Pattern Recognition
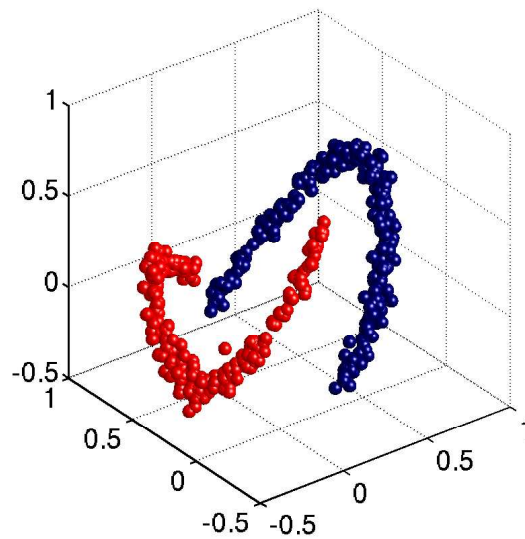
IN4085 laboratory course manual

2017

David M.J. Tax, Marco Loog

Contents

Pattern Recognition

Course contents and goals

This course will provide you with a practical introduction to data analysis and pattern recognition. The techniques are discussed at a level such that you will be able to apply them in your research. The emphasis is on using the computer as a tool for pattern recognition. Starting from the basis for any pattern recognition application, measurements, the topics discussed will be:

- classification;
- representations;
- evaluation;
- feature selection and extraction;
- clustering.

After you have succesfully completed this course, you should:

- understand pattern recognition theory to such an extent that he/she is able to read recent literature on the topic in engineering-oriented journals (e.g. IEEE Tr. on PAMI);
- know which statistical methods to apply to which problems, on which assumptions they are based and how these methods interrelate;
- be able to construct a learning system to solve a given simple problem, using existing software.

Laboratory work

In addition to the Monday lectures, there is an obligatory practical component in which you practice the construction of a patern recognition system. This runs in parallel with the lectures and it given on Wednesday or Thursday. The laboratory work starts with exercises related to the lecture of the week (as given in this manual).

Material

Next to the document you are now reading, the book "Pattern Recognition" by Sergios Theodoridis and Konstantinos Koutroumbas will be used (4th edition, 2009, Academic Press, ISBN 978-1-59749-272-0.).

Although not necessary, it will be useful for you to have access to a computer running MATLAB outside the practical course. The toolbox and data sets used during the course can be downloaded from the course Blackboard site, so that you can experiment and work on your final exercise at home.

Prior knowledge

Basic working knowledge of multivariate statistics and linear algebra is required to follow the course. The basic elements of statistics will be reiterated in the second week. After the first week you are asked to do the self-test in Appendix B and submit your answers. If you feel you are deficient in either statistics or linear algebra, please have a good look at Appendices A and B of the book "Pattern Recognition".

A,B

Examination

The examination of this course will consist of three parts:

1. A number of papers (6-7) will be discussed in the lectures. In preparation, these paper will have to be read, and five non-trivial questions about each paper should be handed in on a separate A4 (with your name and student number) at the beginning of the lecture. These questions are graded as "good" or "insufficient" (20% of the final grade).

2. Lab work is mandatory. As a final assignment, you should construct a working pattern recognition system to solve a given problem (during the lab hours) and write a report on it (40% of the grade, see below).

3. A written exam on pattern recognition theory (40% of the grade).

Final assignment

The final assignment for the practical course is based on a case, in which your group plays the role of a pattern recognition consultancy company (see Appendix A). Your company is faced with an assignment by a client company working on automatic bank cheque processing applications. This client wants you to research the possibility of using pattern recognition techniques to interpret bank account numbers and the monetary amount. To this end, they have supplied you with a standard dataset of handwritten digits ("0", ..., "9") put together by the US National Institute of Standards & Technology, NIST. The digits have already been segmented (isolated) for you, and are stored as individual images. The problem you will work on is how to recognize the digits, given these images. The deliverable consists of a report detailing and motivating the design choices you made, as well as a test of your system on a set of benchmark data withheld from you by the client.

To prepare for this final assignment and practise with the tools you will need to solve this problem, the last few exercises for each of the first 9 weeks will consist of some small experiments on the handwritten digit dataset. During the last few weeks you can then work on the final assignment itself. In the exercises each week, some possible solutions will be presented.

These are definitely not the best solutions, so your final system cannot consist of just the solutions to the exercises each week!

The basic software package used is PRTOOLS. It is introduced in appendix D. It contains many routines to analyze patterns in object measurements given by a vector representation. Preprocessing is needed to convert raw data like images and time signals into the proper representation. PRTOOLS includes a set of routines that enable such preprocessing for image data. In addition, private code may be used. It is important to store routines that automatically generate (intermediate) results, as this will enable to modify and regenerate later datasets. It may also be useful to recompute results for larger sets of objects.

Depending on your programming skills, these last exercises each week may take some time. You are advised to take care that they are at most one to two hours behind schedule.

This manual

This document consists of several parts:

1. exercises for the practical course (Chapters 1-12);

2. the final assignment (Appendix A);

3. a self-test on probability theory (Appendix B);

4. brief introductions to MATLAB, to PRTOOLS, a pattern recognition toolbox used throughout the course; and to `datafiles` for the construction and treatment of general measurements (Appendices C-E).

In the first part of the manual, for each week a number of exercises is given. Please read the text carefully and do the exercises. If you have the idea things are still unclear, ask one of the assistants.

Notation

Most weeks contain a few optional exercises, indicated like this:

_____ OPTIONAL _____

An exercise between these lines is optional. This means that you are not required to do it. It can give you some extra background and tips. Only work on it if you think the subject is interesting and if you have sufficient time left.

_____ END OPTIONAL _____

Some other notational conventions are:

- MATLAB variables and code will be indicated using the `teletype font` (for example: `x`, `mean(x)`). For larger pieces of MATLAB code, we will use the notation:

```
>> % A piece of test Matlab code
>> x = rand(10,2);
>> mean(x)
>> std(x)
```

  Here, `>>` is the Matlab prompt. If pieces of code are not preceded by `>>` it will mean it's wiser to write a script. You can learn more about MATLAB scripts in appendix C.

- An alert sign like the one in the margin here indicates it is essential you read the text next to it carefully.

X, Slides

- A book sign indicates where you can read more on the theory behind the subject discussed. Numbers indicate chapters and sections in "Statistical Pattern Recognition". "Slides" means the theory is discussed in the slides, which can be downloaded in handout format from the Blackboard site.

It might happen that in an exercise you will encounter a MATLAB-command you have never seen before. If this happens, have a look at the `help`. In almost all cases there is an explanation of the command, so you can get an idea of what the function does.

Week 1

Introduction

Objectives When you have done the exercises for this week, you

- should be able to mathematically derive decision boundaries given simple probability density functions,
- should be able to perform simple computations with Bayes' rule,
- should be familiar with working under MATLAB,
- understand some PRTOOLS commands,
- know what an object and a dataset are,
- should be able to construct and investigate and visualize some simple datasets.

1.1 Decision Theory

Exercise 1.1 (a) Assume that we managed to represent objects from a two-class classification problem by a single feature. We know that the objects from class ω_1 have a Gaussian distribution with $\mu_1 = 0$ and $\sigma_1^2 = 1/2$, and the objects from class ω_2 have a Gaussian distribution with $\mu_2 = 1$ and $\sigma_2^2 = 1/2$. Derive the position of the decision boundary when both class priors are equal.

2.2

(b) Again, assume we have a two-class classification problem in a 1D feature space, but now assume that objects from class ω_1 have a uniform distribution between 0 and 1, and objects from class ω_2 have a uniform distribution between 2 and 3. Where is the decision boundary now?

(c) And where is the decision boundary when the objects from class ω_2 have a uniform distribution between 0.5 and 1.5? (The distribution of ω_1 did not change, classes have equal prior.)

(d) And where is the decision boundary when the objects from class ω_2 have a uniform distribution between 0.5 and 2.5? (The distribution of ω_1 did not change, classes have equal prior.)

Exercise 1.2 (a) Assume we represent the objects in a two-class classification problem by a single feature. We know that the objects from class ω_1 have a Gaussian distribution

with $\mu_1 = 0$ and $\sigma_1^2 = 1/2$, and the objects from class ω_2 have a Gaussian distribution with $\mu_2 = 1$ and $\sigma_2^2 = 1/2$. Derive the position of the decision boundary when both class priors are equal, but we have a loss matrix of:

$$L = \begin{bmatrix} 0 & 0.5 \\ 1.0 & 0 \end{bmatrix}. \tag{1.1}$$

(b) Assume again we have a two-class classification problem in a 1D feature space, but now assume that objects from class ω_1 have a uniform distribution between 0 and 1, and objects from class ω_2 have a uniform distribution between 0.5 and 2.5. Given the loss matrix (1.1), where is the decision boundary now?
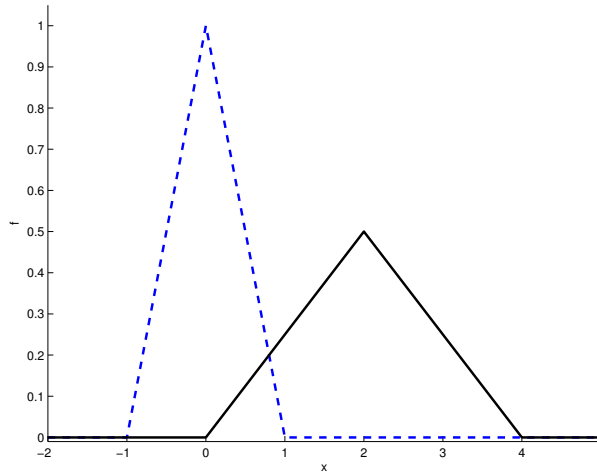
Figure 1.1: The class-conditional probabilities of two classes $p(x|\omega_1)$ (dashed blue line) and $p(x|\omega_2)$ (solid black line) in a 1-dimensional feature space.

Exercise 1.3 In figure 1.1 two triangular-shaped class conditional probability density functions are given. The first one $p(x|\omega_1)$ is indicated by a dashed blue line and the second $p(x|\omega_2)$ with a solid black line. The class priors are assumed equal here.

(a) Again, use the Bayes' rule to derive the class posterior probabilities of the following objects: $x = 3$, $x = -0.5$, $x = +0.5$. To which class are the objects therefore assigned?

(b) Which object is on the decision boundary of the Bayes classifier?

Exercise 1.4 Now assume that class ω_1 in figure 1.1 is twice as small as class ω_2. That means $p(\omega_1) = 1/3$ and $p(\omega_2) = 2/3$.

(a) Compute the posterior probabilities for $x = 3$, $x = -0.5$, $x = 0.5$.

(b) Where is the decision boundary of the Bayes classifier now?

Exercise 1.5 Compute the Bayes error for the class distributions given in figure 1.1, where the classes have equal prior.

1.2 Creating artificial and real datasets

The general approach to attack a pattern recognition problem (or a data analysis problem) is presented in section 1.2 in the book. One key entity in data analysis is the idea of an *object*. We always assume we can represent any object by a set of values, often just measurements. Whatever object we are considering, in order to perform operations on an object by a computer, we have to encode this object by some numbers. An object in real life and in the computer are therefore two different things. Furthermore, in this course we will use the convention that each object is represented by a row vector of measurements (thus an $1 \times d$ array).

When you want to conclude something from data, it is often not from *one* object, but from a *set* of objects. We assume we have a set of objects from which we want to obtain some knowledge. This set is called a dataset. A dataset is a set of n objects and is stored in a $n \times d$ array.

After you have specified the problem you want to solve, you have to collect examples and do measurements on these examples. For that, you have to define what features are likely to be informative for the problem you specified.

Exercise 1.6 (a) Make a fake dataset in MATLAB containing 10 objects with 3 measurements each. Invent the measurement values yourself. Below is an example piece of code which fills the matrix x with 2 objects, each containing 3 measurements:

```
>> x = [ 0.7 0.3 0.2;  2.1 4.5 0];
```

Make your own matrix x.

(b) Compute the means (using `mean`) and standard deviations (`std`) of your 3 measurements of 10 objects. What is the difference between `mean(x)` and `mean(x')`?

Exercise 1.7 (a) When a dataset contains just two features per object, it can be visualized in a scatterplot (we come back to this later). Make a scatterplot by:

```
plot(x(:,1),x(:,2),'b*');
```

Looking at the data matrix you created in Exercise 1.6, find out which object is plotted where in the plot.

(b) When you look at the scatterplot, can you identify outlier objects, or structure in the data?

When the dataset is used to train classifiers, it is also required that for each object a class label is present. This indicates from which class the object originates, according to the expert (i.e. you).

Exercise 1.8 (a) Invent labels for the objects that you defined in the previous question, and store it in a Matlab variable `lab`. The labels can be numbers, like 1 or 2, but you can also use character strings, like 'banana' or 'orange'. Create a PRTOOLS dataset by

```
>> a = prdataset(x,lab)
```

Check if the resulting dataset has the correct number of objects, the correct number of features and correct number of classes (correct here means: what you expect). You can do that by just typing the variable on the Matlab command line:

```
>> a
```

1.2.1   Scatterplot

A scatterplot is the most simple plot you can make: it simply plots the first measurement against the second measurement. If you have three measurements, you can use 3D plots; if you have even more, you will have to select at most three of them by hand (although later we will discuss ways of visualising more measurements at once).

Exercise 1.9 Load the dataset "boomerangs" (use the function `boomerangs` and choose the number of objects to generate).

(a) Use `scatterd(a)` to make a scatterplot of the first two features, and `scatterd(a(:,[2 3]));` for the features 2 and 3.

(b) Use `scatterd(a,3)` for a 3D scatterplot. Find and use the `rotate3d` button to view all data.

(c) Use `scatterd(a,'legend')` to add a legend with the class labels to the scatterplot.

1.2.2   The definition of an object

Up to now, each object was characterised by a vector of measurements. Sometimes the idea of an object is somewhat vague and the user has to decide where an object "starts" and "stops". Imagine, for instance, that we want to build a speaker recogniser. This recogniser should identify the speaker from a recorded speech signal.

Exercise 1.10 (a) Assume we have recorded several sentences of several speakers using a microphone. Now we have a set of very long time signals, say five seconds, recorded at a sampling rate of about 11 kHz. How many samples does the signal contain?

(b) What would you define as the objects, and what as the measurements by which each object is characterised (this is probably not simple!)?

1.2.3   Real image data

Pattern recognition studies automatic ways to find patterns in data. Very often these are images, but other sensor signals like time series (e.g. speech, music, weather) or spectra are equally possible. In addition also non-sensor data like text, web-pages, answers to question-naires and medical tests are often analyzed by pattern recognition tools. In the first example we will deal with simple black-and-white images ('blobs' or 'silhouettes') as given by the so-called Kimia dataset (collected by Benjamin B. Kimia).

```
>> obj = kimia_images      % load 12 objects for 18 classes
>> show(obj,12)
```

This displays 18 classes of objects, each represented by 12 examples. A single image, e.g. object 100 can be displayed by

```
>> figure; show(obj(100,:))
```

The `show` command belongs the the PRTOOLS toolbox. There many more commands in MATLAB to display a single image, e.g. `imshow`, `image` and `imagesc`. Sometimes they demand a change of the color table (`colormap`) or even a change of the intensity scale of the images.

In order to distinguish the classes in the Kimia database in an automatic way, a numeric representation is needed. A first try is to measure the sizes and the perimeters of the blobs. The following commands do this for the entire database using tools from the Matlab Image Processing toolbox (in particular the function `regionprops`):

```
>> x = im_features(obj,obj,{'Area','Perimeter','Eccentricity'});
>> +x
```

The `im_features` commands define the measurements (it is intended to measure object properties in gray value images for which the shape and position are given by a binary (black-and-white) image, therefore the `obj` parameter is supplied twice as we have just binary images). The function `im_features` accepts a `prdatafile`, and it outputs a `prdataset`. This dataset can be shown on the screen by `+x`. A better way for the visualization of 2-dimensional data (i.e. data describing objects by just two numbers) is a scatter plot.

Exercise 1.11 (a) Take the feature set `x` from the above piece of code and select just two classes, e.g. the elephants (class 3) and the camels (class 16) by `featset = seldat(x,[3 16])`. You may inspect the values by `featset = double(featset)`.

(b) In order to understand these numbers you may select the original images by

```
>> a = kimia_images
>> b = seldat(a,[3 16]) % select classes 3 and 16
>> show(b)
```

A single image, e.g. the first camel, can be inspected by

```
>> c = data2im(b,13);    % select image 13
>> figure; imagesc(c);   % show it on the screen
>> colormap gray         % apply gray colormap
```

In this figure the axes show the sizes in pixels. Can you understand the area and perimeter values of `featset`?

(c) Inspect the data in a scatterplot by `scatterd(x,'legend')`. You will now see the first two features. Note the class names in the legend and the feature labels along the axis.

1.3   Measuring objects within PRTools

The PRTOOLS package, that will be used extensively in this course, is based on vector representations of objects derived from object measurements. In order to have a unique toolbox, it has been recently extended (in version 4.1) with possibilities to perform such measurements on large sets raw object measurements. This extension is the so called `prdatafile` object of

PRTOOLS. An introduction to PRTOOLS is presented in D. The `prdatafile` extension is described in E. In the following some introductory examples are presented.

In order to work with PRTOOLS it should be available in the MATLAB search path. This can be verified by `help prtools`. Various datasets can only be loaded if the `prdatasets` or `prdatafiles` directories are in the path. Use `help prdatasets` and `help prdatafiles` to inspect the list of available datasets.

This example is again based on the Kimia Dataset. It consists of a set of 216 black-and-white images, silhouettes of toy objects and animals. There are 18 classes of such objects and for each are 12 images given. In order to speed up processing here we will restrict ourselves to just 6 classes.

Exercise 1.12 Enter the following commands:
```
>> delfigs                     % delete all figures
>> obj1 = kimia_images         % load 12 objects for 18 classes
>> obj1 = seldat(obj1,[3:3:18]);  % select the classes 3,6,9,12,15 and 18
>> figure(1); show(obj1);      % show raw data
>> obj2 = im_box(obj1,0,1);    % put objects in a square box
                               %   (aspect ratio = 1)
>> figure(2); show(obj2);      %   to create square images
>> obj3 = im_rotate(obj2);     % rotate images to same orientation
>> figure(3); show(obj3);      % show data
>> obj4 = im_resize(obj3,[20 20]);% resize images to 20 x 20 pixels
>> figure(4); show(obj4);      % show data
>> obj5 = im_box(obj4,1,0);    % create empty box around objects
>> figure(5); show(obj5);      % show data
>> showfigs                    % show all figures on the screen
```

The raw data in figure(1) shows the objects in bounding boxes of different sizes, to save memory space. In figure(2) the aspects ratios are set equal to one, i.e. for each image as many empty (black) rows or columns are added to make the image square. In figure(3) the images are rotated, and in figure(4) resampled to a 20×20 pixel grid. Some objects may touch the border of their image, which may cause trouble in further processing, so finally empty rows and columns are added to all four sides (figure(5)).

Create a file with the above commands. If you want to better inspect what happens, reduce `obj1` further by a command like `obj1 = obj1(1:12:37,:)`, which selects a single object out of the first 4 classes. Try to understand each of the processing steps.

(a) Find out what the `im_rotate` command does exactly. Why would this be useful/necessary for this set of objects?

(b) Play a little with the image size, i.e. try 10×10 or 30×30 pixels. What is optimal?

The data referred to by the variables `obj1, obj2, ...`, etcetera, are of the `datafile` data

type. The contents can be inspected by
```
>> struct(obj1)

ans =

      files: {{1x19 cell}  {1x19 cell}}
   rootpath: 'C:/Matlab/prdatafiles/kimia'
       type: 'raw'
    preproc: [1x1 struct]
   postproc: [0x0 mapping]
   prdataset: [72x17907 prdataset]
```

Objects of the type `prdatafile` do not store data directly, but just refer to files on disk and the way they should be processed. In (in `rootpath`) the name is stored of the top directory which contains the data. The two cells in the field `files` contain the names of all sub-directories and of all files they contain. Such fields may be better inspected by, e.g. `s = getfiles(obj2); s{1}, s{2}{:}`. All preprocessing is stored in the `preproc` field, e.g. `p = getpreproc(obj6), p(4), p(5)`. (The selection of classes made for `obj2` is stored somewhere else. We will not discuss it here).

