People's Democratic Republic of Algeria
University Ibn Khaldoun Tiaret
Faculty of Natural Sciences and Life

Artificial Intelligence: Applications in Biology and Medicine

Dou El Kefel MANSOURI and Wassim YEZLI

February 15, 2022

Contents

Preface

Who Should Read This Manuscript ?: This manuscript is intended for an audience with a basic understanding of biology or medicine, mathematics, and also programming. Approaching this manuscript without this context is possible, but probably more challenging. The manuscript may also be of keen interest to academics as it presents a set of python language source codes for each AI method presented in the manuscript.

What's In This Manuscript ?: The manuscript is divided into four parts, and each part consists of several chapters. Part I : is an introduction to biology and medicine and their challenges in the twenty-first century. This part also presents an introduction to artificial intelligence and its contributions to these two fields. Part II : introduces the concept of Image Processing. It consists of three chapters including, the image formation, image digitization and basic image processing algorithms. Part III : introduces the concept of Machine Learning. It consists of six chapters including, classification, regression, clustering, deep learning, object recognition, and dimensionality reduction. Each of the aforementioned part ends with the reference section. Part IV : is a gallery of applications.

List of Symbols

- **Pixel** Picture element, the smallest item of information in an image.
- **LED** full Light-Emitting Diode.
- **CCD** Charge Coupled Device.
- **CMOS** Complementary Metal Oxide Semiconductor.
- **IP** Internet Protocol.
- **PCR** Polymerase Chain Reaction.
- **MRI** Resonance Imaging.
- **CT** Computed Tomography.
- **RGB** Red Green Blue.
- **HD** High Definition.
- **4K** Horizontal resolutions of around 4,000 pixels.
- **Python** programming language.
- **DFT** Discrete Fourier Transform.
- **DCT** Discrete Cosine Transform.
- **DWT** Discrete Wavelet Transform.
- **CNN** Convolutional Neural Networks.
- **ANN** Artificial Neural Network.

- **BNN** Biological Neural Network.
- **YOLO** You Only Look Once.
- **DT** Decision Tree.
- **SVM** Support Vector Machine.
- **KNN** k-Nearest Neighbors.
- **QR** Quantile regression.
- **BIRCH** Balanced Iterative Reducing and Clustering using Hierarchies.
- **DBSCAN** Density-Based Spatial Clustering of Applications with Noise.
- **Fuzzy C-Means Clustering**.
- **PCA** Principal Component Analysis.
- **MDS** Multidimensional scaling.
- **LDA** Linear Discriminant Analysis.
- **CCA** Canonical correlation analysis.
- **Isomap** Isometric mapping.
- **KPCA** Kernel Principal Component Analysis.
- **LLE** Locally Linear Embedding.

Part I :

Introduction

Chapter 1

Biology and medicine into the 21st century: Opportunities & Challenges

Chapter 2

Artificial intelligence

Modern biology is witnessing a revolution and the spread of knowledge on an unprecedented scale. This is due to the ever-increasing detail and sophistication, as well as the shift from the principle of reduction to the integrative and organizational principles of living systems. Biology, as a scientific study, raises an obvious problem about the large amount of data that characterizes it and that is obtained from heterogeneous sources. Before long, analyzing biology data was done with traditional laborious methods, often time-consuming. Today, Artificial Intelligence (AI) is expected to dominate biological data thanks to its powerful surveying and classification tool. AI is primarily based on mathematical models that can describe complex biological systems in an accurate manner. In general, most modern AI techniques originate from computer vision, including data processing (for example, image recognition, natural language processing, ...). The real-time concept also has an important place in AI. In this chapter, we present in a simple way both concepts: computer vision and real time.

Computer vision concept

What is Computer vision?

Computer vision is an interdisciplinary scientific field in which computers seek to interpret scenes at the same level as a human visual system can do. Let's look at Figure 2.1 that illustrates a scene composed of various objects.

From the figure, it is understood that a group of children is running

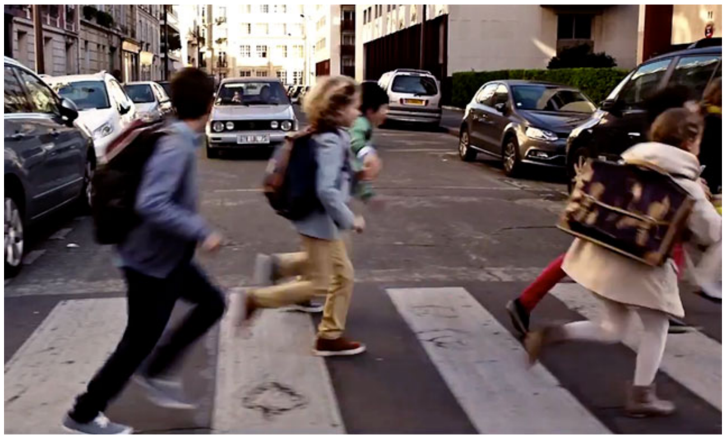


Figure 2.1: Simple example of computer vision interpretation.

through a pedestrian crossing, located on a street that appears to be in a residential area. The image looks recently captured, based on the Volkswagen car parked to the right of the image. The number of children is six. Children's details are not clear enough in the figure, this makes distinguishing between girls and boys a little difficult. Let's now focus on the two boys in the middle of the children group. We can see that the boys are running around smiling. This indicates that there is no danger to children from oncoming cars. We now turn to the set of images in Figure 2.2. One could say at first glance that the images show danger. The danger exposed by the two images above requires very rapid parental intervention. As for the danger presented in the two images below requires the intervention of security services. In fact, the human visual system can arrive at this interpretations very quickly without any effort thanks to the brain which has more than 10^{10} neurons. However, the number of images that cross the human in his daily life, like those presented above, is very large and far exceeds their ability to understand, interpret and make complex decisions based on them.

Computers are extremely efficient solution that perform tasks requiring human intelligence. Thanks to the cameras built around the right resolution, computers are able to collect a very large number of images, and interpret them at the same time while making best decisions. For instance, computer vision system can monitor the highways, neighborhoods, green spaces, and easily inspect object details too small to be seen by the human eye. This improves the speed, accuracy, and effectiveness of human efforts. Indeed,

Figure 2.2: Some images that present a danger.

even though computers are very efficient, the images actually contain a lot of information, making its interpretation difficult for humans, not to mention computers alone. From here, it is important to ask the following question: 'How makes a computer intelligent ?'. As humans, we usually go through a number of steps to quickly understand images and make decisions. We may first observe the images and secondly rely on what we know to explain what we see and finally make the decision. Likewise, computers should use similar steps to reach human intelligence.

Components of a vision system

The common core function of biological and computer vision includes the following units:

– **Lighting.** radiation should be emitted from the part to be inspected so that it can be seen clearly by the camera and processed afterwards. The light sources commonly used to illuminate the parts in machine vision are fluorescent, quartz halogen, LED, mercury, and xenon. It replaces natural sources, such as solar radiation or moonlight and even fire, which was one of the first attempts at inventing the light source. Figure 2.3 shows the lighting principle. Note that computer vision analyzes the reflected light from the object, not the object itself. It should also be noted that the higher the control over the use of light, the higher the image quality and the more precise the analysis.

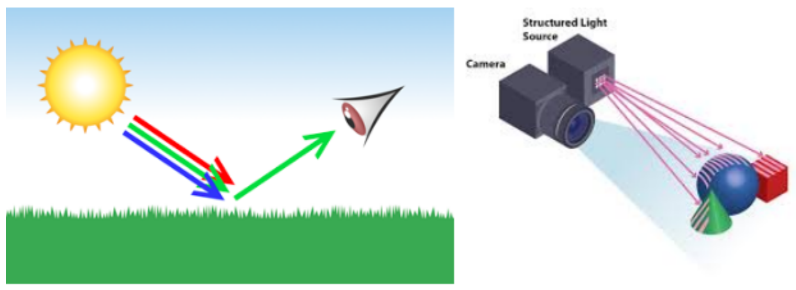


Figure 2.3: Lighting principle.

- **Camera lens.** also known as photographic lens. It captures the image by collecting the radiation received from the object and presents it to the sensor in the form of light. Lens vary in optical quality and price. It might also be permanently fixed to a camera or interchangeable. Figure 2.4 shows an example of camera lens.

Figure 2.4: Example of camera lens.

- **Sensor.** The basic component of digital cameras. It receives the external light in the form of energy, stores it in its cells (pixels), and then turns it into a digital image. The digital image is then sent to the processing unit for analysis. Note that low light produces darker pixels, while bright light gives brighter pixels. Figure 2.5 shows an example of sensor. Sensors can be categorized in several ways, by resolution, frame rate, pixel size, sensor format, structure type (CCD or CMOS), chroma type (color or monochromatic) or shutter type (shutter global or rolling).

- **Processing unit.** processes incoming image data, usually higher di-

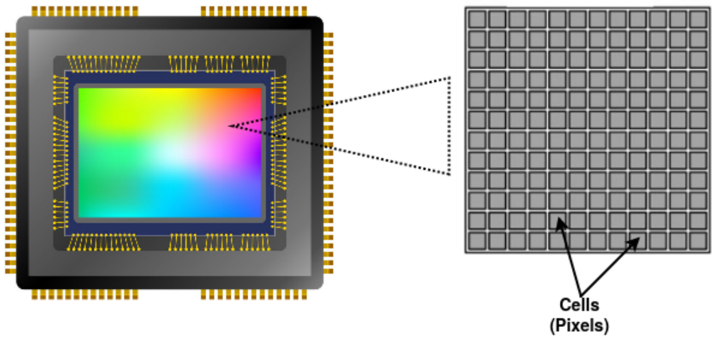


Figure 2.5: Example of sensor.

mensional data. It can take place in a computer or in a standalone vision system. It is always associated with a memory system to store image data, including software to perform the processing. In the context of computers, there are several processing units that differ in width and speed. It is manufactured by various companies such as Intel, AMD, Qualcomm, Motorola, Samsung, IBM, etc. Figure 2.6 shows the Central Processing Unit.

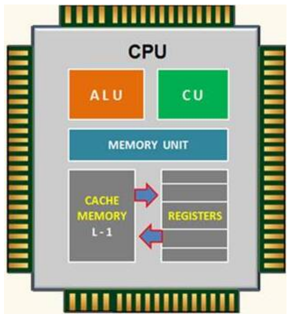


Figure 2.6: Central Processing Unit.

– **Communications.** between the aforementioned set of off-the-shelf components. Communication is done using a conventional RS-232 serial output, Ethernet or Ethernet/IP.

Figure 2.7 briefly illustrates components of machine vision mentioned in this section.

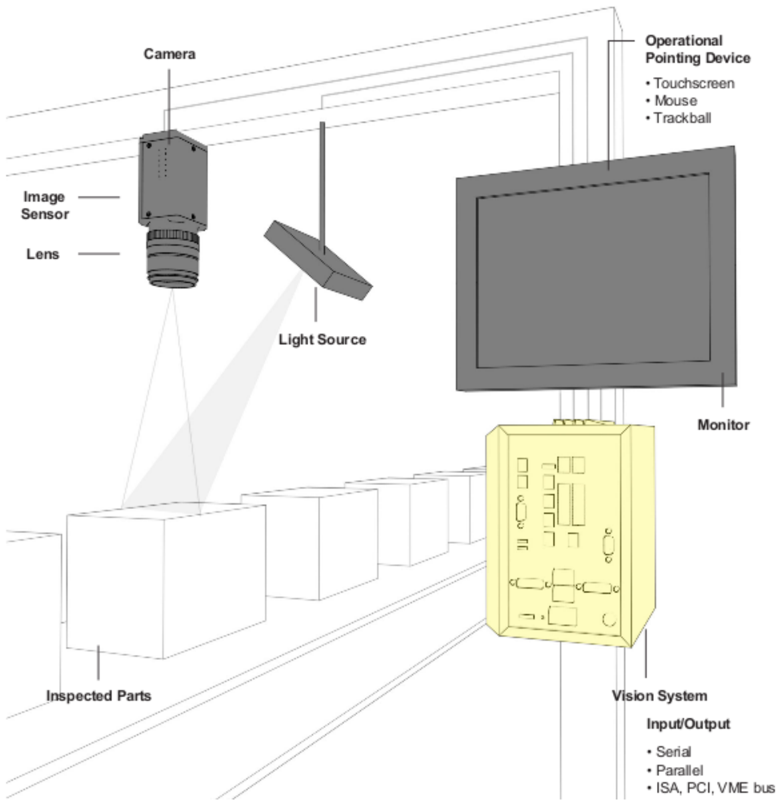


Figure 2.7: Components of machine vision (Figure adopted from url https://www.cognex.com/what-is/machine-vision/components).

Basic stages of computer vision

A computer vision system often consists of an image acquisition system, image processing and statistical analysis programs. Image acquisition system include: an illumination device, a frame-grabber, and a computer. Image processing and statistical analysis rely on specialized algorithms that enhance, recognize, locate and classify objects. For example, neural networks are among the most efficient algorithms for identifying and classifying objects. In addition, systems evolve according to the algorithms that make it work. As a result, surveillance cameras today are radically different from surveillance cameras of the year 2000, because surveillance cameras algorithms are now more sophisticated than the algorithms used in the year 2000. In this handout, we will introduce some computer vision algorithms to help the reader understand the process of computer vision.

Computer vision in Biology and Medicine

Computer vision plays an important role in many fields of biology. For instance, in cell biology, computer vision is used to visualize, measure and also to interpret the content of a cell image through the basic stages of computer vision mentioned above. In biodiversity, computer vision algorithms can make a significant contribution to reducing biodiversity loss caused by rising sea levels, extreme temperatures, and the growing human population. In animal ecology, computer vision solutions can greatly increase the efficiency of monitoring animal species in the natural world. In the food industry, computer vision systems are increasingly used for food quality inspection. Besides, in medicine, computer vision is used in various healthcare applications like precise detection of brain tumors, diagnosis and treatment of Covid-19 and also other complex medical diagnostic tasks covering dermatology, radiology or pathology.

Real-time concept

Generally speaking, we are talking about the real-time principle when we are faced with a problem that involves response time constraints. Suppose an air traffic controller monitors the airspace and gives take-off or landing orders to pilots. If he ever give a landing order to a pilot and his order does not arrive on time, this delay can cause catastrophic events. Another example, let's say you are watching the Champions League final on TV. The match is televised live in the UK by the beIN SPORTS, two thousand kilometers away from your place of residence. You will receive all the match photos live in real-time as if you are on the field. If there is a problem with the transmission, for example the sound comes before the photo, or if there are interruptions that last one minute or more each time, you won't want to continue watching the match.

In fact, a real-time response is critical because any response, even if it is correct, after a deadline is worthless, and the consequences of a delayed response are just as dangerous as the consequences of an incorrect response. From here, it is essential that the request time roughly coincides with the response time. Real-time is generally referred to as real-time systems or real-time algorithms. For instance, airbag deployment systems are an example of real-time systems, and algorithms applied to airbag deployment systems are an example of real-time algorithms. Note that the airbag deployment system uses an algorithm based on the control unit sensor and the frontal impact sensor. It distinguishes collisions along these sensor signals. Like computer vision algorithms, real-time algorithms evolve according to time and need.

Characteristics of a real-time system

Here are some key features of the real-time system:

- **Time Constraint:** means that all tasks assigned to a program should be completed before the deadline.
- **Correctness:** means getting a correct result in a time constraint, and any result after the deadline is not considered correct.
- **Safety:** the system should work for a long time without human intervention and without failure, and it will reset itself when the failure occurs without causing any damage to the data and information.
- **Concurrency:** the system can respond to a certain number of tasks simultaneously and according to the deadline set for each task.
- **Distributed:** the system should work correctly even if its different components are in different geographical locations.
- **Stability:** means that the system should gives stable results in a time constraint even when there are several tasks running at the same time.
- **Scalability:** the system should be able to scale up or down to meet application-specific requirements.

Real-time in Biology and Medicine

In general, when we talk about real-time in biology, we mean real-time PCR also known as quantitative PCR, which is used in diverse applications such as gene expression analysis, the detection of genetically modified organisms, etc. In medicine, the real-time concept is of great importance. For instance, real-time proactively provides live information for monitoring and intervention in care, as well as continuous improvement in patient care.

Chapter 3

Suggested readings

1. Schaffner, K. F. (1993). Discovery and explanation in biology and medicine. University of Chicago press.

2. Winston, P. H. (1992). Artificial intelligence. Addison-Wesley Longman Publishing Co., Inc.

3. Hunter, L. (Ed.). (1993). Artificial intelligence and molecular biology (Vol. 445). Menlo Park: Aaai Press.

4. Langton, C. G. (Ed.). (1997). Artificial life: An overview.

5. Ginsberg, M. (2012). Essentials of artificial intelligence. Newnes.

6. Flasiński, M. (2016). Introduction to artificial intelligence. Switzerland: Springer International Publishing.

7. Pham, T. D., Yan, H., Ashraf, M. W., & Sjöberg, F. (2021). Advances in Artificial Intelligence, Computation, and Data Science. Springer International Publishing.

8. Jahne, B. (Ed.). (2000). Computer vision and applications: a guide for students and practitioners. Elsevier.

9. Kisacanin, B., Bhattacharyya, S. S., & Chai, S. (Eds.). (2008). Embedded computer vision. Springer Science & Business Media.

