CHAPTER 9

Probabilistic models

THE THIRD AND FINAL FAMILY of machine learning models considered in this book are probabilistic models. We have already seen how probabilities can be useful to express a model's expectation about the class of a given instance. For example, a ☞*probability estimation tree* (Section 5.2) attaches a class probability distribution to each leaf of the tree, and each instance that gets filtered down to a particular leaf in a tree model is labelled with that particular class distribution. Similarly, a calibrated linear model translates the distance from the decision boundary into a class probability (Section 7.4). These are examples of what are called *discriminative* probabilistic models. They model the posterior probability distribution $P(Y|X)$, where Y is the target variable and X are the features. That is, given X they return a probability distribution over Y.

The other main class of probabilistic models are called *generative* models. They model the joint distribution $P(Y,X)$ of the target Y and the feature vector X. Once we have access to this joint distribution we can derive any conditional or marginal distribution involving the same variables. In particular, since $P(X) = \sum_y P(Y=y,X)$ it follows that the posterior distribution can be obtained as

$$P(Y|X) = \frac{P(Y,X)}{\sum_y P(Y=y,X)}$$

Alternatively, generative models can be described by the likelihood function $P(X|Y)$, since $P(Y,X) = P(X|Y)P(Y)$ and the target or prior distribution (usually abbreviated to 'prior') can be easily estimated or postulated. Such models are called 'generative' because we can sample from the joint distribution to obtain new data points together with their labels. Alternatively, we can use $P(Y)$ to sample a class and $P(X|Y)$ to sample an instance for that class – this was illustrated for the spam e-mail example on p.29. In contrast, a discriminative model such as a probability estimation tree or a linear classifier models $P(Y|X)$ but not $P(X)$, and hence can be used to label data but not generate it.

Since generative models can do anything that discriminative models do, they may seem preferable. However, they have a number of drawbacks as well. First of all, note that storing the joint distribution requires space exponential in the number of features. This necessitates simplifying assumptions such as independence between features, which may lead to inaccuracies if they are not valid in a particular domain. The most common criticism levied against generative models is that accuracy in modelling $P(X)$ may actually be achieved at the expense of less accurate modelling of $P(Y|X)$. However, the issue is not yet fully understood, and there are certainly situations where knowledge of $P(X)$ provides welcome additional understanding of the domain. For example, we may be less concerned about misclassifying certain instances if they are unlikely according to $P(X)$.

One of the most attractive features of the probabilistic perspective is that it allows us to view learning as a process of reducing uncertainty. For instance, a uniform class prior tells us that, before knowing anything about the instance to be classified, we are maximally uncertain about which class to assign. If the posterior distribution after observing the instance is less uniform, we have reduced our uncertainty in favour of one class or the other. We can repeat this process every time we receive new information, using the posterior obtained in the previous step as the prior for the next step. This process can be applied, in principle, to any unknown quantity that we come across.

Example 9.1 (Spam or not?). Suppose we want to estimate the probability θ that an arbitrary e-mail is spam, so that we can use the appropriate prior distribution. The natural thing to do is to inspect n e-mails, determine the number of spam e-mails d, and set $\hat{\theta} = d/n$; we don't really need any complicated statistics to tell us that. However, while this is the most likely estimate of θ – the maximum a posteriori (MAP) estimate, using the terminology introduced on p.28 – this doesn't mean that other values of θ are completely ruled out. We model this by a probability distribution over θ which is updated each time new information comes in. This is further illustrated in Figure 9.1 for a distribution that is more and more skewed towards spam.

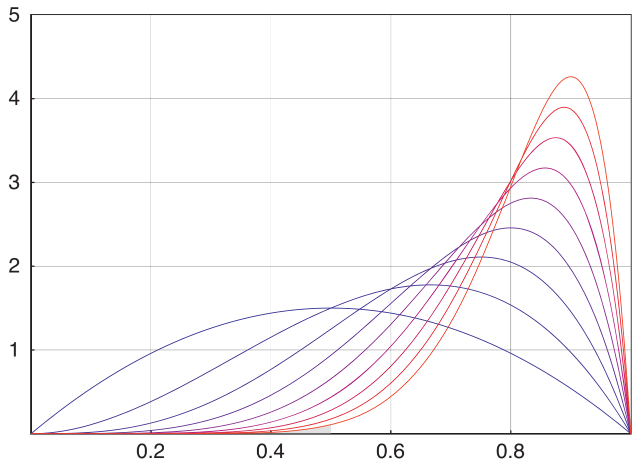


Figure 9.1. Each time we inspect an e-mail, we are reducing our uncertainty regarding the prior spam probability θ. After we inspect two e-mails and observe one spam, the possible θ values are characterised by a symmetric distribution around 1/2. If we inspect a third, fourth, ..., tenth e-mail and each time (except the first one) it is spam, then this distribution narrows and shifts a little bit to the right each time. As you would expect, the distribution for n e-mails reaches its maximum at $\hat{\theta}_{\text{MAP}} = \frac{n-1}{n}$ (e.g., $\hat{\theta}_{\text{MAP}} = 0.8$ for $n = 5$); however, asymmetric distributions like these contain information that cannot be conveyed by single numbers such as the mean or the maximum.

Explicitly modelling the posterior distribution over the parameter θ has a number of advantages that are usually associated with the 'Bayesian' perspective:

- ☞ We can precisely characterise the uncertainty that remains about our estimate by quantifying the spread of the posterior distribution.
- ☞ We can obtain a generative model for the parameter by sampling from the posterior distribution, which contains much more information than a summary statistic such as the MAP estimate can convey – so, rather than using a single e-mail with $\theta = \theta_{\text{MAP}}$, our generative model can contain a number of e-mails with θ sampled from the posterior distribution.
- ☞ We can quantify the probability of statements such as 'e-mails are biased towards ham' (the tiny shaded area in Figure 9.1 demonstrates that after observing one ham and nine spam e-mails this probability is very small, about 0.6%).
- ☞ We can use one of these distributions to encode our prior beliefs: e.g., if we believe that the proportions of spam and ham are typically 50–50, we can take the distribution for $n = 2$ (the lowest, symmetric one in Figure 9.1) as our prior.[1]

[1]Statisticians call a prior that has the same mathematical form as a posterior distribution a *conjugate*

The key point is that *probabilities do not have to be interpreted as estimates of relative frequencies, but can carry the more general meaning of (possibly subjective) degrees of belief*. Consequently, we can attach a probability distribution to almost anything: not just features and targets, but also model parameters and even models. For instance, in the example just given we were considering the distribution $P(\theta|D)$, where D represents the data (i.e., the classes of the inspected e-mails).

An important concept related to probabilistic models is *Bayes-optimality*. A classifier is Bayes-optimal if it always assigns $\arg\max_y P^*(Y = y|X = x)$ to an instance x, where P^* denotes the true posterior distribution. Even if we almost never know the true distribution in a practical situation, there are several ways in which we can make this concrete. For example, we can perform experiments with artificially generated data for which we have chosen the true distribution ourselves: this allows us to experimentally evaluate how close the performance of a model is to being Bayes-optimal. Alternatively, the derivation of a probabilistic learning method usually makes certain assumptions about the true distribution, which allows us to prove theoretically that the model will be Bayes-optimal provided these assumptions are met. For example, later on in this chapter we will state the conditions under which the basic linear classifier is Bayes-optimal. The property is therefore best understood as a yardstick by which we measure the performance of probabilistic models.