It should be emphasized that commands like `obj5 = im_resize(obj4,[20 20])` do not cause any processing of data. They just store the command in the description of the variable. Execution is postponed until data is needed (e.g. for display by the `show` command) or when a `datafile` is converted to a `dataset`. This will be described later.

All preprocessing can be combined in the following way:
```
>> preproc = im_box([],0,1)*im_rotate*im_resize([],[20 20])*im_box([],1,0);
>> obj5 = obj1 * preproc;
```

The ∗-symbol can be interpreted as the "pipe" command in Unix; `preproc` is thereby roughly a small routine that consists of a sequence of operations. When it is applied to `obj1`, PRTOOLS substitutes the `[]` entries by the data before the ∗-symbol.

Exercise 1.13 We will now measure two properties of the set of objects, the x- and y-coordinates of the centers of gravity:
```
>> obj = seldat(obj1,2)        % Use just class 2
>> obj = obj*preproc           % normalize
>> figure(1); show(obj);       % show data
>> mean_set = im_mean(obj)     % measure (x,y) of centers of gravity
>> mean_data = double(mean_set) % convert them (unpack)
```
(a) Do the results correspond to what you expect, given the object images?

The last command, `double(mean_set)`, converts the `datafile` variable `mean_set` to plain numbers (doubles). So in `mean_data` we now have the measurements of the objects in class 2. These can be further analyzed, e.g. by constructing histograms and scatterplots.

(b) Plot the histograms of the (x, y) coordinates of the centers of gravities ("means") of the objects in class 2.

17

(c) Construct a scatterplot of these two measurements.

(d) Also measure the means of the objects in class 13.

(e) Add these to the scatterplot with a different color.

This week you saw the basis of pattern recognition: object definition, data collection, and Bayes' rule. Starting from good measurement data, the rest of the analysis (visualisation, clustering, classification, regression) becomes much easier. Starting from poorly defined objects or poorly sampled datasets with insufficient example objects, your analysis becomes very hard and no clear conclusions will be possible (except that more data is needed).

TAKE-HOME MESSAGES

- **If you collect or receive data, think well about what the objects and measurements are**

- **Visualization is a very important method for giving you a first idea of what your data looks like**

- **At some point, you will have to convert your measurements to identical features for each object (with different values)**

- **Use Matlab's and PRTools' range of visualisation tools to get an idea of the distribution of your data**

Week 2

Classification with Normal Densities

Objectives When you have done the exercises and read the text for this week, you should

- understand how a normal distribution describes multivariate data,
- understand how to visualize and interpret 1D and 2D normal distributions,
- know how to derive decision boundaries and to visualize them for 2D problems,
- know the maximum likelihood estimates for normal models,
- understand the general ML and MAP solution to estimation.

2.1 Normal Probability Density Function

2.1.1 Shapes in 2D

Exercise 2.1 Have a look at Figures 2.3 to 2.6 in the book.

2.4.1

(a) Are the correlations that can be observed in these figures positive, negative, or zero?

(b) What would it mean for the correlation if the ellipsoidal configuration in Figure 2.6 runs from the bottom left to the top right corner instead?

Exercise 2.2 (a) Make a rough sketch of the shape of the ellipsoids of a normal density for which the covariance matrix has the form $\begin{bmatrix} 1 & 0 \\ 0 & 9 \end{bmatrix}$. Take this ellipsoid, what is the ratio of the length of its long axis in comparison with its short axis?

(b) Imagine we rotate the above sketch of ellipsoids clockwise such that their orientation becomes diagonal. Qualitatively, what happens to the variances and the covariances for this new normal distributions? How do they compare to the original variances and covariances as given in the covariance matrix above?

(c) Draw a figure visualizing a 2D normal density in which the two variables/features are completely correlated, i.e., the correlation coefficient between the two variables is 1. Give the covariance matrix that matches your sketch.

Exercise 2.3 (a) Any ideas on how to visualize a 3D normal density?

2.1.2 Parameter Estimation

2.4.1

Exercise 2.4 Consider Equation (2.27) in the book, which provides the general expression for a normal distribution in an l-dimensional space. Here we assume that $l = 2$.

Now assume, in addition, that the covariance matrix Σ is given and equals $\begin{bmatrix} 4 & 1 \\ 1 & 1 \end{bmatrix}$. Derive the maximum likelihood estimate for the 2D mean vector μ.

2.5.2

Exercise 2.5 Consider the MAP solution for the mean, $\hat{\mu}_{MAP}$, in Example 2.5 of the book.

(a) Consider what happens to the estimate for the following limits: $\sigma_\mu^2 \to \infty$, $\sigma_\mu^2 \downarrow 0$, $\sigma^2 \downarrow 0$, and $N \to \infty$.

(b) What is the role of the prior over the mean? What happens with the estimate when there are no observations? What happens to the ML estimate in this case?

2.1.3 Some Manual Labor

Exercise 2.6 Given the following 3D observations: $\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$, $\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$, $\begin{bmatrix} -1 \\ 1 \\ 1 \end{bmatrix}$, $\begin{bmatrix} 2 \\ \pi \\ 1 \end{bmatrix}$

(a) Determine maximum likelihood estimates for the mean, the variances, and the covariance matrix using this 3D data. (Provide exact solutions! No numerical approximate solutions.) Determine the correlation matrix as well. (If you don't know how these maximum likelihood estimates look like or how you calculate them, then make sure you look this up! And yes, Wikipedia is probably fine...)

(b) Redo the exercise in (a) but now only for the observations made in the first and third dimension (or for the first and second feature), i.e., determine the mean, variances, and covariance matrix for this 2D data. (Provide exact solutions! No numerical approximate solutions.)

(c) How do the estimates for the 2D and 3D data relate?

We apply the linear transformation $\begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}$ to the 2D data considered in (b).

(d) How do the new feature vectors look/what are the new coordinates?

(e) Precisely what effect does this transformation have on the maximum likelihood solutions for the mean etc.? What happens to the correlation between the two features?

20

Exercise 2.7 This may be a difficult exercise but it is a rather important one. . .

Again consider Equation (2.27) that provides the general expression for a normal distribution in an l-dimensional space. Consider an $l \times 1$ vector a and a $l \times l$ linear transformation matrix L. Assume we transform our random variables X, which are distributed according to Equation (2.27), into new random variables Y by means of an affine transformation as follows:

$$Y = LX + a.$$

(a) Show that Y is still normally distributed. Express the new mean and covariance matrix in terms of μ, Σ, L, and a.

(b) Check your answer with the outcome in exercise 2.6 (d) and (e).

2.1.4 Some Programming Labor

Exercise 2.8 Generate 1000 random point from a 2D standard normal distribution using `randn`. (Note the discrepancy between the math, in which typically column vectors are employed, and computational software, in which feature vectors often are row vectors. This means, in this case, that the matrix with random numbers should be size 1000×2.)

(a) Turn these random points into a `prdataset a` and `scatterd` the data.

The command `w = gaussm(a)` determines estimates for a normal distribution based on the data set `a`, and stores the estimated parameters in the mapping `w`.

(b) Use `plotm(w)` to visualize the 2D density estimates on top of the scattered data.

(c) Multiply one of the features by 3 and, subsequently, rotate the data over an angle α (say $\alpha = \pi/3$) by means of a rotation matrix (you know. . . $\left[\begin{smallmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{smallmatrix}\right]$. . .). Scatter the new data in a new figure and plot the new contours. Compare to the other figure and see where corresponding points end up.

(d) Investigate for which angle the correlation becomes maximum.

———————————————————— OPTIONAL ————————————————————

Exercise 2.9 (a) Consider the same questions as those in the previous exercise but now use uniformly distributed data, i.e., use `rand` to generate the initial data.

———————————————————— END OPTIONAL ————————————————————

Exercise 2.10 The piece of code below generates 5 data points from a standard normal distribution, estimates its parameters, and displays the estimated densities through a contour plot. This is repeated a number of times and all contour are plotted in the same figure.

(a) Before executing the code. By now, can you predict what the various contour plots will look like? Can you explain why this happens?

```
for i = 1:8
   a = prdataset(randn(5,2));
   scatterd([-3 -3;3 3],'.w') % trick to set the figure axes about right
   plotm(gaussm(a))
   hold on
   pause
end
hold off
```

(b) Execute the code. Pressing the space bar will add another contour plot. Check with yourself wether the plot comes close to your expectations. (Don't fool yourself! Was this indeed what you expected?)

(c) What happens when the number of random feature vectors increases steadily? What if we could perform this exercise with an infinite amount of random data points?

2.2 Bayesian Decisions based on Normal Distributions

Exercise 2.11 Consider two 2D normal distributions with different means but equal covariance matrices. Assume the latter to be equal to a multiple of the identity matrix, i.e., take Σ to be $\begin{bmatrix} c & 0 \\ 0 & c \end{bmatrix}$ for some $c > 0$. The class priors are not necessarily equal.

(a) What shapes can the decision boundary take on? Demonstrate this mathematically starting from the expressions in Section 2.4.2 in the book.

2.4.2

Exercise 2.12 Consider two 2D normal distributions with different means but equal covariance matrices.

(a) What shape does the optimal decision boundary have?

Generate 10 data points per class from a data set that fulfills the above assumptions. Create a data set `a` with these 20 points and these two classes. We can estimate linear discriminants by means of `ldc(a)` and quadratic discriminant boundaries by means of `qdc(a)`.

(b) Given the shape of the optimal decision boundary, how does the boundary look for the trained normal density based quadratic discriminant?

The PRTools command `plotc` plots the decision boundary.

(c) Scatter the data set `a` and plot the decision boundary estimated by means of `ldc(a)` and `qdc(a)`. Can you explain their difference? You might want to revisit the previous question (b).

(d) What happens when the number of points per class increases? What happens in the limit of an infinite number of points?

Exercise 2.13 Generate a 2D data set with 2 classes with 10 samples from a uniform distribution (using `rand`) and 10 samples from a normal distribution (using `randn`), respectively.

(a) Train a `ldc` and a `qdc` on this data, scatter the data, and plot the two decision boundaries (with `plotc`). Determine the how many points are on the wrong side of the boundary.

(b) With `w` a trained classifier and `a` a data set, the labels that the classifier assigns to the data points in `a` can be retrieved using `labeld`: `a*w*labeld`. Check your count from the previous question using this command.

Exercise 2.14 Given two normally distributed classes in 1D. One with mean 0, variance 4, and class prior $\frac{9}{10}$ and one class with mean 1, variance 1, and class prior $\frac{1}{10}$. Determine the decision boundary for this classification problem.

—————————— OPTIONAL ——————————

Exercise 2.15 In case one is dealing with a classification problem in which the classes are multivariate Gaussian, the quadratic classifier is in principle the classifier that will always perform optimally as long as there is enough data to accurately estimate the various parameters like means and covariance matrices. When the class distributions are not necessarily normal, this does not hold true and there are cases in which the normality based linear discriminant performs a better discrimination that the normality based quadratic discriminant.

(a) Construct an artificial 2D problem for which the decision boundary of `ldc(a)` is better than the decision boundary obtained by `qdc(a)`.

(b) Take two uniform rectangular classes. One with bottom left coordinate (-1,-2) and top right coordinate (0,3), the other with bottom left (0,0) and top right (1,1). This should do the 'trick'.

—————————— END OPTIONAL ——————————

Exercise 2.16 (a) Show that the classification performance does not change for `ldc` and `qdc` when the data set it linearly (or even affinely) transformed. If this is too complicated, at least show that scaling the axes does not change the decision on the data.

(b) Construct a simple 1D or 2D example that shows the classifier might essentially change when a nonlinear transformation of the space is performed.

- The decision boundary derived from normal distributions takes on a quadratic form

- Linearly (or affinely) transforming normally distributed data leads to data that is again normally distributed

- LDA and QDA are invariant to scaling or, generally, to affine transformations of the data

- Parameters are estimated form a finite amount of data and will therefore vary

- The new means and (co)variances of some data after an affine transform can be determined using some simple rules

Week 3

Nonparametric Classification

Objectives When you have done the exercises for this week, you

- understand what a Parzen density estimator, and a Parzen classifier is,
- understand what a k-nearest neighbor density estimator, and classifier is,
- know what classifiers are sensitive to rescaling of one of the features.

3.1 Non-parametric density estimation

3.1.1 Histograms

The simplest way to estimate the density of one measurement is by plotting a *histogram*. It is easiest to do this for one measurement (MATLAB has a command `hist`), but you can create a histogram plot of 2 measurements at the same time.

2.5.6

The main problem in using histograms is choosing the right bin size. Making the bins too broad will hide useful information within single bins; making them too small will result in bins containing too few samples to be meaningful.

Exercise 3.1 (a) Generate 10 1D Gaussian datasets and calculate the average histogram and the standard deviation:
```
for i = 1:10
  a = gauss(n,0);
  h(i,:) = hist(+a,-5:5);
end;
errorbar (-5:5, mean(h), std(h));
```
with different numbers of samples `n` (e.g. 10, 100 and 1000).

(b) For what `n` do you think the histogram starts giving a good impression of the true density?

(c) Repeat the above for data obtained from a Laplacian distribution (which is more peaked than the Gaussian); use `a = laplace(n,1)` to generate `n` samples. Does the same hold? Why?

Note how you used 2 dimensions to visualise the 1D histogram. Similarly, we can create 2D histograms using 3 dimensions. In practice, visualising 3D data or a function of 2D data is often as far as we can go with computers.

Exercise 3.2 Generate a 2D Gaussian dataset and use the `hist2` function provided:
```
a = gauss(100,[0 0]);
hist2(a);
```

Note that `hist2` is not a standard MATLAB function, so it is a bit less flexible than the standard `hist` function. To have a look around the histogram, try `help rotate3d`.

(a) For the standard number of bins (10×10), how many samples would you say are needed to get a good idea of the shape of the histogram? Try this by generating various numbers of samples and inspecting whether the histogram changes if you repeat the experiment.

(b) You can also play with the number of bins; see `help hist2`. For 5×5 bins, how many samples do you need? Is the representation still accurate? And for 20×20 bins?

3.1.2 Parzen density estimation

Now we are going to estimate the density using a Parzen density estimator, called `parzenm` in PRTools.

2.5.6

Exercise 3.3 (a) We start with creating a simple dataset with:
```
>> a = gendats([20 20],1,8);
```

(Type `help gendats` to understand what type of data we have now.)

(b) We define the width parameter h for the Parzen kernel:
```
>> h = 0.5;
```

(c) The function `parzenm` estimates a density for a given dataset. In most cases a PRTOOLS `prdataset` is labeled, and these labels are used in the function `parzenm` to estimate a density for each class. To define a Parzen density estimator with a certain width parameter `h` on the entire dataset, ignoring labels, type:
```
>> w = parzenm(+a,h);
```

This mapping can now be plotted along with the data:
```
>> scatterd(+a); plotm(w,1);
```

If your graphs look a little "bumpy", you can increase the grid size PRTOOLS uses for plotting:
```
>> gridsize(100);
```

and try the above again.
(d) Plot the Parzen density estimate for different values of `h`. What is the best value of `h`?

26

When you want to evaluate a fit of a density model to some data, you have to define an error. One possibility is to use the log-likelihood, defined as:

$$\mathrm{LL}(\boldsymbol{X}) = \log\left(\prod_i \hat{p}(\boldsymbol{x}_i)\right) = \sum_i \log\left(\hat{p}(\boldsymbol{x}_i)\right) \tag{3.1}$$

The better the data $\boldsymbol{x}$ fits in the probability density model $\hat{p}$, the higher the values of $\hat{p}(\boldsymbol{x})$ will be. This will result in a high value of $\sum_i \log\left(\hat{p}(\boldsymbol{x}_i)\right)$. When we have different probability density estimates $\hat{p}$, we have to use the one which has the highest value of LL.

Note that when we fill in different values for the width parameters h in the Parzen density estimation, we have different estimates $\hat{p}$. Using the log-likelihood as a criterion, we can optimize the value of this free parameter h to maximise LL.

To get an honest estimate of the log-likelihood, we have to evaluate the log-likelihood (3.1) on a *test set*. That means that we have to make (or measure) new data from the same distribution as where the training data came from. When we would evaluate the log-likelihood on the data on which the probability density was fitted, we would get a too optimistic estimation of the error. We might conclude that we have fitted the data very well, while actually a new dataset from the same distribution does not fit in the density at all! Therefore, if you want to evaluate the performance of an estimated $\hat{p}$, use an independent test set!

Exercise 3.4 Use the data from the same distribution as in the previous exercise to train a Parzen density estimator for different values of h. Compute the log-likelihood of this training set given the estimated densities (for different h):

```
a = gendats([20 20],1,8);                     % Generate data
hs = [0.01 0.05 0.1 0.25 0.5 1 1.5 2 3 4 5]; % Array of h's to try
for i = 1:length(hs)                          % For each h...
  w = parzenm(+a,hs(i));                      %   estimate Parzen density
  LL(i) = sum(log(+(a*w)));                   %   calculate log-likelihood
end;
plot(hs,LL);                                  % Plot log-likelihood as function of h
```

(since w is the estimated density mapping w, the estimated density $\hat{p}$ for objects in a dataset a is given by +(a*w)).

(a) What is the optimal value for h, i.e. the maximal likelihood? Is this also the best density estimate for the dataset?

Exercise 3.5 (a) Use the same data as in the previous exercise, but now split the data into a training and test set of equal size. Estimate a Parzen density on the training set and compute the Parzen density for the test set. Compute the log-likelihood on both the training and test sets for $h = [0.1, 0.25, 0.5, 1, 1.5, 2, 3, 4, 5]$. Plot these log-likelihood vs.

h curves:

```
a = gendats([20 20],1,8);        % Generate data
[trn,tst] = gendat(a,0.5);       % Split into trn and tst, both 50%
hs = [0.1 0.25 0.5 1 1.5 2 3 4 5]; % Array of h's to try
for i = 1:length(hs)             % For each h...
  w = parzenm(+trn,hs(i));       %   estimate Parzen density on trn
  Ltrn(i) = sum(log(+(trn*w)));  %   calculate trn log-likelihood
  Ltst(i) = sum(log(+(tst*w)));  %   calculate tst log-likelihood
end;
plot(hs,Ltrn,'b-'); hold on;   % Plot trn log-likelihood as function of h
plot(hs,Ltst,'r-');            % Plot tst log-likelihood as function of h
```

What is a good choice of *h*?

Exercise 3.6 Use the same procedure as in Exercise 1. Change the number of training objects from 20 per class to 100 per class. What is now the best value of *h*?

3.1.3 Nearest neighbour density estimation

3.3 Estimation of the density by the nearest neighbour method is harder, but PRTOOLS contains an implementation called `knnm`.

Exercise 3.7 (a) Again generate a dataset `a = gendats([20 20],1,8)` and apply the `knnm` density estimator for $k = 1$: `w = knnm(a,k)`. Plot the density mapping using `scatterd(a); plotm(w,1)`. What did you expect to see for $k = 1$? Why don't you see it?

(b) Increase the grid size, e.g. `gridsize(500)`, and plot again.

(c) Try different values of k instead of 1. Judging visually, which do you prefer?

(d) How could you, in theory, optimise k?

Exercise 3.8 Which do you prefer: the Parzen density estimate or the nearest neighbour density estimate?

Although PRTOOLS contains a nearest neighbour density estimator, for illustration purposes we can also make one ourselves. The simplest to construct is the 1-nearest neighbour.

Exercise 3.9 (a) Generate 1D data as you did in the previous exercise, but only 5 objects per class: `a = gendats([5 5],1,8)`. Also generate a grid of data, as follows:

```
>> gx = [-3:0.2:11]';
```

28

(b) Compute the distances from all the grid points to the training set. You can use the `distm` function from PRTOOLS:

`D = sqrt(distm(gx,a));`

(use the `help` to find out why you need to take a `sqrt`). Check the size of matrix `D`. What does a row and a column in this dataset mean?

(c) To find the nearest object in the training set for each of the grid objects, you have to find the minimum over a row (or a column, depends on how you defined your matrix `D`). To find the minimum we first have to order them. To order them along the columns do `sort(+D,1)` and along the rows do `sort(+D,2)`. You will get a sorted distance vector `Dsort` for each of the columns or rows.

