A mi padre, Raúl:
campesino, agricultor, filósofo innato,
quien me heredó su fascinación por el conocimiento.

A mi hija, Manuela,
a quien se la heredaré yo.

ACKNOWLEDGMENTS

This book would not have been possible without the support, first, of the director of the "*Carrera de Ingeniería de Sistemas*" of *Universidad Politécnica Salesiana*, Patsy Prieto, and her whole council, who trusted me and sponsored the project even when it was nothing but a dream flitting around in my head. I also want to thank the authorities of this university for the financial support that made it possible for this book to be not only good and useful from a technical point of view (hopefully), but also beautiful (hopefully, too), due to the great deal of attention that was paid to the aesthetical aspects of its edition. Among these authorities, I owe a special acknowledgement to Jorge Altamirano, who personally advocated for the granting of the funds to hire the specialised professionals that were involved in this "embellishment" of the book.

I am also grateful to the students of the course Artificial Intelligence II, taught by me, who served as "guinea pigs" in this large experiment and gave invaluable feedback about the exercises of this book, helping to perfect them semester after semester. Somehow representing those students, my teaching assistants Gabriel Navas, Sergio Palacios and Jairo Castellano did a great job at writing down my drafts in LaTeX.

My thanks also to those people who selflessly revised this book (without an economic reward, I should say): Bayardo Campuzano and Julio Proaño in the academic aspect; and Michael Byrd, in the grammatical. Thanks to another selfless talent, Claudia Narváez, I have the nice picture of me in the book.

Finally, I would like to thank Andrés Merino, Daniela Andrade and Andrés Hinojosa for their professionalism in doing their job, so important for the above-mentioned aesthetical aspect of the book. And Karina Riera, the love of my life, who chose the colours of the layout in order for them to be agreeable to the female readers of the book.

Contents

Preface

This book had humble origins: it began as a collection of exercises that I had designed for my students of the course *Artificial Intelligence II*, at Universidad Politécnica Salesiana. Why did I bother to create exercises on a subject such widely dispersed as Machine Learning, for which there was such an abundance of material already available? Well, as a teacher I have always compared myself to a trainer. A sport trainer. This kind of guy does not teach you to swim by showing you how to do it while you observe from the bench. He or she gets you into the water.

So that is what I do with my students. I get them into the water. For doing that, however, I need exercises. The kind of exercises you find in any standard book of Calculus, for instance. These nice books are made up of a rather small theoretical section (that almost no student reads) followed by many exercises, which constitute the very core of the chapter. These exercises have straightforward instructions, which lead to unique, unambiguous answers. Fortunately, the book usually includes those answers, so that the students can check their own progress and detect any misunderstanding on time.

The question is: what happened to that good order of things? Strangely enough, when you grow up and pass to increasingly advanced courses, you find that your books differ more and more from that structure. Their theoretical sections get larger and larger, and they dedicate just a couple of pages to the exercises. Besides, these exercises do not have straightforward and unambiguous answers. Sometimes, they only ask for open tasks such as: "think of another algorithm to classify this or that".

This kind of exercises does not work for me as a trainer. That is why I decided to create my own ones, exercises with the same spirit of those of our old Calculus books. And when I had a large enough amount of such exercises, I thought: "wait a minute, why don't we put them together?" And that is how this book was born.

So, this is a book for trainers and their pupils. That is why it contains the word "workout" in its title. It is not meant to show you much theory, you are supposed to know most of it already –maybe from your teacher, your favourite web course or whatever. It is a book for undergraduates, and therefore it requires only a shallow understanding of the maths behind the algorithms.

Each chapter in this book is structured in the following way: first, a short, concise theoretical section, the *Theoretical Briefing* (which probably no student will read); then, the Exercises, a section designed to test your comprehension of the concepts and the mechanics of the algorithms by means of desk checks and stuff like that; and finally the *Practical in MATLAB*, where the problems to be solved by programming (i.e., the most important part of each chapter) reside. In most of the cases, to solve these sections you will need some data, which you will find in the digital files accompanying this book.

Oh, and very importantly: at the end of each chapter you will find the answers to the problems proposed in the *Exercises and Practical in MATLAB* sections. In this way, you will be able to track your progress properly – just like in the good old days!

Holger Ortega
Quito, 2019

PART I

GENERALITIES

This is probably the most important part of the book, especially for the uninitiated. In this part, we introduce the basics of the MATLAB language, focusing on those aspects that will be relevant to the development of the *Practical* sections. Also, you will see the very fundamentals of Machine Learning here: what it does, what kind of data it uses, how this data must be organised, what types of algorithms there are, what they produce and how to evaluate their results.

The introduction to MATLAB will be treated in Chapter 1, in such a way that even if you have not used this language before, you can feel quite confident about your skills for the coming challenges after mastering the material. In Chapter 2, you will learn to code the information from the real world into matrices and vectors, the raw material with which MATLAB works. Knowing this should allow you to use Machine Learning libraries and built-in functions even without understanding them in depth, which makes this chapter very important from a practical point of view. In Chapters 3 and 4, you will know about two fundamental types of problems: *classification* and *regression*. You will not see the algorithms involved, you will only learn the concepts and the metrics to evaluate the results of such algorithms.

Chapter 1

Matrices and MATLAB

1.1 Theoretical Briefing

Matrices. A matrix is an array of numbers. An element (or "entry") of a matrix $\mathbf{A}$ is represented by the symbol a_{ij}, where i is the row and j is the column in which the entry is located.

Dimensions. The dimensions of a matrix are given by the number m of rows and the number n of columns. We then say that the dimensions of the matrix are $m \times n$.

Vector. A vector is a matrix of size $d \times 1$. The number d is called the **dimension** of the vector.

Matrix addition. The addition of two matrices $\mathbf{A}$ and $\mathbf{B}$ is a matrix $\mathbf{X}$ where

$$x_{ij} = a_{ij} + b_{ij} \tag{1.1}$$

for all the values of i and j. We then say that $\mathbf{X} = \mathbf{A} + \mathbf{B}$.

Matrix multiplication. The multiplication of two matrices $\mathbf{A}$ and $\mathbf{B}$ is a matrix $\mathbf{X}$ where

$$x_{ij} = \sum_{k=1}^{m} a_{ik} \times b_{kj} \tag{1.2}$$

for all the values of i and j. We then say that $\mathbf{X} = \mathbf{A} \times \mathbf{B}$.

Transpose. The transpose of a matrix $\mathbf{A}$ (denoted by $\mathbf{A}^T$) is a matrix $\mathbf{X}$ where

$$x_{ij} = a_{ji} \tag{1.3}$$

for all the values of i and j.

MATLAB. MATLAB is a programming language specialised in matrix handling. Actually, its name is short for “Matrix Laboratory”. Any basic variable in MATLAB is considered as a matrix.

Matrix creation. You can use the MATLAB functions `ones`, `zeros`, `eye`, `randi`, `magic`, etc, to create matrices in MATLAB. Also, you can use a manual syntax. For example, to create a 3×3 unitary matrix `A`, use

```
>> A = [1 0 0;0 1 0;0 0 1];
```

It is equally correct if you use commas instead of spaces:

```
>> A = [1,0,0;0,1,0;0,0,1];
```

It is possible to build matrices from other matrices using the same syntax, but with the names of the matrices instead of simple numbers. For instance,

```
>> B = [A;A];
```

Accessing and assigning values in a matrix. For accessing an element in a matrix `A`, use the syntax `A(i,j)`. For instance,

```
>> A(3,2)
```

would return the number 0. For assigning a value, use, for instance:

```
>> A(3,2) = 8;
```

Matrix operations in MATLAB. You can add or multiply two matrices by using the operators “+” and "∗", respectively. Also, a new kind of operation, the **element-wise** operation, is defined in MATLAB. A regular operation is transformed in an element-wise operation by appending a dot (.) to the left of the normal operator. For example,

```
>> X = A.*B;
```

computes the element-wise multiplication between matrices **A** and **B**. This operation will produce a matrix **X** of the same size as **A** and **B**, such that:

$$x_{ij} = a_{ij} \times b_{ij} \tag{1.4}$$

for every i and j.

Control sentences. As in any programming language, there are control sentences in MATLAB. You can find out the syntax and further information about statements such as `for`, `if` and `while` by using the command `doc`. For example, type

```
>> doc while
```

in the *Command Window* to know more about the `while` sentence.

Functions. Functions can be written and saved as independent files in MATLAB. A function file has the extension ".m". To know more about functions, type

```
>> doc function
```

Scripts. A script is a piece of code which can be saved as a file. A script file has also the extension ".m"

Built-in Functions in MATLAB. There are many built-in functions (this is, functions that come with MATLAB), ready to be used by you. We have menctioned several built-in functions in past paragraphs. As another example, take function `sum`. This command will take a matrix and return a row vector containing the sum of elements in each column. Or, if you enter a vector, `sum` will return a simple number, the sum of all elements in the vector.

Documentation. You can access the documentation in general by typing the command `doc` followed by the word you want to search about in the *Command Window*. For instance,

```
>> doc sum
```

1.2 Exercises

Solve the following exercises. You can use a calculator. Do not use MATLAB here.

1. Let **A** and **B** be the matrices

$$\mathbf{A} = \begin{bmatrix} 2 & 1 & -4 & -4 & -2 \\ 2 & -3 & -2 & 3 & 4 \\ -1 & 2 & -4 & 2 & -4 \end{bmatrix}$$

and

$$\mathbf{B} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 2 & 1 \\ 2 & 2 \\ -1 & 2 \end{bmatrix}.$$

And let **C** be the result of the matrix product **A** × **B**.

(a) What should be the size of matrix **C**? (You must be able to tell this even before computing such matrix.)

(b) Compute matrix **C**.

2. Let **D** be the matrix

$$\mathbf{D} = \begin{bmatrix} -1 & 2 & -2 \\ 2 & -1 & 1 \end{bmatrix}.$$

Compute $\mathbf{C}^T + \mathbf{D}$.

3. Let **A** and **B** be the matrices

$$\mathbf{A} = \begin{bmatrix} 2 & 1 & -4 & -4 & -2 \\ 2 & -3 & -2 & 3 & 4 \\ -1 & 2 & -4 & 2 & -4 \end{bmatrix}$$

and

$$\mathbf{B} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 2 & 1 \\ 2 & 2 \\ -1 & 2 \end{bmatrix}.$$

And let **C** be the result of the matrix product **A** × **B**.

(a) What should be the size of matrix **C**? (You must be able to tell this even before computing such matrix.)

(b) Compute matrix **C**.

4. Let **D** be the matrix

$$\mathbf{D} = \begin{bmatrix} -1 & 2 & -2 \\ 2 & -1 & 1 \end{bmatrix}.$$

Compute $\mathbf{C}^T + \mathbf{D}$.

5. Let **A** and **B** be the matrices

$$\mathbf{A} = \begin{bmatrix} 2 & 1 & -4 & -4 & -2 \\ 2 & -3 & -2 & 3 & 4 \\ -1 & 2 & -4 & 2 & -4 \end{bmatrix}$$

and

$$\mathbf{B} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 2 & 1 \\ 2 & 2 \\ -1 & 2 \end{bmatrix}.$$

And let **C** be the result of the matrix product **A** × **B**.

(a) What should be the size of matrix **C**? (You must be able to tell this even before computing such matrix.)

(b) Compute matrix **C**.

6. Let $\mathbf{D}$ be the matrix

$$\mathbf{D} = \begin{bmatrix} -1 & 2 & -2 \\ 2 & -1 & 1 \end{bmatrix}.$$

   Compute $\mathbf{C}^T + \mathbf{D}$.

7. If you wrote the code

```
>> C.*D'
```

   in MATLAB, what would be the result?

8. Let $\mathbf{e}$ and $\mathbf{f}$ be the vectors

$$\mathbf{e} = \begin{bmatrix} 9 \\ -3 \\ 2 \\ -5 \end{bmatrix}$$

   and

$$\mathbf{f} = \begin{bmatrix} 5 \\ -5 \\ 0 \\ 4 \end{bmatrix}.$$

   And let $\mathbf{g}$ be the result of the operation $\mathbf{e}^T \times \mathbf{f}$.

   (a) What should be the size of "matrix" $\mathbf{g}$? (You must be able to tell this even before computing $\mathbf{g}$.)

   (b) Compute $\mathbf{g}$.

9. If you wrote the code

```
>> sum(e.*f)
```

   in MATLAB, what would be the result?

1.3 Practical in MATLAB

General Objective:

To introduce to the student the fundamentals of the MATLAB language.

Specific Objectives:

- The student should be able to create, manipulate and operate matrices in MATLAB.
- The student should be able to use the main flow control statements (`if` and `for` statements) in MATLAB.
- The student should be able to write MATLAB scripts and functions, and to call functions from the scripts.
- The student should be able to access information from files and variables in MATLAB.

Materials:

Computer, MATLAB.

Procedure:

Solve the following exercises in MATLAB

1. Create a script and save it following the format `Pr1_nameLastname.m`. You will write your code for the next numerals in this script.

2. Use the MATLAB function `load` to load into the *Workspace* the matrices contained in the file `matricesPr1.mat`. This file can be found in the folder *Data Sets* and should be previously saved in the *Current Folder* you are working in.

3. Create a 100 × 21 matrix (which you will call "`A`") with the following features: its first column must contain the integers from 1 to 100. Its last three columns must be filled with numbers "8". The element of the 85^{th} row, 12^{th} column, must be -100. The rest of the entries must be zero. (*Hint*: Use the MATLAB functions `zeros` and `ones` for this part. The colon (:) operator could also be useful.)

4. Create a 21 × 500 matrix "`E`" as follows: take the `B` and `C` matrices loaded in numeral 2 and put one below the other (`C` below `B`) to form a 21 × 450 matrix "`bc`". Transpose matrix `D` (also loaded in numeral 2) and put it at the left of the `bc` matrix to form matrix `E`.

5. Compute the matrix multiplication between `A` and `E` and store it in a variable named "`F`". If everything is going right, this variable should be a 100 × 500 matrix.

6. Search for the element in the 54^{th} row, 374^{th} column of matrix `F`:

   (a) By opening the matrix, double-clicking its name in the *Workspace*

   (b) By typing in the *Command Window* the corresponding code to access elements.
   What is the value stored in this entry?

7. Create a function "`myTriplicator(M,A)`" which takes two equally-sized matrices `M` and `A`, and returns a matrix `X` also the same size. This `X` matrix should contain the same elements of matrix `A` (in some cases) and the triple of the elements (in the rest of the cases). Matrix `M` is kind of a "mask" matrix, and it tells the function if it should triple a given element of `A` (depending on whether the corresponding entry in `M` is 1 or 0). For

example, let **M** and **A** be the matrices:

$$\mathbf{M} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix}$$

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

then **X** should be:

$$\mathbf{X} = \begin{bmatrix} 3 & 2 & 3 \\ 4 & 15 & 18 \end{bmatrix}$$

If matrices `M` and `A` are not the same dimensions, the function should just return a message:

```
>> ERROR: Matrix dimensions inconsistent!
```

Write this function:

(a) Using `for` statements.

(b) WITHOUT using them. (*Hint*: use the element-wise product of MATLAB.)

8. Call your function `myTriplicator` from the script, passing as arguments the matrices `M` (the one loaded in numeral 2) and `F` (created in numeral 5). Store the result in a variable called "`X_8`".

9. Use the MATLAB function `sum` to compute the sum of all the elements of matrix `X_8`.

10. Create a function "`extractEquals(A,B)`" which takes two equally-sized matrices `A` and `B`, and returns a matrix `X` also the same size. For each of its entries, this matrix `X` should contain a 0 if the corresponding entries in matrices `A` and `B` are different from each other. If these entries are equal, the matrix `X` should contain that repeated number. For example, let **A** and **B** be:

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

$$\mathbf{B} = \begin{bmatrix} 1 & 8 & 9 \\ 7 & 5 & 4 \end{bmatrix}$$

In this case, function extractEquals would return:

$$\mathbf{X} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 5 & 0 \end{bmatrix}$$

If matrices M and A are not the same size, the function should just return a message:

```
>> ERROR: Matrix dimensions inconsistent!
```

Write this function:

(a) Using for statements.

(b) WITHOUT using them. (*Hint*: use matrix subtraction to identify where the elements are equal. The colon (:) operator and the reshape function could also be useful.)

11. Call your function extractEquals from the script, passing as arguments the matrices F (created in numeral 5) and G (loaded in numeral 2). Store the result in a variable called "X_11".

12. Use the MATLAB function sum to compute the sum of all the elements of matrix X_11.

13. How many non-zero elements are there in matrix X_11? (You can use the built-in function nnz here.)

1.4 Answers to selected exercises

Exercises

1. (a) 3×2

   (b) $\begin{bmatrix} -14 & -16 \\ -2 & 12 \\ 0 & -8 \end{bmatrix}$

2. $\begin{bmatrix} -15 & 0 & -2 \\ -14 & 11 & -7 \end{bmatrix}$

3. $\begin{bmatrix} 14 & -32 \\ -4 & -12 \\ 0 & -8 \end{bmatrix}$

4. (a) 1×1

   (b) 40

5. 40

Practical

6. 788

7. Solution without using `for` statements:

```
function X = myTriplicator(M,A)
if ~isequal(size(M),size(A))
    fprintf('ERROR: Matrix dimensions inconsistent!\n');
    return
end
X = A + 2*(M.*A);
```

9. −3557364

10. Solution without using `for` statements:

```
function X = extractEquals(A,B)
if ~isequal(size(A),size(B))
    fprintf('ERROR: Matrix dimensions inconsistent!\n');
    return
end
C=A-B;
indexes = C~=0;
A=A(:);
A(indexes)=0;
X=reshape(A,size(B));
```

12. −174139

13. 5060

CHAPTER 2

DEFINITIONS AND DATA REPRESENTATION

2.1 Theoretical Briefing

Intelligent systems. An intelligent system is any device (such as a brain or a computer) which processes data from its environment to obtain an output. The more sophisticated and useful the output, the more intelligent the system deserves to be considered.

Feature vectors. The data from the environment of the intelligent system (the "input") is caught by means of the senses (such as an eye or a camera). We will dare here to assert that any piece of information that enters the system as an input (an image, a sound, a fragrance, etcetera) can be written down as a vector. In the Machine Learning argot, such a vector is called a **feature vector**.

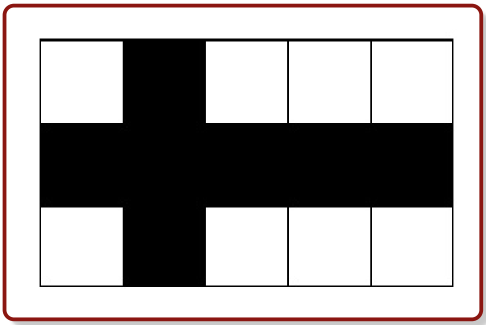


FIGURE 2.1: A simple pixel-made image

For example, in Figure 2.1 we show the image of a black cross. This image has 3×5 pixels and can be represented by the following matrix $\mathbf{A}$:

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 \end{bmatrix}$$

where a 0 represents black, and the 1 represents white. Going one step further, we can take the elements of matrix $\mathbf{A}$, column by column, and put them together in a 15×1 vector $\mathbf{x}_1$ as follows:

$$\mathbf{x}_1 = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 1 \end{bmatrix}$$

This is an example of a feature vector of the image. This vector can be obtained in MATLAB by means of the following line of code:

```
>> x_1 = A(:)
```

We have introduced here the colon (:) MATLAB operator.

Features. Each element of a feature vector is known as a **feature**, or a **coordinate** of the vector. Formally, the j^{th} coordinate of a i^{th} vector will be represented in this book as:

$$x_i{}^{(j)}$$

For instance, the 3^{rd} coordinate of the $\mathbf{x}_1$ vector is:

$$x_1{}^{(3)} = 1$$

The data matrix X. Imagine you want to sell a car, so you need to know how much you can charge for your car. This price depends on several "features": the distance travelled by the car, its age, engine capacity and etcetera. As you might guess, we can assemble a feature vector with these data. Now, each car in the second-hand car market will have such a feature vector. Suppose that there are m cars in this market. We say that there are m **instances**. Then, there will be m feature vectors. We can assemble a matrix stacking these vectors, previously transposed, one over the other, as follows:

$$\mathbf{X} = \begin{bmatrix} \longleftarrow \mathbf{x}_1{}^T \longrightarrow \\ \longleftarrow \mathbf{x}_2{}^T \longrightarrow \\ \vdots \\ \longleftarrow \mathbf{x}_m{}^T \longrightarrow \end{bmatrix}$$

A matrix like this is called a **data matrix**. With the arrows we are trying to express the fact that each feature vector, transposed, is a row vector. Hence, the size of the data matrix is $m \times d$, where d is the dimension of each feature vector. This is the standard Machine-Learning way to represent data.

The y vector. In the last example, the variable "price of a car" depends on the features of the car. For this reason, such a variable is called a **dependent variable**. Each car in the market will have a price, and we can assemble a **y** vector with these m values as follows:

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}$$

Regression and classification problems. In principle, the price of a car can take *any* real value from 0 to ∞; in other words, it is a continuous variable. The problems dealing with continuous variables y_i are called **regression problems**. On the other hand, consider the task of classifying images of the vowels. In this case, we would have five classes, which can be labelled as follows: "1" for *a*, "2" for *e*, "3" for *i* and so on. In this example, each feature vector $\mathbf{x}_i$ (which we can extract from the image in the standard Machine-Learning way) is associated with a y_i value which cannot take any real value in an interval. Instead, this y_i can take only the discrete values 1, 2, 3, 4 or 5. Problems like this one, dealing with discrete variables y_i corresponding to classes, are called **classification problems**.

Training and test. Suppose that you see a symbol like this:

M

Automatically, your brain recalls its name and associates it with a sound, does it not? Nevertheless, there was a time when your brain was not able to do so.

You needed to go through a period of "training" to achieve such a capability. In this period, another intelligent system (your school teacher) used to show you several different images of the same symbol, in different positions, sizes and fonts, while repeating the name and sound of "M" at the same time. This very procedure was repeated for the other letters, generally mixing them up to increase the difficulty of the task. Similarly, we could "train" an artificial intelligent system by using a data matrix $\mathbf{X}$ containing the feature vectors of many images of letters, and a $\mathbf{y}$ vector of the corresponding labels. Together, this $\mathbf{X}$ matrix and its $\mathbf{y}$ vector are called the **training set**. Hopefully, after this training the intelligent system will be able to produce an output (or **prediction**) more or less accurate. These predictions are named $\widehat{y}_1, \widehat{y}_2, \ldots, \widehat{y}_m$. Usually, we collect the predictions in a vector $\widehat{\mathbf{y}}$ as follows:

$$\widehat{\mathbf{y}} = \begin{bmatrix} \widehat{y}_1 \\ \widehat{y}_2 \\ \vdots \\ \widehat{y}_m \end{bmatrix}$$

Also, a tougher challenge can be conceived: to make the system predict the labels for *new* instances, which it has never seen before. This is, instances that were not in the training set. This new $\mathbf{X}$ matrix, together with its $\mathbf{y}$ vector, is called the **test set**.

Visualisation. When the feature vectors in a dataset have one, two or three dimensions, they can be visualised in a Cartesian coordinate system. You will need a straight line, a plane, or a three-dimensional space in each case. To visualise a dataset in MATLAB, use the functions `scatter` or `scatter3`.

2.2 Exercises

Solve the following exercises. You can use a calculator. Do not use MATLAB here

Let $\mathbf{X}$ be the matrix

$$\mathbf{X} = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 12 & 11 & 10 & 9 & 8 & 7 \\ -1 & -2 & -3 & -4 & -5 & -6 \\ -7 & -8 & -9 & -10 & -11 & -12 \end{bmatrix}$$

let $\mathbf{A}$ be the matrix

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & -1 \end{bmatrix}$$

and let $\mathbf{X}_{signs}$ be the matrix

$$\mathbf{X}_{signs} = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \end{bmatrix}$$

1. What is matrix **B** which would result from running the following code in MATLAB?

```
>> g = X(:,5);
>> B = reshape(g,2,2);
```

2. What is matrix **C** which would result from running the following code in MATLAB?

```
>> h = X(3,:);
>> C = reshape(h,3,2);
```

3. What is matrix **D** which would result from running the following code in MATLAB?

```
>> j = A(:);
>> D = reshape(j,5,2);
```

4. Matrix $\mathbf{X}_{signs}$ contains the images of "plus" (+) and "minus" (−) signs. Originally, these images were in the form of 3 × 3 matrices, whose elements were then taken column by column to form the feature vectors, in the standard Machine-Learning way. Write down the **y** vector of labels, using a "0" for class "minus" and a "1" for class "plus".

2.3 Practical in MATLAB

General Objective:

To introduce to the student the standard organisation of data in Machine Learning.

Specific Objectives:

- The student should be able to retrieve simple information from data.
- The student should be able to store information in form of matrices, as usual in Machine Learning.
- The student should be able to create plots of low-dimensional data and qualitatively interpret the plots obtained.

- The student should be able to load information from an Excel file into MATLAB and analyse it.

Materials:

Computer, MATLAB.

Procedure:

Solve the following exercises in MATLAB

1. Create a script and save it following the format `Pr2_nameLastname.m`. You will write your code for the next numerals in this script.
2. Load into the *Workspace* the matrices contained in the file `matricesPr2.mat`. This file can be found in the folder *Data Sets*.
3. One of the matrices loaded in the previous numeral is called `Xnum` and contains the images of four numbers. Each of these images was originally a 5×4 matrix that has been transformed into a feature column vector and then transposed, as explained in the theory. Use the `reshape` function to get the matrices assembled again and visualise them using `imshow`. Write down the `y` vector corresponding to this `Xnum` matrix. (Use as labels the same numbers you see when visualising the images)
4. Now you will do quite the opposite. Matrices `im1`, `im2` and `im3`, loaded in numeral 2, are images of the letters *i*, *u* and *c*. Assemble an "`Xletters`" matrix with them, using the standard procedure described in the theory. If you have done everything right, `Xletters` should be a 3×30 matrix.
5. Use `sum` to compute the sum of elements of each row of matrix `Xletters`. This step is meant to check if everything has been done right.
6. Visualise the data contained in matrices `X1` and `X2`, loaded in numeral 2. Use the MATLAB commands `scatter` and `scatter3` for this. The plots you must obtain are shown in Figures 2.2 and 2.3
7. In folder *Data Sets* you will find a file named `cars.xlsx`. This file contains the real information the author of this book collected to buy a second-hand car in 2016. This information includes (as you can see by opening the file in Excel): distance travelled, age, engine capacity and price. Save `cars.xlsx` to the *Current Folder* and load it using `xlsread`. Store the data in a variable you will call "`carData`".

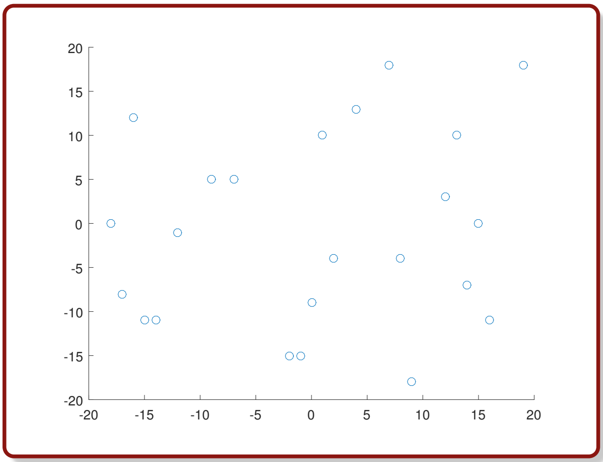


FIGURE 2.2: The data in X1

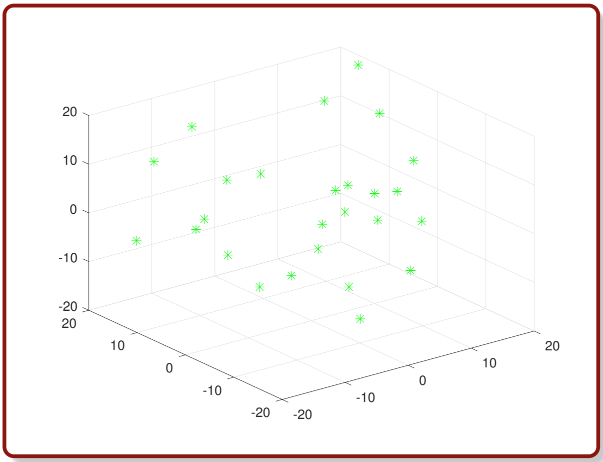


FIGURE 2.3: The data in X2

8. Form an "Xcars" matrix containing the features of the cars. We will consider here as features the distance travelled, the age and the engine capacity of the cars. If everything was done right, you should end up with a 16 × 3 matrix in this step.

9. Extract an "ycars" vector containing the labels of the cars. We will consider here as label (or, rather, as the independent variable) the price of the cars. If everything was done right, you should end up with a 16 × 1 vector in this step.

10. Visualise the data contained in `Xcars`. Use `subplot` to show both the 3-D representation and a view from "above". You can plot this view using the `view` function. Add suitable labels to the plots. The plot you are meant to obtain is shown in Figure 2.4.

