https://doi.org/10.17163/abyaups.146.4

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.

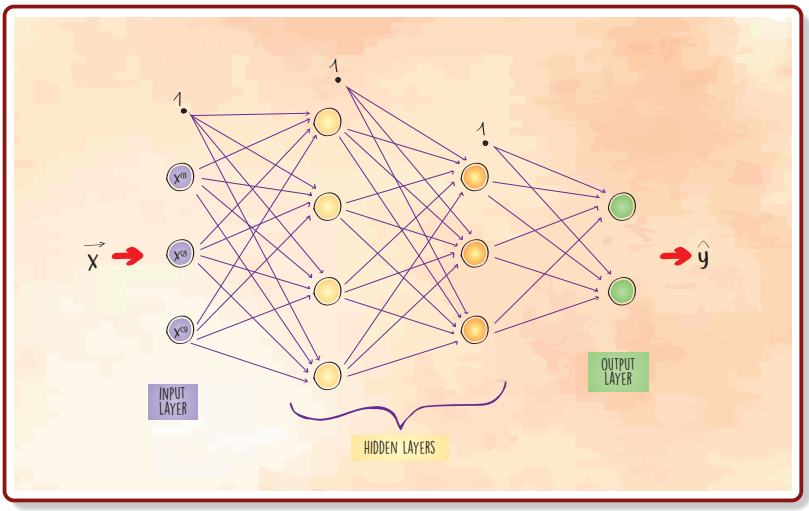


FIGURE 10.1: Architecture of a neural network

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.

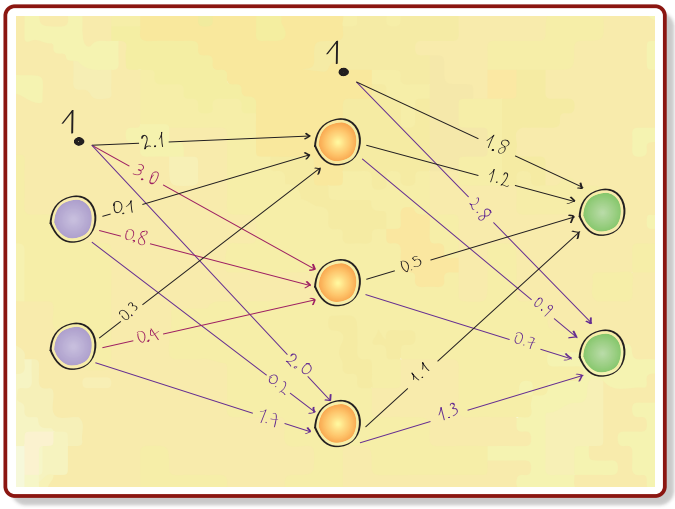


FIGURE 10.2: Example of a neural network with one hidden layer

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 $(\hat{\mathbf{y}})$ 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 ($\widehat{y}$) 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:

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

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. `W1_opt`=
$$\begin{bmatrix} 1.9858 & -0.1206 & -0.1630 & -0.1237 & -0.0325 & -0.0991 & -0.2809 & 0.0092 & -0.0698 \\ 1.9858 & -0.1206 & -0.1630 & -0.1237 & -0.0325 & -0.0991 & -0.2809 & 0.0092 & -0.0698 \\ 1.9858 & -0.1206 & -0.1630 & -0.1237 & -0.0325 & -0.0991 & -0.2809 & 0.0092 & -0.0698 \end{bmatrix}$$
    `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.