(d) To compute the density for each of the grid objects, you have to use the general formula:

$$\hat{p}(\boldsymbol{x}) = \frac{k}{nV(\boldsymbol{x})} \tag{3.2}$$

where $V(\boldsymbol{x})$ is the volume of a hypersphere around $\boldsymbol{x}$ with as radius the *distance* to the nearest neighbour ($k = 1$) and n is the total number of objects in the dataset. The volume of the hypersphere In the 1D dataset this is just two times this distance, so:

`phat = 1./(n*2*Dsort(:,1));`

(e) Plot this density estimate (`scatterd(+a); hold on; plot(gx,phat);`) and save the figure, `saveas(gcf,'figknn','m')`. Are you satisfied with this plot? (Notice the scale of the y-axis! Restrict the axis with the `axis` command).

Exercise 3.10 (a) How can you generalize the previous procedure to use the kth nearest neighbour instead of the first nearest neighbour? Test it by inspecting the density plot.
(b) How would you calculate an optimal value for k?

Exercise 3.11 How should you generalize the procedure in Exercise 3.9 to work in a p-dimensional feature space?

3.1.4 The Parzen and kNN Classifier

In the sections above, two approaches to estimating a density are shown. The first is using the Parzen window approach, the second is using the nearest neighbor approach. When one of these approaches is used to estimate the class conditional probabilities, Bayes' rule can be used to construct a classifier. In the table below, the corresponding PRTOOLS mapping names are listed.

PRTOOLS name	description
parzenc	Parzen density classifier
knnc	k-nearest neighbor classifier

Exercise 3.12 Generate a dataset `a` using the function `gendatb`, and make a scatterplot of the dataset. Next, train a Parzen classifier and a k-nearest neighbor classifier, and

plot the decision boundary using `plotc` (obviously, to train the classifiers, you have to choose a value for `h` and for `k`):

```
>> w = parzenc(a,h);
>> v = knnc(a,k);
>> plotc(w); plotc(v,'r');
```

(a) Try different values for `h` and `k`. What are good values?

(b) How would you make an automatic procedure to optimize the values of `h` and `k` for a classification problem?

3.1.5   Naive Bayes Classifier

In PRTOOLS the naive Bayes classifier is called `naivebc`. In the implementation of PRTOOLS, for each of the features it estimates the class conditional probabilities using a histogram with N bins.

Exercise 3.13 Compare the decision boundary of the `naivebc` with that one of the Parzen classifier and the k-nearest neighbor. Look, for instance, at the datasets `gendats`, `gendatb` and `gendatd`.

2.8
problems

Exercise 3.14 Make exercise 2.39 and 2.41 from the book.

3.2   The scaling problem

In this last section we have a quick look at the scaling problem. It appears that some classifiers are sensitive to the scaling of features. That means, that when one of the features is rescaled to very small or very large values, the classifier will change dramatically. It can even mean that the classifier is not capable of finding a good solution. Here we will try to find out, which classifiers are sensitive to scaling, and which are not.

Exercise 3.15 (a) Generate a simple 2D dataset (for instance, using `gendatb`) and plot the decision boundaries of the six classifiers listed above. Use $k = 1$ for the `knnc`.

(b) Make a new dataset in which the second feature is 10 times larger than the dataset above. Do this by

```
newtrain = a;
newtrain(:,2) = 10*newtrain(:,2)
```

Train six classifiers `nmc,ldc,qdc,fisherc,parzenc,knnc` and plot the decision boundaries.

(c) Which classifiers are affected by this rescaling of the feature space? Why are they affected, and others not?

(d) Is it an advantage or a disadvantage?

- k-NN density estimation followed by the rule of Bayes, results in the k-NN classifier. Similarly, applying Bayes' rule on the Parzen density estimation results in the Parzen classifier,

- Some classifiers are sensitive to the scale of the individual features. Rescaling of features can result in much better or worse classification performances,

Week 4

Linear Classifiers

Objectives When you have done the exercises for this week, you

- know what a percepron classifier does,
- understand the least squares, or Fisher classifier,
- know what the bias-variance dilemma in a classification setting means.

4.1 The perceptron

Exercise 4.1 Make exercise 3.4 from the book.

Exercise 4.2 (a) Generate a simple, linearly separable, dataset `gendats([20,20],2,6)`. Make a scatterplot and check that the dataset is linearly separable. If it is not, generate a new dataset until it is.

Extract from this dataset the feature matrix $\mathbf{X}$ and the label vector (hint: use `getnlab`).

(b) Implement the perceptron algorithm, given on pg. 94 of the book. Don't forget to add the constant term to the features (as explained at the start of section 3.3 from the book)!

(c) Train the perceptron on the data that you generated and plot the normal vector $\mathbf{w}$ and the decision boundary (i.e. all points where $\mathbf{w}^T\mathbf{x} = 0$) in the figure. Does the weight vector make sense: does it separate the classes?

(d) What happens when the classes in the dataset are *not* linearly separable? Check what solution your implementation gives.

4.2 Least squares

Exercise 4.3 Check that equation (3.28) from the book is equal to equation (3.26).

Exercise 4.4 Implement the Least Squares solution, equation (3.45), in Matlab. Test it on the same dataset that you have used for the perceptron (exercise 4.2), by plotting the decision boundary in a scatterplot.

Exercise 4.5 The Least Squares solution is implemented in PRTOOLS as well, and it is called the Fisher classifier, `fisherc`. Train a Fisher classifier on the same data as you used before, and compare the two decision boundaries. Are they equal?

4.3   Bias-variance dilemma

The bias-variance dilemma indicates that when a model is fitted on a finite (small) dataset, there should be made a trade-off between the bias and the variance. The bias is the difference between the average model output, and the 'true' expected output. The variance is the average difference between the specific output of a model, and the average model output. Or, loosely speaking, the bias measures if the model is flexible enough to fit the truth, and the variance measures how stable the estimate is. In this section we try to identify this in some classifiers.

Exercise 4.6 Generate a small Bananaset `a = gendatb([10,10])`, train a Fisher classifier on this, and plot the decision boundary. Now, resample new datasets from the Banana-distribution and train new Fisher classifiers.

Identify which parts of the Banana-distribution are always incorrectly classified by all classifiers. Does this contribute to the bias or the variance?

Identify in which parts of the distribution the classifiers often disagree. Does this contribute to the bias or the variance?

Exercise 4.7 Repeat the previous exercise, but instead of using a Fisher classifier, use a Parzen classifier. How does the variance and the bias change, when you switch from the Fisher classifier to the Parzen classifier?

TAKE-HOME MESSAGES

- **There are several ways of optimizing a linear classifier,**

- **When you have a limited dataset, you need to find a good trade-off between the bias and the variance.**

Week 5

Linear Classifiers, Part 2

Objectives When you have done the exercises and read the text for this week, you should

- be able to derive the expression for the posteriors in the case of logistic discrimination,
- have an idea of the basic shape of the logistic posterior,
- be able to describe to underlying idea of the linear SVM.

5.1 Logistic Regression, Logistic Discrimination, Log-linear Classifier, `loglc`

Exercise 5.1 (a) Consider Equation (3.61) in the book with $M = 2$. Derive an expression for the posterior $p(\omega_2|x)$ (which does not depend on the posterior of the other class ω_1). Section 3.6

(b) Sketch the general shape of $p(\omega_2|x)$ under the logistic model in 1D. Understand its asymptotic behavior and how its form depend on its parameters. Also try to understand how the shape of this function looks in 2D, for example.

Given samples from a two-class problem in 1D with feature values -1 and 1 in the one class and 2 and 3 in the other. Consider the objective function in Equation (3.69) in the book (again with $M = 2$). Section 3.6

(c) Describe the solutions, in terms of formulæ or qualitatively, that maximize Equation (3.69).

(d) Why is the solution from (c) not unique?

5.2 The Support Vector Machine, `svc`

Exercise 5.2 Consider the following 2D two-class data set. Class one contains two points: $\begin{bmatrix} 0 \\ 1 \end{bmatrix}$ and $\begin{bmatrix} 0 \\ 3 \end{bmatrix}$. Class two has a single data point: $\begin{bmatrix} 2 \\ 0 \end{bmatrix}$.

(a) Determine the classifier that maximizes the margin on this classification problem, using a graphical/geometrical reasoning (probably you cannot do the minimization of the support vector error function by hand). How many support vector are obtained.

Shift the first point above, $\left[\begin{smallmatrix} 0 \\ 1 \end{smallmatrix}\right]$, to $\left[\begin{smallmatrix} 0 \\ -1 \end{smallmatrix}\right]$.

(b) How does the new maximum margin classifier look? What happened to the number of support vectors?

Exercise 5.3 (a) Demonstrate, possibly graphically/geometrically, that the support vector classifier is sensitive to feature scaling. Hint: this can be done in 2D based on a training set of size 3 (like in (a) and (b)) and a single test point.

Exercise 5.4 (a) Reconsider the 2D configurations from 5.2 (a)–(b) above and compare the solution of the Fisher classifier to those obtained by means of an SVM. In what cases do they differ? Do you see the pattern?

—————————————————— OPTIONAL ——————————————————

Exercise 5.5 Make exercise 3.17 in the book.

Section 3.8

—————————————————— END OPTIONAL ——————————————————

5.2.1 Linear Classifiers: A First Showdown?

Exercise 5.6 (For this exercise you might want to set up a script...)

Generate a 30-dimensional training set `trn` with 50 samples using `gendats` (read `help gendats`!). Generate, in addition, a test data set `tst` with 1000 objects.

(a) Train the following classifiers: `nmc`, `ldc`, `fisherc`, `loglc`, and `svc`, and test these with the test set. Hint: `testc` can be used to test trained classifiers on a test set (again: read the help!) and with braces one can train (and test) many classifiers in one go. For instance, `trn*{qdc,knnc([],1)}` can be used to train a quadratic discriminant and a 1NN on `trn` simultaneously.

(b) Repeat the previous steps of training and testing a couple of times with newly generated training sets. Overall, which seems the best performing classifier? Can you explain this?

(c) Reduce the training set size to 30 objects and redo the experiment. Does you conclusion change? What happens to `ldc` when the number of samples is so small relative to the dimensionality?

(d) Redo (a) to (c) for samples from `gendatd` instead of `gendats`. Any remarkable changes? What is the main difference between the NMC in the case of a sample size of 50 and 30?

(e) Can you construct a classification problem in which a linear SVM would clearly outperforms a linear discriminant classifier? How about the Fisher classifier v.s. SVM?

36

- **The posteriors of a logistic discriminant have a classical, so-called sigmoidal shape**

- **Generally, the C parameter can critically influence the decision boundary of an SVM**

- **Where, for instance, the linear discriminant classifier is scaling invariant, for SVM feature scaling matters**

- **Different linear classifiers perform best under different circumstances, notably, their relative performance is typically training sample size dependent**

Week 6

Nonlinearity: Neural Networks, Support Vector Machines, and Kernelization

Objectives When you have done the exercises and read the text for this week, you should

- be able to give the link between a kernel function and its (implicit) feature mapping
- be able to kernelize simple, inherently linear classifiers
- have basic understanding of the nonlinearities neural networks can create
- understand the separation potential of linear classifiers in high-dimensional spaces.

6.1 Neural Networks

Exercise 6.1 Consider a 1D data set with two samples in class A $x_1 = 0$, $x_2 = 3$ and one in class B $x_3 = 2$.

(a) Construct a neural network with one hidden layer, using neurons that have a sigmoid activation function, that can perfectly separate class A and B. How many hidden units does it need?

(b) What if one allows an arbitrary activation functions for every setting? With how few neurons can one solve this task consisting of three labeled samples?

Exercise 6.2 The function `bpxnc` implements a simple multilayer neural network, which can be trained by backpropagation. For instance, `w = bpxnc(a,2,1000)` trains on data set `a` a network with a single hidden layer with two nodes.

Investigate for very(!) small networks and very(!) small samples sizes from two classes (XOR?) how such networks behave. (You better type `prwaitbar off`.) Repeat, for example, the same line of code that trains some neural network and plot the different classifiers obtained in the same `scatterd` of the data.

Exercise 6.3 Consider in the book. Consider a sample in all of the seven different regions and label these samples A or B according to the labels in the book, except for the sample Figure 4.8 in area 110, which should also be labeled A (instead of B). So, we have four samples from A and three from B.

(a) Can this problem be solved perfectly with a multilayer *perceptron* (i.e., using hard thresholds as activation functions and not smooth sigmoids) with one (1) hidden layer with three (3) neurons? How can you see this?

(b) How many regions can this network classify correctly at most?

(c) How many different optima does this multilayer perceptron have for which at most one region is being misclassified? That is, in how many different ways can the network reach the maximum number of correctly classified regions as found in b? Explain your answer.

Exercise 6.4 Design a `bpxnc` that works well (and fairly stable?) on the data set `a = gendatb`.

Exercise 6.5 Some arbitrary geometric insight?

(a) In how many parts/polygons can N linear decision boundaries chop up a feature space when its dimensionality is equal to or greater than N?

(b) Consider a two-class problem. How many linear decision boundaries does one typically need to classify N arbitrarily scattered and labeled points in N-dimensional space?

6.2 The Nonlinear Support Vector Machine, `svc`

Exercise 6.6 (a) Assume we have two objects, represented by 1-dimensional feature vectors x and χ. Find a feature mapping ϕ that leads to the inner product $\exp(-(x - \chi)^2)$. Hints: expand the term $-(x - \chi)^2$ and write $\exp(2x\chi)$ as a series based on the Taylor series of the exponential.

(b) What is the dimensionality of the space that ϕ maps a 1-dimensional feature vector to?

Exercise 6.7 Let's kernelize `nmc`.

(a) Express the distance to any class mean in terms of regular inner products between the test point x and, say, the N_C samples x_i^C from class C.

(b) Kernelize the nearest mean classifier by mean of the Gaussian kernel, $K(x, \chi) = \exp(-\frac{\|x-\chi\|^2}{2\sigma^2})$. Can you show that this boils down to something like a Parzen classifier? You may limit yourself to the two-class case.

Exercise 6.8 The function `svc` can be used to both construct linear and nonlinear support vector machines. Check the help to see how kernels different from the linear one can be employed. Type `help proxm` to get an overview of the possible kernels and their parameters.

(a) On `a = gendatb([20 20])`, train an `svc` with a `'radial_basis'` kernel, i.e., the Gaussian kernel, for kernel widths that vary from fairly small (0.1?) to fairly large (10?). Check with a large (enough) independent banana test set how the performance varies for the different choices of kernel widths.

(b) How does the kernel width of `parzenc` relate to the width of the radial basis function?

(c) Why can the `svc`, potentially, perform much faster at test time than the Parzen classifier?

───────────────── OPTIONAL ─────────────────

Exercise 6.9 Consider the linear least squares problem as in of the book. We aim to kernelize its solution mapping $\hat{w}^T x$.

Subsection
3.4.1

    (a) Express the minimizing solution $\hat{w}$ explicitly in terms of the training samples.

    (b) Using the well-known identity $(XX^T)^+ X = X(X^T X)^+$, where $^+$ indicates the Moore-Penrose pseudoinverse, rewrite $\hat{w}^T x$ in terms of inner products between training instances x_i and the test point x.

Exercise 6.10 The next experiment might ask too much time or too much memory.

Construct a data set with a sharp corner in it. Take, for instance `X = rand(1000,2);` and `a = dataset(X,X(:,1)>.5&X(:,2)>.5)`. Study how closely a support vector machine follows the border, and especially the corner, for different kernels and different settings of the penalty term C. Definitely try the different Gaussian kernels with different sigma parameters in combination with various Cs. (If possible, you might want to consider using `libsvc` instead of `svc`.)

───────────────── END OPTIONAL ─────────────────

┌───┐
│   │
│   TAKE-HOME MESSAGES   │
│   │
│ • **The design and training of multilayer neural networks can be a challenge** │
│   │
│ • **The C parameter can critically influence the decision boundary of an SVM, also in the nonlinear case** │
│   │
│ • **Neural nets might get stuck in local optima or, generally, bad solutions** │
│   │
│ • **Classifiers like NMC and the Fisher linear discriminant can be kernelized** │
│   │
│ • **The Gaussian kernel enables one to work implicitly (obviously) in an infinite dimensional space.** │
│   │
└───┘

Week 7

Decision Trees, Combining, and Boosting

Objectives When you have done the exercises and read the text for this week, you should

- understand that pruning might be crucial for decision tree performances
- one can "overprune" and "underprune"
- understand how bagging can potentially improve the performance of a single "instable" classifier
- understand the difference between fixed and trainee combiners

N.B.! The time has come that, during the computer exercises, you better not only spend time on the exercises for the current week but also on the final assignment. You can find the latter in Appendix A.

7.1   Decision Trees

Exercise 7.1 (a) Train a decision tree, using `treec`, with splitting criterion `infcrit` on a `gendats` data set with 50 samples in every class. Vary the degree of pruning up to, say, 12 (`help treec`!) and check how the classification error varies with it on a large enough test data set (i.e., to get a good impression of the true performance of the classifier). (In addition, you might have to average over different training sets to get a good impression of the expected variability of the error rate for different degrees of pruning.)

What can be observed?

(b) Can you explain the previous behavior? In what way do you think the classification boundaries differ when comparing a high degree of pruning with a low degree? You might want to check this using `scatterd` etc. (Note that you might have to put `gridsize` to, say, 1000 or so, and replot the plot.)

Exercise 7.2 (a) Redo the experiments from the previous exercise but now with `gendatd`. What changes in results do you note? Can you explain these?

(b) What if the two class means move closer to each other in the first feature dimension? What kind of change in behavior with varying degrees of pruning do you expect (apart from an increase of the error rate)?

<div align="center">——— OPTIONAL ———</div>

Exercise 7.3 (a) This is an open problem, I think... Redo the experiment in Exercise 7.1 but now with a training set size of 10 per class (so 20 in total). Explain the error rate behavior with varying pruning degrees.

Exercise 7.4 (a) (Another open problem?) Any idea what the splitting criterion `maxcrit` does?

<div align="center">——— END OPTIONAL ———</div>

7.2 Combining

Exercise 7.5 We are going to combine classifiers trained on different representations of hand written digits. We compare these to the individual classifiers and the classifier trained on the data set having all features.

First we construct the various data sets:
`a1 = mfeat_kar, [b1,c1,I,J] = gendat(a1,0.3);`
`a2 = mfeat_zer, b2 = a2(I,:); c2 = a2(J,:);`
`a3 = mfeat_mor, b3 = a3(I,:); c3 = a3(J,:);`
The `kar`, `zer`, and `mor` indicate the type of features we are considering.

(a) Train three nearest mean classifiers on the different data sets:
`w1 = nmc(b1)*classc; w2 = nmc(b2)*classc; w3 = nmc(b3)*classc;`
(The `classc` is a trick to make sure that classifiers output posteriors.) Estimate the three corresponding error rates on the corresponding `c`-data sets, e.g., for the first data set: compute `c1*w1*testc`.

(b) Train and test the same classifier on the data sets `b = [b1 b2 b3]` and `c = [c1 c2 c3]`, which contain all features from the three data sets above. Compare its error rate to those from (a).

Given the trained individual classifiers `w1`, `w2`, and `w3`, a fixed combiner can be constructed as follows: `v = [w1; w2; w3]*meanc`. This is the mean combiner. Other combiners that PRTools allows are `minc`, `maxc`, `prodc`, `medianc`, and `votec` of which it should be clear what kind of combining rule these implement.

(c) Test some combiners, by means of `[c1 c2 c3]*v*testc`, and compare the results to the four earlier error rates you estimated.

(d) Looking back at the exercise, do you understand why the special form of the `gendat` command, in which the `I` and `J` occur, had to be used? What would we have been doing if we would just have generated the `b` and `c` data sets by applying `gendat` to the three `a` data sets individually? Obviously, you may want to check `help gendat` for an initial clue and to see what `I` and `J` contain.

<div align="center">44</div>

Exercise 7.6 We consider the classical use of bagging for which we reconsider exercise 7.1 (a).

(a) Basically, redo the experiments from 7.1 (a) but just compare the unpruned tree with its bagged version. For the latter you need something like `baggingc(trn,treec([],'infcrit',0))`; check `help baggingc` for more on bagging.

(b) You may want to check that even when using `gendatd` instead of `gendats` some improvement can be obtained.

(c) What does the "stabilization" of the decision tree by means of bagging effectively do? You could check a plot of the classification boundary of the single decision tree with a bagged version to get an idea.