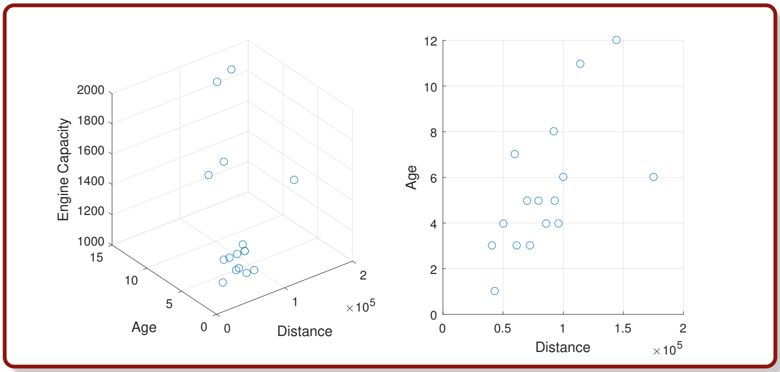


FIGURE 2.4: The data in `Xcars`

11. Visualise the labels (or independent variable) versus the first feature of the data. Add suitable labels to the plot. The plot you are meant to obtain is shown in Figure 2.5 . By looking at this plot, you should be able to answer the following questions: which point corresponds to the cheapest car? Which to the less-travelled car? Which to the most expensive and which to the most travelled?

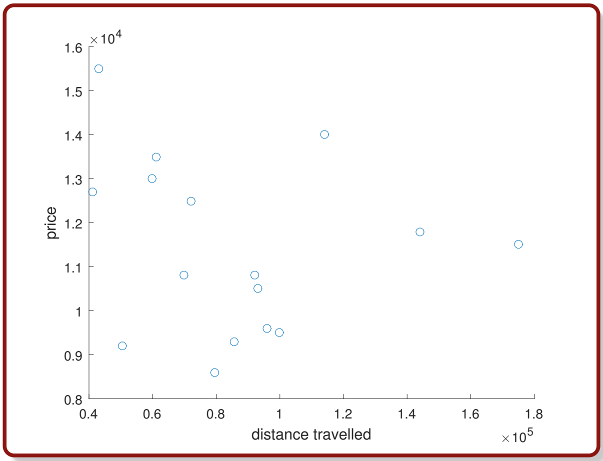


FIGURE 2.5: The independent variable versus a feature of the cars

2.4 Answers to selected exercises

Exercises

1. $\mathbf{B} = \begin{bmatrix} 5 & -5 \\ 8 & -11 \end{bmatrix}$

2. $\mathbf{C} = \begin{bmatrix} -1 & -4 \\ -2 & -5 \\ -3 & -6 \end{bmatrix}$

3. $\mathbf{D} = \begin{bmatrix} 1 & 8 \\ 6 & 4 \\ 2 & 9 \\ 7 & 5 \\ 3 & -1 \end{bmatrix}$

4. $\mathbf{y} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$

Practical

3. y= $\begin{bmatrix} 3 \\ 1 \\ 4 \\ 2 \end{bmatrix}$

5. Sum of elements of first row: 3; second row: 9; third row: 8.

Chapter 3

Classification Problems

3.1 Theoretical Briefing

Classification problems. A problem in which the $\mathbf{y}$ vector consists of class labels c_j is called a "classification problem". This is,

$$y_i \in \{c_1, c_2, \ldots, c_k\}; \quad i = 1, 2, \ldots, m$$

where k is the number of classes and m is the number of instances.

An example of a classification problem is to classify a set of images as those belonging to men and those belonging to women, a task performed by the human brain all the time. In order to mathematise this problem, we need to assign a numerical value to each class (say, "1" for women and "2" for men, or whatever other way). These values are called the class *labels*.

Misclassification error. This is the most basic way of evaluating the performance of a classifier. Let $\widehat{\mathbf{y}}$ be the vector of predictions and $\mathbf{y}$ the vector of actual labels. The misclassification error is defined by

$$\varepsilon = \frac{1}{m} \sum_{i=1}^{m} I(\widehat{y}_i \neq y_i)$$

where $I(x)$ is the indicator function,

$$I(x) = \begin{cases} 1 & \text{if } x \text{ is true} \\ 0 & \text{if } x \text{ is false} \end{cases}$$

In other words, the misclassification error is simply the rate of misclassified instances.

Confusion matrix. This is a more sophisticated measure of the performance of a classifier. We explain what a confusion matrix is by means of the sketch shown in Figure 3.1.

		Predicted				
Actual		c_1	c_2	c_3	$\cdots$	c_k
	c_1					
	c_2			■		
	c_3					
	$\vdots$					
	c_k					

FIGURE 3.1

In the shaded cell, for example, you should put the number of instances of class c_2 which were incorrectly classified as class c_3.

True and false, positives and negatives. Another way of evaluating the performance of a classifier is to count the number of true positives (TP), false positives (FP), false negatives (FN) and true negatives (TN) for any class c_j. "Positive" means that the instance has been classified as being of class c_j; "negative", that it has not. "True" means that the classification was correct; "false", that it was not.

For example, a "false positive" for class "female" means that an image was classified as corresponding to a girl, when it corresponds actually to a boy.

Sensitivity and specificity. Among many other metrics for the performance of a classifier, we have the **sensitivity** and the **specificity** for the "positive" class. As their name could suggest, it is desirable to have a classifier with *high* sensitivity and specificity (the maximum value they can reach is 1), in contrast with the errors, which we expect to be as low as possible. The sensitivity and specificity are defined by Equations 3.1 and 3.2

$$\text{sensitivity} = \frac{TP}{TP + FN} \tag{3.1}$$

$$\text{specificity} = \frac{TN}{TN + FP} \tag{3.2}$$

Geometric representation. You can graphically represent a dataset for a classification problem with 1-D, 2-D or 3-D vectors, at most. For this, use different marks to "paint" the feature vectors, depending on the class given in y.

3.2 Exercises

Solve the following exercises. You can use a calculator. Do not use MATLAB here

Let $\mathbf{X}_1$ be the matrix

$$\mathbf{X}_1 = \begin{bmatrix} 2 & 7 \\ 1 & 1 \\ 1 & 10 \\ -1 & 5 \\ 9 & 3 \\ 1 & 8 \\ 0 & -1 \\ 1 & 5 \\ 6 & 3 \\ 0 & 9 \end{bmatrix}$$

of feature vectors for instances corresponding to 3 classes: (1) Planes, (2) Cars, (3) Buses. Let $\mathbf{y}_1$ and $\widehat{\mathbf{y}}_1$ be the vectors

$$\mathbf{y}_1 = \begin{bmatrix} 2 & 3 & 2 & 2 & 1 & 2 & 3 & 2 & 1 & 2 \end{bmatrix}^T$$

$$\widehat{\mathbf{y}}_1 = \begin{bmatrix} 2 & 3 & 1 & 2 & 1 & 2 & 3 & 1 & 3 & 2 \end{bmatrix}^T$$

of the actual labels corresponding to the feature vectors and the classes predicted by a certain classification algorithm, respectively.
On the other hand, let

$$\mathbf{y}_2 = \begin{bmatrix} 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}^T$$

and

$$\widehat{\mathbf{y}}_2 = \begin{bmatrix} 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}^T$$

be the real labels and the predictions, respectively, for a binary classification with classes “benign” (label: “0”) and “malignant” (label: “1”). As it is usual in medical tests, we will choose as “positive” the malignant condition (please, do not get confused by the fact that a negative condition corresponds to the “positive” result of the test!)

1. Represent, in a Cartesian plane, the feature vectors in $\mathbf{X}_1$, using different colours and markers to distinguish between points of different classes.
2. Compute the misclassification error for the predictions in $\widehat{\mathbf{y}}_1$ (with respect to $\mathbf{y}_1$, obviously).
3. Compute the confusion matrix for the classification shown in $\widehat{\mathbf{y}}_1$ (with respect to $\mathbf{y}_1$).
4. Compute the confusion matrix for the classification shown in $\widehat{\mathbf{y}}_2$ (with respect to $\mathbf{y}_2$).
5. Compute the number of true positives, false positives, false negatives and true negatives for the “positive” class (malignant).
6. Compute the sensitivity and specificity of the classification shown in $\widehat{\mathbf{y}}_2$, for the “positive” class (malignant).

3.3 Practical in MATLAB

General Objective:

To make the student capable of identifying classification problems and computing and interpreting their performance measures.

Specific Objectives:

- The student should be able to create and interpret plots of data in classification problems.
- The student should be able to compute and interpret the misclassification error.
- The student should be able to interpret confusion matrices for classification problems.
- The student should be able to compute and interpret the number of true positives, false positives, false negatives and true negatives in a classification result.

Materials:

Computer, MATLAB.

Procedure:

Solve the following exercises in MATLAB

1. Create a script and save it following the format `Pr3_nameLastname.m`. You will write your code for the next numerals in this script.
2. Load into the *Workspace* the matrices contained in the file `matricesPr3.mat`. This file can be found in the folder *Data Sets*.
3. Create a function "`plotClasses(X,y)`" which takes a matrix `X` of 2-D vectors and a `y` vector of labels, and does not return anything but draws instead a plot showing the vectors in a Cartesian plane. Each point corresponding to a feature vector should be coloured and shaped depending on its class, so each class can be identified by its colour and shape. This function must work either with 2 or 3 classes. Labels in vector `y` could be any set of numbers (by instance: 0 and 1, or -1 and 1, or 1, 2 and 3, or whatever), so `plotClasses` should be able to deal with any of these situations. (*Hint:* you can use the MATLAB function `unique` to attain this.)
4. Try your function on the matrices `X` and `y` loaded in numeral 2. The plot you should obtain is shown in Figure 3.2.

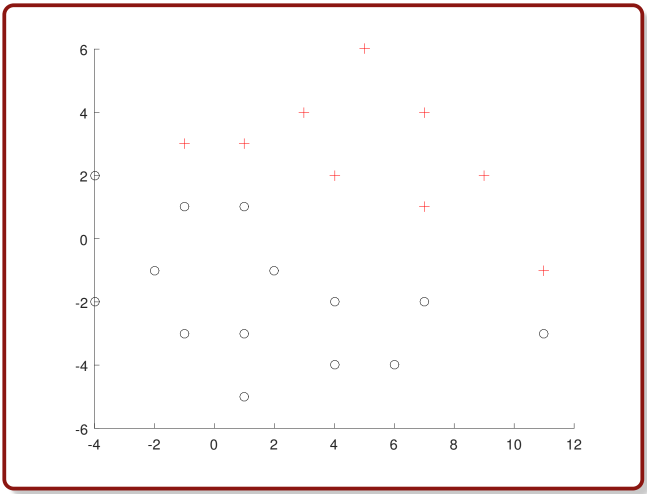


FIGURE 3.2: The data in X and y

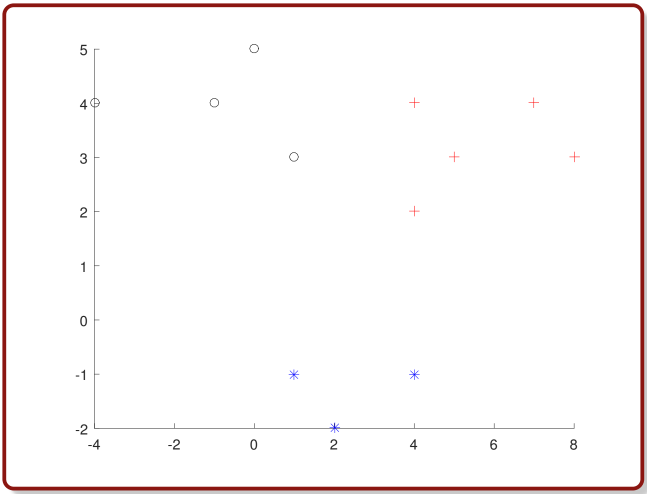


FIGURE 3.3: The data in X2 and y2

5. Try your function on the matrices X2 and y2 loaded in numeral 2. The plot you should obtain is shown in Figure 3.3.

6. In numeral 2 you loaded matrices y_animals and yhat_animals. Assume these matrices contain the actual labels and the labels predicted by a classifier, respectively. Suppose this classifier has been trained to identify three classes of animals, say dogs (class label: -1), cats (class label:

0) and rabbits (class label: 1), from images. Use the MATLAB function `confusionmat` to compute the confusion matrix of the predictions made by the classifier.

7. Write a function "`computeMCE(y,yhat)`", which takes vectors `y` (of true labels) and `yhat` (of the predictions made by a classifier) and returns the misclassification error. (*Challenge*: Do NOT use a `for` loop when writing this function.)

8. Try your function `computeMCE` on vectors `y_animals` and `yhat_animals`.

9. Write a function "`computeCPM(cM,index)`" which takes a confusion matrix "`cM`" and a number "`index`" indicating the index (not the label) of the class taken as "positive"; and computes the following classification performance metrics: true positives, false positives, false negatives, true negatives, sensitivity and specificity for this class. These six numbers must be displayed by the function in the *Command Window* (not returned, just displayed). We suggest the following format:

```
FOR CLASS INDEX c AS "POSITIVE":-------------
True positives: ---
False positives: ---
False negatives: ---
True negatives: ---
Sensitivity: ---
Specificity: ---
```

10. In numeral 2, you loaded a pair of vectors, `y3` and `yhat3`, of real and predicted labels for a classification task, respectively. Use `confusionmat` to compute the confusion matrix for these vectors.

11. Call your function `computeCPM` from the script, passing as argument the confusion matrix obtained in the previous numeral, to compute the performance metrics taking as "positive":

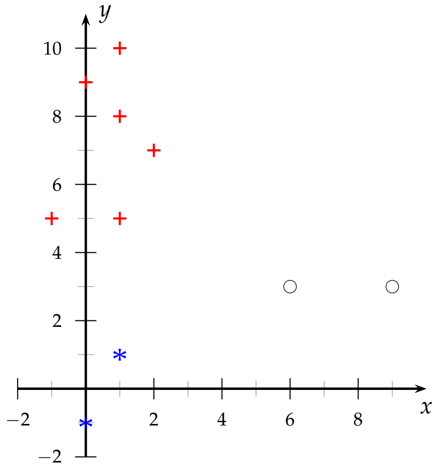
    (a) Class "-1" (you should pass index 1 for this);

    (b) Class "1" (index 2)

3.4 Answers to selected exercises

Exercises

1.



2. 0.3

3. $\begin{bmatrix} 1 & 0 & 1 \\ 2 & 4 & 0 \\ 0 & 0 & 2 \end{bmatrix}$

4.

		Predicted	
		Benign	**Malignant**
Actual	**Benign**	4	2
	Malignant	1	3

5. True positives: 3
   False positives: 2
   False negatives: 1
   True negatives: 4

6. Sensitivity: 75%
   Specificity: 66.7%

Practical

6. $\begin{bmatrix} 8 & 2 & 1 \\ 3 & 9 & 2 \\ 2 & 0 & 7 \end{bmatrix}$

8. 0.294

10.

		Predicted	
		Benign	**Malignant**
Actual	**Benign**	77	3
	Malignant	5	69

11. (a)

```
FOR CLASS INDEX 1 AS ‘‘POSITIVE’’:-------------
True positives: 77
False positives: 5
False negatives: 3
True negatives: 69
Sensitivity: 9.625000e-01
Specificity: 9.324324e-01
```

(b)

```
FOR CLASS INDEX 2 AS ‘‘POSITIVE’’:-------------
True positives: 69
False positives: 3
False negatives: 5
True negatives: 77
Sensitivity: 9.324324e-01
Specificity: 9.625000e-01
```

CHAPTER 4

REGRESSION PROBLEMS

4.1 Theoretical Briefing

Regression problems. Whereas in classification problems the values in the vector $\mathbf{y}$ are class labels (hence, integer numbers), a regression problem is defined by the condition:

$$y_i \in \Re$$

Sometimes, $\mathbf{y}$ is called the "dependent variable".

Mean squared error. The standard way to evaluate the performance of a regression algorithm is by means of the mean squared error, defined by:

$$\varepsilon = \frac{1}{m}\sum_{i=1}^{m}(y_i - \widehat{y}_i)^2 \qquad (4.1)$$

4.2 Exercises

Solve the following exercises. You can use a calculator. Do not use MATLAB here

Let $\mathbf{X}_2$ be the matrix

$$\mathbf{X}_2 = \begin{bmatrix} -2 & 1 \\ 1 & -1 \\ 2 & 1 \\ 1 & 2 \\ 3 & 3 \end{bmatrix}$$

of feature vectors.

Let $\mathbf{y}_2$ be the vector

$$\mathbf{y}_2 = \begin{bmatrix} 0.9 & -0.4 & 1.7 & -1.9 & 0 \end{bmatrix}^T$$

of labels (or values of the "dependent" variable.)

And let $\widehat{\mathbf{y}_2}$ be the vector

$$\widehat{\mathbf{y}_2} = \begin{bmatrix} 1.1 & -0.4 & 1.8 & -1.8 & 0 \end{bmatrix}^T$$

of the values predicted by a certain regression algorithm for these data.

1. Represent, in a Cartesian plane, the feature vectors. Is it possible to represent the values in $\mathbf{y}_2$ in this plot?
2. Compute the mean squared error for this prediction.

Let $\mathbf{X}_1$ be the matrix

$$\mathbf{X}_1 = \begin{bmatrix} -5 \\ -1 \\ 3 \\ 7 \end{bmatrix}$$

of feature "vectors" (in this case, there is only one feature, so the vectors are one-dimensional, and so they are just scalars… don't get confused by this fact!).

Let $\mathbf{y}_1$ be the vectors

$$\mathbf{y}_1 = \begin{bmatrix} -5.2 & -5.3 & -4.3 & -3 \end{bmatrix}^T$$

of labels (or values of the "dependent" variable.)

And let $\widehat{\mathbf{y}_1}$ be the vector

$$\widehat{\mathbf{y}_1} = \begin{bmatrix} -5.59 & -4.83 & -4.07 & -3.31 \end{bmatrix}^T$$

of the values predicted by a certain regression algorithm for these data.

3. Compute the mean squared error of this prediction.
4. Represent, in the horizontal axis of a Cartesian plane, the feature vectors contained in $\mathbf{X}$.
5. Now, represent the values in $\mathbf{y}_1$ in the *vertical* axis of that same plane.
6. Considering the points in the horizontal and vertical axes as ordered pairs, locate the corresponding points in your Cartesian plane and mark them with little circles.
7. Now, represent the values in $\widehat{\mathbf{y}_1}$ in the same vertical axis of that same Cartesian plane. Pairing these new points with the x_i values in the same way as in numeral 6, locate the corresponding points and mark them with little asterisks. Note that it is possible to draw a straight line crossing all of these asterisks (this is why this is called a "linear" regression. We will study this kind of regression in the next chapter).

4.3 Practical in MATLAB

General Objective:

To introduce to the student regression problems and how to compute their performance measure (error).

Specific Objectives:

- The student should be able to create and interpret plots of data in regression problems.
- The student should be able to compute and interpret the mean squared error.

Materials:

Computer, MATLAB.

Procedure:

Solve the following exercises in MATLAB

1. Create a script and save it following the format `Pr4_nameLastname.m`. You will write your code for the next numerals in this script.

2. Load into the *Workspace* the matrices contained in the file `matricesPr5.mat`. This file can be found in the folder *Data Sets*.

3. Create a function "`plotRegression(X,y,yhat)`" which takes as arguments a matrix `X` of one-dimensional feature "vectors", a vector `y` of the values of the dependent variable, and a vector `yhat` of the predictions made by a regression algorithm. This function should not return a value, it only has to draw a Cartesian plane in which the values in `y` are plotted against the values of the feature vectors in `X`. These points are to be represented with blue circles, using the MATLAB function `scatter`. Also, the plot must include the points corresponding to the values in $\hat{y}$ against the values in X. These last points are to be represented as black asterisks (*), and must be connected by straight green lines. (*Challenge*: Do NOT use `for` loops when writing this function.)

4. Test your function `plotRegression` with the matrices "`X`", "`y`" and "`yhat1`" you loaded in numeral 2. You should get a plot like that of Figure 4.1

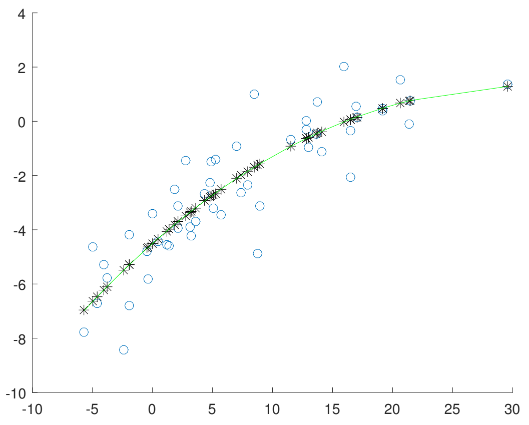


FIGURE 4.1: The regression model in `yhat1`

5. Test your function `plotRegression` with the matrices “`X`”, “`y`” and “`yhat2`” you loaded in numeral 2. You should get a plot like that of Figure 4.2

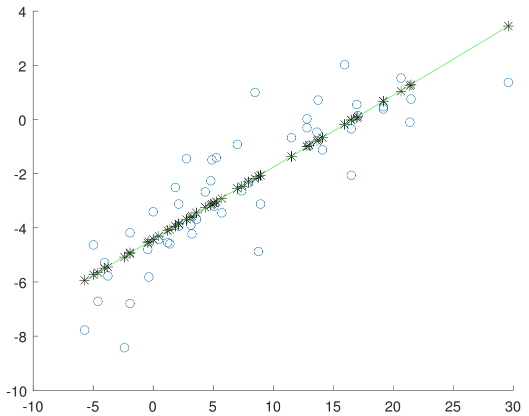


FIGURE 4.2: The regression model in `yhat2`

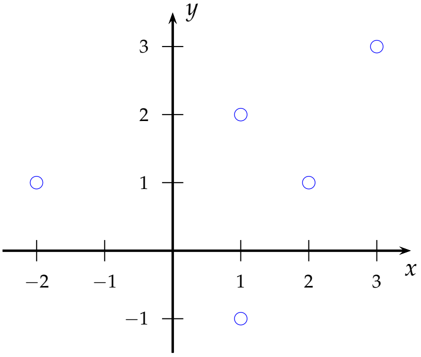
6. Write a function “`computeMSE(y,yhat)`” which takes vectors `y` and `yhat` of true and predicted labels (values of the dependent variable) respectively, and returns the mean squared error of the prediction. (*Challenge*: Do NOT use `for` loops when writing this function.)

7. Use your `computeMSE` function to calculate the mean squared error for (a) the predictions contained in $\widehat{y}_1$, and (b) the predictions contained in $\widehat{y}_2$. Which prediction is the best?

4.4 Answers to selected exercises

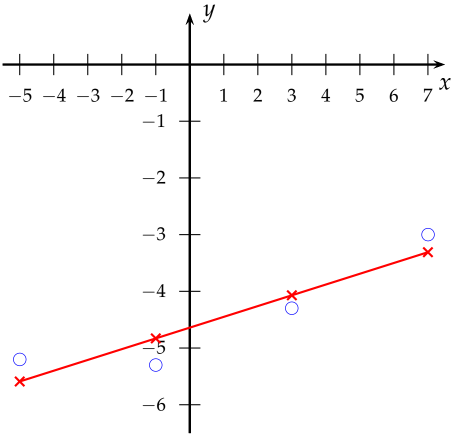
Exercises

1.



2. 0.012
3. 0.1305

7.



Practical

6. 13450

7. 15976

Hence, the prediction $\widehat{y_1}$ is better than the prediction $\widehat{y_2}$.

Part II

Regression

You will see here, at last, a real Machine Learning algorithm! Specifically, *linear regression*, a powerful algorithm that can solve even non-linear regression problems, in spite of its humble name. In Chapter 5, you will learn not only how to have MATLAB executing this algorithm for you, but also the tricks to perform non-linear regression with it. In addition, in Chapter 6 we will take advantage of the beauty of the curves that represent linear-regression solutions in the one-dimensional case, to visually show you two awful problems we deal with in Machine Learning: *overfitting* and *underfitting*.

Chapter 5

Linear Regression

5.1 Theoretical Briefing

Linear regression model. Let **X** and **y** be the data matrix and the dependent variable for a regression problem, respectively. The linear regression model is a mathematical model given by:

$$y_1 = w^{(0)} + w^{(1)}x^{(1)} + w^{(2)}x^{(2)} + \cdots + w^{(0)}dx^{(d)}$$

for each of the instances of this training set. This is, each of the following equations should hold true:

$$\begin{aligned}
1.\ y_1 &= w^{(0)} + w^{(1)}x_1^{(1)} + w^{(2)}x_1^{(2)} + \cdots + w^{(d)}x_1^{(d)} \\
2.\ y_2 &= w^{(0)} + w^{(1)}x_2^{(1)} + w^{(2)}x_2^{(2)} + \cdots + w^{(d)}x_2^{(d)} \\
&\ \vdots \\
m.\ y_m &= w^{(0)} + w^{(1)}x_m^{(1)} + w^{(2)}x_m^{(2)} + \cdots + w^{(d)}x_m^{(d)}
\end{aligned}$$

This system can be written as a matrix equation:

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} = \begin{bmatrix} 1 & x_1^{(1)} & x_1^{(2)} & \cdots & x_1^{(d)} \\ 1 & x_2^{(1)} & x_2^{(2)} & \cdots & x_2^{(d)} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_m^{(1)} & x_m^{(2)} & \cdots & x_m^{(d)} \end{bmatrix} \begin{bmatrix} w^{(0)} \\ w^{(1)} \\ \vdots \\ w^{(d)} \end{bmatrix}$$

Or, in a compact way:

$$\mathbf{y} = \mathbf{X}_{aum} \cdot \mathbf{w} \tag{5.1}$$

Linear regression. The last system of equations is an overdetermined system (it has more equations than unknowns) and it is impossible to find a solution which satisfies all the equations at once. Instead of that, however, we can find a vector **w** which *approximately* satisfies all the equations at once. The mathematical procedure used to find such a solution is called "linear regression".

Method of least squares. One possibility for doing linear regression is the so-called "method of least squares", which is meant to minimise the mean squared error (see Equation 4.1). We will not explain the method of least squares here, we will only tell you how to have MATLAB applying it for you. Just type:

```
>> w = Xaum\y;
```

A mnemonic way to recall this expression is to write Equation 5.1 and then solve for **w**. Multiplying both sides by $\mathbf{X}_{aum}^{-1}$:

$$\mathbf{X}_{aum}^{-1} \cdot \mathbf{y} = \mathbf{X}_{aum}^{-1} \cdot \mathbf{X}_{aum} \cdot \mathbf{w}$$

The inverse of a matrix multiplied by the matrix itself is the identity matrix, so that:

$$\mathbf{X}_{aum}^{-1} \cdot \mathbf{y} = \mathbf{I} \cdot \mathbf{w}$$

This results in:

$$\mathbf{X}_{aum}^{-1} \cdot \mathbf{y} = \mathbf{w}$$

This last expression, $\mathbf{w} = \mathbf{X}_{aum}^{-1} \cdot \mathbf{y}$, should remind you of the MATLAB code written above.

Polynomial regression. Polynomial regression involves higher-degree models. For example, suppose you have only one-dimensional vectors in your data matrix X:

$$\mathbf{X} = \begin{bmatrix} x_1^{(1)} \\ x_2^{(1)} \\ \vdots \\ x_m^{(1)} \end{bmatrix}$$

To simplify, we can write this as:

$$\mathbf{X} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix}$$

Then, you can perform a second-degree polynomial regression (quadratic model) for this data, by creating a matrix

$$\mathbf{X}_{aum} = \begin{bmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ \vdots & \vdots & \vdots \\ 1 & x_m & x_m^2 \end{bmatrix}$$

and using the same code

```
>> w = Xaum\ y;
```

with it.

To perform a third-degree polynomial regression (cubic model), create the matrix:

$$\mathbf{X}_{aum} = \begin{bmatrix} 1 & x_1 & x_1^2 & x_1^3 \\ 1 & x_2 & x_2^2 & x_2^3 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_m & x_m^2 & x_m^3 \end{bmatrix}$$

and so on.

5.2 Exercises

Solve the following exercises. You can use a calculator. Do not use MATLAB here

Let **X** be the matrix

$$\mathbf{X} = \begin{bmatrix} -2 \\ 0 \\ 3 \\ 5 \\ 6 \end{bmatrix}$$

and let **y** be the vector

$$\mathbf{y} = \begin{bmatrix} 5 & 1 & -1 & -3 & -6 \end{bmatrix}^T$$

be the data for a regression problem.

1. Suppose you want to fit a linear model using linear regression (least-squares solution) to these data. Write down the corresponding system of equations (5 equations).
2. Write this system as a matrix equation.
3. The least-squares solution for this system gives as a result the following weight vector:

$$\mathbf{w} = \begin{bmatrix} 2.0885 \\ -1.2035 \end{bmatrix}$$

Compute the $\widehat{\mathbf{y}}$ vector corresponding to this **w**.

4. Compute the mean squared error for this prediction.

5. Suppose now that you want to fit a cubic model using linear regression (least-squares solution) to these data. Write down the corresponding system of equations (5 equations).

6. Write this system as a matrix equation.

7. The least-squares solution for this system gives as a result the following weight vector:

$$\mathbf{w} = \begin{bmatrix} 0.9112 \\ -1.1054 \\ 0.3563 \\ -0.0602 \end{bmatrix}$$

Compute the $\hat{\mathbf{y}}$ vector corresponding to this $\mathbf{w}$.

