Metadata of the chapter that will be visualized in SpringerLink

Book Title	Data Science in Practice	
Series Title		
Chapter Title	Machine Learning: A Concise Overview	
Copyright Year	2019	
Copyright HolderName	Springer International Publishing AG, part of Springer Nature	
Corresponding Author	Family Name	**Duarte**
	Particle	
	Given Name	**Denio**
	Prefix	
	Suffix	
	Role	
	Division	
	Organization	Universidade Federal da Fronteira Sul
	Address	Chapecó, Brazil
	Division	
	Organization	University of Skövde
	Address	Skövde, Sweden
	Email	duarte@uffs.edu.br
		denio.duarte@his.se
Author	Family Name	**Ståhl**
	Particle	
	Given Name	**Niclas**
	Prefix	
	Suffix	
	Role	
	Division	
	Organization	University of Skövde
	Address	Skövde, Sweden
	Email	niclas.stahl@his.se

Abstract	Machine learning is a sub-field of computer science that aims to make computers learn. It is a simple view of this field, but since the first computer was built, we have wondered whether or not they can learn as we do.

Chapter 3
Machine Learning: A Concise Overview

Denio Duarte and Niclas Ståhl

Machine learning is a sub-field of computer science that aims to make computers learn. It is a simple view of this field, but since the first computer was built, we have wondered whether or not they can learn as we do. In 1959, Samuel [40] proposed some procedures to build an algorithm intending to make computers play better checkers than novice players. It was an audacious goal mainly at that time when the available hardware was very limited. However, that shows the importance of machine learning since the first computers were introduced.

Nowadays, users demand computers to perform complex tasks and solve several kinds of new problems, while data are being produced from many devices (e.g., satellites, cell phones, sensors, among others). Researchers in all fields (e.g., statisticians, computer scientists, engineers, to cite some) have started the quest for making computers learn by proposing news techniques to meet the new users demands.

Data are the input of any machine learning system. Data contain examples from a given domain, and machine learning algorithms generalize the examples in the data to build mathematical models. The models can be used to predict new outputs from new examples. The data used as input in the training model are called training data.

This chapter aims to present an overview of machine learning and to serve as a road map to guide interested readers in applying machine learning to everyday problems and giving skills to become a data scientist. It is organized as follows: next section provides an overview of general issues in the field of machine learning. It also presents some examples to help readers to get through to the whole chapter. Section 3.2 discusses supervised machine learning algorithms: algorithms applied

D. Duarte (✉)
Universidade Federal da Fronteira Sul, Chapecó, Brazil
e-mail: duarte@uffs.edu.br; denio.duarte@his.se

D. Duarte · N. Ståhl
University of Skövde, Skövde, Sweden
e-mail: niclas.stahl@his.se

A. Said and V. Torra (eds.), *Data Science in Practice*, Studies in Big Data 46,
https://doi.org/10.1007/978-3-319-97556-6_3

when the training data have labels associated with every example. We present two common supervised learning techniques: regressors and classifiers. Some supervised algorithms for both techniques are also presented. The following section introduces another class of machine learning algorithms: unsupervised learning. The training data for unsupervised algorithm have no labels, so the learner aims at partitioning them into groups. Deep leaning is a new trend in machine learning and it is based on different architectures of artificial neural networks. We dedicate the entire Sect. 3.4 to these. Section 3.5 presents an overview of model assessment approaches. The model assessment is critical to validate the model regarding the quality of the prediction outputs. Another important issue in machine learning is the number of attributes in the training dataset. This issue is called dimension of the dataset. Section 3.6 presents some techniques to reduce the dimensionality to enhance the performance of the algorithms and help to visualize the data. The following section presents some final remarks about machine learning: data preprocessing (feature selection and scaling, missing values), bias, variance, over and underfitting. Finally, Sect. 3.8 concludes this chapter.

3.1 Introduction

Learning is a very complex process, and we cannot say, currently, that a computer can learn. With this in mind, our first definition of machine learning must be carefully reviewed. There is no general agreement about what is learning, however for human beings learning can be defined as (i) functionally as changes in behavior that result from experience, or (ii) mechanistically as changes in the organism that result from experience [6].

Computers are mathematical machines; thus, we have to consider learning as a computer program. Figure 3.1 presents pictorially traditional programming (a) and machine learning (b). Notice that, while traditional programming is concerned with finding the right output based on given inputs and a program, machine learning is concerned with finding a right program (later we call it a model) given a set of inputs and outputs (possibly empty). The learned program can now propose new outputs given new inputs.

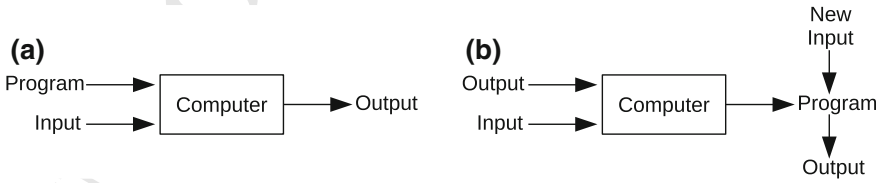


Fig. 3.1 Traditional programming (**a**) and Machine learning (**b**)

Based on Fig. 3.1, we can see how different learning means in human beings and computers. Machine learning algorithms try to create a model that represents the input to propose new outputs. This is the type of learning concerned in this chapter.

Back to the machine learning definition, we point out two definitions. The first one proposed by Samuel [40] who said that machine learning is a field of study that gives computers the ability to learn without being explicitly programmed. Remark that Samuel's definition was one of the first proposed definitions. Almost forty years later, Mitchell [32] proposed a more mathematical view of machine learning: a computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E.

Example 3.1.1 Assume we want to learn whether or not a given email is spam. In this case, we give to the algorithm a set of emails S_e and divide it into two subsets: not spam (S_{nse}) and spam (S_{se}). This step is the experience (E) we give to the algorithm. Based on E, our algorithm classifies emails as spam (and consequently as not spam). This step is related to the task (T). Finally, we want to know how well our algorithm performs in classifying spam emails. This step is related to the performance of our algorithm (P). Our goal is, then, to find a T with as good P as possible. Of course, T and P depend on the quality of E, e.g., if we have a good informative S_e, our machine learning algorithm may classify all our input emails correctly.

Figure 3.2 changes a little bit of the representation of Fig. 3.1. Firstly, we call *Training Set* (X) the input of our machine learning algorithm. Depending on the task we want to accomplish, X may have a label for every example. So, X can be represented as $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$ or $\{x^{(1)}, \ldots, x^{(m)}\}$, where m is the size of X (training set), and $y^{(j)}$ is the label of $x^{(j)}$ $(1 \leq j \leq m)$. The learning algorithm (we have thousand of options) takes X as input and builds a *Model* (also known as a *hypothesis*). After assessing our model (i.e., verify how good our P is), we can feed it with new examples (X') to have predicted outputs. Take into account that, depending on our target task, the predicted outputs can be a class (discrete value), continuous values, clusters, among others. This will be further elaborated in this chapter.

Based on what we have already seen, the machine learning process can be divided into four steps: (i) get the training set X, (ii) choose and implement a learning task based on X, (iii) build a model, and (iv) assess the model with new inputs. Remark that these four steps may be repeated until we have reached a good P. Keep in

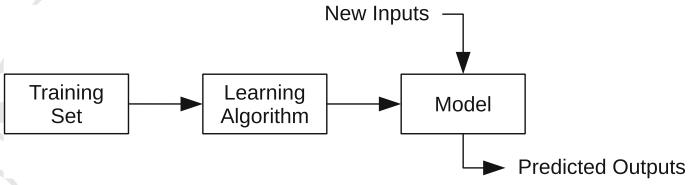


Fig. 3.2 Pictorial representation of the machine learning process

mind that when we have a machine learning problem to solve, the first challenge we face is which machine learning algorithm to use. There are thousands available, and each year other hundreds are proposed [8]. The set of possible available learning algorithms for a given machine learning problem is called *hypothesis space*. To not get lost in this huge set of choices, the learning components can be divided into three [8]:

1. Representation: a task must be represented by some algorithm. We must know what kind of learning we are interested in, and so, the hypothesis space is decreased based on the type of task.
2. Evaluation: the predicted outputs must be evaluated (assessed) to know how good the chosen representation is. Depending on the task, different evaluation functions can be used.
3. Optimization: based on the results of the evaluation component, optimization must be done. The aim of the learning algorithm is to maximize a given performance measure.

Table 3.1 presents some examples for each of the three components. For example, a *Decision Tree* may classify a training set into predefined classes. An evaluation component can be *accuracy*, i.e. how many classes are correctly classified. An *entropy* function can be used to measure the purity of the attributes within the tree, that is, the amount of information that would be needed to classify an example. Table 3.1 gives a little idea of the options we have when we design a machine learning system. Remark that we cannot pick an example for each column to design our system. Each representation has its own set of optimization and evaluation approaches. For example, we can use *Greedy Recursive Partitioning* for *Decision Trees*, and the model can be evaluated by *Accuracy*.

Although there does not exist a simple recipe to choose the best approach from one of the three components, the success (or not) of a learner depends on how well the problem is defined, as well as the quality of the training set. The former helps to find a representation that fits better to the problem, and the latter can be considered an essential *ingredient* of a learning system.

We illustrate what we have seen so far through a running example.

