https://doi.org/10.17163/abyaups.146.1

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 $\mathbf{A}$ and $\mathbf{B}$ 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 $\mathbf{C}$ be the result of the matrix product $\mathbf{A} \times \mathbf{B}$.

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

   (b) Compute matrix $\mathbf{C}$.

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

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

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

3. Let $\mathbf{A}$ and $\mathbf{B}$ 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 $\mathbf{C}$ be the result of the matrix product $\mathbf{A} \times \mathbf{B}$.

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

   (b) Compute matrix $\mathbf{C}$.

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

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

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

5. Let $\mathbf{A}$ and $\mathbf{B}$ 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 $\mathbf{C}$ be the result of the matrix product $\mathbf{A} \times \mathbf{B}$.

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

   (b) Compute matrix $\mathbf{C}$.

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**.

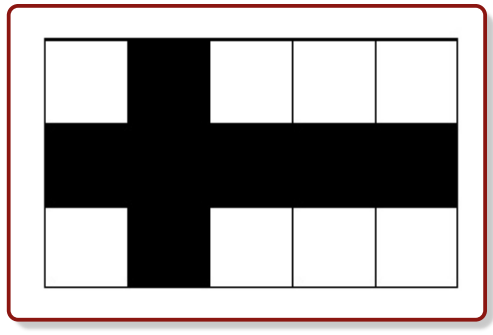


FIGURE 2.1: A simple pixel-made image

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 **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 $\mathbf{y}$ 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 $\mathbf{B}$ which would result from running the following code in MATLAB?

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

2. What is matrix $\mathbf{C}$ which would result from running the following code in MATLAB?

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

3. What is matrix $\mathbf{D}$ 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 $\mathbf{y}$ 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`".

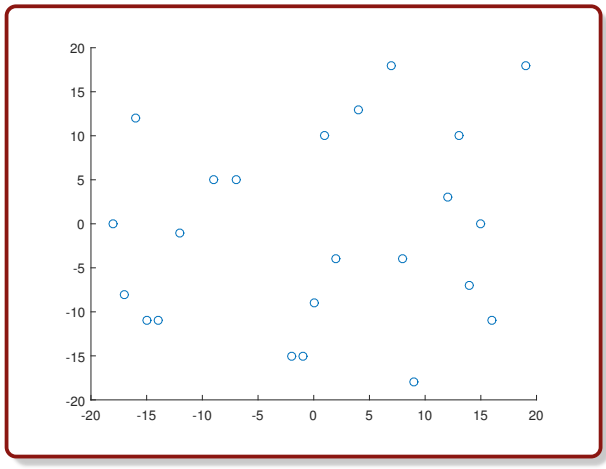


FIGURE 2.2: The data in `X1`

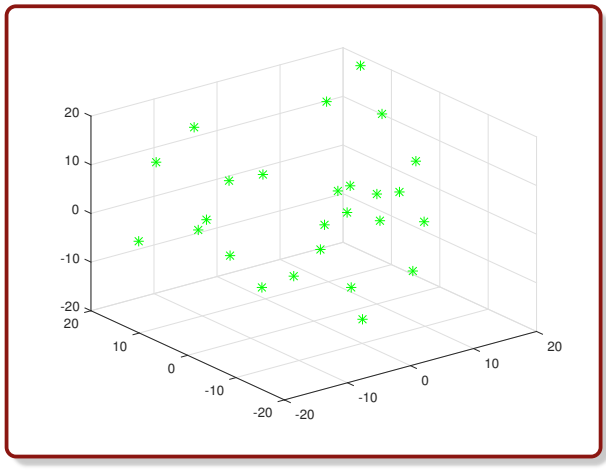


FIGURE 2.3: The data in `X2`

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.

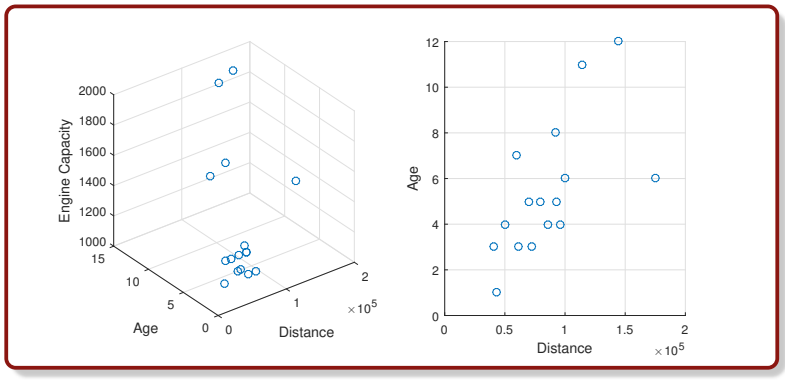


FIGURE 2.4: The data in `Xcars`

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?