10. Szeliski, R. (2010). Computer vision: algorithms and applications. Springer Science & Business Media.

11. Forsyth, D. A., & Ponce, J. (2012). Computer vision: a modern approach. Pearson.

12. Davies, E. R. (2017). Computer vision: principles, algorithms, applications, learning. Academic Press.

13. Mellor, D. H. (1985). Real time. CUP Archive.

14. Liu, F., Narayanan, A., & Bai, Q. (2000). Real-time systems.

15. Dorak, M. T. (Ed.). (2007). Real-time PCR. Taylor & Francis.

16. Mackay, I. M. (2007). Real-time PCR in microbiology (pp. 13-35). Norfolk, UK: Caister Academic Press.

17. Meuer, S., Wittwer, C., & Nakagawara, K. I. (Eds.). (2012). Rapid cycle real-time PCR: methods and applications. Springer Science & Business Media.

18. Akenine-Moller, T., Haines, E., & Hoffman, N. (2019). Real-time rendering. Crc Press.

Part II :

Image Processing

Chapter 4

Image formation

When we talk about biomedical imaging, we are necessarily referring to the various imaging tools used to obtain medical images, such as magnetic resonance imaging (MRI), X-ray computed tomography (CT) imaging, and microscopy. These imaging tools have provided unprecedented accuracy and reliability in disease identification and treatment response. In this section, we provide a brief description of these different imaging modalities.

4.1 X-Ray Imaging

Like radio waves and visible light electromagnetic radiation, X-rays are also a type of radiation that has both a frequency and a wavelength. However, the frequency of X-rays is higher and their wavelength is shorter ranging from 0.124 to 0.0124 nm, and can be compared to the atomic and/or molecular spacing of different materials. These properties allow X-rays to entirety penetrate most materials and interact with them at the microscopic scale, which constitutes a non-destructive volumetric imaging of the molecular structure of objects. Figure 4.1 shows the process generation of X-ray image by typical medical imaging applications. X-rays are emitted by the X-ray tube in the form of photons, which have a much higher energy than that of light. These photons are then collimated by a beam-limiting device. Afterwards, they enter the object (patient), where they may be scattered or absorbed. The X-ray beam absorbed will be proportional to the type and size of object, more X-rays that pass through the object, the higher the optical density of the image. At the end, an image is obtained which contains a two-dimensional distribution of optical density which relates for example to the tissue distribution within the patient.

4.2 Computed Tomography (CT)

Unlike a conventional X-ray which cannot provide depth information from a single image. CT is a new imaging technique that uses a motorized X-ray source and provides a different form of imaging known as cross-sectional imaging. Specifically, computed tomography measure X-ray transmission through a patient for a large number of views. It separates superimposed anatomical details and produces axial sectional images that are used for a variety of diagnostic and therapeutic purposes. Figure 4.2 shows a modern CT scan machine and figure 4.3 shows a CT scan image of a man's brain.

4.3 Magnetic Resonance Imaging (MRI)

Unlike methods that rely on the emission of radiation, Magnetic Resonance Imaging (MRI) is a modern imaging technique that makes use of the phenomenon of nuclear spin resonance and does not emit any ionizing radiation. This technique is very reliable and is widely used to study many organs such as the brain, spine, joints, soft tissues and cardiac cavities, which is often not possible with X-rays, ultrasound or even CT. The main idea of MRI is based on the fact that the hydrogen atoms in the human body are stimulated by an external magnetic field. As a result, protons, a component of hydrogen atoms, try to orient their spins to align with the external magnetic field. When this stimulation is stopped, the spins return to their original position while emitting a weak signal that can be picked up, recorded and processed in the form of an image by a computer system. Figure 4.4 shows a modern MRI and MRI image of a complete body.

4.4 Ultrasound Imaging

Ultrasound is an imaging technique that uses high-frequency sound waves inaudible to humans, greater than 20,000 periods per second (20 kHz), to explore body tissue. The ultrasound beam is emitted by a probe and transmitted at different speeds through body tissue (e.g., 1540 meters per second through soft tissue). Afterwards, the same probe receives several echoes from the part of the body examined. The round-trip time of the echo can be translated into the depth of the echo source. In other words, echoes coming from a deep region of the body are more attenuated than those coming from a superficial region. At the end, the reflected echoes are converted into detailed images visible on the screen. Ultrasound imaging is a popular and inexpensive tool used in various medical disciplines such as obstetrics, gynecology, pediatrics and neurology. Figure 4.5 shows a modern MRI and MRI image of a complete body.

4.5 Microscopy

Microscopy is a powerful biophysical tool commonly used in biological research. It is able to show the detail of small biological structures by achieving resolution at the cellular level. The microscopy is based on the concept of magnification, which is the ratio between the diameter of the image observed under the microscope and the diameter of the object in reality. If a 40X objective is used for example, a total magnification of 400X is obtained (10X $\times$ 40X = 400X). There are many several of microscopes, but for the sake of space, we will focus on optical microscopy, fluorescence microscopy, and electron microscopy. Below, we provide a brief description of these different microscopes.

4.5.1 Optical microscopy

Optical microscopy is an optical instrument used to generate magnified images of small objects using visible light and a lens system. The visible light is a strong point of optical microscopy that maintains its continuity to the present. On the one hand, light is cheap and plentiful and can be manipulated with relatively inexpensive laboratory equipment. On the other hand, light does not irreversibly change the electronic or atomic structure of the material with which it interacts. this allows the observation of natural processes in situ. The lens system is based on using an objective lens close to the specimen which produces a magnified intermediate image (from 10x to100x magnification). An additional magnification is achieved by an ocular (usually gives additional 10x magnification) to produce a virtual image. Hence, if one uses a 40X objective lens, a total magnification of 400X is obtained

(10X $\times$ 40X = 400X). Figure 4.6 shows an example of optical microscope and a magnified image of small object.

4.5.2 Fluorescent microscopy

Fluorescence microscopy is one of the most widely used instruments for examining biological samples. In addition to being an optical microscopy, it also uses fluorescence to study properties of organic or inorganic substances. Fluorescence microscopy uses a light source that is more intense than visible light, which can excite a fluorescent species in the sample of interest. The fluorescent species in turn emits lower energy light of a longer wavelength which produces the magnified image. Figure 4.7 shows an example of fluorescent microscopy and a magnified image of small object.

4.5.3 Electron microscopy

Electron microscopy is also a type of microscopy used to create highly magnified images, up to 2 million times. It is known by its ultrahigh-resolution images, superior to optical microscopes. The high resolution of electron microscopy images results from the use of electron beams as a source of illuminating radiation, instead of a beam of light, to investigate the samples properties and behavior. Like the optical microscopy, the electron microscopy uses a lens system to form images. The lens system controls the electron beam to sweep the surface of the sample to be analyzed. Figure 4.8 shows an example of electron microscopy and a magnified image of small object.

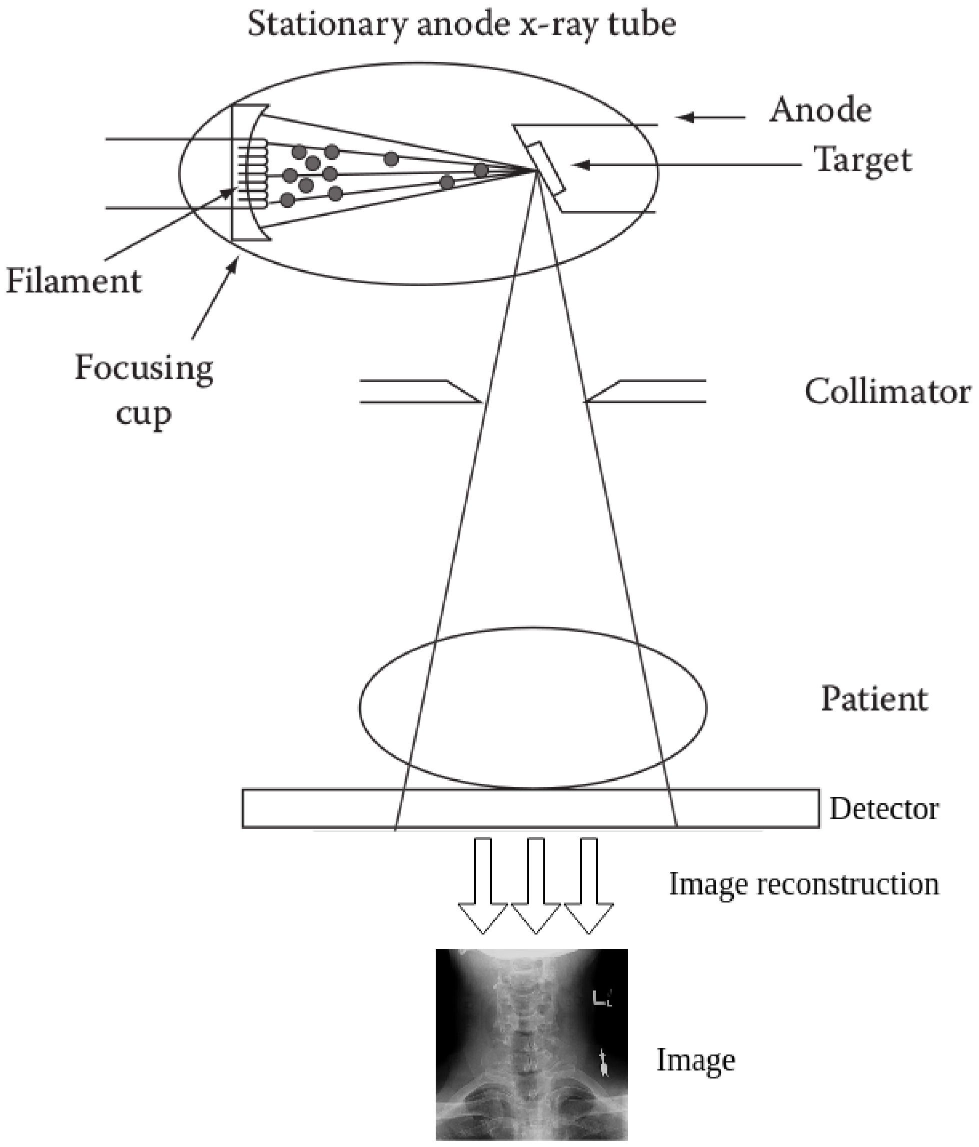


Figure 4.1: Basic principles of X-ray imaging (Figure adapted from book No-13: 978-1-4398-7034).

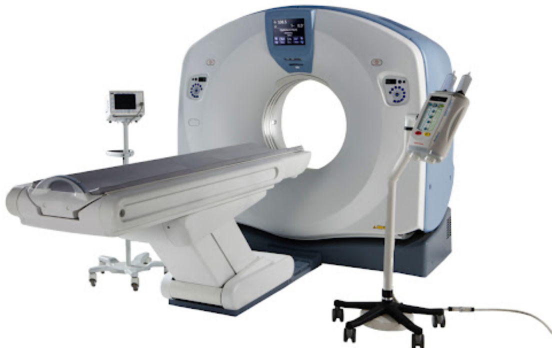


Figure 4.2: X-ray computed tomography. (Figure adopted from url http://www.amradusa.com/equipment/ct/).

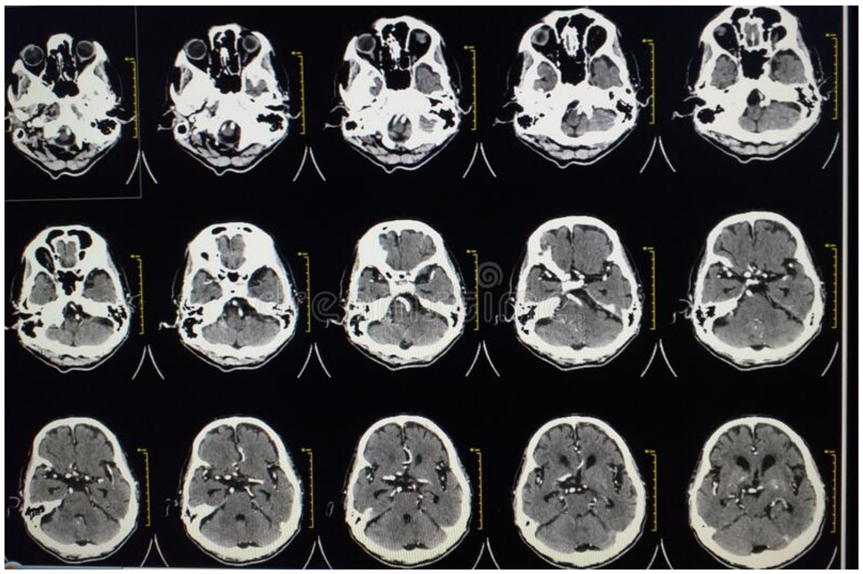


Figure 4.3: CT scan image of a man's brain. (Figure adopted from url https://www.dreamstime.com).

Figure 4.4: MRI machine with full body image. (Figure adapted from url https://irmba.fr/irm_corps_entier.html and https://www.siemens-healthineers.com).

Figure 4.5: Ultrasound machine with baby ultrasound image. (Figure adapted from url https://medi-plus.fr/ and https://fr.dreamstime.com/).

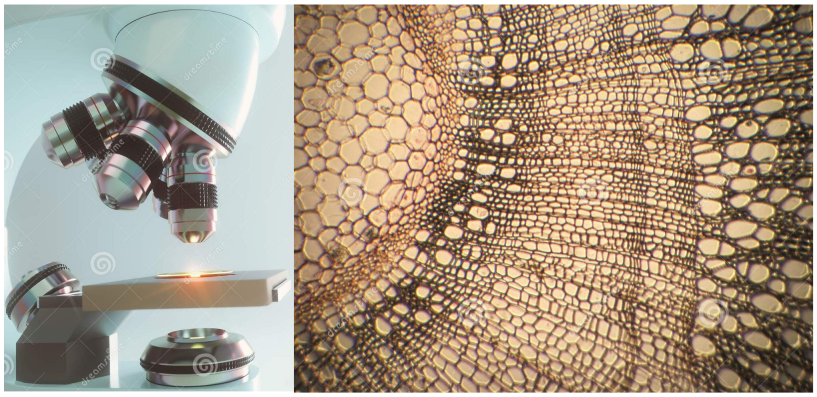


Figure 4.6: Optical microscopy with a plant tissue image. (Figure adapted from url https://fr.dreamstime.com/).

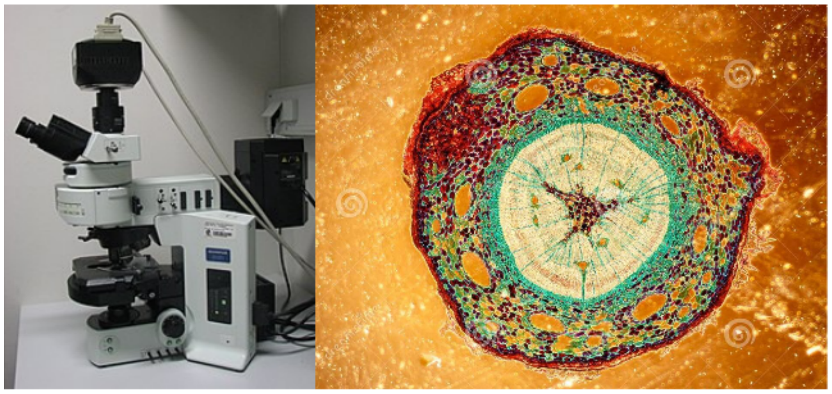


Figure 4.7: Fluorescent microscopy with image. (Figure adapted from Wikipedia).

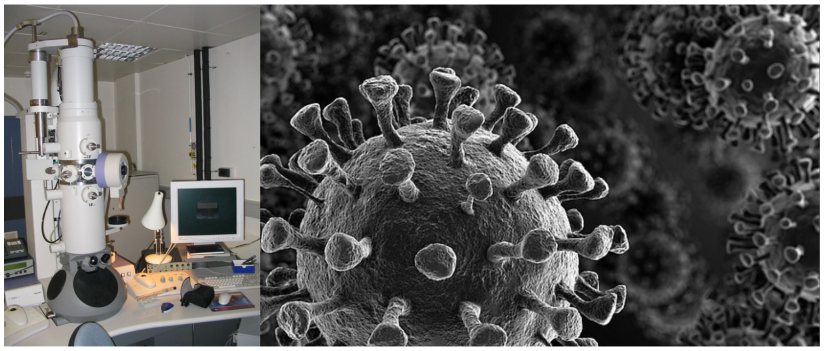
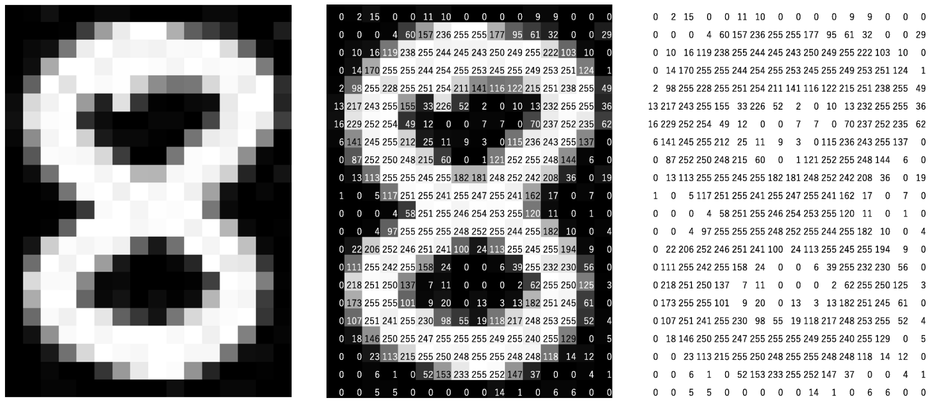


Figure 4.8: Electron microscopy with image. (Figure adapted from Wikipedia).