Table 3.1 Instances of the three components of learning algorithms

Representation	Evaluation	Optimization
Logistic regression	Accuracy	Gradient descent
Neural networks	Precision/Recall	Greedy search
Linear regression	Squared error	Beam search
Decision tree	Cost function	Linear programming
K-means	Root mean square error	Greedy steepest descent
Random forest		Stochastic approximation
Naive bayes		Greedy recursive partitioning

Table 3.2 Wind and temperature affecting pace

Wind speed (km/h)	Temperature (°C)	Pace (min)
10.5	12.3	3.5
8.9	15.4	3.2
20.2	13.7	5.5
5.10	3.1	4.0

Example 3.1.2 Table 3.2 presents an extract of a dataset about running performance (column *Pace*—the number of minutes it takes to cover a kilometer) based on the wind speed and the temperature. Suppose we want to predict a pace based on new information about the weather. X can be seen as (<10.5, 12.3>, <8.9, 15.4>,<20.2, 13.7>, <5.10, 3.1>) and y as (3.5, 3.2, 5.5, 4.0). The size of X is 4 and number of features is 2 (i.e., wind speed and temperature), and we want to predict a continuous value: the pace.

The number of features is usually known as the dimensionality of the dataset. The notion of dimensionality leads to a well-known problem in machine learning: the curse of dimensionality [21]. Considering a dataset as a set of points in a plane, the curse of dimensionality can be stated as follows: (i) learning algorithms generally work with interpolation to build models, (ii) interpolation is only possible if points are close to each other, (iii) if points are spread throughout a high dimensional space, the distance between them is large, and (iv) interpolation-based algorithms cannot build a model.

Besides, high dimensional dataset also leads to two problems: increase of computation cost and non-informative features. Suppose that we have, in our example, the following features: the running shoes price and quality. Although, quality and price are related to each other, in our dataset we can easily discard the feature price without losing essential information for our machine learning system. Notice that if we have a dataset with 200 features, discarding or merging some of them would not be an easy task. We deal with the dimensionality problem later in Sect. 3.6.

Example 3.1.3 Given Example 3.1.2 and $x' = <11.2, 10.0>$, we want to predict a new $\hat{y}$ such that $\hat{y}$ represents a valid pace value for the wind speed and temperature given. To accomplish the prediction, we have to build a model that describes well enough our training set (X). We show how to build a model using a simple linear regression (notice that from Table 3.1 we are choosing a representation for our problem). A model can be

$$\theta_0 + \theta_1 \times x_1^i + \theta_2 \times x_2^i = \hat{y}^i$$

where θ_j are the weights or parameters (sometimes denoted by W), x_k^i represents the kth feature of the ith example in X and $\hat{y}^i$ is the predicted output of the ith

example, and we want a $\hat{y}^i \approx y^i$. θ_0 represents the bias of the model (aka intercept). The challenge is to find θ_0, θ_1, and θ_2 such that our $\hat{y}^i$ value is as close as possible to y^i. Given the matrix $X_{m \times d}$ and the column vector $\Theta_{d+1 \times 1}$, we can implement our model using matrix multiplication. However, $X_{4 \times 2}$ and $\Theta_{3 \times 1}$ are not dimension-compatible. Remark that, in our representation, Θ will always have one more column in relation to X, and we can solve this problem adding a 1's column in X. As 1 is the identity element under multiplication, so, if we multiply θ_0 by 1, its value remains the same. Now, we can represent our model as:

$$\begin{bmatrix} 1 & 10.5 & 12.3 \\ 1 & 8.9 & 15.4 \\ 1 & 20.2 & 13.7 \\ 1 & 5.10 & 3.1 \end{bmatrix} \times \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \end{bmatrix} = \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \hat{y}_3 \\ \hat{y}_4 \end{bmatrix}$$

and, so, $\hat{y} = X \cdot \Theta$.

The next step is to find suitable values for the Θ. If Θ^T=[0, 0.5, −0.2] then $\hat{y}$=[2.79, 1.37, 7.36, 1.93]. Our Θ values made a fair prediction for the first example ($y_1 = 3.5$), but they failed for the others. To find fair values to Θ is an optimization problem (our third component). If we are using linear regression to represent our problem and an error function to evaluate it, we can use an optimization algorithm for finding the minimum of a function (e.g., *Gradient Descent*), and we get the following values $\Theta^T = [3.4, 0.192, -0.132]$, and, then, $\hat{y} = [3.8, 3.1, 5.5, 4.0]$. Finally, to verify if our model is performing well, we have to evaluate it (second component). For linear regression, we can use *Root Mean Squared Error* (RMSE), and the error is 0.16 (see Sect. 3.5 for details). The smallest the value of RMSE, the closer our model is to make good predictions. Given x', we have $3.4 + 0.192 \times 12.0 + -0.132 \times 11.0 = 4.2$, that is, when the wind speed is 12.0 and the temperature is 11.0, the probably pace would be 4.2.

We can evaluate our model using the same set used for training. However, the best way to test our model is against unseen examples. Later in this chapter, we describe some strategies on how to train and test machine learning models.

In this section, we presented an overview of machine learning. In the next sections, we describe approaches for implementing a range of types of algorithms to solve machine learning problems. We consider a broad classification of a learning task (i.e. machine learning algorithm): supervised and unsupervised.[1]

[1] There are other classes of machine learning algorithms: semi-supervised, reinforcement learning, recommender system. In this chapter, we focus on the two most popular ones. We refer [32] to the readers for classes not covered here.

3.2 Supervised Learning

Supervised learning can be applied when the dataset contains a set of labels (possibly unitary) for every example. Therefore, the dataset is divided into X, the features of the examples, and y, the labels. The labels help a learning algorithm to build the predicting model and act as a guide to the learners.

Labels may be discrete (classes) or continuous (numeric values). Depending on the type of label, we can apply classifiers or regressors. Besides, labels make the evaluation of a model easier since we have the ground-truth to compare with the predicted values. Table 3.2 shows a dataset that can be used for supervised algorithms. Remark that the label (*Pace*) represents continuous values, and, so, the dataset can be used as input for regression supervised algorithms. If we change the *pace* to discrete values (e.g., fast, slow, normal, etc.), our problem becomes a classification problem. Note that any regression problem can be turned into a classification problem by binning the continuous target values. Therefore, the first step of machine learning system design is to analyze the dataset to identify which representation must be used. In the following, we describe both regression and classification.

3.2.1 Regression

Regression is a type of supervised machine learning algorithm whose target variables are continuous values. Predicting currency exchange rates, temperatures, and the time when an event may occur are examples of regression problems since the predicted outputs are continuous values. In this section, we present some regression algorithms.

3.2.1.1 Linear Regression

When we face a machine learning problem with continuous target variables, we have to choose a representation based on regression algorithms. Linear regression is the most common algorithm used in regression and serves as the base to understand all other regression algorithms [42].

The mathematical definition of linear regression is given by the following equation:

$$h_\Theta(X) = \theta_0 + \theta_1 \times x_1^{(i)} + \cdots + \theta_m \times x_m^{(i)} \tag{3.1}$$

where θ_j are weights (θ_0 is the bias, and θ_k is the weight for the kth feature of $x^{(i)}$, $1 \leq k \leq m$), and $x_j^{(i)}$ is the jth feature of the ith example in X (dataset).

The best way to understand how linear regression works is to plot a graph with the *features* $\times$ *label*. However, most of the time the dataset is multidimensional, that is, it has more than one feature. Later in this chapter, we discuss dimensionality reduction, but, for the sake of simplicity, we consider only the feature *Temperature* from our

dataset (Table 3.2). Table 3.3 shows the original dataset from Table 3.2 extended with some new examples to have more points in the graph. Figure 3.3 shows the new dataset plotted in a 2D graph ($X \times y$).

Remark that, in Fig. 3.3a, there is a line drawn by $h_\Theta(X)$ with weights that do not represent the data points very well. However, in Fig. 3.3b, the average distance of the points to the line shows that $h_\Theta(X)$ better describes the dataset. We can verify whether or not our model fits the training dataset well by measuring the (Euclidean) distances between the points and the line. In Fig. 3.3, the red lines show pictorially some distances. Remark that one way to evaluate our model is to calculate the average distance between the points and the line.

Example 3.2.1 Based on Fig. 3.3b, we have $h_{([2.3,0.222])}([10]) = 4.52$, that is, when the temperature is 10 degrees, the runner will take 4.52 minutes to run 1 kilometer. For a temperature of 12.3, our model outputs a pace of 5.03 which is higher than the ground-truth value of 3.5 (see Table 3.3).

So far, we have chosen the representation of our model (i.e., Linear Regression), and we still need to choose approaches to evaluate and optimize the model. In Sect. 3.5, we present some approaches for model evaluation. However, if we use *Mean Square Error*, our model $\Theta = [2.3, 0.222]$ gets a score of 0.641. Knowing

Table 3.3 Table 3.2 with just one feature and new examples

Temperature (°C)	Pace (min)
12.3	3.5
15.4	3.2
13.7	5.5
3.1	4.0
11.3	4.6
10.8	4.3
9.7	4.0
4.5	3.5
5.3	3.3
5.2	3.8
7.4	3.4
6.2	4.2
7.8	4.3
8.5	3.8
12.3	5.3
14.3	5.2
13.2	4.4
9.8	4.6
8.3	4.6

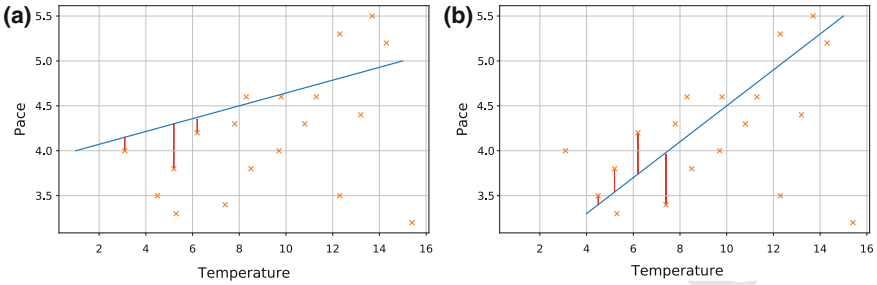


Fig. 3.3 Perform (the drawn line) of two regression models in the same dataset

that close to 0 is better, our model should be optimized. The optimization is the third component of our hypotheses space (see Table 3.1). Gradient Descent is a common solution to optimize (or train) a Linear Regression Model.

