Chapter 4
Learning Multivariate Correlations in Data

4.1 General: Decision Rules, Fitting Criteria, and Learning Protocols

To specify a problem of learning correlation in a data table, one has to distinguish between two parts in the feature set: *predictor*, or *input*, features and *target*, or *output*, features. Typically, the number of target features is small, and in generic tasks, there is just one target feature. Target features are usually difficult to measure or impossible to know beforehand. This is why one would want to derive a decision rule relating predictors and targets so that prediction of targets can be made after measuring predictors only. Examples of learning problems include:

(a) chemical compounds: input features are of the molecular structure, whereas target features are activities such as toxicity or healing effects;
(b) types of grain in agriculture: input features are those of the seeds, ground and weather, and target features are of productivity and protein contents,
(c) industrial enterprises: input features refer to technology, investment and labor policies, whereas target features are of sales and profits;
(d) postcode districts in marketing research: input features refer to demographic, social and economic characteristics of the district residents, target features – to their purchasing behavior;
(e) bank loan customers: input features characterize demographic and income, whereas output features are of (potentially) bad debt;
(f) gene expression data: input features relate to levels of expression of DNA materials in the earlier stages of an illness, and output features to those at later stages.

A *decision rule* predicts values of target features from values of input features. A rule is referred to as a classifier if the target is categorical and as a regression if the target is quantitative. A generic categorical target problem is defined by specifying just a subset of entities labeled as belonging to the class of interest – the correlation problem in this case would be of building such a decision rule that would recognize, for each of the entities, whether it belongs to the labeled class or not.

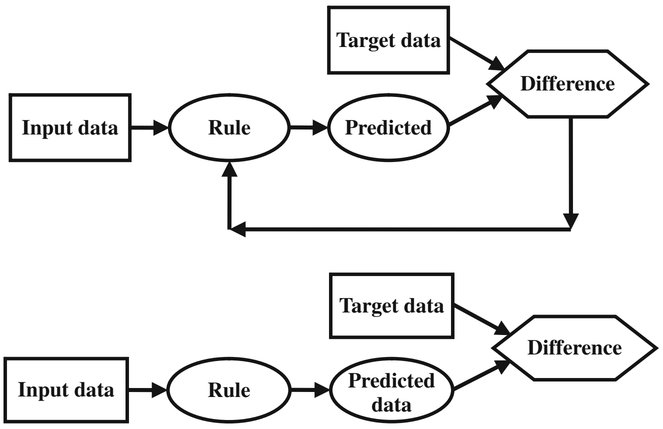

Fig. 4.1 Structure of a training/testing problem: In training, *on the top*, the decision rule is fitted to minimize the difference between the predicted and observed target data. In testing, *the bottom part*, the rule is used to calculate the difference so that no feedback to the rule is utilized

A generic regression problem – the bivariate linear regression – has been considered in Section 3.2; its extension to the multivariate case will be described later in Section 4.3.

A decision rule is learnt over a dataset in which values of the targets are available. These data are frequently referred to as the training data. The idea underlying the process of learning is to look at the difference between predicted and observed target feature values on the training data set and to minimize them over a class of admissible rules. The structure of such a process is presented on the upper part of Fig. 4.1.

The notion that it ought to be a class of admissible rules pre-specified emerges because the training data is finite and, therefore, can be fit exactly by using a sufficient number of parameters. However, this would be valid on the training set only, because the fit would capture all the errors and noise inevitable in data collecting processes. Take a look, for example, at the 2D regression problem on Fig. 4.2 depicting seven points on *(x,u)*-plane corresponding to observed combinations of input feature *x* and target feature *u*.

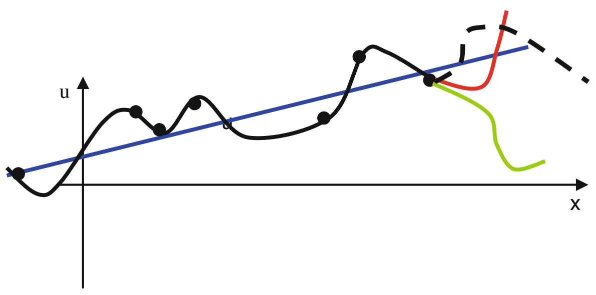

Fig. 4.2 Possible graphs of interrelation between *x* and *u* according to observed data points (*black circles*)

The seven points on Fig. 4.2 can be exactly fitted by a polynomial of 6th order $u = p(x) = a_0 + a_1x + a_2x^2 + a_3x^3 + a_4x^4 + a_5x^6$. Indeed, they would lead to 7 equations $u_i = p(x_i)$ $(i = 1, \ldots, 7)$, so that, in a typical case, the 7 coefficients a_k of the polynomial can be exactly determined. Having N points observed would require an N-th degree polynomial to exactly fit them.

However, the polynomial, on which graph all observations lie, has no predictive power both within and beyond the range. The curve may go either course (like those shown) depending on small changes in the data. The power of a theory – and a regression line is a theory in this case – rests on its generalization power, which, in this case, can be cast down as the relation between the number of observations and the number of parameters: the greater the better. When this ratio is relatively small, statisticians would refer to this as an *over-fitted* rule. The overfitting normally produce very poor predictions on newly added observations. The straight line of Fig. 4.2 fits none of the points, but it expresses a simple and very robust tendency and should be preferred because it summarizes the data much deeper: the seven observations are summarized here in just two parameters, slope and intercept, whereas the polynomial line provides no summary: it involves as many parameters as the data entities. This is why, in learning decision rule problems, a class of admissible rules should be selected first. Unfortunately, as of this moment, there is no model based advice, within the data analysis discipline, on how this can be done, except very general ones like "look at the shapes of scatter plots". If there is no domain knowledge to choose a class of decision rules to fit, it is hard to tell what class of decision rules to use.

A most popular advice relates to the so-called *Occam's razor*, which means that the complexity of the data should be balanced by the complexity of the decision rule. A British monk philosopher William Ockham (*c*. 1285–1349) said: "Entities should not be multiplied unnecessarily." This is usually interpreted as saying that all other things being equal, the simplest explanation tends to be the best one. Operationally, this is further translated as the Principle of Maximum Parsimony, which is referred to when there is nothing better available. In the format of the so-called "Minimum description length" principle, this approach can be meaningfully applied to problems of estimation of parameters of statistic distributions (see P.D. Grünwald 2007). Somewhat wider, and perhaps more appropriate, explication of the Occam's razor is proposed by Vapnik (2006). In a slightly modified form, to avoid mixing different terminologies, it can be put as follows: "Find an admissible decision rule with the smallest number of free parameters such that explains the observed facts" (Vapnik 2006, p. 448). However, even in this format, the principle gives no guidance about how to choose an adequate functional form. For example, which of two functions, the power function $f(x) = ax^b$ or logarithmic one, $g(x) = b\log(x) + a$ both having just two parameters a and b, should be preferred as a summarization tool for graphs on Fig. 4.3?

Another set of advices, not incompatible with those above, relates to the so-called falsifability principle by K. Popper (1902–1994), which can be expressed as follows: "Explain the facts by using such an admissible decision rule which is easiest to falsify" (Vapnik 2006, p. 451). In principle, to falsify a theory one needs to give

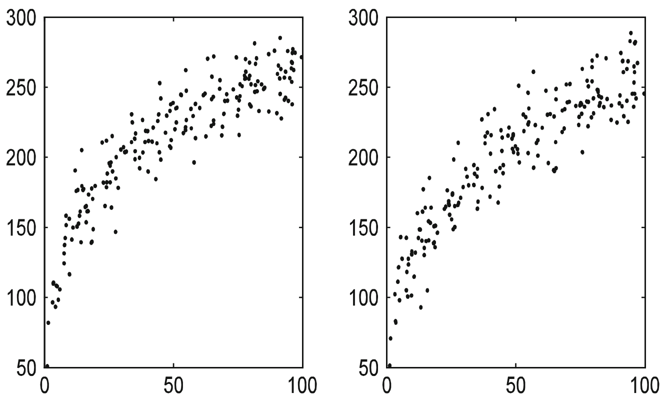

Fig. 4.3 Graph of one of two functions, $f(x) = 65x^{0.3}$ and $g(x) = 50\log(x) + 30$, both with an added normal noise *N(0,15)*, is presented on each plot. Can the reader give an educated guess of which is which? (Answer: *f(x)* is *on the right* and *g(x)on the left*.)

an example contradicting to it. Falsifability of a decision rule can be formulated in terms of the so-called *VC-complexity*, a measure of complexity of classes of decision rules: the smaller VC-complexity the greater the falsifability.

Let us explain the concept of VC-complexity for the case of a categorical target, so that a decision rule to be would be a classifier. However many categorical target features are specified, different combinations of target categories can be assigned different labels, so that a classifier is bound to predict a label. A set of classifiers Φ is said to shatter the training sample if for any possible assignment of the labels, a classifier exactly reproducing the labels can be found in Φ. Given a set of admissible classifiers Φ, the VC-complexity of a classifying problem is the maximum number of entities that can be shattered by classifiers from Φ. For example, 2D points have VC complexity 3 in the class of linear decision rules. Indeed, any three points, not lying on a line, can be shattered by a line; yet not all four-point sets can be shattered by lines, as shown on Fig. 4.4, the left and right parts, respectively.

The VC complexity is an important characteristic of a correlation problem especially within the probabilistic machine learning paradigm. Under conventional conditions of the independent random sampling of the data, a reliable classifier "with probability *a*% will be *b*% accurate, where *b* depends not only on *a*, but also on the sample size and VC-complexity" (Vapnik 2006).

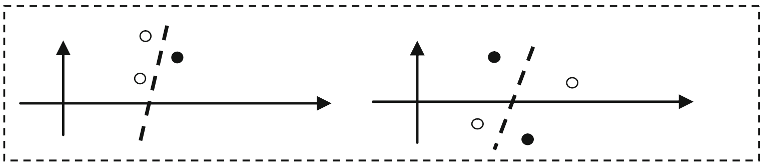

Fig. 4.4 Any two-part split of three points (not on one line) can be made by a linear function, but the presented case on four points cannot be solved by a *line*

The problem of learning correlation in a data table can be stated, in general terms, as follows. Given N pairs (x_i, u_i), $i = 1, \ldots, N$, in which x_i are predictor/input p-dimensional vectors $x_i = (x_{il}, \ldots, x_{ip})$ and $u_i = (u_{il}, \ldots, u_{iq})$ are target/output q-dimensional vectors (usually $q = 1$), build a decision rule

$$\hat{u} = F(x) \tag{4.1}$$

such that the difference between computed $\hat{u}$ and observed u is minimal over a pre-specified class Φ of admissible rules F.

To specify a correlation learning problem one should specify assumptions regarding a number of constituents including:

(i) Type of target
Two types of target features are considered usually: quantitative and categorical. In the former case, Equation (4.1) is usually referred to as regression; in the latter case, decision rule, and the learning problem is referred to as that of "classification" or "pattern recognition".

(ii) Type of rule
A rule involves a postulated mathematical structure whose parameters are to be learnt from the data. The mathematical structures considered further on are:

- *linear* combination of features
- *neural network* mapping a set of input features into a set of target features
- *decision tree* built over a set of features
- *partition* of the entity set into a number of non-overlapping clusters

(iii) Criterion
Criterion of the quality of fitting depends on the framework in which the learning task is formulated. Most popular criteria are: maximum likelihood (in a probabilistic model of data generation), least-squares (data recovery approach) and relative errors. According to the least-squares criterion, the difference between u and $\hat{u}$ is measured with the average squared error,

$$E = < u - \hat{u},\ u - \hat{u} > /N = < u - F(x),\ u - F(x) > /N \tag{4.2}$$

which is to be minimized over all admissible F.

(iv) Training protocol
The rule F is learnt from a training dataset. The way the data becomes available can be referred to as the training protocol. Three popular training protocols are: batch, random and on-line. The batch mode is the case when all training set is available and used at once, the other two refer to cases when data entities come one by one so that the learning goes incrementally. In the random protocol, the data are available at once, yet the learning process is organized incrementally by picking up entities randomly one-by-one, possibly many times each. In contrast, in an on-line protocol each entity comes from an external source and can be seen only once.

4.2 Naïve Bayes Approach

4.2.1 Bayes Decision Rule

Consider a situation in which there is only one target, a binary feature labeling two states of the world corresponding to "positive" and "negative" classes of entities. According to Bayes (1702–1761), all relevant knowledge of the world should be shaped by the decision maker in the form of probability distributions. Then, whatever new data may be observed, they may lead to changing the probabilities – hence the difference between prior probabilities and posterior, data-updated, probabilities. Specifically, assume that, $P(1) = p_1$ and $P(2) = p_2$ are prior probabilities of the two states so that p_1 and p_2 are positive and sum to unity. Assume furthermore that there are two probability density functions, $f_1(x_1,\ x_2,\ \ldots, x_p)$ and $f_2(x_1,\ x_2,\ \ldots,\ x_p)$, defining the generation of observed entity points $x = (x_1, x_2, \ldots, .x_p)$ for each of the classes. That gives us, for any point $x = (x_1, x_2, \ldots, x_p)$ to occur, two probabilities, $P(x/1) = p_1 f_1(x)$ and $P(x/2) = p_2 f_2(x)$, of x being generated from either class. If an $x = (x_1, x_2, \ldots, x_p)$ is actually observed, it leads to a change in probabilities of the classes, from the prior probabilities $P(1) = p_1$ and $P(2) = p_2$ to posterior probabilities $P(1/x)$ and $P(2/x)$, respectively. These can be computed according to the well-known Bayes theorem from the elementary probability theory, so that the posterior probabilities of the classes are

$$P(1|x) = p_1 f_1(x)/f(x) \text{ and } P(2|x) = p_2 f_2(x)/f(x) \tag{4.3}$$

where $f(x) = p_1 f_1(x) + p_2 f_2(x)$.

The decision of which class the entity x belongs to depends on what value, $P(1/x)$ or $P(2/x)$ is greater. The class is assumed to be positive if $P(1/x) > P(2/x)$ or, equivalently,

$$f_1(x)/f_2(x) > p_2/p_1 \tag{4.4}$$

or, negative, if the reverse inequality holds. This rule is referred to as Bayes decision rule. Another expression of the Bayes rule can be drawn by using the difference $B(x) = P(1/x) - P(2/x)$: x is taken to belong to the positive class if $B(x) > 0$, and the negative class if $B(x) < 0$. Equation $B(x) = 0$ defines the so-called separating surface between the two classes.

The proportion of errors admitted by Bayes rule is $1 - P(1/x)$ when 1 is predicted and $1 - P(2/x)$ when 2 is predicted. These are the minimum error rates achievable when both within-class distributions $f_1(x)$ and $f_2(x)$ and priors p_1 and p_2 are known.

Unfortunately, the distributions $f_1(x)$ and $f_2(x)$ are typically not known. Then some simplifying assumptions are to be made so that the distributions could be estimated from the observed data. Among most popular assumptions are: (i) Gaussian probability and (ii) Local independence. Let us consider them in turn:

(i) Gaussian probability

The class probability distributions $f_1(x)$ and $f_2(x)$ are assumed to be Gaussian, so that each can be expressed as

$$f_k(x) = \exp[-(x - \mu_k)^T \Sigma_k{}^{-1}(x - \mu_k)/2]/[(2\pi)^p|\Sigma_k|]^{1/2}$$

where μ_k is the central point, Σ_k the $p\times p$ covariance matrix and $|\Sigma_k|$ its determinant ($k = 1, 2$).

The Gaussian distribution is very popular. There are at least two reasons for that. First, it is treatable theoretically and, in fact, may lead to the least squares criterion within the probabilistic approach. Second, some real-world stochastic processes, especially in physics, can be thought of as having the Gaussian distribution. Typical shapes of a 2D Gaussian density function are illustrated on Fig. 4.5: that with zero correlation on the left and 0.8 correlation on the right.

In the case at which the within-class covariance matrices are equal to each other, the Bayes decision function *B(x)* is linear so that the separating surface $B(x) = 0$ is a hyperplane as explained later in Section 4.4.

(ii) Local independence (Naïve Bayes)

The assumption of local independence states that all variables are independent within each class so that the within-cluster distribution is a product of one-dimensional distributions:

$$f_k(x_1, x_2, \ldots, x_p) = f_{kl}(x_1)f_{k2}(x_2)\ldots f_{kp}(x_p) \tag{4.5}$$

This postulate much simplifies the matters because usually it is not difficult to produce rather reliable estimates of the one-dimensional density functions $f_{kv}(x_v)$ from the training data. Especially simple such a task is when features $x_1,\ x_2,\ \ldots,\ x_p$

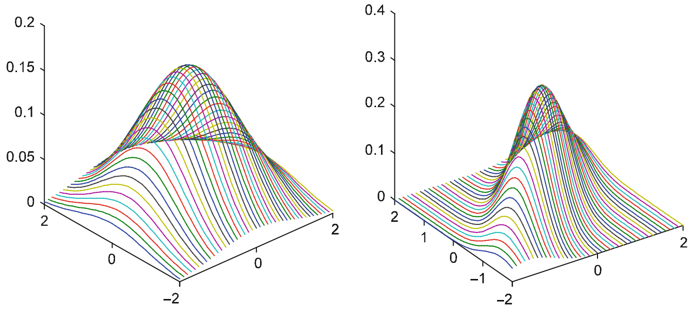

Fig. 4.5 Gaussian bivariate density functions over the origin as the expectation point – with zero correlation *on the left* and 0.8 correlation *on the right*

are binary themselves. In this case Bayes rule is referred to as naïve Bayes rule because in most cases the assumption of independence (4.5) is obviously wrong. Take, for example, the cases of text categorization or genomic analyses – constituents of a text or a protein serving as the features are necessarily interrelated according to the syntactic and semantic structures, in the former, and biochemical reactions, in the latter. Yet naive decision rules based on the wrong assumptions and distributions appear surprisingly good (see discussion in Manning et al. 2008).

Combining the assumptions of local independence and Gaussian distributions in the case of binary variables, one can arrive at equations expressing the conditional probabilities through exponents of linear functions of the variables (as described in Mitchell 2010) so that:

$$P(1/x) = \frac{1}{1+\exp(c_0 + c_1x_1 + ... + c_px_p)},$$
$$P(2/x) = \frac{\exp(c_0 + c_1x_1 + ... + c_px_p)}{1+\exp(c_0 + c_1x_1 + ... + c_px_p)},$$

These equations express what is referred to as logistic regression. Logistic regression is a popular decision rule that can be applied to any data on its own right as a model for the conditional probability, and not necessarily derived from the restrictive independence and normality assumptions.