Exercise 7.7 (a) Generate a data set by means of `gendath` and train two differently behaving linear classifiers on it. Take, for instance, `w1` equal to `nmc` and `w2` to `ldc`. Generate posterior probabilities from these by computing `p1 = a*w1*classc` and `p2 = a*w2*classc` and combine them in one data set by `p = [p1 p2]`.

How many features does this new data set have and why is this so?

(b) Scatter plot the first and third dimension of `p`. What do these dimensions correspond to? Try to understand that the original classifiers correspond to horizontal and vertical lines at 0.5 in this same scatter plot.

(c) Straight lines not parallel to any of the two axes actually combine the posteriors from the two classifiers. Are there combiners possible that indeed seem improve upon the two individual classifiers?

Exercise 7.8 As simply as fixed combiners can be constructed, one can construct trained combiners. An example is `v = [nmc*classc ldc*classc]*fisherc`, which takes the two classifier outputs of `nmc` and `ldc` and combines them through training a `fisherc` on top of it. All is simultaneously trained by executing `w3 = a*v` on a data set `a`.

(a) Take the data set from 7.7—`gendath`, construct a training and a test set, and compare `nmc`, `ldc`, and the trained combiner.

7.3   An AdaBoosting Exercise

Exercise 7.9 The AdaBoost combining scheme, `adaboostc` in PRTools, enables one to experiment with different AdaBoost schemes based on different base classifiers. The classical one is the so-called decision stump (`stumpc`, check the help).

(a) Generate a standard banana data set `a = gentdatb` and visualize the decision boundaries that are obtained by means of boosting when you combine 1, 10, 100, and 1000 base decision stumps, respectively. N.B. You might want to use the following form to suppress annoying messages, while `adaboostc` is training: `adaboostc(a,stumpc,number_of_base_classifiers,[],0)`. The final `0` silences the verbose.

45

(b) When combining 1000 base classifiers, AdaBoost seem to suffer from what is called overtraining. How can you maybe see this in the scatter plot? How could you probably see this from the error rates on an independent test set?

(c) Repeat the experiment but replace the base classifier with `nmc` and with `fisherc`. Are there noteworthy differences between the various solutions obtained? Also compare results between different base classifiers.

(d) What can we expect AdaBoost to do when we employ it in combination with, say, a 1NN or a unpruned decision tree? Could one expect any improvements from such scheme?

<div align="center">

TAKE-HOME MESSAGES

</div>

- **Pruning and bagging can improve the performance of decision trees**

- **Both fixed and trained combiners can outperform individual classifiers in particular settings**

- **AdaBoost can improve simple classifiers but can also overtrain**

- **. . .**

- **Don't forget to work on the final assignment: now and in the weeks to come.**

Week 8

Classifier evaluation

Objectives When you have done the exercises for this week, you

- can explain what sources of performance variability there are,
- understand the difference between train and test error,
- know what a learning curve is,
- explain what crossvalidation is.

8.1 Sources of Variation

Exercise 8.1 In this exercise we investigate the difference in behavior of the error on the training and the test set. Generate a large test set and study the variations in the classification error based on repeatedly generated training sets:

```
>> t = gendath([500 500]);
>> a = gendath([20 20]); t*ldc(a)*testc
```

Repeat the last line e.g. 30 times.

(a) What causes the variation in the error?

Now do the same for different test sets:

```
>> a = gendath([20 20]);
>> w = ldc(a);
>> t = gendath([500 500]); t*w*testc
```

Repeat the last line e.g. 30 times.

(b) Again explain what causes the variance observed in the results.

8.2 Learning Curves

The function `cleval` allows you to calculate so-called learning curves. These are curves that plot classification errors against the number of points in the training data set. (Check `help cleval` for the details.)

In this section we are going to study some learning curves for different data sets and different classifiers. Some of the plots and number that will be generated demonstrate certain salient and/or noteworthy behavior.

After reading a question, and before just implementing the things to be implemented, try to think about the outcome to be expected. Eventually, say at the end of this course, you should not be surprised by many of these things anymore!

8.2.1 Staring at Learning Curves

Exercise 8.2 Generate Highleyman classes (`gendath`) with a 1000 samples per class. Enlarge the feature dimensionality of this set by adding 60 dimensions of class independent randomness, i.e., plain noise. Use `cleval` to generate learning curves for `nmc`, `ldc`, and `qdc` using 64, 128, 256, and 512 objects in the training set (make sure that you repeat often enough. . .). Use `plote` to plot these curves in a single figure (check the help).

(a) Can you explain the overall behavior of these curves?

(b) Explain why the curves intersect. Which classifier performs best?

(c) What do you expect the limiting behavior of learning curves is? That is, if we were able to train on more and more data?

———————————————————————— OPTIONAL ————————————————————————

Exercise 8.3 Redo the previous experiments but substitute `gendath` with `gendats` and `gendatd`. Don't forget to add the 60 noise features.

(a) What kind of typical differences do you see when comparing behaviors for different data sets? Explain these differences?

———————————————————————— END OPTIONAL ————————————————————————

Exercise 8.4 (a) Take your favorite data set and study a learning curve for the first nearest neighbor classier (1NN). What can be expected of the learning curve of the apparent error?

8.2.2 Dipping

Exercise 8.5 Study the learning curve for `nmc` in combination with `gendatc` for very, very, very small training set sizes.

(a) What seems to be happening? And is this expected?

———————————————————————— OPTIONAL ————————————————————————

8.2.3   I Challenge You

Exercise 8.6 This more an challenge than an exercise. You might consider skipping it and move on to the next exercise. Then again, you also might take on this challenge! (Anyway, do keep track of time somewhat.)

(a) Create a two-class problem in which `ldc`, on average, outperforms `qdc` for large sample sizes.

—————————————— END OPTIONAL ——————————————

8.2.4   Feature Curves

Like learning curves that typically plot the classification error against the number of training set samples, making feature curves can also be informative. The latter studies how the classification error varies with varying number of feature dimensionality. We can use the `clevalf` command to perform such study.

Exercise 8.7 Load the data set `mfeat_kar` and make a feature curve for 4, 8, 16, 32, and 64 features using 50% of the data for training based on `qdc`.

(a) When redoing this experiment, would you expect the same curve? Why? Or why not?

(b) What, generally, happens to the learning curve when 40% or 80% of the data is used for training? Can you explain this?

8.3   Cross-Validation

The `prcrossval` function allows you to perform a cross-validation on a given data set using a particular classifier (or even a whole bunch of them in one go).

Exercise 8.8 Generate a small data set using `gendatb`, say, with 10 objects per class.

(a) Using n-fold cross-validation, make plots for the error rates for kNN and 1NN over different values of n. Also calculate the standard deviation of the error estimate, e.g., by performing the cross-validation 10 time (check the help).

(b) What do you notice about the estimated error rates? What is the general trend (maybe you should redo the data generation and the cross-validation a couple of times).

(c) What happen to the variance of the estimates for varying n? Again, we are interested in the general trend.

(d) How would the observations change if one would repeat the experiments with much larger dataset? Would they change?

Exercise 8.9 (a) Let us look back at exercise 1 where a Parzen density is estimated on some data, and use the same data in this exercise. Now let PRTOOLS find the optimal h using leave-one-out cross-validation. This can simply be performed by not specifying h, i.e. calling `w = parzenm(+a);`. To find out what value of h the function actually uses, you can call `h = parzenml(a);`. Does the h found correspond to the one you found to be optimal above?

8.4   The Confusion Matrix

A confusion matrix is of the size $C \times C$, where C is the number of classes. An element $C(i, j)$ encodes the confusion between the classes i and j. More precisely, $C(i, j)$ gives the number of objects that are from i, but are labeled as j. Nothing confusing about it.

Exercise 8.10 To create a confusion matrix you may use something like the following code.

```
>> lab = getlab(test_set);
>> w = fisherc(train_set);
>> lab2 = test_set*w*labeld;
>> confmat(lab,lab2)
```

An alternative is:

```
>> confmat(test_set*w)
```

(a) What is stored in `lab` and what does `lab2` contain?

Exercise 8.11 Load the digits data sets `mfeat_zer` and `mfeat_kar`. Split the data up in a training and a test set (using `gendat`).

(a) What are the error rates on the test sets?

(b) Use `confmat` to study the error distributions more closely. Where do the larger parts of the errors stem from?

(c) Can you explain which of the two feature sets is insensitive to image rotations?

Exercise 8.12 Given a confusion matrix $\left(\begin{smallmatrix} a & b \\ c & d \end{smallmatrix} \right)$ for a two-class problem.

(a) Give an estimate of the expected cost (per object) when misclassifying class 1 as class 2 is twice as expensive as the other way around and assuming that a correct classification incurs no cost.

8.5   Reject Curves

The error rate can be improved simply by *not* classifying certain objects, typically the ones that are near the decision boundary and in that sense difficult to classify reliably. Studying this so-called reject option is facilitated by the functions `reject` and `rejectc`.

Given a classification result, e.g. `d = tst*ldc(trn)`, the classification error is found by `e = testc(d)`. The number of columns in `d` equals the number of classes and an object is assigned

to the class for which the value in its corresponding row is the largest. Basically, the row contains the posterior probabilities (or something equivalent) and therefore will typically sum up to one.

By rejection of objects a threshold is used to determine when this largest value is not sufficiently large. The routine `e = reject(d)` determines the classification error and the reject rate for a set of such threshold values. The errors and reject frequencies are stored in `e`.

Exercise 8.13 Make two data sets both with 1000 samples using `gendatc`. One for training, one for testing.

(a) Train three classifiers `fisherc`, `qdc`, and `knnc([],1)` on the training set and plot the reject curve of the test set.

(b) Explain what you see. Explain the differences between the curves.

(c) What can we maybe conclude from the shape of the curve belonging to `fisherc`? Can you explain why this happens?

(d) Determine the reject rate for which `qdc` makes an error of (about) 5% and compute by `rejectc` the rejecting classifier for this error rate (check the help :-)

(e) Plot the classifier and the rejecting classifier in a scatter-plot. Explain what you see.

8.6   Receiver Operating Characteristic

The receiver operating characteristic (ROC), which is too often wrongly referred to as the receiver-operator curve or the like, is a tool to study the effect of changing decision boundaries in the two-class case due to, for instance, changing misclassification costs or class priors.

Exercise 8.14 Generate two moderately sized Highleyman data sets. Train an `nmc`, an `ldc`, and a 3-`nn` on this data. Using `roc` plot the three ROCs.

(a) Which of the three classifiers performs the best?

(b) If one prefers a very small error on class 1, which classifier would be the chosen one?

- **Cross-validation enables one to compare classifiers even when there is only a single data set available that contains few objects.**

- **In order to further ones understanding of the behavior of particular classifiers, learning curves, both for training and test set, should be studied.**

- **Learning curves also allow one to get more understanding of ones data, e.g. to decide if obtaining more data, possibly in combination with another classifier, might be successful in lowering the error.**

- **A confusion matrix can determine more precisely than a simple error rate where classification errors may stem from.**

- **Errors can be reduced via a reject option. Obviously, rejected objects should be taken care of separately, possibly by a human observer.**

- **ROCs give yet another view of a problem at hand. They allow one to study classification behavior under varying priors and costs.**

Week 9

Clustering

This week, we will discuss the problem of clustering; this practical session is intended to familiarise you with the different techniques that were discussed, more specifically hierarchical clustering, the K-means algorithm and the mixtures-of-Gaussians.

Objectives for this week:

- to learn to use hierarchical clustering on several datasets;
- to learn about the limitations of hierarchical clustering;
- to explore the K-means algorithm by applying it to a variety of datasets;
- to get familiar with mixture-of-Gaussian clustering.

9.1 Hierarchical clustering

Earlier in this course, you learned how to select features, i.e. you learned which *variables* "belong together". This week we will shift the emphasis to determining which *objects* belong together in groups or clusters. This clustering process enables us to extract structure from the data, and to exploit this structure for various purposes such as building a classifier or creating a taxonomy of objects.

The most difficult part of clustering is to determine whether there is *truly* any structure present in the data and if so, what this structure is. To this end, next week we will employ cluster validity measures to estimate the quality of the clustering we have obtained.

In the lectures we discussed hierarchical clustering at length. There are basically two choices that need to be made in hierarchical clustering in order to construct a dendrogram:

1. the dissimilarity measure;

2. the type of linkage.

In this course, we will only employ the Euclidean distance between two samples as a measure of dissimilarity. As you will recall from the lecture, there are three types of linkage: complete, single and average linkage. Once the dendrogram has been constructed, we need to cut the

dendrogram at an appropriate level to obtain a clustering. Simple interactive routines are available to create, visualise and cut the dendrograms.

Exercise 9.1 Start with the `hall` dataset, an artificial dataset with clear structure.

(a) Load the dataset (`load hall`). Use `scatterd` to visualise it. How many clusters are visible in the plot?

(b) What is the most suitable clustering?

An interactive clustering function is provided; it is called `interactclust`. Look at the `help` to familiarise yourself with the inputs. This function performs hierarchical clustering, draws a dendrogram in MATLAB Figure 1 and then waits for user input. The input is provided by positioning the cross-hairs in Figure 1 at the desired vertical level, corresponding to the desired number of clusters. The number of vertical stems of the dendrogram intersected by the horizontal line of the cross-hairs, corresponds to the number of clusters.

After positioning the cross-hairs and clicking on the left-mouse button, the chosen clustering is displayed in two ways. First, coloured rectangles are drawn on the dendrogram to indicate samples that are clustered together. In Figure 2, a scatterplot, colour-coded in the same fashion, is made of the samples, thus revealing the clustering (note that when there are more than two features in the dataset, the clustering is performed taking *all* features into account, but that only the first two features are displayed in Figure 2). The program remains in a loop where new clusterings can be obtained by left-mouse clicking on the dendrogram. To terminate the program, right-click anywhere on the dendrogram.

Exercise 9.2 Load dataset `rnd`. Make a scatterplot. This is a uniformly distributed dataset, with no apparent cluster structure. We will hierarchically cluster this dataset to get an idea of what a dendrogram looks like when there is no structure in the data.

(a) Perform hierarchical clustering with `interactclust` on the `rnd` dataset with complete linkage: `interactclust(+a,'c');`

(b) Cut the dendrogram at arbitrary levels to see how the samples are clustered. Look at the structure of the dendrogram. What is apparent?

(c) Repeat this for single and average linkage. Do you observe the same behavior as with complete linkage?

Exercise 9.3 (a) Perform hierarchical clustering with `interactclust` on the `hall` dataset with complete linkage: what do the lengths of the vertical stems in the dendrogram tell us about the clustering?

(b) Cut the dendrogram at different levels, i.e. inspect different numbers of clusters by left-mouse clicking at the desired level. Can you think of ways in which a good clustering can be defined?

(c) Can you devise a simple rule-of-thumb (in terms of vertical stem lengths) for finding a good clustering in the dendrogram?

Exercise 9.4 (a) Now perform single linkage hierarchical clustering: (`interactclust(+a,'s');`) on the `hall` dataset. Do you notice any significant differences with the complete linkage dendrogram?

(b) Do you notice any differences with the average linkage dendrogram (`interactclust(+a,'a')`)?

Exercise 9.5 (a) Load and plot the `cigars` dataset. What is an appropriate number of clusters? Why?

(b) Perform single linkage hierarchical clustering on the `cigars` dataset. Can you obtain the desired clustering with single linkage hierarchical clustering?

(c) Now perform complete linkage hierarchical clustering on the `cigars` data. Does the optimal number of clusters obtained with single linkage hierarchical clustering also produce acceptable results in complete linkage hierarchical clustering? If not, why not?

Exercise 9.6 (a) Perform complete linkage hierarchical clustering on the `messy` dataset. What is an appropriate number of clusters? Why?

(b) Now perform single linkage clustering on this dataset. Select the same number of clusters that you determined to be optimal with complete linkage. Is it also optimal in this case?

(c) What is an appropriate number of clusters for the single linkage case? Why?

——————————————— OPTIONAL ———————————————

Exercise 9.7 Perform hierarchical clustering on a dataset of your choice. Bear in mind that only the first two dimensions of higher dimensional datasets are displayed in the scatterplot. To visualise other dimensions, reorganise the columns in your dataset so that the appropriate variables are in columns one and two.

——————————————— END OPTIONAL ———————————————

9.2   *K*-means clustering

The operation of this algorithm was explained in the lecture. In this practical session, we will familiarise ourselves with the *K*-means algorithm by applying it in various settings. A function `kmclust` is provided to perform *K*-means clustering. Take a moment to familiarise yourself with the input, output and operation of this function.

14.5.1

Exercise 9.8 The initialisation of the *K*-means algorithm (positioning of the prototypes) can be done in several ways. Two simple ways are:

1. placing the prototypes uniformly distributed in the domain (approximated by the bounding box) of the data; or

2. selecting a number of the samples at random as prototypes.

For reasons of simplicity we recommend the second option.

(a) Can you think of possible advantages and disadvantages to these two initialisation approaches?

(b) Apply `kmclust` with `plotflag` and `stepflag` set on the `cigars` and `messy` datasets for different numbers of prototypes. Is K-means better suited to one of these datasets, and if so, why?

9.2.1 The local minimum problem

You might recall that the K-means algorithm is a simple iterative way of attempting to find those prototype positions that correspond to the global minimum of the total within-scatter. However, there are two factors that could cause this minimisation procedure to get stuck without having reached the global minimum. First, K-means is a procedure that always updates the positions of the prototypes such that the within-scatter *decreases*. Secondly, the within-scatter is not a monotonically decreasing function of prototype positions. This implies that from a specific set of non-optimal prototype positions the road to the optimum (global minimum) contains an initial *increase* in within-scatter before reaching the optimum. If K-means ends up in such a position (local minimum), it gets stuck. The following exercise illustrates this.

Exercise 9.9 (a) Load dataset `triclust`. Inspect it with the aid of a scatterplot. There are clearly three equally spaced, spherical clusters.

(b) Run your K-means algorithm several times, for $g = 3$ clusters, until a solution is found where there are two prototypes in a single cluster, and the third is positioned between the remaining two clusters. Why is this a local minimum?

(c) Does hierarchical clustering also suffer from this problem?

(d) What can we do to overcome the local minimum problem?

9.3 Clustering with a mixture-of-Gaussians

During the lectures, the concept of clustering based on the quality of a mixture-of-Gaussian density fit to the data was discussed. The operation of the Expectation-Maximisation (EM) algorithm, which is employed to estimate the parameters of the mixture model, was also discussed in detail. In the following set of exercises, mixture model clustering will be explored with the aid of the function `em`.

14.2

Exercise 9.10 (a) Load the `triclust` dataset and play around with the function `em` (`em(data,nmodels,type,0,1,0);`). Vary the number of Gaussians employed (`nmodels`) in the mixture model, and also vary the `type` of Gaussian employed. Relate the `type` (`'circular'`,`'aligned'`,`'gauss'`) of the Gaussian to its covariance matrix.

(b) On the `cigars` dataset, fit an unconstrained Gaussian (`type = 'gauss'`) mixture model using the function `em`. For the number of clusters g, assume the following values: $g = 1, 2, 3, 4, 5$. Which g do you expect to be the best? .

(c) Repeat this 20 times (without plotting, i.e. setting the `plot_mix` and `plot_resp` arguments to zero) and plot the average log-likelihood (first output argument of `em`) and its standard deviation as a function of g (hint: use the `errorbar` function). What characteristic of the log-likelihood function indicates the preferable number of models (clusters)?

(d) Do the same for the `rnd` dataset (which has no structure) to confirm the relationship between the log-likelihood curve and the number of clusters. What is the behavior of the curve for the random dataset?

(e) Now try clustering the `messy` dataset. What is the best shape to employ for the Gaussians? What is the optimal number of clusters?

Clustering may be applied for image segmentation, e.g. on the basis of colors.

Exercise 9.11 A full-color image may be segmented by clustering the color feature space. For example, read the `roadsign10` image:

```
>> im = imread('roadsign10.bmp');
>> show(im)
```

The three colors may be used to segment the images based on their pixel values only. Convert the image to a dataset using the `im2feat`command. Use a small subset of pixels for finding four clusters in the 3D color space:

```
>> a = im2feat(im);                    % create a dataset
>> trainingset = gendat(a,500)         % create a small training subset
>> [d,w] = emclust(trainingset,nmc,4)  % cluster the data
```

The retrieved classifier `w` may be used to classify all image pixels in the color space:

```
>> lab = classim(a,w);
>> figure
>> imagesc(lab)                        % view image labels
```

Finally, we will replace the color map used to render the label image to the mean vector of each cluster:

```
>> aa = prdataset(a,lab(:))            % create labeled dataset
>> map = +meancov(aa)                  % compute class means
>> colormap(map)                       % set color map accordingly
```

(a) Find out the correspondences between the image parts (sky, road, etc.) and the clusters by visualizing the outputs of the classifier `w` on the entire image.