Gradient Descent updates the weights (Θ) based on a cost function. Equation (3.2) shows an implementation of a cost function.

$$J(\Theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\Theta(x^{(i)}) - y^{(i)})^2 \tag{3.2}$$

where m is the number of observations in the training dataset. Remark that $h_\Theta(x^{(i)})$ may be replaced by $\hat{y}$. The mathematical definition for the gradient descent is shown in Eq. (3.3).

$$\Theta = \Theta - \alpha\frac{\partial}{\partial\Theta}J(\Theta) \tag{3.3}$$

where α is the learning rate. Equation (3.4) shows Eq. (3.3) with the partial derivative computed for $J(\Theta)$.

$$\Theta = \Theta - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\Theta(x^{(i)}) - y^{(i)})x^{(i)} \tag{3.4}$$

The gradient descent works as follows: (i) we randomly initialize Θ (for linear regression all Θ can be initialized to 0),[2] (ii) define a value to α,[3] and (iii) we run gradient descent until it converges. We can use $J(\Theta)$ to stop the looping. When $J(\Theta)$ stabilizes, we consider that the gradient descent has converged.

Remark that the parameters (called hyper parameters) are essential for building good machine learning systems. The ordinary linear regression needs only a good

[2] For some representations, zero is not a good initial value. Random values from 0 to 1 work in most of cases.

[3] Select a small value to α, say 0.01, plot $J(\Theta)$ to identify how the gradient is converging, increase α (e.g., doubling its value) up to have an expected convergence.

value for the learning rate, but it is not always the case. There are some more sophisticated algorithms that we have to choose values for all the hyper parameters needed.

Sometimes the data points cannot be described by one straight line. In this case, we can still apply linear regression to the dataset. We just need to change our model. If the points are not linearly organized, we can apply polynomial models (in this case, a linear regression may be called a polynomial regression). Suppose we have a dataset with just one feature, and the data points represent a quadratic function. We could build a model as $h_\Theta(X) = \Theta_0 + \Theta_1 \times x_1 + \Theta_2 \times x_1^2$. It is not hard to implement since we insert a new column representing the squared feature. Therefore, the model becomes $h_\Theta(X) = \Theta_0 + \Theta_1 \times x_1 + \Theta_2 \times x_2$, where $x_2 = x_1^2$. Any polynomial function can be used to build a non-linear model. The characteristics of the dataset are the guide for choosing the best one. However, the cost function must be convex to be optimized with gradient descent (the cost function in Eq. 3.2 happens to be convex).

An alternative to training a learning model (i.e., solve the parameters Θ) is to use normal equation. Equation (3.5) gives the mathematical definition of normal equations.

$$\Theta = (X^T \cdot X)^{-1} \cdot X \cdot y \tag{3.5}$$

There are two advantages of normal equation over gradient descent: there is no learning rate and no iteration. However, the matrix that represents the dataset must be invertible, and the computational cost to multiple and invert matrix is high.[4]

Remark that $J(\Theta)$ guides gradient descent in finding the best weights for the model. However, sometimes $J(\Theta)$ fits well the training set ($J(\Theta) \approx 0$) but fails to generalize the test set ($J(\Theta) \gg 0$). This situation (aka overfitting) indicates that our model has been specialized for the training dataset, i.e., the noise in the dataset has been taken into account during the learning process. Overfitting may happen when the dataset is highly dimensional, and some features may be irrelevant in the training step.

There are several techniques to combat the overfitting, the most popular is to add regularization to the (cost) function. The regularization penalizes the function using the weights and some other parameters (see Sect. 3.7.2 for more details).

The algorithm described here for regression problems represents a small part of the hypotheses space for solving this kind of problem, but it can be used as the basis for understanding other regression algorithms.

3.2.2 *Classification*

Classification problem is similar to a regression problem; the only difference is the labels that are discrete values. With this in mind, a regression dataset may be transformed into a classification dataset. We have just to discretize the values, that is, group the continues values into classes. The label (*Pace*) in Table 3.2, for example, can

[4] Multiplication is $O(n^2)$, and inverse is $O(n^3)$.

Table 3.4 The dataset from Table 3.4 with Pace having discrete values

Wind speed (km/h)	Temperature (°C)	Pace	#Class
10.5	12.3	Fast	0
8.9	15.4	Fast	0
20.2	13.7	Normal	1
5.10	3.1	Normal	1

be discretized resulting in a new dataset shown in Table 3.4. The continuous values are transformed into discrete values as follows: (i) fast ($Pace \leq 3.5$), and (ii) normal ($Pace > 3.5$). As we are working with mathematical representations, the classes of the labels should be represented as numerical values. Therefore, *fast* is represented by 0, and *normal* by 1 (column *#Class*). Classes may be binary (as our example) or multi-classes (we could represent the Pace as fast, regular, normal, slow, among others). In the first case, we have $y \in \{0, 1\}$, and in the second case, $y \in \{0, 1, \ldots, k-1\}$ (where k is the number of classes). We focus this section on binary classification since they are more intuitive to understand. Besides, binary classifiers are applied to several situations: classification emails as spam or not, fraudulent transactions, problems regarding winners or losers, among others. In addition, any multi-class problem may be solved by dividing up the problem into many binary classification problems in a so-called *one* versus *all* classifications.

3.2.2.1 Logistic Regression

Logistic regression is one of the simplest and most efficient classifiers. The intuition behind logistic regression is similar to linear regression. We extend Eq. (3.1) to output $\hat{y}$ such that $\hat{y}$ in $[0, 1]$. Equation (3.6) presents a sigmoid (or logistic) function that always returns values between 0 and 1 inclusive and the logistic regression can be defined as $\sigma(h_\Theta(X))$.

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{3.6}$$

Sigmoid functions behavior as follows: $\sigma(z = 0) = 0.5, 0 \leq \sigma(z < 0) < 0.5$, and $0.5 < \sigma(z > 0) \leq 1$ (Fig. 3.4 shows pictorially this behavior). Remark that we may interpret the output of Eq. (3.6) as the probability of $y = 1$ (or $y = 0$). Therefore, the binary classifier can output 1 or 0 based on a threshold defined by the user. If we want an equal distribution, the value returned by $\sigma(z)$ can be rounded, and so we have $y = 1$ when $\sigma(z) > 0.5$, or $y = 0$, otherwise.

If we use the previous cost function in the logistic regression, we would have a non-convex function which could not be optimized with steepest gradient descent. If we try it anyhow, we would risk to find a local minimum instead of the global minimum (Fig. 3.5 shows pictorially this situation: solid circles represent local minima and the open one represents the global minimum). To avoid this, we consider the cost for

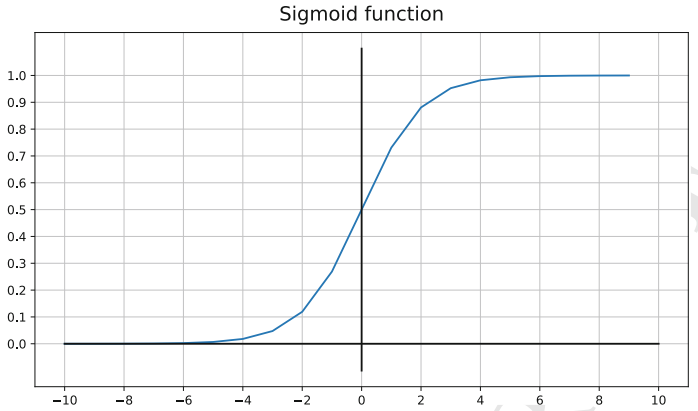


Fig. 3.4 The behavior of a sigmoid function

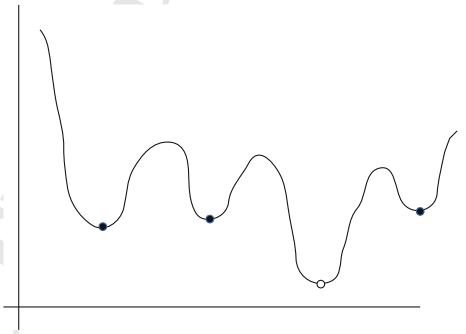


Fig. 3.5 Plot of a nonlinear cost function

$(\hat{y}, y)$ as follows:

$$cost(\hat{y}, y) = \begin{cases} -log(\hat{y}) & \text{if } y = 1 \\ -log(1-\hat{y}) & \text{if } y = 0 \end{cases} \tag{3.7}$$

Remark that the cost is 0 if y and $\hat{y}$ are equal to 1, and increases when $\hat{y} \to 0$. It is the behavior we want since the penalty has to increase when the distance of y and $\hat{y}$ increases. The same reasoning may be applied to when $y = 0$. Note that the Eq. (3.7) can be rewritten as $y \times (-log(\hat{y})) + (1-y) \times (-log(1-\hat{y}))$. Thus, we can define the cost function of linear regression as:

$$J(\Theta) = -\frac{1}{m}[\sum_{i=1}^{m} y^{(i)} log(h_\Theta(x^{(i)})) + ((1-y^{(i)}) log(1-h_\Theta(x^{(i)})))] \tag{3.8}$$

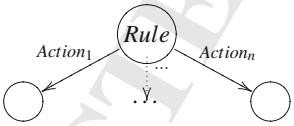
Notice that when $y = 0$, the first part of the summation (the left side of the addition) is canceled since y multiplies the log value, and the same reasoning can be applied to the right side when $y = 1$. The gradient descent remains the same as shown in Eq. (3.3).