4.2.2 *Naïve Bayes Classifier*

Consider a learning problem related to data in Table 4.1: there is a set of entities, which are newspaper articles, divided into a number of categories – there

Table 4.1 An illustrative database of 12 newspaper articles along with 10 keywords. The articles are labeled according to their main subjects – F for feminism, E for entertainment, and H for household

	Keyword									
Article	Drink	Equal	Fuel	Play	Popular	Price	Relief	Talent	Tax	Woman
F1	1	2	0	1	2	0	0	0	0	2
F2	0	0	0	1	0	1	0	2	0	2
F3	0	2	0	0	0	0	0	1	0	2
F4	2	1	0	0	0	2	0	2	0	1
E1	2	0	1	2	2	0	0	1	0	0
E2	0	1	0	3	2	1	2	0	0	0
E3	1	0	2	0	1	1	0	3	1	1
E4	0	1	0	1	1	0	1	1	0	0
H1	0	0	2	0	1	2	0	0	2	0
H2	1	0	2	2	0	2	2	0	0	0
H3	0	0	1	1	2	1	1	0	2	0
H4	0	0	1	0	0	2	2	0	2	0

are three categories in Table 4.1 according to the three subjects: Feminism (F), Entertainment (E) and Household (H). Each article is characterized by its set of keywords presented in the corresponding line. The entries are either 0 – no occurrence of the keyword, or 1 – one occurrence, or 2 – two or more occurrences of the keyword.

The problem is to form a rule according to which any article, including those outside of the collection in Table 4.1, can be assigned to one of these categories using its profile – the data on occurrences of the keywords in the corresponding line of Table 4.1.

Consider the Naïve Bayes decision rule. It assigns each category k with its conditional probability $P(k/x)$ depending on the profile x of an article in question which is similar to Equations in (4.3):

$$P(k/x) = p_k f_k(x) \big/ f(x) \tag{4.6}$$

where $f(x) = \sum_l p_l f_l(x)$. According to the Bayes rule, the category k, at which $P(k/x)$ is maximum, is selected. Obviously, the denominator does not depend on k and can be removed: that category k is selected, at which $p_k f_k(x)$ is maximum.

According to the Naïve Bayes approach, $f_k(x)$ is assumed to be the product of the probabilities of occurrences of the keywords in category k. How one can estimate such a probability? This is not that simple as it sounds.

For example, what is the probability of term "drink" in H category? Probably, it can be taken as 1/4 – since the term is present in only one of four members of H. But what's about term "play" in H – it occurs thrice but in two documents only; thus its probability cannot be taken ¾; yet 2/4 does not seem right either. A popular convention accepts the "bag-of-words" model for the categories. According to this model, all occurrences of all terms in a category are summed up, to produce 31 for category H in Table 4.1. Then each term's probability in category k would be its summary occurrence in k divided by the bag's total. This would lead to a fairly small probability of the "drink" in H, just 1/31. This bias is not that important, however, because what matters indeed in the Naïve Bayes rule is the feature relative contributions, not the absolute ones. And the relative contributions are all right with "drink", "fuel" and "play" contributing 1/31, 6/31 and 3/31, respectively, to H. Moreover, taking the total account of all keyword occurrences in a category serves well for balancing the differences between categories according to their sizes.

Yet there is one more issue to take care of: zero entries in the training data. Term "equal" does not appear at all in H leading thus to its zero probability in the category. This means that any article with an occurrence of "equal" cannot be classed into H category, however heavy evidence from other keywords may be. One would make a point of course that term "equal" has not been observed in H just because the sample of four articles in Table 4.1 is too small, which is a strong argument indeed. To make up for these, another, a "uniform prior" assumption is widely accepted. According to this assumption each term is present once at any category before the count is started. For the case of Table 4.1, this adds 1 to each numerator and 10 to each denominator, which means that the probability of "drink", "equal", "fuel" and "play" in category H will be (1+1)/(31+10) = 2/41, (0+1)/(31+10) = 1/41, (6+1)/(31+10) = 7/41 and (3+1)/(31+10) = 4/41, respectively.

To summarize, the "bag-of-words" model represents a category as a bag containing all occurrences of all keywords in the documents of the category plus one occurrence of each keyword, to be added to every count in the data table.

Table 4.2 contains the prior probabilities of categories, that are taken to be just proportions of categories in the collection, 4 of each in the collection of 12, as well as within-category probabilities of terms (the presence of binary features) computed as described above. Logarithms of these are given too.

Now we can apply Naïve Bayes classifier to any entity presented in the format of Table 4.1 including those in Table 4.1 itself (the training set). Because the probabilities in Table 4.2 are expressed in thousands, we may use sums of their logarithms rather than the probability products; this seems an intuitively appealing operation. Indeed, after such a transformation the score of a category is just the inner product of the row representing the tested entity and the feature scores corresponding to the category. Table 4.3 presents the logarithm scores of article E1 for each of the categories.

It should be mentioned that the Naïve Bayes computations here, as applied to the text categorization problem, follow the so-called multinomial model in which only terms present in the entities are considered – as many times as they occur. Another popular model is the so-called Bernoulli model, in which terms are assumed to be generated independently as binomial variables. The Bernoulli model based computations differ from these on two counts: first, the features are binary indeed so that only binary information, yes or no, of term occurrence is taken, and, second, for each term the event of its absence, along with its probability, is counted too (for more detail, see Manning et al. 2008, Mitchell 2010).

Table 4.2 Prior probabilities for Naïve Bayes rule for the data in Table 4.1 according to the bag-of-word conventions. There are three lines for each of the categories representing, from top to bottom, the term counts from Table 4.1, their probabilities multiplied by 1,000 and rounded to an integer, and the natural logarithms of the probabilities

Category	Prior probability Its logarithm	Total count	Term counts Term probabilities (in thousands) Logarithms of the probabilities									
F	1/3	27	3	5	0	2	2	3	0	5	0	7
			108	162	27	81	81	108	27	162	27	216
	−1.099		4.6	5.1	3.3	4.4	4.4	4.7	3.3	5.1	3.3	5.4
E	1/3	32	3	2	3	6	6	2	3	5	1	1
			95	71	95	167	167	71	95	143	48	48
	−1.099		4.6	4.3	4.6	5.1	5.1	4.3	4.6	5	3.9	3.9
H	1/3	31	1	0	6	3	3	7	5	0	6	0
			49	24	171	98	98	195	146	24	171	24
	−1.099		3.9	3.2	5.1	4.6	4.6	5.3	5	3.2	5.1	3.2

Table 4.3 Computation of category scores for entity E1 (first line) from Table 4.1 according to the logarithms of within-class feature probabilities. There are two lines for each of the categories: that on top replicates the logarithms from Table 4.2 and that on the bottom computes the inner product

Entity E1		2	0	1	2	2	0	0	1	0	0	
		Feature weights (probability logarithms)										
Category	Log(p_k)	Inner product										Score
F	−1.099	4.6	5.1	3.3	4.4	4.4	4.7	3.3	5.1	3.3	5.4	
		2∗4.6	+0+	1∗3.3	+2∗4.4	+2∗4.4	+0+	0+	1∗5.1	+0	+0	35.2
E	−1.099	4.6	4.3	4.6	5.1	5.1	4.3	4.6	5.0	3.9	3.9	
		2∗4.6	+0+	1∗4.6	+2∗5.1	+2∗5.1	+0+	0+	1∗5.0	+0	+0	**39.2**
H	−1.099	3.9	3.2	5.1	4.6	4.6	5.3	5.0	3.2	5.1	3.2	
		2∗3.9	+0	+ 1∗5.1	+2∗4.6	+2∗4.6	+0+	0+	1∗3.2	+0	+0	34.5

Q.4.1. Apply Naïve Bayes classifier in Table 4.2 to article X = (2 2 0 0 0 0 2 2 0 0) which involves items “drink”, “equal”, “relief” and “talent” frequently. **A.** The category scores are: s(F/X) = 35.2, s(E/X) = 35.6, and S(H/X) = 29.4 pointing to Entertainment or, somewhat less likely, Feminism.

Q.4.2. Compute Naïve Bayes category scores for all entities in Table 4.1 and prove that the classifier correctly attributes them to their categories. **A.** See Table 4.4.

Table 4.4 Naïve Bayes category scores for the items in Table 4.1

	Category scores		
Articles	F	E	H
F1	**37.7006**	35.0696	29.3069
F2	**28.9097**	25.9362	21.5322
F3	**24.9197**	20.1271	14.8723
F4	**38.2760**	34.6072	30.0000
E1	34.2349	**37.9964**	33.3322
E2	37.2440	**42.1315**	40.2435
E3	43.1957	**44.5672**	40.8398
E4	21.1663	**22.9203**	19.4367
H1	25.8505	29.3940	**34.5895**
H2	34.9290	40.4527	**42.7490**
H3	29.9582	35.3573	**38.3227**
H4	24.7518	28.8344	**34.8408**

4.2.3 Metrics of Accuracy

4.2.3.1 Accuracy and Related Measures: Presentation

Consider a generic problem of learning a binary target feature, so that all entities belong to either class 1 or class 2. A decision rule, applied to an entity, generates a "prediction" which of these two classes the entity belongs to. The classifier may return some decisions correct and some erroneous. Let us pick one of the classes as that of our interest, say 1, then there can be two types of errors: false positives (FP) – the classifier says that an entity belongs to class 1 while it does not, and false negatives (FN) – the classifier says that an entity does not belong to class 1 while it does.

Let it be, for example, a lung screening device for testing against a lung cancer. Whilst established in a hospital cancer ward, on a selected sample of 200 patients sent by local surgeries for investigation, it may produce results that are presented in Table 4.5. Its rows correspond to the diagnosis by the screening device and the columns to the results of further, more elaborate and definitive, tests. This is a cross-classification contingency table, and it is frequently referred to as a confusion table.

There are 94 true positives TP and 98 true negatives TN in the table so that the total accuracy of the device can be rated as $(94 + 98)/200 = 0.96 = 96\%$. Respectively, the numbers of false positives FP = 7, and false negatives FN = 1 sum to 8 leading to 4% error rate. Yet there are significant differences between these two showing that the device is in fact better than the totals show. Indeed, the 7 FP are not that important, because patients with the suspected cancer will be investigated further in depth anyway so that their No-status will be restored, with the cost of further testing. In contrast, 1 FN may go out of the medical system and get their cancer untreated with the potential loss of life because of the error. This is an example of different costs associated with FP and FN errors. The device made just one serious error: of 95 true cancer cases, one error. The TP rate, the proportion of correctly identified true cases, frequently referred to as recall or sensitivity, $94/95 = 98.9\%$, is impressive indeed. On the other hand, the precision, that is, the proportion of the 94 TP cases related to all cancer predicted cases, 101, is somewhat smaller, just 93% to reflect that FP rate is 7%. The difference between precision and sensitivity is somewhat averaged in the value of accuracy rate, 96% in this case, so that the accuracy rate works reasonably well here as a single characteristic of the quality of the testing device.

Yet in a situation in which there is a great disparity in the sizes of Yes and No classes, the accuracy rate fails to reflect the results properly. Consider, for example, results of the same device at a random sample of 200 individuals who have not been sent for the screening by doctors but rather volunteered to be screened from public at large (Table 4.6).

Table 4.5 Confusion table of patients' lung screening test results

		True lung cancer		
		Yes	No	Total
Device's	Yes	94	7	101
diagnosis	Not	1	98	99
Total		95	105	200

Table 4.6 Contingency table of volunteers' lung screening test results

		True lung cancer		
		Yes	No	Total
Device's diagnosis	Yes	2	2	4
	Not	1	195	196
Total		3	197	200

The accuracy rate at Table 4.6 is even greater than that at Table 4.5, $(2+195)/200 = 98.5\%$. Yet both sensitivity, $2/3 = 66.7\%$, and precision, $2/4 = 50\%$, are quite mediocre. The high accuracy rate is caused by the very high specificity, the proportion of correctly identified No cases, $195/197 = 98.9\%$, and by the fact that there are very few Yes cases.

As to a single measure adequately reflecting sensitivity and precision, the one most popular is their harmonic mean, the F-measure, which is equal to $F = 2/(1/(2/3) + 1/(2/4)) = 2/(3/2 + 4/2) = 4/7 = 57.1\%$.

Case study 4.1. Prevalence and Quetelet coefficients

If one looks at the record of the screening device according to Table 4.6 and compares that with the prevalence of the cancer at the sample, 3 cases of 200 – the difference is impressive indeed. This difference is exactly what is caught up in the concept of Quetelet coefficient $q(l/k)$ (see Section 3.3.3.2) at row $k = 1$ and column $l = 1$. This takes the relative difference between the conditional probability $P(1/1) = 2/4$ and the average probability $P(l = 1) = 3/200$ which is referred to sometimes as the prevalence: $q(1/1) = (2/4 - 3/200)/(3/200) = 2*200/(3*4) - 1 = 32.33 = 3233\%$, quite a change. This high value probably explains the difference in sensitivity and specificity between Tables 4.6 and 4.5.

Indeed, a similar Quetelet coefficient at Table 4.5 is $q(1/1) = 94*200/(101*95) - 1 = 0.96 = 96\%$, a less than a 100% increase, which may convey the idea that Table 4.5 is much more balanced than Table 4.6. The accuracy measure works well at balanced tables and it does not at those that are not.

4.2.3.2 Accuracy and Related Measures: Formulation

In general, the situation can be described by a confusion, or contingency, table between two sets of categories related to the class being predicted (1 or not) and the true class (1 or not), see Table 4.7. Of course, if one changes the class of interest, the errors will remain errors, but their labels will change: false positives regarding class 1 are false negatives when the focus is on class 2, and vice versa.

Table 4.7 A statistical representation of the match between the true class and predicted class. The entries are counts of the numbers of co-occurrences

		True class		
		1	Not	Total
Predicted class	1	True positives	False positives	TP+FP
	Not	False negatives	True negatives	FN+TN
Total		TP+FN	FP+TN	N

Among popular indexes scoring the error or accuracy rates are the following:

FP rate = FP/(FP + TN) – the proportion of false positives among those not in 1; 1-FP rate is referred to sometimes as specificity – it shows the proportion of correct predictions among other, not class 1, entities.
TP rate = TP/(TP + FN) – the proportion of true positives in class 1; in information retrieval, this frequently is referred to as recall or sensitivity.
Precision = TP/(TP + FP) – the proportion of true positives in the predicted class 1.

These reflect each of the possible errors separately. There are indexes that try to combine all the errors, too. Among them the most popular are:

Accuracy = (TP + TN)/N – the total proportion of accurate predictions. Obviously, 1-Accuracy is the total proportion of errors.
F-measure = 2/(1/Precision + 1/Recall) – the harmonic average of Recall and Precision.

The latter measure is getting more popularity than the former because the Accuracy counts both types of errors equally, which may be at odds with the common sense in those frequent situations at which errors of one type are "more expensive" than the others. Recall, for example, the case of medical diagnostics in Tables 4.5–4.6: a tumor wrongly diagnosed as malignant would cost much less than the other way around when a deadly tumor is diagnosed as benign. F-measure, to some extent, is more conservative because it, first, combines rates rather than counts, and, second, utilizes the harmonic mean which tends to be close to the minimum of the two, as can be seen from the statements in Q.4.3 and Q.4.4.

Q.4.3. Consider two positive reals, a and b, and assume, say that $a < b$. Prove that the harmonic mean, $h = 2/(1/a + 1/b)$ stays within the interval between a and $2a$ however large the difference b–a is. **A.** Take b be $b = ka$ at some $k > 1$. Then

$h = 2/(1/a + 1/(ka)) = 2\,ka/(1+k)$. The coefficient at a, $2\,k/(1+k)$, is less than 2, which proves the statement.

Q.4.4. Consider two positive real values, a and b, and prove that their mean, $m = (a+b)/2$, and harmonic mean, $h = 2/(1/a + 1/b)$, satisfy equation $mh = ab$. A. Take the product $mh = [(a+b)/2][2/(1/a+1/b)]$ and perform elementary algebraic operations.

More elaborate representation of errors of the two types can be achieved with the so-called receiver operating characteristics (ROC) graphs analysis (see, for example, Fawcett 2006). ROC graphs are especially suitable in the cases of classifiers that have a continuous output such as Bayes classifiers. ROC graph is a 2D Cartesian plane plotting TP rate against FP rate so that the latter is shown on x-axis and the former, on y-axis (see Fig. 4.6)

To be specific, let us take a Bayes classifier's rule in (4.4) and change the ratio p_2/p_1 for an arbitrary threshold $d>0$. Take now $d = d_1$ for a specific d_1, so that the rule now predicts class 1 if $f1(x)/f2(x) > d_1$. Count the proportions of true and false positives, *tp1* and *fp1*, at this threshold and put the point $(fp1,\ tp1)$ onto a ROC graph. Then change d to d_2 and count the rates, *tp2* and *fp2*, at this threshold. If, say, $d_2 > d_1$, then the TP rate can only decrease, because the number of positive predictions can only decrease. The FP rate, in a regular case, should increase at $d_2 > d_1$ so that point (fp2, ft2) would go to the right and above the former point on ROC plot. In this way, by step-by-step changing the threshold d, one can obtain a ROC curve such as curves "a" and "b" on the plot of Fig. 4.6. Such a curve can be utilized as a characteristic of the classifier under consideration that can be used, for instance, for selection of suitable levels of TP and FP rates. In the case shown on Fig. 4.6, one can safely claim that classifier "a" is superior to that of "b", because at each FP rate level, TP rate of "a" is greater than that of "b".

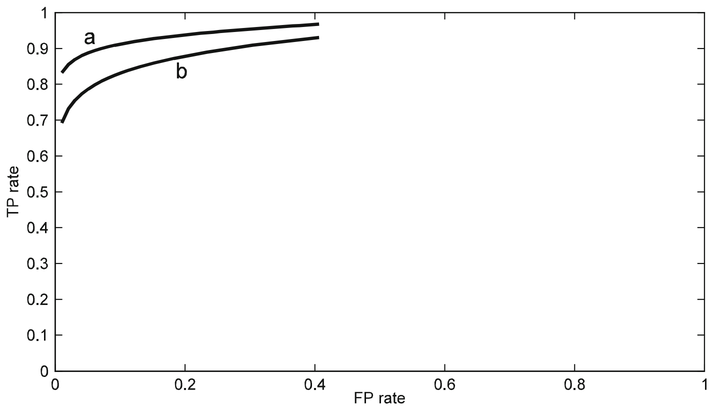

Fig. 4.6 ROC curves for two classifiers; that of *a* is superior to that of *b*

4.3 Linear Regression

P4.3.1 Linear Regression: Presentation

Let us extend the notion of linear regression from the bivariate case considered in Section 3.1 to multivariate case, when several features can be used as predictors for a target feature.

