Medical Image Analysis (8DC00)

Release v0.1

Daniel Krahulec

Sep 17, 2023

SOFTWARE GUIDE

Course website: https://github.com/tueimage/8dc00-mia
Virtual reader: https://8dc00-mia-docs.readthedocs.io/en/latest/

This course is a sequel to the second year introductory imaging course. In that course the basic principles of image analysis were covered. In 8DC00 we will concentrate on the more advanced image analysis methods and how they can be used to tackle clinical problems. Topics covered include image registration and computer-aided diagnosis (CAD).

CHAPTER

ONE

INTERACTIVE NOTEBOOKS

1.1 Help for Jupyter and Python

Read the following sections carefully before you start working on the notebooks.

The first part of this notebook provides an explanation of the fundamental steps for installing your Python distributor, configuring your Python environment, and using basic Python commands. It will also give you important information on how to debug your code in case something does not work as expected. The second part describes the general setup of all notebooks, teaches you how to work with Jupyter and how to quickly and efficiently solve problems while doing the exercises.

Contents:

1.1.1 1 Python programming skills

In this course, we will be working with Anaconda (a Python distribution platform). The following instructions give an overview of essential steps prior to using Jupyter notebooks on Windows.

1.1 Python installation and configuration

Here is how to install your Python distribution platform:

1. Download and install Anaconda (it automatically comes with the latest Python version)
2. Follow the instructions in the dialog window. Make sure to check the box **Add Anaconda to my PATH environment variable** in order to be able to use Jupyter notebooks.
3. Installation will follow
4. To check whether the path to Anaconda has been added to your environment variables, go to *Edit the system environment variables* in the start menu, and click the *Environment Variables* button in the dialogue window.

1.2 Using Python terminal, setting up a Python environment

To progress efficiently in this course, you will need to install additional Python packages that are not included in the basic Anaconda Python distribution. It is recommended to install these packages in a dedicated Python environment. A Conda environment is a directory in which you can install files and packages such that their dependencies will not interact with other environments, which is very useful if you develop code for different courses or research projects. These packages can either be installed using a conda .yml file or manually using the `conda` and/or `pip` package managers. To run the complete development environment for this course, you need to install six additional Python packages: `jupyter, matplotlib, numpy, scikit-learn, scipy and spyder` (spyder is optional).

1. Open the Anaconda terminal from the Start menu on Windows
2. Create a `conda` environment: In (Anaconda) command prompt, write `conda create --name myenv` (to create an environment with a specific Python version, specify the version at the end of this command line `python=3.8`; and to add specific packages to the environment, specify them afterwards in the same command line, e.g. `conda create -n myenv python=3.8 scipy=0.15.0 numpy nibabel`). Check the requirements file for the package versions you need to install.

Here is an example you can follow for this course:

```
conda create --name 8dc00 python=3.8                          # create a new environment␣
↪called `8dc00`
conda activate 8dc00                                           # activate this environment
conda install matplotlib jupyter numpy scikit-learn scipy     # install the required␣
↪packages
```

Using both ways, the default destination folder for your newly created Python environment will be in `C:\path-to-anaconda\envs\myenv`. **Note!** You have to activate the `8dc00` environment every time you start working on the assignments (`conda activate 8dc00`).

1.3 Implementation of basic engineering and mathematical techniques

Best way to learn the basics of programming is to study Python essentials in the Essential Skills notebook of the course. Additionally, a comprehensive reference book with examples on applying mathematical models as well as machine learning in Python can be found in the book Python for Science and Engineering by Hans-Peter Halvorsen.

1.4 How to efficiently search for solutions to Python errors

Code bugs, glitches and unexpected behavior occur frequently whenever you develop code snippets, test your implementation or integrate your solution into someone else's code. What is usually time-consuming and demotivating for students, is searching for solutions to Python errors that may show an utterly confusing explanation on the screen. You copy the error text, open your browser, paste it, and a long list of sometimes completely unrelated solutions is thrown in front of you. Yes, this can be very frustrating.

The good news is that errors in Python have a very specific form, called a *traceback*. Though intimidating at times, tracebacks inform you broadly about what went wrong in your program, including indication of the line of code where the error ocurred and what type of error it was. Tracebacks may have multiple levels (reaching up to 20 levels deep!), which results in long error messages. Note however, that the length of these error statements does not reflect the severity of the problem as the messages contain all functions that were called upon before the error was encountered. You will typically find the error at the bottom of the traceback messages. Most commonly seen tracebacks include:

- `SyntaxError` (describes a "grammar" issue related to the syntax of the program)
- `IndentationError` (is related to how your code is indented)
- `NameError` (shows up when a variable definition is missing, does not exist, or its name is misspelt)
- `IndexError` (Python indexing starts at 0; this error occurs when you try to wrongly access list or array elements)
- `FileNotFoundError` (occurs when the file you aim to read is not found in the given destination on your disk)
- `IOError` (appears when you are trying to read a file that is open for writing or vice versa)

You may find examples of traceback errors on this educational website on errors and exceptions in Python.

Sometimes you are referred to the documentation pages of a certain library, where it is clearly described how to use a function, and how to fill in its mandatory input parameters. Check for example the numpy documentation to understand the structure of documenting Python libraries. Apart from documentation resources, probably the most comprehensive repository of various hacks, solutions, workarounds and tips for programmers can be found on Stack Overflow, where enthusiastic programmers post solutions to miscellaneous problems and glitches found in codes of users from all around the globe. If you still cannot find your solution, post your question on Stack Overflow, and an answer will be available for you soon.

Although Google is a friendly debugging assistant (and some programmers have learnt a programming language on a simple trial-error basis), prevention in programming is key to obtaining a functional code. General advice is to program defensively, i.e. assume errors will arise and write test code first to detect problems in an early stage. Small tests with pre- and postconditions will help you determine what the code is supposed to eventually do.

1.1.2 2. Jupyter notebook workflow

2.1 General information about notebooks

Getting started with Jupyter

We recommend using *Jupyter Notebook* to follow the exercises and run the example code (also see the Essential Skills module). An alternative is Jupyter Lab which has a bit more advanced functionality that some might find useful. It is best if you change the directory to the directory containing the code before starting Jupyter Notebook. Similarly, you can start the integrated development environment *Spyder* by typing `spyder` in the Anaconda Prompt.

To open a Jupyter notebook editor, you have several options:

1. Open Anaconda Navigator (may take some time to open), and launch Jupyter
2. Open a Windows command prompt / Windows Powershell, and type `jupyter notebook` (note the space in between); this way will only work if you have added Anaconda to your Path

Digital reader

For a quicker view of all Jupyter noteboooks you will be working on in this course, we have prepared an online reader 8DC00-website, which gives you the option to study the notebooks before opening them locally in your Jupyter environment. The reader also allows for an easy access to study all notebooks without the need for launching any interactive environment. This will especially be useful in your preparation for the final exam.

Code and data repository structure

To get started, you have to save the course's GitHub repository to your local machine (either as a ZIP archive or by running the command `git clone <link_to_repository>`), say into a folder named `8DC00` on your machine. Once downloaded or cloned, you will see the following folder and file structure:

```
8DC00
.
|____code
| |____registration.py
| |____registration_tests.py
| |____registration_util.py
| |____registration_project.py
| |____...
|____data
|____reader
| |____0.1_Software_guide.ipynb
| |____1.1_Geometrical_transformations.ipynb
| |____1.2_Point-based_registration.ipynb
| |____...
|____README.md
|____requirements.txt
```

The code for this course is organised in Python modules per topic (e.g. `registration.py`) stored in the `code` folder, each containing the Python functions (either complete or to be completed by you) particular to the topic of the exercise. These modules are referred to from the relevant Jupyter notebooks.

The testing functions (e.g. `registration_tests.py`) can be used validate the code that you developed. These functions are often already called from within the Jupyter notebooks, although some of these tests might fail if they do not yet contain completed code. Helper functions are provided in modules ending with `_util.py`.

The Jupter notebooks in the `reader` folder contain all exercise and project instructions, mostly structured according to a narrative interspersed with code snippets and example figures. The README provides the order (with links) in which the exercises can be followed.

Finally, the `data` folder contains all of the data necessary to complete the exercises and projects. Hardcoded filenames in the `_tests.py` modules are referenced to the `8DC00` folder. You might have to change these filenames if you program and run your code from outside the notebooks.

Exercises on image registration

In this set of exercises, you will first implement Python definitions for computing transformation matrices for different geometrical transformations. Then, you will implement code for converting a transformation matrix into a homogeneous form. All information needed for implementing these functions can be found in corresponding lecture slides and/or previous parts of this notebook. In the beginning, you will apply the transformations to geometric objects, however, the same functions will be later used for image transformation.

Exercises on computer-aided diagnostics and neural networks

In this set of exercises, you will implement linear regression and logistic regression methods, apply them to simplistic datasets and then evaluate and analyze the results. As with the other sets of exercises, the goal is to help you better study and understand the material and lay down the ground work for the corresponding mini-project. You should not wait to complete all exercises before moving to work on the project. For example, after completing the exercises on linear regression, you can already start with the linear regression experiments required for the project work.

Notation

Vectors and matrices are represented by a bold typeface, matrices with uppercase and vectors with lowercase letters, e.g. the matrix $\mathbf{X}$, the vector $\mathbf{w}$ etc. Compare this with the notation for scalars: X, w. In-line Python function (i.e. definition) names, commands, files and variables are represented in a highlighted monospace font, e.g. `X`, `w`, `imshow(I)`, `some_python_definition()`, `some_file.py` etc.

Activity icons

To help you understand what is expected from you in different parts of the notebooks, we have incorporated the following activity icons (top-down: STUDY, IMPLEMENT/TEST, ANSWER):

2.2 User interface and useful commands in Jupyter notebooks

Jupyter notebooks is an interactive computing environment. There is a comprehensive documentation describing the Notebook Basics, where you can learn about what happens when you first start the Jupyter notebook server and the dashboard appears in front of you. When working with the notebooks in this course, you will see the User Interface which allows you to run code, work on exercises and answer questions interactively. Instead of using the UI buttons and interactive tools, you may prefer to use keyboard commands optimized for efficient work with the notebooks. Here are a couple of most useful commands you may find useful in this course:

- Basic navigation: `Enter` (enter edit mode), `Esc` (enter command mode), `Shift-Enter` (confirm editing)
- Saving notebooks: `s` (save)
- Change cell types: `m` (markdown), `y` (code)

- Cell creation: `a` (add cell above), `b` (add cell below)
- Cell editing: `c` (copy cell), `v` (paste cell), `d, d` (delete cell), `z` (undo deletion), `x` (cut cell)

Note! It may well be that you cannot view Jupyter notebooks on the GitHub webpage correctly. Therefore, it is essential that you clone the GitHub repository to your local folder (free to choose by yourself), and work locally.

2.3 Debugging and editing your Python code directly in Jupyter notebook

Editing code directly in Jupyter, which offers no linking, auto-complete or other comforts of a decent editor, might sometimes be difficult. There are several open-source Integrated Development Environments (IDEs) enabling fast and efficient software development, code editing and debugging. Examples of these tools are PyCharm, MS Visual Studio Code or Sublime Text, to name some. On the bright side, such code editors offer miscellaneous utilities, such as auto-complete, suggestions for code enhancements, automatic installation of missing Python libraries, etc. While all these features make it much easier to develop your functionalities, setting up an IDE might be cumbersome, especially if you have never worked with any code editing software before. Eventually, these IDEs yield larger benefits when working on extensive projects that entail much more code writing, integration, and testing compared with what is necessary in this course.

While working on your notebooks, unexpected events may occur. If so, the first aid for you may be the documentation page What to do when things go wrong describing how to proceed when Jupyter fails to start, your kernel cannot be launched, a notebook does not load or does not work in a browser.

Therefore, it is essential you learn how to debug and edit your Python code directly in Jupyter notebooks (in a web browser). You can do so by making use of the so-called **magic commands**. Magic commands are IPython kernel enhancements of the normal Python code, dedicated to problem solving. An extensive list of magic commands with examples of their use can be found on the website called 28 Jupyter Notebook Tips and Tricks. Below, we will mention some of those magic commands which you will see in the Jupyter notebooks of this course.

Cell execution history

As long as your Python kernel is active, there is an input history logging the code execution of each cell. This comes in handy when you have accidentally deleted a cell.

Autoreload

The notebook typically needs to be restarted whenever you edit the code of an already imported module or package. To avoid making it tedious, we use the following two magic commands:

```
%load_ext autoreload
%autoreload 2
```

%debug and the IPython debugger

For debugging, you can use the `%debug` command. Whenever you encounter an error or exception, just open a new notebook cell, type %debug and run the cell. Then, a command line will be opened, where you can perform code testing and inspect all variables up to the line which triggered the error. Type `n` and hit `Enter` to run the next line of code (The $\rightarrow$ arrow shows you the current position). Use `c` to continue until the next breakpoint. `q` quits the debugger and code execution.

```
1 first = 5
2 second = 0
3 result = first/ 0
```

```
---------------------------------------------------------------
ZeroDivisionError                         Traceback
<ipython-input-49-b23363565227> in <module>()
      1 first = 5
      2 second = 0
----> 3 result = first/ 0

ZeroDivisionError: division by zero
```

```
1 %debug
```

```
> <ipython-input-50-a2d401806d89>(1)<module>()
----> 1 _jupyterlab_variableinspector_dict_list()

5
ipdb> first
ipdb> second
```

Another option is to make use of the IPython debugger library. Import the library as set_trace (`from IPython.core.debugger import set_trace`) and use the `set_trace()` in any code cell of your notebook to create one or more breakpoints. Executed cell will stop evaluating code at the first breakpoint and open a command line for detailed inspection. In case any of your imported modules or functions do not work, you may also deploy the debugger there.

In [*]:
```
1 from IPython.core.debugger import set_trace
2
3 first = 5
4 second = 3
5 set_trace()
6 result = first + second
```

```
ipdb> second
--Return--
None
> <ipython-input-1-3951b25a3b60>(5)<module>()
      2
      3 first = 5
      4 second = 3
----> 5 set_trace()
      6 result = first + second

3
ipdb> first
```

JupyterLab extensions

The Jupyter project is under constant development and a plethora of extensions for the user interface including more notebooks viewers have been available as JupyterLab extensions. Among the various tools JupyterLab offers, advanced debugging functionalities may come in handy. Nevertheless, these additional Jupyter API enhancers are absolutely not mandatory to install for the purpose of our course.

1.2 Topic 1.1: Geometrical transformations

This notebook combines theory with exercises to support the understanding of geometrical transformations in medical image analysis. Implement all functions in the `code` folder of your cloned repository, and test it in this notebook after implementation by importing your functions to this notebook. Use available markdown sections to fill in your answers to questions as you proceed through the notebook.

Contents:

1. Review of linear algebra
2. Introduction to medical image registration
   - Applications of registration
   - Classification of registration methods
   - Causes of medical image misalignment
3. Geometrical transformations (theory and exercises)

   3.1 Rigid transformations

   3.2 Nonrigid transformations

   3.3 Transform composition

   3.4 Homogeneous coordinates

References:

[1] Fitzpatrick, J.M., Hill, D.L. and Maurer Jr, C.R., Image registration. LINK

- Rigid transformations: Fitzpatrick, J.M., et al. Image registration, section 8.2.1
- Non-rigid transformations: Fitzpatrick, J.M., et al. Image registration, section 8.2.2

[2] Kolter, Z. Do, C. Linear algebra review and reference. LINK

[3] Maintz JB, Viergever MA. A survey of medical image registration. Med Image Anal. 1998;2(1):1–36. LINK

[1]:
```
%load_ext autoreload
%autoreload 2
```

1.2.1 1. Review of linear algebra

For animated explanation of linear algebra, you may refer to many YouTube channels, e.g. Essence of linear algebra. It is recommended that you also read chapters 1-3 of the document by Kolter, Z. Do, C. Linear algebra review and reference.

Scalars

A scalar is a single number, and can be represented by integers, real numbers, rational numbers, etc. Scalars are denoted with italic font: a, n, t.

Vectors

In mathematical terms, a vector is a 1-D array of numbers, while in physics terms, a vector is simply an arrow pointing in space, defined by its length and direction. While in physics, vectors do not have a common origin, in linear algebra, they typically start from the root of a coordinate system (e.g. x,y,z). Computer scientists define vectors as ordered lists of numbers. Vectors can be real, binary, integer, etc. An example notation for type and size of real vectors: $\mathbb{R}^n$

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \tag{1.1}$$

In Python, we can create a vector of three elements using numpy (np) arrays or the function arange.

[2]:
```
import numpy as np # importing the numpy library

x = np.array([0,1,2]) # arrays are type-set in square brackets
print(f'\nVector x is:\n {x}')

y = x.reshape(3,1) # the reshape function takes rows and columns as input arguments
print(f'\nVector y is:\n {y}. It has length of {len(y)} elements, and shape {y.shape}.')
```

```
Vector x is:
 [0 1 2]

Vector y is:
 [[0]
 [1]
 [2]]. It has length of 3 elements, and shape (3, 1).
```

Matrices

A matrix is a finite-dimensional rectangular array of numbers arranged in rows and columns. Whenever referring to a matrix element, it is common to first list row (i) before collumn (j) indices. The type and shape of matrix A can be denoted as $\boldsymbol{A} \in \mathbb{R}^{m \times n}$, where elements are arranged as follows:

$$\boldsymbol{A} = \begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix} \tag{1.2}$$

In Python, we can create a matrix using the np.matrix() function, specifying row elements in comma-separated blocks:

[3]:
```
M = np.matrix([[1,2],[3,4]])
print("\nMatrix M: \n", M)
```

```
Matrix M:
 [[1 2]
 [3 4]]
```

Matrix (Dot) Product

There are various operations that are done on matrices. Besides addition and subtraction, these operations can include multiplication by a scalar, a vector or another matrix. Scalar multiplication means that each element of a matrix is multiplied by a scalar. If we multiply an $m \times n$ matrix by a vector, then the output is a linear combination of all columns ($\boldsymbol{C} = \boldsymbol{AB}$, where $\boldsymbol{C}_{i,j} = \sum_k \boldsymbol{A}_{i,k} \boldsymbol{B}_{k,j}$). It is however important to realize that *order matters*, i.e. matrix multiplication is not commutative ($\boldsymbol{AB} \neq \boldsymbol{BA}$).

[4]:
```
# Example of matrix multiplication

A = np.matrix([[1,2,1], [0,2,1]])
B = np.matrix([[1,2,0], [0,3,1], [-2,1,1]])

# Possible multiplication
print("\nMultiplication of A*B: \n", A*B)

# Multiplication is not commutative (you can check the output error by uncommenting the␣
↪line below)
#print("\nMultiplication of B*A: \n", B*A)
```

```
Multiplication of A*B:
 [[-1  9  3]
 [-2  7  3]]
```

Matrix transpose

The transpose of a matrix can be thought of as a mirror image across the main diagonal. The first column becomes the first row, the second column becomes the second row, etc. An $n \times m$ matrix is said to be symmetric if $\mathbf{A} = \mathbf{A}^\top$, and skew symmetric if $\mathbf{A}^\top = -\mathbf{A}$. In other words, $(\boldsymbol{A}^\top)_{i,j} = \boldsymbol{A}_{i,j}$, and $(\boldsymbol{AB})^\top = \boldsymbol{B}^\top \boldsymbol{A}^\top$.

[5]:
```
# Matrix transpose is done using the transpose() function
M = np.matrix([[1,2],
               [3,4]])

print("\nMatrix M: \n", M)
print("\nMatrix M transposed: \n", M.transpose())
```

```
Matrix M:
 [[1 2]
 [3 4]]
```

(continues on next page)

(continued from previous page)

```
Matrix M transposed:
 [[1 3]
 [2 4]]
```

Identity matrix

An identity matrix of size $n \times n$ is a square matrix with ones on its main diagonal and all other elements equal to zero, i.e. $\forall \boldsymbol{x} \in \mathbb{R}^n, \boldsymbol{I}_n \boldsymbol{x} = \boldsymbol{x}$.

$$\boldsymbol{I} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{1.3}$$

[6]:
```
# Identity matrix can be created using the eye() or identity () function
N = np.eye(3)
M = np.identity(2, dtype = float) #dtype determines the data type of a variable

print("\nMatrix N with eye(): \n", N)
print("\nMatrix M with identity(): \n", M)
```

```
Matrix N with eye():
 [[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]

Matrix M with identity():
 [[1. 0.]
 [0. 1.]]
```

Matrix inversion

A matrix inverse is a matrix which when multiplied by the original matrix will yield an identity matrix. Matrix inverse is denoted as: $\boldsymbol{A}^{-1}\boldsymbol{A} = \boldsymbol{I}_n$. However, not all matrices have their inverse form. The inverse of a matrix exists only if the matrix is non-singular ($\det A \neq 0$). A system of equations can be solved using matrix inverse as follows:

$$\begin{aligned} \boldsymbol{A}\boldsymbol{x} &= b && (1.4) \\ \boldsymbol{A}^{-1}\boldsymbol{A}\boldsymbol{x} &= \boldsymbol{A}^{-1}b && (1.5) \\ \boldsymbol{I}_n\boldsymbol{x} &= \boldsymbol{A}^{-1}b && (1.6) \\ & && (1.7) \end{aligned}$$

[7]:
```
# Matrix inversion can be calculated using the numpy function linalg.inv():

# Taking a 3 * 3 matrix
A = np.array([[6, 1, 1], [4, -2, 5], [2, 8, 7]])

A_inv =  np.linalg.inv(A)
# Calculating the inverse of the matrix
print("\nThe inverse of matrix A is: \n", A_inv.round(3))
```

```
The inverse of matrix A is:
 [[ 0.176 -0.003 -0.023]
 [ 0.059 -0.131  0.085]
 [-0.118  0.15   0.052]]
```

Special matrices and vectors

There is a plethora of special matrices and vectors in linear algebra, the explanation of which is beyond the scope of this notebook. To name a couple of examples, a unit vector is a vector of length 1: $||\boldsymbol{x}||_2 = 1$; a symmetric matrix is a matrix which is equal to its transposed form: $\boldsymbol{A} = \boldsymbol{A}^\top$; and a square matrix is said to be orthogonal or orthonormal if its transpose is equal to the inverse ($\boldsymbol{A}^{-1} = \boldsymbol{A}^\top$) of that matrix: $\boldsymbol{A}^\top\boldsymbol{A} = \boldsymbol{A}\boldsymbol{A}^\top = \boldsymbol{I}$.

Systems of equations

Via the so-called augmented matrices, one can solve systems of equations. This process is not vastly different from what you normally do when solving single equations. Augmented matrices contain all equation arguments as rows. The operations on these rows consist of switching two rows, multiplication of a row by a nonzero number, and replacing a row by a multiple of another row added to it. Any row operation can be undone by another inverse row operation. Here, we show an example in Python to solve the following system of equations:

$$\begin{aligned} 8x + 3y - 2z &= 9 \\ -4x + 7y + 5z &= 15 \\ 3x + 4y - 12z &= 35 \end{aligned} \tag{1.8}$$

[8]:
```
# Solving systems of equations is easy using the numpy function linalg.solve():

A = np.array([[8, 3, -2], [-4, 7, 5], [3, 4, -12]])
b = np.array([9, 15, 35])
x = np.linalg.solve(A, b)

print("\nSolution for given system of equations [x,y,z] is: \n", x.round(2))
```

```
Solution for given system of equations [x,y,z] is:
 [-0.58  3.23 -1.99]
```

Norms

Norms are defined as functions that measure the magnitude of a matrix or vector. E.g. the distance of a vector from its origin is called a Euclidean norm, which can also be defined as the square root of the inner product of a vector with itself. The norm of a matrix expresses the magnitude of that matrix regardless of the number of its elements. Vector norms have the following three properties:

$$\begin{aligned} f(\boldsymbol{x}) = 0 &\implies \boldsymbol{x} = 0 \\ f(\boldsymbol{x} + \boldsymbol{y}) &\leq f(\boldsymbol{x}) + f(\boldsymbol{y}) \quad \text{(triangle inequality)} \\ \forall \alpha \in \mathbb{R}, f(\alpha\boldsymbol{x}) &= |\alpha| f(\boldsymbol{x}) \end{aligned} \tag{1.11}$$

(1.12)

(1.13)

(1.14)

[9]:
```
# Norms can be calculated in Python using the numpy function linalg.norm()
# This function returns one of the seven matrix norms
# or one of the infinite vector norms depending upon the value of its parameters.

# initialize vector
x = np.arange(20)

# compute norm of vector
x_norm = np.linalg.norm(x)

print("\nVector norm of x is: \n", x_norm.round(2))
```

```
Vector norm of x is:
 49.7
```

Determinant

Determinant of a matrix is a special number defined only for square matrices, representing the matrix in terms of a real number which can be used to solve systems of linear equations and finding matrix inverse. Determinant of a transformation matrix $\mathbf{T}$ is the signed area of a unit square shape after transforming with $\mathbf{T}$. The sign reflects whether the orientation has changed or not. The determinant of a 2×2 matrix is calculated as the subtraction of cross-diagonal element multiplication. It is common to use the absolute value of the determinant:

$$\det \begin{pmatrix} a & b \\ c & d \end{pmatrix} = |ad - bc| \tag{1.15}$$

[10]:
```
# The determinant of a matrix can be calculated using the numpy function linalg.det():

M = np.matrix([[1,2],[3,4]])

M_det = abs(np.linalg.det(M))

print("\nDeterminant of matrix M: \n", M_det)
```

```
Determinant of matrix M:
 2.0000000000000004
```

1.2.2 2. Introduction to (medical) image registration

Image registration is the determination of a geometrical transformation that aligns one view of an object with another view of that object or another object. The term "view" refers to a two-, three-dimensional image or the physical representation of an object in space. An example of two-dimensional image types may be x-ray projections captured as a digital radiograph or a light projection in a video frame. Three-dimensional images can be collected by imaging modalities commonly used in hospital settings, e.g. computed tomography (CT) or magnetic resonance (MR) imaging scanners. Generally, images are stored as discrete arrays of intensity values, and in medical applications, the object in each view will represent an anatomical region of interest. Explained mathematically, the inputs of registration are two views, which we map together by matching points positioned in one view to points in another view.

Applications of registration

Registration may be applied to various purposes. It allows us to combine information from different sources (MR-guided radiotherapy planning) or investigate longitudinal changes in e.g. post-treatment patient monitoring. Moreover, registration procedures are employed when studying group changes across multiple subjects in a trial. Last but not least, registration can aid in segmentation tasks when mapping atlases with anatomical model priors to a newly acquired medical image, or when performing e.g. motion-induced image artefact corrections.

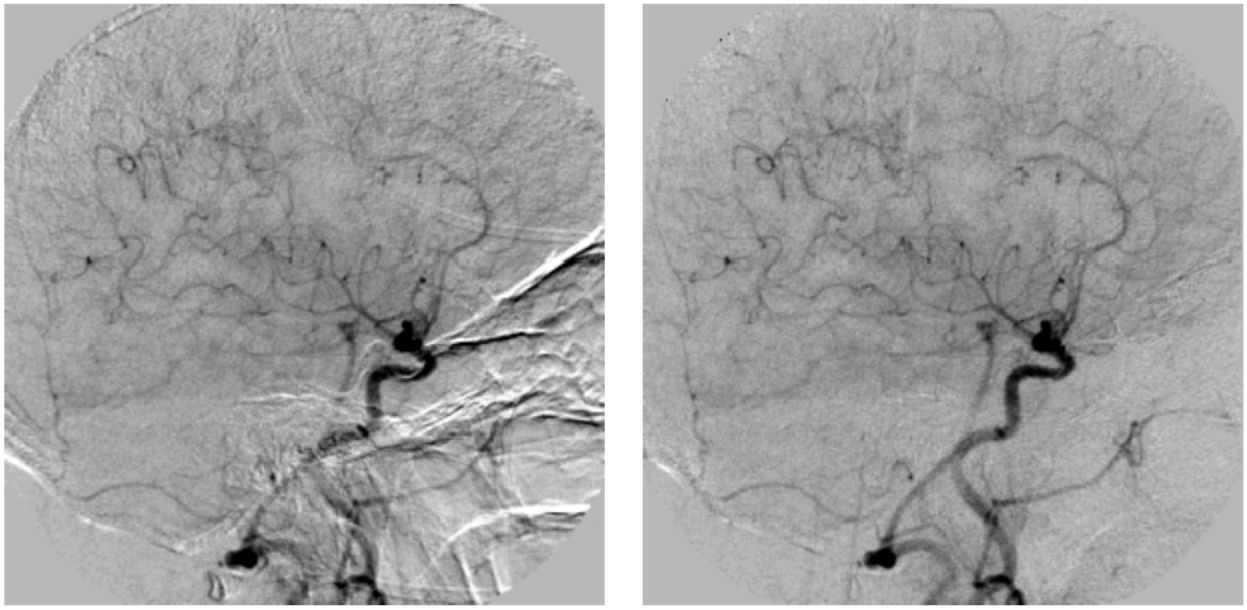
Classification of registration methods

There is a complex categorization of registration methods in the field. The following eight categories have been proposed in relevant literature (Fitzpatrick, J.M., et al. Image registration, section 8.1.2): image dimensionality, registration basis, geometrical transformation, degree of interaction, optimization procedure, modalities, subject, and object. In medical applications, we typically work with individual two-dimensional slices or three-dimensional image volumes, which may be acquired sequentially over time or as a series of multiple 3D volumes (e.g. diffusion MR images). Registration may be performed via various bases, using either a location in respective views (point-based registration) or intensity similarities (intensity-based registration). Further classification of registration methods can be based on geometrical transformations, i.e. which geometrical manipulation (rigid, affine, nonlinear, etc.) are applied for alignment between two different spaces. Registration can be either automatic or semi-automatic, depending on the amount of human interaction during the registration process. The quality of registration output is estimated continuously during the procedure either as a closed-form solution or iteratively. At last, registration methods are stratified according to the amount of modalities they involve (multi-modal, intra-modal), the subjects involved in a trial (inter-subject, intra-patient, atlas-based), and commonly also the anatomical object of interest (e.g. brain, liver, etc.).

Causes of medical image misalignment

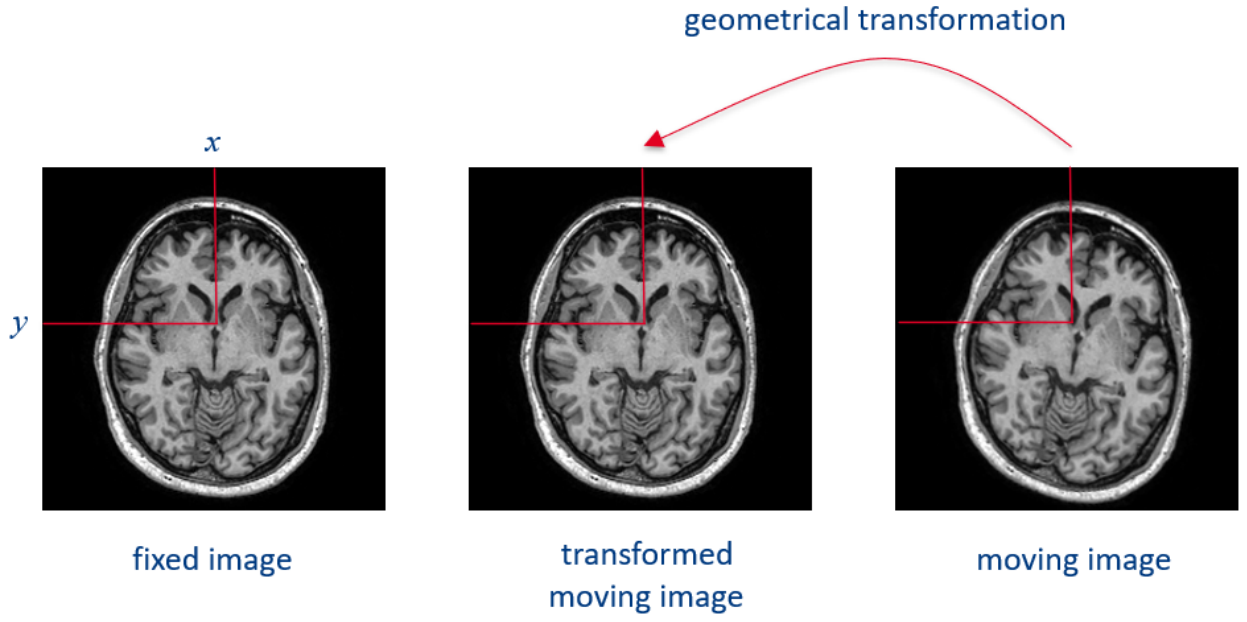
There are various reasons for misalignments across multiple images / volumes in a series or two images acquired at different time points. In measurements where patient compliance is crucial, different patient positioning, physiological movements of organs (heartbeat, breathing, cerebrospinal fluid flow), patient motion during image acquisition, and distortions caused by imaging systems (e.g. due to the design of imaging sequences in magnetic resonance imaging) can cause data misalignment. In cases where image acquisition includes interventions (e.g. surgery, chemotherapy, biopsy), users benefit from image registration as well.

An example of how digital subtraction angiography benefits from image registration:



1.2.3 3. Geometrical transformations (theory and exercises)

In registration operations, each involved view is defined by a coordinate system. As written above, registration is the act of mapping points from space X to space Y. A successful registration is achieved when the point y in space Y is approximately equal or completely correspondent to x' after a transformation T has been applied to point x in X. In the picture below, geometrical transformation T aligns the moving axial brain scan with the fixed one.



1.2.4 3.1 Rigid transformations

Rigid transformations (Fitzpatrick, J.M., et al. Image registration, section 8.2.1) are geometrical alignments of two objects that preserve distances, the planarity of surfaces and angles between straight lines. The so-called rigid registration problems involve object translation and rotation.

Translation

Translation is arguably the "simplest" geometrical transformation that can be applied to an object. Assuming that the coordinates of a 2D geometric object are stored in the variable X (the first row contains the horizontal coordinates and the second row contains the vertical coordinates), translation of the geometric object can be performed by adding a 2D translation vector X_t to every vertex of X, as shown in the Python example below.

[11]:
```
import numpy as np

# An example of translation in Python:
X = np.matrix([[1,2,3],[1,4,6]])
Xt = [4,5]

X[0,:] = X[0,:] + Xt[0]
X[1,:] = X[1,:] + Xt[1]

print("\nTranslation of the first vector X[0,:]: \n", X[0,:])
print("\nTranslation of the second vector X[1,:]: \n", X[1,:])
```

```
Translation of the first vector X[0,:]:
 [[5 6 7]]

Translation of the second vector X[1,:]:
 [[ 6  9 11]]
```

Rotation

Image rotation is a rigid transformation that requires a rotation angle θ defining the number of degrees for rotation. Typically, rotation is done about image origin (e.g. x_0, y_0).

[12]:
```
# An example of rotation in Python using numpy and scipy:
X = np.matrix([[1,2,3],[1,4,6]])

X_rotated_90 = np.rot90(X)

from scipy.ndimage import rotate
X_rotated_270 = rotate(X, angle = 270, reshape=True)

print("\nRotation by 90 degrees: \n", X_rotated_90)
print("\nRotation by 270 degrees: \n", X_rotated_270)
```

```
Rotation by 90 degrees:
 [[3 6]
 [2 4]
 [1 1]]
```

(continues on next page)

(continued from previous page)

```
Rotation by 270 degrees:
 [[1 1]
 [4 2]
 [6 3]]
```

1.2.5 3.2 Nonrigid transformations

Nonrigid transformations (Fitzpatrick, J.M., et al. Image registration, section 8.2.2) are essential in registration operations for interpatient comparisons of rigid anatomies, as well as intrapatient registration of anatomical structures with artifactual distortions induced during image acquisition of nonrigid anatomies (e.g. MRI scans of beating heart). Nonrigid transformations include scaling, where the straightness of lines and the angles between them are preserved (used e.g. to suppress calibration errors in MR scanners), and affine transformations, where the angle between lines may be changed (used e.g. in deskewing a CT image after improper gantry angle recording). Further examples of nonrigid registrations comprise projective, perspective and curved registration methods (see chapters...)

