# Chapter 8
# Validation

## 8.1 Validating data mining techniques

This book presents details for some of the most frequently used data mining techniques in the field of agriculture. As pointed out in Chapter 1, data mining techniques can be mainly divided into clustering and classification techniques. Clustering techniques are used when there is not any previous knowledge about the data, and hence a partition in clusters grouping similar data is searched. When a training set is available, classification techniques can be applied. In such cases, the training set is exploited for classifying data of unknown classification. The training set can be exploited in two ways: it can be used directly for performing the classification, or it can be used for setting up the parameters of a model which fits the data.

Chapter 3 presents the most frequently used clustering algorithm, the $k$-means algorithm, and many of its variants. Samples in a set of data are partitioned into clusters; each cluster groups a subset of samples very similar to one another. The similarities between the samples are measured using a distance function. Each cluster contains the samples closest to the center of the cluster. An error function monitoring the distances between the samples and the centers is used to evaluate the quality of a given partition in clusters. Chapter 7 introduces the simultaneous partition of the samples and their features in biclusters. In this case, the quality of the biclusters is evaluated using error functions as well, where the variance in the elements of a bicluster is measured. These error functions depend on the kinds of biclusters that are searched.

The classification techniques discussed in this book are the $k$-nearest neighbor (Chapter 4), the artificial neural networks (Chapter 5), the support vector machines (Chapter 6), and the supervised biclustering (Chapter 7). All these techniques require the use of a training set. $k$-nearest neighbor exploits such a training set directly for classifying samples with unknown classification. An unclassified sample is compared to similar samples in the training set, and the classification is assigned in accordance with the ones such similar samples have. As before, the similarities between samples are measured through distance functions. Artificial neural networks consist of a set

A. Mucherino et al., *Data Mining in Agriculture*, Springer Optimization and Its Applications 34, 161
DOI: 10.1007/978-0-387-88615-2_8,

of neurons performing simple tasks and connected to each other in a structure that resembles the human brain. A neural network can be trained using the information available in a training set. During this phase, they are supposed to learn from the data and generalize from them. Once trained, a neural network should be able to classify unknown samples because of the information extracted during the learning phase from the known samples of the training set. Similarly, support vector machines learn from a training set how to classify unknown data. They are linear classifiers and can be extended to nonlinear cases. The basic assumption is that a classifier able to separate two distinct classes of samples with a larger margin is a better classifier. Finally, supervised biclustering uses a training set of samples for simultaneously classifying in biclusters the samples themselves and even their features. Therefore, not only the samples are categorized in classes, but even their features, so that the features responsible for the classification of a class can be identified.

In the clustering techniques discussed in this book, an error function is usually used for finding the best partition in clusters or biclusters of the data. Such error function gives an evaluation of the quality of a given partition: the lower is the error function value, the higher is the quality. This can be considered as an evaluation of the quality of the solution. However, even when the error function has a small value, the obtained partition may not be accurate. For instance, let us suppose that the $k$-means algorithm is used with different values for the $k$ parameter. For a given $k$, the error function values show the best partition among a set of partitions in $k$ clusters. Unfortunately, if $k$ changes, and $k_1$ and $k_2$ are for instance used, then the error function values cannot be used for comparing the partitions in $k_1$ clusters to the partitions in $k_2$ clusters. Therefore, sometimes validation techniques are needed when clustering methods are used. Reference [98] presents a survey of validation techniques applied to clustering methods. This survey takes into account even clustering methods that are not presented in this book.

The situation is different when dealing with classification techniques. In the $k$-nearest neighbor approach, an unknown sample is classified considering the classification of its neighbors in the training set. The accuracy of the classification depends on the value chosen for the parameter $k$, and some $k$ values may be good for some types of applications and not as good for other types of applications. Although the method provides a simple and often effective classification, unfortunately, the accuracy of the classification needs verification. In the neural network approach, an error function is defined for monitoring how the network fits the data during the learning phase. This error function evaluates the mean error occurring when the network is used for classifying the samples of a training set. Once the network is trained, and eventually also pruned, it can be used for classifying unknown samples. Even in this case, the network is not able to provide an estimation for the accuracy of the classification, and therefore the results need to be validated in a different way. In general, all the classification techniques using a training dataset are able to estimate the accuracy of their classification on the known data only.