(b) Is it possible to improve the result by using more clusters?

- **Clustering is not the same as classification: it does not use the class labels. Clustering is an *unsupervised learning technique*, while classification is a *supervised learning technique***

- **Although the examples here were mostly in 2D, clustering can be performed for data sets with any number of features**

- **A mixture-of-Gaussians is a model between a single Gaussian density and a Parzen density estimate**

Week 10

Cluster validation

In this week, we will explore ways to guide our choice when we judge clustering results and determine the number of clusters.

Objectives for this week:

- to learn to use cluster validity measures;
- to learn how the within-scatter, jumps in the fusion graph and the Davies-Bouldin cluster validity measure can be employed to extract the "optimal" number of clusters from a dataset;

10.1   Cluster validation

In the lectures this week, various clustering approaches were presented. Last week you applied hierarchical clustering with different linkage types and the Euclidean dissimilarity measure as well as K-means clustering to several datasets. One aspect of clustering that we avoided last week was the determination of the "optimal" number of clusters in the dataset. More specifically, given a particular hierarchical clustering, one needs to determine where to cut the dendrogram to produce a clustering of the data and one needs to know which value of g to use as input in the K-means clustering algorithm. Last week you had a small taste of cluster validation when you tried to determine the optimal number of clusters to employ in mixture-of-Gaussians clustering algorithm.

This practical session is intended to familiarise you with the different cluster validity measures that were discussed in the lecture. We will achieve this by:

1. employing the fusion graph as a simple, intuitive measure of clustering tendency in hierarchical clustering;

2. coding and applying the within-scatter in the K-means clustering algorithm to estimate the optimal number of clusters;

3. coding and applying the Davies-Bouldin index to various algorithms and datasets.

10.1.1 Fusion graphs

The *fusion graph* plots the *fusion level* as a function of the number of clusters (g). For example, the fusion level at $g = 2$ represents the (single, complete, average) link distance between the clusters that are merged to create two clusters from three clusters. A simple heuristic to determine the number of clusters in hierarchical clustering is to cut the dendrogram at the point where we observe a large jump in the fusion graph.

Exercise 10.1 (a) Why is this a reasonable heuristic to employ?

The following three exercises focus on the estimation of the number of clusters based on the fusion graph.

Exercise 10.2 (a) Load the `triclust` dataset. Perform single linkage hierarchical clustering and display the fusion graph by setting the third optional argument: `interactclust(triclust,'s',1)`. Where do you observe the largest jump?

(b) Cut the dendrogram at this position, to check whether this is a reasonable number of clusters. Now perform complete linkage hierarchical clustering. Does the fusion graph give a clear indication of the number of clusters in the data? If not, why not?

Exercise 10.3 (a) Load the `hall` dataset. Perform single linkage hierarchical clustering and display the fusion graph. What do you observe in the fusion graph?

Exercise 10.4 (a) Finally, load the `messy` dataset. Perform single linkage hierarchical clustering. According to the fusion graph, where should the dendrogram be cut?

(b) Does a satisfactory clustering result from cutting the dendrogram at this point? Motivate.

(c) Now perform complete linkage clustering. Is the clustering suggested by the fusion graph better or worse than the clustering obtained with single linkage clustering?

10.1.2 The within-scatter criterion

Exercise 10.5 (a) Now, generate a two-cluster, 2D dataset: `a = +gendats([50,50],2,d)`, for `d = 10`. Make a scatterplot of the data.

(b) Recall that the function `kmclust` outputs the within-scatter associated with the obtained clustering. For the generated two-cluster dataset, compute the cluster within-scatter for $g = [1, 2, 3, 5, 10, 20, 40, 50, 100]$, and compute the *normalised* within-scatter values as a function of g (normalise by dividing all within-scatter values by the within-scatter for $g = 1$). Repeat this 10 times, and employ the `errorbar` function to plot the average and standard deviation of the cluster within-scatter as a function of g.

(c) What are the most prominent characteristics of this curve?

(d) Now for `d = 5`, generate a new dataset. For this dataset, generate the same normalised within-scatter plot as in the previous exercise, i.e. the within-scatter as a function of g. What is the biggest difference between the datasets? What is the biggest difference between the within-scatter curves?

Exercise 10.6 Generate a dataset consisting of a single Gaussian, i.e. use `d = 0`. Plot the same within-scatter curve. **(a)** How does this curve differ from the other curves?

10.1.3 The Davies-Bouldin index

D.L. Davies and D.W. Bouldin[1] introduced a cluster separation measure which is based on both the within-scatter of the clusters in a given clustering and the separation between the clusters. Formally, this measure is known as the Davies-Bouldin index (DBI). It assumes that clusters are spherical, and that a desirable clustering consists of compact clusters that are well-separated.

Suppose we wish to compute the DBI for a clustering consisting of n objects assigned to g clusters. We can compute a score for every possible pair of clusters in this clustering, which is inversely proportional to the distance between the cluster means and directly proportional to the sum of the within-scatters in the pair of clusters. This score is given by

$$R_{jk} = \frac{\sigma_j + \sigma_k}{\|\boldsymbol{\mu}_j - \boldsymbol{\mu}_k\|}, \quad j, k = 1, 2, \ldots, g; \quad k \neq j. \tag{10.1}$$

Here $\boldsymbol{\mu}_j$ is the mean of all the objects in cluster j and σ_j is the within scatter of cluster j, given by:

$$\sigma_j = \sqrt{\frac{1}{n_j} \sum_{\boldsymbol{x}_i \in C_j} \|\boldsymbol{x}_i - \boldsymbol{\mu}_j\|^2}, \tag{10.2}$$

where C_j is the set of objects associated with cluster j and n_j is the number of objects in cluster j. The score, R_{jk}, is small when the means of clusters j and k are far apart and the sum of the within-scatter for these clusters is small. Since cluster j can be paired with $g-1$ other clusters, resulting in $g-1$ pair-scores, $R_{jk}, j = 1, 2, \ldots, g; k \neq j$, a *conservative* estimate of the cluster score for cluster j, when paired with all other clusters, is obtained by assigning the maximal pair-score with cluster j:

$$R_j = \max_{k=1,2,\ldots,g; \ k \neq j} R_{jk}. \tag{10.3}$$

The Davies-Bouldin index of the complete clustering is then determined by averaging these maximal pair-scores for all clusters:

$$I_{DB} = \frac{1}{g} \sum_{j=1}^{g} R_j. \tag{10.4}$$

---

[1]IEEE Transactions on Pattern Analysis and Machine Intelligence 1, pp. 224–227, 1979.

Exercise 10.7 We will employ the Davies-Bouldin index to evaluate the clustering produced by hierarchical clustering. We will do so for a range of clusters in order to determine the optimal number of clusters, i.e. the best level to cut the dendrogram at. To achieve this we employ the function `hdb()`.

(a) The function `hdb()` has as inputs a dataset, the linkage type, the distance measure and the range of clusters (start,stop) for which you wish to evaluate the quality of the clustering. It computes, for each clustering, the DBI. Familiarize yourself with the operation of this function.

(b) Load the `triclust` dataset and make a scatterplot. What do you expect the DBI curve as a function of the number of clusters to look like?

(c) Now apply `hdb` to this dataset with Euclidean distance, complete linkage clustering, starting at 2 clusters and stopping at 10. What is evident from the DBI curve?

(d) Apply `hdb` to the `hall` dataset with Euclidean distance, complete linkage clustering, starting at 2 clusters and stopping at 20. What do you observe in the obtained curve? Can you explain your observation?

(e) Finally, apply `hdb` to the `cigars` dataset with Euclidean distance, complete linkage clustering, starting at 2 clusters and stopping at 20. Inspect the DBI curve. Do you detect the expected number of clusters? Now try single linkage clustering. Does this help? Why (not)?

10.1.4 Log-likelihood and information criteria

Last week, you already looked at how to use log-likelihood to judge the number of clusters in exercise 9.10.

Exercise 10.8 Repeat exercise 9.10, but generate plots of the Akaike Information Criterion (AIC) and the Bayesian Information Criterion (BIC) instead.

TAKE-HOME MESSAGES

- **Some cluster criteria indicate the optimal number of clusters at the point where the curve flattens, e.g. fusion graphs and within-scatter, while other criteria reveal the optimal number of clusters at an extremum**

- **Most cluster validity measures assume a model of a typical cluster (e.g. spherical) and will therefore not detect a good clustering if clusters have a different shape**

Week 11

Feature Selection

Objectives When you have done the exercises and read the text for this week, you should

- have an understanding of the computational complexity of feature selection
- have an understanding of the approximative nature of selection schemes
- have an understanding of some basic properties of scatter matrices
- have a basic idea of the concept of VC-dimensions.

Exercise 11.1 Given a data set with 100 features. Say you want to find the 5 best features. How many feature sets do you need to check for this in case we have to rely on an exhaustive search?

11.1 Selection Experiments

Exercise 11.2 Check the help of a couple standard search strategies for features selection, e.g. `featself`, `featselb`, `featsellr`, `featselp`, etc.

(a) Create 100 samples from a 10D "difficult" distribution using `a = gendatd(100,10)`. Apply a few of these feature selection strategies together with the `'maha-s'` criterion (what does `'maha-s'` do?).

(b) Which features are the best two and do the algorithms pick them out? And if not, do they agree among each other on the optimal choice?

Exercise 11.3 Load the diabetes data set using `a = diabetes`. This data set is a real-world data set (though an old one). The aim is, or rather, was to predict the presence of diabetes mellitus in people based on the eight measurements made. Some of the features included are diastolic blood pressure, triceps skin fold thickness, and age.

(a) Split up the data set in 50% for training and the rest for testing. Based on the training set, what is the optimal 5-dimensional subspace for the fisher classifier? (Hint: you can loop you way out of this but it might be more handy to use some feature from `nchoosek`; read the help carefully.)

(b) What is the optimal 5-dimensional space for a 1NN classifier?

(c) What is the error attained on the test set for these classifiers with their respective optimal feature spaces.

(d) Check some search strategies and selection criteria (`help feateval`) and see whether they are able to pick out the same 5-dimensional subspace. If not, do at least the error rates on the test set come close in case one uses the same classifier?

(e) Do another 50-50 split of the original data set. When redoing some of the feature selections, do you find back the same feature sets?

Exercise 11.4 This code you get for free: `a = mfeat_zer`

```
[trn,tst] = gendat(a,.04)
clsf = ldc
featnum = [1:5:47]

prwaitbar off
prwarning off

mf = max(featnum)
e0 = clevalf(trn,clsf,featnum,[],1,tst);
rp = randperm(size(a,2));
er = clevalf(trn(:,rp),clsf,featnum,[],1,tst(:,rp));
[w,r] = featseli(trn,'eucl-m',mf)
e1 = clevalf(trn*w,clsf,featnum,[],1,tst*w);
[w,r] = featself(trn,'eucl-m',mf)
e2 = clevalf(trn*w,clsf,featnum,[],1,tst*w);
[w,r] = featselb(trn,'eucl-m',mf)
e3 = clevalf(trn*w,clsf,featnum,[],1,tst*w);

figure
plote({e0,er,e1,e2,e3})
legend({'o','r','i','f','b'})
```

(a) Run it a couple of times and have a look at the output, both in the figure and in the command window, and try to understand it. Also have a look at the code of course. E.g. what does `randperm` do in the code and what is `rp` used for? What does `featseli` do? And `'eucl-m'`?

(b) Can you explain what the backward feature selection does if you want to extract `mf` features from dataset `a`? Using this explanation, can you then explain why the feature curve for the backward selection looks so poor?

(c) Can you find a selection strategy, which optimal performance is, by and large, better than the best performing one in (a)? Recall that there are not only so-called filter approaches .

11.2 Stuff on Scatter Matrices

Exercise 11.5 Given a set of feature vectors $\{x_i\}$ for which the sample covariance matrix equals C and given a linear transformation A. Show that the covariance matrix for the

64

transformed feature vectors $\{Ax_i\}$ equals ACA^T.

Exercise 11.6 Note: this might be a tiresome exercise but if you have too much time on your hands... One way or the other, you should know that $S_m = S_b + S_w$ for a fact!

─────────────────────── OPTIONAL ───────────────────────

(a) Show that $\Sigma_i = E[(x - \mu_i)(x - \mu_i)^T] = \frac{1}{2}E[(x - x')(x - x')^T]$ in which x and x' are equally distributed, i.e., according to the distribution of class i.

(b) Exercise 5.12 from the book. Show that the mixture scatter matrix is the sum of the within-class and between-class scatter matrices. The fact proven in (a) can be very Ex. 5.12
helpful here.

─────────────────────── END OPTIONAL ───────────────────────

Exercise 11.7 (a) Variation to Exercise 5.18 from the book (where the book is slightly inaccurate). Convince yourself that if the number of classes equals M that the rank of Ex. 5.18
the between scatter matrix S_b is *at most* $M - 1$. You can either try to proof this, which might be a bit though, or you can develop some insight be staring at the formula or doing some experiments in, say, Matlab.

(b) In which (two) different cases can the rank of S_b be smaller than $M - 1$? Note that this applies similarly to any estimate of a covariance matrix based on M samples.

─────────────────────── OPTIONAL ───────────────────────

Exercise 11.8 Make Exercise 5.20 from the book. (Note that S_{xw} actually equals S_w?)

Ex. 5.20

─────────────────────── END OPTIONAL ───────────────────────

11.3   More...

Exercise 11.9 Determine what is VapnikChervonenkis dimension for classifiers with a Sect. 5.10
quadratic decision boundary in 1D and 2D? (Again, there is no need to find an ac- tual proof.)

- **Reducing the dimensionality by means of feature selection can improve performance**

- **Procedures are approximate and will, typically, not find the optimal features**

- **Applying feature selection strategies easily becomes rather time consuming**

- **Especially in the small sample setting, feature selection can be "unstable"**

- $S_m = S_b + S_w$.

Week 12

Feature Extraction

Objectives When you have done the exercises and read the text for this week, you should

- have an understanding of some PCA / KL transform basics
- have an understanding of LDA basics
- have an understanding of some properties of scatter matrices
- know how to kernelize PCA

Exercise 12.1 Make sure that you had enough practice with last week's exercises.

12.1   Principal Component Analysis Karhunen-Loève Transform

Exercise 12.2 Load the diabetes data set using `a = diabetes`. We revisit Exercise 11.3 from the foregoing week.

(a) Consider one of the better performing feature selection strategies and compare its performance (e.g. using `clevalf`), for various feature dimensionalities with the performance of PCA (`pcam`). Do this based on two quite differently behaving classifiers, say, a complex and a simple one.

(b) Does a normalization of the individual features lead to an improved error rate when using PCA?

Exercise 12.3 (a) Load the `mfeat_pix` data set. Scatter this data set. What objects does this data set describe and what are the features that are used to describe them?

(b) Split up the set into a training set containing 50% of the data. The other 50% goes into the test set. Calculate a mapping `w` on the training set that consist of the first 2 PCs. Display, in two separate figures, the 2-dimensional training set and test set obtained by means of PCA. What noticeable difference between the scatter plots, from a classification point of view, can be observed?

(c) Now split up the set into a training set containing 5% of the data. Again the rest goes into the test set. Determine a PCA on the training set. What is the maximum number of features that PCA can extract? Why is this the case?

(d) Compare feature curves obtained with the original data sets and some obtained with the PCA transformed data sets for a couple of classifiers (e.g. using something like `clevalf(trn,classifier,[5:5:90],[],1,tst)`). Is there any benefit of the PCA feature space over the pixel feature space? Do you observe peaking for any of the classifiers?

12.2 Scatter Matrices, Repeating. . .

Ex. 5.18

Exercise 12.4 (a) Variation to Exercise 5.18 from the book (where the book is slightly inaccurate). Convince yourself that if the number of classes equals M that the rank of the between scatter matrix S_b is *at most* $M - 1$. You can either try to proof this, which might be a bit though, or you can develop some insight be staring at the formula or doing some experiments in, say, Matlab.

(b) In which (two) different cases can the rank of S_b be smaller than $M - 1$? Note that this applies similarly to any estimate of a covariance matrix based on M samples.

12.3 Supervised Feature Extraction: the Fisher Mapping

Exercise 12.5 (a) Consider three arbitrary points in a 2-dimensional feature space. One is from one class and the two others from the other. That is, we have a two-class problem. What (obviously?) is the 1-dimensional subspace that is optimal according to the Fisher mapping/LDA? What value would the Fisher criterion take on in this case? Explain your answers.

(b) Consider an arbitrary two-class problem in three dimensions with three points in one class and one in the other. How can one determine a 1-dimensional subspace in this 3D space that optimizes the Fisher criterion? (Like in (a), a geometric view maybe seems the easiest here.)

(c) Again we take a two-class problem with four objects in 3D but now both classes contain the same number of points. Again determine the 1D subspace that Fisher gives.

(d) When dealing with two points from two different classes in three dimensions, how many essentially different Fisher optimal 1D subspaces are there?

Exercise 12.6 Take `a = mfeat_pix` but now split up the set into a training set containing 10% of the data. Again the rest goes into the test set. Determine a PCA on the training set.

(a) What is the maximum number of features that Fisher can extract? Why is this the case?

(b) Calculate a mapping `w` on the training set that consist of the first 3 components provided by `fisherm`. Display the three (3) different 2-dimensional scatters of the

training set after the Fisher mapping (use the 'gridded' option in scatterd). What happened here? Explain what you see (hint: reconsider the solutions you found in the previous simple, low-dimensional examples).

(c) Use the mapping from (b) and apply it to the test set. Display, in a different figure, the three (3) different 2-dimensional scatters. What noticeable difference between the scatter plots can be observed?

(d) Would you say the Fisher mapping is overtrained?

(e) Does the mapping with PCA look better or worse than the mapping using Fisher/LDA? Explain your answer.

(f) Can you come up with a combination of PCA and Fisher that performs better than using PCA or Fisher separately? Does it depend on the choice of classifier?

12.4 Kernel PCA

Exercise 12.7 Assume that the overall mean of a data set is zero and that all feature vectors are stored in the rows of the matrix X.

(a) Show that the eigenvector of $X^T X$ with the largest eigenvalue is equal to the largest principal component.

(b) Given that v is a (column) eigenvector with eigenvalue λ of XX^T, show that $X^T v$ is an eigenvector of $X^T X$. What is the eigenvalue associated to the eigenvector $X^T v$?

(c) Using $X^T v$, a (row) feature vector x from the original space can be mapped onto the first PC by $xX^T v$.

(d) Describe now how one can kernelize PCA: describe the procedure to do the actual calculation. All ingredients are there. (Hint: realize what information XX^T and xX^T contain.)

Take-home messages

- **Reducing the dimensionality by means of feature extraction can improve performance**

- **Procedures are approximate and will, typically, not find the optimal features**

- **Especially in the small sample setting, supervised methods can overtrain**

- **PCA can be kernelized.**

Week 13

Dissimilarity Representation

Objectives When you have done the exercises and read the text for this week, you should

- understand how dissimilarities relate to kernels and features,
- understand how dissimilarities can represent an object,
- get `DisTools` to work,
- be able to employ `DisTools` to your pattern recognition problems.

Any feature based representation `a` (e.g. `a = gendath(100)`) can be converted into a (dis)similarity representation `d` using the `proxm` mapping:

```
>> w = proxm(b,par1,par2); % define some dissimilarity measure
>> d = a*w;                 % apply to the data
```

in which the representation set `b` is a small set of objects. In `d` all (dis)similarities between the objects in `a` and `b` are stored (depending on the parameters `par1` and `par2`, see `help proxm`). `b` can be a subset of `a`. The dataset `d` can be used similarly as a feature based set. A dissimilarity based classifier using a representation set of 5 objects per class can be constructed and visualised by:

```
>> b = gendat(a,5); % the representation set
>> w = proxm(b);    % define an Euclidean distance mapping to the objects in b
>> c = a*w;         % construct a dissimilarity representation for a
>> scatterd(c*pca(c,2)); % compute and plot a PCA
```

Exercise 13.1 Generate a set of 50 objects per class by `gendatb`.