Case study 4.2. Linear regression for Market town data

Consider feature Post expressing the number of post offices in Market towns (Table 1.4 on pp. 14) and try to relate it to other features in the table. It obviously relates to the population. For example, towns with population of 15,000 and greater are those and only those where the number of post offices is 5 or greater. This correlation, however, is not as good as to give us more guidance in predicting Post from the Population. For example, at the seven towns whose population is from 8,000 to 10,000 any number of post offices from 1 to 4 may occur, according to the table. This could be attributed to effects of services such as a bank or hospital present at the towns. Let us specify a set of features in Table 1.4 that can be thought of as affecting the feature Post, to include in addition to Population some other features – PS-Primary schools, Do – General Practitioners, Hos- Hospitals, Ba- Banks, Sst – Superstores, and Pet– Petrol Stations; seven features altogether, taken as the set of input variables (predictors).

What we want is to establish a linear relation between the set and target feature Post. A linear relation is an equation representing Post as a weighted sum of input features plus a constant intercept; the weights can be any reals, not necessarily positive. If the relation is supported by the data, it can be used for various purposes such as analysis, prediction and planning.

In the example of seven Market town features used for linearly relating them to Post Office feature, the least-squares optimal weight coefficients are presented in Table 4.8. Each weight coefficient shows how much the target variable would change on average if the corresponding feature is increased by a unity, while the others do not change. One can see that increasing population by a thousand would give a similar effect as adding a primary school, about 0.2, which may seem absurd in the example as Post Office variable can have only integer values. Moreover, the linear function format should not trick the decision maker into thinking that increasing different input features can be done independently: the features are obviously not independent so that increase of, say, the population will lead to respectively adding new schools for the additional children. Still, the weights show relative effects of the features – according to Table 4.8, adding a doctor's surgery in a town would lead to maximally possible increase in post offices. The maximum value is assigned to the intercept in this case. What this may mean? Is it the number of post offices in an empty town with no population, hospitals or petrol stations? Certainly not. The intercept expresses that part of the target variable which is relatively independent

Table 4.8 Weight coefficients of input features at Post Office as target variable for Market towns data

Feature	Weight
Pop_Res	0.0002
PSchools	0.1982
Doctors	0.2623
Hospitals	−0.2659
Banks	0.0770
Superstores	0.0028
Petrol	−0.3894
Intercept	0.5784

of the features taken into account. It should be also pointed out that the weight values are relative not to just feature concepts but specific scales in which features measured. Change of a feature scale, say ten fold, would result in a corresponding, inverse, change of its weight (due to the linearity of the regression equation). This is why in statistics, the relative weights are considered for the scales expressed in units of the standard deviation. To find them, one should multiply the weight for the current scale by the feature's standard deviation (see Table 4.9).

The standardized weights are well justified when input features are mutually uncorrelated – indeed, they show the pair-wise correlation with the target feature. Yet in a situation of correlated features, like this, they seem to have much less definite interpretation, except for showing the changes of the target in units of the standard deviations, although some claim that they also reflect feature's correlation with the target or even importance for predicting the target. An argument against their usage as a correlation measure is that, in fact, a regression coefficient multiplied by the standard deviation loses its "purity" as a measure of correlation to the target at constant levels of the other features because the standard deviation does not pertain to constant features. An argument against their usage as measures of importance for prediction is that the standardized coefficient has nothing to do with the change of the coefficient of determination when the corresponding feature is removed from the equation of regression.

Table 4.9 Different indexes to express the idea of importance of a feature in the Post regression problem

Feature	Weights in natural scales, w	Standard deviations, s	Weights in standardized scales, w∗s
Pop_Res	0.0002	6, 193.2	1.3889
PSchools	0.1982	2.7344	0.5419
Doctors	0.2623	1.3019	0.3414
Hospitals	−0.2659	0.5800	−0.1542
Banks	0.0770	4.3840	0.3376
Superstores	0.0028	1.7242	0.0048
Petrol	−0.3894	1.637	−0.6375

J. Bring (1994) proposes to kill two birds with one stone: to clean up the standard deviations from the non-constancy of the other features, which are claimed to reflect the changes in the coefficients of determination. Specifically, take the variance of a feature and take off the proportion of it unexplained by the linear regression of it through the other features. The square root of the result represents the partial standard deviation, which is proportional to the so-called "t-value", and, in the original squared form, to the change of the coefficient of determination inflicted by the removal of the feature from the list of the explanatory variables (Bring 1994). Unfortunately, this is not that simple, as the next case study 4.3 shows.

Case study 4.3. Using feature weights standardized

Table 4.10 presents the feature weights standardized with both the original and partial standard deviations as well as the absolute reductions of the original coefficient of determination 0.8295 after removal of the corresponding variables. There is a general agreement between the absolute values of the first column and those in the third column, but the second column has little in common with either of them. A general analysis of a simpler problem of relation between the regression coefficients and correlation coefficients between the target and input features can be found in Waller and Jones (2010).

Amazingly, the convenient standardization involves negative weights, specifically at features Petrol and Hospitals. This can be an artifact of the method, related to the effect of "replication" of features. One can think of Hospitals being a double for Doctors, and Petrol, for Superstores. Thus, before jumping to conclusions, one should check whether the minus disappears if the "replicas" are removed from the set of features. As Table 4.11 shows, not in this case: the negative weights remain, though they slightly change, as well as other weights. This illustrates that the interpretation of linear regression coefficients as weights should be cautious and restrained.

Table 4.10 Standardized weight coefficients of input features at Post Office as target variable for Market towns

Feature	Weights in standard deviation scales	Weights with partial standard deviations	Determination coefficient reduction
Pop_Res	1.3889	1,602.00	0.0247
PSchools	0.5419	1.02	0.0077
Doctors	0.3414	0.64	0.0055
Hospitals	−0.1542	0.41	0.0023
Banks	0.3376	2.27	0.0059
Superstores	0.0048	1.07	0
Petrol	−0.6375	0.96	0.0251

Table 4.11 Weight coefficients for reduced set of features at Post Office as target variable for Market towns data

Feature	Weight
POP_RES	0.0003
PSchools	0.1823
Hospitals	−0.3167
Banks	0.0818
Petrol	−0.4072
Intercept	0.5898

In our example, coefficient of determination $\rho^2 = 0.83$, that is, the seven features explain 83% of the variance of Post Office feature, and the multiple correlation is $\rho = 0.91$. Curiously, the reduced set of five features (see Table 4.11) contributes almost the same, 82.4% of the variance of the target variable. This may make one wonder whether just one Population feature could suffice for doing the regression. This can be tested with the 2D method described in Section 3.2 or with the nD method of this section.

According to the formulation of the multivariate linear regression method further in F3.3, the estimated parameters must be feature weight coefficients – no room for an intercept in the formula. To accommodate the intercept, a fictitious feature whose all values are unities is introduced. That is, an input data matrix X with two columns is to be used: one for the Population feature, the other for the fictitious variable of all ones. According to (4.10), this leads to the slope 0.0003 and intercept 0.4015, though with somewhat reduced coefficient of determination, which is $\rho^2 = 0.78$ in this case. From the prediction point of view this may be all right, but the reduced feature set looses on interpretation.

F4.3.2 Linear Regression: Formulation

The problem of linear regression can be formulated as a particular case of the correlation learning problem with just one quantitative target variable u and linear admissible rules so that

$$u = w_1x_1 + w_2x_2 + \ldots + w_px_p + w_0$$

where $w_0, w_1, \ldots, w_p$ are unknown weights, parameters of the model.

For any entity $i = 1, 2, \ldots, N$, the rule-computed value of u

$$\hat{u}_i = w_1x_{i1} + w_2x_{i2} + \ldots + w_px_{ip} + w_0$$

differs from the observed one by $d_i = |\hat{u} - u_i|$, which may be zero – when the prediction is exact. To find $w_1, w_2, \ldots, w_p, w_0$, one can minimize the summary square error

$$D^2 = \Sigma_i {d_i}^2 = \Sigma_i (u_i - w_1 * x_{i1} - w_2 * x_{i2} - \ldots - w_p * x_{ip} - w_0)^2 \tag{4.7}$$

over all possible parameter vectors $w = (w_0,\ w_1, \ldots, w_p)$.

To make the problem treatable in terms of linear operations, a fictitious feature x_0 is introduced such that all its values are 1 : $x_{i0} = 1$ for all $i = 1, 2, \ldots, N$. Then criterion D^2 can be expressed as $D^2 = \Sigma_i (u_i - < w_i, x_i >)^2$ using the inner products $< w, x_i >$ where $w = (w_0, w_1, \ldots, w_p)$ and $x_i = (x_{i0},\ x_{i1},\ \ldots,\ x_{ip})$ are $(p + 1)$-dimensional vectors of which all x_i are known whereas w is not. From now on, the unity feature x_0 is assumed to be part of data matrix X in all correlation learning problems in this text.

The criterion D^2 in (4.7) is but the squared Euclidean distance between the N-dimensional target feature column $u = (u_i)$ and vector $\hat{u} = Xw$ whose components are $\hat{u}_i = < w, x_i >$. Here X is $N \times (p + 1)$ matrix whose rows are x_i (augmented with the component $x_{i0} = 1$, thus being $(p + 1)$-dimensional) so that Xw is the matrix product of X and w. Vectors defined as Xw for all possible w's form $(p + 1)$-dimensional vector space, referred to as X-span.

Thus the problem of minimization of (4.7) can be reformulated as follows: given target vector u, find its projection $\hat{u}$ in the X-span space. The global solution to this problem is well-known: it is provided by a matrix P_X applied to u:

$$\hat{u} = P_X u$$

where P_X is the so-called orthogonal projection operator, an $N \times N$ matrix, defined as:

$$P_X = X(X^T X)^{-1} X^T$$

so that

$$\hat{u} = X(X^T X)^{-1} X^T u \text{ and } w = (X^T X)^{-1} X^T u \tag{4.8}$$

Matrix P_X projects every N-dimensional vector u to its nearest match in the $(p + 1)$-dimensional X-span space. The inverse $(X^T X)^{-1}$ does not exist if the rank of X, as it may happen, is less than the number of columns in X, $p + 1$, that is, if matrix $X^T X$ is singular or, equivalently, the dimension of X-span is less than $p + 1$. In this case, the so-called pseudo-inverse matrix $(X^T X)^+$ can be used as well. This is not a big deal computationally: for example, in MatLab one just puts pinv(X^TX) instead of inv(X^TX).

The quality of approximation is evaluated by the minimum value D^2 in (4.7) averaged over the number of entities and related to the variance of the target variable. Its complement to 1, the coefficient of determination, is defined by the equation

$$\rho^2 = 1 - D^2/(N\sigma^2(u)) \tag{4.9}$$

The coefficient of determination shows the proportion of the variance of u explained by the linear regression. Its square root, ρ, is referred to as the coefficient of multiple correlation between u and $X = \{x_0, x_1,\ x_2,\ \ldots,\ x_p\}$.

4.4 Linear Discrimination and SVM

P4.4.1 Linear Discrimination and SVM: Presentation

Discrimination is an approach to address the problem of drawing a rule to distinguish between two classes of entity points in the feature space, a "yes" class and "no" class, such as for instance a set of banking customers in which a, typically very small, subset of fraudsters constitutes the "yes" class and that of the others the "no" class. On Fig. 4.7, entities of "yes" class are presented by circles and of "no" class by squares.

The problem is to find a function $u = f(x)$ that would separate the two classes in such a way that *f(x)* is positive for all entities in the "yes" class and negative for all the entities in the "no" class. When the discriminant function *f(x)* is assumed to be linear, the problem is of linear discrimination. It differs from that of the linear regression in that aspect that the target values here are binary, either "yes" or "no", so that this is a classification rather than regression, problem.

The classes on Fig. 4.7 can be discriminated by a straight – dashed – line indeed. The dotted vector *w*, orthogonal to the "dashed line" hyperplane, represents a set of coefficients at the linear classifier represented by the dashed line. Vector *w* also shows the direction at which function $f(x) =< w, x > -b$ grows. Specifically, *f(x)* is 0 on the separating hyperplane, and it is positive above and negative beneath that. With no loss of generality, *w* can be assumed to have its length equal to unity. Then, for any *x*, the inner product $< w, x >$ expresses the length of vector *x* along the direction of *w*.

To find an appropriate *w*, even in the case when "yes" and "no" classes are linearly separable, various criteria can be utilized. A most straightforward classifier is defined as follows: put 1 for "yes" and –1 for "no" and apply the least-squares criterion of linear regression. This produces a theoretically sound solution approximating the best possible – Bayesian – solution in a conventional statistics model. Yet, in spite of its good theoretical properties, least-squares solution may be not

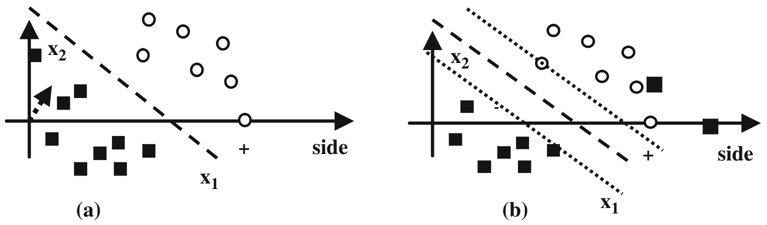

Fig. 4.7 A geometric illustration of a separating hyper-plane between classes of *circles and squares*. The *dotted vector w* on (**a**) is orthogonal to the hyper-plane: its elements are hyper-plane coefficients, so that it is represented by equation $< w, x > -b = 0$. Vector *w* also points at the direction: at all points above the *dashed line*, the *circles* included, function $f(x) =< w, x > -b$ is positive. The *dotted lines* on (**b**) show the margin, and the *squares and circle* on them are support vectors

necessarily the best at some data configurations. In fact, it may even fail to separate the positives from negatives when they are linearly separable. Consider the following example.

Worked example 4.1. A failure of Fisher discrimination criterion

Let there be 14 2D points presented in Table 4.12 (first line) and displayed in Fig. 4.8a. Points 1,2,3,4,6 belong to the positive class (dots on Fig. 4.8a), and the others to the negative class (stars on Fig. 4.8a). Another set, obtained by adding to each of the components a random number, according to the normal distribution with zero mean and 0.2 the standard deviation; is presented in the bottom line of Table 4.12 and Fig. 4.8b. The class assignment for the disturbed points remains the same.

The optimal vectors *w* according to formula (4.8) are presented in Table 4.13 as well as that for the separating, dotted, line in Fig. 4.8d.

Note that the least-squares solution depends on the values assigned to classes, leading potentially to an infinite number of possible solutions under different numerical codes for "yes" and "no". A popular discriminant criterion of minimizing the ratio of a "within-class error" over "out-of-class error", proposed by R. Fisher in his founding work of 1936, in fact, can be expressed with the least-squares criterion as well. Just change the target as follows: assign N/N_1, rather than +1, to "yes" class and $-N/N_2$ to "no" class, rather than –1 (see Duda et al. 2001, pp. 242). This means that Fisher's criterion may also lead to a failure in a linear separable situation.

Q.4.5. Why only 10, not 14, points are drawn on Fig. 4.8b? **A.** Because each of the points 11–14 doubles a point 7–10.

Q.4.6. What would change if the last four points are removed so that only points 1–10 remain? **A.** The least-squares solution will be separating again.

By far the most popular set of techniques, Support Vector Machine (SVM), utilize a different criterion – that of maximum margin. The margin of a point *x*, with

Table 4.12 *x–y* coordinates of 14 points as given originally and perturbed with a noise generated from the Gaussian distribution N(0, 0.2)

Entity #		1	2	3	4	5	6	7	8	9	10	11	12	13	14
Original data	*x*	3.00	3.00	3.50	3.50	4.00	1.50	2.00	2.00	2.00	1.50	2.00	2.00	2.00	1.50
	y	0.00	1.00	1.00	0.00	1.00	4.00	4.00	5.00	4.50	5.00	4.00	5.00	4.50	5.00
Perturbed data	*x*	2.93	2.83	3.60	3.80	3.89	1.33	1.95	2.13	1.83	1.26	1.98	1.99	2.10	1.38
	y	−0.03	0.91	0.98	0.31	0.88	3.73	4.09	4.82	4.51	4.87	4.11	5.11	4.46	4.59

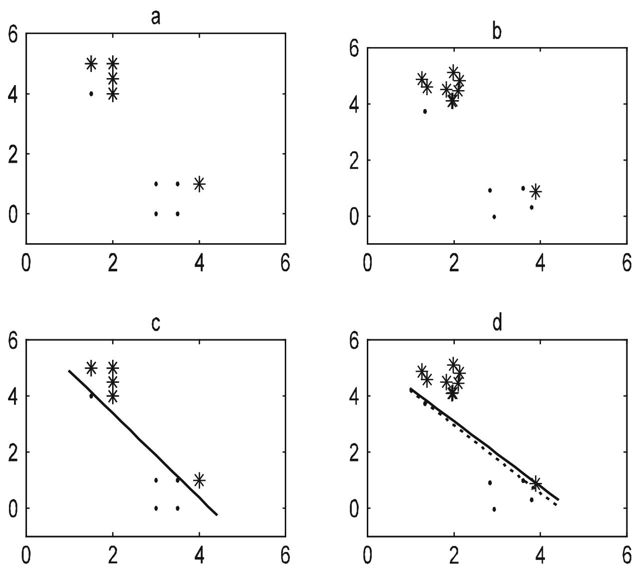

Fig. 4.8 Figures (**a**) and (**b**) represent the original and perturbed data. The least *squares optimal separating line* is added in Figures (**c**) and (**d**) shown by *solid*. Entity 5 falls into the "*dot*" class according to the *solid line* in (**d**); *a separating line* is shown *dotted* there

respect to a hyperplane, is the distance from x to the hyperplane along its perpendicular vector w (Fig. 4.7a), which is measured by the absolute value of inner product $< w, x >$. The margin of a class is defined by the minimum value of the margins of its members. Thus the criterion requires, like L_∞, finding such a hyperplane that maximizes the minimum of class margins, that is, crosses the middle of the line between the nearest entities of two classes. Those entities that fall on the margins, shown by dotted lines on Fig. 4.7b, are referred to as support vectors; this explains the method's name.