Let us leave translation aside for now and focus on the other, more complex geometrical transformations. The identity, scaling, reflection and shearing transformations (or any combination of these transformations) can be performed by multiplying the matrix of coordinates X with an appropriate transformation matrix T. Here is an example of Python code that compute transformation matrices for the identity transformation (which is not really a transformation) and scaling:

Scaling

Scaling can be performed for example in the following way: `X_scaled = scale(2,3)*X`. To verify this, we can use the provided `test_object()` function in the `registration_util.py` module that returns a test geometrical object in the shape of the letter *F*, and plot the original object and a scaled version of it (the provided `plot_object()` function in the `registration_util.py` module can be used to plot the geometrical object) as follows:

[13]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import registration as reg
import registration_util as util

def identity():
    T = np.eye(2)
    return T

def scale(sx, sy):
    T = np.array([[sx,0],[0,sy]])
    return T

X = util.test_object(1)
X_scaled = reg.scale(2, 3).dot(X)

fig = plt.figure(figsize=(5,5))
ax1  = fig.add_subplot(111)
```

(continues on next page)

(continued from previous page)

```
ax1.grid()
util.plot_object(ax1, X)
util.plot_object(ax1, X_scaled)
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
Cell In[13], line 3
      1 get_ipython().run_line_magic('matplotlib', 'inline')
      2 import matplotlib.pyplot as plt
----> 3 import registration as reg
      4 import registration_util as util
      6 def identity():

ModuleNotFoundError: No module named 'registration'
```

Reflection

Reflection is the mirror transformation of an image along a given axis.

Shearing

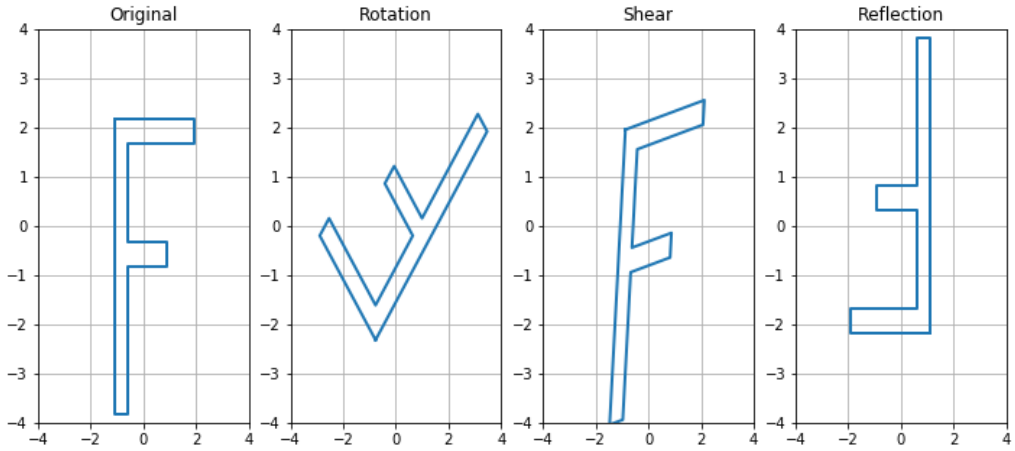
Shearing transformation (a.k.a transvection) is a type of transformation where each point is displaced by a distance proportional to the perpendicular distance from the point's parallel line. In 3D, planes are sheared instead of points.

[14]:
```
# An example of reflection in Python:
X = np.matrix([[1,2,3],[1,4,6]])

X_right_left = np.fliplr(X)
X_upside_down = np.flipud(X)

print(X_right_left)
print(X_upside_down)
```

```
[[3 2 1]
 [6 4 1]]
[[1 4 6]
 [1 2 3]]
```

Exercise 3.2.1:

Implement functions that return transformation matrices for 2D rotation, shear and reflection. You can find the templates for these three function definitions in `SECTION 1` of the `registration.py` module. To test your implementation, run the `transforms_test()` script from the `registration_tests.py` module and make sure that the output matches the figure below.



[15]:
```
%matplotlib inline
from registration_tests import transforms_test

transforms_test()
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
Cell In[15], line 2
      1 get_ipython().run_line_magic('matplotlib', 'inline')
----> 2 from registration_tests import transforms_test
      4 transforms_test()

ModuleNotFoundError: No module named 'registration_tests'
```

?

***Question 3.2.1*:**

What is rigid and what affine transformation? How many degrees of freedom dothese two types of transformations have in 2D?

Type your answer here

?

Question 3.2.2:

What is the minimum number of corresponding point pairs needed to fit a 2D affine transform? How about 3D? Motivate your answer.

Type your answer here

1.2.6 3.3 Transform composition

Geometrical transformations can be combined by multiplying transformation matrices. These compositions may involve rotation and translation, or even rotation, scaling, shearing, reflection and translation altogether. For example, the following (conceptual) command first applies a 90° rotation to an object and then a vertical reflection: `X_t = reg.reflect(-1,1).dot(reg.rotate(np.pi/2)).dot(X)`.

Question 3.3.1:

Would the result be different if the two transformations in the example above are applied in reversed order? Motivate your answer.

Type your answer here

Question 3.3.2:

How can you compute the inverse of an affine transformation represented with a transformation matrix?

Type your answer here

Exercise 3.3.1:

Test a few more examples of combining transformations. Save the examples in the `combining_transforms()` script template in the `registration_tests.py` module.

[16]:
```
%matplotlib inline
from registration_tests import combining_transforms
combining_transforms()
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
Cell In[16], line 2
      1 get_ipython().run_line_magic('matplotlib', 'inline')
----> 2 from registration_tests import combining_transforms
      3 combining_transforms()

ModuleNotFoundError: No module named 'registration_tests'
```

1.2.7 3.4 Homogeneous coordinates

As mentioned in the beginning of the previous exercise, translation can be performed by adding a translation vector to the coordinates of an object. Translation can be combined with other geometrical transformations, for example:

[17]:
```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
import registration as reg
import registration_util as util

X = util.test_object(1)

# translation vector
t = np.array([10, 20])

# rotate by 45 deg.
X_rot = reg.rotate(np.pi/4).dot(X)

# translate by 10 in the horizontal and 20 in the vertical direction
X_rot_tran = np.empty(shape=X.shape)
X_rot_tran[0,:] = X_rot[0,:] + t[0];
X_rot_tran[1,:] = X_rot[1,:] + t[1];

fig = plt.figure(figsize=(5,5))
ax1  = fig.add_subplot(111)
ax1.grid()
util.plot_object(ax1, X)
util.plot_object(ax1, X_rot_tran)
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
Cell In[17], line 4
      2 import numpy as np
      3 import matplotlib.pyplot as plt
----> 4 import registration as reg
      5 import registration_util as util
      7 X = util.test_object(1)

ModuleNotFoundError: No module named 'registration'
```

However, this way of combining translation with other transformations can be a bit cumbersome (it somewhat complicates the mathematical notation and implementation in code). The transformations that you have implemented in the previous exercise can be straightforwardly combined with translation by converting the transformation matrix to homogeneous form. This matrix can then be applied to the homogeneous coordinates (details can be found in the lecture slides). The function `c2h()` given below (also available in the `registration_util.py` module) implements conversion from Cartesian coordinates to homogeneous coordinates. As you can see from the code, this conversion is performed by simply adding an additional row of coordinates with all ones:

[18]:
```
%matplotlib inline
import registration_util as util

X = util.test_object(1)
Xh = util.c2h(X)

print('X:\n{}\n'.format(X))
print('Xh:\n{}\n'.format(Xh))
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
Cell In[18], line 2
      1 get_ipython().run_line_magic('matplotlib', 'inline')
----> 2 import registration_util as util
      4 X = util.test_object(1)
      5 Xh = util.c2h(X)

ModuleNotFoundError: No module named 'registration_util'
```

***Exercise 3.4.1*:**

Implement the function called `t2h()` in the `registration_util.py` module that converts a transformation matrix and a translation vector to a transformation matrix in homogeneous form. The template for this definition is already provided in the module file. To test your function, verify that the `t2h_test()` script results in the same object as the example above (note that the function `plot_object()` also works with homogeneous coordinates):

[19]:
```
%matplotlib inline
from registration_tests import t2h_test

t2h_test()
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
Cell In[19], line 2
      1 get_ipython().run_line_magic('matplotlib', 'inline')
----> 2 from registration_tests import t2h_test
      4 t2h_test()

ModuleNotFoundError: No module named 'registration_tests'
```

The rotation transformation rotates the objects counterclockwise around the origin of the coordinate system. To perform rotation around an arbitrary point, the following sequence of transformations must be applied:

1. Translate the object so the arbitrary rotation point is translated to the origin of the coordinate system

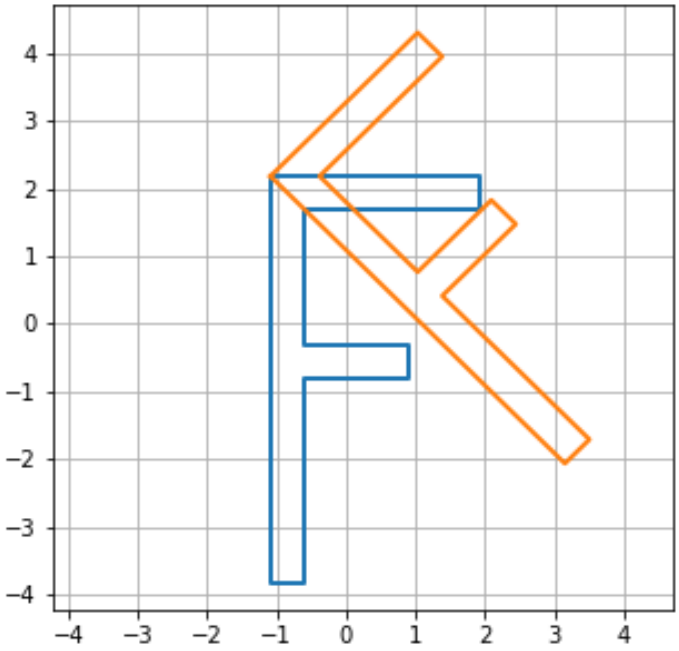
2. Rotate the object
3. Translate the object back so that the arbitrary rotation point is in the original location.

These three transformations can be combined by multiplying the three homogeneous transformation matrices. Combining transformation matrices in homogeneous form works in the same way as the "regular" transformation matrices, i.e. by matrix multiplication.

#

Exercise 3.4.2:

Write an example that rotates the test object by 45° around the first vertex (hint: the first vertex is `X[:,0]` and `t2h(reg.identity(), Xt)` is a homogeneous transformation matrix that performs only translation). Save the example in the provided `arbitrary_rotation()` template in the `registration_tests.py` module. The result should match the one shown in the figure below.



[20]:
```
%matplotlib inline

import sys
sys.path.append("../code")
from registration_tests import arbitrary_rotation

arbitrary_rotation()
```

```
Traceback (most recent call last):

  File ~/checkouts/readthedocs.org/user_builds/8dc00-mia-docs/envs/latest/lib/python3.8/
↪site-packages/IPython/core/interactiveshell.py:3508 in run_code
    exec(code_obj, self.user_global_ns, self.user_ns)

  Cell In[20], line 5
    from registration_tests import arbitrary_rotation
  File ../code/registration_tests.py:144
    def ls_affine_test():
    ^
IndentationError: expected an indented block
```

Question 3.4.1:

Assuming you have implemented the missing functionality correctly, will the following line of code result in an apparent clockwise or counter-clockwise rotation of the image? Motivate your answer. (Hint: think about the coordinate system of the image, also shown in the figures illustrating forward and inverse mapping above.)

```
It = image_transform(I, t2h(rotate(pi/4), [0 0]))
```

Type your answer here

1.3 Topic 1.2: Point-based registration

This notebook combines theory with exercises to support the understanding of point-based registration in medical image analysis. Implement all functions in the `code` folder of your cloned repository, and test it in this notebook after implementation by importing your functions to this notebook. Use available markdown sections to fill in your answers to questions as you proceed through the notebook.

Contents:

1. Point-based registration (theory)
   - *Optimization*
   - Evaluation of image registration accuracy
2. Point-based transformations (theory and exercises)

   2.1 Inverse mapping

   2.2 Least squares solution to an overdetermined system of linear equations

   2.3 Least squares fitting of an affine transformation

References:

[1] Point-based registration: Fitzpatrick, J.M., et al. Image registration, chapter 8.3

[1]:
```
%load_ext autoreload
%autoreload 2
```

1.3.1 1. Point-based registration (theory)

Image registration driven by a reliable (set of) reference point(s) on both the fixed and moving views is referred to as point-based (Fitzpatrick, J.M., et al. Image registration, chapter 8.3). Selected points in the fixed image that are considered reliable, and are typically called *fiducial points* or *fiducials*, can be part of two groups of distinguishable features: *intrinsic features* (anatomical landmarks, such as the intersection of the central sulcus with the midline of the brain), or *extrinsic features* (implanted markers, e.g. on a head coil).

Unlike in landmark-based registration, marker-based registration tends to be more precise due to its independence of anatomical structures which may sometimes be hard to discern. Fiducial points within markers may be produced via automated methods that typically assign location to the centroid of the marker.

A perfect fiducial alignment is typically impossible due to multiple reasons. There is always some fiducial localization error of the marker, and misalignments, shifts or distortions of the markers may occur relative to the voxel grid. The aim of fiducial-guided registration is to minimize the variance in the two views. Generally, the effective mean displacement will be smaller in magnitude if the marker is larger (ideally beyond the voxel size) and vice versa. Larger markers spanning more than two voxels can be more accurately localized, thereby improving the registration outcome. Reasons are three-fold:

1. When the marker is large, the fraction of partially filled voxels within it is lower
2. Erroneous shifts are cancelled out when the marker partially fills more voxels
3. Noise averaging over a larger number of voxels

Example: Assuming two images misaligned by translation, a simple registration algorithm that is efficient in this example would consist of these steps:

- Mark the location of some well discernible features in the fixed image
- Mark the corresponding location in the moving image
- Compute the translation as $t = x' - x$
- Transform the moving image by translating it with $-\mathbf{t}$

It is therefore unrealistic to look for an algorithm that will find a transformation that results in a perfect alignment of all corresponding fiducial pairs. However, we can design an algorithm that will find a transformation that results in the best possible alignment given that fact that there will always be some error. The better an algorithm aligns the two views, the lower this error becomes.

How to find the transformation that aligns fiducials at the lowest possible alignment error?

- Step 1: Write the error as a function of the transformation (affine registration)
- Step 2: Find the minimum of the error function w.r.t. the transformation

Optimization

Optimization involves finding the best parameters according to an objective function, which is either minimised or maximised. If we have a method that finds the maximum of a function, it can be easily used to find a minimum by inverting the function.

Question 1.1:

Why is full search of the parameter space not the most efficient optimization approach in 2D? Ideally, explain by example.

Type your answer here

Question 1.2:

What would be a better solution?

Type your answer here

Evaluation of image registration accuracy

Image registration can be evaluated by computing the registration error for some target corresponding point pairs. The target points should be selected in locations that are relevant for some treatment or diagnosis. Basically, this is the same as the process of selecting corresponding point pairs to compute the image transformation. The following steps should be made when evaluating registration accuracy:

1. Perform image registration (compute the transformation matrix $\mathbf{T}$)
2. Annotate some target corresponding point pairs in the fixed and moving images. These must be different from the corresponding points used to compute the transformation $\mathbf{T}$, and located at locations that are relevant for some treatment or diagnosis.
3. Transform the points from the moving image
4. Compute the target registration error as the average distance between points in the fixed image and the transformed moving points.

***Question 1.3*:**

In the registration evaluation procedure, the target corresponding points must be different from the points used to compute the image transformation. Why?

Type your answer here

1.3.2 2. Point-based transformations (theory and exercises)

The term ''image transformation" refers to the transformation of pixel spatial coordinates. Images are stored as arrays of values where each corresponds to a pixel intensity. In addition to the intensity, in medical imaging, each pixel is associated with spatial coordinates (these are in some world coordinate system where the pixel intensity value appears), and extent (the physical extent of a pixel).

In the exercises below, we assume that the pixel indices correspond to the spatial coordinates. Moreover, we assume that all images have pixels of the same size and shape (unit size isotropic). Unlike in these exercises, the concepts of physical pixel size and spatial coordinates are vital in practice.

The problem with forward mapping of the coordinates are gaps, and overlaps. These can be avoided by using inverse mapping and interpolation.

Following steps describe the inverse mapping of an image with a transformation $\mathbf{T}$:

1. Define a grid of the output image
2. Map the grid points to the input image with inverse transform $\mathbf{T}^{-1}$.
3. Determine the intensity value at those locations with image interpolation

In the previous exercises you have implemented functions for computing transformation matrices and applied them to geometric objects. In these exercises you will first write a Python function that performs image transformation by the inverse mapping method. Then, you will implement a function that performs linear least squares fitting. All necessary information for implementing these functions can be found in the lecture slides. You can use the functions that you implement in this section to perform point-based affine image registration in the project work.

2.1 Inverse mapping

Transforming an image results in transforming the locations of the image pixels. The most obvious method for transforming an image is to apply the geometric transformation to all pixel locations in an input image (the image that is being transformed) in order to determine where those pixel should be located in the output image (the transformed image) and then "fill in" the corresponding intensity values. This approach is called *forward mapping* and is illustrated in the figure below.

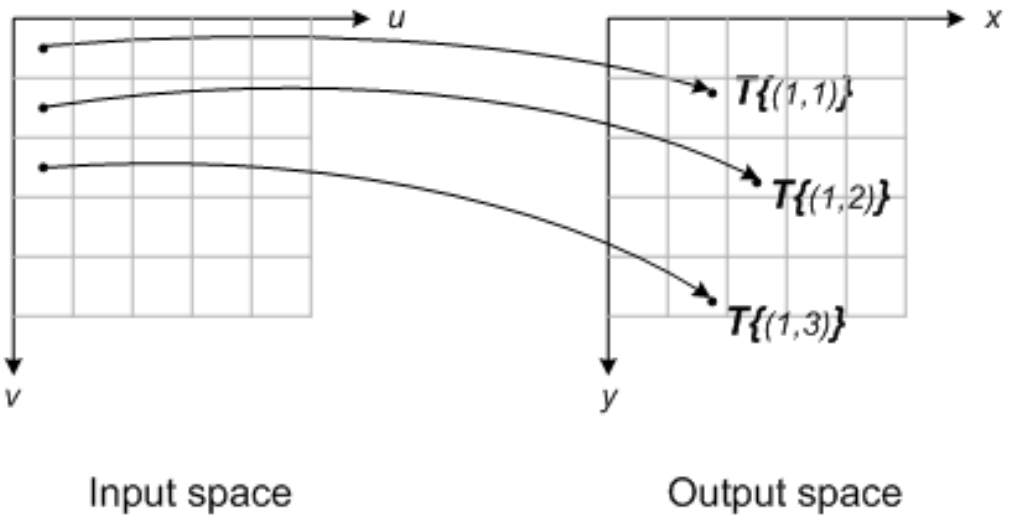


Figure from Steve on Image Processing and MATLAB

The forward mapping method can be problematic as some pixels in the output image might not "receive" a value (resulting in gaps), while some pixels might "receive" multiple values (resulting in overlaps) from the input image. These problems can be avoided by using an approach called *inverse mapping* illustrated in the figure below. Inverse mapping works by transforming the locations of the output image back to the original image by applying the inverse of the geometric transformation. The values for the pixels of the transformed image can be obtained by interpolation at the determined location in the original image. This avoids the problem of gaps and multiple values of the forward mapping method.

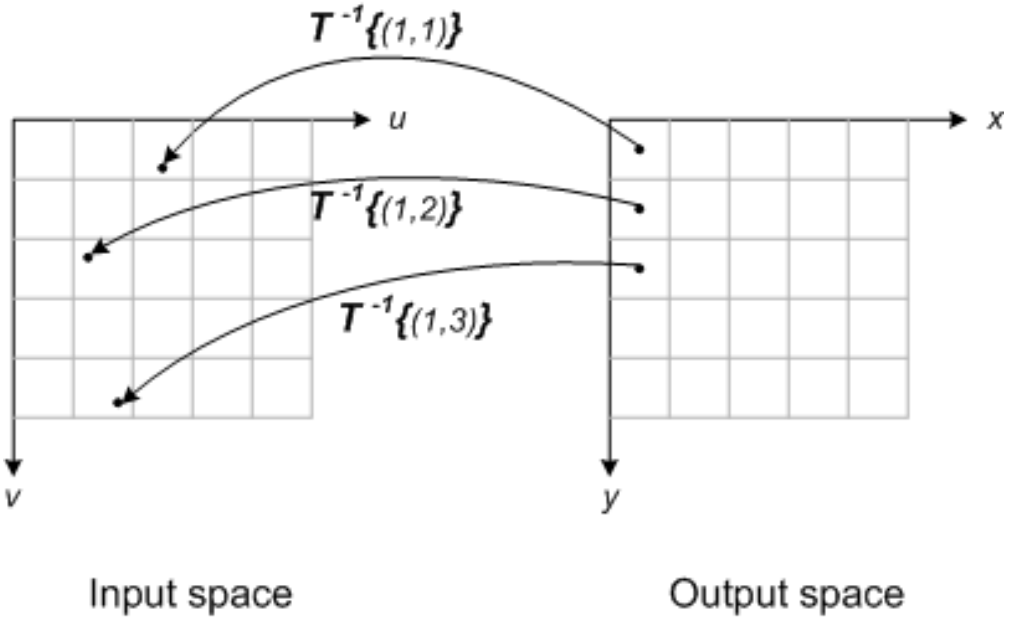


Figure from Steve on Image Processing and MATLAB

?

Question 2.1.1:

A template for implementing an image transformation function with inverse mapping is provided in the `image_transform()` function in `SECTION 2` of the `registration.py` module. Read the documentation for the `numpy.meshgrid()` function that is used in the first part of this function (you can quickly look up the documentation by clicking here. Briefly explain what the following line of code does (what are the inputs and outputs?):

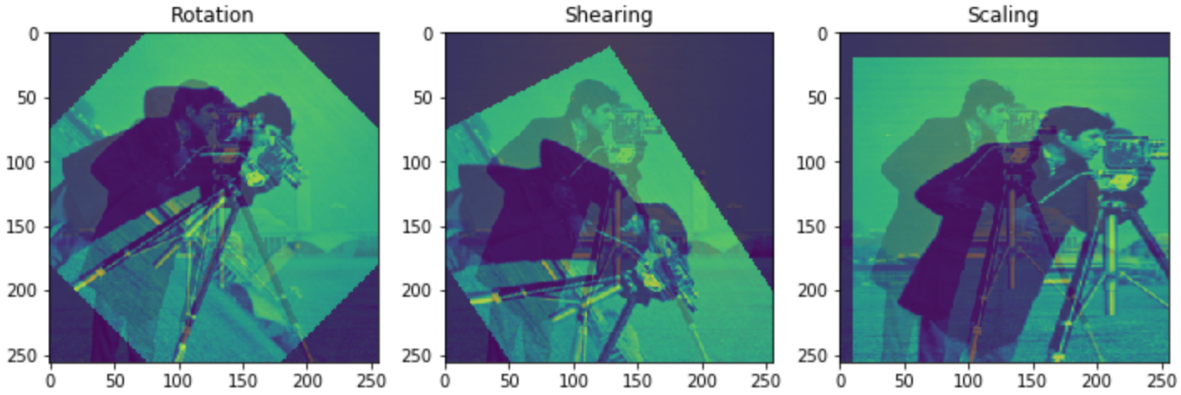
```
x = np.arange(0, output_shape[1])
y = np.arange(0, output_shape[0])
xx, yy = np.meshgrid(x, y)
```

Type your answer here

Exercise 2.1.1:

Implement the missing functionality in the `image_transform()` function. You will find the function in the `registration.py` module. It is only missing a few lines of code that performs inverse mapping of the coordinates, and tests for exceptions.

Once you have finalized your implementation, test it below. Run `image_transform_test()` from `SECTION 2` of the `registration_tests.py` module and make sure that the output matches the result in the figure below.



[2]:
```
import numpy as np
import sys
sys.path.append('../code')
sys.path.append('../data')

from registration_tests import image_transform_test
image_transform_test()
```

```
Traceback (most recent call last):

  File ~/checkouts/readthedocs.org/user_builds/8dc00-mia-docs/envs/latest/lib/python3.8/
↪site-packages/IPython/core/interactiveshell.py:3508 in run_code
    exec(code_obj, self.user_global_ns, self.user_ns)

  Cell In[2], line 6
    from registration_tests import image_transform_test
  File ../code/registration_tests.py:144
```

(continues on next page)

(continued from previous page)

```
    def ls_affine_test():
    ^
IndentationError: expected an indented block
```

2.2 Least-squares solution to an overdetermined system of linear equations

A set of linear equations can be written in matrix form in the following way:

$$\begin{cases} a_{1,1}w_1 + a_{1,2}w_2 + \ldots + a_{1,n}w_n = b_1 \\ a_{2,1}w_1 + a_{2,2}w_2 + \ldots + a_{2,n}w_n = b_2 \\ \vdots \\ a_{m,1}w_1 + a_{m,2}w_2 + \ldots + a_{m,n}w_n = b_m \end{cases}$$

$$\begin{bmatrix} a_{1,1} & a_{1,2} & \ldots & a_{1,n} \\ a_{2,1} & a_{2,2} & \ldots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \ldots & a_{m,n} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix} \tag{1.16}$$

$$\mathbf{A}\mathbf{w} = \mathbf{b} \tag{1.17}$$

where $\mathbf{w}$ is an $nx1$ column-vector of the unknown variables w_i, $\mathbf{A}$ is an mxn matrix of the known coefficients and $a_{i,j}$ and $\mathbf{b}$ is an $mx1$ column-vector of the known constant terms b_i. Solving the set of equations means finding the values of w_i that satisfy the set of equations.

- When $m < n$ the equations have no unique solution.
- When $m = n$ the equations have a unique solution.
- When $m > n$ the equations are overconstrained and there may not be an exact solution for $\mathbf{z}$. In this case, what is often consider is minimization of the squared error.

Exercise 2.2.1:

In `SECTION 2` of the `registration.py` module, `ls_solve()` contains a template for a function that finds the least squares solution for $\mathbf{w}$. Implement the missing functionality of that function.

Test your implementation by solving the following system of equations:

$$\begin{cases} 3w_1 + 4w_2 = 1 \\ 5w_1 + 6w_2 = 2 \\ 7w_1 + 8w_2 = 3 \\ 17w_1 + 10w_2 = 4 \end{cases} \tag{1.18}$$

In order to do so, you have to create the $\mathbf{A}$ matrix and the $\mathbf{B}$ vector in Python and then call the `ls_solve()` function. Implement your code in the `ls_solve_test()` script in the `registration_tests.py` module.

The found solution should be $\mathrm{w} = [0.0694, 0.2842]^\top$.

[3]:
```
%matplotlib inline
import sys
sys.path.append("../code")
from registration_tests import ls_solve_test

ls_solve_test()
```

```
Traceback (most recent call last):

  File ~/checkouts/readthedocs.org/user_builds/8dc00-mia-docs/envs/latest/lib/python3.8/
↪site-packages/IPython/core/interactiveshell.py:3508 in run_code
    exec(code_obj, self.user_global_ns, self.user_ns)

  Cell In[3], line 4
    from registration_tests import ls_solve_test
  File ../code/registration_tests.py:144
    def ls_affine_test():
    ^
IndentationError: expected an indented block
```

***Question 2.2.1*:**

For which equation does the solution result in the largest error?

Type your answer here

2.3 Least-squares fitting of an affine transformation

In point-based image registration, the goal is to find a transformation that aligns the moving with the fixed image given a set of corresponding points in the two images. In the case of affine registration, we make an assumption that the two sets of points are related trough the transformation matrix in the following way:

$$\mathbf{T}\mathbf{X}' = \mathbf{X} \tag{1.19}$$

$$\begin{bmatrix} w_1 & w_2 & w_3 \\ w_4 & w_5 & w_6 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x'_1 & x'_2 & \dots & x'_n \\ y'_1 & y'_2 & \dots & y'_n \\ 1 & 1 & \dots & 1 \end{bmatrix} = \begin{bmatrix} x_1 & x_2 & \dots & x_n \\ y_1 & y_2 & \dots & y_n \\ 1 & 1 & \dots & 1 \end{bmatrix} \tag{1.20}$$

In the previous expression, $\{x_i, y_i\}$ are the coordinates of the points in the fixed image, $\{x'_i, y'_i\}$ are the coordinates of the corresponding points in the moving image and w_i are the elements of the transformation matrix (for example, w_3 and w_6 are the translation parameters). If we transpose both sides of this equation, it will immediately become obvious that this expression defines two systems of linear equations:

$$\begin{bmatrix} x'_1 & y'_1 & 1 \\ x'_2 & y'_2 & 1 \\ \vdots & \vdots & \vdots \\ x'_m & y'_m & 1 \end{bmatrix} \begin{bmatrix} w_1 & w_4 & 0 \\ w_2 & w_5 & 0 \\ w_3 & w_6 & 1 \end{bmatrix} = \begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ \vdots & \vdots & \vdots \\ x_m & y_m & 1 \end{bmatrix} \tag{1.21}$$

The two systems of equations are:

$$\begin{bmatrix} x'_1 & y'_1 & 1 \\ x'_2 & y'_2 & 1 \\ \vdots & \vdots & \vdots \\ x'_m & y'_m & 1 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \text{ and } \begin{bmatrix} x'_1 & y'_1 & 1 \\ x'_2 & y'_2 & 1 \\ \vdots & \vdots & \vdots \\ x'_m & y'_m & 1 \end{bmatrix} \begin{bmatrix} w_4 \\ w_5 \\ w_6 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} \tag{1.22}$$

The first system gives the solution for w_1, w_2 and w_3. Similarly, the second system gives the solution for w_4, w_5 and w_6.

Exercise 2.3.1:

Implement least squares fitting of an affine transform in the provided `ls_affine()` function template in `SECTION 2` of the `registration.py` module. You have to form the $\mathbf{b}$ vector for the two systems of equations. The $\mathbf{A}$ matrix is the same for both systems and already implemented with the line $A = Xm'$. Then call `ls_solve()` to solve the two systems. Finally, you have to use the computed parameters to form a homogeneous transformation matrix (e.g. the first row of the transformation matrix will be the solution for the first linear system of equations).

Test your implementation by calling `ls_affine_test()` from `SECTION 2` of the `registration_tests.py` module. This function applies some arbitrary affine transformation to a test object, and then transforms the object back to the original with a transformation that is computed with `ls_affine()`. If your implementation is correct the retrieved object should match the original object in the displayed figure.

[4]:
```
%matplotlib inline
import sys
sys.path.append("../code")
from registration_tests import ls_affine_test

ls_affine_test()
```

```
Traceback (most recent call last):

  File ~/checkouts/readthedocs.org/user_builds/8dc00-mia-docs/envs/latest/lib/python3.8/
↪site-packages/IPython/core/interactiveshell.py:3508 in run_code
    exec(code_obj, self.user_global_ns, self.user_ns)