Therefore, it is important to validate the classifiers used in the classification process. Validation techniques can be used for this purpose. Usually, the available training set is divided at least in two parts. The first part is actually used as training set

and the second part is used for validation purposes. The latter part is usually called *validation* or *testing* set. Both names, in general, can refer to the set of known samples used for evaluating the quality of the classifications. In some cases, however, validation and testing sets are actually two different objects. As an example, during the learning phase of a neural network, the parameters of the network are improved step by step and they converge toward the optimal values. Therefore, at each iteration, the parameters can be used for classifying samples different from the ones in the training set. This allows one to check if the parameters are converging to optimal values, or if there is overfitting, during the learning phase. The set of samples used in this case is usually referred to as the validation set. Once the network has been trained, then a set of known samples can be used to check the quality of the classifications obtained by the network. This last set of known samples is referred to as the testing set.

In the following sections, three validation techniques are presented and for each of them an example in MATLAB® is provided. For simplicity, regression models and the simple $k$-nearest neighbor rule are validated on a random set of samples in a two-dimensional space in MATLAB. For more details about these techniques, the reader may consider Andrew Moore's lecture that can be found on the Internet [167].

## 8.2 Test set method

The training set contains the information needed for performing the classification of unknown samples. It consists in a set of pairs grouping samples and their corresponding classifications. All the other samples which are not contained in the training set have an unknown classification, and hence they cannot be used for validation purposes.

The *test set* method is based on the following idea. Since only the samples in the training set have a known classification, the idea is to split the training set in two parts: a part which is actually used as a training set, and another part used for the validation. In general, 70% of the data can be used as a training set, and the remaining (30%) can be used for the validation process.

Let us suppose that the $k$-nearest neighbor rule is used for classifying a set of unknown samples. To validate the effectiveness of this rule, 30% of the training set is classified using the remaining 70% of the training set. Since samples in both cases are taken from the training set, their classification is known, and therefore the classification obtained by the $k$-nearest neighbor rule can be validated. Similarly, a neural network or a support vector machine can be trained using 70% of the training set, and then the accuracy of the classification provided by the trained network or support vector machine can be evaluated on the remaining 30% of the training set.

### *8.2.1 An example in MATLAB*

In this example, a linear regression model is validated by using the test set method. It is supposed that a set of points in a two-dimensional space is available, and that

it is needed to model these points by linear regression. These points can represent measurements of a certain process that it is known to be linear. The available set of points is used as a training set: the general rule governing the process needs to be discovered from this set. Once the regression model has been found, it should be able to approximate with an acceptable accuracy the points of the training set, and it should also be able to generalize to other unknown points.

In order to validate the quality of the regression model, the test set method can be applied. Following this method, the original training set has to be divided in two parts. Let us suppose the training set contains the following 10 points:

$$(1, 4), \quad (2, 2), \quad (3, 3), \quad (4, 1.7), \quad (5, 1)$$
$$(6, 1.2), \quad (7, 1.5), \quad (8, 1.9), \quad (9, 2.3), \quad (10, 2.7).$$

Three of these points (30%) can be used for validating the model, while the other seven points (70%) are used as a training set for finding the model. One issue can be how to decide the points to place in the validation set and the ones to place in the actual training set. This separation can be done in a totally random way, but there might be cases in which this can lead to problems. Let us consider, for instance, that the three points having the smallest $x$ value are used as a validation set, whereas the others are used as a training set. The following MATLAB code has been used for performing the validation and generating Figure 8.1.

```
x = [1 2 3 4 5 6 7 8 9 10];
y = [4 2 3 1.7 1 1.2 1.5 1.9 2.3 2.7];
x1 = x(4:10);
y1 = y(4:10);
x2 = x(1:3);
y2 = y(1:3);
plot(x1,y1,'ks','MarkerSize',16,'MarkerEdgeColor','k','MarkerFaceColor',
     [.49 1 .63])
hold on
plot(x2,y2,'ko','MarkerSize',16,'MarkerEdgeColor','k','MarkerFaceColor',
     [.87 1 .23])
c = polyfit(x1,y1,1);
xx = 0:0.1:12;
yy = polyval(c,xx);
plot(xx,yy,'k')
err = abs(y(1) - polyval(c,x(1)))
```