It should be noted that the classes are not necessarily linearly separable; moreover in most cases they are not. Therefore, the SVM technique is accompanied with a non-linear transformation of the data into a high-dimensional space which is more likely to make the classes linear-separable, as illustrated on Fig. 4.9. Such

Table 4.13 Coefficients of straight lines on Fig. 4.8

	Coefficients at		
	X	Y	Intercept
LSE at original data	−1.2422	−0.8270	5.2857
LSE at perturbed data	−0.8124	−0.7020	3.8023
Dotted at perturbed data	−0.8497	−0.7020	3.7846

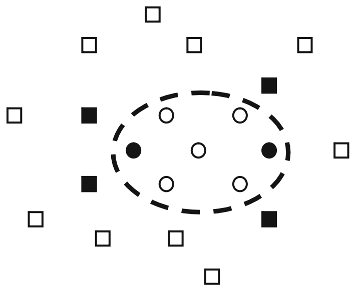
Fig. 4.9 Illustrative example of 2D entities belonging to two classes, *circles* and *squares*. The *separating line* in the space of Gaussian kernel is shown by the *dashed oval*. The support entities are shown by *black*

a non-linear transformation is provided by the so-called kernel function. The kernel function imitates the inner product in the high-dimensional space and is represented by a between-entity similarity function such as that defined by formula (4.12) on p. 140.

The intuition behind the SVM approach is this: if the population data – those not present in the training sample – concentrate around training data, then having a wide margin would keep classes separated even after other data points are added (see Fig. 4.7). One more consideration comes from the Minimum Description Length principle: the wider the margin, the more robust the separating hyperplane is and the less information of it needs to be stored. A criticism of the SVM approach is that the support vector machine hyperplane is based on the borderline objects – support vectors – only, whereas the least-squares hyperplanes take into account all the entities so that the further away an entity is the more it may affect the solution, because of the quadratic nature of the least-squares criterion. Some may argue that both borderline and far away entities can be rather randomly represented in the sample under investigation so that neither should be taken into account in distinguishing between classes: it is some "core" entities of the patterns that should be separated – however, there has been no such an approach taken in the literature so far.

Worked example 4.2. SVM for Iris dataset

Consider Iris dataset standardized by subtracting, from each feature column, its midrange and dividing the result by the half-range.

Take Gaussian kernel in (4.15) to find a support vector machine surface separating Iris class 3 from the rest. The resulting solution embraces 21 supporting entities (see Table 4.14), along with their "alpha" prices reaching into hundreds and even, on two occasions, to the maximum boundary 500 embedded in the algorithm.

There is only one error with this solution, entity 78 wrongly recognized as belonging to taxon 3. The errors increase when we apply a cross-validation techniques, though. For example, "leave-all-one-out" cross-validation leads to nine errors: entities 63, 71, 78, 82 and 83 wrongly classified as belonging to taxon 3 (false positives), while entities 127, 133, 135 and 139 are classified as being out of taxon 3 (false negatives).

Table 4.14 List of support entities in the problem of separation of taxon 3 (entities 101–150) in Iris data set from the rest (thanks to V. Sulimova for computation)

N	Entity	Alpha	N	Entity	Alpha
1	18	0.203	12	105	2.492
2	28	0.178	13	106	15.185
3	37	0.202	14	115	52.096
4	39	0.672	15	118	15.724
5	58	13.630	16	119	449.201
6	63	209.614	17	127	163.651
7	71	7.137	18	133	500
8	78	500	19	135	5.221
9	81	18.192	20	139	16.111
10	82	296.039	21	150	26.498
11	83	200.312			

F4.4.2 Linear Discrimination and SVM: Formulation

F4.4.2.1 Linear Discrimination

The problem of linear discrimination can be stated as follows. Let a set of N entities in the feature space, $x_i = (x_{i0}, x_{il}, x_{i2}, \ldots, x_{ip})$ $i = 1, 2, \ldots, N$ is partitioned in two classes, sometime referred to as patterns, a "yes" class and a "no" class, such as for instance a set of banking customers in which a, typically very small, subset of fraudsters constitutes the "yes" class and that of the others the "no" class. The problem is to find a function $u = f(x_0, x_1, x_2, \ldots, x_p)$ that would discriminate the two classes in such a way that u is positive for all entities in the "yes" class and negative for all entities in the "no" class. When the discriminant function is assumed to be linear so that $u = w_1x_1 + w_2x_2 + \ldots + w_px_p + w_0$ at constant $w_0,\ x_1, \ldots, w_p$, the problem is of linear discrimination.

To make it quantitative, define $u_i = 1$ if i belongs to the "yes" class and $u_i = -1$ if i belongs to the "no" class. The intercept w_0 is referred to, in the context of the discrimination/classification problem, as bias.

A linear classifier is defined by a vector w so that if $\hat{u}_i =< w, x_i >> 0$, predict $\mathring{u}_i = 1$; if $\hat{u}_i =<< w, x_i >< 0$, then predict $\mathring{u}_i = -1$; that is, $\mathring{u}_i = sign(< w, x_i >)$. (Here the sign function is utilized as defined by the condition that $sign(a) = 1$ when $a > 0, = -1$ when $a < 0$, and $= 0$ when $a = 0$.)

To find an appropriate w, even in the case when "yes" and "no" classes are linearly separable, various criteria can be utilized. A most straightforward classifier is defined by the least-squares criterion of minimizing (4.7). This produces

$$w = (X^TX)^{-1}X^Tu \tag{4.10}$$

Note that formula (4.10) leads to an infinite number of possible solutions because of the arbitrariness in assigning different u-labels to different classes. A slightly different criterion of minimizing the ratio of the "within-class error" over "out-of-class error" was proposed by R. Fisher (1936). Fisher's criterion, in fact, can be expressed with the least-squares criterion if the output vector u is changed for u_f as follows: put N/N_1 for the components of the first class, instead of +1, and put $-N/N_2$ for the entities of the second class, instead of –1. Then the optimal w (4.10) at $u = u_f$ minimizes the Fisher's discriminant criterion (see Duda, Hart, Stork, 2001, p. 242).

Solution (4.10) has two properties related to the Bayes decision rule. It appears the squared summary difference between the least-square error linear decision rule function $<w, x>$ and Bayes function *B(x)* approaches minimum when N grows infinitely (Duda et al. p. 243). Moreover, the least-squares linear decision rule is the Bayes function *B(x)* if the class probability distributions $f1(x)$ and $f2(x)$ are Gaussian with coinciding covariance matrices, so that they can be expressed with formula:

$$f_i(x) = \exp[-(x - \mu_i)^T \Sigma^{-1}(x - \mu_i)/2]/[(2\pi)^p|\Sigma|]^{1/2}$$

where μ_i is the central point and $\sum$ the *pxp* covariance matrix of the Gaussian distribution. In fact, in this case the optimal $w = \Sigma^{-1}(\mu_1 - \mu_2)$(see Duda, Hart, Stork, p. 40).

F4.4.2.2 Support Vector Machine (SVM) Criterion

Another criterion would put the separating hyperplane just in the middle of an interval drawn through closest points of the different patterns. This criterion produces what is referred to as the support vector machine since it heavily relies on the points involved in the drawing of the separating hyperplane (as shown on the right of Fig. 4.7). These points are referred to as support vectors. A natural formulation would be like this: find a hyperplane $H :< w, x >= b$ with a normed w to maximize the minimum of absolute values of distances $| < w, x_i > -b|$ from points x_i belonging to each of the classes. This, however, is rather difficult to associate with a conventional formulation of an optimization problem because of the following irregularities:

(i) an absolute value to maximize,
(ii) the minimum over points from each of the classes, and
(iii) w being of the length 1, that is, normed.

However, these all can be successfully tackled. The issue (i) is easy to handle, because there are only two classes, on the different sides of H. Specifically, the distance is $< w, x_i > -b$ for "yes" class and $- < w, x_i > +b$ for "no" class – this removes the absolute values. The issue (ii) can be taken care of by uniformly using inequality constraints

$< w, x_i > -b \geq \lambda$ for x_i in "yes" class and
$- < w, x_i > +b \geq \lambda$ for x_i in "no" class

and maximizing the margin λ with respect to these constraints. The issue (iii) can be addressed by dividing the constraints by λ so that the norm of the weight vector becomes $1/\lambda$, thus inversely proportional to the margin λ. Moreover, one can change the criterion now because the norm of the ratio w/λ is minimized when λ is maximized. Denote the "yes" class by $u_i = 1$ and "no" class by $u_i = -1$. Then the problem of deriving a hyperplane with a maximum margin can be reformulated, without the irregularities, as follows: find b and w such that the norm of w or its square, $< w, w >$, is minimum with respect to constraints

$$u_i(< w, x_i > -b) \geq 1 \quad (i = 1, 2, \ldots, N)$$

This is a problem of quadratic programming with linear constraints, which is easier to analyze in the format of its dual optimization problem. The dual problem can be formulated by using the so-called Lagrangian, a common concept in optimization, that is, the original criterion penalized by the constraints weighted by the so-called Lagrangian multipliers that are but penalty rates. Denote the penalty rate for the violation of i-th constraint by α_i. Then the Lagrangian can be expressed as

$$L(w, b, \alpha) =< w, w>/2 = \Sigma_i \alpha_i (u_i(< w, x_i > -b) - 1),$$

where $< w, w >$ has been divided by 2 with no loss of generality, just for the sake of convenience. The optimum solution minimizes L over w and b, and maximizes L over non-negative α. The first order optimality conditions require that all partial derivatives of L are zero at the optimum, which leads to equations $\Sigma_i \alpha_i u_i = 0$ and $w = \Sigma_i \alpha_i u_i x_i$. Multiplying the latter expression by itself leads to equation $< w, w >= \Sigma_{ij} \alpha_i \alpha_j u_i u_j < x_i, x_j >$. The second item in Lagrangian L becomes equal to $\Sigma_i \alpha_i u_i < w, x_i > -\Sigma_i \alpha_i u_i b - \Sigma_i \alpha_i =< w, w > -0 - \Sigma_i \alpha_i$. This leads us to the following, dual, problem of optimization regarding the Lagrangian multipliers, which is equivalent to the original problem: Maximize criterion

$$\Sigma_i \alpha_i - \Sigma_{ij} \alpha_i \alpha_j u_i u_j < x_i, x_j>/2 \tag{4.11}$$

subject to $\Sigma_i \alpha_i u_i = 0$ and $\alpha_i \geq 0$.

Support vectors are defined as those x_i for which penalty rates are positive, $\alpha_i > 0$, in the optimal solution – only they contribute to the optimal vector $w = \Sigma_i \alpha_i u_i x_i$; the others have zero coefficients and disappear.

It should be noted that the margin constraints can be violated, which is not difficult to take into account – by using non-negative values η_i expressing the sizes of violations:

$$u_i(< w, x_i > -b) \geq 1 - \eta_i \quad (i = 1, 2, \ldots, N)$$

in such a way that they are minimized in a combined criterion $< w, w>/2 + C\Sigma_i \eta_i$ where C is a large "reconciling" coefficient that is a user-defined parameter. The dual problem for the combined criterion remains almost the same as above, in spite of the fact that an additional set of dual variables, β_i, needs to be introduced as corresponding to the constraints $\eta_i \geq 0$. Indeed, the Lagrangian for the new problem can be expressed as

$$L(w, b, \alpha, \beta) =< w, w>/2 - \Sigma_i \alpha_i (u_i(< w, x_i > -b) - 1) - \Sigma_i \eta_i (\alpha_i + \beta_i - C)$$

which differs from the previous expression by just the right-side item. This implies that the same first-order optimality equations hold, $\Sigma_i \alpha_i u_i = 0$ and $w = \Sigma_i \alpha_i u_i x_i$, plus additionally $\alpha_i + \beta_i = C$. These latter equations imply that $C \geq \alpha_i \geq 0$ because β_i are non-negative.

Since the additional dual variables are expressed through the original ones, $\beta_i = C - \alpha_i$, the dual problem can be shown to remain unchanged and it can be solved by using quadratic programming algorithms (see Vapnik 2001 and Schölkopf and Smola 2005). Recently, approaches have appeared for solving the original problem as well (see Groenen et al. 2008).

F4.4.2.3 Kernels

Situations at which patterns are linearly separable are very rare; in real data, classes are typically well intermingled with each other. To attack these typical situations with linear approaches, the following trick can be applied. The data are nonlinearly transformed into a much higher dimensional space in which, because of both nonlinearity and higher dimension, the classes may be linearly separable. The transformation can be performed only virtually because of specifics of the dual problem: dual criterion (4.11) depends not on individual entities but rather just inner products between them. This property obviously translates to the transformed space, that is, to transformed entities. The inner products in the transformed space can be computed with the so-called kernel functions $K(x, y)$ so that in criterion (4.11) inner products $< x_i, x_j >$ are substituted by the kernel values $K(x_i, x_j)$. Moreover, by substituting the expression $w = \Sigma_i \alpha_i u_i x_i$ into the original discrimination function $f(x) =< w, x > -b$ we obtain its different expression $f(x) = \Sigma_i \alpha_i u_i < x, x_i > -b$, also involving inner products only, which can be used as a kernel-based decision rule in the transformed space: x belongs to "yes" class if $\Sigma_i \alpha_i u_i K(x, x_i) - b > 0$.

It is convenient to define a kernel function over vectors $x = (x_v)$ and $y = (y_v)$ through the squared Euclidean distance $d^2(x, y) = (x_1 - y_1)^2 + \ldots + (x_V - y_V)^2$ because matrix $(K(x_i, x_j))$ in this case is positive definite – a defining property of matrices of inner products. Arguably, the most popular is the Gaussian kernel defined by:

$$K(x, y) = \exp(-d^2(x, y)) \tag{4.12}$$

Q.4.7. Consider a full set B$_n$ of 2$^{\mathrm{n}}$ binary 1/0 vectors of length n like those presented by columns below for $n = 3$:

1	0	0	0	0	1	1	1	1
2	0	0	1	1	0	0	1	1
3	0	1	0	1	0	1	0	1

These columns can be considered as integers coded in the binary number system; moreover, they are ordered from 0 to 7. Prove that this set shutters any subset of n (or less) points.
A. Indeed, let S be a set of elements $i_1, i_2, \ldots, i_n$ in B_n that are one-to-one labeled by numbers from 1 to n. Consider any partition of S in two classes, S$_1$ and S$_2$. Assign 0 to each element of S$_1$ and 1 to each element of S$_2$.The partition follows that vector of B$_n$ that corresponds to the assignment.

Q.4.8. Consider set B_n defined above. Prove that its rank is n, that is, there are n columns in matrix B_n that form a base of the space of n-dimensional vectors. **A.** Take, for example, n columns e_p that contain unity at p-th position whereas other n–1 elements are zero ($p = 1, 2, \ldots n$). These obviously are mutually orthogonal and any vector $x = (x_1, \ldots, x_n)$ can be expressed as a linear combination $x = \Sigma_p x_p e_p$, which proves that vectors e_p form a base of the n-dimensional space.

Q.4.9. What is VC-dimension of the linear discrimination problem at an arbitrary dimension $p \geq 2$? **A.** p+1, because each subset of p points can be separated from the others by a hyperplane, but there can be such $(p + 1)$-point configurations that cannot be shattered using liner separators.

4.5 Decision Trees

P4.5.1 General: Presentation

Decision tree is a structure used for learning and predicting quantitative or nominal target features. In the former case it is referred to as a regression tree, in the latter, classification tree. This structure can be considered a multivariate extension of contingency tables in such a way that only meaningful combinations of feature categories are involved.

As illustrated on Fig. 4.10, a decision tree recursively partitions the entity set into smaller clusters by splitting a parental cluster over a single feature. The root of a decision tree corresponds to the entire entity set. Each node corresponds to a subset of entities, cluster, and its children are the cluster's parts defined by values of a single predictor feature x. Note that the trees on Fig. 4.10 are binary: each interior node is split in two parts. This is a most convenient format, currently used in most popular programs. Only binary trees are considered in this section.

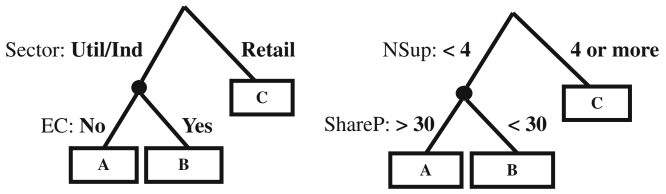

Fig. 4.10 Decision trees for three product based classes of Companies, A, B, and C, made using categorical features, *on the left*, and quantitative features, *on the right*

Decision trees are built from top to bottom in such a way that every split is made to maximize the homogeneity of the resulting subsets with respect to a desired target feature. The splitting stops either when the homogeneity is enough for a reliable prediction of the target feature values or when the set of entities is too small to consider its splits reliable. A function scoring the extent of homogeneity to decide of the stopping is, basically, a measure of correlation between the partition of the entity set being built and the target feature.

When the process of building a tree is completed, each terminal node is assigned with a value of the target that is determined to be characteristic for that node, and thus should be predicted at the conditions leading to the node. For example, both trees on Fig. 4.10 are precise – each terminal class corresponds to one and only one product, which is the target feature, so that each of the trees gives a precise conceptual description of all products by conjunctions of the corresponding branch values. For example, product A can be described as that which is not in Utility sector, nor E commerce utilized in the production process (left-side tree) or as that in which less than 4 suppliers are involved and the share price is greater than 30. Both descriptions are fitting here since both give no errors at all.

Decision trees are very popular because they are simple to understand, use, and interpret. However, one should properly use them, because the decision rules produced with them can be overly simplistic and frequently imprecise. Their effectiveness much depends on the features and samples selected for the analysis. As always in learning correlation, a simpler tree is preferred to a complex one because of the over-fitting problem: a complex tree is more likely reflect noise in the data rather than the true tendencies.

In the next section, we discuss popular homogeneity scoring functions and then proceed to the process of classification tree building, in yet another section.

F4.5.2 General: Formulation

To build a binary decision tree, one needs the following information:

(a) set of input features X,
(b) an output feature u,
(c) a scoring function $W(S, u)$ that scores admissible partitions S against the output feature,

(d) rule for obtaining admissible partitions,
(e) stopping criterion
(f) rule for pruning long or unreliable branches, and
(g) rule for the assignment of u-values to terminal nodes.

Let us comment on each of these items:

(a) The input features are, typically, quantitative or nominal. Quantitative features are handled rather easily by testing all possible splits of their ranges. More problematic are categorical features especially those with many categories because the number of possible binary splits can be very large. However, this issue does not emerge at all if categorical features are preprocessed into the quantitative format of binary dummy variables corresponding to individual categories (which is advocated in this text too, see more detail in Section 5.1). Indeed, each of the dummy variables admits only one split – that separating the corresponding category from the rest, which reduces the number of possible splits to consider to the number of categories – an approach advocated by Loh and Shih (1997). A number of such splits can be done in sequence to warrant that any combination of categories is admissible in this approach too.