(a) Compute a dissimilarity matrix for using the city-block distance. Visualize the result by multi-dimensional scaling.

(b) Use a randomly selected representation set of 5 objects per class. Compute a dissimilarity representations using `proxm`. Visualize the result by PCA.

(c) Reduce the size of the representation set to 2 objects by feature selection and inspect the result by `scatterd`.

In most applications the dissimilarity representation is computed on raw data like images and not on an already given feature representation. We will use distances resulting from a study by Zongker and Jain on deformable templates between 10 classes of handwritten digits ('0' - '9'). Each class is given by 200 examples, resulting in a 2000 x 2000 dissimilarity matrix. We will however just use two of them. For handling dissimilarity matrices the following commands from the `DisTools` toolbox are useful:

- `selcdat`: selection of some classes from a dataset representing a dissimilarity matrix

- `genddat`: split of a dissimilarity matrix in a training set and a test set. Note that such a split effects the numbers of rows as well as columns.

- `psem`: pseudo-Euclidean embedding of a dissimilarity matrix. Note that this command returns a signature describing the numbers of positive and negative eigenvalues found during (pseudo)-Euclidean embedding. (Currently, its use is discouraged.)

- `psdistm`: computation of squared distances in a pseudo-Euclidean space with given signature. (Currently, its use is discouraged.)

- `testkd`: 1-NN classification error based on a given set of distances between a test set and a training set.

Exercise 13.2 Load the Zongker dataset by `readzongker` and inspect its size. Select the just two classes (1 and 2) using `selcdat`. Use for the following experiments training sets and test sets of 100 objects per class.

(a) Use the entire dissimilarity matrix of the two classes (size 400 x 400) for embedding by `psem` into a 10-dimensional space. Project all dissimilarity data into this space. Generate the training and test set and compute the errors of the 1-NN and `qdc` classifiers.

──────────── OPTIONAL ────────────

(b) Select a random set of 10 objects for representation (5 from each class) and determine the corresponding dissimilarity matrix and use it for embedding. Project all dissimilarity data into this pseudo-Euclidean space. Generate the training and test set and compute the errors of the 1-NN and `qdc` classifiers.

──────────── END OPTIONAL ────────────

(c) Use the same representation set for constructing a dissimilarity space (just by reducing the numbers of columns of the total dissimilarity matrix). Generate a training set and a test set and compute in the dissimilarity space the errors of the 1-NN and `qdc` classifiers.

(d) Compare the above results with 1-NN errors for the same test set using the representation set as training set, as well as using the entire training set.

(e) What representation yields the best classifiers? Which representation is preferred if for test objects just the computation of 10 dissimilarities can be afforded?

- **There are other object representations than the standard feature vector representation!**

Appendix A

Final assignment

A.1   Case

For the purpose of this final assignment, a case is introduced. Assume your group plays the role of a pattern recognition consultancy company. Your company is faced with an assignment by a client company working on automatic bank cheque processing applications. This client wants you to research the possibility of using pattern recognition techniques to classify individual digits in bank account numbers and the monetary amount.

They are interested in two scenarios:

1. the pattern recognition system is trained once, and then applied in the field;

2. the pattern recognition system is trained for each batch of cheques to be processed.

In pattern recognition terms, Scenario 1 means the amount of training data available is large (in this case, at least 200 and at most 1000 objects per class), whereas for Scenario 2 it is much smaller (at most 10 objects per class).

A.2   Input: the NIST dataset

To help you to construct your system, the client has supplied you with a standard dataset of handwritten digits put together by the US National Institute of Standards & Technology, NIST (see `http://www.nist.gov/`). This set consists of 2800 images of handwritten digits for each of the 10 classes, "0","1", ..., "9". They were scanned from forms filled out by volunteers, thresholded in black-and-white and automatically segmented in images of 128 x 128 pixels. Sometimes parts of other digits are visible in an image. In order to save diskspace, we determined for each digit a bounding box, thereby causing images to have different sizes.

The data can be loaded and displayed as:

```
>> a = prnist([0:9],[1:40:1000])
>> show(a)
```

This loads, for each of the 10 digits, 25 objects out of the available 1000 into a `prdatafile` a. The result is shown in figure A.1.

Figure A.1: Subset of the NIST digit dataset

A.3 Expected output: deliverables

The client requires two deliverables:

- a report detailing the design choices you made *for both scenarios*, including detailed motivation. This report should discuss the following:

  1. the choice of representation (by pixels, features or dissimilarities);
  2. the actual features or dissimilarity measure used;
  3. for each representation, whether feature reduction is possible and, if so, what number of features should be used;
  4. for each representation, the optimal classifier and its type, i.e. parametric (`nmc`, `ldc`, `qdc`, `fisherc`, `loglc`), non-parametric (`knnc`, `parzenc`) or advanced (neural networks, support vector classifiers, one-class classifiers, combining classifiers);
  5. the estimated performance of the optimal classifier on novel data.

- a test of your system on a set of benchmark data withheld from you by the client (see below). For Scenario 1, your system should have lower than 5% test error; for Scenario 2, your target is 25% test error.

Note that each choice is expected to be backed up by either evidence or solid reasoning.

In the last exercise(s) for each week you have already experimented somewhat with the handwritten digit dataset. However, it is important to question each step as you construct your final digit recognition system, for two reasons:

- for Scenario 1, you should use as much training data as possible for the construction of your system (but *at least* 200 objects per class), rather than the smaller sets you used earlier; this may invalidate some of your earlier conclusions;

- you are supposed to optimise the overall performance of the system, rather than just the individual steps.

A.4   The benchmark

It is common for clients of your company to withhold data from you, which they can then use to test your system and verify whether the performance you claim to obtain is actually valid. In this case, the client has supplied you with a function you can use to benchmark your system yourself, without having access to the actual data used. It goes without saying that you may not use the test error to optimise your system in any way.

The procedure agreed upon is as follows:

- write a function `a = my_rep(m)` in which `m` is a NIST `prdatafile` set and `a` the resulting dataset;

- compute your classifier as PRTOOLS classifier `w` in which dimension reduction and classification are incorporated, so `a*w*labeld` should generate the proper labels;

- call `e = nist_eval(filename,w,n)` in which `filename` is the filename for the `my_rep` routine, `w` is your classifier and `n` is the desired number of digits per class to be tested (which you can leave empty, if you want to test on all data). A typical call therefore looks like: `e = nist_eval('my_rep',w);`. The maximum value of `n` is 100. Evaluation may take a long time. The fraction of classification errors is returned in `e`.

In your report, give the performance obtained using the classifiers selected for both scenarios and compare these to the performance predicted using the training data. Are there large differences; if so, what could be their cause?

A.5   Grading

Your report will be graded by the supervisors, acting as the client company. If the minimum performance requirements (see section A.3 are not met, a passing grade will not be given. The report is graded on the results obtained, the choices based on their results and the motivation behind these choices. The maximum grade that can be obtained for the report is an 8.

The final two points can be earned as follows:

- Include a section "Live test" in your report, in which you report on the performance obtained on actual handwritten digits and compare it to the expected performance and the benchmark performance. To this end, you should:
    - use a scanner to read in an image of a sheet of paper on which you have written a test set of digits;

77

– segment the individual digits out of the resulting image;

– classify the digit images.

You will have to organize the scanner yourself.

- Include a section "Recommendations" to your report, in which you advise your client in detail on possible steps to improve performance in light of the overall system they are constructing. For example:

  – will having more training data help?

  – would other features help?

  – does the application require reject options?

  – does a single classifier suffice for all digits on a bank cheque?

  – what is the role of time constraints?

  – etc.

A.6   Feedback

While you work on the final assignment during the last weeks, the supervisors will of course be available to answer questions and comment on your results.

Appendix B

Self-evaluation probability theory and linear algebra

In the course on Pattern Recognition, we assume a certain familiarity with probability theory and linear algebra. You should be able to answer the following questions without too many problems and, in principle, only using pen and paper. If you find you have difficulties in doing so, please consult Theodoridis (Appendices A and B), or some other textbook of your liking, to refresh your memory.

Exercise B.1 Probability and combinatorics

From an ordinary deck of 52 cards, someone draws a card at random.

(a) What is the probability that it is a black ace or a red queen?

(b) What is the probability that it is a face card or a black card?

(c) What is the probability that it is neither a heart nor a queen?

(d) If two random cards were missing from the deck, what is the probability that it is a spade?

Exercise B.2 Expectation, (co)variance, sample estimate, and correlation

The time it takes for a student to finish a statistics aptitude test has probability density function (PDF)

$$f(x) = \begin{cases} \frac{3}{8}x^2 & 0 < x < 2 \\ 0 & \text{otherwise} \end{cases} \tag{B.1}$$

(a) Determine the expected time it takes for a randomly selected student to finish the test.

(b) Determine the standard deviation of this time.

(c) Given 3 students who finish their test in 0 hours (wrong examination room), 2 in 1 hour, 1 in 2 hours, calculate the sample mean μ and standard deviation σ for this sample from $f(x)$.

(d) Given the previous numbers, we define now three outcomes: ($hour = 1, nr.students = 3$), ($hour = 2, nr.students = 2$) and ($hour = 3, nr.students = 1$).

So each outcome has two variables *hour* and *nr.students*, and we assume that these three outcomes are equally probable. Now determine the covariance and the correlation between the hours spend on the examination and the number of students that spend this amount of time.

Exercise B.3 The normal distribution

Suppose that the IQ of students at Delft University is normally distributed with mean 125 and standard deviation 10, respectively. What is the probability that of 20 randomly selected Delft students, five have an IQ of at least 135? Use the table at `https://www.stat.tamu.edu/~lzhou/stat302/standardnormaltable.pdf`, in which the cell in the row with label r and the column with label c gives the value of the standard normal CDF $\Phi(x)$ for $x = r + c$.

Exercise B.4 Correlation, covariance and independence

A stick of length 1 is broken into two pieces at a random point (drawn from a uniform distribution).

(a) Calculate the covariance between the lengths of the two pieces.

(b) Calculate the correlation coefficient between the lengths of the two pieces.

(c) Are the lenghts independent?

Exercise B.5 The multivariate normal distribution

Let X_1 and X_2 be random variables with a joint multivariate normal distribution with mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$. Writing $\boldsymbol{x} = [x_1, x_2]^T$:

$$f(\boldsymbol{x}) = \frac{1}{2\pi|\boldsymbol{\Sigma}|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{x} - \boldsymbol{\mu})\right)$$

Prove that if X_1 and X_2 are not correlated, they are also independent (note that this only holds for the normal distribution!).

Exercise B.6 Bayes' rule

Suppose that for the general population, 1 in 1000 people carries the human immunodeficiency virus (HIV). A test for the presence of HIV yields either a positive (+) or negative (-) response. Suppose further that the test gives the correct answer 95% of the time. Finally, assume that the prior of having the disease, i.e. $P(H)$, is negligibly small compared to not having it.

(a) Calculate $P(-|H)$, the conditional probability that a person tests negative given that that person does have HIV.

(b) Calculate $P(H|+)$, the conditional probability that a randomly chosen person has HIV given that the test is positive.

(c) In a small town of 100 people, what is the probability that at least one person tests positive for HIV? .

Exercise B.7 Eigenvalues and eigenvectors

Given the matrix $\boldsymbol{A} = \begin{pmatrix} 6 & 2 \\ 2 & 9 \end{pmatrix}$.

(a) Check that 5 is an eigenvalue of this matrix.

(b) Check that the vector $(1, 2)^T$ is an eigenvector of $\boldsymbol{A}$. What is its eigenvalue?

(c) Write out the eigenvalue decomposition for $\boldsymbol{A}$. .

Exercise B.8 Transformations and projections

Given a $n \times N$-matrix $\boldsymbol{L}$, with $N > n$, and various N dimensional vectors $\boldsymbol{a}_i$.

(a) What is the dimensionality of $\boldsymbol{La}_i$?

(b) $\boldsymbol{L}^+$ denotes the Moore-Penrose pseudo-inverse. What is the dimensionality of $\boldsymbol{L}^+\boldsymbol{La}_i$ and why can't we just use the standard inverse in this expression?

(c) What is the relation between $\boldsymbol{L}^+\boldsymbol{La}_i$ and $\boldsymbol{La}_i$ if $\boldsymbol{L}$ is (row-)orthogonal (or more exact (row-)orthonormal)? How is such a transformation also called? What can you say about the relation if $\boldsymbol{L}$ is not orthogonal?

Appendix C

Introduction to Matlab

Ela Pękalska and Marjolein van der Glas

C.1   Getting started with Matlab

MATLAB is a tool for mathematical (technical) calculations. First, it can be used as a scientific calculator. Next, it allows you to plot or visualize data in many different ways, perform matrix algebra, work with polynomials or integrate functions. You can create, execute and save a sequence of commands to make your computational process automatic. Finally, MATLAB is also a user-friendly programming language, which gives the possibility to handle mathematical calculations in an easy way. In summary, as a computing/programming environment, MATLAB is especially designed to work with data sets as a whole such as vectors, matrices and images. Therefore, PRTOOLS, a toolbox for Pattern Recognition purposes, and DIPLIB, a toolbox for Image Processing, have been developed under MATLAB.

On most systems, after logging in, you can enter MATLAB with the system command `matlab` and exit with the command `exit` or `quit`. Running MATLAB creates one or more windows on your screen. The most important is the MATLAB *Command Window*, which is the place where you interact with MATLAB. The string `>>` (or `EDU>>` for the Student Edition) is the MATLAB prompt. When the Command window is active, a cursor appears after the prompt, indicating that MATLAB is waiting for your command.

C.1.1   Input via the command-line

MATLAB is an interactive system; commands followed by `Enter` are executed immediately. The results are, if desired, displayed on the screen. Execution of a command is only possible when the command is typed according to the rules. Table C.1 shows a list of commands used to solve indicated mathematical equations (a, b, x and y are numbers). Below you find basic information to help you starting with MATLAB:

- Commands in MATLAB are executed by pressing `Enter` or `Return`. The output will be displayed on the screen immediately. Try the following:
  ```
  >> 3 + 7.5
  >> 18/4 - (3 *  7)
  ```

Note that spaces are **not** important in MATLAB.

- The result of the last performed computation is ascribed to the variable `ans`, which is an example of a MATLAB built-in variable. It can be used on the next command line. For instance:

```
>> 14/4
ans =
      3.5000
>> ans^(-6)
ans =
    5.4399e-04
```

`5.4399e-04` is a computer notation of $5.4399*10^{-4}$. Note that `ans` is always overwritten by the last command.

- You can also define your own variables (more on variables in Section C.2.1), e.g. `a` and `b` as:

```
>> a = 14/4
a =
      3.5000
>> b = a^(-6)
b =
    5.4399e-04
```

- When the command is followed by a semicolon ';', the output is suppressed. Compare:

```
>> 3 + 7.5
>> 3 + 7.5;
```

- It is possible to execute more than one command at the same time; the commands should then be separated by commas (to display the output) or by semicolons (to suppress the output display), e.g.:

```
>> sin(pi/4), cos(pi); sin(0)
ans =
      0.7071
ans =
      0
```

Note that the value of `cos(pi)` is not printed.

- By default, MATLAB displays only 5 digits. The command `format long` increases this number to 15, `format short` reduces it to 5 again. For instance:

```
>> 312/56
ans =
    5.5714
>> format long
>> 312/56
ans =
    5.57142857142857
```

- The output may contain some empty lines; this can be suppressed by the command `format compact`. In contrast, the command `format loose` will insert extra empty lines.

Mathematical notation	MATLAB command
$a + b$	a + b
$a - b$	a - b
ab	a * b
$\frac{a}{b}$	a / b or b \ a
x^b	x^b
$\sqrt{x}$	sqrt(x) or x^0.5
$|x|$	abs(x)
π	pi
$4 \cdot 10^3$	4e3 or 4*10^3
i	i or j
$3 - 4i$	3-4*i or 3-4*j
$e,\ e^x$	exp(1), exp(x)
$\ln x,\ \log x$	log(x), log10(x)
$\sin x,\ \arctan x, ...$	sin(x), atan(x),...

Table C.1: Translation of mathematical notation to MATLAB commands.

- MATLAB is case sensitive, for example, a is written as `a` in MATLAB; `A` will result then in an error.

- All text after a percent sign `%` until the end of a line is treated as a comment. Enter e.g. the following:

```
>> sin(3.14159)           % this is an approximation of sin(pi)
```

  You should skip comments while typing the examples.

- Previous commands can be fetched back with the $\boxed{\uparrow}$-key. The command can also be changed, the $\boxed{\leftarrow}$ and $\boxed{\rightarrow}$-keys may be used to move around in a line and edit it.

C.1.2   `help`-facilities

MATLAB provides assistance through extensive online help. The `help` command is the simplest way to get help. It displays the list of all possible topics. To get a more general introduction to `help`, try:

```
>> help help
```

If you already know the topic or command, you can ask for a more specified help. For instance:

```
>> help elfun
```

provides information on the elementary math functions of MATLAB. The topic you want help on must be exact. The `lookfor` command is useful if you do not know the exact name of the command or topic, e.g.:

```
>> lookfor inverse
```

displays a list of all commands, with a short description, for which the word `inverse` is included in its help-text. Besides the `help` and `lookfor` commands, there is also a separate

mouse driven help. The `helpwin` command opens a new window on screen which can be browsed in an interactive way.

C.1.3 Interrupting a command or program

Sometimes you might spot an error in your command or program. Due to this error it can happen that the command or program does not stop. Pressing `Ctrl-C` (or `Ctrl-Break` on PC) forces MATLAB to stop the process. After this, the MATLAB prompt (`>>`) re-appears. This may take a while, though.

C.1.4 Workspace issues

If you work in the Command Window, MATLAB memorizes all commands that you entered and all variables that you created. These commands and variables reside in the MATLAB *workspace* and they might be easily recalled when needed, e.g. to recall previous commands, the $\boxed{\uparrow}$-key is used. Variables can be verified with the commands `who`, which gives a list of variables present, and `whos`, which includes also extra information. Assuming that you performed all commands given in Section A.1, after typing `who` you would obtain:

```
>> who
Your variables are:
a         ans        b         x
```

The command `clear <name>` deletes the variable `<name>` from the MATLAB workspace, `clear` or `clear all` removes all variables. This is useful when starting a new task. For example:

```
>> clear a x
>> who
Your variables are:
ans       b
```

C.1.5 Saving and loading data

The easiest way to save or load MATLAB variables is by using (clicking) the **File** menu-bar, and then the **Save Workspace as...** and **Load Workspace...** items. Alternatively, the MATLAB commands `save` and `load` can be used. `save` allows for saving your workspace variables either into a binary file or an ASCII file. Binary files automatically get the '.mat' extension. For ASCII files, it is recommended to add a '.txt' or '.dat' extension. Study the use of the `save` command:

```
>> s1 = sin(pi/4);
>> c1 = cos(pi/4); c2 = cos(pi/2);
>> str = 'hello world';        % a string
>> save data                   % saves all variables in binary format to data.mat
>> save numdata s1, c1         % saves numeric variables: s1 and c1 to numdata.mat
>> save allcos.dat c* -ascii   % saves c1,c2 in 8-digit ascii format to allcos.dat
```

The `load` command allows for loading variables into the workspace. It uses the same syntax as `save`. It might be useful to clear the workspace before each `load` and check which variables

are present in the workspace (use `who`) after loading. Assuming that you have created the files above, the variables can be now loaded as:

```
>> load data                    % loads all variables from data.mat
>> load data s1 c1              % loads only specified variables from data.mat
```

It is also possible to read ASCII files that contain rows of space separated values. Such a file may contain comments that begin with a percent character. The resulting data is placed into a variable with *the same* name as the ASCII file (without the extension). Check, for example:

```
>> load allcos.dat             % loads data from allcos.dat into variable allcos
>> who                         % lists variables present in the workspace now
```

C.2   Mathematics with vectors and matrices

The basic element of MATLAB is a matrix (or an array). Special cases are:

- a 1×1-matrix: a scalar or a single number;
- a matrix existing only of one row or one column: a vector.

Note that MATLAB may behave differently depending on input, whether it is a number, a vector or a 2D matrix.

C.2.1   Assignments and variables

Formally, there is no need to declare (i.e. define the name, size and the type of) a new variable in MATLAB. A variable is simply created by an assignment (e.g. `a = 15/4`). Each newly created numerical variable is *always* of the `double` type. You can change this type by converting it into e.g. the `single` type[1].