The `x` vector is initialized with the $x$ coordinates of the whole set of points, and the `y` vector is initialized with their $y$ coordinates. In the vectors `x1` and `y1` are then placed the points that are actually used for computing the regression model. In `x2` and `y2` are instead placed the remaining points, the ones that are used for the validation. In this example, the compact symbologies `1:3` and `4:10` are used for considering vectors whose first component is 1 (or 4), whose last component is 3 (or 10) and having distance between any consecutive components equal to 1 (for details see Appendix A). These points separated in this way are then printed by using the function `plot`. Note that many options are used for controlling the symbols used when the points are drawn. In particular, the points in the validation set are marked by circles, and the points in the training set are marked by squares.

The function `polyfit` is able to find the coefficient of the linear function that better approximates the points in `x1` and `y1` (see Section 2.4). The specified degree for the

polynomial is 1, because a linear model is searched. The output of the function `polyfit` is placed in the vector `c`, which is soon used as input in the function `polyval` that evaluates the polynomial in the $x$ coordinates stored in `xx`. The vector `xx` is created so that it contains all the $x$ coordinates in `x`. The vector `yy` generated by `polyval` contains the corresponding $y$ coordinates. The vectors `xx` and `yy` are finally given as input to the function `plot`.

Figure 8.1 shows that the found linear regression does not give a good approximation of the points placed in the validation set. For instance, the error `err` computed on the point (`x(1)`,`y(1)`) has value 3.59. This is not a small error, if it is compared to the coordinates of the points. This error is larger than 3 times the difference between two consecutive $x$ components. If the points (`x1`,`y1`) and (`x2`,`y2`) are instead chosen in the following way

```
x1 = x([1 2 4 5 7 8 10]);
y1 = y([1 2 4 5 7 8 10]);
x2 = x([3 6 9]);
y2 = y([3 6 9]);
```

then the accuracy grows. Figure 8.2 shows the new-found linear regression. In this case, the whole set of points is represented better by the chosen 70% of the original training set. This brings to a reduction of the overall error on the points in the validation set. The largest error is here due to the points (`x6`,`y6`) and it corresponds to 0.85. In general, more than one random division of the training set could be considered and the test set method applied for each of these divisions.