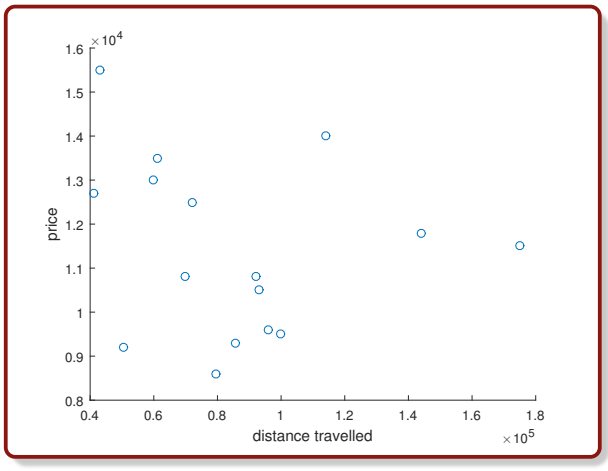


FIGURE 2.5: The independent variable versus a feature of the cars

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

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					
			c_1	c_2	c_3	$\cdots$	c_k
Actual	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.

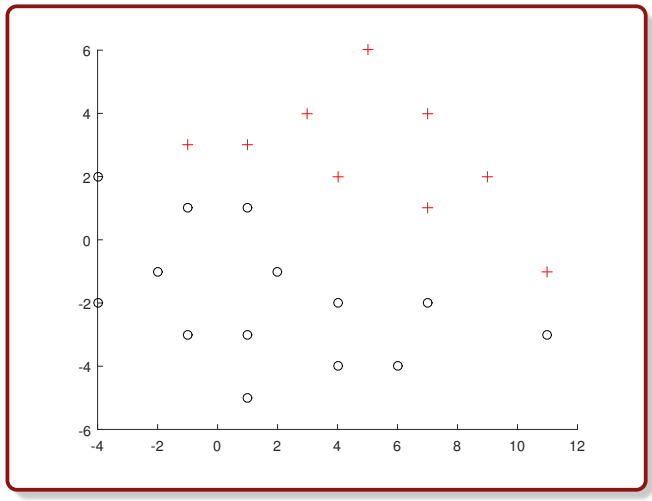


FIGURE 3.2: The data in `X` and `y`

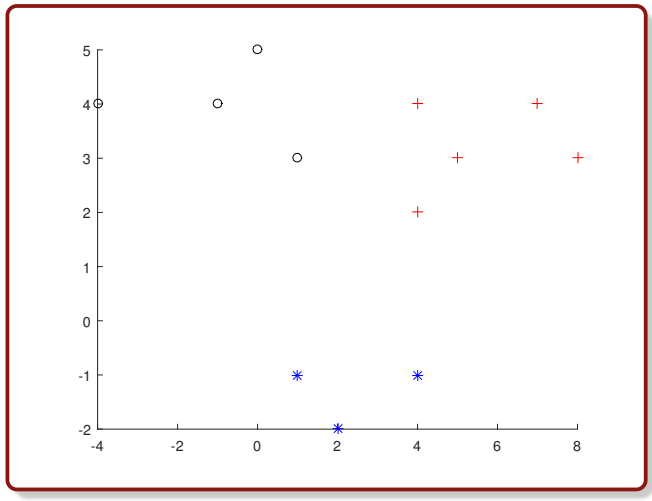


FIGURE 3.3: The data in `X2` and `y2`

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":

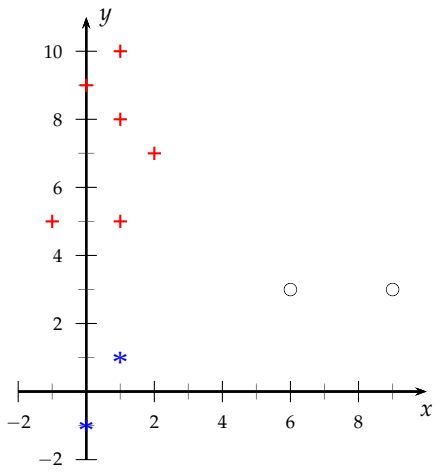
    (a) Class "-1" (you should pass index 1 for this);