  Cell In[4], line 4
    from registration_tests import ls_affine_test
  File ../code/registration_tests.py:144
    def ls_affine_test():
    ^
IndentationError: expected an indented block
```

1.4 Topic 1.3: Image similarity metrics

This notebook combines theory and exercises on image similarity metrics in medical image analysis. Implement all functions in the `code` folder of your cloned repository, and test it in this notebook after implementation by importing your functions to this notebook. Use available markdown sections to fill in your answers to questions as you proceed through the notebook.

Contents:

References:

[1] Image similarity metrics: Fitzpatrick, J.M., et al. Image registration, section 8.5.1

[1]:
```
%load_ext autoreload
%autoreload 2
```

1.4.1 1. Probability theory

Random variables

Random variables map the outcomes of random phenomena to numbers. Remember the example with coin tossing in the lecture? There, we had a random variable X (outcome of the coin toss), and another random variable Y (the number of heads in a series of 3 tosses). To represent possible values and the respective probabilities of the magnitude of a random variable, we use probability distribution functions. In a similar way, we can define medical image intensities as random variables.

Probability mass function (a.k.a probability distribution table)

Random phenomenon: Pick a random pixel location. In this case, the pixel intensity can be treated as a random variable. Each outcome from the random phenomenon we are studying can be associated with a probability. If a random variable X can have a finite set of possible values, we can define a function that maps each possible value to a probability. This function is called **probability mass function** (PMF), and expresses a *discrete probability distribution*.

Probability mass function:

$p_X(x) = P(X = x)$

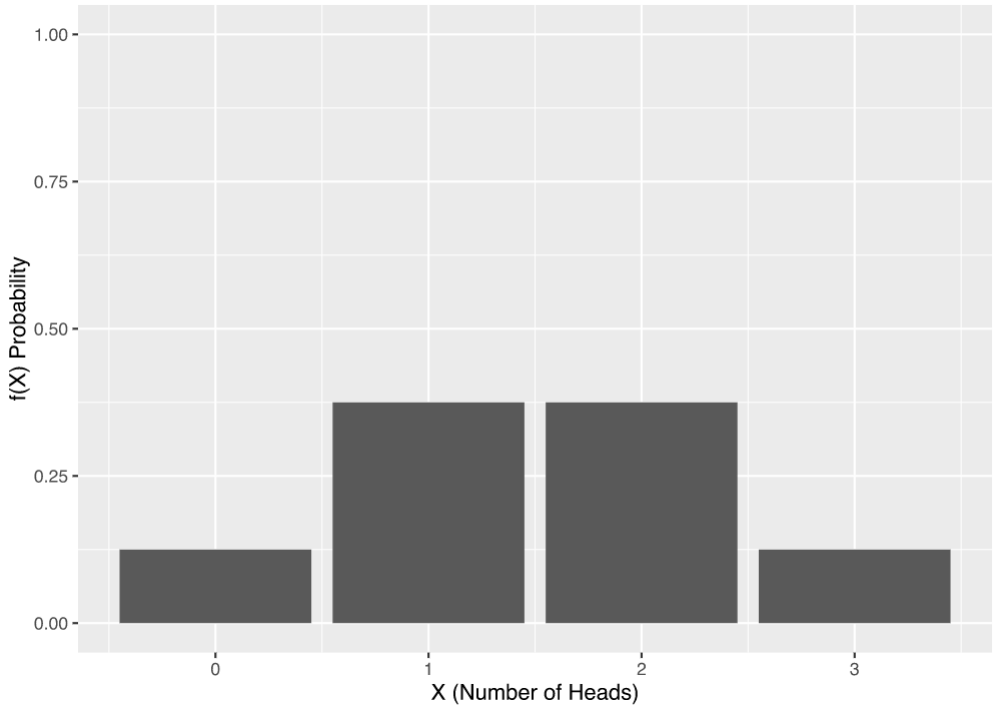


Figure from Fong Chun Chan's Blog

What if we have two random variables? For example, the pixel intensity in two images. In such case, we can define a joint probability mass function:

$p_{X,Y}(x, y) = P(X = x, Y = y)$

PMF can be used to determine the probability of an observation being exactly equal to a discrete target value. But how can we define the probability mass function for the image intensities? We can use image histogram for this purpose by counting the number of occurrences of each intensity value in the image. In order to treat the counts of the histogram as probability values, we must normalize the histogram in such a way that all values sum to 1. This is the probability mass function for the pixel intensity as a random variable.

Probability density function

Probability mass function is defined for discrete random variables. In case of continuous random variables, however, their probabilities are not directly measurable, and we therefore calculate the probability as the proportion of times. Imagine you had a random variable that measured the price of a diamond. Now, what is the probability that a single diamond's price is exactly equal to e.g. 150 USD? The probability of getting a diamond for that exact price would be very low, if any at all. Therefore, a given value of a variable on a continuous scale cannot be assigned a probability. We therefore need to think in terms of intervals instead of individual outcomes. For continuous random variables, which can take infinite number of possible values, we can define the **probability density function** (PDF), where the probability of $y \in [a, b]$ is equivalent to the integral of the PDF between a and b:

$P(a \leq y \leq b) = \int_a^b f(Y)\, dy$

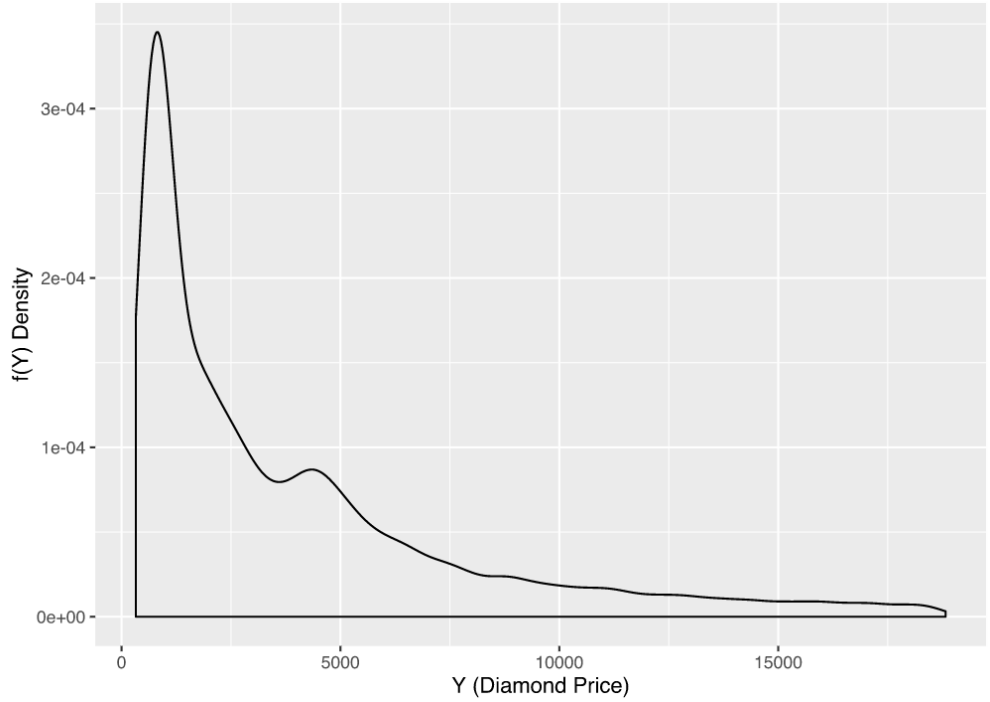


Figure from Fong Chun Chan's Blog

Bayes' rule

Bayes' rule is a very useful formula that we will use later in the computer-aided diagnosis notebooks of this course. The so-called Bayes' theorem gives the probability of an event based on new information that is, or may be related, to that event. Mathematically, the Bayes' theorem can be expressed as follow:

$p_{X|Y} = \frac{p_{Y|X}(x|y)p_Y(y)}{p_X(x)}$,

where X and Y are events and $P(Y) \neq 0$, and:

- $p_{X|Y}$ is the probability of event X occurring given event Y is true; also known as the posterior probability of X given Y
- $p_{Y|X}(x|y)$ is the likelihood of X given a fixed Y
- $p_X(x)$ and $p_Y(y)$ are the probabilities of observing the two events without any given conditions; also known as marginal or prior probabilities
- X and Y are events (must not be the same)

Bayes' theorem is typically utilized in diagnostic decision-making, e.g. to find out if there is a certain clinical manifestation in a patient before images are acquired. Given the prevalence of a disease, a radiologist is able to first estimate the marginal probability of the disease and afterwards assess medical images based on this prior. The Bayes' rule enables to derive positive predictive and negative predictive values in radiologists' pre-assessment tasks. Furthermore, this probability theorem also has its utility in cases with similar imaging findings in different diagnoses to calculate the probability at which certain imaging characteristics pertain to rare or common diagnoses (regardless of complete clinical contexts). The Bayes' theorem is also used in algorithms for medical image artefact corrections, such as in MRI and perfusion-weighted images to reduce noise. Bayesian inference has a wide range of applications in AI-driven radiology software.

1.4.2 2. Image similarity metrics

Due to the prevalence of 3D volumes in medical imaging, the term *voxel similarity measures* is typically used to jointly address these methods. In practice, algorithms perform registration between two images based on a voxel subset, which is either randomly chosen or defined by a grid. In other applications, segmentation algorithms aid registration by preselecting a subset of voxels comprising specific regions of interest. At last, similarity measures may be applied on e.g. image gradients instead of voxel values themselves. More details on image similarity metrics can be found in Fitzpatrick, J.M., et al. Image registration, section 8.5.1.

In this section, you will implement two such image similarity metrics: correlation and mutual information. The computation of the mutual information between two images relies on their joint histogram, so one of the exercises deals with the implementation of this intermediate step. In the project work section you will use the similarity metrics to find the optimal rotation transformation that aligns two images.

Before you start, load your favorite test image in Python. You're going to use this image to test your implementation. Some of the examples below assume you work with images of type `uint8`, i.e. pixel intensities in the $[0, 255]$ range, but the provided functions are equipped to work with arbitrary image types.

Question 2.1:

How can we measure the quality of a registration task? Which consideration is important when selecting target points for which we measure registration performance?

Type your answer here

2.0 Sum of square differences

Let I and J be two images and i the pixel locations. A simple and intuitive intensity-based measure of the similarity of I and J is the sum of squared differences (SSD). The SSD will be equal to zero provided that both images are correctly aligned, and will grow with increasing registration error (misalignment). If I is the fixed image in a registration problem, and J is the moving image transformed with a transformation T, the similarity measure will be a function of the transformation. It can be shown that this measure is optimal when two images differ only by Gaussian noise. This is an implicit assumption of this measure, which does not hold for inter-modality registration, and is rarely true for intra-modality registration (e.g. MRI noise is non-Gaussian due to artifacts, which leads to changes between acquisitions, etc.). Nevertheless, SSD can still be successfully used in intra-modality registration. A possible drawback of this

similarity measure is that it can be sensitive to "outlier" intensity differences. An SSD algorithm can be denoted as finding the transformation T to minimize for images I and J:

$$\mathbf{SSD} = \sum_i |I(i) - B'(i)|^2 \quad \forall\, i \in A \cap B'$$

(1.23)

2.1. Normalized cross-correlation

Another measure making slightly less assumptions is called (normalized) cross-correlation (CC). Normalized CC assumes there is a linear relationship between pixel intensities in two images, which frequently is the case for registration of images acquired with the same modality. The normalized cross-correlation between two images I and J with pixels i and respective mean intensities $\overline{I}$ and $\overline{J}$ is:

$$C = \frac{\sum_{i=1}^{n}(I(i) - \overline{I})(J(i) - \overline{J})}{\sqrt{\sum_{i=1}^{n}(I(i) - \overline{I})^2 \sum_{j=1}^{n}(J(i) - \overline{J})^2}} \tag{1.24}$$

where n is the number of image pixels. If we reshape the 2D images to vectors (in Python this can be done with `numpy.reshape()`), the expression for the normalized cross-correlation can be rewritten using vector multiplication operators (which will also make it more clear how to implement it in Python):

$$C = \frac{(\mathbf{u} - \overline{I})^{\top}(\mathbf{v} - \overline{J})}{\sqrt{(\mathbf{u} - \overline{I})^{\mathrm{T}}(\mathbf{u} - \overline{I})}\sqrt{(\mathbf{v} - \overline{J})^{\mathrm{T}}(\mathbf{v} - \overline{J})}} \tag{1.25}$$

where $\mathbf{u}$ and $\mathbf{v}$ are vectors of the pixels intensities of the images I and J, respectively:

$$u = \begin{bmatrix} I(1) \\ I(2) \\ \vdots \\ I(n) \end{bmatrix}, v = \begin{bmatrix} J(1) \\ J(2) \\ \vdots \\ J(n) \end{bmatrix} \tag{1.26}$$

Exercise 2.1.1:

The provided function `correlation()` in `SECTION 3` of the `registration.py` module contains a template for implementing the normalized cross-correlation metric. Most of the functionality such as parameter checking and preprocessing of the images is already implemented. The only piece of code that is missing is the computation of the normalized cross-correlation using the equation above.

Implement the missing functionality in the `correlation()` function. Note that the mean intensity is already subtracted from the images.

Exercise 2.1.2:

Test your implementation using the template `correlation_test()` script provided in `SECTION 3` of the `registration_tests.py` module. For example, you can make sure that `correlation(I,I)`, i.e. the normalized cross-correlation of any image with itself, returns 1. Use some other properties of normalized cross-correlation in order to further test your implementation. (**Tip**: How does a linear transformation of the intensities of the images affect the normalized cross-correlation coefficient?)

[2]:
```
%matplotlib inline
import sys
sys.path.append("../code")
from registration_tests import correlation_test

correlation_test()
```

```
Traceback (most recent call last):

  File ~/checkouts/readthedocs.org/user_builds/8dc00-mia-docs/envs/latest/lib/python3.8/
↪site-packages/IPython/core/interactiveshell.py:3508 in run_code
    exec(code_obj, self.user_global_ns, self.user_ns)

  Cell In[2], line 4
    from registration_tests import correlation_test
  File ../code/registration_tests.py:144
    def ls_affine_test():
    ^
IndentationError: expected an indented block
```

Question 2.1.1:

Under which assumptions is the normalized cross-correlation the optimal similarity metric?

Type your answer here

2.2. Joint histogram

The `joint_histogram()` function in `SECTION 3` of the `registration.py` module contains an almost complete implementation of computation of the joint histogram of two images. We use the joint histogram as an estimate of the joint probability mass function (PMF) of two images. This function informs us of the probability that two intensities co-occur (appear together) at the same location in the two images.

Exercise 2.2.1:

Go over the implementation and make sure you understand the functionality of all the steps in the code. Implement the last missing step in the computation of the joint histogram.

Question 2.2.1:

One of the parameters of the function is num bins, which defines the number of bins of the joint histogram. The default value for this parameter in this implementation is chosen to be 16. We mostly work with 8-bit images that have 256 possible values for the pixel intensities, which means that num bins can go as high as 256. However, there is a trade-off between choosing num bins too low or too high. What is this trade-off?

Type your answer here

2.3 Mutual Information

Compared to the above measures, mutual information (MI) makes very few a priori assumptions about registered objects, which is why it can be applied to larger dimensional registration and many other imaging situations.

Intuitively, MI tries to find out how much information we have about the pixel intensity at the same location in image J provided that we know the pixel intensity value at some location in the fixed image I. MI is therefore essentially a reduction in the uncertainty of Y due to the knowledge of I. Given the joint PMF of two images and the two marginal PMF's, the mutual information between the two images can be computed with the following formula:

$$MI(I,J) = \sum_{i=1}^{n}\sum_{j=1}^{n} p_{I,J}(i,j)\log\frac{p_{I,J}(i,j)}{p_I(i)p_J(j)} \tag{1.27}$$

The unit of MI depends on the particular log function: when using the natural logarithm, the unit is nats, when using base 2 logarithm, the unit is bits. In its essence, MI is a measure of the "compactness" of the joint PMF of two images. When the two images are well registered, the joint PMF is compact. When the two images are not well aligned the joint PMF is "spread out".

Remember that the joint histogram is an estimate of the joint PMF. Thus, in the previous equation, we can "plug in" the joint histogram for $p_{I,J}$, and analogously, the marginal histograms (the histograms of the individual images) for p_I and p_J. The two sum operators go over all bins in the joint histogram.

Exercise 2.3.1:

A template for implementation of mutual information given a joint histogram of two images is given in the Python function `mutual_information()` in `SECTION 3` of the `registration.py` module. As before, the file already contains all the pre-processing steps but the actual computation of the mutual information is missing. The only missing piece of code in the template file is implementation of the above formula for mutual information. Implement the missing functionality.

Exercise 2.3.2:

Use some of the properties of mutual information to test your implementation. Write these tests in the provided `mutual_information_test()` script in `SECTION 3` of the `registration_tests.py` module. (**Tip**: What would be the mutual information of two random noise images? You can generate random noise `uint8` images with `np.random.randint(255, size=(512, 512))`)

[3]:
```
%matplotlib inline
import sys
sys.path.append("../code")
from registration_tests import mutual_information_test

mutual_information_test()
```

```
Traceback (most recent call last):

  File ~/checkouts/readthedocs.org/user_builds/8dc00-mia-docs/envs/latest/lib/python3.8/
↪site-packages/IPython/core/interactiveshell.py:3508 in run_code
    exec(code_obj, self.user_global_ns, self.user_ns)

  Cell In[3], line 4
    from registration_tests import mutual_information_test
  File ../code/registration_tests.py:144
    def ls_affine_test():
    ^
IndentationError: expected an indented block
```

Exercise 2.3.3:

An alternative formula for mutual information is:

$$MI(I, J) = H(I) + H(J) - H(I, J)$$

In the previous equation, $H(I)$ and $H(J)$ is the entropy of the images I and J and $H(I, J)$ is their joint entropy.

Find out the equation for the entropy and implement mutual information, using this formula, in the `mutual_information_e()` function in `SECTION 3` of the `registration.py` module. Test your implementation with the provided `mutual_information_e_test()` script. Make sure that both implementations output equal values (very small differences are possible due to rounding errors).

[4]:
```
%matplotlib inline
import sys
sys.path.append("../code")
from registration_tests import mutual_information_e_test

mutual_information_e_test()
```

```
Traceback (most recent call last):

  File ~/checkouts/readthedocs.org/user_builds/8dc00-mia-docs/envs/latest/lib/python3.8/
↪site-packages/IPython/core/interactiveshell.py:3508 in run_code
```

(continues on next page)

(continued from previous page)

```
    exec(code_obj, self.user_global_ns, self.user_ns)

  Cell In[4], line 4
    from registration_tests import mutual_information_e_test
  File ../code/registration_tests.py:144
    def ls_affine_test():
    ^
IndentationError: expected an indented block
```

?

Question 2.3.1:

When is mutual information preferable over sum of squared errors and normalized cross-correlation as an image similarity metric? Motivate your answer.

Type your answer here

?

Question 2.3.2:

The output of `mutual_information()` is described as "*mutual information in nat units*". What change in the code would you have to make to output the mutual information in bits? Does it make a difference which units you output when you use the mutual information metric in practice (for example, to perform image registration)?

Type your answer here

1.5 Topic 1.4: Intensity-based registration

This notebook combines theory with exercises to support the understanding of intensity-based registration in medical image analysis. Implement all functions in the `code` folder of your cloned repository, and test it in this notebook after implementation by importing your functions to this notebook. Use available markdown sections to fill in your answers to questions as you proceed through the notebook.

Contents:

1. Intensity-based registration
2. Optimization for intensity-based registration
   - Gradient ascent/descent
3. Intensity-based similarity metrics (exercises)

   3.1 Numerical differentiation

   3.2 Similarity as a function of image transformation

   3.3 Similarity as a function of rotation

References:

[1] Intensity-based registration: Fitzpatrick, J.M., et al. Image registration, chapter 8.5

[1]:
```
%load_ext autoreload
%autoreload 2
```

1.5.1 1. Intensity-based registration

Besides points and surface features, image intensity is an alternative registration basis. It is even the most widely used registration basis. In general, the term *intensity* refers to scalar values of image pixels or voxels, which are used to calculate transformations between two images. Compared with point-based registration, intensity-based registration (Fitzpatrick, J.M., et al. Image registration, chapter 8.5) requires less user interaction as it works by iterative optimization of an intensity-based similarity measure (the concepts of similarity measures are explained in notebook 1.3_Registration_image-similarity-metrics). When one of the images is being transformed, the similarity measures are a function of the image transformation. This is "step 1" in our general approach to registering two images. "Step 2" is finding the parameters that find the transformation that maximizes the similarity between two images.

Intensity-based registration methods are relatively easy to automate and require few manual steps. However, their application is restricted to a limited range of images given the need for image preprocessing. Algorithms exploiting intensity-based image registration can be used for various purposes: registration of images with different dimensionality; intermodal and intramodal registration; and registration involving complex transforms, to name some.

1.5.2 2. Optimization for intensity-based registration

General procedure for maximizing similarity functions is:

1. Start with some initial values for the parameters (e.g. transformation T).
2. Slightly update the parameters in such a way that the similarity will slightly increase.
3. Repeat until the similarity *stops increasing*.

Gradient ascent / descent:

To optimize similarity functions in intensity-based registration, we typically use gradient ascent (to localize function maximum) or gradient descent (to localize function minimum). In other words, these numerical methods help us find the minimum of the error or the maximum of the similarity in registration. To find the minimum and maximum of a function, we can compute the derivative and set it to zero (in case of more variables, set all partial derivatives to zero).

Gradient ascent algorithm for maximizing a function $f(\mathbf{w})$:

1. Choose some initial values of the parameters $\mathbf{w}$
2. Calculate the value for the gradient of $f(\mathbf{w})$ for the current parameters
3. Update the parameters in the direction of the gradient: $\mathbf{w} \leftarrow \mathbf{w}+\mu\nabla_{\mathbf{w}} f(\mathbf{w})$

If we want to minimize the function we move in the direction opposite of the gradient (gradient descent): $\mathbf{w} \leftarrow \mathbf{w} - \mu \nabla_{\mathbf{w}} f(\mathbf{w})$

The parameter μ is called learning rate. It controls how fast we move towards the maximum (minimum). If μ is too small, the maximum (or minimum) might not be reached in reasonable time. If μ is too large, the maximum (minimum) might be missed. Initialization is important. Different starting points will result in different found maxima (and not always global).

1.5.3 3. Intensity-based image registration (exercises)

3.1 Numerical differentiation

Numerical differentiation refers to finding the value of a derivative of a given function at a given point without the need to analytically differentiate the function. This technique can be very useful, for example, when the analytical expression for the derivative is too complex and computationally expensive to evaluate. In such a case it might be significantly faster to approximate the derivative instead of computing its exact value.

A simple expression that approximates the derivative of a function $f(x)$ is:

$$\frac{d}{dx} f(x) \approx \frac{f(x+h) - f(x)}{h} \tag{1.28}$$

where h is some very small positive number. When h approaches zero this expression becomes the true value of the derivative:

$$\frac{d}{dx} f(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h} \tag{1.29}$$

A better approximation of the derivative is the symmetric difference quotient given by the following expression:

$$\frac{d}{dx} f(x) \approx \frac{f\left(x + \frac{h}{2}\right) - f\left(x - \frac{h}{2}\right)}{h} \tag{1.30}$$

Numerical differentiation can also be used to approximate the partial derivatives of a function with more than one variable, for example:

$$\frac{\partial}{\partial x} f(x,y) \approx \frac{f\left(x + \frac{h}{2}, y\right) - f\left(x - \frac{h}{2}, y\right)}{h} \tag{1.31}$$

$$\frac{\partial}{\partial y} f(x,y) \approx \frac{f\left(x, y + \frac{h}{2}\right) - f\left(x, y - \frac{h}{2}\right)}{h} \tag{1.32}$$

and in turn the gradient:

$$f(x,y) = \begin{bmatrix} \frac{\partial}{\partial x} f(x,y) \\ \frac{\partial}{\partial y} f(x,y) \end{bmatrix} \approx \begin{bmatrix} \frac{f\left(x+\frac{h}{2},y\right) - f\left(x-\frac{h}{2},y\right)}{h} \\ \frac{f\left(x,y+\frac{h}{2}\right) - f\left(x,y-\frac{h}{2}\right)}{h} \end{bmatrix} \tag{1.33}$$

Exercise 3.1.1:

In the provided template for the `ngradient()` function in `SECTION 4` of the `registration.py` module, implement the computation of the gradient of a function with numerical differentiation using the symmetric difference quotient.

Exercise 3.1.2:

Test your implementation of `ngradient()`. An easy way to test this function is to numerically compute the gradient and then verify with the analytical expression. For example, since $\frac{d}{dx}e^x = e^x$ the the numerical derivative $\frac{d}{dx}e^x$ should have approximately the same value as e^x. Write your test cases in the provided `ngradient_test()` script in `SECTION 4` of the `registration_tests.py` module.

[2]:
```
%matplotlib inline
import sys
sys.path.append("../code")
from registration_tests import ngradient_test

ngradient_test()
```

```
Traceback (most recent call last):

  File ~/checkouts/readthedocs.org/user_builds/8dc00-mia-docs/envs/latest/lib/python3.8/
↪site-packages/IPython/core/interactiveshell.py:3508 in run_code
    exec(code_obj, self.user_global_ns, self.user_ns)

  Cell In[2], line 4
    from registration_tests import ngradient_test
  File ../code/registration_tests.py:144
    def ls_affine_test():
    ^
IndentationError: expected an indented block
```

Question 3.1.1:

The `ndgradient()` function can be used to perform optimization with the gradient ascent/descent method. Describe in short how this algorithm works. What is the role of the learning rate parameter in gradient descent/ascent?

Type your answer here

3.2 Similarity as a function of image transformation

In the previous section, you have analyzed how the similarity between two images changes as a function of the rotation of one of the images. The goal of this exercise is to write a Python function that, given two images and the parameters of some transformation between them, will output the similarity measure. This function can then be used in combination with `ndgradient()` from the previous exercise to perform gradient based optimization of the transformation parameters.

The function `rigid_corr()` in `SECTION 4` of the `registration.py` module computes the normalized cross-correlation between a fixed and a moving image transformed with rigid transformation. The three parameters of the rigid transformation (rotation angle and 2D translation vector) are passed to the function as a vector `x`.

Here is an example of how to use this function to numerically compute the derivative for a set of parameters:

```
import numpy as np
import matplotlib.pyplot as plt
from registration_utils import ngradient

I = plt.imread('some_fixed_image.tif')
Im = plt.imread('some_moving_image.tif')

# create an instance of rigid_corr for this particular pair of images
rigid_corr_I_Im = lambda x: rigid_corr(I, Im, x)

x = [np.pi/4, 10/100, 20/100]

# computes the numerical gradient at x
g = reg.ndgradient(rigid_corr_I_Im, x)
```

In this code snippet, we first create an instance of the function `rigid_corr()` where the first to input parameters (the fixed and moving image) are preset. The new function `rigid_corr_I_Im()` now has only a single input parameter - the vector `x` that stores the rotation angle and the translation. `rigid_corr_I_Im()` can be used with `ndgradient()` to compute the gradient of the similarity function at a particular point (in this example for the point `x = [pi/4, 10/100, 20/100]`).

Question 3.2.1:

Let's assume that after executing this code snippet, the computed value for the derivative at point `x = [pi/4, 10, 20]` is `g = [10, -5, 30]`. Will increasing the rotation angle (the first parameter of `x`) by a very small amount increase or decrease the similarity between the fixed and transformed moving image? Motivate your answer.

Type your answer here

Exercise 3.2.1:

Using `rigid_corr()` as an example, implement the following two functions in `SECTION 4` of the `registration.py` module:

1. `affine_corr()` that computes the normalized cross correlation for a pair of images as a function of affine transformation, and
2. `affine_mi()` that computes the mutual information between a pair of images as a function of affine transformation.

The only thing that you need to change is the length of the parameter vector, which for affine registration should contain the rotation, scaling, shearing and translation parameters, the computation of the transformation matrix and for `affine_mi()` the function call that computes the similarity measure.

3.3 Similarity as a function of rotation

Let's put the implementations of correlation and mutual information functions to some use. You are going to compute the similarity between an image and a rotated version of that image for different rotation angles. The `registration_metrics_demo()` Python function contains code for performing this analysis. Study the function and make sure you understand what it does (you can skip the part about visualization of the results).

Exercise 3.3.1:

Run the demo and describe and analyze the results.

[3]:
```
%matplotlib inline
import sys
sys.path.append("../code")
from registration_tests import registration_metrics_demo

registration_metrics_demo()
```

```
Traceback (most recent call last):

  File ~/checkouts/readthedocs.org/user_builds/8dc00-mia-docs/envs/latest/lib/python3.8/
↪site-packages/IPython/core/interactiveshell.py:3508 in run_code
    exec(code_obj, self.user_global_ns, self.user_ns)

