



Delft University of Technology

## Pattern Recognition Introduction and Terminology

Duin, Robert P.W.; Pekalska, Elzbieta

**Publication date**  
2016

**Document Version**  
Final published version

**Citation (APA)**  
Duin, R. P. W., & Pekalska, E. (2016). *Pattern Recognition: Introduction and Terminology*. Delft University of Technology.

**Important note**  
To cite this publication, please use the final published version (if applicable).  
Please check the document version above.

**Copyright**  
Other than for strictly personal use, it is not permitted to download, forward or distribute the text or part of it, without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license such as Creative Commons.

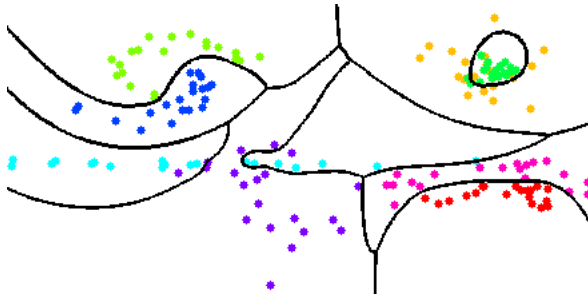
**Takedown policy**  
Please contact us and provide details if you believe this document breaches copyrights.  
We will remove access to the work immediately and investigate your claim.

*This work is downloaded from Delft University of Technology.  
For technical reasons the number of authors shown on this cover page is limited to a maximum of 10.*

# Pattern Recognition: Introduction and Terminology

Robert P.W. Duin and Elżbieta Pękalska  
*37 Steps*

August 23, 2016



This is a free ebook.  
Donations are appreciated: [37steps.com/donations](https://37steps.com/donations).

## About this ebook

This ebook gives the starting student an introduction into the field of pattern recognition. It may serve as reference to others by giving intuitive descriptions of the terminology. The book is the first in a series of ebooks on topics and examples in the field.

Our goal is an informal explanation of the concepts. For thorough mathematical descriptions we refer to the textbooks and lectures. In ten chapters the topics of pattern recognition are summarized and its terminology is introduced. In the [glossary](#) about 200 terms are described. All glossary terms are linked, forward and backward by hypertext. In the [glossary chapter](#) external links are provided to internet pages, papers, tutorials, Wikipedia entries, examples, etcetera. Internal links are in [dark blue](#) in order to preserve the readability. External links are in [blue](#).

This ebook is offered by the authors of a website on pattern recognition tools, <http://37steps.com/>. Here more information, software, data and examples can be found. The book itself does not assume the use of specific software. The code for generating the examples, however, is written in Matlab using [PRTTools](#). It can be inspected by clicking on the figures or example links.

## How to read

A very simple and useful pdf reader for this document is [Sumatra PDF](#). It uses by default the backspace as the back-button. Other pdf readers should be adjusted such that returning to the previous page is as a handy shortcut available. It is often needed for browsing through this ebook. See [here](#) for some tips. In this case a web browser should be open next to the book. Alternatively, the document may be read inside a web browser. Use single page view if possible.

The medium size version of the book has been formatted such that it is suitable for a computer screen. The small size version is suitable for an e-reader like the Kindle. Here external links may not work. Reading and browsing through the ebook may still be of interest to get acquainted to the pattern recognition terminology.

## About 37Steps

The website <http://37steps.com> has been created by the authors after they left university. They have been engaged with pattern recognition for decades and put together their insights, experiences and software. All material is for free. However, donations are appreciated: <http://37steps.com/donations>.

# Contents

|           |                                              |           |
|-----------|----------------------------------------------|-----------|
| <b>1</b>  | <b>Introduction</b>                          | <b>4</b>  |
| 1.1       | Recognition and consciousness . . . . .      | 4         |
| 1.2       | Creating artificial PR systems . . . . .     | 5         |
| <b>2</b>  | <b>A small example</b>                       | <b>8</b>  |
| <b>3</b>  | <b>Review of PR problems</b>                 | <b>14</b> |
| 3.1       | Pattern recognition applications . . . . .   | 14        |
| 3.2       | Data types . . . . .                         | 15        |
| <b>4</b>  | <b>The PR system in operation</b>            | <b>17</b> |
| <b>5</b>  | <b>PR system design</b>                      | <b>19</b> |
| <b>6</b>  | <b>Representation</b>                        | <b>22</b> |
| 6.1       | Object representation . . . . .              | 22        |
| 6.2       | Vector representations . . . . .             | 24        |
| 6.3       | Dimension reduction . . . . .                | 27        |
| <b>7</b>  | <b>Generalization</b>                        | <b>30</b> |
| 7.1       | Class models or decision functions . . . . . | 30        |
| 7.2       | Classifiers . . . . .                        | 32        |
| <b>8</b>  | <b>Evaluation</b>                            | <b>35</b> |
| 8.1       | Error estimation . . . . .                   | 35        |
| 8.2       | Learning curves . . . . .                    | 37        |
| 8.3       | Feature curves . . . . .                     | 39        |
| 8.4       | Accuracy guidelines . . . . .                | 41        |
| <b>9</b>  | <b>Exploratory data analysis</b>             | <b>43</b> |
| 9.1       | Cluster analysis . . . . .                   | 43        |
| 9.2       | Visualization . . . . .                      | 46        |
| <b>10</b> | <b>Glossary</b>                              | <b>49</b> |

# Chapter 1

## Introduction

### 1.1 Recognition and consciousness

For a very small child a walk into the world is a walk into the wild: all is new and exciting. Every **object**, every sound, every shape provides a new experience. Consequently, it does not know how to interpret what is happening, nor is it able to walk or to grasp objects properly. It does not know how to hold an item or how to walk. That is beneficial, as it also does not know about danger and, as a result, can freely explore the world around. Such an exploration cannot continue forever. If the child has to become independent of its protecting environment, it will have to know and to understand, it will have to act and to deal with the world in a sensible way.

The miracle of memory helps to overcome the state of ignorance. It converts the chaos of the wild into a world where we can organize the information and find our way. Thanks to the memory we find our way back to a safe place or to a place where the food has been stored. Thanks to the memory we learn to avoid the danger. We recognize the past in the present. Thanks to successes and disappointments we learn how to deal with them. Memory yields **consciousness**. And consciousness is the basis for **recognition** and understanding.

How is this possible? The human abilities of **consciousness** and **recognition** are miracles as large as the physical basis of gravity and light. They are there. We can build some models, but we are still far from understanding how the light that enters the eye generates the word or the idea 'tree' in the mind. At the bottom there is the first principle: we may say that we understand, but what do we mean by that? Do we understand understanding? The entire field of **pattern recognition** is an effort to come somewhat closer to this understanding. This can be done in a scientific way, or by attempts to create artificial devices that mimic the human ability. Here we will give an introduction how results of the first are used for the second.

The present is never equal to the past. There are always differences. Every street, every tree or every person that we meet is different from all streets, trees and persons we have seen before. How do we know that the new place where we find ourselves is a street anyway? How do we know that the figure on the quilt is a tree? Or, how do we know that we know the person who is standing at the door? Even if she is our partner and we have lived with her for 37 years the question how we recognize her is intriguing.

There may be just minor differences between what we see in the very moment and what we have left behind this morning and that is still in our memory. But, why are some differences minor while others are major? How do we judge that? How do we know which differences are essential? In one way or the other we are able to judge that different **observations** refer to the same **object** we have seen before. Even, in the case we have never seen a particular **object**, we are able to recognize to which **class** of similar **objects** it belongs.

The ability to judge the **similarity** between **objects** or events is called **generalization**. Given a few examples, sometimes even a single one, we are able to tell whether a new, unseen **object** belongs to the same group. Its similarity to previously observed objects is sufficiently large. Human beings are **pattern recognizers**, not just because of this **recognition** ability, but especially because we are aware of it. We can handle it, we can teach the **patterns** to others and discuss with them our **observations**.

The **generalization** skill develops our **consciousness** further. It constitutes the basis of any science, in particular, the natural sciences. The question how we do this, how we come from **observations** to memory and to **generalization** is thereby the basic scientific question of **pattern recognition**.

## 1.2 Creating artificial PR systems

From the early development of computers, scientists and engineers tried to imitate the human **recognition** ability by mechanical means, either

partially or in its entirety. Two types of main results have been obtained from these efforts so far.

First, scientific results have been gained: a better understanding of the human perception, reasoning and the ability to gain new knowledge which may be applied in a changing environment. This resulted in more insight in the human senses and the neural system. To some extent, this understanding can be expressed in mental, psychological and philosophical terms. Herewith **observations**, facts and existing knowledge are combined by reasoning yielding some new conclusions.

Attempts to design sensors, computers and programs that imitate such processes bring an additional prospect to the investigation of possible biological models. An ever returning difficulty, however, is the relation of low level phenomena occurring in the senses and the nerves with high level understanding and conceptual thinking. How do externally measurable, physical and biological processes generate the internal **observation** of **recognition** and understanding? This problem is related to the so-called **semantic gap** and is one of the main unsolved scientific questions.

Second, many contributions to the engineering practise have been created. Various **pattern recognition** systems have been developed that are of practical use, as for the assistance in medical diagnosis, industrial inspection, personal identification and man-machine interaction. Very often, they are not based on a detailed simulation of the human processes, but on specific approaches to the problem at hand.

It is striking and interesting to observe that artificial **recognition** devices, especially the ones that **learn from examples**, are almost not, or just superficially based on a modeling of the human perception and learning abilities. This can be compared with other biological studies. We know a lot of how birds fly, but airplanes are constructed differently. One of the reasons is that the artificial systems may serve different purposes: they need to be more stable and should sometimes be faster and larger, at the cost of a reduced flexibility.

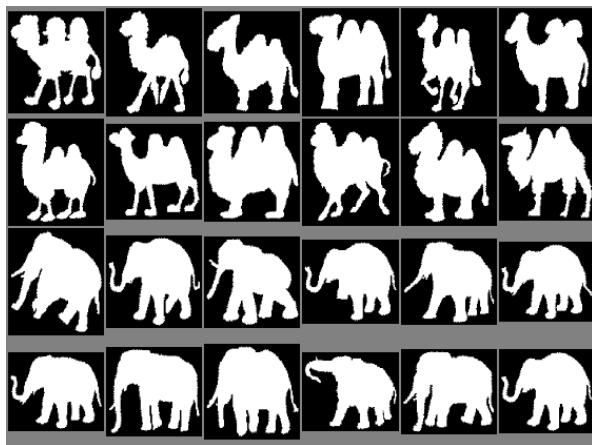
In this book, we will focus on the [pattern recognition](#) research aiming at the development of automatic systems. We will especially deal with the possibilities of these systems to learn from sets of examples. A general, global description of such systems will be presented, resulting in a intuitive characterization of the various steps and procedures that can be distinguished in their design and operation. This will be illustrated by simple examples.

The main goal of the book is to give the student a first introduction into the terminology used in the field of [pattern recognition](#). In the final chapter a glossary is presented in which short characterizations of the main terms are given with backward references to the places where they have been introduced or used for the first time. A few non-standard terms are added for completeness. For most other ones holds that they used in the literature in various different ways. Therefore, strict definitions are not presented here. It is expected that readers have some implicit pattern recognition ability and are able to learn from the examples as they have been made available.

## Chapter 2

### A small example

After all these introductory words it is more than time to discuss a simple example. Let us take a small dataset, the so-called Kimia images. This is a set of silhouettes of 2D figures. [Figure 1](#) shows two of the classes, elephants and camels, each given by 12 examples. Suppose we take out one elephant and show it to somebody who has no knowledge about these animals, e.g. a 3-year old boy. Would he be able to name it if we would show him the other 23 figures and name for each of them the class? It is probable that he wouldn't have any problem with that.



*Figure 1: Two [classes](#) in the Kimia dataset*

Let us now try to do the same with a computer. We need to compare figures in a numeric way: compute some type of distances or similarities. To that end a comparable [representation](#) has to be established. As the images have all different sizes, counting pixel differences is not straightforward. We will try something very elementary and compute the sizes (blob areas in pixels) and perimeter lengths (in pixels). They are graphically represented in the [scatter plot](#) of [Figure 2](#). This is a picture of what is called a [feature space](#). This is a vector space in which [objects](#),

here the animal blobs, are represented by characteristic properties, here area and perimeter length.

If the **features** really correspond to characteristic properties, then similar **objects** are close in **feature space**. Similar **objects** are expected to belong to the same class. In the plot the **objects** are encircled that have nearest neighbors of a different class. The fraction of encircled **objects** gives an indication of the usefulness of this **feature space** for recognizing new, similar **objects** of the same **classes**.

A problem with this **feature space** is that the area is dominating over the perimeter as the spread of its values is much larger. If **features** are considered to be equally important, **feature spaces** should be created in which they are equally **scaled**. This is done in the next **scatter plot**, **Figure 3**, by dividing all **feature** values by their corresponding standard deviation.

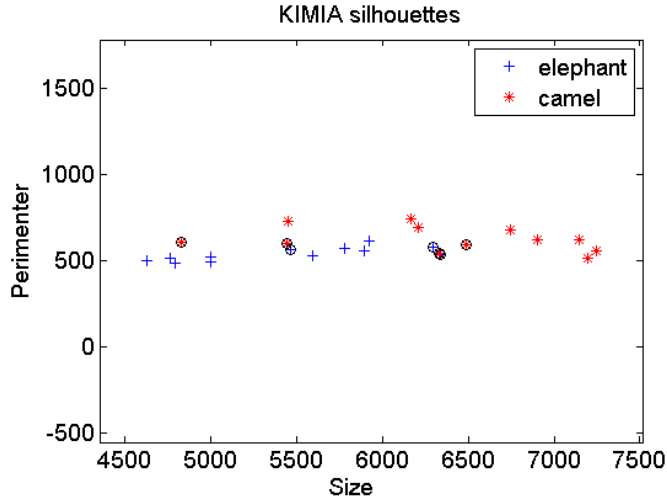


Figure 2: *Scatter plots of two classes in the Kimia dataset based on perimeter and area. Objects are encircled for which the nearest neighbor is of a different class. Two erroneously classified objects are almost on top of each other. Note that the perimeter is hardly of significance as the values of the area feature are much larger.*

In this case the result of the [scaling](#) is that less [objects](#) are assigned to the wrong class, indicating that the scaled [feature space](#) is better for [classification](#). The improvement (5 instead of 7 errors), however, is in this case not significant. It is certainly not true that rescaling is always better. In some cases it is known that the [features](#) as they are measured are informative. This should not be spoiled. Anyway, whether [scaling](#) is appropriate or not should always be considered.

Once a [feature space](#) is found that is appropriate for classifying new [objects](#) with unknown [class memberships](#), it can be used for computing a definite [classifier](#). Several procedures are possible. In [Figure 4](#) the result of the same [classifier](#) as used above is shown using the entire available [training set](#). For new [objects](#) the same [features](#), area and perimeter should be measured and the same scaling factor as found for the [training set](#) should be applied.

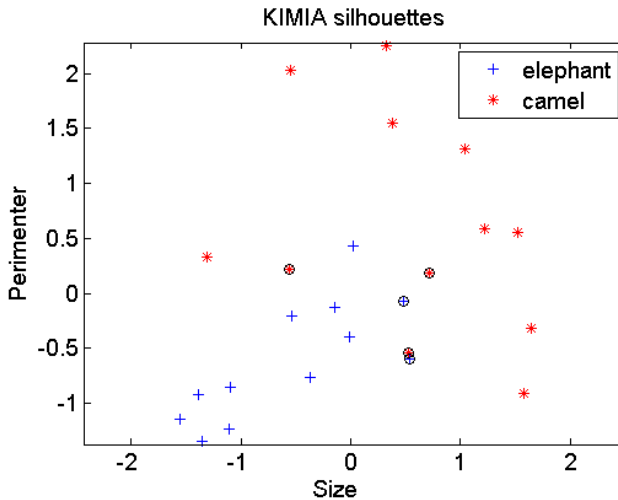


Figure 3: *Scatter plots* of two classes in the Kimia dataset based on perimeter and area. The values of perimeter and area are [rescaled](#) using their standard deviation. [Objects](#) are encircled for which the nearest neighbor is of a different class.