8. Compute the mean squared error for this prediction.

5.3 Practical in MATLAB

General Objective:

To make the student capable of computing and interpreting the linear regression solution for a regression problem.

Specific Objectives:

- The student should be able to compute the simple linear regression solution for a problem using MATLAB.
- The student should be able to plot the linear regression solution in order to visually interpret the result.
- The student should be able to compute the simple linear regression solution with multiple features for a problem using MATLAB.

Materials:

Computer, MATLAB.

Procedure:

Solve the following exercises in MATLAB

1. Create a script and save it following the format `Pr5_nameLastname.m`. You will write your code for the next numerals in this script.

2. In folder *Data Sets* you will find a file named "`cars.xlsx`". This file contain the real information the author of this book collected to buy a second hand car in 2016. This information includes (as you can see by opening

the file in Excel): distance travelled, age, engine capacity and price. Save cars.xlsx to the Current Folder and load it using xlsread. Store the data in a variable you will call "carData".

3. Extract the first column of this carData matrix (the distance travelled by the cars) and call this vector "X_km".
4. Extract the last column (the price of the cars) and call this vector "ycars". This will be considered as the dependent variable in this regression problem.
5. Write a function "myLinearRegression(X,y)" which takes a matrix X of size $m \times d$ (where m is the number of instances and d is the dimensionality of the vectors) and a vector y of labels (values of the dependent variable). This function should return a vector yhat of the predictions after performing linear regression. More specifically, the function must perform a simple linear regression, using the "\" MATLAB operator. (Challenge: Do NOT use for loops when writing this function.)
6. Use your function myLinearRegression to obtain a vector yhat_km corresponding to the values in X_km.
7. Use your function plotRegression (written in Practical 1.4) to plot the values in ycars and yhat_km against those in X_km. Add suitable labels to the axes.
8. By looking at the plot obtained in the last numeral, find the point which is below the green line and is farthest from it. This corresponds, roughly, to the car that is the "best buy". What is the price and the distance travelled by this car?
9. Load into the workspace the matrices contained in the file matricesPr5.mat. This file can be found in the folder *Data Sets*.
10. Use your function myLinearRegression to obtain a vector yhat_1, the linear regression solution corresponding to matrix X and vector y loaded in numeral 9. Use plotRegression to plot the values in y and yhat_1 against those in X.
11. Use your function computeMSE (written in Practical 1.4) to compute the mean squared error of the predictions in yhat_1.
12. Use your function myLinearRegression to obtain a vector yhat_2, the predictions given by the quadratic model fit to the data. Plot this solution.

13. Compute the mean squared error of the predictions in `yhat_2`.

14. Use your function `myLinearRegression` to obtain a vector `yhat_3`, the predictions given by the `cubic` model fit to the data. Plot this solution.

15. Compute the mean squared error of the predictions in `yhat_3`.

16. Use your function `myLinearRegression` to obtain a vector `yhat_2`, the predictions given by the 4th-grade model fit to the data. Plot this solution.

17. Compute the mean squared error of the predictions in `yhat_4`.

5.4 Answers to selected exercises

Exercises

1. $$\begin{cases} w^{(1)} - 2w^{(2)} &= & 5 \\ w^{(1)} &= & 1 \\ w^{(1)} + 3w^{(2)} &= & -1 \\ w^{(1)} + 5w^{(2)} &= & -3 \\ w^{(1)} + 6w^{(2)} &= & -6 \end{cases}$$

2. $$\begin{bmatrix} 1 & -2 \\ 1 & 0 \\ 1 & 3 \\ 1 & 5 \\ 1 & 6 \end{bmatrix} \times \begin{bmatrix} w^{(1)} \\ w^{(2)} \end{bmatrix} = \begin{bmatrix} 5 \\ 1 \\ -1 \\ -3 \\ -6 \end{bmatrix}$$

3. $$\widehat{\mathbf{y}} = \begin{bmatrix} 4.4956 \\ 2.0885 \\ -1.5221 \\ -3.9292 \\ -5.1327 \end{bmatrix}$$

4. 0.6655

5. $$\begin{cases} w^{(1)} - 2w^{(2)} + 4w^{(3)} - 8w^{(4)} &= & 5 \\ w^{(1)} &= & 1 \\ w^{(1)} + 3w^{(2)} + 9w^{(3)} + 27w^{(4)} &= & -1 \\ w^{(1)} + 5w^{(2)} + 25w^{(3)} + 125w^{(4)} &= & -3 \\ w^{(1)} + 6w^{()} + 36w^{(3)} + 216w^{(4)} &= & -6 \end{cases}$$

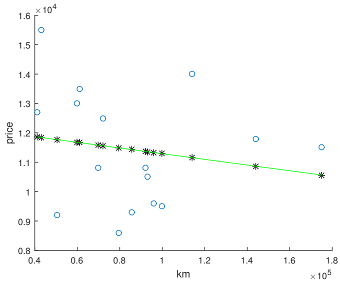
6. $$\begin{bmatrix} 1 & -2 & 4 & -8 \\ 1 & 0 & 0 & 0 \\ 1 & 3 & 9 & 27 \\ 1 & 5 & 25 & 125 \\ 1 & 6 & 36 & 216 \end{bmatrix} \times \begin{bmatrix} w1 \\ w2 \\ w3 \\ w4 \end{bmatrix} = \begin{bmatrix} 5 \\ 1 \\ -1 \\ -3 \\ -6 \end{bmatrix}$$

7. $$\widehat{\mathbf{y}} = \begin{bmatrix} 5.0285 \\ 0.9112 \\ -0.8225 \\ -3.2282 \\ -5.8890 \end{bmatrix}$$

8. 0.0209

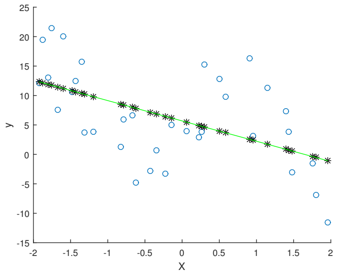
Practical

7.



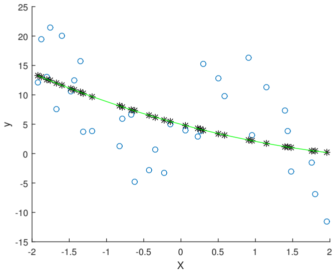
8. Price: 8600 USD; Distance travelled: 79600 km

10.



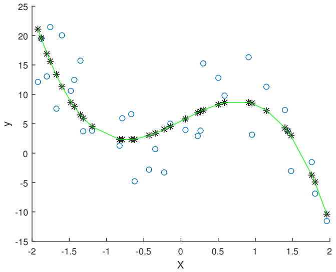
11. 45.28

12.



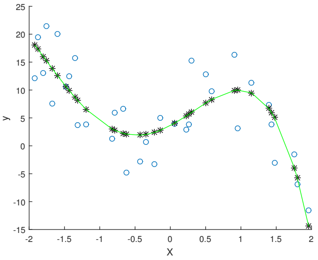
13. 44.93

14.



15. 23.68

16.



17. 20.78

Chapter 6

Overfitting and underfitting: visualisation

6.1 Theoretical Briefing

Same data, different models. It is possible to find several models to fit a given dataset. We saw an example of this in the previous chapter, in which n-th degree polynomial regression was introduced. Each value of n corresponds to a different model, and so it is possible to generate, in principle, infinitely many models to fit the data. Also, in coming chapters we will see other algorithms which can generate even more models. Having such an abundance, how to choose the "best" one?

Evaluation of models. To choose a model, apply performance measures such as mean squared errors, misclassification errors and confusion matrices, and pick out the one with lowest error. Quite straightforward. Nevertheless, this chapter is meant to show you that, most of the time, too much perfection is as bad as too much laxness.

Underfitting. In the first place, let us talk about too much relaxation. We refer here to models that really "take it easy" and give a very lazy prediction about the data, without taking into account particularities and details. The most relaxed model for a classification problem, for example, would be one which predicts the same class for every instance (say, the class "woman" for all of the photos of a given set, regardless of the actual gender of the person depicted.) In Fig 6.1 we show a regression model which predicts the same value of y for every value of x (you will obtain this plot in the Practical.) We say that this kind of models **underfits** the data.

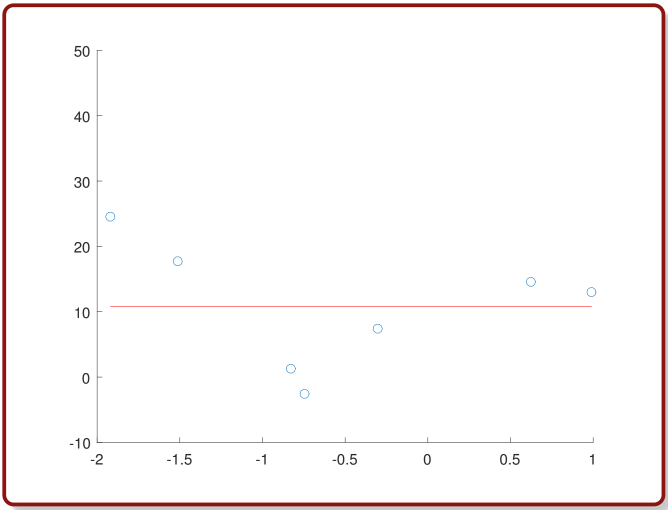


FIGURE 6.1: A model that underfits the data.

Overfitting. You could think, then, that the best model is one which perfectly fits all of the data. In Fig 6.2 we see a linear regression model that attains this (you will obtain this plot in the Practical as well.)

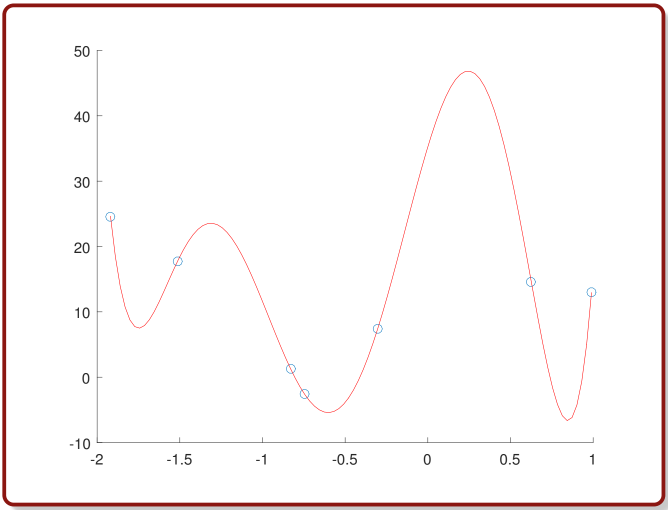


FIGURE 6.2: A model that overfits the data.

Nevertheless, one desired characteristic of a model is its **generalisation capability.** This is, if the model is presented with a new instance, not used previously in the training phase, it should predict a coherent value of y. Let us show that this is not the case with the model of Figure 6.2. Consider the value of $x = 0.3$. By looking at the plot in Figure 6.2, what is the value of y for this x? Well, it is about 50! Clearly, this is an exaggerated prediction, out of place with respect to the other values. We say that this model does not have a good generalisation capability, and this is because it **overfits** the data.

Dealing with overfitting. In future chapters, we will see how to overcome problems such as overfitting. One possibility is using a validation set to select the best model, a technique that we will use in Chapter 11.

6.2 Exercises

Solve the following exercises. You can use a calculator. Do not use MATLAB here

Let

$$\mathbf{X} = \begin{bmatrix} -2 & -1 & 1 & 2 & 3 & 4 & 5 & 6 & 8 \end{bmatrix}^T$$

and

$$\mathbf{y} = \begin{bmatrix} 1 & 3 & 5 & 4 & 2 & 5 & 7 & 4 & 5 \end{bmatrix}^T$$

be the data for a regression problem.

1. On graph paper, draw a Cartesian plane and plot the data in $\mathbf{X}$ in the horizontal axis and the data in $\mathbf{y}$ in the vertical one.

2. Draw a straight line that fits well to the data (according to your intuition).

3. Draw a curve that overfits the data (again, according to your intuition).

4. In Fig 6.3, we show a plot in MATLAB of the 1^{st}-degree polynomial regression for this data. According to this model, what is the value of $\widehat{y}$ for $x = 7.5$ (approximately)?

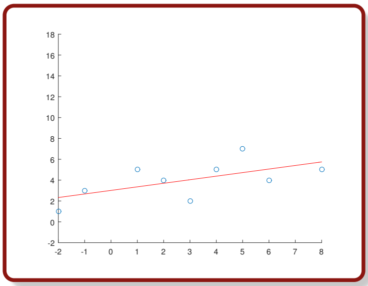


FIGURE 6.3

5. In Fig 6.4, we show a plot in MATLAB of the 8^{th}-degree polynomial regression for this data. According to this model, what is the value of $\widehat{y}$ for $x = 7.5$ (approximately)?

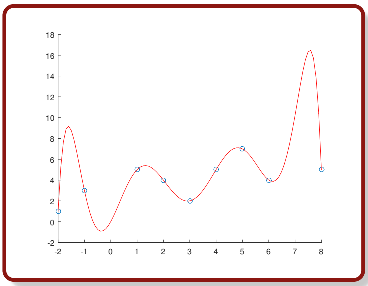


FIGURE 6.4

6.3 Practical in MATLAB

General Objective:

To make the student capable of computing and comparing the solutions given by different models to a given data set.

Specific Objectives:

- The student should be able to compute different solutions for a data set given as example.
- The student should be able to graphically represent the solutions and identify the problems of overfitting and underfitting in these graphical representations.
- The student should be able to estimate the error for each solution and identify the problems of overfitting and underfitting through these performance measures.

Materials:

Computer, MATLAB.

Procedure:

Solve the following exercises in MATLAB

1. Create a script and save it following the format `Pr6_nameLastname.m`. You will write your code for the next numerals in this script.
2. Load into the *Workspace* the matrices contained in the file `matricesPr6.mat`. This file can be found in the folder *Data Sets*.
3. In numeral 2 you loaded a data matrix `X` of one-dimensional vectors and a vector y of the corresponding labels. Use `scatter` to plot the data contained in `X` and `y`. Use the MATLAB command `ylim` to set the limits in the y axis as -10 and 50.
4. By looking at the plot obtained in the previous numeral, make a guess about what the labels (the values of y_i) should be for the “vectors” contained in the matrix `Xnew`. Do not compute anything here, just use your

intuition.

$$\mathbf{X}_{new} = \begin{bmatrix} -1.7500 \\ -1.3000 \\ 0.2500 \\ 0.8500 \end{bmatrix}$$

5. Write a function "`plot_nModel(X,y,n)`", which takes a data matrix `X` of one-dimensional feature vectors, a vector `y` of labels and a parameter `n`; and does not return anything but plots the data in `X` and `y` together with the model corresponding to a n-th degree polynomial regression (see the *Theoretical Briefing* of Chapter 5.)

6. Use your function `plot_nModel(X,y,n)` to obtain plots of the models with:

   a. $n = 0$

   b. $n = 1$

   c. $n = 2$

   d. $n = 3$

   e. $n = 6$

   Set the limits of the y axis as -10 and 50 in all the cases.

7. Write a function "`nModel(X,y,n)`", which takes a data matrix `X` of one-dimensional feature vectors, a vector `y` of labels and a parameter `n`; and returns the weight vector, `w`, corresponding to a n-th degree polynomial regression.

8. Use your function `nModel` to get the vectors `w` and `yhat` for the `Xnew` matrix given in numeral 4, corresponding to the models with:

   a. n = 0

   b. n = 1

   c. n = 2

   d. n = 3

   e. n = 6

   Let us call these vectors `yhat_0`, `yhat_1`, `yhat_2`, `yhat_3` and `yhat_6`, correspondingly. In the *Answers* section, you can see the vectors you should obtain.

9. The author of this book made his own guess about the labels for `Xnew` (see numeral 4). His intuitions are contained in vector $\mathbf{y}_{guess}$.

$$\mathbf{y}_{guess} = \begin{bmatrix} 20 \\ 10 \\ 10 \\ 17 \end{bmatrix}$$

Compute the mean squared error, with respect to these intuitions, of the predictions contained:

a. In **yhat_0**

b. In **yhat_1**

c. In **yhat_2**

d. In **yhat_3**

e. In **yhat_6**

10. Discuss with your classmates about which model was the best and how the phenomena of overfitting and underfitting were unveiled through this practical.

6.4 Answers to selected exercises

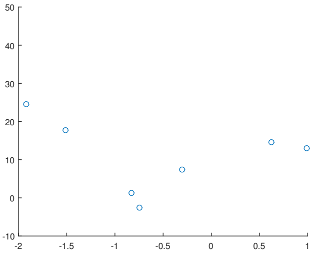
Exercises

4. Approximately, yhat = 5

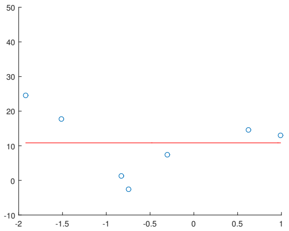
5. Approximately, yhat = 17

Practical

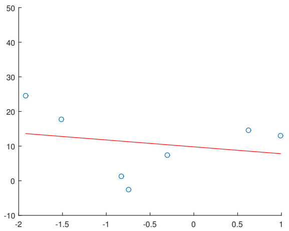
3.



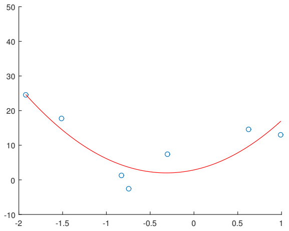
6. (a)



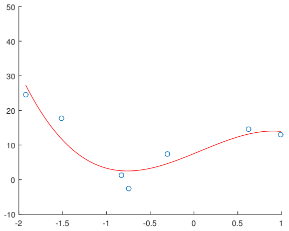
(b)



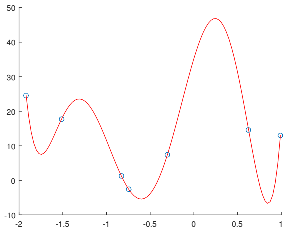
(c)



(d)



(e)



8. (a) $\texttt{yhat\_0} = \begin{bmatrix} 10.8386 \\ 10.8386 \\ 10.8386 \\ 10.8386 \end{bmatrix}$

(b) $\texttt{yhat\_1} = \begin{bmatrix} 13.2707 \\ 12.3750 \\ 9.2898 \\ 8.0955 \end{bmatrix}$

(c) $\texttt{yhat\_2} = \begin{bmatrix} 20.0764 \\ 10.5280 \\ 4.7923 \\ 13.8713 \end{bmatrix}$

(d) $\texttt{yhat\_3} = \begin{bmatrix} 19.6855 \\ 6.9666 \\ 10.0951 \\ 14.0033 \end{bmatrix}$

(e) $\texttt{yhat\_6} = \begin{bmatrix} 7.4999 \\ 23.5441 \\ 46.8416 \\ -6.6499 \end{bmatrix}$

9. (a) 30.8249

(b) 32.6795

(c) 9.2984

(d) 4.5723

(e) 564.0793

Part III

Historic Algorithms

We study here three algorithms developed in the 40s, 50s and 60s of the twentieth century; and we do so because they deserve to be studied, even now, after many years of their appearance. The McCulloch-Pitts neuron, studied in Chapter 7, is the mathematical unit from which many algorithms have been built, including the modern *Neural Networks* and *Deep Learning* algorithms; the *Perceptron* algorithm, Chapter 8, was the first one with the ability to "learn" (modify its weights according to the data); and *Adaline*, Chapter 9, is a pioneer in the application of neural computation to regression problems.

Most likely, you will never use these algorithms in real applications nowadays. Knowing them, however, will help you appreciate the great effort of human ingenuity that has led us to our modern intelligent systems. Also, understanding the behaviour of McCulloch-Pitts neurons will be the basis for a deep comprehension of Neural Networks in coming chapters.

Chapter 7

McCulloch-Pitts Neuron

7.1 Theoretical Briefing

Biological neurons. A neuron is a cell. A cell specialised for signal processing and transmission. In Figure 7.1, we show the morphology and basic components of a neuron.

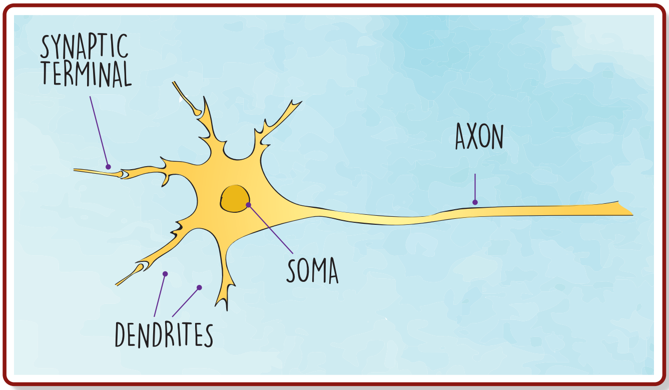


Figure 7.1: Scheme of the biological neuron

The neuron consists of a central **soma**, surrounded by branches or extensions called **dendrites**. The signals coming from other neurons through their **synaptic terminals**, arrive at these dendrites. Then, our neuron itself can emit a signal through its **axon**. At its end (not shown), the axon has synaptic terminals as well, which in turn deliver signals to other neurons' dendrites. Nevertheless, there is a detail: synaptic terminals and dendrites are not connected physically. Looking closer, we would see that there is a gap between them, a gap called **synaptic cleft**. The communication across the synaptic cleft is carried out by means of intermediate molecules, called **neurotransmitters**. Neurotransmitters are stored in tiny bags called **synaptic vesicles**, located in the synaptic terminals. When a signal arrives through the synaptic terminal, the neurotransmitters act as "filters" of the signal: if there are scarce neurotransmitters, for example, you will have a weak signal entering the neuron, however strong the original signal was.

Firing of neurons. Imagine the following experiment: you insert an electrode to stimulate the interior of a neuron and another electrode to measure its response (in millivolts). What would you get? Here are some results: even without any stimulus, there is a voltage of around -70 mV in the interior of the neuron with respect to the exterior. (This voltage is called **resting potential**.) If you apply a stimulus, you can depolarise the neuron (with a positive stimulus) or hyperpolarise it (with a negative one). There is a special value, -50 mV, called the **threshold potential**. If you do not surpass this value, the neuron reacts only proportionally to the stimulus applied. But if you do surpass it, the response of the neuron is kind of exaggerated: its potential shoots up to +40 mV! This value is called the **action potential**. We say in this case that the neuron has **fired**.

McCulloch-Pitts neuron. The American scientists Warren McCulloch (who studied philosophy, psychology and medicine) and Walter Pitts (logician, autodidact) created in 1943 the first model of an artificial neural network, made up from abstract units now called **McCulloch-Pitts neurons**. In Figure 7.2, we show a sketch of a McCulloch-Pitts neuron.

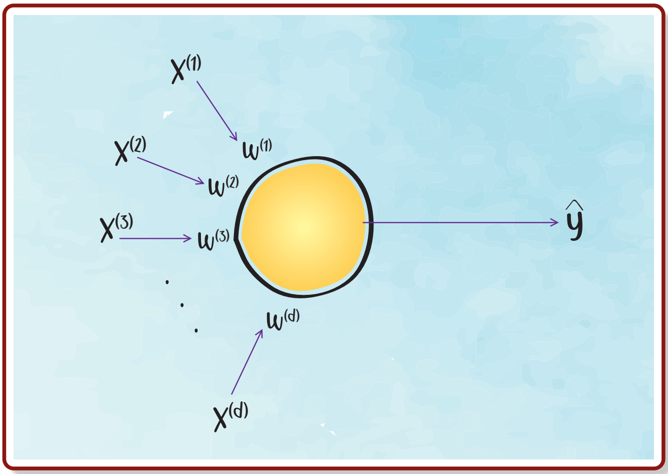


FIGURE 7.2: Scheme of the McCulloch-Pitts neuron

This abstract unit mimics the biological neuron in several ways: it receives an input signal (mathematically, a feature vector $\mathbf{x}$). Each one of the components of the feature vector goes through a "dendrite" and is multiplied by a **weight** (analogous to the neurotransmitters). These contributions are summed up in the "soma", represented here by the central circle. The outgoing arrow represents the axon, which delivers the output signal, $\hat{y}$. The sum obtained in the "soma" is given by:

$$s = \sum_{j=1}^{d} x^{(j)} w^{(j)} \tag{7.1}$$

Or, in a compact way:

$$s = \mathbf{x}^T \cdot \mathbf{w} \tag{7.2}$$

where

$$\mathbf{w} = \begin{bmatrix} w^{(1)} & w^{(2)} & w^{(3)} & \dots & w^{(d)} \end{bmatrix}^T$$

is the weight vector of the neuron. Other elements "copied" from nature are the threshold and the firing: if you surpass a threshold value, the neuron fires. Otherwise, it does not. This is, mathematically:

$$y = \begin{cases} 1 & \text{if} \quad s > \theta \\ 0 & \text{if} \quad s \leq \theta \end{cases} \tag{7.3}$$

where "1" means *firing*, "0" means *no firing* and θ is the threshold.

Working with data matrices. As usual, we can enter several feature vectors at once, by arranging them in a data matrix $\mathbf{X}$. In this case, we could apply the McCulloch-Pitts criteria simultaneously by using matrix calculus:

$$\mathbf{s} = \mathbf{X} \cdot \mathbf{w} \tag{7.4}$$

where $\mathbf{s}$ will be a $m \times 1$ vector. Then, we should apply the "firing" criterion (Equation 7.3) to each of the components of $\mathbf{s}$, in order to obtain the vector $\hat{\mathbf{y}}$ of outputs (dimension: $m \times 1$) for the m instances.

McCulloch-Pitts neuron as a classifier. Since the output of a McCulloch-Pitts neuron can be either a "0" or a "1", you can use it to solve classification problems with two classes. For instance, suppose you are interested in classifying images from men and women by genre. To do this using a McCulloch-Pitts neuron, assign the label "1" to women and "0" to men (or the other way round, if you feel offended.)

Activation functions. A function that takes the sum s (see Equation 7.1) and computes the output y of a neuron is called an **activation function**, represented by $f(s)$. This is,

$$y = f(s) \tag{7.5}$$

The activation function corresponding to Equation 7.3 is:

$$f(s) = \begin{cases} 1 & \text{if} \quad s > \theta \\ 0 & \text{if} \quad s \leq \theta \end{cases} \tag{7.6}$$

Of course, this is not the only activation function that exists. In the coming chapters we will see other possibilities.

7.2 Exercises

Solve the following exercises. You can use a calculator. Do not use MATLAB here

Let **X** be the matrix and **y** be the vector

$$\mathbf{X} = \begin{bmatrix} -10 & 6 \\ -5 & 2 \\ -1 & 5 \\ 2 & 9 \\ 2 & -3 \\ 5 & 8 \\ 6 & -1 \\ 8 & -4 \\ 9 & 9 \\ 11 & 5 \\ 12 & 10 \\ 13 & 13 \\ 14 & -3 \\ 19 & 2 \\ 21 & 3 \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

of data for a classification problem, where **y** is the vector of actual labels.

1. Locate, in a Cartesian plane, the feature vectors. Use different colours and markers to represent their actual classes.
2. Compute $\hat{\mathbf{y}}$, the vector of predictions given for the data in **X** by a McCulloch-Pitts neuron with threshold θ=0 and weight vector

$$\mathbf{w} = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$$

3. In the same Cartesian plane of numeral 1, locate the misclassified instances by overlapping their predicted markers over the actual ones.
4. Compute the confusion matrix of this prediction.
5. Compute the number of true positives, false positives, false negatives and true negatives for class "0".

7.3 Practical in MATLAB

General Objective:

To make the student capable of computing predictions for a classification problem using a McCulloch-Pitts neuron with given weights and threshold.

Specific Objectives:

- The student should be able to implement a McCulloch-Pitts neuron in MATLAB.
- The student should be able to use the implemented neuron to compute the predictions for a set of feature vectors.
- The student should be able to evaluate the performance of McCulloch-Pitts neurons.

Materials:

Computer, MATLAB.

Procedure:

Solve the following exercises in MATLAB

1. Create a script and save it following the format `Pr7_nameLastname.m`. You will write your code for the next numerals in this script.
2. Load into the *Workspace* the matrices contained in the file `matricesPr7.mat`. This file can be found in the folder *Data Sets*.
3. Write a function "`myMcC_P(X,w,Theta)`", which returns a vector `yhat` of the predictions given for a data matrix `X` by a McCulloch-Pitts neuron with weight vector `w` and a threshold `Theta`. (*Challenge*: Do not use `for` loops to write this function.)
4. In numeral 2 you loaded four 3-D vectors: `x1`, `x2`, `x3` and `x4`. Assemble a data matrix "`X1`" with these vectors, in the standard Machine-Learning way. Then, use your function `myMcC_P` to compute the predictions for the data in `X1` by a McCulloch-Pitts neuron with threshold $\theta = 3$ and weight

vector

$$\mathbf{w} = \begin{bmatrix} 2 \\ -1 \\ -3 \end{bmatrix}$$

5. In numeral 2 you loaded two 20-D vectors: `x5` and `x6`. Assemble a data matrix "`X2`" with these vectors, in the standard Machine-Learning way. Then, use your function `myMcC_P` to compute the predictions for the data in `X2` by a McCulloch-Pitts neuron with threshold $\theta = 1$ and weight vector `w2`, also loaded in numeral 2.

