Since $\frac{p_i}{1-p_i}$ takes values between 0 and ∞, $\log(\frac{p_i}{1-p_i})$ takes values between $-\infty$ and $+\infty$. Taking exponentials on both sides of (4.31) leads to the equation

$$p_i = \frac{e^{\mathbf{w}\cdot\mathbf{x}}}{1+e^{\mathbf{w}\cdot\mathbf{x}}}. \tag{4.32}$$

This is the logistic regression assumption. It is interesting to note that if we assume that the distribution of the characteristic values of the goods and of the bads is multivariate normal, as was suggested in section 4.2, then this example satisfies the logistic regression assumption. Again assume that the means are $\boldsymbol{\mu}_G$ among the goods and $\boldsymbol{\mu}_B$ among the bads with common covariance matrix $\boldsymbol{\Sigma}$. This means that $E(X_i|G) = \mu_{G,i}$, $E(X_i|B) = \mu_{B,i}$, and $E(X_iX_j|G) = E(X_iX_j|B) = \Sigma_{ij}$.

The corresponding density function in this case as given in (4.15) is

$$f(\mathbf{x}|G) = (2\pi)^{-\frac{p}{2}}(\det\Sigma)^{-\frac{1}{2}}\exp\left(\frac{-(\mathbf{x}-\boldsymbol{\mu}_G)\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu}_G)^{\mathbf{T}}}{2}\right), \tag{4.33}$$

where $(\mathbf{x}-\boldsymbol{\mu}_G)$ is a vector with 1 row and p columns while $(\mathbf{x}-\boldsymbol{\mu}_G)^{\mathbf{T}}$ is its transpose, which is the same numbers represented as a vector with p rows and 1 column. If p_B is the proportion of the population who are bad and p_G is the proportion of the population who are good, then the log of the probability odds for customer i who has characteristics $\mathbf{x}$ is

$$\begin{aligned}\log\left(\frac{p_i}{1-p_i}\right) &= \log\left(\frac{p_G f(\mathbf{x}|G)}{p_B f(\mathbf{x}|B)}\right)\\ &= \mathbf{x}\cdot\boldsymbol{\Sigma}^{-1}2(\boldsymbol{\mu}_B-\boldsymbol{\mu}_G)^{\mathbf{T}} + (\boldsymbol{\mu}_G\cdot\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_G^{\mathbf{T}} + \boldsymbol{\mu}_B\cdot\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_B^{\mathbf{T}}) + \log\left(\frac{p_G}{p_B}\right).\end{aligned} \tag{4.34}$$

Since this is a linear combination of the x_i, it satisfies the logistic regression assumption. However, other classes of distribution also satisfy the logistic assumption, including ones that do not lead to linear discriminant functions if the Bayes loss approach of section 4.2 is applied. Consider, for example, the case where the characteristics are all binary and independent of each other. This means that

$$\begin{aligned}&\text{Prob}(X_i = 1|G) = p_G(i); \qquad \text{Prob}(X_i = 0|G) = 1-p_G(i);\\ &\text{Prob}(X_i = 1|B) = p_B(i); \qquad \text{Prob}(X_i = 0|B) = 1-p_B(i).\end{aligned}$$

Hence if p_G, p_B are the prior probabilities of goods and bads in the population

$$\text{Prob}(G|\mathbf{x}) = \frac{\text{Prob}(\mathbf{x}|G)p_G}{\text{Prob}(\mathbf{x})} = \frac{\prod_i p_G(i)^{x_i}(1-p_G(i))^{1-x_i}p_G}{\text{Prob}(\mathbf{x})}, \tag{4.35}$$

then

$$\begin{aligned}\log\left(\frac{\text{Prob}(G|\mathbf{x})}{\text{Prob}(B|\mathbf{x})}\right) &= \Sigma_i x_i(\log(p_G(i)) - \log(p_B(i)))\\ &\quad + \Sigma_i(1-x_i)(\log(1-p_G(i)) - \log(1-p_B(i))) + \log\left(\frac{p_G}{p_B}\right)\\ &= \Sigma_i x_i\left(\log\left(\frac{p_G(i)(1-p_B(i))}{p_B(i)(1-p_G(i))}\right)\right) + \Sigma_i\log\left(\frac{1-p_G(i)}{1-p_B(i)}\right) + \log\left(\frac{p_G}{p_B}\right).\end{aligned} \tag{4.36}$$

This is again of the form of (4.31) and hence satisfies the logistic regression assumption.

The only difficulty with logistic regression compared with ordinary regression is that it is not possible to use the ordinary least-squares approach to calculate the coefficients **w**. Instead one has to use the maximum likelihood approach to get estimates for these coefficents. This leads to an iterative Newton–Raphson method to solve the equations that arise. With the power of modern computers this is no problem, even for the large samples that are often available when building credit scorecards.

One of the surprising results is that although theoretically logistic regression is optimal for a much wider class of distributions than linear regression for classification purposes, when comparisons are made on the scorecards developed using the two different methods on the same data set, there is very little difference in their classification. The difference is that linear regression is trying to fit the probability p of defaulting by a linear combination of the attributes while logistic regression is trying to fit $\log(\frac{p}{1-p})$ by a linear combination of the attributes. As Figure 4.3 shows, if one maps a linear shift of p and $\log(\frac{p}{1-p})$, then they are very similar until either p becomes close to 0 or close to 1. In the credit-scoring context, this means that the scores being produced by the two methods are very similar except for those where the probability of default is very low or very high. These are the applicants for whom it should be easy to predict whether they will default. For the more difficult regions of default prediction—around $p = 0.5$—the two curves are very similar. This may explain why there is less variation in the methods than one would have expected.

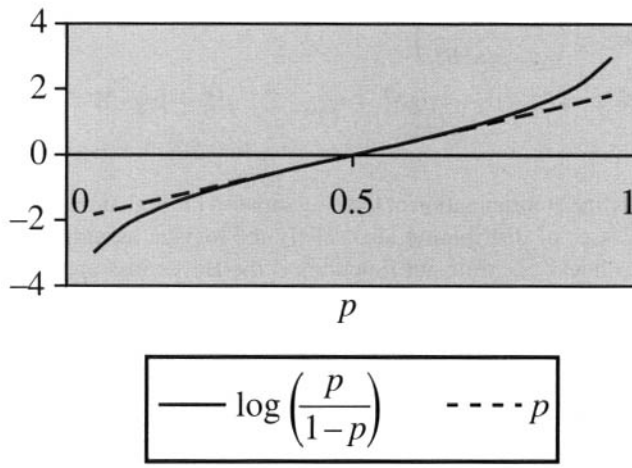


Figure 4.3. *Graph of* $\log(\frac{p}{1-p})$ *and* $ap + b$.

4.6 Other nonlinear regression approaches

Two other nonlinear functions have been used in credit scoring. The first of these is the probit function, which was used in credit scoring by Grablowsky and Talley (1981). In probit analysis, if $N(x)$ is the cumulative normal distribution function so that

$$N(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{y^2}{2}} dy,$$

then the aim is to estimate $N^{-1}(p_i)$ as a linear function of the characteristics of the applicant, so

$$N^{-1}(p_i) = \mathbf{w} \cdot \mathbf{x}_i^{\mathrm{T}} = w_0 + w_1 x_{1i} + w_2 x_{2i} + \cdots + w_p x_{pi}. \tag{4.37}$$

Again, although p_i takes only values between 0 and 1, $N^{-1}(p_i)$ takes values between $-\infty$ and $+\infty$ and so allows the linear function to vary over this full range. One way to think about the probit approach is to say that

$$W = \mathbf{w} \cdot \mathbf{x}_i^{\mathrm{T}} = w_1 X_1 + w_2 X_2 + \cdots + w_p X_p$$

is a measure of the goodness of an applicant, but whether the applicant is actually bad or good depends on whether the value of W is greater or less than a cutoff level C. If we assumed that C is not fixed but is a variable with standard normal distribution, then this gives the same probability of applicants being good as the probit equation (4.37). One can never test this assumption since the W is not a variable with any real meaning. However, one can always fit this model to any sample of past applicants. As in the case of logistic regression, one uses maximum likelihood estimation to obtain the values $\mathbf{w}$ and again one has to use iterative techniques to solve for these values. There are situations where an iteration procedure may not converge, and then alternative iterative procedures must be tried.

The other approach used in credit scoring and that is very common in economic models is tobit analysis. The tobit transformation assumes that one can estimate p_i by

$$p_i = \max\{\mathbf{w} \cdot \mathbf{x}_i^{\mathrm{T}}, 0\} = \max\{w_0 + w_1 x_{1i} + w_2 x_{2i} + \cdots + w_p x_{pi}, 0\}. \tag{4.38}$$

In this case, one is dealing with the mismatch in limiting values between the two sides of the regression approach to discriminant analysis (4.22) by limiting the right-hand side to be positive. Since there are many economic situations where the variable in which one is interested shows itself only when it is positive, the statistical analysis of tobit regressions is fully worked out and many of the standard statistical packages will produce estimates of the parameters. An example of where tobit analysis may be appropriate is if one assumes that for consumers debt = income − expenditure and then one decides to regress debt on income and expenditure. One has to use tobit analysis since negative debt does not show itself as debt.

In the credit-scoring context, there is something unsatisfactory about the asymmetry of the tobit transformation. It has dealt with the difficulty of estimating negative probabilities but not of estimating probabilities greater than 1. A more symmetrical model would be

$$p_i = \mathrm{Min}\{1, \max\{\mathbf{w} \cdot \mathbf{x}_i^{\mathrm{T}}, 0\}\}, \tag{4.39}$$

but unfortunately the analysis and parameter estimation for that transformation is not available in standard packages.

Probit and tobit are not approaches that find much favor with credit-scoring practitioners since they are more concerned with getting the decision correct for the less clear-cut cases than with the fact that the probability for a clear bad is 0.05 or −0.05. However, it has to be remembered that the statistical techniques do not know this, and their objective is to minimize the total sum of the errors. Thus an outlandish case might affect the parameters considerably and hence change the classifications in the difficult region. This suggests one might try a two-stage approach. In the first stage, one tries to estimate the likely cutoff region, while in the second stage, one concentrates on getting the classification in that region to be as accurate as possible.

4.7 Classification trees (recursive partitioning approach)

A completely different statistical approach to classification and discrimination is the idea of classification trees, sometimes called recursive partitioning algorithms (RPA). The idea

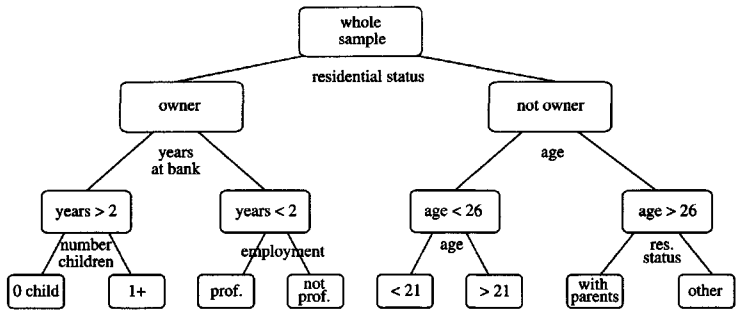


Figure 4.4. *Classification tree.*

is to split the set of application answers into different sets and then identify each of these sets as good or bad depending on what the majority in that set is. The idea was developed for general classification problems independently by Breiman and Friedman in 1973, who described a number of statistical applications (but not credit scoring) in their book (Breiman et al. 1984). Its use in credit scoring quickly followed (Makowski 1985, Coffman 1986). The idea was also picked up in the artificial intelligence literature. Similar ideas were used in classification problems, and useful computer software was developed for its implementation. Although the software has different names—CHAID and C5, for example—the basic steps are the same in both literatures (Safavian and Landgrebe 1991).

The set of application data A is first split into two subsets so that looking at the sample of previous applicants, these two new subsets of application attributes are far more homogeneous in the default risk of the applicants than the original set. Each of these sets is then again split into two to produce even more homogeneous subsets, and the process is repeated. This is why the approach is called recursive partitioning. The process stops when the subsets meet the requirements to be terminal nodes of the tree. Each terminal node is then classified as a member of A_G or A_B and the whole procedure can be presented graphically as a tree, as in Figure 4.4.

Three decisions make up the classification tree procedure:

- what rule to use to split the sets into two—the splitting rule;
- how to decide that a set is a terminal node—the stopping rule;
- how to assign terminal nodes into good and bad categories.

The good-bad assignment decision is the easiest to make. Normally, one would assign the node as good if the majority of sample cases in that node are good. An alternative is to minimize the cost of misclassification. If D is the debt incurred by misclassifying a bad as a good and L is the lost profit caused by misclassifying a good as a bad, then one minimizes the cost if one classifies the node as good when the ratio of goods to bads in that node in the sample exceeds $\frac{D}{L}$.

The simplest splitting rules are those that look just one step ahead at the result of the proposed split. They do this by finding the best split for each characteristic in turn by having some measure of how good a split is. Then they decide which characteristic's split is best under this measure. For any continuous characteristic X_i, one looks at the splits $\{x_i < s\}$, $\{x_i \geq s\}$ for all values of s and finds the value of s where the measure is best. If X_i is a categorical variable, then one looks at all possible splits of the categories into two and checks the measure under these different splits. Usually one can order the categories in increasing

good:bad ratios and know that the best split will divide this ordering into two groups. So what measure is used? The most common is the Kolmogorov–Smirnov statistic, but there are at least four others—the basic impurity index, the Gini index, the entropy index, and the half-sum of squares. We look at each in turn and use them to calculate the best split in the following simple example.

Example 4.1. Residential status has three attributes with the numbers of goods and bads in each attribute in a sample of a previous customer, shown in Table 4.1. If one wants to split the tree on this characteristic, what should the split be?

Table 4.1.

Residential status	Owner	Tenant	With Parents
Number of goods	1000	400	80
Number of bads	200	200	120
Good:bad odds	5:1	2:1	0.67:1

4.7.1 Kolmogorov–Smirnov statistic

For a continuous characteristic X_i, let $F(s|G)$ be the cumulative distribution function of X_i among the goods and $F(s|B)$ be the cumulative distribution among the bads. Assuming bads have a greater propensity for low values of X_i than the goods and that the definitions of the costs D and L are as in the earlier paragraph, the myopic rule would be to split on the value s which minimizes

$$LF(s|G)p_G + D(1 - F(s|B))p_B. \tag{4.40}$$

If $Lp_G = Dp_B$, this is the same as choosing the Kolmogorov–Smirnov distance between the two distributions, as Figure 4.5 shows; i.e., one wants to minimize $F(s|G) - F(s|B)$ or more obviously maximize $F(s|B) - F(s|G)$.

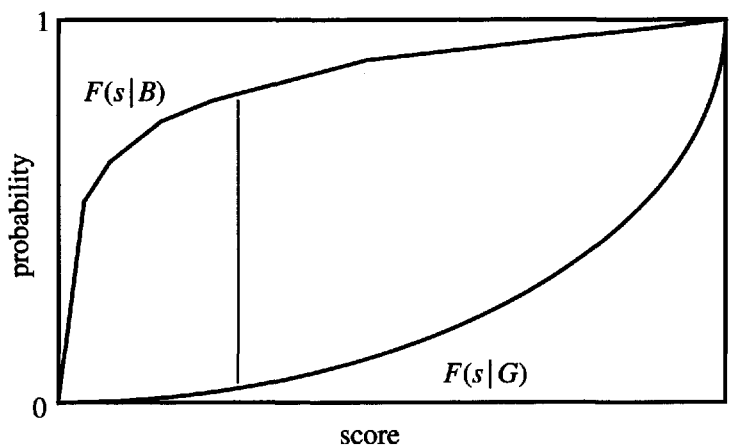


Figure 4.5. *Kolmogorov–Smirnov distance.*

If one thinks of the two subgroups as the left (l) and the right (r) group, then this is like maximizing the difference between $p(l|B)$, the probability a bad appears in the left group (i.e., $F(s|B)$ in the continuous case), and $p(l|G)$, the probability a good appears in the left

group (i.e., $F(s|G)$ in the continuous case). One can rewrite $p(l|B)$ as $\frac{p(B|l)p(l)}{p(B)}$ using Bayes's theorem, and this is easier to calculate. Thus for categorical and continuous variables, the Kolmogorov–Smirnov (KS) criterion becomes the following: Find the left-right split that maximizes

$$\text{KS} = |p(l|B) - p(l|G)| = \left| \frac{p(B|l)}{p(B)} - \frac{p(G|l)}{p(G)} \right| \cdot p(l). \tag{4.41}$$

For Example 4.1, it is obvious from the good:bad odds that the best split must either be with parent in one group and owner + tenant in the other, or tenant and with parent in one group and owner in the other:

l = parent, r = owner + tenant:

$$p(l|B) = \frac{120}{520} = 0.231, \quad p(l|G) = \frac{80}{1480} = 0.054, \quad \text{KS} = 0.177,$$

l = parent + tenant, r = owner:

$$p(l|B) = \frac{320}{520} = 0.615, \quad p(l|G) = \frac{480}{1480} = 0.324, \quad \text{KS} = 0.291.$$

Thus the best split is parent + tenant in one group and owner in the other.

4.7.2 Basic impurity index $i(v)$

There is a whole class of impurity index measures that try to assess how impure each node v of the tree is, where purity corresponds to that node being all of one class. If one then splits the node into a left node l and a right node r with the proportion going into l being $p(l)$ and the proportion going into r being $p(r)$, one can measure the change in impurity that the split has made by

$$I = i(v) - p(l)i(l) - p(r)i(r). \tag{4.42}$$

The greater this difference, the greater the change in impurity, which means the new nodes are much more pure. This is what we want so we choose the split that maximizes this expression. This is equivalent to minimizing $p(l)i(l) + p(r)i(r)$. Obviously, if there was no split with a positive difference, then one should not split the node at all.

The most basic of these impurity indices is to take $i(v)$ to be the proportion of the smaller group in that node so that

$$\begin{aligned} i(v) &= p(G|v) \quad \text{if } p(G|v) \leq 0.5, \\ i(v) &= p(B|v) \quad \text{if } p(B|v) < 0.5. \end{aligned} \tag{4.43}$$

For Example 4.1, this gives the following calculation of the indices to see which is the better split, where v is the whole set before any split:

l = parent, r = owner + tenant:

$$i(v) = \frac{520}{2000} = 0.26, \quad p(l) = \frac{200}{2000} = 0.1, \quad i(l) = \frac{80}{200} = 0.4,$$

$$p(r) = \frac{1800}{2000} = 0.9, \quad i(r) = \frac{400}{1800} = 0.22,$$

$$I = 0.26 - 0.1(0.4) - 0.9(0.22) = 0.02;$$

$$
\begin{aligned}
&l = \text{parent} + \text{tenant}, \quad r = \text{owner}:\\
&i(v) = \frac{520}{2000} = 0.26, \quad p(l) = \frac{800}{2000} = 0.4, \quad i(l) = \frac{320}{800} = 0.4,\\
&p(r) = \frac{1200}{2000} = 0.6, \quad i(r) = \frac{200}{1200} = 0.167,\\
&I = 0.26 - 0.4(0.4) - 0.6(0.167) = 0.
\end{aligned}
$$

This suggests the best split is parent in one group and owner + tenant in the other.

Although this looks useful, appearances can be deceptive. I in the second split is 0 because the bads are the minority in all three nodes v, l, and r. This will always happen if the same group is in the minority in all three nodes, and for many credit situations this is the case. Since all splits give the same difference—0—this criterion is useless to help decide which is the best. Moreover, Breiman et al. (1984) gave an example in which there were 400 goods and 400 bads. One partition split this into one class of 300 goods and 100 bads and the other of 100 goods and 300 bads, while another partition split it into one class of just 200 goods and the other of 200 goods and 400 bads. Both these splits have the same index difference $i(v)$, but most people would think the second partition that has been able to identify a complete group of goods is a better split. What is needed is an index that rewards the purer nodes more than this one does.

4.7.3 Gini index

Instead of being linear in the proportion of the impurity probability, the Gini index is quadratic and so puts relatively more weight on purer nodes. It is defined by

$$
\begin{aligned}
&i(v) = p(G|v)p(B|v),\\
&\text{so} \quad G = p(G|v)p(B|v) - p(l)(G|l)p(B|l) - p(r)(G|r)p(B|r).
\end{aligned}
\tag{4.44}
$$

In the example we are interested in, this gives

$$
\begin{aligned}
&l = \text{parent}, \quad r = \text{owner} + \text{tenant}:\\
&i(v) = \left(\frac{1480}{2000}\right)\left(\frac{520}{2000}\right) = 0.1924,\\
&p(l) = \frac{200}{2000} = 0.1, \quad i(l) = \left(\frac{80}{200}\right)\left(\frac{120}{200}\right) = 0.24,\\
&p(r) = \frac{1800}{2000} = 0.9, \quad i(r) = \left(\frac{400}{1800}\right)\left(\frac{1400}{1800}\right) = 0.1728,\\
&I = 0.1924 - 0.1(0.24) - 0.9(0.1728) = 0.01288;\\
&l = \text{parent} + \text{tenant}, \quad r = \text{owner}:\\
&i(v) = \left(\frac{520}{2000}\right)\left(\frac{1480}{2000}\right) = 0.1924,\\
&p(l) = \frac{800}{2000} = 0.4, \quad i(l) = \left(\frac{320}{800}\right)\left(\frac{480}{800}\right) = 0.24,\\
&p(r) = \frac{1200}{2000} = 0.6, \quad i(r) = \left(\frac{200}{1200}\right)\left(\frac{1000}{1200}\right) = 0.1389,\\
&I = 0.1924 - 0.4(0.24) - 0.6(0.1389) = 0.01306.
\end{aligned}
$$

This index suggests that the best split is parent + tenant in one node and owner in the other node.

4.7.4 Entropy index

Another nonlinear index is the entropy one, where

$$i(v) = -p(G|v)\ln(p(G|v)) - p(B|v)\ln(p(B|v)). \tag{4.45}$$

As its name suggests, this is related to the entropy or the amount of information in the split between the goods and bads in the node. It is a measure of how many different ways one could end up with the actual split of goods to bads in the node, and it is related to the information statistic used to in coarse classifying measurements (see section 8.7 for more details).

Using this measure the splits in Example 4.1 are measured as follows:

l = parent, r = owner + tenant:

$$i(v) = -\left(\frac{520}{2000}\right)\ln\left(\frac{520}{2000}\right) - \left(\frac{1480}{2000}\right)\ln\left(\frac{1480}{2000}\right) = 0.573,$$

$$p(l) = \frac{200}{2000} = 0.1, \quad i(l) = -\left(\frac{80}{200}\right)\ln\left(\frac{80}{200}\right) - \left(\frac{120}{200}\right)\ln\left(\frac{120}{200}\right) = 0.673,$$

$$p(r) = \frac{1800}{2000} = 0.9, \quad i(r) = -\left(\frac{400}{1800}\right)\ln\left(\frac{400}{1800}\right) - \left(\frac{1400}{1800}\right)\ln\left(\frac{1400}{1800}\right) = 0.530,$$

$$I = 0.573 - 0.1(0.673) - 0.9(0.530) = 0.0287;$$

l = parent + tenant, r = owner:

$$i(v) = -\left(\frac{520}{2000}\right)\ln\left(\frac{520}{2000}\right) - \left(\frac{1480}{2000}\right)\ln\left(\frac{1480}{2000}\right) = 0.573,$$

$$p(l) = \frac{800}{2000} = 0.4, \quad i(l) = -\left(\frac{320}{800}\right)\ln\left(\frac{320}{800}\right) - \left(\frac{480}{800}\right)\ln\left(\frac{480}{800}\right) = 0.673,$$

$$p(r) = \frac{1200}{2000} = 0.6, \quad i(r) = -\left(\frac{200}{1200}\right)\ln\left(\frac{200}{1200}\right) - \left(\frac{1000}{1200}\right)\ln\left(\frac{1000}{1200}\right) = 0.451,$$

$$I = 0.573 - 0.4(0.673) - 0.6(0.451) = 0.0332.$$

Hence this index also suggests the best split is with parent + tenant in one node and owner in the other node.