    (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:

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

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 $\widehat{y}$ 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

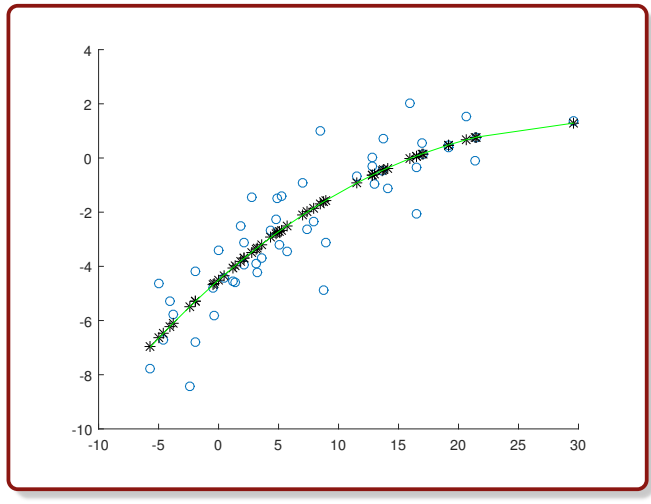


FIGURE 4.1: The regression model in `yhat1`

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

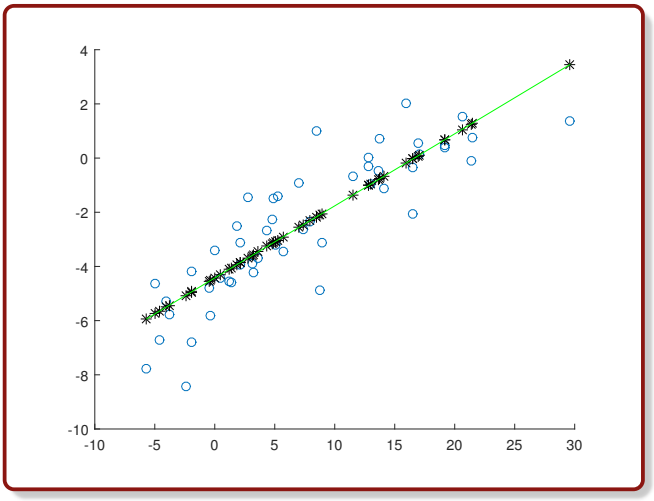


FIGURE 4.2: The regression model in `yhat2`

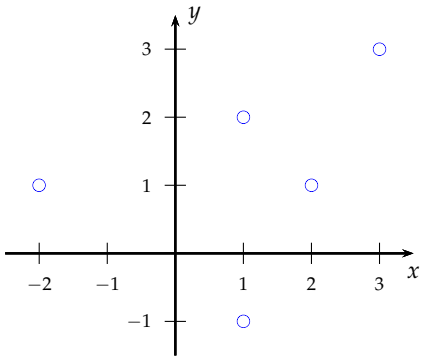
6. Write a function "`computeMSE(y,yhat)`" which takes vectors `y` and `yhat` of true and predicted labels (values of the dependent variable) respectively, and returns the mean squared error of the prediction. (*Challenge*: Do NOT use `for` loops when writing this function.)

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

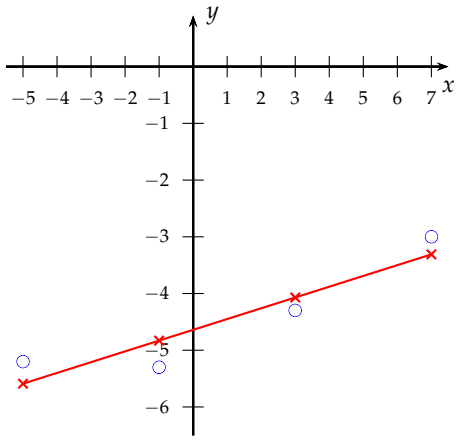
1.



2. 0.012

3. 0.1305

7.



Practical

6. 13450

7. 15976

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