Since many models discussed in previous chapters are able to estimate class probabilities and hence are discriminative probabilistic models, it is worth pointing out that the choice of a single model, often referred to as *model selection*, does not necessarily lead to Bayes-optimality – even if the model chosen is the one that performs best under the true distribution. To illustrate this, let m^* be the best probability estimation tree we have learned from a sufficient amount of data. Using m^* we would predict $\arg\max_y P(Y = y|M = m^*, X = x)$ for an instance x, where M is a random variable ranging over the model class m^* was chosen from. However, these predictions are not necessarily Bayes-optimal since

$$\begin{aligned}
P(Y|X=x) &= \sum_{m\in M} P(Y, M=m|X=x) && \text{by marginalising over } M \\
&= \sum_{m\in M} P(Y|M=m, X=x)P(M=m|X=x) && \text{by the chain rule} \\
&= \sum_{m\in M} P(Y|M=m, X=x)P(M=m) && \text{by independence of } M \text{ and } X
\end{aligned}$$

Here, $P(M)$ can be interpreted as a posterior distribution over models after seeing the training data (the MAP model is therefore $m^* = \arg\max_m P(M = m)$). The final

prior – in this case we have used the Beta distribution, which is conjugate to the binomial distribution. Conjugate priors not only simplify the mathematics, but also allow more intuitive interpretations: in this case we pretend we have already inspected two e-mails, one of which was spam – a very useful idea that we have in fact already used in the form of the ☞*Laplace correction* in Section 2.3.

expression in the preceding derivation tells us to average the predictions of all models, weighted by their posterior probabilities. Clearly, this distribution is only equal to $P(Y|M = m^*, X = x)$ if $P(M)$ is zero for all models other than m^*, i.e., if we have seen sufficient training data to rule out all but one remaining model. This is obviously unrealistic.[2]

The outline of the chapter is as follows. In Section 9.1 we will see some useful connections between the geometric perspective and the probabilistic viewpoint, which come about when features are normally distributed. This allows us, as already mentioned, to state the conditions under which the basic linear classifier is Bayes-optimal. In Section 9.2 we consider the case of categorical features, leading to the well-known naive Bayes classifier. Section 9.3 revisits the linear classifier from a probabilistic perspective, which results in a new training algorithm explicitly aimed at optimising the posterior probability of the examples. Section 9.4 discusses ways to deal with hidden variables. Finally, in Section 9.5 we briefly look at compression-based learning methods, which can be given a probabilistic interpretation by means of information-theoretic notions.

9.1 The normal distribution and its geometric interpretations

We can draw a connection between probabilistic and geometric models by considering probability distributions defined over Euclidean spaces. The most common such distributions are *normal distributions*, also called *Gaussians*; Background 9.1 recalls the most important facts concerning univariate and multivariate normal distributions. We start by considering the univariate, two-class case. Suppose the values of $x \in \mathbb{R}$ follow a *mixture model*: i.e., each class has its own probability distribution (a *component* of the mixture model). We will assume a Gaussian mixture model, which means that the components of the mixture are both Gaussians. We thus have

$$P(x|\oplus) = \frac{1}{\sqrt{2\pi}\sigma^{\oplus}} \exp\left(-\frac{1}{2}\left[\frac{x-\mu^{\oplus}}{\sigma^{\oplus}}\right]^2\right) \qquad P(x|\ominus) = \frac{1}{\sqrt{2\pi}\sigma^{\ominus}} \exp\left(-\frac{1}{2}\left[\frac{x-\mu^{\ominus}}{\sigma^{\ominus}}\right]^2\right)$$

where $\mu^{\oplus}$ and $\sigma^{\oplus}$ are the mean and standard deviation for the positive class, and $\mu^{\ominus}$ and $\sigma^{\ominus}$ are the mean and standard deviation for the negative class. This gives the following likelihood ratio:

$$\mathrm{LR}(x) = \frac{P(x|\oplus)}{P(x|\ominus)} = \frac{\sigma^{\ominus}}{\sigma^{\oplus}} \exp\left(-\frac{1}{2}\left[\left(\frac{x-\mu^{\oplus}}{\sigma^{\oplus}}\right)^2 - \left(\frac{x-\mu^{\ominus}}{\sigma^{\ominus}}\right)^2\right]\right) \tag{9.1}$$

[2]Note that we do not require the two distributions to be equal, but rather that they reach the same maximum for Y. It is not hard to demonstrate that this, too, is not generally the case.

The univariate normal or Gaussian distribution has the following probability density function:

$$P(x|\mu,\sigma) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) = \frac{1}{E} \exp\left(-\frac{1}{2}\left[\frac{x-\mu}{\sigma}\right]^2\right) = \frac{1}{E} \exp\left(-z^2/2\right), \quad E = \sqrt{2\pi}\sigma$$

The distribution has two parameters: μ, which is the mean or expected value, as well as the median (i.e., the point where the area under the density function is split in half) and the mode (i.e., the point where the density function reaches its maximum); and σ, which is the standard deviation and determines the width of the bell-shaped curve.

$z = (x-\mu)/\sigma$ is the *z-score* associated with x; it measures the number of standard deviations between x and the mean (it has itself mean 0 and standard deviation 1). It follows that $P(x|\mu,\sigma) = \frac{1}{\sigma} P(z|0,1)$, where $P(z|0,1)$ denotes the *standard normal distribution*. In other words, any normal distribution can be obtained from the standard normal distribution by scaling the x-axis with a factor σ, scaling the y-axis with a factor $1/\sigma$ (so the area under the curve remains 1), and translating the origin over μ.

The *multivariate normal distribution* over d-vectors $\mathbf{x} = (x_1,\ldots,x_d)^{\mathrm{T}} \in \mathbb{R}^d$ is

$$P(\mathbf{x}|\boldsymbol{\mu},\boldsymbol{\Sigma}) = \frac{1}{E_d} \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right), \quad E_d = (2\pi)^{d/2}\sqrt{|\boldsymbol{\Sigma}|} \tag{9.2}$$

The parameters are the mean vector $\boldsymbol{\mu} = (\mu_1,\ldots,\mu_d)^{\mathrm{T}}$ and the d-by-d covariance matrix $\boldsymbol{\Sigma}$ (see Background 7.2 on p.200). $\boldsymbol{\Sigma}^{-1}$ is the inverse of the covariance matrix, and $|\boldsymbol{\Sigma}|$ is its determinant. The components of $\mathbf{x}$ may be thought of as d features that are possibly correlated.

If $d = 1$, then $\boldsymbol{\Sigma} = \sigma^2 = |\boldsymbol{\Sigma}|$ and $\boldsymbol{\Sigma}^{-1} = 1/\sigma^2$, which gives us the univariate Gaussian as a special case. For $d = 2$ we have $\boldsymbol{\Sigma} = \begin{pmatrix} \sigma_1^2 & \sigma_{12} \\ \sigma_{12} & \sigma_2^2 \end{pmatrix}$, $|\boldsymbol{\Sigma}| = \sigma_1^2\sigma_2^2 - (\sigma_{12})^2$ and $\boldsymbol{\Sigma}^{-1} = \frac{1}{|\boldsymbol{\Sigma}|}\begin{pmatrix} \sigma_2^2 & -\sigma_{12} \\ -\sigma_{12} & \sigma_1^2 \end{pmatrix}$. Using z-scores we derive the following expression for the bivariate normal distribution:

$$P(x_1,x_2|\mu_1,\mu_2,\sigma_1,\sigma_2,\rho) = \frac{1}{E_2} \exp\left(-\frac{1}{2(1-\rho^2)}(z_1^2+z_2^2-2\rho z_1 z_2)\right), \quad E_2 = 2\pi\sigma_1\sigma_2\sqrt{1-\rho^2} \tag{9.3}$$

where $z_i = (x_i-\mu_i)/\sigma_i$ for $i = 1,2$, and $\rho = \sigma_{12}/\sigma_1\sigma_2$ is the *correlation coefficient* between the two features.

The *multivariate standard normal distribution* has $\boldsymbol{\mu} = \mathbf{0}$ (a d-vector with all 0s) and $\boldsymbol{\Sigma} = \mathbf{I}$ (the d-by-d identity matrix), and thus $P(\mathbf{x}|\mathbf{0},\mathbf{I}) = \frac{1}{(2\pi)^{d/2}} \exp\left(-\frac{1}{2}\mathbf{x}\cdot\mathbf{x}\right)$.

Background 9.1. The normal distribution.

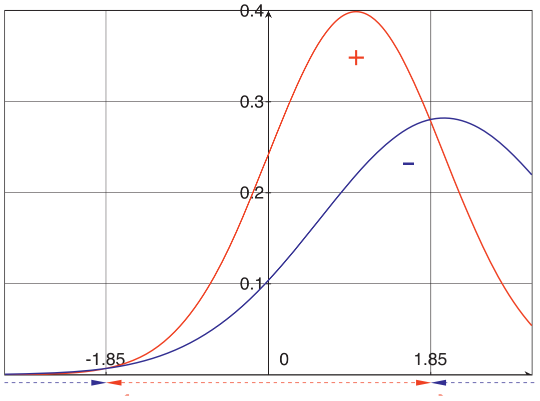


Figure 9.2. If positive examples are drawn from a Gaussian with mean and standard deviation 1 and negatives from a Gaussian with mean and standard deviation 2, then the two distributions cross at $x = \pm 1.85$. This means that the maximum-likelihood region for positives is the closed interval $[-1.85, 1.85]$, and hence the negative region is non-contiguous.