4.7.5 Maximize half-sum of squares

The last measure we look at is not an index but comes instead from the χ^2 test, which would check if the proportion of goods is the same in the two daughter nodes we split into. If this χ^2 statistic Chi is large, we usually say the hypothesis is not true; i.e., the two proportions are not the same. The larger the value, the more unlikely they are the same, but the latter could be reinterpreted to say the greater the difference between them there is. That is what we want from a split, so we are led to the following test.

If $n(l)$ and $n(r)$ are the total numbers in the left and right modes, then maximize

$$\text{Chi} = n(l)n(r) - \frac{(p(G|l) - p(G|r))^2}{n(l) + n(r)}.$$

Applying this test to the data in Example 4.1 gives

$$l = \text{parent}, \quad r = \text{owner} + \text{tenant}:$$

$$n(l) = 200, \quad p(G|l) = \frac{80}{200} = 0.4,$$

$$n(r) = 1800, \quad p(G|r) = \frac{1400}{1800} = 0.777, \quad \text{so Chi} = 25.69;$$

$$l = \text{parent} + \text{tenant}, \quad r = \text{owner}:$$

$$n(l) = 800, \quad p(G|l) = \frac{480}{800} = 0.6,$$

$$n(r) = 1200, \quad p(G|r) = \frac{1000}{1200} = 0.833, \quad \text{so Chi} = 26.13.$$

Hence this measure is maximized when the "with parents" class and the tenants class are put together. One can construct examples where the tests suggest different splits are best.

There are other methods of splitting. Breiman et al. (1984) suggested that a better criterion than just looking at the next split would be to consider what the situation is after r more generations of splits. This takes into account not just the immediate improvement caused by the current split but also the long-run strategic importance of that split. It is usually the case that one is splitting using different characteristics at different levels of the tree, and also different characteristics come to the fore for different subsets at the same level of the tree, as Figure 4.4 illustrates. In this way, the tree picks up the nonlinear relationships between the application characteristics.

Although we talk about when we stop the tree and recognize a node as a terminal node, it is perhaps more honest to talk of a stopping and pruning rule. This emphasizes that we expect the initial tree to be too large and it will need to be cut back to get a tree that is robust. If one had a tree where every terminal node had only one case from the training sample in it, then the tree would be a perfect discriminator on the training sample but it would be a very poor classifier on any other set. Thus, one makes a node a terminal node for one of two reasons. The first reason is the number of cases in the node is so small that it makes no sense to divide it further. This is usually when there are fewer than 10 cases in the node. The second reason is the split measurement value if one makes the best split into two daughter nodes is hardly any different from the measurement value if one keeps the node as is. One has to make "hardly any difference" precise in this context, and it could be that the difference in the measure is below some prescribed level β.

Having obtained such an overlarge tree, one can cut back by removing some of the splits. The best way to do this is to use a holdout sample, which was not used in building the tree. This sample is used to estimate empirically the expected losses for different possible prunings of the tree. Using this holdout sample and a classification tree T, define T_G (T_B) to be the set of nodes that are classified as good (bad). Let $r(t, B)$ be the proportion of the holdout sample that is in node t and that are classified as bad and $r(t, G)$ be the proportion of the sample that is in t and is good. Then an estimate of the expected loss is

$$r(T) = \sum_{t \in T_G} Dr(t, B) + \sum_{t \in T_B} Lr(t, G). \tag{4.46}$$

If $n(T)$ is the number of nodes in tree T, define $c(T) = r(T) + dn(T)$ and prune a tree T^* by looking at all subtrees of T^* and choosing the tree T that minimizes $c(T)$. If $d = 0$, we end up with the original unpruned tree, while as d becomes large, the tree will consist of only one node. Thus the choice of d gives a view of how large a tree is wanted.

4.8 Nearest-neighbor approach

The nearest-neighbor method is a standard nonparametric approach to the classification problem first suggested by Fix and Hodges (1952). It was applied in the credit-scoring contect first by Chatterjee and Barcun (1970) and later by Henley and Hand (1996). The idea is to choose a metric on the space of application data to measure how far apart any two applicants are. Then with a sample of past applicants as a representative standard, a new applicant is classified as good or bad depending on the proportions of goods and bads among the k nearest applicants from the representative sample—the new applicant's nearest neighbors.

The three parameters needed to run this approach are the metric, how many applicants k constitute the set of nearest neighbors, and what proportion of these should be good for the applicant to be classified as good. Normally, the answer to this last question is that if a majority of the neighbors are good, the applicant is classified as good; otherwise, the applicant is classified as bad. However, if, as in section 4.2, the average default loss is D and the average lost profit in rejecting a good is L, then one could classify a new applicant as good only if at least $\frac{D}{D+L}$ of the nearest neighbors are good. This criterion would minimize the expected loss if the likelihood of a new applicant being good is the proportion of neighbors who are good.

The choice of metric is clearly crucial. Fukanaga and Flick (1984) introduced a general metric of the form

$$d(\mathbf{x}_1, \mathbf{x}_2) = (\mathbf{x}_1 - \mathbf{x}_2)\mathbf{A}(\mathbf{x}_1)((\mathbf{x}_1 - \mathbf{x}_2)^{\mathbf{T}})^{\frac{1}{2}}, \tag{4.47}$$

where $\mathbf{A}(x)$ is a $p \times p$ symmetric positive definite matrix. $\mathbf{A}(\mathbf{x})$ is called a local metric if it depends on $\mathbf{x}$ and is called a global metric if it is independent of $\mathbf{x}$. The difficulty with the local metric is that it picks up features of the training set that are not appropriate in general, and so most authors concentrate on global metrics. The most detailed examination of nearest-neighbors approach in the credit-scoring context was by Henley and Hand (1996), who concentrated on metrics that were mixtures of Euclidean distance and the distance in the direction that best separated the goods and the bads. One gets this direction from Fisher's linear discriminant function of section 4.3. Thus if $\mathbf{w}$ is the p-dimensional vector defining that direction, given in (4.21), Henley and Hand suggest a metric of the form

$$d(\mathbf{x}_1, \mathbf{x}_2) = \{(\mathbf{x}_1 - \mathbf{x}_2)^{\mathbf{T}}(\mathbf{I} + D\mathbf{w} \cdot \mathbf{w}^{\mathbf{T}})(\mathbf{x}_1 - \mathbf{x}_2)\}^{\frac{1}{2}}, \tag{4.48}$$

where I is the identity matrix. They perform a large number of experiments to identify what might be a suitable choice of D. Similarly, they choose k, the ideal number of nearest neighbors, by experimenting with many choices of k. Although there are not large variations in the results, the best choice of D was in the range 1.4 to 1.8. The choice of k clearly depends on the size of the training sample, and in some cases changing k by one makes a noticeable difference. However, as Figure 4.6 suggests, in the bigger picture there is not much difference in the rate of misclassifying bads for a fixed acceptance rate as k varies over quite a range from 100 to 1000 (with a training sample of 3000). To avoid picking a locally poor value of k, one could smooth k by choosing a distribution for k. Thus for each point, there may be a different number of nearest neighbors. However, satisfactory results can be obtained without resorting to this level of sophistication.

Nearest-neighbor methods, although they are far less widely in credit scoring than the linear and logistic regression approaches, have some potentially attractive features for actual implementation. It would be easy to dynamically update the training sample by adding new

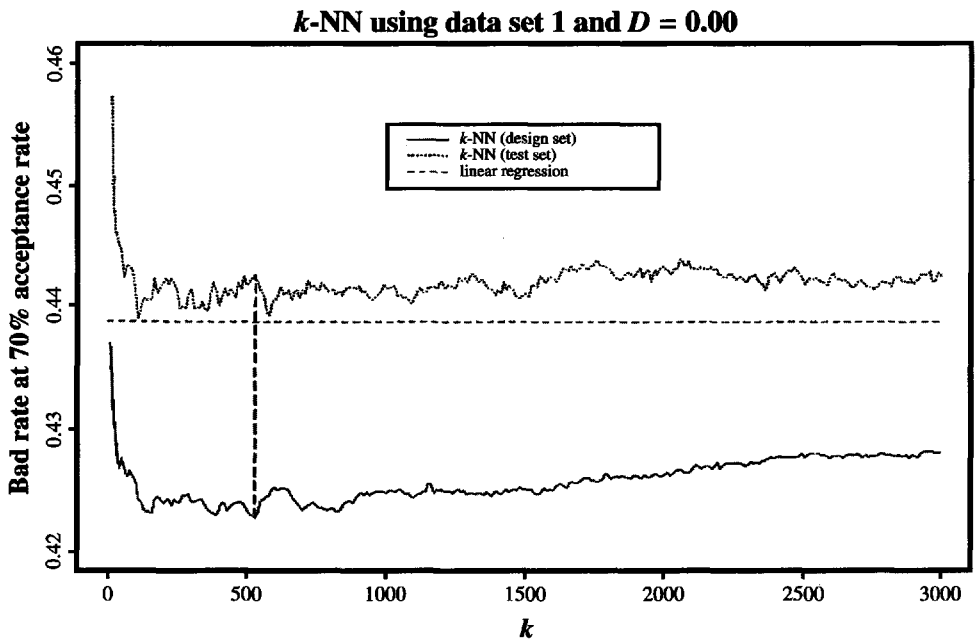


Figure 4.6. *Default (bad) rate in nearest-neighbor systems as k—size of neighbourhood—varies.*

cases to the training sample when it is known if they are good or bad and dropping the cases that have been in the sample longest. This would partially overcome the need to update the scoring system regularly because of the changes in the population, though a metric d like (4.48) would need to be updated as well to take account of the population shifts and this could not be done dynamically. The concern that there is a lot of calculation needed to work out whether a new case is good or bad—finding out which are its k nearest neighbors in the training sample—is misplaced as modern computers can do such calculations in a few seconds. However, in many ways, finding a good metric in the first place is almost the equivalent of the regression approach to scorecard building. Thus many users are content to stop at that point and use a traditional scorecard. As is the case with the decision tree approach, the fact that the nearest-neighbors technique is not able to give a score for each particular applicant's characteristics deprives users of an anchor that helps them feel that they understand what the system is really doing.

It is also the case that monitoring the performance of the system is nigh on impossible. How does one know when the original metric is no longer appropriate? In section 8.12, we discuss the various ways one can do this for regression base scorecards, but those methods will not work for the nearest-neighbor approach.

4.9 Multiple-type discrimination

There may be circumstances in credit scoring where the applicants need to be divided into more than two groups. As was argued earlier in this chapter, one has to be careful about this because the groupings formed may be due to the way the organization goes about its decision making. However, some organizations may split past applicants into the goods whom they

want as customers, those whom they do not want because they default, and those whom they do not want because they do not use the loan facility enough for the organization to make a profit. Similarly, in the U.S., consumers can default or they can seek the protection of personal bankruptcy against their debts. In that case, the sample of past applicants can be split into goods, defaulters, and bankrupts since it is considered that the characteristics of the latter two groups are different. A detailed analysis of bankruptcy is found in Chapter 13.

If one wants to divide the population into g groups, most of the previous methods will continue to apply if suitable modifications are made. Thus one can change the decision-making approach to discriminant analysis as follows.

Suppose that $c(i, j)$ is the cost of misassigning an applicant of group j to group i, and let p_j be the proportion of the population in group j. Let $p(\mathbf{x}|j)$ be the probability that applicants in group j have application attributes $\mathbf{x}$ (similar to $p(\mathbf{x}|G)$ in section 4.2). Then the probability that an applicant with attributes $\mathbf{x}$ is in group j, which we define as $P(j|\mathbf{x})$, satisfies

$$P(j|\mathbf{x}) = \frac{p_j p(\mathbf{x}|j)}{\Sigma_i p_i p(x|i)}. \tag{4.49}$$

If one wants to minimize the expected loss, one would assign applicant with an attribute $\mathbf{x}$ to group i if

$$\Sigma_j c(i, j) p_j p(\mathbf{x}|j) < \Sigma_j c(k, j) p_j p(\mathbf{x}|j) \quad \text{for all } k,\ k \neq i. \tag{4.50}$$

In a similar way, one can extend the logistic regression approach to classifying three or more groups by taking (4.32) as the probability the customer is in one class with similar forms for estimates of probabilities of being in the other classes. Although the statistical methods of multiple group classification are well developed, there has been little application of these ideas in credit scoring.

Chapter 5

Nonstatistical Methods for Scorecard Development

5.1 Introduction

The original idea in the development of credit scorecards was to use statistical analysis of a sample of past customers to help decide which existing or new customers were likely to be satisfactory. One can also look at nonstatistical approaches to the same problem. This has happened to all such classification problems in the last 25 years. Until the 1980s, the only approaches were statistical ones, but then it was realized (Freed and Glover 1981a, 1981b) that finding the linear function of the characteristics that best discriminates between groups can be modeled as a linear programming problem. The linear programming approach measures goodness of fit by taking the sum of the absolute errors involved or the maximum error involved. This is discussed in section 5.2. If one wants to take the number of cases where the discrimination is incorrect as a measure of goodness of fit, then one has to introduce integer variables into the linear program, and this leads to the integer programming models of section 5.3.

The 1970s saw tremendous research in the area of artificial intelligence, whereby researchers tried to program computers to perform natural human skills. One of the most successful attempts was expert systems, in which computers were given a database of information on a certain field obtained from experts in that field and a mechanism for generating rules from the information. The computer programs then used this combination to analyze new situations and come up with ways to deal with the new situation, which would be as good as experts would suggest. Successful pilot systems were developed for medical diagnosis, and since this is essentially a classification problem, researchers considered using expert systems in credit scoring. So far this has had limited success, as outlined in section 5.6.

In the 1980s another variant on the artificial intelligence–based approach to classification problems, neural networks, suddenly came to the fore and has continued to be a very active area of research. Neural networks are ways to model the decision process in a problem in the way the cells in the brain use neurons to trigger one another and hence set up learning mechanisms. A system of processing units is connected together, each of which gives an output signal when it receives input signals. Some of the processing units can receive external input signals and one can give an output signal. The system is given a set of data where each example is a set of input signals and a specific output signal. It tries to learn from this data how to reproduce the relationship between the input and output signals by adjusting the way each processing unit relates its output signal to the corresponding input signal. If the

input signals are taken to be the characteristics of a customer and the output signal is whether their credit performance is good or bad, one can see how this approach can be used in credit scoring. Section 5.4 looks at the application of neural networks in credit scoring.

One can also think of the development of a scorecard as a type of combinatorial optimization problem. One has a number of parameters—the possible scores given to the various attributes—and a way to measure how good each set of parameters is (say, the misclassification error when such a scorecard is applied to a sample of past customers). There have been a number of generic approaches to solving such problems in the last decade—simulated annealing, tabu search, and genetic algorithms. The last of these has found some applications in pilot projects in credit scoring. These ideas are described in section 5.5.

This chapter outlines these approaches and their relative advantages and disadvantages and concludes by describing the results of comparisons of their classification accuracy and those of statistically based methods in credit and behavioral scoring applications.

5.2 Linear programming

Mangasarian (1965) was the first to recognize that linear programming could be used in classification problems where there are two groups and there is a separating hyperplane, i.e., a linear discriminant function, which can separate the two groups exactly. Freed and Glover (1981a) and Hand (1981) recognized that linear programming could also be used to discriminate when the two groups are not necessarily linearly separable, using objectives such as minimization of the sum of absolute errors (MSAE) or minimizing the maximum error (MME).

Recall that the decision that has to be made by the lending organization is to split A, the set of all combinations of values of the application variables $\mathbf{X} = (X_1, X_2, \ldots, X_p)$ can take, into two sets: A_G, corresponding to the answers given by the goods, and A_B, the set of answers given by the bads. Suppose one has a sample of n previous applicants. If n_G of the sample are goods, then for ease of notation we assume these are the first n_G in the sample. The remaining n_B of the sample $i = n_G + 1, \ldots, n_G + n_B$ are bad. Assume applicant i has characteristics $(x_{i1}, x_{i2}, \ldots, x_{ip})$ in response to the application variables $\mathbf{X} = (X_1, X_2, \ldots, X_p)$. We want to choose weights or scores $(w_1, w_2, \ldots, w_p)$ so that the weighted sum of the answers $w_1X_1 + w_2X_2 + \cdots + w_pX_p$ is above some cutoff value c for the good applicants and below the cutoff value for the bads. If the application variables have been transformed into binary variables, then one can think of the weights w_i as the scores for the different answers given.

Usually, one cannot hope to get a perfect division of the goods from the bads, so one introduces variables a_i, all of which are positive or zero, which allow for possible errors. Thus if applicant i in the sample is a good, we require $w_1x_{i1} + w_2x_{i2} + \cdots + w_px_{ip} \geq c - a_i$, while if applicant j is a bad, we require $w_1x_{j1} + w_2x_{j2} + \cdots + w_px_{jp} \leq c + a_j$. To find the weights $(w_1, w_2, \ldots, w_p)$ that minimize the sum of the absolute values of these deviations (MSD), one has to solve the following linear program:

$$\begin{array}{lll} \text{Minimize} & a_1 + a_2 + \cdots + a_{n_G+n_B} & \\ \text{subject to} & w_1x_{i1} + w_2x_{i2} + \cdots + w_px_{ip} \geq c - a_i, & 1 \leq i \leq n_G, \\ & w_1x_{i1} + w_2x_{i2} + \cdots + w_px_{ip} \leq c + a_i, & n_G + 1 \leq i \leq n_G + n_B, \\ & a_i \geq 0, & 1 \leq i \leq n_G + n_B. \end{array} \tag{5.1}$$

If instead one wished to minimize the maximum deviation (MMD), one simplifies to the same error term in each constraint, namely,

$$\begin{array}{ll} \text{Minimize} & a \\ \text{subject to} & w_1x_{i1} + w_2x_{i2} + \cdots + w_px_{ip} \geq c - a, \qquad 1 \leq i \leq n_G, \\ & w_1x_{i1} + w_2x_{i2} + \cdots + w_px_{ip} \leq c + a, \quad n_G + 1 \leq i \leq n_G + n_B, \\ & a \geq 0. \end{array} \tag{5.2}$$

In credit scoring, an advantage of linear programming over statistical methods is that if one wants a scorecard with a particular bias, linear programming can easily include the bias into the scorecard development. Suppose, for example, that X_1 is the binary variable of being under 25 or not and X_2 is the binary variable of being over 65, and one wants the score for under 25s to be higher than for retired people. All that is necessary is to add the constraint $w_1 \geq w_2$ to the constraints in (5.1) or (5.2) to obtain such a scorecard. Similarly, one could ensure that the weighting on the application form variables exceeds those on the credit bureau variables, where the former are the variable X_1 to X_s and the latter are X_{s+1} to X_p, by adding to (5.1) or (5.2) constraints for each applicant that

$$w_1x_{i1} + w_2x_{i2} + \cdots + w_sx_{is} \geq w_{s+1}x_{is+1} + w_{s+2}x_{is+2} + \cdots + w_px_{ip} \quad \text{for all } i. \tag{5.3}$$

However, one has to be slightly careful because in the linear programming formulation one has made the requirements that the goods are to have scores higher than or the same as the cutoff score and bads are to have scores lower that or the same as the cutoff score since linear programming cannot deal with strict inequalities. This means that if one were able to choose the cutoff score as well as the weights, there is always a trivial solution where one puts the cutoff at zero and all the weights at zero also, i.e., $w_i = 0$ for all i. Thus everyone has a zero total score and sits exactly on the cutoff. One way out of this would seem to be to set the cutoff score to be some nonzero value, say, 1. However, Freed and Glover (1986a, 1986b) pointed out this is not quite correct and one will need to solve the model twice—once with the cutoff set to a positive value and once with it set to a negative value.

To see this, consider the following two examples with one variable X_1 and three applicants.

Example 5.1. In Figure 5.1(a), one has the two goods having values 1 and 2, respectively, and the bad has a value 0. It is easy to see that if the cutoff is 1, one can choose $w_1 = 1$ and there are no errors. For Figure 5.1(b), however, the linear program that minimizes MSD is

$$\begin{array}{ll} \text{Minimize} & a_1 + a_2 + a_3 \\ \text{subject to} & 2w \leq 1 + a_1, \quad w \geq 1 - a_2, \quad 0 \geq 1 - a_3. \end{array}$$

This is solved by taking $w = \frac{1}{2}$ with $a_1 = 0$, $a_2 = 0.5$, and $a_3 = 1$, so the total error is 1.5. Yet if we allow the cutoff point to be -1, then minimizing MSD becomes

$$\begin{array}{ll} \text{Minimize} & a_1 + a_2 + a_3 \\ \text{subject to} & 2w \leq -1 + a_1, \quad w \geq -1 - a_2, \quad 0 \geq -1 - a_3. \end{array}$$

This is solved exactly by letting $w = -\frac{1}{2}$ with 0 total error.

However, if we try to solve Figure 5.1(a) with a cutoff of -1, then the best we can do is $w = -\frac{1}{2}$, with a total error of 1.

The point is that if the goods generally have higher characteristic values than the bads, then one wants the characteristics to have positive weights and hence a positive cutoff score. If the goods tend to have lower values in each characteristic than the bads, then to get the total score for a good to be higher than the total score for a bad, the weights w_i need to

Figure 5.1.

be negative. This will make the total scores all negative, and so the cutoff value will be a negative score.

Fixing the cutoff value causes another problem, namely, those cases where the ideal cutoff score would have been zero. Formally, this means that the solutions obtained by linear programming are not invariant under linear transformations of the data. One would hope that adding a constant to all the values of one variable would not affect the weights the scorecard chose nor how good its classification was. Unfortunately, that is not the case, as the following example shows.

Example 5.2. There are two characteristic variables (X_1, X_2), and let there be three good cases with values $(1, 1)$, $(1, -1)$, and $(-1, 1)$ and three bad cases with values $(0, 0)$, $(-1, -1)$, and $(0.5, 0.5)$. If one is trying to minimize the sum of the deviations using a cutoff boundary of $w_1X_1 + w_2X_2 = 1$, then by symmetry one can assume the cutoff goes through the point $(1, -1)$ and so is of the form $(c + 1)X_1 + cX_2 = 1$ (see Figure 5.2). With $0.5 < c$, the deviation for $(-1, 1)$ is 2 and for $(0.5, 0.5)$ is $c - 0.5$, and so the deviation error is minimized at 2 by the scorecard $1.5X_1 + 0.5X_2$.

If we add 1 to the values of both variables so that the goods have values $(2, 2)$, $(2, 0)$, and $(0, 2)$ and the bads have values $(1, 1)$, $(0, 0)$, and $(1.5, 1.5)$, then the cutoff that minimizes MSD is $0.5X_1 + 0.5X_2 = 1$ (see Figure 5.2) with a deviation of 0.5, which is quite a different scorecard.

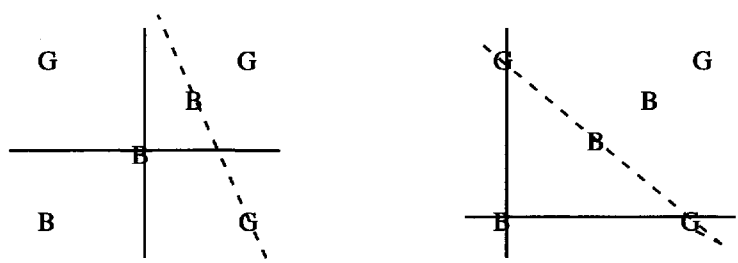


Figure 5.2. *Points in Example* 5.2.