Chapter 5

Image Digitization

During the image capture process, light accumulates in the sensor cells, indicated by pixels. Each cell then converts to a numeric value. At the end of conversion process, one get a data table. The latter is known as the digital image. Figure 5.1 shows the conversion process. As shown in the figure,



0	2	15	0	0	11	10	0	0	0	0	9	9	0	0	0
0	0	0	4	60	157	236	255	255	177	95	61	32	0	0	29
0	10	16	119	238	255	244	245	243	250	249	255	222	103	10	0
0	14	170	255	255	244	254	255	253	245	255	249	253	251	124	1
2	98	255	228	255	251	254	211	141	116	122	215	251	238	255	49
13	217	243	255	155	33	226	52	2	0	10	13	232	255	255	36
16	229	252	254	49	12	0	0	7	7	0	70	237	252	235	62
6	141	245	255	212	25	11	9	3	0	115	236	243	255	137	0
0	87	252	250	248	215	60	0	1	121	252	255	248	144	6	0
0	13	113	255	255	245	255	182	181	248	252	242	208	36	0	19
1	0	5	117	251	255	241	255	247	255	241	162	17	0	7	0
0	0	0	4	58	251	255	246	254	253	255	120	11	0	1	0
0	0	4	97	255	255	255	248	252	255	244	255	182	10	0	4
0	22	206	252	246	251	241	100	24	113	255	245	255	194	9	0
0	111	255	242	255	158	24	0	0	6	39	255	232	230	56	0
0	218	251	250	137	7	11	0	0	0	2	62	255	250	125	3
0	173	255	255	101	9	20	0	13	3	13	182	251	245	61	0
0	107	251	241	255	230	98	55	19	118	217	248	253	255	52	4
0	18	146	250	255	247	255	255	255	249	255	240	255	129	0	5
0	0	23	113	215	255	250	248	255	255	248	248	118	14	12	0
0	0	6	1	0	52	153	233	255	252	147	37	0	0	4	1
0	0	5	5	0	0	0	0	0	14	1	0	6	6	0	0

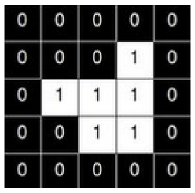
Figure 5.1: Conversion process.

the image on the left represents the first step of the conversion process. It is a grayscale image where each pixel represents a value from 0 to 255 (0: black, 255 white). For binary images, the pixel has two values, 0 for "black" and 1 for white". For color images, the pixel has three values from 0 to 255, representing Red-Green-Blue. Thus, any color can be encoded as a mixture of red, green, and blue, as presented in the table 5.1. Recall that the value is interpreted as the amount of light hitting a cell in sensor. Figure 5.2 presents

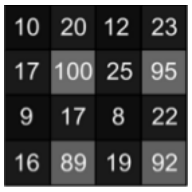
Table 5.1: Colors in numeric values.

Color	Red value	Green value	Blue value
black	0	0	0
White	255	255	255
Red	255	0	0
Blue	0	0	255
Green	0	128	0
Yellow	255	255	0
Purple	128	0	128
orange	255	165	0

an example of digital coding for a binary image, grayscale image and also an RGB image.



(a)



(b)

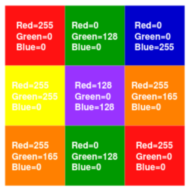

(c)

Figure 5.2: (a) Binary image. (b) Grayscale image. (c) RGB image

5.1 Sampling

Since the output of most image sensors represents an analog signal which can have infinite values requiring infinite memory to store them, it is important to proceed with the sampling. Sampling represents a process of recording an analog signal at regular discrete moments of time, ie, breaking up a continuous signal to a discrete signal. Therefore, a continuous image turns into a discrete image by selecting values on the continuous image and discarding others. This procedure is generally occur in the sensor. Figure 5.3 and Algorithm 5.1 show the sampling process [Code adopted from https://www.fatalerrors.org/].

Listing 5.1: Sampling in Python

```
import numpy as np
from skimage import data
```

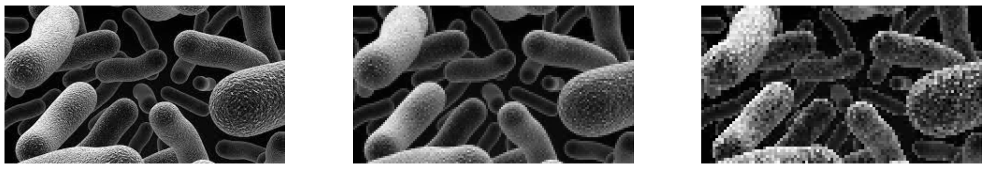

(a) (b) (c)

Figure 5.3: Example of sampling. (a) Original image of bacteria. (b)(c) Sampled images.

```
from matplotlib import pyplot as plt
image=data.coffee()
ratio=100
image1=np.zeros(( int(image.shape[0]/ratio),
                  int(image.shape[1]/ratio),
                 image.shape[2]),dtype='float32')
for i in  range(image1.shape[0]):
    for j in  range(image1.shape[1]):
        for k in  range(image1.shape[2]):
            delta=image[i*ratio:(i+1)*ratio,j*ratio:(j+1)*
            ratio,k]
            image1[i,j,k]=np.mean(delta)
```

5.2 Quantization

Since the sampling points represent continuous values, quantization allows these continuous values to be replaced by discrete values. Thus, an image encoded on 256 bits is quantized so that it is encoded on 8 bits. Figure 5.4 and Algorithm 5.2 show the quantization process [Code adopted from https://www.fatalerrors.org/].

Listing 5.2: Quantization in Python

```
from skimage import data
from matplotlib import pyplot as plt
image=data.coffee()
ratio=128  # Set quantization ratio
for i in  range(image.shape[0]):
    for j in  range(image.shape[1]):
        for k in  range(image.shape[2]):
            image[i][j][k]= int(image[i][j][k]/ratio)*ratio
```

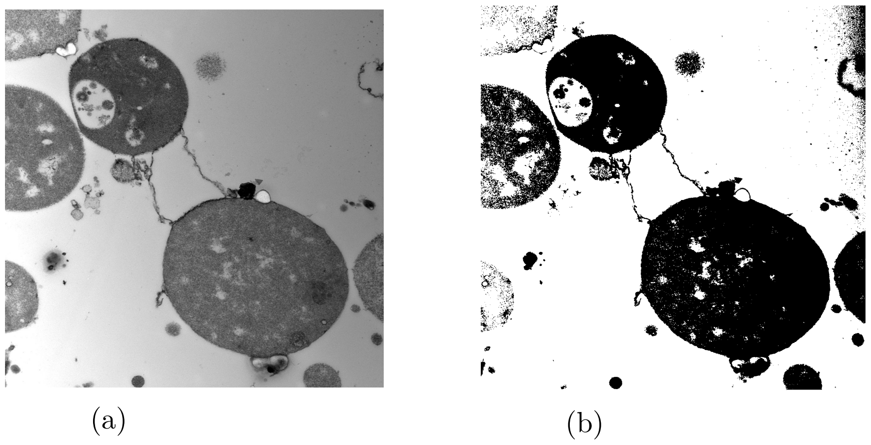


Figure 5.4: Example of quantization. (a) Original image of bacteria. (b) Quantized images.

5.3 Resolution

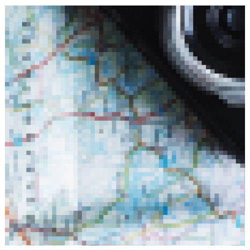
The quality of the image depends mainly on its definition, ie, the number of pixels it can contain horizontally and vertically. The more the number of pixels increases, the more the quality of the image increases. Thus, for ultra high definition 4K, we are talking about an image whose definition is 3840 pixels horizontally and 2160 pixels vertically, i.e. a definition of 3840×2160 pixels. Table 5.2 and Figure 5.5 summarize the main image definitions.

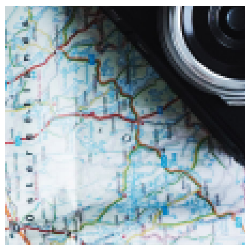
Table 5.2: Summary table of the main image definitions.

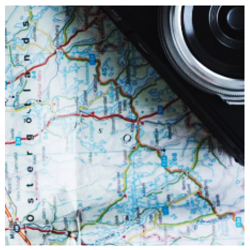
Name definition	**Norm**	**Image definition**	**Common names**
480p	DVD	720×480 pixels	SD, definition standard, quality DVD
720p	HD Ready	1280×720 pixels	HDTV, HD 720p, 720p, HD Ready
1080p	Full HD	1920×1080 pixels	HDTV 1080p, HD 1080p, 1080p, Full HD
2160p	UHDTV1	3840×2160 pixels	4K, UHD 4K, Ultra HD 4K, UHD-4K, 2160p
4320p	UHDTV2	7680×4320 pixels	8K, UHD 8K, Ultra HD 8K, UHD-8K, 4320p

Now suppose two images that have the same definitions, for example 3840×2160 pixels, but the first has 75 inches diagonal (1 inch = 2.54 cen-

(a) 480p (DVD)


(b) 720p (HD Ready)


(c) 1080p (Full HD)


(d) 2160p (4k)

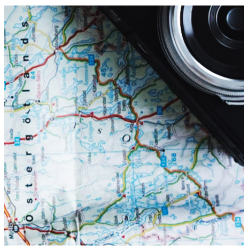

(e) 4320p (8k)

Figure 5.5: The effect of increasing the definition.

timeters), the equivalent of 51.2 pixels per inch, and the other 55 inches or 69.8 dpi (see Figure 5.6). The number of pixels is identical for the two im-

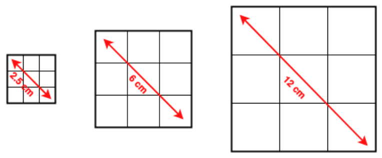


Figure 5.6: Image clarity according to pixel size.

ages but they are more tight on the image of smaller dimension. Thus, the first image will be clearness than the second. In fact, in this example we are talking about the concept of resolution. The higher the resolution, the smaller and more numerous the pixels, and the finer the image. Figure 5.7 illustrates an example which more clearly presents the principle of resolution. The resolution is calculated by the following equation (Eq. 5.1).

$$Resolution\;(dots\;per\;inch : dpi) = \frac{number\;of\;dots}{number\;of\;inches} \tag{5.1}$$

Thus, the resolution of an image that is 2837 pixels wide and its actual size is 10 cm wide, is: 720 dpi.

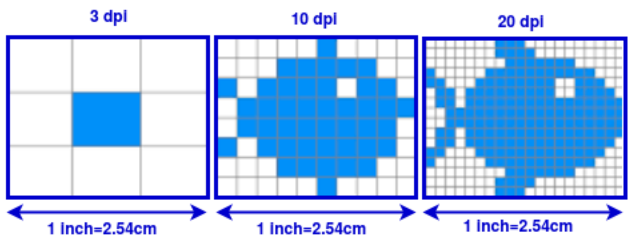


Figure 5.7: Resolution vs Pixel Dimension.

Proof. dots per inch? = 2837 / (10 / 2,54) = 720 dpi

Computer screens are usually used at 72 or 75 dpi. professional printing equipment often operate at 4800 dpi or more. This is essential to have very good quality prints, such as magazines or books. Note that the more pixels there are in the image, the more memory space increases to store the pixels.

Chapter 6

Basic image processing algorithms

Image processing consists of performing certain operations on an image, in order to extract useful information from it or to obtain an improved image. Generally speaking, there are three levels of image processing operations: *Low level*, relates to primitive operations (eg noise reduction, contrast enhancement, etc.). *Intermediate level*, operations of extracting attributes (for example, edges, contours, regions, etc.). *High Level*, operations of analysis and interpretation of the content of a scene. Each level encompasses a set of algorithms. In this section, we present some simple and useful algorithms of each level implemented using python.

6.1 Low level algorithms

6.1.1 Read image

The digital image sent from sensor to processing unit is often stored in a file in the computer. The file is composed of *metadata* (file name, size, type, and so on), *data*, and *EOF*, a special character that indicates the end of the file. In order to process the file, one must first upload it by finding the file path. Algorithm 6.1 loads the image file, reports the format, mode, and size, then shows the result (see Fig.6.1) [Code adopted from `https://machinelearningmastery.com/`].

Listing 6.1: Read image in Python

```python
# load and show an image with Pillow
from PIL import Image
# load the image
image = Image. open('File.jpg')
# summarize some details about the image
print(image. format)
print(image.mode)
print(image.size)
# show the image
image.show()
```

Result:

Listing 6.2:

```
JPEG
RGB
(640, 360)
```

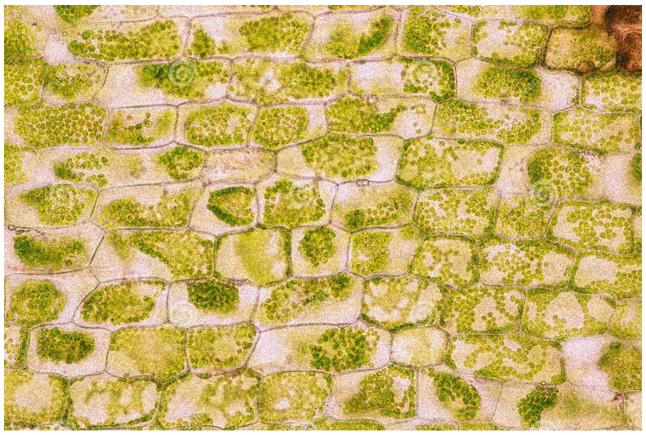


Figure 6.1: Uploaded image (plant cells).

6.1.2 Resize image

Images captured by the sensor often vary in size, therefore, we need to establish a base size for all images. On another side, image changes from 1392 × 1040 to 696 × 520 pixels results in a reduction in image size, and therefore a reduction in the time required to process the resized image. Sometimes resizing an image is necessary to find out more information by exploring all possible angles of the image. Algorithm 6.3 loads the image, reports the shape of the image, then resizes it to have a width and height of 200 pixels (see Fig.6.2) [Code adopted from https://machinelearningmastery.com/].

Listing 6.3: Resize image in Python

```python
# resize image and force a new shape
from PIL import Image
# load the image
image = Image. open('File.jpg')
# report the size of the image
print(image.size)
# resize image and ignore original aspect ratio
img_resized = image.resize((200,200))
# report the size of the thumbnail
print(img_resized.size)
```

Result:

Listing 6.4:

```
(640, 360)
(200, 200)
```

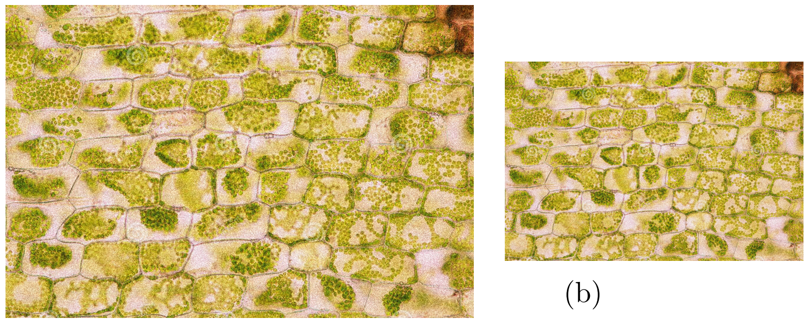

(a) (b)

Figure 6.2: Resized image. (a) Original image. (b) Image after resizing.

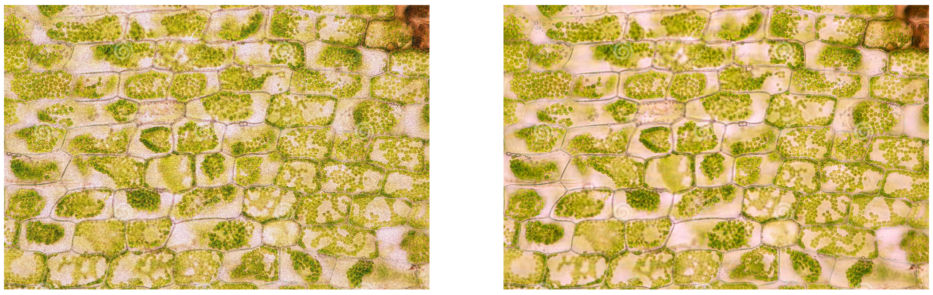
6.1.3 Remove noise

Any digital image is subject to some perturbations due to different intrinsic (i.e., sensor) and extrinsic (i.e., environment) conditions which are often not possible to avoid in practical situations. Denoising process is important to smooth images and remove unwanted noise. Algorithm 6.5 loads the image and remove noise (see Fig.6.3) [Code adopted from https://www.geeksforgeeks.org/].

Listing 6.5: Denoise image in Python

```python
# importing libraries
import numpy as np
import cv2
from matplotlib import pyplot as plt
# Reading image from folder where it is stored
img = cv2.imread('File.jpg')
# denoising of image saving it into dst image
dst = cv2.fastNlMeansDenoisingColored(img,
 None, 10, 10, 7, 15)
# Plotting of source and destination image
plt.subplot(121), plt.imshow(img)
plt.subplot(122), plt.imshow(dst)
plt.show()
```

Result:



(a) (b)

Figure 6.3: Denoised image. (a) Original (noisy) image. (b) After removing noise.