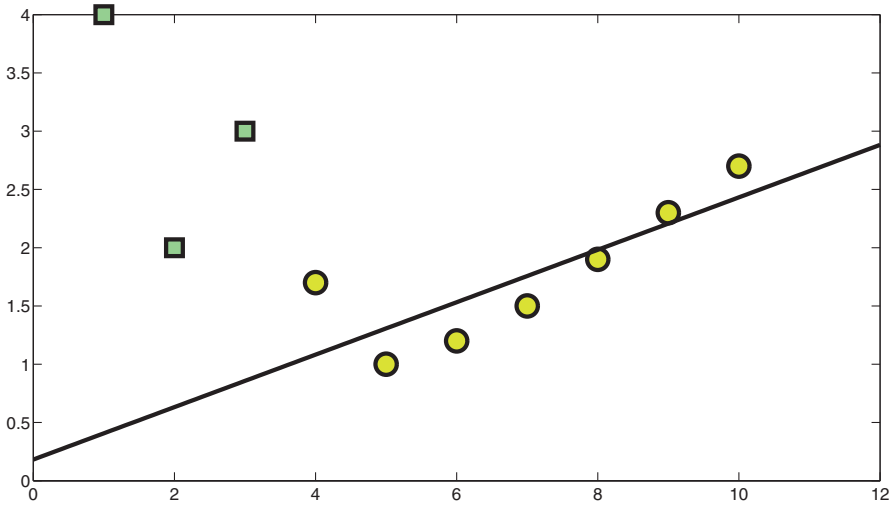


**Fig. 8.1** The test set method for validating a linear regression model.

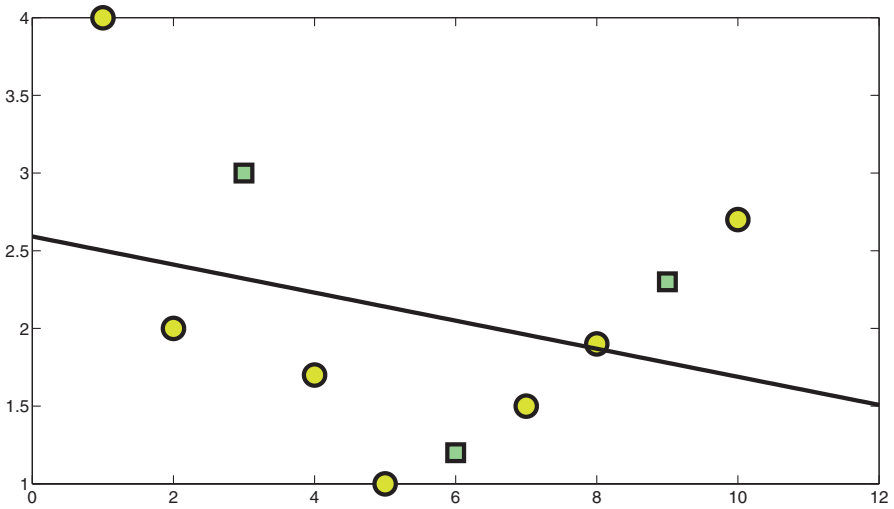


**Fig. 8.2** The test set method for validating a linear regression model. In this case, a validation set different from the one in Figure 8.1 is used.

## 8.3 Leave-one-out method

There are two disadvantages in using the test set method for validation. First, a consistent part of the training set is actually not used as a training set, but it is used as a validation set. Second, the validation set is generally randomly extracted from the original training set, and it may not be a good representative of the whole set. Therefore, the original training set has to be reduced for applying this method, and it may be a problem if there is not much data available. Furthermore, the validation set may not provide an accurate validation. For instance, if only one sample of a certain class is contained in the validation set, which the accuracy of classifications in this class is evaluated only on such a sample, and this is statistically irrelevant.

The *leave-one-out* method overcomes these problems. As the name suggests, the validation is performed by leaving only one sample out of the training set: all the samples except the one left out are used as a training set, and the classification method is validated on the sample left out. If this procedure is performed only once, then the result would be statistically irrelevant as well. The procedure is indeed performed as many times as the number of samples in the training set, that one by one are taken out of the training set. The overall accuracy of the classifications of the samples left out gives an evaluation of the classification method.

### *8.3.1 An example in MATLAB*

A quadratic regression model is validated in the example discussed in this section. Let us suppose that the same set of points used in the example in Section 8.2.1

is available, but this time it is known that the model fitting these points has to be quadratic. In practice, the parabola that better fits the points is searched. As before, the available set of points can be used as a training set for finding the quadratic regression model which is able to approximate the points in the training set and even unknown points.

The leave-one-out method is used for evaluating the quality of several quadratic models that can be generated from the set of points. In particular, each model is created by using the whole training set except only one point, which is later used for the validation. The following MATLAB code can be used for building one of these quadratic models leaving out the point (`x1`,`y1`):

```
x = [1 2 3 4 5 6 7 8 9 10];
y = [4 2 3 1.7 1 1.2 1.5 1.9 2.3 2.7];
x1 = x;
y1 = y;
x1(1) = [];
y1(1) = [];
x2 = x(1);
y2 = y(1);
plot(x1,y1,'ks','MarkerSize',16,'MarkerEdgeColor','k','MarkerFaceColor',
     [.49 1 .63])
hold on
plot(x,y,'ko','MarkerSize',16,'MarkerEdgeColor','k','MarkerFaceColor',
     [.87 1 .23])
c = polyfit(x1,y1,2);
xx = 0:0.1:12;
yy = polyval(c,xx);
plot(xx,yy,'k')
err = abs(y(1) - polyval(c,x(1)))
```

It is supposed that the original training set is the same used in Section 8.2.1, containing points whose $x$ and $y$ coordinates are stored in `x` and `y`, respectively. In `x1` and `y1` are specified the points that are part of the training set. In the code, `x1` and `y1` are initially set equal to `x` and `y`, and then the first component of both of them is deleted. The instruction `x1(1) = []` actually removes the component `1` of `x1`, since it assigns to `x1(1)` an empty matrix `[]`. The validation set contains in this case only one point, whose $x$ and $y$ coordinates are stored in `x2` and `y2`. Figure 8.3(a) is generated by the two calls to the function `plot`, separated by the instruction `hold on`.