Various modifications of the formulation have been suggested to overcome these problems such as requiring the one constraint in (5.1) to become

$$w_1x_{i1} + w_2x_{i2} + \cdots + w_px_{ip} \leq c - e + a_i, \quad n_G + 1 \leq i \leq n_G + n_B \tag{5.4}$$

so there is a gap between the two regions. However, then one has to develop techniques to decide into which set the points in this gap should be put. These difficulties have been overcome in the case when the mean values of the characteristics of the goods and the bads are different by using the normalization proposed by Glover (1990), namely, by adding the constraint

$$\sum_{j=1}^{p}\left(n_B\sum_{i=1}^{n_G}x_{ij}-n_G\sum_{i=n_G+1}^{n_G+n_B}x_{ij}\right)w_j=1. \tag{5.5}$$

One of the most general of the linear programming formulations was proposed by Freed and Glover (1986b). As well as the errors or external deviations a_i, by which a constraint is not satisfied, it also looks at the internal deviations e_i, which are how far from the boundary are the scores of the applicants who are correctly classified. Freed and Glover proposed an objective that was a weighted combination of the maximum internal and external deviations and the sum of the absolute values of the internal and external deviations. This is solved by the following linear program:

$$\begin{aligned}
&\text{Minimize} \quad k_0a_0-l_0e_0+\sum_{i=1}^{n_G+n_B}k_ia_i+\sum_{i=1}^{n_G+n_B}l_ie_i\\
&\text{subject to} \quad w_1x_{i1}+w_2x_{i2}+\cdots+w_px_{ip}\geq c-a_0-a_i+e_0+e_i, \quad 1\leq i\leq n_G,\\
&\qquad\qquad w_1x_{i1}+w_2x_{i2}+\cdots+w_px_{ip}\leq c+a_0+a_i-e_0-e_i,\\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad n_G+1\leq i\leq n_G+n_B,\\
&\qquad\sum_{j=1}^{p}\left(n_B\sum_{i=1}^{n_G}x_{ij}-n_G\sum_{i=n_G+1}^{n_G+n_B}x_{ij}\right)w_j=1,\\
&\qquad\qquad\qquad\qquad a_i,e_i\geq 0, \qquad\qquad 0\leq i\leq n_G+n_B.
\end{aligned} \tag{5.6}$$

This formulation of the problem always gives a nontrivial solution and is invariant under linear transformation of the data.

The practical complaint about the linear programming approach is that since it has no statistical underpinning, one cannot evaluate whether the estimated parameters are statistically significant. Ziari et al. (1997) suggested a way to estimate the parameters by using jackknife or bootstrap resampling techniques. Both procedures assume that the sample used in the linear program represents the whole population of clients. By resampling a subset from the original sample, each new subsample will generate different parameter estimates. The distribution of these estimates reflects some information about the distribution of the parameters obtained from the full sample. In the jackknife procedure, a subset of t clients is left out of the sample and the linear program is solved on the remaining ones (usually $t = 1$). This is repeated using different subsets of t to get estimates for the parameters. In the bootstrap approach, a sample of $n = n_G + n_B$ is taken with replacement from the original data set of the same size and the linear programming parameters are calculated. This is repeated a number of times, and the mean and standard deviation of these bootstrap estimates of the parameters give approximations for the standard deviation in the original parameter estimates. Further details of bootstrapping and jackknifing are found in section 7.4.

Another advantage claimed for regression over linear programming is that in the former you can introduce variables one at a time into the scorecard starting with the most powerful—the forward approach of many statistical packages. This means you can develop lean and mean models wherein you decide in advance the number m of characteristics you want and regression then finds the m that are most discriminating. Nath and Jones (1988) showed how the jackknife approach applied to linear programming can be used to select the m most powerful characteristics. It does, however, involve solving the linear program a large number of times.

The review papers by Erenguc and Koehler (1990) and Nath, Jackson, and Jones (1992) compared the linear programming and regression approaches to classification on several data

sets (but none of the sets was credit data). Their results suggest that the linear programming approach, while competitive, does not classify quite as well as the statistical methods.

5.3 Integer programming

Linear programming models minimize the sum of the deviation in the credit score of those who are misclassified. However, a more practical criterion would be to minimize the number of misclassifications, or the total cost of misclassifying if it is thought that D, the cost of misclassifying a bad as a good is very different from L, the cost of misclassifying a good as a bad. Building scorecards under these criteria can again be done by linear programming, but as some of the variables will have to be integer (0, 1, etc.), this technique is called integer programming. Such a model was given by Koehler and Erenguc (1990) as follows:

$$\begin{array}{lll} \text{Minimize} & L(d_1 + \cdots + d_{n_G}) + D(d_{n_{G+1}} + \cdots + d_{n_{G+B}}) & \\ \text{subject to} & w_1 x_{i1} + \cdots + w_p x_{ip} \geq c - M d_i, & 1 \leq i \leq n_G, \\ & w_1 x_{i1} + \cdots + w_p x_{ip} \leq c + M d_i, & n_G + 1 \leq i \leq n_G + n_B, \\ & 0 \leq d_i \leq 1, & d_i \text{ integer.} \end{array} \tag{5.7}$$

Thus d_i is a variable that is 1 if customer i in the sample is misclassified and 0 if not. Again, as it stands (5.7) is minimized by $c = 0$, $w_i = 0, i = 1, \ldots, p$, so one has to add normalization conditions, for example,

$$\begin{aligned} &\sum_{j=1}^{p} (s_j^+ + s_j^-) = 1, \\ &\qquad 0 \leq s_j^+, \quad s_j^- \leq 1, \quad \text{and} \quad s_j^+, s_j^- \text{ integer}, \quad j = 1, \ldots, p, \\ &-1 + 2s_j \leq w_j \leq 1 - 2s_j, \quad j = 1, \ldots, p. \end{aligned} \tag{5.8}$$

This set of constraints requires one of the s_i^+, s_i^- to be 1 and the corresponding w_i to be either greater than 1 or less than -1 (equivalent to forcing c to be positive or negative). Thus it is similar to requiring that

$$\sum_{j=1}^{p} w_i = 1 \quad \text{and} \quad c = +1 \text{ or } -1.$$

Joachimsthaler and Stam (1990) and Koehler and Erengac (1990) found that the integer model (5.7) is a better classification model than the linear programming models. However, it has two major disadvantages. First, it takes much longer than linear programming to solve and hence can deal with only very small sample sets with number of cases in the hundreds. Second, there are often a number of optimal solutions with the same number of misclassifications on the training set but with quite different performances on holdout samples.

The computational complexity of the solution algorithms for integer programming is a real difficulty, which probably rules it out for commercial credit-scoring applications. Several authors suggested ways to exploit the structure of the classification problem to increase the speed of the branch-and-bound approach to solving integer programming in this case (Rubin 1997, Duarte Silva and Stam 1997). However, this still makes the approach feasible only when the sample set is 500 or smaller.

Various authors added extra conditions and secondary objectives to (5.7) to ensure a classification that is robust in the holdout sample, as well as minimizing the number of

misclassifications in the training sample. Pavar, Wanarat, and Loucopoulus (1997) suggested models that also maximize the difference between the mean discriminant scores; Bajgier and Hill (1982) sought to minimize the sum of the exterior deviations (the difference between the score of the misclassified and the cutoff score) as well as the number of misclassifications. Rubin (1990) took a secondary goal of maximizing the minimum interior deviation—the difference between the score of the correctly classified and the cutoff score.

Thus far, integer programming has been used to solve the classification problem with the minimum number of misclassifications criterion, but it can also be used to overcome two of the difficulties that arise in the linear programming approach to minimizing the absolute sum of errors. The first is the problem about removing trivial solutions and ensuring the weights are invariant if the origin shifts in the data. This was overcome in the minimum misclassification formulation (5.7) by using the normalization (5.8). A similar normalization can be applied to the MSD or MMD problem. This would lead to the following integer program to minimize the sum of the absolute value of the deviations (MSD) (see Glen (1999)):

$$
\begin{aligned}
\text{Minimize} \quad & a_1 + a_2 + \cdots + a_{n_G+n_B} \\
\text{subject to} \quad & (w_1^+ - w_1^-)x_{i1} + (w_2^+ - w_2^-)x_{i2} + \cdots + (w_p^+ - w_p^-)x_{ip} \geq c - a_i, \\
& \qquad\qquad 1 \leq i \leq n_G, \\
& (w_1^+ - w_1^-)x_{i1} + (w_2^+ - w_2^-)x_{i2} + \cdots + (w_p^+ - w_p^-)x_{ip} \leq c + a_i, \\
& \qquad\qquad n_{G+1} \leq i \leq n_G + n_B, \\
& (w_1^+ + w_1^-) + (w_2^+ + w_2^-) + \cdots + (w_p^+ + w_p^-) = 1, \\
& cs_j^+ \leq w_i^+ \leq s_j^+, \quad cs_j^- \leq w_i^- \leq s_j^-, \quad c = 1, \ldots, p, \\
& \qquad\qquad s_j^+ + s_j^- \leq 1, \quad j = 1, \ldots, p,
\end{aligned}
\tag{5.9}
$$

where $w_i^+, w_i^- \geq 0$, $0 \leq s_j^+, s_j^- \leq 1$, s_j^+, s_j^- integer, $j = 1, \ldots, p$, and c is any positive constant less than 1.

These conditions guarantee that at most one of w_i^+, w_i^- is positive, but some of the $w_i^\pm$ must be nonzero. The result is the same as the normalization in (5.8). With these extra $2p$ integer variables, one can ensure there are no trivial solutions and the results are invariant under change of origin.

Another criticism of the linear programming approach was that it was hard to choose the best scorecard using only m characteristics. Jackknifing methods were suggested to overcome this problem, but one can also use the integer variable formulation of (5.9) to find the best scorecard using only m characteristics. All one needs to do is to add another constraint to (5.9), namely,

$$
\sum_{j=1}^{p} (s_j^+ + s_j^-) = m. \tag{5.10}
$$

This ensures that only m of the weights w_i^+, w_i^- are positive and so only m of the characteristics are nonzero. One could solve this problem for different values of m to get a view of which characteristics should be entered when m variables are already included. One technical difficulty is that several different scorecards may be optimal using m characteristics, so going to $m + 1$ characteristics might mean that several characteristics leave the scorecard and several new ones enter. One can overcome this by finding all optimal m characteristic scorecards as follows. If solving the problem with (5.9) and (5.10) gives a solution where the nonzero characteristics are the set $C = \{i_1, i_2, \ldots, i_m\}$, add the constraint

$$
\sum_{j \in C} (s_j^+ + s_j^-) \leq m - 1
$$

and solve again. This will give a different solution and one can check if it gives the same objective function. If it does not, the solution is unique, but if it does, then repeat the process until all the optimal solutions are found.

5.4 Neural networks

Neural networks were originally developed from attempts to model the communication and processing information in the human brain. In the brain, large numbers of dendrites carry electrical signals to a neuron, which converts the signals to a pulse of electricity sent along an axon to a number of synapses, which relate information to the dendrites of other neurons. The human brain has an estimated 10 billion neurons (Shepherd and Koch, 1990). Analogous to the brain, a neural network consists of a number of inputs (variables), each of which is multiplied by a weight, which is analogous to a dendrite. The products are summed and transformed in a "neuron" and the result becomes an input value for another neuron.

5.4.1 Single-layer neural networks

A single-layer neural network consists of the components just described where instead of the transformed value becoming the input for another neuron, it is the value we seek. Thus it may predict whether a case is to be accepted or rejected. A single-layer neural network may be represented diagrammatically as in Figure 5.3.

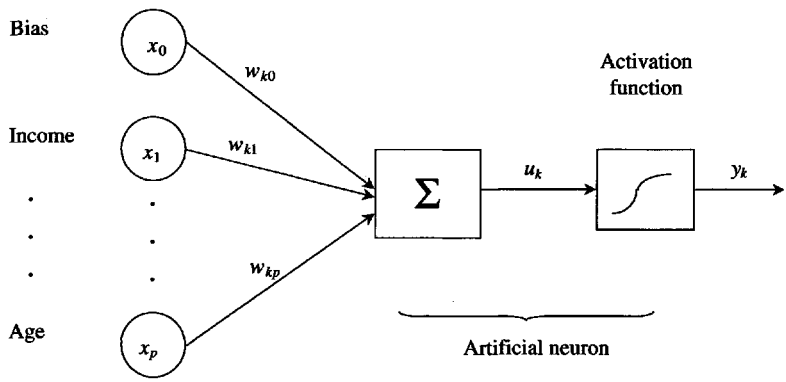


Figure 5.3. *A single-layer neural network.*

We can represent a single-layer neural network algebraically as

$$u_k = w_{k0}x_0 + w_{k1}x_1 + \cdots + w_{kp}x_p = \sum_{q=0}^{p} w_{kq}x_q, \tag{5.11}$$

$$y_k = F(u_k). \tag{5.12}$$

Each of $x_1, \ldots, x_p$ is a variable, such as a characteristic of a credit card applicant. Each takes on a value known as a signal. The weights, often known as synaptic weights, if positive are known as excitory because they would increase the corresponding variable, and if negative are called inhibitory because they would reduce the value of u_k for positive variables. Notice that the subscripts on each weight are written in the order (k, p), where k indicates the neuron to which the weight applies and p indicates the variable. In a single-layer neuron, $k = 1$

because there is only one neuron. Notice also that variable x_0 is ascribed the value $+1$ so that the $w_{k0}x_0$ term in (5.11) is just w_{k0}, often known as bias. This has the function of increasing or decreasing u_k by a constant amount.

The u_k value is then transformed using an activation (or transfer or squashing) function. In early networks, this function was linear, which severely limited the class of problems that such networks could deal with. Various alternative transfer functions are used, and they include the following:

- Threshold function:

$$\begin{aligned} F(u) &= 1 \quad \text{if } u \geq 0, \\ &= 0 \quad \text{if } u < 0, \end{aligned} \tag{5.13}$$

  which implies that if u is 0, or greater the neuron outputs a value of 1; otherwise, it outputs a 0.

- Logistic function:

$$F(u) = \frac{1}{1 + e^{-au}}. \tag{5.14}$$

Both are represented in Figure 5.4. The value of a in the logistic function determines the slope of the curve. Both functions constrain the output of the network to be within the range $(0, 1)$. Sometimes we wish the output to be in the range $(-1, +1)$ and the tanh function, $F(u) = \tanh(h)$, is used.

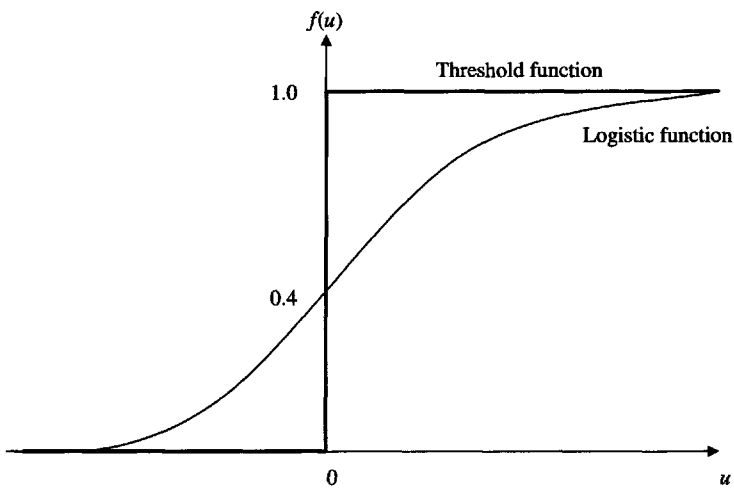


Figure 5.4. *Threshold and logistic activation functions.*

Given the values for the weights and the transfer function, we can predict whether an application for a credit card is to be accepted or rejected by substituting the applicant's characteristics into (5.11), calculating the value of y_k from (5.12), and comparing this value with a cutoff value.

A model consisting of a single neuron and a threshold activation function is known as a perceptron. Rosenblatt (1958, 1960) showed that if the cases to be classified were linearly separable, that is, they fall either side of a straight line if there are two input signals

(and of a hyperplane when there are p signals), then an algorithm that he developed would converge to establish appropriate weights. However, Minsky and Papert (1969) showed that the perceptron could not classify cases that were not linearly separable.

In 1986, Rumelhart, Hinton, and Williams (1986a, 1986b) showed that neural networks could be developed to classify nonlinearly separable cases using multiple-layer networks with nonlinear transfer functions. At approximately the same time, methods for estimating the weights in such models using back-propagation were described in Rumelhart and McClelland (1986), Parker (1982), and LeCun (1985). Since this is the most commonly used method, we describe it below.

5.4.2 Multilayer perceptrons

A multilayer perceptron consist of an input layer of signals, an output layer of output signals (different y_v values), and a number of layers of neurons in between, called hidden layers. Each neuron in a hidden layer has a set of weights applied to its inputs which may differ from those applied to the same inputs going to a different neuron in the hidden layer. The outputs from each neuron in a hidden layer have weights applied and become inputs for neurons in the next hidden layer, if there is one; otherwise, they become inputs to the output layer. The output layer gives the values for each of its member neurons whose values are compared with cutoffs to classify each case. A three-layer network is shown in Figure 5.5.

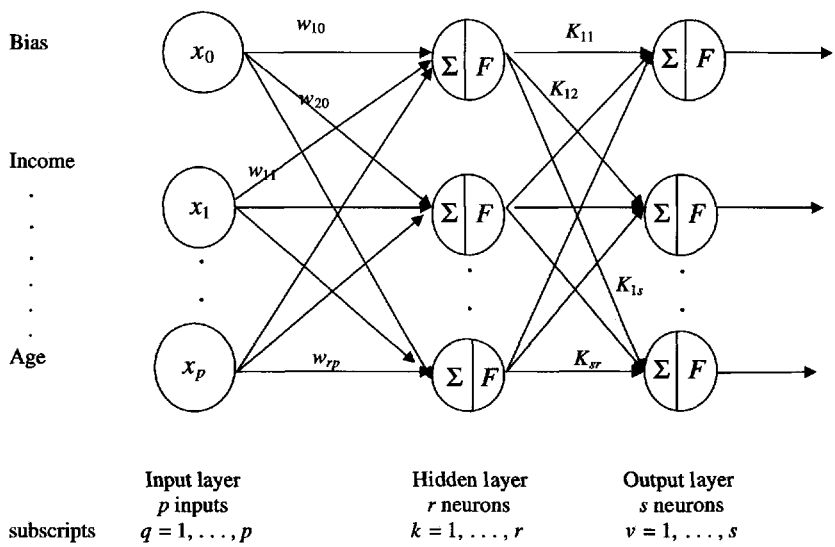


Figure 5.5. *A multilayer perceptron.*

We can represent a multilayer perceptron algebraically as follows. We do this for Figure 5.5. From (5.11) and (5.12), we have

$$y_k = F_1\left(\sum_{q=0}^{p} w_{kq}x_q\right), \tag{5.15}$$

where the subscript 1 on F indicates it is the first layer after the input layer. The y_k, $k = 1, \ldots, r$, are the outputs from the first hidden layer. Since the output of one layer is an

input to the following layer, we can write

$$z_v = F_2\left(\sum_{k=1}^{r} K_{vk} y_k\right) = F_2\left(\sum_{k=1}^{r} K_{vk}\left(F_1\left(\sum_{q=0}^{p} w_{kq} x_q\right)\right)\right), \tag{5.16}$$

where z_v is the output of neuron v in the output layer, $v = 1, \ldots, s$, F_2 is the activation function in the output layer, and K_{vk} is the weight applied to the y_k layer that joins neuron k in the hidden layer and neuron v in the output layer.

The calculation of the vector of weights is known as training. There are many such methods, but the most frequently used method is the back-propagation algorithm. Training pairs, consisting of a value for each of the input variables for a case and the known classification of the case, are repeatedly presented to the network and the weights are adjusted to minimize some error function.

5.4.3 Back-propagation algorithm

Initially all weights are set equal to some randomly chosen numbers. A training pair is selected and the x_p values used to calculate the z_v values using (5.16). The difference between the z_v values and the known values, o_v, are calculated. This is known as a forward pass. The backward pass consists of distributing the error back through the network in proportion to the contribution made to it by each weight and adjusting the weights to reduce this portion of the error. Then a second training pair is selected and the forward and backward passes are repeated. This is repeated for all cases, called an epoch. The whole procedure is repeated many times until a stopping criterion is reached.

The change in the weights effected when a new case is presented is proportional to the first derivative of the error term with respect to each weight. This can be understood as follows. Define the error when the training case t is presented, $e_v(t)$, as

$$e_v(t) = o_v(t) - y_v(t), \tag{5.17}$$

where $o_v(t)$ is the actual observed outcome for case t in neuron v and $y_v(t)$ is the predicted outcome. The aim is to choose a vector of weights that minimizes the average value over all training cases of

$$E(t) = 0.5 \sum_{v=1}^{s} e_v^2(t), \tag{5.18}$$

where s is the number of neurons in the output layer. This average value is

$$E_{\text{mean}}(t) = \frac{1}{N} \sum_{t=1}^{N} E(t), \tag{5.19}$$

where N is the number of training cases.

For any neuron v in any layer c, we can write

$$u_v^{[c]} = \sum_{k=0}^{r} w_{vk} y_k^{[c-1]}, \tag{5.20}$$

$$y_v^{[c]} = F(u_v^{[c]}), \tag{5.21}$$

which are simply (5.11) and (5.12) rewritten to apply to any neuron in any layer rather than just to the input layer and first neuron.

Hence the partial derivative of $E(t)$ with respect to weight w_{vk} can, according to the chain rule, be written as

$$\frac{\partial E(t)}{\partial w_{vk}(t)} = \frac{\partial E(t)}{\partial e_v(t)} \cdot \frac{\partial e_v(t)}{\partial y_v(t)} \cdot \frac{\partial y_v(t)}{\partial u_v(t)} \cdot \frac{\partial u_v(t)}{\partial w_{vk}(t)}. \tag{5.22}$$

From (5.18),

$$\frac{\partial E(t)}{\partial e_v(t)} = e_v(t). \tag{5.23}$$

From (5.17),

$$\frac{\partial e_v(t)}{\partial y_v(t)} = -1. \tag{5.24}$$

From (5.21),

$$\frac{\partial y_v(t)}{\partial u_v(t)} = F'(u_v(t)). \tag{5.25}$$

From (5.20),

$$\frac{\partial u_v(t)}{\partial w_{vk}(t)} = y_k(t). \tag{5.26}$$

By substitution,

$$\frac{\partial E(t)}{\partial w_{vk}(t)} = -e_v(t) \cdot F'(u_v(t)) \cdot y_k(t). \tag{5.27}$$

The change in weights between the forward pass and the backward pass is therefore

$$\Delta w_{vk}(t) = -\eta \frac{\partial E(t)}{\partial w_{vk}(t)} = \eta \delta_v(t) y_k(t), \tag{5.28}$$

where $\delta_v(t) = e_v(t)F'(u_v(t))$. The constant η, called a training rate coefficient, is included to alter the change in w so as to make the changes smaller or larger. Smaller values improve accuracy but extend the training time. Equation (5.28) is known as the delta rule (or the Widrow–Hoff rule).

The implementation of this rule depends on whether neuron v is in the output layer or a hidden layer. If neuron v is in the output layer, the value of e_v is directly observable because we know both the observed outcome o_v and the predicted outcome y_v. If neuron v is in a hidden layer, one component in e_v, o_v, is not observable. In this case, we still use (5.28), except that we must calculate $\delta_v(t)$ in a different way. The method used is to calculate the value of δ for each neuron in the output layer, multiply each δ by the weight that connects the neuron for which it was calculated and a neuron in the previous layer, and then sum these products for each neuron in the previous layer. Figure 5.6 illustrates this for two neurons in the output layer.