Bear in mind that *undefined* values cannot be assigned to a variable. So, the following is not possible:

```
>> clear x;                     % to make sure that x does not exist
>> f = x^2 + 4 * sin (x)
```

It becomes possible by:

```
>> x = pi / 3;  f = x^2 + 4 * sin (x)
```

Here are some examples of different types of MATLAB variables (check their types by using `whos`):

```
>> M  = [1 2; 3 4; 5 6]        % a matrix
>> 2t = 8                      % what is the problem with this command?
>> c  = 'E'                    % a character
>> str = 'Hello world'         % a string
```

There is also a number of built-in variables, e.g. `pi`, `eps` or `i`, presented in Table C.2. In addition to creating variables by assigning values to them, another possibility is to copy one

---

[1]A variable `a` is converted into a different type by performing e.g. `a = single(a)`, etc.

Variable name	Value/meaning
`ans`	the default variable name used for storing the last result
`pi`	$\pi = 3.14159...$
`eps`	the smallest positive number that added to 1, creates a result larger than 1
`inf`	representation for positive infinity, e.g. `1/0`
`nan` or `NaN`	representation for `not-a-number`, e.g. `0/0`
`i` or `j`	$i = j = \sqrt{-1}$
`nargin/nargout`	number of function input/output arguments used
`realmin/realmax`	the smallest/largest usable positive real number

Table C.2: Built-in variables in MATLAB.

variable, e.g. `b` into another, e.g. `a`. In this way, the variable `a` is automatically created (if `a` already existed, its previous value is now lost):

```
>> b = 10.5;
>> a = b;
```

If `min` is the name of a function (see Section C.4.2), then `a` defined as:

```
>> b = 5; c = 7;
>> a = min (b,c);
```

will call that function, with the values `b` and `c` as parameters. The result of this function (its return value) will be written (assigned) into `a`.

C.2.2   Vectors

Row vectors are lists of numbers separated either by commas or by spaces. They are examples of simple arrays. First element has index 1. The number of entries is known as the *length* of the vector (the command `length` exists as well). Their entities are referred to as *elements* or *components*. The entries must be enclosed in `[ ]`:

```
>> v = [-1 sin(3) 7]
v =
   -1.0000    0.1411    7.0000
```

A number of operations can be performed on vectors. A vector can be multiplied by a scalar, or added/subtracted to/from another vector with *the same* length, or a number can be added/subtracted to/from a vector. A particular value can be also changed or displayed. All these operations are carried out element-by-element. Vectors can be also built from the

already existing ones:

```
>> v = [-1 2 7]; w = [2 3 4];
>> z = v + w                        % an element-by-element sum
z =
     1     5    11
>> t = [2*v, -w+2]
ans =
    -2     4    14     0    -1    -2
>> v(2) = -1                        % change the 2nd element of v
v =
    -1    -1     7
```

A colon notation is an important shortcut, used when producing row vectors (see Table C.3 and `help colon`):

```
>> 2:5
ans =
     2     3     4     5
```

In general, `first:step:last` produces a vector of entities with the value `first`, incrementing by the `step` until it reaches `last`:

```
>> -0.2:0.5:1.7
ans =
   -0.2000    0.3000    0.8000    1.3000
>> 5.5:-2:-2.5                      % a negative step is also possible
ans =
    5.5000    3.5000    1.5000   -0.5000   -2.5000
```

Parts of vectors can be extracted by using a colon notation:

```
>> r = [-1:2.5:6, 2, 3, -2];    % r = [-1 1.5 4 2 3 -2]
>> r(2:2:6)                      % get the elements of r from the positions 2, 4 and 6
ans =
    1.5     2    -2
```

To create *column* vectors, you should separate entries by a semicolon ';' or by new lines. The same operations as on row vectors can be done on column vectors. However, you cannot for example add a column vector to a row vector. To do that, you need an operation called

transposing, which converts a column vector into a row vector and vice versa:

```
>> f = [-1; 3; 5]               % a column vector
f =
    -1
     3
     5
>> f'                           % f' is a row vector
ans =
    -1     3     5
>> v = [-1 2 7];                % a row vector
>> f + v                        % you cannot add a column vector f to a row vector v
??? Error using ==> +
Matrix dimensions must agree.
>> f' + v
ans =
    -2     5    12
```

You can now compute the inner product between two vectors x and y, $x^T y = \sum_i x_i\, y_i$, in a simple way:

```
>> v = [-1; 2; 7];           % v is now a column vector; f is also a column vector
>> f' * v                    % this is the inner product! f * v' is the outer product
ans =
    42
```

C.2.3   Matrices

An $n \times k$ matrix is a rectangular array of numbers having n rows and k columns. Defining a matrix in MATLAB is similar to defining a vector. The generalization is straightforward, if you know that a matrix consists of row vectors (or column vectors). Commas or spaces are used to separate elements in a row, and semicolons are used to separate individual rows. For example, a matrix can be defined as:

```
>> A = [1 2 3; 4 5 6; 7 8 9]   % row by row input
A =
    1   2   3
    4   5   6
    7   8   9
```

Transposing a vector changes it from a row to a column or the other way around. This idea can be extended to a matrix, where transposing interchanges rows with the corresponding

Command	Result
`A(i,j)`	A_{ij}
`A(:,j)`	j-th column of A
`A(i,:)`	i-th row of A
`A(k:l,m:n)`	$(l - k + 1) \times (n - m + 1)$ matrix with elements A_{ij} with $k \leq i \leq l$, $m \leq j \leq n$
`v(i:j)'`	'vector-part' $(v_i, v_{i+1}, \ldots, v_j)$ of vector v

Table C.3: Manipulation of (groups of) matrix elements.

columns, as in the example:
```
>> A2 = [1:4; -1:2:5], A2'
A2 =
     1     2     3     4
    -1     1     3     5
ans =
     1    -1
     2     1
     3     3
     4     5
>> size(A2), size(A2')          % returns the size (dimensions) of a matrix
ans =
     2     4
ans =
     4     2
```

There are also built-in matrices of size specified by the user (see Table C.4). A few examples are given below:
```
>> I = eye(2)                   % the 2-by-2 identity matrix
I =
     1     0
     0     1
>> B = randn(1,2)               % a vector of normally distributed random numbers
B =
   -0.4326   -1.6656
>> r = [1 3 -2]; R = diag(r)    % create a diagonal matrix with r on the diagonal
R =
     1     0     0
     0     3     0
     0     0    -2
```

It is often needed to build a larger matrix from the smaller ones:

```
>> x = [4; -1]; y = [-1 3]; X = [x y']
X =                               % X consists of the columns: x and y'
    4    -1
   -1     3
>> T = [-1 3 4; 4 5 6]; t = 1:3;
>> A = [[T; t] ones(3,2)]        % add a row vector t, then add two columns of 1's
A =
   -1    3    4    1    1
    4    5    6    1    1
    1    2    3    1    1
```

It is possible to manipulate (groups of) matrix elements (see Table C.3). A part can be extracted from a matrix in a similar way as from a vector. Each element in the matrix is indexed by a row and a column to which it belongs, e.g. `A(i,j)` denotes the element from the i-th row and the j-th column of the matrix A.

```
>> A(3,4)
ans =
    1
>> A(4,3)                        % A is a 3-by-5 matrix! A(4,3) does not exist
???  Index exceeds matrix dimensions.
```

It is easy to extend a matrix automatically. For the matrix A it can be done e.g. as follows:

```
>> A(4,3) = -2                   % assign -2 to the position (4,3); the uninitialized
A =                              % elements become zeros
   -1    3    4    1    1
    4    5    6    1    1
    1    2    3    1    1
    0    0   -2    0    0
>> A(4,[1:2,4:5]) = 4:7;         % assign A(4,1)=4, A(4,2)=5, A(4,4)=6 and A(4,5)=7
```

Different parts of the matrix A can be now extracted:

```
>> A(:,2)                        % extract the 2nd column of A; what will you get?
>> A(4,:)                        % extract the 4th row of A
ans =
    4    5   -2    6    7
>> A(2:3,[1,5])
    4    -1
    1     1
```

Table C.4 shows some frequently used matrix operations and functions. A few examples are

given below:
```
>> B = [1 4 -3; -7 8 4]; C = [1 2; 5 1; 5 6];
>> (B + C')./4                % add matrices and divide all elements of result by 4
ans =
    0.5000    2.2500    0.5000
   -1.2500    2.2500    2.5000
>> D = [1 -1 4; 7 0 -1];
>> D = B * D'; D^3             % D^3 is equivalent to D * D * D
ans =
    -4205       38390
     3839     -150087
>> D.^3 - 2                    % for all elements: raise to the power 3, subtract 2
ans =
    -3377         998
       -1     -148879
```

The concept of an empty matrix [] is also very useful in MATLAB. For instance, a few columns or rows can be removed from a matrix by assigning an empty matrix to it. Try for example:
```
>> C = [1 2 3 4; 5 6 7 8; 1 1 1 1];
>> D = C; D(:,2) = []          % now a copy of C is in D; remove the 2nd column of D
>> C ([1,3],:) = []            % remove the 1st and 3rd rows from C
```

C.3   Control flow

A control flow structure is a block of commands that allows conditional code execution and making loops.

C.3.1   Logical and relational operators

To use control flow commands, it is necessary to perform operations that result in logical values: TRUE or FALSE, which in MATLAB correspond to 1 or 0 respectively (see Table C.5 and `help relop`). The relational operators <, <=, >, >=, == and ~= can be used to compare two arrays of the same size or an array to a scalar. The logical operators &, | and ~ allow for the logical combination or negation of relational operators. The logical & and | have equal precedence in MATLAB, which means that those operators associate from left to right. In addition, three functions are also available: `xor`, `any` and `all` (use `help` to find out more).

The command `find`

You can extract all elements from the vector or the matrix satisfying a given condition, e.g. equal to 1 or larger than 5, by using logical addressing. The same result can be obtained via

the command `find`, which return the positions (indices) of such elements. For instance:

```
>> x = [1 1 3 4 1];
>> i = (x == 1)
i =
    1    1    0    0    1
>> j = find(x == 1)        % j holds indices of the elements satisfying x == 1
j =
    1    2    5
>> y = x(i), z = x(j)
y =
    1    1    1
z =
    1    1    1
```

`find` operates in a similar way on matrices. It reshapes first the matrix `A` into a column vector, i.e. all columns all concatenated one after another. Therefore, e.g. `k = find (A > 2)` is a list of indices of elements larger than 2 and `A(k)` gives the values. The row and column indices can be returned by `[I,J] = find (A > 2)`.

C.3.2 Conditional code execution

Selection control structures, `if`-blocks, are used to decide which instruction to execute next depending whether *expression* is TRUE or not. The general description is given below. In the examples below the command `disp` is frequently used. This command displays on the screen the text between the quotes.

- `if ...  end`

```
[frame=single,framesep=3mm,xrightmargin=2cm,label=Syntax]
if logical_expression
   statement1
   statement2
   ....
end
[frame=single,framesep=3mm,xrightmargin=1.2cm,label=Example]
if (a > 0)
  b = a;
  disp ('a is positive');
end
```

- `if ...  else ...  end`

```
[frame=single,framesep=3mm,xrightmargin=1.7cm,label=Syntax]
if logical_expression
   block of statements
   evaluated if TRUE
else
   block of statements
   evaluated if FALSE
end
[frame=single,framesep=3mm,xrightmargin=0.4cm,label=Example]
```

- if ... elseif ... else ... end


```
[frame=single,framesep=3mm,xrightmargin=0cm,label=Syntax]
if logical_expression1
   block of statements evaluated
   if logical_expression1 is TRUE
elseif logical_expression2
   block of statements evaluated
   if logical_expression2 is TRUE
else
   block of statements evaluated
   if no other expression is TRUE
end
[frame=single,framesep=3mm,xrightmargin=1.9cm,label=Example]
if (height > 190)
  disp ('very tall');
elseif (height > 170)
  disp ('tall');
elseif (height < 150)
  disp ('small');
else
  disp ('average');
end
```

Another selection structure is `switch`, which switches between several cases depending on an expression, which is either a scalar or a string. Learn more by using `help`.

C.3.3 Loops

Iteration control structures, *loops*, are used to repeat a block of statements until some condition is met:

- the `for` loop that repeats a group of statements a *fixed* number of times;


```
[frame=single,framesep=3mm,xrightmargin=0.8cm,label=Syntax]
for index = first:step:last
  block of statements
end
[frame=single,framesep=3mm,xrightmargin=1.8cm,label=Example]
sumx = 0;
for i=1:length(x)
   sumx = sumx + x(i);
end
```

You can specify any `step`, including a negative value. The `index` of the `for`-loop can be also a vector. See some examples of possible variations:

95

```
[frame=single,framesep=3mm,xrightmargin=0cm,label=Example 1]
for i=1:2:n
  ...
end
[frame=single,framesep=3mm,xrightmargin=0cm,label=Example 2]
for i=n:-1:3
  ....
end
[frame=single,framesep=3mm,xrightmargin=0cm,label=Example 3]
for x=0:0.5:4
  disp(x^2);
end
[frame=single,framesep=3mm,xrightmargin=-0.2cm,label=Example 4]
for x=[25 9 81]
  disp(sqrt(x));
end
```

- `while` loop, which evaluates a group of commands as long as *expression* is TRUE.

```
[frame=single,framesep=3mm,xrightmargin=2.8cm,label=Syntax]
while expression
  statement1
  statement2
  statement3
  ...
end
[frame=single,framesep=3mm,xrightmargin=1.9cm,label=Example]
N = 100;
iter = 1;
msum = 0;
while iter <= N
  msum = msum + iter;
  iter = iter + 1;
end;
```

C.4   Script and function m-files

C.4.1   Script m-files

Script m-files are useful when the number of commands increases or when you want to change values of some variables and re-evaluate them quickly. Formally, a script is an external file that contains a sequence of MATLAB commands. However, it is not a function, since there are no input/output parameters and the script variables remain in the workspace. **When you run a script, the commands in it are executed as if they have been entered**

96

through the keyboard. To create a script, open the MATLAB editor (go to the **File** menu-bar, choose the **Open** option and then **m-file**; the MATLAB Editor Window will appear), enter the lines listed below and save as `sinplot.m`:

```
x = 0:0.2:6; y = sin(x); plot(x,y);
title('Plot of y = sin(x)');
```

The script can be run by calling `sinplot`. Note that m-script file must be saved in one of the directories in MATLAB's path. The `sinplot` script affects the workspace. Check:

```
>> clear                  % all variables are removed from the workspace
>> who                    % no variables present
>> sinplot                % run the script
>> who
Your variables are:
x         y
```

These generic names, `x` and `y`, may be easily used in further computations and this can cause *side effects*. They occur, in general, when a set of commands change variables other than the input arguments. Remember that the commands within a script have access to all variables in the workspace and all variables created in this script become a part of the workspace. Therefore, it is better to use function m-files (see Section C.4.2) for a specific problem to be solved.

C.4.2 Function m-file

Functions m-files are true subprograms, as they take input arguments and/or return output parameters. They can call other functions, as well. Variables defined and used inside a function, different from the input/output arguments, are invisible to other functions and to the command environment. The general syntax is:

```
function [outputArgs] = function_name (inputArgs)
```

`outputArgs` are enclosed in `[ ]`:

- a comma-separated list of variable names;
- `[ ]` is optional when only one argument is present;
- functions without `outputArgs` are legal[2].

`inputArgs` are enclosed in `( )`:

- a comma-separated list of variable names;
- functions without `inputArgs` are legal.

MATLAB provides a structure for creating your own functions. The first line should be a definition of a new function (called a *header*). After that, a continuous sequence of comment lines, explaining what the function does, should appear. Since they are displayed in response to the `help` command, also the expected input parameters, returned output parameters and synopsis should be described there. Finally, the remainder of the function is called *the body*. Function m-files terminate execution when they reach the end of the file or, alternatively, when the command `return` is encountered. The name of the function and the name of

---

[2]In other programming languages, functions without output arguments are called *procedures*.

the file stored should be *identical*. As an example, the function `average`, stored in a file `average.m`, is defined as:

Here is an another example of a function:

```
function [avr,sd] = stat(x)
%STAT Simple statistics; computes the average and standard deviation of a vector x.
n   = length(x);
avr = sum(x)/n;
sd  = sqrt(sum((x - avr).^2)/n);
return;
```

Warning: The functions `mean` and `std` already exist in MATLAB. As long as a function name is used as a variable name, MATLAB can not perform the function. Many other, easily appealing names, such as `sum` or `prod` are reserved by MATLAB functions, so be careful when choosing your names.

When controlling the proper use of parameters, the function `error` may become useful. It displays an error message, aborts function execution, and returns to the command environment. Here is an example:

```
if (a >= 1)
  error ('a must be smaller than 1');
end;
```


C.4.3 Scripts vs. functions

The most important difference between a script and a function is that *all* script's parameters and variables are externally accessible (i.e. in the workspace), where function variables are not. Therefore, a script is a good tool for documenting work, designing experiments and testing. In general, create a function to solve a given problem for arbitrary parameters; use a script to run functions for specific parameters required by the assignment.

Command	Result
`n = rank(A)`	n becomes the rank of matrix A
`x = det(A)`	x becomes the determinant of matrix A
`x = size(A)`	x becomes a row-vector with 2 elements: the number of rows and columns of A
`x = trace(A)`	x becomes the trace (sum of diagonal elements) of matrix A
`x = norm(v)`	x becomes the Euclidean length of vector v
`C = A + B`	sum of two matrices
`C = A - B`	subtraction of two matrices
`C = A * B`	multiplication of two matrices
`C = A .* B`	'element-by-element' multiplication (A and B are of equal size)
`C = A ^ k`	power of a matrix ($k \in Z$; can also be used for A^{-1})
`C = A .^ k`	'element-by-element' power of a matrix
`C = A'`	the transposed of a matrix; A^T
`C = A ./ B`	'element-by-element' division (A and B are of equal size)
`X = A \ B`	finds the solution in the least squares sense to the system of equations $AX = B$
`X = B / A`	finds the solution of $XA = B$, analogous to the previous command
`C = inv(A)`	C becomes the inverse of A
`C = null(A)`	C is an orthonormal basis for the null space of A obtained from the singular value decomposition
`C = orth(A)`	C is an orthonormal basis for the range of A
`C = rref(A)`	C is the reduced row echelon form of A
`L = eig(A)`	L is a vector containing the (possibly complex) eigenvalues of a square matrix A
`[X,D] = eig(A)`	produces a diagonal matrix D of eigenvalues and a full matrix X whose columns are the corresponding eigenvectors of A
`S = svd(A)`	S is a vector containing the singular values of A
`[U,S,V] = svd(A)`	S is a diagonal matrix with nonnegative diagonal elements in decreasing order; columns of U and V are the accompanying singular vectors
`x = linspace(a,b,n)`	generates a vector x of n equally spaced points between a and b
`x = logspace(a,b,n)`	generates a vector x starting at 10^a and ended at 10^b containing n values
`A = eye(n)`	A is an $n \times n$ identity matrix
`A = zeros(n,m)`	A is an $n \times m$ matrix with zeros (default $m = n$)
`A = ones(n,m )`	A is an $n \times m$ matrix with ones (default $m = n$)
`A = diag(v)`	gives a diagonal matrix with the elements $v_1, v_2, \ldots, v_n$ on the diagonal
`X = tril(A)`	X is lower triangular part of A
`X = triu(A)`	X is upper triangular part of A
`A = rand(n,m)`	A is an $n \times m$ matrix with elements uniformly distributed between 0 and 1
`A = randn(n,m)`	ditto - with elements standard normal distributed
`v = max(A)`	v is a vector with the maximum value of the elements in each column of A or v is the maximum of all elements if A is a vector
`v = min(A)`	ditto - with minimum
`v = sum(A)`	ditto - with sum

Table C.4: Frequently used matrix operations and functions.

Command	Result
`a = (b > c)`	`a` is 1 if `b` is larger than `c`. Similar are: `<`, `>=` and `<=`
`a = (b == c)`	`a` is 1 if `b` is equal to `c`
`a = (b ~= c)`	`a` is 1 if `b` is not equal `c`
`a = ~b`	logical complement: `a` is 1 if `b` is 0
`a = (b & c)`	logical AND: `a` is 1 if `b` = TRUE AND `c` = TRUE
`a = (b | c)`	logical OR: `a` is 1 if `b` = TRUE OR `c` = TRUE

Table C.5: Relational and logical operations.