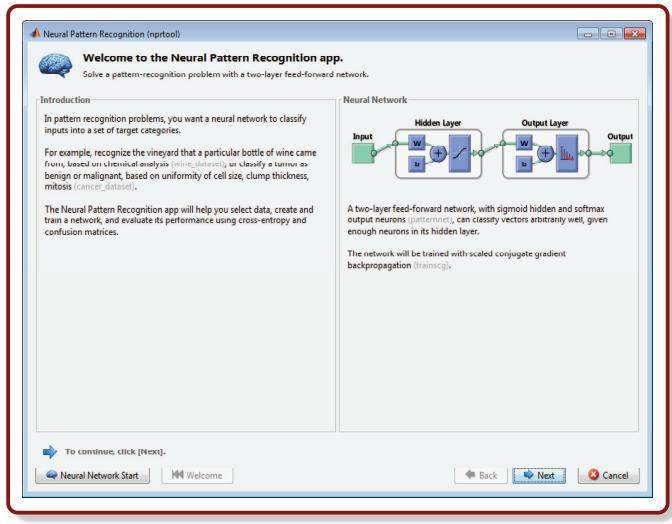


FIGURE 12.1: Interface of the *Neural Net Pattern Recognition* app

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 **y** vector of labels. Instead, transform the **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, **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 **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.

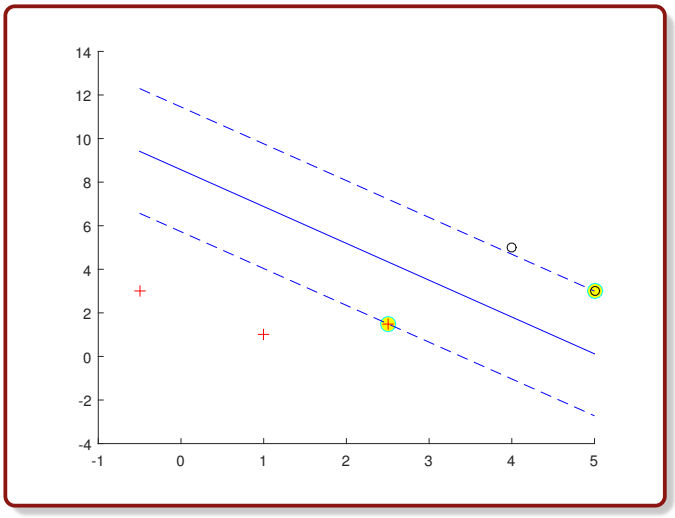


FIGURE 13.1: Plot of the data showing the slab and the support vectors.

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".

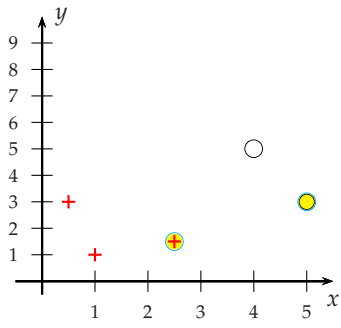
11. In numeral 2, you uploaded a dataset made of matrix `X2` and vector `y2`. If you plot this data, you will see that it is not linearly separable. On this dataset, use `fitcsvm` as you did in numeral 4, this time to get a structure "`SVMModel2`". Then plot your results in the following way: first, 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

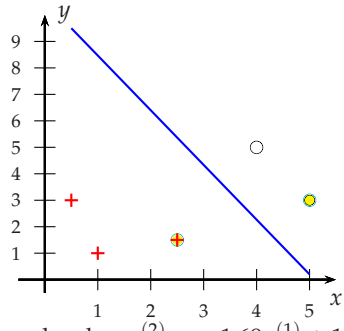
Exercises

2.



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

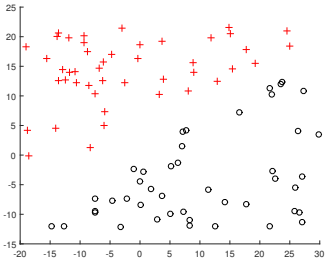
4.



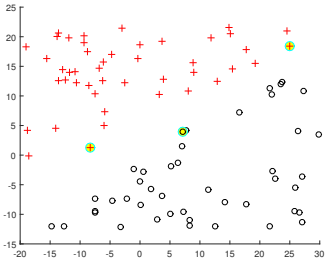
5. Upper border: $x^{(2)} = -1.69x^{(1)} + 11.45$
   Lower border: $x^{(2)} = -1.69x^{(1)} + 5.73$

Practical

3.