In general,

$$\delta_k^{[c-1]} = F_k^{[c-1]\prime} \sum_{v=1}^{s} \delta_v^{[c]} w_{vk}^{[c]}. \tag{5.29}$$

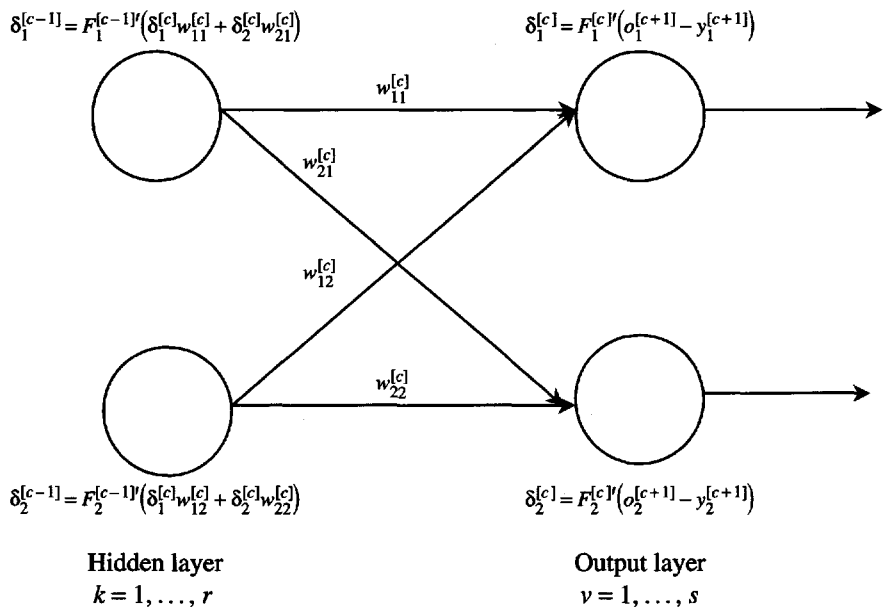


Figure 5.6. *Back-propagation.*

The δ value for any neuron in one layer is distributed recursively, according to the connecting weights, to neurons in the previous layer, and then, for each previous layer neuron, the weighted δ's connecting to it are summed. This gives a value of δ for the previous layer. The procedure is repeated back, layer by layer, to the original set of weights from the input layer.

The change in weights connecting any layer, $[c-1]$, where variables are subscripted k, to the next layer, $[c]$, where variables are subscripted v, is therefore, from (5.28) and (5.29),

$$\Delta w_{vk} = \eta \delta_v^{[c]} y_k^{[c-1]}. \tag{5.30}$$

This rule is often amended by the inclusion of a momentum term, $\alpha \cdot \Delta w_{vk}(t-1)$:

$$\Delta w_{vk}(t) = \alpha \Delta w_{vk}(t-1) + \eta \delta_v^{[c]} y_k^{[c-1]}. \tag{5.31}$$

This is now called the generalized delta rule because (5.30) is a special case of (5.31), where $\alpha = 0$. The momentum may be added to speed up the rate at which the w values converge while reducing the effect of the calculated weights oscillating—that is, on successive iterations alternately increasing and decreasing because the error surface in weight space has a local minimum. Haykin (1999) showed that for the weights to converge, α must be set within the range ± 1. In practice it is normally set to $0 < \alpha < +1$.

The back-propagation algorithm is a gradient descent method. Intuitively, we can think of a diagrammatic representation if we consider only two inputs, as in Figure 5.7. The very idealized surface represented by $ABCD$ represents the size of the error E as a function of the vector of weights. The aim is to find the weight vector $\mathbf{w}^*$ that minimizes the error, as at point E. Our initial choice of weight vector $\mathbf{w}_a$ gives an error value indicated by point F for training case 1 from a feed-forward pass. During a backward pass the weight vector is changed to be $\mathbf{w}_b$ to give an error indicated by point G. The move from point F to point G given by the algorithm is an approximation to the change which would be the

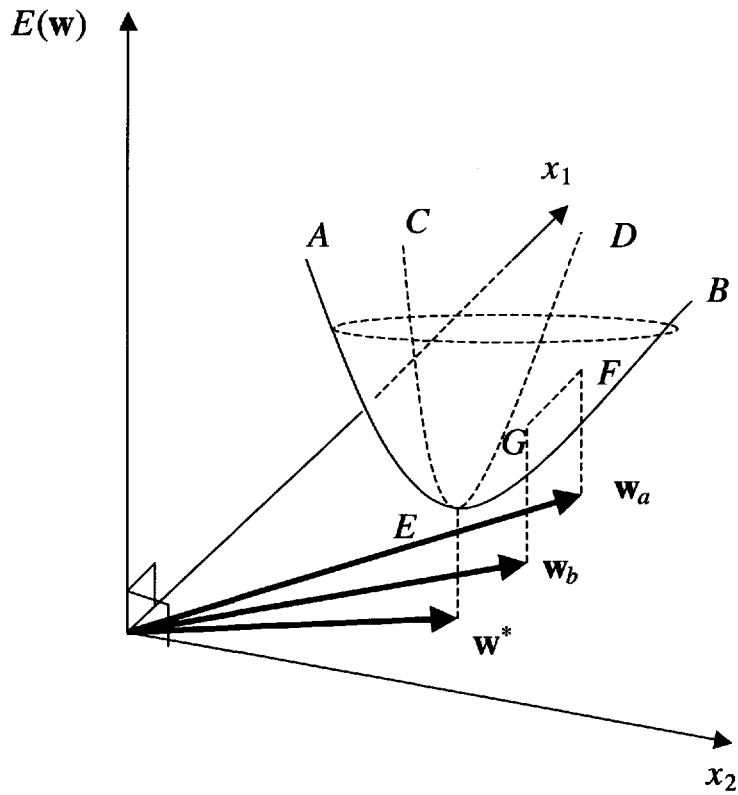


Figure 5.7. *Back-propagation and an error surface.*

greatest descent toward E. In practice, the surface $ABCD$ will contain very many bumps and wrinkles giving many local maxima and minima. To identify a minimum value of the error, we can require the first partial derivative of the error surface with respect to the weight vector $\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \ldots$ to be zero. The algorithm would then be stopped when this occurred. However, this is extremely time-consuming and requires knowledge of these derivatives. An alternative method used in practice is to stop the algorithm when the absolute change in $E_{\text{mean}}(\mathbf{w})$ is sufficiently small—for example, less than 0.01 in each epoch.

5.4.4 Network architecture

A number of aspects of a neural network have to be chosen by the experimenter. These include the number of hidden layers, the number of neurons in the hidden layers, and the error function to use (we have, so far, considered only one as represented by (5.17) and (5.18)). Here we concentrate on the number of hidden layers.

First, consider why we have hidden layers. If we have a single layer, we can classify only groups that are linearly separable. When we include hidden layers together with a nonlinear activation function, the final network can correctly classify cases into classes that are not linearly separable. The hidden layers model the complicated nonlinear relationships between the variables and transform the input variables into hidden space, which is further transformed. As such, these hidden layers extract features in the data. In particular, introducing a single nonlinear hidden layer results in a network that will give a value above or

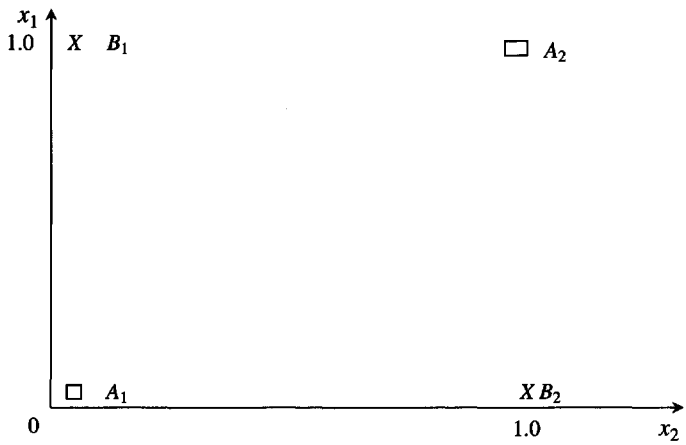


Figure 5.8. *Data for the XOR problem.*

below a cutoff value in a convex region of the input variables. Having a second hidden layer will allow these convex regions to be combined, which can give nonconvex or even totally separate regions. Hand (1997) argued that for this reason, "in principle two hidden layers are sufficient for any problem."

The effect of introducing a hidden layer and nonlinear activation function can be shown using the XOR problem. Consider just two inputs (perhaps characteristics of credit card applicants) which have values of 0 or 1. Cases in class A have either $x_1 = 0$ and $x_2 = 0$ or $x_1 = 1$ and $x_2 = 1$. Cases in class B have either $x_1 = 1$ and $x_2 = 0$ or $x_1 = 0$ and $x_2 = 1$. In Figure 5.8, we have plotted these four possibilities in a graph with axes x_1 and x_2. No straight line can separate the two groups. However, if the values of x_1 and x_2 are weighted and subject to a nonlinear transformation to give y_1 and y_2 values, these y_1 and y_2 values may then be used to separate the groups by using a straight line. A hidden layer consisting of a nonlinear transformation function of the sum of the weighted inputs has been included. If weights with values $+1$ in all four cases, together with biases of $-\frac{3}{2}$ and $-\frac{1}{2}$, are applied, we have

$$u_1 = x_1 + x_2 - \frac{3}{2}, \qquad u_2 = x_1 + x_2 - \frac{1}{2}.$$

If we then use the activation function

$$\begin{aligned} u_1 < 0 &\Rightarrow y_1 = 0, \\ u_1 \geq 0 &\Rightarrow y_1 = 1, \\ u_2 < 0 &\Rightarrow y_2 = 0, \\ u_2 \geq 0 &\Rightarrow y_2 = 1, \end{aligned}$$

then the four cases have values as plotted in Figure 5.9. Class A cases, which in terms of (x_1, x_2) values were (0, 0) or (1, 1), now become, in terms of (y_1, y_2) values, (0, 0) or (1, 1). Class B cases, which in x space were (0, 1) or (1, 0), now become, in y space, (0, 1) and (0, 1). Now a straight line can separate the two groups. This straight line represents the output layer. However, for practical reasons more than two layers may occasionally be used.

In addition, a practitioner needs to decide how many neurons should be included in each hidden layer. The optimal number is not known before training begins. However, a number of heuristics are available for guidance (Garson 1998).

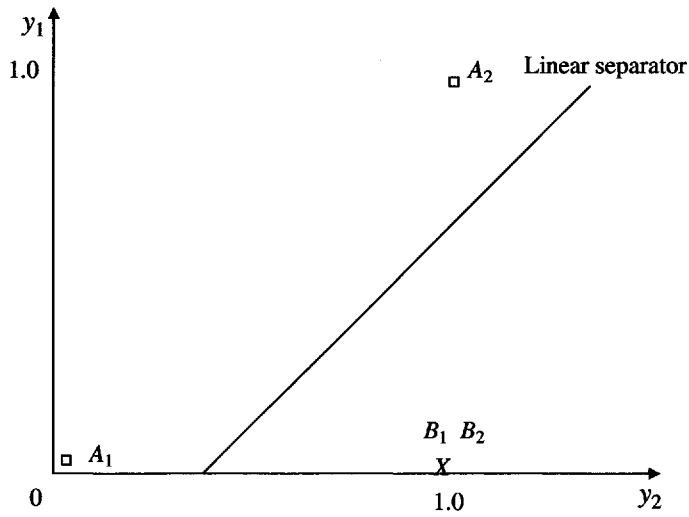


Figure 5.9. *Transformed data.*

5.4.5 Classification and error functions

In credit scoring, we are often interested in classifying individuals into discrete groups, for example, good payers and poor payers or good, poor, and bad payers (Desai et al. 1996, 1997). In principle, when we wish to classify cases into Z groups, we need Z outputs from a multilayer perceptron. Using results by White (1989) and Richard and Lipman (1991), it can be shown that a multilayer perceptron, trained by back-propagation to minimize the value of (5.19) using a finite number of independent cases and identically distributed values of inputs, leads asymptotically to an approximation of the posterior probabilities of class membership (see Haykin 1999, Bishop 1995). Therefore, provided the training sample fulfils these conditions, we can adopt the decision rule that we allocate a case to a group C_g, $g = 1, \ldots, Z$, if the output value of the output layer on which that group has been trained, $F_g(\mathbf{x})$, is greater than the output value of the output layer on which any other group has been trained, $F_h(\mathbf{x})$—that is, if

$$F_g(\mathbf{x}) > F_h(\mathbf{x}), \quad g \neq h. \tag{5.32}$$

However, the error function of (5.19) is not necessarily the best error function for classification problems. It is derived from maximizing the likelihood that the overall network weights and functions generate the set of output values, o_v, which are themselves deterministically determined with additional normally distributed noise. But in classification problems, the output values are binary variables: a case either is a member of a group or is not a member.

Instead of (5.19), we may derive an alternative error function. Suppose we have a network with one output per group g, y_{vg}. The probability of observing the output values, given the input vector $x(t)$, is y_g and the distribution of these probabilities is

$$P(\mathbf{o}(t)|\mathbf{x}(t)) = \prod_{g=1}^{Z} (y_g^t)^{o_g^t}. \tag{5.33}$$

Forming the likelihood function, taking its log, and multiplying by -1, we gain

$$E_2 = -\sum_t \sum_{g=1}^{Z} O_g^t \ln y_g^t. \tag{5.34}$$

This is the relative entropy criterion.

Since the y_v values are to be interpreted as probabilities, they need to have the following properties: $0 \leq y_{vg} \leq 1$ and $\sum_{g=1}^{Z} y_{vg} = 1$. To achieve this, the softmax activation function is often used:

$$y_g = \frac{e^{u_g}}{\sum_{g=1}^{Z} e^{u_g}}. \tag{5.35}$$

Precisely this error and activation function were used by Desai et al. (1997) in their comparison of the classificatory performance of logistic regression, linear discriminant analysis, and neural networks.

5.5 Genetic algorithms

Very simply, a genetic algorithm (GA) is a procedure for systematically searching through a population of potential solutions to a problem so that candidate solutions that come closer to solving the problem have a greater chance of being retained in the candidate solution than others. GAs were first proposed by Holland (1975) and have analogies with the principle of evolutionary natural selection proposed by Darwin (1859).

5.5.1 Basic principles

Suppose that we wish to calculate the parameters $a_1, a_2, \ldots, a_p, b_1, b_2, \ldots, b_p$, and c in the following credit-scoring equation to classify applicants for a loan:

$$f(x_i) = a_1 x_{i1}^{b_1} + a_2 x_{i2}^{b_2} + \cdots + a_p x_{ip}^{b_p} + c, \tag{5.36}$$

where $x_{i1}, \ldots, x_{ip}$ are characteristic values for applicant i.

Once the parameters are estimated, an applicant may be classified as good or bad according to whether $f(x_i)$ is greater than or less than 0.

The procedures followed in the GA method of calculation are shown in Figure 5.10. First, the population of candidate values for the a's, b's, and c's is selected. For example, the range of possible a_1 values may be -1000 to $+1000$ and so on for each a, the range for b_1 may be 0 to 6, and so on. For the purposes of the algorithm, each number in the solution is represented in its binary form. A solution to the calculation problem is a complete set of {0, 1} values for $a_1, \ldots, a_p, b_1, \ldots, b_p$, and c.

At this stage, we introduce some terminology. A collection of 0s and 1s is known as a string or chromosome. Within a string are particular features or genes, each of which takes on particular values or alleles. Thus a solution to the credit-scoring function problem consists of sets of genes arranged in a row, each gene having a value of 0 or 1 and each set relating to $a_1, x_1, n_1, a_2, x_2, n_2$, and so on. The whole row is a chromosome or string.

At the second stage, a number of solutions are selected for inclusion in the intermediate population. These may be chosen randomly unless the analyst has prior knowledge of more appropriate values. To select members of the intermediate population, the performance of

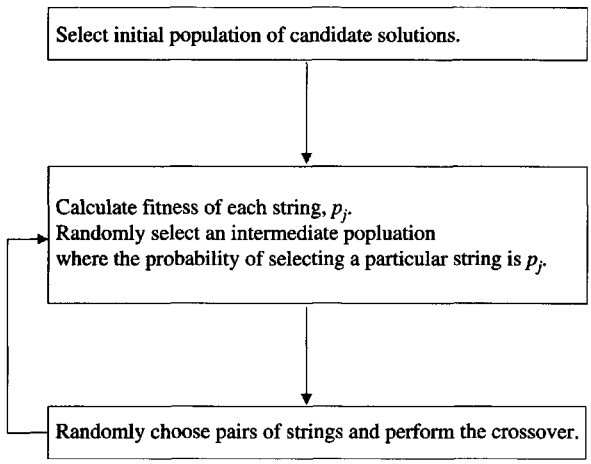


Figure 5.10. *Procedural stages of a genetic algorithm.*

each solution in the initial population is calculated. The performance is often called fitness. In the credit-scoring problem, the fitness might be calculated as the percentage of cases correctly classified. Let j indicate a particular solution. We wish to compare the fitness of each solution, f_j, but the value of f_j depends on the fitness function used. To avoid this, we calculate the normalized fitness function for each candidate solution, p_j, as

$$p_j = \frac{f_j}{\sum_{j=1}^{n_{\text{pop}}} f_j}, \tag{5.37}$$

where n_{pop} is the number of candidate solutions in the population. The intermediate population is selected by randomly selecting strings from the initial population, where p_j is the probability that a string is selected on any occasion. This may be implemented by spinning a roulette wheel where the portion of the circumference that corresponds to each string is equal to p_j of the whole. The wheel is spun n_{pop} times.

At this second stage, an intermediate population with members of the original population only is created. No new strings are created. In the third stage, new strings are created. A given number of pairs of solutions from the intermediate population is chosen and genetic operators are applied. A genetic operator is a procedure for changing the values within certain alleles in one or a pair of strings. Either of two operators may be followed: crossover and mutation. Each chromosome has the same chance of selection for crossover, p_c, which is determined by the analyst. This may be implemented by repeatedly generating a random number r, and if the value of r for the kth number is less than p_c, the kth chromosome is selected.

Crossover is implemented when the first or last n bits of one string (indexed from, say, the left) are exchanged with the first or last n bits of another string. The value of n would be chosen randomly. For example, in the following two parent strings, the last five bits have been exchanged to create two children:

$$0110|11100 \rightarrow 011010110,$$
$$1100|10110 \rightarrow 110011100.$$

The children replace their parents within the population.

Mutation is implemented when an element within a string is randomly selected and its value flipped, that is, changed from 0 to 1 or vice versa. The analyst selects the probability p_m that any element will be mutated.

The selected chromosome, including the children resulting from crossover and after mutation, form the new population. The second and third stages are then repeated a chosen number of times.

The parameters chosen by the analyst are the number of candidate solutions in the population, the probabilities of crossover and of mutation (and hence the expected number of crossovers and mutations), and the number of generations. Michalewicz (1996) suggested some heuristic rules and that the population size of 50–100, with p_c between 0.65 and 1.00 and p_m of 0.001 to 0.01, is often used. Thus Albright (1994) in his application of GAs to the construction of credit union score cards used a population of 100 solutions and up to 450 generations with p_c around 0.75 and p_m around 0.05.

Equation (5.36) is one example of a scoring function that has been used in the scoring literature. Usually the literature does not reveal such detail. A paper that does give such detail (Yobas, Crook, and Ross 2000) has a different function. This can be explained by the chromosome of a solution in Figure 5.11.

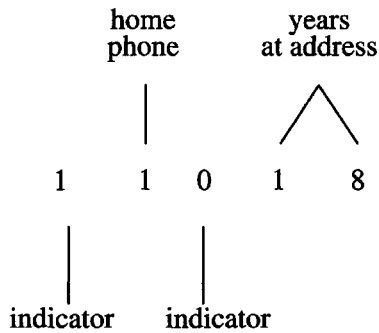


Figure 5.11.

Here there are five genes arranged in groups consisting of two parts. Each group relates to a characteristic (e.g., home phone). Within each group, the first gene, a single bit, indicates whether the variable enters the classification rule (value = 1 if it does, 0 if it does not). The second gene indicates the value of the variable (in the case of binary variables) or the second and third genes indicate a range of values (in the case of continuous variables). If a case fulfils the condition represented by the chromosome, it is predicted to be a good payer; otherwise, it is a bad payer. The condition shown in the chromosome in Figure 5.11 is that the applicant has a home phone. Years at address does not enter the classification rule because the indicator for this variable has the value 0. If the indicator for years at address had been 1, this would also have been relevant to the prediction. In that case the condition would read, "if the applicant has a home phoneand his/her years at address is in the range 1 to 8 inclusive, then (s)he is a good payer."

5.5.2 Schemata

In this section we consider how similarity of values in particular portions within strings can speed up the search for the optimal solution, and so we try to give an understanding of how

and why GAs work. To do this, it is helpful to consider a particular problem. Suppose that we wish to use a GA to find the value of x that maximizes $y = -0.5x^2 + 4x + 100$. By simple differentiation, we know that the answer is 4.

Suppose we set the population to be integer values between 0 and 15. Then we randomly choose four candidate solutions, which are shown in Table 5.1. Examination of Table 5.1 suggests that strings in which the value furthest to the left is 0 have a greater relative fitness p_j than the others.

Table 5.1. *Candidate solutions for a genetic algorithm.*

String no.	String	x (decimal value)	y (fitness)	p_j
1	1 0 1 0	10	90.0	0.2412
2	0 1 1 1	7	103.5	0.2775
3	1 1 0 0	12	72.0	0.1930
4	0 0 1 1	2	107.5	0.2882

A schema (a term first used by Holland (1968)) is a template that states the values in particular positions within a string. If we use the symbol $*$ to mean "don't care" what the value is, we can see that the schema $0 * *1$ is possessed by strings 2 and 4 but not by 1 and not by 3. The similarity of strings is used to show why a GA is likely to progress toward an optimum solution.

Notice that each string matches 2^l schemata. (In each position, there is either the particular 0 or 1 value or a $*$ and there are l positions.) If there are n strings in a population, between 2^l and $n2^l$ different schemata match the population, depending on how many strings have the same schema.

Now consider the effects of the creation of a new population on the expected number of schemata within it. We show that as new generations are created, the expected number of strings in a given-size population that match fitter schemata will increase. That is, fitter schemata will come to dominate the population of solutions.