If we plot our dataset in a vector space, logistic regression draws a line that separates the data points based on their classes. This separation is called a decision boundary. However, sometimes the classes cannot be separated by a straight line, and in this case, one solution is to transform the feature matrix into a higher dimensional space by adding new features with higher degree. Another solution is to apply another algorithm to the problem, for example, Support Vector Machines (SVM) [15].

One-versus-all is one of the approaches to deal with multi-class classification. This approach learns one class at a time, and thus, a dataset with k classes has k sets of Θs. For example, in a dataset with three classes (0, 1, and 2), we keep the label of first class and update the others to 1; we do the same for the second class, update the others to 0, and so on. To verify in which class an example e_i belongs, we apply every learned Θ to e_i, and the highest one corresponds to the predicted class of e_i.

3.2.2.2 Decision Trees

Decision trees are another representation for solving classification problems. The model is based on decision rules implemented in the nodes of a tree. This model is more understandable for humans than logistic regression. See a pictorial representation below:



Basically, every node represents a test to be performed on a single attribute, and a child node is accessed depending on the result of the test. The testing is repeated until it reaches a leaf node, and finally the class is found. Figure 3.6 presents a decision tree for the dataset from Table 3.4. Remark that the tree covers all cases of the dataset, and we may conclude that a runner will have a normal pace under temperatures below 12.3.

The choice of the attributes for each rule and node has a major role in the success of a decision tree. The criterion is based on the information gain of the attributes (features). The attribute that gives the greatest information gain becomes the root of the tree, and the internal nodes follow the ranking of the information gain. Entropy calculates the (im)purity of an attribute regarding the classes, and it may be used to calculate the information gain (i.e. how well an attribute can describe a given class) [3]. Equation (3.9) gives the mathematical definition of an entropy for a subset X_i:

$$H(X_i) = -p_i^+ log_2(p_i^+) - p_i^- log_2(p_i^-) \tag{3.9}$$

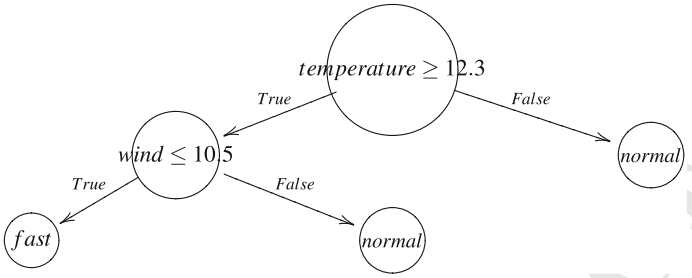


Fig. 3.6 A decision tree built from dataset in Table 3.4

where p_i^+ is the probability that a randomly taken example in X_i is positive and can be estimated by the relative frequency $p_i^+ = \frac{n_i^+}{n_i^+ + n_i^-}$; the same reasoning is used to calculate p_i^-.

The entropy for every value i of an attribute *attr* in X is calculated as follows (considering the *attr* has K different values):

$$H(X, attr) = \sum_{i=1}^{K} P(X_i) \times H(X_i)$$

where $P(X_i)$ is the probability of an example belonging to X_i and can be estimated by the relative size of subset X_i in X: $P(X_i) = \frac{|X_i|}{|X|}$. Finally, the mathematical definition of a knowing attribute *attr* information gain is given by Eq. (3.10).

$$I(X, attr) = H(X) - H(S, attr) \tag{3.10}$$

where $H(X)$ is the entropy of the whole dataset. Remark that if X is well balanced for the classes, $H(X)$ will be close to its maximum ($\approx$1).

Example 3.2.2 Let X be the dataset from Table 3.4, the result from Eq. (3.9) is: $H(X) = \frac{2}{2+2} \times log_2(\frac{2}{2+2}) - \frac{2}{2+2} \times log_2(\frac{2}{2+2}) = 1$. This means that X is well balanced. It is easy to see since X is composed of two positive and two negative examples. Let t_1 be the temperature greater or equal to 12.3 and the class normal be a positive example, the entropy is: $H(X, t_1) = \frac{1}{3} \times log_2(\frac{1}{3}) - \frac{2}{3} \times log_2(\frac{2}{3}) = 0.92$. For the temperate below 12.3, the entropy is by definition 1 since there is no negative example, and logarithm of zero is not defined. The total entropy of the attribute temperature is $H(X, temperature) = \frac{1}{4} \times 1 + \frac{3}{4} \times 0.92 = 0.94$. Finally, the information gain is $I(X, temperature) = 1 - 0.94 = 0.06$. If we calculate the information gain for a wind speed less or equal to 10.5, we will have the same result. That is, both attribute have the same information gain, and so both can be the root of the decision tree. Remember that the tree is organized by following the information gain of the attributes in descending order.

The equations and formulas above consider that the attribute values are discrete, but it is not always the case: decision tree can be induced from numerical attributes as well. An approach to discretize the values follows: an attribute *attr* is sorted, the range of each class for *attr* is calculated, and the ranges are ranked by information gain. Each range corresponds to a discrete value of *attr*. A range for an attribute *attr* can be [10, 20], i.e., $attr > 10 \wedge attr < 20$.

Decision trees are the basis for lots of other tree classifiers. One of the most effective one is the random forest. Roughly, random forest trees combine into an ensemble. N random samples are selected from the dataset X; each sample is used to build a decision tree with some samples of X. Therefore, random forest trees use N decision trees to build the best model for a given dataset.

Decision trees are proposed to deal with classification problems. However, there are several approaches to adapt decision tree algorithms to regression problems [26]. A regression tree is similar to a classification tree, except that the label y takes continuous numerical values and a regression model is fitted to each node to give the predicted values of y.

3.3 Unsupervised Learning

When the dataset has no labels, that is, there is no previous classification of the examples, we apply an unsupervised learning algorithm. The goal is to infer classes or groups from the dataset without the help of the labels. In this case, the dataset is in the form $X = \{x^{(1)}, \ldots, x^{(m)}\}$. Unsupervised learning is less objective than supervised learning, since there are no labels to guide the user for the analysis. The domain of the dataset must be known by the user to build useful models, otherwise, the results may not be understandable.

Although, unsupervised learning is harder to model than a supervised learning, the importance of such techniques is growing since there are more unlabeled data than the labeled ones. Besides, many learning problems are related to unsupervised problems: recommendation, classification of customer behaviors in a website, market segmentation, among others. Clustering is the most popular technique for unsupervised learning.

Clustering is about discovering semantically related groups in an unlabeled dataset. The number of groups (aka clusters) is defined by the user based on his/her knowledge of a dataset X. For example, let's say X represents examples of heights and weights of people, and we want to separate them into 3 *T-shirts* sizes (e.g., S, M, and L). The dataset can be split into 3 clusters, and, based on the user knowledge, each cluster represents the $height \times weight$ characteristic for each *T-shirts* size.

Data clustering has been used for [18] (i) gaining insight into data, generate hypotheses, detect anomalies, and identify salient features, (ii) identifying the degree of similarity among forms or organisms, and (iii) for organizing the data and summarizing it through cluster prototypes.

K-means is one of the most popular and easy to understand clustering algorithms [19]. The basic idea is to define k centroids that help to build the clusters. Every example in the dataset will be associated to one of the k centroids. The dataset is seen as a set of data points in a plane, and the algorithm tries to group them into clusters by measuring the distance between a given data point and a centroid.

Data: $X = \{x^{(1)}, \ldots, x^{(m)}\}$, K centroids $\mu_1, \ldots, \mu_K$
Result: A set of K centroids for X
Initialize the centroids (either $K \subset X$ or data points picking from the plane);
repeat
  for *each* $x^{(i)}$ *in* X **do**
    $c^{(i)} = \underset{k}{argmin}(\sum_{k=1}^{K} ||x^{(i)} - \mu_k||^2)$; // assign $x^{(i)}$ to the closest centroids
  end
  for *each* μ_i *in* K **do**
    // update the centroids with the average of the points associated to them
    $\mu_i = \frac{1}{|c^{(i)}|} \sum_{x^{(j)} \in c^{(i)}} x^{(j)}$;
  end
until K *converge*;

Algorithm 1: K-means pseudo code

Algorithm 1 presents a K-means pseudo-code. The two internal loops are the main parts of the algorithm. The first one associates each data point (an example from X) to a given centroids. The second loop updates the centroids positions by averaging the points associated to them. The main loop is repeated until the data points get stable in relation to their centroids.

Figure 3.7 shows pictorially the above steps of K-means (rounded points are centroids, and diamonds represent the data points). First, three centroids are picked from the hyperplane ((a), points black, red, and yellow). Next, all data points are associated to a centroid (b), new centroids are calculated ((c), the dotted arrow shows the source and target of each centroid), and finally, the data points are updated (d). Those steps are repeated until the centroids do not change anymore. The initial centroids play an essential role in K-means algorithms, and the resulting clusters may be different depending on the initial centroids.

Although, there are methods for selecting the so-called correct number of clusters (e.g., Silhouette and CH index methods [11]), the user knowledge plays an essential role to define the number of centroids (the number of clusters). Each cluster will have a semantic meaning in relationship to the domain of the dataset for the expert in the domain.