In order to have a reliable estimate of the [classification accuracy](#) of this [classifier](#), an additional [test set](#) is needed, representative for the application and independent of the [training set](#). If the entire [design set](#) (the union of [training set](#) and [test set](#)) is used for [training](#) the [classifier](#), it is expected to be better, i.e. to have a lower [classification error](#). Using this [design set](#) or one of its subsets ([training set](#) or [test set](#)), however, it will be optimistically biased as they are used for [training](#) this [classifier](#). Strategies for dealing with this dilemma, computing the best [classifier](#) or estimating a reliable [classification error](#), are discussed in [section 8.1](#).

To end this chapter we summarize the steps that have been taken in the design of the [pattern classifier](#) and the main concepts used.

**Representation** If we want to build automatic [recognition](#) system for real world [objects](#) they should be represented in a way they can be compared by a computer. Here we used a camera to obtain black-and-white [images](#) in which the [objects](#) are blobs. The areas and perimeters of the blobs are used to construct 2D vectors in a [feature space](#).

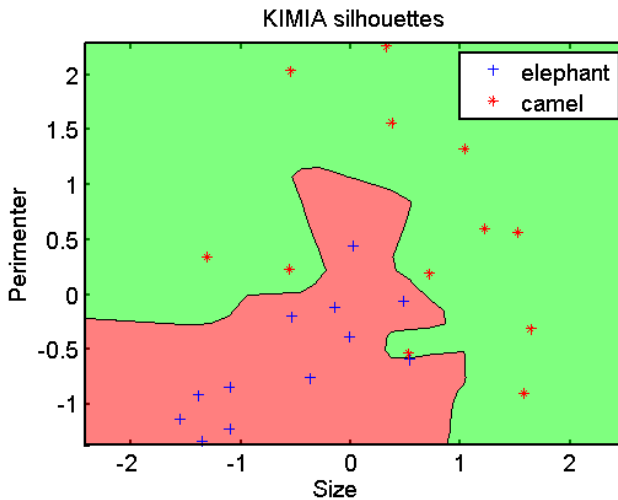


Figure 4: The [classifier](#) computed for the entire available dataset.

**Adaptation** By studying the [representation](#) it was found that the area dominated as it showed much larger values and thereby much larger differences between the [objects](#). The variations were [normalized](#) by dividing by the standard deviation to give [features](#) an equal spread. Consequently, distances between [objects](#) are now equally influenced by the two [features](#).

**Evaluation** We need to find how good a particular [representation](#) is. This is useful for finding better [representations](#) and for optimizing an [adaptation](#). See also [chapter 6](#) and [chapter 8](#).

**Generalization** The final system should be such that it is possible to apply it to new, unseen [objects](#), that have not been used in the design. In the above example we used the computation of a [classifier](#) as a particular kind of [generalization](#). See also [chapter 7](#)

In addition the following concepts have been used:

**Object** Used for real world physical [objects](#) and events as well as for their [representations](#), e.g. in a [feature space](#).

**Feature** A characteristic, measurable property of [objects](#) useful for [representation](#) and [generalization](#). Characteristic is meant here as contributing to the distinction of [objects](#) of different [classes](#). [Features](#) are sometimes called [attributes](#).

**Class** A set of [objects](#) that, as such, has to be distinguished from other [objects](#) in the problem.

**Label** Often used as the name of the [class](#) to which a particular [object](#) belongs. Sometimes it is just a number, being an index in the list of names of classes to be distinguished.

**Training set** The set of [objects](#) with known [class labels](#) used for computing a [classifier](#).

**Test set** The set of [objects](#) with known [class labels](#) used for estimating the [classification error](#) of a [trained classifier](#).

**Design set** The total set of **objects** that is used in designing a **recognition** system. Subsets are used **training**, **evaluation** or special steps in the **adaptation**.

**Classifier** The final function that distinguishes the classes of interest.

The entire technology of the creation of a **recognition** system based on given **objects** is sometimes referred to as **learning from examples**.

# Chapter 3

## Review of PR problems

Here some examples will be given of [Pattern Recognition](#) (PR) applications and the types of data that the analyst in this field may encounter.

### 3.1 Pattern recognition applications

By its nature PR can be applied in any field in which [observations](#) are studied that can be represented in a numeric way. It is only worthwhile to use a PR approach if the problems can be formulated as [classification](#) problems: it should be of interest to assign new [objects](#) to some class. Moreover, this problem should not be easily solvable by other means, e.g. a threshold on a sensor, or a perfectly fitting physical model. Applications areas of PR are often fields that recently became of interest, or in which new sensors have been introduced, or that are very complicated. In such situations a sufficiently useful physical may not be available or is infeasible. PR may be of help to start an analysis.

In almost all the natural sciences PR techniques are used in one way or the other. In some of them, e.g. astronomy, chemometrics and taxonomy, very similar techniques are used under different names. [Classification](#) procedures have here been studied independently and sometimes even before PR was established as an area of research. In some way the field of PR as a separate technology became of interest after it was found that in various disciplines similar problems arose. Various fields with many researchers continued independently. This might be regretted, but it has definite its positive sides as thereby different paths are followed and different solutions are found.

Important application areas to which many PR researchers contribute are biology, health, medical imaging, psychology, human behavior, ecology, seismics, space engineering, aeronautics, oceanology, navigation, transport, computer vision and speech. In all these areas new sensors or new measurement technologies are introduced. Often the possibilities that

arise grow much faster than the possibilities to process, analyze and interpret the generated data.

In addition to these areas of science and engineering, there are various applications of societal interest, e.g. security, forensics, mail delivery, personal identification by biometrics, signature verification, fraud detection and computer crimes. Sometimes similar sensors as in scientific research are used like photo sensors and microphones, but also entirely different types of measurements may be used. Examples are text and computer events.

The common ground between many of these application areas is the much smaller set of different data types. As far as they are similar, general procedures can be developed. A review of the main data types is presented in the next section.

## 3.2 Data types

Here some common data types are discussed as they are encountered by an analyst in the PR field studying **observations** on real world **objects**.

**Independently observed features** Features measured by separate sensors, e.g. the weight and size of cars in an application of car type **recognition**. It should be emphasized that such **features** are independently measured, but the obtained **feature values** are not statistically independent as they involve the same **object**.

**Time signals** Sensor outputs observed as a function of time, e.g. speech. Such data may run continuously with out a clear start or end. If the signals are properly segmented, e.g. into words, **objects** can be aligned: at the same time sample the same phoneme is expected. **Objects** can thereby be compared as a function of time.

**Spectra** Unsegmented time signals can be transformed into spectra by a Fourier analysis. Time signals of different length may be compared by spectra over the same frequency interval. Examples are

the [recognition](#) of types of earthquakes in seismics and speaker [recognition](#) from audio signals.

**Images** This is a very common data type. There are various subtypes. The pixel values can be binary, gray, RGB and even multi-band (more than three colors) and hyperspectral. In the latter case an entire spectrum per pixel is measured, like in remote sensing. As for the time signals, [objects](#) in [images](#) may be positioned on arbitrary places, or might be aligned after [segmentation](#). In addition [objects](#) may have different sizes and may be really 2D or just 2D views of 3D [objects](#).

**Text** For instance newspaper articles to be classified on topic, or letters to be [classified](#) on author.

**Symbolic sequences** Sequences of symbols, e.g. representing DNA structures or a series of events in an internet stream.

**Graphs** Complicated [objects](#) like an image of a building can be represented by a [graph](#): connections between elements and their properties are used as [attributes](#) of nodes and edges.

In advanced applications combinations of such datatypes may arise, e.g. videos (images as a function of time, combined with sound). Sometimes one datatype is transformed into another, e.g. a time signal may be represented as a series of spectra over short time periods, a so-called spectrogram. This looks like an image, but [segmentation](#) might be needed to locate a specific event. Such an event may be represented by a some [features](#) like time length and the power of some specific [features](#).

# Chapter 4

## The PR system in operation

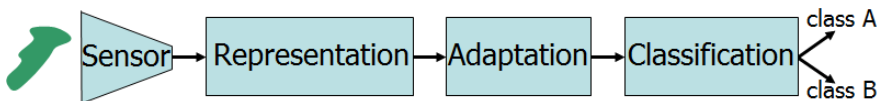
A completed **pattern recognition** system, as we want to design it, may look like shown in **Figure 5**. We will discuss the separate elements.

**Object to be recognized** This **object**, possibly an event, is a part of the world outside the **recognition** system. It has an unknown **class** and it is the task of the system to derive this class from its **observations**. An example is the heart beat of patient that should be classified as normal/abnormal.

**Sensor** This can be a single sensor or a set of sensors. In our example it may consist of a few electrodes measuring the ECG (electrocardiogram) adjusted to the breast of the patient.

**Representation** The time signals retrieved from the electrodes cannot be directly used. By **segmentation** separate heart beats have to be isolated. Next they should be sampled in a standard way, e.g. every heart beat should be represented by 256 samples. In this way an **object representation** is realized suitable for the **classification** of heart beats.

**Adaptation** The 256 time samples should be adapted to the **classifier** found in designing the system. As such they might not yet characterize the differences between normal and abnormal heart beats sufficiently. Moreover, for some **classifiers** the number of 256 is too large. So in this step the initial **representation** is adapted to a final one that suits the final step. This may result into a few **features**.



*Figure 5: Pattern recognition system.*

**Classification** Finally, the **classifier** found during design, takes the **features** as an input and an optimized function maps the **features** found for a single heart beat to normal/abnormal. Sometimes a score between 0 and 1 is returned. They may also be combined over a series of heart beats.

The two steps called here **representation** and **adaptation** are often combined into a single **representation** step. Often this step is called **feature extraction**. We split it here into two, to emphasize that there is a part that is entirely defined by the designer of the system, independent of the **design set**, and a part that is optimized by **training** the **classifier**. We will return on that later in **chapter 6**

# Chapter 5

## PR system design

The design of a system as described in the previous chapter is based on two sources of knowledge. The expertise of the analyst as well as the application expert (often not the same person) is needed for various choices like the sensors and the initial signal processing. Also a gross list of possible useful and measurable **features** has to be created by these two persons.

The second source of knowledge is the **design set**: a set of **objects**, preferably drawn at random from the same source as the future **objects** to be classified. All or many of these **objects** should have a known **class label**: their true **class memberships** should be known. They are used for optimizing the **representation** (the **adaptation** of an initial **representation** to the demands of the **classifier**) and for **classifier training**. Training a **classifier** is also an optimization step, but often with a different criterion than the **adaptation**.

Usually the **design set** is stored for an initial **representation**, chosen on the basis of the background knowledge of the application. It may be randomly split into two or three different sets:

**Training set** This is the set used for adapting the **representation** (if needed) and **training** the **classifier**. See Figure 6.

**Validation set** This set is only used for designing advanced systems or when one wants to validate the choice of an algorithm or **classifier**. See Figure 7.

**Test set** After optimizing the **recognition** system, its performance is estimated by a **test set**. See Figure 8.

Sometimes the **training set** is used for **validation** as well. These sets serve a similar task: optimizing the **recognition** system. It should be

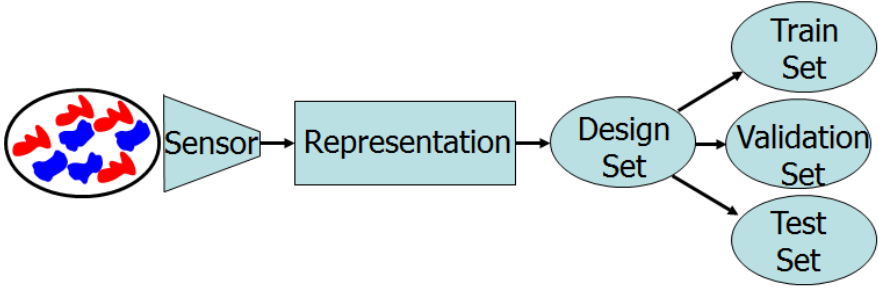


Figure 6: The *design set* is split at random into different subsets.

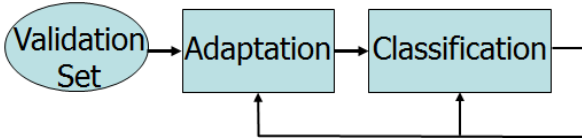


Figure 7: The *validation set* is used for optimizing choices for algorithms and classifiers.

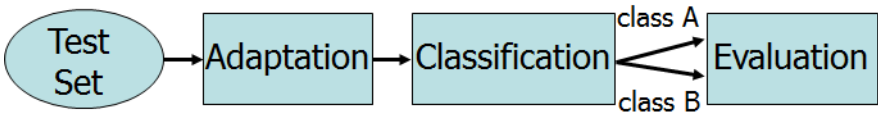


Figure 8: The *test set* is used for an *evaluation* of the final system.

taken into account, however, that after **training** a **classifier** with a particular dataset, the performance measured on the basis of this set might be treacherously good. Using the **training set** for **evaluation** is in general a bad idea.

More often the **test set** is used for **validation**. Consequently the **test set** is used multiple times, e.g. for choosing an algorithm for adapting the **representation** to the **classifier**, for choosing a good **classifier** on the basis of their performances after **training**, and finally for reporting the performance of the entire system to the costumer or supervisor. This

usage is very common, but formally wrong. Whenever a dataset has been used for performance estimation, followed by a change of the system in an attempt to improve it, this dataset has in fact been used for [training](#). The resulting system is (somewhat) adapted to that dataset and consequently this dataset cannot be used anymore for an independent [evaluation](#), see [chapter 8](#).

The ultimate consequence of this reasoning is that a [test set](#) can only be used once. In practise this is usually prohibitive as the available dataset will be limited in size. System designers usually prefer to use all available data for optimizing the system and take it for granted that their final performance estimate is biased.

# Chapter 6

## Representation

### 6.1 Object representation

The task of the **representation** step is to make **objects** numerically comparable. If  $x$  and  $y$  are the **representations** of two **objects** then it should be possible to compute  $d_{xy} = D(x, y)$  in which  $D$  is some distance measure and  $d_{xy}$  is a scalar reflecting the difference between  $x$  and  $y$ . For simple sensors or for an advanced definition of  $D$  no additional **representation** is needed. Otherwise, the sensor outputs have to be transformed into a **representation** for which distance functions are available. Some examples:

**Graphs**, a **structural representation** build by nodes and edges. An example is the **graph representation** of **images** by segments (nodes) with connections (the graph edges) to neighboring segments. The computation of **graph** distances is not trivial, especially not if the nodes or edges have **attributes**.

**Symbolic sequence**, a **structural representation** by a sequence of symbols, like text or a DNA sequence.

**Sampled domain**, like **images** and time signals. In order to compute distances between **images** or between time signals it is advantageous if they all have the same size.

**Features**, a set of relevant **object** properties. The same **features** have to be measured for all **objects** in order to make the comparable.

The **structural representation** will not be further discussed here. On the basis of computed distances between structures a so-called **dissimilarity representation** can be built, see below.

**Objects** given as **images** or time signals may need further processing before they can constitute a proper initial **representation**. Here are some possibilities:

**Segmentation** This is of interest for **images** and time signals. All pixels or time samples outside the domain of interest should be removed. After that the **object representations** have different sizes and cannot easily be compared directly. So **segmentation** is often followed by the computation of **features** of the segments.