The MATLAB function `polyfit` is used for creating the quadratic regression model. It receives as inputs the actual training set through the vectors `x1` and `y1`, and the degree of the approximating polynomial, which is 2 in this example. The provided output consists of the obtained polynomial coefficients, stored in the vector `c`. In order to draw the polynomial, a vector `xx` is defined and the function `polyval` is called, similarly as in the example showed in Section 8.2.1. Another call of the function `plot` finally draws the quadratic regression in Figure 8.3(a).

The error occurring when the point left out, (`x(1),y(1)`) in this case, is compared to the corresponding point (`x(1),polyval(c,x(1))`) is 0.75. Following the leave-one-out method, the same procedure has to be repeated leaving out all the points of the training set, one by one. Figure 8.3(b) shows the quadratic regression obtained leaving out the point (`x(4),y(4)`). In Figure 8.4(a) the point (`x(7),y(7)`) is left out, and in Figure 8.4(b) the point (`x(10),y(10)`) is left out. The obtained errors are 0.01 when (`x(4),y(4)`) is left out, 0.11 when (`x(7),y(7)`) is left out, and 0.44

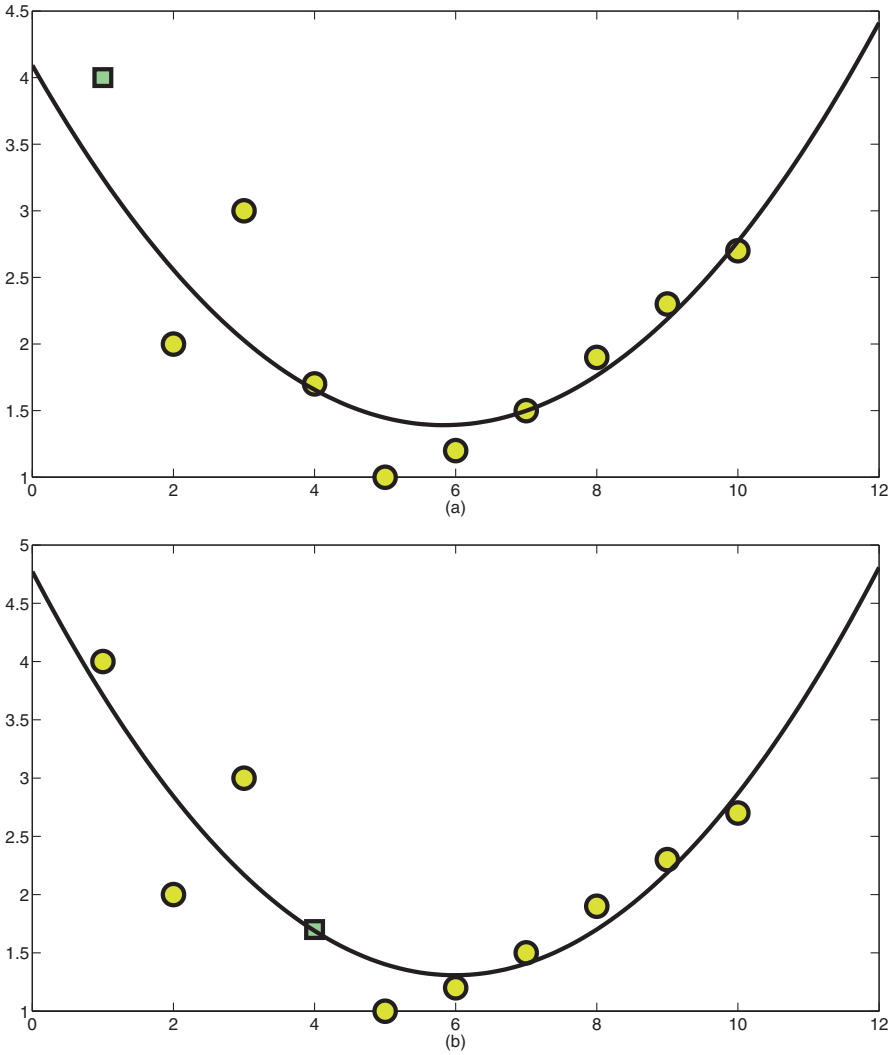


**Fig. 8.3** The leave-one-out method for validation. (a) The point (`x(1)`,`y(1)`) is left out; (b) the point (`x(4)`,`y(4)`) is left out.

when (`x(10)`,`y(10)`) is left out. The errors on the other points of the training set, when left out, have similar values. Therefore, in general, this regression model can be considered sufficiently accurate, since such errors are quite small.