The K-means algorithm is a clustering algorithm based on partition, i.e., the idea behind is to consider the center of data the points as the center of the corresponding cluster. Another category of clustering algorithms are those based on a hierarchy. This kind of algorithm builds a hierarchical relationship among data to cluster them. Each data point, in the beginning, is a cluster itself. The closest clusters are merged.

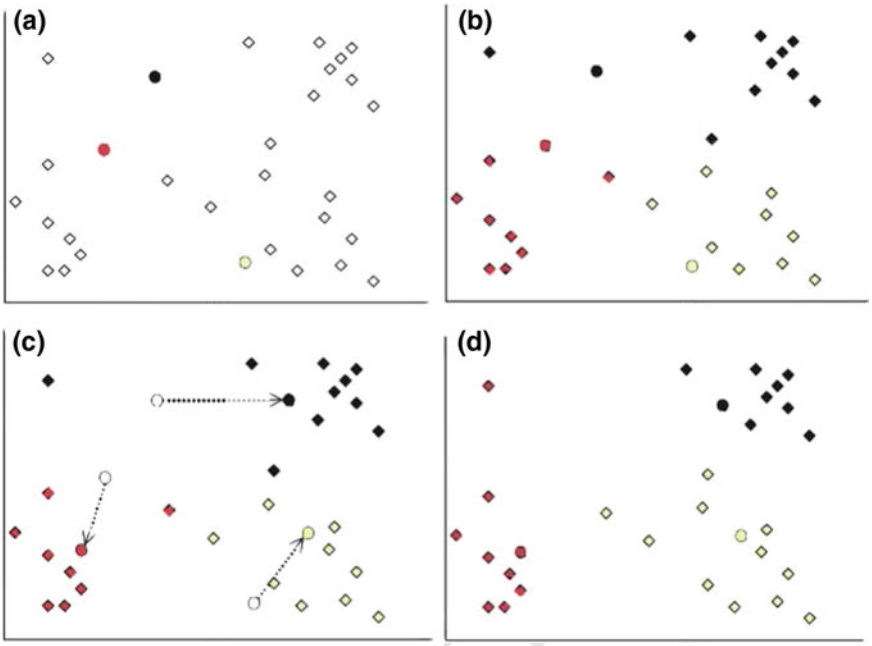


Fig. 3.7 First steps of K-means algorithm

The merge operation builds a dendrogram representing the nested clusters. The dendrogram shows the pattern and similarities of the clusters. Dendrograms can be seen as a hierarchical representation of the clusters showing the similarity or dissimilarity of them.

3.4 Artificial Neural Networks and Deep Learning

Deep learning is a type of representation learning where the machine itself learn several internal representations from raw data to perform regression or classification [23]. This is in contrast to more classical machine learning algorithms which often require carefully engineered features that are based on domain expertise [1]. Deep learning models are built up in a layer-wise structure where each layer learn a set of hidden representations, that in many cases cannot be understood by a human observer. The representations in each layer are non-linear compositions of the representations in the previous layer. This allows the model to first learn very simple representations in the first layers which are then combined into more and more complex and abstract representations for each layer. An example of this is that when deep learning models are used on images, they often start by learning to detect edges and strokes [44]. These are then combined into simple objects, objects that then are combined into

even more complex objects for each layer. Since each layer only learns from the representation of the previous layer, a general purpose learning algorithm, such as back propagation [22], can be used to train a given network.

3.4.1 *Artificial Neural Networks*

Most algorithms in deep learning are based on artificial neural networks [23]. In contrast to deep learning the field of artificial neural networks has been around for some time. It all started in 1943 when McCulloch and Pitts, a neuroscientist and mathematician, defined a mathematical model of how they believed a neuron worked in a biological brain [28]. The next step came in 1949 when Hebb came up with a rule that made it possible to train an artificial neuron to learn and subsequently recognize a set of given patterns [16]. In 1958 Rosenblatt, a psychologist, further generalised the works of McCulloch and Pitts and proposed a model, called the *perceptron*, for an artificial neuron [39]. The mathematical definition of a perceptron is given in Eq. (3.11), and a graphical representation is shown in Fig. 3.8.

$$y = f\left(\left(\sum_i x_i * w_i\right) + b\right) \tag{3.11}$$

The perceptron was then further analysed and developed by Minsky and Papert [30]. In the analysis of the perceptron, Minsky and Papert showed that a single perceptron was not sufficient to learn certain problems (e.g., nonlinear problems), for example, the XOR problem. Instead, they argued that multi-layered perceptrons were needed to solve such problems. However such networks were not possible to train at that time, this lead to an AI winter and very little research on ANNs were conducted on neural networks for some time. This has changed during the years,

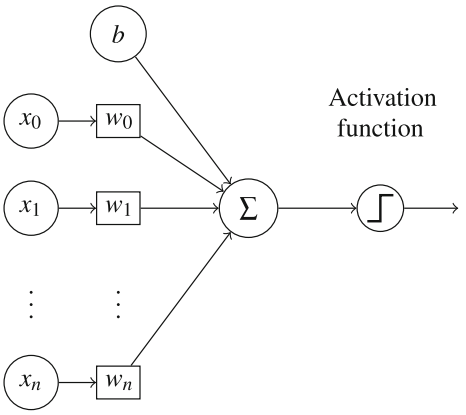


Fig. 3.8 A graphical illustration of a perceptron. The output of a perceptron is an activation function applied to the weighted sum of the inputs plus a bias. The mathematical definition of a perceptron is given by Eq. (3.11)

thanks to the increase in computational and improvements to the methodology, such as the introduction of the backpropagation algorithm, unsupervised pre-training [9] and the rectified linear unit [34]. These improvements have allowed researchers to build networks with many hidden layers, so called deep neural networks [23]. In the following sections, we present several different architectures of artificial neural networks used in deep learning.

3.4.2 *Feedforward Neural Networks*

A feedforward neural network is an artificial neural network where information only moves in one direction; thus feedforward networks are acyclical and therefore free of loops. The layout of a typical feedforward network is shown in Fig. 3.9. The most basic feedforward network is the perceptron [39] where the output is an activation function applied to the weighted sum of the input plus a bias. If the sigmoid function, described in Eq. (3.6), is used as the activation function a single perceptron performs exactly the same task as logistic regression (see Sect. 3.2.2). A standard architecture of feedforward networks is to arrange multiple neurons in interconnected layers. Each neuron in any layer, except the final output layer, has directed connections to all neurons in the subsequent layer. This types of networks are called *multilayer perceptrons*. As with the perceptron, the output that each neuron will propagate to the next layer is an activation function applied to the weighted sum of all inputs plus a bias. As long as the activation function is differentiable, it is possible to calculate how the output will change if any of the weights is changed, and thus the network can be optimized with gradient based methods.

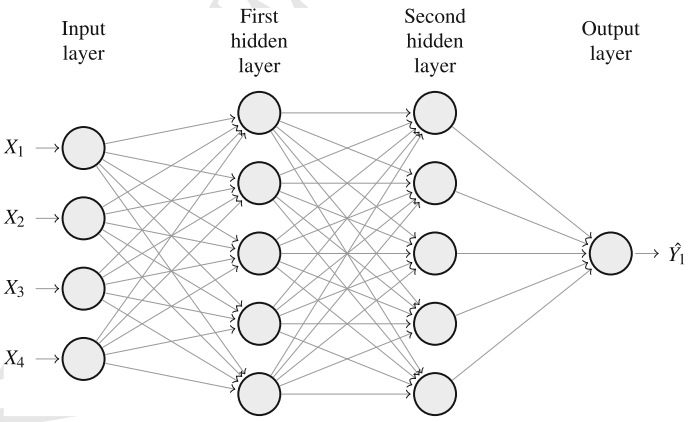


Fig. 3.9 The layout of a multilayer perceptron with two hidden layers, each having 5 neurons

3.4.3 Convolutional Neural Networks

Convolutional neural networks (CNNs) are a special type of neural networks that are mainly used in image analysis [25], but some researchers have used CNNs for natural language processing [24]. The main idea behind the CNN architecture is that basic features in a small area of an image can be analysed independently of its position and the rest of the image. Thus an image can be split up into many small patches. Each patch can then be analysed in the same way and independently of the other patches. The information from each patch can then be merged, to create a more abstract representation of the image.

This scheme is implemented in a CNN using two different steps; convolutional and sub-sampling steps. In a convolutional step, a feedforward neural network is applied to all small patches of the image, generating several maps of hidden features. In the sub-sampling step, the size of the feature map is reduced. This is often done by reducing a neighborhood of features to a single value. The most common way for this reduction is to either represent the neighborhood with the maximum or the average value. These two steps are then combined into a deep structure with several layers.

It has been shown that a CNN learns to detect general and simple patterns in the first layers, such as detecting edges, lines, and dots [44]. The abstraction of the learned features will increase in each layer. If, for example, the first layer detects edges and dots, the next layer may combine these edges and dots into simple patters. These patterns may then be combined into more complex and abstract objects in the next layer. One of the main benefits of this approach is that the CNN learns translation invariant features. Thus a CNN can learn general features about objects in an image independently of their position within the image.

3.4.4 Recurrent Neural Networks

A recurrent neural network (RNN) is a type of artificial neural network where there are cyclical connections between neurons, unlike feedforward networks which are acyclical [37]. This allows the network to keep an inner state allowing it to act on information from previous input to the network, thus exhibit dynamic temporal behaviour. This makes RNNs optimal for the analysis of sequential data, such as text [29] and time series [5]. One big problem with recurrent neural networks, which also occurs in deep feedforward networks, is that the gradients in the backpropagation will either go to zero or infinity [36]. This has however been partially solved by the introduction of special network architectures, such as the long short term memory (LSTM) [14] and the gated recurrent unit (GRU) [4].