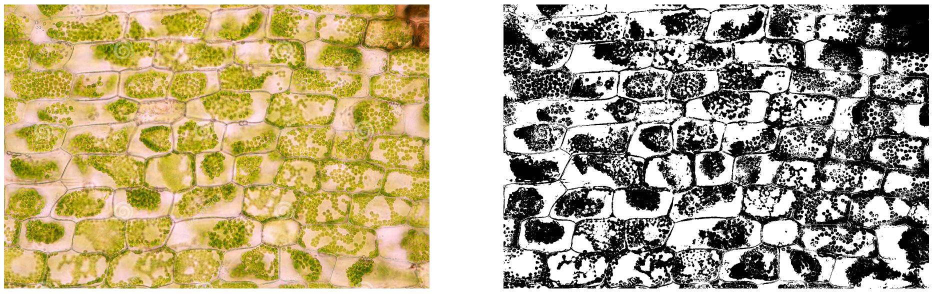
6.1.4 Binarization (thresholding)

Consists of converting the original image to just two levels of gray (black and white) in order to speed up future treatment process. Thresholding can be used to detect e.g., cells, proteins, DNA, glycogen, acids, etc. Algorithm 6.6 shows the binarization steps (see Fig.6.4) [Code adopted from `https://www.geeksforgeeks.org/`].

Listing 6.6: Binarization in Python

```python
# organizing imports
import cv2
import numpy as np
# path to input image is specified and image
# is loaded with imread command
image1 = cv2.imread('File.jpg')
# cv2.cvtColor is applied over the
# image input with applied parameters
# to convert the image in grayscale
img = cv2.cvtColor(image1, cv2.COLOR_BGR2GRAY)
# applying  thresholding  technique on the input
# image all pixels value above 120 will be set to
#255
ret, thresh = cv2.threshold(img, 120, 255,
cv2.THRESH_BINARY)
# the window showing output image
cv2.imshow('Binary Threshold', thresh)
# De-allocate any associated memory usage
if cv2.waitKey(0) & 0xff == 27:
cv2.destroyAllWindows()
```

Result:



(a) (b)

Figure 6.4: Binarization of the image. (a) Original image. (b) binarized image.

6.2 Intermediate level algorithms

6.2.1 Edge detection

An important step in image processing, used to separate the objects from each other before identifying their content (see Fig.6.5). For instance, the separation of nuclei from cytoplasm plays an essential role in a wide spectrum of clinical and research settings. Below the edge detection algorithm 6.7 [Code adopted from `https://www.geeksforgeeks.org/`].

Listing 6.7: Edge detection in Python

```python
from PIL import Image , ImageFilter
img = Image. open(r"File.jpg")
# Converting the image to greyscale , as Sobel
#Operator requires input image
#to be of mode Greyscale (L)
img = img.convert("L")
# Calculating Edges using the passed laplican Kernel
final = img. filter(ImageFilter.Kernel((3, 3),
(-1, -1, -1, -1, 8, -1, -1, -1, -1), 1, 0))
final.save("EDGE_sample.png")
```

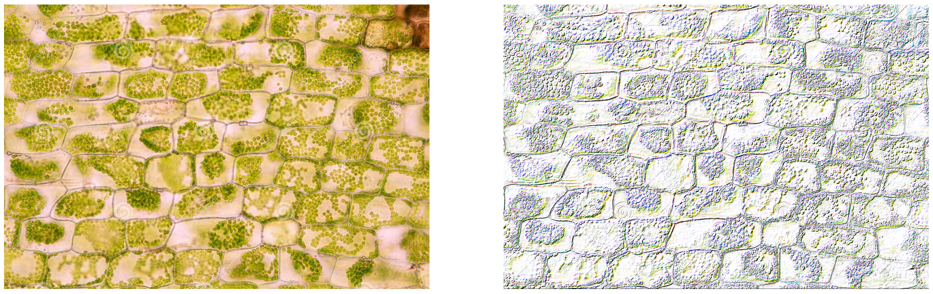

(a) (b)

Figure 6.5: Edge detection (a) image input. (b) image output.

6.2.2 Discrete Fourier Transform (DFT)

DFT was introduced by Jean Baptiste Joseph Fourier in 1822. It is one of the most important mathematical theories in signal processing and probably one of the most commonly used techniques in image processing. DFT determines the frequency spectra of digital signals. Let $F(k)$ be a Frequency spectra of a signal $f(n)$, $n = 0, ..., N-1$. The DFT is mathematically defined as follows:

$$F(k) = \sum_{n=0}^{N-1} f(n) e^{-\frac{2\pi}{N}kn} \tag{6.1}$$

For images, 2D DFT represents a natural extension of Fourier analysis to phenomena in the R^2 space. It is used to convert an image from spatial domain to frequency domain, in other words, it separates high frequency coefficients from low frequency coefficients (see Fig.6.6). 2D DFT is mathematically defined as follows:

$$F(u,v) = \frac{1}{MN} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} f(m,n) e^{-i2\pi(um/M+vn/N)} \tag{6.2}$$

Below the 2D DFT algorithm 6.8 [Code adopted from https://thepythoncodingbook.com/].

Listing 6.8: 2D Discrete Fourier Transform (2D DFT) in Python

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

image_filename = "image.png"

def calculate_2dft( input):
    ft = np.fft.ifftshift( input)
    ft = np.fft.fft2(ft)
    return np.fft.fftshift(ft)

# Read and process image
image = plt.imread(image_filename)
image = image[:, :, :3].mean(axis=2)  # Convert to grayscale
plt.set_cmap("gray")
ft = calculate_2dft(image)
plt.subplot(121)
plt.imshow(image)
plt.axis("off")
plt.subplot(122)
plt.imshow(np.log( abs(ft)))
plt.axis("off")
plt.show()
```

Result:

(a)

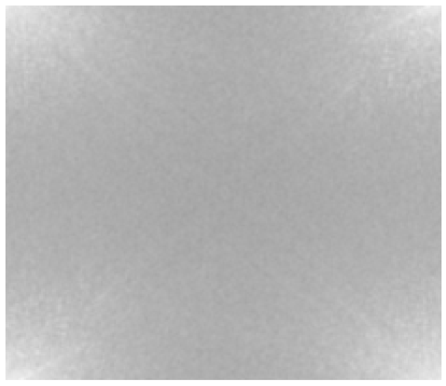

(b)

Figure 6.6: 2D DFT (a) image input. (b) image output.

6.2.3 Discrete Cosine Transform (DCT)

DCT was first proposed by Nasir Ahmed in 1972. It is a widely used transformation technique in signal processing and image processing. DCT is close to DFT but its projection kernel is a cosine which creates real coefficients, unlike the DFT, whose kernel is a complex exponential and therefore creates complex coefficients. (see Fig.6.7). 2D DCT is mathematically defined as follows:

$$F(p,q) = \alpha_p \alpha_q \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} A_{mn} cos \frac{\pi(2m+1)p}{2M} cos \frac{\pi(2n+1)q}{2N} \tag{6.3}$$

where 0 $\leq$ p $\leq$ M-1, 0 $\leq$ q $\leq$ N-1 and

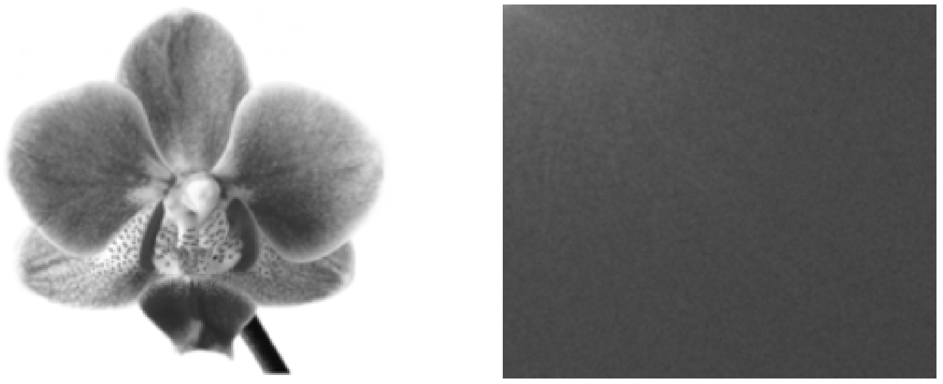
$$\alpha_p = \begin{cases} \frac{1}{\sqrt{M}} & p = 0 \\ \sqrt{\frac{2}{M}} & 1 \leq p \leq M-1. \end{cases}$$

$$\alpha_q = \begin{cases} \frac{1}{\sqrt{N}} & q = 0 \\ \sqrt{\frac{2}{N}} & 1 \leq q \leq N-1. \end{cases}$$

Below the 2D DCT algorithm 6.9 [Code adopted from `https://docs.scipy.org/`].

Listing 6.9: 2D Discrete Cosine Transform (2D DCT) in Python

```python
from scipy.fftpack import dct, idct

from skimage.io import imread
from skimage.color import rgb2gray
import numpy as np
import matplotlib.pylab as plt

image_filename = "fruit.png"

# implement 2D DCT
def dct2(a):
    return dct(dct(a.T, norm='ortho').T, norm='ortho')

# Read and process image
image = plt.imread(image_filename)
imF = dct2(image)
plt.show()
```



(a) (b)

Figure 6.7: 2D DCT (a) image input. (b) image output.

6.2.4 Discrete Wavelet Transform (DWT)

DWT was first invented by Alfréd Haar in 1909, but the most commonly used DWT was formulated by Ingrid Daubechies in 1988. Its main advantage over Fourier transforms is its temporal resolution as it captures both frequency and location information. Generally, the DWT can be expressed by the following equation:

$$F(a,b) = \int_{-\infty}^{\infty} f(x)\psi^*_{a,b} x d(x) \tag{6.4}$$

DWT is a representation of a square-integrable function by a certain orthonormal series generated by a wavelet. The Two-dimensional DWT (2D DWT) is an extension of one-dimensional DWT which is of particular interest to image processing and computer vision applications (see Fig.6.8).

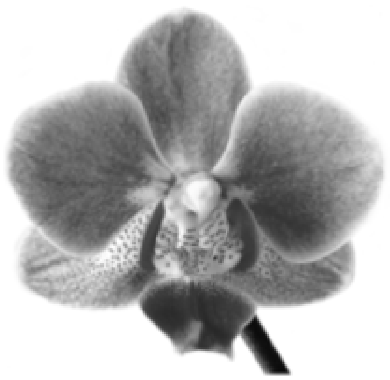
Below the Wavelet transform algorithm 6.10 [Code adopted from https://mahotas.readthedocs.io/].

Listing 6.10: Wavelet transform in Python

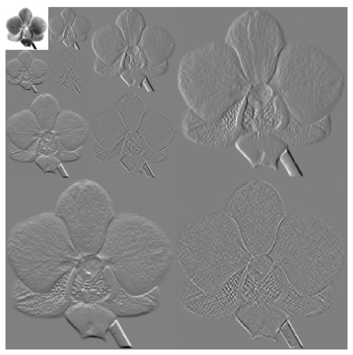
```
# importing various libraries
import numpy as np
import mahotas
import mahotas.demos
from mahotas.thresholding import soft_threshold
from matplotlib import pyplot as plt
from os import path
# loading image
f = mahotas.demos.load('File', as_grey = True)
# making ply gray
plt.gray()
# showing image
print("Image")
plt.imshow(f)
plt.show()
# Transform using D8 Wavelet to obtain transformed
#image t
t = mahotas.daubechies(f, 'D8')
# showing transformed image
print("Transformed Image")
plt.imshow(t)
plt.show()
```

6.3 High level algorithms

Image analysis and interpretation includes classification techniques, recognition, extraction or feature selection, etc. We cover all of these principles in the following chapters.



(a)



(b)

Figure 6.8: Wavelet transform (a) image input. (b) image output.

Chapter 7

Suggested readings

1. Greenberg, J. (Ed.). (2018). X-Ray Diffraction Imaging: Technology and Applications. CRC Press.

2. Najarian, K., & Splinter, R. (2012). Biomedical signal and image processing. Taylor & Francis.

3. Haidekker, M. (2010). Advanced biomedical image analysis. John Wiley & Sons.

4. Sternberg, S. R. (1983). Biomedical image processing. Computer, 16(01), 22-34.

5. Guan, L. (Ed.). (2017). Multimedia image and video processing. CRC press.

6. Wheeler, A., & Henriques, R. (Eds.). (2017). Standard and super-resolution bioimaging data analysis: a primer. John Wiley & Sons.

7. Pitas, I. (2000). Digital image processing algorithms and applications. John Wiley & Sons.

8. Bhabatosh, C. (1977). Digital image processing and analysis. PHI Learning Pvt. Ltd.

9. Gonzalez, R. C., & Woods, R. E. (2008). Digital image processing, 3rd edition, Prentice Hall, Upper Saddle River, NJ.

10. Castleman, K. R. (1993). Digital image processing.

11. Bovik, A. C. (2010). Handbook of image and video processing. Academic press.

12. Haskell, B. G., Puri, A., & Netravali, A. N. (1996). Digital video: an introduction to MPEG-2. Springer Science & Business Media.

13. Poynton, C. (2012). Digital video and HD: Algorithms and Interfaces. Elsevier.

14. Winkler, S. (2005). Digital video quality: vision models and metrics. John Wiley & Sons.

15. Umesh, P. (2012). Image processing in python. CSI Communications, 23(2).

16. Kinser, J. M. (2018). Image Operators: Image Processing in Python. CRC Press.

17. Chityala, R., & Pudipeddi, S. (2020). Image processing and acquisition using Python. Chapman and Hall/CRC.

Part III :

Machine learning

Chapter 8

Object Recognition

Object recognition is a computer vision technique very useful in a variety of applications such as identification and diagnosis of diseases, discovery of antibiotics, revealing antibiotic mechanisms of action, prediction of glycemic responses, prediction of gestational age and time to delivery in pregnant women, the plant science and plant breeding, location of pollution zones, etc. Object recognition task is inspired by humans. Humans recognize objects very easily, in a robust, selective and fast manner. This is due to the power of the human brain and the neurobiological foundations on which he relies. Formally, humans can perform object recognition based on previously acquired information, called *reference information* or *labeled data*, without which new objects cannot be recognized. Human's knowledge of the *lion* makes him flee from the *puma*, which is very similar to it. In fact, some features of the lion are similar to the puma, for instance the size of the body, the shape of the nose, mouth and eyes. The information obtained by human about the lion is only these features, which he will inevitably find in the image of the puma, so he classifies it as dangerous. More precisely, human memory cannot save all the data on the lion. In contrast, it saves important features, such as those listed above, and uses them to make further comparisons with other animals. Although objects recognition by machines is a very complicated operation, it is nevertheless experiencing a rapid adoption rate in various and diverse industries thanks to the recent advances in hardware and deep learning technology. Figure 8.1 shows an example of object recognition.

Generally, the main steps of object recognition are presented in the following diagram:

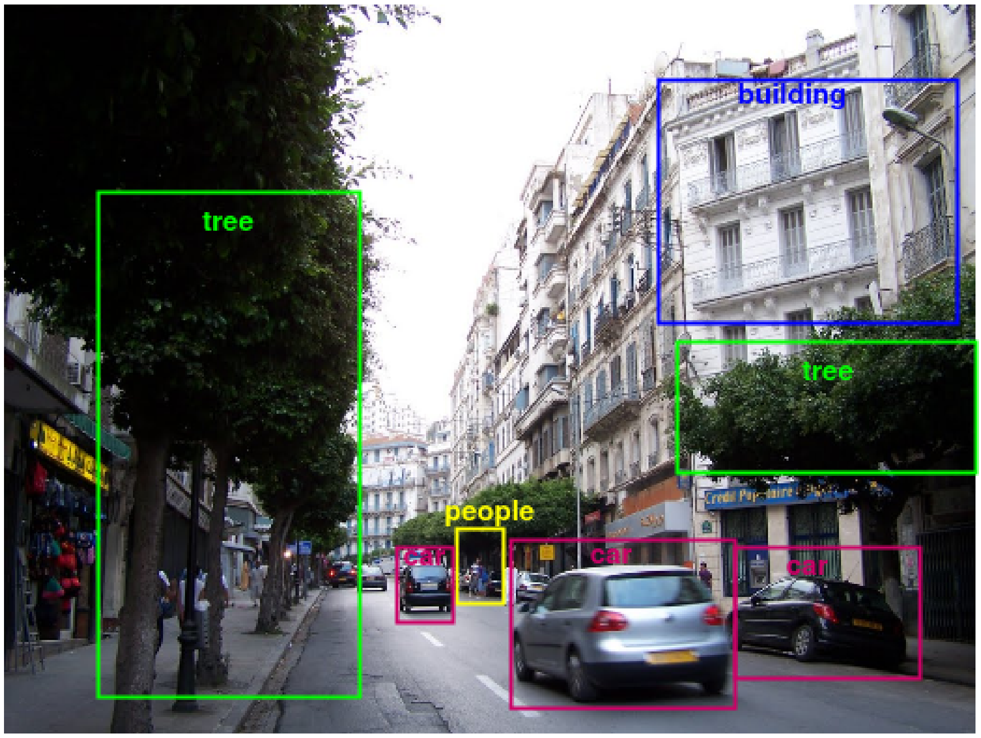


Figure 8.1: Object recognition

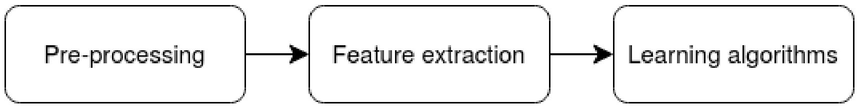


Figure 8.2: Steps in object recognition

- *Pre-processing:* to correct the image, improve contrast, etc., roughly, prepare the image to be recognized. This step is already discussed in chapter 6.
- *Feature extraction:* extracting the important information contained in the image to help classification.
- *Learning algorithm for classification:* learn classification algorithms how to take into account the features extracted in step 2 to generate class labels as an output.

Both steps, 2 and 3, will be well discussed in the following chapters. In the following, we list down some object detection algorithms one must know.

8.1 Faster R-CNN

Faster R-CNN is a Deep Convolutional Neural Networks (DCNN) developed from traditional Artificial Neural Networks (ANN) and used for object detection. Before explaining the Faster R-CNN method, let us explain the ANN and the CNN methods.

- **Artificial Neural Networks (ANN)**: The ANN is a computational system inspired by the Biological Neural Networks (BNN) that can be used for predictive modeling, adaptive control and other applications. Basically, the biological neuron is a cell that can be found in brain. Hence, the BNN consists of synapse, dendrites that receive signals (impulses) from other neurons and axon that transmits the signals to other cells. In the other side, the ANN consists of the inputs which replaces the dendrites in the BNN, the hidden layer or f(x), where all the computations are performed and the outputs that replace the axon. Figure 8.3 shows the components of the BNN and the ANN. The ANN computes the sum of the inputs multiplied by its weight to output the result. The simplest form of ANN is the Perceptron. Algorithm 8.1 shows the Perceptron steps [Code adopted from `https://scikit-learn.org/`].
.

Listing 8.1: ANN perceptron in Python