6. In numeral 2 you loaded a data matrix `X` with 2-D feature vectors, and a `y` vector of actual labels. Use your function `myMcC_P` to compute the predictions for the data in **X** by a McCulloch-Pitts neuron with threshold $\theta = 10$ and weight vector

$$\mathbf{w} = \begin{bmatrix} 2 \\ -1 \end{bmatrix}$$

7. Write a function "`plotClasses_McC_P(X,y,yhat)`", which does not return anything but creates a plot with the feature vectors in `X` (suppose they are 2-D), with the following characteristics: vectors of class "0", black; class "1", red; actual classification (given by the labels in `y`), circles; predicted classification (given by the labels in `yhat`), stars. Use the MATLAB function `legend` to add a legend to the plot, indicating the actual and predicted classes. (You can look at the answer of numeral 8 to get a better idea of what is expected of your function `plotClasses_McC_P`.)

8. Call your function `plotClasses_McC_P` from the script, passing as arguments the matrix `X` and the vector `y` mentioned in numeral 6, and the vector `yhat` computed in the same numeral.

9. Compute the misclassification error for the predictions obtained in numeral 6.

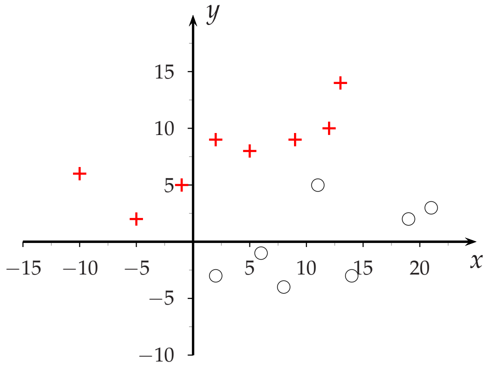
10. Compute the confusion matrix for the predictions obtained in numeral 6.

11. Compute the number of true positives, false positives, false negatives and true negatives for class "1".

7.4 Answers to selected exercises

Exercises

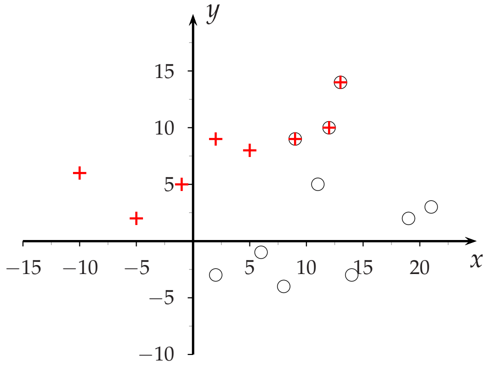
1. (a)



(b) We present the vector transposed, to save space:

$$\mathbf{y}^T = [1\ 1\ 1\ 1\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0]$$

(c)



(d)

		Predicted	
		0	**1**
Actual	**0**	7	0
	1	3	5

(e) True positives: 7
False positives: 3

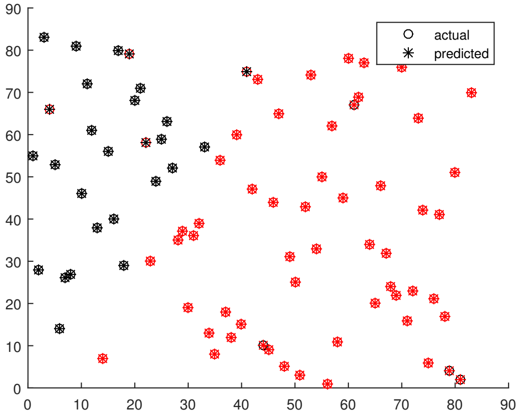
False negatives: 0
True negatives: 5

Practical

4. $\begin{bmatrix} 0 & 0 & 1 & 0 \end{bmatrix}^T$

5. $\begin{bmatrix} 1 & 0 \end{bmatrix}^T$

8.



9. 0.0964

10.

		Predicted	
		0	**1**
Actual	**0**	23	4
	1	4	52

11. True positives: 52
False positives: 4
False negatives: 4
True negatives: 23

CHAPTER 8

PERCEPTRON

8.1 Theoretical Briefing

Artificial neuron for the perceptron. With subtle differences (created for mathematical reasons) the neuron for the perceptron is basically a McCulloch-Pitts neuron. In Figure 8.1, we depict the perceptron unit.

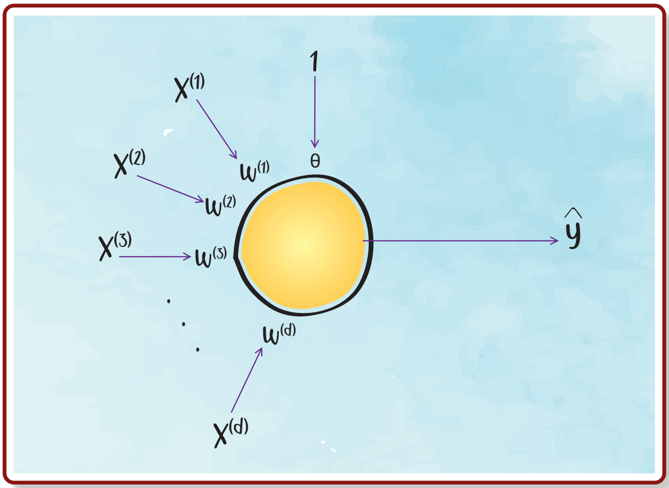


FIGURE 8.1: The artificial neuron for the perceptron

Sum in a perceptron. As you might have noted, the neuron in Figure 8.1 incorporates an additional "dendrite" or entry. The value of the signal for this entry is always 1. Always. Therefore, quite reasonably, this artificial dendrite is called a **bias** entry. The corresponding weight is called θ and should remind you of the threshold of the McCulloch-Pitts neuron. The sum inside the "soma" of the neuron is, then:

$$s = \theta + \sum_{j=1}^{d} x^{(j)} w^{(j)} \tag{8.1}$$

We can write this equation in a more compact way:

$$s = \mathbf{x}_{aug}^T \times \mathbf{w} \tag{8.2}$$

where

$$\mathbf{x}_{aug} = \begin{bmatrix} 1 & x^{(1)} & x^{(2)} & x^{(3)} & \cdots & x^{(d)} \end{bmatrix}^T$$

and

$$\mathbf{w} = \begin{bmatrix} \theta & w^{(1)} & w^{(2)} & w^{(3)} & \cdots & w^{(d)} \end{bmatrix}^T$$

both of which are $(d+1) \times 1$ vectors. The “aug” subscript stands for “augmented”, since we have appended a “1” at the top of the feature vector.

Activation function for the perceptron. The activation function, $f(s)$, for the perceptron is:

$$f(s) = \begin{cases} 1 & \text{if} \quad s > 0 \\ -1 & \text{if} \quad s \leq 0 \end{cases} \tag{8.3}$$

Sum for a data matrix. Again, it is possible to collect several feature vectors in a data matrix $\mathbf{X}$ and use this whole matrix as the entry for the perceptron. The vector of sums for such a matrix would be, then:

$$\mathbf{s} = \mathbf{X}_{aug} \times \mathbf{w} \tag{8.4}$$

where

$$\mathbf{X}_{aug} = \begin{bmatrix} 1 & \longleftarrow \mathbf{x}_1^T \longrightarrow \\ 1 & \longleftarrow \mathbf{x}_2^T \longrightarrow \\ \vdots & \vdots \\ 1 & \longleftarrow \mathbf{x}_m^T \longrightarrow \end{bmatrix} \tag{8.5}$$

(In other words, $\mathbf{X}_{aug}$ is the just the same usual matrix $\mathbf{X}$, but with a column of ones appended at its left.)

The two-dimensional case. For the two-dimensional case, we have the chance to visualise what a perceptron does. To see how to do this, consider Equations 8.1 and 8.3. From the latter, you can infer that something special must happen at 0. Therefore, taking into account Equation 8.1, you can infer that something special must happen when

$$\theta + \sum_{j=1}^{d} x^{(j)} w^{(j)} = 0 \tag{8.6}$$

Since $d = 2$ in the two-dimensional case, the last equation turns out to be

$$\theta + x^{(1)}w^{(1)} + x^{(2)}w^{(2)} = 0 \tag{8.7}$$

But this is the equation of a straight line! To see how this is true, let us solve for $x^{(2)}$. By doing this, we have:

$$x^{(2)} = \frac{-w^{(1)}}{w^{(2)}}x^{(1)} + \frac{-\theta}{w_2} \tag{8.8}$$

which is totally analogous to the equation $y = mx + b$, the equation of a straight line. So, a *weight vector corresponds to a **straight line** in the two-dimensional case*. The aim of the perceptron algorithm will be then to find a straight line which separates the class "1" vectors from the class "-1" vectors. Generalising, we will state that a weight vector corresponds to a **plane** in the three-dimensional case and that, in higher-dimensional spaces, it corresponds to a **hyperplane**; both on a mission to separate class "1" vectors from class "-1" vectors.

The perceptron algorithm. Created in 1957 by American psychologist Frank Rosenblatt, this was the first algorithm to perform "learning". That is, to "tune" the weight vector until the classification is correct (graphically, until the line, plane or hyperplane, separates the instances correctly.) As input, the algorithm takes a data set made up by $\mathbf{X}$ (dimension $m \times d$) and $\mathbf{y}$ (dimension $m \times 1$), and returns the "tuned" weight vector $\mathbf{w}$. In Algorithm 1 we show the perceptron learning algorithm in pseudocode.

In Algorithm 1, $\mathbf{x}_{aug}^{(i)}$ represents the i^{th} feature vector, augmented. This is,

$$\mathbf{x}_{aug}^{(i)} = \begin{bmatrix} 1 & x_i^{(1)} & x_i^{(2)} & x_i^{(3)} & \cdots & x_i^{(d)} \end{bmatrix}^T$$

The stopping criterion for the perceptron algorithm, shown in line 3 of Algorithm 1, can be changed in order to soften the expectations of the algorithm. That is, instead of looking for perfection, the algorithm is only required to reach a "goal error", to be set by the user. For doing this, just change line 3 of Algorithm 1 for:

```
    while misclassification error > goal error do
```

A mathematical expression such as that of line 6 is called a *learning rule*. Note that in the second term of this expression, a **scalar multiplication** appears. Check your books on physics or linear algebra to recall this concept, if needed.

Algorithm 1 The perceptron learning algorithm

```
$\mathbf{w} \leftarrow \mathbf{w}_0$    %random initialisation
Compute current misclassification error
while misclassification error $\neq 0$ do
    for $i = 1 : m$ do
        if $\widehat{y}_i \neq y_i$ then
            $\mathbf{w} \leftarrow \mathbf{w} + y_i \cdot \mathbf{x}_{aug}^{(i)}$
        end if
    end for
    Compute current misclassification error
end while
```

8.2 Exercises

Solve the following exercises. You can use a calculator. Do not use MATLAB here

Let

$$\mathbf{X} = \begin{bmatrix} 0 & -2 \\ 1 & 4 \\ 2 & -1 \\ 4 & 4 \\ 4 & -3 \\ 5 & 5 \\ 6 & 2 \end{bmatrix}$$

$$\mathbf{y} = \begin{bmatrix} 1 & -1 & 1 & -1 & 1 & -1 & 1 \end{bmatrix}^T$$

be the data matrix and the vector of actual labels for a classification problem, respectively.

1. Represent the feature vectors in a Cartesian plane. Use different marker shapes to differentiate among classes: little circles for vectors with “-1” label and plus signs for vectors with “1” label.

2. Take

$$\mathbf{w} = \begin{bmatrix} -7 \\ 2 \\ 1 \end{bmatrix}$$

as an initial weight vector. Draw the straight line that corresponds to this vector, in the same plot you made in numeral 1.

3. Execute, by hand, the Perceptron algorithm for these data and complete the Table 8.1 (in the first column write down how many times the algorithm has entered the **while** loop at that moment). NOTE: This table is incomplete, since the algorithm will have to enter the **while** loop four times until having all the instances correctly classified. Nevertheless, this is enough as a didactic exercise.

TABLE 8.1

Times at while	i	$\mathbf{x}_i^T$	$\hat{\mathbf{y}}_i$	**Is** $\hat{\mathbf{y}}_i \neq \mathbf{y}_i$ **? (y/n)**	$\mathbf{w}^T$
0	–	–	–	–	[-7 2 1]
Misclassification error:					
1	1				
	2				
	3				
	4				
	5				
	6				
	7				
Misclassification error:					
2	1				
	2				[-5 6 -4]

8.3 Practical in MATLAB

General Objective:

To make the student capable of training and using a perceptron.

Specific Objectives:

- The student should be able to plot the perceptron solution together with the data for a classification problem, and interpret this plot.
- The student should be able to implement a perceptron to get predictions for a data matrix.
- The student should be able to implement the perceptron learning algorithm.

Materials:

Computer, MATLAB.

Procedure:

Solve the following exercises in MATLAB

1. Create a script and save it following the format `Pr8_nameLastname.m`. You will write your code for the next numerals in this script.

2. Load into the *Workspace* the matrices contained in the file `matricesPr8.mat`. This file can be found in the folder *Data Sets*.

3. In the folder *"MATLAB Functions"*, you will find a function `plotPerceptron.m`. Save it in your *Current Folder* and use it to make a plot of the data loaded in numeral 2, passing as weight vector:

$$\mathbf{w} = \begin{bmatrix} -30 \\ 2 \\ 3 \end{bmatrix}$$

4. Create a function "`perceptronOutput(X,w)`", which takes a data matrix `X` and a weight vector `w`, and computes a vector "`yhat`" containing the predictions given by the corresponding perceptron.

5. Use your function `perceptronOutput` to compute the predictions of the perceptron with the weight vector given in numeral 3, over the matrix `X` loaded in numeral 2.

6. Create a function "`perceptronLearning(X,y,w_ini)`", which takes a data matrix `X`, a vector of actual labels `y` and an initial weight vector `w_ini` and executes the Perceptron algorithm. This function must return the weight vectors generated during the learning process, collected in a matrix which we will call "`W_values`". This matrix will contain all the values of `w`, transposed, one below the other, starting with the initial value `w_ini`. Also, this function must display in the *Command Window* the following things: a message telling how many times it is entering the `while` loop; which instance it is analysing at that moment; the initial misclassification error; and the misclassification error each time it gets out the `while` loop. You can see the answer to numeral 7 to get a better idea of what we are asking for.

7. Use your function `perceptronLearning` with the following data:

$$\mathbf{X} = \begin{bmatrix} -1 & -1 \\ 1 & -1 \\ -1 & 1 \\ 1 & 1 \end{bmatrix}$$

$$\mathbf{y} = \begin{bmatrix} -1 & -1 & -1 & 1 \end{bmatrix}^T$$

$$\mathbf{w}_{ini} = \begin{bmatrix} 0.5 & 1 & 1 \end{bmatrix}^T$$

(a) What is the resulting `W_values` matrix?

(b) What are the messages printed in console?

8. Try your function `perceptronLearning` on the data loaded in numeral 2 to obtain a matrix "`W_values2`".

   (a) How many times does the algorithm enter the `while` loop?

   (b) What is the initial misclassification error?

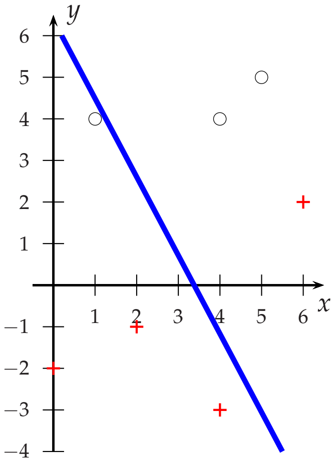
   (c) What is the misclassification error before entering the `while` loop for the last time?

9. Use the function `plotPerceptron` to plot the data with the last six weight vectors in `W_values2` and arrange them in a 2 × 3 array using **subplot**.

8.4 Answers to selected exercises

Exercises

2.

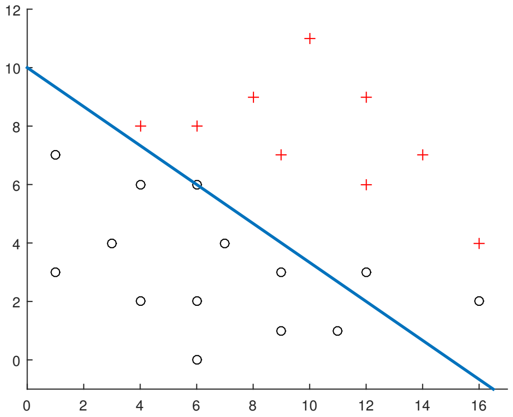


3.

Times at while	i	$\mathbf{x}_i^T$	$\widehat{\mathbf{y}}_i$	**Is $\widehat{\mathbf{y}}_i \neq \mathbf{y}_i$? (y/n)**	$\mathbf{w}^T$
0	–	–	–	–	[-7 2 1]
Misclassification error: 0.7143					
1	1	[0 -2]	-1	yes	[-6 2 -1]
	2	[1 4]	-1	no	[-6 2 -1]
	3	[2 -1]	-1	yes	[-5 4 -2]
	4	[4 4]	1	yes	[-6 0 -6]
	5	[4 -3]	1	no	[-6 0 -6]
	6	[5 5]	-1	no	[-6 0 -6]
	7	[6 2]	-1	yes	[-5 6 -4]
Misclassification error: 0.2857					
2	1	[0 -2]	1	no	[-5 6 -4]
	2	[1 4]	-1	no	[-5 6 -4]

Practical

3.



5. Your `yhat` should be a 23×1 vector with twelve numbers "-1" followed by eleven numbers "1".

7. (a) $\texttt{W\_values} = \begin{bmatrix} 0.5 & 1 & 1 \\ -0.5 & 0 & 2 \\ -1.5 & 1 & 1 \end{bmatrix}$

(b)

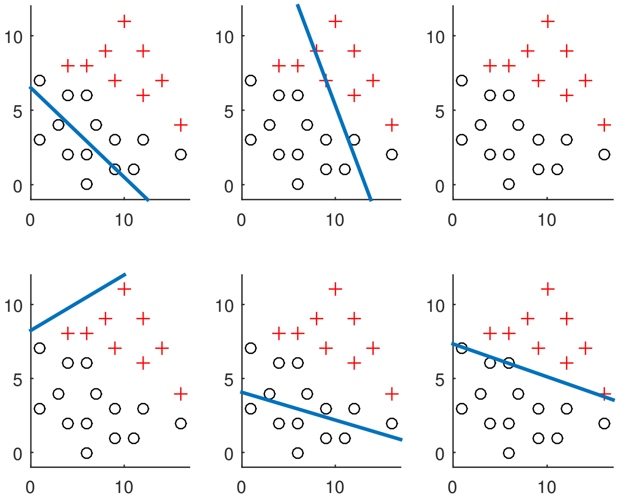
```
The initial misclassification error is: 0.500000
Entering the while for the 1th. time, instance 1...
Entering the while for the 1th. time, instance 2...
Entering the while for the 1th. time, instance 3...
Entering the while for the 1th. time, instance 4...
The misclassification error now is: 0.000000
```

8. (a) 69

(b) 0.086957

(c) 0.347826

9.



Chapter 9

Adaline

9.1 Theoretical Briefing

Artificial neuron for the ADALINE. The artificial neuron for this algorithm will be exactly the same as that for the perceptron (see Figure 8.1.)

Sum in an ADALINE. The sum in the ADALINE is the same as that for the perceptron (see Equations 8.1, 8.2 and 8.4.)

Activation function. Now, this is the first difference between the perceptron and the ADALINE. For the latter, the activation function is simply given by:

$$f(s) = s \tag{9.1}$$

Note that whereas the result from the activation function in the perceptron is discrete (i.e., it can take only discrete values, namely -1 and 1), the output from the activation function shown in Equation 9.1 is *continuous*. This implies that $\widehat{y} \in \Re$. In other words, *ADALINE is a regression algorithm*.

Error for ADALINE. Since ADALINE is a regression algorithm, the error to be computed here should be the mean squared error (see Equation 4.1.) However, in order to have some terms simplified when doing the math, a new error has been defined, with a small modification from the original one:

$$\varepsilon = \frac{1}{2}\sum_{i=1}^{m}(y_i - \widehat{y}_i)^2 \tag{9.2}$$

We will use Equation 9.2 when computing the error in all this section.

The ADALINE algorithm. Standing for Adaptive Linear Neuron, ADALINE was developed by American engineers Bernard Widrow and Ted Hoff in 1960. The corresponding learning algorithm is shown in Algorithm 2.

A new parameter has been introduced in line 5 of this algorithm, the so-called **learning rate**, represented here by the Greek letter γ. By tuning this parameter, a user can regulate how long the step will be from the previous to the next **w** vector. Using this mathematical device, a trade-off should be attained: for too large values of γ, the algorithm can become unstable and miss its target; for too small ones, it could take too long for the algorithm to hit it.

Algorithm 2 The ADALINE learning algorithm

```
1: $\mathbf{w} \leftarrow \mathbf{w}_0$     %random initialisation
2: Compute current error
3: while error $>$ goal error do
4:     for $i = 1 : m$ do
5:         $\mathbf{w} \leftarrow \mathbf{w} + \gamma(\mathbf{y}_i - \widehat{\mathbf{y}}_i)\mathbf{x}_{aug}^{(i)}$
6:     end for
7:     Compute current error
8: end while
```

9.2 Exercises

Solve the following exercises. You can use a calculator. Do not use MATLAB here

Let

$$\mathbf{X} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

$$\mathbf{y} = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{bmatrix}^T$$

be the data for a regression problem.

1. Execute, by hand, the ADALINE algorithm, and complete the Table 9.1 with the first iterations of the algorithm. Take $\gamma = 0.3$ as the learning rate, a goal error of 0.001 and an initial weight vector of

$$\mathbf{w}_o = \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix}^T$$

(NOTE: This table is incomplete, since the algorithm will have to enter twenty-one times the while loop until overcoming the goal error. Nevertheless, this is enough as a didactic exercise.)

TABLE 9.1

Times at while	i	x_i^T	y_i	$\hat{y}_i$	w^T
0	–	–	–	–	[1 1 1 1]
Mean squared error (ADALINE): 13.5					
1	1	[0 0 1]	1	2	[0.7 1 1 0.7]
	2	[0 1 0]	2	1.7	
	3	[0 1 1]	3		
	4	[1 0 0]			
	5				
	6				
	7				
Mean squared error (ADALINE):					
2	1				
	2				

9.3 Practical in MATLAB

General Objective:

To make the student capable of training and using an ADALINE.

Specific Objectives:

- The student should be able to implement an ADALINE to get predictions for a data matrix
- The student should be able to implement the ADALINE learning algorithm.
- The student should be able to graphically represent the ADALINE solution for a regression problem with one-dimensional data.

Materials:

Computer, MATLAB.

Procedure:

Solve the following exercises in MATLAB

1. Create a script and save it following the format `Pr9_nameLastname.m`. You will write your code for the next numerals in this script.

2. Load into the *Workspace* the matrices contained in the file `matricesPr9.mat`. This file can be found in the folder *Data Sets*.

3. Create a function "`adalineOutput(X,w)`", which takes a data matrix `X` and a weight vector `w`, and computes the vector `yhat` of predictions given by the corresponding ADALINE.

4. Use your function `adalineOutput` to compute the predictions of an ADALINE with weight vector
$$\mathbf{w} = \begin{bmatrix} -2 \\ 5 \end{bmatrix}$$
over the four first instances in the matrix `X` loaded in numeral 2.

5. Create a function "`adalineLearning(X,y,w_ini,gamma,goal_error)`", which takes a data matrix `X`, a vector of actual labels `y`, an initial weight vector `w_ini`, a learning rate `gamma` and a `goal_error` to overcome, and executes the ADALINE algorithm. This function must return the weight vectors generated during the learning process, collected in a matrix which we will call "`W_values`". This matrix will contain all the values of `w`, transposed, one below the other, starting with the initial value `w_ini`. Also, this function must display in the *Command Window* the following things: a message telling how many times it is entering the `while` loop; which instance it is analysing at that moment; the initial error; and the error each time it gets out the `while` loop. You can see the answer to numeral 6 to get a better idea of what we are asking for.

6. Use your function `adalineLearning` with learning rate `gamma`=0.2 and a goal error of 2 with the following data:
$$\mathbf{X} = \begin{bmatrix} -1 & -1 \\ -1 & 2 \\ 0 & 0 \\ 1 & 0 \\ 1 & -1 \\ 3 & 0 \end{bmatrix}$$
$$\mathbf{y} = \begin{bmatrix} 1 & 3 & 2 & 2 & 2 & 3 \end{bmatrix}^T$$
$$\mathbf{w}_{ini} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

   (a) What is the resulting `W_values` matrix?

   (b) What are the messages printed in console?

7. Now you will work with some data from a repository, the UCI Machine Learning Repository. Go to: `http://archive.ics.uci.edu/ml/datasets/Concrete+Compressive+Strength` and download the Excel file "`Concrete\_Data.xls`", which is located in the *Data Folder*. Save this file in your *Current Folder*.

8. Once you have an Excel file in your Current Folder, you can read it using the MATLAB command `xlsread`. Use this command to store the data in `Concrete_Data.xls` in a matrix you will call "`dataC`".

9. Form matrices "`X_concrete`" and "`y_concrete`" extracting the adequate columns from `dataC`. To know which attributes are the features of the vectors and which is the dependent variable, read the *Attribute Information* section in the web page you opened above.

10. Use your function `adalineLearning` on this data, with learning rate `gamma` = 0.000001 and a goal error of 82000. Initialise the algorithm with a vector `w_ini` of suitable size (i.e., 9 × 1) and which entries are 0.005 (all of them.). This is,

$$\texttt{w\_ini} = \begin{bmatrix} 0.005 \\ 0.005 \\ \vdots \\ 0.005 \end{bmatrix}_{9\times 1}$$

    (a) How many times does the algorithm enter the `while` loop?

    (b) What is the size of the resulting `W_values` matrix?

    (c) What is the final error?

11. Use your function `adalineLearning` on the matrix `X` and the vector `y` that you loaded in numeral 2, with learning rate `gamma`=0.01 and a goal error of 21. Initialise the algorithm with a vector

$$\texttt{w\_ini} = \begin{bmatrix} 0.1 \\ 0.1 \end{bmatrix}$$

    (a) How many times does the algorithm enter the `while` loop?

    (b) What is the size of the resulting `W_values` matrix?

    (c) What is the final error?

12. Plot the data together with the models given by the weight vectors number 1, 75, 149 and 373 (contained in `W_values`) and arrange them in a 2 × 3 array using `subplot`. See the answer to this numeral to get a better idea of what you are meant to achieve. (*Hint*: In order to plot the models, you can create a vector "`x_plot`" using the MATLAB command **linspace**, and then compute the value of $\widehat{\mathbf{y}}$ for each of the values of this vector. In this way, you will have a "continuous" set of points, which you can plot using `plot`).

9.4 Answers to selected exercises

Exercises

1.

Times at while	i	x_i^T	y_i	$\hat{y}_i$	w^T
0	–	–	–	–	[1 1 1 1]
Mean squared error (ADALINE): 13.5					
1	1	[0 0 1]	1	2	[0.7 1 1 0.7]
	2	[0 1 0]	2	1.7	[0.79 1 1.09 0.7]
	3	[0 1 1]	3	2.58	[0.92 1 1.22 0.83]
	4	[1 0 0]	4	1.92	[1.54 1.63 1.22 0.83]
	5	[1 0 1]	5	3.99	[1.84 1.93 1.22 1.13]
	6	[1 1 0]	6	4.99	[[2.15 2.23 1.52 1.13]
	7	[1 1 1]	7	7.03	[2.14 2.22 1.51 1.12]
Mean squared error (ADALINE): 5.67					
2	1	[0 0 1]	1	3.26	[1.46 2.22 1.51 0.44]
	2	[0 1 0]	2	2.97	[1.17 2.22 1.22 0.44]

Practical

4. $\begin{bmatrix} -10.7879 \\ -9.7778 \\ -8.1616 \\ -7.9596 \end{bmatrix}$

6. (a) $\texttt{W\_values} = \begin{bmatrix} 0 & 0 & 0 \\ 0.2 & -0.2 & -0.2 \\ 0.8 & -0.8 & 1 \\ 1.04 & -0.8 & 1 \\ 1.39 & -0.45 & 1 \\ 1.80 & -0.04 & 0.59 \\ 2.06 & 0.75 & 0.59 \end{bmatrix}$

(b)

```
error = 15.500000
Entering the while for the 1th. time, instance 1...
Entering the while for the 1th. time, instance 2...
Entering the while for the 1th. time, instance 3...
Entering the while for the 1th. time, instance 4...
Entering the while for the 1th. time, instance 5...
Entering the while for the 1th. time, instance 6...
error = 1.375781
```

10. (a) 23 times

    (b) 23691 × 9

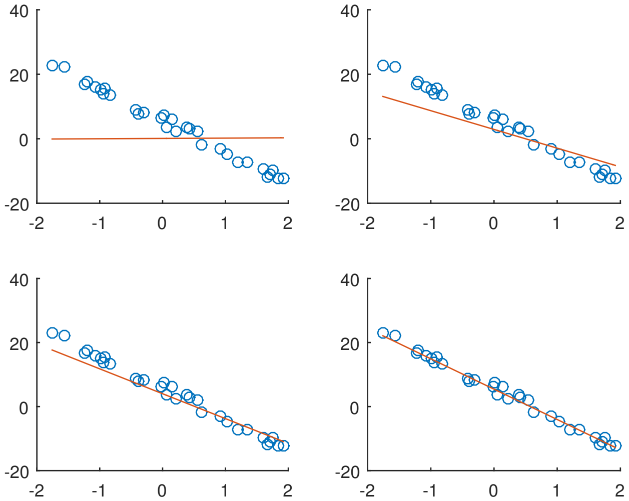
    (c) 81940.199

11. (a) 12 times

    (b) 373 × 2

    (c) 19.92

12.