3.4.5 Generative Adversarial Networks

Generative adversarial networks (GANs) were introduced by Ian Goodfellow in 2014 [13]. The idea behind a GAN is that we have two artificial neural networks that compete. The first network, called the generator, tries to generate examples following the same distribution as the collected data. While the second network, called the discriminator, tries to distinguish between the examples that are generated by the generator and the data that are sampled from the real data distribution.

The training of these two networks consists of two phases where the first part aims to train the generator and the second to train the discriminator. In the first phase, the generator creates several examples and gets information about how the discriminator would judge these examples, and in which direction to change these examples so that they are more likely to pass as real data to the discriminator. In the second phase, several generated and real examples are presented to the discriminator, that classifies them as real or generated. The discriminator is then given the correct answers and how to change its settings to preform better when classifying future examples. This can be compared to the competition between a money counterfeiter and a bank. The task of the counterfeiter is to generate fake money, and the bank should be able to determine if money is faked or not. If the counterfeiter gets better at creating new fake money, the bank must take new measures to discover the fake money and if the bank gets better at discovering fake money, the counterfeiter must come up with better and creative ways to create new money. The hope when training a GAN is that the generating network and the discriminating network will reach a stalemate where they are both good at their tasks. Successful works including GANs, are the generation of images of human faces [12], images of hotel rooms [38] and the generation of text tags to images [31].

3.5 Model Evaluation

There are many representations to build models from data. In the previous sections, we have seen some of them. However, we need to evaluate the built model to check how well it performs on unseen examples, that is, how well it generalizes the training dataset. The design of machine learning system, as stated before, is composed of several steps: (i) the choice of a dataset as input, (ii) the choice of a representation for a learner, (iii) an approach to optimize the model, and, finally, (iv) an evaluation of the model. The evaluation (or assessment) must be done in a dataset not used for training. Basically, the original dataset is split into two subsets: the training and the test sets. The usual approach to dividing the dataset is as follows: 70% for training and 30% for testing. Remark that the number of examples in each subset depends on the number of example in the original dataset, and the selection of examples for each subset must be balanced (mainly in classification problems), that is, each subset must have representative information of the domain to be modeled.

Another important remark is that, during the training phase, we have to test our algorithm (or algorithms) using different hyper parameters (e.g., learning rate, node purity, number of clusters, etc.). In this scenario, we may also divide the training set into cross-validation sets (or split the original dataset into three subsets: training, validation, and test). Therefore, the training data is used to training some learning algorithms. In the validation set, the performance of trained algorithms are evaluated, and thus, the best one is chosen to model our problem. Moreover, the test data is used to evaluate the chosen model against new examples.

In the training set, we use a loss function (or another similar function) to verify whether or not our model is converging. When we are satisfied with the results in the training set, our model is run against the test set, and, depending on the representation used, we choose a metric to evaluate the predictions made by the model. It is clear that a metric for classification is different from a metric for regression, and it is not the same for unsupervised approaches. In the following, we present some metrics for the representations discussed in the previous sections.

3.5.1 Regression

The regression learners predict continuous values, and the metrics find how close a predicted value is to the real value (the ground-truth value). Therefore, most of the approaches are based on the distance between the ground-truth and predicted values (represented by $y - \hat{y}$).

Some popular metric functions for regression are:

- R^2 score (aka coefficient of determination) is a number that indicates the proportion of the variance in the predicted output from the real output. It is calculated as follows: $R^2 = 1 - \frac{\sum_{i=1}^{m}(y^{(i)} - \hat{y}^{(i)})^2}{\sum_{i=1}^{m}(y^{(i)} - \bar{y}^{(i)})^2}$, where m is the size of the (test) dataset, $\hat{y}$ is the predict value, and $\bar{y}$ is the average of ground-truth values. The closer R^2 is to one, the better.
- Mean square error (MSE) measures the average of the squares of the errors of the predict and real value. The error means the difference between the two values (predicted and truth). The differences are also called residuals. The mathematical definition is given as follows: $MSE = \frac{1}{m}\sum_{i=1}^{m}(y^{(i)} - \hat{y}^{(i)})^2$. The closer MSE is to zero, the better.
- Root mean square error (RMSE) is a measure of the differences between values (sample and population values) predicted by a model or an estimator and the truth-ground values. RMSE is the square root of the value calculated by MSE. So, its mathematical definition is $RMSE = \sqrt{2}MSE$. The closer RMSE is to zero, the better.

- Mean absolute error (MAE) is similar to MSE, but it does not square the error. The absolute value of the difference is used instead. It is defined as follows: $MSE = \frac{1}{m}\sum_{i=1}^{m} |y^{(i)} - \hat{y}^{(i)}|$.
- Mean absolute percent error (MAPE) is another metric to evaluate regression models, and the error expressed in generic percentage terms is: $MAPE = (\frac{1}{n}\sum_{i=1}^{m} \frac{|y^{(i)} - \hat{y}^{(i)}|}{|y^{(i)}|}) \times 100$

MAPE and MAE are less sensitive to the occasionally very large error because they do not square the errors. Therefore, if we want our model to ignore big prediction errors, MAPE and MAE may be used. However, the metric which is considered as *one size fits all* is RMSE [43].

3.5.2 Classification

The metrics for classification problems are a little bit easier to apply on the model than the regression ones. Roughly speaking, the metrics are based on counting how many predicted classes equals to the observed ones. We use the binary classification to present the metrics since it is more intuitive to understand. The same reason is applied to multi-class prediction.

The simpler way to evaluate a classifier is when the classes are well balanced in the dataset (training and test). Accuracy is the metric for this scenario. The predicted classes are matched against to the observed ones, and the number of matched ones is divided by the size of the dataset: $\frac{1}{m}\sum_{i=1}^{m} y^{(i)} == \hat{y}^{(i)}$, considering that *false* is 0, and *true* is 1.

In most of the cases, the classes in a dataset are skewed, that is, the number of classes is not balanced. For instance, in a dataset with examples of benign and malignant tumors, maybe most of the examples are labeled as benign tumors. If 96% of the tumors are labeled as benign, and the model outputs benign for every example, its accuracy will be 96% (a very good accuracy); however, we know that the model is not able to predict malignant tumors.

To overcome this problem, several metrics have been proposed, and we discuss three of them: precision, recall, and F_1-score.[5] First, we present some definitions to help present the metrics above. Table 3.5 presents a taxonomy of class classification that is used to classify the predicted classes in relation to the ground-truth ones. The true positive (TP) means that the classifier matches the positive classes, and the false positive (FP) implies that the classifier outputs negative classes as positive ones. The same reasoning for negative classes: true negative (TN) indicates negative classes are predicted correctly, and false negative (FN) indicates positive classes are predicted as negative ones. We remark that the confusion matrix can be extended for the multi-class problem [2].

[5] F_1-score is a specialization of F_β-score that is not covered in this chapter.

Table 3.5 The confusion matrix

	Positive	Negative
Positive	True positive (TP)	False positive (FP)
Negative	False negative (FN)	True negative (TN)

Precision, recall, and F_1-score can be defined as follows:

$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

$$F_1\text{-}score = \frac{2 \times precision \times recall}{precision + recall}$$

The precision metric is used when we want exactness, that is, our classifier is covering the positive classes confidently. On the other hand, recall means completeness, that is, how many positive examples our classifier has missed. If we want to balance between recall and precision, F_1-score gives the harmonic mean of precision and recall.

Remark that accuracy can be also calculated as follows: $\frac{TP+TN}{TP+FP+TN+FN}$.

3.5.3 Clustering

The evaluation of the clusters resulted from a clustering algorithm is a not easy task since there are no true labels to compare with the clusters. The evaluation can be divided into two categories: internal and external. The internal category measures the quality using the training data, and the external category uses the external data (test set). However, the external evaluation is not completely accurate as compared to the methods for supervised learning [10].

Silhouette Coefficient is a (internal) metric to measure the quality of the built clusters. It is a popular method that combines both cohesion (similarity between an object and its cluster) and separation (similarity between an object and other clusters). The silhouette coefficient for an individual object (example or data point) can be computed as follows: (i) given an example e_i, calculate its average distance to the other examples in the same cluster (a_{ei}), (ii) do the same using the other clusters (b_{ei}), and (iii) the silhouette coefficient for e_i is $s_i = \frac{b_i - a_i}{max(b_i, a_i)}$. The average of all coefficients s_k can be calculated to find the clustering coefficient.

Remark that the coefficient can be a value between -1 and 1. A negative value of s_i is not desirable since it indicates that the average distance of e_i to its cluster is

greater than to the other clusters. On the other hand, a positive value of s_i is an ideal value, and $s_i = 1$ indicates that the the average distance of a_{ei} is 0.

Rand Index (RI) is a metric for external evaluation. It compares the predicted clusters to the real clusters (manually assigned by an expert user), and it is similar to the accuracy metric for supervised algorithms. Here is how to calculate RI:

1. Let X be a dataset, C_p be a clustering (set of clusters) build by the clustering algorithm (predicted), and C_r be a set of ground-truth clusters;
2. TP is the number of examples (data points) belonging to the same clusters in C_p and C_r;
3. TN is the number of examples (data points) belonging to different clusters in C_p and C_r;
4. FP is the number of examples (data points) belonging to a cluster in C_p but to a different cluster in C_r;
5. FN is the number of examples (data points) belonging to a different cluster in C_p but the same cluster in C_r
6. Rand Index is calculated as follows: $RI = \frac{TP + TN}{TP + TN + FP + FN}$.

We remark that RI is calculated exactly as accuracy.