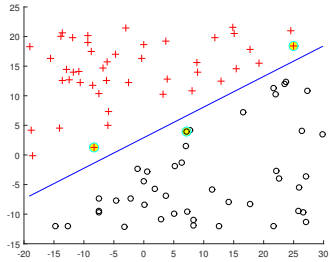


5.

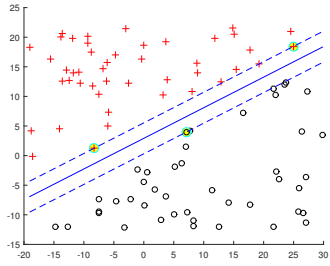


6. slope=0.5170; intercept=2.9171;

7.



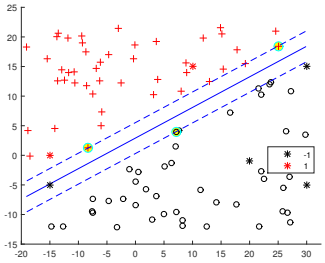
8.



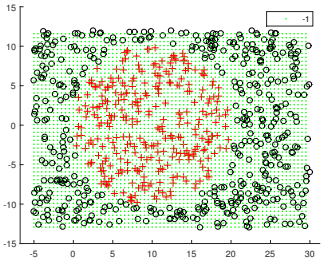
9.

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

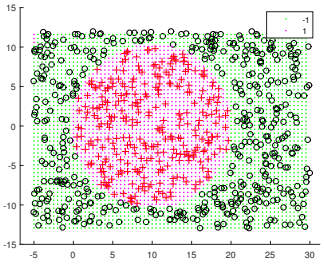
10.



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(x_new, x_i)
    store d(x_new, x_i) in a vector of distances, d
end for
sort distances in d from lowest to greatest.
take the k first elements in this sorted d.
assemble a vector a with the labels corresponding to these k first elements.
y_new ← mode(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 $\mathbf{X}$, representing with a little black circle the vectors of class 0; and with a red "$\times$" 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 $\mathbf{X}$. Put these values in a vector called "$\mathbf{d}$".
6. Sort the vectors in $\mathbf{X}$ from the one corresponding to the minimal value in $\mathbf{d}$ 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 $\mathbf{X}$ and the vector of corresponding labels, $\mathbf{y}$.
- 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.

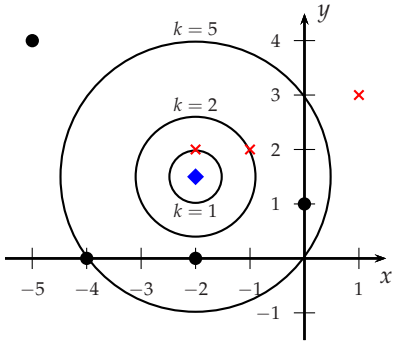
14. Compute the confusion matrix for these predictions.

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

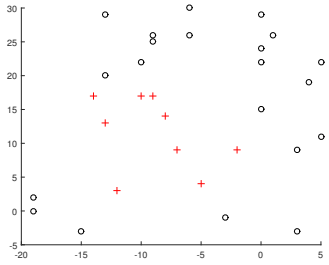
5. $\mathbf{d} = \begin{bmatrix} 1 & 9 & 4 & 5 & 2 & 13 & 32 \end{bmatrix}^T$

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

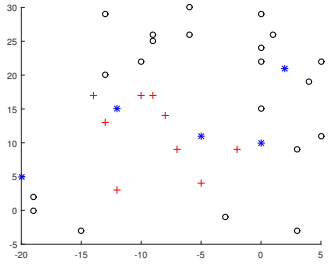
7. (a) 0
   (b) 0
   (c) 0

Practical

3.

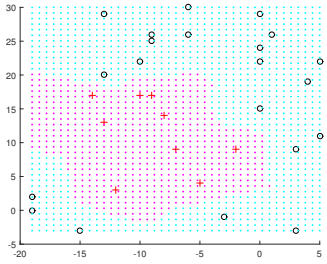


4.



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

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
```