PART IV

CLASSIFICATION ALGORITHMS

Since classification is a very important task for artificial intelligent systems (think of Facebook recognising a person in a picture, or YouTube recommending you videos), good classification algorithms will be crucial for your career as a Machine Learning scientist. So, you have arrived to the very core of this book, the part that introduces the modern classification algorithms, used in state-of-the-art systems.

This part dedicates three of its chapters to one of the most important classification algorithms ever: *Neural Networks*. Chapters 10 and 11 are here to lay out the conceptual foundations of neural networks and Chapter 12 to show you the MATLAB app that you could use in real projects. The other chapters deal with *Support Vector Machines*, another favourite in the Machine Learning community (Chapter 13) and *k-Nearest Neighbours* (Chapter 14), probably the simplest one but anyway widely used because it does not disappoint in many real applications, in spite of its simplicity.

Chapter 10

Neural Networks: Forward Propagation

10.1 Theoretical Briefing

History. The idea of connecting artificial neurons to form networks has been with us since the days of McCulloch and Pitts. Nevertheless, truly functional neural networks were not possible until the development of the **backpropagation algorithm**, first applied to neural networks, in the form that is used today, by the American scientist Paul Werbos in 1982 [Schmidhuber:2015].

Architecture. The architecture of the neural networks used nowadays is shown in Figure 10.1.

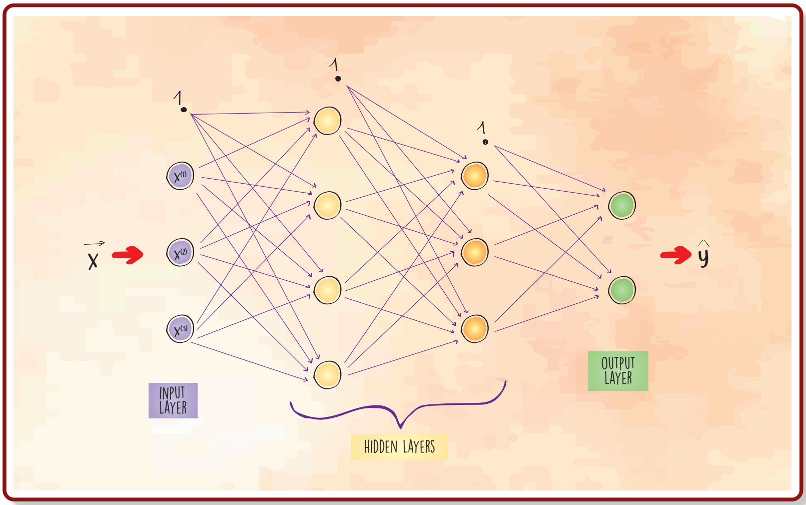


Figure 10.1: Architecture of a neural network

You can see here that the network is formed by layers of neurons with connections between them. There are no connections between neurons of the same layer. The information (in form of feature vectors, as ever) enters the network via the **input layer**, and the result is shown in the **output layer**. In every layer, except in the output layer, there are **bias neurons**, represented here by black little circles. The layers in the middle are called **hidden layers** and serve as intermediate processors of the information. Specifically, Figure 10.1 represents a four-layered network for 3-D feature vectors (since it has three neurons in the input layer), which can classify them into two classes (since there are two neurons in the output layer.)

You can implement a network with an arbitrary number of hidden layers, but the most used has only one hidden layer (i.e. three layers in total), see Figure 10.2. We will only use three-layered networks in this book.

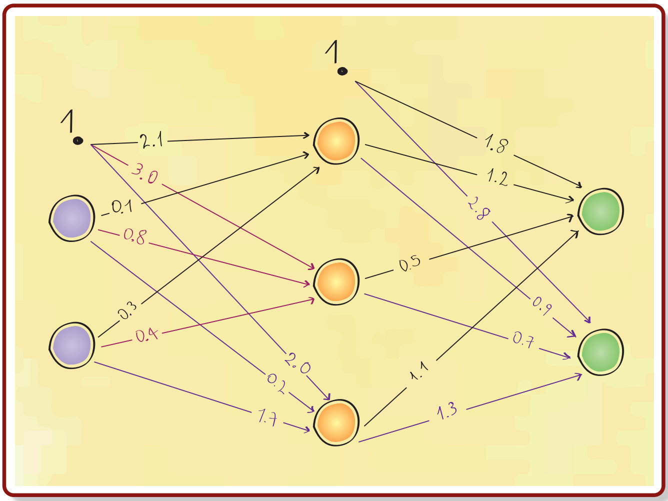


FIGURE 10.2: Example of a neural network with one hidden layer

Notations. Consider the three-layered neural network shown in Figure 10.2. For such a network, the layers are labelled by the numbers

$$c = 1, 2, 3 \tag{10.1}$$

The weight going from the j^{th} neuron of layer c to the i^{th} neuron of layer $(c+1)$ is denoted by:

$$w_{ij}^{(c)} \tag{10.2}$$

Note the order "to-from" in the subscript of this notation. For example, in Figure 10.2 we have that

$$w_{3,1}^{(1)} = 0.2; w_{1,2}^{(2)} = 0.5; w_{2,2}^{(2)} = 0.7; w_{3,0}^{(1)} = 2.0 \tag{10.3}$$

(Note that the bias neuron has been labelled with index "0".)

Weight matrices. The weights coming from a layer c can be put together, all of them, in a matrix $\mathbf{W}^{(c)}$, as shown in Equation 10.4

$$\mathbf{W}^{(c)} = \begin{bmatrix} w_{1,0}^{(c)} & w_{1,1}^{(c)} & w_{1,2}^{(c)} & \cdots \\ w_{2,0}^{(c)} & w_{2,1}^{(c)} & w_{2,2}^{(c)} & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix} \tag{10.4}$$

For example, for the network shown in Figure 10.2, $\mathbf{W}^{(2)}$ would be:

$$\mathbf{W}^{(2)} = \begin{bmatrix} 1.8 & 1.2 & 0.5 & 1.1 \\ 2.8 & 0.9 & 0.7 & 1.3 \end{bmatrix} \tag{10.5}$$

Sums in the hidden layer. Given a data matrix $\mathbf{X}$ and the weight matrix $\mathbf{W}^{(1)}$, you can compute the matrix of sums in the hidden layer, $\mathbf{S}^{(2)}$, using Equation 10.6

$$\mathbf{S}^{(2)} = \mathbf{X}_{aug} \cdot \left(\mathbf{W}^{(1)}\right)^T \tag{10.6}$$

In this case it is a matrix instead of a vector of sums because you are analysing several instances at once. In fact, each row of $\mathbf{S}^{(2)}$ will be a transposed vector of sums for the corresponding instance. Obviously, then, there will be m rows in this matrix.

Activation function. Several activation functions (usually sigmoid functions) are used in neural networks. Among them, the most popular (and the only one we will use in this book) is the so-called **logistic function**. This is a sigmoid function given by:

$$f(s) = \frac{1}{1+e^{-s}} \tag{10.7}$$

Sum in the output layer. For computing the matrix of sums in the output layer, first compute the matrix of activations in the hidden layer, $\mathbf{A}^{(2)}$. Naturally, this must be done by applying the sigmoid function given by Equation 10.7 over each of the elements of $\mathbf{S}^{(2)}$. Then, form a matrix $\mathbf{A}_{aug}^{(2)}$ by appending a column of ones at the left of $\mathbf{A}^{(2)}$ (this is to include the bias neuron of the hidden layer.) Finally, compute $\mathbf{S}^{(3)}$ using Equation 10.8

$$\mathbf{S}^{(3)} = \mathbf{A}^{(2)}_{aug} \cdot \left(\mathbf{W}^{(2)}\right)^{T} \tag{10.8}$$

This matrix will contain, in its rows, the transposed vectors of sums in the output layer.

Output of the neural network. Applying the sigmoid function given by Equation 10.7 over each of the elements of $\mathbf{S}^{(3)}$, we will get the matrix of activations, $\mathbf{A}^{(3)}$. But, what does this matrix say? To decipher its message, you will need to read each row independently. Remember: every row corresponds to an instance. So, for example, read the third row to know what the network's prediction for the third instance is. Now, remember that each row is a transposed vector of activations, so that each element of this third vector corresponds to a neuron in the output layer. What is the prediction of the neural network for this third instance, then? For answering this question, just look at the index of the "most activated" neuron! That will be the index of the class predicted.

10.2 Exercises

Solve the following exercises. You can use a calculator. Do not use MATLAB here

Let

$$\begin{array}{lll}
w^{(1)}_{1,1} = -0.6, & w^{(1)}_{2,1} = -0.6, & w^{(1)}_{1,2} = 0.2, \\
w^{(1)}_{2,2} = 0.2, & w^{(2)}_{1,1} = 0.6, & w^{(2)}_{1,2} = 0.6, \\
w^{(1)}_{1,0} = 0.3, & w^{(1)}_{2,0} = 0.3, & w^{(2)}_{1,0} = -0.6, \\
w^{(1)}_{3,1} = -0.6, & w^{(1)}_{3,2} = 0.2, & w^{(1)}_{3,0} = 0.3, \\
w^{(2)}_{2,0} = 0.6, & w^{(2)}_{2,1} = -0.6, & w^{(2)}_{2,2} = -0.6, \\
w^{(2)}_{2,3} = -0.6, & w^{(2)}_{1,3} = 0.6 &
\end{array}$$

be the weights of a neural network, and let

$$\mathbf{X} = \begin{bmatrix} -3 & 1 \\ -4 & 2 \\ -2 & 3 \\ 5 & 1 \\ 4 & -1 \end{bmatrix}$$

be a data matrix.

1. Draw a schematic diagram of this network, writing down the numerical values of the weights over the corresponding arrows in your plot.

2. Write down the matrices $\mathbf{W}^{(1)}$ and $\mathbf{W}^{(2)}$.

3. Compute the matrix of activations in the hidden layer of this neural network, for the data contained in $\mathbf{X}$.

4. Compute the vector of activations in the output layer for the 4th instance.

5. Compute the output $(\widehat{\mathbf{y}})$ given by this network for matrix $\mathbf{X}$.

10.3 Practical in MATLAB

General Objective:

To make the student capable of using a neural network with one hidden layer to compute the predicted classes for a data matrix.

Specific Objectives:

- The student should be able to implement a neural network with one hidden layer.
- The student should be able to use the implemented network to get the predictions for any input data.
- The student should be able to evaluate the performance of the implemented network.

Materials:

Computer, MATLAB.

Procedure:

Solve the following exercises in MATLAB

1. Create a script and save it following the format `Pr10_nameLastname.m`. You will write your code for the next numerals in this script.
2. Load into the *Workspace* the matrices contained in the file `matricesPr10.mat`. This file can be found in the folder *Data Sets*.
3. Write a function "`nnOutput(X,W1,W2)`", which takes as arguments a data matrix `X` and the two weight matrices, `W1` and `W2`, of a neural network with one hidden layer. This function must process the data using forward propagation and return a vector "`yhat`" of predictions given by the network for the feature vectors contained in `X`. These predictions must be the classes (more specifically, the indexes of the classes) corresponding to the instances. (*Challenge*: Do NOT use `for` loops when writing this function.) Write your function in such a way that it accepts feature vectors of any dimensionality.
4. Try your function `nnOutput` with the weight matrices `W1` and `W2` you loaded in numeral 2, to get the predictions ($\hat{y}$) for the following feature

vectors:

$$\mathbf{x}_1 = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}, \; \mathbf{x}_2 = \begin{bmatrix} -2 \\ 3 \\ 1 \end{bmatrix}, \; \mathbf{x}_3 = \begin{bmatrix} 5 \\ -5 \\ 4 \end{bmatrix}$$

(NOTE: you might want to build a data matrix $\mathbf{X}$ with these feature vectors to feed the neural network. If you do so, you will get all the predictions collected in a single vector $\widehat{\mathbf{y}}$, as usual.)

5. One of the matrices loaded in numeral 2 is called "`X_dig`" and contains the images of 25 hand-written digits [a]. Each of these images was originally a 20 × 20 matrix that has been transformed into a feature column vector and then transposed to build the matrix `X_dig`, in the standard Machine-Learning way. Use the MATLAB command `reshape` to get the matrices assembled again and visualise them using `imshow`. Write down the y vector corresponding to this `X_dig` matrix. (Use as labels the same numbers you see when visualising the images, with one exception: the digit "0" must be written with label "10". This is because indexes are natural numbers, in which the zero is not included.)

6. Use your function `nnOutput` to compute the predictions of the neural network with weight matrices `W1_dig` and `W2_dig` (also loaded in numeral 2) over the `X_dig` matrix.

7. Compute the confusion matrix for the predictions obtained in the previous numeral.

8. Looking at the confusion matrix obtained, answer the following questions:

   (a) With which number is the "4" mistaken by the network?

   (b) With which number is the "6" mistaken by the network?

   (c) With which number is the "0" mistaken by the network?

[a] Taken from the Coursera course *Machine Learning – Stanford University*, Week 5 (Neural Networks Learning), Programming Assignment.

10.4 Answers to selected exercises

Exercises

2. $\mathbf{W}^{(1)} = \begin{bmatrix} 0.3 & -0.6 & 0.2 \\ 0.3 & -0.6 & 0.2 \\ 0.3 & -0.6 & 0.2 \end{bmatrix}$

$\mathbf{W}^{(2)} = \begin{bmatrix} -0.6 & 0.6 & 0.6 & 0.6 \\ 0.6 & -0.6 & -0.6 & -0.6 \end{bmatrix}$

3. $\mathbf{A}^{(2)} = \begin{bmatrix} 0.9089 & 0.9089 & 0.9089 \\ 0.9569 & 0.9569 & 0.9569 \\ 0.8909 & 0.8909 & 0.8909 \\ 0.0759 & 0.0759 & 0.0759 \\ 0.0911 & 0.0911 & 0.0911 \end{bmatrix}$

4. $\mathbf{a}^{(3)} = \begin{bmatrix} 0.3862 \\ 0.6138 \end{bmatrix}$

5. $\widehat{\mathbf{y}} = \begin{bmatrix} 1 & 1 & 1 & 2 & 2 \end{bmatrix}^T$

Practical

4. $\widehat{\mathbf{y}} = \begin{bmatrix} 1 \\ 1 \\ 2 \end{bmatrix}$

5. $\mathbf{y} = \begin{bmatrix} 1 \\ 6 \\ 4 \\ 7 \\ 7 \\ 9 \\ 8 \\ 3 \\ 6 \\ 1 \\ 10 \\ 7 \\ 4 \\ 4 \\ 9 \\ 6 \\ 6 \\ 8 \\ 8 \\ 5 \\ 1 \\ 2 \\ 8 \\ 10 \\ 4 \end{bmatrix}$

7. $\begin{bmatrix} 3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \end{bmatrix}$

8 (a). With digit “9” (which is quite reasonable.)

(b). With digit “5” (which is quite reasonable.)

(c). With digit “8” (which is even more reasonable, if you look at the image of this instance. Even a human classifier could mistake this “0” for an “8”!)

Chapter 11

Neural Networks: Validation and Test

11.1 Theoretical Briefing

Hyperparameters. When you train a machine learning algorithm, you "tune" the values of its **parameters**. For example, in a three-layered neural network you will compute the weight matrices $\mathbf{W}^{(1)}$ and $\mathbf{W}^{(2)}$, using the backpropagation algorithm. There are other values, however, that are not pre-determined and are not computed during the training process. These values are related to the configuration of the algorithm, and are called **hyperparameters**. For example, in a three-layered neural network some hyperparameters are: the number of neurons in the hidden layer, the maximum number of iterations, etcetera.

Overfitting and underfitting. In Chapter 6, we introduced two common problems a machine learning algorithm can run into: **overfitting** and **underfitting**. We saw that neither fitting "too tightly" nor "too loosely" are good things; we always look for an intermediate, equilibrated solution. Here we will see how to find this intermediate solution through a process called **validation**, which looks for the values of the hyperparameters that avoid overfitting and underfitting. The selection of these values is sometimes called **model selection**.

Training, validation and test sets. The process of validation begins with the splitting of the data set into three parts: training set (around 70% of the instances), validation set (15%) and test set (15%). A common practice (not used in this book) is to randomly shuffle the instances before splitting the data set. Warning: when you split the data set, you must split both the $\mathbf{X}$ matrix and the $\mathbf{y}$ vector correspondingly, so that you will end up with three matrices (usually named $\mathbf{X}_{train}$, $\mathbf{X}_{val}$ and $\mathbf{X}_{test}$) and three vectors ($\mathbf{y}_{train}$, $\mathbf{y}_{val}$ and $\mathbf{y}_{test}$.)

Validation step. After the splitting, you must perform the validation step, which consists in training the algorithm (the neural network, in this case) *with the training set*. The result of training, as always, will be the matrices $\mathbf{W}^{(1)}$ and $\mathbf{W}^{(2)}$. Using them, together with the $\mathbf{X}_{val}$ set, compute $\widehat{\mathbf{y}}_{val}$. Then compute the validation error (the one between $\widehat{\mathbf{y}}_{val}$ and $\mathbf{y}_{val}$.) Repeat this for several possible combinations of the hyperparameters and choose the combination with the lowest validation error. We will assign here the adjective "optimal" both to these hyperparameters and to the respective parameters.

Test step. Finally, use a neural network with the optimal parameters and hyperparameters and perform forward propagation over $\mathbf{X}_{test}$ to compute $\widehat{\mathbf{y}}_{test}$. Compute the test error (the one between $\widehat{\mathbf{y}}_{test}$ and $\mathbf{y}_{test}$.) You can also compute other performance measures here. By computing performance measures in the test set, you are assuring the objectivity and fairness of the evaluation.

11.2 Exercises

Solve the following exercises. You can use a calculator. Do not use MATLAB here

1. Suppose you perform the validation step for a classification problem with a neural network, in order to decide the optimal value of the hyperparameter "hidden layer size". You train the network in a certain training set and evaluate its performance both in the training set and the validation set, obtaining the following results:

Hidden layer size	Training error	Validation error
5	0.5	0.3
10	0.4	0.1
15	0.1	0.2
20	0.2	0.4

Which hidden layer size would you choose?

2. Suppose you have a data matrix $\mathbf{X}$ of dimensions 2100×180 and a vector $\mathbf{y}$ of labels which dimension is 2100 x 1. You split this set into three sets: training (70% of the data), validation (15%) and test (15%) sets.

   (a) What are the dimensions of matrix $\mathbf{X}_{train}$?

   (b) What are the dimensions of matrix $\mathbf{y}_{train}$?

   (c) What are the dimensions of matrix $\mathbf{X}_{validation}$?

   (d) What are the dimensions of matrix $\mathbf{y}_{validation}$?

   (e) What are the dimensions of matrix $\mathbf{X}_{test}$?

   (f) What are the dimensions of matrix $\mathbf{y}_{test}$?

11.3 Practical in MATLAB

General Objective:

To make the student capable of performing validation and test steps when solving a classification problem.

Specific Objectives:

- The student should be able to understand and use a function written by the author of this book to perform the backpropagation algorithm and make a neural network "learn" the weight matrices.
- The student should be able to perform the validation step to determine the optimal values of the parameters for a neural network.
- The student should be able to perform the test step to assess the ability of a neural network to generalise.

Materials:

Computer, MATLAB.

Procedure:

Solve the following exercises in MATLAB

1. Create a script and save it following the format `Pr11_nameLastname.m`. You will write your code for the next numerals in this script.
2. Go to `https://archive.ics.uci.edu/ml/datasets/breast+cancer+wisconsin+(original)` and click on *Data Folder* ▷ *breast-cancer-wisconsin.data.* This data was obtained from the University of Wisconsin Hospitals, Madison, from Dr. William H. Wolberg [16], and concerns a very important problem: benign-malignant classification to detect breast cancer. This particular data set consists of 9 features of the cells (please read the *Attribute Information* to know more about this), as seen in samples taken by means of breast biopsy from 699 patients.
3. Copy the data found in the previous numeral and paste it into Excel, in order to generate a .xlsx file. You shall call this file "`cancerData.xlsx`". (*Hint*: to do this after pasting the data in Excel, go to Data, *Text to Columns*, etcetera.)

4. Save the `cancerData.xlsx` file in your *Current Folder* and load it in your Worspace using the MATLAB function `xlsread`. Store the data in a matrix “`dataC`”.

5. Form the matrix `X` as follows: extract from `dataC` the columns corresponding to the features “Clump Thickness”, “Uniformity of Cell Size”, “Uniformity of Cell Shape”, “Marginal Adhesion”, “Single Epithelial Cell Size”, “Bland Chromatin”, “Normal Nucleoli” and “Mitoses”. We will leave aside the feature “Bare Nuclei”, because it contains missing values, represented by question marks (?) in the original set and by “`NaN`” in the matrix `dataC`.

6. Form the vector y extracting the last column of matrix `dataC`. However, you should be aware of the fact that, in this column, “2” stands for “benign” and “4” stands for “malignant” (as you can read in the Attribute Information). This is not a good thing. In order to use a neural network on this data, a pair of suitable labels would be: “1” for “benign” and “2” for malignant. So please change the label “2” for “1” and label “4” for “2” in your y vector. (*Challenge*: Do NOT use a `for` loop to do this.)

7. Now, you are going to split the `X` and `y` matrices into three parts: the training set (489 instances), the validation set (105 instances) and the test set (the remaining 105). Specifically, take for your training set all the instances from instance number 1 to instance number 489; for validation, from instance number 490 to 594; and for test, from 595 to the end.

8. In the folder *MATLAB Functions* you will find a function named “`nnLearning.m`”. Save it to your *Current Folder* to be used in the next numerals. This function takes as arguments a data matrix `X`, a label vector `y`, the number of classes involved and the following parameters: hidden layer size, a regularisation coefficient λ and the maximum number of iterations for the optimisation function. With this information, the function performs the backpropagation algorithm to learn the weight matrices $\mathbf{W}^{[1]}$ and $\mathbf{W}^{[2]}$, which are returned by the function in a cell array. [a]

9. Now you are going to perform the validation step to decide the optimal values for the hyperparameters “hidden layer size”, “regularisation coefficient λ” and “maximum number of iterations”. For doing this, effectuate a grid search (i.e., explore every possible combination) for the following values of the hyperparameters:

$$\begin{gathered} \textit{hidden layer size} \in \{1,2,3,4,5\} \\ \lambda \in \{0.5,1,1.5\} \\ \textit{maximum number of iterations} \in \{1,2,3,4,\ldots,20\} \end{gathered}$$

For each of these possible combinations (300 in total), do the following: (a) Run the function `nnLearning` on the training set; (b) Recover the matrices $\mathbf{W}^{[1]}$ and $\mathbf{W}^{[2]}$ returned by the function; (c) Use your function `nnOutput` *over the validation set* with these matrices to obtain a `yval_hat`; (d) Compute the validation error. Register all the obtained values in a matrix containing the information like this:

hidden layer size	λ	maximun number of iterations	Validation error
1	0.5	1	
1	0.5	2	
1	0.5	3	
1	0.5	4	
⋮	⋮	⋮	⋮
5	1.5	20	

(*Hint*: An efficient way for doing this could be to use nested `for` loops and fill one row of the matrix in each iteration. Obviously, the size of this matrix will be 300×4)

10. The next step in performing model selection is to select the combination with the lowest validation error. Look at the table obtained in the previous numeral and search for this lowest value. What is the corresponding hidden layer size? What is the corresponding λ? What is the corresponding maximum number of iterations? (Name this variables `hidd_opt`, `lambda_opt` and `maxIter_opt`, correspondingly.)

11. Run the function `nnLearning` on the training set, using `hidd_opt`, `lambda_opt` and `maxIter_opt` as parameters. This is to get the "optimal" weight matrices, the ones corresponding to the minimal validation error. Recover these matrices from the cell array returned by `nnLearning` and call them `W1_opt` and `W2_opt`. You can see the correct matrices in the *Answers* section.

12. Now you are going to perform the test step. Use your function `nnOutput` *over the test set* to obtain a "`ytest_hat`". Obviously, you have to use `W1_opt` and `W2_opt` for it.

13. How well did your neural network perform in the test set? To answer this, you can visually compare both `ytest` and `ytest_hat`. Also, compute:

    (a) The test error.

    (b) The confusion matrix.

    (c) The number of true positives, false positives, false negatives and true negatives for class "malignant". Discuss these numbers with your classmates.

[a] AUTHOR ATTRIBUTION: Function `nnLearning` was written by the author of this book, based on the material available online for the Programming Assignment of Week 5, Machine Learning course on Coursera, by Andrew Ng, Stanford University. The `fmincg` function inside it was written by Carl Edward Rasmussen, © 2001 and 2002.

11.4 Answers to selected exercises

Exercises

1. 10 neurons in the hidden layer
2. (a) 1470 × 180

   (b) 1470 × 1

   (c) 315 × 180

   (d) 315 × 1

   (e) 315 × 180

   (f) 315

Practical

10. `hidd_opt` = 3

    `lambda_opt` = 0.5

    `maxIter_opt` = 17

11. $$\texttt{W1\_opt}=\begin{bmatrix} 1.9858 & -0.1206 & -0.1630 & -0.1237 & -0.0325 & -0.0991 & -0.2809 & 0.0092 & -0.0698 \\ 1.9858 & -0.1206 & -0.1630 & -0.1237 & -0.0325 & -0.0991 & -0.2809 & 0.0092 & -0.0698 \\ 1.9858 & -0.1206 & -0.1630 & -0.1237 & -0.0325 & -0.0991 & -0.2809 & 0.0092 & -0.0698 \end{bmatrix}$$

    $$\texttt{W2\_opt}=\begin{bmatrix} -2.4088 & 2.8936 & 2.8936 & 2.8936 \\ 2.3231 & -2.8746 & -2.8746 & -2.8746 \end{bmatrix}$$

13. (a) 0.019

    (b)

		PREDICTED	
		benign	**malignant**
Actual	**benign**	81	2
	malignant	0	22

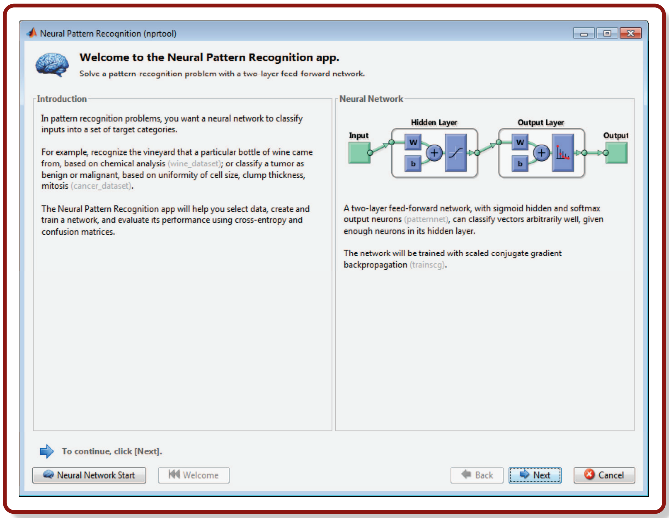
    (c) True positives: 22
    False positives: 2
    False negatives: 0
    True negatives: 81

Chapter 12

Neural Network Pattern Recognition App

12.1 Theoretical Briefing

The app. You can find the *Neural Net Pattern Recognition* app (see Figure 12.1) in the *APPS* section of MATLAB. Although it is a quite user-friendly app, we will briefly describe it in this section. For more information, you can go to the *Documentation* page for this app.



Neural Pattern Recognition (nprtool)

Welcome to the Neural Pattern Recognition app.

Solve a pattern-recognition problem with a two-layer feed-forward network.

Introduction

In pattern recognition problems, you want a neural network to classify inputs into a set of target categories.

For example, recognize the vineyard that a particular bottle of wine came from, based on chemical analysis (wine_dataset); or classify a tumor as benign or malignant, based on uniformity of cell size, clump thickness, mitosis (cancer_dataset).

The Neural Pattern Recognition app will help you select data, create and train a network, and evaluate its performance using cross-entropy and confusion matrices.

Neural Network

A two-layer feed-forward network, with sigmoid hidden and softmax output neurons (patternnet), can classify vectors arbitrarily well, given enough neurons in its hidden layer.

The network will be trained with scaled conjugate gradient backpropagation (trainscg).

To continue, click [Next].

Figure 12.1: Interface of the *Neural Net Pattern Recognition* app

Technical features. The *Neural Network Pattern Recognition* app trains a neural network with one hidden layer, performing validation and test steps automatically. It then displays several performance measures: misclassification error, confusion matrices, cross entropy and ROC curves. Also, it allows you to save the weight matrices to use them later. These matrices, together with more information, are stored in an object whose name by default is “net”. To save this object, go to the *Save Data to Workspace* section, in the *Save Results* window.