**Scaling** For **objects** observed under possible different circumstances (e.g. light conditions) it is necessary to adjust the amplitudes properly. The best way to do this is application dependent. Possibilities are to multiply the entire signal such that their means or maxima are equal (e.g. divide by such a number by which it becomes one). Time signals like speech may be better **normalized** on their standard deviation (or mean amplitudes). For some sensors it is needed that minima (zero level) become equal.

**Centering** In case **objects** have some samples around them that do not belong to the **objects** themselves, they should be positioned such that they can be compared pixel wise (sample by sample). The best way to do this is by centering as the **object** borders are not always well defined.

**Resampling** Depending on the next steps it may be needed that **objects** are represented by the same numbers of samples.

**Rotation** For many **objects** in **images** their rotation is not important for **recognition**. In order to make an efficient comparison a standard rotated view may be used. Sometimes it is needed to put restrictions to such a rotation, e.g. to avoid that the digits '6' and '9' become equal.

**Normalization** All the above and possible others are sometimes referred to as **object** normalization.

Object properties that are not relevant for **class** differences are named **invariants**. Examples may be position in an **image** or size. In the design of a **pattern recognition** system they should be removed if possible. During **training** of a **recognition** system the insignificance of such properties may be discovered automatically, but this requires a **training set** that is sufficiently large. It is thereby more efficient to remove them. The above **normalization** steps may take care of some of them.

## 6.2 Vector representations

A very common, almost 'standard' **representation** is the vector space. All **objects** are represented by vectors having the same length and using the same set of properties as their elements. The **scatter plot** in **Figure 3** is an example of a 2-dimensional **vector representation**. Usually vector spaces have higher dimensionality, from tens to possibly over 1000.

The popularity of the **vector representation** is caused by the availability of many tools for analyzing sets of vectors. **Classes** may be studied by computing their **densities**, estimated from the **training set**. **Classifiers** in a  $k$ -dimensional vector space are  $(k-1)$ -dimensional linear or nonlinear surfaces separating **class** regions in the space. The data (sets of **objects**) may be found to lie (with sufficient accuracy) in a linear or nonlinear subspace of possibly much lower dimensionality, from which it can be concluded that the relevant **features** describing the **object variability** are a (non)linear function of the used **object** properties.

Multi-variate statistics, linear vector algebra and matrix manipulation are important tools to study **vector representations**. We will summarize three ways to arrive at such a **representation**.

**Feature representation** The space built by the possibly relevant **object** properties is called the **feature space**. They may be derived from **images** or time signals according to some preprocessing, but they may also be measured directly by a dedicated sensor, e.g. a temperature. As **features** can be based on entirely different physical phenomena, their **scaling** may be different as well. **Normalizing**

their variabilities as discussed in [section 6.1](#) may be needed to avoid not-intended build-in preferences for some [features](#).

It should be emphasized that the [feature representation](#) is reducing: some [object](#) differences are not taken into account. Consequently different [objects](#) can have the same [representations](#) in the [feature space](#). If this happens for [objects](#) of different classes, the classes overlap. [Classifiers](#) may then be built on the basis of [class](#) density estimators.

**Pixel representation** In case it is not clear what [features](#) should be defined, one may try to sample the [images](#) or time signals by which the [objects](#) are given. The [normalization](#) steps discussed in [section 6.1](#) are now relevant. In particularity it is necessary that all [object representations](#) have the same size in pixels or samples. Moreover, alignment by shifting and rotation may be needed as well.

If the [images](#) of all [objects](#) have the same number of pixels, an [image](#), and thereby the [objects](#), can be represented in a vector space. It has as many dimensions as the [image](#) has pixels. An example is the MNIST dataset, a heavily used benchmark consisting of 70000 [images](#) of 28x28 pixels. These are gray value [images](#). The [pixel representation](#) is thereby 784. If color (RGB) [images](#) are used like this, the dimensionality would be three times larger.

Instead of [images](#), samplings of time signals or spectra may be used in the same fashion. In these cases time samples or frequency amplitudes are used instead of pixel intensities to build the vector space. Spectra are usually already automatically aligned (same frequencies have the same meaning). They thereby constitute a more simple example of a [pixel representation](#) than [images](#). (Although time samples and frequencies are not pixels, still '[pixel representation](#)' is sometimes used for lack of something better).

In case [objects](#) are entirely defined by their [image](#) (e.g. 2D [objects](#)) or time signal (e.g. speech), an automatic [recognition](#) system gets the same information as a human observer. It may thereby reach the same or even a better performance. If human [recognition](#) is error free, or [class labels](#) are assigned in a consistent way by the human expert, classes will not overlap in a [pixel space](#).

**Dissimilarity representation** The good aspect of the **pixel representation** is that any, by the sensors observed **object** change, is reflected in the **representation**. A severe drawback is that neighboring pixels are expected to be highly correlated. In the **representation**, however, their relation is similar to the relation of two remote pixels. The entire **object** is teared into pieces, which are used independently for **representation**. Neighborhood relations are lost.

The **dissimilarity representation** tries to do both, be sensitive for all **object** differences, but treat neighboring pixels different than remote pixels. The application expert is asked to design a **dissimilarity** measure  $d_{xy} = D(x, y)$  between **objects** such that this is realized. Examples are the Hausdorff distance between shapes and the Earthmover distance between spectra. Next a set of prototype **objects**  $P = p_i, i = 1, \dots, k$  has to be defined, called the **representation set**. Any **object**  $x$  is now represented by a vector of distances **d** to the representation set:  $\mathbf{d} = (D(x, p_1), \dots, D(x, p_k))$ . This constitutes the **dissimilarity space**.

Any distance measure between **objects** can be used for this **representation**. Preferably they should have the above discussed properties. Other choices appear sometimes to perform surprisingly well, including the **Euclidean distance** in **feature spaces** and **pixel spaces**. An explanation for this is that such a **dissimilarity space** is a non-linear transformation of the original space. Consequently, simple linear **classifiers** in the **dissimilarity space** correspond to non-linear **classifiers** in the original **representation space**. The non-linearity used for this is not arbitrary, but is in some way natural for the given original **representation**.

There is another way than the **dissimilarity space** to arrive at a vector space from dissimilarities. That is **embedding**: finding a vector space in which the distances are equal to the given dissimilarities. In general such a space is non-Euclidean as it cannot always be based on an **Euclidean geometry**. In such cases it cannot easily be used for **generalization**.

The **feature space** is traditionally the main **representation**. The vector spaces resulting from the **pixel representation** and the **dissimilarity representation** are formally not **feature spaces** as their bases are just indirectly constituted by **object** properties. It is very common, however, to use "feature space" in referring to a **pixel space** or a **dissimilarity space**.

## 6.3 Dimension reduction

**Generalization** procedures like **classifiers** sometimes heavily focus on their performance on the **training set**. If this set is small in relation to the dimensionality of the **feature space** they may **overtrain**, i.e. they adapt more to the local noise than to the global **class** differences as a result of the **curse of dimensionality**. To avoid this, separate **dimension reduction** procedures are used. There are two different approaches:

**Feature extraction** finds combinations (often linear) of the given **features** such that the **class separability** is maintained as good as possible. Criteria are usually not **classification performance** (as this will introduce **overtraining** again), but general measures of the preserved information content.

**Feature selection** is more restrictive as it selects **features** instead of searching for good combinations. Selection may be preferred over extraction as the final **feature representation** is here in terms of a reduced set of original **features**. This may give more insight in what is important for **recognition**. Moreover, sensors related to not-selected **features** may be removed.

Note that by both approaches a subspace of the original **feature space** is found. **Class separability** will reduce or, at best, remain equal. It is impossible that it will increase. However, **classifiers** may have a more easy job by which the final performance increases.

The most important algorithms for reduction and selection are:

**Principal Component Analysis (PCA)** is an **unsupervised approach** (not using **class labels** of the **training set**) for linear **feature extraction**. It maximizes the explained variance.

**Linear Discriminant Analysis (LDA)**, also named Fisher Mapping, is a **supervised approach** (using **class labels** of the **training set**) for linear **feature extraction**. It maximizes the **between scatter** (differences in **class means**) over the **within scatter** (object differences within a class, averaged over the classes).

**Individual selection** takes the best **features** according to their individual performance.

**Forward selection** starts with the best single **feature** and extends the selection iteratively with the **feature** that maximizes the performance of the selected set.

**Backward selection** starts with all **features** and reduces the selection iteratively with the **feature** that least reduces the performance of the selected set.

There are more advanced selection algorithms like **branch and bound** and **floating search**. They only make sense for large **training sets** as extended searches bear the danger of **overtraining**. Moreover, they are very time consuming. Also the backward selection, mentioned above, has to be treated with care as it starts with measuring the performance of all **features**. This is already tricky when **feature selection** is done because of a too large dimensionality of the space.

If one has a much too large **feature** set w.r.t. the **training set** size, e.g. 1000 to be reduced to 10, a feasible approach might be:

1. Select by individual selection 50 **features**.
2. Select by forward selection 20 out of these 50.
3. Select by backward selection 10 out of 20.

Several criteria are possible in [feature selection](#). They can be divided in [filter](#) and [wrapper](#) approaches. In [filter](#) approaches a global separability criterion is used like the Mahalanobis distance. This is comparable to [LDA](#) in [feature extraction](#).

In the [wrapper](#) approach a [classifier](#) is used and its performance used as a criterion. As [training](#) has to be repeated for all steps this can be very time consuming. Moreover, it is needed to have a separate [test set](#) for measuring the performance. Even more advanced and time consuming is to do [cross-validation](#), see [section 8.1](#). Although [wrapper](#) approaches are popular in the literature, there is again the danger of [overtraining](#) caused by the repeated use of a [test set](#). It may be used thousands of times and starts thereby to become a [training set](#).

For the selection and optimization of [representations](#) various criteria are used, but there is no such thing as a 'ground truth'. However, at the end a [representation](#) is as good as the [generalization](#) that can be based on it. We discuss here no specific procedures for the [evaluation](#) of [representations](#) and just refer to the [evaluation chapter 8](#).

# Chapter 7

## Generalization

### 7.1 Class models or decision functions

Once we have a proper **representation**, it can be used for **learning from examples** procedures. Such procedure estimates an unknown property of an **object** given a set of example **objects** for which the property is known. This is also called **generalization**. It is based on the assumption that some properties are heavily dependent on others. Thereby it is not needed to observe them directly. They can be indirectly determined from other **observations**.

In PR **generalization** is mainly used to find the **class** of an **object**. This is particularly of interest when there is no measurement possible that reveals the **class** and only a human expert is able to name it. For example, there is no *single* measurable property that determines that a chair is a chair. It can only be established in an automatic way from a *set* of measurable properties. A set of examples of known chairs, collected by a human expert, in this context often called the **teacher**, has to be used for **training** a **generalization** algorithm, e.g. a **classifier**. This results into a **trained classifier** which is a function of measurable properties. Depending on the outcome of this function a new **object** may be **classified** as a chair.

Here some ways will be discussed to find **classifiers** for **objects** represented in a vector space. Such a **classifier** determines from the set of **labeled object examples**, the **training set**, for every other point in the vector space what its **label** might be. (A **labeled object** is an **object** with known **class membership**). The output of a **classifier** might be a unique **crisp label**, a set of possible labels, or for all or the main classes a **confidence**.

A classification **confidence** is a number between 0 and 1. The larger the more likely the related class. As an extreme **objects** are sometimes **rejected**. This implies that for the supplied **representation** vector of the

object to be classified not sufficient information is available to make any sensible decision. Most classifiers don't have an explicit reject option, but from thresholds on confidences rejects may be decided.

There are two opposite approaches from which classifiers can be built.

**Class models** For each class a model is build using objects of that class only, usually a probability density function. The procedures for the estimation of density functions can be distinguished in parametric and non-parametric approaches. By parametric estimation the parameters of some standard function are estimated, e.g. the mean and covariance matrix a normal distribution(also called Gaussian distribution). Non-parametric estimators, like the Parzen density estimator, replace every sample by elementary function, also called kernel, and average them.

As a result of the density estimation for every point in the vector space a distance or a confidence for that class will be found, independent of possible other classes. So this is a class-by-class approach. The advantage of this approach is that if classes have known prior probabilities (class frequencies of objects to be classified), these can be used to weight the models.

**Decision functions** In this case objects of different classes are used simultaneously for a direct estimate of the decision function between the classes. Decision functions often act between two classes or between groups of classes (so called binary classification problems). In case of more than two classes (multi-class classification problems) they can be used in two fashions: one-against-one and one-against-rest. In the first case classifiers are found for all pairs of classes. In the second each classifier tries to separate a single class form all other. In both cases a second level of decision making has to be defined for a final classification.

Some decision functions, e.g. the ones based on neural networks, directly implement a multi-class classifier.

The advantage of the decision function approach is that just is estimated what is needed. Directly training a decision function is

more efficient than the detour over estimating full **class** models. A disadvantage is that the use of **class prior probabilities** is for this approach difficult, tricky or impossible.

## 7.2 Classifiers

Here the most important **classifiers** are shortly summarized.

**Template matching** A few representative **objects** (**prototypes**) are selected by the **teacher**, at least one per **class**. Classification is done by assigning the **class** of the nearest to the **object** to be **classified**.

**Nearest mean** This is like template matching. There is now a single **prototype** per **class** which is defined as the mean of all **training objects** available for that **class**.

**1-Nearest neighbor** This is also like **template matching**, but now all available **training objects** are used as **prototypes**.

**Fisher classifier** This is a two-class **classifier** (**discriminant**). It is defined as the plane perpendicular to the direction in **feature space** that maximizes the **between scatter** over the **within scatter**, see **LDA**. It is also known as **Fisher's Linear Discriminant (FLD)**. In case of equal **class priors** it is almost identical to the **Bayes normal classifier** assuming equal **covariances**.

**Bayes classifier** This **classifier** selects the **class** with the highest **posterior probability**, determined by the **class** densities and the **class priors**. This is the minimum error **classifier** (the error it realizes is sometimes called **Bayes error**). Usually densities are unknown and should be estimated. See **Bayes normal** and **Parzen** below.

**Bayes normal** The **Bayes classifier** assuming **normal distributions** for all classes. If the **class covariance matrices** are assumed to be equal the **classifier** is linear (almost identical to **LDA**), otherwise quadratic (**QDA**).

**Parzen classifier** The **Parzen classifier** is a **Bayes classifier** using a **kernel** density estimator for the **class** distributions. Depending on the implementation the same or different **kernels** are used for the classes. The choice of the **kernel** function is usually a spherical **normal distribution**. Its width should be optimized or chosen by the teacher.

**k-Nearest neighbor ( $k$ NN)** The  $k$ -Nearest neighbor classifier selects the majority **class** of the  $k$ -nearest neighbors in the **training set**.

**Naive Bayes** This is another **Bayes classifier** based on estimated densities, now assuming independent **features**. Thereby **class** densities are estimated by multiplying **feature** density estimates. The latter can be based on histograms, **Parzen** densities, **normal distributions**, etcetera.

**Decision tree** Usually a binary **decision tree** is used in which the nodes are single **feature** thresholds. During **training** for every node the next best **feature** is selected for the part of the **training set** that arrives in the given node. There are various criteria and stopping strategies.

**Neural network** Originally neural networks are studied as a simulation of the human nervous system. Later they are used as well for building artificial **recognition** systems. This resulted in a large set of architectures with very different properties. Consequently, the performance of a neural network **classifier** heavily depends on the choices made by the analyst, and thereby on his skills. The main properties of neural network **classifiers** are that they can be **trained** by very large datasets, **training** is computationally demanding, the **classification** function is usually nonlinear and results can be reasonably good.