Appendix D

Introduction to PRTools

David M.J. Tax and Robert P.W. Duin

D.1   Introduction

The more than 100 routines offered by PRTOOLS in its present state represent a basic set covering largely the area of statistical pattern recognition. In order to make the evaluation and comparison of algorithms more easy a set of data generation routines is included, as well as a small set of standard real world datasets.

This manual treats just the basic objects and constructs in the PRTOOLS toolbox. It does not give an exhaustive explanation of all possibilities in the toolbox. To be more explicit, this manual *does not* treat:

- feature selection,

- error estimation techniques (cross-validation etc.),

- combining classifiers,

- clustering,

- applications on images.

Note: the manual tries to show the basic constructs and ideas behind the toolbox. In the actual application of the toolbox in practice, the user should frequently use the `help`. For each function defined in the PRTOOLS toolbox an extensive help text is present, explaining the syntax, the arguments, the function and the possible default values for arguments.

D.2   PRTools

PRTOOLS deals with sets of labeled objects and offers routines for generalising such sets into functions for data mapping and classification. An object is a `k`-dimensional vector of feature values. It is assumed that for all objects in a problem *all* values of the same set of features are given (and that there are no missing values). The space defined by the actual set of features is called the *feature space*. Objects are represented as points or vectors in this space.

A classification function assigns labels to new objects in the feature space. Usually, this is not done directly, but in a number of stages in which the initial feature space is successively mapped into intermediate stages, finally followed by a classification. The concept of mapping spaces and dataset is thereby important and constitutes the basis of many routines in the toolbox.

PRTOOLS makes use of the possibility offered by MATLAB 5 to define 'classes' and 'objects'. These programmatic concepts should not be confused with the classes and objects as defined in Pattern Recognition. Two classes have been defined: `prdataset` and `mapping`. A large number of operators (like `*`, `[]`) and MATLAB commands have been overloaded and have thereby a special meaning when applied to a dataset and/or a mapping.

In the coming sections it is explained how `prdataset`s and `prmapping`s can be created, modified and applied.

D.3   Dataset

The central data structure of PRTOOLS is the `prdataset`. It primarily consists of a set of objects represented by a matrix of feature vectors. Attached to this matrix is a set of labels, one for each object and a set of feature names. Moreover, a set of a priori probabilities, one for each class, is stored. In most help files of PRTOOLS, a dataset is denoted by `a`. In almost any routine this is one of the inputs. Almost all routines can handle multi-class object sets.

prdataset	
`prdataset`	Define dataset from data matrix and labels
`getdata`	Retrieve data from dataset
`getlabels`	Retrieve object labels from dataset
`getfeat`	Retrieve feature labels from dataset
`genlab`	Generate dataset labels
`renumlab`	Convert labels to numbers

Sets of objects may be given externally or may be generated by one of the data generation routines of PRTOOLS. Their labels may also be given externally or may be the result of a cluster analysis.

A dataset containing 10 objects with 5 random measurements can be generated by:
```
>> data = rand(10,5);
>> a = prdataset(data)
10 by 5 dataset with 0 classes: []
```

In the previous example no labels were supplied, therefore one class is detected. Labels can be added to the dataset by:
```
>> data = rand(10,5);
>> labs = ['A'; 'A'; 'A'; 'B'; 'A'; 'A'; 'B'; 'B'; 'B'; 'B'];
>> a = prdataset(data,labs)
10 by 5 dataset with 2 classes: [5  5]
```

In most cases the objects in the dataset are ordered (such that the first `n1` objects belong to class `A`, the next `n2` to class `B`, etc). In this case it is easier to use the `genlab` routine for the generation of the labels:

```
>> data = rand(10,5);
>> labs = genlab([5 5],['A';'B']);
>> a = prdataset(data,labs)
10 by 5 dataset with 2 classes
```

Next to class labels, it is also possible to supply feature labels. To set this parameter, the keyword `featlab` should be given before:

```
>> data = rand(10,3);
>> labs = genlab([5 5],['A';'B']);
>> feats = ['Feat1';'Feat2';'Feat3'];
>> a = prdataset(data,labs,'featlab',feats)
10 by 3 dataset with 2 classes: [5  5]
```

There are numberous other parameters which can be defined inside the dataset definition. The class prior probabilities can be defined, the dataset name, cost matrices can be given, it can be indicated whether features are numerical or categorical, etc. etc. See for more information about the possible parameters `help prdataset`.

All the values can be set by their respective commands. For instance for setting the prior:

```
>> a = setprior(a,[0.4 0.6]);
```

Analogous you can use `setcost`, `setname` or `setfeatdom`.

Various items stored in a dataset can be retrieved by the `get` series of commands:

```
>> name = getname(a);
>> prob = getprior(a);
>> nlab = getnlab(a);
>> lablist = getlablist(a);
>> [m,k,c] = getsize(a);
```

in which `nlab` are numeric labels for the objects (`1, 2, 3, ...`) referring to the true labels stored in the rows of `lablist`. The size of the dataset is `m` by `k`, `c` is the number of classes (equal to `max(nlab)`).

The original data matrix can be retrieved by `double(a)` or by `+a`. The labels in the objects of `a` can be retrieved `labels = getlab(a)`, which is equivalent to `labels = lablist(nlab,:)`. The feature labels can be retrieved by `featlist = getfeat(a)`.

Note: In many respects a `prdataset` object can be compared with a normal MATLAB matrix. Normal matrix operations like `a.^2`, `abs(a)`, `a(:,3)`, `a(1,:)`, `size(a)` etc, can directly be applied to `prdataset` objects. In some cases MATLAB routines cannot cope with the `prdataset` object. In these cases the user has to use `+a` instead of `a`.

Another way to inspect a dataset `a` is to make a scatterplot of the objects in the dataset. For this the function `scatterd` is supplied. This plots each object in the dataset `a` in a 2D graph, using a coloured marker when class labels are supplied. When more than 2 features are present in the dataset, the first two are used. When you want to make a scatterplot of two other features, you have to explicitly extract them by `scatterd(a(:,[2 5]))`.

prdataset generation	
gauss	Generation of multivariate Gaussian distributed data
gendatb	Generation of banana shaped classes
gendatc	Generation of circular classes
gendatd	Generation of two difficult classes
gendath	Generation of Higleyman classes
gendatl	Generation of Lithuanian classes
gendatm	Generation of many Gaussian distributed classes
gendats	Generation of two Gaussian distributed classes

Exercise D.1 Generate an artificial dataset (one from the `prdataset` generation table) containing 3 objects per class.

Exercise D.2 Inspect the data matrix in the `prdataset` object by using the + operator and make a scatterplot of the dataset.

Exercise D.3 Can you point out which object is plotted where in the the scatterplot? Check if each object is coloured according to their class label.

Datasets can be combined by `[a; b]` if `a` and `b` have equal numbers of features and by `[a b]` if they have equal numbers of objects.

Exercise D.4 Make a dataset containing 2 classes with 5 objects per class and call it `a1` (for instance from `gendats`). Make a second dataset containing 2 classes with 5 objects per class and call it `a2` (for instance from `gendatb`).

Combine the two datasets using

```
>> b = [a1; a2]
```

and make a scatterplot.

Next combine the two datasets using

```
>> c = [a1 a2]
```

and again make a scatterplot. In what do the two datasets differ? Check your opinions by using `size(b)` and `size(c)`.

Creating subsets of datasets can be done by `a(I,J)` in which `I` is a set of indices defining the desired objects and `J` is a set of indices defining the desired features. In all these examples the a priori probabilities set for `a` remain unchanged.

Exercise D.5 Use the dataset `c` from the previous exercise to extract feature 2 and 4. Make a scatterplot and compare it with the data matrix. Where is the first object in the data matrix plotted in the scatterplot?

There is a large set of routines for the generation of arbitrary normally distributed classes (`gauss`), and for various specific problems (`gendatc`, `gendatd`, `gendath`, `gendatm` and

prdataset manipulation	
gendat	Extraction of subsets of a given data set
gendatk	Nearest neighbour data generation
gendatp	Parzen density data generation
gendat	Extraction of test set from given dataset
prdata	Read data from file and convert into a dataset

gendats). There are two commands for enriching classes by noise injection (gendatk and gendatp). These are used for the general test set generator gendatt. A given dataset can be split randomly into a training set and a test set using gendat.

Exercise D.6 Generate a dataset using gendatb containing 5 objects per class.

> Enlarge this dataset by making a new dataset using gendatk and gendatp and appending this dataset to your original dataset.

> Make a scatterplot of the original and the second dataset, to check that it worked.

Note: have a look at the help if it is not clear how you should use a function! ☞∎

Exercise D.7 Make a dataset with 100 objects per class. Generate two new datasets trainset and testset with 60 and 40 objects per class respectively.

> Make two scatterplots of the datasets. Do they differ much?

D.4   Datafile

The above described prdataset class refers to data that are stored in the computer memory and not on disk. It has thereby its limitations w.r.t. the numbers of objects and features that can be used. Moreover, it is assumed that objects are already represented by a fixed number of vector elements (features, dissimilarities).

The recently introduced (PRTools version 4.1) class prdatafile offers several ways to start with objects stored in files as raw images, time signals or mat-files. There is thereby no limitation to the size. a prdatafile is a special type of prdataset by which many operations defined on a prdataset are supported. In addition there are tools to define several types of preprocessing filters and feature measurement commands. At some moment all objects in a prdatafile are represented by the same features and it can be converted to a prdataset. After that all procedures for dimension reduction, cluster analysis, classification and evaluation can be used.

An important difference between a prdatafile and a prdataset is that operations on the first are just stored in the prdatafile as a defined processing which is only executed at the moment it is converted to a prdataset. Operations on a prdataset are directly executed.

D.5 Mapping

Data structures of the class `mapping` store trained classifiers, feature extraction results, data scaling definitions, nonlinear projections, etc. They are usually denoted by `w`.

mapping	
`prmapping`	Define mapping
`getlab`	Retrieve labels assigned by mapping

A mapping `w` is often created by training a classifier on some data. For instance, the nearest mean classifier `nmc` is trained on some data `a` by:

```
>> a = gendatb(20);
>> w = nmc(a)
Nearest Mean, 2 to 2 trained classifier --> affine
```

`w` by itself, or `display(w)` lists the size and type of a classifier as well as the routine which is used for computing the mapping `a*w`.

When a mapping is trained, it can be applied to a dataset, using the operator `*`:

```
>> b = a*w
Banana Set, 20 by 2 dataset with 2 classes: [7   13]
```

The result of the operation `a*w` is again a dataset. It is the classified, rescaled or mapped result of applying the mapping definition stored in `w` to `a`.

mappings and classifiers	
`classc`	Converts a mapping into a classifier
`labeld`	General classification routine for trained classifiers
`testc`	General error estimation routine for trained classifiers

All routines operate in multi-class problems. For mappings which change the labels of the objects (so the mapping is actually a classifier) the routines `labeld` and `testc` are useful. `labeld` and `testc` are the general classification and testing routines respectively. They can handle any classifier from any routine.

Linear and polynomial classifiers	
`klldc`	Linear classifier by KL expansion of common cov matrix
`loglc`	Logistic linear classifier
`fisherc`	Fisher's discriminant (minimum least square linear classifier)
`ldc`	Normal densities based linear classifier (Bayes rule)
`nmc`	Nearest mean classifier
`nmsc`	Scaled nearest mean classifier
`perlc`	Linear classifier by linear perceptron
`pfsvc`	Pseudo-Fisher support vector classifier
`qdc`	Normal densities based quadratic (multi-class) classifier
`udc`	Uncorrelated normal densities based quadratic classifier

Nonlinear classifiers	
`knnc`	k-nearest neighbour classifier (find k, build classifier)
`mapk`	k-nearest neighbour mapping routine
`testk`	Error estimation for k-nearest neighbour rule
`parzenc`	Parzen density based classifier
`parzenml`	Optimization of smoothing parameter in Parzen density estimation.
`parzen_map`	Parzen mapping routine
`testp`	Error estimation for Parzen classifier
`edicon`	Edit and condense training sets
`treec`	Construct binary decision tree classifier
`tree_map`	Classification with binary decision tree
`bpxnc`	Train feed forward neural network classifier by backpropagation
`lmnc`	Train feed forward neural network by Levenberg-Marquardt rule
`rbnc`	Train radial basis neural network classifier
`neurc`	Automatic neural network classifier
`rnnc`	Random neural network classifier
`svc`	Support vector classifier

D.6   Training and testing

There are many commands to train and use mappings between spaces of different (or equal) dimensionalities. For example:

> if `a` is an `m` by `k` dataset (`m` objects in a `k`-dimensional space)
> and `w` is a `k` by `n` mapping (map from `k` to `n` dimensions)
> then `a*w` is an `m` by `n` dataset (`m` objects in a `n`-dimensional space)

Mappings can be linear (e.g. a rotation) as well as nonlinear (e.g. a neural network). Typically they can be used for classifiers. In that case a `k` by `n` mapping maps an `k`-feature data vector on the output space of an `n`-class classifier (exception: 2-class classifiers like discriminant functions may be implemented by a mapping to a 1D space like the distance to the discriminant, `n = 1`).

Mappings are of the data type `mapping`, have a size of `[k,n]` if they map from `k` to `n` dimensions. Mappings can be instructed to assign labels to the output columns, e.g. the class names. These labels can be retrieved by

> `labels = getlab(w);` before the mapping, or
> `labels = getlab(a*w);` after the dataset `a` is mapped by `w`.

Mappings can be learned from examples, (labeled) objects stored in a dataset `a`, for instance by training a classifier:

> `w3 = ldc(a);` the normal densities based linear classifier
> `w2 = knnc(a,3);` the 3-nearest neighbor rule
> `w1 = svc(a,'p',2);` the support vector classifier based on a 2nd order
> polynomial kernel

The mapping of a test set `b` by `b*w1` is now equivalent to `b*(a*v1)` or even, irregularly but very handy to `a*v1*b` (or even `a*ldc*b`). Note that expressions are evaluated from left to right, so `b*a*v1` may result in an error as the multiplication of the two datasets (`b*a`) is executed first.

D.7 Example

In this example a 2D Highleyman dataset `A` is generated, 100 objects for each class. Out of each class 20 objects are generated for training, `C` and 80 for testing, `D`. Three classifiers are computed: a linear one and a quadratic one, both assuming normal densities (which is correct in this case) and a Parzen classifier. Note that the data generation use the random generator. As a result they only reproduce if they use the original seed. After computing and displaying classification results for the test set a scatterplot is made in which all classifiers are drawn.

```
%PREX_PLOTC  PRTools example on the dataset scatter and classifier plot
help prex_plotc
echo on
                % Generate Higleyman data
A = gendath([100 100]);
                % Split the data into the training and test sets
[C,D] = gendat(A,[20 20]);
                % Compute classifiers
w1 = ldc(C);        % linear
w2 = qdc(C);        % quadratic
w3 = parzenc(C);    % Parzen
w4 = lmnc(C,3);      % neural net
                % Compute and display errors
                % Store classifiers in a cell
W = w1,w2,w3,w4;
                % Plot errors
disp(D*W*testc);
                % Plot the data and classifiers
figure
                % Make a scatter-plot
scatterd(A);
                % Plot classifiers
plotc(w1,w2,w3,w4);
echo off
```

Appendix E

Introduction to datafiles

Robert P.W. Duin and David M.J. Tax

E.1 Introduction

Since the introduction of PRTOOLS 4.1 the toolbox is extended to avoid some of the limitations that were imposed by the use of the `prdataset` object. The first limitation of the dataset object is that all objects in a dataset should have the same (number of) features. In particular when one starts with raw data, for instance when one is using images or timeseries with different sizes, the processing has to be done outside PRTOOLS. A second limitation is that datasets can become very large. So large in fact, that the dataset object cannot be stored in the memory. To remove these limitations the `prdatafile` object is introduced.

A datafile in PRTOOLS is a generalization of the dataset concept, and they inherit most of the properties of a dataset. The datafile allows the distribution of a large dataset over a set of files, stored on disk. Datafiles are mainly an administration about the files and directories in which the objects are stored. A datafile is, like a dataset, a set consisting of M objects, each described by k features. k may be unknown and varying over different objects, in which case it is set to zero.

Almost all operations defined for datasets are also defined for datafiles, with a few exceptions. Operations on datafiles are possible as long as they can be stored (e.g. filtering of images for raw datafiles, or object selection by `gendat`). Furthermore, commands that are able to process objects sequentially, like `nmc` and `testc` can be executed on datafiles. The use of untrained mappings in combination with datafiles is a problem, as they have to be adapted to the sequential use of the objects. Mappings that can handle datafiles are indicated in the Contents file.

Different types of datafiles are defined:

raw Every file is interpreted as a single object in the dataset. These files may, for instance, be images of different size. It is assumed that objects from different classes are stored each in a different directory. The directory name is used as the class label.

cell All files should be mat-files containing just a single variable being a cell array. Its elements are interpreted as objects. The file names will be used as labels during construction. This may be changed by the user afterwards.

pre-cooked In this case the user should supply a command that reads the file and converts it to a dataset. More than one object may be stored in a single file. Furthermore, the command should define the class labels for the objects stored in the dataset.

half-baked All files are mat-files, containing a single labeled dataset.

mature This is a datafile created by PRTOOLS, using the `savedatafile` command after execution of all preprocessings defined for the datafile.

The most natural way for a user to create a datafile, is using the option **raw**.

Whenever a raw datafile is sufficiently defined by pre- and postprocessing it can be converted into a dataset. If this is still a large dataset, not suitable for the available memory, it should be stored by the `savedatafile` command and is ready for later use. If the dataset is sufficiently small it can be directly converted into a dataset by `prdataset`.

E.2 Operations on datafiles

The main commands specific for datafiles are:

`prdatafile`	constructor. It defines a datafile on a directory.
`addpreproc`	adds preprocessing commands (low level command)
`addpostproc`	adds postprocessing commands (low level command)
`filtm`	user interface to add preprocessing to a datafile.
`savedatafile`	executes all defined pre- and postprocessing and stores the result as a dataset in a set of matfiles.
`prdataset`	conversion to dataset

Most functions that were defined in MEASTOOLS are also available as mappings that operate on datafiles. In particular, all examples as they were defined in the MEASTOOLS appendix work in an idential way:

```
% load digits as measurement variables
>> a = load_nist([5,9],[1:20]); figure; show(a)
% add rows and columns to create square figure (aspect ratio 1)
>> b = im_box(a,[],1);          figure; show(b)
% resample by 16 x 16 pixels
>> c = im_resize(b,[16,16]);    figure; show(c)
% compute means
>> d = im_mean(c);
% convert to PRTools dataset and display
>> d = prdataset(d,[],'featlab',char('mean-x','mean-y');
>> scatterd(x,'legend');
```

E.3 Image Processing

Before measurering features, images might have to be preprocessed. PRTOOLS contains a series of `mapping` routines to facilitate this. If the desired routine is not available, they can easily be built from standard MATLAB or `DipImage` routines using `filtm`.

mappings for image processing	
im_gray	Convert any image to a gray-value image
im_invert	Invert an image by subtracting it from its maximum
im_gaussf	Gaussian filter
im_minf	Minimum filter
im_maxf	Maximum filter
im_stretch	Grey-value stretching
im_hist_equalize	Histogram equalization
im_threshold	Thresholding
im_berosion	Binary erosion
im_bdilation	Binary dilation
im_bpropagation	Binary propagation
im_label	Label binary image
im_select_blob	Selection of largest blob in a binary image
im_box	Bounding box for binary image
im_center	Center binary image in center of gravity
im_resize	Resize images
im_scale	Scale binary image
im_rotate	Rotate image
im_fft	Image FFT
im_cols	Conversion of full color images to 3 grey value images

A number of more specific routines for measuring features in images are available:

prfilters for features of images	
im_moments	Computes various image moments, central, Hu and Zernike.
im_mean	Computes just the centre of gravity.
im_profile	Computes the horizontal and vertical image profiles.
im_measure	Measures various object features.

The last routine, im_measure, computes a single set of features. If the image contains a set of objects, it has first to be split by im2meas into images containing a single object, in order to obtain a measurement set in which each object refers to a single physical object. The im_measure-routine makes use of the DipImage package. The user has to take care that this is loaded.