  Cell In[3], line 4
    from registration_tests import registration_metrics_demo
  File ../code/registration_tests.py:144
    def ls_affine_test():
    ^
IndentationError: expected an indented block
```

***Question 3.3.1*:**

Run the demo again but this time compute the similarity of the T1w image with a rotated version of the T2w image for different angles (note that the T1w and T2w images in this example are registered). Describe and analyze the results. Would the normalized cross-correlation metric be suitable to register the T1w and T2w images? Which of the two analyzed metrics would be more appropriate? Motivate your answer.

Type your answer here

1.6 Topic 1.5: Validation in medical image analysis

This notebook combines theory with questions to support the understanding of validation metrics in medical image analysis. Use available markdown sections to fill in your answers to questions as you proceed through the notebook.

Contents:

1. Validation (concepts)

   1.1 Quality characteristics

   - Accuracy (bias)
   - Precision (variation), reproducibility, reliability, replicability
   - *Robustness*
   - *Efficiency*
   - Fault detection

   1.2 Ground truth

   - Ground truth from real data
   - Ground truth from phantoms
   - Data representativeness
   - Statistical significance

   1.3 Measures of quality

   - Segmentation - quality measures
   - Registration - quality measures
   - (Computer-aided) detection - quality measures

2. Common limitations of performance metrics in biomedical image analysis
   - Small structures
   - Image artifacts
   - Overlap measurements
   - Over- and undersegmentation
   - Single-object bias
   - Metric combination
   - Choosing the right metric for a given task

References:

[1] Measures of quality: Toennies Klaus, D. Guide to Medical Image Analysis - Methods and Algorithms, Chapter 13.1

[2] Ground truth: Toennies Klaus, D. Guide to Medical Image Analysis - Methods and Algorithms, Chapter 13.2

[3] Limitations of performance metrics: Reinke et al. Common Limitations of Image Processing Metrics: A Picture Story.

[4] Assessment of registration errors: Fitzpatrick, M. Visualization, Image-Guided Procedures and Modeling, 7261:1–12, SPIE Medical Imaging (2009).

[1]:
```
%load_ext autoreload
%autoreload 2
```

1.6.1 1. Validation (concepts)

Validation of medical image analysis methods is the estimation of correctness of certain results from tests of a method on a representative sample set. In e.g. software design, validation is the evaluation of the degree to which user needs (performance requirements) are met, i.e. whether the right software is being built. In medical image analysis, we usually talk about technical validation, where the aim is to evaluate the performance of computing algorithms with respect to e.g. segmentation accuracy. Validation is used in various image computing classes (registration, segmentation, detection, classification, quantification).

Prior to performing validation, suitable data needs to be selected, comparison measures need to be chosen, and a norm (e.g. ground-truth, explained below) needs to be defined. **Remember that every validation study must have a hypothesis on performance (e.g. outcome is better than…), and a ground truth (gold standard) is essential**. Validation can provide information about our method with respect to another method used to generate the same results (cross-validation). It is mandatory to document a detailed description of the validation procedure together with a well-founded justification of selected measures, as it allows potential new users of the method to investigate the validity of the arguments used to build the validation scenario.

1.1 Quality characteristics

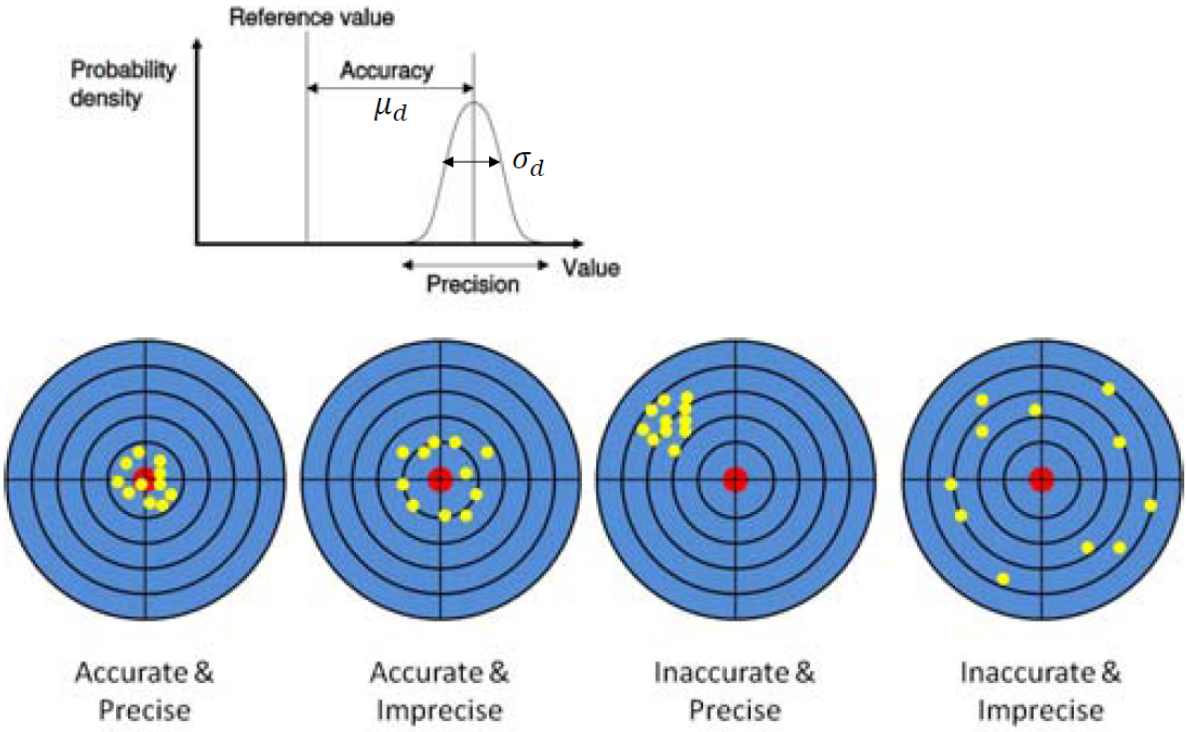
There are a number of quality characteristics used in validation of medical image analysis methods:

Accuracy (bias)

Accuracy determines the deviation of results from known ground truth. It is computed via a measure of quality (section 1.3) comparing results with some norm. Accuracy is calculated as the ratio between true/false positives and negatives: $\mathrm{A} = \frac{(\mathrm{TP}+\mathrm{TN})}{(\mathrm{TP}+\mathrm{FP}+\mathrm{FN}+\mathrm{TN})}$.

Precision (variation), reproducibility, reliability, replicability

These characteristics measure the extent to which equal or similar input produces equal or similar results. Reliable methods produce output within a given range of variation (e.g. in terms appearance). A method is reproducible, if two runs of this method with the exact same input and setup produce the exact same results. Replicability of a method can be determined if two runs of a method with the same input and same setup arrive at similar conclusions.



Robustness

Robustness of a method characterizes the change of the quality of an analysis result if conditions deviate from assumptions made for analysis (e.g., when noise level increases or if object appearance deviates from prior assumptions). For example, robustness of a segmentation algorithm is the ability of an algorithm to persist in sufficient performance despite abnormalities in the input images (e.g. due to patient motion). Reproducibility and robustness are also important performance indicators in case of varying image data (different scanners, hospitals, patient populations, etc.).

Efficiency

The effort which must be exerted to achieve an analysis result is described by efficiency. You may recall that there are semi-automated methods that require some degree of human interaction or expert knowledge. These factors contribute to the overall determination of a method's efficiency.

Fault detection

The ability to discover possible faults while an analysis method is being applied is called fault detection. It is a very useful feature, because it requires the method to test for reliability of its own results.

1.2 Ground truth

You may remember one of the lecture slides with the following statement: *"In medical image analysis, the truth is difficult to come by, since the reason for producing images in the first place was to gather information about the human body that cannot be accessed otherwise."*.

Ground truth is a conceptual term relative to the knowledge of the truth concerning a specific question (the "ideal expected result"). In validation, all measures of quality estimation for an analysis method require comparison of the method's produced results with the true information. Ground truth data can be either real or artificial, however, it is never completely certain whether selected data are representative of the desired ground truth. See also chapter 13.2 of the Guide to Medical Image Analysis by Tonnies, Klaus D

Ground truth from real data

Ground truth based on real data can be created by applying the currently established best method to it if such method exists at all. An example is the use of mutual information and spline-based non-rigid registration for registering MR brain images. An often encountered problem is proving that the conditions under which a standard is applied, are comparable with those conditions under which they are considered to be an established standard. Moreover, the implementation of the established methods is rarely available, even though these days, more implementations become open-source or integrated in widely used freely downloadable software packages.

If an established method is missing, human experts may help produce ground truth data through *manual data annotation*. This approach requires a lot of effort both from the method's developer, as well as the expert who has to carry out the analysis on several datasets, document findings, and sometimes it is desirable to have the expert analyze the data(sets) multiple times (intra-observer variability) to increase the significance of the results. The developer must provide a sufficiently good user interface for the expert to avoid bias by the input component quality. Sometimes it may be more beneficial to ask more experts and measure (inter-observer) variability. In such case, it is crucial to define what is meant by agreement among all (e.g. agreement by all / the majority of observers, etc.).

An algorithm for the validation of image segmentation that estimates reference standard based on a set of segmentations is called STAPLE (Simultaneous Truth and Performance Level Estimation).

Ground truth from phantoms

Phantoms can be used as ground truth as well. They are classified as follows:

Based on real data

- cadaver phantoms (human or animal)
- artificial hardware phantoms (e.g. CT and MRI slices generated in the Visible Human Project)

Based on simulated data

- software phantoms representing the reconstructed image or the imaged measurement distribution
- mathematical simulations (e.g. Shepp-logan phantom)

Phantoms are characteristic for specific properties (material, measurement properties, influences from image reconstruction, shape properties), according to which they are applied in different tasks. Phantoms are only useful in validation analyses when results have been generated in them. For a detection task, a couple of locations must be specified, and for registration tasks, fiducial markers have to be implanted, for example. Material and measurement properties are often idealized. Image artefacts are typically simulated, e.g. by using zero-mean Gaussian noise to simulate detector noise; smoothing data to evoke partial volume effects or through inclusion of artificial shading to model signal fluctuations.

The advantage of a software phantom is that it is more straightforward to account for anatomical variation by creating several phantoms with different shapes, unlike in hardware phantoms, where anatomical variation can hardly be modelled. Examples of software phantoms include the BrainWeb phantom; the Field II ultrasound simulation program; the XCAT phantom; or the dynamic MCAT heart phantom simulating a moving heart.

Data representativeness

To make a (ground truth) dataset representative, all data properties that may have an impact on the performance of an analysis method should be reflected in it. Representativeness can be enforced by:

- separation between test and training data (leave-one-out technique in classification tasks); if optimal parameters have to be determined for a method, it is unacceptable to validate the results on ground-truth data which has been used to arrive at the optimal parameter value
- identification of sources of variation (all should be covered by the ground truth data) and outlier detection (experts can help)
- robustness with respect to parameter variation (e.g. changes in input thresholds)

Statistical significance

While your analysis results may seem satisfactory, there is a chance that they are statistically insignificant due to low number of samples in your validation set. Significance of an experimental outcome can be indicated by the well-known p-value. For example, the probability of less than 1% that a result arose by chance would be expressed as $p < 0.01$. Significance can be calculated via the *Student's t-test*, which helps you find out if there is a statistical difference between two compared groups.

1.3 Measures of quality

Quality is determined by the kind of analysis which has been conducted on a dataset:

Task	Quality measure
Segmentation	Correspondence between the segmented object and a reference segmentation
Registration	Deviation from the correct registration transformation
Computer-aided detection (CAD)	Ratio between correct and incorrect decisions

See also chapter 13.1 of the Guide to Medical Image Analysis by Tonnies, Klaus D

Segmentation - quality measures

When segmenting an object in an image, a measure of comparison between some reference g (usually a ground truth) and the segmented object f is required. Mutual correspondence may be determined by calculating volumetric overlap, overlap between object and background or performing distance measurements (of boundary deviations). In 3D cases, volumetric measurements aim to count the number of voxels in both the segmented object and the reference norm weighted by the volume covered by each voxel.

Overlaps between objects f and g can be calculated by measures that count over-segmentation (number of elements) and under-segmentation.

The next measures often used for quality assessment are *Dice similarity coefficient* (DSC) a.k.a *Sørensen–Dice coefficient* (d), *Intersection over union* (i) and the *Jaccard index* (j):

$$d = \frac{2|F \cap G|}{|F| + |G|} \ , \tag{1.34}$$

$$i = \frac{\mathrm{DSC}}{2 - \mathrm{DSC}} \ , \text{and} \tag{1.35}$$

$$j = \frac{|F \cap G|}{|F \cup G|} \ , \tag{1.36}$$

where $F \cap G$ is the size of elements (voxels) in overlap, and $|F|$, $|G|$ are the sizes of individual volumes. The coefficient is equal to 1 in case of perfect correspondence; otherwise it is smaller than 1. In the medical image analysis community, the Dice coefficient is more popular, and therefore also more often present in literature.

Neither Dice nor Jaccard indices can be used to measure outliers (e.g. in tasks where organ boundaries are to be segmented as part of access planning in surgery). In minimally invasive procedures, it is crucial to determine the deviation of the segmented boundary from the true boundary. This can be done by *Hausdorff distance* (HD) between F and G. The Hausdorff distance is defined as the maximum of all shortest distances d between points in F and G. Since this measure is highly sensitive to image artefacts, the quantile Hausdorff distance is used, where distances of largest outliers are averaged. It is computed from a quantile of a histogram of distances from F to G and from G to F:

$h^q = \max(t_q(d(f, G)), t_q(d(g, F)))$(1.37)

Registration - quality measures

Registration aims to find a geometric transformation that maps an n-dimensional image onto another one, bringing both images into alignment. In case of different dimensionalities of the registered objects, the transformation includes a projection step of the scene from higher dimension to the scene of lower dimension. The steps to evaluate registration accuracy when working with point-based registration are explained in *section 1 of notebook 1.2*.

The quality of a registration method can be measured as the average deviation of known transformation parameters based on comparisons between vector fields (for non-rigid registration) or differences in global rotation and translation (for rigid transformation). Another way of assessing quality for a registration task is to compute locations of fiducials after registration, however, **one should never use the same corresponding point pairs/fiducials or image similarity metric were used for optimization when computing the registration transformation!**

We use Fiducial Localization Error (FLE), Fiducial Registration Error (FRE) and Target Registration Error (TRE) to evaluate registration accuracy:

- FLE quantifies the error in determining the location of a point which is used to estimate the registration transformation.
- FRE is the error of the fiducial markers following registration, i.e. $||T(\mathbf{p_f}) - \mathbf{p_m})||$, where T is the estimated transformation and $\mathbf{p_f}$, $\mathbf{p_m}$ are the points that were **used for estimation**.
- TRE computes the error of the target fiducials following registration, i.e. $||T(\mathbf{p_f}) - \mathbf{p_m})||$, where T is the estimated transformation and $\mathbf{p_f}$, $\mathbf{p_m}$ are the points that were **not used for estimation**.

It is important to remember that FRE should never be utilized as a surrogate for TRE as the two error measures are uncorrelated given a specific registration task. Typically, we can only estimate the distribution of TRE as it is spatially varying. A good TRE depends on using a good fiducial configuration. More information on FRE and TRE can be found in this article.

If the transformation is unknown, image similarity metrics (see *notebook 1.3*), and the Structural Similarity Index (SSIM) can be used. The SSIM is a perceptual image quality measure indicating whether two images are very similar or the same (a value of $+1$) or very different (a value of -1).

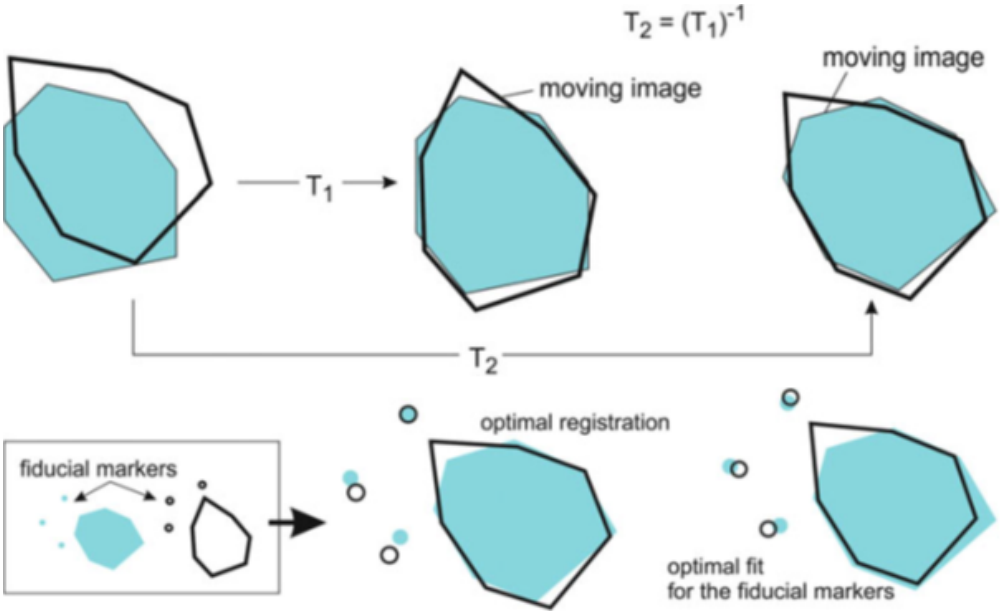


Figure from Guide to Medical Image Analysis - Methods and Algorithms

(Computer-aided) detection - quality measures

In detection tasks, an object is either found or not found while the object is or is not present in the data. The quality of detection is measured by *sensitivity* (a.k.a recall rate) and *specificity* (a.k.a precision rate):

- True positives (TP) are those detections belonging to the data and rightly resulted as positive.
- True negatives (TN) are those objects not present in the data and rightly resulted as negative.
- False positives (FP) are those objects that do not belong to the data, but were detected as present.
- False negatives (FN) are results that belong to the data, but were classified as absent.

Sensitivity can be calculated as $\frac{\mathrm{TP}}{\mathrm{TP+FN}}$, while specificity is defined as $\frac{\mathrm{TN}}{\mathrm{TN+FP}}$. A good detection method would produce as many TP and TN as possible. FPs (e.g. tumor detected, though absent) and FNs (e.g. tumor overlooked) may have various consequences, and are therefore measured as two types of error (type-I error, and type-II error). The so-called *confusion matrix* listing detection results in an organized way, specifies a two-class classification problem:

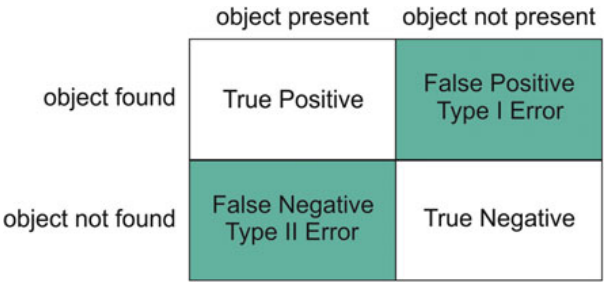


Figure from Guide to Medical Image Analysis - Methods and Algorithms

These metrics are commonly used in detection tasks involving medical images. Interestingly, they are also very important when interpreting the performance of any test (e.g., airport security, breast cancer screening, quality assurance in companies, etc.).

In practice, a trade-off between specificity and sensitivity is often targeted. In detection tasks, the ratio of sensitivity versus specificity is measured by the *receiver operator characteristic* (ROC). The ROC curve can also serve as a measure of human operator performance when several operators performed the same task.

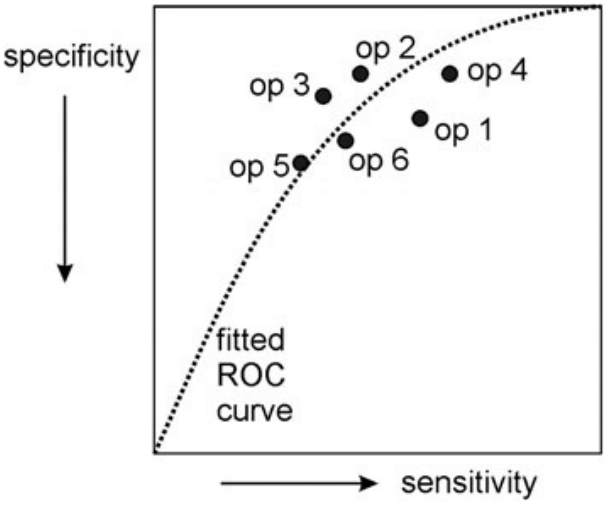


Figure from Guide to Medical Image Analysis - Methods and Algorithms

1.6.2 2. Common limitations of performance metrics used for segmentation tasks

Recent meta-analytical research has detected major issues in algorithm validation. Most of these flaws are related to the practical use of some performance metrics in a given analysis task. One of the core issues in medical image analysis is the choice of inappropriate metrics Maier-Hein, L. et al. (2018). In the same publication, it has been reported that image segmentation is the most popular of all medical image processing tasks taking into account international challenges. In these competitions, the chosen metrics significantly influence the rankings of various methods, and it was found out that researchers are missing guidelines for choosing the right metric for a given problem. More information can be found in the article Common Limitations of Image Processing Metrics: A Picture Story.

Small structures

It is important to understand the mathematical properties of a metric before applying it to a given task. Segmentation of small structures, such as brain lesions (e.g. multiple sclerosis) often employs Dice scores, which may not be an appropriate metric because of the often unknown pathological outlines and high inter-observer variability in such tasks. The predictions of two algorithms may differ only by one pixel, yet the impact on the Dice score outcome is substantial (see figure below).

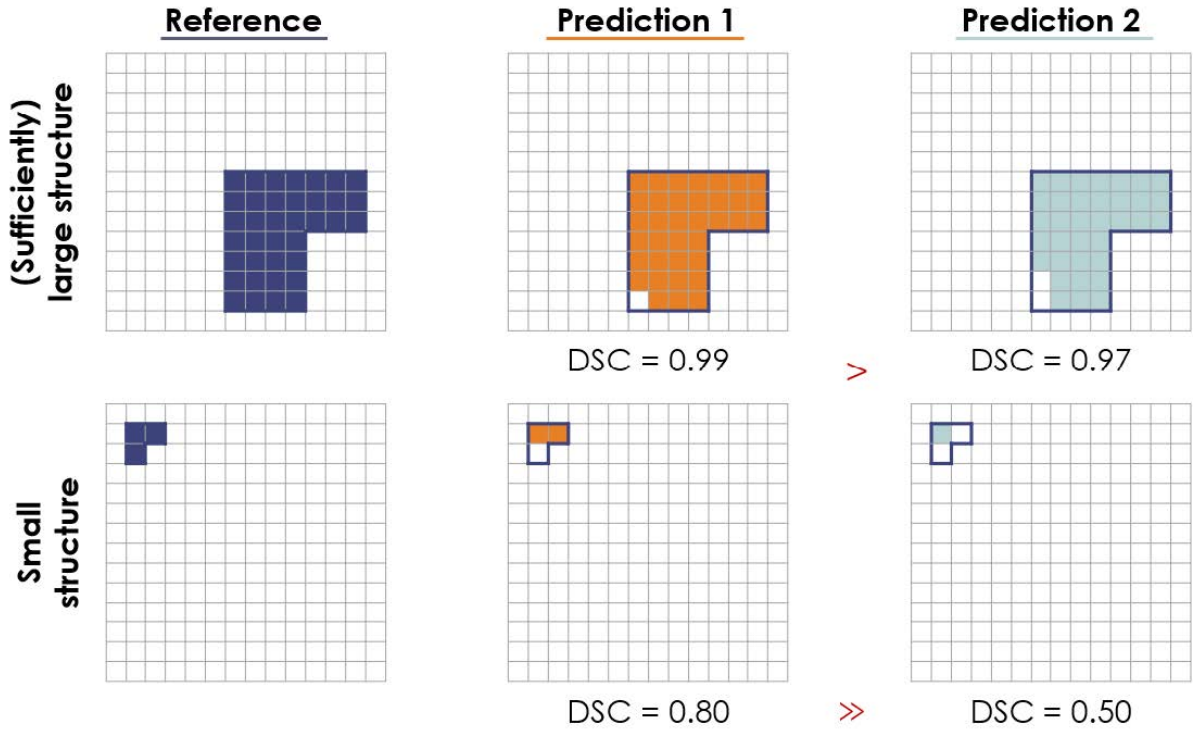


Figure from Common Limitations of Image Processing Metrics: A Picture Story

Image artifacts

Similar issues may arise in the presence of image artifacts such as noise or errors in reference annotations. As seen in the figure below, a single erroneous pixel in the reference annotation may lead to a large performance decrease.

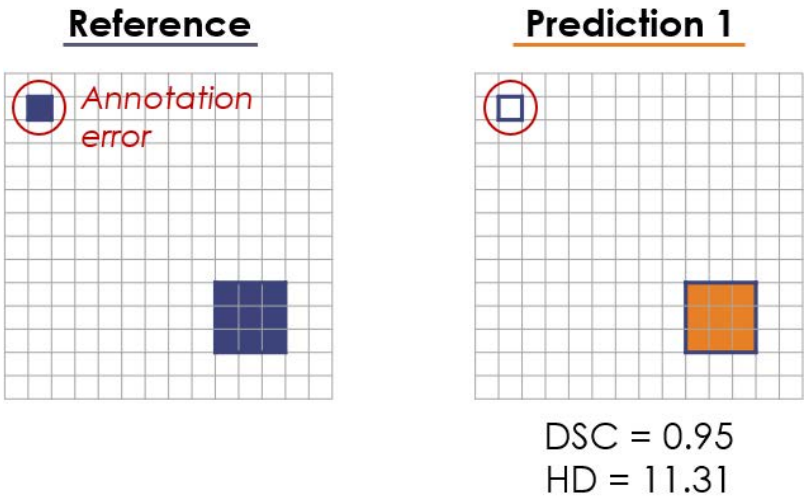


Figure from Common Limitations of Image Processing Metrics: A Picture Story

Overlap measurements

In overlap measurements, dedicated metrics are incapable of discovering differences in shapes, which may have huge impact e.g. on radiotherapy applications. Completely different predictions may therefore lead to the exact same DSC value.

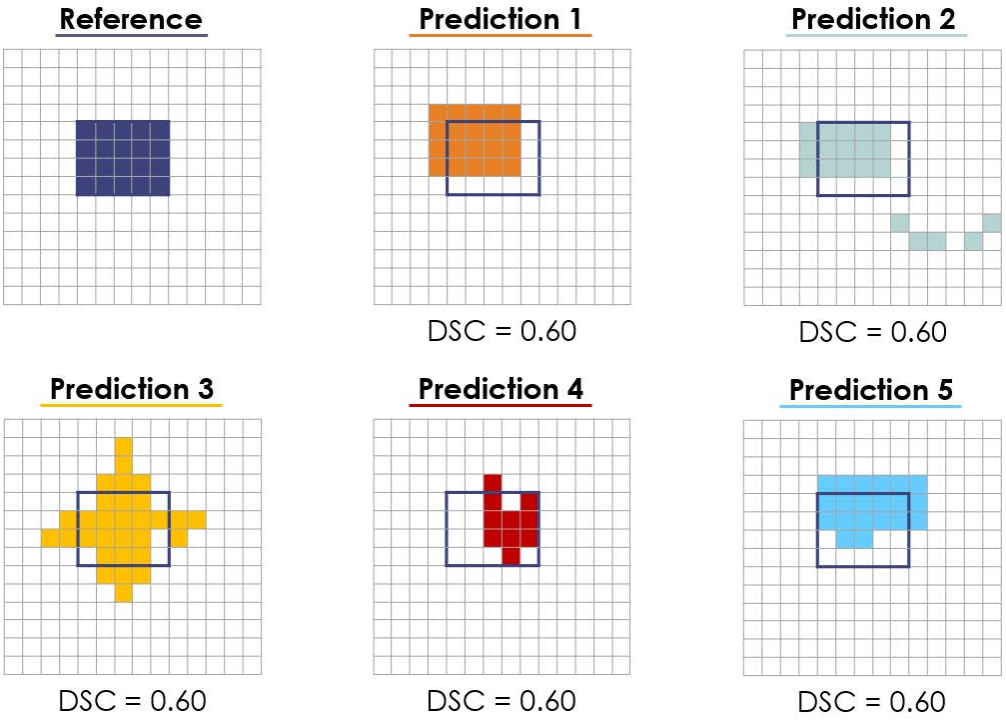


Figure from Common Limitations of Image Processing Metrics: A Picture Story

Over- and undersegmentation

In some applications detecting over- and undersegmentation, the DSC metric does not represent these performance indicators reliably, while HD is invariant to these properties.

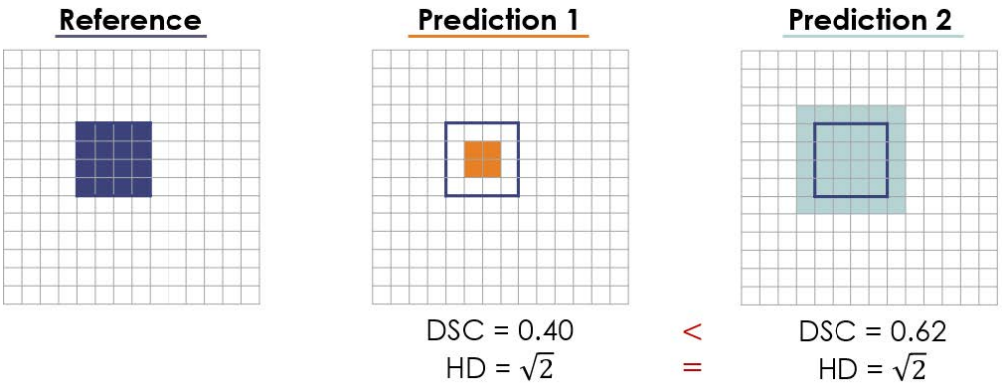


Figure from Common Limitations of Image Processing Metrics: A Picture Story

Single object bias

Commonly, segmentation metrics, such as DSC, are applied to detection and localization problems as well. In general, the DSC tends to be strongly biased against single objects, which is why its application in detection tasks should be avoided. An example where DSC underperforms, can be seen below.

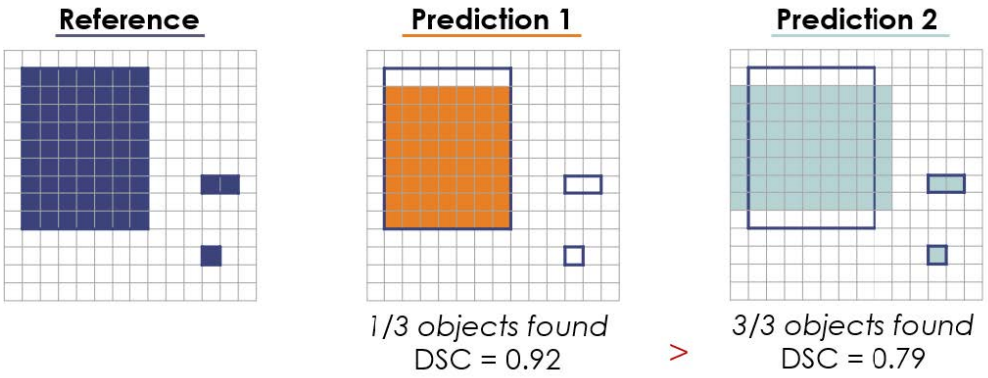


Figure from Common Limitations of Image Processing Metrics: A Picture Story

Metric combination

Metrics are typically combined over all test cases to produce overall ranking. However, this can be detrimental in case of missing values (NA) and lead to a substantially higher DSC or varying HD compared to setting missing values to zero. Moreover, a single metric usually does not reflect all important features for algorithm validation. Through the combination of multiple metrics helps mitigate the problem, it has to be kept in mind that some metrics are mathematically related to each other, such as DSC and Intersection over union (IoU) (see above). Thus combining related metrics will not change the ranking, and only metrics measuring different properties should be aggregated.

Choosing the right metric for a given task

The selection of the most appropriate metric depends on your biomedical question and the characteristics of its problem:

- What is the size, volume and shape of structures?
- Are there image artefacts?
- What is the annotation quality?
- Is computation time relevant?
- Is there any reference available?
- Do we prefer higher sensitivity or specificity?

Question 1:

Describe a situation where volume computation would be an appropriate criterion for measuring the quality of a segmentation task. When should it not be used?

Type your answer here

Question 2:

What information about segmentation quality is revealed by the Hausdorff distance? Please describe a scenario where this measure is important to rate a segmentation method.

Type your answer here

***Question 3*:**

What needs to be made sure when selecting test data for ground truth?

Type your answer here

***Question 4*:**

Why is it necessary to carry out manual segmentation several times by different and by the same person if it shall be used for ground truth? How is the information that is gained from these multiple segmentations used for rating the performance of an algorithm?

Type your answer here

1.7 Project 1: Image registration

Contents:

1.7.1 Goal

Develop Python code for point-based and intensity-based (medical) image registration. Use the developed code to perform image registration and evaluate and analyze the results.

The dataset you will be using in the first mini-project originates from the MRBrainS medical image analysis challenge. It consists of 30 traverse slices of MR brain scans with two different sequences: T1-weighted and T2-FLAIR (5 patients $\times$ 3 slices per patient $\times$ 2 modalities). Please see the Getting started assignment below for more details on the dataset.

1.7.2 Deliverables

Code and a report describing your implementation, results and analysis. There is no hard limit for the length of the report, however, concise and short reports are **strongly** encouraged. Aim to present your most important findings in the main body of the report and (if needed) any additional information in an appendix. The following report structure is suggested for the main body of the report:

1. Introduction
2. Methods
3. Results
4. Discussion

The introduction and result sections can be very brief in this case (e.g. half a page each). The discussion section should contain the analysis of the results. The report must be submitted as a single PDF file. The code must be submitted as a single archive file (e.g. zip or 7z) that is self-contained and can be used to reproduce the results in the report.

Note that there is no single correct solution for the project. You have to demonstrate to the reader that you understand the methods that you have studied and can critically analyze the results of applying the methods. Below, you can find a set of assignments (guided project work) that will help you get started with the project work and, when correctly completed, will present you with a **minimal solution**. Solutions which go beyond these assignments are of course encouraged.

1.7.3 Assessment

The rubric that will be used for assessment of the project work is given in this table

[1]:
```
%load_ext autoreload
%autoreload 2
```

1.7.4 Guided project work

A. Getting started

As an introduction, you will get familiar with the dataset that will be used in the first mini-project and the control point selection tool that can be used to annotate corresponding points in pairs of related images. The annotated points can later be used to perform point-based registration and evaluation of the registration error.

Dataset

The image dataset is located in the image_data subfolder of the code for the registration exercises and project. The image filenames have the following format: `{Patient ID}_{Slice ID}_{Sequence}.tif`. For example, the filename `3_2_t1.tif` is the second slice from a T1-weighted scan of the third patient. Every T1 slice comes in two versions: original and transformed with some random transformation that can be identified with the `_d` suffix in the filename. This simulates a registration problem where you have to register two image acquisitions of the same patient (note however that some of the transformations that were used to simulate the second set of images are not realistic for brain imaging, e.g. brain scans typically do not encounter shearing between consecutive acquisitions).

Question 1:

With this dataset we can define two image registration problems: T1 to T1 registration (e.g. register `3_2_t1_d.tif` to `3_2_t1.tif`) and T2 to T1 registration (e.g. register `3_2_t2.tif` to `3_2_t1.tif`). Which one of these can be considered inter-modal image registration and which one intra-modal image registration?

Selecting corresponding point pairs

A function called `cpselect` is provided to select control points in two different images. This function provides two numpy arrays of cartesian coordinates, one array for each image, of points selected in the two images. The coordinate format is a numpy array with the X and Y on row 0 and 1 respectively, and each column being a point.

Calling the function will cause a new interactive window to pop up, where you will see your two images and some instructions. For convenience, the instructions can also be found below:

- First select a point in Image 1 and then its corresponding point in Image 2. This pattern should be repeated for as many control points as you need. If you do not follow this pattern, the output arrays will be incorrect.
- Left Mouse Button to create a point.
- Right Mouse Button/Delete/Backspace to remove the newest point.
- Middle Mouse Button/Enter to finish placing points.

Task 1:

Test the functionality of `cpselect` by running the following code example:

[2]:
```
import sys
sys.path.append("../code")
import registration_util as util

I_path = '../data/image_data/1_1_t1.tif'
Im_path = '../data/image_data/1_1_t1_d.tif'

X, Xm = util.cpselect(I_path, Im_path)
```

(continues on next page)

(continued from previous page)

```
print('X:\n{}'.format(X))
print('Xm:\n{}'.format(Xm))
```

```
---------------------------------------------------------------------------
ImportError                               Traceback (most recent call last)
Cell In[2], line 8
      5 I_path = '../data/image_data/1_1_t1.tif'
      6 Im_path = '../data/image_data/1_1_t1_d.tif'
----> 8 X, Xm = util.cpselect(I_path, Im_path)
     10 print('X:\n{}'.format(X))
     11 print('Xm:\n{}'.format(Xm))

File ~/checkouts/readthedocs.org/user_builds/8dc00-mia-docs/checkouts/latest/docs/source/
↪code/registration_util.py:77, in cpselect(imagePath1, imagePath2)
     74 image2 = plt.imread(imagePath2)
     76 #ensure that the plot opens in its own window
---> 77 get_ipython().run_line_magic('matplotlib', 'qt')
     79 #set up the overarching window
     80 fig, axes = plt.subplots(1,2)

File ~/checkouts/readthedocs.org/user_builds/8dc00-mia-docs/envs/latest/lib/python3.8/
↪site-packages/IPython/core/interactiveshell.py:2417, in InteractiveShell.run_line_
↪magic(self, magic_name, line, _stack_depth)
   2415     kwargs['local_ns'] = self.get_local_scope(stack_depth)
   2416 with self.builtin_trap:
-> 2417     result = fn(*args, **kwargs)
   2419 # The code below prevents the output from being displayed
   2420 # when using magics with decodator @output_can_be_silenced
   2421 # when the last Python token in the expression is a ';'.
   2422 if getattr(fn, magic.MAGIC_OUTPUT_CAN_BE_SILENCED, False):

File ~/checkouts/readthedocs.org/user_builds/8dc00-mia-docs/envs/latest/lib/python3.8/
↪site-packages/IPython/core/magics/pylab.py:99, in PylabMagics.matplotlib(self, line)
     97     print("Available matplotlib backends: %s" % backends_list)
     98 else:
---> 99     gui, backend =␣
↪self.shell.enable_matplotlib(args.gui.lower() if isinstance(args.gui, str) else args.gui)
    100     self._show_matplotlib_backend(args.gui, backend)

File ~/checkouts/readthedocs.org/user_builds/8dc00-mia-docs/envs/latest/lib/python3.8/
↪site-packages/IPython/core/interactiveshell.py:3603, in InteractiveShell.enable_
↪matplotlib(self, gui)
   3599         print('Warning: Cannot change to a different GUI toolkit: %s.'
   3600                 ' Using %s instead.' % (gui, self.pylab_gui_select))
   3601         gui, backend = pt.find_gui_and_backend(self.pylab_gui_select)
-> 3603 pt.activate_matplotlib(backend)
   3604 configure_inline_support(self, backend)
   3606 # Now we must activate the gui pylab wants to use, and fix %run to take
   3607 # plot updates into account

File ~/checkouts/readthedocs.org/user_builds/8dc00-mia-docs/envs/latest/lib/python3.8/
↪site-packages/IPython/core/pylabtools.py:360, in activate_matplotlib(backend)
```

(continues on next page)

(continued from previous page)

```
    355 # Due to circular imports, pyplot may be only partially initialised
    356 # when this function runs.
    357 # So avoid needing matplotlib attribute-lookup to access pyplot.
    358 from matplotlib import pyplot as plt
--> 360 plt.switch_backend(backend)
    362 plt.show._needmain = False
    363 # We need to detect at runtime whether show() is called by the user.
    364 # For this, we wrap it into a decorator which adds a 'called' flag.

File ~/checkouts/readthedocs.org/user_builds/8dc00-mia-docs/envs/latest/lib/python3.8/
↪site-packages/matplotlib/pyplot.py:271, in switch_backend(newbackend)
    268 # have to escape the switch on access logic
    269 old_backend = dict.__getitem__(rcParams, 'backend')
--> 271 backend_mod = importlib.import_module(
    272     cbook._backend_module_name(newbackend))
    274 required_framework = _get_required_interactive_framework(backend_mod)
    275 if required_framework is not None:

File ~/checkouts/readthedocs.org/user_builds/8dc00-mia-docs/envs/latest/lib/python3.8/
↪importlib/__init__.py:127, in import_module(name, package)
    125             break
    126         level += 1
--> 127 return _bootstrap._gcd_import(name[level:], package, level)

File <frozen importlib._bootstrap>:1014, in _gcd_import(name, package, level)

File <frozen importlib._bootstrap>:991, in _find_and_load(name, import_)

File <frozen importlib._bootstrap>:975, in _find_and_load_unlocked(name, import_)

File <frozen importlib._bootstrap>:671, in _load_unlocked(spec)

File <frozen importlib._bootstrap_external>:783, in exec_module(self, module)

File <frozen importlib._bootstrap>:219, in _call_with_frames_removed(f, *args, **kwds)

File ~/checkouts/readthedocs.org/user_builds/8dc00-mia-docs/envs/latest/lib/python3.8/
↪site-packages/matplotlib/backends/backend_qt5agg.py:7
      4 from .. import backends
      6 backends._QT_FORCE_QT5_BINDING = True
----> 7 from .backend_qtagg import (    # noqa: F401, E402 # pylint: disable=W0611
      8     _BackendQTAgg, FigureCanvasQTAgg, FigureManagerQT, NavigationToolbar2QT,
      9     FigureCanvasAgg, FigureCanvasQT)
     12 @_BackendQTAgg.export
     13 class _BackendQT5Agg(_BackendQTAgg):
     14     pass

File ~/checkouts/readthedocs.org/user_builds/8dc00-mia-docs/envs/latest/lib/python3.8/
↪site-packages/matplotlib/backends/backend_qtagg.py:9
      5 import ctypes
      7 from matplotlib.transforms import Bbox
----> 9 from .qt_compat import QT_API, _enum
```

(continues on next page)

(continued from previous page)

```
     10 from .backend_agg import FigureCanvasAgg
     11 from .backend_qt import QtCore, QtGui, _BackendQT, FigureCanvasQT

File ~/checkouts/readthedocs.org/user_builds/8dc00-mia-docs/envs/latest/lib/python3.8/
↪site-packages/matplotlib/backends/qt_compat.py:135
    133         break
    134     else:
--> 135         raise ImportError(
    136             "Failed to import any of the following Qt binding modules: {}"
    137             .format(", ".join(_ETS.values())))
    138 else:  # We should not get there.
    139     raise AssertionError(f"Unexpected QT_API: {QT_API}")

ImportError: Failed to import any of the following Qt binding modules: PyQt6, PySide6,␣
↪PyQt5, PySide2
```

1.7.5 B. Point-based registration

Point-based affine image registration

From the provided dataset for this project, select one pair of T1 image slices (e.g. `3_2_t1.tif` and `3_2_t1_d.tif`) and use `my_cpselect` to select a set of corresponding points. Then, compute the affine transformation between the pair of images with `ls_affine` and apply it to the moving image using `image_transform`.

Repeat the same for a pair of corresponding T1 and T2 slices (e.g. `3_2_t1.tif` and `3_2_t2.tif`).

Evaluation of point-based affine image registration

***Question 2*:**

Describe how you would estimate the registration error. (Hint: Should you use the same points that you used for computing the affine transformation to also compute the registration error?) How does the number of corresponding point pairs affect the registration error? Motivate all your answers.

1.7.6 C. Intensity-based registration

Comparing the results of different registration methods

The following Python script (provided as `intensity_based_registration_demo()`) performs rigid intensity-based registration of two images using the normalized-cross correlation as a similarity metric:

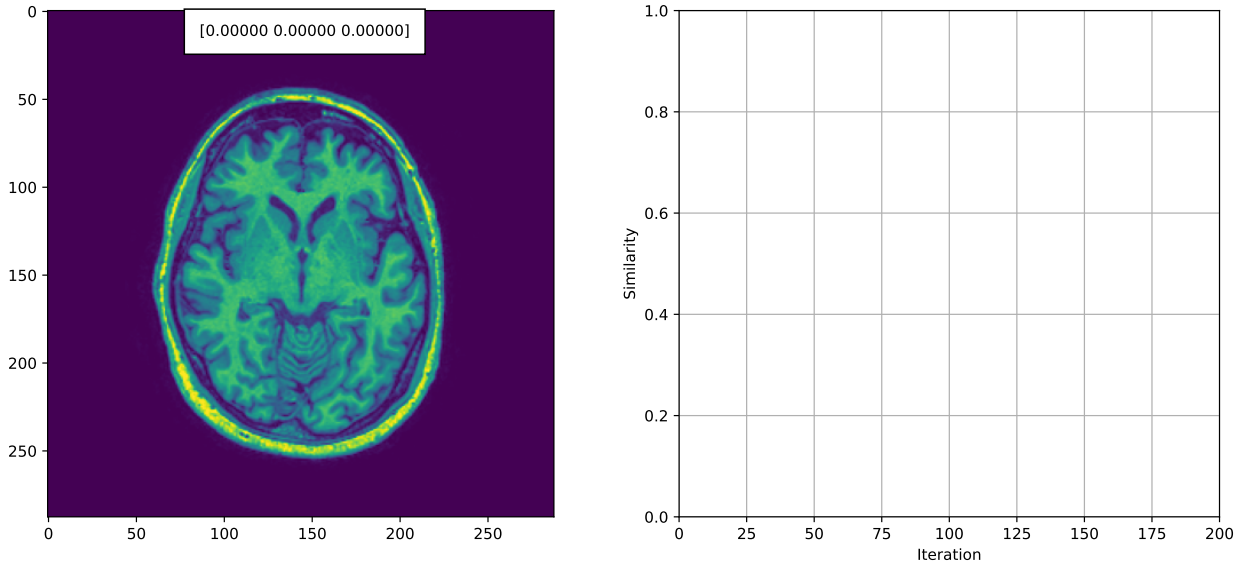
[3]:
```
%matplotlib inline
import sys
sys.path.append("../code")
from registration_project import intensity_based_registration_demo

intensity_based_registration_demo()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[3], line 6
      3 sys.path.append("../code")
      4 from registration_project import intensity_based_registration_demo
----> 6 intensity_based_registration_demo()

File ~/checkouts/readthedocs.org/user_builds/8dc00-mia-docs/checkouts/latest/docs/source/
↪code/registration_project.py:74, in intensity_based_registration_demo()
     71 x += g*mu
     73 # for visualization of the result
---> 74 S, Im_t, _ = reg.rigid_corr(I, Im, x, return_transform=True)
     76 clear_output(wait = True)
     78 # update moving image and parameters

File ~/checkouts/readthedocs.org/user_builds/8dc00-mia-docs/checkouts/latest/docs/source/
↪code/registration.py:342, in rigid_corr(I, Im, x, return_transform)
    339 SCALING = 100
    341 # the first element is the rotation angle
--> 342 T = rotate(x[0])
    344 # the remaining two element are the translation
    345 #
    346 # the gradient ascent/descent method work best when all parameters
   (...)
    351 # scaled down version of the translation vector to this function
    352 # and then scale it up when computing the transformation matrix
    353 Th = util.t2h(T, x[1:]*SCALING)

File ~/checkouts/readthedocs.org/user_builds/8dc00-mia-docs/checkouts/latest/docs/source/
↪code/registration.py:46, in rotate(phi)
     35 def rotate(phi):
     36     # 2D rotation matrix.
     37     # Input:
   (...)
     43     # TODO: Implement transformation matrix for rotation.
```

(continues on next page)

(continued from previous page)

```
     44     #-------------------------------------------------------------------#
---> 46     return T