## 8.4 *k*-fold method

As previously observed, the test set method may not be very efficient as a validation method because the validation set takes data from the training set and because

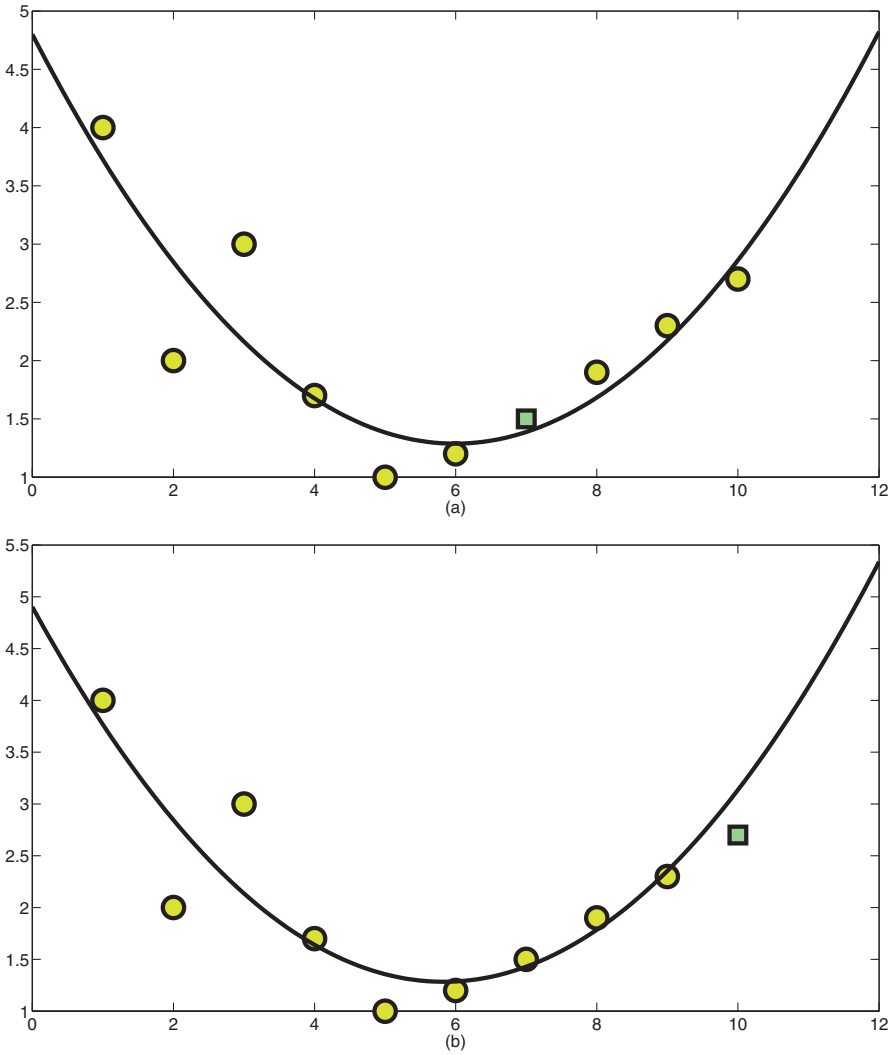


**Fig. 8.4** The leave-one-out method for validation. (a) The point (`x(7),y(7)`) is left out; (b) the point (`x(10),y(10)`) is left out.

these data may not be a good representative of the original set. These problems are overcome if the leave-one-out method is instead used. In this case, indeed, only one sample is taken out of the training set at a time, and hence the amount of data actually used as a training set is not reduced. Moreover, all the samples, one by one, are also used for testing the accuracy of the classification, overcoming the problem of using a validation set that may not be a good representative of the whole set of data. The leave-one-out method seems to be the optimal choice, but it actually introduces another issue. This issue is related to the computational cost of the validation method.

