations, such as computing the inverse of a matrix, you can use high-level (library) functions (e.g., from NumPy). Show the implementation of the linear regression algorithm in your report (you need not show boilerplate code, loading of the data , etc.).
2. (10 points) The task is to build a non-linear model $h : \mathbb{R} \to \mathbb{R}$ of the data in `PCB.dt`. The model should be of the form

$$h(x) = \exp(ax + b) \tag{2}$$

[2]Download from `https://github.com/christian-igel/ML/tree/main/data`

with parameters $a, b \in \mathbb{R}$.

The parameters can be learned using linear regression in the following way. Before applying linear regression, transform the output data (the y values) by applying the natural logarithm.

Then build an affine linear model using the transformed targets. By doing so, you effectively learn the (non-linear) model (2).

That is, you have to perform the following steps:

- Construct the data set $S' = \{(x_1, \ln y_1), \ldots, (x_N, \ln y_n)\}$.
- Fit a model $h'(x) = ax + b$ to S'.
- Obtain the final model as $h(x) = \exp(h'(x))$

Build the model, report the two parameters of the model as well as the mean-squared-error of the model h computed over the training data set S.

3. (10 points) In this particular example, linear regression without the non-linear transformation gives a lower error. Do not get confused by this: In the above procedure we train for a low error "on logarithmic scale", and minimizing this error may not minimize the error on the original scale, which is the error you should consider in this exercise. The idea is that the data looks more linear on logarithmic scale, this is why a model of the form (2) (a standard *allometric* equation, Huxley and Teissier, 1936) has been considered in literature.

   Provide an example showing that

$$\arg\min_{a,b} \sum_{i=1}^{N} (y_i - h(x_i))^2 \neq \arg\min_{a,b} \sum_{i=1}^{N} (\ln y_i - (ax_i + b))^2 \ . \tag{3}$$

   Note two important things: First, the transformation is non-linear. Second, we are considering a transformation of the output space that changes the error/objective function – the learning goal. In task 6, we change the input representation and compare approaches using the same objective function.

   Provide an instructive example with two data points. An easy way is to assume that the output is independent of the input and a data set $S = \{(x_1, y_1), (x_2, y_2)\}$ with $x_1 = x_2$ and $y_1 \neq y_2$.

4. (5 points) Plot the data and the model output. The plot must have proper axis labels and a legend. In *all* plots in this assignment, plot the logarithm of the PCB concentration versus the (not transformed) age.

5. (10 points) Compute the coefficient of determination R^2 as

$$1 - \frac{\sum_{i=1}^{N} (y_i - h(x_i))^2}{\sum_{i=1}^{N} (y_i - \overline{y}))^2} \ , \tag{4}$$

where $\overline{y}$ denotes the mean of the training labels. Discuss this quantity. What does it mean if R^2 is 1 and especially if R^2 is 0? Can R^2 be negative?

6. (10 points) Now let us build a non-linear model

$$h(x) = \exp(a\sqrt{x} + b) \tag{5}$$

with $a, b \in \mathbb{R}$. This is the same as before plus additionally applying the non-linear input transformation $x \mapsto \sqrt{x}$ to the input data.

One can view this as the inputs x being mapped to a feature space $\mathcal{Z}$ by the *feature map* $\phi(x) = \sqrt{x}$.

Build the model and report the mean-squared-error. Then plot the target (on logarithmic scale) and the model output *over the original inputs* (linear scale). That is, the unit of the x-axis should be years.

Compute R^2 for the new model and the transformed labels. Discuss the result in comparison to the previous model.

Deliverables: Source code (main parts also in the report); plot of data and model output; mean-squared error, parameters of regression model, discussion of R^2; counterexample showing the difference when fitting in non-linearly transformed output space; mean-squared error of model with transformed inputs, plot of data and the model given by the second model (note axis scaling in years), comparison of R^2 values of the two different models

References

C. A. Bache, J. W. Serum, W. D. Youngs, and D. J. Lisk. Polychlorinated biphenyl residues: Accumulation in cayuga lake trout with age. *Science*, 177 (4055):1191–1192, 1972.

D. M. Bates and D. G. Watts. *Nonlinear Regression Analysis and Its Applications*, volume 2. Wiley New York, 1988.

J. S. Huxley and G. Teissier. Terminology of relative growth. *Nature*, 137(3471): 780–781, 1936.

D. Kahneman. *Thinking, fast and slow.* Farrar, Straus and Giroux, New York, 2011.

Y. LeCun, C. Cortes, and C. J. C. Burges. The MNIST database of handwritten digits, 1994. MNIST was created in 1994 and released in 1998.

Appendix: Python Tips

Before you begin, ensure you have the following installed:

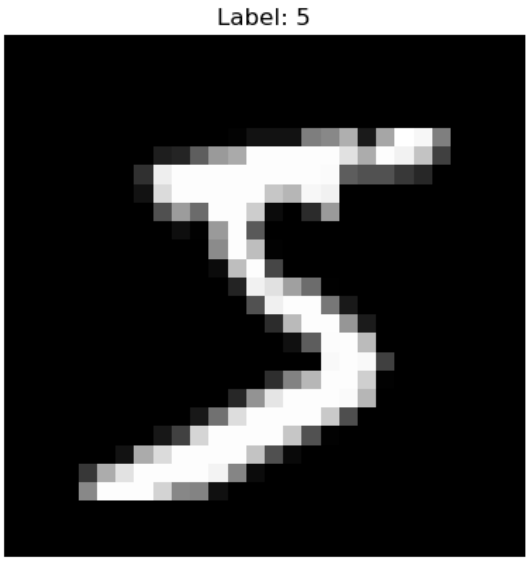
- Python (recommended version 3.x)
- NumPy library
- Matplotlib library