Suppose the population is set at n_{pop} strings. The expected number of strings that match schema s that will exist in the new generation at time $(t + 1)$ equals

$$\left(\begin{array}{l}\text{number of strings}\\ \text{which match}\\ \text{schema } s, \text{ time } t\end{array}\right) \times \left(\begin{array}{l}\text{probability that strings}\\ \text{which match } s \text{ are selected}\end{array}\right) \times (\text{number of selections}). \tag{5.38}$$

Let $m(s, t)$ denote the number of strings that match s at time t. This is the first term. Remember from (5.37) that in the selection process, the probability that a string j is selected, p_j, is the fitness of the string j divided by the total fitness of all strings in the population. Thus the probability that a string that matches schema s is selected is the average fitness of the strings that match s divided by the total fitness of all strings in the population. That is,

$$\frac{f(s, t)}{\sum_{j=1}^{n_{\text{pop}}} f_j}, \tag{5.39}$$

where $f(s, t)$ is the average fitness of all strings that match s at time t. Finally, the number of selections is just the population size. Substituting into (5.38) gives

$$m(s, t+1) = m(s, t) \cdot \frac{f(s, t)}{\sum_{j=1}^{n_{\text{pop}}} f_j} \cdot n_{\text{pop}}, \tag{5.40}$$

where $m(s, t+1)$ is the expected number of strings that match s at time $t+1$. Writing

$$\frac{\sum_{j=1}^{n_{\text{pop}}} f_j}{n_{\text{pop}}}$$

as $\bar{f}$, the average fitness of all strings in the population, we can rewrite (5.40) as

$$m(s, t+1) = m(s, t) \cdot \frac{f(s, t)}{\bar{f}}. \tag{5.41}$$

Equation (5.41) shows that as each generation is created, if the average fitness of schema s is greater than that in a population, then the expected number of strings in the population with this schema will increase. Correspondingly, the expected number of strings within the population with a schema with a relatively low fitness will decrease. In fact, it can be shown that if the relative fitness of a schema exceeds the average, the number of strings with this schema increases exponentially as further generations are created. To show this, suppose that $f(s, t) = (1+k)\bar{f}$, that is, the average fitness of a schema is a constant proportion of $\bar{f}$ larger than $\bar{f}$. Then from (5.41), $m(s, 1) = m(s, 0)(1+k)$, $m(s, 2) = m(s, 1)(1+k) = m(s, o)(1+k)^2$, and in general $m(s, t) = m(s, 0)(1+k)^t$.

We now amend (5.41) to include crossover and mutation. To understand this, we need to define the order and the defining length of a schema. The order of a schema H, labeled $O(H)$ is the number of numbers (0 and 1) in fixed positions in the schema. The defining length of a schema (denoted δ) is the difference in positions between the first and last fixed numbers in the schema. For example, in the schema

$$10 * * * * 1 *,$$

the order is 3 and the defining length is $7 - 1 = 6$.

Now consider crossover. Consider two schemata:

$$S_1: \quad 11 * * * * * 01,$$
$$S_2: \quad * * * 11 * * * *.$$

Since the crossover point is chosen randomly and each point to be chosen is equally likely to be chosen, and supposing that the crossover operator is a swapping of the last four elements between two strings, crossover is more likely to destroy S_1 than S_2. The last four elements of S_2 are "don't care," and so, if replaced by any string, the new child would still be an example of schema S_2. On the other hand, it is much less likely that a string with 01 as the last two elements would be selected to replace the last four elements of S_1. The relevant difference between S_1 and S_2 is the defining length, which is 8 for S_1 $(9-1)$ and 1 for S_2 $(5-4)$.

Thus the greater the defining length, the greater the probability that the schema will be destroyed. In particular, if the crossover location is chosen from $(L-1)$ sites, the probability that a schema will be destroyed is $\frac{\delta}{(L-1)}$ and the probability that it will survive is

$$1 - \left(\frac{\delta}{L-1}\right). \tag{5.42}$$

Given that the probability that a string is selected for crossover is p_c, the probability of survival can be amended to $1 - p_c(\frac{\delta}{L-1}.)$. However, it is possible that because of the

particular positions of the particular numbers (0, 1s) in two strings that are to be crossed over, after crossover one of the schema matching the two strings will still match the child. For example, if the two strings to be mated began with 11 and ended with 01—and thus both are an example of S_1 above—then crossover at a point between the second 1 and before the 0 would give a child that still matched S_1. Therefore, the probability that a schema survives is at least $1 - p_c(\frac{\delta}{L-1}.)$.

We can now amend (5.41) to allow for crossover. $m(s, t+1)$ is the expected number of strings that match s after the creation of a new generation. Equation (5.41) allowed for selection only. However, when we also allow for crossover, the expected number of strings that match s in the new generation equals the number in the reproduced population multiplied by the chance that the string survives. Remembering the "at least" expression of the last paragraph, we can amend (5.41) to be

$$m(s, t+1) \geq m(s, t) \cdot \frac{f(s, t)}{\bar{f}} \cdot \left(1 - \left(\frac{p_c \delta}{L-1}\right)\right). \tag{5.43}$$

For a given string length $(L-1)$, given defining length (δ), and given probability of selection for crossover (p_c), our original interpretation of (5.41) applies. However, in addition, we can say that the expected number of strings that match schema s will increase between generations if the probability of crossover decreases and/or if the defining length of the strings in the solution decreases.

Equation (5.43) can be further amended to allow for mutation. (See a text on GAs for this amendment, for example, (Goldberg 1989).) Intuitively, it can be appreciated that as the order (i.e., number of fixed bit values) increases, given the probability that a bit is flipped, the lower the expected number of strings that match a schema in the following generation will be. There is a greater chance that one of the fixed bit values will be flipped and that the child fails to match schema s.

Equation (5.43), as amended by the preceding paragraph, allows us to conclude that the number of strings matching schemata with above-average fitness, which have a short defining length and low order, will increase exponentially as new generations of a given size are created. This is known as the schemata theorem.

5.6 Expert systems

In general terms, an expert system is a collection of processes that emulate the decision-making behavior of an expert. An expert system consists of a number of parts. First, there is a knowledge base, which usually consists of rules. Second is a series of facts that, in the inference engine, are matched with the rules to provide an agenda of recommended actions. This inference engine may provide an explanation for the recommendations and the system may have a facility for updating knowledge. The rules, often called production rules, may be of the IF . . . , THEN . . . form. For example, a rule might be

IF annual payments exceed 50% of annual income, THEN the loan will not be repaid.

The construction of the knowledge base is one of the earlier stages in the construction of an expert system. This is done by a knowledge engineer who repeatedly discusses with an expert the behavior and decisions made by the expert whose behavior is to be emulated. The knowledge engineer may represent the heuristics followed by an expert in symbolic logic. The rules may be extracted by the knowledge engineer directly or with the use of specialized software. An example of the derivation of such rules is the training of a neural network

so that when new facts—that is, application characteristics—are fed into it, a decision is given (Davis, Edelman, and Gammerman 1992). The rules created from a neural network are uninterpretable, and yet one aim of an expert system is to provide explanations for the actions recommended. Therefore, recent research in the area of knowledge acquisition, or machine learning, is in symbolic empirical learning (SEL), which is concerned with creating symbolic descriptions where the structure is not known in advance. An example of a SEL program is Quinlan's C5 (a development from his ID3), which constructs decision trees from input data (see section 4.7 above).

The expert system may consist of literally thousands of rules—for example, XCON, DEC's expert system for configuring computers, contained 7000 rules. In an expert system, the rules need not be sequential, and the matching of facts to rules is complex. Such matching is carried out by algorithms such as the Rete algorithm, devised by Forgy (1982).

From the foregoing, it can be seen that an expert system is particularly applicable when the decision maker makes decisions that are multiple and sequential or parallel and where the problem is ill-defined because of the multiplicity of decisions that can be made—in short, where it is not efficient to derive an algorithmic solution.

Relatively few examples of an expert system used for credit scoring have been published, and because the details of such systems are usually proprietary, none that have been published give exact details. However, an example by Talebzadeh, Mandutianu, and Winner (1994) describes the construction of an expert system, called CLUES, for deciding whether to underwrite a mortgage loan application, which was received by the Countrywide Funding Corporation.

They considered using expert systems, neural nets, mentoring systems, and case-based reasoning. They chose expert systems rather than neural networks, for example, because neural networks require large numbers of cases for retraining and so were less amenable to updating than an expert system into whose knowledge base additional rules could be inserted. In addition, unlike neural nets, expert systems could provide reasons for a recommendation. Loan applications to Countrywide contained 1500 data items, on the basis of which an underwriter decides whether to grant a loan by considering the strengths and weaknesses of the applicant. Knowledge engineers interviewed and observed underwriters as they decided whether to grant a loan.

Talebzadeh, Mandutianu, and Winner (1994) identified three types of analysis that each loan officer undertook: an examination of the ability of the borrowers to make the payments, an examination of the past repayment performance of the application, and consideration of the appraisal report. Eventually, the expert system had around 1000 rules, which evaluated each item within each of the three analyses, for example, assets and income.

Extensive comparisons between the decisions made by the system and those made by underwriters showed that virtually all the decisions made by CLUES were confirmed by the underwriters.

In another example of an expert system, Jamber et al. (1991) described the construction of a system by Dun and Bradstreet to produce credit ratings of businesses. After fine-tuning the rules, the system agreed with the experts in 98.5% of cases in a test sample. In another example, Davis, Edelman, and Gammerman (1992) achieved agreement between their system and that of loan officers for consumer loans in 64.5% of test cases, but their "expert system" was really a neural network applied to application data rather than the construction of a knowledge base from the behavior of experts. In a related context, Leonard (1993a, 1993b) constructed an expert system to predict cases of fraud among credit card users. His rule extraction technique was an algorithm by Biggs, De Ville, and Suen (1991), which constructs decision trees from raw financial data.

5.7 Comparison of approaches

In the last two chapters, a number of methods for developing credit-scoring systems were outlined, but which method is best? This is difficult to say because commercial consultancies have a predilection to identify the method they use as best. On the other hand, comparisons by academics cannot reflect exactly what happens in the industry since some of the significant data, like the credit bureau data, are too sensitive or too expensive to be passed on to them by the users. Thus their results are more indicative than definitive. Several comparisons of classification techniques have been carried out using medical or scientific data but fewer by using credit-scoring data.

Table 5.2 shows the results of five comparisons using credit-scoring data. The numbers should be compared across the rows but not between the rows because they involve different populations and different definitions of good and bad.

Table 5.2. *Comparison of classification accuracy for different scoring approaches.*

Authors	Linear reg.	Logistic reg.	Classification trees	LP	Neural nets	GA
Srinivisan (1987b)	87.5	89.3	93.2	86.1	—	—
Boyle (1992)	77.5	—	75.0	74.7	—	—
Henley (1995)	43.4	43.3	43.8	—	—	—
Yobas (1997)	68.4	—	62.3	—	62.0	64.5
Desai (1997)	66.5	67.3	—	—	66.4	—

The entries in the table are the percent correctly classified if the acceptance rate. However, there is no difference made between goods classified as bads and vice versa. In Srinivasan's comparison (1987b) and Henley's (1995) work, classification trees is just the winner; Boyle et al.'s (1992) and Yobas, Crook, and Ross's (1997) linear regression classifies best, while the paper by Desai et al. (1997) suggests logistic regression is best. However, in almost all cases, the difference is not significant.

The reason for the similarity may be the flat maximum effect. Lovie and Lovie (1986) suggested that a large number of scorecards will be almost as good as each other as far as classification is concerned. This means there can be significant changes in the weights around the optimal scorecard with little effect on its classification performance, and it perhaps explains the relative similarity of the classification methods.

This relative stability of classification accuracy to choice of method used has prompted experts to wonder if a scoring system is also relatively stable to the choice of customer sample on which the system is built. Can one build a scorecard on one group of customers and use it to score a different group of customers; i.e., are there generic scorecards? Platts and Howe (1997) experimented by trying to build a scorecard that could be used in a number of European countries. Overstreet, Bradley, and Kemp (1992) tried to build a generic scorecard for U.S. credit unions by putting together a sample of customers from a number of unions because each union did not have enough customers to build a stable scoring system. In both cases, the results were better than not using a scorecard at all, but the classification accuracy was well below what is usual for credit scorecards. Thus it does seem the systems are very sensitive to differences in the populations that make up the scorecard and suggest that segmenting the population into more homogeneous subgroups is the sensible thing to do. Further details of these generic scorecards are found in section 12.2.

If classification accuracy is not the way to differentiate between the approaches, what should be used? An obvious answer is the special features the different methods bring to the scoring process. The regression approaches, both linear and logistic, have all the

underpinning of statistical theory. Thus one can perform statistical tests to see whether the coefficients (the scores) of an attribute are significant, and hence whether that attribute should really be in the scorecard. Correlation analysis shows how strongly related are the effects of different characteristics and so whether one really needs both in the scorecard. The regressions can be performed using all the characteristics or by introducing them one by one, each time introducing the one that increases the existing scorecard's discrimination the most. Thus these tests allow one to identify how important the characteristics are to the discrimination and whether two characteristics are essentially measuring the same effect. This allows one to drop unimportant characteristics and arrive at lean, mean, and robust scorecards. It is also useful in identifying which characteristics should be included and which can be dropped when application forms are being updated or information warehousing systems are being developed.

The linear programming approach has to be extended to integer programming, which is impractical for commercial systems, to give similar tools on the relative importance of variables, as outlined in section 5.3. However, linear programming deals very easily with constraints that lenders might impose on the scorecard. For example, they may want to bias the product toward younger customers and so require the score for being under 25 to be greater than the score for being over 65. This is easily done in any of the linear programming formulations by adding the constraint $w(\text{under } 25) \geq w(\text{over } 65)$. One could require other constraints; for example, the weighting on residential states is no more than 10% of the total score, or the weighting on income must be monotone in income. Regression approaches find it almost impossible to incorporate such requirements. One of the other advantages of linear programming is that it is easy to score problems with hundreds of thousands of characteristics, and so splitting characteristics into many binary attributes causes no computational difficulties, whereas the statistical analysis of such data sets with large number of variables can cause computational problems.

The methods that form groups like classification trees and neural networks have the advantage that they automatically deal with interactions between the characteristics, whereas for the linear methods, these interactions have to be identified beforehand and appropriate complex characteristics defined. For example, Table 5.3 describes data on the percentage goods in different categories of residential status and phone ownership.

Table 5.3. *Percent of goods in residential status and phone ownership.*

	Own phone	**No phone**	
Owner	95%	50%	90%
Tenant	75%	65%	70%
	91%	60%	

Since the percentage of goods is greater among owners than tenants (90% to 70%) and among those with a phone than those without a phone (91% to 60%), the highest score in a linear system will be of owners who have a phone and the lowest of tenants who do not have a phone. In fact, owners who do not have a phone are significantly worse, and this would not be picked up by a linear system if it had not been identified beforehand. Classification trees and neural networks do tend to pick up such interactions. Some scorecard developers will use classification trees to identify major interactions and then segment on the characteristic involved in several such interactions (see section 8.6), i.e., in this example, they build separate linear scorecards for owners and for tenants. Age is one of the most common variables on which to segment.

Nearest-neighbor methods and genetic algorithms have been insufficiently used in practice to detail their extra features. However, it is clear that nearest-neighbor methods could be used to develop continuously updated scorecards. When the history on past customers is sufficiently long to make an accurate judgment on whether they are good or bad, then it can be added to the sample, while very old sample points could be deleted. Thus the system is being continuously updated to reflect the trends in the population. Genetic algorithms allow a wide number of scorecards to be considered in the construction of a fit population.

One final point to make in any comparison of methods regards how complex the lender wants the system to be. To be successfully operated, it may be necessary to sacrifice some classification accuracy for simplicity of use. One extreme of this is the scorecard developed for Bell Telephone and described by Showers and Chakrin (1981). The scorecard consisted of 10 yes-no questions and the cutoff was at least 8 yeses. Thus a scorecard with a very small number of binary attributes may be successfully implemented by nonexperts, while more complex scorecards will fail to be implemented properly and hence lose the classification accuracy built into them (Kolesar and Showers 1985).

Chapter 6

Behavioral Scoring Models of Repayment and Usage Behavior

6.1 Introduction

The statistical and nonstatistical classification methods described in the last two chapters can be used to decide whether to grant credit to new applicants and to decide which of the existing customers are in danger of defaulting in the near or medium-term future. This latter use is an example of behavioral scoring—modeling the repayment and usage behavior of consumers. These models are used by lenders to adjust credit limits and decide on the marketing and operational policy to be applied to each customer. One can also model consumers' repayment and usage behavior using Markov chain probability models. Such models make more assumptions about the way consumers behave, and thus they can be used to forecast more than just the default probability of the consumer.

In section 6.2, we explain how the classification methods are used to obtain behavioral scores. Section 6.3 describes the uses and variants of this form of behavioral score. Section 6.4 describes the Markov chain models used in behavioral scoring, where the parameters are estimated using data from a sample of previous customers. Section 6.5 explains how this idea can be combined with dynamic programming to develop Markov decision process models that optimize the credit limit policy. Section 6.6 considers some of the variants of these models, especially ones where the population is segmented and different models are built for each segment. It also looks at the way the parameters for the model can be estimated and at the tests that are used to check that the assumptions of the models are valid. Section 6.7 outlines the Markov chain models where the parameter values for an individual customer are estimated and updated using that individual's performance. This Bayesian approach to modeling contrasts with the orthodox approach of the models in sections 6.4 and 6.5.

6.2 Behavioral scoring: Classification approaches

The main difference between the classification approaches of behavioral scoring and credit scoring is that more variables are available in the former. As well as all the application-form characteristics and the data from the credit bureau (whose values are regularly updated into the scoring system), there are characteristics that describe the repayment and usage behavior of the customer. These are obtained from a sample of histories of customers as follows. A particular point of time is chosen as the observation point. A period preceding this point, say the previous 12 to 18 months, is designated the performance period, and the characteristics

of the performances in this period are added to the credit bureau and application information. A time—say, 12 months after the observation point—is taken as the outcome point, and the customer is classified as good or bad depending on their status at that time. Figure 6.1 shows the relationship between the observation and outcome points and the performance period.

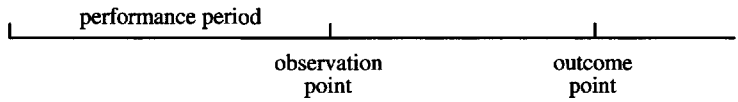


Figure 6.1. *Behavioral scoring time line.*

The extra performance variables in behavioral scoring systems include the current balance owed by the account and various averages of this balance. There will be similar records of the amount repaid in the last month, six months, etc., as well as the amount of new credit extended and the usage made of the account over similar periods. Other variables refer to the status of the account, such as the number of times it had exceeded its limits, how many warning letters had been sent, and how long since any repayment had been made. Thus there can be a large number of very similar performance variables with strong correlations. Usually, one chooses only a few of these similar variables to be in the scoring system. Thus a common first step is to apply a linear regression of the status of the account at the outcome point against these similar variables and to leave in only those that have the greatest impact. These can be either the ones with the most significant coefficients if all variables are in the regression or those that enter first if an iterative regression procedure is applied, with one new variable entering at each stage. For more details, see section 8.8, on choosing the characteristics.

The definition of the status of the account at the outcome point is usually compatible with that used in the application scoring system. Thus common definitions of bad are accounts that have three, possibly consecutive, months of missed payments during the outcome period.

6.3 Variants and uses of classification approach–based behavioral scoring systems

A very common variant of the classification-based behavioral scoring system outlined in the previous section is to segment the population and apply different scoring systems to the different segments. The population can be segmented on characteristics like age that have strong interactions with other characteristics. If one uses a classification tree, then the highest-level splits are good indications of what the appropriate segments of the population might be to deal with such interaction effects.

Another more obvious segmentation is between new and established customers. The former may have insufficient history for the performance variables to have values. Someone who opened an account within the last six months cannot have an average balance for the last year or the last half-year. A separate behavior scorecard is built for those with so little history. Not surprising, these scorecards put greater weight on the application-form data than do scorecards for more mature accounts. It is then necessary to calibrate the scorecards so that there are no discontinuities in score and hence in good:bad odds at the time when customers move from one scorecard to another. Ways to ensure consistency in calibrating the scorecard are outlined in section 8.12, and this consistency is required for all portfolios of segmented scorecards.

Another variant on the standard behavioral scoring methodology deals with the length of the outcome period. Since one is anxious to identify the behavior of all defaulters and not just these who are well on the way to defaulting, one wants to make this a reasonable period of time; 12 or 18 months is normal. However, with a performance period of 12 months, one is building a scoring system on a sample that is two to three years old. Population drift means that the typical characteristics of the population as well as the characteristics that are indicative of bad outcomes may start to change over such a period. To overcome this, some organizations take a two-stage approach, using an outcome point only six months after the observation point. First, they identify what behavior in a six-month period is most indicative of the customer going bad in the next 12 months, using the normal definition of bad. This is usually done using the classification tree approach on a sample with a suitably long history. This bad behavior in the six-month period is then taken as the definition of bad for a behavioral score using more recent data with an outcome period of just the last six months. Thus one can use data that are only 12 or 18 months old depending on the performance period used.

One of the problems with behavioral scoring is how the score should be used. Using the typical lengths of outcome period and definition of bad, the score is a measure of the risk that the customer will default in the next 12 months, assuming the operating policy is the same as that being used in the sample period. It can therefore be argued that using this score to change the operating policy invalidates the effectiveness of the score. Yet this is what is done when one uses the behavioral score to set the credit limits. The assumption is that those with higher behavioral scores, and hence less risk of defaulting at the current credit limit policy, should be given higher credit limits. This is analogous to saying that only those who have no or few accidents when they drive a car at 30 mph in the towns should be allowed to drive at 70 mph on the motorways. It does not accept that there might be other skills needed to drive faster. Similarly, there may be other characteristics that are needed to manage accounts with large credit limits, which are not required in managing accounts with limited available credit. If one accepts that this is a leap of faith, then it is a reasonable one to make, and one that many lenders are willing to accept. What is commonly done is to cut the behavioral score into a number of bands and to give a different credit limit to each band, increasing the limit with the behavioral score. Other organizations think of the behavioral score as a measure of the risk involved and take another measure, such as credit turnover, of the return involved. They then produce a matrix of behavioral score bands and credit turnover bands with different credit limits in each; see Table 6.1 as an example.

Table 6.1. *Example of a credit limit matrix.*

Monthly credit turnover / **Behavioral score**	**<£50**	**£50–£150**	**£150–£300**	**£300+**
<200	£0	£0	£500	£500
200–300	£0	£500	£1000	£2000
300–500	£2000	£3000	£3000	£5000
500+	£5000	£5000	£5000	£5000

This then raises the question of how these limits are chosen. Ideally, one would like to have a profit model, where one could put in the behavioral score as a measure of the riskiness of a customer and the variables that measure the profit of the customer and then find the optimal credit limit. Lenders do not usually have such models and so are reduced to setting

credit limits using hunches and experience. A more scientific approach advocated by Hopper and Lewis (1992) is the champion-versus-challenger approach, where they suggest trying out new credit policies on subsets of the customers and comparing the performance of these subsets with the group using the existing policy. This approach emphasizes the point that it takes time to recognize whether a new policy is better or worse than the old one. Increasing credit limits gives an immediate boost to the usage or sales and so apparently to the profit, but it takes another 6 to 12 months for the consequent increase in bad debt to become apparent. Thus such competitions should be kept going for at least 12 months.

The classification-based behavioral scores have other uses, such as estimating whether a customer will continue to use the credit line, whether they will close the account and move to another lender, and whether they can be persuaded to buy other products from the lender. Chapter 10 looks at some of these other applications in detail, but it is sufficient to say that in some cases one needs to know further information like the age of the account or when certain events occurred. Customers in the U.K. have, until the recent changes in car registration lettering, tended to change their cars about the same time of year every two, three, or four years. Thus knowing when the last car loan was taken out is likely to be a significant factor in whether a customer is thinking about getting another car loan.

6.4 Behavioral scoring: Orthodox Markov chain approach

The idea of building a model of the repayment and usage behavior of a consumer, as opposed to the black box approach of classification-based behavioral scoring, was first suggested in the early 1960s. The idea is to identify the different states that a borrower's account can be in, and then to estimate the chance of the account moving from one state at one billing period to another at the next. The states depend mainly on information concerning the current position of the account and its recent history, but they can also depend on the initial application information. Thus the current balance, the number of time periods overdue, and the number of reminder letters in the last six months might be typical information used. The object is to define states in which the probability of moving to any particular state at the next billing period is dependent only on the current state the account is in and not on its previous history. This is the definition of a Markov chain. A more formal definition follows.