Binary matrix of labels. Do not try to feed the *Neural Net Pattern Recognition* app with our typical $\mathbf{y}$ vector of labels. Instead, transform the $\mathbf{y}$ vector into what we have called a **binary matrix of labels**. We call it “binary” because it is a matrix formed only by zeros and ones, where the position (column index) of a “1” in a certain row indicates the label for the instance corresponding to that row, and the rest of entries are zero.

Glossary. Some technical terminology is different in this MATLAB app from what we have been using in this book. These are the terms you might get confused about:

Inputs.- Data matrix, $\mathbf{X}$

Targets.- Binary matrix, $\mathbf{Y}_{bin}$

Samples.- This term appears in the *Select Data* window and refers to the instances. Notice that, by default, this app assumes that the instances are organised in columns, quite the contrary to what we are used to. So, you will probably need to change the setting “Samples are:” to “matrix rows”.

Percent error.- This term appears in the *Train Network* window, and refers to our misclassification error.

12.2 Exercises

Solve the following exercises. You can use a calculator. Do not use MATLAB here

1. The *Neural Net Pattern Recognition* app in MATLAB accepts as “Targets” only binary matrices. Write down the binary matrix, $\mathbf{Y}_{bin}$, corresponding to the following $\mathbf{y}$ vector:

$$\mathbf{y} = \begin{bmatrix} 3 & 3 & 2 & 5 & 4 & 1 \end{bmatrix}^T$$

12.3 Practical in MATLAB

General Objective:

To make the student capable of using the *Neural Net Pattern Recognition* app of MATLAB.

Specific Objectives:

- The student should be able to generate the data to feed the neural network provided by the *Neural Net Pattern Recognition* app of MATLAB.
- The student should be able to train the network and save it for future use.
- The student should be able to use the saved net to make predictions over new data.

Materials:

Computer, MATLAB.

Procedure:

Solve the following exercises in MATLAB

1. Create a script and save it following the format `Pr12_nameLastname.m`. You will write your code for the next numerals in this script.

2. Go to `https://archive.ics.uci.edu/ml/datasets/iris` and click on *Data Folder* ▷ *iris.data.* There you will find the so called "Iris data set", perhaps the most famous data set in the Machine Learning community. Iris is a genus of plants, which many species can be identified by some features of their flowers. You can read in the *Attribute Information* which features are included in this set.

3. Copy the data found in the previous numeral and paste it into Excel, in order to generate a .xlsx file. You shall call this file "`irisData.xlsx`". (*Hint:* to do this after pasting the data in Excel, go to *Data*, *Text to Columns*, etcetera.) In the 5^{th} column, replace the names of the species for a label: 1 for *setosa*, 2 for *versicolor* and 3 for *virginica*.

4. Save the `irisData.xlsx` file in your *Current Folder* and load it in your *Worspace* using the MATLAB function `xlsread`. Store the data in a matrix `dataIris`.

5. Form the matrix `X` with the columns corresponding to the four features mentioned in the *Attribute Information*. You should end up with a 150×4 matrix at this point.

6. Form the vector `y` with the labels (contained in the 5^{th} column of matrix `dataIris`) of the species of the flowers. You should end up with a 150×1 vector at this point.

7. Write a function "`lab2bin(y)`", which takes a vector `y` of labels and returns the same information but in a binary way. This is, it returns the corresponding $\mathbf{Y}_{bin}$ matrix.

8. Use your function `lab2bin` to obtain the `Ybin` matrix corresponding to your Iris data set.

9. Run your script. After doing this, your data has been loaded and you are ready to use the Neural Net Pattern Recognition app. You can find it in the *APPS* tab in MATLAB.

10. Open the app and then click on *Next*. You will end up in the *Select Data window*. Select as *Inputs* the matrix `X` you got in numeral 5. Select as *Targets* your `Ybin` matrix. Select the option *Matrix rows*, instead of the *Matrix columns* option set by default.

11. Clicking on *Next* you will reach the *Validation and Test Data*, and *Network Architecture* windows. In the first, leave unchanged the values set by default: 15% for validation and 15% for testing. In the second, change the number of hidden neurons to 20. Click on *Next* to create the network.

12. In the *Train Network* window, click on *Train*. This action makes the app execute the backpropagation algorithm to obtain the optimal weight matrices $\mathbf{W}^{(1)}$ and $\mathbf{W}^{(2)}$, performing the validation and test steps as well. After this, you can click on *Plot Confusion* to see the corresponding confusion matrices, which should have a "fat" diagonal. Also, the misclassification errors are displayed in the table of results.

13. Click on *Next* several times until you reach the *Save Results* window. In the section *Save Data to Workspace*, select the option "Save network to MATLAB network object named:", changing the default name "net" for "netIris". Click on *Save Results* and *Finish*.

14. Suppose you find five new iris flowers and want to know which species each of them belongs to. You measure their features and find the following results:

ID	Sepal, length [cm]	Sepal, width [cm]	Petal, length [cm]	Petal, width [cm]
1	5.1	3.5	1.4	0.2
2	6.2	3.1	5.3	1.9
3	5.1	3.1	1.3	0.2
4	6.8	3.3	4.1	1.1
5	5.9	2.9	4.6	1.3

Use your previously-trained neural network to get the predicted classes for them. To do this, use the object `netIris` you got in the previous numeral, writing the following line of code:

```
>> predictions = netIris(Xnew’);
```

where `Xnew` is a data matrix containing the five feature vectors corresponding to the new flowers.

What species does the network predict for each flower? Check the *Answers* section to see if your predictions are correct.

12.4 Answers to selected exercises

Exercises

1. $\mathbf{Y}_{bin} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix}$

Practical

14.

ID	Species predicted
1	*Setosa*
2	*Virginica*
3	*Setosa*
4	*Versicolor*
5	*Versicolor*

Chapter 13

Support Vector Machines

13.1 Theoretical Briefing

Linearly separable datasets. Suppose you have a two-dimensional dataset for a classification problem, with two classes. If the vectors corresponding to those two classes can be separated by a straight line when represented on a Cartesian plane, then we would call this set a linearly separable dataset. In higher dimensions, the same concept corresponds to any set whose vectors can be separated by a hyperplane. A hyperplane is a geometrical entity with a linear equation of the form

$$a_d x^{(d)} + a_{d-1} x^{(d-1)} + \cdots + a_1 x^{(1)} + a_0 = 0 \tag{13.1}$$

where d is the dimension of the feature space.

Support Vector Machines. By definition, it is always possible to find a straight line which separates regions of different classes in a linearly separable dataset (in two dimensions.) But it seems obvious that, if such a line exists, then an infinite number of other lines can be found that satisfy the same condition. **Support vector machines** (SVM) is an algorithm that finds the optimal of these lines, where "optimal" means that the line is as far as possible from the vectors in both sides. Roughly speaking. In higher dimensional spaces, just change the line for a hyperplane and the concept remains the same.

Support vectors. Let us get back to the two-dimensional case. The work done by SVM can be understood as follows. Imagine a linearly separable dataset with two classes. Then imagine a straight line (the "decision boundary") separating both classes. Now imagine two lines, parallel to the decision boundary and passing through some training vectors. These lines should satisfy the condition that no vectors are located in the space between them (see Figure 13.1 to help your imagination.) Let us call this empty space the *slab*. Then, what

SVM does is to find the solution for which the slab is as wide as possible. The vectors through which the borders of the slab pass are what we call **support vectors**.

Kernels. All our explanations about the SVM algorithm have so far assumed that the dataset is linearly separable. What if not? In this case, we can use a mathematical trick: to map the feature space to a higher dimensional space. By doing so, we can transform a set of, say, two-dimensional vectors, into a set of three-dimensional vectors; in such a way that this new set is separable by a plane even though the original set was not separable by a line. The usage of kernels take advantage of this property, although the mathematical details are beyond the scope of this book. Suffice to say, use kernels when you need to deal with non linearly-separable datasets.

The MATLAB function for SVM. The recommended function to apply the SVM algorithm in MATLAB is `fitcsvm`. As you can see in the documentation, one possible syntax is:

```
SVMModel = fitcsvm(X,y)
```

where `X` is the data matrix $m \times d$ that we are well accustomed to, `y` is the label vector $m \times 1$, and `SVMModel` is a structure containing the parameters obtained by the training and the hyperparameters involved. Some of the fields returned by SVMModel are: "`SupportVectors`", a matrix containing the support vectors, one in each row; "`Beta`", a vector of coefficients of the equation for the decision boundary; and, "`Bias`", a scalar representing the bias for this equation.

Equation for the decision boundary. Given the vector of coefficients `Beta` (that we will call "$\boldsymbol{\beta}$" here) and `Bias` (that we will call "b"), the prediction of this SVM for any vector $\mathbf{x}_i$ is:

$$\widehat{\mathbf{y}}_i = \begin{cases} 1 & \text{if} \quad f(\mathbf{x}_i) \geq 0 \\ -1 & \text{if} \quad f(\mathbf{x}_i) < 0 \end{cases} \tag{13.2}$$

where

$$f(\mathbf{x}) = \mathbf{x}^T \boldsymbol{\beta} + b = 0 \tag{13.3}$$

Decision boundary, two-dimensional case. Displaying the vectors for the two-dimensional case, the last equation becomes:

$$\begin{bmatrix} x^{(1)} & x^{(2)} \end{bmatrix} \begin{bmatrix} \beta^{(1)} \\ \beta^{(2)} \end{bmatrix} + b = 0$$

This is,

$$x^{(1)}\beta^{(1)} + x^{(2)}\beta^{(2)} + b = 0$$

This is the equation of the decision boundary (a straight line, in this case). Using some basic analytic geometry, it can be shown that the slope and the intercept with the vertical axis for this line are:

$$\text{slope} = -\frac{\beta^{(1)}}{\beta^{(2)}}$$

$$\text{intercept} = -\frac{b}{\beta^{(2)}}$$

13.2 Exercises

Solve the following exercises. You can use a calculator. Do not use MATLAB here

Let

$$\mathbf{X} = \begin{bmatrix} 4 & 5 \\ -0.5 & 3 \\ 5 & 3 \\ 2.5 & 1.5 \\ 1 & 1 \end{bmatrix}$$

and

$$\mathbf{y} = \begin{bmatrix} -1 & 1 & -1 & 1 & 1 \end{bmatrix}^T$$

be the data for a classification problem

1. Plot this data on paper, in a Cartesian plane, using different markers and colours for different classes.
2. Using the MATLAB function `fitcsvm`, we get the following matrix in the field "SupportVectors":

$$\begin{bmatrix} 5 & 3 \\ 2.5 & 1.5 \end{bmatrix}$$

   Use this information to identify the support vectors in the graph plotted in the previous numeral. Draw a circle surrounding these support vectors.
3. In the same structure returned by `fitcsvm`, we obtain the number 3 in the `Bias` field and in the `Beta` field, the matrix

$$\begin{bmatrix} -0.59 \\ -0.35 \end{bmatrix}$$

   Use this information to compute the slope and the intercept with the vertical axis of the decision boundary.
4. Using this information, draw the decision boundary in the same graph done in numeral 2.
5. In Figure 13.1, we show (in blue dashed lines) the borders of the "slab" corresponding to the SVM model trained. Find the equation of these borders.

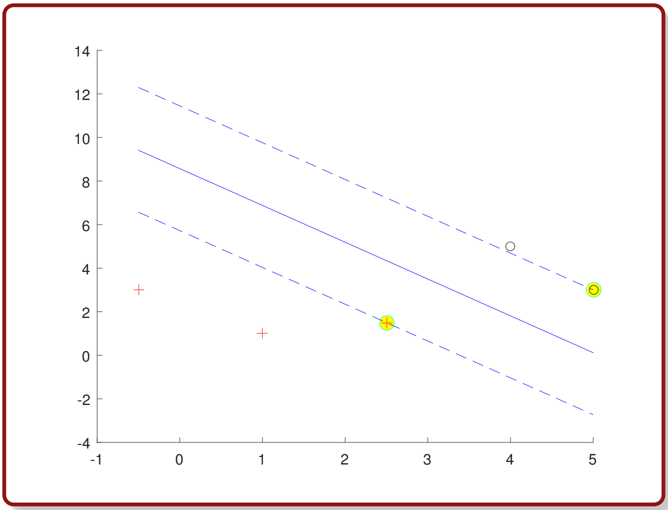


FIGURE 13.1: Plot of the data showing the slab and the support vectors.

13.3 Practical in MATLAB

General Objective:

To make the student capable of using the MATLAB function to perform the Support Vector Machine algorithm and to understand its results.

Specific Objectives:

- The student should be able to use the MATLAB function `fitcsvm` and the variables returned by it to graphically visualise the support vectors computed by the algorithm for a linearly separable two-dimensional dataset.
- The student should be able to plot the “slab” separating the regions of different classes, computed by the SVM algorithm for the same training set, and the respective decision boundary.
- The student should be able to use the model returned by the SVM algorithm to compute the predictions given by the algorithm for new feature vectors.
- The student should be able to train a SVM with a kernel for a non-linearly separable dataset, to use it to compute predictions and to graphically represent its results.

Materials:

Computer, MATLAB.

Procedure:

Solve the following exercises in MATLAB

1. Create a script and save it following the format `Pr13_nameLastname.m`. You will write your code for the next numerals in this script.

2. Load into the workspace the matrix contained in the file `matricesPr13.mat`. This file can be found in the folder *Data Sets*.

3. In numeral 2, you loaded a data matrix `X` and the corresponding labels in `y`, for a classification problem with two classes, -1 and 1. Use your function `plotClasses` to graphically represent this data. Note that this is a linearly separable dataset.

4. Use the MATLAB function `fitcsvm` to execute the SVM algorithm upon the dataset conformed by `X` and `y`. By typing `SVMModel = fitcsvm(X,y);` you will get a structure called "`SVMModel`", containing the information from the training performed by SVM.

5. From the `SVMModel` structure, recover the field named "`SupportVectors`" and store it in a variable "`sv`". Use the information in `sv` to plot the support vectors, surrounding them by a cyan circumference filled with yellow (see the *Answers* section to understand what you are meant to attain.)

6. From the `SVMModel` structure, recover the fields "`Beta`" and "`Bias`" and use this information to compute the slope and the intercept of the corresponding hyperplane (the decision boundary returned by SVM for this problem). In the *Theoretical Briefing* section you can find equations that could help you with these calculations.

7. Use the information computed in the previous numeral to plot the decision boundary, superposing it to the graph generated in numeral 5. You can use the MATLAB function `plot` here.

8. In the same graph, plot the limits of the "slab". This is, plot the straight lines parallel to the decision boundary that pass through the support vectors.

9. In numeral 2, you loaded a matrix `Xnew`, containing 6 feature vectors whose classification we do not know. Use `predict` with the `SVMModel` structure, to figure out what the predictions of this SVM for such vectors are.

10. Use the MATLAB function `gscatter` to plot the feature vectors in `Xnew`, locating them in the same graph attained in numeral 8. Represent by black stars the vectors classified as “-1” and with red stars the ones classified as “1”.

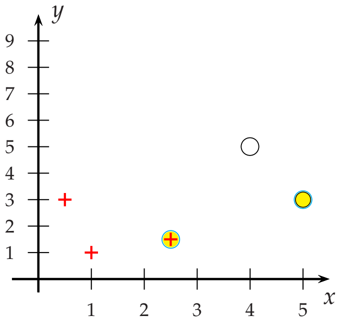
11. In numeral 2, you uploaded a dataset made of matrix `X2` and vector `y2`. If you plot this data, you will see that it is not linearly separable. On this dataset, use `fitcsvm` as you did in numeral 4, this time to get a structure “`SVMModel2`”. Then plot your results in the following way: first, create a grid of points using `meshgrid`, with a pace of 0.5 from one point to the other; second, use `predict` with the `SVMModel2` structure to get the predictions of this SVM for each of the points of the grid; and third, use `gscatter` to represent each point according to its classification. Please note how the algorithm classifies all of the vectors as “-1”, utterly failing to perform its task.

12. Repeat the steps of the previous numeral, but this time use SVM with a polynomial kernel. Look at the documentation for `fitcsvm` to figure out how to do this. In the resulting plot, note how the algorithm coherently classifies the points of the grid (at least most of them!)

13.4 Answers to selected exercises

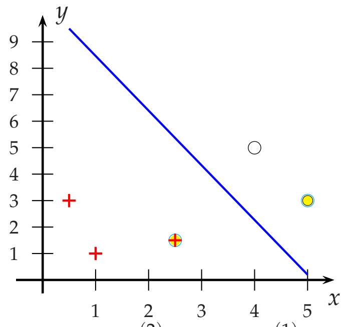
Exercises

2.



3. *Slope* = -1.69; *intercept* = 8.57

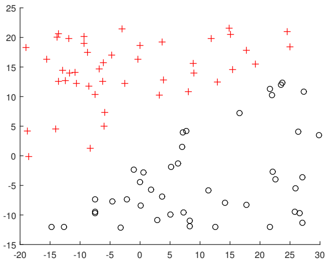
4.



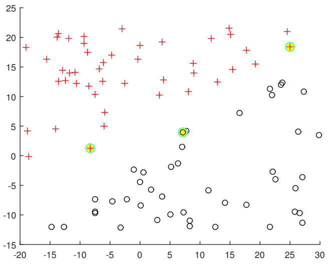
5. Upper border: $x^{(2)} = -1.69x^{(1)} + 11.45$
   Lower border: $x^{(2)} = -1.69x^{(1)} + 5.73$

Practical

3.

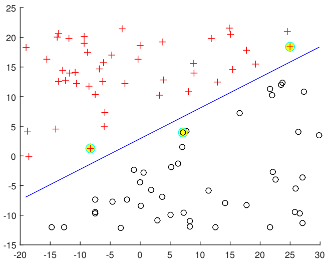


5.

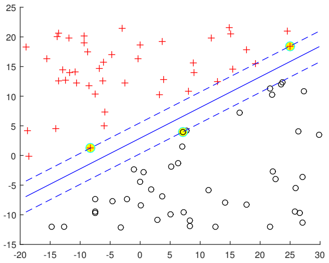


6. slope=0.5170; intercept=2.9171;

7.



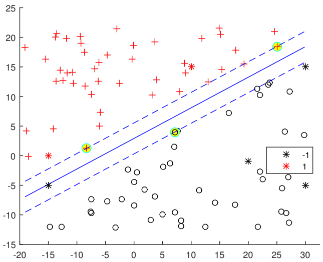
8.



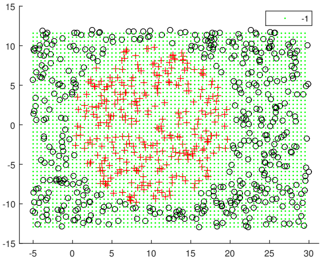
9.

$$\widehat{\mathbf{y}} = \begin{bmatrix} -1 & 1 & 1 & -1 & -1 & -1 \end{bmatrix}^T$$

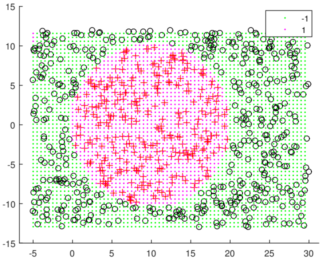
10.



11.



12.



Chapter 14

k-Nearest Neighbours

14.1 Theoretical Briefing

The basic idea. As you have seen many times in this book, a feature vector can be considered as a point in a space of d dimensions (remember, for example, the plots made by your function `plotClasses` in two dimensions.) Given a new instance (a new point in the space of d dimensions), the k-nearest neighbours algorithm is a classification algorithm that assigns to it the class of the k-nearest points in that space. More precisely, since these points can have different classes, it assigns to the new point the class corresponding to the majority of the neighbours.

The algorithm. Let $S = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_m, y_m)\}$ be a training set, and let $\mathbf{x}_{new}$ be a new feature vector, for which we want to predict the class y_{new}. The k-nearest neighbours algorithm is detailed then in Algorithm 3. At line 2, a distance must be computed. You can use any definition of distance, though the most common is the Euclidean distance, Equation 14.1; or the squared Euclidean distance, Equation 14.2

$$d(\mathbf{x}_1, \mathbf{x}_2) = \sqrt{\sum_{j=1}^{d} \left(x_1^{(j)} - x_2^{(j)} \right)^2} \tag{14.1}$$

$$d^2(\mathbf{x}_1, \mathbf{x}_2) = \sum_{j=1}^{d} \left(x_1^{(j)} - x_2^{(j)} \right)^2 \tag{14.2}$$

Graphical interpretation. For 2-D feature vectors, a graphical interpretation is feasible to better understand the k-nearest neighbours algorithm. Let $\mathbf{x}_{new}$ be a new 2-D feature vector, for which we want to predict the class. This vector is just a point in the Cartesian plane, and you can use a drawing compass to find its nearest neighbours. For this, just place the point of the compass on the $\mathbf{x}_{new}$ point and draw circles with increasingly greater radius, until the circle contains k training points; then assign to this $\mathbf{x}_{new}$ vector the class most represented inside the circle. A similar procedure can be conceived for 1-D and 3-D vectors. For the latter ones, a sphere instead of a circle would do the work.

Algorithm 3 The *k-NN* Algorithm

```
for $i = 1 : m$ do
    compute $d(\mathbf{x}_{new}, \mathbf{x}_i)$
    store $d(\mathbf{x}_{new}, \mathbf{x}_i)$ in a vector of distances, $\mathbf{d}$
end for
sort distances in $\mathbf{d}$ from lowest to greatest.
take the $k$ first elements in this sorted $\mathbf{d}$.
assemble a vector $\mathbf{a}$ with the labels corresponding to these $k$ first elements.
$y_{new} \leftarrow \text{mode}(\mathbf{a})$
return $y_{new}$
```

Voronoi Tessellation. Since the k-nearest neighbours algorithm can be applied to any point in the feature space, the dataset determines a division of the space in zones belonging to different classes. A diagram showing this division is a type of **Voronoi tessellation** which can be drawn as follows: scan the space point by point (with an arbitrary step between them) and execute the k-nearest neighbours algorithm for each point. "Paint" each point with a colour distinctive of the class found.

14.2 Exercises

Solve the following exercises. You can use a calculator. Do not use MATLAB here

Let

$$\mathbf{X} = \begin{bmatrix} -2 & 0 \\ -4 & 0 \\ -1 & 2 \\ -2 & 2 \\ 0 & 1 \\ 1 & 3 \\ -5 & 4 \end{bmatrix}$$

and

$$\mathbf{y} = \begin{bmatrix} 0 & 0 & 1 & 1 & 0 & 1 & 0 \end{bmatrix}^T$$

be the data for a classification problem. And let

$$\mathbf{X}_{new} = \begin{bmatrix} -2 & 1.5 \\ -3 & 0 \end{bmatrix}$$

be a matrix whose vectors you want to classify using the k-NN algorithm.

1. On graph paper, draw a Cartesian plane and locate there the vectors contained in **X**, representing with a little black circle the vectors of class 0; and with a red "×" the vectors of class 1.

2. In this same Cartesian plane, locate the first vector contained in $\mathbf{X}_{new}$, representing it with a blue diamond.

3. With a compass centred in this blue diamond, draw a circumference which passes through its nearest neighbour, a circumference through its second-nearest neighbour and a circumference through its fifth-nearest neighbour.

4. So, what is the class of the blue vector? Apply the k-NN criterion with:
   (a) k=1
   (b) k=2
   (c) k=5

5. Compute the squared distances from the second vector in $\mathbf{x}_{new}$ to each of the vectors in **X**. Put these values in a vector called "**d**".

6. Sort the vectors in **X** from the one corresponding to the minimal value in **d** to the one corresponding to the maximum. Form a matrix $\mathbf{X}_{sorted}$ with these vectors.

7. So, what is the class of this second vector? Apply the k-NN criterion with:
   (a) k=1
   (b) k=2
   (c) k=3

14.3 Practical in MATLAB

General Objective:

To make the student capable of using the k-NN algorithm and to graphically represent its results in the two-dimensional case.

Specific Objectives:

- The student should be able to use the k-NN algorithm to get predictions for a new dataset, given a data matrix **X** and the vector of corresponding labels, **y**.

- The student should be able to plot the Voronoi tessellation of the two-dimensional feature space due to a certain dataset.

Materials:

Computer, MATLAB.

Procedure:

Solve the following exercises in MATLAB

1. Create a script and save it following the format `Pr14_nameLastname.m`. You will write your code for the next numerals in this script.

2. Load into the workspace the matrices contained in the file `matricesPr14.mat`. This file can be found in the folder *Data Sets*.

3. Use your function `plotClasses` to visualise the data contained in matrices `X` and `y`. This is the training data for this part of the practical. (*Note*: the plot shown in the *Answers* section represents the class 0 vectors as little black circles and the class 1 vectors as red crosses.)

4. In the same Cartesian plane, locate the feature vectors contained in `Xnew`. Represent them as blue asterisks. (*Note*: We do not have the labels for these vectors, but you can infer their class by looking at their neighbourhoods in this graphical representation.)

5. Write a function "`kNNOutput(X,y,Xtest,k)`", which takes as arguments a data matrix `X`, the corresponding vector `y` of labels, a matrix `Xnew` and the `k` parameter of the k-NN algorithm; and returns a vector "`yhat`" of predicted classes for the vectors.

6. Use your function `kNNOutput` with the matrices `X` and `y` loaded in numeral 2, to get the predictions of the k-NN algorithm for the vectors contained in `Xnew`:

   (a) For k=1

   (b) For k=2

   (c) For k=3

7. Create a function "`plotVoronoi(X,y)`", which takes a data matrix `X` and the corresponding vector `y` of labels; and draws the data and the Voronoi tessellation of the feature space due to these data. (Specifications: make a grid of approximately 40 x 40 points and use the colour cyan to represent class 0, and the colour magenta to represent class 1.)

8. Go to `https://www.kaggle.com/primaryobjects/voicegender` and download the corresponding dataset. The feature vectors here contain 20 acoustic properties of the recorded voice of 3168 individuals, half of them male and half female. The aim of the problem is to guess the gender of the individual from these features. (You can read more in the description shown in the webpage.)[5]

9. Separate the data as follows: for the training set, take the first 1108 male instances, followed by the first 1108 female instances; for the validation set, take the next 238 male instances, followed by the next 238 female instances; for the test set, take the last 238 male instances, followed by the last 238 female instances. After doing this, you should end up with a 2216 x 20 `Xtrain` matrix, a 476 x 20 `Xval` matrix and a 476 x 20 `Xtest` matrix. Vectors `ytrain`, `yval` and `ytest` should be of dimensions 2216x1, 476x1 and 476x1, respectively.

10. Perform the validation step to select the best model (the best value for k, in this case.) For doing this, call your function `kNNOutput` for five different values of k: 1, 2, 3, 4 and 5. For each value, register the corresponding misclassification error (you can use your function `computeMCE` to calculate this) in a table.

11. Which value of k was the best one?

12. With this best value of k, call your function `kNNOutput` to get the predictions `(yhat)` for the test set.

13. Compute the misclassification error for these predictions.

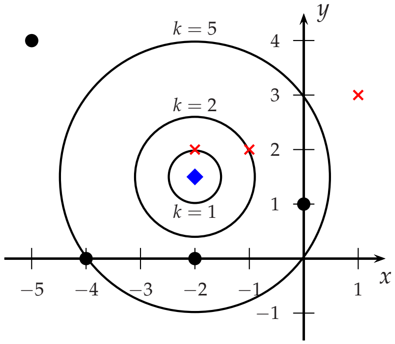
14. Compute the confusion matrix for these predictions.

15. Use your function `computeCPM` to obtain the number of true positives, false positives, false negatives, true negatives, sensitivity and specificity for class “male”.

14.4 Answers to selected exercises

Exercises

3.



4. (a) Class 1

   (b) Class 1

   (c) Class 0

5. $\mathbf{d} = \begin{bmatrix} 1 & 9 & 4 & 5 & 2 & 13 & 32 \end{bmatrix}^T$

6. $\mathbf{X}_{sorted} = \begin{bmatrix} -2 & 0 & -1 & -2 & -4 & 1 & -5 \\ 0 & 1 & 2 & 2 & 0 & 3 & 4 \end{bmatrix}^T$

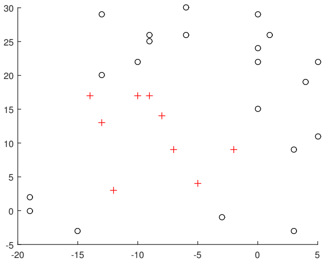
7. (a) 0

   (b) 0

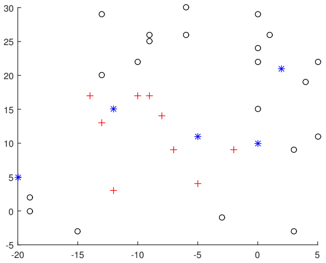
   (c) 0

Practical

3.



4.