```python
from sklearn.datasets import load_digits
from sklearn.linear_model import Perceptron
X, y = load_digits(return_X_y=True)
clf = Perceptron(tol=1e-3, random_state=0)
clf.fit(X, y)
clf.score(X, y)
```

- **Convolutional Neural Network (CNN):** CNN is a type of artificial neural network mainly developed to solve difficult image-driven pattern recognition tasks. It consists of three basic components: convolution layer, pooling layer and the output layer. Figure 8.4 and 8.5 shows the Concept of CNN.

  1. *Convolutional layer:* In this layer a convolution operation is applied to the input in order to to extract the high-level features such as edges, from the input image.

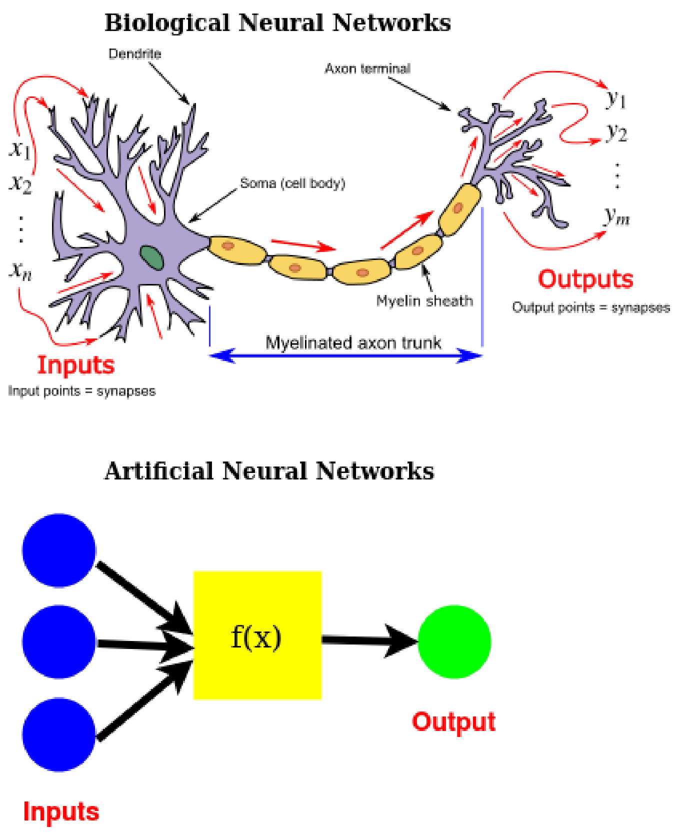


Figure 8.3: Comparison between the BNN and the ANN [figure adapted from Wikipedia].

2. *Pooling layer:* In this layer the dimension of the inputs is roughly reduced using a simple operation sush as the *max* that takes the largest value in the pool region and neglects all others.
3. *Fully-connected layer:* This layer is used to classify the data.

Algorithm 8.2 shows the CNN steps [Code adopted from `https://www.tensorflow.org/tutorials/images/cnn/`].

Listing 8.2: CNN in Python

```
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt
(train_images, train_labels), (test_images, test_labels) =
```

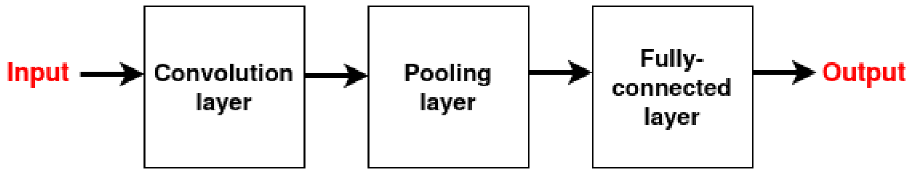


Figure 8.4: Components of CNN.

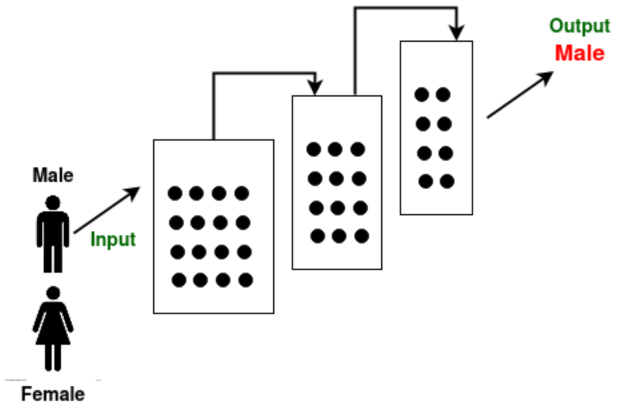


Figure 8.5: CNN – Image Classification.

```
datasets.cifar10.load_data()

# Normalize pixel values to be between 0 and 1
train_images, test_images=train_images/255.0,
test_images/255.0
class_names = ['airplane', 'automobile', 'bird', 'cat',
'deer','dog', 'frog', 'horse', 'ship',
'truck']
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu',
input_shape=(32, 32, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.summary()
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10))
model. compile(optimizer='adam',
    loss=tf.keras.losses.SparseCategoricalCrossentropy
    (from_logits=True),metrics=['accuracy'])

history = model.fit(train_images, train_labels, epochs=10,
                    validation_data=(test_images,
                    test_labels))
test_loss, test_acc = model.evaluate(test_images,
test_labels, verbose=2)
```

The Faster R-CNN is an extension of CNN which is also based on the ANN. It consists of of two components: a fully convolutional Region Proposal Network (RPN) that proposes candidate regions followed by the Fast R-CNN detector that uses the proposed regions. Algorithm 8.3 shows the Faster R-CNN steps [Code adopted from https://www.analyticsvidhya.com/blog/].

Listing 8.3: Faster R-CNN in Python

```
# as the images are in train_images folder, add train_images
before the image name
for i in range(data.shape[0]):
    data['format'][i] = 'train_images/' + data['format'][i]

# add xmin, ymin, xmax, ymax and class as per the format
required for i in range(data.shape[0]):
    data['format'][i] = data['format'][i] + ',' +
     str(train['xmin'][i]) + ',' + str(train['ymin']
    [i]) + ',' + str(train['xmax'][i]) + ',' +
     str(train['ymax'][i]) + ',' + train['cell_type'][i]

data.to_csv('annotate.txt', header=None, index=None, sep=' ')
cd keras-frcnn
python train_frcnn.py -o simple -p annotate.txt
python test_frcnn.py -p test_images
```

8.2 YOLO

You Only Look Once (YOLO) is a state-of-the-art neural network-based real-time object detection system. It is extremely fast compared to other techniques. Although several techniques have proven their superiority in the field of object detection, such as Faster R-CNN, nevertheless YOLO has proven their superiority over them. On the one hand, YOLO is extremely fast compared to various techniques, including Faster R-CNN. On the other hand, YOLO takes the entire image into account when making its predictions, unlike techniques based on sliding window and region proposal. Figure 8.6 shows the Concept of YOLO.

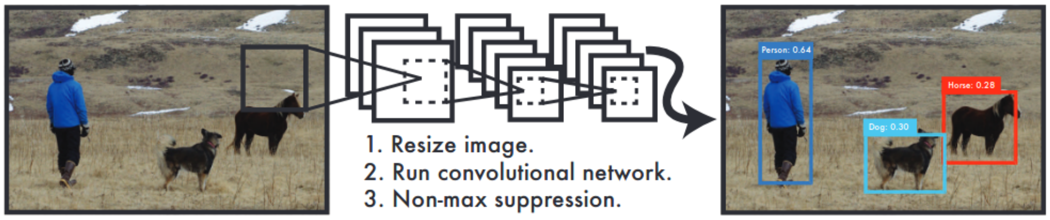


Figure 8.6: YOLO Principle [Figure adopted from the CVPR conference].

Algorithm 8.4 shows the YOLO steps [Code adopted from https://cloudxlab.com/blog/object-detection-yolo-and-python-pydarknet/].

Listing 8.4: YOLO in Python

```python
import numpy as np
import time
import cv2
INPUT_FILE='data/dog.jpg'
OUTPUT_FILE='predicted.jpg'
LABELS_FILE='data/coco.names'
CONFIG_FILE='cfg/yolov3.cfg'
WEIGHTS_FILE='yolov3.weights'
CONFIDENCE_THRESHOLD=0.3
LABELS =  open(LABELS_FILE).read().strip().split("\n")
np.random.seed(4)
COLORS = np.random.randint(0, 255, size=( len(LABELS), 3),
        dtype="uint8")
net = cv2.dnn.readNetFromDarknet(CONFIG_FILE, WEIGHTS_FILE)
image = cv2.imread(INPUT_FILE)
(H, W) = image.shape[:2]
# determine only the *output* layer names that we need from
# YOLO
ln = net.getLayerNames()
ln = [ln[i[0] - 1] for i in net.getUnconnectedOutLayers()]
blob = cv2.dnn.blobFromImage(image, 1 / 255.0, (416, 416),
        swapRB=True, crop=False)
net.setInput(blob)
start = time.time()
layerOutputs = net.forward(ln)
end = time.time()
```

```
print("[INFO] YOLO took {:.6f} seconds". format(end - start))
# initialize our lists of detected bounding boxes,
confidences, and class IDs, respectively
boxes = []
confidences = []
classIDs = []
# loop over each of the layer outputs
for output in layerOutputs:
        # loop over each of the detections
        for detection in output:
                # extract the class ID and confidence (i.e.,
                probability) of
                # the current object detection
                scores = detection[5:]
                classID = np.argmax(scores)
                confidence = scores[classID]
                # filter out weak predictions by ensuring the
                #detected probability is greater than the
                #minimum
        probability if confidence > CONFIDENCE_THRESHOLD:
                        # scale the bounding box coordinates
                        #back relative to the size of the image,
                        #keeping in mind that YOLO actually
                        # returns the center (x, y)-coordinates
                        #of the bounding
                        # box followed by the boxes' width and
                        #height
                        box = detection[0:4] *
                        np.array([W, H, W, H])
                        (centerX, centerY, width, height)
                        = box.astype("int")
                        # use the center (x, y)-coordinates to
                        #derive the top and left corner of the
                        #bounding box
                        x =  int(centerX - (width / 2))
                        y =  int(centerY - (height / 2))
                        # update our list of bounding box
                        #coordinates, confidences,
                        # and class IDs
                        boxes.append([x, y,  int(width),
                         int(height)])
                        confidences.append( float(confidence))
                        classIDs.append(classID)
# apply non-maxima suppression to suppress weak, overlapping
#bounding
```

```
# boxes
idxs = cv2.dnn.NMSBoxes(boxes, confidences,
CONFIDENCE_THRESHOLD,
        CONFIDENCE_THRESHOLD)
# ensure at least one detection exists
if  len(idxs) > 0:
        # loop over the indexes we are keeping
        for i in idxs.flatten():
                # extract the bounding box coordinates
                (x, y) = (boxes[i][0], boxes[i][1])
                (w, h) = (boxes[i][2], boxes[i][3])
                color = [ int(c) for c in COLORS[classIDs[i]]]
                cv2.rectangle(image, (x, y), (x + w, y + h),
                color, 2)
                text = "{}: {:.4f}". format(LABELS[classIDs[i]],
                confidences[i])
                cv2.putText(image, text, (x, y - 5),
                cv2.FONT_HERSHEY_SIMPLEX,0.5, color, 2)
# show the output image
cv2.imwrite("example.png", image)
```

Chapter 9

Classification

Classification is a machine learning-based technique used to organize data into relevant categories. It is of great importance as it covers a wide range of fields such as biology, medicine, chemistry, agriculture, etc. In biology, the classification is used for two totally different purposes, often in combination, namely, the identification and creation of natural groups. Classification is roughly divided into three main categories including, binary classification, multiclass classification and multilabel classification. In this chapter, we present the different most used algorithms of each category.

9.1 Naive Bayes

Naive Bayes is a simple and very relevant supervised classifier for real, discrete and streaming data. It is based on *Bayes' theorem* and a set of conditional independence assumptions between the features. Mathematically, Bayes' theorem is expressed by the following equation (9.1):

$$P(\mathbf{X} \mid \mathbf{Y}) = \frac{P(\mathbf{Y} \mid \mathbf{X})P(\mathbf{X})}{P(\mathbf{Y})} = \frac{P(\mathbf{X} \cap \mathbf{Y})}{P(\mathbf{Y})} \tag{9.1}$$

$$P(\mathbf{Y} \mid \mathbf{X}) = \frac{P(\mathbf{X} \cap \mathbf{Y})}{P(\mathbf{X})} \tag{9.2}$$

$$P(\mathbf{X} \cap \mathbf{Y}) = \frac{cardinal(\mathbf{X} \cap \mathbf{Y})}{cardinal(\Omega)} \tag{9.3}$$

$$P(\mathbf{X}) = \frac{cardinal(\mathbf{X})}{cardinal(\Omega)} \tag{9.4}$$

Where, $P(\mathbf{X})$ and $P(\mathbf{Y})$ are the probabilities of observing $\mathbf{X}$ and $\mathbf{Y}$ respectively without any given conditions. $P(\mathbf{X} \cap \mathbf{Y})$ is the probability of both $\mathbf{X}$ and $\mathbf{Y}$ being true. $cardinal(\Omega)$ is the data size. $cardinal(\mathbf{X})$ is the number of elements in $\mathbf{X}$. $P(\mathbf{X} \mid \mathbf{Y})$ is the probability of $\mathbf{X}$ occurring given that $\mathbf{Y}$ is true. $P(\mathbf{Y} \mid \mathbf{X})$ is the probability of $\mathbf{Y}$ occurring given that $\mathbf{X}$ is true.

For the sake of simplicity, let us consider the following example. The table below presents a set of 100 persons, males and females. We are looking to know the conditional probability that a person is affected by COVID-19 given that he is a ”male”.

	Male	Female	Total
COVID19$^+$	8	12	20
COVID19$^-$	32	48	80
Total	40	60	100

$$P(\mathbf{COVID19^+} \mid \mathbf{Male}) = \frac{P(\mathbf{COVID19^+} \cap \mathbf{Male})}{P(\mathbf{Male})} = \frac{12}{60} = 0.2.$$

Now, we come back to **Naive Bayesian classifier**. Mathematically, Naive Bayes is expressed by the following equation (9.5):

$$P(\mathbf{C} \mid \mathbf{X}) = P(\mathbf{C} \mid \mathbf{x_1}) \times P(\mathbf{C} \mid \mathbf{x_2}) \times ... \times P(\mathbf{C} \mid \mathbf{x_n}) \tag{9.5}$$

$$P(\mathbf{C} \mid \mathbf{x_i}) = \frac{P(\mathbf{x_i} \mid \mathbf{C})P(\mathbf{C})}{P(\mathbf{x_i})} \tag{9.6}$$

Where, $P(\mathbf{C})$ is the prior probability of class, $P(\mathbf{x_i})$ is the prior probability of predictor, $P(\mathbf{C} \mid \mathbf{X})$ is the posterior probability of class ($\mathbf{C}$: target) given predictor ($\mathbf{x_i}$, attributes), $P(\mathbf{x_i} \mid \mathbf{C})$ is the likelihood which is the probability of predictor given class.

Now back to COVID-19 example. We are looking to know the conditional probability that a person is affected by COVID-19 given that he is a male, with a high temperature and cough.
$P(\mathbf{COVID19^+})$=0.5,
$P(\mathbf{Male})$=0.5,
$P(\mathbf{Temperature} \uparrow)$=0.65,
$P(\mathbf{Cough})$=0.8,
$P(\mathbf{Male} \mid \mathbf{COVID19^+}) = 0.8$,

$P(\textbf{Temperature} \uparrow \mid \textbf{COVID19}^{+}) = 0.7,$
$P(\textbf{Cough} \mid \textbf{COVID19}^{+}) = 0.9.$

$$\begin{aligned} P(\textbf{COVID19}^{+} \mid \textbf{Male}, \textbf{Temperature} \uparrow, \textbf{Cough}) &= P(\textbf{COVID19}^{+} \mid \textbf{Male}) \\ &\times P(\textbf{COVID19}^{+} \mid \textbf{Temperature} \uparrow) \times P(\textbf{COVID19}^{+} \mid \textbf{Cough}) \\ &= 0.96. \end{aligned}$$

Algorithm 9.1 shows the Naive Bayes steps [Code adopted from https://scikit-learn.org/].

Listing 9.1: Naive bayes in Python