Definition 6.1. *Let* $\{X_0, X_1, X_2, X_3, \dots\}$ *be a collection of random variables that take values in one of* M *states. The process is said to be a finite-valued Markov chain if*

$$\text{Prob}\{X_{t+1} = j | X_0 = k_0,\ X_1 = k_1, \dots, X_{t-1} = k_{t-1},\ X_t = i\} = P\{X_{t+1} = j | X_t = i\} \tag{6.1}$$

for all t *and* i, j, *where* $1 \le i, j \le M$. *The conditional probabilities* $P\{X_{t+1} = j | X_t = i\}$ *are called transition probabilities and represented* $p_t(i, j)$, *and the probability properties mean that one requires that* $p_t(i, j) \ge 0$ *and* $\sum_j p_t(i, j) = 1$.

The matrix of these probabilities is denoted P_t, so $(P_t)(i, j) = p_t(i, j)$. The Markov property (6.1) means that one can obtain the distribution of the X_t given the value of X_0 by multiplying the matrices $P_0, P_1, \dots, P_{t+1}$ together since

$$P\{X_{t+1} = j | X_0 = i\} = \sum_{k(1)\cdots k(t-1)} P\{X_{t+1} = j | X_t = k(t)\} P\{X_t = k(t) | X_{t+1} = k(t-1)\}$$
$$\cdots P\{X_2 = k(2) | X_1 = k(1)\} P\{X_1 = k(1) | X_0 = i\}$$

$$= \sum_{k(1)\cdots k(t-1)} p_0(i,k(1))p_1(k(1),k(2))\cdots p_t(k(t),j)$$
$$= (P_0 \cdot P_1 \cdot \cdots \cdot P_t)(i,j). \tag{6.2}$$

If the $p_t(i,j) = p(i,j)$ for all t, i, and j, the process is a stationary Markov chain. In this case, the k-stage transition probabilities are obtained by multiplying P by itself k times, so

$$P\{X_{t+1} = j | X_0 = i\} = \sum_{k(i)\cdots k(t-1)} p_0(i,k(1))p_1(k(1),k(2))\cdots p_t(k(t),j)$$
$$= (P \cdot P \cdot \cdots \cdot P)(i,j) = P^{t+1}(i,j). \tag{6.3}$$

If π_{t+1} is the distribution of X_{t+1}, so $\pi_{t+1}(i) = \text{Prob}(X_{t+1} = i)$, then (6.3) corresponds to

$$\pi_{t+1} = P\pi_t = P^{t+1}\pi_0 \tag{6.4}$$

in vector notation. For aperiodic Markov chains (ones where there is no $k > 2$, so the $p^n(i,j) \neq 0$ for some i, j only if k divides n exactly), in the long run π_t converges to a long-run distribution π^*. Replacing π_t and π_{t+1} by π^* in (6.4) shows that π^* must satisfy

$$\pi^* = P\pi^*. \tag{6.5}$$

The states of the Markov chain divide into persistent and transient ones. Persistent states i are ones where the chain is certain to return to and correspond to states where $\pi_i^* > 0$; i.e., there is a positive probability of being in them in the long run. Transient states i are ones where the chain has a probability of less than 1 of ever returning to and corresponding to states where $\pi_i^* = 0$.

Consider the following Markov chain model.

Example 6.1. One of the simplest examples of this type of model is to take the status of a credit account as one of the following states $\{NC, 0, 1, 2, \ldots, M\}$, where NC is no-credit status, where the account has no balance; 0 is where the account has a credit balance but the payments are up to date; 1 is where the account is one payment overdue; i is where the account is i payments overdue; and one assumes M payments overdue is classified as default. The transition matrix of the Markov chain would be

From/To	***NC***	**0**	**1**	**2**	**...**	***M***
NC	$p(NC,NC)$	$p(NC,0)$	0	0	...	0
0	$p(0,NC)$	$p(0,0)$	$p(0,1)$	0	...	0
1	$p(1,NC)$	$p(1,0)$	$p(1,1)$	$p(1,2)$	...	0
2	$p(2,NC)$	0	$p(2,1)$	$p(2,2)$	...	0
...	...	...	...	...	...	...
M	$p(M,NC)$	$p(M,0)$	0	0	...	$p(M,M)$

(6.6)

This is the model described by Kallberg and Saunders (1983), in which the accounts jump from state to state. The data in their example led to that stationary transition matrix

From/To	***NC***	**0**	**1**	**2**	**3**
NC	0.79	0.21	0	0	0
0	0.09	0.73	0.18	0	0
1	0.09	0.51	0	0.40	0
2	0.09	0.38	0	0	0.55
3	0.06	0.32	0	0	0.62

(6.7)

Thus if one starts with all the accounts having no credit $\pi_0 = (1, 0, 0, 0, 0)$, after one period the distribution of accounts is $\pi_1 = (0.79, 0.21, 0, 0, 0)$. After subsequent periods, it becomes

$$\begin{aligned} \pi_2 &= (0.64, 0.32, 0.04, 0, 0), \\ \pi_3 &= (0.540, 0.387, 0.058, 0.015, 0), \\ \pi_4 &= (0.468, 0.431, 0.070, 0.023, 0.008), \\ \pi_5 &= (0.417, 0.460, 0.077, 0.028, 0.018), \\ \pi_{10} &= (0.315, 0.512, 0.091, 0.036, 0.046). \end{aligned} \tag{6.8}$$

This proves a useful way for estimating the amount of bad debt (state 3) that will appear in future periods. After 10 periods, it estimates that 4.6% of the accounts will be bad. This approach seems to be more robust to variations in sales volumes than more standard forecasting techniques, which use lagged values of sales volumes in their forecasts.

One of the first Markov chain models of consumer repayment behavior was a credit card model suggested by Cyert, Davidson, and Thompson (1962), in which each dollar owed jumped from state to state. The Cyert model had difficulties, however, with accounting conventions. Suppose an account was £20 overdue—£10 was three months overdue and £10 was one month overdue. If the standard payment is £10, and £10 is paid this month (so the account is still £20 overdue), is £10 of this four months overdue and £10 two months overdue or is £10 two months overdue and £10 one month overdue? This problem was addressed by van Kueler, Spronk, and Corcoran (1981), who used a partial payment of each amount compared with paying off the oldest debt suggested by Cyert, Davidson, and Thompson (1962). Other variants of the model were suggested by Corcoran (1978), who pointed out that the system would be even more stable if different transition matrices were used for accounts of different-size loans. Two similar models were investigated by Kallberg and Saunders (1983), in which the state space depends on the amount of the opening balance and whether payments made were (1) less than the total outstanding balance but greater than the required amount, (2) within a small interval of the required amount, (3) some payment but significantly less than the required amount, or (4) no payment at all.

One can define much more sophisticated Markov chain models in which each state s in the state space has three components, $s = (b, n, i)$, where b is the balance outstanding, n is the number of current consecutive periods of nonpayment, and i represents the other characteristics of importance. If one defines the expected one-period reward the lender makes from a customer in each state, one can calculate the expected total profit from the customer under any credit limit policy. In fact, one can go further and calculate the credit limit policy that maximizes the rewards for a given level of bad debt or minimizes the bad debt for a given level of reward. To see this, consider the following examples in the mail-order context.

Example 6.2. Let the states of consumer repayments be $s = (b, n, i)$, where b is the balance outstanding, n is the number of periods of nonpayment, and i are the other characteristics. From the sample of customer histories, we estimate the following:

$t(s, a)$ the probability an account in state s repays a next period;

$w(s, i')$ the probability an account in state s changes its other characteristics to i' next period;

$r(s)$ the expected value of orders placed per period by those in state s.

If no more orders were allowed, we can use dynamic programming to calculate the chance of defaulting, $D(s)$, given the account is in state s. We assume that default corresponds to

N successive periods of nonpayment. The means that

$$\begin{aligned} D(b, N, i) &= 1 \quad \text{for all } b, i, j, \\ D(0, n, i) &= 0 \quad \text{for all } i, n < 1, \end{aligned} \tag{6.9}$$

whereas

$$D(b, n, i) = \sum_{i', a \neq 0} t(s, a) w(s, i') D(b - a, 0, i') + \sum_{i'} t(s, 0) w(s, i') D(b, n + 1, a). \tag{6.10}$$

This follows since if the system is in state (b, n, i) and there is a payment of a and if i changes to i', which occurs with probability $t(s, a)w(s, i)$, the state of the system becomes $(b - a, 0, i)$. With probability $t(s, 0)w(s, i')$, there is no payment and the state moves to $(b, n + 1, i')$. One can use (6.8) and (6.10) to calculate $D(b, n, i)$ for each state since, as Figure 6.2 shows, the jumps in state move the state either closer to the bottom of the grid or closer to the right-hand side, and the value $D(b, n, i)$ is known on these line edges.

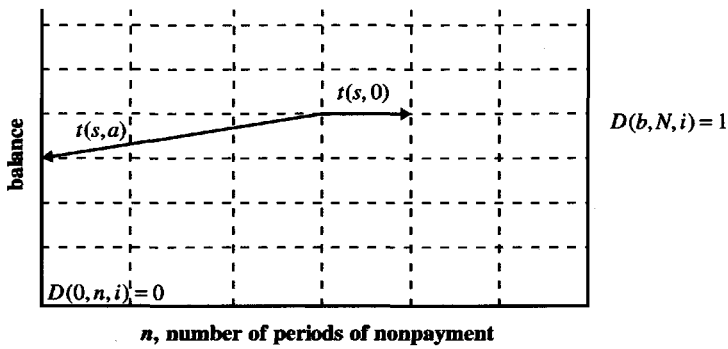


Figure 6.2. *Calculation of D.*

It is obvious that the chance of default $D(b, n, i)$ increases as b increases—if n and i are kept fixed—since there have to be more repayments before the balance is paid off.

Using past data, we are able to calculate the default probability for each state using the above calculations and also the expected value of the orders in each state. The states can then be ordered in increasing probability of default, say, for example, Table 6.2, which is a hypothetical nine-state example, where the total expected value of orders is £1,000. The lenders then have to decide what default level and value of orders lost is acceptable. If they say a default level of .1 is acceptable, then they will accept orders in the states up to s_8 and reject those in s_4, s_6, and s_5 and so lose $\frac{60+40+50}{1000} = 15\%$ of the orders. If they say that a level of .025 is acceptable, then they will reject orders from those in states s_9, s_8, s_4, s_6, and s_5 and so lose $\frac{120+80+60+40+50}{1000} = 25\%$ of the orders. Thus management has to choose which default level D^* is acceptable. Since $D(b, n, i)$ is increasing in b, one can solve

$$D(L(n, i), n, i) = D^* \tag{6.11}$$

to find the credit limit $L(n, i)$ for each state (n, i).

Table 6.2. *Value and default probability at each state.*

State	s_7	s_1	s_3	s_2	s_9	s_8	s_4	s_6	s_5
Default probability	.001	.004	.008	.02	.06	.09	.12	.15	.2
Value of orders (£)	50	150	250	200	120	80	60	40	50

6.5 Markov decision process approach to behavioral scoring

The last Markov chain model of the previous section had two elements—states, describing the repayment performance of the consumers with a transition matrix describing the dynamics of the repayment behavior, and a reward function, describing the value to the organization of the consumer being in that state. If one adds a third element—decisions the lender can make that impact on the rewards and the transitions between the states—one has a Markov decision process that is an example of stochastic dynamic programming.

Formally, a Markov decision process is a stochastic process X_t, $t = 1, 2, \ldots$, taking values in a state space S. For each state $i \in S$, there is a set of actions $k \in K_i$, and one of these has to be chosen each time the process enters a state. The result of choosing action k in state i is that there is an immediate reward $r^k(i)$, and with probability $p^k(i, j)$ the system moves into state j at the next period. The objective is to choose the actions $k \in K_i$ so as to maximize some function of the rewards. This could be the reward over a finite number of periods or the reward over an infinite number of periods. In the latter, one criterion is the expected discounted total reward. In this, one takes the net present value of the future rewards, as developed in (3.1) and (3.2), and writes the discounting term as β, where $\beta = \frac{1}{1+i}$ if i is the interest rate given. Thus the rewards in period two are discounted by a factor β, those in period three by a factor β^2, and so on. The second possible criterion in the infinite horizon case is the expected average reward. This is the limit as n goes to infinity of the average reward over the first n periods. Whatever criterion is used, under very mild conditions, which always hold if the state space is finite and the number of actions per state is finite (see Puterman 1994 for details), the optimal value and the optimal actions satisfy what is known as the Bellman optimality equation. For example, if $v_n(i)$ is the optimal total reward over n periods with the system starting in state i, then

$$v_n(i) = \max_{k \in ki} \left\{ r^k(i) + \sum_j p^k(i, j) v_{n-1}(j) \right\} \quad \text{for all } i \in S \text{ and all } n = 1, 2, \ldots. \tag{6.12}$$

Similarly, if β is the discount factor and $v(i)$ is the optimal expected discounted reward with the system starting in state i, then

$$v(i) = \max_{k \in ki} \left\{ r^k(i) \sum_j p^k(i, j) \beta v(j) \right\} \quad \text{for all } i \in S. \tag{6.13}$$

Standard analysis (see Puterman 1994) shows that the actions that maximize the right-hand side of (6.13) make up the optimal policy for the discounted infinite horizon problem, while the maximizing actions of the right-hand side of (6.12) form the optimal policy for maximizing the total reward over a finite horizon. Thus solving the optimality equations (6.12) and (6.13) will give both the optimal values and the optimal actions. The finite horizon optimality equations (6.12) can be solved by first deciding what the boundary conditions of

$v_0(.)$ should be (normally $v_0(.) = 0$). Then one can solve for $v_1(i)$. Having solved for $v_1(.)$, this allows the equation for $v_2(.)$ to be solved and this process can be repeated to obtain $v_n(.)$ for all n. This process of value iteration can also be used for solving the infinite horizon discounted reward problems with optimality equation (6.13). One solves the following set of equations iteratively:

$$v_n(i) = \max_{k \in K} \left\{ r^k(i) + \sum_{j \in S} p^k(i, j) \beta v_{n-1}(j) \right\} \quad \text{for all } i \in S, \quad n = 1, 2, \ldots. \tag{6.14}$$

It can be shown that the solutions $v_n(0)$ converge to $v(0)$, the solution of (6.13). Similarly, as Puterman (1994) proved, after some value n^*, all the iterations $n \geq n^*$ have maximizing actions that also maximize the infinite horizon problem.

Returning to the models of the previous section, we can now extend them to become Markov decision processes. Consider the following extension of Example 6.2.

Example 6.3. The states of consumer repayment behavior are $s = (b, n, i)$, where b is the balance outstanding, n is the number of periods since the last payment, and i is any other information. The actions are the credit limit L to set in each state. The Markov decision process behavioral scoring system is well determined once we have defined $p^L(s, s')$ and $r^L(s)$. Define $p^L(s, s')$ by estimating the following from past data:

$t^L(s, a)$ the probability account in state s with credit limit L repays a next period;

$q^L(s, e)$ the probability account in state s with credit limit L orders e next period;

$w^L(s, i')$ the probability account in state s with credit limit L changes to information state i'.

Then

$$\begin{aligned}
p^L(b, n, i; b + e - a, 0, i') &= t^L(s, a) q^L(s, e) w^L(s, i'), \quad b + e - a \leq L, \quad a > 0, \\
p^L(b, n, i; b - a, 0, i') &= t^L(s, a) \left(q^L(s, 0) + \sum_{e \geq L - b + a} q^L(s, e) \right) w^L(s, i'), \quad a > 0, \\
p^L(b, n, i; b + e, n + 1, i') &= t^L(s, 0) q^L(s, e) w^L(s, i'), \quad b + e \leq L, \\
p^L(b, n, i; b, n + 1, i') &= t^L(s, 0) \left(q^L(s, 0) + \sum_{e \geq L - b} q^L(s, e) \right) w^L(s, i').
\end{aligned} \tag{6.15}$$

The reward to the firm is the proportion f of the purchase amount that is profit for the lender, less the amount lost when a customer defaults. If we assume that N consecutive periods of nonpayment is what the company considers as default, then

$$r^L(b, n, i) = f \sum_e e q^L(s, e) - b t^L(s, 0) \delta(n - (N - 1)). \tag{6.16}$$

If one assumes profits are discounted by β each period, then let $v_n(s)$ be the optimal total discounted expected profit over an n-period horizon starting in state s. The optimality equation that $v_n(s)$ must satisfy is

$$v_n(s) = \max_L \left\{ r^L(s) + \sum_{s' \in S} p^k(s, s') \beta v_{n-1}(s') \right\} \quad \text{for all } s \in S, \quad n = 1, 2 \ldots. \tag{6.17}$$

This would give a credit limit policy where the limit L is a function of the state s and the number of periods n until the end of the time horizon. To make the policy independent of the period, one needs to consider the infinite horizon expected discounted profit criterion, where the optimal value $v(s)$ starting in state s satisfies

$$v(s) = \max_{L} \left\{ r^{L}(s) + \sum_{s' \in S} p^{k}(s, s') \beta v(s') \right\} \quad \text{for all } s \in S, \quad n = 1, 2, \ldots. \tag{6.18}$$

The actions that maximize the right-hand side of the (6.18) gives the optimal credit limits that maximize profit.

6.6 Validation and variations of Markov chain models

The Markov chain and Markov decision process models of the last two sections make more assumptions about the form of consumer repayment behavior than do the statistical classification models. One still needs, however, to use the data on previous applicants to estimate the parameters of the probability model chosen as well as check that the data support the assumptions of the model. Following are the estimators used to estimate the parameters and the tests needed to check the assumptions of the model.

6.6.1 Estimating parameters of a stationary Markov chain model

Suppose one takes the simplest model where the state space of the Markov chain is S which has m values and the transition probabilities from one state to another are the same for all time periods; i.e., the chain is stationary. Thus one needs to estimate $p(i, j), i, j \in S$, from a sample of histories of R previous customers. The rth customer's history is $s_0^r, s_1^r, s_2^r, \ldots, s_T^r$.

Let $n^r(i)$ be the number of times state i appears in this sequence from times 0 to $T-1$, and let $n^r(i, j)$ be the number of times the sequence i followed by j appears in times 1 to T. Let $n(i) = \sum_{r \in R} n^r(i)$ and $n(i, j) = \sum_{r \in R} n^r(i, j)$ be the total number of times state i and sequence i, j appeared in times 1 to T of all histories. Bartlett (1951) and Hoel (1954) showed that the maximum likelihood estimate of $p(i, j)$ is

$$\hat{p}(i, j) = \frac{n(i, j)}{n(i)}. \tag{6.19}$$

This is the obvious estimate.

6.6.2 Estimating parameters of nonstationary Markov chains

If one believes that the process is a Markov chain but the transition probabilities vary depending on the period under question, one has a nonstationary Markov chain. In this case, $p_t(i, j)$ is the probability the chain moves from state i to state j in period t. If one has a sample of histories of previous customers, let $n_t(i), i \in S$, be the number who are in state i at period t, whereas let $n_t(i, j)$ be the number who move from i at period t to state j at the next state. The maximum likelihood estimator of $p_t(i, j)$ is then

$$\hat{p}_t(i, j) = \frac{n_t(i, j)}{n_t(i)}. \tag{6.20}$$

Models in which the transition matrix changes every period need lots of data to estimate the parameters and are not very useful for future predictions. It is more common to assume the chain has a seasonal effect, so t is indexed by the month or the quarter of the year. Hence we would estimate $p_{\text{Jan}}(i, j)$ by using (6.20) over all cases when $t =$ January. Other models imply that the trend is more important than the season, so one assumes the transition matrix varies from year to year but stays constant within the year. Hence one assumes that there is a $p_{1999}(i, j)$, $p_{2000}(i, j)$, etc., and uses (6.20), where t is all the 1999 periods, to estimate $p_{1999}(i, j)$.

6.6.3 Testing if $p(i, j)$ have specific values $p^0(i, j)$

If one wants to test if $p(i, j)$ have specific values $p^0(i, j)$ for all $j \in S$ (recall that S has m values), one can use the result (Anderson and Goodman 1957) that

$$\sum_{j \in S} \frac{n(i)(\hat{p}(i, j) - p^0(i, j))^2}{p^0(i, j)} \tag{6.21}$$

has a χ^2 asymptotic distribution with $m - 1$ degrees of freedom. Thus one would accept this hypothesis at a significance level α (say, 95%) if (6.21) has a value that is less than the α-significant part of the χ^2 distribution with $m - 1$ degrees of freedom.

Since the variables $n(i)(\hat{p}(i, j) - p^0(i, j))^2$ for different i are asymptotically independent, the test quantities (6.21) can be added together and the result is still a χ^2 variable with $m(m - 1)$ degrees of freedom. Thus if one wants to check whether $\hat{p}(i, j) = p^0(i, j)$ for all i, j, one calculates

$$\sum_{i \in S} \sum_{j \in S} \frac{n(i)(\hat{p}(i, j) - p^0(i, j))^2}{p^0(i, j)} \tag{6.22}$$

and tests it using the χ^2 variable tests with $m(m - 1)$ degrees of freedom.

6.6.4 Testing if $p_t(i, j)$ are stationary

Checking whether rows of transition matrices at different periods are essentially the same is similar to checking on the homogeneity of contingency tables, and thus the χ^2 tests of the latter work in the former case (Anderson and Goodman 1957).

Thus for a given state i, we are interested in whether the hypothesis that the ith row of the t-period transition matrix $p_t(i, j)$, $j = 1, \ldots, m$, is the same for all periods t; i.e., $p_t(i, j) = p(i, j)$, $j = 1, \ldots, m$. Let $\hat{p}_t(i, j)$, $\hat{p}(i, j)$, $j = 1, 2, \ldots, m$, be the estimates for these probabilities obtained by (6.19) and (6.20). Then to test the hypothesis, calculate

$$\sum_{t=1}^{T-1} \sum_{j \in S} \frac{n_t(i)[\hat{p}_t(i, j) - \hat{p}(i, j)]^2}{\hat{p}(i, j)}. \tag{6.23}$$

If the null hypothesis is true and there are $T + 1$ periods in total $(0, 1, 2, \ldots, T)$, this has the χ^2 distribution with $(m - 1)(T - 1)$ degrees of freedom.

The previous test looked at the time independence of transitions from state i only, but normally one would like to check that the whole transition matrix is time independent. This corresponds to the null hypothesis that $p_t(i, j) = p(i, j)$ for all $i = 1, \ldots, m, j = 1, \ldots, m$,

and $t = 0, \ldots, T-1$. Since $p_t(i, j)$ and $p(i, j)$ for different values of i are asymptotically independent random variables, one can also show that

$$\chi^2 = \sum_{i \in S} \chi_i^2 = \sum_{i \in S} \sum_{t=1}^{T-1} \sum_{j \in S} \frac{n_t(i)[\hat{p}_t(i, j) - \hat{p}(i, j)]^2}{\hat{p}(i, j)} \tag{6.24}$$

has a χ^2 distribution with $m(m-1)(T-1)$ degrees of freedom, and one can apply the appropriate χ^2 test.

An alternative approach is to recognize that

$$L_i = -2 \log \prod_{t=0}^{T-1} \prod_{j=1}^{m} \left(\frac{\hat{p}(i, j)}{\hat{p}_t(i, j)} \right)^{n_t(i,j)} \tag{6.25}$$

is χ^2 with $(m-1)(T-1)$ degrees of freedom as a test for the homogeneity of row i. $\sum_{i \in S} L_i$ has a χ^2 distribution with $m(m-1)(T-1)$ degrees of freedom and so can be used to test if the whole matrix is stationary. Notice that if the matrix is nonstationary, then we cannot estimate the transition matrix for the current year until several months into the year. In addition, forecasting future years can be done only if it is assumed they are like certain previous years.