Loading and Visualizing MNIST-5-6-Subset Data

```
import numpy as np
import matplotlib.pyplot as plt
```

```
# Load the data from MNIST-5-6-Subset.txt
# Change the path as needed
data_file_path = "MNIST-5-6-Subset/MNIST-5-6-Subset.txt"
data_matrix = np.loadtxt(data_file_path).reshape(1877, 784)

# Load the labels from MNIST-5-6-Labels.txt
# Change the path as needed
labels_file_path = "MNIST-5-6-Subset/MNIST-5-6-Subset-Labels.txt"
labels = np.loadtxt(labels_file_path)
```

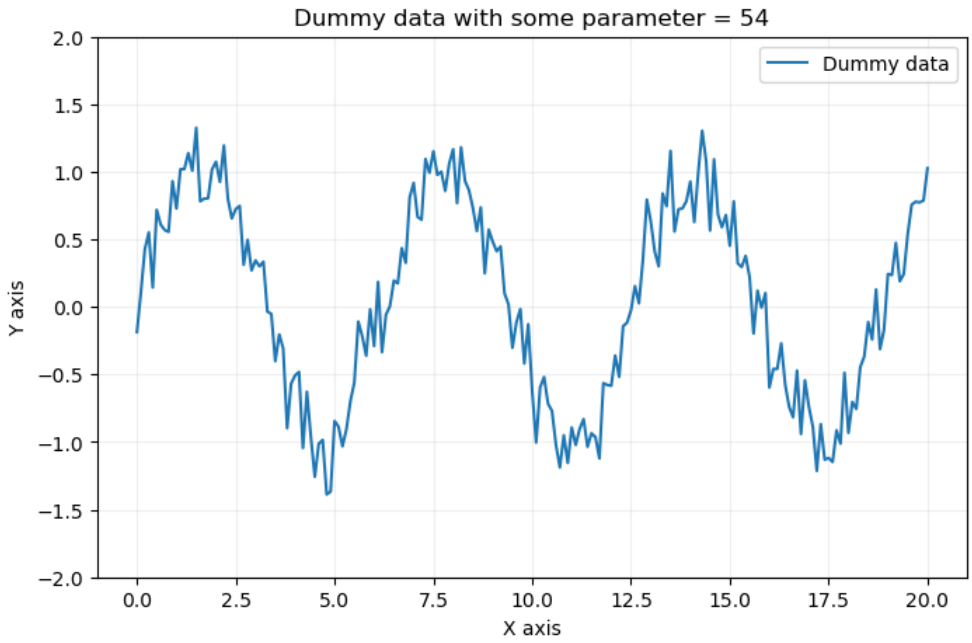
```
# Assuming you want to visualize the first image
# Change the index as needed
image_index = 0
image_data = data_matrix[image_index]
selected_label = int(labels[image_index])

# Visualize the image using Matplotlib
# We transpose the image to make the number look upright.
plt.imshow(image_data.reshape(28,28).transpose(1,0), cmap='gray')
plt.title(f"Label: {selected_label}")
plt.axis('off')  # Turn off axis
plt.show()
```



Setting up a figure with axis labels, legend and title

```
# Dummy data, x and y
x = np.arange(0, 20.1, 0.1)
y = np.sin(x) + np.random.normal(0, 0.2, len(x))
some_parameter = 54

# Initialise figure (fig) and axis (ax)
fig, ax = plt.subplots(figsize=(8,5))

# Plot in axis, add label to data
ax.plot(x, y, label='Dummy data') # (*)

# Set labels and title
ax.set_xlabel('X axis')
ax.set_ylabel('Y axis')
ax.set_title(f'Dummy data with some parameter =␣
 ↪{some_parameter}')

# Add grid
ax.grid(alpha=0.2)

# Set axes limits
```

```
ax.set_ylim(-2,2)

# Add legend (remember to label the data as shown above (*))
ax.legend()

# Show plot
plt.show()

# Save plot to some local path
fig.savefig('validation_err.png')
```



Other useful Numpy functions: `cumsum`, `sort` **and** `argsort`

```
# Creating an example array
data = np.array([5, 2, 8, 1, 6])

# 1)
# Calculating cumulative sum using cumsum
cumulative_sum = np.cumsum(data)
print("Original data:", data)
```

```
print("Cumulative sum:", cumulative_sum)
# Documentation for np.cumsum: https://numpy.org/doc/stable/
 ↪reference/generated/numpy.cumsum.html

# 2)
# Sorting the array using sort
sorted_data = np.sort(data)
print("\nOriginal data:", data)
print("Sorted data:", sorted_data)
# Documentation for np.sort: https://numpy.org/doc/stable/
 ↪reference/generated/numpy.sort.html

# 3)
# Getting indices that would sort the array using argsort
sorted_indices = np.argsort(data)
print("\nOriginal data:", data)
print("Sorted indices:", sorted_indices)
# Documentation for np.argsort: https://numpy.org/doc/stable/
 ↪reference/generated/numpy.argsort.html

# 4)
# Accessing elements in sorted order using sorted indices
sorted_data_using_indices = data[sorted_indices]
print("\nOriginal data:", data)
print("Sorted data using indices:", sorted_data_using_indices)
```

```
1)
Original data: [5 2 8 1 6]
Cumulative sum: [ 5  7 15 16 22]

2)
Original data: [5 2 8 1 6]
Sorted data: [1 2 5 6 8]

3)
Original data: [5 2 8 1 6]
Sorted indices: [3 1 0 4 2]

4)
Original data: [5 2 8 1 6]
Sorted data using indices: [1 2 5 6 8]
```