```
from sklearn.datasets import load_data
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
X, y = load_data(return_X_y=True)
X_train, X_test, y_train, y_test =
train_test_split(X, y, test_size=0.5, random_state=0)
gnb = GaussianNB()
y_pred = gnb.fit(X_train, y_train).predict(X_test)
print("Number of mislabeled points out of a total %d points:%d"
```

9.2 Decision Trees (DTs)

DTs is a powerful supervised classifier commonly used in operations research and machine learning. A tree, in graph theory, is an undirected, acyclic and connected graph. It is composed of a *root* node, an *internal* nodes, and a *leaf* or *terminal* nodes. All nodes store information and are often labeled with numbers or letters. Figure 9.1 shows an example of tree.

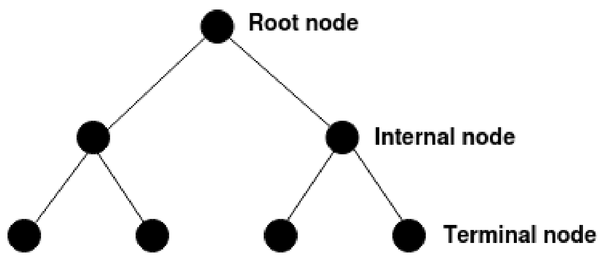


Figure 9.1: Example of a tree

DTs use a decision model in the form of a tree whose leaves contain all potential decisions. Reaching the potential decisions at the ends of the branches is based on the decisions made at each stage. Let's consider the following example related to military service in Algeria (see Figure 9.2). In the figure, the DTs relies on 3 features of the dataset, namely *sex*, *age* and *health*, to predict whether a person is fit to join the algerian military or not. Algorithm 9.2 shows the Decision Trees steps [Code adopted from `https://scikit-learn.org/`].

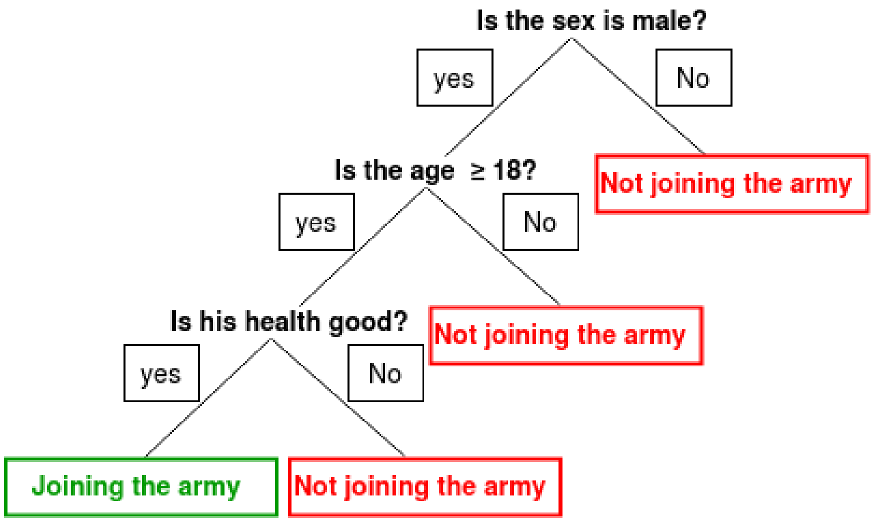


Figure 9.2: Example of Decision Trees

Listing 9.2: Decision Trees in Python

```python
from sklearn.datasets import load_data
from sklearn.tree import DecisionTreeClassifier
 from sklearn.tree import export_data
 X = load_data()
 decision_tree = DecisionTreeClassifier(random_state=0,
 max_depth=2)
 decision_tree = decision_tree.fit(X.data, X.target)
 result = export_data(decision_tree, feature_names
 =X['feature'])
 print(result)
```

9.3 Support Vector Machines (SVM)

SVM is a supervised learning method developped by Vapnik in 1995. It is very powerful for general classification, regression and even outlier detection. It is flexible, theoretically very solid, and simple to use even without a great knowledge of the data. Basically, SVM finds a hyperplane that optimally separates two classes by maximizing the margin between the classes' closest points. Figure 9.3 shows an example of optimal linear separation by SVM in a two dimensional space. Note that, the points on the boundaries are called support vectors.

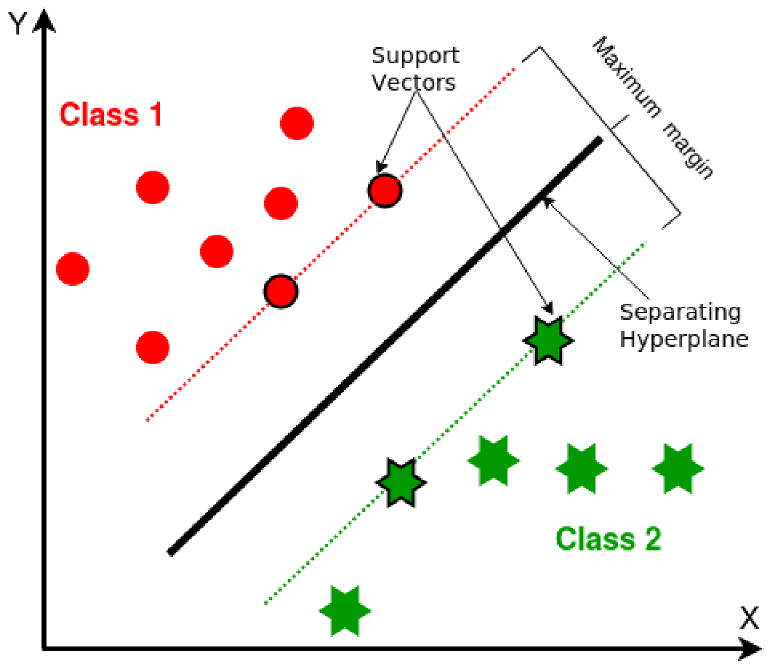


Figure 9.3: Linear separation by SVM

The points in the above example are linearly separable, but in reality the data is rarely linearly separable. For this, SVM often rely on the use of kernels such as polynomial, Gaussian kernel, Radial basis function (RBF), sigmoid etc, when dealing with nonlinear problems. Algorithm 9.3 shows the SVM steps [Code adopted from `https://scikit-learn.org/`].

Listing 9.3: SVM in Python

```python
from sklearn import svm
X = [[0, 0], [1, 1]]
y = [0, 1]
clf = svm.SVC()
clf.fit(X, y)
```

9.4 *k*-Nearest Neighbors (KNN)

KNN is a supervised learning method, mostly used to solve classification and regression problems. It ranks new data points based on the rank of neighbors, assuming similar items are nearby. Basically, KNN is broken down into three steps:

- Step 1: *Calculate the Similarity*: The distance between the new data and all other data already classified is measured. Euclidean Distance is the most commonly used method for calculating distance. It is represented by the following equation:

$$d(p,q) = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2} \tag{9.7}$$

- Step 2: *Find K-Nearest Neighbors*: One selects K nearest data points, i.e the smaller distances. K is a parameter defined at the start of the algorithm.

- Step 3: *Making predictions*: Predict a class value for new data. This by assigning the new data points to the class for which the number of neighbors is maximum.

Figure 9.4 outlines the basic steps mentioned above.

Algorithm 9.4 shows the KNN steps [Code adopted from https://scikit-learn.org/].

Listing 9.4: KNN in Python

```python
from sklearn.neighbors import NearestNeighbors
import numpy as np
X = data
nbrs = NearestNeighbors(n_neighbors=2, algorithm='ball_tree')
.fit(X)
distances, indices = nbrs.kneighbors(X)
```

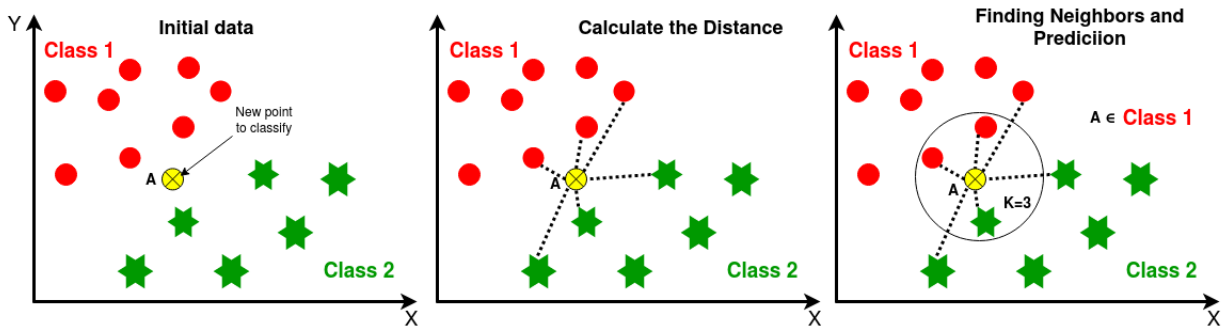


Figure 9.4: KNN steps.

Chapter 10

Regression

Regression is a statistics-based technique used to predict continuous numeric outputs where an order relation is defined. It covers a wide range of fields such as business, survey analysis, neuroimaging, etc. In this chapter, we present a set of regression techniques with a brief description of each.

10.1 Linear Regression

Linear Regression is a supervised machine learning algorithm that seeks to establish a linear relationship between two or more variables. It has many practical uses such as prediction and forecasting. Contrary to the classification, the predicted output is continuous and has a constant slope. Generally, the simple linear regression model is stated in the following form:

$$y = \beta_0 + \beta_1 x + \epsilon \tag{10.1}$$

Where, y is the dependent variable, β_0 is the intercept, β_1 is the slope of the linear regression line, x is the independent variable and ϵ is the random error. The most commonly used method to solve the above equation is that of least squares estimation. This later finds parameters β_0 and β_1 that minimize the following residual sum of squares:

$$(\beta_0, \beta_1) = \arg \min_{(\beta_0, \beta_1)} \sum_{i=1}^{n} [y_i - (\beta_0 + \beta_1 x)]^2 \tag{10.2}$$

After solving the equation 10.2, one obtains:

$$\beta_1 = \frac{\sum_{i=1}^{n} (y_i - \bar{y})(x_i - \bar{x})}{\sum_{i=1}^{n} (x_i - \bar{x})^2} \tag{10.3}$$

$$\beta_0 = \bar{y} - \beta_1 \bar{x} \tag{10.4}$$

Algorithm 10.1 shows the linear regression steps [Code adopted from https://scikit-learn.org/].

Listing 10.1: Linear regression in Python

```python
import numpy as np
from sklearn.linear_model import LinearRegression
X = np.array([[1, 1], [1, 2], [2, 2], [2, 3]])
y = np.dot(X, np.array([1, 2])) + 3
reg = LinearRegression().fit(X, y)
```

10.2 Ridge Regression

Ridge regression is a linear regression technique introduced by Hoerl and Kennard in 1970. It is used when the data suffers from multicollinearity, that is, when the independent variables are highly correlated. Ridge regression can be seen as a regularized version of linear least squares regression by shrinking the coefficients or weights of the regression model towards zero. This is achieved by imposing a penalty term (l_2-norm) which is equal to the square of the coefficient. The Ridge regression is represented by the following equation:

$$\sum_{i=1}^{n}(y_i - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} \beta_j^2 \tag{10.5}$$

where λ is a regularization penalty. If $\lambda = 0$ then the equation is the basic linear regression. As we increase the value of λ, the value of the coefficient tends towards zero leading to both low variance and low bias. Algorithm 10.2 shows the Ridge regression steps [Code adopted from https://scikit-learn.org/].

Listing 10.2: Ridge regression in Python

```python
from sklearn.linear_model import Ridge
import numpy as np
n_samples, n_features = 10, 5
rng = np.random.RandomState(0)
y = rng.randn(n_samples)
X = rng.randn(n_samples, n_features)
clf = Ridge(alpha=1.0)
clf.fit(X, y)
```

10.3 Lasso Regression

Least Absolute Shrinkage and Selection Operator or Lasso is a linear regression technique developed by Robert Tibshirani in 1996. The difference between Lasso and Ridge regression is in the penalty function. Ridge regression uses the squared Euclidean norm and Lasso uses the l_1-norm. The lasso regression loss function is therefore represented by the following equation:

$$\sum_{i=1}^{n}(y_i - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} |\beta_j| \tag{10.6}$$

In addition to the regularization, lasso also performs variable selection to enhance the prediction accuracy and interpretability of the resulting statistical model. Algorithm 10.3 shows the Lasso steps [Code adopted from https://scikit-learn.org/].

Listing 10.3: Lasso in Python

```
from sklearn import linear_model
clf = linear_model.Lasso(alpha=0.1)
clf.fit([[0,0], [1, 1], [2, 2]], [0, 1, 2])
print(clf.coef_)
print(clf.intercept_)
```

10.4 Quantile regression (QR)

QR is an extension of linear regression method, generally used when the data does not satisfy the assumptions for linear regression (eg. financial economics). Contrary to the classical linear regression that estimates the mean response of the dependent variable dependent on the independent variables, the Quantile regression estimates the median, as well as all other quantiles, of a set of data across a distribution based on the variables within that distribution. QR minimizes a weighted sum of the positive and negative error terms as follow:

$$\tau \sum_{y_i > \hat{\beta_\tau}' X_i} |y_i - \hat{\beta_\tau}' X_i| + (1-\tau) \sum_{y_i < \hat{\beta_\tau}' X_i} |y_i - \hat{\beta_\tau}' X_i| \tag{10.7}$$

Where τ is the quantile level. Algorithm 10.4 shows the Lasso steps [Code adopted from https://scikit-learn.org/].

Listing 10.4: QR in Python

```python
from sklearn.linear_model import QuantileRegressor

quantiles = [0.05, 0.5, 0.95]
predictions = {}
out_bounds_predictions = np.zeros_like(y_true_mean,
dtype=np.bool_)
for quantile in quantiles:
    qr = QuantileRegressor(quantile=quantile, alpha=0)
    y_pred = qr.fit(X, y_normal).predict(X)
    predictions[quantile] = y_pred

    if quantile ==  min(quantiles):
        out_bounds_predictions = np.logical_or(
            out_bounds_predictions, y_pred >= y_normal
        )
    elif quantile ==  max(quantiles):
        out_bounds_predictions = np.logical_or(
            out_bounds_predictions, y_pred <= y_normal
        )
```

Chapter 11

Clustering

Clustering is an unsupervised technique based on machine learning used to group data. Unlike classification, in clustering there are no labels for the training data and the groups are not defined in advance but are created based on a measure of similarity which should be clear and have a practical meaning so that data assigned to the same group or cluster should be as similar as possible and data assigned to different groups should be as different as possible. In this chapter, we present a set of clustering techniques with a brief description of each.

11.1 *k*-means

k-means is one of the most popular partition-based unsupervised clustering algorithms. The 'k' refers to the number of clusters which is fixed beforehand, and 'means' refers to the averaging or centroid of the data. Basically, k-means partitions n observations into the k clusters such that the sum of the distances between observations and their respective clusters centroids is minimized. Clusters centroids are updated by an iterative calculation will continue until certain convergence criteria are satisfied. Figure 11.1 and Algorithm 11.1 show the k-means process [Code adopted from `https://scikit-learn.org/`].

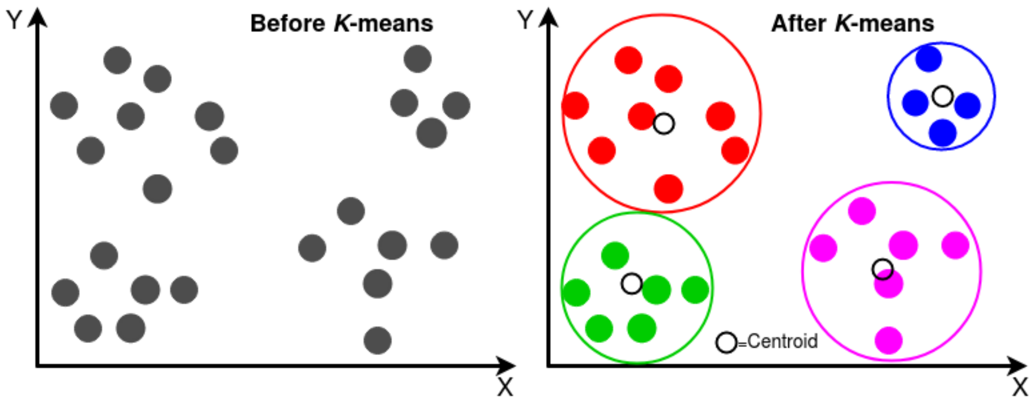


Figure 11.1: k-means clustering.

Listing 11.1: k-means in Python

```python
from sklearn.cluster import KMeans
import numpy as np
X = data
kmeans = KMeans(n_clusters=2, random_state=0).fit(X)
kmeans.labels_
kmeans.predict([[0, 0], [12, 3]])
kmeans.cluster_centers_
```

11.2 BIRCH

BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies) is a Hierarchy-based unsupervised clustering algorithm. It is very useful when the data sets are very large. BIRCH summarizes large datasets into smaller and dense regions called Clustering Feature (CF) entries. It then applies a clustering algorithm on the leaves of the CF tree. Figure 11.2 and Algorithm 11.2 show the BIRCH process [Code adopted from `https://acervolima.com/`].

Listing 11.2: BIRCH in Python

```python
# Import required libraries and modules
import matplotlib.pyplot as plt
from sklearn.datasets.samples_generator import make_blobs
from sklearn.cluster import Birch

```

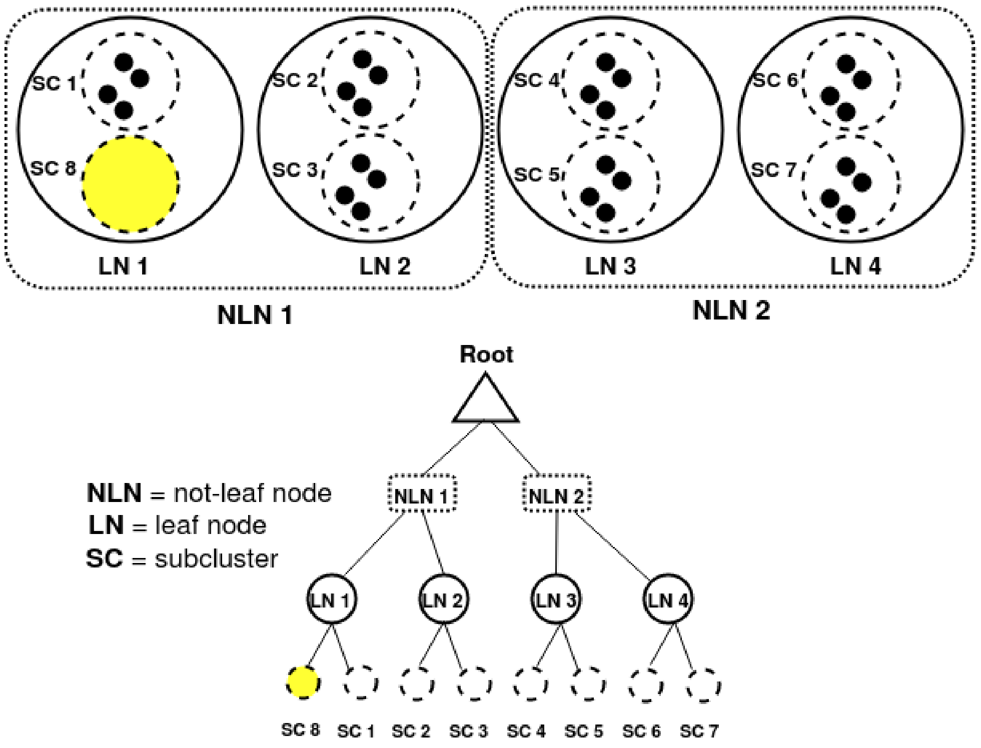


Figure 11.2: BIRCH clustering.