Thus far, we have assumed that the process is Markov so that all we need to estimate the transition probabilities $p(i, j)$ for this period is the current state of the system i. If this were not the case, we would need to know the history of the process and in particular what previous state k it was in. This is a way to check on the Markovity of the chain.

6.6.5 Testing that the chain is Markov

Take a sample of histories of previous customers and define $n_t(i)$, $n_t(i, j)$ to be the number of times that a customer was in state i at time t and the number of times that a customer was in state i at time t followed by moving to j at $t+1$. Similarly define $n_t(i, j, k)$ to be the number of times that a customer was in state i at time t followed by being in j at time $t+1$ and k at time $t+2$. For a stationary chain, estimate that

$$\hat{p}(i, j, k) = \frac{\sum_{t=0}^{T-2} n_t(i, j, k)}{\sum_{t=0}^{T-2} n_t(i, j)}. \tag{6.26}$$

These are estimators of $p(i, j, k) = p_{ijk}$, the probability of moving from i to j to k. The Markovity of the chain corresponds to the hypothesis that $p_{1jk} = p_{2jk} = \cdots = p_{mjk} = p_{jk}$ for $j, k = 1, 2, \ldots, m$. One can again make parallels with contingency tables, saying that for a fixed j, etc., the rows $p_{1jk}, p_{2jk}, \ldots, p_{mjk}$ are homogeneous. Thus a χ^2 test again works. Let

$$\chi_j^2 = \sum_{i \in S} \sum_{K \in S} \frac{n^*(i, j)[\hat{p}(i, j, k) - \hat{p}(j, k)]^2}{\hat{p}(j, k)}, \tag{6.27}$$

where

$$\hat{p}(j, k) = \frac{\sum_{t=1}^{T-1} n_t(j, k)}{\sum_{t=1}^{T-1} n_t(j)}$$

and

$$n^*(i, j) = \sum_{t=1}^{T-1} n_t(i, j).$$

Then it has a χ^2 distribution with $(m-1)^2$ degrees of freedom.

This test checks on the non-Markovity or otherwise of state j. To check that the whole chain is Markov, i.e., $p_{ijk} = p_{jk}$ for all $i, j, k = 1, 2, \ldots, m$, one computes

$$\chi^2 = \sum_j \chi_j^2 = \sum_{i,j,k} \frac{n^*(i, j)[\hat{p}(i, j, k) - \hat{p}(j, k)]^2}{\hat{p}(j, k)} \tag{6.28}$$

and checks whether this value falls inside or outside the significant value for a χ^2 test with $m(m-1)^2$ degrees of freedom.

If it is found the chain is not Markov, one can recover a useful model of consumer behavior in one of two ways. One can either complicate the state space by including information about previous states visited as part of the current state description or segment the population into separate groups and build a Markov chain model for each group. In the first approach, if the consumer is thought of as being in one of the states i in S at each period, then the state of the system at period t would be extended to be of the form $(i(t), i(t-1), \ldots, i(t+1-r))$, where $i(s)$ is the state the consumer was in at period s. Thus the state includes the consumer's current repayment behavior and their repayment behavior in the previous $r-1$ periods. One can then apply the estimates (6.19) and (6.20) and the tests (6.27) and (6.28) to check if the chain is Markov on this extended state space. This really corresponds to estimating the probabilities $p(i_1, i_2, \ldots, i_r)$ that the process goes through the sequence $i_1, i_2, \ldots, i_r$ of states and checking whether these probabilities are independent of the first components (see Anderson and Goodman 1957). If one requires the state at time t to include $(i(t), i(t-1), \ldots, i(t+1-r))$, the Markov chain is said to be of order r. (A normal Markov chain is of order 1.)

One can segment the population into a number of groups in different ways, for example, by splitting on characteristics like age or amount of data available, and thus have different chains for new customers and established customers. Edelman (1999) developed different chains for different products and different months of the year. He then looked to see whether it was better to segment by product or by period. Another segmentation that has received some attention because it has proved very successful in labor mobility and in consumer brand preferences is the mover-stayer model.

6.6.6 Mover-stayer Markov chain models

In the mover-stayer model, the population is segmented into two groups—stayers, who never leave their initial states, which is usually up-to-date repayment, and movers, who make transitions according to a Markov chain. The assumption is that there is a whole group of people who will never leave the "nonrisky" safe states, while others are more volatile in their behavior. Assume that the movers move between states according to a Markov chain with transition matrix P, whose entries are $p(i, j)$, $i, j \in S$. Assume that $s(i)$ is the percentage of the population who are stayers in state i. Hence let $s = \sum_{i \in S} s(i)$ be the proportion of stayers. Then if D is the diagonal matrix with entries $(s(1), s(2), \ldots, s(m))$, the one-period transitions are given by the matrix

$$D + (1-s)P. \tag{6.29}$$

However, the two-period transitions (changes in state two periods apart) are given by the matrix $D + (1 - s)P^2$ and the t-period transitions by $D + (1 - s)P^t$.

If one wishes to fit such a model to data on previous customers, one needs to estimate $s(i), i \in S$, and $p(i, j), i, j \in S$. These estimations were calculated in two papers (Frydman, Kallberg, and Kao 1985 and Frydman 1984). Suppose one has the credit history for customers over $T + 1$ periods, $t = 0, 1, 2, \ldots, T$, and $n_t(i), n(i) = \sum_{t=0}^{T-1} n_t(i), n_t(i, j)$, and $n(i, j) = \sum_{t=0}^{T-1} n_t(i, j)$ are defined, as before, as the number of customers in state i at time t, the total number of visits by customers to state i in periods 0 to $T - 1$, the number of i, j transitions made at time t, and the total number of i, j transitions made, respectively. Let n be the total number of customers in the sample and let $s(i)$ be the proportion of customers who are in state i for all $T + 1$ periods of the credit history. The estimators of the $s(i)$ and $p(i, j)$ are not straightforward because there is a probability $(p(i, i))^T$ that a mover who starts in state i in period 1 will stay there for the rest of the time and so has the same behavior as a stayer. If $\hat{p}(i, i)^T$ is the estimator for that behavior, it would then seem reasonable that the maximum likelihood estimator $\hat{s}(i)$ for $s(i)$ would satisfy

$$\hat{s}(i) = \begin{cases} \left(\dfrac{ns(i) - n_0(i)(\hat{p}(i, i))^{T-1}}{n_0(i)(1 - \hat{p}(i, i)^{T-1})} \right) & \text{if } ns(i) - n_0(i)\hat{p}(i, i)^T > 0, \\ 0 & \text{otherwise.} \end{cases} \tag{6.30}$$

$\hat{p}(i, i)$ satisfies

$$\begin{aligned} &\hat{p}(i, i)(n(i) - Tns(i)) \\ &\quad = (n(i, i) - Tns(i)) + (\hat{p}(i, i))^T((n(i) - Tn_0(i))\hat{p}(i, i) - n(i, i) + Tn_0(i)) \end{aligned} \tag{6.31}$$

(see Frydman et al. 1985). If T is very large and thus $\hat{p}(i, i)^T$ goes to zero, the second term on the right-hand side disappears, and then one can think of $n(i) - Tns(i)$ as the number of times that movers are in state i and $(n(i, i) - Tns(i))$ as the number of times that movers move from i to i. This extra term is a compensation because with finite T, one may be misclassifying some movers as stayers. Having solved (6.31) to obtain $\hat{p}(i, i)$, we can compute $\hat{p}(i, k)$ iteratively beginning with $k = 1$ by

$$\hat{p}(i, k) = \frac{n(i, k)\left(1 - \hat{p}(i, i) - \sum_{r=1, r\neq i}^{k-1} \hat{p}(i, r)\right)}{\sum_{r=k, r\neq i}^{m} n(i, r)}. \tag{6.32}$$

Having obtained these estimates—no matter how unpleasant they are—one can now test whether the new model is an improvement on the standard Markov chain and use likelihood ratio tests; see (Frydman, Kallberg, and Kao 1985), for example.

6.7 Behavioral scoring: Bayesian Markov chain approach

Section 6.4 looked at behavioral scoring systems based on Markov chain models, where the parameters were estimated from samples of previous customers. This is very much the orthodox statistical approach, where one assumes that such probabilities are fixed and are the same for groups of similar customers. The alternative approach, Bayesian statistics, is to assume that such probabilities are subjective and one's view on their value is likely to change as new information becomes available. In behavioral scoring systems, this implies that the probability of a consumer repaying an amount in the next period depends very much on that

consumer's history and will change from period to period depending on whether repayment was made. The parameters of the model are automatically updated in light of payment or nonpayment each period, and in the long run the parameters of an individual customer will depend totally on the credit history of that customer.

Example 6.4 (fixed-amount repayment loan). Consider the simple case in which there is a fixed amount a of a loan that has to be paid each period. Thus the only unknown is whether there will be a repayment next period. Hence one has a Bernoulli random variable, $X = 1$ if repayment and 0 if no repayment, and one has to estimate $p = \text{Prob}(X = 1)$. At any point, the state of the system must reflect one's belief of the distribution of the values of p. In Bayesian statistics, one chooses beliefs that are part of a family of distributions so that when new information appears, the updated belief distribution will still be part of that family. These families are the conjugate distributions for the variable that one is trying to estimate. In the case of a Bernoulli variable, the beta distributions are the family of conjugate distributions.

Using a slightly different parameterization from normal, $B(r, n)$, a beta distribution with parameters (r, n), where $0 \le r \le n$, has density function

$$f_{r,m}(p) = \frac{(m-1)!}{(r-1)!(m-r-1)!} p^{r-1}(1-p)^{m-r-1}, \quad 0 \le p \le 1; \quad 0 \text{ outside this range.} \tag{6.33}$$

Suppose that the initial (prior) belief f_0 about the value of p is $B(r, m)$ and then a repayment is made. Then the belief at the start of the next period f_1 should satisfy

$$f_1(p) = \frac{P(X_1 = 1|p) f_0(p)}{P(X_1 = 1)} = \frac{p\left(\frac{(m-1)!}{(r-1)!(m-r-1)!} p^r (1-p)^{m-r}\right)}{P(X_1 = 1)}. \tag{6.34}$$

Since

$$P(X_1 = 1) = \frac{(m-1)!}{(r-1)!(m-r-1)!} \int_0^1 p \cdot p^{r-1}(1-p)^{m-r-1} dp = \frac{r}{m},$$

then

$$f_1(p) = \frac{m!}{r!(m-r-1)!} p^{r+1}(1-p)^{m-r};$$

i.e., the new distribution is $B(r+1, m+1)$. Similarly, if there is no payment,

$$f_1(p) = \frac{P(X_1 = 0|p) f_0(p)}{P(X_1 = 0)} = \frac{(1-p)\left(\frac{(m-1)!}{(r-1)!(m-r-1)!} p^r (1-p)^{m-r}\right)}{P(X_1 = 0)}. \tag{6.35}$$

Since $P(X_1 = 0) = \frac{m-r}{m}$, then

$$f_1(p) = \frac{m!}{(r-1)!(m-r)!} p^r (1-p)^{m+1-r}.$$

This is the density function for $B(r, m+1)$. With this parameterization, one can think of m as the number of payment periods for which there is history and r as the number of payments in that time.

Thus in the repayment loan case, the state of the customer is given as (b, n, r, m), where b is the balance still left to pay, n is the number of periods since the last payment, and the belief about the probability of repayment next period is a beta distribution, $B(r, m)$. Let $D(b, n, r, m)$ be the probability the customer will default on the loan if the repayment amount is a. Then the transitions in the Markov chain mean that this default probability satisfies

$$D(b,n,r,m) = \frac{r}{m} D(b-a, 0, r+1, m+1) + \left(1 - \frac{r}{m}\right) D(b, n+1, r, m+1)$$
$$\text{for } n = 0, 1, 2, \ldots, N, \quad 0 \le r \le m, \tag{6.36}$$
$$\text{with } D(0,n,r,m) = 0 \quad \text{and} \quad D(b,N,r,m) = 1.$$

The boundary conditions follow if one takes default to be N consecutive periods of nonpayment. In a sense, we have a model very similar to that in Example 6.3; see Figure 6.2. The difference, as Figure 6.3 implies, is that the probability of the jumps varies from period to period according to the history of the process.

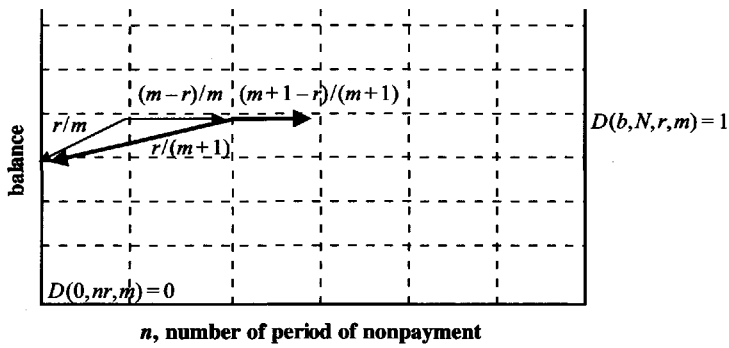


Figure 6.3. *Calculation of* $D(b, n, r, m)$.

To implement this model, one has to decide what is the prior distribution on the repayment probability at the beginning of the loan—say, (r_0, n_0). One could take it to be an uninformative prior $r_0 = 1, n_0 = 2$, which says all probabilities are equally likely. Better would be to perform some credit scoring and relate (r_0, n_0) to the credit score with $\frac{r_0}{n_0}$ increasing as the credit score increases. Whatever is chosen, after m periods in which there have been r repayments, the belief about the repayment probability is $B(r + r_0, m + m_0)$, where the actual repayment history of the individual (r, m) dominates the initial condition (r_0, m_0).

Example 6.5 (Bierman–Hausman charge card model). The first Bayesian model of repayment behavior was suggested by Bierman and Hausman (1970) in the context of a charge card, for which the amount borrowed has to be completely repaid each period before more can be borrowed the next month. They modeled this as a Markov decision process, where the state space is (r, m), the parameter of the beta distribution describing the belief of the repayment probability. Let P be the profit obtained this period when credit is given and collected and let D $(D < 0)$ be the loss (in current terms) when credit is given but not collected. If costs and profits are discounted by β each period (Srinivasan and Kim (1987a) describe a more realistic discounting), then $v(r, m)$, the maximum expected total profit that

can be obtained from a customer in state (r, m), satisfies

$$v(r,m) = \max\left\{\frac{r}{m}\left(P + \beta v(r+1, m+1) + \left(1 - \frac{r}{m}\right)\right)(D + \beta v(r, m+1)); 0\right\}. \quad (6.37)$$

The two actions in this model are to allow credit or not, and once credit is refused, it will never be allowed to the customer again. The model can be solved using value iteration as outlined in section 6.5.

Bierman and Hausman sought to extend the model by allowing the decision to be the amount of credit that can be extended, rather than the all-or-nothing decision of (6.37). They assumed that the amount of credit offered is some fraction Y of a standard amount. They allowed for the repayment probability to depend on the credit allowed by saying Prob{repayment$|Y = y\} = p_y$, where p was the probability of repaying the standard amount. This extension shows the limitations of this Bayesian approach. What one wants is that if the initial belief p on the repayment probability of the standard amount, $f_0(p)$, is a beta distribution, then the beliefs after the first repayment period are still in the beta family. These are the equivalent calculations to (6.34) and (6.35). Let $X_1 = y$ if there is full repayment of the credit, which amounts to y of the standard amount, and $X_1 = (0, y)$ if there is no repayment of a fraction y. Calculations akin to (6.34) show that if $f_0(p)$ is $B(r, m)$ and $X_1 = y$,

$$f_1(p) = \frac{P(X_1 = y|p) f_0(p)}{P(X_1 = y)} = \frac{(m+y)!}{(r+y-1)!(m-r-1)!} p^{r+y-1}(1-p)^{m-r-1}. \quad (6.38)$$

This is still a beta distribution. However, if there is no repayment $(X_1 = (0, y))$,

$$f_1(p) = \frac{P(X_1 = (0,y)|p) f_0(p)}{P(X_1 = (0,y))} \propto (1 - p^y) p^{r-1}(1-p)^{m-r-1}. \quad (6.39)$$

This is not a beta distribution. Bierman and Hausman overcome this by assuming that after every nonpayment, the expected discounted payoff will be zero; i.e., no more profit will be made from that customer. This leads to the optimality equation for $v(r, m)$ of

$$v(r,m) = \max_y \left\{\left(\frac{r}{m}\right)^y \left(Py + \beta v(r+y, m+y) + \left(1 - \left(\frac{r}{m}\right)^y\right)\right)(-Dy); 0\right\}, \quad (6.40)$$

where P and D are the profit and loss from a standard amount of credit repaid or defaulted on. The first terms correspond to lending y of the standard amount and the last term to lending nothing.

If one feels that this assumption of no further profit after one nonpayment is too severe, then a different set of problems arises. As the posterior beliefs of the repayment parameter p are no longer beta distributions, the state of the customer can no longer be described by (r, m). Instead, the state of the customer must involve the whole history of payments and nonpayments, i.e., $((y_1), (0, y_2), (y_3), \ldots, (y_n))$ would be a description of someone who paid y_1 in period 1, missed paying in period 2, paid y_3 in period 3, etc. Thus the size of the state space blows up exponentially and makes calculations nearly impossible. Dirickx and Wakeman (1976) investigated this approach and showed that the posterior distributions can be expressed as a linear combination of beta distributions and that if the optimal decision in any period was to advance no credit $(y = 0)$, then this would remain optimal at all subsequent periods. However, it did not seem possible to use the model for practical calculations.

Example 6.6 (credit card repayment model). Modeling credit card or mail-order repayments is more complicated since the required repayment level R is usually some percentage

of the balance outstanding and so can vary from month to month as new items are purchased. Thomas (1992) looked at a model in which there are two parameters, described by Bayesian beliefs. The first is again the probability p that the customer pays something next period. The second is that there is a maximum affordable repayment amount M that the customer can repay, so if $M < R$, the customer will pay M, and if $M \geq R$, the customer will pay between R and M. As before, assume that the belief about p is given by a beta distribution with parameters (r, m). To deal with the belief about M, split the repayment amount into k possible levels, $1, 2, 3, \ldots, k$, etc. (Think of it as £1, £2, £3, ... for convenience.) Instead of estimating M, one defines Bernoulli random variables M_1, M_2, M_3, where

$$p_i = P\{M_i = 1\} = P\{M \geq i | M \geq i-1\} \quad \text{for } i = 1, 2, \ldots, K. \tag{6.41}$$

Thus $M_i = 1$ means that if the affordable repayment amount is at least £$i-1$, it will be at least £i. Given these marginal conditional distributions M_i, one can reconstruct the distribution of M, at least as a discrete variable with values £1, £2, etc. Since the M_i are all Bernoulli random variables and independent of each other, one can describe the belief about each probability p_i by a beta distribution with parameters (r_i, m_i). Concentrating on the repayment aspect of the model, the state of a consumer is given by $(b, n, r, m, r_i, m_i, r_2, m_2, \ldots, r_k, m_k)$, where b is the balance outstanding, n is the number of periods since a repayment, r and m are the parameters of the beta distribution describing the repayment parameter p, and r_i and m_i are the parameters of the beta distribution describing the marginal probabilities of the affordable amount, p_i. The transitions are as follows:

$$\begin{aligned}
&\text{if no payment,} && (b, n, r, m, r_1, m_1, \ldots, r_K, m_K) \\
&&& \quad \to (b, n+1, r, m+1, r_1, m_1, \ldots, r_K, m_K); \\
&\text{if payment } a < R, && (b, n, r, m, r_1, m_1, \ldots, r_K, m_K) \\
&&& \quad \to (b-a, 0, r+1, m+1, r_1+1, m_1+1, \ldots, r_{a+1}, m_a+1, \\
&&& \qquad r_{a+1}, m_{a+1}+1, \ldots, r_K, m_K); \\
&\text{if payment } a \geq R, && (b, n, r, m, r_1, m_1, \ldots, r_K, m_K) \\
&&& \quad \to (b-a, 0, r+1, m+1, r_1+1, m_1+1, \ldots, \\
&&& \qquad r_{a+1}, m_a+1, r_{a+1}, m_{a+1}, \ldots, r_K, m_K).
\end{aligned} \tag{6.42}$$

The idea here is that if there is no repayment, then the mean of the belief goes down from $\frac{r}{m}$ to $\frac{r}{m+1}$. If payment is made for an amount a greater than R, then the belief the customer can pay at all levels up to a goes up from $\frac{r_i}{m_i}$ to $\frac{r_{i+1}}{m_{i+1}}$. The beliefs about paying above a do not change. If, however, the payment is $a < R$, then the belief the customer can pay at all levels up to a goes up again, but the belief the consumer can pay at level $a+1$ goes down from $\frac{r_{a+1}}{m_{a+1}}$ to $\frac{r_{a+1}}{m_{a+1}+1}$. The probability that a repayment will be made next period is $\frac{r}{m}$, and the expected affordable repayment amount is

$$E(M) = \frac{r_1}{m_1}\left(1 + \frac{r_2}{m_2}\left(1 + \frac{r_3}{m_3}\left(1 + \cdots\left(1 + \frac{r_K}{m_K}\right)\right)\right)\right). \tag{6.43}$$

Although the state space is much larger than in the simple model of Example 6.4 and (6.37), it does not suffer from the exponential expansion that occurred in the extension of Example 6.5. One can add on the purchase side by saying there is a probability $q(e)$ that the customer will take e of new credit next period as in (6.15) to develop this into a Markov chain or Markov decision process model.

Chapter 7
Measuring Scorecard Performance

7.1 Introduction

Having built a credit or behavioral scorecard, the obvious question is, "How good is it?" This begs the question of what we mean by good. The obvious answer is in distinguishing the goods from the bads because we want to treat these groups in different ways in credit-scoring systems—for example, accepting the former for credit and rejecting the latter. Behavioral scoring systems are used in a more subtle way, but even if we stick to the idea that we will measure the scoring system by how well it classifies, there are still problems in measuring its performance. This is because there are different ways to define the misclassification rate, mainly due to the sample that we use to check this rate. If we test how good the system is on the sample of customers we used to build the system, the results will be much better than if we did the test on another sample. This must follow because built into the classification system are some of the nuances of that data set that do not appear in other data sets. Thus section 7.2 looks at how to test the classification rate using a sample, called the holdout sample, separate from the one used to build the scoring system. This is a very common thing to do in the credit-scoring industry because of the availability of very large samples of past customers, but it is wasteful of data in that one does not use all the information available to help build the best scoring system. There are times, however, when the amount of data is limited, for example, when one is building a system for a completely new group of customers or products. In that case, one can test the validity of the system using methods that build and test on essentially the same data set but without causing the misclassification errors to be optimistically biased. Section 7.3 looks at the cross-validation methods of leave-one-out and bootstrapping, which do this, while section 7.4 shows how jackknifing can also be used for this purpose.

There are a number of standard ways to describe how different two populations are in their characteristics. These can be used in the case of scorecard-based discrimination methods to see how different the scores are for the two groups of goods and bads. These measure how well the score separates the two groups, and in section 7.5, we look at two such measures of separation—the Mahalanobis distance and Kolmogorov–Smirnov statistics. These give a measure of what is the best separation of the groups that the scorecard can make. One can extend the Kolmogorov–Smirnov approach to get a feel for how good the separation of the two groups is over the whole range of the scores. One way of doing this will generate a curve—the receiver operating characteristic (ROC) curve—which is a useful way

of comparing the classification properties of different scorecards. This curve and the related Gini coefficient are considered in section 7.6. Finally, in section 7.7, we look at some ways that the performance of the scorecard can be checked and, if necessary, corrected without having to completely rebuild the whole system.