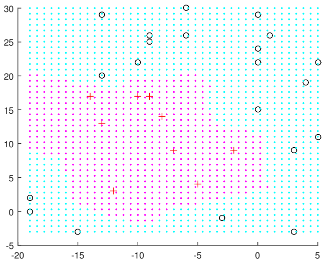


6. (a) $\widehat{\mathbf{y}} = \begin{bmatrix} 0 & 0 & 1 & 1 & 1 \end{bmatrix}^T$

   (b) $\widehat{\mathbf{y}} = \begin{bmatrix} 0 & 0 & 1 & 0 & 1 \end{bmatrix}^T$

   (c) $\widehat{\mathbf{y}} = \begin{bmatrix} 0 & 0 & 1 & 0 & 1 \end{bmatrix}^T$

7.



10.

k	Misclassification error
1	0.3319
2	0.3613
3	0.3151
4	0.3298
5	0.3172

11. $k = 3$

13. 0.395

14.

		PREDICTED	
		Females	**Males**
ACTUAL	**Females**	103	135
	Males	53	185

15.

```
FOR CLASS INDEX 2 AS ‘‘POSITIVE’’:-------------
True positives: 185
False positives: 135
False negatives: 53
True negatives: 103
Sensitivity: 7.773109e-01
Specificity: 4.327731e-01
```

Part V

Problems in Implementation Time

Machine Learning is full of subtleties, and many things just do not result as they are supposed to. In this part, we address a number of those problems that make our lives harder, and see some of the solutions that have been invented to deal with them.

In Chapter 15, we address the problem of dissimilar ranges (when the components of the feature vectors are in very different orders of magnitude); in Chapter 16, you will learn to use some graphical tools to detect the old problems of overfitting and underfitting in your algorithms (and help you make decisions about them), and in Chapter 17 we introduce a famous technique, *Principal Component Analysis*, which is our best weapon against the so-called "curse of dimensionality", an unbearably high dimensionality of the feature vectors that in many cases can be reduced without losing of relevant information.

Chapter 15

Feature Scaling

15.1 Theoretical Briefing

Dissimilar ranges. One of the problems that can arise when applying machine learning to real data is that the features may contain values in very different ranges. For example, the cars of our Practical 2.3 were characterised by the following features: distance travelled, age and engine capacity. The range of the first for a typical second-hand car can be around 100000 km (order of magnitude: 5), while the range for the second could be around 5 years (order of magnitude: 1). These differences usually affect the efficiency of our classification and regression algorithms.

Mean normalisation. One possible solution to the problem of dissimilar ranges is **mean normalisation**. This procedure drags all of the values in a feature vector to the same order of magnitude. We explain this technique as follows:

Let $\mathbf{z}$ be a vector of values z_i, with $i = 1, 2, \ldots, m$. Then each value z_i can be mapped into a new, normalised value, z_i^{norm}, using Equation 15.1.

$$z_i^{norm} = \frac{z_i - \overline{\mathbf{z}}}{max(\mathbf{z}) - min(\mathbf{z})} \tag{15.1}$$

where $max(\mathbf{z})$ is the maximum value in $\mathbf{z}$, $min(\mathbf{z})$ is the minimum, and $\overline{\mathbf{z}}$ is the mean of the values, given by Equation 15.2.

$$\overline{\mathbf{z}} = \frac{1}{m} \sum_{i=1}^{m} z_i \tag{15.2}$$

Standardisation. This is another possible solution to the problem of dissimilar ranges. Let $\mathbf{z}$ be a vector of values z_i, with $i = 1, 2, \ldots, m$. Then each value z_i can be mapped into a new, standardised value, z_i^{stand}, using Equation 15.3.

$$z_i^{stand} = \frac{z_i - \overline{\mathbf{z}}}{\sigma} \tag{15.3}$$

where σ is the standard deviation of the values in $\mathbf{z}$.

15.2 Exercises

Solve the following exercises. You can use a calculator. Do not use MATLAB here

Let

$$\mathbf{X} = \begin{bmatrix} -2.1 & 18000 \\ -1.4 & 20500 \\ 7 & 40000 \\ 7.7 & 30000 \end{bmatrix}$$

be a data matrix.

1. Compute the mean of the values corresponding to the first feature.
2. Compute the mean of the values corresponding to the second feature.
3. Compute the standard deviation of each feature.
4. Compute $\mathbf{X}^{(norm)}$, the matrix of mean-normalised values.
5. Compute $\mathbf{X}^{(stand)}$, the matrix of standardised values.

15.3 Practical in MATLAB

General Objective:

To make the student capable of solving the problem of dissimilar ranges in datasets.

Specific Objectives:

- The student should be able to apply mean normalisation to a dataset.
- The student should be able to apply standardisation to a dataset.

Materials:

Computer, MATLAB.

Procedure:

Solve the following exercises in MATLAB

1. Create a script and save it following the format `Pr15_nameLastname.m` You will write your code for the next numerals in this script.
2. Go to `https://archive.ics.uci.edu/ml/datasets/default+of+credit+card+clients` and download the corresponding dataset. As you can read in the descriptions, this dataset contains the information about payment history and other data (such as gender and marital status) of 30000 clients in Taiwan. This information could help financial institutions to predict whether a client will default on a payment of his/her credit card

or not. Use xlsread to read the corresponding file in MATLAB. Generate the data matrix **X** and the label vector **y**, using label "1" for class "No" and label "2" for class "Yes" ("Yes" = default payment; "No" = not default payment).

3. Divide your dataset in a training set and a test set. For the former, take the 25500 first instances; and the remaining instances for the latter. Call your matrices and vectors "Xtrain", "Xtest", "ytrain" and "ytest". If you have done everything right, you should end up with the following dimensions: Xtrain, 25500 × 23; Xtest, 4500 × 23; ytrain, 25500 × 1; ytest, 4500 × 1.

4. In the folder *MATLAB Functions* you will find a function named "nnLearning2.m". Save it to your *Current Folder* to be used in the next numerals. This function takes as arguments a data matrix X, a label vector y, the number of classes involved and the following parameters: hidden layer size, a regularisation coefficient lambda and the maximum number of iterations for the optimisation function. With this information, the function performs the backpropagation algorithm to learn the weight matrices $\mathbf{W}^{(1)}$ and $\mathbf{W}^{(2)}$, which are returned by the function in a cell array.[a]

5. Use the function nnLearning2 to perform the backpropagation algorithm with the training set (Xtrain and ytrain.) Use as hyperparameters the following: hidden layer size, 100; regularisation coefficient (lambda), 1; maximum number of iterations, 100.

6. With the weight matrices returned by the function in the previous numeral, use your function nnOutput to compute the vector of predictions for the test set. Call this vector "yhat_test".

7. Use confusionmat to compute the confusion matrix between ytest and yhat_test. Also, use your function computeMCE to compute the misclassification error.

8. Now, let us take a minute to analyse the results. We have obtained a misclassification error of just 21.2%. Not bad, uh? Well, actually, the classification was worse than bad, if you think better about it. Take a look at the confusion matrix: the column for the predicted class "Yes" is full of zeros, which means that our neural network has classified all of the clients as "good" clients (i.e., not incurring default payment), regardless of their features. We could say that this network is too "naive". It lacks

the malice necessary to detect the potential defaulters. That is terrible! If you were a banker, you would hate this network!

9. Taking into account the point of view of a banker, very interested in spotting potential defaulters, we will set now the class "Yes" as the "positive" class. (Do not get confused by the fact that the "positives" are the cases in which the clients incur in payment default! It is like in medicine, where "positive" means something negative: that you have the disease.) For this positive class, then, compute the sensitivity (this is, the rate of defaulters correctly identified.) You can use your function `computeCPM` for this.

10. Write a function "`meanNormalise(X)`", which takes a data matrix `X` as argument, and performs mean normalisation on it. This function should return a new data matrix, of the same dimension as the original, with the data normalised.

11. Use your function `meanNormalise` to perform mean normalisation on the matrices `Xtrain` and `Xtest`. Repeat the steps of numerals 5 and 6, but this time with the normalised data. Use the same hyperparameters given in numeral 5. Then, evaluate the performance of this new neural network by computing:

    (a) The misclassification error.

    (b) The confusion matrix.

    (c) The sensibility and specificity for the "positive" class.

12. Write a function "`standardise(X)`", which takes a data matrix `X` as argument, and performs standardisation on it. This function should return a new data matrix, of the same dimension as the original, with the data standardised.

13. Write a function "`standardise(X)`", which takes a data matrix `X` as argument, and performs standardisation on it. This function should return a new data matrix, of the same dimension as the original, with the data standardised.

14. (a) The misclassification error.

    (b) The confusion matrix.

    (c) The sensibility and specificity for the "positive" class.

[a] AUTHOR ATTRIBUTION: Function `nnLearning2` was written by the author of this book, based on the material available online for the Programming Assignment of Week 5, *Machine Learning* course on Coursera, by Andrew Ng, Stanford University. The `fmincg` function inside it was written by Carl Edward Rasmussen, © 2001 and 2002.

15.4 Answers to selected exercises

Exercises

1. 2.8
2. 27125
3. 5.2694 and 10020
4. $\begin{bmatrix} -0.5 & -0.415 \\ -0.429 & -0.301 \\ 0.429 & 0.585 \\ 0.5 & 0.131 \end{bmatrix}$
5. $\begin{bmatrix} -0.930 & -0.911 \\ -0.797 & -0.661 \\ 0.797 & 1.285 \\ 0.930 & 0.287 \end{bmatrix}$

Practical

7.

		Predicted	
		No	**Yes**
Actual	**No**	3544	0
	Yes	956	0

Missclassification error: 21.2%

9. Sensitivity: 0% (This value reveals the poor performance of the classifier!)

11. (a) 17.2

(b)

		Predicted	
		No	**Yes**
Actual	**No**	3421	123
	Yes	653	303

(c) Sensitivity: 31.7%; specificity: 96.5%

13. (a) 17.2

(b)

		Predicted	
		No	**Yes**
Actual	**No**	3362	182
	Yes	590	366

(c) Sensitivity: 38.3%; specificity: 94.9%

CHAPTER 16

LEARNING CURVES

16.1 Theoretical Briefing

Learning curves. The behaviour of a predictive model as the size of the training set increases, can help us detect the presence of problems such as overfitting and underfitting. Specifically, how the training and test errors evolve will give us good clues to spot them, and therefore we draw graphs with the errors in the ordinates and the size of the training set, m, in the abscises. This plots are called **learning curves**.

Detecting overfitting. Think of it: since "overfitting" means that a model is too tight, too adjusted to the training set, the training error will be always low in models that suffer from this problem, while, at the same time, its inability to generalise will keep the test error high, no matter how many instances you add to your training set. The plot of such models will present, then, a pair of curves with a great gap between them, as shown schematically in Figure 16.1.

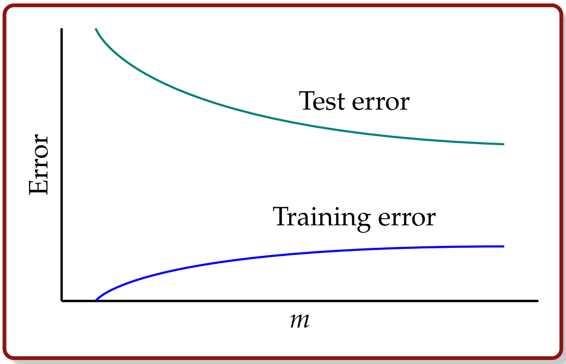


FIGURE 16.1: Learning curves for an overfitting model.

Detecting underfitting. On the other hand, models suffering from underfitting will present two curves with a small gap between them, both at a high error, as shown in Figure 16.2. The reason for this is that an underfitting model is too "simple" and biased, and fails to predict well, both in the training and the test sets.

Adding more training instances. It is a well-known fact in Machine Learning that algorithms work better with large training sets. Therefore, it seems obvious that adding more instances to the training set will improve the performance of an algorithm. However, learning curves teach us a lesson: this is not always the case. When a model suffers from overfitting or underfitting, adding more instances, no matter how many, will not help. So, before taking the effort of collecting more and more data, check your learning curves!

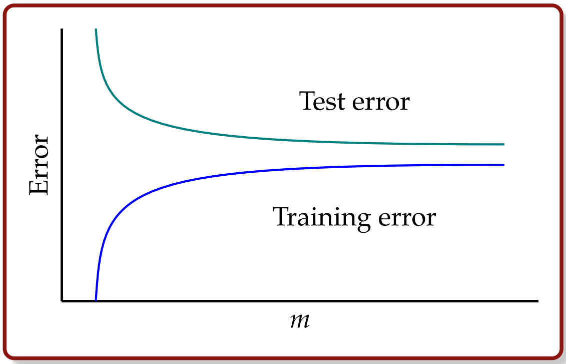


FIGURE 16.2: Learning curves for an underfitting model.

16.2 Exercises

Solve the following exercises. You can use a calculator. Do not use MATLAB here

Suppose you have a classification algorithm with certain hyperparameters. To assess the goodness of this algorithm, you use it with training sets of different sizes: first, one of only 5 instances; then, one of 100; then, one obtained by adding 200 more training instances; then, 200 more, and so on. In each case, you measure the training error and the test error, the latter for the model applied over an independent test set.

1. Assume that the values you obtain for this experiment are the ones shown in Table 16.1. Draw (by hand) the learning curve corresponding to this model. What would you conclude? Is this model underfitting or overfitting the data?

2. Assume that the values you obtain for this experiment are the ones shown in Table 16.2. Draw (by hand) the learning curve corresponding to this model. What would you conclude? Is this model underfitting or overfitting the data?

TABLE 16.1: Results for experiment (a)

m	Training error	Test error
5	0	0.98
100	0.1	0.93
300	0.15	0.89
500	0.25	0.86
700	0.26	0.85
900	0.3	0.82
1100	0.3	0.77
1300	0.35	0.79
1500	0.37	0.76

TABLE 16.2: Results for experiment (b)

m	Training error	Test error
5	0.5	0.98
100	0.65	0.9
300	0.68	0.84
500	0.7	0.83
700	0.71	0.84
900	0.74	0.84
1100	0.75	0.82
1300	0.74	0.83
1500	0.76	0.82

16.3 Practical in MATLAB

General Objective:

To make the student capable of plotting and interpreting learning curves.

Specific Objectives:

- The student should be able to plot the learning curve for any model in MATLAB, previously generating a table with the errors for different training-set sizes.

- The student should be able to interpret the results, telling from the plots whether a model is balanced or if it underfits or overfits the data.

Materials:

Computer, MATLAB.

Procedure:

Solve the following exercises in MATLAB

1. Create a script and save it following the format `Pr16nameLastname.m`. You will write your code for the next numerals in this script.

2. Load into the *Workspace* the matrices contained in the file `matricesPr16.mat`. This file can be found in the folder *Data Sets*. The matrices and label vectors you will find here were generated from the Letter Recognition Data Set of the UCI Machine Learning Repository (`http://archive.ics.uci.edu/ml/datasets/Letter+Recognition`). This data set concerns the classification of 16-D feature vectors in one of the 26 letters of the English alphabet. The features come from black-and-white images (statistical moments and edge counts measured from them). The labels found in the original data set have been transformed into numerical labels following the correspondence: "A"="1", "B"="2", "C"="3", and so on.

3. In the folder *MATLAB Functions* you will find a function named "`nnLearning2.m`". Save it to your Current Folder to be used in the next numerals. This function takes as arguments a data matrix `X`, a label vector `y`, the number of classes involved and the following parameters: hidden layer size, a regularisation coefficient `lambda` and the maximum number of iterations for the optimisation function. With this information, the function performs the backpropagation algorithm to learn the weight matrices $W^{(1)}$ and $W^{(2)}$, which are returned by the function in a cell array.[a]

4. Use the function `nnLearning2` to get the weight matrices $\mathbf{W}^{(1)}$ and $\mathbf{W}^{(2)}$ of a neural network with the following hyperparameters: hidden layer size: 2, `lambda` $=$ 50 and maximum number of iterations: 350. Use as training set only the first 5 instances of `Xtrain1` and `ytrain1` (you loaded these matrices in numeral 2). Then use your function `nnOutput` (you wrote this function in Chapter 10) with matrices $W^{(1)}$ and $W^{(2)}$ to get the predictions "`yhat`", for these 5 instances. Also, use `nnOutput` to compute the

predictions “yhat_test” for the 6000 instances in Xtest (you loaded this matrix in numeral 2). Finally, compute the misclassification error (you can use your function computeMCE for this) for both yhat and yhat_test. We call these errors the “training error” and the “test error”, respectively.

5. Repeat the procedure described in the last numeral, but this time using as training set the first 105 instances of Xtrain1 and ytrain1. Then use the first 205 instances. Then, the first 305 instances, and so on (until 3905, since Xtrain1 contains 4000 instances), in order to generate kind of a “table” in MATLAB containing the size of the training sets and the training and test errors corresponding to each of these sets. In the *Answers* section we include a part of this table for you to verify if everything is going well.

6. Use plot to draw the learning curves corresponding to the table you made in the last numeral. Place adequate labels, legends and title for the plot, using xlabel, ylabel, legend and title. Also, use xlim and ylim to limit the ranges for the m values from 0 to 4000 and for the error from 0 to 1. See the *Answers* section to get a better idea of what you are meant to do.

7. We have named the last plot as “Learning curves for an underfitting model”. From the Theoretical Briefing section you can infer that this is actually an underfitting model, because the training and test error are quite high (around 90% of the examples, misclassified!) and close to each other. Nothing strange, since we have used only 2 neurons in the hidden layer of the network! Now, what would happen if we increase the size of the training set? In this numeral, you will do so, by adding to your training set the 10000 more instances to be found in the Xadd and yadd matrices (you loaded them in numeral 2.) In this way, you will end up with an “Xtrain2” matrix of size 14000 $\times$ 16 and an “ytrain2” 14000 $\times$ 1 vector.

8. By repeating the procedure detailed in numerals 4 and 5, generate a “table” in MATLAB for this extended training set, but this time take the following values for m: 5, 505, 1005, and so on (until 13505, since Xtrain2 contains 14000 instances.) In the *Answers* section we include a part of this table for you to verify if everything is going well.

9. Plot the learning curves corresponding to the last table. Place adequate labels, legends and title for the plot, and use xlim and ylim to limit the ranges for the m values from 0 to 14000 and for the error from 0 to 1. See

the Answers section to get a better idea of what you are meant to do. Has the addition of 10000 instances helped to reduce the error?

10. Repeat the procedure detailed in numerals 4 and 5, with Xtrain1 and ytrain1, and with the following hyperparameters: hidden layer size: 1000, lambda $=0$ and maximum number of iterations: 350. (You could guess just from considering the huge amount of neurons in the hidden layer and the null regularisation, that this neural network will overfit the data!). Make your table with the following values of m: 5, 105, 205,…, 3905.

11. Plot the learning curves for this table. Please note the overfitting in the great gap between the training-error curve and the test-error curve, especially at the beginning (actually, with $m = 5$ we have zero error (perfect score) in the training set…but a 91% of examples misclassified in the test set!).

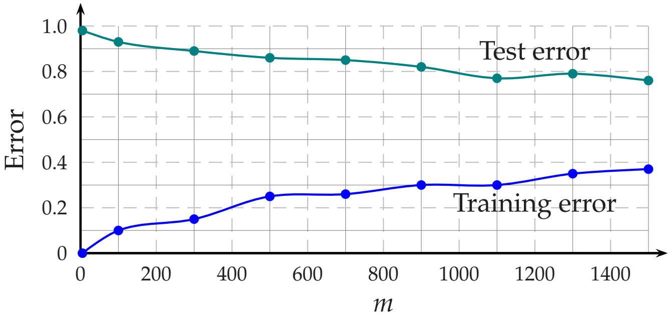
12. Generate a new table and learning curves, this time over Xtrain2 and ytrain2, with hidden layer size: 50, lambda $= 20$, maximum number of iterations: 350, and $m \in \{5, 505, 1005, \ldots, 13505\}$. This is our example of a balanced model, with relatively few hidden neurons (and therefore a reasonable running time), that nevertheless attains an acceptable performance, with an error that tends to 0.2 as the size of the training set increases.

[a] AUTHOR ATTRIBUTION: Function nnLearning2 was written by the author of this book, based on the material available online for the Programming Assignment of Week 5, *Machine Learning* course on Coursera, by Andrew Ng, Stanford University. The fmincg function inside it was written by Carl Edward Rasmussen, © 2001 and 2002.)

16.4 Answers to selected exercises

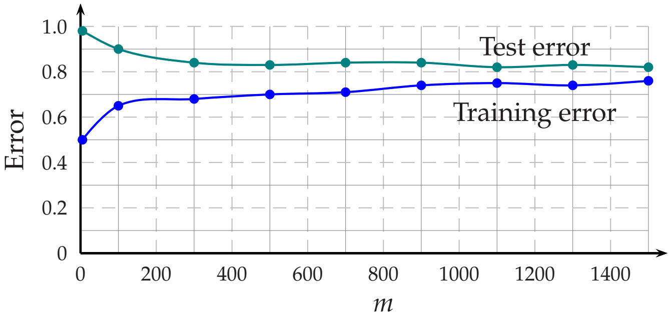
Exercises

1.



This model is overfitting the data, since there is a big gap between the training-error and the test-error curves

2.



This model is underfitting the data, since there is no big gap between the training-error and the test-error curves but they both are at high levels, the error moving around 80%

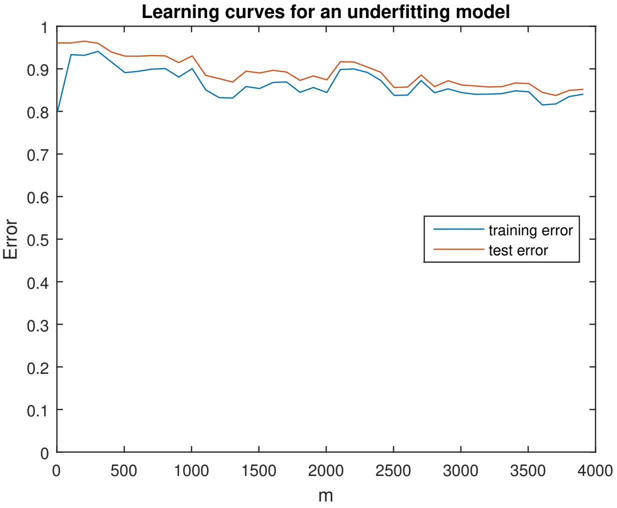
Practical

4. Training error: 0.8; test error: 0.9608

5.

m	Training error	Test error
5	0.8	0.9608
105	0.9333	0.9608
205	0.9317	0.9650
⋮	⋮	⋮
3905	0.8402	0.8518

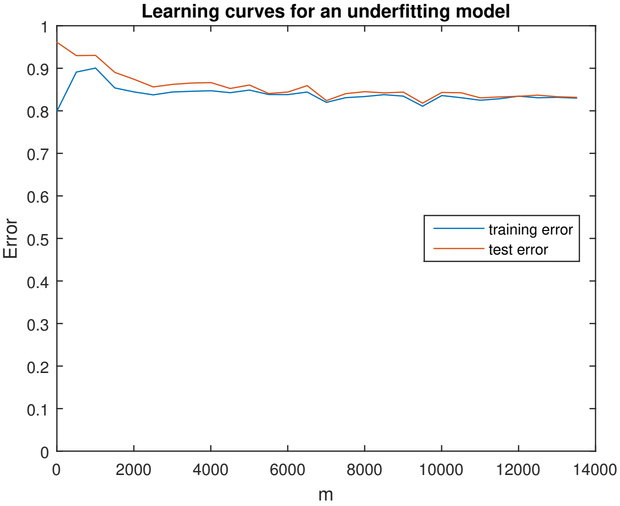
6.



8.

m	Training error	Test error
5	0.8	0.9608
505	0.891	0.9298
1005	0.9005	0.9303
⋮	⋮	⋮
13505	0.8298	0.8317

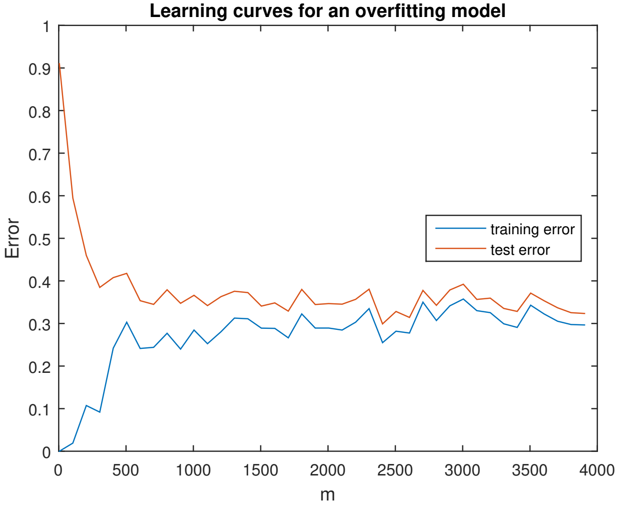
9.



10.

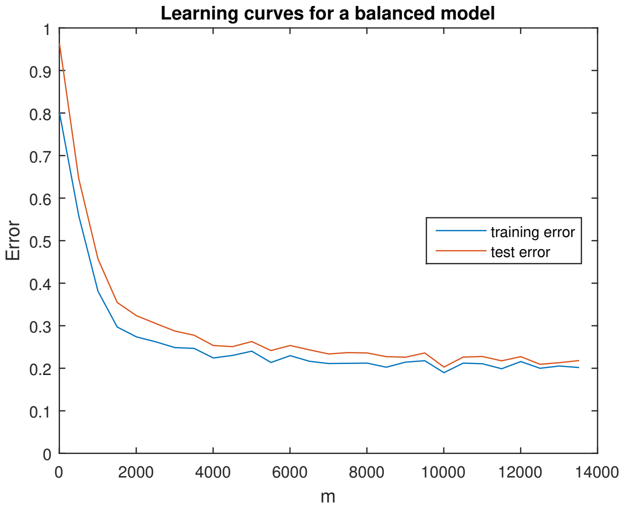
m	Training error	Test error
5	0	0.91
105	0.019	0.5947
205	0.1073	0.4598
⋮	⋮	⋮
3905	0.2965	0.3235

11.



12.

m	Training error	Test error
5	0.8	0.9623
505	0.5584	0.6457
1005	0.3811	0.457
⋮	⋮	⋮
13505	0.2018	0.218


Learning curves for a balanced model
Error
m
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
2000
4000
6000
8000
10000
12000
14000
training error
test error

Chapter 17

Principal Component Analysis

17.1 Theoretical Briefing

Reduction of dimensionality. Without going into detail, it seems clear that when dealing with the dimensionality of feature vectors, the premise “the fewer the better” is applicable. Having too many dimensions will require more space for storage and more time for computations, for instance. In this chapter, then, we look for a method to reduce the dimensionality of the feature vectors in our datasets, retaining the valuable information and getting rid of the superfluous.

Principal components. Given a certain dataset, it often happens that the feature vectors are spread in a certain direction more than in others. We could call this direction the “first principal component”. This direction accounts for the greatest variance of the data and therefore is the most important in terms of the discriminative information needed for, say, a classification task. Sorting from greatest to lowest variance, then, we can sort the components from the “most important” to the “less important” one. Let d be the dimension of the feature space. Then, there will be d such components.

Principal component analysis (PCA). After centring the data, so that the mean of the vectors is zero, this algorithm looks for the d components, sorted from the most important (the first principal component) to the less important. For doing this, the MATLAB implementation of PCA uses the singular value decomposition algorithm, which lies beyond the scope of this book.

Centring the Data. In order to make sense of the information produced by PCA, it will be necessary to perform its first step ourselves. This is, centring the data. To centre your feature vectors, you just need to subtract off the mean of them.

The MATLAB function for PCA. To perform PCA on our data, we will use the MATLAB function pca. As you can see in the documentation, one possible syntax is:

```
[coeff,score] = pca(X)
```

where X is the $m \times d$ data matrix that we are well accustomed to, and the matrices returned are "coeff", a $d \times d$ matrix containing the d eigenvectors (or unit vectors in the principal directions); and "score", a $m \times d$ matrix containing the coordinates of the m vectors in the space of principal components. The coeff matrix is $d \times d$ because each eigenvector has d dimensions. One way to represent the information contained in coeff is shown in Figure 17.1. One way to visualise the information contained in the score matrix is shown in Figure 17.2, for the two-dimensional case. This figure shows that the elements of the i-th row of score are the coordinates of the i-th feature vector, but in the principal-component space. Since this new space has also d dimensions and there are m feature vectors, the size of score must be $m \times d$.

$$\mathbf{coeff} = \begin{bmatrix} \uparrow & \uparrow & & \uparrow \\ \mathbf{e}_1 & \mathbf{e}_2 & \dots & \mathbf{e}_d \\ \downarrow & \downarrow & & \downarrow \end{bmatrix}_{d \times d}$$

FIGURE 17.1: A visual representation of the coeff matrix.

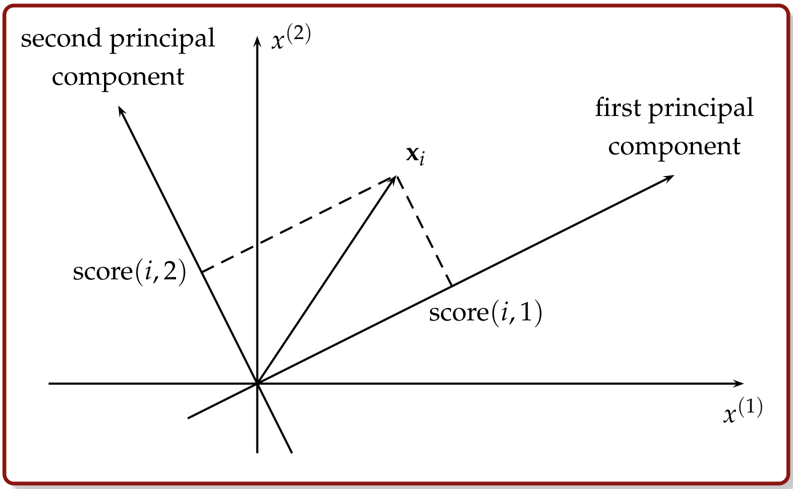


FIGURE 17.2: Geometrical meaning of the elements of the i-th row of score.