NameError: name 'T' is not defined
```



Task 2:

By changing the similarity function and the initial parameter vector, you can also use this script to perform affine registration and use mutual information as a similarity measure. Do not forget to also change the transformation for the visualization of the results.

Using the provided dataset and the functions that you have implemented in the exercises, perform the following series of experiments:

1. Rigid intensity-based registration of two T1 slices (e.g. `1_1_t1.tif` and `1_1_t1_d.tif`) using normalized cross-correlation as a similarity measure.
2. Affine intensity-based registration of two T1 slices (e.g. `1_1_t1.tif` and `1_1_t1_d.tif`) using normalized cross-correlation as a similarity measure.
3. Affine intensity-based registration of a T1 and a T2 slice (e.g. `1_1_t1.tif` and `1_1_t2.tif`) using normalized cross-correlation as a similarity measure.
4. Affine intensity-based registration of two T1 slices (e.g. `1_1_t1.tif` and `1_1_t1_d.tif`) using mutual information as a similarity measure.
5. Affine intensity-based registration of a T1 slice and a T2 slice (e.g. `1_1_t1.tif` and `1_1_t2.tif`) using mutual information as a similarity measure.

Describe, analyze and compare the results from each experiment. If a method fails, describe why you think it fails. Note that you will most likely have to try different values for the learning rate in each experiment in order to find the one that works best.

1.8 Topic 2.1: Linear regression

This notebook combines theory with exercises to support the understanding of linear regression in computer-aided diagnosis. Implement all functions in the `code` folder of your cloned repository, and test it in this notebook after implementation by importing your functions to this notebook. Use available markdown sections to fill in your answers to questions as you proceed through the notebook.

Contents:

1. Linear regression (theory)
2. Implementing linear regression
3. Polynomial regression and model selection
4. k-Nearest neighbor classifier

References:

[1] Schneider, Astrid et al. "Linear regression analysis: part 14 of a series on evaluation of scientific publications." Deutsches Arzteblatt international vol. 107,44 (2010): 776-82. LINK

[2] k-Nearest neighbor classifier: LINK

1.8.1 1. Linear regression (theory)

Linear regression is an indispensable tool for statistical analysis and can be considered the most basic building block of neural networks. In its simplest terms, univariate linear regression helps estimate the association between a **continuous** dependent variable (outcome) and an independent explanatory variable (predictor) by fitting a linear equation to observed data. Multivariate linear regression then uses two or more independent variables to predict certain outcome. In medical applications, linear regression allows for the identification of prognostically important risk factors (e.g. weight, blood pressure, etc.).

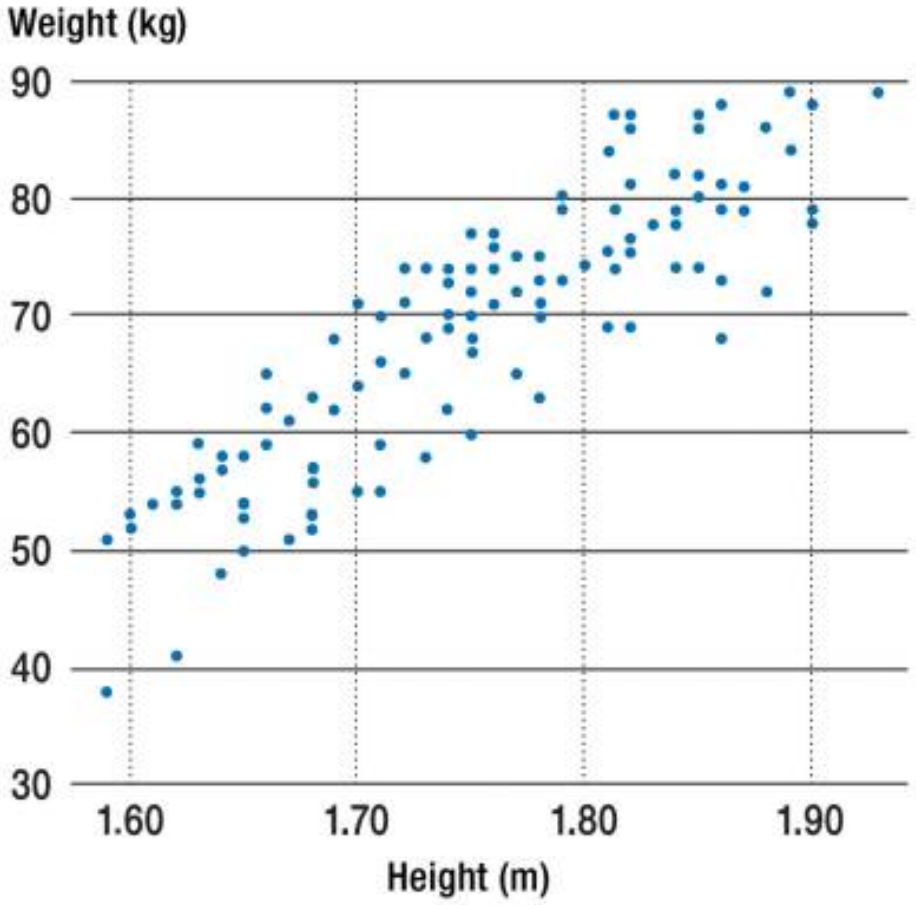


Figure from Linear Regression Analysis (Deutsches Ärzteblatt International)

The equation for linear regression is expressed as $Y = a + bX$, where X is the independent variable and Y is the dependent variable. b denotes the slope of the linear regression line, and a is the intercept (y at $x = 0$).

To graphically visualize linear relationship and its strength between two variables, a scatterplot is commonly used. A fitted regression line (via the least squares method) across all data points then shows either an increasing or decreasing trend. Numerically, the association strength between two variables can be evaluated using the correlation coefficient (R^2), followed by calculating the p-value to determine statistical significance.

After fitting a regression line to a group of data, deviations from the fitted line to the observed values (the so-called *residuals*) allow the observer to inspect the validity of their assumption and accept/reject the hypothesis that a linear relationship exists.

Computed regression lines may be affected by *outliers* (data points lying far away from the main data cluster in the scatterplot). Depending on their position, outliers may have a major impact on the computed trend since these data points may represent erroneous data. The effect of outliers as well as influential observations (horizontally distant points) should be properly investigated and such data potentially removed.

1.8.2 2. Implementing linear regression

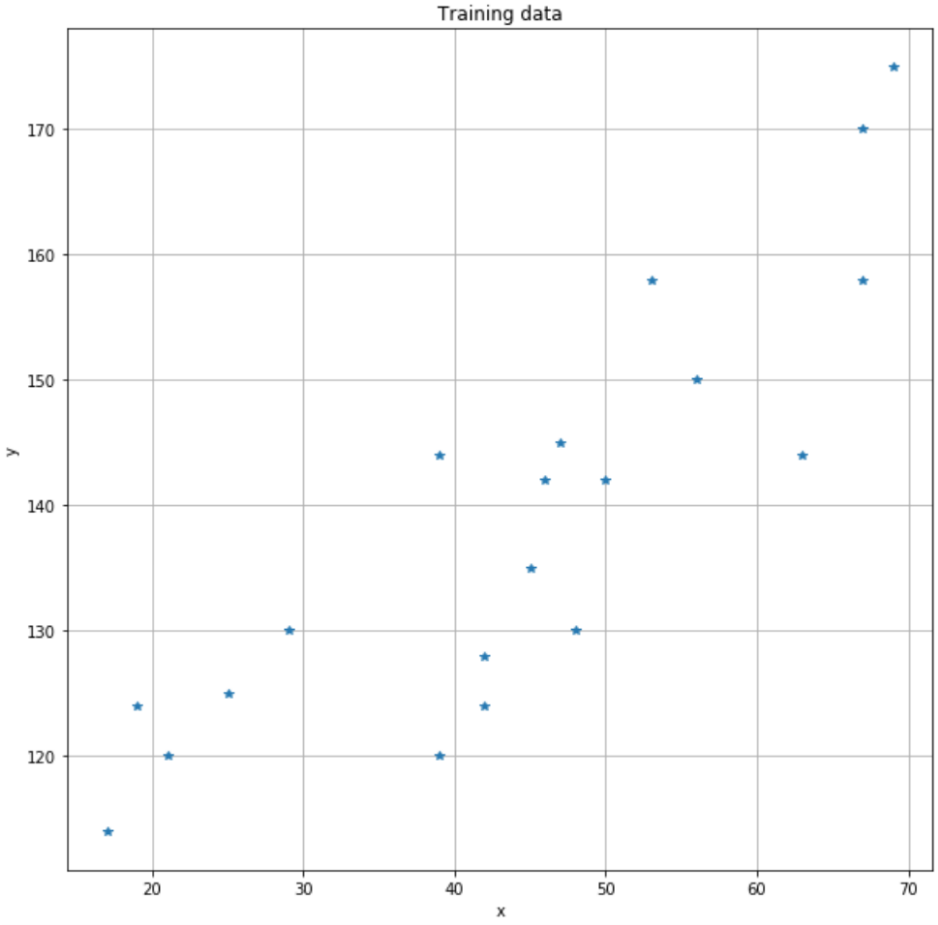
The optimal parameters of a linear regression model given a training dataset of features **X** and targets **y** can be obtained with the closed-form solution for minimization of the loss function:

$$J(\theta) = \|X\theta - y\|_2^2$$
$$\nabla_\theta J = 0$$
$$\theta = \left(X^\top X\right)^{-1} X^\top y$$

The function `ls_solve()` that you have implemented in the point-based registration practical (`SECTION 2` of the `registration.py` module) can be reused to solve for the parameters θ.

The `linear_regression()` Python script in `SECTION 1` of the `cad_tests.py` module reads a toy dataset split into training, validation and testing subsets, computes the parameters of a linear regression model and visualizes the results for the training and testing datasets. The toy dataset consists of a single feature and a target variable. For example, the target that we want to predict can be a person's systolic blood pressure and age can be the single feature that describes the person. Such a "small" problem is not often encountered in practice but it can be very illustrative for this technique (in the project work you will work with a more "practical" medical image analysis problem).

The first section of `linear_regression()` loads the training, validation and testing datasets that will be used for training and evaluation of the linear regression model. It also shows a plot of the feature vs. the target variable. We can observe from the plot that the value of the target tends to increase together with the value of the feature.



?

Question 2.1:

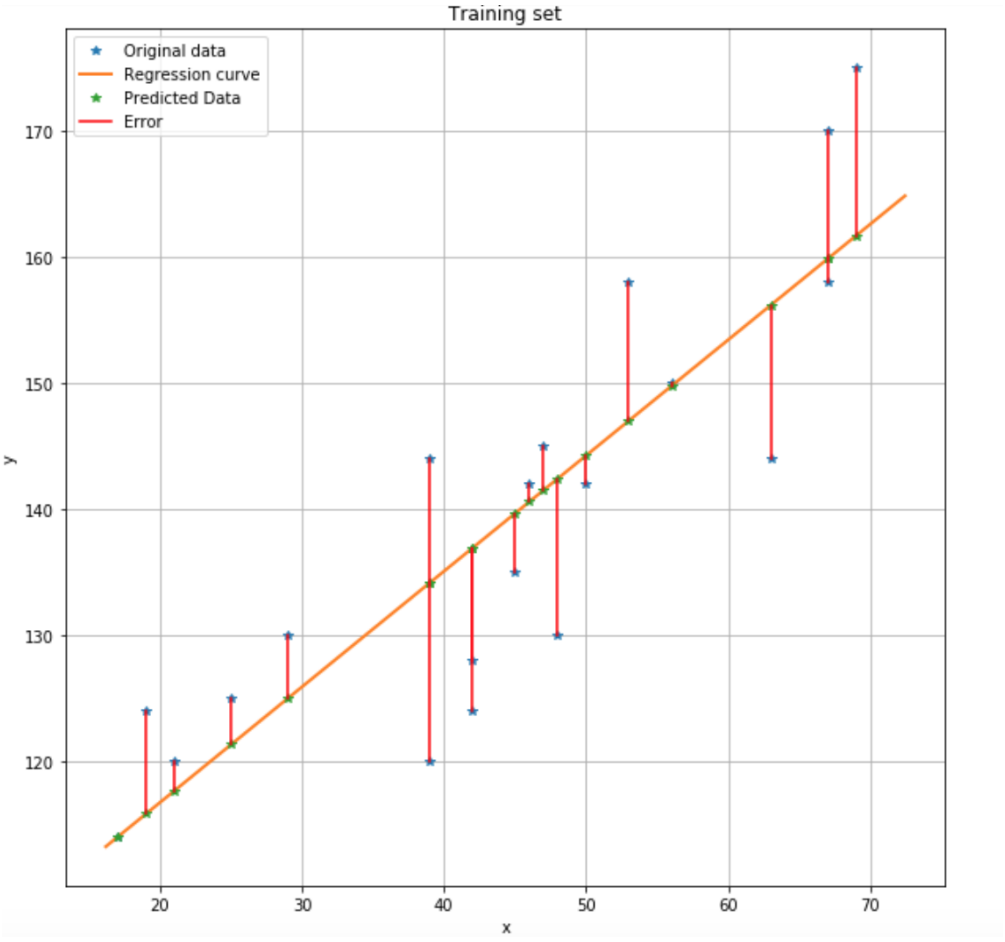
What is role of these three subsets in training and evaluating machine learning models?

Type your answer here

Exercise 2.1:

Implement the missing functionality of `linear_regression()` that computes the parameters `Theta` of the linear regression model. Note that you will have to add a column of all ones to the data matrix, for which you can use the provided `addones()` function in the `cad_util.py` module.

If you have implemented this correctly, the results for the training set should look like in the figure below.



[1]:
```
%matplotlib inline
import sys
sys.path.append("../code")
from cad_tests import linear_regression

E_validation, E_test = linear_regression()
```

```
Traceback (most recent call last):

  File ~/checkouts/readthedocs.org/user_builds/8dc00-mia-docs/envs/latest/lib/python3.8/
↪site-packages/IPython/core/interactiveshell.py:3508 in run_code
    exec(code_obj, self.user_global_ns, self.user_ns)

  Cell In[1], line 4
    from cad_tests import linear_regression
  File ../code/cad_tests.py:424
    def rotate_using_eigenvectors_test(X, Y, v):
    ^
IndentationError: expected an indented block
```

Exercise 2.2:

How can you compute the error of the linear regression model for the optimal parameters? Implement this at the end of `linear_regression()`.

[2]:
```
print(E_validation)
print(E_test)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[2], line 1
----> 1 print(E_validation)
      2 print(E_test)

NameError: name 'E_validation' is not defined
```

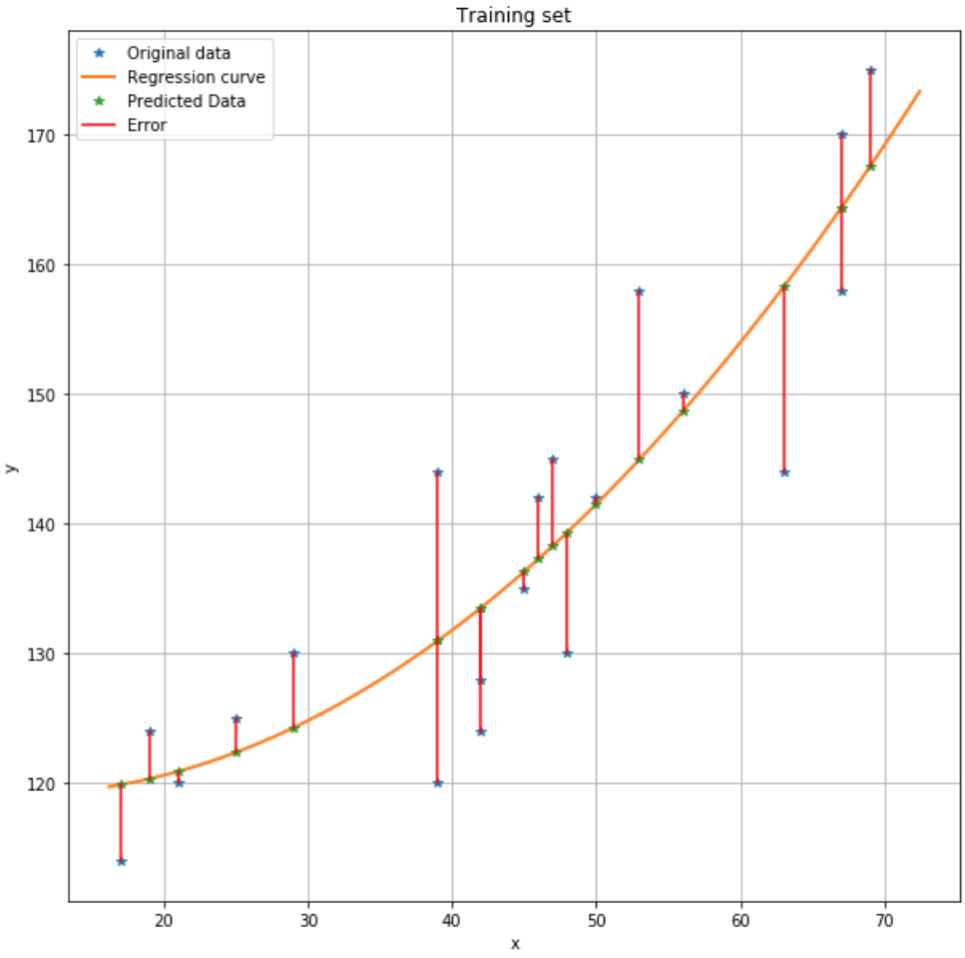
1.8.3 3. Polynomial regression and model selection

Suppose that after examining the results from the linear regression model, your conclusion is that a quadratic model might be a better fit for the data. Instead of a line, the fitted model now resembles a parabola, which is described by the equation $y = ax^2 + bx + c$.

Exercise 3.1:

Use the existing code for linear regression to implement and evaluate such a model. You can make a copy of `linear_regression()` called `quadratic_regression()` and work there.

If you have implemented this correctly, the results for the training set should look like in the figure below:



[3]:
```
%matplotlib inline
import sys
sys.path.append("../code")
from cad_tests import quadratic_regression

E_validation, E_test = quadratic_regression()
print(E_validation)
print(E_test)
```

```
Traceback (most recent call last):

  File ~/checkouts/readthedocs.org/user_builds/8dc00-mia-docs/envs/latest/lib/python3.8/
↪site-packages/IPython/core/interactiveshell.py:3508 in run_code
    exec(code_obj, self.user_global_ns, self.user_ns)

  Cell In[3], line 4
    from cad_tests import quadratic_regression
  File ../code/cad_tests.py:424
```

(continues on next page)

(continued from previous page)

```
    def rotate_using_eigenvectors_test(X, Y, v):
    ^
IndentationError: expected an indented block
```

***Question 3.1*:**

You now have implemented both linear and quadratic regression. Compare the quadratic regression to the linear regression model. Which model would you choose and why?

Type your answer here

***Question 3.2*:**

After choosing one of the two models, you have to report the error. For which dataset should you report the error?

Type your answer here

1.8.4 4. Nearest neighbor classifier

The k-Nearest Neighbor is a type of supervised learning algorithm used both in regression and classification tasks (e.g. to classify a CT or MRI scan as benign or malignant based on given features). Given N training vectors, the k-NN algorithm tries to predict the class for the test data (e.g. a feature vector $\mathbf{c}$) by calculating the distance between $\mathbf{c}$ and other training points. The variable k represents the selected number of points which is closest to $\mathbf{c}$.

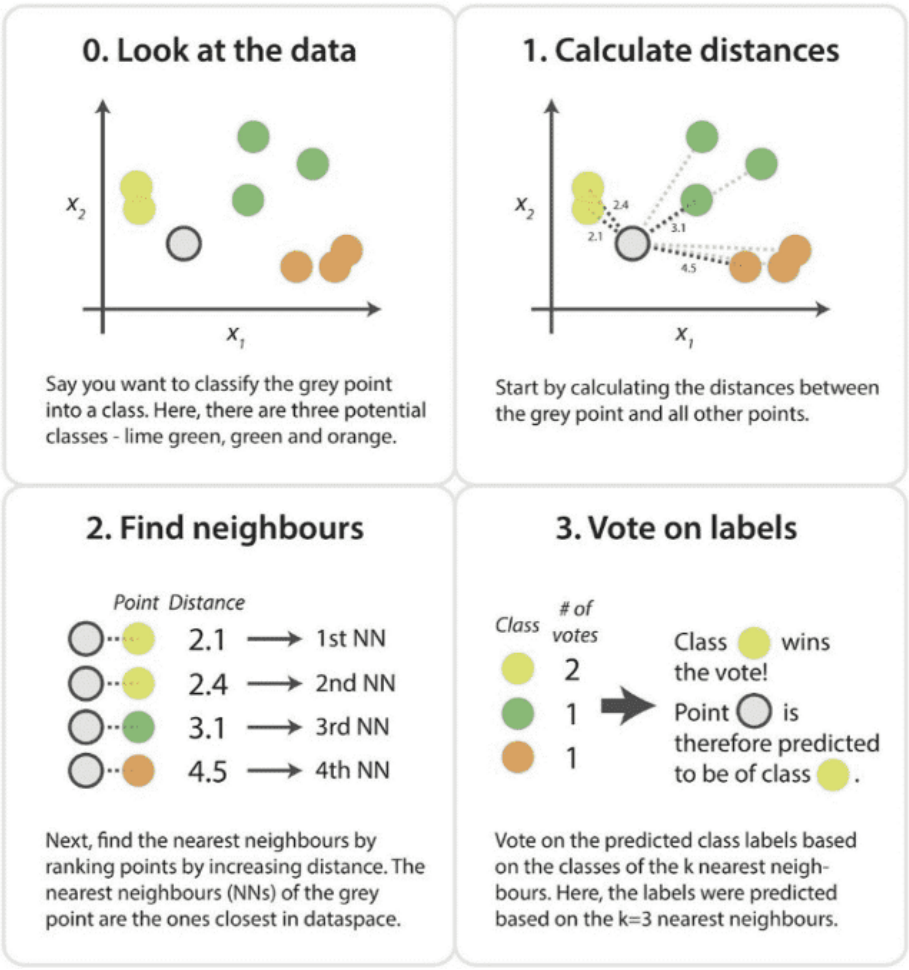


Figure from Antony Christopher on K-Nearest Neighbor

In classification tasks, k-NN algorithm is typically used to identify the category or class of a particular data point (or dataset) which is newly added to the space of two known categories, e.g. `A` and `B`. How does the algorithm determine the class or category? For a new example with features $x_{new} = [x_1, x_2]$, predict the class $\hat{y_{new}}$ as follows:

1. Specify the amount of neighbors k (e.g. 5)
2. Compute the distance from the new point to the k training samples. The most frequently used distance metric is the Euclidean distance calculated as $d(x_{new}, x_i) = \sqrt{(x_{new,1} - x_{i,1})^2 + (x_{new,2} - x_{i,2})^2}$ (Note: another often used metric is L1-distance)
3. Count the number of data points in each category among the k neighbors according to the Euclidean distance, sort them, and pick the nearest ones
4. Determine the class of the k nearest training samples
5. Assign to x_{new} the majority class of its nearest training samples (neighbors)
6. Algorithm has finished

Now, the question remains how to select the values of k. In general, the higher the value of k, the lesser the chance of erroneous classification. However, one has to keep in mind that every iteration of the distance calculation is computationally expensive. One cannot select the most applicable k-value via any pre-defined statistical methods. If we were to choose the optimal value of k based on the performance on the training set, we would always select $k = 1$ since the training error would be 0. Hence, we need to choose k based on the performance on an independent test set. The test

set should be independent in the sense that the examples that it contains should by no means be related to the ones in the training set.

1.9 Topic 2.2: Logistic regression

This notebook combines theory with exercises to support the understanding of logistic regression in computer-aided diagnosis. Implement all functions in the `code` folder of your cloned repository, and test it in this notebook after implementation by importing your functions to this notebook. Use available markdown sections to fill in your answers to questions as you proceed through the notebook.

Contents:

1. Logistic regression (theory)
2. Implementing the components of logistic regression
3. Implementing logistic regression
4. Generalization and overfitting

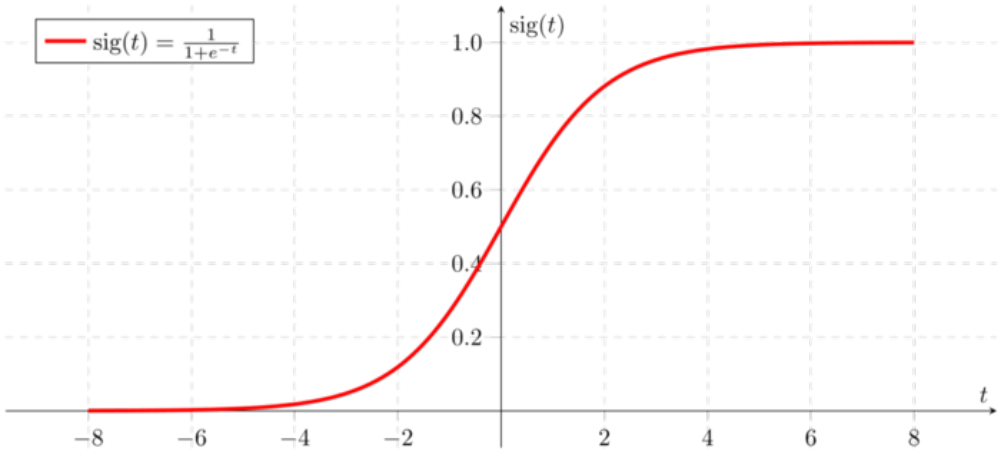
References:

[1] Peng, Chao-Ying J. An Introduction to Logistic Regression Analysis and Reporting, The Journal of Educational Research (2002) LINK

[1]:
```
%load_ext autoreload
%autoreload 2
```

1.9.1 1. Logistic regression (theory)

The logistic regression classifier can be considered as an extension of linear regression. However, logistic regression predicts whether something is True or False instead of predicting a continuous variable like height, for instance. Instead of fitting a line to the data, logistic regression fits an "S"-shaped curve (the sigmoid function) ranging from 0 to 1:



Logistic curve thus predicts the probability of an observation being classified into certain group. Logistic regression tests if a variable's effect on the prediction is significantly different from 0. Its ability to provide probabilities and classify new samples using continuous and discrete measurements makes it a popular machine learning approach. Logistic

regression does not have the same concept of residuals unlike linear regression, i.e. the least squares method cannot be applied and the correlation R^2 cannot be calculated. Instead, the concept of maximum likelihood is used. In medical applications, logistic regression serves for mortality prediction in injured patients or as a predictor of developing a certain disease.

1.9.2 2. Implementing the components of logistic regression

For a binary classification problem (a classification problem with two classes), logistic regression predicts the probability that a sample $\mathbf{x}$ belongs to one of the classes:

$$p(y = 1|\mathbf{x}) = \sigma\left(\boldsymbol{\theta}^\top \mathbf{x}\right)$$

We can view this expression as passing the output from a linear regression model $\boldsymbol{\theta}^\intercal \mathbf{x}$ through the sigmoid function $\sigma(\cdot)$ that "squashes" the value between 0 and 1 making it possible to be interpreted as a probability.

The loss function for logistic regression is the negative log-likelihood (NLL):

$$J(\theta) = -\sum_{i=1}^{N} y_i \log p\left(y = 1|\mathbf{x}_i, \theta\right) + (1 - y_i) \log\left\{1 - p\left(y = 1|\mathbf{x}_i, \theta\right)\right\}$$

Compared to linear regression, there is no closed-form solution for the optimal parameters of the model (we cannot set the derivative of $J(\boldsymbol{\theta})$ to zero and solve for $\boldsymbol{\theta}$). The NLL loss is optimised with the gradient descent method, similar to intensity-based image registration covered in the Registration topic of this course.

The provided `logistic_regression()` Python script in `SECTION 2` of the `cad_tests.py` module implements all necessary steps for training a logistic regression model on a toy dataset. However, the code will not work as is because two of the functions it depends on (`sigmoid()` and `lr_nll()`) are not implemented yet.

Exercise 2.1:

Implement the computation of the sigmoid function in `sigmoid()` in `SECTION 2` of the `cad.py` module. You will test your implementation in the next exercise.

Exercise 2.2:

Implement the computation of the negative log-likelihood in `lr_nll` in `SECTION 2` of the `cad.py` module. You will test your implementation in the next exercise.

***Question 2.1*:**

Suppose that you have two logistic regression models that predict $p(y = 1|\mathbf{x})$ and a validation dataset with three samples with labels 1, 0 and 1. The first model predicts the following probabilities for the three validation samples: 0.9, 0.4 and 0.7. The second model predicts 0.7. 0.5 and 0.9. Which of the two models has a better performance on the validation set? How did you come to this conclusion?

Type your answer here

1.9.3 3. Implementing logistic regression

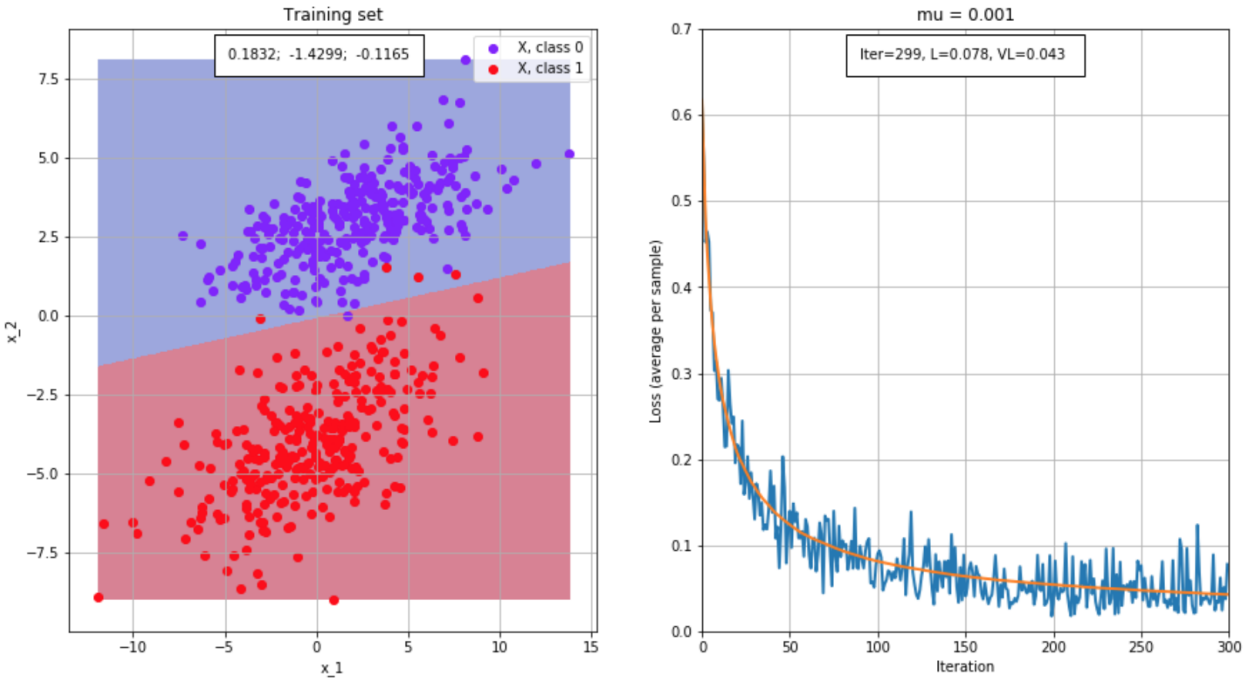
***Exercise 3.1*:**

The provided `logistic_regression()` Python script implements all necessary steps for training a logistic regression model on a toy dataset.

The first part of the script generates and visualizes a dataset for a binary classification problem. The code generates both a training and a validation dataset, which can be used to monitor for overfitting during the training process. The second part implements training of logistic regression with stochastic gradient descent. The training process is visualized in two ways: a scatter plot of the training data along with the linear decision boundary, and a plot of the training and validation loss as a function of the number of iterations (this is similar to the plot of the similarity vs. the number of iteration for intensity-baser image registration).

Read through the code and comments and make sure you understand what it does (you can skip the visualization part as it is not relevant for understanding logistic regression and stochastic gradient descent).

If you have implemented `sigmoid()` and `lr_nll()` correctly and run `logistic_regression()`, the results should look like on the figure below (it will most likely not be exactly the same as the toy dataset is randomly generated).



[2]:
```
%matplotlib inline
import sys
sys.path.append("../code")
from IPython.display import display, clear_output, HTML
from cad_tests import logistic_regression

logistic_regression()
```

```
Traceback (most recent call last):

  File ~/checkouts/readthedocs.org/user_builds/8dc00-mia-docs/envs/latest/lib/python3.8/
↪site-packages/IPython/core/interactiveshell.py:3508 in run_code
    exec(code_obj, self.user_global_ns, self.user_ns)