```
# Generating 600 samples using make_blobs
dataset, clusters = make_blobs(n_samples = 600, centers = 8,
cluster_std = 0.75, random_state = 0)

# Creating the BIRCH clustering model
model = Birch(branching_factor = 50, n_clusters = None,
threshold = 1.5)

# Fit the data (Training)
model.fit(dataset)

# Predict the same data
pred = model.predict(dataset)
```

11.3 DBSCAN

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is a density-based unsupervised clustering algorithm proposed by Martin Ester in 1996. It looks for high-density base samples and develops clusters from them. Basically, DBSCAN requires two parameters:

- ϵ: used to defined the neighborhood around points. If it is too small then large part of the data will be considered as outliers. If it is too large the the majority of data points will be in the same clusters.
- $MinPts$: is the minimum number of data points within ϵ radius. It is based on the size of the dataset i.e. if the dataset is large then the value of MinPts should be higher.

Figure 11.3 and Algorithm 11.3 show the DBSCAN process [Code adopted from `https://github.com/`].

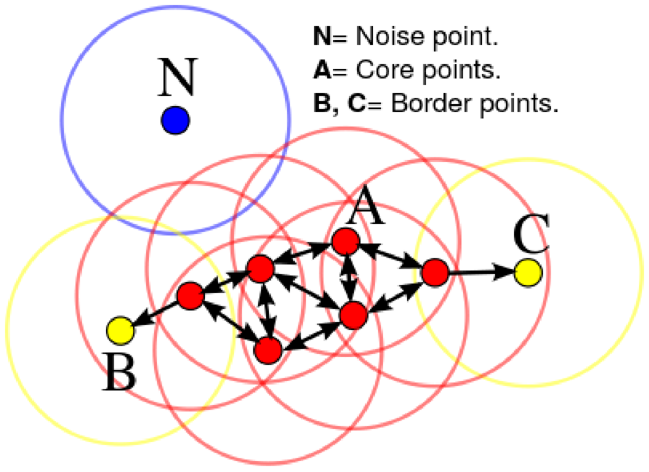


Figure 11.3: DBSCAN clustering [adopted from wikipedia].

Listing 11.3: DBSCAN in Python

```
# Compute DBSCAN
db = DBSCAN(eps=0.3, min_samples=10).fit(X)
core_samples_mask = np.zeros_like(db.labels_, dtype= bool)
core_samples_mask[db.core_sample_indices_] = True
labels = db.labels_

```

```
# Number of clusters in labels, ignoring noise if present.
n_clusters_ =  len( set(labels)) - (1 if -1 in labels else 0)
n_noise_ =  list(labels).count(-1)
```

11.4 FCM

FCM (Fuzzy C-Means Clustering) is a fuzzy theory-based unsupervised clustering algorithm developed by J.C. Dunn in 1973. It is very similar to the k-means, the difference is that in k-means each data point can only belong to one cluster, on the other hand in FCM each data point can belong to several clusters. For instance, the Cyan color is a color halfway between blue and green. Instead to belonging to only blue [blue = 1, green = 0] or only green [blue = 0, green = 1], the Cyan can belong to blue [blue = 0.5] and green [green = 0.5]. Basically, FCM is divided on three steps:

- *Define a number of clusters centers*: Clusters are randomly selected.
- *Calculate the fuzzy membership*: By assigning data points to clusters. Each point belonging to every cluster to a certain degree.
- *Iteration*: Recalculate the cluster centers and the degree of membership in each cluster.

Algorithm 11.4 show the FCM process [Code adopted from https://towardsdatascience.com/].

Listing 11.4: FCM in Python

```
import numpy as np
from fcmeans import FCMmy_model = FCM(n_clusters=2)
# we use two cluster as an example
my_model.fit(X) ## X, numpy array. rows:samples columns:
#features
```

Chapter 12

Dimensionality Reduction

We previously indicated that high-dimensional original data cannot be processed by human, and we have given the example of lion and puma. We confirmed that the reason for this is the limited capabilities of the human brain. It may be different for machines. Doubling the computing capacity of machines, for example processors or memories, with the aim of increasing the number of training samples can lead to improvements such as saving time and increasing the accuracy of the calculations. This solution may be inevitable, but it is very expensive. For this, various solutions have been proposed to mitigate the curse of dimensionality on the one hand, and reduce the high cost on the other. In fact, high-dimensional data tend to be noisy with an enormous amount of redundant features, this may significantly affect the performance of machines and the accuracy of the calculations. Dimensionality reduction by selecting or extracting important features from original data is one of the effective solutions used in this context. Dimensionality reduction is roughly divided into linear methods and nonlinear methods. In the following, we present some methods of each group.

12.1 Linear methods

Linear dimensionality reduction methods are of great importance for the analysis of high-dimensional data with noise. Thanks to their simple geometric interpretations and attractive computational properties (e.g., covariance, dynamic structure, correlation, margin between data classes), it is possible to produce a low-dimensional linear mapping of the original high-dimensional data while by preserving the characteristics of interest in the original data. In this section, we present four linear methods, including: PCA, MDS, LDA and CCA.

12.1.1 Principal Component Analysis (PCA)

One of the most widely used unsupervised methods for feature extraction and data visualization. It was originally formulated by Pearson in 1901, it subsequently experienced many extensions. Principal component analysis can be divided into five steps, which are presented in the following diagram:

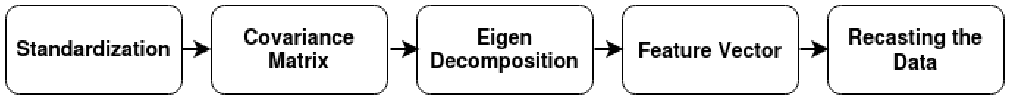


Figure 12.1: Steps in PCA.

- *Standardization:* used to ensure that all the features are along the same scale, and thus avoid biased results. Mathematically, this can be done by the following equation (Eq.12.1):

$$Standardized\ feature\ value = \frac{value - mean}{standard\ deviation} \tag{12.1}$$

- *Covariance matrix:* used to understand how two features vary with each other and see if there is any relationship between them. Mathematically, this can be done by the following equation (Eq.12.2):

$$cov_{x,y} = \frac{\sum_{i=1}^{N}(x_i - \bar{x})(y_i - \bar{y})}{N-1} \tag{12.2}$$

- *Eigen decomposition:* used to get the Eigenvectors which are the principal components of the covariance matrix and the eigenvalues which are their corresponding magnitude. Mathematically, an eigenvector satisfies the following equation (Eq.12.3):

$$A\vec{v} = \lambda\vec{v} \tag{12.3}$$

  where A is a square matrix, $\vec{v}$ is the eigenvector and λ is an eigenvalue.

- *Feature vector:* used as the first step towards dimensionality reduction. In this step, we choose between keeping all the components obtained in the previous step or rejecting those of less importance (low eigenvalues) and forming a feature vector. Mathematically, this can be done by plotting the cumulative sum of the eigenvalues as in the following equation (Eq.12.4):

$$\frac{\lambda_j}{\sum_{j=1}^{d} \lambda_j} \tag{12.4}$$

  If one must choose between three components, ie the eigenvectors *v1*, *v2* and *v3* of the feature vector in the previous step, then if the formula shows that 95% of the variance is captured in the two largest principal components then it is acceptable to choose the first two principal components to constitute the projection matrix and reducing dimensionality by 1.

- *Recasting the data:* used to project the original data onto a new subspace of lower dimensionality. This is done by multiplying the transpose of the original data by the transpose of the feature vector as in the following equation (Eq.12.5):

$$New\ data = Original\ data^T \times Feature\ vector^T \tag{12.5}$$

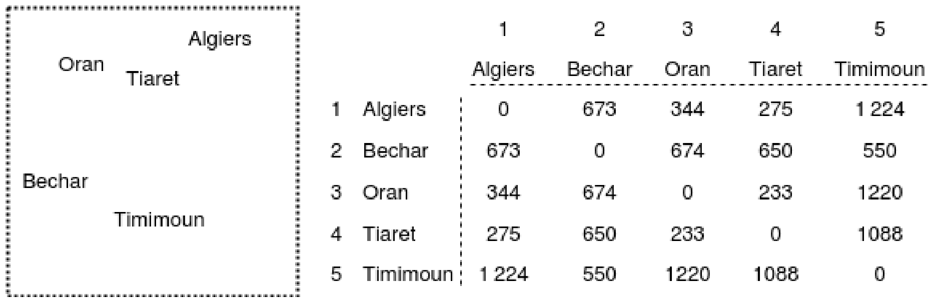
Algorithm 12.1 shows the PCA steps [Code adopted from https://github.com/].

Listing 12.1: PCA in Python

```python
def pca(X, n_components=2):

    # Presprocessing - Standard Scaler
    X_std = StandardScaler().fit_transform(X)

    #Calculate covariance matrix
    cov_mat = np.cov(X_std.T)

    # Get eigenvalues and eigenvectors
    eig_vals, eig_vecs = np.linalg.eigh(cov_mat)

    # flip eigenvectors' sign to enforce deterministic output
    eig_vecs, _=extmath.svd_flip(eig_vecs, np.empty_like
    (eig_vecs).T)

    # Concatenate the eigenvectors corresponding to the highest
    #n_components eigenvalues
    matrix_w = np.column_stack([eig_vecs[:,-i] for i in

     range(1,n_components+1)])

    # Get the PCA reduced data
    Xpca = X_std.dot(matrix_w)

    return Xpca
```

12.1.2 Multidimensional scaling (MDS)

MDS is a visual representation tool that locates objects (eg. colors, faces,...) in an Euclidean space according to their pairwise distances or dissimilarities. Thus, similar objects are close together on the graph, while less similar objects or have longer distances are far apart. Figure 12.2 illustrates the principle of MDS.



		1 Algiers	2 Bechar	3 Oran	4 Tiaret	5 Timimoun
1	Algiers	0	673	344	275	1 224
2	Bechar	673	0	674	650	550
3	Oran	344	674	0	233	1220
4	Tiaret	275	650	233	0	1088
5	Timimoun	1 224	550	1220	1088	0

Figure 12.2: the principle of MDS.

As shown in the figure, *Algiers* is far from *Timimoun* about 1224 km, but it is close to *Tiaret* (275 km). As the map of Algeria is very large, we can delete *Tiaret*. Thus, MDS will serve as a dimensionality reduction. Mathematically, MDS satisfies the following equation (Eq.12.6):

$$\sum_{i \inf j} (d_{ij}|x_i - x_j|)^2 \tag{12.6}$$

Where $x_i \in R^K$ is the coordinates of the ith object in K-dimensional Euclidean space and D is a dissimilarity matrix. $|x_i - x_j| = \sqrt{(x_i - x_j)^T (x_i - x_j)}$.

Algorithm 12.2 shows the MDS steps [Code adopted from https://github.com/].

Listing 12.2: MDS in Python

```python
def mds():
  D = np.loadtxt('DISTANCES.txt', delimiter=',')
  C = np.genfromtxt('CITIES.txt', dtype = 'str', delimiter="\n")
  n =  len(D)
  H = np.eye(n) - np.ones((n,n))/n
  B = -0.1 * H.dot(D**2).dot(H)

  evals, evecs = np.linalg.eigh(B)
  idx = np.argsort(evals)[::-1]
  evals = evals[idx]
  evecs = evecs[:,idx]

  w = [0,1]
  L = np.diag(np.sqrt(evals[w]))
  U = evecs[:,w]
  Y = U.dot(L)
  return Y, C
```

12.1.3 Linear Discriminant Analysis (LDA)

LDA is a supervised dimensionality reduction technique used in several fields such as Bioinformatics, chemistry, etc. It is very similar to PCA, but in addition to finding the component axes that maximize the variance of data, it seeks the projection that maximizes between-class variability while minimizing within-class variability. In other words, LDA transforms the original data into a lower dimensional space while maintaining the class-discriminatory information. It can be divided into three steps:

- *Separability between-class:* In this step, a distance between the means of different classes is calculated.
- *Separability within-class:* In this step a distance between the means and the samples of each class is calculated.
- *Lower Dimensional Space:* This step enables the dimensionality reduction.

Mathematically, LDA satisfies the following equation (Eq.12.7):

$$maximize \ \frac{tr(w^T S_B w)}{tr(w^T S_W w)} \ subject \ to \ w \in \mathbf{O}^{r\times d} \tag{12.7}$$

$$S_W = \sum_{i=1}^{n} (x_i - \mu_{c_i})(x_i - \mu_{c_i})T \tag{12.8}$$

$$S_B = \sum_{i=1}^{n} (\mu_{c_i} - \mu)(\mu_{c_i} - \mu)T \tag{12.9}$$

Where, S_W is the within-class covariance matrix and S_B is the between-class covariance matrix. μ is the global data mean and μ_{c_i} is the class mean associated with x_i. Algorithm 12.3 shows the LDA steps [Code adopted from `https://scikit-learn.org/`].

Listing 12.3: LDA in Python

```
import numpy as np
from sklearn.discriminant_analysis
import LinearDiscriminantAnalysis
X = np.array([[-1, -1], [-2, -1], [-3, -2], [1, 1], [2, 1],
[3, 2]])
y = np.array([1, 1, 1, 2, 2, 2])
clf = LinearDiscriminantAnalysis()
clf.fit(X, y)
LinearDiscriminantAnalysis()
print(clf.predict([[-0.8, -1]]))
```

12.1.4 Canonical correlation analysis (CCA)

Canonical correlation analysis studies the linear relationships between two sets of variables (within and between sets) in order to know if the two sets describe the same phenomenon, and therefore one can do without one of them. Mathematically, Eq.12.10 and Eq.12.11 govern the CCA.

$$\left(\sum_{xy}^{'} \sum_{xx}^{-1} \sum_{xy} - \rho^2 \sum_{yy} \right) Y = 0 \tag{12.10}$$

$$\left(\sum_{xy} \sum_{yy}^{-1} \sum_{xy}^{'} - \rho^2 \sum_{xx} \right) X = 0 \tag{12.11}$$

Where, ρ_{ij} is the correlation coefficient and $\sum_{xy}'$ is the transpose of $\sum_{xy}$ which represents the covariance.

$$\rho_{ij} = \frac{\sum_{xy}}{\sqrt{\sum_{ii}\sum_{jj}}} \tag{12.12}$$

Algorithm 12.4 shows the CCA steps [Code adopted from https://scikit-learn.org/].

Listing 12.4: CCA in Python

```
from sklearn.cross_decomposition import CCA
X = [[0., 0., 1.], [1.,0.,0.], [2.,2.,2.], [3.,5.,4.]]
Y = [[0.1, -0.2], [0.9, 1.1], [6.2, 5.9], [11.9, 12.3]]
cca = CCA(n_components=1)
cca.fit(X, Y)
CCA(n_components=1)
X_c, Y_c = cca.transform(X, Y)
```

12.2 Nonlinear methods

Linear methods are very powerful but suffer from being based on linear models and often miss important nonlinear structures in the data. Nonlinear methods, also called manifold learning, are also very powerful and can be applied when the original high dimensional data contains nonlinear relationships. In this section, we present three nonlinear methods, including: Isomap, KPCA and LLE.

12.2.1 Isometric mapping (Isomap)

Isomap can be viewed as an extension of MDS but it uses geodesic distances instead of euclidean ones in MDS. Figure 12.3 shows the difference between geodesic and euclidean distance.

Isomap can be divided into three steps:

- *Nearest neighbor:* In this step, a neighborhood graph is constructed.
- *Shortest-path graph:* In this step, a shortest path between two nodes A and B is computed, using Dijkstra's algorithm for instance (estimate geodesics).

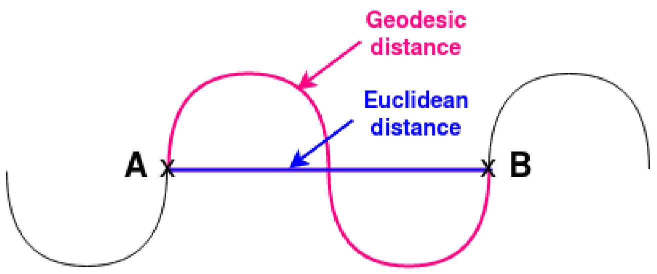


Figure 12.3: Geodesic vs. Euclidean distances between A and B

- *MDS with geodesic distance:* In this step, the MDS method is applied using a geodesic distance as input distance to find the lower-dimensional representation of original data.

Algorithm 12.5 shows the Isomap steps [Code adopted from `https://github.com/`].

Listing 12.5: Isomap in Python