   Since this approach involves one feature at a time only, missing values are not of an issue, because all the relevant information such as means and frequencies can be reasonably well estimated from those values that are available – this is a stark contrast with the other multivariate techniques.
(b) In principle, the decision tree format does not prevent from using multiple target features – just single-target criteria should be summed up when there are several targets (Mirkin 1985). However, all current internationally available programs involve only single target feature. Depending on the scale of the target feature, the learning task differs as well as terminology. Specifically, if the target feature is quantitative, a decision tree is referred to as a regression tree, and if the target feature is categorical, a decision tree is referred to as a classification tree. Yet classification trees may differ on the learning task: (a) learning a partition, if the target is nominal, and (b) learning a category. This section focuses only on the task of learning a classification tree with a partitional target.
(c) Given a decision tree, its terminal nodes (leaves) form a partition S, which is considered then against the target feature u with a scoring function measuring the overall correlation $W(S, u)$. This suggests a context of the analysis of correlation between two features, see Sections 3.3 and 3.4. If the target u is quantitative, then a tabular regression of u over S should be analyzed and scored. Unfortunately, in the data mining literature, this natural approach is not appreciated; thus, the most natural scoring function, the correlation ratio, is not popular. In contrast, at a categorical target, two most popular scoring functions, Gini index and Pearson chi-squared, fit perfectly in the framework of contingency tables and Quetelet indexes described in Section 3.4.1.2, as will be shown in this section further on. Moreover, it will be mathematically proven that these

two can be considered as implementations of the same approach of maximizing the contribution to the data scatter of the target categories – the only difference being the way the dummy variables representing the categories are normalized: (i) no normalization to make it Gini index or (ii) normalization by Poissonian standard deviations so that less frequent categories get more important, to make it Pearson chi-squared. This sheds a fresh light on the criteria and suggests the user a way for choosing between the two.

(d) Admissible partitions conventionally are obtained by splitting the entity set corresponding to one of the current terminal nodes over one of the features. To make it less arbitrary, most modern programs do only binary splits. That means that any node may be split only in two parts: (i) that corresponding to a category and the rest, for a categorical feature or (ii) given an *a*, those "less than *a*" and those "greater than *a*", for a quantitative feature. This text attends to this approach as well. All possible splits are tested and that producing the largest value of the criterion is actually made, after which the process is reiterated.

(e) Stopping rule typically assumes a degree of homogeneity of sets of entities, that is, clusters, corresponding to terminal nodes and, of course, their sizes: too small clusters are not stable and should be excluded.

(f) Pruning: In some programs, the size of a cluster is unconstrained so that in the process of splitting nodes over features, some split parts may become very small and, thus, unreliable as terminal nodes. This makes it useful to prune the tree after it is computed, usually by merging the small subset nodes into greater agglomerations. This is typically done not according to the splitting criterion *W(S,u)* but according to more local considerations such as testing whether proportions of the target categories in a cluster are similar to those used at the assignment of *u* values to terminal nodes or by removing nodes with small chi-squared values (see, for a review, Esposito et al. 1997).

(g) Assigning a terminal node with a *u* category conventionally is done by just averaging its values over the node entities if *u* is quantitative or according to the maximum probability of an *u* category. Then the quality of quantitative prediction is accessed, as usual, by computing the differences between observed and predicted values of u, and their variance of course. In the nominal target case, this leads to an obvious estimate of the probability of the error: unity minus the maximum probability; these then are averaged over the terminal nodes of the decision tree. To make the error's estimate more robust, cross-validation techniques are used. Consider, say, a ten fold cross validation. The entity set is randomly divided into ten equal-sized subsets. Each of them is used as a testing ground for a decision tree built over the rest: these errors are averaged and given as the error's estimate to the tree built over the entire entity set. These techniques are beyond the scope of the current text.

It should be mentioned that the assignment of a category to a terminal cluster in the tree can be of an issue in some situations: (i) if no obvious winning category occurs in the cluster, (ii) if the category of interest is quite rare, that is, when *u*'s distribution is highly skewed. In this latter case using Quetelet coefficients relating the node proportions with those in the entire set may help by revealing some great improvements in the proportions, thus leading to interesting tendencies discovered (Mirkin 1985).

4.5.3 Measuring Correlation for Classification Trees

P4.5.3.1 Three Approaches to Scoring the Split-to-Target Correlation: Presentation

The process of building a classification tree is, basically, a process of splitting clusters into smaller parts driven by a measure of correlation between the partition S being built and the target feature u. Since our focus here is the case of nominal u's only, the target feature is represented by a partition T which is known to us on the training set.

How to define a function $w(S, T)$ to score correlation between the target partition T and partition S being built? Three possible approaches are:

1. A popular idea is to use a measure of uncertainty, or impurity, of a partition and score the goodness of split S by the reduction of uncertainty achieved when the split is made. If it is Gini index, or nominal variance, which is taken as the measure of uncertainty, the reduction of uncertainty is the popular impurity function utilized in a popular decision tree building program CART (Breiman et al. 1984). If it is entropy, which is taken as the measure of uncertainty, the reduction of uncertainty is the popular Information gain function utilized in another popular decision tree building program C4.5 (Quinlan 1993).
2. Another idea would be to use a popular correlation measure defined over the contingency table between partitions S and T such as Pearson chi-squared. Indeed Pearson chi-squared is used for building decision trees in one more popular program, SPSS (Green and Salkind 2003), as a criterion of statistical independence criterion, though, rather than a measure of association. Yet because Pearson chi-squared is equal to the summary relative Quetelet index (see Section 3.4.1.2), it is a measure of association, and it is in this capacity that Pearson chi-squared is used in this text. Moreover, both the impurity function and Information gain mentioned above also are correlation measures defined over the contingency table as shown in the formulation part of this section. Indeed, the Information gain is just the mutual information between S and T, a symmetric function, and the impurity function, the summary absolute Qutelet index.
3. One more idea comes from the discipline of analysis of variance in statistics (see Section 3.3): the correlation can be measured by the proportion of the target feature variance taken into account by the partition S. How come? The variance is a property of a quantitative feature, and we are talking of a target partition here. The trick is that each class of the target partition is represented by the corresponding dummy feature, which is equal to 1 at entities belonging to the class and 0 at the rest. Each of them can be treated as quantitative, as explained in Section 2.3,

so that the summary explained proportion would make a measure of correlation between *S* and *T*. What is nice in this approach, that it is uniform across different types of feature scales: both categorical and quantitative features can be treated the same, which is not the case with other approaches. Although this approach has been advocated by the author for a couple of decades (see, for example, Mirkin 2005), no computational program has come out of it so far. There is a good news though: both the impurity function and Pearson chi-squared can be expressed as the summary explained proportion of the target variance, under different normalizations of the dummy variables course. To get the impurity function (Gini index), no normalization is needed at all, and Pearson chi-squared emerges if each of the dummies is normalized by the square root of its frequency. That means that Pearson chi-squared is underlied by the idea that more frequent classes are less contributing. This might suggest the user to choose Pearson chi-squared if they attend to this idea, or, in contrast, the impurity function if they think that the frequencies of target categories are irrelevant to their case.

There have been developed a number of myths about classification tree building programs and correlation scoring functions involved in them. The following comments are purported to shed light on some of them.

Comment 4.1. There is an opinion lurking in some comments on the web that of two popular programs, CART (Breiman et al. 1984) and CHAID (Green and Salkind 2007), the former is more oriented at prediction whereas the latter, at description. The reason for this perhaps can be traced to the fact that CART involves the impurity function that is defined as the reduction in uncertainty whereas CHAID involves Pearson chi-squared as a measure of the deviation from statistical independence. Yet this opinion is completely undermined by the fact that the measures have very similar predictive powers shaped as the summary Quetelet indexes, the only difference being that one of them involves the relative Quetelet indexes, and the other abosolute ones (see Statements 4.5.2.1 (b) and 4.5.2.2 (b)).

Comment 4.2. The difference between impurity function and Pearson chi-squared amounts to just different scaling options for the dummy variables representing classes of the target partition *T* (see items (c) in Statements 4.5.2.1 and 4.5.2.2). The smaller *T* classes get rescaled to larger values, thus contributing more, when using Pearson chi-squared.

Comment 4.3. Pearson chi-squared introduced to measure the deviation of a bivariate distribution from the statistical independence appears also to signify a purely geometric concept, the contribution to the data scatter (see (a) and (c) in Statement 4.5.2.2 on p. 150). This leads to a different advice regarding the zeros in a contingency table. According to classical statistics, the presence of zeros in a contingency table contradicts the hypothesis of statistical independence so that the data are to be trimmed to avoid zeros. However, in the context of data scatter decompositions, the chi-squared is just a contribution with no statistical independence involved so that the presence of zeros is of no issue in this context: thus, no data trimming is needed.

F4.5.3.2 Scoring Functions for Classification Trees: Formulation

Conventional Definitions and Quetelet Coefficients

Consider an entity set I with a pre-specified partition $T = \{T_l\}$ – which can be set according to categories l of a nominal feature – that is to be learnt by producing a classification tree. At each step of the tree building process, a subset $J \subseteq I$ is to be split into a partition $S = \{S_k\}$ in such a way that S is as close as possible to *T(J)* which is the overalp of T and J. The question is: how the similarity between S and *T(J)* is to be measured? When $S = T(J)$, there is no confusion between the two. Otherwise, it is the contingency table (see Section 3.3) between S and $T(J)$, $P = (p_{kl})$ where p_{kl} is the proportion of J- entities in $S_k \cap T_1$, that expresses the confusion, which is why it is frequently referred to as a confusion table in this context.

One idea for assessing the extent of similarity is to use a correlation measure over the contingency table such as averaged Quetelet coefficients, Q and A, or chi-squared X^2, as discussed in Section 3.4.1.2.

Seemingly another idea is to score the extent of reduction of uncertainty over *T(J)* obtained when S becomes available. This idea works like this: take a measure of uncertainty of a feature, in this case partition *T(J)*, $\upsilon(T(J))$, and evaluate it at each of S-classes, $\upsilon(T(S_k))$, $k = 1, \ldots, K$. Then the average uncertainty on these classes will be $\sum_{k=1}^{K} p_{k+}\upsilon(T(S_k))$, where p_{k+} are proportions of entities in classes S_k, so that the reduction of uncertainty is equal to

$$\upsilon(T(J)/S) = \upsilon(T(J)) - \sum_{k=1}^{K} p_{k+}\upsilon(T(S_k)) \tag{4.13}$$

Of course a function like (4.13) can be considered a measure of correlation over the contingency table P as well, but a nice feature of this approach is that it can be extended from nominal features to quantitative ones – just with an uncertainty index over quantitative T-features (see Q.4.11)

Two very popular measures defined according to (4.13) are so-called impurity function (Breiman at al. 1984) and information gain (Quinlan 1993).

The impurity function builds on Gini coefficient as a measure of variance (see Section 1.3). Let us recall that Gini index for partition T is $G(T) = 1 - \sum_{l=1}^{L} p_l^2$ where p_l is the proportion of entities in T_l. If J is partitioned in clusters $S_k, k = 1, \ldots, K$ and S form a contingency table of relative frequencies $P = (p_{kl})$. Then the reduction (4.13) of the value of Gini coefficient due to partition S is equal to $\Delta(T(J), S) = G(T(J)) - \sum_k p_k G(T(S_k))$. This index $\Delta(T(J), S)$ is referred to as impurity of S over partition T. The greater the impurity the better the split S.

It is not difficult to prove that $\Delta(l, S)$ relates to Quetelet indexes from Section 3.3. Indeed, $\Delta(T, S) = A(T, S)$ where $A(T, S)$ is the summary absolute Quetelet index defined by Equation (3.22) in Q.3.24. Indeed, $\Delta(T, S) = G(T) - \sum_k p_k G(T(S_k) = 1 - \sum_l p_{+l}^2 - \sum_k (p_{k+} - \sum_l p_{kl}^2/p_{k+})$, where p_{+l} is the proportion of l-th category (class) in set J. This implies indeed that $\Delta(T, S) = \sum_l p_{kl}^2/p_{k+} - \sum_l p_{+l}^2$, which proves the statement.

The information gain function builds on entropy as a measure of uncertainty (see Section 2.3). Let us recall that entropy of partition T is $H(T) = -\sum_{l=1}^{L} p_l \log(p_l)$ where p_l is the proportion of entities in T_l. If J is partitioned in clusters S_k, $k = 1, \ldots, K$, partitions T and S form a contingency table of relative frequencies $P = (p_{kl})$. Then the reduction (4.13) of the value of entropy due to partition S is equal to $I(T(J), S) = H(T(J)) - \sum_k p_k H(T(S_k))$. This index $I(T(J), S)$ is referred to as the information gain due to S. In fact, it is equal to a popular characteristic of the cross-classification of T and S, the mutual information defined as $I(T, S) = H(T) + H(S) - H(ST)$ where $H(ST)$ is entropy of the bivariate distribution represented by contingency table P. (The J argument is omitted here as irrelevant to the statement.)

Please note that the mutual information is symmetric with regard to S and T, in contrast to the impurity function. To prove the statement let us just put forward the definition of the information gain and use the property of logarithm that $\log(a/b) = \log(a) - \log(b)$:

$$I(T,S) = H(T) - \sum\nolimits_k p_k H(T(S_k)) = H(T) + \sum\nolimits_k p_{k+} \sum\nolimits_l p_{kl} \log(p_{kl}/p_{k+}) =$$
$$= H(T) - \sum\nolimits_k p_{k+} \log(p_{k+}) + \sum\nolimits_{k,l} p_{kl} \log(p_{kl}) = H(T) + H(S) - H(ST),$$

which completes the proof.

The reduction of uncertainty measures are absolute differences that much depend on the measurement scale and, also, on values of $\upsilon(T)$ and $\upsilon(S)$. This is why it can be of advantage to use relative versions of the reduction of uncertainty measures normalized by $\upsilon(T)$ or $\upsilon(S)$ or both. For example, popular program C4.5 (Quinlan 1993) uses the information gain normalized by $H(S)$ and referred to as the information gain ratio.

Confusion Measures as Contributions to the Data Scatter

Once again we consider a nominal feature over an entity set I of cardinality N (in fact, I and N can be changed for any other symbols – these are just notations in this section), represented by partition $T = \{T_l\}$, and a clustering partition $S = \{S_k\}$ designed from available features to approximate T. This time, though, we are not going to use their contingency table $P = (p_{kl})$, to see the co-occurrence frequencies p_{kl} emerging from a different perspective.

Specifically, assign each target class (category) T_l with a binary variable x_l, a dummy, which is just a 1/0 N-dimensional vector whose elements $x_{il} = 1$ if i $\in T_l$ and $x_{il} = 0$, otherwise ($l = 1, \ldots, L$). Use these dummies as quantitative features to build a tabular regression of each over the partition S. Consider, first, the average of x_l within cluster S_k: the number of unities among x_{il} with $i \in S_k$ is obviously N_{kl}, the size of the intersection $S_k \cap T_l$ because $x_{il} = 1$ only if $i \in S_k$. That means that the within S_k average of x_l is equal to $c_{kl} = N_{kl}/N_{k+}$ where N_{k+} stands for the size of S_k, or, p_{kl}/p_{k+} in terms of the relative contingency table P.

Let us now standardize each feature x_l by a scale shift a_l and rescaling factor $1/b_l$, according to the conventional formula $y_l = (x_l - a_l)/b_l$. This will correspondingly change the averages, so that within-cluster averages of standardized features y_l are equal to $c_{kl} = (p_{kl}/p_{k+} - a_l)/b_l$. In mathematical statistics, the issue of standardization is just a routine transforming the probabilistic density function to a standardized format. Things are different in data analysis, since no density function is assigned to data usually. The scale shift is considered as positioning the data against a backdrop of the "norm", whereas the act of rescaling is to balance feature "weights" (see Section 5.1 for discussion). Therefore, choosing the feature means as the "norms" should be reasonable. The mean of feature x_l is obviously the proportion of unities in it, which is p_{+l} in notations related to contingency table P. In fact, the remainder of this section can be considered as another reason for using $a_l = p_{+l}$. The choice of rescaling factors is somewhat less certain, though using all $b_l = 1$ should seem reasonable too because all the dummies are just 1/0 variables measured in the same scale. Incidentally, 1 is the range of x_l. Some other values related to x_l's dispersion could be used as well. With the scale shift value specified, the within cluster average can be expressed as

$$c_{kl} = \frac{p_{kl} - p_{k+}p_{+l}}{p_{k+}b_l}.$$

Let us refer to formula (3.13) on p. 94 in which the feature scatter is presented as the sum of two parts, one explained by partition S and the other unexplained. Using symbolic of this section, the explained part of x_l's scatter can be expressed as $B_l = \sum_{k=1}^{K} N_{k+}{c_{kl}}^2$. This is the sum of contributions of individual clusters. By using (3.17) each of the individual contributions is equal to

$$B_{kl} = N_{k+}\frac{(p_{kv} - p_{k+}p_{+l})^2}{{p_{k+}}^2 b_l^2} = N\frac{(p_{kv} - p_{k+}p_{+l})^2}{p_{k+}b_l^2} \tag{4.14}$$

Accordingly, the total contribution of partition S to the total scatter of the set of standardized dummies representing partition T is equal to

$$B(T/S) = \sum_{k=1}^{K}\sum_{l=1}^{L} B_{kl} = N\sum_{k=1}^{K}\sum_{l=1}^{L}\frac{(p_{kv} - p_{k+}p_{+l})^2}{p_{k+}b_l^2} \tag{4.15}$$

The total contribution (4.15) reminds us of both the averaged relative Quetelet coefficient (3.19) and the averaged absolute Quetelet coefficient (3.22). The latter, up to the constant N of course, emerges at all rescaling factors $b_l = 1$. The former emerges when rescaling factors $b_l = \sqrt{p_l}$. The square root of the frequency has an appropriate meaning – this is a good estimate of the standard deviation in Poisson model of the variable: according to this model, N_{+l} unities are thrown randomly into the fragment of memory assigned for the storage of vector x_l. In fact, at this scaling system, $B\,(T/S) = \mathrm{X}^2$, Pearson chi-squared!

Let us summarize the proven facts (Mirkin 1996).

Statement 4.5.2.1. The impurity function can be equivalently expressed as

(a) The reduction of Gini uncertainty index of partition T when partition S is taken into account;
(b) The averaged absolute Quetelet index $a(l/k) = p_{kl}/p_{k+} - p_{+l}$ of the same effect;
(c) The total contribution of partition S to the summary data scatter of the set of dummy 1/0 features corresponding to classes of T and standardized by subtracting the mean with no rescaling.