Let's first consider the case that both components have the same standard deviation, i.e., $\sigma^{\oplus} = \sigma^{\ominus} = \sigma$. We can then simplify the exponent in Equation 9.1 as follows:

$$\begin{aligned}
-\frac{1}{2\sigma^2}\left[(x-\mu^{\oplus})^2-(x-\mu^{\ominus})^2\right] &= -\frac{1}{2\sigma^2}\left[x^2-2\mu^{\oplus}x+\mu^{\oplus 2}-(x^2-2\mu^{\ominus}x+\mu^{\ominus 2})\right]\\
&= -\frac{1}{2\sigma^2}\left[-2(\mu^{\oplus}-\mu^{\ominus})x+(\mu^{\oplus 2}-\mu^{\ominus 2})\right]\\
&= \frac{\mu^{\oplus}-\mu^{\ominus}}{\sigma^2}\left[x-\frac{\mu^{\oplus}+\mu^{\ominus}}{2}\right]
\end{aligned}$$

The likelihood ratio can thus be written as $\mathrm{LR}(x) = \exp\left(\gamma(x-\mu)\right)$, with two parameters: $\gamma = (\mu^{\oplus}-\mu^{\ominus})/\sigma^2$ is the difference between the means in proportion to the variance, and $\mu = (\mu^{\oplus}+\mu^{\ominus})/2$ is the midpoint between the two class means. It follows that the maximum-likelihood decision threshold (the value of x such that $\mathrm{LR}(x) = 1$) is $x_{\mathrm{ML}} = \mu$.

If $\sigma^{\oplus} \neq \sigma^{\ominus}$, the x^2 terms in Equation 9.1 do not cancel. This results in two decision boundaries and a non-contiguous decision region for one of the classes.

Example 9.2 (Univariate mixture model with unequal variances). Suppose $\mu^{\oplus} = 1$, $\mu^{\ominus} = 2$ and $\sigma^{\ominus} = 2\sigma^{\oplus} = 2$, then $\mathrm{LR}(x) = 2\exp\left(-[(x-1)^2-(x-2)^2/4]/2\right) = 2\exp\left(3x^2/8\right)$. It follows that the ML decision boundaries are $x = \pm(8/3)\ln 2 = \pm 1.85$. As can be observed in Figure 9.2, these are the points where the two Gaussians cross. In contrast, if $\sigma^{\ominus} = \sigma^{\oplus}$ then we get a single ML decision boundary at $x = 1.5$.

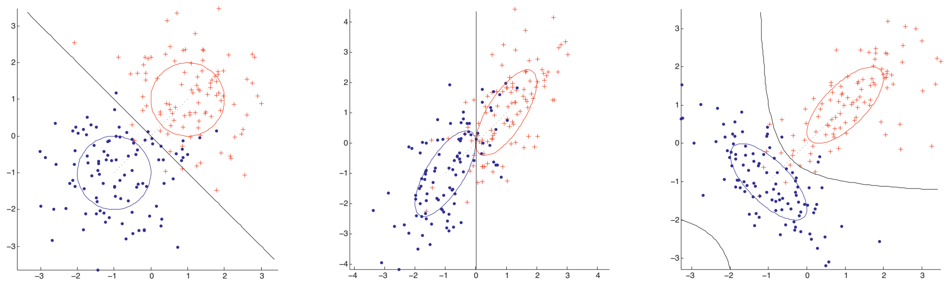


Figure 9.3. **(left)** If the features are uncorrelated and have the same variance, maximum-likelihood classification leads to the basic linear classifier, whose decision boundary is orthogonal to the line connecting the means. **(middle)** As long as the per-class covariance matrices are identical, the Bayes-optimal decision boundary is linear – if we were to decorrelate the features by rotation and scaling, we would again obtain the basic linear classifier. **(right)** Unequal covariance matrices lead to hyperbolic decision boundaries, which means that one of the decision regions is non-contiguous.

Non-contiguous decision regions can also occur in higher-dimensional spaces. The following example demonstrates this for $m = 2$.

Example 9.3 (Bivariate Gaussian mixture). We use Equation 9.3 on p.267 to obtain explicit expressions for the ML decision boundary in the bivariate case. Throughout the example we assume $\mu_1{}^{\oplus} = \mu_2{}^{\oplus} = 1$ and $\mu_1{}^{\ominus} = \mu_2{}^{\ominus} = -1$.

(*i*) If all variances are 1 and both correlations are 0, then the ML decision boundary is given by $(x_1-1)^2+(x_2-1)^2-(x_1+1)^2-(x_2+1)^2 = -2x_1-2x_2-2x_1-2x_2 = 0$, i.e., $x_1+x_2 = 0$ (Figure 9.3 (left)).

(*ii*) If $\sigma_1{}^{\oplus} = \sigma_1{}^{\ominus} = 1$, $\sigma_2{}^{\oplus} = \sigma_2{}^{\ominus} = \sqrt{2}$ and $\rho^{\oplus} = \rho^{\ominus} = \sqrt{2}/2$, then the ML decision boundary is $(x_1-1)^2+(x_2-1)^2/2-\sqrt{2}(x_1-1)(x_2-1)/\sqrt{2}-(x_1+1)^2-(x_2+1)^2/2+\sqrt{2}(x_1+1)(x_2+1)/\sqrt{2} = -2x_1 = 0$ (Figure 9.3 (middle)).

(*iii*) If all variances are 1 and $\rho^{\oplus} = -\rho^{\ominus} = \rho$, then the ML decision boundary is given by $(x_1-1)^2+(x_2-1)^2-2\rho(x_1-1)(x_2-1)-(x_1+1)^2-(x_2+1)^2-2\rho(x_1+1)(x_2+1) = -4x_1-4x_2-4\rho x_1x_2-4\rho = 0$, i.e., $x_1+x_2+\rho x_1x_2+\rho = 0$, which is a hyperbole. Figure 9.3 (right) illustrates this for $\rho = 0.7$. Notice that the bottom left of the instance space is a positive decision region, even though it contains no training examples and it is closer to the negative mean than to the positive mean.

Notice the circles and ellipses in Figure 9.3, which provide a visual summary of the covariance matrix. By projecting the shape for the positive class down to the x-axis we obtain the interval $[\mu_1{}^{\oplus}-\sigma_1{}^{\oplus}, \mu_1{}^{\oplus}+\sigma_1{}^{\oplus}]$ – i.e., one standard deviation around the mean – and similar for the negative class and the y-axis. Three cases can be distinguished: (*i*) both x and y standard deviations are equal and the correlation coefficient is zero, in which case the shape is a circle; (*ii*) the standard deviations are different and the correlation coefficient is zero, which means the shape is an ellipse parallel to the axis with the largest standard deviation; (*iii*) the correlation coefficient is non-zero: the orientation of the ellipse gives the sign of the correlation coefficient, and its width varies with the magnitude of the correlation coefficient.[3] Mathematically, these shapes are defined by setting $f(\mathbf{x})$ in $\frac{1}{E_d}\exp\left(-\frac{1}{2}f(\mathbf{x})\right)$ to 1 and solving for $\mathbf{x}$, in order to capture the points that are one standard deviation away from the mean. For the bivariate case this leads to $(z_1^2+z_2^2-2\rho z_1 z_2) = 1-\rho^2$, which can be translated into an elliptic equation for x_1 and x_2 by expanding the z-scores. Notice that for $\rho = 0$ this is a circle around the origin, and when $\rho \to 1$ this approaches the line $z_2 = z_1$ (we can't put $\rho = 1$ because this leads to a singular covariance matrix).

In the general multivariate case the condition $(\mathbf{x}-\boldsymbol{\mu})^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu}) = 1$ defines a hyper-ellipse, because $\boldsymbol{\Sigma}^{-1}$ satisfies certain properties.[4] For a standard normal distribution, one-standard-deviation contours lie on a hyper-sphere (a circle in d dimensions) defined by $\mathbf{x}\cdot\mathbf{x} = 1$. A very useful geometric intuition is that, just as hyper-spheres can be turned into arbitrary hyper-ellipses by scaling and rotation, any multivariate Gaussian can be obtained from the standard Gaussian by scaling and rotation (to obtain the desired covariance matrix) and translation (to obtain the desired mean). Conversely, we can turn an arbitrary multivariate Gaussian into a standard normal distribution by translation, rotation and scaling, as was already suggested in Background 1.2 on p.24. This results in decorrelated and normalised features.