```python
def isomap(data, n_components=2, n_neighbors=6):
    """
    data: input image matrix of shape (n,m) if dist=False,
    square distance matrix of size (n,n) if dist=True
    n_components: number of components for projection
    n_neighbors: number of neighbors for distance matrix
    computation
    """
    # Compute distance matrix
    data, _ = distance_mat(data, n_neighbors)

    # Compute shortest paths from distance matrix
    from sklearn.utils.graph import graph_shortest_path
    graph = graph_shortest_path(data, directed=False)
    graph = -0.5 * (graph ** 2)

    # Return the MDS projection on the shortest paths graph
    return mds(graph, n_components)
```

12.2.2 Kernel PCA (KPCA)

KPCA can be viewed as a nonlinear generalization of principal component analysis (PCA), using techniques of kernel methods (eg., gaussian kernel). KPCA projects a linearly inseparable data set into a higher dimensional space in which the same data set becomes linearly separable. i.e, before performing a PCA, the nonlinear data are mapped into a higher-dimensional feature space. Figure 12.4 illustrates the principle of KPCA. Algorithm 12.6 shows

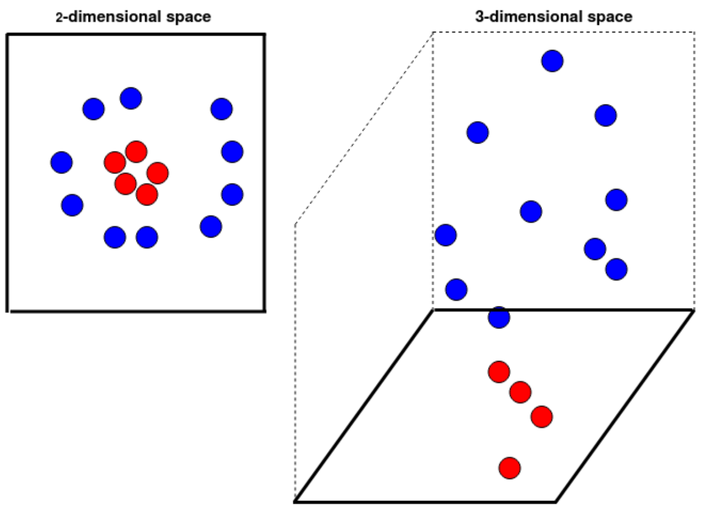


Figure 12.4: the principle of KPCA

the KPCA steps [Code adopted from `https://nirpyresearch.com/`].

Listing 12.6: KPCA in Python

```python
def ker_pca(X, n_components=3, gamma = 0.01):

    #Euclidean distances
    dist = euclidean_distances(X, X, squared=True)

    #Gaussian kernel matrix
    K = np.exp(-gamma * dist)
    Kc = KernelCenterer().fit_transform(K)

    # Eigenvalues and eigenvectors of the kernel matrix
    eig_vals, eig_vecs = np.linalg.eigh(Kc)

    eig_vecs, _=extmath.svd_flip(eig_vecs, np.empty_like
    (eig_vecs).T)

    Xkpca = np.column_stack([eig_vecs[:,-i] for i in

     range(1,n_components+1)])

    return Xkpca
```

12.2.3 Locally Linear Embedding (LLE)

LLE is an unsupervised learning algorithm used to solve globally nonlinear problems using locally linear fitting. It computes low-dimensional embeddings while preserving the neighborhood of the high-dimensional inputs. LLE can be divided into three steps:

- *Nearest neighbor:* In this step, a neighborhood graph is constructed, using the euclidean distance.
- *Reconstruction data:* In this step, weights are computed to optimally reconstruct data from its nearest neighbours.
- *Low-dimensional embedding :* In this step, the coordinates of the original high-dimensional data are computed in the low-dimensional space.

Algorithm 12.7 shows the LLE steps [Code adopted from `https://scikit-learn.org/`].

Listing 12.7: LLE in Python

```
from sklearn.manifold import LocallyLinearEmbedding
X, _ = load_data(return_X_y=True)
X.shape
embedding = LocallyLinearEmbedding(n_components=2)
X_transformed = embedding.fit_transform(data)
X_transformed.shape
```

Chapter 13

Suggested readings

1. Sammut, C., & Webb, G. I. (Eds.). (2011). Encyclopedia of machine learning. Springer Science & Business Media.

2. Mohri, M., Rostamizadeh, A., & Talwalkar, A. (2018). Foundations of machine learning. MIT press.

3. Alpaydin, E. (2020). Introduction to machine learning. MIT press.

4. Pham, T. T. (2018). Applying Machine Learning for Automated Classification of Biomedical Data in Subject-Independent Settings. Springer.

5. Hart, P. E., Stork, D. G., & Duda, R. O. (2000). Pattern classification. Hoboken: Wiley.

6. Anzai, Y. (2012). Pattern recognition and machine learning. Elsevier.

7. Weenink, D. (2003). Canonical correlation analysis. In Proceedings of the Institute of Phonetic Sciences of the University of Amsterdam (Vol. 25, pp. 81-99). Amsterdam: University of Amsterdam.

8. Cunningham, J. P., & Ghahramani, Z. (2015). Linear dimensionality reduction: Survey, insights, and generalizations. The Journal of Machine Learning Research, 16(1), 2859-2900.

9. Abe, S. (2005). Support vector machines for pattern classification (Vol. 2, p. 44). London: Springer.

10. Matloff, N. (2017). Statistical regression and classification: from linear models to machine learning. Chapman and Hall/CRC.

11. Halgamuge, S. K., & Wang, L. (Eds.). (2005). Classification and clustering for knowledge discovery (Vol. 4). Springer Science & Business Media.

12. Freund, R. J., Wilson, W. J., & Sa, P. (2006). Regression analysis. Elsevier.

13. Chatterjee, S., & Hadi, A. S. (2013). Regression analysis by example. John Wiley & Sons.

14. Celebi, M. E. (Ed.). (2014). Partitional clustering algorithms. Springer.

15. Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep learning. MIT press.

16. Ketkar, N., & Santana, E. (2017). Deep learning with Python (Vol. 1). Berkeley: Apress.

17. Sejnowski, T. J. (2018). The deep learning revolution. MIT press.

18. Stevens, E., Antiga, L., & Viehmann, T. (2020). Deep learning with PyTorch. Manning Publications.

Part IV :

Applications Gallery

13.1 Application 1

Dimensionality reduction using PCA

Let's consider the following data matrix to be the score of 6 patients.

$$\mathbf{X} = \begin{array}{c} \\ p1 \\ p2 \\ p3 \\ p4 \\ p5 \\ p6 \end{array} \begin{array}{c} \begin{array}{cc} s1 & s2 \end{array} \\ \begin{pmatrix} 2 & 1 \\ 3 & 5 \\ 4 & 3 \\ 5 & 6 \\ 6 & 7 \\ 7 & 8 \end{pmatrix} \end{array}$$

1- Compute the mean vector:

$\bar{\mathbf{X}} = \frac{1}{n}\sum_{i=1}^{n} x_i = \frac{x_1 + x_2 + \cdots + x_n}{n}$, hence

$\bar{\mathbf{X}} = \begin{bmatrix}(2+3+4+5+6+7)/6 & (1+5+3+6+7+8)/6\end{bmatrix} = \begin{bmatrix}4.5 & 5\end{bmatrix}$

2- Compute the covariance matrix:

$cov_{x,y} = \frac{\sum_{i=1}^{N}(x_i - \bar{x})(y_i - \bar{y})}{n}$

$$\mathbf{D} = x_i - \bar{\mathbf{X}} = \begin{bmatrix} -2.5 & -4 \\ -1.5 & 0 \\ -0.5 & -2 \\ 0.5 & 1 \\ 1.5 & 2 \\ 2.5 & 3 \end{bmatrix}$$

$$\mathbf{D^T} = \begin{bmatrix} -2.5 & -1.5 & -0.5 & 0.5 & 1.5 & 2.5 \\ -4 & 0 & -2 & 1 & 2 & 3 \end{bmatrix}$$

Covariance matrix **C** :

$$\mathbf{C} = \frac{1}{6} \times [\mathbf{D^T} \times \mathbf{D}] = \frac{1}{6} \times \begin{bmatrix} 17.5 & 22 \\ 22 & 34 \end{bmatrix} = \begin{bmatrix} 2.92 & 3.67 \\ 3.67 & 5.67 \end{bmatrix}$$

3- Compute the Eigenvectors & Eigenvalues:

The eigenvalues of **C** are calculated using the following equation:

$$det(\mathbf{C} - \lambda I) = 0.$$

where, λ is an eigenvalue for $\mathbf{C}$ and $\mathbf{I}$ is an identity matrix.

$$det(\mathbf{C} - \lambda I) = 0 \implies \begin{bmatrix} 2.92 & 3.67 \\ 3.67 & 5.67 \end{bmatrix} - \lambda \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 2.92 - \lambda & 3.67 \\ 3.67 & 5.67 - \lambda \end{bmatrix} = 0$$

Hence,

$$(2.92–\lambda)(5.67–\lambda)–(3.67 \times 3.67) = \lambda^2–8.59\lambda + 3.09 = 0$$

Solving this equation, we get, $\lambda_1 = \mathbf{8.22}$ and $\lambda_2 = \mathbf{0.38}$
The eigenvectors are calculated using the following equation:

$$(\mathbf{C} - \lambda_i) \times v_i.$$

One can obtain two eigenvectors: $v_1 = [x_1, x_2]$ and $v_2 = [x'_1, x'_2]$.
If we use λ_1 then:

$$\begin{bmatrix} 2.92 - 8.22 & 3.67 \\ 3.67 & 5.67 - 8.22 \end{bmatrix} \times \underbrace{\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}}_{v_1} = \begin{bmatrix} -5.3 & 3.67 \\ 3.67 & -2.55 \end{bmatrix} \times \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

Hence, we get the following system:

$$\begin{cases} -5.3 \times x_1 + 3.67 \times x_2 = 0 \\ 3.67 \times x_1 - 2.55 \times x_2 = 0 \end{cases}.$$

If we use λ_2 then:

$$\begin{bmatrix} 2.92 - 0.38 & 3.67 \\ 3.67 & 5.67 - 0.38 \end{bmatrix} \times \underbrace{\begin{bmatrix} x'_1 \\ x'_2 \end{bmatrix}}_{v_2} = \begin{bmatrix} 2.54 & 3.67 \\ 3.67 & 5.29 \end{bmatrix} \times \begin{bmatrix} x'_1 \\ x'_2 \end{bmatrix}$$

Hence, we get the following system:

$$\begin{cases} 2.54 \times x'_1 + 3.67 \times x'_2 = 0 \\ 3.67 \times x'_1 - 5.29 \times x'_2 = 0 \end{cases}.$$

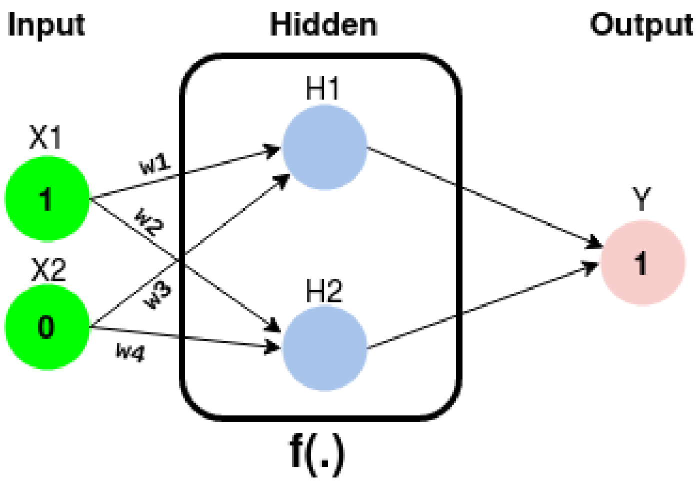
By simplifying the above systems, we get the values of the two eigenvectors which are the principal components.

13.2 Application 2

Classification using Neural Network

Let's consider the following data matrix:

$$\textbf{Input} = \begin{array}{c} \begin{array}{cc} X_1 & X_2 \end{array} \\ \begin{pmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{pmatrix} \end{array} \quad \textbf{Output} = \begin{array}{c} Y = X_1 \oplus X_2 \\ \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} \end{array}$$



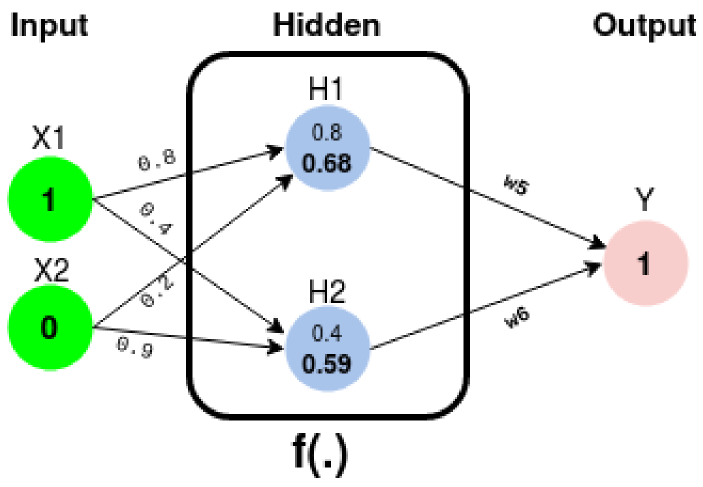
Let's assume the sigmoid function $f(x)$ is used in the activation function of the neural network.

$$f(x) = \frac{1}{1+e^{-x}}$$

Let's assume $b = 0$, [w1, w2]= [0.8, 0.4], [w3, w4]= [0.2, 0.9].

$\mathbf{H}_1 = f((w1 \times \mathbf{X}_1) + (w3 \times \mathbf{X}_2) + b) \implies \mathbf{H}_1 = f((0.8 \times 1) + (0.2 \times 0) + 0)$.
$\mathbf{H}_1 = f(0.8) = 0.68$.

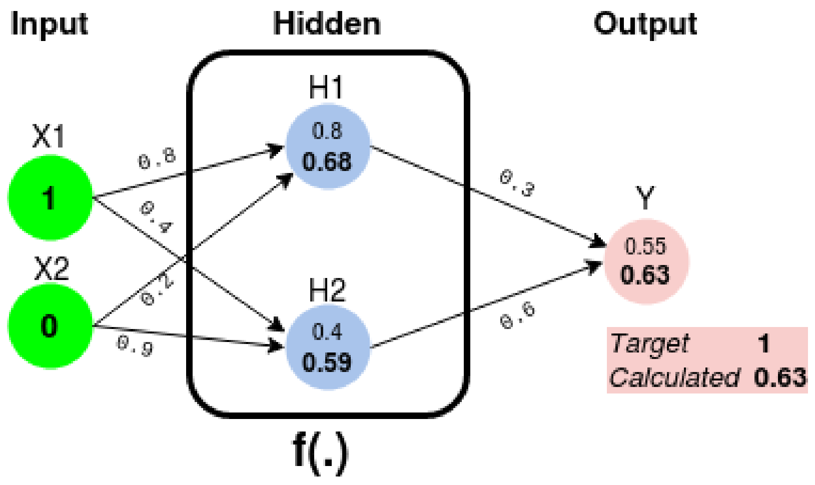
$\mathbf{H}_2 = f((w2 \times \mathbf{X}_1) + (w4 \times \mathbf{X}_2) + b) \implies \mathbf{H}_2 = f((0.4 \times 1) + (0.9 \times 0) + 0)$.



$\mathbf{H}_2 = f(0.4) = 0.59.$

Now, suppose [w5, w6]= [0.3, 0.6]

$\mathbf{Y} = f((w5\times\mathbf{H}_1)+(w6\times\mathbf{H}_2)+b) \implies \mathbf{H}_1 = f((0.3\times0.68)+(0.6\times0.59)+0).$
$\mathbf{Y} = f(0.55) = 0.63.$



The value of the output neuron (target) should have 1, the output value is 0.63. It is wrong because of the initial weights used which are chosen at random. To fix this we use backpropagation by adjusting the weights to improve the network.

$$\begin{bmatrix} w_5^* \\ w_6^* \end{bmatrix} = \begin{bmatrix} w_5 \\ w_6 \end{bmatrix} - a \times \Delta \times \begin{bmatrix} H_1 \\ H_2 \end{bmatrix}$$

$$\begin{bmatrix} w_1^* & w_3^* \\ w_2^* & w_4^* \end{bmatrix} = \begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} - a \times \Delta \times \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} \times \begin{bmatrix} w_5 \\ w_6 \end{bmatrix}$$

Where, $\Delta = (claculated - target)$ and a is used in a smarting manner. We repeat the same process of backward and forward pass until error is close or equal to zero.