Statement 4.5.2.2. The Pearson chi-square function can be equivalently expressed as

(a) A measure of statistical independence between partitions T and S;
(b) The averaged relative Quetelet index $q(l/k) = (p_{kl}/p_{k+} - p_{+l})/p_{+l}$ between partitions T and S;
(c) The total contribution of partition S to the summary data scatter of the set of dummy 1/0 features x_l corresponding to classes T_l and standardized by subtracting the mean and dividing the result by $b_l = \sqrt{p_l}$.

Statement 4.5.2.3. The information gain can be equivalently expressed as

(a) The reduction of entropy of partition T when partition S is taken into account;
(b) The mutual information $H(T) + H(S) - H(TS)$ between T and S;

C4.5.3.3 Computing Scoring Functions with MatLab: Computation

```
function a=gini(p)
  tot=sum(sum(p));
% total
  pr=p/tot;
  rp=sum(pr');
% row sums
  cp=sum(pr);
%column sums
  ps=pr.*pr;
  rps=sum(ps');
  ir=find(rp>0);
  tr=rps(ir)./rp(ir)
  a1=sum(tr);
  a2=sum(cp.*cp);
  a=a1-a2;
  return
```

```
function a=chi(p)
tot=sum(sum(p));
% total
  pr=p/tot;
  rp=sum(pr');
  cp=sum(pr);
  ir=find(rp>0);
 % nonzero rows
  ic=find(cp>0);
 %nonzero columns
  ps=pr.*pr;
  ip=rp'*cp;
  psi=ps(ir,ic);
  ipi=ip(ir,ic);
  tp= psi./ipi;
  a1=sum(sum(tp));
  a=a1-1;
  return
```

```
function a=ing(p)
 p=p+1;
% to avoid zeros
 tot=sum(sum(p));
 pr=p/tot;
 rp=sum(pr');
 cp=sum(pr);
 pl=log2(pr);
 pp=pr.*pl;
 rpp=sum(pp');
 a1=sum(rpp.*rp);
 tp=cp.*log2(cp);
 a2=sum(tp);
 a=a1-a2;
  return
```

Three functions discussed above, Gini index, Pearson chi-squared, and Information gain can be coded as presented in columns of the box above where input p is a contingency matrix. Due to a holistic nature of MatLab computation, it is possible to organize the computation without looping through the matrix elements.The subroutines gini, chi and ing in the box can be considered pseudocodes of the functions for coding in any other language as well.

Q.4.10. Consider the variance as an uncertainty measure for a quantitative feature y. Define the uncertainty reduction measure according to formula (4.13), with T changed for y of course, and prove that it is equal to the numerator of the correlation measure – the part of variance of y explained by its tabular regression over S. **A.** The summary contribution of S to the data scatter is equal to $B = \sum_{k=1}^{K} c_k^2|S_k| = \sum_{i\in I} y_i^2 - \sum_{k=1}^{K} \sigma_k^2|S_k|$ where σ_k^2 is the within-cluster variance of y (see (3.13) in Section 3.3). Then $B = N(\sigma^2 - \sum_{k=1}^{K} p_k\sigma_k^2)$ where σ^2 is the variance of the standardized feature y (note that the mean of y is 0!) and p_k the proportion of entities in cluster S_k. The last equation clearly shows that the explained part of v is $B = N\sigma^2\eta^2$. If y has been z-score standardized so that $\sigma^2 = 1$, B equals the correlation ratio.

Q.4.11. What is the formula of summary contribution B of partition S to the set of dummy features representing partition T when they have been normalized by dividing by their Bernoullian standard deviations $b_l = \sqrt{p_{+l}(1 - p_{+l})}$?

Q.4.12. Consider a partition $\mathrm{S} = \{\mathrm{S_k}\}$ $(\mathrm{k} = 1, 2, \ldots, \mathrm{K})$ on J and a set of categorical features $\mathrm{v} \in \mathrm{V}$, each with a set of categories L(v). The category utility function (Fisher 1987) scores partition S against the feature set according to formula:

$$u(S) = \frac{1}{K}\sum_{k=1}^{K} p_k[\sum_{v\in V}\sum_{l\in L(v)} p(v = l/S_k)^2 - \sum_{v\in V}\sum_{l\in L(v)} p(v = l)^2]$$

The term in the square brackets is the increase in the expected number of attribute values that can be predicted given a class, S_k, over the expected number of attribute values that could be predicted without using the class. The assumed prediction strategy follows a probability-matching approach. According to this approach, entities arrive one-by-one in a random order, and the category l is predicted for them with the frequency reflecting its probability, $P(l/k)$ if the class S_k is known, or $p_k = N_k/N$ if information of the class S_k is not provided. Factors p_k weigh classes S_k according to their sizes, and the division by K takes into account the differences in the numbers of clusters: the smaller the better. Prove that the category utility function *u(S)* is the sum of impurity functions $\Delta(l, s)$ over all features $l \in l$ related to the number of clusters, that is, *u(S)* $=\sum_{l\in L} \Delta(l, S)/K$.

4.5.4 Building Classification Trees

Building of a classification tree is a recursive process: starting from the entire data set, partition a cluster into a number of parts according to one of the features. To make the partitions less arbitrary, only binary splits are involved in most of the update programs. That means that any node may be split only in two parts: (i) that corresponding to a category and the rest, for a categorical feature, or (ii) given a threshold a, those "less than or equal to a" and those "greater than a", for a quantitative feature. This approach naturally comes when the data are preprocessed by "enveloping" categories into the corresponding "quantitative" dummy features, that assign a unity to every object falling into the category, and a zero to all the rest. Indeed, at $a = 0$, such a dummy feature would split the set in two parts – that for the corresponding category and the rest. Given a cluster, the choice of feature and threshold a for doing the split is driven by a correlation scoring function, be it Information gain, Pearson chi-squared, Gini index or anything else.

A cluster is not to be split anymore if it is smaller than a user defined threshold TS (TS = 10 is set further on) or is homogeneous enough. We use two different homogeneity tests: (a) large enough proportion of a target category in the cluster, say, above 80%, and (b) small enough value of the scoring function which is set to be 0.03 for Gini index, 0.08 for Pearson chi-squared, and 0.15 for Information gain. These levels of magnitude reflect the functions' ranges: Gini index is very close to 0 hardly reaching 0.5 at all, Pearson chi-squared, related to N, changes between 0 and 1 because it cannot be greater than the number of split parts minus 1, and Information gain can have larger values when the number of target categories is 3 or more. This sets the stopping conditions.

Worked example 4.3. Classification tree for Iris dataset

At Iris dataset with its three taxa, Iris setosa and Iris versicolor and Iris virginica, taken as target categories, all the three scoring functions – Impurity (Gini) function, Pearson chi-squared and Information gain – lead to the same classification tree, presented on Fig. 4.11.

The tree of Fig. 4.11 was found with program clatree.m, see p. 380 in Appendix. It comprises three leaf clusters: A, consisting of all all 50 Iris setosa specimens; B, containing 54 entities of which 49 are of Iris versicolor and 5 of Iris virginica;

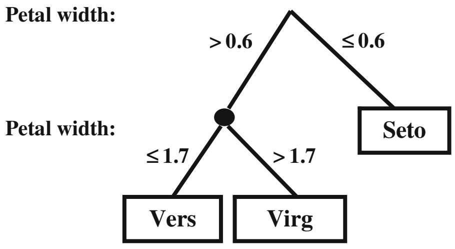

Fig. 4.11 Classification tree for the three-taxon partition at Iris dataset found by using each of Gini, Pearson chi-squared and Information gain scoring functions

Table 4.15 Values of Gini index at the best split of each feature on Iris dataset clusters in Fig. 4.11

	First split		Second split	
Feature	Value	Gini	Value	Gini
w1	5.4	0.228	6.1	0.107
w2	3.3	0.127	2.4	0.036
w3	1.9	0.333	4.7	0.374
w4	0.6	0.333	1.7	0.390

C, containing 46 entities of which 45 are of Iris virginica and 1 of Iris versicolor. Altogether, this misplaces 6 entities leading to the accuracy of 96%. Of course, the accuracy would somewhat diminish if a cross-classification scheme is applied (see Loh and Shih, 1997, who draw a slightly different tree for Iris dataset).

Let us take a look at the action of each variable at each of the two splits in Table 4.15. Each time features w3 and w4 appear to be most contributing, so that at the first split, at which w3 and w4 give the same impurity value, w4 made it through just because it is the last maximum which is remembered by the program.

The tree involves just one feature, w4: Petal width, split twice, first at w4 = 0.6 and then at w4 = 1.7. The Pearson chi-squared value (related to N of course) is 1 at the first split and 0.78 at the second. The Impurity function grows by 0.33 at the first split and 0.39 at the second. The fact that the second value is greater than the first one may seem to be somewhat controversial. Indeed, the first split is supposed to be the best, so that it is the first value that ought to be maximum. Nevertheless, this opinion is wrong: if the first split was at w4 = 1.7 that would generate just 0.28 of impurity value, less than the optimal 0.33 at w4 = 0.6. Why? Because the first taxon has not been extracted yet and grossly contributes to a higher confusion (see the top part in Table 4.16).

Table 4.16 Confusion tables between a split and target partition on Iris dataset

Target partition classes	Iris setosa	Iris versicolor	Iris virginica	Total
Full set				
w4≤1.7	50	49	5	104
w4>1.7	0	1	45	46
Total	50	50	50	150
First cluster removed				
w4≤1.7	0	49	5	54
w4>1.7	0	1	45	46
Total	0	50	50	100

Project 4.1. Prediction of learning outcome at Student data

Consider the Student dataset and ask whether students' learning successes can be predicted from other features available (Occupation, Age, Number of children)? By looking at Table 1.5, it is hardly can be expected that marks can be predicted in this way. Therefore, let us divide students in three groups: I – not so good performers (average mark is less than 50), II – good performers (average mark between 50 and 70 inclusive), and III – excellent performers (average mark higher than 70). To do this, we compute the average mark over the three subjects (SE, OOP, and CI) and create a partition of students *T* as described; the distribution of *T* appears to be I–25, II–58, III–17.

We have a 100×5 matrix X to explore the correlation between X and T, the three columns, $1, 2, 3$, being dummy variables for Occupation categories (IT, BA, AN), column 4 for Age, and column 5 for Number of children. The two conventional stopping criteria, the cluster's size and prevalence of a target class, are not sufficient at this data, because after one or two splits, the program just chips away small fragments of clusters without much improving them. This corresponds to the situations at which the scoring function does not show much improvements either. Therefore, we utilize one more criterion – the minimum value of the scoring function below which there is no splitting. Since the three scoring functions we use have different ranges, the thresholds must be different too. At this study, the threshold is set at 0.03 for Gini index, 0.08 for Pearson chi-squared, and 0.15 for Information gain. The minimum cluster size is taken at 10, and the prevalence of a target class at 80%.

The classification tree found with Gini index is presented on Fig. 4.12. The distributions of target categories in clusters on Fig. 4.12 are presented in Table 4.17. Bold font highlights four terminal clusters as well as high or low proportions of target classes in clusters. High proportions here are those greater than 70% and low proportions are those smaller than 5%.

Tree on Fig. 4.12 is driven by two features: AN Occupation, that structures the set rather well – one split part, those of AN occupation, get more than 70% of category I, and none of category III, and the other of category II. All further divisions are over feature Age; the 12 students in cluster 8 are rather specific – these are of AN occupation aged between 22 and 28, so that 75% of them are in category I, an improvement over parental cluster 6. Cluster 4 of younger not-AN students seems an attempt at drawing a cluster to predict category III – it has a highest jump in its proportion, to 36.4 from 17% in the entire set (cluster 1). The 25 older people

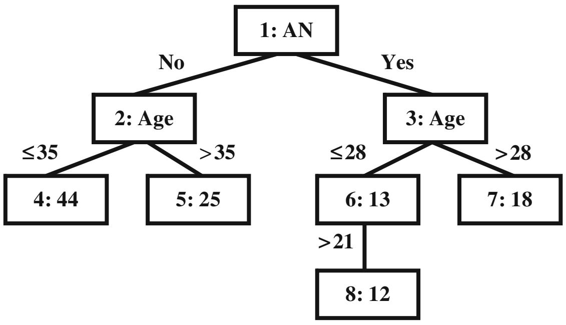

Fig. 4.12 Classification tree on students data targeting partition T of students in three categories found using Gini index. The legend *Number: A* presents, at a split cluster, *A* as the split variable or, at an unsplit cluster, *A* as the size (the number of students in it)

Table 4.17 Distributions of target classes in clusters of tree on Fig. 4.12, per cent

	Clusters in tree on Fig. 4.12							
Target categories	1	2	3	**4**	**5**	6	**7**	**8**
I	25.0	2.9	**74.2**	**2.3**	**4.0**	69.2	**77.8**	**75.0**
II	58.0	**72.5**	25.8	61.4	**92.0**	30.8	22.2	25.0
III	17.0	24.6	**0**	36.4	**4.0**	**0**	**0**	**0**
Gini index at split	0.168	0.046	0.035			0.048		
Cluster size	100	69	31	44	25	13	18	12

among not-AN students are overwhelmingly, 92%, in category II. More splits would have followed if we had decreased the minimum acceptable value of Gini index, say from 0.03 to 0.01.

How well this tree would fare at prediction? To address this question properly, one should either conduct a cross-classification test as explained in the end of Section 4.5.2 or set aside a random testing set before using the rest for building a tree, after which see the levels of errors on the testing set.

Yet for the illustrative purposes, let us calculate the prediction error by using tree on Fig. 4.12. This is done by using the terminal clusters 4, 5, 7, 8 comprising 44, 25, 18, 12 elements, respectively. They total to 99, not 100, because of chipping off an element from cluster 6 to make it into cluster 7. That means: for students in AN category aged 21 or less, no prediction of their learning success level will be made; the classifier takes what is referred to as reject option (comprising approximately 1% of future cases if our sample is representative). According to the data in Table 4.17, the optimal prediction rule would predict then category II at cluster Cluster 4 (with error 100 – 61.4 = 38.6%), category II again, at cluster 5 (with error 100 – 92 = 8%), and category I at clusters 7 and 8 (with errors 22.2% and 25.0%, respectively). The average error is the sum of the individual cluster errors weighted by their relative sizes, (38.6∗44+8∗25+22.2∗18+25∗12)/99 = 26.2%.

What happens, if we use the parental cluster 6 instead of the chipped cluster 8? First thing – no reject option is involved then. Second, the error somewhat increases as should be expected: (38.6∗44 + 8∗25 + 22.2∗18 + 30.8∗13)/100 = 27.0 %.

Figure 4.13 presents trees found by using Pearson chi-squared (a) and Information gain (b). In contrast to Gini index, decreasing the increment threshold does not much help at Information gain: chipping here and there rather than splits will be added. The change of splitting Age value to 30 at cluster 2 on tree (a) does lead to some improvements: the 45 older students are 82.2% in category II. Yet among the 24 younger students, 45.8% belong to category III (leaving 54.2% in category II and 0 in category I).

With this example, one can see that the 90–100% precision is not easy to achieve. That is, a terminal node may have rather modest proportions of target categories, like cluster 5 on Figure 4.13a: about 54% of II category and 46% of III category. Conventional thinking would label the node as an II category predictor because the share of II is greater than half.

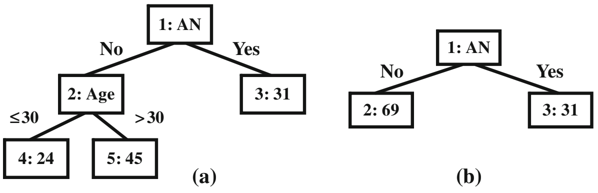

Fig. 4.13 Classification trees on Student dataset targeting partition T of students in three occupation categories found using Pearson chi-squared (**a**) and Information gain (**b**). The legends are of format "Number: A" where "A", at a split cluster, is the split variable or, at an unsplit cluster, the cluster's size

Yet, one should note that, in fact, the proportion 54% is smaller than that, 58%, in the entire set, which means that in fact these conditions, Not_AN and younger age, less than 31, wash out some of II category. It is a case when the style of Quetelet's thinking may produce a better description. This thinking goes beyond proportions in the terminal node and requires comparing the category shares at the node with that in the whole sample. In contrast to a reduction of II category, this cluster boasts a dramatic increase of III category – from 17% in the entire set to 46% in the cluster, 29%. This difference would be picked up by the absolute Quetelet coefficient which is equal to Gini index. Even more dramatic is the relative increase, (45.8–17)/17 = 170%. It is this increase that has been picked up by Pearson chi-squared scoring function, because it is driven by the relative Quetelet coefficient.

Q.4.13. Drawing a lift chart in marketing research. Consider a marketing campaign advertising a product. There is a 1,000 strong sample from the set of targeted customers whose purchasing behavior is known because of prior campaigns. The sample is composed of clusters of a classification tree with different response (that is, purchasing) rates (see Table 4.18). To plan an effective campaign, marketing researchers use what is called a lift chart – a visual representation of the response rate. The x-axis of a lift chart shows the percentiles of the sample, say, from 10 to 100%. On y-axis, the so-called lifts are presented. Given a group of customers, the lift is defined as the ratio of the group's response rate to the baseline response rate, which is the response rate for the entire sample. On the lift chart, the percentiles of the sample are taken in the descending order of the lift. Both baseline and percentile lifts are presented on the chart. Build a lift chart for the sample. **A.** First, we calculate the baseline rate which is the average of the response rates in Table 4.18 weighted by the cluster proportions: $r = 0.1{*}30 + 0.4{*}10 + 0.25{*}4 + 0.25{*}0 = 8\%$.

Table 4.18 Proportions of four clusters in a sample of 1,000 customers and their purchasing behavior (response rate)

Cluster share, %	10	40	25	25
Response rate, %	30	10	4	0

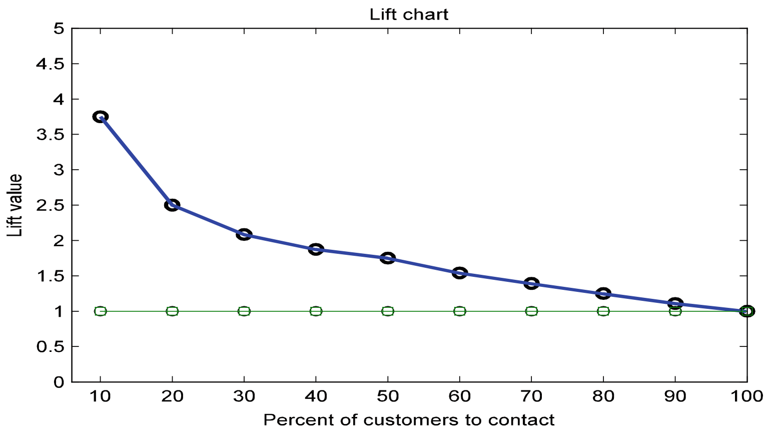

Fig. 4.14 Lift chart for data in Table 4.18

Now we take the most responsive 10% of the customers and calculate their lift value: 30/8 = 3.75. Next, we take the most responsive 20% of the sample, that is the first cluster plus a hundred customers from the second cluster and see their response rate – there should be 30 customers from the first cluster plus 10 from the second who have purchased the product, which gives 40/200 = 20% response rate leading to the lift value of 20/8 = 2.5. Next percentile, 30% of the sample is composed of the first cluster plus 200 customers from the second cluster leading to 50/300 = 17.7% response rate and lift 2.2. In this way, chart presented on Fig. 4.14 is computed.