The general form of the likelihood ratio can be derived from Equation 9.2 on p.267 as

$$\mathrm{LR}(\mathbf{x}) = \sqrt{\frac{|\boldsymbol{\Sigma}^{\ominus}|}{|\boldsymbol{\Sigma}^{\oplus}|}}\exp\left(-\frac{1}{2}\left[(\mathbf{x}-\boldsymbol{\mu}^{\oplus})^{\mathrm{T}}(\boldsymbol{\Sigma}^{\oplus})^{-1}(\mathbf{x}-\boldsymbol{\mu}^{\oplus}) - (\mathbf{x}-\boldsymbol{\mu}^{\ominus})^{\mathrm{T}}(\boldsymbol{\Sigma}^{\ominus})^{-1}(\mathbf{x}-\boldsymbol{\mu}^{\ominus})\right]\right)$$

where $\boldsymbol{\mu}^{\oplus}$ and $\boldsymbol{\mu}^{\ominus}$ are the class means, and $\boldsymbol{\Sigma}^{\oplus}$ and $\boldsymbol{\Sigma}^{\ominus}$ are the covariance matrices for each class. To understand this a bit better, assume that $\boldsymbol{\Sigma}^{\oplus} = \boldsymbol{\Sigma}^{\ominus} = \mathbf{I}$ (i.e., in each class the features are uncorrelated and have unit variance), then we have

$$\begin{aligned}\mathrm{LR}(\mathbf{x}) &= \exp\left(-\frac{1}{2}\left[(\mathbf{x}-\boldsymbol{\mu}^{\oplus})^{\mathrm{T}}(\mathbf{x}-\boldsymbol{\mu}^{\oplus}) - (\mathbf{x}-\boldsymbol{\mu}^{\ominus})^{\mathrm{T}}(\mathbf{x}-\boldsymbol{\mu}^{\ominus})\right]\right)\\ &= \exp\left(-\frac{1}{2}\left[||\mathbf{x}-\boldsymbol{\mu}^{\oplus}||^2 - ||\mathbf{x}-\boldsymbol{\mu}^{\ominus}||^2\right]\right)\end{aligned}$$

[3]A common mistake is to think that the angle of rotation of the ellipse depends on the correlation coefficient; in fact, it is solely determined by the relative magnitudes of the marginal standard deviations.

[4]Specifically, $\mathbf{x}^{\mathrm{T}}\mathbf{A}\mathbf{x}$ defines a hyper-ellipse if $\mathbf{A}$ is symmetric and positive definite. Both properties are satisfied if $\mathbf{A}$ is the inverse of a non-singular covariance matrix.

It follows that LR($\mathbf{x}$) = 1 for any $\mathbf{x}$ equidistant from $\boldsymbol{\mu}^{\oplus}$ and $\boldsymbol{\mu}^{\ominus}$. But this means that the ML decision boundary is a straight line at equal distances from the class means – in which we recognise our old friend, the basic linear classifier! In other words, *for uncorrelated, unit-variance Gaussian features, the basic linear classifier is Bayes-optimal*. This is a good example of how a probabilistic viewpoint can justify particular models.

More generally, as long as the per-class covariance matrices are equal, the ML decision boundary will be linear, intersecting $\boldsymbol{\mu}^{\oplus} - \boldsymbol{\mu}^{\ominus}$ in the middle, but not at right angles if the features are correlated. This means that the basic linear classifier is only Bayes-optimal in this case if we first decorrelate and normalise the features. With non-equal class covariances the decision boundary will be hyperbolic. So, the three cases in Figure 9.3 generalise to the multivariate case.

We have now seen several examples of how the normal distribution links the probabilistic and geometric viewpoints. The multivariate normal distribution essentially translates distances into probabilities. This becomes obvious when we plug the definition of ☞*Mahalanobis distance* (Equation 8.1 on p.237) into Equation 9.2:

$$P(\mathbf{x}|\boldsymbol{\mu},\boldsymbol{\Sigma}) = \frac{1}{E_d}\exp\left(-\frac{1}{2}\left(\mathrm{Dis}_M(\mathbf{x},\boldsymbol{\mu}|\boldsymbol{\Sigma})\right)^2\right) \tag{9.4}$$

Similarly, the standard normal distribution translates Euclidean distances into probabilities:

$$P(\mathbf{x}|\mathbf{0},\mathbf{I}) = \frac{1}{(2\pi)^{d/2}}\exp\left(-\frac{1}{2}\left(\mathrm{Dis}_2(\mathbf{x},\mathbf{0})\right)^2\right)$$

Conversely, we see that *the negative logarithm of the Gaussian likelihood can be interpreted as a squared distance*:

$$-\ln P(\mathbf{x}|\boldsymbol{\mu},\boldsymbol{\Sigma}) = \ln E_d + \frac{1}{2}\left(\mathrm{Dis}_M(\mathbf{x},\boldsymbol{\mu}|\boldsymbol{\Sigma})\right)^2$$

The intuition is that the logarithm transforms the multiplicative probability scale into an additive scale (which, in the case of Gaussian distributions, corresponds to a squared distance). Since additive scales are often easier to handle, log-likelihoods are a common concept in statistics.

Another example of the link between the geometric and the probabilistic perspective occurs when we consider the question of estimating the parameters of a normal distribution. For example, suppose we want to estimate the mean $\boldsymbol{\mu}$ of a multivariate Gaussian distribution with given covariance matrix $\boldsymbol{\Sigma}$ from a set of data points X. The principle of *maximum-likelihood estimation* states that we should find the value of $\boldsymbol{\mu}$ that maximises the joint likelihood of X. Assuming that the elements of X were independently sampled, the joint likelihood decomposes into a product over the individual data points in X, and the maximum-likelihood estimate can be found as follows:

$$\begin{aligned}
\hat{\boldsymbol{\mu}} &= \arg\max_{\boldsymbol{\mu}} \prod_{\mathbf{x}\in X} P(\mathbf{x}|\boldsymbol{\mu},\boldsymbol{\Sigma}) \\
&= \arg\max_{\boldsymbol{\mu}} \prod_{\mathbf{x}\in X} \frac{1}{E_d} \exp\left(-\frac{1}{2}\left(\mathrm{Dis}_M(\mathbf{x},\boldsymbol{\mu}|\boldsymbol{\Sigma})\right)^2\right) && \text{using Equation 9.4} \\
&= \arg\min_{\boldsymbol{\mu}} \sum_{\mathbf{x}\in X} \left[\ln E_d + \frac{1}{2}\left(\mathrm{Dis}_M(\mathbf{x},\boldsymbol{\mu}|\boldsymbol{\Sigma})\right)^2\right] && \text{taking negative logarithms} \\
&= \arg\min_{\boldsymbol{\mu}} \sum_{\mathbf{x}\in X} \left(\mathrm{Dis}_M(\mathbf{x},\boldsymbol{\mu}|\boldsymbol{\Sigma})\right)^2 && \text{dropping constant term and factor}
\end{aligned}$$

We thus find that the maximum-likelihood estimate of the mean of a multivariate distribution is the point that minimises the total squared Mahalanobis distance to all points in X. For the identity covariance matrix $\boldsymbol{\Sigma} = \mathbf{I}$ we can replace Mahalanobis distance with Euclidean distance, and by Theorem 8.1 the point minimising total squared Euclidean distance to all points in X is the arithmetic mean $\frac{1}{|X|}\sum_{\mathbf{x}\in X}\mathbf{x}$.