Projections. To reduce the dimensionality of the data, take not all of the principal components, but only a subset of them. In practice, this means taking only a subset of the columns of the `score` matrix. (Of course, you will prefer to take the columns at the left of this matrix rather than those at the right.) Geometrically, this is equivalent to **projecting** the data onto a lower-dimensional space. For instance, you can project three-dimensional vectors onto a plane, or two-dimensional vectors onto a line (see Figure 17.3.)

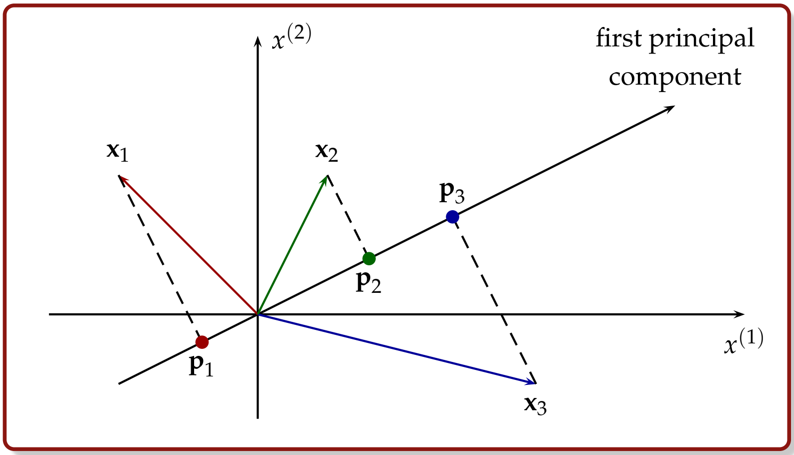


FIGURE 17.3: Projections of two-dimensional vectors onto a one-dimensional space.

In Figure 17.3, we have represented three vectors in a two-dimensional space and their projections onto the first principal component (allegedly.) From your old Physics classes, you can recall that, for any vector $\mathbf{v}$, Equation 17.1 holds.

$$\mathbf{v} = |\mathbf{v}| \cdot \mathbf{u}_v \tag{17.1}$$

where $|\mathbf{v}|$ is the magnitude of $\mathbf{v}$ and $\mathbf{u}_v$ is its unit vector. You can use this equation to compute the coordinates of vectors $\mathbf{p}_1$, $\mathbf{p}_2$ and $\mathbf{p}_3$, realising that their magnitudes are in the first column of the matrix `score` and that the unit vector for all of them is the eigenvector $\mathbf{e}_1$.

17.2 Exercises

Solve the following exercises. You can use a calculator. Do not use MATLAB here.

Let

$$\mathbf{X} = \begin{bmatrix} -1 & 1 \\ 2 & 3 \\ 3 & 3 \\ 5 & 4 \\ 7 & 4 \end{bmatrix}$$

be a data matrix. By using the MATLAB function pca on this data, we get the following matrix of coefficients:

$$\textbf{coeff} = \begin{bmatrix} 0.93 & -0.36 \\ 0.36 & 0.93 \end{bmatrix}$$

Also, the following matrix of scores is returned by this function:

$$\textbf{score} = \begin{bmatrix} -4.64 & -0.35 \\ -1.12 & 0.43 \\ -0.19 & 0.07 \\ 2.04 & 0.28 \\ 3.90 & -0.44 \end{bmatrix}$$

1. Centre the data and write down a matrix $\mathbf{X}_c$ containing these centred vectors.
2. Plot the data in $\mathbf{X}_c$ in a Cartesian plane.
3. On the same graph, plot the two vectors contained in the "**coeff**" matrix.
4. Using the information given in the "**score**" matrix, compute the positions of the projections of the five points of $\mathbf{X}_c$ onto the principal component direction. (*Hint*: take only the first column of "score" for your calculations, since this column contains the coordinates of the points in the first-component axis. Then, use Equation 17.1). In the *Answers* section, we show these positions arranged in a matrix **P**.
5. In the same graph drawn in numerals 2 and 3, locate the points to which the coordinates in matrix **P** correspond. Note that all of these five points belong to a straight line (drawn as a blue dashed line in the *Answers* section), and that this line corresponds to the principal component direction.

17.3 Practical in MATLAB

17.4 PRACTICAL IN MATLAB

General Objective:

To make the student capable of using the MATLAB function to perform the Principal Component Analysis on a dataset and of understanding its results.

Specific Objectives:

- The student should be able to use the MATLAB function pca and the variables returned by it to graphically visualise the principal components of the data and their projections.
- The student should be able to use the MATLAB function pca to reduce

the dimensionality of the iris dataset and to perform a classification task on the reduced set.

Materials:

Computer, MATLAB.

Procedure:

Solve the following exercises in MATLAB

1. Create a script and save it following the format `Pr17_1nameLastname.m` You will write your code for the next numerals in this script.

2. Load into the *Workspace* the matrix and the neural-network object contained in the file `matricesPr17.mat` This file can be found in the folder *Data Sets*.

3. In numeral 2, you loaded a data matrix `X`. Create now a matrix "`Xcentred`", with the feature vectors centred (read "Centring the Data" in the *Theoretical Briefing*.) Use `scatter` to plot the data contained in `Xcentred`. (Note how the "centre" of the data is now at (0,0).)

4. Make MATLAB perform the Principal Component Analysis on this data, by typing `[coeff,score,latent] = pca(X);`

5. As explained in the *Theoretical Briefing*, the matrix `coeff` contains the p eigenvectors obtained in the PCA, in descending order of variance (this is, in descending order of importance). Use the MATLAB function `line` to plot the two eigenvectors returned by **pca** in numeral 4, making the first eigenvector appear thicker than the second (look at the *Answers* section to see what you are meant to attain.) Also, use `axis equal` to make both axis have the same scale, in order to see the orthogonality of the two eigenvectors.

6. Use `scatter` and `line` to plot both the data in `Xcentred` and the direction of the principal component. To do this, you can just prolong the principal eigenvector in both directions. (For the plot shown in the `Answers` section, we have multiplied the eigenvector by 11 in each direction.)

7. In the same plot drawn in the previous numeral, visualise the orthogonal projections of the vectors onto the principal component direction. To do this, you will need to use the information contained in the matrix `score`

obtained in numeral 4. As explained in the Theoretical Briefing, this matrix contains the coordinates of the feature vectors in the space defined by the eigenvectors. Since you have to plot the projections onto the first principal component only, you should only use the first column of the `score` matrix. In the plot shown in the *Answers* section, we have represented these projections as red dots; and then linked the projections with the original vectors through blue lines.

8. Now, you will work with real data: the iris dataset again (already used in Chapter 12.) Go to `https://archive.ics.uci.edu/ml/datasets/iris` to download the data or use the Excel file generated in Chapter 12. Then form a matrix "`Xiris`" and a vector "`yiris`" with the feature vectors and the labels, respectively. If everything has been done right, your matrix `Xiris` should be 150 × 4 and `yiris`, 150 × 1.

9. In numeral 2, you loaded an object called "`net_iris`", which basically contains a neural network trained with this iris dataset. Use `net_iris` to get the predictions of this neural network for the data in `Xiris` and compute:

   a The misclassification error.

   b The confusion matrix.

10. Use `pca` to perform Principal Component Analysis on the data in `Xiris`. Remember that the feature vectors in this dataset are 4-D. Our objective here is to reduce the dimensionality of the data to two, this is, our new feature vectors will be 2-D. To attain this, take the coordinates corresponding to the first two principal components and form a matrix "`Xiris2`". Naturally, `Xiris2` will be a 150 × 2 matrix.

11. Use `scatter` to plot the 2-D vectors in `Xiris2`, using different markers and colours to differentiate vectors of different classes (as you did with your function `plotClasses` in Chapter 3). Also, use the MATLAB functions `xlabel`, `ylabel` and `legend` to add information to your plot (See the *Answers* section to know better what are you expected to attain.)

12. Use the MATLAB app "Neural Net Pattern Recognition" to train a neural network with the data in `Xtrain2`. Please note that, even after having reduced the dimensionality of the data to half of its size, you still can get excellent classifications of the data. We cannot include an answer in this numeral (due to the randomness in the initialisation of the neural

networks), but you can see by yourself that this is true by retraining the net several times. Eventually, you will even run into a zero error in the test set (i.e., a perfect classification) with this reduced set!

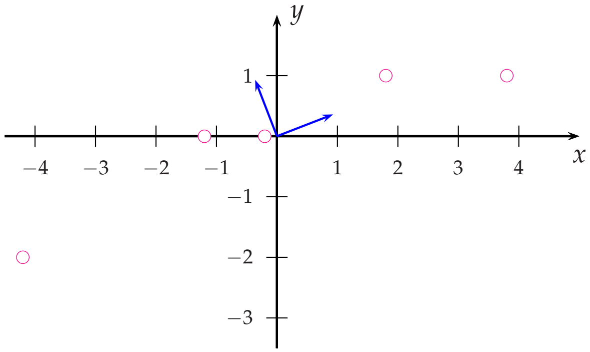
17.5 Answers to selected exercises

Exercises

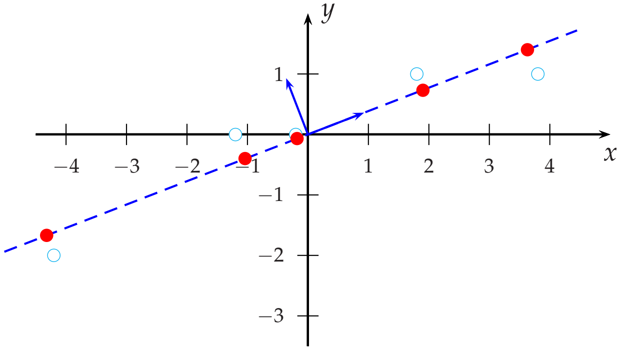
1. $\mathbf{X}_c = \begin{bmatrix} -4.2 & -2 \\ -1.2 & 0 \\ -0.2 & 0 \\ 1.8 & 1 \\ 3.8 & 1 \end{bmatrix}$

4. $\mathbf{P} = \begin{bmatrix} -4.32 & -1.67 \\ -1.04 & -0.40 \\ -0.18 & -0.07 \\ 1.90 & 0.73 \\ 3.63 & 1.40 \end{bmatrix}$

3.

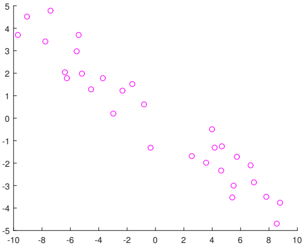


5.

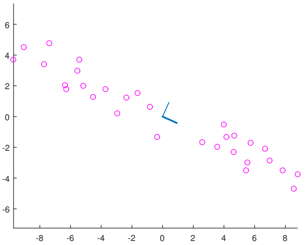


Practical

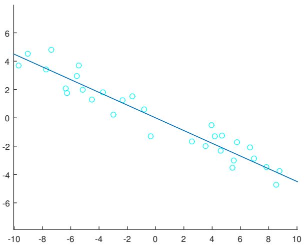
3.



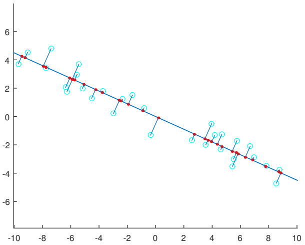
5.



6.



7.

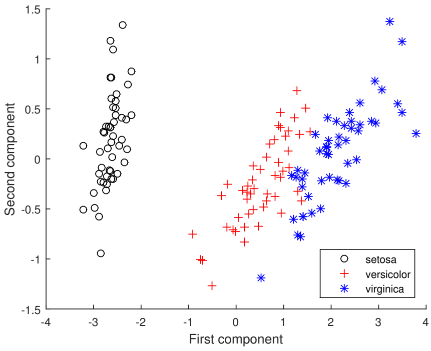


9. (a) 0.2

(b)

		PREDICTED		
		setosa	versicolor	virginica
ACTUAL	setosa	50	0	0
	versicolor	0	47	3
	virginica	0	0	50

11.



PART VI

UNSUPERVISED LEARNING

As you might have realised, all of the previous algorithms studied in this book work with input data consisting of two components: a matrix **X** (made of feature vectors) and a vector **y** (made of the corresponding labels or the values of the dependent variable.) This kind of algorithms is said to perform "supervised learning". In contrast, there are algorithms that work only with a matrix **X** (without the **y** vector). These algorithms are said to perform "unsupervised learning", and their task is usually to make a **clustering** of the data.

In this part, we study a simple but extremely popular algorithm to attain unsupervised learning: *k-means*.

Chapter 18

k-Means Algorithm

18.1 Theoretical Briefing

Clustering. Given a set of feature vectors (especially if they are taken from real situations) it is very likely that they form **clusters**. Clusters are defined by distances: vectors of a given cluster are closer to each other than they are to vectors in other clusters. **Clustering** is the action of assigning a label (the label of any of the clusters) to each of the vectors in a dataset.

The algorithm. The most famous algorithm to perform clustering is called *k-means*. This algorithm takes a dataset $\mathbf{X}$ and returns the centroids of k clusters for this data. Let $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m$ be the feature vectors in $\mathbf{X}$. Then, the k-means algorithm is the one shown in Algorithm 4

Algorithm 4 The *k-means* algorithm.

```
Initialise the centroids $\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_k$
for $iteration = 1 : n$ (or until fulfilling a stopping criterion) do
    for $l = 1 : m$ do
        compute $d_1, d_2, \ldots, d_k$ (the distances between $\mathbf{x}_l$ and $\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_k$)
        assign $\mathbf{x}_l$ to the cluster corresponding to the lowest distance
    end for
    for $j = 1 : k$ do
        take all the vectors in the j-th cluster. Call this set $S_j$ and the number of elements in it, $n_j$
        update the j-th centroid using $\mathbf{c}_j^{(new)} = \frac{1}{n_j} \sum_{i \in S_j} \mathbf{x}_i$
    end for
end for
return $\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_k$
```

Distance measure. Like in the k-nearest neighbours algorithm, you can use here the Euclidean distance, Equation 14.1. In the *Exercises* section of this chapter, we will use that definition of distance.

Initialisations. The first line in Algorithm 4 tells about initialising the centroids. Although in the *Exercises* and *Practical* of this chapter we give the initial centroids as data, in real life the implementation of this algorithm should include random initialisations. One good possibility is to take k vectors, randomly chosen from **X**, as the initial centroids. Another possibility is just to take k random vectors anywhere in the feature space.

Stopping criteria. The simplest stopping strategy you could adopt with *k-means* is the one used in the *Practical* section of this book: fix a number of iterations to be performed. However, in a real application you would not be able to properly guess how many iterations are necessary to get a good clustering. Therefore, another option must be used. A good possibility is to measure how much the clusters move while the iterations advance. If they are no longer moving, or if they are scarcely moving, stop the algorithm.

18.2 Exercises

Solve the following exercises. You can use a calculator. Do not use MATLAB here

Let

$$\mathbf{X} = \begin{bmatrix} 3 & 1 \\ -1 & 0 \\ -2 & 0 \\ 3 & 0 \\ 0 & -1 \\ 3 & -1 \end{bmatrix}$$

be the data matrix for a clustering problem.

1. Plot the data in a Cartesian plane, identify two clusters and try to guess where the centroids should be. What are the coordinates of these centroids, approximately?
2. Execute, by hand, the k-means algorithm for the data in matrix X and complete the Table 18.1. (Use, as initial centroids, the vectors $[-2\ \ 3]^T$ and $[7\ \ -1]^T$.)

   In the *Update* steps, use Equations 18.1 and 18.2.

$$\mathbf{c}_1^{(new)} = \frac{1}{n_1} \sum_{i \in S_1} \mathbf{x}_i \qquad (18.1)$$

$$\mathbf{c}_2^{(new)} = \frac{1}{n_2} \sum_{i \in S_2} \mathbf{x}_i \qquad (18.2)$$

TABLE 18.1

iteration	*l*	Distance to centroid # 1	Distance to centroid # 2	Assigned to cluster
1	1	5.39	4.47	2
	2	3.16		1
	3	5		
	4			
	5			
	6			
	Update centroid # 1. This is, compute $\mathbf{c}_1^{(new)}$			
	Update centroid # 2. This is, compute $\mathbf{c}_2^{(new)}$			
2	1	5.39	4.47	2
	2			
	3			
	4			
	5			
	6			
	Update centroid # 1. This is, compute $\mathbf{c}_1^{(new)}$			
	Update centroid # 2. This is, compute $\mathbf{c}_2^{(new)}$			

18.3 Practical in MATLAB

General Objective:

To make the student capable of implementing the k-means algorithm and to graphically represent its results in the two-dimensional case.

Specific Objectives:

- The student should be able to use the k-means algorithm to compute the centroids of the clusters for a data matrix X.
- The student should be able to graphically visualise the evolution of the centroids –how they "move" through the feature space when the algorithm develops.
- The student should be able to cluster the data in a data matrix X, graphically identifying the clusters by means of different colours.

Materials:

Computer, MATLAB.

Procedure:

Solve the following exercises in MATLAB

1. Create a script and save it following the format `Pr18_nameLastname.m` You will write your code for the next numerals in this script.

2. Load into the workspace the matrix contained in the file `matricesPr18.mat`. This file can be found in the folder *Data Sets*.

3. Write a function "`k_meansLearning(X,k,n_iter,centroids_ini)`",which takes as arguments a data matrix `X`, the number `k` of desired clusters, the number of iterations `n_iter` and a matrix `centroids_ini`, containing the k initial centroids. To follow the style we are used to, `centroids_ini` will contain the vectors transposed and stacked one above the other. That is, `centroids_ini` will be a matrix of dimension $k \times d$, where d is the dimension of the vectors. Your function `k_meansLearning` should return a $k \times d$ matrix "`centroids`" containing the centroids which result after applying the k-means algorithm to this data. For didactical purposes, you will not take the convergence of the algorithm as its stopping criteria (as it should be, following Algorithm 4). Instead, you will have your algorithm halting simply when it has fulfilled the number of iterations given by `n_iter`.

4. In numeral 2, you loaded a data matrix `X`. Use the MATLAB function **scatter** to visualise the data contained in this matrix.

5. Use your function `k_meansLearning` on the matrix `X` mentioned in the previous numeral, to compute the centroids given by the k-means algorithm for this data. Use as parameters: `k`$=$ 34 and `n_iter`$=$ 3. Set `centroids_ini` as the matrix:

$$\begin{bmatrix} -1 & 4 \\ 2 & 5 \\ 0 & 0 \end{bmatrix}$$

6. Write a function "`k_meansHistory(X,k,n_iter,centroids_ini)`", which does exactly what the `k_meansLearning` function does, with the difference that it returns not only the last group of centroids, but all the groups of centroids obtained by the algorithm (one group in each iteration). Since each group of centroids is stored in a matrix, this function cannot return a simple single matrix. Instead, it should return a cell array "`centroidHistory`" containing the matrices. Information about cell arrays is available in the MATLAB documentation.

7. Try your `k_meansHistory` function with the same data and the same parameters as those given in numeral 5. The groups of centroids you should

end up with are shown in the *Answers* section.

8. Write a function "`plotkMeans(X,centroidHistory)`", which takes as arguments a data matrix `X` and a cell array `centroidHistory` containing the groups of centroids the k-means algorithm has gone through. This function should plot the data contained in X and the evolution of the centroids. You can have a better idea of what you are expected to attain, by looking at the answer of the next numeral in the `Answers` section.

9. Use your function `plotkMeans` on the `centroidHistory` cell array obtained in numeral 7.

10. Use your function `k_meansHistory` on the matrix `X` loaded in numeral 2, using as parameters: `k`$=3$ and `n_iter`$=20$. Again, set `centroids_ini` as the matrix.

$$\begin{bmatrix} -1 & 4 \\ 2 & 5 \\ 0 & 0 \end{bmatrix}$$

As should be obvious to you, this time your function will return 21 matrices in the `centroidHistory` cell array. Use `plotkMeans` on this cell array to visualise the evolution of the centroids in this case. What do you see? What was the effect of performing 20 iterations of the k-means algorithm?

11. Now, let us change the initialisation. Use `k_meansHistory` on the same matrix X and using as parameters: `k`$=3$ and `n_iter`$=4$. This time, however, set `centroids_ini` as the matrix.

$$\begin{bmatrix} -10 & -4 \\ 5 & -10 \\ 15 & 10 \end{bmatrix}$$

Visualise the results using `plotkMeans`.

12. Write a function "`k_meansOutput(Xnew,centroids)`", which takes as arguments a matrix `centroids` and a matrix `Xnew` of feature vectors; and returns the cluster-assignation for all of the vectors in `Xnew`. These assignations consist of the indexes of the clusters, which are in turn given by the order in which they are stored in the matrix `centroids`. For example,

let `centroids` be the matrix:

$$\begin{bmatrix} -8.2 & 2.3 \\ 6.8 & 4.8 \\ 0.8 & -4.1 \end{bmatrix}$$

Then, cluster #1 would be the cluster corresponding to the centroid $[-8.2, 2.3]^T$, and so on. Your function `k_meansOutput(Xnew,centroids)` should return the assignations in a vector of indexes we will call "`cluster_assignations`".

13. Try your function `k_meansOutput` upon the matrices:

$$\mathbf{X}_{new} = \begin{bmatrix} -10 & -5 \\ 0 & -15 \\ -5 & 10 \\ 10 & -1 \end{bmatrix}$$

and

$$\textbf{centroids} = \begin{bmatrix} -8.2 & 2.3 \\ 6.8 & 4.8 \\ 0.8 & -4.1 \end{bmatrix}$$

to get the indexes of the clusters assigned to each of the four vectors in $\mathbf{X}_{new}$.

14. Create a plot showing the assignations given by `k_meansOutput` for the 450 vectors in the matrix `X` loaded in numeral 2, with respect to the same matrix **centroids** given in the previous numeral. Use different colours for the clusters. Specifically, use green for cluster #1, blue for cluster #2 and magenta for cluster #3. Also, show the centroids as black stars.

18.4 Answers to selected exercises

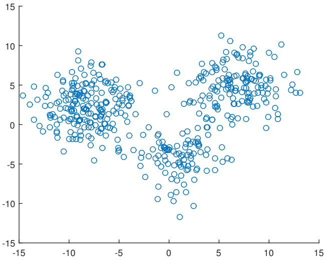
Exercises

2.

iteration	*l*	Distance to centroid # 1	Distance to centroid # 2	Assigned to cluster
1	1	5.39	4.47	2
	2	3.16	8.06	1
	3	5.00	5.10	1
	4	5.83	4.12	2
	5	4.47	7.00	1
	6	6.40	4.00	2
	Update centroid # 1: $\mathbf{c}_1^{(new)} = \begin{bmatrix}0.333\\-0.333\end{bmatrix}$			
	Update centroid # 2: $\mathbf{c}_2^{(new)} = \begin{bmatrix}3\\0\end{bmatrix}$			
2	1	2.98	1.00	2
	2	1.37	4.00	1
	3	1.70	1.00	2
	4	2.69	0	2
	5	0.75	3.16	1
	6	2.75	1.00	2
	Update centroid # 1: $\mathbf{c}_1^{(new)} = \begin{bmatrix}-0.5\\-0.5\end{bmatrix}$			
	Update centroid # 2: $\mathbf{c}_2^{(new)} = \begin{bmatrix}2.75\\0\end{bmatrix}$			

Practical

4.



5. `centroids`$=\begin{bmatrix}-8.2348 & 2.3388\\ 6.7733 & 4.8441\\ 0.7872 & -4.1161\end{bmatrix}$

7. Group 1:

$$\begin{bmatrix}-1 & 4\\ 2 & 5\\ 0 & 0\end{bmatrix}$$

Group 2:

$$\begin{bmatrix}-8.2883 & 3.1015\\ 6.9356 & 5.0550\\ -1.1125 & -3.1631\end{bmatrix}$$

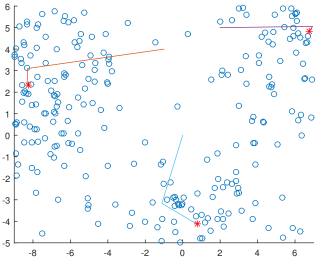
Group 3:

$$\begin{bmatrix} -8.3094 & 2.4412 \\ 6.7733 & 4.8441 \\ 0.4988 & -3.9973 \end{bmatrix}$$

Group 4:

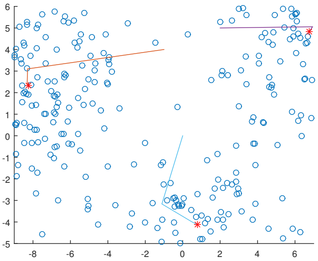
$$\begin{bmatrix} -8.2348 & 2.3388 \\ 6.7733 & 4.8441 \\ 0.7872 & -4.1161 \end{bmatrix}$$

9.



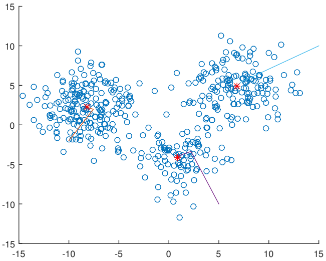
(NOTE: The plot obtained with the `plotkMeans` function has been cropped using `xlim` and `ylim` to show a "zoom" of the area of interest.)

10.

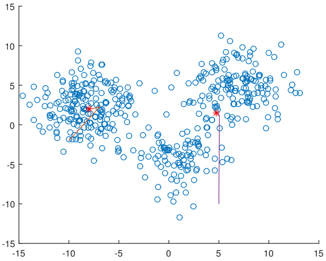


Actually, there is no apparent effect at all! This becomes clear when you look at the `centroidHistory` cell array, to find out that the algorithm reaches its final solution at the third iteration; and, after that, the centroids remain unchanged. Indeed, k-means is well known for its speed, given a good initialisation.

11.

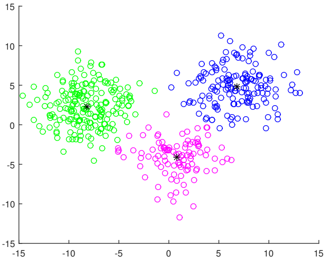


12.



14. `cluster_assignations` $= \begin{bmatrix} 1 \\ 3 \\ 1 \\ 2 \end{bmatrix}$

15.



Bibliography

[1] Classify patterns with a shallow neural network. https://uk.mathworks.com/help/deeplearning/gs/classify-patterns-with-a-neural-network.html;jsessionid=2c390a63bc8487fb3cd1df651d31.

[2] Machine learning. https://www.coursera.org/learn/machine-learning.

[3] Tara H. Abraham. *Rebel genius: Warren S. McCulloch's transdisciplinary life in science*. The MIT Press, 2016.

[4] David Barber. *Bayesian Reasoning and Machine Learning*. Bayesian Reasoning and Machine Learning. Cambridge University Press, 2012.

[5] Kory Becker. Gender recognition by voice. https://www.kaggle.com/primaryobjects/voicegender, 2017.

[6] Christopher M. Bishop. *Pattern recognition and machine learning*. Springer, 2013.

[7] Bonifacio Martín del Brío and Alfredo Sanz Molina. *Redes neuronales y sistemas difusos*. Alfaomega Ra-Ma, 2001.

[8] R.A. Fisher. Iris data set. https://archive.ics.uci.edu/ml/datasets/iris.

[9] Stephen Marsland. *Machine Learning: an Algorithmic Perspective*. CRC Press, 2011.

[10] Gualtiero Piccinini. The first computational theory of mind and brain: A close look at mcculloch and pitts's "logical calculus of ideas immanent in nervous activity". *Synthese*, 141(2):175–215, 2004.

[11] Frank Rosenblatt. *The Perceptron, a Perceiving and Recognizing Automaton*. Report: Cornell Aeronautical Laboratory. Cornell Aeronautical Laboratory, 1957.

[12] J. Schmidhuber. Deep Learning. *Scholarpedia*, 10(11):32832, 2015. revision #184887.

[13] David J. Slate. Letter recognition data set. `http://archive.ics.uci.edu/ml/datasets/Letter+Recognition`.

[14] Pedro Isasi Viñuela and Inés Galván León. *Redes de neuronas artificiales un enfoque práctico*. Pearson, 2004.

[15] Bernard Widrow. An adaptive "adaline" neuron using chemical "memistors", 1553-1552, 1960.

[16] William H. Wolberg. Breast cancer wisconsin (original) data set. `https://archive.ics.uci.edu/ml/datasets/breast+cancer+wisconsin+(original)`.

[17] I-Cheng Yeh. Concrete compressive strength data set. `https://archive.ics.uci.edu/ml/datasets/Concrete+Compressive+Strength`.

[18] I-Cheng Yeh. Default of credit card clients data set. `https://archive.ics.uci.edu/ml/datasets/default+of+credit+card+clients`.