Metrics are essential tools to evaluate a machine learning system. Each one must be carefully studied to understand the behavior of the built model. Some metrics can be affected by noise in the data (aka outliers), and others may smooth the effects of noise.

3.6 Dimensionality Reduction

Dimensionality reduction plays an essential role in machine learning. Its goal is to decrease the number of features of a dataset. As an example, let's suppose that we want to identify objects in a set of images. Each image is a 100×100 pixels, thus we have 10, 000 features. An approach to reduce the dimensionality can bring the number of feature to 1000. Therefore, dimensionality reduction may be applied to:

- Compress data in the main and secondary storage.
- Speed up learning algorithms.
- Visualize the dataset in 2D or 3D planes.

The reduction may also merge the more correlated features into one (or more). For instance, a dataset can have a feature f_1 that represents the height in centimeters and another feature f_2 that also represents the height but in inches. Based on the correlation of f_1 and f_2, a dimensionality reduction technique may merge both features to form a new one.

Given a dataset $X = \{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$ (possibly without labels $y^{(i)}$), the dimensionality reduction aims to transform $x^{(i)} \in \mathbb{R}^d$ into $z^{(i)} \in \mathbb{R}^k$ (where $k < d$), resulting in $X' = \{(z^{(1)}, y^{(1)}), \ldots, (z^{(m)}, y^{(m)})\}$. Figure 3.10 shows a 3D dataset

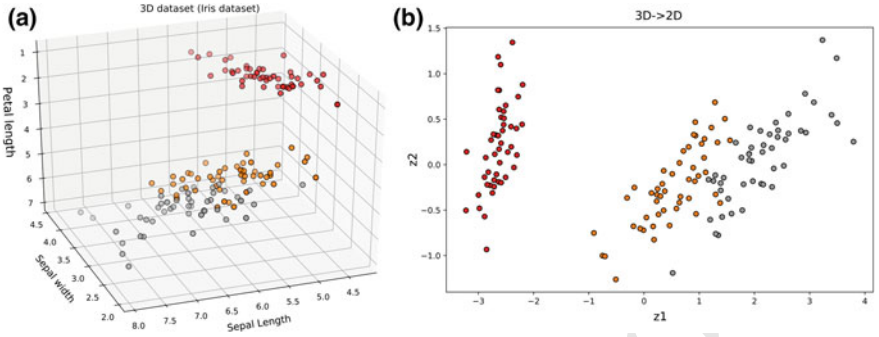


Fig. 3.10 A 3D dataset reduced to 2D dataset

(three features a) reduced to a 2D dataset (two features b). The reduction was done using the PCA technique.

3.6.1 Principal Component Analysis (PCA)

PCA reduces the dimensionality of a dataset by projecting vectors onto the plane and minimizing the projection distance error between the points and the projected vector. It can be described as follows: given a dataset with d dimensions, find k vectors $\mu^{(1)}$, $\ldots$, $\mu^{(k)}$ onto which to project the data, so as to minimize the projection squared error.

To find the vectors the following steps must be done: (i) find the covariance matrix of the dataset: $\Sigma = \frac{1}{m}\sum_{i=1}^{m} x^{(i)} \cdot x^{(i)^T}$ (Σ is a $d \times d$ matrix), calculate the eigenvector[6] U of Σ (U is a $d \times d$ matrix), and reduce X to k-dimensions based on U as follows $X_{reduced} = U[:, 1:k]^T \cdot X^T$. Remark that the new number of dimensions is taken from U which represents the vectors $\mu(i)$. We can approximately reconstruct X from $X_{reduced}$ and $U[:, 1:k]$ as follows $X_{app} = U[:, 1:k] \cdot X_{reduced}$

The ideal number of new dimensions (k) is found giving the percentage of variance to retain. If we want to retain 99% of variance, the threshold φ must be set to 0.01. We find the threshold as follows: $\frac{\frac{1}{m}\Sigma_{i=1}^{m}|x^{(i)}-x_{app}^{(i)}|^2}{\frac{1}{m}\Sigma_{i=1}^{m}|x^{(i)}|} \leq \varphi$. To find the best k, we test the variance retained with $k = 1, \ldots, d-1$ until we reach the desired threshold.

PCA performs a linear mapping to merge points in the plane. However, some features in a dataset may have more complex polynomial relationships. Therefore, there are some techniques to deal with nonlinear dimensionality reduction. t-Distributed Stochastic Neighbor Embedding (t-SNE) is an approach based on probability distributions with a random walk on neighborhood graphs to find the structure within the data [27]. t-SNE converts distances between data in the original space to probabilities.

[6] Definition of eigenvectors can be found in traditional books of linear algebra.

Another approach that works with nonlinear dimensionality is an extension of PCA: nonlinear PCA. Nonlinear PCA is performed using a five-layer neural network [35] that captures the complex nonlinear relationship between the features.

The decision in using a linear or nonlinear approach must be based on the characteristic of the dataset; however, sometimes it is not easy to identify whether or not features are linearly or nonlinearly correlated. A rule of thumb is to start with PCA. If it does not work well, use more sophisticated techniques.

3.7 Final Remarks

The design of a machine learning system is composed of several steps: from choosing a domain (dataset) to building a model to accurately predict new information about the given domain. This section closes this chapter by presenting some issues that are orthogonal to the subjects discussed so far. An essential step in machine learning is to prepare the dataset as a proper input of a learning algorithm. The quality of the dataset has a high impact on the performance of machine learning-based methods. Thus, in this section, we present some issues about preprocessing the data and checking the behavior of the learning algorithm.

3.7.1 Data Preprocessing

Some characteristic of a dataset may have adverse influence on the learning results or even may be not suitable for certain learning algorithm classes. Several learning algorithms deal with only numerical values, and, in this case, features that are not numerical must be either discarded or transformed into numerical values (discretization may be applied). For example, a feature that stores gender as m and f may have the values replaced by the numerical values 1 and 2, respectively. The contrary is also true: a numerical value can be transformed into a string to speed up some learning algorithms (e.g., decision trees).

A dataset can also have features on very different scales, that is, one feature can store values on the order of thousands, and another on the order of ten. An example is the price of an apartment and number of bathrooms. In this case, we can use a technique called feature scaling, that is, all features are scaled to a same range of values. The two most common technique are: standardization ($\frac{X-\bar{X}}{\sigma}$) and rescaling ($\frac{X-min(X)}{max(X)-min(X)}$) feature rescaling. The former makes all features to have the average close to 0, and the latter the features will have values between 0 and 1. The feature scaling helps an optimizer to converge faster.

Missing data is another problem that must be solved during the preprocessing stage.[7] Three approaches may be used to address it: analyze only the available data, impute missing values in the dataset, and use a learning algorithm that deals with missing data.

For the first case, examples or features are deleted from the dataset. This is recommended when there are not many missing values. The dataset size allows the removal of some examples or features, and the values are randomly missing.

Missing values' imputation aims to replace missing values with some plausible values. The new values are calculated based on some traditional statistic methods (e.g., mean, the most frequent value, or median), or some other more sophisticated approaches (e.g., Expectation Maximization [7], Shell Neighbor Imputation [45]).

The third approach is to use a learning algorithm that integrates components to deal with missing values. Probably, these extensions of traditional learning algorithms use some statistical methods cited previously, e.g., [41].

In the dataset preprocessing step, we can also drop unnecessary features (aka attribute or feature selection) or create new features based on the existing ones (aka attribute or feature transformation). Imagine a dataset with characteristics of cars, and we want to learn the safety of the cars (e.g., low, medium, high). A feature like license plate will be not important for the learning algorithm. On the other hand, we can build new features from existing ones to improve the learning algorithm performance, e.g., based on the *weight* of the car and its *horsepower*, a new feature *power-to-weight ratio* can be created. The feature creation is very useful, for example, when we want to build a polynomial model for linear regression (as we saw in Sect. 3.2.1.1).

Future selection is a largely used tool in preprocessing dataset to improve a learning algorithm's performance, and there are many approaches to accomplish it. Most of them identify the relevance of a feature in relationship to the others. Based on the relevance, a subset of features can be extracted from the original dataset [33].

The preprocessing step plays an essential role to build good machine learning systems. There is no good rule of thumb to guide during this step. However, the best thing to do is to test several approaches by assessing the results. Another issue to consider is that an approach may fit very well in one dataset but may have poor performance in another one.

3.7.2 *Bias, Variance, Under, and Overfitting*

During the learning step, the model may suffer from some learning problems. The most two common ones are under and overfitting. They are intimately related to the bias and the variance of the model. Bias and variance are used to identify some issues in assessing the ability of a learning method to generalize.

[7] For the sake of simplicity, we consider an invalid value (e.g., mixed characters and numerical values) for a feature as missing data too.

Roughly speaking, variance means how much structure from the dataset the model has learned, while bias means how much structure from the dataset the model has not learned. That is, bias is a learner tendency to learn the same wrong thing consistently, and variance is the tendency to learn random things regardless the dataset [8]. Intuitively, a biased model has a poor performance in the training set and in the test set (as expected) while, a model with variance has good performance in the training set but poor performance in the test set. When a model has high variance, it means that the model has caught all the details of the training set (including noise and outliers), thus cannot be generalized to unseen examples.

Bias and variance can be identified by verifying the performance of the model in both the training and test set. Considering that a model performs better in the training set, the behavior in the test set must follow the performance in the test sets. The ideal scenario would be with low bias and low variance, that is, neither does the model make a strong assumption regarding the dataset nor does it learn useless characteristics from the dataset.