If the training set contains $n$ samples, then, following the leave-one-out method, the used classification method needs to be trained and applied $n$ times. If $n$ is large enough, this can be computationally demanding.

The optimal choice between the speed of the test set method and the reliability of the leave-one-out method is the *k-fold* method. In this method, the samples are partitioned in $k$ groups. Then, for each of these $k$ groups, the classification method is performed using as a training set the original set without the samples contained in one of these groups. After that, the group left out from the training set is used as a validation set. Note that if $k = n$, then the $k$-fold method corresponds to the leave-one-out method. If $k = 4$, then one iteration of the $k$-fold method, in which about 25% of the training set is devoted to the validation, is similar to the test set method. Therefore, the choice of a value for the parameter $k$ is very important as it provides the trade-off between accuracy and computational speed.

### *8.4.1 An example in MATLAB*

In this example, an application of the $k$-nearest neighbor method is validated by using the $k$-fold method. The example is carried out in the MATLAB environment and the code used for performing it is the following one:

```
[x,y] = generate(100,0.2);
[class] = hmeans(100,x,y,2);
plotp(100,x,y,class);
xA = x(1:50);
yA = y(1:50);
xB = x(51:100);
yB = y(51:100);
classA = class(1:50);
classB = class(51:100);
[class] = knn(50,xA,yA,2,50,xB,yB,classB);
plotp(50,xA,yA,class)
[class] = knn(50,xB,yB,2,50,xA,yA,classA);
plotp(50,xB,yB,class)
```

The MATLAB functions used in this example have been discussed in the previous chapters and their source codes are available in the book. The function `generate` is used for creating a random set of points in a two-dimensional space. One hundred points are generated, and they are randomly separated in two subgroups having a margin equal to 0.2 (see Section 3.6 and Figure 3.16). The chosen margin is quite wide, so that a clustering method is able to discover easily this pattern in the data. In particular, the function `hmeans` is used for partitioning the points in two parts. The partition is stored through the vector `class`, whose components can have value 1 or 2 (Section 3.6, Figure 3.20). This set of points and its partition are used as a training set for the application of the $k$-nearest neighbor. The call to the function `plotp` generates Figure 8.5.

The $k$-fold method is used for validating the application of the $k$-nearest neighbor method in which the training set is the one generated above. For simplicity, the parameter $k$ in $k$-fold is set to 2, and therefore the training set is divided in 2 parts only. The division in 2 parts can be performed randomly, or the strategy used here