  Cell In[2], line 5
    from cad_tests import logistic_regression
  File ../code/cad_tests.py:424
    def rotate_using_eigenvectors_test(X, Y, v):
    ^
IndentationError: expected an indented block
```

?

Question 3.1:

What is the difference between "regular" gradient descent and stochastic gradient descent? What is the advantage of one over the other?

Type your answer here

?

Question 3.2:

In the figure above, the training loss curve has a noisy appearance, whereas the validation loss curve is relatively smooth. Why is this the case (**Tip**: How will the appearance of the training loss curve change if you increase the batch size parameter?).

Type your answer here

?

***Question 3.3*:**

Based on the training curves in the figure above, would you say that the model has overfitted the training dataset? Motivate your answer.

Type your answer here

***Question 3.4*:**

Assuming that you have trained a model and are satisfied with the generalization performance, how can you use the model to predict the class label y for an unknown test sample $\mathbf{x}$. (**Tip**: Remember that the model predicts a probability. How can this probability be converted to a binary class label?).

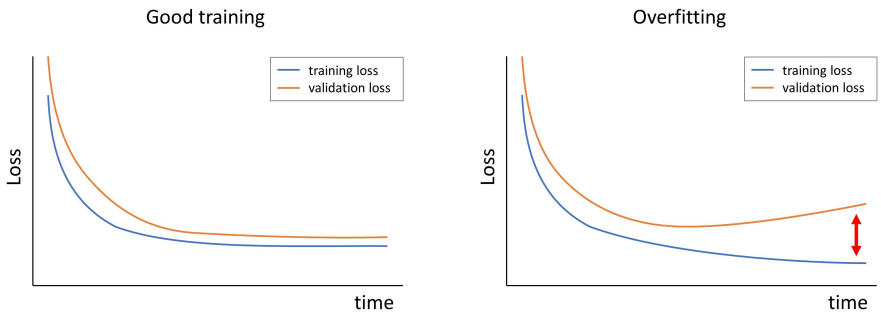
Type your answer here

1.9.4 4. Generalization and overfitting

Generalization and overfitting are crucial terms in machine learning algorithms. Generalization describes a model's ability to make accurate predictions based on new data previously absent from the training dataset. Model's generalization capability can be thought of as a success measure in making accurate predictions. However, if a model has been trained too well on training data, it will make inaccurate predictions on new data. The opposite holds as well. Underfitting can happen when a model has been trained insufficiently.

In practice, three datasets are used in deep learning research:

1. **Training set** - The training set is used for training the model (i.e. iteratively updating the network weights to minimize the error).
2. **Validation set** - At the inference phase, the validation set is used for two purposes:
    - Check for model overfitting: Sometimes the model is able to 'remember' all the training examples, which means that the model will not generalize well to unseen data at the inference phase. Overfitting can be detected by inspecting the training and validation loss over time. The loss function is often based on the error between the model prediction and the desired output, this means that you want to minimize the loss function. As you can see in the figure below, the training and validation loss show the same pattern when the model is not overfitting (left figure). When the model is overfitting to the training data, you see that the training and validation loss start to diverge after a certain number of epochs (right figure), this tells you that the model is not generalizing well to new data.



- Tuning of model parameters: Many parameters (e.g. number of layers, loss function, learning rate, etc.) influence the performance of the model for a specific task. The performance can often be measured with a quantitative metric because the desired output (ground truth) is known for the validation set. By systematically adapting model parameters and evaluating the performance, the optimal parameters can be chosen.

3. **Test set** - The test set is used to show the final performance of the model on an unseen set. This performance can give an indication of how the model will perform when it is implemented in for example the clinic (with the assumption that the test set resembles the real clinical data in terms of population and acquisition protocol).

1.10 Topic 2.3: Building blocks of neural networks

This notebook combines theory with exercises to support the understanding of fundamental building blocks of neural networks. Implement all functions in the `code` folder of your cloned repository, and test it in this notebook after implementation by importing your functions to this notebook. Use available markdown sections to fill in your answers to questions as you proceed through the notebook.

Contents:

1. Learning process of a neural network
2. Backpropagation
3. Implementation of a neural network

References:

[1] Deep feedforward networks: LINK

Nowadays most automated medical image analysis tasks are carried out using deep neural networks. These large networks are often seen as black box models, even though the outputs of the networks can 'directly' be calculated from the inputs. With the simple examples given in this notebook we aim for you to understand how neural networks learn. After you completed the exercises of this notebook you are able to:

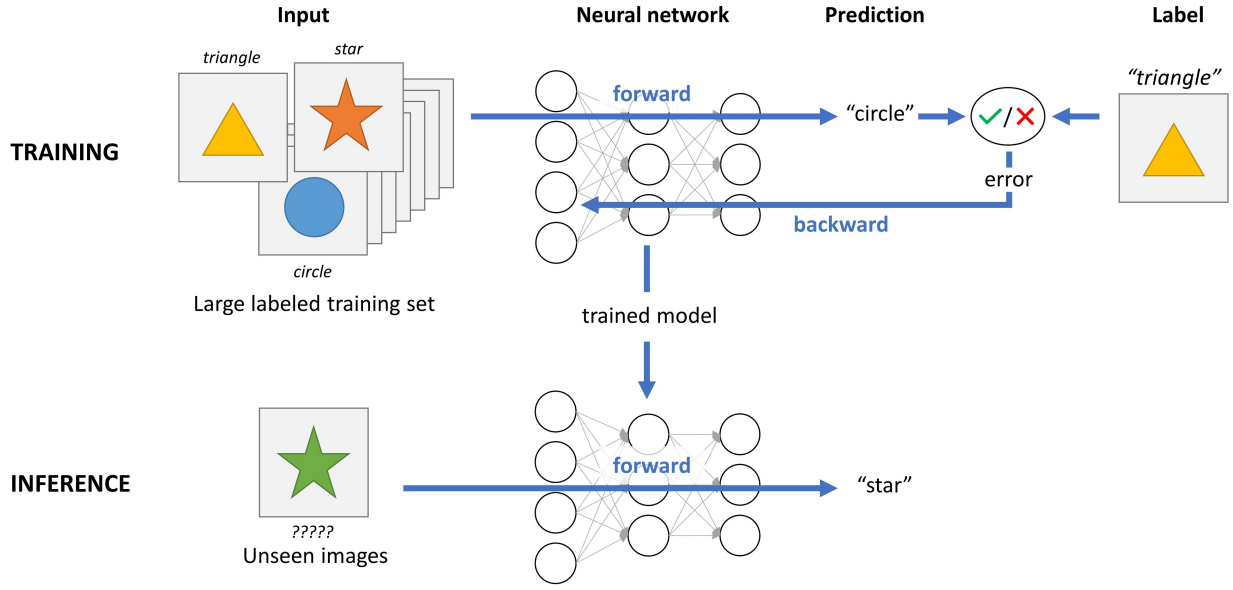
- explain the fundamental principles behind the learning process of a neural network.
- manually train a simple neural network by doing backpropagation.
- implement a small neural network in python that can be used for the CAD project work.

[1]:
```
%load_ext autoreload
%autoreload 2
```

1.10.1 1. Learning process of a neural network

Let's first understand how a neural network learns. As humans we are capable of learning many tasks throughout our lives. For example, we can easily distinguish cats from dogs in a picture, but we were not able to do this as newborns and we had to learn it along the way. In our upbringing, constant feedback is given by parents and teachers to ensure we can recognize different animals or objects. Eventually, you simply know which animal you observe by taking a quick look at the animal. However, when it comes to rare animals, our distinguishing skills are poor since we have not seen enough examples of these rare animals in our lives.

The working principle of a neural network is similar. During the training process, known data is fed into the neural network, and the network makes a prediction about what the data represents. Any error in the prediction is used as feedback. As the training process continues, the network weights are adjusted (using backpropagation) until the network starts to make accurate predictions. Then, the model can be used to make predictions for unseen images in the inference stage. The learning process is visualized in the figure below. As you can see, the model is only trained on three classes of images (triangles, stars and circles), therefore it will never be able to classify other shapes. However, the model is able to classify a green star as a star, because the training data consists of a large variety of colors, even though a star with this exact color is not seen during the training stage.



1.10.2 2. Backpropagation

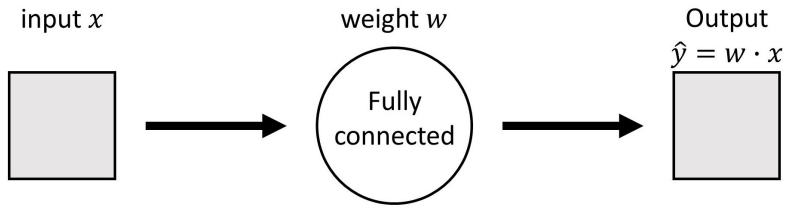
Now that you understand the idea of how a neural network can learn, let's have a look at the calculus behind the learning process. As mentioned before, you want to minimize the loss function, which is done by updating network weights in the backward pass. Backpropagation is carried out by taking small steps in the descending direction of the slope in the loss function. In this notebook we will walk through an example of backpropagation to understand what is really happening in the backward pass.

The following videos and book are suggested for a more in-depth explanation of backpropagation:

1. Video 1: What is backpropagation really doing?
2. Video 2: Backpropagation calculus
3. deeplearningbook.org - chapter 6

Simple neural network

The neural network that will be used for this backpropagation example is as follows:



This network is used to predict a value of $\hat{y}$, given the input x, where both x and y are scalars. This network contains only one fully connected layer (without a bias), therefore the output can be calculated as $\hat{y} = w \cdot x$, where w is the network weight. Keep in mind that normally a neural network has millions of weights, but just for the sake of manually carrying out backpropagation, we use a neural network of one single weight.

Training set

The model needs to be trained to obtain the optimal value for the weight w. Normally a large training set is used to find the optimal values for all the weights, but for simplification purposes, our training set consists of a single input-output pair, which is as follows:

Input (x)	Desired output (y)
1.5	0.5

Because this is such an easy example we know that the solution to this optimization problem is $w = \frac{y}{x} = \frac{0.5}{1.5} \approx 0.33$. However, normally neural networks are used for much more complex optimization problems with millions of parameters (weights) and many more training examples. Therefore, the best solution cannot just simply be calculated like this, and an iterative training approach is needed where the network weights are optimized one step at a time.

Model initialization

This optimization process predicts the output $\hat{y}$ given an input x and a weight w. The weight w is updated such that the predicted output $\hat{y}$ becomes more similar to y. To start this optimization process, the model weight w is initialized with a random value, let's say 0.8. We can now calculate the predicted value (after zero epochs, i.e. at initialization) of $\hat{y}$, given that $x = 1.5$ and $w = 0.8$:

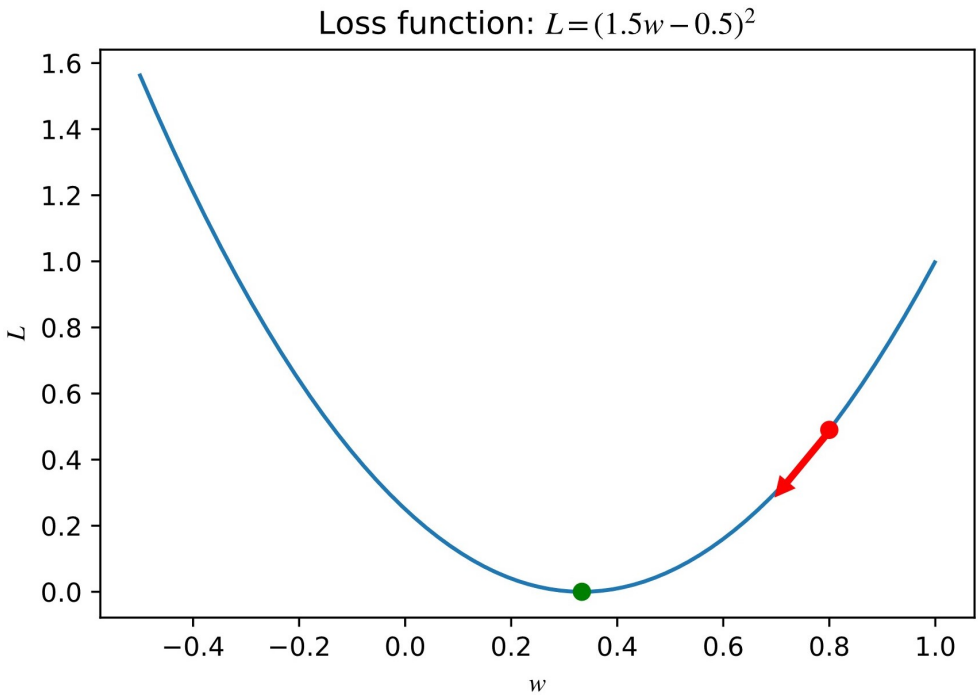
Epoch	Input (x)	Desired output (y)	Weight (w)	Predicted output ($\hat{y}$)
0 (init)	1.5	0.5	0.8	1.2

Training & loss function

Now the question is how the model needs to be trained such that the predicted output reaches the desired output of 0.5. For this training process, a loss function is defined. The model will try to minimize the value of the loss function, and therefore the loss function gives the model feedback on how the network weights should be updated. The loss function for this example is defined as the squared difference between the predicted and the desired output:

$$L = (\hat{y} - y)^2 \tag{1.38}$$

The loss function with respect to the weight is visualized for the given training pair in the following figure. It can be seen that the loss function is a parabola with a minimum around 0.33 (green dot), which is in line with the solution we calculated earlier.



You can see that for the current weight, $w = 0.8$ (red dot), the loss function is not at the minimum. The backpropagation algorithm seeks to minimize the loss by descending along the loss function (red arrow), which is called gradient descent. To take a descending step in the direction of the slope, the derivative of the loss function needs to be calculated.

***Exercise 2.1*:**

- Given the model $\hat{y} = wx$ and the loss function $L = (\hat{y} - y)^2$, find the derivative of the loss function with respect to the weight: $\frac{\partial L}{\partial w}$. (**Tip**: Use the chain rule: $\frac{\partial L}{\partial w} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w}$)
- Fill in the values of the training set: $x = 1.5$ and $y = 0.5$.

Learning rate

After calculating the derivative, the model weight is updated by taking a step along the slope. Therefore, a step size needs to be formulated with care. If the step size is too small, it will take many steps before the minimum is reached, or a more complex model can get stuck in a local minimum. But if the step size is too large, the minimum will not exactly be reached because the red dot 'bounces' around the minimum. This step size is usually called the learning rate. For this example, we take a learning rate (r) of 0.1.

Now the weight can be updated according to the gradient descent and the learning rate as follows:

$$w_{new} = w_{old} - r\frac{\partial L}{\partial w} \tag{1.39}$$

After one step (i.e. after one epoch, since we have a training set size of 1), the weight is updated to

$$w_{new} = 0.8 - 0.1\frac{\partial L}{\partial w}(x = 1.5, y = 0.5, w = 0.8) \approx 0.59, \tag{1.40}$$

Which means that the updated predicted value is: $\hat{y} = 0.59 \cdot 1.5 \approx 0.89$.

Exercise 2.2:

Calculate the weights and predicted outputs for the next epochs until the model converges (i.e. the weight is approximately 0.33). Fill in (and continue) the following table:

Epoch	Input (x)	Desired output (y)	Weight (w)	Predicted output ($\hat{y}$)
0 (init)	1.5	0.5	0.8	1.2
1	1.5	0.5	0.59	0.89
2	1.5	0.5		
3	1.5	0.5		
n	1.5	0.5		

Question 2.1:

After approximately how many epochs does the model converge?

Type your answer here

Question 2.2:

What is the reason for the fast convergence in the beginning of the training and the slow convergence later on, despite the fact that the step size (learning rate) is constant?

Type your answer here

Following model training

This process is normally done using python because the model is much more complex and then you want to follow the training process to know when the model is done training. To know this you can inspect the loss curve during training.

Exercise 2.3:

In the following python cell, the example model is implemented. To run the code, you need to **add an extra line** in the `model_training()` function in `SECTION 3` of the `cad_tests.py` module. Define in the formula of $\frac{\partial L}{\partial w}(w, x, y)$ which you obtained from the first part of Exercise 2.1. Once you have added the missing line, you can test it below. Inspect the training curve and check the results of exercise Exercise 2.1 with the table that is printed by this code cell.

[2]:
```
%matplotlib inline
import sys
import matplotlib.pyplot as plt
sys.path.append('../code')

from cad_tests import model_training

model_training()
```

```
Traceback (most recent call last):

  File ~/checkouts/readthedocs.org/user_builds/8dc00-mia-docs/envs/latest/lib/python3.8/
↪site-packages/IPython/core/interactiveshell.py:3508 in run_code
    exec(code_obj, self.user_global_ns, self.user_ns)

  Cell In[2], line 6
    from cad_tests import model_training
  File ../code/cad_tests.py:424
    def rotate_using_eigenvectors_test(X, Y, v):
    ^
IndentationError: expected an indented block
```

Question 2.3:

What do you think of the training process when looking at the loss plot?

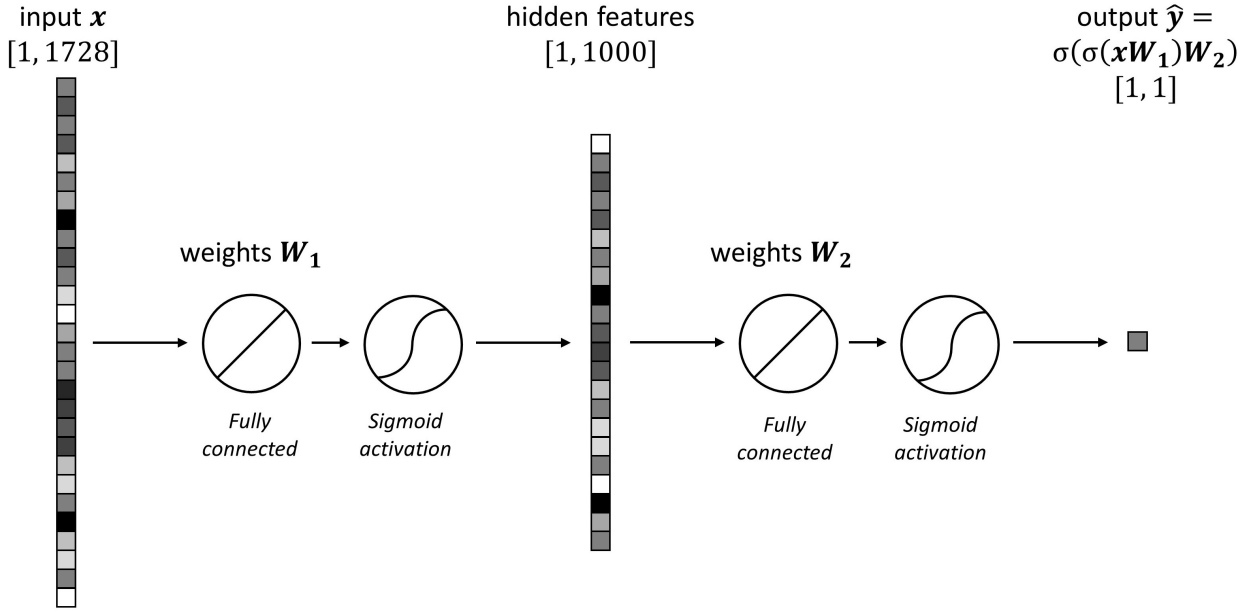
- Do you think the model was trained for enough epochs? Explain your answer.
- Do you think the step size of 0.1 was appropriate for the given model? Explain your answer.
- How could you in a real application (objectively) define when the model finished training?

Type your answer here

1.10.3 3. Neural network implementation

Project computer-aided diagnosis (CAD)

Before continuing with the exercises in this notebook, make sure that you have read the description of the *CAD project*, especially for the binary classification. You do not need to complete the project before starting the following exercises, but just read the description to understand the task. The description states that one image has $24 \times 24 \times 3 = 1728$ features, where the factor 3 comes from the RGB channels. Optimally, these features are used in the image space and captured with a convolutional neural network, but for simplification purposes and carrying out a numpy implementation we are still going to used these 1728 features flattened in a 1D vector. We will use the following neural network, consisting of two fully connected layers and their activation functions (sigmoid activation: σ).



As you can see, the number of features decrease every layer. They start at 1728, then go to 1000 and the model outputs only a single value for a given input. This output is a prediction on whether the nuclei is large (output is 1) or small (output is 0).

Data loading & preprocessing

To implement a neural network we need the training, validation and test set. In this implementation we will be using a supervised network, which means we also need labels (whether the nuclei should be classified as large or small). The following python cell loads the data and applies preprocessing.

***Exercise 3.1*:**

Run the following python cell, try to understand what is happening and inspect the example images. You will find the definition of the function `data_preprocessing()` in the `Training` class of `SECTION 3` of the `cad_tests.py` module.

[3]:
```
%reset_selective -f regex
import matplotlib.pyplot as plt

from cad_tests import Training
t = Training()

t.data_preprocessing()
```

```
Traceback (most recent call last):

  File ~/checkouts/readthedocs.org/user_builds/8dc00-mia-docs/envs/latest/lib/python3.8/
↪site-packages/IPython/core/interactiveshell.py:3508 in run_code
    exec(code_obj, self.user_global_ns, self.user_ns)

  Cell In[3], line 4
    from cad_tests import Training
  File ../code/cad_tests.py:424
    def rotate_using_eigenvectors_test(X, Y, v):
    ^
IndentationError: expected an indented block
```

Define initialization values

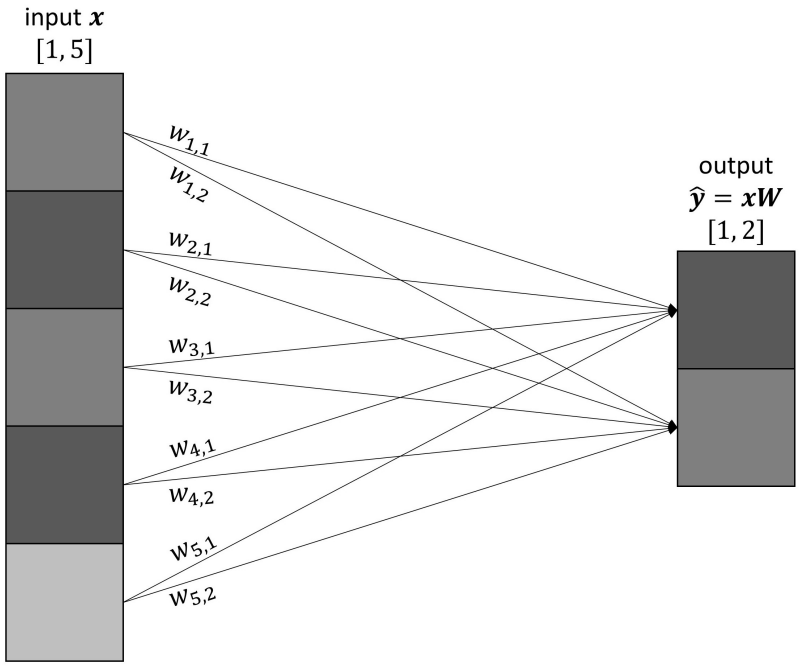
To train a model, several initialization variables should be defined.

Some values are fixed or are already chosen by us (such as the `learning_rate`, `batchsize` and the number of features in the input, hidden and output layer (`in_features`, `n_hidden_features`, `out_features`, respectively)). This means that the sizes of the weight matrices can be calculated.

?

Question 3.1:

To understand the size of the weight matrices, we show you a small example of a fully connected network, as can be seen in the following figure.



For this case, the input features $\mathbf{x}$ have the shape `[1,5]` and the output features $\hat{\mathbf{y}}$ have the shape `[1,2]`. We also know that $\hat{\mathbf{y}} = \mathbf{x}\mathbf{W}$.

- What is the shape of the matrix $\mathbf{W}$?
- What would happen with the shape of $\mathbf{W}$ if a batch of images is given to the network (i.e. shapes of $\mathbf{x}$ and $\hat{\mathbf{y}}$ are `[batchsize,5]` and `[batchsize,2]`?

Type your answer here

Exercise 3.2:

Let's go back to the original nuclei problem with the two layer network. Complete the `TODO` block of code in the `define_shapes()` function in the `Training` class of `SECTION 3` of the `cad_tests.py` module by defining the shapes of the weight matrices `w1` and `w2`. When complete, run the python cell below.

[4]:
```
t.define_shapes()
```

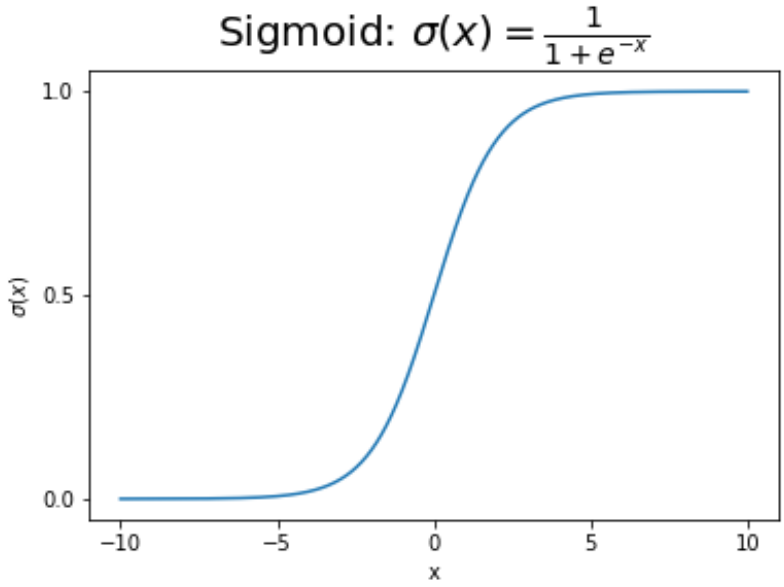
```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[4], line 1
----> 1 t.define_shapes()

NameError: name 't' is not defined
```

Model functions

Several functions need to be defined before implementing the model.

Activation function

As mentioned before, both fully connected layers are followed by a sigmoid activation function, which is defined as:



This activation is used to obtain an eventual output between zero and one, which indicates the probability of a nuclei being large. Such an activation layer simply maps the pixel intensities (x in the plot above) to another range, and no model weights are involved in an activaiton layer.

Loss function

The squared difference between the predicted output $\hat{\mathbf{y}}$ and the desired output (the label) $\mathbf{y}$ is again used for the loss function:

$$L = (\hat{\mathbf{y}} - \mathbf{y})^2 \tag{1.41}$$

Forward and backward pass

For training the model, we need to apply the forward and backward pass. In a forward pass the prediction $\hat{\mathbf{y}}$ is calculated for a given input $\mathbf{x}$ with the following steps (assume all hidden layers before or after activation $\mathbf{h1}$, $\mathbf{h2}$ and $\mathbf{h3}$):

$$\mathbf{h_1} = \mathbf{x}\mathbf{W_1} \tag{1.42}$$

$$\mathbf{h_2} = \sigma(\mathbf{h_1}) \tag{1.43}$$

$$\mathbf{h_3} = \mathbf{h_2}\mathbf{W_2} \tag{1.44}$$

$$\hat{\mathbf{y}} = \sigma(\mathbf{h_3}) \tag{1.45}$$

Subsequently the backward pass is carried out for updating the weights such that the loss function decreases. The derivatives with respect to the two weight matrices $\mathbf{W_1}$ and $\mathbf{W_2}$ are defined (by the chain rule) as follows:

$$\frac{\partial \mathbf{L}}{\partial \mathbf{W_1}} = \frac{\partial \mathbf{L}}{\partial \hat{\mathbf{y}}} \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{h_3}} \frac{\partial \mathbf{h_3}}{\partial \mathbf{h_2}} \frac{\partial \mathbf{h_2}}{\partial \mathbf{h_1}} \frac{\partial \mathbf{h_1}}{\partial \mathbf{W_1}} = 2(\hat{\mathbf{y}} - \mathbf{y}) \cdot \sigma\prime(\mathbf{h3}) \cdot \mathbf{W_2} \cdot \sigma\prime(\mathbf{h1}) \cdot \mathbf{x} \tag{1.46}$$

$$\frac{\partial \mathbf{L}}{\partial \mathbf{W_2}} = \frac{\partial \mathbf{L}}{\partial \hat{\mathbf{y}}} \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{h_3}} \frac{\partial \mathbf{h_3}}{\partial \mathbf{W_2}} = 2(\hat{\mathbf{y}} - \mathbf{y}) \cdot \sigma\prime(\mathbf{h3}) \cdot \mathbf{h_2} \tag{1.47}$$

Exercise 3.3:

The forward pass and the derivatives of the backward pass are implemented in the `forward()` and `backward` in `SECTION 3` of the `cad_tests.py` module, respectively. The model weights are not yet updated, and the new model weights need to be defined as a function of the old weights (`w1` and `w2`), `learning_rate`, and the derivatives (`dL_dw1` and `dL_dw2`). Complete the missing implementation. (**Tip:** Have a look at the first part of this notebook).

Model training & validation

The backpropagation example in the first part of this notebook is based on a single input-label training pair, but this implementation is based on a training set of 14607 image-label pairs. As discovered in Question 3.1, the number of examples given as input (in that case either 1 or `batchsize`) does not affect the size of the weight matrix. Therefore you can choose to show more image-label pairs at the same time such that less passes are needed to 'see' the entire training set.

For this model we choose a batchsize of 128, which means that 128 image-label pairs are given to the model at a time and then the model updates its weights according to the losses of these 128 pairs. Subsequently the next batch of images is given to the model and after batch several iterations the model completed a full epoch where the whole training set is seen once.

Normally, the model needs to train for many epochs to converge. The function `launch_training()` in `SECTION 3` of the `cad_tests.py` module implements a training for 100 epochs. As you can see, there are two for-loops: one for-loop for the epoch and one for-loop for the batch. After every epoch is completed, the validation set is fed through the network (all at once) to calculate the validation loss and the model performance (accuracy) over time.

Exercise 3.4:

Inspect the `launch_training()` function of the `Training` class in more detail, and run the following code cell. **NOTE**: It takes some time to do all the calculations.

[5]:
```
import sys
sys.path.append('../code')

t.launch_training()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[5], line 4
      1 import sys
      2 sys.path.append('../code')
----> 4 t.launch_training()

NameError: name 't' is not defined
```

Question 3.2:

Inspect the results of the training.

- Do you think the model is suitable for this task, and that it is trained well?
- What is remarkable about the loss curves? What is this phenomena called?
- Could you think of solutions for reducing the gap between training and validation loss?

Type your answer here

Final model performance

Inspecting the model's training and performance on the validation set is useful for making adaptations to your model to eventually obtain the best model for this task. After the model is fully optimized, you want to present you final performance on a test set that has not yet been used in the optimization process. In the following Python cell, the complete test set is fed through the network and the final test accuracy is given. All test predictions are visualized in a histogram with the color of the given label.

Question 3.3:

Run the following cell below.

- If you completed the entire code correctly a test accuracy of 0.76 is given, did you obtain the same result?
- What is noticeable about the shown histogram?

[6]:
```
t.pass_on_test_set()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[6], line 1
----> 1 t.pass_on_test_set()

NameError: name 't' is not defined
```

Type your answer here

1.11 Topic 2.4: Unsupervised learning, PCA

This notebook combines theory with code exercises to support the understanding of (un)supervised machine learning and principal component analysis in computer-aided diagnosis. **Please note:** Cells in this notebook must be executed *in order* as the code often relies on cells run above it!

Contents:

1. Supervised vs. unsupervised learning
   - Examples of unsupervised machine learning methods
2. Principal component analysis (theory)
   - 2.1 Motivation
   - 2.2 Basics of PCA
   - 2.3 Dimensionality reduction using PCA
   - 2.4 Intuitive interpretations and principal components
3. Principal component analysis (exercises)

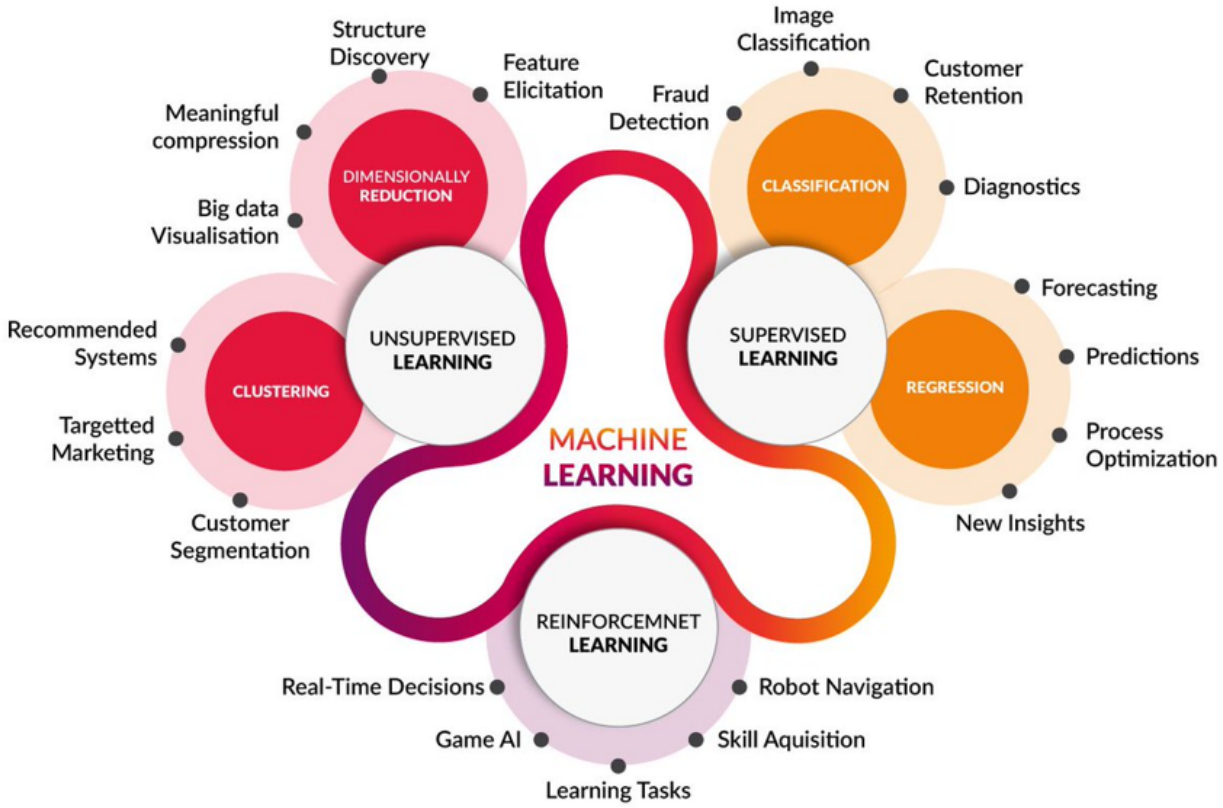
References:

[1] Principal component analysis: Jolliffe, I. T., Cadima J. PCA: a review and recent developments (2016)

[1]:
```
%load_ext autoreload
%autoreload 2
```

1.11.1 1. Supervised vs. unsupervised learning

While supervised machine learning (e.g. classification or regression) serves to develop a predictive model based on input and output data (i.e. ground truth, labels), unsupervised machine learning (e.g. clustering or dimensionality reduction) utilizes machine learning algorithms to cluster unlabeled data by finding hidden patterns without the need for manual human intervention.



Supervised learning requires a lot of known transformations for training and it is necessary to use ground truth (labels) to calculate the loss. Supervised machine learning methods can either be *fully supervised* (e.g. 2D FlowNet or 3D U-net, mostly patch-based; see generation of ground truth transformations and large deformation diffeomorphic metric mapping - LDDMM) or *weakly supervised*, utilizing overlap between segmentations or a similarity metric combined with ground truth (e.g. registration between ultrasound and MRI images using convolutional neural networks - CNNs or generative adversarial networks - GANs).

Despite its relative computational complexity due to a high volume of training data, unsupervised learning is widely used in computer vision tasks (for visual perception and object recognition) as well as in medical imaging to guide radiologists and pathologists towards accurate diagnoses. There is no need for ground truth (labels); instead, unsupervised methods often use a spatial transformer layer. A spatial transformer is a learnable module that explicitly allows for spatial manipulation of data within a given network. They are differentiable, modularizable for existing neural networks and serve for active transformation of feature maps.

Examples of unsupervised machine learning methods

1. Variational autoencoders
2. Generative adversarial networks (GANs)
3. Multi-scale methods:

   3.1 RegNet

   3.2 ConvNet

   3.3 pgCNN

   3.4 HoVer-Net
4. VoxelMorph (U-net)

4.1 Cycle-consistent VoxelMorph

1.11.2 2. Principal component analysis (theory)

This section is meant to teach you about **principal component analysis** (PCA), a technique that is commonly used in medical image analysis to reduce the dimensionality of data (for example, features for a classification or segmentation task).

We will begin with some motivation *why* PCA (and dimensionality reduction in general) is useful. Next, we will show PCA applied to a simple Gaussian dataset, and then to the nuclei dataset.

2.1 Motivation

The ultimate goal of any classification/segmentation technique is to use training data to make accurate predictions about future, unlabeled data. In other words, the goal is to *generalize* well to new data. This can be achieved by many different choices for classifiers and many different combinations of features. For example, when tasked with predicting the systolic blood pressure of a patient given some clinical data, we could use many different clinical values: the low-density lipoprotein (LDL) blood levels of the patient, their high-level lipoprotein (HDL), but also their age, weight and height.

Choosing which combination of features to use for a classifier is a non-trivial task, for a variety of reasons:

- For most complex tasks, it is inherently unclear what number of features is 'enough'.
- It may not always be intuitive what features are actually discriminatory between your classes.
- Increasing the number of features may actually decrease the performance of the classifier.

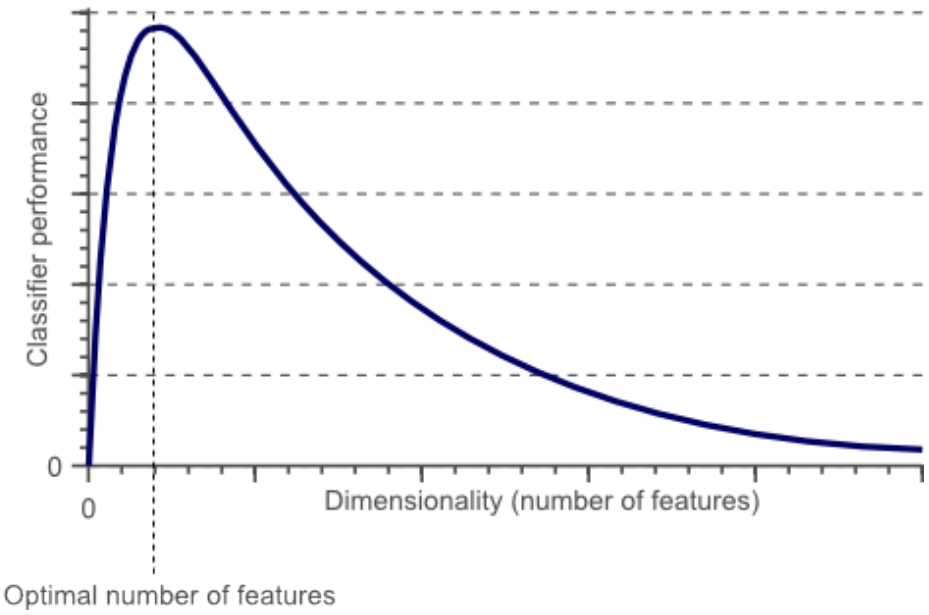


Figure 2.1.1: As dimensionality (the number of features) increases, a classifier's performance can increase until some optimum. Further increasing the dimensionality for the same amount of training samples can result in a decrease in classifier performance.

Curse of dimensionality

It is worth elaborating on the last point: increasing the number of features may inadvertently lower classifier performance. This is because of something called the *curse of dimensionality*. We can illustrate the curse of dimensionality with a simple example, where we try to classify pictures of dogs from cats using a linear classifier.

In the image below, we show how the feature space of the classifier grows by adding one new feature (dimension) at a time. You can image that we might begin by classifying dogs from cats by their color, but we find that this doesn't offer a linear separability of our dataset (there is no one line we can draw that perfectly separates the two animals). We see the same for two features (e.g. color and size). However, when we use three features (e.g. color, size and weight), we *can* linearly separate the two classes, thus giving us perfect classification accuracy!

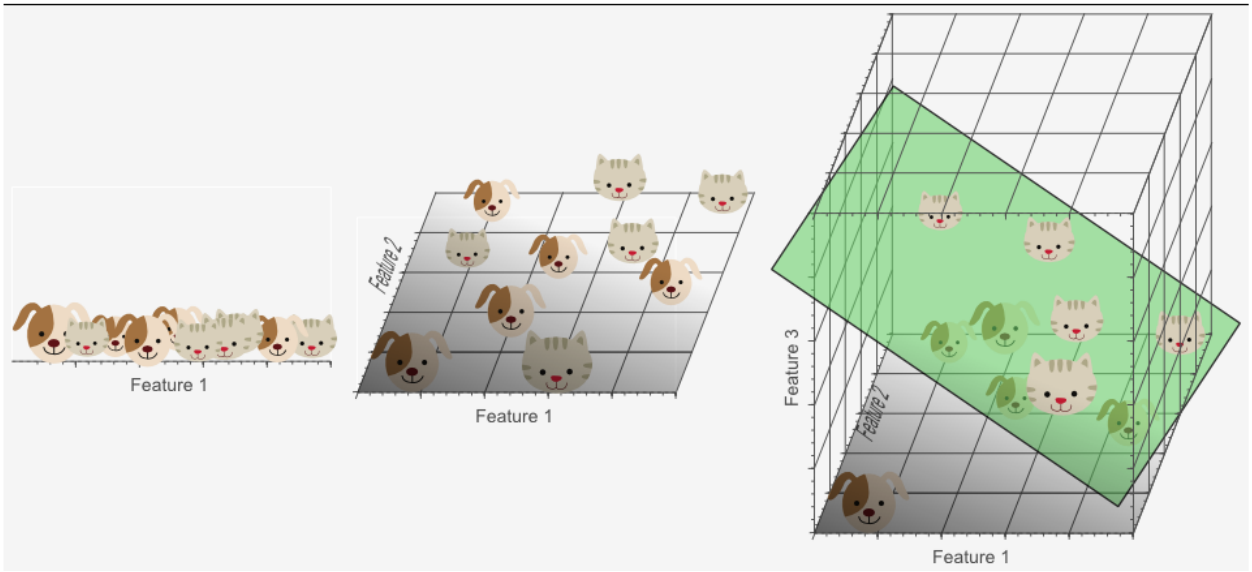


Figure 2.1.2: The more features we use, the higher the likelihood we can separate the classes with a hyperplane.

While it looks like adding more features has improved classification performance, **this is actually not the case**. Consider the 'density' of all of the training samples (represented by cat/dog icons) within the feature space: as the number of dimensions goes up, the feature space becomes increasingly sparse. As you might intuitively feel, it is much easier to find a hyperplane that splits two classes perfectly in a sparse feature space than in a very densely filled feature space - there are simply far less constraints on what shape the hyperplane needs to take on. In fact, if you were to take an infinite amount of features, the probability of a training sample laying on the wrong side of a hyperplane would become infinitely small and your classification performance on the training samples would be perfect.

In this way, the curse of dimensionality causes **overfitting** on the training set: the learned hyperplane does not reflect actual real-world differences between cats and dogs (the test set), but instead reflects the appearance of individual cats and dogs that just happen to appear in our training set. In this instance, a much better choice would be to do linear classification on two features: while performance on the training set would be lower, the model would *generalize* much better to the new, unseen data of the test set.

2.2 Basics of PCA

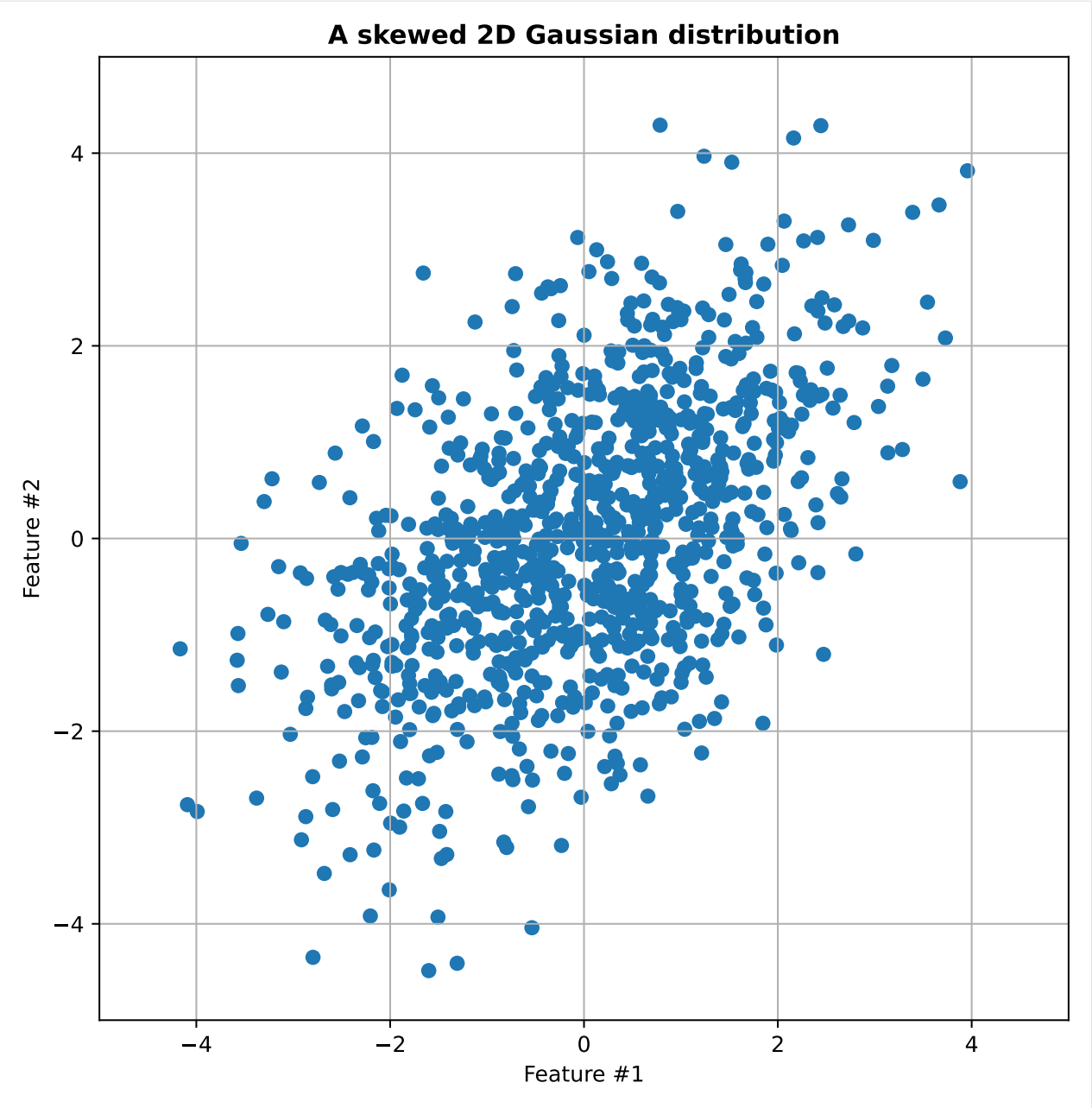
After the cat/dog example, we are left with an obvious question: "if we shouldn't use too many features due to the curse of dimensionality, which features *should* we use?" PCA is often used to determine which features are most suitable for a classification problem. We begin by illustrating the basic principles of PCA on a Gaussian dataset.

[2]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import sys

def plot_data(X, title = None, xlabel = None, ylabel = None, ax = None):

    # If no pre-existing axis given, make new plot
    if ax is None:
        fig = plt.figure(figsize = (8,8))
        ax = fig.add_subplot(111)
        ax.grid()

    # Plot and format figure
    ax.scatter(X[:,0], X[:,1])
    ax.set_xlim(-5, 5)
    ax.set_ylim(-5, 5)
    ax.set_title(title, fontweight = 'bold')
    ax.set_xlabel(xlabel)
    ax.set_ylabel(ylabel)

def draw_vector(start, end, ax = None):
    ax = ax or plt.gca()
    arrowprops = dict(arrowstyle = '->',
                      linewidth = 4,
                      shrinkA = 0, shrinkB = 0)
    ax.annotate('', end, start, arrowprops = arrowprops) # Draw vector from coords␣
↪(start) to (end)
```

Our dataset is defined as an M-by-2 matrix $\mathbf{X}$ containing M points sampled from a two dimensional Gaussian $\mathcal{N}$ with $\mu_1 = 0, \mu_2 = 0$

and a covariance matrix $\mathbf{\Sigma} = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$. Because of the covariance matrix, there is a positive skew upwards when plotting the data.