As a final example of how geometric and probabilistic views of the same problem can be strongly connected I will now demonstrate how the ☞*least-squares solution to a linear regression problem* (Section 7.1) can be derived as a maximum-likelihood estimate. For ease of notation we will look at the univariate case discussed in Example 7.1. The starting point is the assumption that our training examples (h_i, y_i) are noisy measurements of true function points $(x_i, f(x_i))$: i.e., $y_i = f(x_i) + \epsilon_i$, where the ϵ_i are independently and identically distributed errors. (Notice the slight change of notation as y_i is now no longer the true function value.) We want to derive the maximum-likelihood estimates $\hat{y}_i$ of $f(x_i)$. We can derive this if we assume a particular noise distribution, for example Gaussian with variance σ_2. It then follows that each y_i is normally distributed with mean $a + bx_i$ and variance σ^2, and thus

$$P(y_i|a,b,\sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{\left(y_i - (a+bx_i)\right)^2}{2\sigma^2}\right)$$

Since the noise terms ϵ_i are independent for different i, so are the y_i and so the joint probability over all i is simply the product of n of these Gaussians:

$$\begin{aligned}
P(y_1,\ldots,y_n|a,b,\sigma^2) &= \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{\left(y_i - (a+bx_i)\right)^2}{2\sigma^2}\right) \\
&= \left(\frac{1}{\sqrt{2\pi\sigma^2}}\right)^n \exp\left(-\frac{\sum_{i=1}^{n}\left(y_i - (a+bx_i)\right)^2}{2\sigma^2}\right)
\end{aligned}$$

For ease of algebraic manipulation we take the negative natural logarithm:

$$-\ln P(y_1,\ldots,y_n|a,b,\sigma^2) = \frac{n}{2}\ln 2\pi + \frac{n}{2}\ln\sigma^2 + \frac{\sum_{i=1}^{n}\left(y_i - (a+bx_i)\right)^2}{2\sigma^2}$$

Taking the partial derivatives with respect to a, b and σ^2 and setting to zero in order to maximise the negative log likelihood gives the following three equations:

$$\sum_{i=1}^{n} y_i - (a + bx_i) = 0$$
$$\sum_{i=1}^{n} \left(y_i - (a + bx_i)\right) x_i = 0$$
$$\frac{n}{2}\frac{1}{\sigma^2} - \frac{\sum_{i=1}^{n}\left(y_i - (a+bx_i)\right)^2}{2(\sigma^2)^2} = 0$$

The first two equations are essentially the same as derived in Example 7.1 and give us $\hat{a} = \overline{y} - \hat{b}\overline{x}$ and $\hat{b} = \sigma_{xy}/\sigma_{xx}$, respectively. The third equation tells us that the sum of squared residuals is equal to $n\sigma^2$ and gives the maximum-likelihood estimate of the noise variance as $\left(\sum_{i=1}^{n}\left(y_i - (a+bx_i)\right)^2\right)/n$.

It is reassuring that the probabilistic viewpoint allows us to derive (ordinary) east-squares regression from first principles. On the other hand, a full treatment would require noise on the x-values as well (total least squares), but this complicates the mathematics and does not necessarily have a unique solution. This illustrates that *a good probabilistic treatment of a machine learning problem achieves a balance between solid theoretical foundations and the pragmatism required to obtain a workable solution*.

9.2 Probabilistic models for categorical data

To kill time during long drives to some faraway holiday destination, my sisters and I would often play games involving passing cars. For example, we would ask each other to look out for cars that had a particular colour, were from a particular country or had a particular letter on the numberplate. A binary question such as 'is the car blue?' is called a *Bernoulli trial* by statisticians. They are modelled as a binary random variable whose probability of success is fixed over each independent trial. We used a Bernoulli distribution to model the event of an e-mail being ham in Example 9.1. On top of such a random variable, other probability distributions can be built. For example, we may want to guess how many of the next n cars are blue: this is governed by the binomial distribution. Or the task may be to estimate how many cars we need to see until the first Dutch one: this number follows a geometric definition. Background 9.2 will help to refresh your memory regarding the main definitions.

Categorical variables or features (also called discrete or nominal) are ubiquitous in machine learning. Perhaps the most common form of the Bernoulli distribution models whether or not a word occurs in a document. That is, for the i-th word in our vocabulary we have a random variable X_i governed by a Bernoulli distribution. The joint distribution over the *bit vector* $X = (X_1, \ldots, X_k)$ is called a *multivariate Bernoulli distribution*. Variables with more than two outcomes are also common: for example,

The *Bernoulli distribution*, named after the Swiss seventeenth century mathematician Jacob Bernoulli, concerns Boolean or binary events with two possible outcomes: success or 1, and failure or 0. A Bernoulli distribution has a single parameter θ which gives the probability of success: hence $P(X=1)=\theta$ and $P(X=0)=1-\theta$. The Bernoulli distribution has expected value $\mathbb{E}[X]=\theta$ and variance $\mathbb{E}\left[(X-\mathbb{E}[X])^2\right]=\theta(1-\theta)$.
The *binomial distribution* arises when counting the number of successes S in n independent Bernoulli trials with the same parameter θ. It is described by

$$P(S=s)=\binom{n}{s}\theta^s(1-\theta)^{n-s} \text{ for } s\in\{0,\dots,n\}$$

This distribution has expected value $\mathbb{E}[S]=n\theta$ and variance $\mathbb{E}\left[(S-\mathbb{E}[S])^2\right]=n\theta(1-\theta)$.
The *categorical distribution* generalises the Bernoulli distribution to $k\geq 2$ outcomes. The parameter of the distribution is a k-vector $\boldsymbol{\theta}=(\theta_1,\dots,\theta_k)$ such that $\sum_{i=1}^k\theta_i=1$.
Finally, the *multinomial distribution* tabulates the outcomes of n independent and identically distributed (i.i.d.) categorical trials. That is, $\mathbf{X}=(X_1,\dots,X_k)$ is a k-vector of integer counts, and

$$P(\mathbf{X}=(x_1,\dots,x_k))=n!\frac{\theta_1^{x_1}}{x_1!}\cdots\frac{\theta_k^{x_k}}{x_k!}$$

with $\sum_{i=1}^k x_i=n$. Notice that setting $n=1$ gives us an alternative way of stating the categorical distribution as $P(\mathbf{X}=(x_1,\dots,x_k))=\theta_1^{x_1}\cdots\theta_k^{x_k}$, with exactly one of the x_i equal to 1 and the rest set to 0. Furthermore, setting $k=2$ gives an alternative expression for the Bernoulli distribution as $P(X=x)=\theta^x(1-\theta)^{1-x}$ for $x\in\{0,1\}$. It is also useful to note that if $\mathbf{X}$ follows a multinomial distribution, then each component X_i follows a binomial distribution with parameter θ_i.
We can estimate the parameters of these distributions by counting in a straightforward way. Suppose $a\ b\ a\ c\ c\ b\ a\ a\ b\ c$ is a sequence of words. We might be interested in individual words being a or not, and interpret the data as coming from 10 i.i.d. Bernoulli trials, which would allow us to estimate $\hat{\theta}_a=4/10=0.4$. This same parameter generates a binomial distribution of the number of occurrences of the word a in similar sequences. Alternatively, we can estimate the parameters of the categorical (word occurrences) and multinomial (word counts) distributions as $\hat{\boldsymbol{\theta}}=(0.4,0.3,0.3)$.

It is almost always a good idea to smooth these distributions by including *pseudo-counts*. Imagine our vocabulary includes the word d but we haven't yet observed it, then a maximum-likelihood estimate would set $\hat{\theta}_d=0$. We can smooth this by adding a virtual occurrence of each word to our observations, leading to $\hat{\boldsymbol{\theta}}'=(5/14,4/14,4/14,1/14)$. In the case of a binomial this is the Laplace correction.

Background 9.2. Probability distributions for categorical data.

every word position in an e-mail corresponds to a categorical variable with k outcomes, where k is the size of the vocabulary. The multinomial distribution manifests itself as a *count vector*: a histogram of the number of occurrences of all vocabulary words in a document. This establishes an alternative way of modelling text documents that allows the number of occurrences of a word to influence the classification of a document.

Both these document models are in common use. Despite their differences, they both assume independence between word occurrences, generally referred to as the *naive Bayes assumption*. In the multinomial document model, this follows from the very use of the multinomial distribution, which assumes that words at different word positions are drawn independently from the same categorical distribution. In the multivariate Bernoulli model we assume that the bits in a bit vector are statistically independent, which allows us to compute the joint probability of a particular bit vector $(x_1, \dots, x_k)$ as the product of the probabilities of each component $P(X_i = x_i)$. In practice, such word independence assumptions are often not true: if we know that an e-mail contains the word 'Viagra', we can be quite sure that it will also contain the word 'pill'. In any case, the experience is that, while the naive Bayes assumption almost certainly leads to poor probability estimates, it often doesn't harm ranking performance. This means that, provided the classification threshold is chosen with some care, we can usually get good classification performance too.