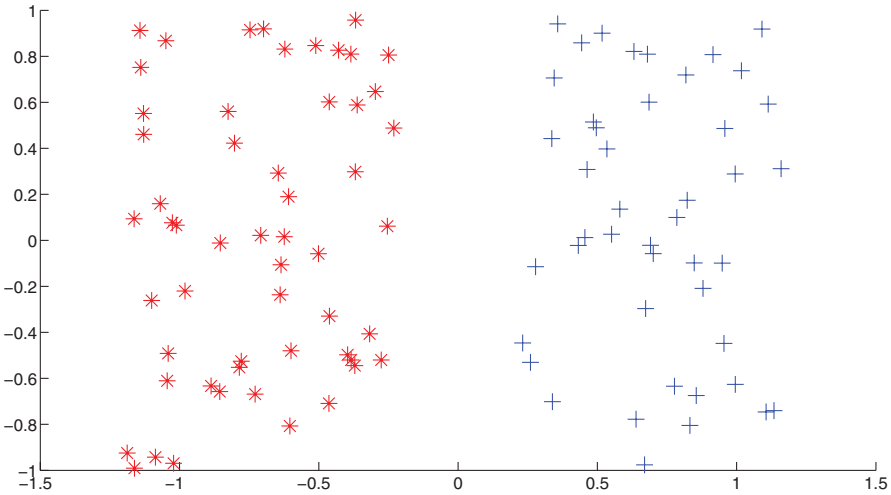


**Fig. 8.5** A set of points partitioned in two classes.

can be implemented. `xA` and `yA` are defined so that they contain the first 50 points stored in `x` and `y`; `xB` and `yB`, instead, are defined so that they contain the last 50 points stored in `x` and `y`. The vectors `classA` and `classB` are defined similarly. The $k$-nearest neighbor method must be applied twice. The training set is specified by `xB`, `yB` and `classB` and the points stored in `xA` and `yA` are classified. Successively, the training set is specified by `xA`, `yA` and `classA` and the points stored in `xB` and `yB` are instead classified. The function `plotp` is used for plotting the points in `xA` and `yA`, where the vector `class` is the one just obtained by the function `kNN`. The obtained result is shown in Figure 8.6(a). Successively, the function `plotp` is used again for printing the points `xB` and `yB` marked in accordance with the classification given by the $k$-nearest neighbor. This other plot is shown in Figure 8.6(b). If Figure 8.5 and Figures 8.6(a) and 8.6(b) are compared, it is easy to see that the points are correctly classified by the $k$-nearest neighbor in both the cases. This classification method on this simple example is therefore validated.

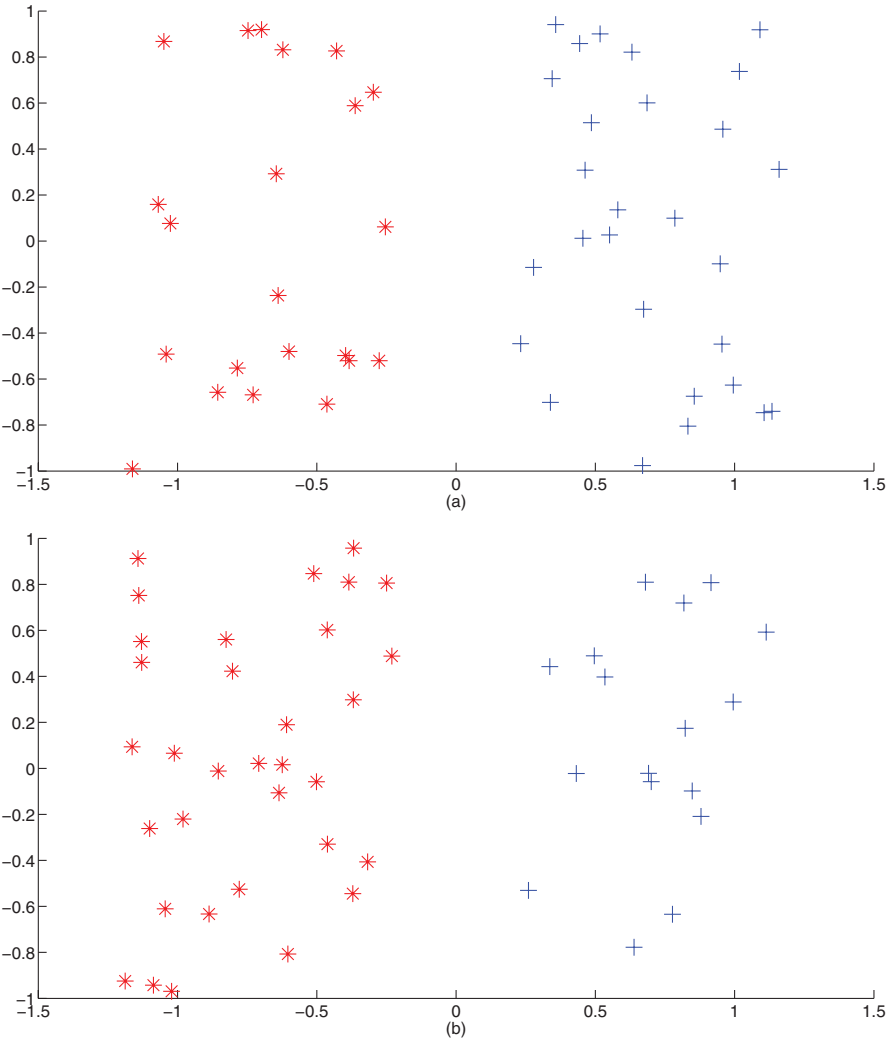


**Fig. 8.6** The results obtained applying the $k$-fold method. (a) Half set is considered as a training set and the other half as a validation set; (b) training and validation sets are inverted.