[3]:
```
n_samples = 1000
X = np.random.multivariate_normal([0,0], [[2,1], [1,2]], n_samples) # Make 2D gaussian␣
↪dataset
plot_data(X, "A skewed 2D Gaussian distribution", "Feature #1", "Feature #2")
```



PCA works by finding the 'principal components' of an N-dimensional dataset (here $N = 2$). One reasonable way to think of principal components is that they are the directions in which the dataset shows the most variation, i.e. the largest spread in values. Typically, these directions of large variance are the interesting parts of the dataset: imagine a dataset with 100 features, 98 of which barely have any spread, and 2 of which show large amounts of variance. You can intuitively imagine that these 2 features, because of their variance, must have some power to discriminate between classes.

Principal components are always orthogonal to one another and together form a new basis that we can use to transform the data. As a result of the transformation, the data will be linearly uncorrelated, meaning the covariance matrix will be diagonal: $\mathbf{\Sigma} = \begin{pmatrix} a & 0 \\ 0 & b \end{pmatrix}$, with $\{a, b\} \geq 0$. This also means that the directions of greatest variances are now aligned with the axes: the first coordinate is now the first principal component, the second coordinate is the second principal component, and so on.

Mathematical background

The process begins by centering the data X (M samples by 2 features) around the origin by subtracting the means of each variable from that column:

$$\hat{\mathbf{X}} = \mathbf{X} - \bar{\mathbf{X}} \tag{1.48}$$

We can find the principal components for the matrix $\hat{\mathbf{X}}$ by calculating its covariance matrix $\mathbf{\Sigma}$ and calculating the corresponding eigenvalues and eigenvectors of $\mathbf{\Sigma}$. The eigenvectors represent the principal components of the data and the eigenvalues represent the amount of variance explained by that principle component (proving that this is the case is outside of the scope of this course).

We can calculate the covariance matrix with the following formula:

$$\mathbf{\Sigma} = \frac{1}{M-1}\mathbf{X}^\mathbf{T}\mathbf{X} \tag{1.49}$$

The easiest way to then calculate the eigenvectors and eigenvalues is to factorize $\mathbf{\Sigma}$ using singular value decomposition (SVD). SVD gives us the following expression:

$$\mathbf{\Sigma} = \mathbf{UsV} \tag{1.50}$$

where $\mathbf{U}$ contains the eigenvectors $\mathbf{u}^{(i)}$ in the columns, ordered by largest to smallest variance:

$$\mathbf{U} = \begin{bmatrix} | & | & & | \\ \mathbf{u}^{(1)} & \mathbf{u}^{(2)} & \dots & \mathbf{u}^{(n)} \\ | & | & & | \end{bmatrix} \tag{1.51}$$

and $\mathbf{s}$ is a vector containing the eigenvalues λ_i:

$$\mathbf{s} = [\lambda_1, \lambda_2, \dots \lambda_n] \tag{1.52}$$

Now, we can simply multiple $\mathbf{U}$ with our data $\hat{\mathbf{X}}$ to transform it to the new basis:

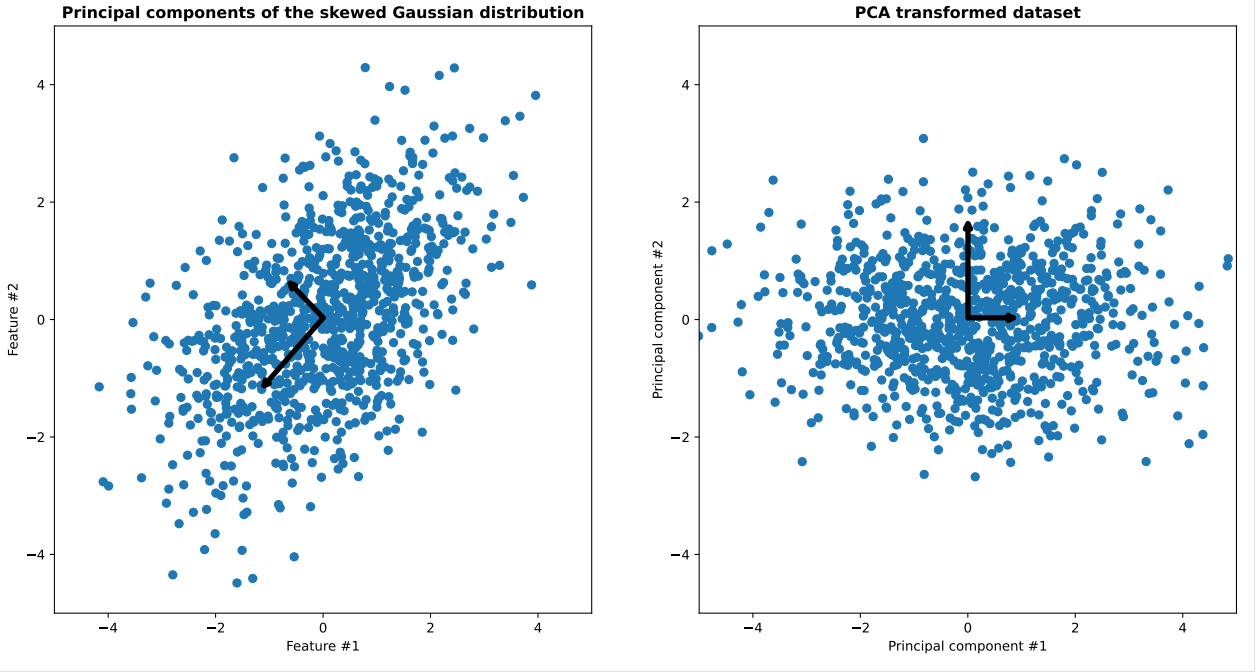
$$\mathbf{X}_{\text{PCA}} = \mathbf{U}^\mathbf{T}\hat{\mathbf{X}} \tag{1.53}$$

In code, this looks like the following:

[4]:
```
def pca_transform(X):
    n_samples = X.shape[0]

    X_mean = np.mean(X, axis = 0)

    X_hat = X - X_mean # Center data

    sigma_hat = 1/(n_samples-1)*X_hat.T.dot(X_hat) # Calculate covariance matrix,␣
↪alternative is np.cov(X)

    U, s, V = np.linalg.svd(sigma_hat) # Do singular value decomposition to get eigen␣
↪vector/values

    X_pca = U.dot(X_hat.T) # Transform dataset using eigenvectors

    return X_pca.T, U, s

X_pca, U, s = pca_transform(X)
```

We plot the original dataset and the PCA transformed dataset side by side. Superimposed on the plots, we show the principal components, before and after rotation. Note how the vectors point in the direction of the most variance.

[5]:
```
fig, axes = plt.subplots(1,2, figsize = (16,8))

X_mean = np.mean(X, axis = 0)

# We plot vectors from (X_mean) - i.e. the data center - to (X _mean +␣
↪sqrt(eigenvalue)*eigenvector)
# We use sqrt(eigenvalue) as a scaling factor to show the relative "importance" of the␣
↪eigenvectors
# But note that this has no semantic meaning/significance!

plot_data(X, "Principal components of the skewed Gaussian distribution", "Feature #1",
↪"Feature #2", axes[0])
draw_vector(X_mean, X_mean + np.sqrt(s[0])*U[:,0], axes[0])
draw_vector(X_mean, X_mean + np.sqrt(s[1])*U[:,1], axes[0])

plot_data(X_pca, "PCA transformed dataset", "Principal component #1", "Principal␣
↪component #2", axes[1])
draw_vector(X_mean, X_mean + np.sqrt(s[0])*np.array([0,1]), axes[1])
draw_vector(X_mean, X_mean + np.sqrt(s[1])*np.array([1,0]), axes[1])
```



2.3 Dimensionality reduction using PCA

In the previous section, we showed the basic principles of PCA. However, dimensionality reduction is obviously not very important if you only have two features. PCA becomes much more useful for datasets where there are too many features to plot in a human understandable way. In that case, we can select only a subset of the first n eigenvectors of matrix $\mathbf{U}$ (principal components) to do our transformation with. This means we project the data onto fewer axes and get a lower dimensional dataset! For example, if we have a 100-dimensional dataset and we choose $n = 10$, we only take the first 10 columns of $\mathbf{U}$ and use this $\mathbf{U}_{\text{reduced}}$ to project 100 dimensions onto 10 dimensions.

We usually decide on the number n as follows: we ask *how many principal components we need to retain :math:`geq 95% ` of the dataset's variance.* To reiterate: in dimensionality reduction, we care about retaining as much as the data's variance as possible, while using as little dimensions as possible.

We can describe 'retained variance' with the eigenvalues of the principal components: for n vectors and a k-dimensional dataset, the retained variance r is:

$$r = \sum_{i=1}^{n} \lambda_i \Big/ \sum_{i=1}^{K} \lambda_i \tag{1.54}$$

In other words, we divide the variance that the first n principal components retain by the total variance of all principal components. The variance of a principal component is represented by its eigenvalue.

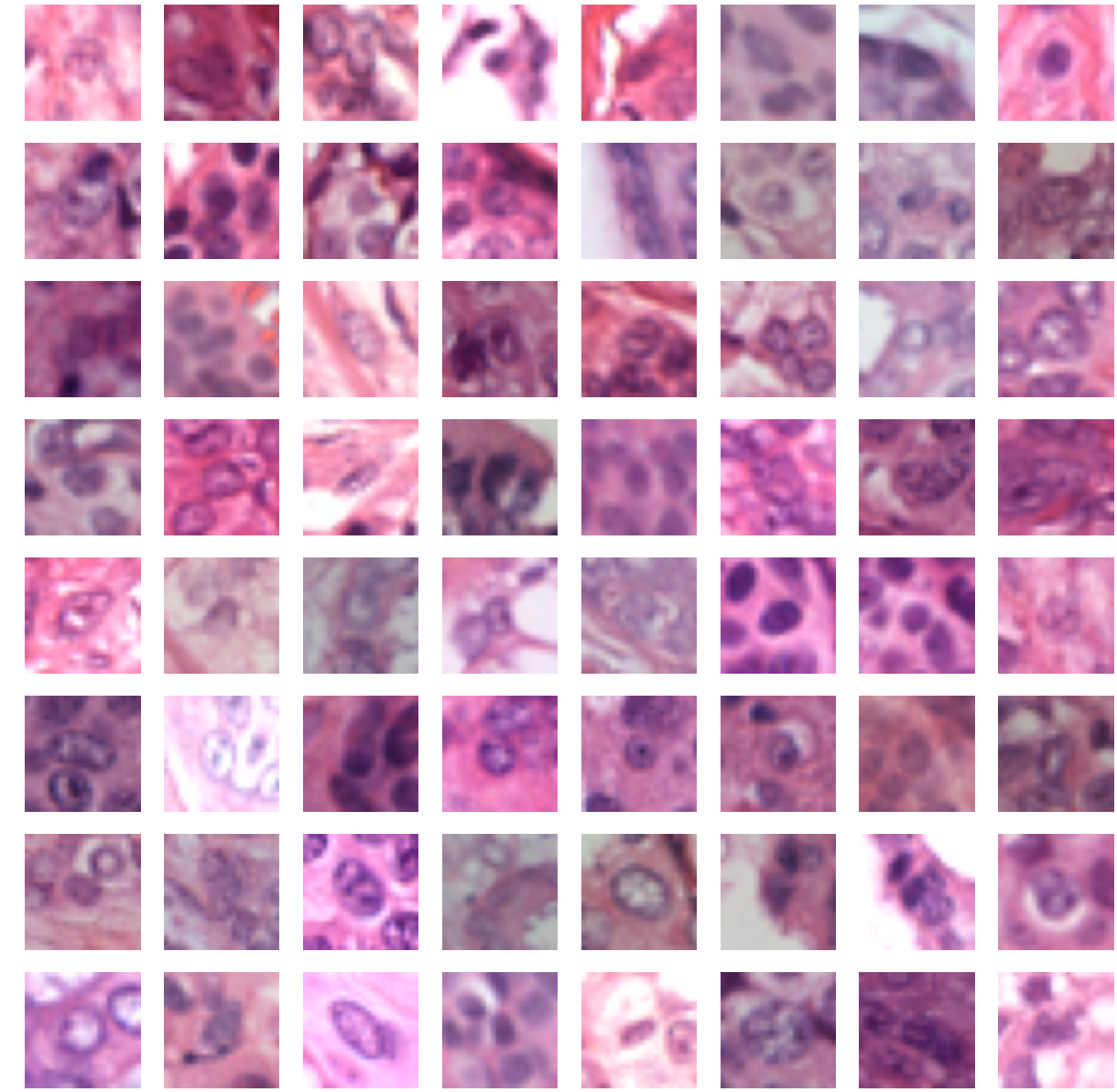
A graphical example: cell nuclei

Let's give an example using a high dimensional data set: we use the cell nuclei dataset from the CAD project.

[6]:
```
from scipy.io import loadmat

def plot_series(images, shape = (5,5), stochastic = True):
    n,m = shape
    ix = np.random.randint(0, images.shape[-1], n*m) if stochastic else np.arange(0, n*m)

    fig, axs = plt.subplots(n, m, figsize=(n, m))
    axs = axs.ravel()
    for i, j in zip(range(n*m), ix):
        axs[i].imshow(images[:,:,:,j])
        axs[i].axis('off');

    return axs
```

[7]:
```
# Load dataset
fn = '../data/nuclei_data.mat'
mat = loadmat(fn)

images = mat["training_images"]       # shape (24, 24, 3, 21910)
images_y = mat["training_y"]          # shape (21910, 1)
```

[8]:
```
# Visualize - randomized, so refresh to see new nuclei!
axs = plot_series(images, shape = (8,8))
```



In the CAD project, you must perform linear regression on the raw pixel values of the images to predict the surface area of nuclei. Here, we don't do regression, but purely investigate how many principal components we need to properly represent this high dimensional dataset, for the purposes of dimensional reduction. Because the images are of shape $(24, 24, 3)$, we get a total of $24 \times 24 \times 3 = 1728$ features per image. As our training set consists of 21910 images, we will have a $(21910, 1728)$ shaped dataset $\mathbf{X}$.

[9]:
```
shapes = images.shape
num_features = shapes[0]*shapes[1]*shapes[2]
X = images.reshape(num_features, shapes[3]).T.astype(float)
print("The shape of our dataset X is: ", X.shape)
```

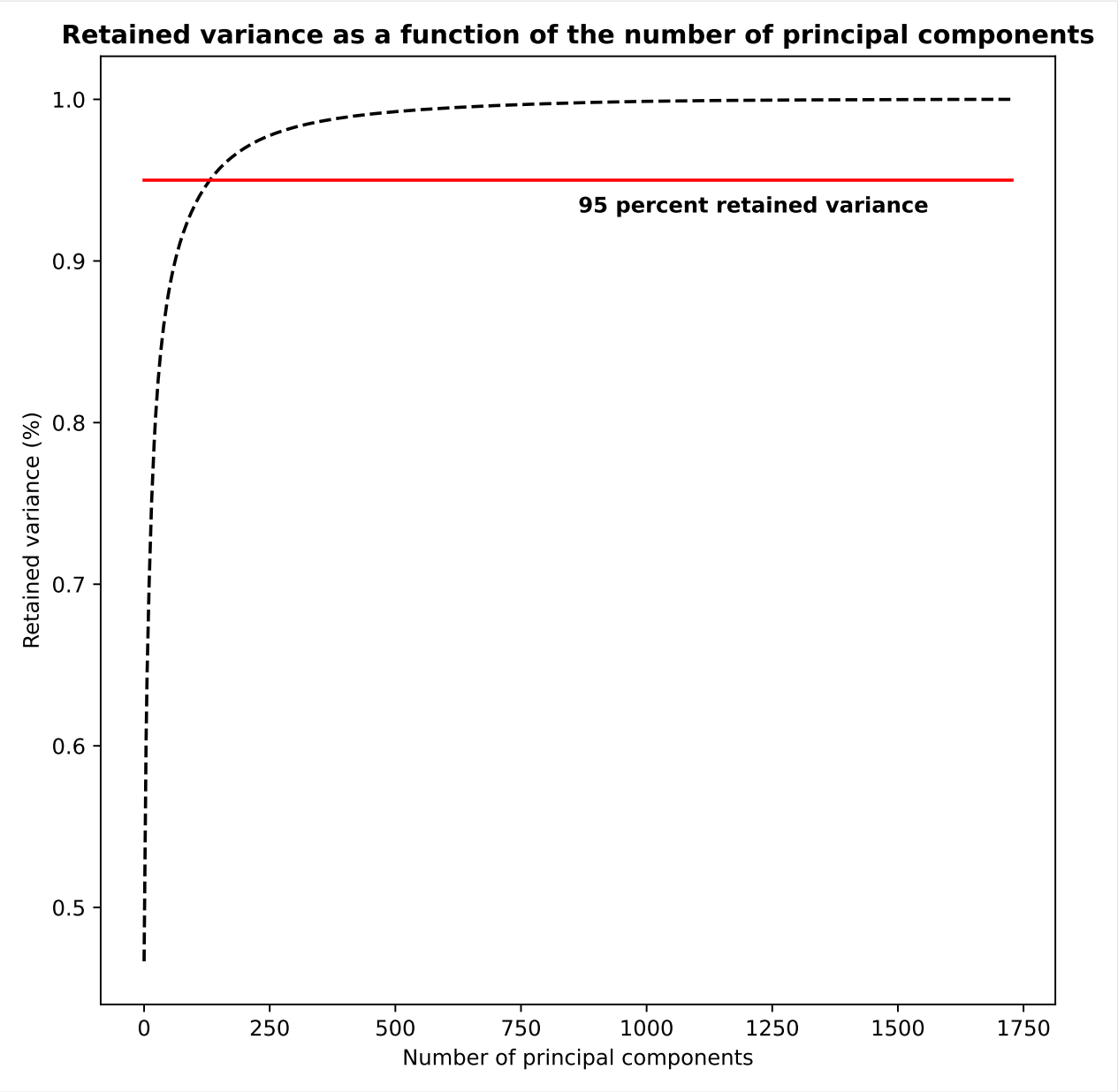
```
The shape of our dataset X is:  (21910, 1728)
```

Let's perform PCA on our dataset and see how many principal components we need to retain 95% of the data variance.

[10]:
```
X_pca, U, s = pca_transform(X)

# Every iteration i, divide variance retained in s[0:i] by total variance
r = np.array([s[0:i].sum() / np.sum(s) for i in range(1, len(s)+1)])
n = np.argmax(r > 0.95)
print("The number of principal components needed to retain 95 percent data variance is:
↪", n+1)
```

```
The number of principal components needed to retain 95 percent data variance is:  132
```

[11]:
```
plt.figure(figsize = (8,8))
plt.plot(r, 'k--')
plt.plot(range(len(r)), np.ones(len(r)) * 0.95, 'r-')
plt.xlabel("Number of principal components")
plt.ylabel("Retained variance (%)")
plt.text(1728/2, 0.93, "95 percent retained variance", fontweight = "bold")
plt.title("Retained variance as a function of the number of principal components",␣
↪fontweight = 'bold');
```



Remarkably, we only need 132 principal components (dimensions) to retain 95 percent of our data variance, even for a 1728 dimensional dataset. We can now easily reduce the dimensionality of our dataset by transforming the data with the first 132 eigenvectors. In a real scenario, we could now use this transformed dataset to perform regression (and also classification) techniques on. In general, because of the lower dimensionality, techniques will be less likely to overfit.

[12]:
```
X_hat = X - np.mean(X, axis = 0)
U_reduced = U[0:132]
X_pca = U_reduced.dot(X_hat.T).T # Our new, transformed (and lower dimensional) dataset
print("The new shape of our data matrix now is: ", X_pca.shape)
```

```
The new shape of our data matrix now is:  (21910, 132)
```

2.4 Intuitive interpretations of principle components

Lastly, we show two additional intuitive interpretations of principle components.

We have also established that principle components can be thought of the orthogonal directions of variation in our dataset. However, beyond just 'direction', we can also visualize this variation in the form of an image! Consider the fact that the eigenvectors are of the shape $(1, 1728)$ and can thus be reshaped back into the original shape of the image, $(24, 24, 3)$. In this form, the eigenvectors represent the *principle modes of variation* in the image.

We visualize this with the code in the cell below. Per eigenvector, we reshape the vectors to images (and rescale the values to [0,255]). The first eigenvector (top left) is a disk, which makes sense: since our dataset depicts nuclei, typically round, centered objects, the most variation exists in that form. As we go further right and down, this variation becomes more abstract.

[13]:
```
# From here, we use the sklearn PCA function, it's much better optimized than our own␣
↪code.
sys.path.append("../code")
from sklearn.decomposition import PCA
from cad_PCA import reconstruction_demo, reshape_and_rescale

# Rescaling is necessary because the eigenvectors are not in [0,255] domain
reconstructed_ims = np.stack([reshape_and_rescale(U[:,i]) for i in range(U.shape[-1])],␣
↪axis = -1)
axes = plot_series(reconstructed_ims, shape = (5,5), stochastic = False)
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
Cell In[13], line 3
      1 # From here, we use the sklearn PCA function, it's much better optimized than␣
↪our own code.
      2 sys.path.append("../code")
----> 3 from sklearn.decomposition import PCA
      4 from cad_PCA import reconstruction_demo, reshape_and_rescale
      6 # Rescaling is necessary because the eigenvectors are not in [0,255] domain

ModuleNotFoundError: No module named 'sklearn'
```

Second of all, we can use the eigenvectors in the matrix $\mathbf{U}$ to reconstruct our images from the transformed dataset $\mathbf{X}_{\text{PCA}}$. We simply multiply the data with $\mathbf{U}$ again, because:

$$\mathbf{X}_{\text{PCA}} = \mathbf{U}^{\mathbf{T}}\hat{\mathbf{X}}\mathbf{X}_{\text{rec}} = \mathbf{U}\mathbf{X}_{\text{PCA}} = \mathbf{U}\mathbf{U}^{\mathbf{T}}\hat{\mathbf{X}} = \mathbf{I}\hat{\mathbf{X}} \tag{1.55}$$

Then we just have to add the mean back to the data ("uncentering" it), and we get back our original dataset $\mathbf{X}$.

$$\mathbf{X} = \mathbf{X}_{\text{rec}} + \bar{\mathbf{X}} \tag{1.56}$$

Just like with the eigenvectors, we can reshape the individual rows of the dataset $\mathbf{X}$ back into an image. However, we can also reconstruct and reshape the images with only a subset of the eigenvectors, so with $\mathbf{U}_{\text{reduced}}$. In the demo below, we show the resulting images for the first 200 eigenvectors.

[14]:
```
pca = PCA()
pca.fit(X)
reconstruction_demo(X, pca)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
```

(continues on next page)

(continued from previous page)

```
Cell In[14], line 1
----> 1 pca = PCA()
      2 pca.fit(X)
      3 reconstruction_demo(X, pca)

NameError: name 'PCA' is not defined
```

As you can see, as the number of principle components used to reconstruct the images with increases, the quality of the images becomes better. This is because we able to utilize more information of the dataset in the reconstruction. In effect, we observe that each principle component is a linear combination of all features of our dataset.

1.11.3 3. Principal component analysis (exercises)

Exercise 3.1:

Use the following:

```
generate_gaussian_data(100, [0, 0], [0, 0], [[3, 1],[1, 1]], [[3, 1],[1, 1]])
```

to generate a dataset with correlated features. Calculate the mean and covariance matrix of the data using `mean` and `cov` and compare them to the parameters you used as input. Write your implementation in the `covariance_matrix_test()` function in `SECTION 4` of the `cad_tests.py` module.

[15]:
```
%matplotlib inline
import sys
sys.path.append("../code")
from cad_tests import covariance_matrix_test
X, Y, sigma = covariance_matrix_test()
```

```
Traceback (most recent call last):

  File ~/checkouts/readthedocs.org/user_builds/8dc00-mia-docs/envs/latest/lib/python3.8/
↪site-packages/IPython/core/interactiveshell.py:3508 in run_code
    exec(code_obj, self.user_global_ns, self.user_ns)

  Cell In[15], line 4
    from cad_tests import covariance_matrix_test
  File ../code/cad_tests.py:424
    def rotate_using_eigenvectors_test(X, Y, v):
    ^
IndentationError: expected an indented block
```

Question 3.1:

Is there a difference? How could you increase or decrease this difference?

Type your answer here

Exercise 3.2:

Compute the eigenvectors and eigenvalues of the covariance matrix using:

```
w, v = np.linalg.eig(cov)
```

(the column `v[:,i]` is the eigenvector corresponding to the eigenvalue `w[i]`).

Inspect the eigenvectors and eigenvalues. What two properties can you name about the eigenvectors? How can you verify these properties (describe the operations, or give a line of Python code). For the eigenvalues, which eigenvalue is the largest and which is the smallest?

You can sort the eigenvalues and eigenvectors as follows:

```
ix = np.argsort(w)[::-1] #Find ordering of eigenvalues
w = w[ix] #Reorder eigenvalues
v = v[:, ix] #Reorder eigenvectors
```

Write your implementation in `eigen_vecval_test()` in `SECTION 4` of the `cad_tests.py` module.

[16]:
```
%matplotlib inline
import sys
sys.path.append("../code")
from cad_tests import eigen_vecval_test
v, w = eigen_vecval_test(sigma)
```

```
Traceback (most recent call last):

  File ~/checkouts/readthedocs.org/user_builds/8dc00-mia-docs/envs/latest/lib/python3.8/
↪site-packages/IPython/core/interactiveshell.py:3508 in run_code
    exec(code_obj, self.user_global_ns, self.user_ns)

  Cell In[16], line 4
    from cad_tests import eigen_vecval_test
  File ../code/cad_tests.py:424
    def rotate_using_eigenvectors_test(X, Y, v):
    ^
IndentationError: expected an indented block
```

Exercise 3.3:

Rotate the data using `v` in the function `rotate_using_eigenvectors_test()` in `SECTION 4` of the `cad_tests.py` module. This is similar to what you did in the registration project, only now instead of getting the angle of rotation, `v` is already the rotation matrix.

[17]:
```
%matplotlib inline
import sys
sys.path.append("../code")
from cad_tests import rotate_using_eigenvectors_test
X_rotated = rotate_using_eigenvectors_test(X, Y, v)
```

```
Traceback (most recent call last):

  File ~/checkouts/readthedocs.org/user_builds/8dc00-mia-docs/envs/latest/lib/python3.8/
↪site-packages/IPython/core/interactiveshell.py:3508 in run_code
    exec(code_obj, self.user_global_ns, self.user_ns)

  Cell In[17], line 4
    from cad_tests import rotate_using_eigenvectors_test
  File ../code/cad_tests.py:424
    def rotate_using_eigenvectors_test(X, Y, v):
    ^
IndentationError: expected an indented block
```

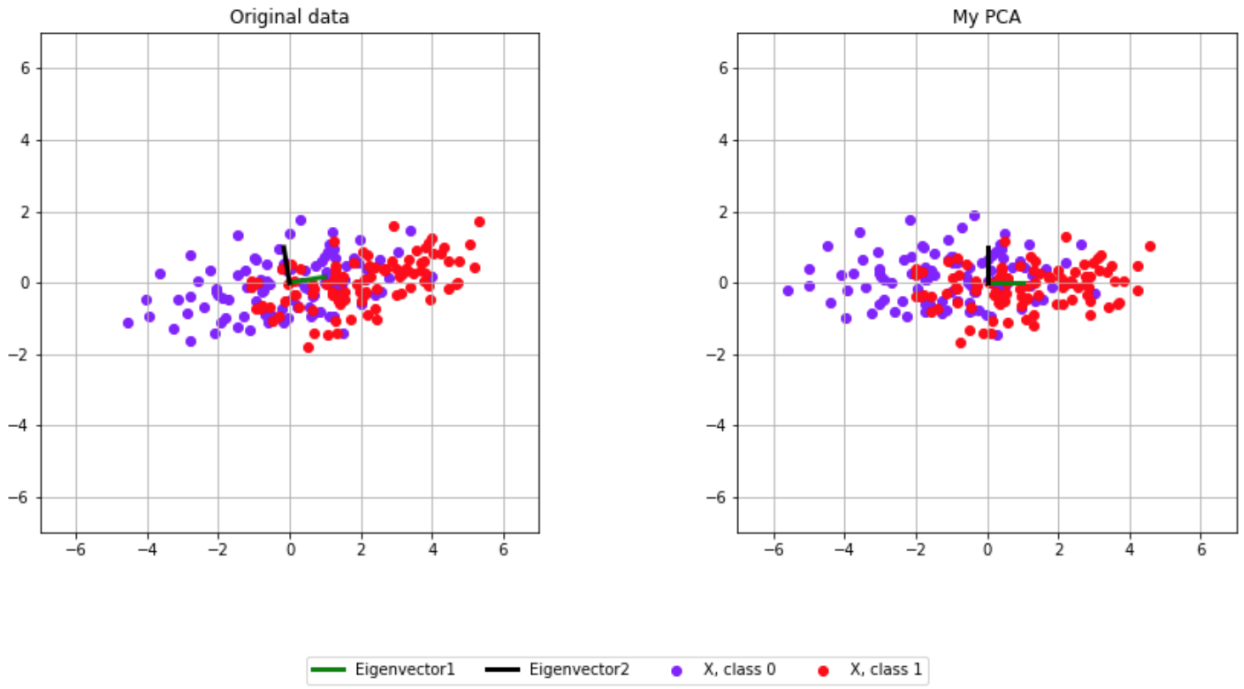
Question 3.2:

In most literature you will see the notation $v^T * X$, but this will not work on our dataset because of how the dataset is defined (rows = samples, columns = dimensions). Instead use $X_{pca} = v^T * X^T$ and $X_{pca} = X_{pca}^T$. What can you say about the covariance matrix of `Xpca`?

Type your answer here

Exercise 3.4:

Complete the missing functionality in the function `mypca()` in `SECTION 2` of the `cad.py` module. Test the function by running the `test_mypca()` script located in `SECTION 4` of the `cad_tests.py` module. This will plot the original data, and the data after `test_mypca()` is applied. Here is how the result might look:



[18]:
```
%matplotlib inline
import sys
sys.path.append("../code")
from cad_tests import test_mypca

test_mypca()
```

```
Traceback (most recent call last):

  File ~/checkouts/readthedocs.org/user_builds/8dc00-mia-docs/envs/latest/lib/python3.8/
↪site-packages/IPython/core/interactiveshell.py:3508 in run_code
    exec(code_obj, self.user_global_ns, self.user_ns)

  Cell In[18], line 4
    from cad_tests import test_mypca
  File ../code/cad_tests.py:424
    def rotate_using_eigenvectors_test(X, Y, v):
    ^
IndentationError: expected an indented block
```

?

***Question 3.3*:**

You might have noticed when editing `mypca()` that there is an additional output, `fraction_variance`. This vector stores how much variance is accounted for by the first, first two, first three etc principal components. How much variance is the first principal component responsible for in the Gaussian data you just generated? How would you need to modify the covariance matrix of the data, in order to decrease the amount of variance in the first principal component? You can test your hypothesis by modifying the properties of the Gaussian data created at the start of `test_mypca()`.

Note that not any matrix is a valid covariance matrix so if you just enter random numbers you are likely to get an error. To start, the matrix needs to be symmetric, and the diagonal values need to be positive. Furthermore, the covariance cannot be large if the variance is small. You can read about how to verify this here: https://math.stackexchange.com/questions/1522397/how-to-tell-is-a-matrix-is-a-covariance-matrix.

Type your answer here

1.12 Project 2: Computer-aided diagnosis

Contents:

References:

[1] Veta M., van Diest P.J., Pluim J.P.W. 2016. Cutting Out the Middleman: Measuring Nuclear Area in Histopathology Slides Without Segmentation. Medical Image Computing and Computer-Assisted Intervention. LINK

[2] Graham, S., Vu, Q. D., Raza, S. E. A., Azam, A., Tsang, Y. W., Kwak, J. T., & Rajpoot, N. 2019. Hover-net: Simultaneous segmentation and classification of nuclei in multi-tissue histology images. Medical Image Analysis, 58, 101563. LINK

1.12.1 Goal

Implement and apply linear regression, logistic regression, a neural network and k-NN for classifying nuclei size in histopathology images, and evaluate and analyze the results.