**Support vector machine (SVM)** The **SVM** is in the research communities of machine learning and **pattern recognition** a very frequently applied and studied **classifier**. It is based on a solid theoretical foundation called Empirical Risk Minimization (ERM) which

aims to maximize the distances of the **training objects** to the **classifier**. Originally it is designed for linear **decision functions**, but by the use of **kernels** and the so-called **kernel trick** also non-linear functions can be computed.

**Adaboost** This **classifier** is based on combining a large set of simple, usually linear, **weak classifiers**, resulting in a complex, non-linear architecture. Iteratively additional **base classifiers** are added emphasizing difficult (often erroneously classified) **objects** in the **training set**. (A **base classifier** is one of the constituting **classifiers** in a **combined classifier**)

**Random decision forest** Also this **classifier combines** many simple **base classifiers**, in this case **decision trees** with just a few nodes based on randomly selected **features**.

The bottom five **classifiers** in the above list have many adjustable parameters to be set by the user. Finding the best values is challenging. A well known approach is a so-called **grid search**: systematically a large set of parameter combinations is evaluated, often by **cross-validation**. There is a high risk of **overtraining**.

# Chapter 8

## Evaluation

**Evaluation** is an essential aspect of the design of **pattern recognition** systems. Every choice, every **training** step, as well as the overall set-up are guided by expectations resulting from past experience or estimates of the realized **accuracy**. Without **evaluation** there is no learning. It is thereby crucial to have proper procedures, but also to be aware of the general behavior of the performance as a function of **training set** size and dimensionality.

In the next sections short reviews are given of **error estimation** procedures, **section 8.1**, the error as a function of the size of the **training set** (**learning curves**), **section 8.2**, and the dimensionality (**feature curves**), **section 8.3**. Finally some general considerations and guidelines for the **accuracy** are presented, **section 8.4**.

### 8.1 Error estimation

The **classification error** can be estimated by counting the number of erroneous **classifications** made in a set of **objects** with known true **class labels** (a **test set**). This set should be representative for the future **objects** to be **classified**. Usually this is realized by random sampling. The **objects** are thereby i.i.d. (independent and identically distributed) random variables. The fraction of errors, the **test error**, is in that case an unbiased estimate of the expected **classification error**.

This **error estimation** procedure (**testing**) can be performed over the set of **classes** or **class-by-class** and weighted with the probabilities of encountering objects of a particular **class**, the **class priors**. This should be done when the **class frequencies** in the available set of **objects** are different from the known, true **class prior probabilities**.

The **objects** in the **training set** are not independent from the **classifier** to be tested. Formally they should not be used. It is expected they will result in an optimistically biased error estimate. However, the **training**

**set error**, also called **apparent error** or **resubstitution error**, can be informative in comparison with the **true error** (the error for an infinite independent **test set**), estimated by the available **test set**. The difference is a measure for the bias, and thereby for **overtraining**. The larger, the more the **classifier** has been adapted to the noise in the **training set**, instead of the **class** differences.

An independent **test set** is needed for an unbiased estimate of the **true classification error**. To get its variance as small as possible the size of the **test set** should be as large as possible. Larger **test sets**, however, will result in smaller **training sets**, as they are subsets of the same **design set**. This results in worse **classifiers**. Thereby, there is a trade-off between very accurate estimates of the performance of a bad **classifier** (large **test sets**, small **training sets**), and inaccurate estimates of the performance of a good **classifier** (small **test sets**, large **training sets**).

A common compromise for the above dilemma is a 50-50 split of the **design set** in equal parts for **training sets** and for **testing**. The final **classifier**, delivered to the customer, will be **trained** on the entire **design set**. The performance of a **classifier trained** by a dataset half of this size will not be very much worse. This is the **hold-out error**, which is slightly pessimistic.

The roles of **test set** and **training set** in a 50-50 split may be reversed and the results averaged. This is called 2-fold **cross-validation**. The entire **design set** participates in testing by which the variance in the resulting estimate in comparison with the hold-out procedure is reduced. It is, however, equally pessimistic as it estimates an error of a **classifier** that is **trained** by just 50% of the data used for the final **classifier**.

Instead of 2-fold **cross-validation** also **n-fold cross-validation** can be used by a rotation procedure. In this approach  $n$  **classifiers** are **trained** by a fraction of  $(n - 1)/n$  of the **design set** and tested by the remaining **objects**, a fraction of  $1/n$  of the **design set**. Again, all **objects** are used for testing, but they test **classifiers** that are very close to the final **classifier**. They are more similar to each other as well as to the final **classifier** for larger values of  $n$ . The variance is thereby smaller (**classifiers** are

more equal) and the estimate is even less pessimistic. The cost of this procedure is that **training** times are increased by a factor of  $n$ . Common choices for  $n$  are  $n = 5$  and  $n = 10$ .

The  $n$  **classifiers**, based on random subsets of a  $(n - 1)/n$  fraction of the **design set**, are all somewhat different. A repetition of the **cross-validation** will generate another set of  $n$  **classifiers**. Averaging the performances of a number (e.g. 10) of **repeated  $n$ -fold cross-validations** will yield a somewhat more accurate estimate the **classification error**. The variance in the performance results can be used to test whether differences between **classifiers** are significant.

A special case is the **leave-one-out cross-validation (LOO)**. It uses as many folds as the size of the **design set**. Every **classifier** is **trained** by all-but-one **objects** in the **design set** and is tested by just the single **object** that was left out. In this case makes no sense as it would generate the same result. The tested **classifiers** are all almost equal to the final one. Consequently the estimated error, usually called the **LOO-error** has a minimum pessimistic bias.

## 8.2 Learning curves

Here we discuss why plotting and analyzing **learning curves** may reveal some interesting characteristics of **classifiers**. A **learning curve** shows the performance or **classification error** of a **classifier** as a function of the size of the **training set**. As the error has to be estimated from an independent **test set** it can not be studied up to the size of the **design set**.

In **Figure 9** an example is given based on the Iris dataset, a 3-class dataset (different types of the Iris flower) with 50 **objects** per **class**, given by 4 **features**. Random subsets of 2-40 **objects** per **class** are used for **training**, the remaining ones for testing. This is repeated 100 times to get smooth results.

The plot shows the averaged estimates of the error on the **test set** as well as on the **training set**, the so called **apparent error**. The **test error**

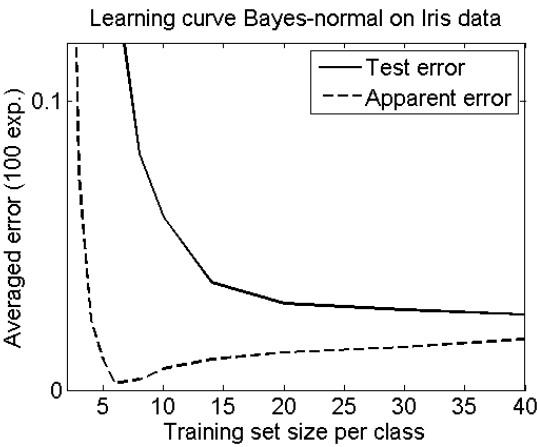


Figure 9: *Learning curve of Bayes Normal for the Iris dataset. The error curve for the test set is shown as well (apparent error).*

approaches some asymptotic value. It can be observed that this value has not been reached yet as the curve still goes down. Enlarging the dataset is thereby expected to yield better results.

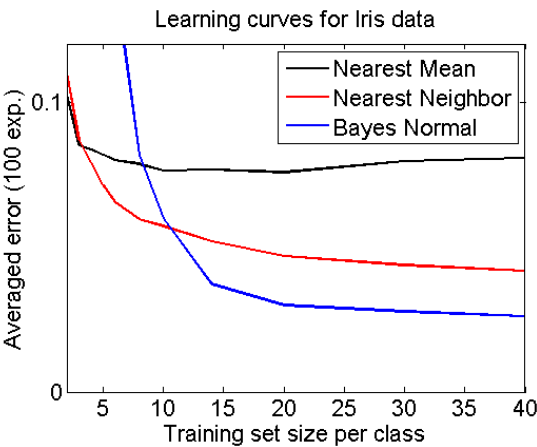


Figure 10: *Learning curves for the Iris dataset.*

The **apparent error** reaches a minimum for 6 **training objects** per **class**. For less **objects** no proper **normal distributions** can be estimated. For increasing sample sizes after 6 **objects** per **class** the **classifier** better and better generalizes as the estimated distributions approach the real distributions. It is expected that the two curves approach the same asymptotic value. The **test error** from above, the **apparent error** from below.. The difference between the two errors can be understood as the amount of **overtraining**. It is caused by the adaptation of the **classifier** to the noise in the data (caused by small datasets) instead of to the true **class** distributions.

In **Figure 10** the **learning curves** for three **classifiers** are shown. They differ in complexity. A general tendency of such curves is that low-complexity **classifiers** do relatively well for small **training sets**, while that high-complexity **classifiers** need large **training sets**. It can be seen that the most simple **classifier**, the nearest mean, shows a reasonable performance for just a few **training objects**, while the most complex **classifier**, **Bayes Normal**, at moment suffers from the noise in the data. Because of its complexity it is sensitive for noise, but for large **training sets** its sensitivity can well be used to shape a good **decision function**.

This scissor phenomenon is very characteristic for **classifiers** with a different complexity. It shows that there is no unique best **classifier** for some problems, but that it depends on the size of the **training set**.

## 8.3 Feature curves

The behavior of the **classifier** performance as a function of the dimensionality is significant for selecting **features** and their number. **Feature curves** show an estimate of the true **classification error** as a function of the number of **features**. In **Figure 11** and **Figure 12** some examples are given based on the **Bayes normal classifier**.

For **classifiers** trained by a small **training set**, **feature curves** may show a minimum for a low number of **features**. This shows **overtraining** for larger dimensionalities, see the sections on **dimension reduction** 6.3 and

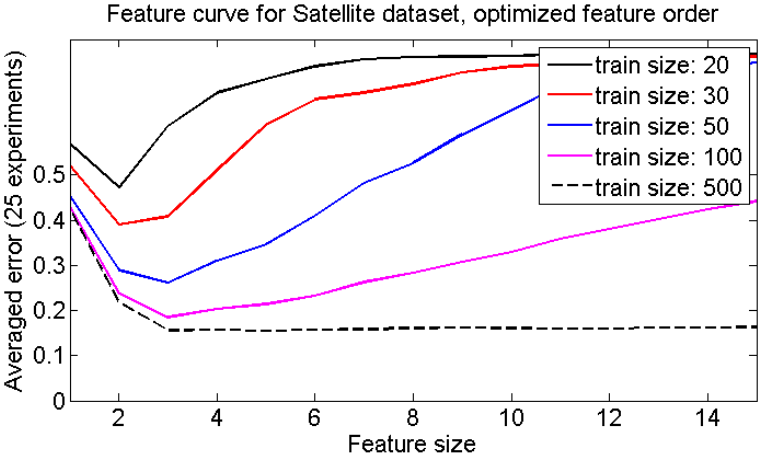


Figure 11: Satellite dataset *feature curves*, optimized order.

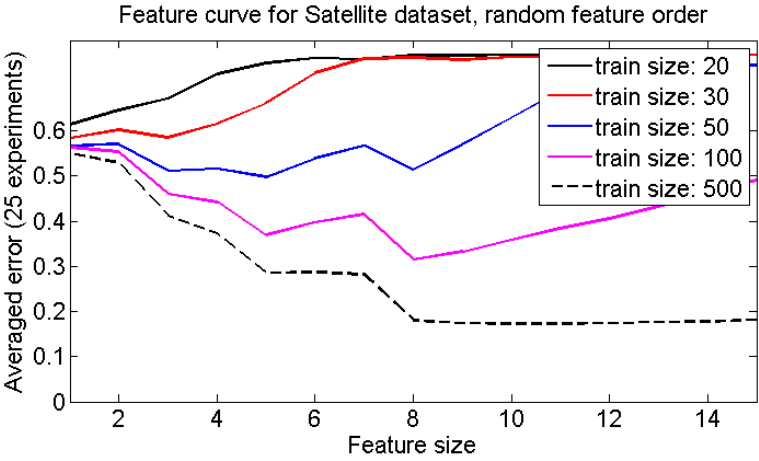


Figure 12: Satellite dataset *feature curves*, random order.

error estimations 8.1. The larger the training set, the more features can be used.

The result depends on the order of the features. In Figure 11 the features are ranked according to a forward selection procedure based on the Mahalanobis distance. In Figure 12 features are randomly ordered.

The dataset has 36 **features** of which most are uninformative. Consequently, in a random order many uninformative **features** may be used in the beginning. After 8 **features** it is almost sure that the most significant **features** (or very similar copies) are included.

## 8.4 Accuracy guidelines

### 8.4.1 The number of features.

From the discussions on the **evaluation** in **chapter 8** it is clear that there is a relation between **training set** size and **feature** size: the more **features** are added, the larger the **training set** should be to obtain a similar **accuracy** for the **trained** result. This holds for the increasing number of parameters involved in the **classifier**. The added **features**, however, might be informative, so in spite of the increasing noise in the parameter estimates, the **classifier** error may still decrease. The optimal number of **features** for a given size of the **training set** thereby depends on the separability added by the new **features**. The sensitivity for the noise in the data also depends on the characteristics of the chosen **classifier**.

Consequently, there is no general rule for the number of **features**, unless their distribution, the size of the **training set** and the chosen **classifier** are specified.

### 8.4.2 The size of the training set

By studying the **learning curves** it can be concluded that in expectation the more **training objects** the better, but that at some moment this will hardly help for the given **representation** and **classifier**. When the **apparent error** is close the **test error** the asymptotic value is reached. As the asymptotic behavior is exponential a very rough rule of thumb is that by doubling the **training set** size not more will be gained in **accuracy** than the difference with the error at half the present size.

**Classifiers** that are based on **covariances** need at least more **objects** than **features** for estimating a non-singular matrix that can be inverted, otherwise a proper **regularization** is needed. **Classifiers** that are based

on density estimation need to fill the space. As a rule of thumb it is sometimes stated that [training sets](#) of 5-10 times the dimensionality are needed. This holds per [class](#) as the densities are computed class-wise. However, it appears that with some [regularization](#) for much less samples reasonable results may be obtained.

### 8.4.3 The size of the test set

Here clear answers are possible. If one estimates an error with true value  $\epsilon$ , based on an independent randomly selected (i.i.d) [test set](#) of size  $n$ , then the standard deviation of the estimate is  $\sqrt{\epsilon \times (1 - \epsilon)/n}$ . If one likes to have this smaller than (e.g.)  $0.1\epsilon$  it directly follows for small  $\epsilon$  that  $n > 100/\epsilon$ .

This result is for some problems shocking. In case  $\epsilon = 0.1$  a [test set](#) of size  $n = 1000$  is needed to estimate the error with a standard deviation of 0.01. This results in a 95% [confidence](#) interval of about  $0.08 < \epsilon < 0.12$ , which is not very accurate. A [test set](#) of 1000 [objects](#), however, is for many applications very large.

# Chapter 9

## Exploratory data analysis

The final target of **pattern recognition** is to design a system that, by the two steps of **representation** and **generalization**, is able to correctly classify new **objects** with a sufficient **accuracy**. There are well defined **evaluation** tools to predict or measure the obtained performance. In its design, however, some additional knowledge of the data characteristics may be helpful to select and adjust the appropriate procedure. In this chapter some approaches are discussed that may be helpful for this.

### 9.1 Cluster analysis

In **cluster analysis** a grouping of the **objects** is studied without using **class** information supplied by the teacher. It is thereby an **unsupervised learning** technique (like **PCA**). The target is to find **clusters**: subgroups of similar **objects** in a given set of **objects**. We consider two types of **representations**, vector spaces and dissimilarities.