7.2 Error rates using holdout samples and 2 x 2 tables

In section 4.2, we defined the decision making in a credit-scoring system as follows: $\mathbf{X} = (X_1, X_2, \ldots, X_p)$ are the application variables, and each applicant gives a set of answers $\mathbf{x}$ to these variables and thereafter is found to be either good (G) or bad (B). Assuming that the application characteristics are continuous (a similar analysis works for the discrete case), let $f(\mathbf{x})$ be the distribution of application characteristics, $f(G|\mathbf{x})$ be the probability of being a good if the application characteristics were $\mathbf{x}$, and $f(B|\mathbf{x}) = 1 - f(G|\mathbf{x})$ be the probability of being a bad if those are the characteristics.

The optimal or Bayes error rate is the minimum possible error rate if one knew these distributions completely over the whole population of possible applicants,

$$e(\text{Opt}) = \int \min\{f(B|\mathbf{x}), f(G|\mathbf{x})\} f(\mathbf{x}) d\mathbf{x}. \tag{7.1}$$

Any given credit-scoring system built on a sample S of n consumers estimates the functions $f(G|\mathbf{x})$ and $f(B|\mathbf{x})$ and in light of this defines two regions of answers A_G and A_B, in which the applicants are classified as good or bad. The actual or true error for such a system is then defined as

$$e_S(\text{Actual}) = \int_{A_G} f(B|\mathbf{x}) f(\mathbf{x}) d\mathbf{x} + \int_{A_B} f(G|\mathbf{x}) f(\mathbf{x}) d\mathbf{x}. \tag{7.2}$$

This is the error that the classifier built on the sample S would incur when applied to an infinite test set. The difference occurs because one has used only a finite sample S to build the estimator. One is usually interested in estimating $e_S(\text{Actual})$, but if one had to decide what system to use before seeing the data and adapting the system to the data, then one would take the expected error rate $e_n(\text{Expected})$, which is the expectation of $e_S(\text{Actual})$ over all samples of size n.

The difficulty in calculating $e_S(\text{Actual})$ is that one does not have an infinite test set on which to calculate it and so one has to use a sample S^* to estimate it on. Hence one calculates

$$e_S(S^*) = \int_{A_G} f(B|\mathbf{x}) f_{S^*}(\mathbf{x}) d\mathbf{x} + \int_{A_B} f(G|\mathbf{x}) f_{S^*}(\mathbf{x}) d\mathbf{x}, \tag{7.3}$$

where $f_{S^*}(\mathbf{x})$ is the distribution of characteristics $\mathbf{x}$ in the sample S^*.

The obvious thing is to check the error on the sample on which the classifier was built and so calculate $e_S(S)$. Not surprisingly, this underestimates the error considerably, so $e_S(S) < e_S(\text{Actual})$. This is because the classifier has incorporated in it all the quirks of the sample S even if these are not representative of the rest of the population. Hence it is far better to test the data on a completely independent sample S^*. This is called the holdout sample, and it is the case that the expected value of $e_S(S^*)$ over all samples excluding S is $e_S(\text{Actual})$. This means that this procedure gives an unbiased estimate of the actual error.

In credit scoring, the cost of the two errors that make up the error rate are very different. Classifying a good as a bad means a loss of profit L, while classifying a bad as a good means

an expected default of D, which is often considerably higher than L. Thus instead of error rates one might look at expected loss rates, where the optimal expected loss $l(\text{Opt})$ satisfies

$$l(\text{Opt}) = \int \min\{Df(B|\mathbf{x}), Lf(G|\mathbf{x})\} f(\mathbf{x}) d\mathbf{x}. \tag{7.4}$$

In analogy with (7.2) and (7.3), the actual or true loss rate for a classifier based on a sample S is

$$l_S(\text{Actual}) = \int_{A_G} Df(B|\mathbf{x}) f(\mathbf{x}) d\mathbf{x} \int_{A_B} Lf(G|\mathbf{x}) f(\mathbf{x}) d\mathbf{x}, \tag{7.5}$$

while the estimated rate using a test sample S^* is

$$l_S(S^*) = \int_{A_G} Df(B|\mathbf{x}) f_{S^*}(\mathbf{x}) d\mathbf{x} \int_{A_B} Lf(G|\mathbf{x}) f_{S^*}(\mathbf{x}) d\mathbf{x}. \tag{7.6}$$

Bayes's theorem will confirm that the expression for the actual loss rate in (7.5) is the same as that given in (4.12).

So how does one calculate $e_S(S^*)$ and $l_S(S^*)$, having built a classifier on a sample S and having a completely independent holdout sample S^* available? What we do is compare the actual class G or B of each customer in the sample S^* with the class that the scorecard predicts. The results are presented in a 2×2 table called the confusion matrix (see Table 7.1), which gives the numbers in each group.

Table 7.1. *General confusion matrix.*

		True class		
		G	B	
Predicted class	G	g_G	g_B	g
	B	b_G	b_B	b
		n_G	n_B	n

For example, we might have the confusion matrix given in Table 7.2. In this sample, n, n_G, and n_B (1000, 750, 250) are fixed as they describe the sample chosen. Thus really only two of the four entries g_G, b_G, g_B, and b_B are independent. The actual error rate is calculated as $\frac{b_G+g_B}{n} = \frac{250}{1000} = 0.25$. If the losses are $L = 100$, $D = 500$, then their actual loss per customer is $\frac{Lb_G+Dg_B}{n} = \frac{(100\cdot150)+(500\cdot100)}{1000} = 65$.

Table 7.2. *Example of a confusion matrix.*

		True class		
		G	B	
Predicted class	G	600	100	700
	B	150	150	300
		750	250	1000

One can use confusion matrices to compare systems or even to try to decide on the best cutoff score when a scorecard had been developed. In the latter case, changing the scorecard in the example in Table 7.2 may lead to the confusion matrix in Table 7.3.

Table 7.3. *Confusion matrix with a different cutoff.*

		True class		
		G	*B*	
Predicted class	*G*	670	130	800
	B	80	120	200
		750	250	1000

The actual error rate is $\frac{130+80}{1000} = 0.21$, which suggests this is a better cutoff. However, the actual expected loss rate is $\frac{(100 \cdot 80)+(500 \cdot 130)}{1000} = 73$, which is higher than the expected loss with the other cutoff. So perhaps the other system was superior. This difference between the two ways of trying to decide which system to go for is very common. Sometimes in credit scoring, one will find that the error rate is minimized by classifying everyone as good and so accepting them all. It is a brave—and foolish—credit analyst who suggests this and thus makes himself redundant!

One approach that is useful in comparing the difference between the way two credit-scoring systems perform on a holdout sample is to look at the swap sets. This is the group of people in the holdout sample, who are classified differently by the two systems. The swap sets for Tables 7.2 and 7.3 might look like Table 7.4.

Table 7.4. *Example of a swap set analysis.*

	True	
	G	*B*
Predicted *G* by Table 7.2, *B* by Table 7.3	50	10
Predicted *B* by Table 7.2, *G* by Table 7.3	120	40

This table cannot be calculated from the two confusion matrices, although from these we can see that 100 more in the sample were predicted *B* by Table 7.2 and *G* by Table 7.3 than were predicted *G* by Table 7.2 and *B* by Table 7.3. There could be in this case a number of people who move a different way to the overall trend. The fact that $\frac{50+10+120+40}{1000} = 0.22$ of the population change between the two scorecards suggests that the scorecards are quite different. If one looks only at the swap sets caused by changing the cutoff, then obviously no customers will have their own classifications changed against the movement. Hence one of the rows in the swap sets will be 0.

7.3 Cross-validation for small samples

The standard way to assess credit-scoring systems is to use a holdout sample since large samples of past customers usually are available. One wants the holdout sample to be independent of the development sample but to be similar to the population on which it is to be used. This causes a problem because of population drift, wherein the characteristics of consumers change over time. The new customers on whom the scorecard is to be used may be different from those of two or three years ago, on whom it was built and tested.

However, there are also situations in which not much data are available to build a scorecard. This could be because what is being assessed is a new type of loan product, or the

lender is targeting a new consumer group. It could also be that it is an area like scoring of commercial customers, where the population itself is very limited. In these cases, we want to use all the data to build the system and so cannot afford to cut the size of the development sample, yet we want as true an estimate of the error rate of the classification as is possible.

The way around this is cross-validation. Subsets of the sample are used to test the performance of the scorecard built in the remainder of the sample. This process is repeated for different subsets and the results are averaged. Two ways have been suggested for choosing the subsets: leave-one-out—a single consumer is used as a test set for the scorecard built on the other $n-1$, and this is repeated for each consumer in the sample; and rotation—the sample is split into m different subsets, say, $m = 5$ or 10, and each one in turn is used as a test set for the scorecard built on the rest.

The leave-one-out method is so much the archetypal cross-validation method that it is often called cross-validation. If $\{S-i\}$ is the sample with customer i missing, one calculates

$$e_S(\text{Leave}) = \sum_{i=1}^{m} e_{S-i}\{i\}. \tag{7.7}$$

It is where the sample is used to its fullest in building the scorecard, but it has the disadvantage that n scorecards have to be built, which could be very time-consuming. Modern computing power makes this less of a burden than might have been expected, and in the linear regression and linear programming approaches to scorecard building, one can use the fact that only one sample data point is different between any two samples to cut down considerably the calculation of the new scorecard. The scorecard built on the full sample is used in practice, and the average error rate of the n scorecards built on the $n-1$ points in turn is an unbiased estimator of the actual error rate, although in practice it has quite a high variance.

The rotational approach considerably cuts down the number of scorecards that need to be built, from n to m, but the scorecards built are less similar to those built on the full sample than in the leave-one-out case. Hence if one uses the full sample scorecard in practice, the average of the error rates of the m scorecards built is not as robust an estimator of the actual error rate as the leave-one-out estimator.

7.4 Bootstrapping and jackknifing

Cross-validation approaches still split the sample into two subsets—one for building the scorecard and the other for testing it. Bootstrapping and jackknifing methods seek to estimate the bias in $e_S(\text{Actual}) - e_S(S)$ and try to remove it.

In bootstrapping, a sample, say, R, equal in size to the original sample is chosen from it, allowing repetition. Thus the same consumer can appear several times in the sample. This sample with possible repetition is taken as the development sample, and the original sample is used to test the scorecard. The procedure is repeated m times using samples chosen again with repetition allowed; call the samples $R_1, R_2, \ldots, R_m$. The idea is that one is trying to estimate the bias between the true and the apparent error rate $e_S(\text{Actual}) - e_S(S)$. For the scorecards built on the samples R, the apparent rate is $e_R(R)$ and the claim is that the true rate is $e_R(S)$. This assumes that S is, in fact, the whole population for these bootstrap estimators. Hence $e_R(S) - e_R(R)$ gives a good estimate of the bias that we are interested in, and so one takes the estimator for $e_S(\text{Actual})$ to be

$$e_S(S) + (e_S(\text{Actual}) - e_S(S)) \approx e_S(S) + \sum_{i=1}^{m} (e_{R_i}(S) - e_{R_i}(R_i)). \tag{7.8}$$

One can improve on this initial bootstrap estimation by recognizing that there are bound to be customers who are both in S and in R, and so these sets are not independent. If there are n customers in sample S, the chance a particular one does not appear in a specific R is $(1 - \frac{1}{n})^n$ (i.e., $(1 - \frac{1}{n})$ is the chance the customer is not sampled each time one of the n customers is sampled). Letting $n \to \infty$, this converges to $e^{-1} = 0.368$. The easiest way to see this is to take logs and to recognize that $n \log(1 - \frac{1}{n})$ tends to -1. Thus there is a 0.368 chance a consumer does not appear in sample R and a 0.632 chance that the consumer does appear in sample R. It then turns out that an improved estimate of the actual error rate is obtained by taking

$$0.368 e_S(S) + 0.632 \sum_m e_{R_m}(S - R_m), \tag{7.9}$$

where $(S - R_m)$ is the set of customers who are in S but not in R_m.

The jackknife method operates very similarly to the leave-one-out estimator in that one leaves out each of the sample customers in turn and builds a scorecard on the remaining $n - 1$. However, it then calculates the apparent error rate of this scorecard $e_{[S-i]}(S - i)$, and hence the average apparent rate is

$$\bar{e}_{S-1}(S - 1) = \sum_{i=1}^{n} e_{S-i}(S - i). \tag{7.10}$$

Similarly, one calculates $e_{[S-i]}(S)$ and the average $\bar{e}_{S-1}(S) = \sum_{i=1}^{n} e_{S-i}(S)$, and one can show that $(n - 1)(\bar{e}_{S-1}(S) - \bar{e}_{S-1}(S - 1))$ is a good estimate of the bias between estimating on the true set (in this case, S) and the set on which the scorecard was built (the $S - i$ in this case). Thus the jackknife estimate $e_S(\text{Jack})$ of $e_S(\text{Actual})$ is taken as

$$e_S(\text{Jack}) = e_S(S) + (n - 1)(\bar{e}_{S-1}(S) - \bar{e}_{S-1}(S - 1)). \tag{7.11}$$

It can be shown (Hand 1997) that there is a relationship between the jackknife and the leave-one-out estimate of $e_S(\text{Actual})$, namely,

$$e_S(\text{Jack}) = e_S(\text{Leave}) + (e_S(S) - \bar{e}_{S-1}(S)). \tag{7.12}$$

7.5 Separation measures: Mahalanobis distance and Kolmogorov–Smirnov statistics

A number of measures used throughout statistics describe how far apart the characteristics of two populations are. If one has a scoring system that gives a score to each member of the population, then one can use these measures to describe how different the scores of the goods and the bads are. Thus these approaches can be used only for credit-scoring systems that actually give a score, like the regression approaches or linear programming. They cannot be used for the credit-scoring systems that group, like classification trees, or where a score is not explicit, like neural networks. Moreover, they describe the general properties of the scorecard and do not depend on which cutoff score is used. This is useful in that these measures give a feel for the robustness of the scorecard if the cutoff score is changed and may be useful in determining what the cutoff score should be. However, when it comes to it, people will want to know how well the scorecard will predict, and to know that one needs to have chosen a specific cutoff score so that one can estimate the error rates and confusion matrices of sections 7.2 and 7.3.

For accurate estimates, we should calculate these measures on a holdout sample, which is independent of the development sample on which the scorecard was built. However, often speed and the fact that in many statistical packages it is much easier to calculate the measures on the development sample than on the holdout sample mean that one calculates them first on the development sample. Since they are only indicators of the relative effectiveness of different scorecards, the assumption is made that if one is much better than the other on the development sample, it will remain so on the holdout sample.

The first measure, the Mahalanobis distance, appeared earlier in the discussion on the Fisher approach to using linear regression as a classification method in section 4.3. There we found the linear combination Y of the application variables so that M, the difference between the sample mean of Y for the goods and the sample mean Y of the bads, divided by the standard deviation in Y for each group was as large as possible. This M is the Mahalanobis distance and is a measure of by how much the scores of the two groups of the goods and the bads differ. Formally, if $n_G(s)$ and $n_B(s)$ are the numbers of goods and bads with score s in a sample of n, where there are n_G goods and n_B bads, the $p_G(s) = \frac{n_G(s)}{n_G}$ $(p_B(s) = \frac{n_B(s)}{n_B})$ are the probabilities of a good (and bad) having a score s. Then $m_G = \sum s p_G(s)$ and $m_B = \sum s p_B(s)$ are the mean scores of the goods and bads. Let σ_G and σ_B be the standard deviation of the scores of the goods and the bads, calculated as

$$\sigma_G^2 = \left(\sum_s s^2 p_G(s) - m_G^2\right)^{\frac{1}{2}}, \qquad \sigma_B^2 = \left(\sum_s s^2 p_B(s) - m_B^2\right)^{\frac{1}{2}}. \tag{7.13}$$

Let σ be the pooled standard deviation of the goods and the bads from their respective means; it is calculated as follows:

$$\sigma = \left(\frac{n_G\sigma_G^2 + n_B\sigma_B^2}{n}\right)^{\frac{1}{2}}. \tag{7.14}$$

The Mahalanobis distance M is then the difference between the mean score of the two groups, suitably standardized

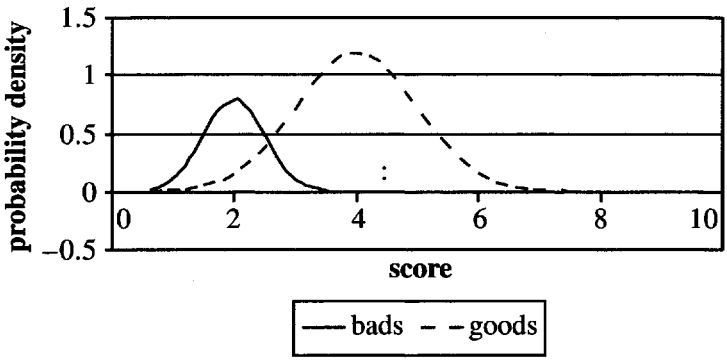
$$M = \frac{m_G - m_B}{\sigma}. \tag{7.15}$$

This is indifferent to any linear scaling of the score, and as Figure 7.1 suggests, one would assume that if a scorecard has a large Mahalanobis distance, it will be a better classifier. In Figure 7.1, the dotted lines represent possible cutoff scores, and the errors in the figure on the left with the smaller M are much greater than those on the right.

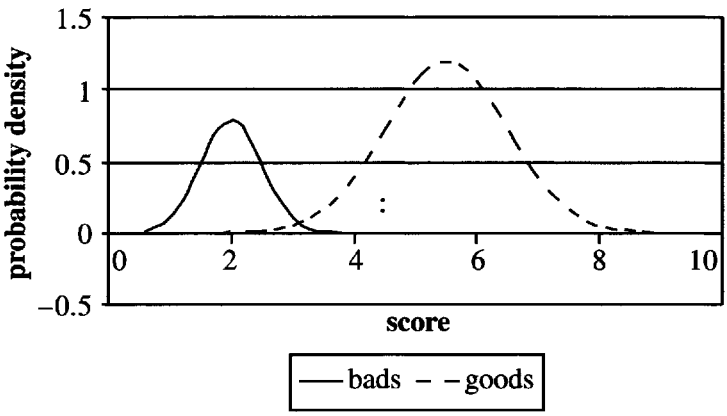
The Mahalanobis distance measures how far apart the means of the goods score and the bads score are. The Kolmogorov–Smirnov statistic measures how far apart the distribution functions of the scores of the goods and the bads are. Formally, if $P_G(s) = \sum_{x \le s} p_G(x)$ and $P_B(s) = \sum_{x \le s} p_B(x)$ (or the sums replaced by integrals if the scores are continuous), then the Kolmogorov–Smirnov (KS) statistic is

$$\text{KS} = \max_s |P_G(s) - P_B(s)|. \tag{7.16}$$

In Figure 7.2, the Kolmogorov–Smirnov statistic is the length of the dotted line at the score that maximizes the separation in the distribution function. If the distribution functions are sufficiently regular, then the Kolmogorov–Smirnov distance occurs at the score where the good and bad histograms in Figure 7.1 cross. As mentioned in section 4.7, the Kolmogorov–Smirnov statistic for an attribute, rather than the score, is used to find the best splits in a classification tree.



(a)



(b)

Figure 7.1. (a) *Goods and bads similar.* (b) *Goods and bads different.*

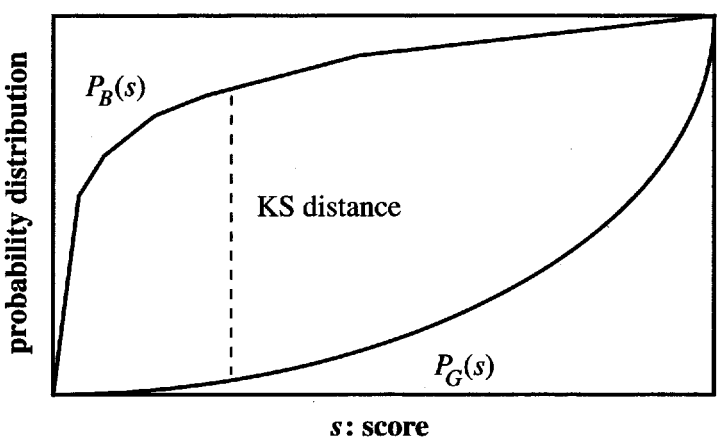


Figure 7.2. *Kolmogorov–Smirnov distance.*

7.6 ROC curves and Gini coefficients

Kolmogorov–Smirnov statistics can be displayed, as in Figure 7.2, by plotting two curves, but it is possible to display the same information on one curve by plotting $P_G(s)$ against $P_B(s)$. The result is a curve as in Figure 7.3, where each point on the curve represents some score s, and its horizontal distance is $P_B(s)$ and its vertical value is $P_G(s)$.

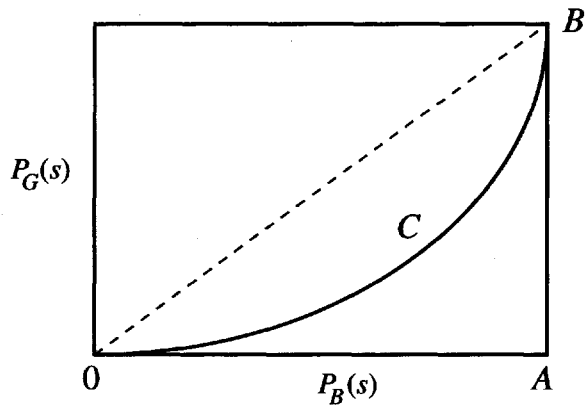


Figure 7.3. *ROC curve.*

This is the ROC curve, sometimes called the Lorentz diagram. It describes the classification property of the scorecard as the cutoff score varies. The best possible scorecard would have an ROC curve that goes all the way along the horizontal axis before going up the vertical axis. Thus point A would correspond to a score s^*, where $P_B(s^*) = 1$ and $P_G(s^*) = 0$; i.e., all of the bads have scores less than s^* and none of the goods do. An ROC curve along the diagonal OB would correspond to one where at every score $P_G(s) = P_B(s)$, so the ratio of goods to bads is the same for all score ranges. This is no better than classifying randomly given that one knows the ratio of good to bads in the whole population. Thus the further from the diagonal the ROC curve is, the better the scorecard. If one scorecard has a ROC curve that is always further from the diagonal than the ROC curve of another scorecard, then the first scorecard dominates the second and is a better classifier at all cutoff scores. More common is to find ROC curves that cross so one scorecard is a better classifier in one score region and the other is better in the other region.

The ROC curve's name suggests its origin in estimating classification errors in transmitting and receiving messages. Using classification as a tool in epidemiology has introduced two other terms that are found in scorecard measurement. If the bads are thought of as the cases one needs to detect, then the sensitivity (Se) is the proportion of bads who are predicted as bad, while the specificity (Sp) is the proportion of goods who are predicted as good. If prediction is by a scorecard, with those below the cutoff score predicted to be bad and those above the cutoff score predicted to be good, then the sensitivity Se is $P_B(s)$, while the specificity Sp is $1 - P_G(s)$. Thus the ROC curve can be thought of as plotting 1− specificity against sensitivity.

One can also use the confusion matrix to calculate sensitivity and specificity. Using the notation of Table 7.1, $\text{Se} = \frac{b_B}{n_B}$, while $\text{Sp} = \frac{g_G}{n_G}$. Recall that since n_G and n_B are fixed, there are only two variables in the confusion matrix, so specifying Se and Sp defines it exactly.

Recall that the further the ROC curve is from the diagonal the better. This suggests that the larger the shaded area in Figure 7.3 between the ROC curve and the diagonal, the