C4.5.5 Building Classification Trees: Computation

Consider an entity set I along with a nominal target feature represented by partition T of I as well as a set of quantitative input features X (some or all of X-features may be binary dummy variables corresponding to categories). At each step of the process of building a classification tree a cluster $J \subseteq I$ is to be split according to a feature x_v from X in two clusters, S_1 and S_2 so that $S_1 = \{i | i \in J \textit{ and } x_{iv} \leq y\}$ and $S_2 = \{i | i \in J \textit{ and } x_{iv} > y\}$ where y is a value of x_v. The choice of x_v and y is guided by a scoring function *W(S,T)* defined over the contingency table P cross-classifying T by S. That implies that a cluster, as an element of the hierarchical structure being built, should maintain at least the following data: (i) its entity set, (ii) its parental cluster, (iii) feature x_v over which it has been split, (iv) splitting value y, (v) the inequality, $\leq$ or $>$, in the cluster defining predicate. The process starts at the universal cluster consisting of the entire set I. The process stops if either of two conditions holds: (a) $|J| < n$, where n is a pre-specified threshold on the minimum number of entities in a cluster, and (b) if the frequency of a T-cluster is greater than a pre-specified threshold α. To make testing of (b) easier, each cluster should bear one more feature – (vi) the distribution of T in it. One more useful piece of data supplied with a cluster would be (vii) a signal of whether it may or may not be split again.

The recursive nature of the process, as well as the presence of a set of data to accompany each cluster, would make it a fitting subject of an object oriented code. Yet since the object oriented part of MatLab is not quite native in it, a procedural construction will be described in this section. This construction involves two parts, provided that computing scoring function *W(T,S)* over contingency table *P*, has been implemented: (A) finding the best split over a feature, and (B) building a hierarchy of the best splits.

4.5.5.1 Finding the Best Split Over a Feature: Computation

A pseudocode, or MatLab function, msplit.m, takes in a column-feature x, a partition of the set of its indices, t, as a cell array of t-classes, and a string with the name of a scoring method. It produces partition s, the feature splitting value y, and the value of scoring function ma. The stages of computation are annotated within the code.

```
function [g,ma,y]=msplit(x,t,method)
n=length(x);
%-------preparing the set of split value candidates
xv=union(x,x);%set of x values sorted
ll=length(xv);
rl=length(t);
if ll==1 %feature x is constant
    g{1}=[1:n];
    ma=0;
    y=max(x);
else
    for k=1:(ll-1) %loop over splitting values
        f{1}=find(x<=xv(k)); %first split set
        f{2}=setdiff([1:n],f{1}); % the rest
        for ik=1:2; for il=1:rl
              p(ik,il)=length(intersect(f{ik},t{il}));
            end
        end %  contingency table p
        switch method
            case 'gini'
                res=gini(p);
            case 'chi'
                res=chi(p);
            case 'ing'
                res=ing(p);
            otherwise
                disp('The method is wrong ');
                pause(10);
        end
    %---------looking for the best split
        if res>ma
            ma=res;
            g=f;
            y=xv(k);
        end
    end
end
```

4.5.5.2 Organizing a Recursive Split Computation and Storage

The computation is organized in code clatree.m printed in the Appendix p. 380. Here are just a few comments on its structure. Consider a set of ss clusters stored in a cell structure indexed from 1 to ss; in the beginning, the structure stores just the universal cluster *I* and its features at ss = 1. Of these clusters, those in the end, starting from index tt ≥ ss are eligible for splitting. The newly split clusters are indexed by index bb starting from bb = ss+1. (Note that with this system of indexing, there is no need to assign clusters with a label informing that they should not be split anymore: the clusters to split can only be fresh ones!) After split parts are put in the structure, the indices are updated.

There can be a number of stopping criteria that are to be set in the very beginning of the program: it stops when no clusters eligible for splitting remain. In the current version of program clatree.m, three types of stopping criteria are employed. First is the number of entities, TS: a cluster with a smaller number of entities cannot make it into the tree and of course cannot be split further. Second, the dominant proportion of the target classes, ee: a cluster is not split anymore if this has been reached. And the third stopping criterion is tin, a threshold on the scoring function value: if it is less then tin at a split, the cluster is not split.

4.6 Learning Correlation with Neural Networks

4.6.1 General

P4.6.1.1 Artificial Neuron and Neural Network: Presentation

Neural network is one of the most popular structures used for predictions of target features. It is a network of artificial neurons modeling the neuron cell in a living organism. A neuron cell fires an output when its summary input becomes higher than a threshold. Dendrites bring signal in, axons pass it out, and the firing occurs via synapse, a gap between neurons, that makes the threshold (see Fig. 4.15).

This is modeled in an artificial neuron as follows (see Fig. 4.16). A neuron model is drawn as a set of input elements connected to an output. The connections are assigned with wiring weights.

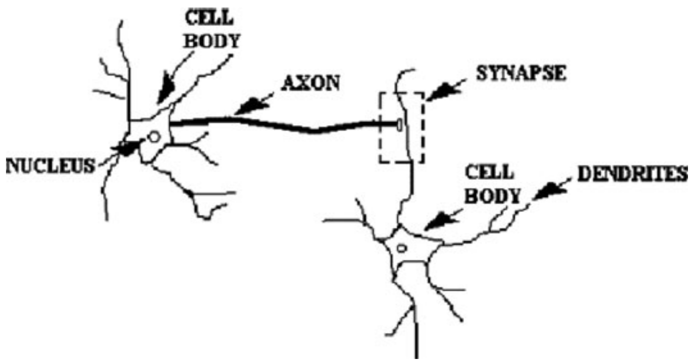

Fig. 4.15 Structure of neuron cells (from http://www.compeng.dit.ie/staff/tscarff/Music_technology/music/neuron_structure.htm visited 17 December 2010)

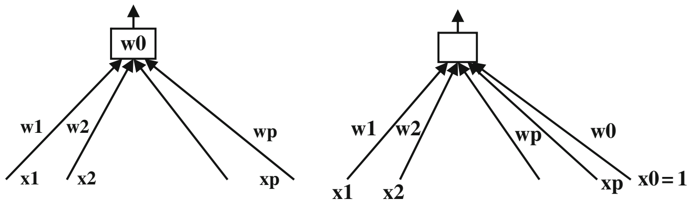

Fig. 4.16 A scheme of an artificial neuron, *on the left*. The same neuron with the firing threshold shown as a wiring weight on the fictitious input always equal to 1 is *on the right*

The input signals are data features or other neurons' outputs. The output element receives a combined signal, the sum of feature values weighted by the wiring weights. The output compares this with a firing threshold, otherwise referred to as a bias, and fires an output depending on the results. Ideally, the output is 1 if the combined signal is greater than the threshold, and –1 if it is smaller. This is, in fact, what is called the sign function of the difference, *sign(x)*, which is 1, 0 or –1 if *x* is positive, zero or negative, respectively. This activation function is overly straightforward sometimes. Instead, the so-called sigmoid and symmetric sigmoid functions are considered as smooth exponent-based counterparts to sign(x). Their graphs are shown alongside with that for *sign(x)* on Fig. 4.17. Sometimes the output element is assumed as doing no transformation at all, just passing the combined signal as the neuron's output, which is referred to as a linear activation function.

The firing threshold, or bias, hidden in the box in neuron on the left on Fig. 4.16, can be made explicit if one more, fictitious, input is added to the neuron.

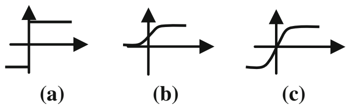

Fig. 4.17 Graphs of sign (**a**), sigmoid (**b**) and symmetric sigmoid (**c**) functions

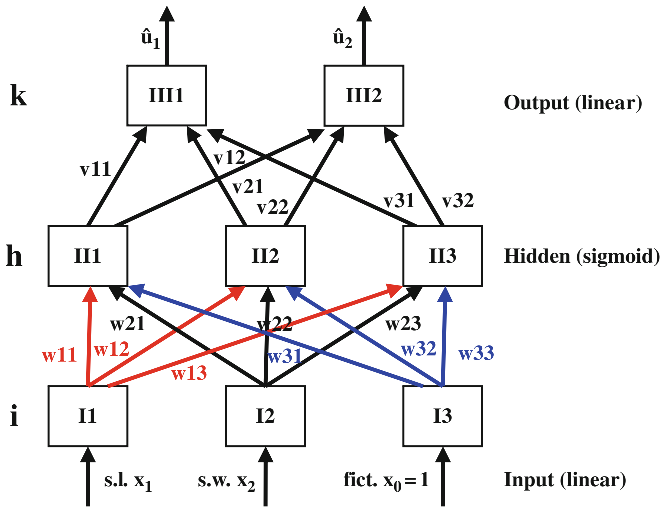

Fig. 4.18 A feed-forward network with two input and two output features (no feedback loops). Layers: Input (I, indexed by *i*), Output (III, indexed by *k*) and Hidden (II, indexed by *h*)

This input is always equal to 1 so that its wiring weight is always added to the combined input to the neuron. It is assumed to be equal to minus the bias so that the total sum is the difference between the combined signal and the bias. In the remainder, we assume that the bias, with the minus sign, is always explicitly present among the wiring weights in this way (see Fig. 4.16 on the right).

Artificial neurons can be variously combined in neural networks. There have been defined many specific types of neural network structures, referred to as architectures, of which the most generic is a three-layer structure with no feedback connections, such as presented on Fig. 4.18 in the next section. There are two outbound layers, the input and output ones, and one intermediate layer which is referred to as a hidden layer. This is why such a structure is referred to as a one hidden-layer neural network (NN).

Network on Fig. 4.18 is designed as a one-hidden-layer NN for predicting petal sizes of Iris features from their sepal sizes. Recall that in Iris data set, each of 150 specimens is presented with four features which are the length and width of petals (features w3 and w4) and sepals (features w1 and w2). It is likely that the sepal sizes and petal sizes are related.

In fact, the further material can be used for building an NN for modeling correlation between any inputs and outputs – the only possible difference, in numbers of input and/or output units, plays no role in the organization of computations.

This neural network consists of the following layers:

(a) Input layer that accepts three inputs: a bias input $x_0 = 1$ as explained above (see Fig. 4.16 on the right) as well as sepal length and width; these are combined to be inputs to each of the neurons at the hidden layer.

(b) Output layer producing an estimate for petal length and width with a linear activation function. Its input is the output signals from the hidden layer. No fictitious input $x_0 = 1$ is assumed here because the activation function here just passes the combined signal through without a threshold.
(c) Hidden layer consisting of three neurons. Each of them takes a combined input from the first layer and applies to it its sigmoid activation function. The output signals of these three neurons constitute inputs to the output layer. The architecture allows for any number of hidden neurons with no changes in the computations.

The one-hidden-layer structure is generic in NN theory. It has been proven, for instance, that such a structure can exactly learn any subset of the set of entities. Moreover, any pre-specified mapping of inputs to outputs can be approximated up to a pre-specified precision with such a one-hidden-layer network, if the number of hidden neurons is large enough (Cybenko 1989).

F4.6.1.2 Activation Functions and Network Function: Formulation

Two popular activation functions, besides the *sign* function $\overset{\circ}{u}_i = sign(\hat{u}_i)$, are the *linear* activation function, $\overset{\circ}{u}_i = \hat{u}_i$ and *sigmoid* activation function $\overset{\circ}{u}_i = s(\hat{u}_i)$ where

$$s(x) = (1 + e^{-x})^{-1} \tag{4.16}$$

is a smooth analogue to the sign function, except for the fact that its output is between 0 and 1 rather than –1 and 1 (see Fig. 4.17b). To imitate the *sign* function, we first double the output interval and then subtract 1 to obtain what is referred to as a symmetric sigmoid or hyperbolic tangent:

$$th(x) = 2s(x) - 1 = 2(1 + e^{-x})^{-1} - 1 \tag{4.17}$$

This function, illustrated on Fig. 4.17c, in contrast to sigmoid *s(x)*, is symmetric: $th(-x) = -th(x)$, like *sign(x)*, which can be useful in some contexts.

The sigmoid activation functions have nice mathematical properties; they are not only smooth, but their derivatives can be expressed through the functions themselves, see Q.4.14 and (4.24).

Let us express now the function of the one-hidden-layer neural network presented on Fig. 4.18. Its wiring weights between the input and hidden layer form a matrix $W = (w_{ih})$, where i denotes an input, and h a hidden neuron, $h = 1, 2, \ldots, H$ where H is the number of hidden neurons. The wiring weights between the hidden and output layers form matrix $V = (v_{hk})$, where h denotes a hidden neuron and k an output.

Layers I and III are assumed to be linear giving no transformation to their inputs; all of the hidden layer neurons will be assumed to have a symmetric sigmoid as their activation function.

To find out an analytic expression for the network, let us work it out layer by layer. Neuron h in the hidden layer receives, as its input, a combined signal

$$z_h = w_{1\,h}x_1 + \ w_{2\,h}x_2 + w_{3\,h}x_0$$

which is h-th component of vector $z = \Sigma_i x_i * w_{ih} = x*W$ where x is a 1×3 input vector. Then its output will be $th(z_h)$. These constitute an output vector $th(z) = th(x*W)$ that is input to the output layer. Its k- th node receives a combined signal $\Sigma_h v_{hk} * th(z_h)$ which is k-th component of the matrix product $th(z)*V$, that is passed as the NN output $\hat{u}$. Therefore, the NN on Fig. 4.18 transforms input x into output $\hat{u}$ according to the following formula

$$\hat{u} = th(x*W)*V \tag{4.18}$$

which combines linear operations of matrix multiplication with a nonlinear symmetric sigmoid transformation. If matrices W, V are known, (4.18) computes the function $u = F(x)$ in terms of th, W, and V. The problem is to fit this model with training data provided, at this instance, by the Iris data set.

4.6.2 Learning a Multi-layer Network

Given all the wiring weights W, between the input and hidden layers, and wiring weights V, between the hidden and output layers, as well as pre-specified hidden layer activation functions, the NN on Fig. 4.18 takes an input of the sepal length and width and transforms it into estimates of the corresponding petal length and width.

The quality of the estimates can be measured by the average squared error. The better adapted weights W and V are, the smaller the error. Where the weights come from? They are learnt from the training data.

Thus the problem is to estimate weight matrices W and V at the training data in such a way that the average squared error is minimized.

The machine learning paradigm is based on the assumption that a learning device adapts itself incrementally by facing entities one by one. This means that the full sample is assumed to be never known to the device so that global solutions, such as the orthogonal projection used in linear discrimination, are not applicable. In such a situation an optimization algorithm that processes entities one by one should be applied. Such is the gradient method, also referred to as the steepest descent.

This method relies on the so-called gradient of the function to be optimized. The gradient is a vector that can be derived or estimated at any admissible solution, that is, matrices W and V. This vector shows the direction of the steepest ascent over the optimized function considered as a surface. Its elements are the so-called partial derivatives of the optimized function that can be derived according to rules of calculus. The gradient is useful for maximizing a criterion, but how one can do minimization with the steepest ascent? Easily, by moving in the opposite direction, that is, taking minus gradient.

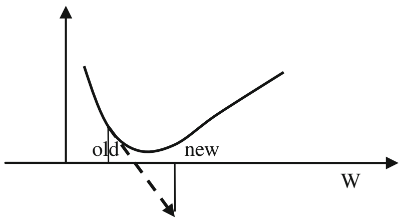

Fig. 4.19 The importance of properly choosing the step in the steepest descent process: if the leap is too big, the new state may be worse than the old one

Assume, we have some estimates of matrices *W* and *V* as well as their gradients, that is, matrices *gW* and *gV*, whose components express the steepest ascent direction of changes in *W* and *V*. Then, according to the method of steepest descent, the matrices V and W should be moved in the direction of –*gW* and –*gV* with the control of the length of the step by a factor referred to as the learning rate. The equations expressing the move from the old state to the new one are as follows:

$$V(new) = V(old) - \mu{*}gV, \;\; W(new) = W(old) - \mu{*}gW \qquad (4.19)$$

where μ is the learning rate (step size). The importance of properly choosing the step size is illustrated on Fig. 4.19.

The gradient of the criterion of squared error is defined by: (a) matrices *W* and *V*, (b) error value itself, and (c) input feature values. This is why it is convenient to apply this approach when entities come in a sequence so that each individual entity gives an estimate of the gradient and, accordingly, the move to a new state of matrices *W* and *V* according to Equation (4.19). The sequence of entities is natural when the learning is done on the fly by processing entities in the order of their arrival. In the situations when all the entities have been already collected in a data set, as the Iris data set, the sequence is organized artificially in a random order. Moreover, as the number of entities is typically rather small (as it is in the case of just 150 Iris specimens) and the gradient process is rather slow, it is usually not enough to process all the entities just once. The processing of all the entities in a random order constitutes *an epoch*. A number of epochs need to be executed until the matrices *V* and *W* are stabilized.

Worked example 4.4. Learning Iris petal sizes

Consider, at any Iris specimen, its two sepal sizes as the input and its two petal sizes as the output. We are going to find a decision rule relating them in the format of a one-hidden-layer NN.

At the Iris data, the architecture presented on Fig. 4.18 and program nnn.m implementing the error back propagation algorithm leads to the average errors at each of the output variables presented in Table 4.19 at different numbers of hidden neurons *h*. Note that the errors are given relative to feature ranges.

The number of elements in matrices V and W here are five-fold of the number of hidden neurons, thus ranging from 15 at the current setting of three hidden neurons

Table 4.19 Relative error values in the predicted petal dimensions with full Iris data after 5,000 epochs

	Relative error, per cent	
Number of hidden neurons	Petal length	Petal width
3	5.36	8.84
6	4.99	8.40
10	4.98	8.15

to 50 when this grows to 10. One can see that the increase in the numbers of hidden neurons does bring some improvement, but not that great – probably not worth doing.