Using a naive Bayes model for classification

Assume that we have chosen one of the possible distributions to model our data X. In a classification context, we furthermore assume that the distribution depends on the class, so that $P(X|Y = \texttt{spam})$ and $P(X|Y = \texttt{ham})$ are different distributions. The more different these two distributions are, the more useful the features X are for classification. Thus, for a specific e-mail x we calculate both $P(X = x|Y = \texttt{spam})$ and $P(X = x|Y = \texttt{ham})$, and apply one of several possible decision rules:

maximum likelihood (ML)	– predict $\arg\max_y P(X = x|Y = y)$;
maximum a posteriori (MAP)	– predict $\arg\max_y P(X = x|Y = y)P(Y = y)$;
recalibrated likelihood	– predict $\arg\max_y w_y P(X = x|Y = y)$.

The relation between the first two decision rules is that ML classification is equivalent to MAP classification with a uniform class distribution. The third decision rule generalises the first two in that it replaces the class distribution with a set of weights learned from the data: this makes it possible to correct for estimation errors in the likelihoods, as we shall see later.

Example 9.4 (Prediction using a naive Bayes model). Suppose our vocabulary contains three words a, b and c, and we use a multivariate Bernoulli model for our e-mails, with parameters

$$\boldsymbol{\theta}^{\oplus} = (0.5, 0.67, 0.33) \qquad \boldsymbol{\theta}^{\ominus} = (0.67, 0.33, 0.33)$$

This means, for example, that the presence of b is twice as likely in spam (+), compared with ham.

The e-mail to be classified contains words a and b but not c, and hence is described by the bit vector $\mathbf{x} = (1, 1, 0)$. We obtain likelihoods

$$P(\mathbf{x}|\oplus) = 0.5 \cdot 0.67 \cdot (1 - 0.33) = 0.222 \qquad P(\mathbf{x}|\ominus) = 0.67 \cdot 0.33 \cdot (1 - 0.33) = 0.148$$

The ML classification of $\mathbf{x}$ is thus spam. In the case of two classes it is often convenient to work with likelihood ratios and odds. The likelihood ratio can be calculated as $\frac{P(\mathbf{x}|\oplus)}{P(\mathbf{x}|\ominus)} = \frac{0.5}{0.67}\frac{0.67}{0.33}\frac{1-0.33}{1-0.33} = 3/2 > 1$. This means that the MAP classification of $\mathbf{x}$ is also spam if the prior odds are more than $2/3$, but ham if they are less than that. For example, with 33% spam and 67% ham the prior odds are $\frac{P(\oplus)}{P(\ominus)} = \frac{0.33}{0.67} = 1/2$, resulting in a posterior odds of $\frac{P(\oplus|\mathbf{x})}{P(\ominus|\mathbf{x})} = \frac{P(\mathbf{x}|\oplus)}{P(\mathbf{x}|\ominus)}\frac{P(\oplus)}{P(\ominus)} = 3/2 \cdot 1/2 = 3/4 < 1$. In this case the likelihood ratio for $\mathbf{x}$ is not strong enough to push the decision away from the prior.

Alternatively, we can employ a multinomial model. The parameters of a multinomial establish a distribution over the words in the vocabulary, say

$$\boldsymbol{\theta}^{\oplus} = (0.3, 0.5, 0.2) \qquad \boldsymbol{\theta}^{\ominus} = (0.6, 0.2, 0.2)$$

The e-mail to be classified contains three occurrences of word a, one single occurrence of word b and no occurrences of word c, and hence is described by the count vector $\mathbf{x} = (3, 1, 0)$. The total number of vocabulary word occurrences is $n = 4$. We obtain likelihoods

$$P(\mathbf{x}|\oplus) = 4!\frac{0.3^3}{3!}\frac{0.5^1}{1!}\frac{0.2^0}{0!} = 0.054 \qquad P(\mathbf{x}|\ominus) = 4!\frac{0.6^3}{3!}\frac{0.2^1}{1!}\frac{0.2^0}{0!} = 0.1728$$

The likelihood ratio is $\left(\frac{0.3}{0.6}\right)^3\left(\frac{0.5}{0.2}\right)^1\left(\frac{0.2}{0.2}\right)^0 = 5/16$. The ML classification of $\mathbf{x}$ is thus ham, the opposite of the multivariate Bernoulli model. This is mainly because of the three occurrences of word a, which provide strong evidence for ham.

Notice how the likelihood ratio for the multivariate Bernoulli model is a product of factors $\theta_i^{\oplus}/\theta_i^{\ominus}$ if $x_i = 1$ in the bit vector to be classified, and $(1-\theta_i^{\oplus})/(1-\theta_i^{\ominus})$ if $x_i = 0$. For the multinomial model the factors are $\left(\theta_i^{\oplus}/\theta_i^{\ominus}\right)^{x_i}$. One consequence of this is that the multinomial model only takes the presence of words into account, whereas in the multivariate Bernoulli model absent words can make a difference. In the previous example, not containing word b corresponds to a factor of $(1-0.67)/(1-0.33) = 1/2$ in the likelihood ratio. The other main difference between the two models is that multiple occurrences of words are treated like duplicated features in the multinomial model, through the exponential 'weight' x_i. This becomes clearer by taking the logarithm of the likelihood ratio, which is $\sum_i x_i(\ln\theta_i^{\oplus} - \ln\theta_i^{\ominus})$: this expression is linear in $\ln\theta_i^{\oplus}$ and $\ln\theta_i^{\ominus}$ with x_i as weights. Notice that this does not mean that naive Bayes classifiers are linear in the sense discussed in Chapter 7 unless we can demonstrate a linear relationship between $\ln\theta$ and the corresponding feature value. But we can say that naive Bayes models are linear in a particular space (the 'log-odds' space) obtained by applying a well-defined transformation to the features. We will return to this point when we discuss ☞*feature calibration* in Section 10.2.

The fact that the joint likelihood ratio of a naive Bayes model factorises as a product of likelihood ratios of individual words is a direct consequence of the naive Bayes assumption. In other words, the learning task decomposes into univariate tasks, one for each word in the vocabulary. We have encountered such a decomposition before when we discussed ☞*multivariate linear regression* in Section 7.1. There, we saw an example of how ignoring feature correlation could be harmful. Can we come up with similar examples for naive Bayes classifiers? Consider the situation when a particular word occurs twice in the vocabulary. In that case, we have the same factor occurring twice in the product for the likelihood ratio, and are effectively giving the word in question twice the weight of other words. While this is an extreme example, such double-counting does have noticeable effects in practice. I previously gave the example that if a spam e-mail contains the word 'Viagra', it is also expected to contain the word 'pill', so seeing the two words together should not give much more evidence for spam than seeing the first word on its own, and the likelihood ratio for the two words should not be much higher than that of the first word. However, multiplying two likelihood ratios larger than 1 will result in an even larger likelihood ratio. As a result, the probability estimates of a naive Bayes classifier are often pushed too far towards 0 or 1.

This may not seem such a big deal if we are only interested in classification, and not in the probability estimates as such. However, *an often overlooked consequence of having uncalibrated probability estimates such as those produced by naive Bayes is that both the ML and MAP decision rules become inadequate*. Unless we have evidence that the model assumptions are satisfied, the only sensible thing to do in this case is to invoke the *recalibrated likelihood decision rule*, which requires one to learn a weight vector

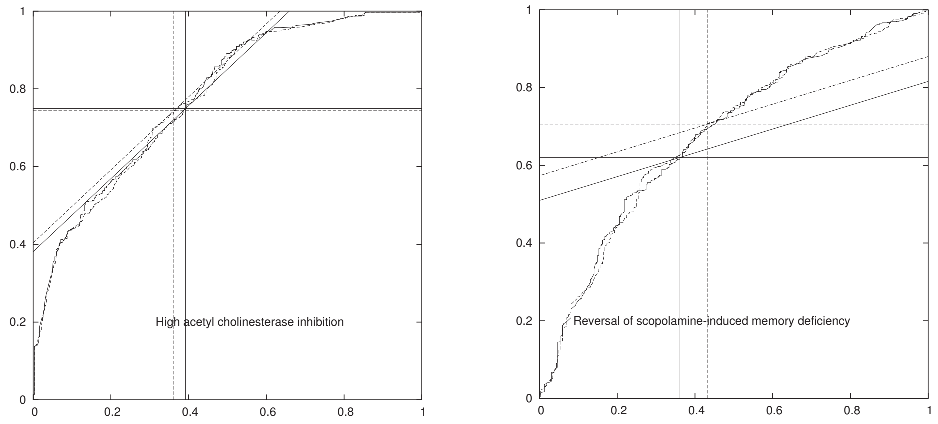


Figure 9.4. (left) ROC curves produced by two naive Bayes classifiers (solid line: a variant of the multivariate Bernoulli model; dashed line: a variant of the multinomial model). Both models have similar ranking performance and yield almost the same – more or less optimal – MAP decision threshold. **(right)** On a different data set from the same domain, the multinomial model's MAP threshold is slightly better, hinting at somewhat better calibrated probability estimates. But since the slope of the accuracy isometrics indicates that there are about four positives for every negative, the optimal decision rule is in fact to always predict positive.