**Generalization** is not necessarily the target of **cluster analysis**. However, it may be followed by the computation of a **classifier** to assign new **objects** to one of the detected **clusters**. Here we will restrain to possible **clustering** procedures.

Like for **classification**, distances and densities play an important role in the design of **clustering** procedures. There are three kinds of approaches. For each of them the main algorithms will be shortly characterized.

#### 9.1.1 Hierarchical techniques

These are characterized by the construction of a hierarchy of **clusters**. On the lowest level every **object** is defined as a separate **cluster**. On every level above that the two nearest **clusters** are merged, until in the top level all **objects** are collected into a single **cluster**. The various techniques differ in the way the distance between two **clusters** is defined. The three main distance definitions are:

**Single linkage:** the distance between two **clusters** is the minimum distance between any two **objects** of the two **clusters**.

**Complete linkage:** the distance between two **clusters** is the maximum distance between any two **objects** of the two **clusters**.

**Average linkage:** the distance between two **clusters** is the average distance of all distances between two **objects** of the two **clusters**. This is different but might be similar as the distance between the **cluster** means of the **objects** are given by a **vector representation**.

These approaches can be used for **objects** given by a **dissimilarity representation** as well as by a **vector representation**. In the latter case **object** distances are usually defined by the **Euclidean metric**.

The entire hierarchy can be represented by a **dendrogram**: a **graph tree** in which every node represents the merge of two **clusters** and the size of the edges is determined by the distance between the corresponding **clusters**. The **dendrogram** is used to determine an appropriate **cluster** level and to find outlier **objects** that do not belong to any **cluster**. This post-analysis determines the number of **clusters** that is found.

### 9.1.2 Partitional techniques

These techniques are based on a desired number of **clusters**. They start from some initial clustering, which might be random. In an iterative procedure the clustering is implicitly or explicitly optimized according to some procedure. Some well known algorithms are:

**k-Means:** In every iteration a new clustering is defined by assigning all **objects** to the nearest **cluster** mean derived from the previous clustering.

**k-Centers:** This is also called k-Medians. In every iteration a new clustering is defined by assigning all **objects** to the nearest **cluster** center derived from the previous clustering. The center of a **cluster** is the **object** in the **cluster** for which the distance to the most remote **object** in the **cluster** is minimum.

**EM clustering:** Expectation-Maximization clustering. This is in fact a [generalization](#) of the above procedures. In every iteration a new clustering is defined by assigning all [objects](#) to the [cluster](#) to which it fits best according to a model describing the [clusters](#). These models are derived from the previous clustering. A well known option for these models is the [normal distributions](#). In PRTools an arbitrary [classifier](#) can be used.

### 9.1.3 Mode seeking

In the mode seeking strategy [clusters](#) correspond to the modes (local maxima) of the density function. [Objects](#) are assigned to the [cluster](#) of the mode that is found by following the density gradient upwards. The two main procedures are based on the non-parametric density estimates based on [kernels](#) and nearest neighbors. In both cases there is width parameter that influences the smoothness of the density estimate and thereby the number of modes and the number of [clusters](#).

**Mean shift:** This procedure is based on [Parzen](#) density estimates. The modes are found by following the gradient from a number of start points ([objects](#)). This is time consuming and mainly feasible in for lower dimensional vector spaces.

**k-NN mode seeking:** In the k-NN approach densities are related to the distance to the k-th neighbor. Pointers are set from all [objects](#) to the [object](#) with the highest density in their neighborhood. Following pointers is fast and modes are uniquely defined ([objects](#) that point to themselves).

The mean shift algorithm requires less [objects](#) than k-NN mode seeking and is frequently used for color image [segmentation](#). k-NN mode seeking has no restrictions for the dimensionality and can be applied to very large datasets by two steps in which the distances of all [objects](#) to all other [objects](#) have to be computed.

## 9.2 Visualization

**Visualization** tools may give the analyst some insight into the set of **objects** under study. This can be helpful to select an appropriate strategy for **representation** or **generalization**. Here three types of tools are shortly summarizes,

### 9.2.1 Scatterplots

Multi-dimensional data can be visualized by 2D or pseudo-3D projections. It has to be realized that if the intrinsic dimension is high many **objects** may be shown close to each other in the projection, but that have very large distances. Three well-known techniques for linear projection are:

**PCA:** Principal component analysis finds, by an **eigenvalue decomposition** of the **covariance matrix**, the linear subspace that shows the largest variances.

**LDA:** Linear discriminant analysis or Fisher mapping shows the subspace in which the **between scatter** (the variances of the **class** means) is optimized with respect to the **within scatter** (the variances within the classes).

**KLT:** Karhunen-Loeve transform is sometimes used as an alternative name for **PCA**. Sometimes it is distinguished from that by computing an **eigenvalue decomposition** of the average of the **class covariance matrices**. It thereby computes the linear subspace that shows the largest **within scatter**.

It is likely that in high-dimensional datasets **objects** are in a lower-dimensional non-linear subspace. Three popular techniques for finding nonlinear mappings are:

**MDS:** Multi-dimensional scaling is a general, classical procedure that aims to create a low-dimensional vector subspace in which the **objects** are positioned such that some measure for the difference

between the original and realized distances (called the stress) is minimized.

**KPCA** **Kernel PCA**. This is a kernelized version of **PCA**. The non-linearity is determined by the choice of the **kernel**.

**tSNE** A recent procedure called t-distributed stochastic neighbor **embedding**. Like the other procedures similar **objects** are modeled by nearby points and dissimilar **objects** are modeled by distant points in the **scatter plot**. In this case there is, however, a high emphasis on the first. The criterion used here is to minimize the Kullback-Leibler divergence between the original and realized distributions of **objects** distances.

### 9.2.2 Graph trees

Somewhat more general than a **scatter plot** of a (non-linear) subspace is to show relations between **objects** is a **graph**. The nodes are the **objects** and the edges show something of their relations. A **graph tree** is an undirected **graph** in which any two nodes are connected by exactly one path. Some examples:

**Decision tree**: This is a visual **representation** of the **Decision Tree classifier** mentioned in **section 7.2**. Every node is based on a subset of the original **training set**. The top node contains the entire set. At every node the subset is split (using some simple **classifier**, often a threshold on a **feature**) into some subsets. When the subsets are (sufficiently) homogeneous the node is not split further and a **class label** is assigned to such an end node. An original **vector representation** is hereby implicitly split into a number of cells containing (almost entirely) **training objects** of the same class.

**Dendrogram**: A **dendrogram** shows by a **graph tree** the result of a **hierarchical clustering**. The structure is similar to the **decision tree**. A top node contains all **objects**. The end nodes are the basic **clusters** or the individual **objects**. The edges combine the nodes hierarchically.

**MST** In a **Minimum Spanning Tree** all **objects** are a node and the edges represent their distances. The spanning of such a **graph tree** is the sum of all distances that are represented by an edge. The MST is the tree with the minimum spanning. It is the result of the single linkage **hierarchical clustering** is by that procedure sequentially the **clusters** of the two most neighboring **objects** in different **clusters** are connected.

### 9.2.3 Curves

Various function can be plotted to characterize some properties of a (labeled) dataset. Here are some of the main ones.

**Eigenspectra:** The spectrum of a **covariance matrix** is the set of its **eigenvalues**. With the not so often used word **eigenspectrum** we point to the curve that plots the ranked eigenvalues. It shows graphically how many **eigenvectors** are significant, which is the dimensionality of the linear subspace in which most of the variability is concentrated. The cumulative **eigenspectrum** may show this even more clearly, e.g. for what dimensionality 95% of the total variance (the sum of all eigenvalues) is reached.

**Density plots:** Plots of a two-dimensional **probability density distribution** can be shown by equal-density curves in a **scatter plot**. The density plot of a single **feature** is usually shown as a function (the density) along a **feature** axis. Multiple curves in the same plot may be used to study **densities** of different classes, showing the **class separability** of that **feature**.

**Learning curves:** These show some performance measure (e.g. the **classification error**) as a function of the size of the **training set**, see [section 8.2](#).

**Feature curves:** These show some performance measure (e.g. the **classification error**) as a function of the dimensionality, see [section 8.3](#).

# Chapter 10

## Glossary

This chapter summarizes the terminology as it is used in the previous chapters. Some additional terms are included as well. Almost all occurrences of the terms in the book, including the ones in this chapter, are linked to the corresponding glossary entry. The headings of the glossary entry are linked backward to the most significant place in the main text.

At the end the description of every glossary entry backward links are provided to pages, chapters and sections. In addition, external links are given to papers, Wikipedia entries, tutorials, video's and other internet locations. A browser is needed to support this. For a number of entries, also links are supplied to Matlab examples written by PRTools.

**Adaboost** An advanced [combined classifier](#) based on sequentially generating a large set of [weak classifiers](#) emphasizing [training objects](#) that are often incorrectly classified by one of these (boosting). [page-34](#) [wiki](#) [tutorial](#) [paper](#) [video](#) [example](#)

**Adaptation** This is not a specific pattern recognition term. It is used here for the step by which the [representation](#) is optimized for the generalization, e.g. for [classification](#). [page-12](#) [page-17](#)

**Apparent error** Also called [resubstitution error](#) or [training error](#). It is the fraction of [objects](#) used for designing ([training](#)) a [classifier](#) that is incorrectly classified by that [classifier](#). It is usually a positively biased estimator of the [true classification error](#). [page-35](#) [example](#)

**Attribute** A symbolic or numeric property of an [object](#) that might be useful to determine its class. The word [feature](#) is used for this as well. Different [objects](#) however may have different numbers of attributes. Usually, the same set of [features](#) is measured for all [objects](#) in a single problem. Thereby, [objects](#) can be represented by a feature [vector](#), or by a set of [attributes](#). [page-12](#)

**Backward search** See [backward selection](#).

**Backward selection** The selection of [features](#) or [prototypes](#) on the basis of the performance decrease after removal from an already selected set. The selection is usually started with a large set of best performing individuals, or with the entire available set. See also [feature selection](#). [page-28](#) [example](#) [example](#)

**Base classifier** One of the constituting [classifiers](#) in a [combined classifier](#) like [adaboost](#) or a [random forest](#). See also [ensemble classifier](#). [example](#)

**Bayes classifier** This is a [classifier](#) based on the [Bayes rule](#). It combines given [class prior](#) probabilities  $P(\omega)$  with [class probability densities](#)  $P(x|\omega)$  for the [object vector representation](#)  $x$  such that the [classification error](#)  $\epsilon$  is minimum under the assumption that these probabilities and densities hold for the [objects](#) to be classified. If this assumption is violated (e.g. by using bad density estimates) the [classifier](#) may be far from optimal.

For a two-class problem with  $\omega \in A, B$  this [classifier](#) can be written as:

$$S(x) = P(x|A)P(A) - P(x|B)P(B),$$

$$\text{if } S(x) > 0 \text{ then } A \text{ else } B$$

For a multi-class problem with classes  $\omega \in \Omega$  the [classifier](#) a more general formulation is:

$$\hat{\omega}(x) = \operatorname{argmax}_{\omega \in \Omega} \{P(x|\omega)P(\omega)P(x)\}$$

As the denominator  $P(x)$  does not depend on  $\omega$ , it can be omitted. The rule thereby simplifies to:

$$\hat{\omega}(x) = \operatorname{argmax}_{\omega \in \Omega} \{P(x|\omega)P(\omega)\}$$

[page-32](#) [wiki](#) [example](#)

**Bayes normal classifier** The Bayes classifier using density estimates based on the [training set](#) assuming [normal distributions](#). The result is either a quadratic [classifier](#) (different [covariance matrices](#)), also named [QDA](#), or a linear [classifier](#) (equal covariance matrices), also named [LDA](#). [page-32](#) [tutorial](#) [example](#)

**Bayes error** The classification error made by the Bayes classifier for the case its assumptions are correct. By definition this is the lowest classification error that can be achieved for the given problem.[page-32](#) [tutorial](#) [wiki](#) [example](#)

**Bayes rule** For a class  $\omega$  and an object representation vector  $x$ , this rule relates the class posterior probability  $P(\omega|x)$  with the class probability density  $P(\omega|x)$ , the class prior probability  $P(\omega)$  and the joint probability density function  $P(x)$  over the set of classes:

$$P(\omega|x) = \frac{P(x|\omega)P(\omega)}{P(x)} = \frac{P(x|\omega)P(\omega)}{\sum_{\omega \in \Omega} P(x, \omega)}$$

[tutorial](#) [video](#) [wiki](#) [wiki](#)

**Between scatter** The estimated variance or covariance of the means of a collection of sets of data points. It is thereby the scatter of the set means. The averaged scatter of the sets is called the within scatter. The two scatters constitute together the scatter of the combined sets. [page-28](#) [paper](#) [paper](#) [wiki](#)

**Binary classification** A classification problem in which a discriminant between two classes has to be found. [page-31](#)

**Boosting** Training classifiers by emphasizing the objects that are likely to be misclassified. It is specifically used in training ensemble classifiers, e.g. by the adaboost algorithm. [video](#) [wiki](#)

**Branch and bound** The optimization of some selection (e.g of features or prototypes) on the basis of a monotonic criterion, based on backward selection. The branch and bound algorithm minimizes the search tree by using the monotonicity. See also [feature selection](#). [page-28](#) [paper](#) [wiki](#)

**Class** A class is a set of objects that within a given context is recognized as similar. Such a class has usually a unique name, the class name. The individual objects within a class have a label that refers to this name. If the class is human defined it is sometimes also called a concept. [page-12](#)

**Class frequency** The frequency of which [objects](#) of a particular [class](#) in a dataset. It can be used as an estimate for the [class prior](#) in case the dataset is representative for the [classification problem](#). [page-35](#)

**Class label** A pointer assigned to an [object](#) that refers to its [class](#) or [class name](#). Note the inconsistency in terminology: an [object label](#) is the same pointer, from an [object](#) to a [class](#). [page-12](#)

**Class membership** The [class](#) to which a [class label](#) points, in case of [crisp labels](#), or the value(s) of the [soft labels](#). [page-10](#)

**Class name** A name assigned to a [class](#) of [objects](#). Sometimes very symbolic like A or B, sometimes with a very practical meaning like 'healthy' or 'diseased'.

**Class separability** Some general measure related to the possible [classification performance](#) given an [object representation](#) and a [training set](#). [page-27](#) [tutorial](#)

**Class posterior** The [probability](#)  $P(\omega|x)$  of a particular [class](#)  $\omega$  for a given [object](#)  $x$ . The difference with the [class prior](#) is that the [posterior probability](#) depends on the observed [object](#). See also [confidence](#). [page-32](#) [paper](#)

**Class prior** The [probability](#)  $P(\omega)$  that an arbitrary [object](#) belongs to [class](#)  $\omega$ . The difference with the [class posterior](#) is that the prior is independent of an observed [object](#). [page-35](#)

**Classification** The assignment of a [class](#) (in fact a [class name](#)) to an [object](#) by [evaluating](#) a [trained classifier](#) for that [object](#). [page-14](#) [page-18](#) [paper](#) [tutorial](#) [wiki](#)

**Classification accuracy** This is one minus the [classification error](#). Sometimes, the accuracy is also multiplied by 100 and presented as a percentage. See also [classification performance](#). [page-35](#)

**Classification error** The [probability](#) that an arbitrarily selected [object](#) from a set of classes is incorrectly classified by a given [trained classifiers](#). Alternatively it is used for the fraction of [objects](#) of a given finite set of [objects](#) that is incorrectly classified. See also [apparent](#)

error (or [training error](#)), [test error](#) and [true error](#). Sometimes, this error is also multiplied by 100 and presented in percentage. [page-35](#)  
[section 8.1](#) [paper example](#)

**Classification performance** A general expression referring to how well a [classifier](#) classifies unseen [objects](#). There is no sharp mathematical definition of the performance. In general, the higher the performance, the higher the [classification accuracy](#) and thereby the lower the [classification error](#). [paper paper tutorial](#)

**Classification problem** The problem for which a [classifier](#) has to be found. Usually it is defined for a given [representation](#) and [design set](#). See also [recognition problem](#). Some specific problems are the [one-class classification](#) problem (finding a boundary around a single class), the [binary classification](#) problem (finding a [discriminant](#) between two classes) and the [multi-class classification](#) problem (finding a classifier between a number of classes, possibly two). [page-31](#)  
[wiki](#)

**Classifier** A classifier is a rule that assigns a [class label](#) to any [object](#) in a particular [object representation](#).