Here are a few suggestions for further work on this example:

1. Find values of E for the errors reported in Table above.
2. Take a look at what happens if the data are not normalized.
3. Take a look at what happens if the learning rate is increased, or decreased, ten times.
4. Extend the table above for different numbers of hidden neurons.
5. Try petal sizes as input with sepal sizes as output.
6. Try predicting only one size over all input variables.

Worked example 4.5. Predicting marks at Student dataset

Let us embark on an ambitious task of predicting students mark at the Students data – we partially dealt with this in Section 4.4. The nnn.m program leads to the average errors in predicting student marks over three subjects, as presented Table 4.20 at different numbers of hidden neurons h. Surprisingly, the prediction works rather well: the errors are on the level of 3 points only, more or less independently on the number of hidden neurons utilized.

F4.6.2.1 Fitting Neural Networks and Gradient Optimization: Formulation

Steepest Descent for the Square Error Criterion with Linear Rules

In machine learning, the assumption is that the decision rule is learnt incrementally by using entities one by one. That is, the global solutions involving the entire sample

Table 4.20 Average absolute error values in the predicted student marks over all three subjects, with full Student data after 5,000 epochs

H	|e1|	|e2|	|e3|	# param.
3	2.65	3.16	3.17	27
6	2.29	3.03	2.75	54
10	2.17	3.00	2.64	90

are not applicable. In such a situation an optimization algorithm that processes entities one by one should be applied. The most popular is the gradient method, also referred to as the steepest descent.

This method relies on the gradient of the function to be optimized. If we are to minimize function *f(x)* over *x* spanning a subspace *D* of the *n*-dimensional vector space R^n, we can utilize its gradient *gf* for this purpose. The gradient *gf* at $x \in D$ is a vector consisting of the *f*'s partial derivatives over all components of *x*, under the assumption that a full derivative, geometrically corresponding to the tangential hyperplane, does exist. This vector shows the direction of the steepest ascent of *f(x)*, so that its opposite vector –*gf* shows the opposite direction which is considered as that of the steepest descent of *f(x)*. The method of steepest descent produces a sequence of points $x(0), x(1), x(2), \ldots$ starting from an arbitrary $x(0)$ by using recursive equation

$$X(t+1) = x(t) - \mu_t * gf(x(t)) \tag{4.19'}$$

where parameter μ_t denotes the length of the step to go from *x(t)* in the direction of the steepest descent, referred to as the learning rate in machine learning. The sequence *x(t)* is guaranteed to converge to the minimum point at a constant $\mu_t = \mu$ if *f(x)* is strictly convex, so that there is a sphere of a finite radius such that *f(x)* is always greater than its lower part, as shown on the right of Fig. 4.20 (see B. Polyak 1987).

The process converges if *f(x)* is a convex function and μ_t tends to *0* when *t* grows to infinity, but not too fast so that the sum of the series $\Sigma_t \mu_t$ is infinity. This guarantees that the moves from *x(t)* to $x(t+1)$ are small enough to not over-jump the point of minimum but not that small to stop the sequence short of reaching the optimum by themselves.

If *f(x)* is not convex however, the sequence reaches just one of the local optima depending on the starting point *x(0)* (see Fig. 4.21). Luckily, the square error in

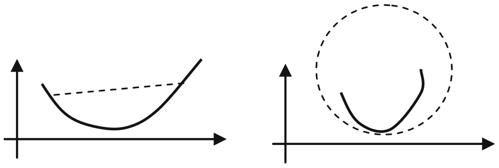
Fig. 4.20 A convex function, *on the left* and strictly convex function, *on the right*

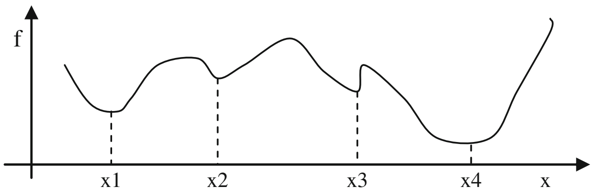

Fig. 4.21 Points x1 to x4 are points of local minimum for the function whose graph is drawn with the *line*. The global minimum is only one of them, x4

the problem of linear discriminant analysis is strictly convex so that the steepest descent sequence converges to the optimum from any initial point. This gives rise to the algorithm described in the following section.

Learning Wiring Weights with Error Back Propagation

The problem of learning a neuron network is to find weight matrices W and V minimizing the squared difference between u observed and $\hat{u}$ computed according to (4.20):

$$E = d(u, \hat{u}) = < u - th(x{*}W){*}V,\ u - th(x{*}W){*}V> /2 \tag{4.20}$$

over the training entity set. The division by 2 is made to avoid factor 2 in the derivatives of E that has been already encountered in Section 4.4.2.2 (see Haykin 1999).

Specifically, with just two outputs on Fig. 4.18, the error function is

$$E = [(u_1 - \hat{u}_1)^2 + (u_2 - \hat{u}_2)^2]/2 \tag{4.20$'$}$$

where $u_1 - \hat{u}_1$ and $u_2 - \hat{u}_2$ are differences between the actual and predicted values of the two outputs.

Steepest descent Equation (4.19) for learning V and W can be written component-wise:

$$v_{hk}(t+1) = v_{hk}(t) - \mu{*}\partial E/\partial v_{kh}, w_{ih}(t+1) = w_{ih}(t) - \mu{*}\partial E/\partial w_{ih}\ (i \in I,\ h \in II,\ k \in III) \tag{4.21}$$

To make these computable, let us express the derivatives explicitly; first those at the output, over v_{hk}:

$$\partial E/\partial v_{hk} = -(u_k - \hat{u}_k){*}\partial \hat{u}_k/\partial v_{hk}$$

To advance, notice that $\partial \hat{u}_k/\partial v_{hk} = th\,(zh)$, since $\hat{u}_k = \Sigma_j th\,(zh)\,{*}v_{hk}$. Putting this into equation above makes

$$\partial E/\partial v_{hk} = -(u_k - \hat{u}_k){*}th\,(zh)\,. \tag{4.22}$$

Regarding the second layer, of W, let us find the derivative $\partial E/\partial w_{ih}$ which requires more chain based derivations. Specifically,

$$\partial E/\partial w_{ij} = \Sigma_k[-(u_k - \hat{u}_k){*}\partial \hat{u}_k/\partial w_{ij}].$$

Since $\hat{u}_k = \Sigma_j th(\Sigma_i x_i{*}w_{ij}){*}v_{jk}$, this can be expressed as

$$\partial \hat{u}_k/\partial w_{ij} = v_{jk}{*}th'(\Sigma_i x_i{*}w_{ij}){*}x_i.$$

The derivative $th'(z)$ can be expressed through $th(z)$ as explained in Q.4.14 later, which leads to the following final expression for the partial derivatives:

$$\partial E/\partial w_{ij} = -\Sigma_k[(u_k - \hat{u}_k)*v_{jk}]*(1 + th(z_j))(1 - th(z_j))*x_i/2 \tag{4.23}$$

Equations (4.19), (4.22) and (4.23) lead to the following rule for processing an entity, or instance, in the error back-propagation algorithm as applied to neural network on Fig. 4.18.

1. *Forward computation (of the output $\hat{u}$ and error).* Given matrices V and W, upon receiving an instance (x, u), the estimate $\hat{u}$ of vector u is computed according to the neural network as formalized in Equation (3.18), and the error $e = u - \hat{u}$ is calculated.
2. *Error back-propagation (for estimation of the gradient elements).* Each neuron receives the relevant error estimate, which is
   $-e_k = -(u_k - \hat{u}_k)$ for (4.22) for output neurons $k(k = III1, III2)$ or
   $-\Sigma_k[(u_k - \hat{u}_k)*v_{hk}]$, for (4.23) for hidden neurons $h(j = II1, II2, II3)$
   [the latter can be seen as the sum of errors arriving from the output neurons according to the corresponding wiring weights].
   These are used to adjust the derivatives (4.22) and (4.23) by multiplying them with local data depending on the input signal, which is $th(z_h)$, for neuron k's source h in (4.22), and $th'(z_h)x_i$ for neuron h's source i in (4.23).
3. *Weights update.* Matrices V and W are updated according to formula (4.19).

What is nice in this procedure is that the computation can be done locally, so that every neuron processes only the data that are available to this neuron, first from the input layer, then backwards, from the output layer. In particular, the algorithm does not change if the number of hidden neurons is changed from $h = 3$ on Fig. 4.18, to any other integer $h = 1, 2, \ldots$; nor it changes if the number of inputs and/or outputs changed.

C4.6.2.2 Error Back Propagation: Computation

For a data set available as a whole, "offline", due to the specifics of the binary target variables and activation functions, such as *th(x)* and *sign*(x), which have –1 and 1 as their boundaries, the data in the NN context are frequently pre-processed to make every feature's range to lie between –1 and 1 and the midrange to be 0. This can be done by using the conventional shifting and rescaling formula for each feature v, $y_{iv} = (x_{iv} - a_v)/b_v$, at which b_v is equal to the half-range, $b_v = (M_v - m_v)/2$, and shift coefficient a_v, to the mid-range, $a_v = (M_v + m_v)/2$. Here M_v denotes the maximum and m_v the minimum of feature v.

The practice of digital computation, with a limited number of digits used for representation of reals, shows that it is a good idea to further expand the ranges into a [–10, 10] interval by multiplying afterwards all the entries by 10: in this range, digital numbers stored in computer arguably lead to smaller computation errors than in the range [–1, 1] if they are closer to 0.

The implementation of the method of gradient descent for learning neural networks cannot be straightforward because the minimized squared error depends both on the wiring weight matrices *V* and *W* and input/output pairs *(x,u)*, yet there is no way to freely change the latter – the process is bound by the set of observations. This is why the observed pairs (x_i, u_i), the instances, are used as triggers to the steepest descent changes in matrices *V* and *W*. Specifically, given *V* and *W*, the instances are put one by one, in a random order, to see what are the discrepancies between the observed *u* and computed *û*. When all of the instances have been entered, their order is randomly changed and they are ready to be put all over again – this is referred to as a new "epoch". The matrices *V* and *W* are changed either at each (x_i, u_i) instance, using the errors *û–u* locally, or after an epoch, using the accumulated errors.

The error back propagation algorithm, with the local changes of matrices *V* and *W*, can be formulated as follows.

A. Initialize weight matrices $W = (w_{ih})$ and $V = (v_{hk})$ by using random normal distribution N(0, 1) with the mean at 0 and the variance 1.
B. Standardize data to [–10, 10] ranges and 0 averages as described above.
C. Formulate halting criterion as explained below and run a loop over epochs.
D. Randomize the order of entities within an epoch and run a loop of the error back-propagation *instance processing procedure*, below, in that order.
E. If Halt-criterion is met, end the computation and output results: *W, V, û, e,* and *E*. Otherwise, execute D again.

The best halting criterion, according to the nature of the steepest descent process should be at

(i) Matrices *V* and *W* stabilized. Unfortunately, in real world computations this criterion requires by far too many iterations, so that in practice the matrices fail to converge. Thus, other stopping criteria are used.
(ii) The difference between the average values (over iterations within an epoch) of the error function becomes smaller than a pre-specified threshold, such as 0.0001.
(iii) The number of epochs performed reaches a pre-specified threshold such as 5,000.

Instance Processing Procedure

Specifics of the NN structure and function provide for simple and effective rules for processing individual entities in the procedure of the steepest descent. Before updating the wiring weights according to Equation (4.19), two following steps are executed:

1. *Forward computation of the estimated output and its error.* Upon receiving a training instance input feature values, they are processed by the neuron network to produce an estimate of the output, after which the error is computed as the difference between real and estimated output values.

2. *Error back-propagation for estimation of the gradient.* The computed error of the output is back-propagated through the network. Each neuron of the output layer corresponds to a specific output feature and, thus, receives the error in this feature. Each neuron of the hidden layer receives a combined error signal from all output neurons weighted by the corresponding wiring weights. These are used to adjust the gradient elements by using the hidden neuron activation function as described in section “Learning Wiring Weights with Error Back Propagation”.

In the Appendix A4, a Matlab code nnn.m is presented for learning NN weights with the error back propagation algorithm according to the NN of Fig. 4.18. Two parameters of the algorithm, the number of neurons in the hidden layer and the learning rate, are its input parameters. The output is the minimum level of error achieved and the corresponding weight matrices V and W.

The code includes the following steps:

1. *Loading data.* It is assumed that all data are in subfolder Data. According to the task, this can be either iris.dat or stud.dat or any other dataset.
2. *Normalizing data.* This is done by shifting each column to its midrange with the follow-up dividing it by the half-range, after which all data set is multiplied by 10, to have them in [–10, 10] scale as described above.
3. *Preparing input and output* training sub-matrices. This is done after the decision has been made of what features fall in the former and what features fall in the latter categories. In the case of Iris data, for example, the target is petal data (features w3 and w4) and input is sepal measurements (features w1 and w2) as described. In the case of Students data, the target can be students’ marks on all three subjects (CI, SP and OOP), whereas the other variables (occupation categories, age and number of children), input.
4. *Initializing the network.* This is done by: (a) specifying the number of hidden neurons H, (b) filling in matrices W and V with random (0,1) normally distributed values, and (c) setting a loop over epochs with the counter initialized at zero.
5. *Organizing a loop over the entities.* For setting a random order of entities to be processed, the Matlab command randperm(n) for making a random permutation of integers 1, 2,. . ., n can be used.
6. *Forward pass*. Given an entity, the output is calculated, as well as the error, using the current V, W and activation functions. The program uses the symmetric sigmoid (4.17) as the activation function of hidden neurons.
7. *Error back-propagation*. Gradient matrices for V and W according to formulas (3.22) and (3.23) are computed.
8. *Weights V and W update.* Having the gradients computed and learning rate accepted as the input, updated W and V are computed according to (4.19).
9. *Halt-condition.* This includes both the level of precision, say 0.01, and a threshold to the number of epochs, say, 5,000. If either is reached, the program halts.

Q.4.14. Prove that the derivatives of sigmoid (4.16) or hyperbolic tangent (4.17) functions appear to be simple polynomials of selves. Specifically,

$$s'(x) = \left(\left(1 + e^{-x}\right)^{-1}\right)' = (-1)\left(1 + e^{-x}\right)^{-2}(-1)\,e^{-x} = s(x)\,(1 - s(x))\,, \tag{4.24}$$

$$th'(x) = [2{*}s(x) - 1]' = 2{*}s(x)' = 2{*}s(x){*}(1 - s(x)) = (1 + th(x)){*}(1 - th(x))/2 \tag{4.24$'$}$$

Q.4.15. Find a way to improve the convergence of the process, for instance, with adaptive changes in the step size values.

Q.4.16. Use k-fold cross validation to provide estimates of variation of the results regarding the data change.

Q.4.17. Develop a scoring function for learning a category by using the contribution of the partition to be built to the category.

4.7 Summary

The goal of this chapter is to present a variety of techniques for learning correlation from data. Most popular concepts – Bayes classifiers, decision trees, neuron networks and support vector machine – are presented along with more generic linear regression and discrimination. Some of these are accompanied with concepts that are interesting on their own such as the bag-of-words model or kernel. The description, though, is rather fragmentary, except perhaps the classification trees for which a number of theoretical results is invoked to show their firm relations to bivariate analysis, first, summary Quetelet indexes in contingency tables and, second, normalization options for dummy variables representing target categories.

Overall, the chapter contents reflect the current state of the art on the subject of learning correlations from data. Perhaps the subject is too big and major advances are a matter of future rather than the past.

References

Breiman, L., Friedman, J.H., Olshen, R.A., Stone, C.J.: Classification and Regression Trees. Wadswarth, Belmont, CA (1984)

Bring, J.: How to standardize regression coefficients. Am. Stat. **48**(3), 209–213 (1994)

Cybenko, G.: Approximation by superposition of sigmoidal functions. Math. Control Signals Systems **2**(4), 303–314 (1989)

Duda, R.O., Hart, P.E., Stork D.G.: Pattern Classification. Wiley-Interscience, New York, NY, ISBN 0-471-05669-3 (2001)

Esposito, F., Malerba, D., Semeraro, G.: A comparative analysis of methods for pruning decision trees. IEEE Trans. Pattern Anal.Mach. Intell. **19**(5), 476–491 (1997)

Fawcett, T.: An introduction to ROC analysis. Pattern Recognition Letters **27**, 861–874 (2006)

Fisher, R.A.: The use of multiple measurements in taxonomic problems. Annu. Eugen. **7**, Part II, 179–188 (1936); also in “Contributions to Mathematical Statistics” (Wiley, New York, NY, 1950)

Fisher, D.H.: Knowledge acquisition via incremental conceptual clustering. Machine Learning, **2**, 139–172 (1987)

Green, S.B., Salkind, N.J.: Using SPSS for the Windows and Mackintosh: Analyzing and Understanding Data. Prentice Hall, Upper Saddle River, NJ (2003)

Groenen, P.J.F., Nalbantov, G.I., Bioch, J.C.: SVM-Maj: A majorization approach to linear support vector machines with different hinge errors. Adv. Data Anal. Classification **2**(1), 17–43 (2008)

Grünwald, P.D.: The Minimum Description Length Principle, MIT Press (2007)

Haykin, S. S.: Neural Networks, 2nd edn. Prentice Hall, ISBN 0132733501 (1999)

Loh, W.-Y., Shih, Y.-S.: Split selection methods for classification trees. Stat. Sin. **7**, 815–840 (1997)

Manning, C.D., Raghavan, P., Schütze, H.: Introduction to Information Retrieval. Cambridge University Press, Cambridge, England (2008)

Mirkin, B.: Methods for Grouping in Socioeconomic Research. Finansy i Statistika Publishers, Moscow (in Russian) (1985)

Mirkin, B.: Mathematical Classification and Clustering. Kluwer Academic Press, Dordrecht/Boston (1996)

Mirkin, B.: Clustering for Data Mining: A Data Recovery Approach. Chapman & Hall/CRC, London, ISBN 1-58488-534-3 (2005)

Mitchell, T.M.: Machine Learning. McGraw Hill, New York, NY (2010)

Polyak, B.: Introduction to Optimization. Optimization Software, Los Angeles, CA, ISBN: 0911575146 (1987)

Quinlan, J.R.: C4.5: Programs for Machine Learning. Morgan Kaufmann, San Mateo, CA (1993)

Schölkopf, B., A.J. Smola, A.J.: (2005) Learning with Kernels. The MIT Press, Cambridge, MA (2005)

Vapnik, V.: Estimation of Dependences Based on Empirical Data, 2d edn. Springer Science + Business Media Inc. (2006)