over the classes, in order to correct for the estimation errors in the likelihoods. Specifically, we want to find weights w_i such that predicting $\arg\max_y w_y P(X = x|Y = y)$ results in the smallest possible loss – e.g., the number of misclassified examples – over a test set. For two classes this can be solved by the same procedure we considered for ☞*turning rankers into classifiers* in Section 2.2. To see this, notice that for two classes the recalibrated likelihood decision rule can be rewritten as

- ☞ predict positive if $w^{\oplus} P(X = x|Y = \oplus) > w^{\ominus} P(X = x|Y = \ominus)$ and negative otherwise; which is equivalent to
- ☞ predict positive if $P(X = x|Y = \oplus)/P(X = x|Y = \ominus) > w^{\ominus}/w^{\oplus}$ and negative otherwise

This demonstrates that in the two-class case we really have just one degree of freedom, as multiplying the weights by a constant does not affect the decisions. In other words, what we are interested in is finding the best threshold $t = w^{\ominus}/w^{\oplus}$ on the likelihood ratio, which is essentially the same problem as finding the best operating point on an ROC curve. The solution is given by the point on the highest accuracy isometric. Figure 9.4 illustrates this on two real-life data sets: in the left figure we see that the MAP decision threshold is more or less optimal, whereas in the right figure the optimal point is in the top right-hand corner.

For more than two classes, finding a globally optimal weight vector is computationally intractable, which means that we need to resort to a heuristic method. In Section 3.1 such a method was demonstrated for three classes. The idea is to fix the weights one by one, using some ordering of the classes. That is, we use the two-class procedure to optimally separate the i-th class from the previous $i-1$ classes.

Training a naive Bayes model

Training a probabilistic model usually involves estimating the parameters of the distributions used in the model. The parameter of a Bernoulli distribution can be estimated by counting the number of successes d in n trials and setting $\hat{\theta} = d/n$. In other words, we count, for each class, how many e-mails contain the word in question. Such relative frequency estimates are usually smoothed by including *pseudo-counts*, representing the outcome of virtual trials according to some fixed distributions. In the case of a Bernoulli distribution the most common smoothing operation is the Laplace correction, which involves two virtual trials, one of which results in success and the other in failure. Consequently, the relative frequency estimate is changed to $(d+1)/(n+2)$. From a Bayesian perspective this amounts to adopting a uniform prior, representing our initial belief that success and failure are equally likely. If appropriate, we can strengthen the influence of the prior by including a larger number of virtual trials, which means that more data is needed to move the estimate away from the prior. For a categorical distribution smoothing adds one pseudo-count to each of the k categories, leading to the smoothed estimate $(d+1)/(n+k)$. The *m-estimate* generalises this further by making both the total number of pseudo-counts m and the way they are distributed over the categories into parameters. The estimate for the i-th category is defined as $(d+p_i m)/(n+m)$, where p_i is a distribution over the categories (i.e., $\sum_{i=1}^{k} p_i = 1$). Notice that smoothed relative frequency estimates – and hence products of such estimates – can never attain the extreme values $\hat{\theta} = 0$ or $\hat{\theta} = 1$.

Example 9.5 (Training a naive Bayes model). We now show how the parameter vectors in the previous example might have been obtained. Consider the following e-mails consisting of five words a, b, c, d, e:

e_1: $b\ d\ e\ b\ b\ d\ e$	e_5: $a\ b\ a\ b\ a\ b\ a\ e\ d$
e_2: $b\ c\ e\ b\ b\ d\ d\ e\ c\ c$	e_6: $a\ c\ a\ c\ a\ c\ a\ e\ d$
e_3: $a\ d\ a\ d\ e\ a\ e\ e$	e_7: $e\ a\ e\ d\ a\ e\ a$
e_4: $b\ a\ d\ b\ e\ d\ a\ b$	e_8: $d\ e\ d\ e\ d$

We are told that the e-mails on the left are spam and those on the right are ham, and so we use them as a small training set to train our Bayesian classifier. First,

E-mail	$\#a$	$\#b$	$\#c$	Class
e_1	0	3	0	+
e_2	0	3	3	+
e_3	3	0	0	+
e_4	2	3	0	+
e_5	4	3	0	–
e_6	4	0	3	–
e_7	3	0	0	–
e_8	0	0	0	–

E-mail	$a?$	$b?$	$c?$	Class
e_1	0	1	0	+
e_2	0	1	1	+
e_3	1	0	0	+
e_4	1	1	0	+
e_5	1	1	0	–
e_6	1	0	1	–
e_7	1	0	0	–
e_8	0	0	0	–

Table 9.1. (left) A small e-mail data set described by count vectors. **(right)** The same data set described by bit vectors.

we decide that d and e are so-called *stop words* that are too common to convey class information. The remaining words, a, b and c, constitute our vocabulary.

For the multinomial model, we represent each e-mail as a count vector, as in Table 9.1 (left). In order to estimate the parameters of the multinomial, we sum up the count vectors for each class, which gives $(5,9,3)$ for spam and $(11,3,3)$ for ham. To smooth these probability estimates we add one pseudo-count for each vocabulary word, which brings the total number of occurrences of vocabulary words to 20 for each class. The estimated parameter vectors are thus $\hat{\boldsymbol{\theta}}^{\oplus} = (6/20, 10/20, 4/20) = (0.3, 0.5, 0.2)$ for spam and $\hat{\boldsymbol{\theta}}^{\ominus} = (12/20, 4/20, 4/20) = (0.6, 0.2, 0.2)$ for ham.

In the multivariate Bernoulli model e-mails are represented by bit vectors, as in Table 9.1 (right). Adding the bit vectors for each class results in $(2,3,1)$ for spam and $(3,1,1)$ for ham. Each count is to be divided by the number of documents in a class, in order to get an estimate of the probability of a document containing a particular vocabulary word. Probability smoothing now means adding two pseudo-documents, one containing each word and one containing none of them. This results in the estimated parameter vectors $\hat{\boldsymbol{\theta}}^{\oplus} = (3/6, 4/6, 2/6) = (0.5, 0.67, 0.33)$ for spam and $\hat{\boldsymbol{\theta}}^{\ominus} = (4/6, 2/6, 2/6) = (0.67, 0.33, 0.33)$ for ham.

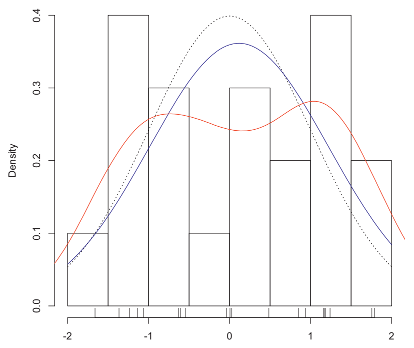
Many other variations of the naive Bayes classifier exist. In fact, what is normally understood as 'the' naive Bayes classifier employs neither a multinomial nor a multivariate Bernoulli model, but rather a multivariate categorical model. This means that features are categorical, and the probability of the i-th feature taking on its l-th value for class c examples is given by $\theta_{il}^{(c)}$, under the constraint that $\sum_{l=1}^{k_i} \theta_{il}^{(c)} = 1$, where k_i

is the number of values of the i-th feature. These parameters can be estimated by smoothed relative frequencies in the training set, as in the multivariate Bernoulli case. We again have that the joint probability of the feature vector is the product of the individual feature probabilities, and hence $P(F_i, F_j|C) = P(F_i|C)P(F_j|C)$ for all pairs of features and for all classes.