Some classifiers may [reject](#) some [objects](#). Some classifiers may assign multiple [class labels](#). Instead of or in addition to [class labels](#) classifiers may output [class posteriors](#), [confidence](#), distances or densities for all possible [classes](#). In the additional step, the most likely [class](#) or [classes](#) have to be determined.

The word classifier is used for both, [untrained](#) as well as [trained classifiers](#). The first refers to the rule, the way the classifier is [trained](#). Sometimes this is also called a [learner](#). The second refers to the realized version, the function that assigns [labels](#) or [confidences](#). In [section 7.2](#) a set of classifiers is listed. [page-10](#) [page-13](#) [page-30](#)  
[section 7.2](#) [paper wiki example](#)

**Cluster** A set of [objects](#) that are similar in their [representation](#). It depends on the problem and the [representation](#) whether classes and [clusters](#) coincide. In a single [class](#) sometimes several [clusters](#) can be distinguished. [page-43](#)

**Cluster analysis** The study of a dataset by comparing the results of various clusterings. [page-43](#) [section 9.1](#) [paper](#) [paper](#) [tutorial](#) [video](#) [wiki example](#)

**Clustering** The process of finding [clusters](#). It is a way of [unsupervised learning](#). Three different approaches are the [hierarchical techniques](#), the [partitional clustering](#) and [mode seeking](#). [page-43](#) [paper](#) [video](#) [example](#)

**Combined classifier** An advanced [classifier](#) constituted by a set of more simple [classifiers](#) (called [base classifiers](#)). They are combined by the combining [classifier](#) according to some [combining rule](#). If the [base classifiers](#) are of the same type, the combined classifier is called an [ensemble classifier](#). Well known examples are [adaboost](#) and the [random forest classifier](#). A [neural network classifier](#) may also be considered as a combined classifier. [paper example](#)

**Combining rule** The way a set of [classifiers](#) is combined into a single one. The combining rule can be a fixed rule like majority voting, the average or the product of [posterior probabilities](#). It can also be a trained combiner. [paper](#) [paper](#) [tutorial](#) [example](#)

**Concept** A concept is a general idea, or something conceived in the mind. In [pattern recognition](#) it is sometimes used for a [class](#) or a [cluster](#) for which the underlying set of [objects](#) is the complete set of specific examples for which the concept is applicable (occasionally a single representative or typical [object](#) is also called a concept). The concept is the step that [consciousness](#) makes after recognizing [patterns](#) in [observations](#) and before a name (a word) has been assigned to it. [wiki](#)

**Confidence** A value between 0 and 1 that indicates the likelihood that a particular outcome (e.g. [class membership](#)) is correct. This term is used instead of [posterior probability](#) when probabilities in the strict sense are not well defined. Class confidences may, like [class posteriors](#), be interpreted as [soft labels](#). [page-30](#) [paper](#) [example](#)

**Consciousness** The ability of a thinker, observer or actor to observe his own state of mind. [page-4](#)

**Covariance** A statistical measure for the linear dependence of two stochastic variables:

$$\text{cov}(x, y) = E[(x - E[x])(y - E[y])]$$

The covariance of a variable with itself is called variance. In [pattern recognition](#) the covariance is often used to study the relation of two [features](#). See also [covariance matrix](#). [page-31](#)

**Covariance matrix** This symmetric matrix encodes all [covariances](#) of a set stochastic variables, e.g. the [features](#) used for [representation](#). Multi-dimensional [Gaussian distributions](#) are fully described by their mean vector and the covariance matrix. [page-31](#) [video](#) [wiki example](#)

**Crisp label** [Labels](#) point from an [object](#) to a class. They are assigned by an expert or estimated during automatic [classification](#). If there is no uncertainty or ambiguity such a [label](#) is unique and symbolic. It just contains the name of the class. See also [soft labels](#). [page-30](#)

**Cross-validation** An [evaluation](#) system in which the [design set](#) is repeatedly split into a [training set](#) and a [test set](#). Each time the system is [trained](#) and [tested](#). Next, the roles of [training](#) and [test sets](#) are reversed, and the obtained performance estimates are averaged.

In  $n$ -fold cross-validation the [design set](#) is split into  $n$  partitions. In  $n$  rounds  $n - 1$  partitions are used for [training](#) and one for [testing](#). Afterwards all performance estimates are averaged. Common values for  $n$  are 5 or 10. As the result depends on the random partition,  $n$ -fold cross-validation is sometimes repeated a number of times to get a more stable result, needed for a reliable comparison of different [training](#) systems. [page-36](#) [paper](#) [paper](#) [video](#) [wiki example](#)

**Curse of dimensionality** Statistics in high-dimensional spaces tend to be bad. This is related to Rao's paradox, the [peaking phenomenon](#) and to [overtraining](#). [page-27](#) [paper](#) [quora](#) [wiki example](#)

**Decision forest** A combined classifier constituted by of a large set of decision tree classifiers. Some or all might be very simple, weak classifiers like the decision stump. Also random forest. [page-34](#) [paper](#) [wiki](#)

**Decision function** A function of observable variables that makes a crisp choice between two or more options. A classifier is a decision function. [page-31](#)

**Decision stump** A weak classifier based on a decision tree consisting of just a single decision step, usually optimized for a random subset of features and/or a random subset of the training set. See decision forest and adaboost. [paper](#) [wiki](#) [example](#)

**Decision tree** A classifier based on a sequence of elementary decisions (usually thresholds on a feature) that can be understood as a graph tree. At the root the object to be classified comes in. By the sequence of decisions it arrives at an end node that is assigned to one of the classes. [page-33](#) [page-47](#) [paper](#) [tutorial](#) [wiki](#)

**Dendrogram** A graph tree used for visualizing the results of hierarchical clustering. The tope node stands for the entire dataset. Downward nodes show splits into smaller clusters. Bottom nodes may correspond to the individual objects. The lengths of the edges can be related to the clustering strength. [page-44](#) [page-47](#) [wiki](#) [example](#) [example](#)

**Density** Often used as a short for probability density. [page-48](#)

**Density estimation** The estimation of a probability density function (PDF), usually in a vector space, on the basis of a training set. PDFs are used in the Bayes classifier. This results in classifiers like Bayes normal, Parzen and naive Bayes, depending on the procedures and assumptions used for estimating the density. [page-31](#) [wiki](#) [wiki](#) [example](#)

**Design set** The total set of objects that is available for the designer of a pattern recognition system. It may be split (repeatedly) into

subsets like the [training set](#), [validation set](#) and [test set](#). [page-13](#)  
[page-19](#) [page-36](#)

**Dimension reduction** Transformation of a given [vector representation](#), usually a [feature space](#) into another one with a lower dimension than the original one, while maintaining the information content. This content is either expressed in the [object variability](#) (unsupervised) or the [class separability](#) (supervised). [page-27](#) [section 6.3](#) [wiki](#)

**Dipping** The phenomenon that the [learning curve](#) of some [classifier](#) for a particular [classification problem](#) shows an optimum for some size of the [training set](#). [paper](#) [example](#)

**Discriminant** A function that decides between two possibilities, in [pattern recognition](#) usually between two [classes](#). [Decision function](#) and [classifier](#) are more commonly used than 'discriminant'. The latter is mainly found in [Fisher's Linear Discriminant](#) (FLD) and [Linear Discriminant Analysis](#) (LDA). [wiki](#) [example](#)

**Dissimilarity** A measure for the difference between two [objects](#). This can be a proper metric but may also violate the triangle inequality or be asymmetric. The main characteristics of a dissimilarity measure are usually: (1) the property of identity holds: it is zero if and only if the [objects](#) are identical, and (2) it is monotonic: the more different the [objects](#) the larger the dissimilarity. See [dissimilarity representation](#) and [dissimilarity space](#). [post](#)

**Dissimilarity representation** In this [representation](#) [objects](#) are represented by pairwise dissimilarities to other [objects](#), either directly computed from the real world [observations](#) or from another [representation](#), e.g. [features](#) or [graphs](#). The next step may be the postulation of a [dissimilarity space](#) or [embedding](#). [page-22](#) [page-26](#)  
[post](#) [paper](#) [paper](#) [tutorial](#)

**Dissimilarity space** The vector space defined by the dissimilarities to a [representation set](#) of [objects](#). The vector length (space dimensionality) is thereby equal to the size of the [representation set](#). This set can be the entire [training set](#), a selection of these [objects](#), but possibly also a set of entirely different [objects](#). [Classifiers](#) in this

space can be [trained](#) in a similar way as in a [feature space](#). [page-26 post paper tutorial example](#)

**Eigenvalue decomposition** The decomposition of a square matrix  $X$  into a diagonal matrix  $D$  and a full matrix  $V$  such that  $XV = VD$ . The diagonal elements of  $D$  are the eigenvalues and the row vectors of  $V$  constitute a set of orthonormal eigenvectors.

An important application in [pattern recognition](#) is the eigenvalue decomposition of the [covariance matrix](#) of some [vector representation](#) of the objects. If the eigenvectors are ranked according to decreasing eigenvalues then the first  $n$  eigenvectors constitute the linear subspace for which the sum of the squared distances of the [objects](#) to this subspace is minimal. It is assumed that the dataset mean has been shifted to the origin first. See also [eigenspectrum](#). [page-46 wiki example](#)

**Eigenspectrum** A plot of the ranked eigenvalues. Usually applied to the [covariance matrix](#) of a [vector representation](#) of a set of [objects](#). It shows the data variances in the directions of the eigenvectors. Often the cumulative eigenspectrum is preferred for visualization purposes. [page-48 paper example example](#)

**Embedding** The computation of a set of vectors in an Euclidean space for which the distances are equal to a given set of dissimilarities or the inner products are equal to a given set of similarities (isometric embedding), see [Euclidean geometry](#) and [MDS](#). [page-26 wiki wiki](#)

**Ensemble classifier** A subset of the family of [combined classifiers](#). In an ensemble all base classifiers are of the same type, e.g. [decision stumps](#) used in [adaboost](#). [wiki tutorial example example](#)

**Error estimation** Procedure to estimate the [classification error](#) of a classifier, e.g. by using a [test set](#) or [cross-validation](#). [page-35 section 8.1 paper paper example](#)

**Euclidean geometry** A geometry based on Euclid's axiomatic system, the Elements. The traditional [vector representation](#) in [pattern recognition](#) is usually equipped with a Euclidean geometry. Distances are based on the Euclidean metric. [page-26 wiki wiki](#)

**Evaluation** Test of the performance of (a part of) a [pattern recognition](#) system. This may best be done by an independent [test set](#). [page-12](#) [page-35](#) [chapter 8](#) [short-tutorial](#) [long-tutorial](#) [video](#) [example](#)

**Feature** A symbolic or numeric property of a real world [object](#) that might be useful to determine its class. The word '[attribute](#)' is used for this as well. Different [objects](#) however may have different numbers of [attributes](#), while usually for all [objects](#) in the same problem the same features can be measured. Thereby [objects](#) may be represented by a [feature vector](#), or by a set of [attributes](#). [page-12](#) [page-15](#)

**Feature curve** A plot that shows the performance of a trainable system (e.g. a [classifier](#)) as a function of the number of [features](#) used for training. [page-39](#) [page-48](#) [section 8.3](#) [example](#) [example](#)

**Feature extraction** The process of determining good [features](#) for a [feature representation](#) of [objects](#). This may refer to raw data like images or time signals, but also to already given representation. In the latter case the aim is to simplify the [representation](#), e.g. by [dimension reduction](#). Examples are [PCA](#) and [LDA](#). [page-18](#) [page-27](#) [tutorial](#) [video](#) [wiki](#) [example](#)

**Feature representation** The representation of objects by features resulting in a [feature space](#). [page-24](#)

**Feature scaling** See [scaling](#). [page-23](#) [wiki](#) [example](#)

**Feature selection** Reducing the dimensionality of a [feature space](#) by the selecting of a subset of the [features](#). Procedures for [feature selection](#) consist of a criterion (a [filter](#) approach or a [wrapper](#)) and a strategy (e.g. [individual selection](#), [forward selection](#), [backward selection](#), [branch and bound](#), [floating search](#)). [page-27](#) [paper](#) [paper](#) [post](#) [tutorial](#) [video](#) [video](#) [wiki](#) [example](#) [example](#)

**Feature space** The vector space defined by the [feature vectors](#). This concept is so familiar in statistical [pattern recognition](#) and machine

learning that sometimes also vector spaces used for [object representation](#) but created otherwise, are also called feature spaces. Examples are the [kernel space](#) and the [dissimilarity space](#) which are defined by functions of input [features](#) or distances between [objects](#). [page-8](#)

**Feature vector** The vector storing all relevant properties of a real world [object](#) in a particular well defined order. It may also be the unfolded set of image pixels ([pixel space](#)) or a sampling of a spectrum or a time signal. If feature vectors are used to represent a set of [objects](#) in a vector space it is necessary that the [images](#), spectra or signals are converted to the same size before sampling.