The overfitting and underfitting problems are usually fixed using a (cross) validation dataset. Models and their parameters are trained in the training set, ranked in the validation set, and the best model is evaluated in the test set. Another tool widely used to combat the over/underfitting is to add a regularization term to the evaluation function. For example, we can add to the gradient descent equation (Eq. 3.4) a regularization term:

$$\Theta = \Theta - \alpha[\frac{1}{m}\sum_{i=1}^{m}(h_\Theta(x^{(i)}) - y^{(i)})x^{(i)} + \lambda\sum_{j=1}^{m}\Theta_j] \tag{3.12}$$

where λ represents the strength of the regularization. Clearly, if λ is equal to 0 then there is no regularization (or penalty).

Multi-label and multi-target[8] algorithms are classifiers that learn a vector of values from the observed data (examples). Most traditional supervised learning algorithms are extended to deal with multi-label or multi-target values. We do not cover these extensions in this chapter, and details can be found in [17].

3.8 Conclusions

In this chapter, we presented a comprehensive overview of various aspects of machine learning. The subjects introduced are widely discussed in the literature; however, this chapter intends to be a starting point for those who are interested in applying machine learning techniques to the real-world problems.

[8] Also known as multivariate or multi-output regression.

Even though, we find many researches on machine learning, it still remains a young field with many under-explored research opportunities. In addition, it has a lot of *folk wisdom* that can be hard to come by, but helps its development [8, 20].

Besides, machine learning plays an important role for data science. The knowledge brought by machine learning in building models for prediction makes it an essential tool for those wanting to extract information and knowledge from data.

Acknowledgements Denio Duarte is partially funded by Coordenadoria de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) under process number 88881.119081/2016-01—Brazil during his visit to Skövde Artificial Intelligence Laboratory (SAIL) at University of Skövde (HiS).

References

1. Bengio, Y., Courville, A., & Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *35*(8), 1798–1828.
2. Boutell, M. R., Luo, J., Shen, X., & Brown, C. M. (2004). Learning multi-label scene classification. *Pattern Recognition*, *37*(9), 1757–1771.
3. Bratko, I., Michalski, R. S., & Kubat, M. (1999). *Machine learning and data mining: Methods and applications*.
4. Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2015). Gated feedback recurrent neural networks. In *International Conference on Machine Learning* (pp. 2067–2075).
5. Connor, J. T., Martin, R. D., & Atlas, L. E. (1994). Recurrent neural networks and robust time series prediction. *IEEE Transactions on Neural Networks*, *5*(2), 240–254.
6. De Houwer, J., Barnes-Holmes, D., & Moors, A. (2013). What is learning? On the nature and merits of a functional definition of learning. *Psychonomic Bulletin & Review*, *20*(4), 631–642.
7. Dempster, A. P., Laird, N. M., & Rubin, D. B. (1977) Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 1–38.
8. Domingos, P. (2012). A few useful things to know about machine learning. *Communications of the ACM*, *55*(10), 78–87.
9. Erhan, D., Bengio, Y., Courville, A., Manzagol, P. A., Vincent, P., & Bengio, S. (2010). Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research 11*, 625–660.
10. Färber, I., Günnemann, S., Kriegel, H. P., Kröger, P., Müller, E., & Schubert, E., et al. (2010). On using class-labels in evaluation of clusterings. In *Multiclust: 1st International Workshop on Discovering, Summarizing and Using Multiple Clusterings Held in Conjunction with KDD* (p. 1)
11. Fujita, A., Takahashi, D. Y., & Patriota, A. G. (2014). A non-parametric method to estimate the number of clusters. *Computational Statistics & Data Analysis*, *73*, 27–39.
12. Gauthier, J. (2014). Conditional generative adversarial nets for convolutional face generation. In *Class Project for stanford CS231N: Convolutional neural networks for visual recognition* (Vol. 2014, No. 5, p. 2). Winter Semester
13. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., & Ozair, S., et al. (2014). Generative adversarial nets. In *Advances in neural information processing systems* (pp. 2672–2680).
14. Graves, A. (2013). *Generating sequences with recurrent neural networks*. arXiv:1308.0850.

15. Gunn, S. R. (1998). Support vector machines for classification and regression. Technical report, Faculty of Engineering, Science and Mathematics–School of Electronics and Computer Science.
16. Hebb, D. (1949). *The organization of behavior: A neuropsychological theory*. Wiley
17. Izenman, A. J. (2008). Modern multivariate statistical techniques. Regression, classification and manifold learning.
18. Jain, A. K. (2010). Data clustering: 50 years beyond k-means. *Pattern Recognition Letters*, *31*(8), 651–666.
19. Jain, A. K., Murty, M. N., & Flynn, P. J. (1999). Data clustering: A review. *ACM Computing Surveys (CSUR)*, *31*(3), 264–323.
20. Jordan, M. I., & Mitchell, T. M. (2015). Machine learning: Trends, perspectives, and prospects. *Science*, *349*(6245), 255–260.
21. Keogh, E., & Mueen, A. (2010). *Curse of dimensionality*. US: Springer.
22. Le Cun, Y., Touresky, D., Hinton, G., & Sejnowski, T. (1988). A theoretical framework for back-propagation. In *The connectionist models summer school* (Vol. 1, pp. 21–28).
23. LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, *521*(7553), 436–444.
24. LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278–2324.
25. LeCun, Y., Jackel, L., Bottou, L., Brunot, A., Cortes, C., & Denker, J., et al. (1995). Comparison of learning algorithms for handwritten digit recognition. In *International Conference on Artificial Neural Networks*, Perth, Australia (Vol. 60, pp. 53–60).
26. Loh, W. Y. (2011). Classification and regression trees. In *Wiley interdisciplinary reviews: Data mining and knowledge discovery* (Vol. 1, No. 1, pp. 14–23).
27. Maaten, L. V. D., & Hinton, G. (2008). Visualizing data using t-SNE. *Journal of Machine Learning Research 9*, 2579–2605.
28. McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, *5*(4), 115–133.
29. Mikolov, T., & Zweig, G. (2012). Context dependent recurrent neural network language model. *SLT*, *12*, 234–239.
30. Minsky, M., & Papert, S. (1969). *Perceptrons*.
31. Mirza, M., & Osindero, S. (2014). *Conditional generative adversarial nets*. arXiv:1411.1784.
32. Mitchell, T. M. (1997). *Machine learning* (1st ed.). New York, NY, USA: McGraw-Hill Inc.
33. Molina, L. C., Belanche, L., & Nebot, A. (2002). Feature selection algorithms: A survey and experimental evaluation. In *Proceedings of the 2002 IEEE International Conference on Data Mining* (pp. 306–313).
34. Nair, V., & Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning* (ICML 2010) (pp. 807–814).
35. Oja, E. (1997). The nonlinear PCA learning rule in independent component analysis. *Neurocomputing*, *17*(1), 25–45.
36. Pascanu, R., Mikolov, T., & Bengio, Y. (2013). On the difficulty of training recurrent neural networks. *ICML*, *3*(28), 1310–1318.
37. Pineda, F. J. (1987). Generalization of back-propagation to recurrent neural networks. *Physical Review Letters*, *59*(19), 2229.
38. Radford, A., Metz, L., & Chintala, S. (2015). *Unsupervised representation learning with deep convolutional generative adversarial networks*. arXiv:1511.06434.
39. Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, *65*(6), 386.
40. Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, *3*(3), 210–229.
41. Smola, A. J., Vishwanathan, S. V. N., & Hofmann, T. (2005). Kernel methods for missing variables. In R. G. Cowell, & Z. Ghahramani (eds.) *Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics* (pp. 325–332). Society for Artificial Intelligence and Statistics.

42. Weisberg, S. (2005). *Applied linear regression* (Vol. 528). Wiley
43. Willmott, C. J. (1982). Some comments on the evaluation of model performance. *Bulletin of the American Meteorological Society*, *63*(11), 1309–1313.
44. Zeiler, M. D., & Fergus, R. (2014) Visualizing and understanding convolutional networks. In *European Conference on Computer Vision* (pp. 818–833). Springer.
45. Zhang, S. (2011). Shell-neighbor method and its application in missing data imputation. *Applied Intelligence*, *35*(1), 123–133.

441329_1_En_3_Chapter ☑ TYPESET ☐ DISK ☐ LE ☑ CP Disp.:13/7/2018 Pages: 33 Layout: T1-Standard

Author Queries

Chapter 3

Query Refs.	Details Required	Author's response
	No queries.	

MARKED PROOF

Please correct and return this set

Please use the proof correction marks shown below for all alterations and corrections. If you wish to return your proof by fax you should ensure that all amendments are written clearly in dark ink and are made well within the page margins.

Instruction to printer	*Textual mark*	*Marginal mark*
Leave unchanged	··· under matter to remain	New matter followed by
Insert in text the matter indicated in the margin		or (x2)
Delete	through single character, rule or underline or through all characters to be deleted	or (x2)
Substitute character or substitute part of one or more word(s)	through letter or through characters	new character or new characters
Change to italics	under matter to be changed	
Change to capitals	under matter to be changed	
Change to small capitals	under matter to be changed	
Change to bold type	under matter to be changed	
Change to bold italic	under matter to be changed	
Change to lower case	Encircle matter to be changed	
Change italic to upright type	(As above)	
Change bold to non-bold type	(As above)	
Insert 'superior' character	through character or where required	or under character e.g. or
Insert 'inferior' character	(As above)	over character e.g.
Insert full stop	(As above)	
Insert comma	(As above)	
Insert single quotation marks	(As above)	or and/or or
Insert double quotation marks	(As above)	or and/or or
Insert hyphen	(As above)	
Start new paragraph		
No new paragraph		
Transpose		
Close up	linking characters	
Insert or substitute space between characters or words	through character or where required	
Reduce space between characters or words	between characters or words affected	