The size of the cell nuclei of the tumor in breast cancer patients can be indicative of the outcome. Large nuclei size indicates more aggressive tumor and in turn worse prognosis for the patient. As part of their routine work, pathologists make qualitative evaluation of the size of the nuclei by examining the tissue under a microscope. Quantitative measurement (e.g. by manual segmentation) is a much better solution, however, it is unfeasible as it takes additional time away from the busy pathologists. A solution to this problem is to develop an automatic method for measurement of nuclei area.

All data required for this mini-project is provided with the code handout. In the exercises, you applied regression and classification methods on toy datasets, and in the project work you will apply the same methods to a dataset of RGB images of nuclei with size 24×24 pixels. The images originate from the dataset that was previously described in Veta et al. (2015).

1.12.2 Deliverables

There is no hard limit for the length of the report, however, concise and short reports are **strongly** encouraged. Aim to present your most important findings in the main body of the report and (if needed) any additional information in an appendix. The following report structure is suggested for the main body of the report:

1. Introduction
2. Methods
3. Results
4. Discussion
5. Reading assignment (see below)

The introduction and result sections can be very brief in this case (e.g. half a page each). The discussion section should contain the analysis of the results.

The report must be submitted as a single PDF file. The code must be submitted as a single archive file (e.g. zip) that is self-contained and can be used to reproduce the results in the report.

Note that there is not a single correct solution for the project. You have to demonstrate to the reader that you understand the methods that you have studied and can critically analyze the results of applying the methods. Below, you can find a set of assignments (guided project work) that will help you get started with the project work and when correctly completed will present you with a **minimal solution**. Solutions which go beyond these assignments are of course encouraged.

Code and a report describing your implementation, results and analysis.

1.12.3 Assessment

The rubric that will be used for assessment of the project work is given in this table

[1]:
```
%load_ext autoreload
%autoreload 2
```

1.12.4 Guided project work

A. Linear regression for nuclei area measurement

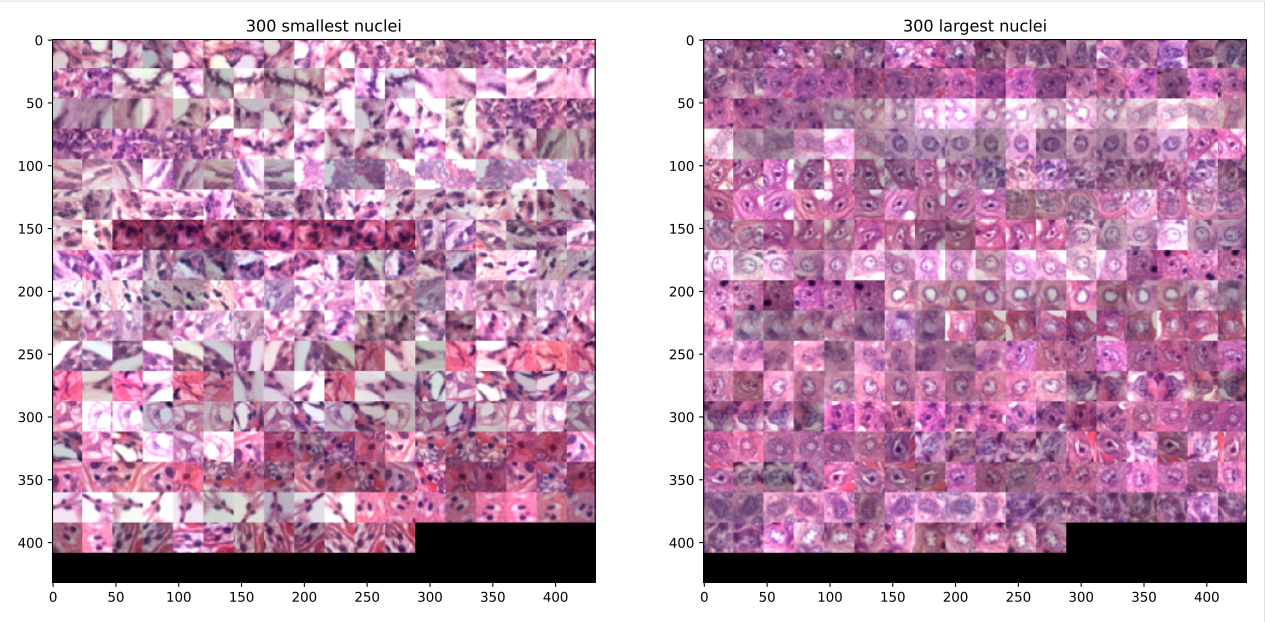
The Python function `nuclei_measurement()` implements training of a linear regression model for measuring the area of nuclei in microscopy images. The dataset for this problem consists of small RGB images of size 24×24 pixels with a nucleus in the center. Such images can be obtained, for example, by cropping from larger images after performing a nuclei detection step. The targets are the areas of the nucleus in the center of the image obtained by manual measurement. The linear regression model that we are going to train will enable us to automatically measure the size of new, previously unseen samples (without resorting to manual measurement).

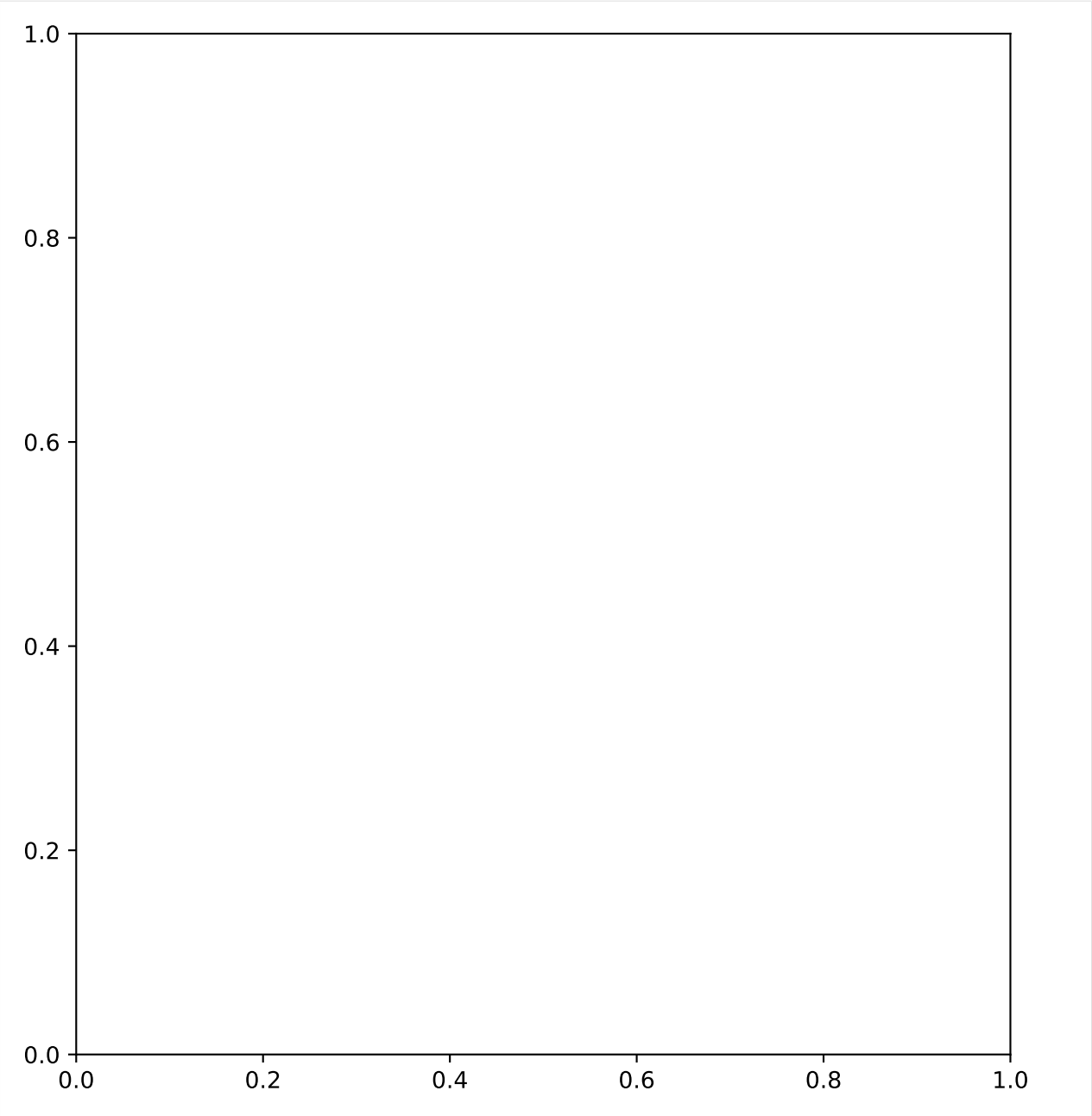
The first section of code loads and prepares the dataset. The data is already split into a training and testing set, each containing more than $20,000$ samples (a validation dataset is not needed as we are not going to perform model selection, i.e. we are going to stick to linear regression). The last few lines of the first section of code visualize the 300 smallest and 300 largest nuclei in the training dataset.

In this example, we are not going to perform feature extraction but use the raw pixel values as features. Since each sample is an RGB image with size 24×24 pixels, we end up with $24 \times 24 \times 3 = 1728$ features. Locate the code that reshapes each image into a feature vector and make sure you understand how it works.

[2]:
```
%matplotlib inline
import sys
sys.path.append("../code")
from cad_project import nuclei_measurement

nuclei_measurement()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[2], line 6
      3 sys.path.append("../code")
      4 from cad_project import nuclei_measurement
----> 6 nuclei_measurement()

File ~/checkouts/readthedocs.org/user_builds/8dc00-mia-docs/checkouts/latest/docs/source/
↪code/cad_project.py:59, in nuclei_measurement()
     57 fig2 = plt.figure(figsize=(16,8))
     58 ax1  = fig2.add_subplot(121)
---> 59 line1, = ax1.plot(test_y, predicted_y, ".g", markersize=3)
     60 ax1.grid()
     61 ax1.set_xlabel('Area')

NameError: name 'predicted_y' is not defined
```


300 smallest nuclei
300 largest nuclei
0
50
100
150
200
250
300
350
400


1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

***Task 1*:**

Implement the missing functionality for training a linear regression model for automatic measurement of the nuclei area. Evaluate the performance on the independent test dataset. The next lines of code plot the predicted vs. the actual area. What is your analysis of the results shown in the plot?

***Task 2*:**

Train a new linear regression model with a reduced number of training samples. Which model results in larger error on the testing set and why?

B. Logistic regression for nuclei classification

The Python function `nuclei_classification()` implements the training of a logistic regression model that classifies nuclei into the classes "large" (class label $y = 1$) and "small" (class label $y = 0$). Examine the code and comments and make sure that you understand what it does. One notable difference from before is that this code uses the analytical expression for the gradient of the loss function, instead of computing it numerically with `ngradient` as before. Using `ngradient` will also work, but is much slower. The script is mostly complete. The only missing component is the values for the parameters of the training process.

[3]:
```
%matplotlib inline
import sys
sys.path.append("../code")
from cad_project import nuclei_classification
from IPython.display import display, clear_output

nuclei_classification()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[3], line 7
      4 from cad_project import nuclei_classification
      5 from IPython.display import display, clear_output
----> 7 nuclei_classification()

File ~/checkouts/readthedocs.org/user_builds/8dc00-mia-docs/checkouts/latest/docs/source/
↪code/cad_project.py:103, in nuclei_classification()
     93 training_x, validation_x, test_x = util.reshape_and_normalize(training_images,␣
↪validation_images, test_images)
     95 ## training linear regression model
     96 #--------------------------------------------------------------------#
     97 # TODO: Select values for the learning rate (mu), batch size
   (...)
    100 # fast training of an accurate model for this classification problem.
    101 #--------------------------------------------------------------------#
--> 103 xx = np.arange(num_iterations)
    104 loss = np.empty(*xx.shape)
    105 loss[:] = np.nan
```

(continues on next page)

(continued from previous page)

```
NameError: name 'num_iterations' is not defined
```

***Task 3*:**

1. Select values for the learning rate, batch size and number of iterations (these are sometimes called hyper-parameters of the model), as well as initial values for the model parameters that will result in fast training of an accurate model for this classification problem. Note that if you don't choose the hyper-parameters and initial parameters well, the resulting loss might be out of range of the plot.

   Experiment with a few variations of the parameters and analyze and compare the resulting loss curves. Describe how the different hyper-parameters influence the training process.

2. Instead of running gradient descent for a fixed number of iterations, can you propose a stopping criterion for the training?

3. Report the classification accuracy for your best trained model.

4. Reduce the size of the training set by a very large factor (e.g. 0.5% of the original number of samples). Train the model with this reduced number of samples. Does the model overfit the training dataset? How did you come to this conclusion?

C. Neural network training for nuclei classification

The code in SECTION 3 of the `cad_test.py` file, enabled you to train a neural network that classifies whether the input image contains a large or small nuclei. During the training of this network the hyper parameters (i.e. batch size, learning rate, number of iterations, ...) have been kept fixed. However, these parameters can be a crucial factor in optimizing your model results. Therefore we are going to investigate the effect of these parameters on the loss curve behaviour and accuracy of the model. To put the improvements in perspective, we are also going to compare these results to the Logistic Regression model (see Task 4.1).

***Task 4*:**

1. Select values for the learning rate, batch size and number of iterations similar to those used in Task 4. Analyze and compare the resulting loss curves with the loss curves obtained from the logistic regression. What do you observe? Also, report the accuracy of the methods you compare. Is there a best one?

2. Now, with a fixed set of hyper parameters, try to change the size of the hidden layer and report the obtained loss curves. How do the number of parameters of the model influence the loss curve? Can you think of an appropriate number of parameters for this dataset?

D. Using k-NN for nuclei classification

In Notebook 2.1 we have introduced the k-NN algorithm, in Notebook 2.4 the PCA method was introduced. In this task we would like to combine these two to create a new classifier. How does a clustering algorithm hold up against the algorithm you have implemented thus far?

Task 5:

1. Use the training set of the cell nuceli data for the k-NN algorithm. Next, use the test set to classify the images with this algorithm. How did you choose the parameter k? What happens to the accuracy when you first use PCA to reduce the dimensions on the data, how many components do you keep?
2. Report the accuracy, and compare it to the best performing neural network, logistic regression and linear regression models. What do you see?

E. Reading assignment

In recent literature, various deep learning-based methods have been proposed for cell nuclei segmentation and classification. In this reading assignment, you are asked to carefully study the paper by Graham et al. (2019).

Task 6:

In a separate section of your project report (~ half a page), compare your own linear (section A) and logistic (section B) regression-based methods and your small neural network (section C) with the deep neural network proposed by Graham et al. (2019). Start with giving a brief summary of the method proposed by Graham et al. What are the advantages of their method, and what are its weak points/disadvantages?

1.13 Active shape models

This (optional) notebook combines theory with exercises to support the understanding of active shape models for object detection in medical image segmentation. Implement all functions in the `code` folder of your cloned repository, and test it in this notebook after implementation. Use available markdown sections to fill in your answers.

Contents:

1. Active shape models

- Using ASMs for segmentation tasks

References:

[1] Active shape models: Cootes et al. Active Shape Models - Their Training and Application, Computer Vision and Image Understanding (1994)

[2] Chapter 11.5 of the Guide to Medical Image Analysis by Tonnies, Klaus D

[1]:
```
%load_ext autoreload
%autoreload 2
```

1.13.1 1. Active shape models

In object detection, model-based vision allows for the recognition and location of known objects or patterns despite the presence of occlusive phenomena (e.g. noise). But what if the appearance of the object varies? This is where rigid models become inefficient. Active shape models (ASMs) are statistical models of the shape of objects which iteratively deform to fit to an example of the object in a new image. Iterative deformation is achieved by active **segmentation** preceded by registration of a model to the new image. For a detailed explanation of ASM priciples, **please read carefully the following article**: Cootes et al. Active Shape Models - Their Training and Application, Computer Vision and Image Understanding (1994)

In principle, an ASM aims to find shapes and acceptable variations of an object in a new image based on the model created from a sufficiently large training dataset. Variations in an active shape from a training phase are used to predict variation of unknown objects.

An ASM describes a K-dimensional shape that has L boundary points in a *shape feature vector* $\mathbf{s} = (s_0, s_1, ..., s_N) = (x_{1,1}x_{1,2}...x_{1,L}, x_{2,1}x_{2,2}...x_{2,L}, ..., x_{K,1}, ...x_{K,L})$, where $x_{k,l}$ denotes the *k*th component of the *l*th boundary point $\mathbf{x}_i$.

With increasing K dimension of the shape feature vector, more samples in the feature space are needed to compute a reliable estimate.

An ASM is defined by its probability density function (PDF) that reflects deformation within an object class. Computation of a probabilistic shape model from training samples is typically performed as follows:

1. Identify several points on the object boundary to select landmarks (Note: semantic equivalence of selected landmarks across all training data needs to be assured)
    - primary landmarks are anatomical landmarks equivalent to anatomical locations (e.g. the brain's Sylvian fissure)
    - secondary landmarks are other image features, e.g. ridge intersections on the brain surface
    - tertiary landmarks are used to represent curvature of the shape boundary
2. Align landmarks within a common coordinate system
3. Decorrelate the estimated covariance matrix to obtain uncorrelated features with eigenvectors of that covariance matrix
4. Clean up the feature space by only keeping significant variations that lie below some percentage of the total variance in the training data.

As you may correctly anticipate, there are several bottlenecks present in the process of estimating the probabilistic shape model. Semantic equivalence is often difficult, let alone impossible to determine. Moreover, human interaction is required for landmark detection, which is rarely feasible in medical practice. Hence, landmark detection is conducted based on local attributes (e.g. curvatures), geometric shape features or registration of atlases. Moreover, landmark alignment based on invalid assumptions about the object's coordinate system may cause wrong shape variation. Last but not least, the limited amount of training samples typically present in practice leads to a decrease in the significance of the estimated probability distribution.

Further information about active shape models as well as active appearance models can be found in chapter 11.5 of the Guide to Medical Image Analysis by Tonnies, Klaus D.

Using ASMs for segmentation tasks

As mentioned above, most of the issues related to estimating the probability density function of an ASM represent classification problems. Hence, the ASM approach is suitable for segmentation tasks. The main idea is to apply an ASM such that its shape is aligned and deformed to fit a potential shape instance in an image. ASM-assisted segmentation is done in the following steps:

1. Register an active shape model with new image data that contain a shape which is not accounted for in the shape model in terms of position, orientation and scale

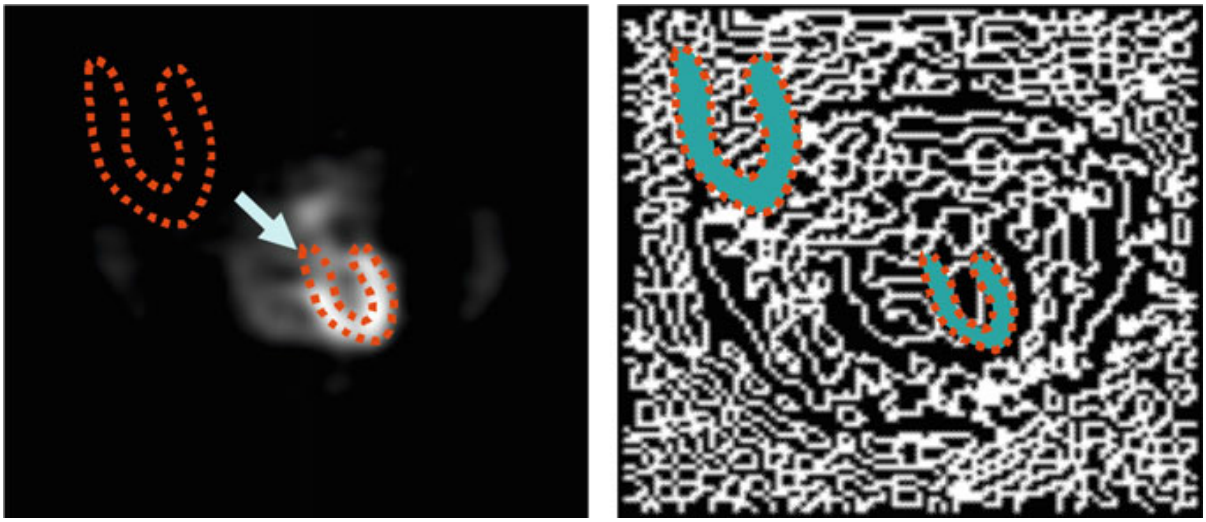


Figure from Toennies K.D. Guide to Medical Image Analysis

2. Apply local deformations of the shape model to fit the object in the target image, thereby creating a new estimate for computing the next pose estimate (positions of the model shape with deformations)

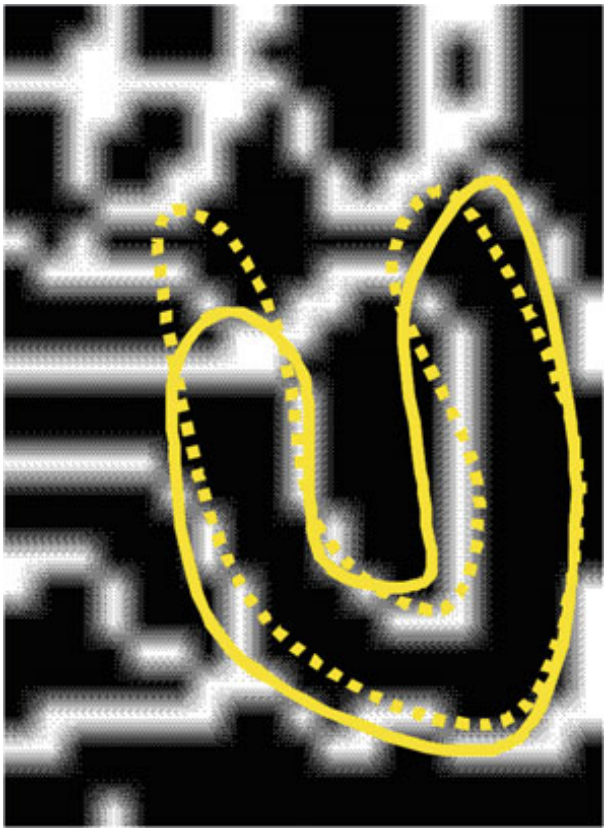


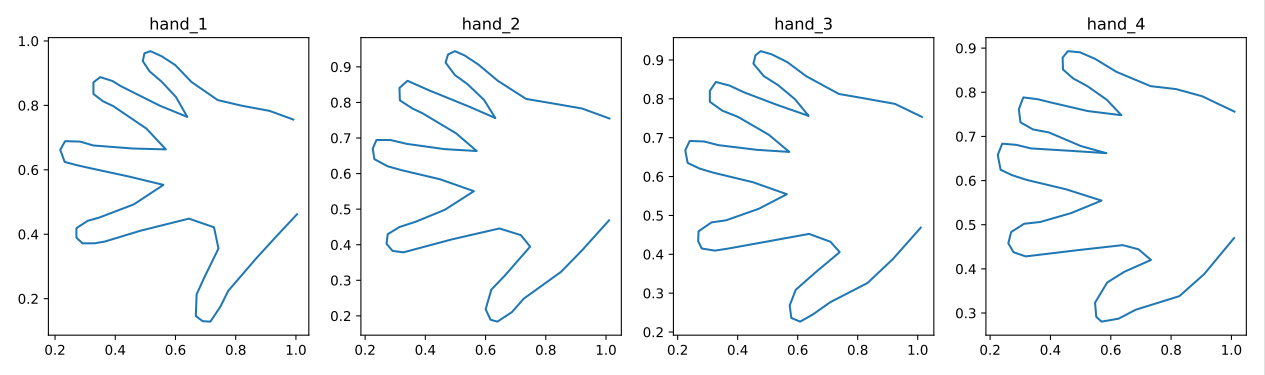
Figure from Toennies K.D. Guide to Medical Image Analysis

3. Compute as many pose estimates as are required for convergence.

Exercise 1.1:

In the `plot_hand_shapes()` function in `active_shape_models.py`, we start with loading `coordinates.txt` which contains coordinates of 40 hand shapes, each represented by 56 points. Dimensions 1 to 56 store the x-coordinate and dimensions 57 to 112 store the y-coordinate. Let's plot a few shapes to examine the variation. What do you think the mean shape will look like? Compute it to verify your guess. Implement this functionality in the `plot_hand_shapes()` function.

[2]:
```
%matplotlib inline
import sys
sys.path.append("../code")
from active_shape_models import plot_hand_shapes
plot_hand_shapes()
```



Exercise 1.2:

Apply `mypca` (from the previous exercises on PCA) to the coordinates data. How many dimensions are needed to describe 98% of the variance? Store only the vectors corresponding to these dimensions in `v`. Implement your code in the `pca_hands()` function of the `active_shape_models.py` module.

[3]:
```
%matplotlib inline
import sys
import warnings
warnings.filterwarnings('ignore')
sys.path.append("../code")
from active_shape_models import pca_hands
num_dims, v_new, _, _ = pca_hands()
print('Number of dimensions explaining 98% variance: {}'.format(num_dims))
print('Eigenvectors for these dimensions (shape): {}'.format(v_new.shape))
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[3], line 7
      5 sys.path.append("../code")
      6 from active_shape_models import pca_hands
----> 7 num_dims, v_new, _, _ = pca_hands()
      8 print('Number of dimensions explaining 98% variance: {}'.format(num_dims))
      9 print('Eigenvectors for these dimensions (shape): {}'.format(v_new.shape))

File ~/checkouts/readthedocs.org/user_builds/8dc00-mia-docs/checkouts/latest/docs/source/
↪code/active_shape_models.py:53, in pca_hands()
     47 coordinates =  np.loadtxt(fn)
     48 #-----------------------------------------------------------------#
     49 # TODO: Apply PCA to the coordinates data.
     50 #-----------------------------------------------------------------#
     51 # Note: this function also needs to return the eigenvectors v and the
     52 # eigenvalues w (you will need these in the next exercise)
---> 53 return num_dims, v_new, v, w

NameError: name 'num_dims' is not defined
```

Exercise 1.3:

Create a loop in the `test_remaining_variance()` function of the `active_shape_models.py` module to go through the dimensions left in `v` and compute a variation that this dimension produces. For the weight, you might want to use the corresponding eigenvalue multiplied by a small scaling factor, like 5. What is the main variation that you notice?

Note: If you see the warning **ComplexWarning: Casting complex values to real discards the imaginary part**, just ignore it.

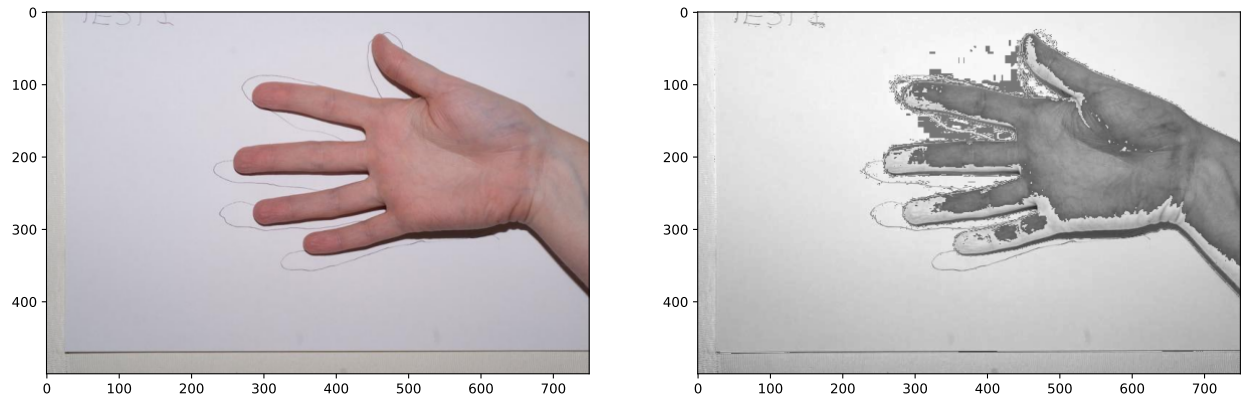
[4]:
```
%matplotlib inline
import sys
sys.path.append("../code")
from active_shape_models import test_remaining_variance
test_remaining_variance()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[4], line 5
      3 sys.path.append("../code")
      4 from active_shape_models import test_remaining_variance
----> 5 test_remaining_variance()

File ~/checkouts/readthedocs.org/user_builds/8dc00-mia-docs/checkouts/latest/docs/source/
↪code/active_shape_models.py:60, in test_remaining_variance()
     58 coordinates =  np.loadtxt(fn)
     59 mn = np.mean(coordinates, axis=0)
---> 60 num_dims, v_new, v, w = pca_hands()
     62 fig = plt.figure(figsize=(15,10))

File ~/checkouts/readthedocs.org/user_builds/8dc00-mia-docs/checkouts/latest/docs/source/
↪code/active_shape_models.py:53, in pca_hands()
     47 coordinates =  np.loadtxt(fn)
     48 #----------------------------------------------------------------#
     49 # TODO: Apply PCA to the coordinates data.
     50 #----------------------------------------------------------------#
     51 # Note: this function also needs to return the eigenvectors v and the
     52 # eigenvalues w (you will need these in the next exercise)
---> 53 return num_dims, v_new, v, w

NameError: name 'num_dims' is not defined
```

Exercise 1.4:

Let's load the image `test001.jpg` from the `dataset_hands` folder, and view it in grayscale. If you were to plot the hand template on top of this image, what do you expect to happen? Verify your hypothesis. Implement your code in the `plot_hand_grayscale()` function of the `active_shape_models.py` module.

[5]:
```
%matplotlib inline
import sys
sys.path.append("../code")
from active_shape_models import plot_hand_grayscale
plot_hand_grayscale()
```



Exercise 1.5:

Let's transform your mean hand shape into a 2x56 dataset with `initialpos = [[meanhand[0,:56]], [meanhand[0,56:112]]]`. Think about the registration exercises you did before. Define a transformation matrix (you can try out yourself what numbers are needed) and use it to plot the hand template close to the hand in the image. Implement your code in the `test_transformed_hand()` function of the `active_shape_models.py` module.

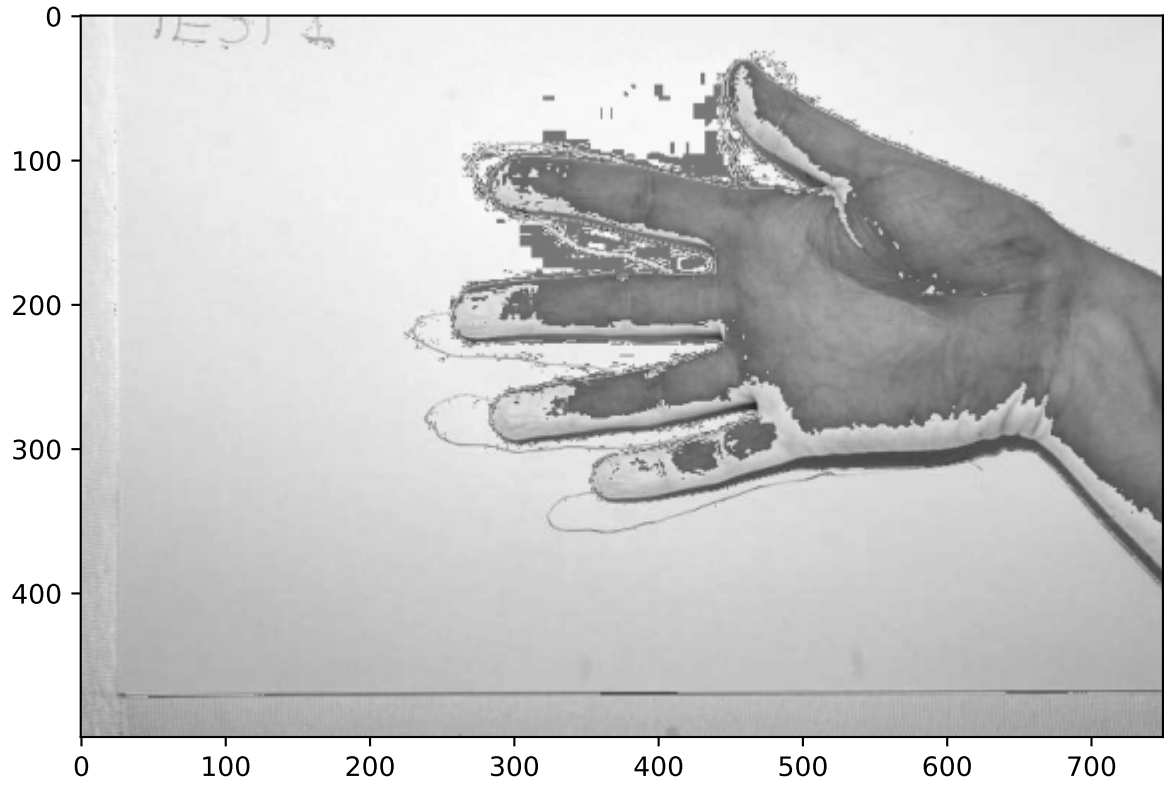
[6]:
```
%matplotlib inline
import sys
sys.path.append("../code")
from active_shape_models import test_transformed_hand
test_transformed_hand()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[6], line 5
      3 sys.path.append("../code")
      4 from active_shape_models import test_transformed_hand
----> 5 test_transformed_hand()

File ~/checkouts/readthedocs.org/user_builds/8dc00-mia-docs/checkouts/latest/docs/source/
↪code/active_shape_models.py:111, in test_transformed_hand()
    109 ax2 = fig.add_subplot(122)
    110 ax2.imshow(img2, cmap='gray')
```

(continues on next page)

(continued from previous page)

```
--> 111 ax2.plot(shape_t[0,:], shape_t[1,:], 'r')

NameError: name 'shape_t' is not defined
```



Question 1.1:

Consider an active shape model for segmentation of the ventricles in the sample brain images. Describe which steps you would need to do for the data that is available to us, to train a shape model of the ventricles.

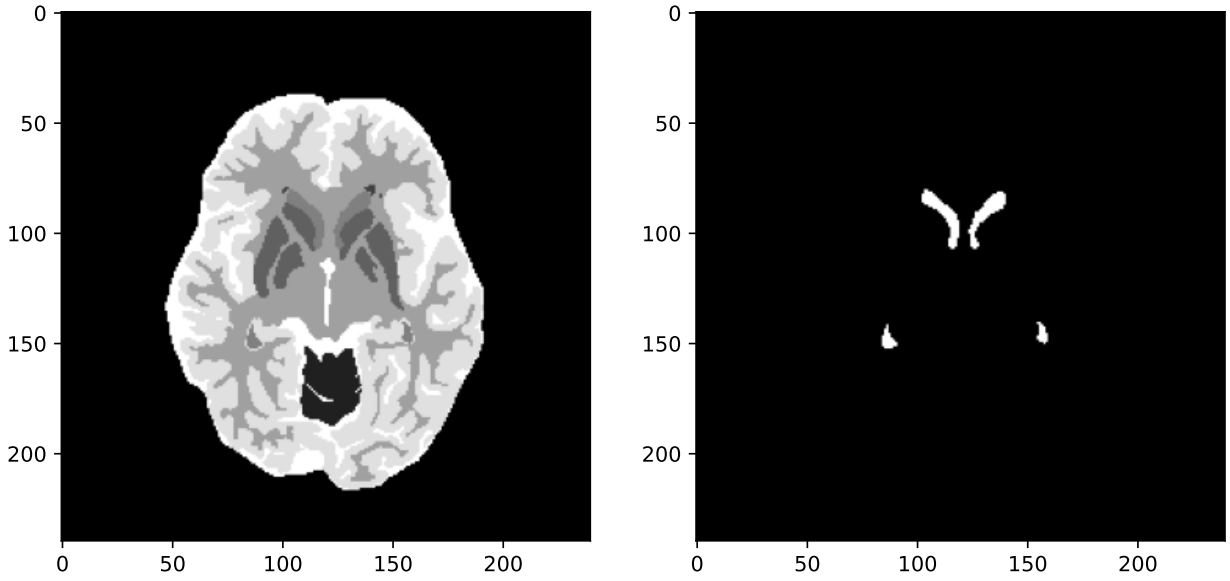
Type your answer here

Exercise 1.6:

You can inspect the mask of the ventricles in the sample brain images using the code below.

[7]:
```
%matplotlib inline
import sys
import matplotlib.pyplot as plt
sys.path.append("../code")
GT = plt.imread('../data/dataset_brains/1_1_gt.tif')
gtMask = GT == 4
fig = plt.figure(figsize=(10,10))
ax1  = fig.add_subplot(121)
ax1.imshow(GT)
ax2  = fig.add_subplot(122)
ax2.imshow(gtMask)
```

[7]:
```
<matplotlib.image.AxesImage at 0x7fd45b81a6d0>
```



?

Question 1.2:

Look at the ventricle masks for different subjects and different slices. Based on the shapes that you see, what difficulties do you think you might face, if you wanted to train an active shape model? How could you modify the dataset to overcome these difficulties?

Type your answer here