**Filter** In general a filter transforms an input (stream) by a fixed procedure into an output (stream). In relation with [feature selection](#) it has the specific meaning that it refers to criteria that are different, usually less complex, than the final [classifier](#) to be used after [feature selection](#). The latter would be the [wrapper](#) approach, which is more focussed to the final use of the [features](#) to be selected. The filter approach, however, has the advantage that it might be faster (more simple to compute) and may avoid [overtraining](#). Filtering is especially important if [feature selection](#) is done because the computation of the target [classifier](#) is troublesome for the given [feature](#) size. [page-29](#) [paper](#) [wiki example](#)

**Fisher classifier** Usually called Fisher's Linear Discriminant (FLD). This is the traditional linear [classifier](#) optimizing the [between scatter](#) w.r.t. the [within scatter](#). This [classifier](#) is based on the same criterion as [Linear Discriminant Analysis \(LDA\)](#). Also the classifier is thereby occasionally called LDA. [page-32](#) [paper](#) [paper](#) [video](#) [wiki example](#)

**Fisher mapping** The orthonormal transformation of a [feature space](#) that maximizes the [class between scatter](#) w.r.t. the average [class within scatter](#). It is also called LDA. [video](#) [wiki example](#)

**FLD Fisher's Linear Discriminant**, see [Fisher classifier](#) [page-32](#) [paper](#) [video](#) [wiki example](#)

**Forward search** See [forward selection](#).

**Forward selection** The selection of [features](#) or [prototypes](#) on the basis of the performance increase after addition to an already selected set. The selection is started with the best performing individual. See also [feature selection](#). [page-28](#) [example](#) [example](#)

**Floating search** A [forward selection](#) procedure of [features](#) or [prototypes](#) including [backward steps](#). See also [feature selection](#). [page-28](#) [paper](#)

**Gaussian distribution** See [normal distribution](#). [page-31](#) [tutorial video](#) [wiki](#) [example](#)

**Generalization** Generalization is the step from the [observations](#) of a set of [objects](#) to their common properties or [concept](#). In logic this is called induction. In [pattern recognition](#) it is the process of finding [clusters](#), [classes](#) or typical [objects](#). If for a new [object](#), that was not used in the generalization, its most similar cluster, [class](#) or typical [object](#) can now be determined. Thereby a property may be predicted that has not been measured. [page-5](#) [page-12](#) [page-30](#) [chapter 7](#) [wiki](#) [wiki](#)

**Graph** A symbolic representation of data: nodes (or vertices) connected by edges. In [pattern recognition](#) used for [visualization](#), [representation](#) (see [graph representation](#)) and [generalization](#) (see [hierarchical clustering](#) and [decision trees](#)). [page-47](#) [wiki](#)

**Graph tree** A connected [graph](#) in which any two nodes (vertices) are connected by exactly one path. [page-47](#) [wiki](#)

**Graph representation** A structural object representation by a set of nodes partially connected by edges. Nodes (vertices) and edges may be attributed. [page-16](#) [page-22](#) [paper](#) [paper](#) [wiki](#)

**Grid search** A way to find an optimal parameter setting, e.g. hyperparameters used for [regularization](#). The possibly multi-dimensional domain of the parameters (e.g. of a [classifier](#)) is sampled by a grid and for every parameter vector on the grid a criterion (e.g. the

performance of the [classifier](#)) is determined. The best parameter vector is used. [page-34](#) [wiki](#)

**Hierarchical clustering** A [clustering procedure](#) in which small [clusters](#) (initially possibly single [objects](#)) are iteratively merged into large ones, eventually into a single large [cluster](#). Hierarchical clustering may be visualized by a [dendrogram](#). [page-43](#) [subsection 9.1.1](#) [paper](#) [tutorial](#) [video](#) [wiki](#) [example](#)

**Hold-out error** The [classification error](#) of a [classifier](#) trained by the available [design set](#), estimated by the [classification error](#) found for a [classifier](#) trained by a randomly chosen subset of the [design set](#) by applying it to the unused part of the [design set](#) (the hold-out set). Sometimes the subsets have equal size and the roles of the two sets is reversed as well and results are averaged: 2-fold [cross-validation](#). [page-36](#) [paper](#) [wiki](#) [example](#)

**Image** A one-, two-, or multi-dimensional set of pixels. For many [pattern recognition](#) applications it is relevant that the assumption holds that neighboring pixels have a higher correlation than more remote ones. This information is lost by a straight forward [pixel representation](#). [page-16](#)

**Individual selection** The selection of [features](#) or [prototypes](#) on the basis of their individual contribution to some performance measure. See also [feature selection](#). [page-28](#) [example](#) [example](#)

**Intrinsic dimensionality** Formally this is the number of variables to represent a signal. For [vector representations](#) of [objects](#) this corresponds the dimensionality of the, possibly nonlinear, subspace that spans these [objects](#) with sufficient accuracy. [wiki](#) [example](#)

**Input space** Machine learning terminology for the space of the given (vector) [representation](#). It is usually used in relation with [kernel representations](#) as it refers to the input space for which [kernels](#) are computed. The input space is in this case identical to what is called [feature space](#) in [pattern recognition](#). [paper](#)

**Invariant** A property, often a (possible) feature, that has no relation with a particular class. Objects of that class may have all values of the invariant. For example, rotation angle is an invariant for most character classes, but not for the numbers '6' and '9', as they will be confused by rotation. [page-24](#)

**Iris dataset** A classical dataset in [pattern recognition](#). It is one of the first real world datasets used to illustrate (by R.A. Fisher) the operation and performance of a [classifier](#) ([Fisher's Linear Discriminant](#)). [data](#) [paper](#) [wiki](#) [example](#)

**Kernel** A function  $K(x, y)$  of two [objects](#)  $x$  and  $y$ . It is either computed from a [representation](#) like a [feature space](#), or from the raw data of  $x$  and  $y$  directly. A [training set](#) of  $m$  [objects](#) is converted by the kernel in an  $m \times m$  matrix. This may be used directly as a [representation](#), but much more often the [kernel trick](#) is used (when possible) to train [classifiers](#) in [kernel space](#). See also [dissimilarity representation](#) and [dissimilarity space](#). [page-33](#) [example](#)

[Kernels](#) are also used in [non-parametric density estimation](#), e.g. [Parzen](#), and in the [classifiers](#) based on it. [page-33](#) [example](#)

**Kernel space** If a [kernel](#) is separable, i.e. if  $K(x, y) = \phi(x)\phi(y)$  then the space  $\phi(x)$  and  $\phi(y)$  refer to is the kernel space. If the kernel satisfies the Mercer conditions, e.g. a Gaussian [kernel](#) (a radial basis [kernel](#)) or a polynomial kernel, this space is a Hilbert space. In [support vector machines](#) the [kernel space](#) is only used implicitly by the [kernel trick](#). [paper](#) [tutorial](#) [wiki](#) [wiki](#)

**Kernel trick** If a [kernel](#) is separable, i.e. if  $K(x, y) = \phi(x)\phi(y)$ , which is the case if  $K()$  fulfills the Mercer conditions, then the [kernel](#) may be interpreted as the inner product in the [kernel space](#). As some transformations and [classifiers](#), in particular the [support vector machines](#), can be written in terms of inner products only, they implicitly base their result on a virtual [kernel space](#), which might be of an infinite dimensionality. See also the Wikipedia article on this topic. [page-33](#)[post](#) [tutorial](#) [video](#) [wiki](#)

**KLT The Karhunen-Loeve Transform.** It is sometimes used as an alternative name for [PCA](#). In some toolboxes it is used for a slightly different use of [PCA](#), e.g. [PCA](#) based on the averaged class [covariance matrix](#) instead of the overall [covariance matrix](#). [page-46 example](#)

**$k$ NN The  $k$ -nearest neighbor classifier.** New objects are assigned to the majority class of the  $k$  [nearest neighbors](#) in the [training set](#). [page-33 example](#)

**KPCA Kernel PCA**, an extension of [PCA](#) by using the [kernel trick](#). Consequently a non-linear subspace is found, depending on the choice of the [kernel](#). Often this subspace is 2D and the purpose is [visualization](#) by a [scatter plot](#). [page-47 wiki example](#)

**Label** In general this is a tag or a pointer assigned to some entity in order to link it to some other entity. In a [pattern recognition](#) context it is often a short for [class label](#) as it links an [object](#) to a [class](#) by the name of that class. [page-12](#)

**Labeled object** An [object](#) with a known [class label](#). It may be used in a [training set](#) for [training](#) a [classifier](#) or in a [test set](#) for [error estimation](#). [page-30](#)

**LDA Linear Discriminant Analysis** stands for both, the linear transformation to the sub-space of a vector space under consideration for which the [class between scatter](#) is maximized w.r.t. the average [class within scatter](#), as well as for the [classifier](#) that assigns new objects to the [class](#) with the highest [posterior probability](#) under the assumption of Gaussian class distributions with equal [covariance matrices](#). LDA is also called Fisher mapping. [page-28](#) [page-32](#) [page-46 paper](#) [paper](#) [post](#) [tutorial](#) [wiki example](#)

**Learner** Sometimes used as alternative for an [untrained classifier](#): the learning rule to find a [classifier](#) from data. It is especially used in discussions on [weak](#) and [strong classifiers](#) (learners).

**Learning curve** A plot that shows the performance of a trainable system (e.g. a [classifier](#)) as a function of the size of the [training set](#). [page-37](#) [page-48 section 8.2 example example](#)

**Learning from examples** This is the general goal of [pattern recognition](#). One may gain knowledge from a [teacher](#), but if this is not available one should learn from experience, from the [observations](#) as they arise: [learning from examples](#). [page-6](#) [page-30](#)

**LOO Leave-One-Out cross-validation.** A [cross-validation](#) procedure which uses as many folds as [objects](#) in the [design set](#). In each fold a classifier is trained on  $n-1$  [objects](#) and [tested](#) on the remaining [object](#). As this is a systematic procedure, repeating makes no sense as it will result in the same [error estimate](#). [page-37](#) [example](#)

**LOO-error** The [classification error](#) as estimated by [LOO cross-validation](#). [page-37](#)

**Mahalanobis distance** The distance in a vector space between two sets of [objects](#), or two distributions  $A$  and  $B$ , defined by their means  $\mu_A$  and  $\mu_B$  and their common [covariance matrix](#)  $\Sigma$ .

$$D(A, B) = \sqrt{(\mu_A - \mu_B)^T \Sigma^{-1} (\mu_A - \mu_B)}$$

Using the same metric also the distance between an object  $x$  and a set or distribution of objects  $A$  may be defined:

$$D(A, B) = \sqrt{(x - \mu)^T \Sigma^{-1} (x - \mu)}$$

or between two [objects](#)  $x$  and  $y$  within a set or distribution with [covariance matrix](#)  $\Sigma$

$$D(A, B) = \sqrt{(x - y)^T \Sigma^{-1} (x - y)}$$

Note that if  $\Sigma$  equals the identity matrix the Mahalanobis distance equals the [Euclidean distance](#). [page-40](#) [wiki](#)

**MDS Multi-Dimensional Scaling** is a nonlinear technique for [embedding](#) a set of [objects](#) with a given [dissimilarity matrix](#) in a Euclidean vector space such that the distances in this space correspond to the given matrix. Often this vector space is 2D and the purpose is visualization by a [scatter plot](#). [page-46](#) [wiki](#) [example](#)

**Mode seeking** A clustering procedure based on the local maxima (modes) of the probability density function of all objects. [page-45 subsection 9.1.3](#) [paper](#) [paper](#) [wiki](#) [example](#)

**MST Minimal Spanning Tree:** A [graph tree](#) representing a set of objects as the nodes of the tree used for visualization the dataset. The length of the edges are the distances between the connected objects. The sum of all represented lengths is minimized. [page-48 video](#) [wiki](#)

**Multi-class classification** A [classification problem](#) in which a number of classes [classes](#) (two or more) have to be distinguished. Not all [classifiers](#) are able to distinguish directly more than two [classes](#). Such [binary classifiers](#) need to a [one-against-one](#) or a [one-against-rest](#) strategy to solve a general multi-class problem. [page-31 paper](#) [paper](#) [tutorial](#) [wiki](#) [example](#)

**Naive Bayes** Used for [classification](#) procedures based on the [Bayes classifier](#) and the assumption that all [features](#) are independent. Consequently, the [density functions](#) can be estimated [feature by feature](#), followed by a multiplication. Usually histograms are used for estimating the [feature densities](#). [page-33post](#) [tutorial](#) [video](#) [wiki](#)

**Nearest mean** [Classification](#) rule in which [objects](#) are assigned to the class of the nearest [class](#) mean derived from the [training set](#). [page-32 paper](#) [example](#)

**Nearest neighbor** The closest [object](#) in a set of [objects](#) (usually the [training set](#)) of a given [object](#) (e.g. a [test object](#)). The 1-nearest-neighbor rule (1-NN) is one of the traditional [classifiers](#). The  $k$ NN classifier, which selects the majority class among the  $k$  nearest neighbors in the [training set](#), approximates the [Bayes classifier](#) for an appropriate choice of  $k$ . The optimization of  $k$  is based on the assumption that the [training set](#) is selected according to the true [class density](#) distributions. [page-32 paper](#) [scholar](#) [video](#) [wiki](#) [wiki](#) [example](#)

**Neural network** Originally designed as a simulation of the nervous system. In [pattern recognition](#) it is a flexible [trainable](#) tool for defining

non-linear subspaces and non-linear [classifiers](#). It consists of a large set of simple units, called neurons, combining many inputs into a single output. Usually they are organized in layers. A non-linear neural network has at least three layers: the input layer (connected to the [features](#)), a hidden layer and an output layer (the [class confidences](#)). A very simple, degenerated neural network is the [perceptron](#), consisting out of a single neuron. Neural networks are usually trained sequentially. [page-33](#) [paper](#) [tutorial](#) [wiki](#) [wiki](#)

**Non-parametric estimation** Model estimation, e.g. a [probability density function](#), not based on optimizing a parametric model, e.g. a [normal distribution](#), but on a non-parametric function of all observations, e.g. a [kernel density estimator](#) ([Parzen](#)). See also [parametric estimation](#). [page-31](#) [tutorial](#) [wiki](#) [example](#)

**Normal distribution** A very common bell shaped one-dimensional or multi-dimensional distribution. It is very common as for many other distributions holds that the means of a set of independently randomly drawn random variables asymptotically approximate a normal distribution. The normal distribution,  $f(x, \mu, \Sigma)$  depends on just the class mean  $\mu$  and the covariance matrix  $\Sigma$ : [wiki](#)

$$f(x, \mu, \Sigma) = \frac{1}{(2\pi)^{k/2} \sqrt{|\Sigma|}} e^{-(x-\mu)^T \Sigma^{-1} (x-\mu)/2}$$

Normal distributions with the given parameters are symbolically denoted by  $N(\mu, \Sigma)$ . The [Bayes normal classifiers](#) are based on the assumption of normally distributed classes. Normal distributions are also named [Gaussian distributions](#). [page-31](#) [tutorial](#) [video](#) [wiki](#) [example](#)

**Normalization** Used to describe the process of removing [object](#) differences that are not relevant for the [classification](#), e.g. rotation or size. [page-23](#) [example](#)

**Object** A real world entity that can be physically observed. Also used for a [representation](#) of such an entity. Examples are 2-dimensional

items like photos, characters, plotted curves; 3-dimensional items like chair, tables, cells, airplanes; but also time dependent events like speech, gestures, movies. More abstract entities like a voice, a persons identity, style of writing, a composition are sometimes called an object. Another way to phrase it is that an object is a realization of a [concept](#), which corresponds to the [pattern recognition](#) terminology: a member (element) of a [class](#) (set). [page-8](#) [page-12](#) [page-15](#) [page-17](#)

**Object label** A pointer to the [class](#) to which an [object](#) belongs or may belong, e.g. its name. Note the inconsistency in terminology: a [class label](#) is often used in way: the same pointer from an [object](#) to a [class](#).

**Object representation** A formal description of an [object](#) that facilitates the comparison of [objects](#) and the [generalization](#) of sets of similar [objects](#) to a [class](#). Examples are the [feature representation](#), [pixel representation](#), [dissimilarities](#), [kernels](#) and [graphs](#). [page-22](#) [section 6.1](#) [paper](#)

**Object variability** A measure that expresses how much the [objects](#) in a given set differ in general.

**Observation** A single measurement or a set of measurements made from a single [object](#). [page-6](#) [page-15](#)

**One-against-one** A scheme for solving the [multi-class classification](#) problem in which [binary classifiers](#) ([discriminants](#)) are trained between all pairs of [classes](#). On a second level these [classifiers](#) are [combined](#). [page-31](#) [example](#)

**One-against-rest** A scheme for solving the [multi-class classification](#) problem in which [binary classifiers](#) ([discriminants](#)) are trained between all single [classes](#) and the remaining ones. On a second level these [classifier](#) are [combined](#). [page-31](#) [example](#)

**One-class classification** A [classification problem](#) in which a boundary around a single [class](#) has to be found. [paper](#) [paper](#) [thesis](#) [website](#) [wiki](#)

**Overfitting** [Classifiers](#) and transformations that are optimized too long or by a too sensitive procedure may adapt to the noise in the data instead of to the for [generalization](#) relevant information. Such procedures may be qualified as [overtraining](#), the result as overfitting. [wiki example](#)

**Overtraining** Originally overtraining refers to [training](#) a [neural network](#) too long, by which it adapts to the noise in the data. Now overtraining is also used for any [classifier](#) that by bad parameter settings, too small [training sets](#), too many dimensions or too many parameters to be optimized adapts to the noise. Other terms for (about) the same phenomenon are [curse of dimensionality](#), [overfitting](#), [peaking](#) and [Rao's paradox](#). [page-27](#) [page-35](#) [page-39](#) [post paper](#) [wiki example](#)

**Parametric estimation** Model estimation, e.g. a [probability density function](#), based on optimizing a parametric model, e.g. a [normal distribution](#), and not on a non-parametric function of all observations, e.g. a [kernel density estimator](#) ([Parzen](#)). See also [non-parametric estimation](#). [page-31](#) [tutorial](#) [wiki](#) [wiki](#)

**Partitional clustering** A [clustering procedure](#) which starts with an initial, possibly random, [clustering](#) with the desired number of [clusters](#). Iteratively it is optimized according to some specific criterion. Examples are k-Means, k-Centers and Expectation-Maximization (EM) [clustering](#). [page-44](#) [subsection 9.1.2](#) [paper](#) [wiki example](#)

**Parzen** [Kernel](#) based procedures for non-parametric [density estimation](#). Usually a spherical [normal distribution](#) is used as [kernel](#). Its width, also called the smoothing parameter, should be adapted to size of the [training set](#). If done properly the Parzen [classifier](#), which is based on the Parzen density estimates, is asymptotically a [Bayes classifier](#). [page-31](#) [page-33](#) [paper](#) [paper](#) [post paper](#) [tutorial example](#) [example](#)

**Pattern** The word 'pattern' is overly used for slightly different but related concepts. The main one is the subset of similar [objects](#) in a larger set (a [class](#) or a cluster). It is however also used for the

entire [similarity](#) structure in a collection of [objects](#) as well as for a single [object](#) which is typical for a set of similar [objects](#).

**Pattern recognition** This is the process or ability of finding patterns in a set of [objects](#). It also refers to the scientific domain that studies such processes as well as to the technology of creating artificial systems that can do this. The main sub-domains are [representation](#) and [generalization](#). Pattern recognition is related to but slightly different from the fields of artificial intelligence and machine learning. As pattern recognition refers to both, a human ability as well as a research domain, it may be labeled as an art as well as a science. [page-6](#) [paper](#) [tutorial](#) [wiki](#)

**PCA Principal Component Analysis** finds a orthogonal transformation for a vector space that decorrelates a specified [covariance matrix](#)  $C$ . It results in a set of eigenvectors (the new axes)  $V$  and the variances (diagonal terms) of the transformed [covariance matrix](#)  $D$ , also called eigenvalues. Thereby

$$V^{-1}CV = D$$

The orthonormal transformation is  $V^T = V^{-1}$  as the [covariance matrix](#) of the transformed space is

$$\mathbb{E}[V^T x (V^T x)^T] = V^T \mathbb{E}(xx^T)V = V^{-1}CV = D$$

It is assumed without loss of generality that  $x$  has zero mean.

In practical applications often just the eigenvectors corresponding the the largest eigenvalues are used as they stand for the main directions of variance. Other directions are then neglected under the assumption that they correspond to noise. As this is effectively a type of averaging it may improve accuracy.

PCA is related to the [Karhunen-Loeve Transform](#) KLT and to [LDA](#) or [Fisher mapping](#). [page-28](#) [page-46](#) [post](#) [tutorial](#) [wiki](#) [example](#)

**Peaking phenomenon** The phenomenon that the expected [classification performance](#) computed for a constant size of the [training set](#) as a function of the number of [features](#) (the [feature curve](#)) may show

an optimum for some small number of [features](#). See also [overtraining](#) and [curse of dimensionality](#) [post paper](#) [paper](#) [example](#)

**Perceptron** A simple [neural network](#) classifier consisting out of a single neuron. The perceptron [classifier](#) is linear and [trained](#) sequentially. It is sometimes used as a [weak classifier](#). [wiki](#) [example](#)

**Pixel representation** [Images](#) can be represented by their pixels. The unfolded 1-dimensional vector constructed from 2-dimensional or even multi- dimensional [images](#) is then used for building a [feature space](#). There are two problems with this approach: it results in very high-dimensional spaces and small shifts or rotations of the [image](#) may result into a large jump of the representing vector. A good property of the pixel representation is that all information available in the [image](#) is preserved, in contrast to [features](#) measured from the image. [page-25](#) [example](#)

**Pixel space** Sometimes used as a short for '[feature space](#) based on pixels'. Objects like [images](#) and spectra are in this way represented by a [feature vector](#) with all pixels or samples as elements. A set of 64x64 grey value [images](#) is thereby represented by a set of vectors in a 4096 dimensional space. [page-25](#) [example](#)

**Posterior probability** In [classification](#) this is the [probability](#)  $P(\omega|x)$  that a particular [object](#), given by an [observation](#)  $x$ , belongs to the class  $\omega$ . [page-32](#) [paper](#) [wiki](#)

**Prior knowledge** All knowledge of measurements, models and probabilities that may be helpful to classify a particular [object](#)  $x$ , except any measurement on  $x$  itself. [wiki](#)

**Prior probability** In [classification](#) this is the [probability](#)  $P(\omega)$  for a particular [class](#)  $\omega$  of an [object](#)  $x$  without taking into account any measurement on  $x$  itself. [page-31](#) [tutorial](#) [video](#) [example](#)

**Probability** A number between 0 and 1 which is the likeliness that a particular event will occur. It is thereby the expectation of the fraction of the number of times that this event occurs in the total

number of events under consideration. See also [probability density](#). [wiki](#)

**Probability density** This is used for describing the [probability](#) of events given by a (set of) continuous variable(s). If the probability density of an event  $x$  is  $f(x)$  then the [probability](#) that  $x$  is in the interval  $[x - \Delta, x + \Delta]$  is  $\int_{x-\Delta, x+\Delta} f(x)dx$ .  $f(x)$  is also named probability density function (PDF), probability distribution or just [density](#). See [density estimation](#) for the relation with [classifiers](#). [page-31](#) [tutorial](#) [wiki](#)

**Prototype** In general, something that is typical for a larger class. In [pattern recognition](#) it is often used for [objects](#) that are representative for a [class](#) or for a [classification problem](#). [page-32](#) [example](#)

**QDA Quadratic Discriminant Analysis** is used for the [Bayes](#) normal classifier allowing different [covariance matrices](#) for the [classes](#). [page-32](#) [tutorial](#) [wiki](#) [example](#)

**Random forest** A combined classifier based on an ensemble of [decision tree](#) trained by a random subset of the [training set](#) and a random subset of the [features](#). See also [decision forest](#). [page-34](#)

**Recognition** In the strict sense the recognition of an [object](#) is equivalent to its [classification](#). In a broader sense it includes the (development of the) [representation](#) as well of the [classifier](#). [page-4](#)

**Recognition problem** The application for which a [pattern recognition](#) system has to be designed. The problem may include not only [representation](#) and [generalization](#), but also the choice of sensors and the collection of a [design set](#). [wiki](#)

**Regularization** A general term for the various ways to avoid [overtraining](#) in optimizing a [classifier](#), e.g. adding noise to the data, adding noise to an optimization step, restrictions on the size of the step, adding a cost term to the criterion and early stopping (a limit on the number of optimization steps). [page-41](#) [video](#) [wiki](#) [example](#)

**Reject Classification** option by which no [class](#) is assigned to the input [object](#). Instead it is rejected, i.e. returned for a better [classifier](#), a human specialist or just put aside. Reasons for rejection can be that the [object](#) is for every [class](#) an outlier, or it is a borderline case. [page-30](#) [paper](#) [paper](#) [example](#) [example](#)

**Representation** The [representation](#) of an [object](#) is a description of that [object](#) in terms of measurable [observations](#) that enables the a numeric or logic comparison with other [objects](#) in the same problem. [page-8](#) [page-11](#) [page-17](#) [chapter 6](#) [post](#) [tutorial](#)

**Representation set** The representation set is a set of [objects](#) used in a [dissimilarity representation](#) to represent other [objects](#) by their dissimilarities to the [objects](#) in the representation set. [page-26](#) [paper](#)

**Resubstitution error** The [classification error](#) estimated by resubstituting the [training set](#) used for training the [classifier](#). Also [apparent error](#) or [training error](#). [page-35](#) [example](#)

**Scaling** Used to modify [feature spaces](#) in such a way that the variability of [features](#) (e.g. their variances, mean class variances or domains) become equal. [page-9](#) [page-23](#) [post](#) [wiki](#)

**Scatter matrix** The scatter is a synonym for the estimated variance or [covariance matrix](#), based on observed data points. If the set of data points has been split into subsets, the scatter of the total set is split into the [between scatter](#) and the [within scatter](#). See also [LDA](#) and [Fisher classifier](#). [wiki](#)

**Scatter plot** A plot that shows the vector locations of [object representation](#) in a 2D or 3D space, or a projection of a higher dimensional vector space. [page-8](#) [page-46](#) [subsection 9.2.1](#) [example](#)

**Segmentation** The isolation of an object to be recognized from a larger observation containing other objects and/or a disturbing background. Examples are a single character on a page with more text and images, a phoneme in a speech signal, a face in an picture from a street with more people. [page-16](#) [page-23](#) [wiki](#) [wiki](#) [wiki](#)

**Semantic gap** The jump in the processing of [observations](#), from physical phenomena, senses, nervous system to understanding. What first was just a physically measurable signal gets suddenly a meaning. [page-6](#) [wiki](#)

**Similarity** A measure expressing how much two [objects](#) are equal. Usually similarities are positive, sometimes even restricted to the  $[0,1]$  interval. See also [dissimilarity](#). [page-5](#) [scholar](#) [wiki](#)

**Soft label** Labels point from an [objects](#) to a [class](#). They are assigned by an expert or estimated during automatic [classification](#). In both cases there might be uncertainty or ambiguity. This can be solved by replacing the traditional [crisp labels](#) by soft labels, usually numbers between 0 and 1. Every [object](#) has a soft label value for every class. They don't necessarily sum to one. They might be interpreted in several ways: uncertainties, probabilities or fuzzy [class membership](#). See also [crisp label](#). [scholar](#) [wiki](#)

**Strong classifier** This is a [classifier](#) that, after training, is expected to perform significantly better than random assignment, also called a strong [learner](#). The opposite of a strong [classifier](#) is a [weak classifier](#) (weak learner). [wiki](#)

**Structural representation** The [representation](#) of an [object](#) by some structural means, e.g. (sequences of) symbols or a [graph](#). [page-22](#) [paper](#) [paper](#)

**Supervised learning** This is learning from examples under supervision of a [teacher](#). This almost always refers to a [teacher](#) who assigns desired [class labels](#) to the example [objects](#). [page-28](#) [wiki](#)

**SVM Support Vector Machine.** A linear [classifier](#) or regression function that is optimized for a distance based criterion as a linear function of a subset of the given vectors ([training objects](#)), the support vectors. Thanks to the [kernel trick](#) also non-linear [classifiers](#) can be [trained](#). The result depends on the choice of the [kernel](#). [page-33](#) [tutorial](#) [wiki](#) [example](#)

**Template matching** A [classification](#) rule in which [objects](#) are assigned to the most [similar object](#) example. Such examples are usually selected by the [teacher](#) or found automatically by a [prototype](#) selection procedure. Originally templates are used for the [recognition](#) of simple [objects](#) like characters by optical comparison. [page-32](#) [wiki](#)

**Teacher** The application expert who is able to collect examples, [label](#) them, and define a [representation](#), e.g. suggesting features or defining a [dissimilarity measure](#). [page-30](#)

**Testing** [Evaluation](#) of a system by a [test set](#), preferably selected independent from the [training set](#). See [evaluation](#) and [error estimation](#). [page-35](#) [example](#)

**Test error** The [classification error](#) as estimated by a [test set](#). [page-35](#) [example](#)

**Test set** A set of [objects](#) with given [class labels](#) used for the evaluation of a [classifier](#). A good test set is representative for the set of [objects](#) to be classified later by the [classifier](#) and is not used during [training](#). [page-12](#) [page-19](#) [page-35](#) [page-36](#) [example](#)

**Trained classifier** A [classifier](#) trained by a [training set](#), ready for [evaluation](#) by a [test set](#) or to be applied in practice. A trained [classifier](#) might be just a linear function. The way it is trained (the [untrained classifier](#)) may not be visible anymore. [page-30](#)

**Training** The optimization of a mapping or a [classifier](#) for a given set of [objects](#), the [training set](#). There are several types of training procedures, e.g. analytically, based on a single estimate of the parameter (e.g. the [Fisher classifier](#)), iteratively, by optimizing a criterion based on the entire training set (e.g. [SVM](#)), or sequentially, object by object (e.g. [perceptron](#)) [page-30](#)

**Training error** The [classification error](#) of a [classifier](#) based on the set of [objects](#) used for training the [classifier](#), its [training set](#). [example](#)

**Training set** The set of [objects](#) used for optimizing a mapping or a [classifier](#). [page-10](#) [page-12](#) [page-19](#) [page-30](#) [page-35](#) [example](#)

**True classification error** The [classification error](#) as made by the [classifier](#) on the [objects](#) of the problem for which it was [trained](#). This error can be estimated by a [test set](#) randomly sampled from these [objects](#). [page-35](#)

**tSNE** [t-distributed Stochastic Neighbor Embedding](#). It is a low-dimensional [embedding](#) of [objects](#) such that similar [objects](#) are modeled by nearby points and dissimilar [objects](#) are modeled by distant points with a high emphasis on the first. Usually [objects](#) are embedded in a 2D subspace and the purpose is [visualization](#) by a [scatter plot](#). [page-47](#) [wiki example](#)

**Unseen objects** This expression refers to [objects](#) in the [design set](#) that are different from all [objects](#) in the [training set](#).

**Unsupervised learning** This is learning from examples without the [supervision of a teacher](#). This almost always refers to a learning from [objects](#) for which no [class labels](#) have been given. Examples are [cluster analysis](#) and [PCA](#). [page-43](#) [page-28](#) [wiki](#)

**Untrained classifier** A [classifier](#) for which the functional form and a criterion are given but for which the parameters have not yet been specified. They need to be optimized by the use of a [training set](#).

**Validation** Synonym for [evaluation](#) and for [testing](#). [page-19](#)

**Validation set** A subset of the [design set](#), usually different from the [training set](#), used to optimize some [training step](#). [page-19](#)

**Vector representation** An [object representation](#) by a vector space. This is the most popular [representation](#) as there are many tools to analyze sets of vectors. [page-24](#) [section 6.2](#)

**Visualization** Various tools are used to demonstrate in some graphical way the data, the processing and the results of [pattern recognition](#) problems or the properties of the algorithms to tackle them: [scatter plots](#), [graphs](#), [decision trees](#), [dendrograms](#), [learning curves](#) and [feature curves](#). [page-46](#) [section 9.2](#)

**Weak classifier** This is a [classifier](#) that, after training, is expected to perform just slightly better than random assignment, also called a weak [learner](#). Usually such [classifiers](#) can be [trained](#) fast and store just a few parameters. They are thereby popular as a [base classifiers](#) in [combined classifiers](#) like [adaboost](#). The opposite of a weak [classifier](#) is a [strong classifier](#) ([strong learner](#)). [page-34 wiki example](#)

**Within scatter** The estimated variance or [covariance](#) of a set of data points. There is formally no difference with just the [scatter](#). By within scatter, however, the difference with [between scatter](#) is emphasized as it restrict itself to the [scatter](#) of a single set, while the [between scatter](#) expresses the [scatter](#) between sets. [page-28 paper wiki](#)

**Wrapper** This is an approach in [feature selection](#) in which as criterion the performance of the target [classifier](#) is used. For each [feature](#) set to be [evaluated](#) this performance has to be estimated, usually by [cross-validation](#). This makes the procedure computationally intensive and may result in [overtraining](#), especially if the performance is estimated by data that is also used in designing the final [classifier](#). If this danger exists, e.g. when the original [feature](#) size is large, the so-called [filter](#) approach may be preferred, which is based on the use of a more simple criterion. [page-29 paper wiki example](#)