Notice, by the way, that conditional independence is quite different from unconditional independence: neither implies the other. To see that conditional independence does not imply unconditional independence, imagine two words that are very likely to occur in spam, but they are independent (i.e., the probability of both of them occurring in a spam e-mail is the product of the marginal probabilities). Imagine further that they are very unlikely – but also independent – in ham. Suppose I tell you an unclassified e-mail contains one of the words: you would probably guess that it is a spam e-mail, from which you would further guess that it also contains the other word – demonstrating that the words are not unconditionally independent. To see that unconditional independence does not imply conditional independence, consider two different independent words, and let an e-mail be spam if it contains at least one of the words and ham otherwise, then among spam e-mails the two words are dependent (since if I know that a spam e-mail doesn't contain one of the words, then it must contain the other).

Another extension of the naive Bayes model is required when some of the features are real-valued. One option is to discretise the real-valued features in a pre-processing stage: this will be discussed in Chapter 10. Another option is to assume that the feature values are normally distributed within each class, as discussed in the previous section. In this context it is worth noting that the naive Bayes assumption boils down to assuming a diagonal covariance matrix within each class, so that each feature can be treated independently. A third option that is also used in practice is to model the class-conditional likelihood of each feature by a non-parametric density estimator. These three options are illustrated in Figure 9.5.

In summary, the naive Bayes model is a popular model for dealing with textual, categorical and mixed categorical/real-valued data. Its main shortcoming as a probabilistic model – poorly calibrated probability estimates – are outweighed by generally good ranking performance. Another apparent paradox with naive Bayes is that it isn't particularly Bayesian at all! For one thing, we have seen that the poor probability estimates necessitate the use of reweighted likelihoods, which avoids using Bayes' rule altogether. Secondly, in training a naive Bayes model we use maximum-likelihood parameter estimation, whereas a fully fledged Bayesian approach would not commit to a particular parameter value, but rather employ a full posterior distribution. Personally, I think the essence of naive Bayes is the decomposition of joint likelihoods into marginal likelihoods. This decomposition is evocatively visualised by the Scottish tartan pattern in Figure 1.3 on p.31, which is why I like to call naive Bayes the 'Scottish classifier'.



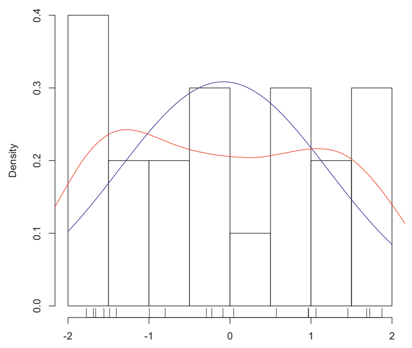


Figure 9.5. **(left)** Examples of three density estimators on 20 points sampled from a normal distribution with zero mean and unit variance (dotted line)). A histogram is a simple non-parametric method which employs a fixed number of equal-width intervals. A kernel density estimator (in red) applies interpolation to obtain a smooth density function. The solid bell curve (in blue) is obtained by estimating the sample mean and variance, assuming the true distribution is normal. **(right)** Here, the 20 points are sampled uniformly from $[-2,2]$, and the non-parametric methods generally do better.

9.3 Discriminative learning by optimising conditional likelihood

In the introduction to this chapter we distinguished between generative and discriminative probabilistic models. Naive Bayes models are generative: after training they can be used to generate data. In this section we look at one of the most commonly used discriminative models: *logistic regression*.[5] The easiest way to understand logistic regression is as a linear classifier whose probability estimates have been logistically calibrated using the method described in Section 7.4, but with one crucial difference: calibration is an integral part of the training algorithm, rather than a post-processing step. While in generative models the decision boundary is a by-product of modelling the distributions of each class, logistic regression models the decision boundary directly. For example, if the classes are overlapping then logistic regression will tend to locate the decision boundary in an area where classes are maximally overlapping, regardless of the 'shapes' of the samples of each class. This results in decision boundaries that are noticeably different from those learned by generative classifiers (Figure 9.6).

Equation 7.13 on p.222 expresses the likelihood ratio as $\exp\big(\gamma(d(\mathbf{x})-d_0)\big)$ with $d(\mathbf{x}) = \mathbf{w}\cdot\mathbf{x}-t$. Since we are learning the parameters all at once in discriminative learning, we can absorb γ and d_0 into $\mathbf{w}$ and t. So the logistic regression model is

[5]Notice that the term 'regression' is a bit of a misnomer here, since, even though a probability estimator approximates an unknown function, the training labels are classes rather than true function values.

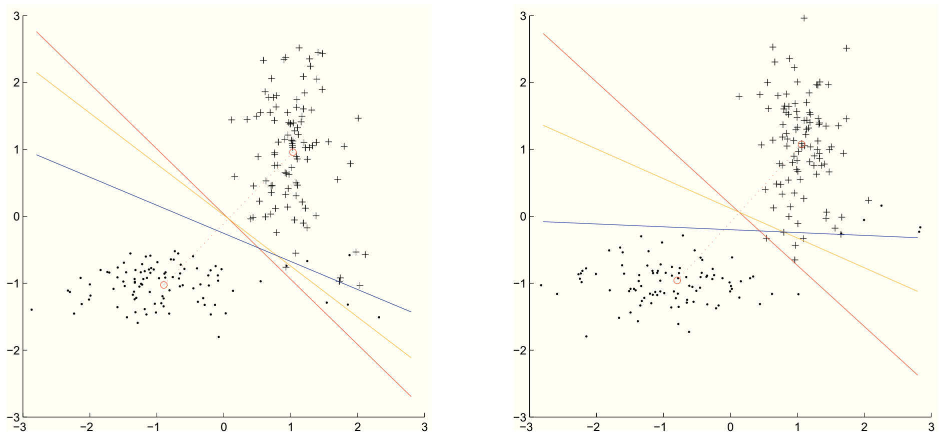


Figure 9.6. (left) On this data set, logistic regression (in blue) outperforms the basic linear classifier (in red) and the least squares classifier (in orange) because the latter two are more sensitive to the shape of the classes, while logistic regression concentrates on where the classes overlap. **(right)** On this slightly different set of points, logistic regression is outperformed by the other two methods because it concentrates too much on tracking the transition from mostly positive to mostly negative.

simply given by

$$\hat{p}(\mathbf{x}) = \frac{\exp(\mathbf{w}\cdot\mathbf{x}-t)}{\exp(\mathbf{w}\cdot\mathbf{x}-t)+1} = \frac{1}{1+\exp(-(\mathbf{w}\cdot\mathbf{x}-t))}$$

Assuming the class labels are $y = 1$ for positives and $y = 0$ for negatives, this defines a Bernoulli distribution for each training example:

$$P(y_i|\mathbf{x}_i) = \hat{p}(\mathbf{x}_i)^{y_i}(1-\hat{p}(\mathbf{x}_i))^{(1-y_i)}$$

It is important to note that the parameters of these Bernoulli distributions are linked through $\mathbf{w}$ and t, and consequently there is one parameter for every feature dimension, rather than for every training instance.

The likelihood function is

$$\mathrm{CL}(\mathbf{w},t) = \prod_i P(y_i|\mathbf{x}_i) = \prod_i \hat{p}(\mathbf{x}_i)^{y_i}(1-\hat{p}(\mathbf{x}_i))^{(1-y_i)}$$

This is called *conditional likelihood* to stress that it gives us the *conditional* probability $P(y_i|\mathbf{x}_i)$ rather than $P(\mathbf{x}_i)$ as in a generative model. Notice that our use of the product requires the assumption that the y-values are independent given $\mathbf{x}$; but this is an entirely reasonable assumption and not nearly as strong as the naive Bayes assumption of $\mathbf{x}$ being independent within each class. As usual, the logarithm of the likelihood function is easier to work with:

$$\mathrm{LCL}(\mathbf{w},t) = \sum_i y_i \ln \hat{p}(\mathbf{x}_i) + (1-y_i)\ln(1-\hat{p}(\mathbf{x}_i)) = \sum_{\mathbf{x}^{\oplus}\in Tr^{\oplus}} \ln \hat{p}(\mathbf{x}^{\oplus}) + \sum_{\mathbf{x}^{\ominus}\in Tr^{\ominus}} \ln(1-\hat{p}(\mathbf{x}^{\ominus}))$$

We want to maximise the log-conditional likelihood with respect to these parameters, which means that all partial derivatives must be zero:

$$\nabla_{\mathbf{w}}\mathrm{LCL}(\mathbf{w},t) = \mathbf{0}$$
$$\frac{\partial}{\partial t}\mathrm{LCL}(\mathbf{w},t) = 0$$

Although these equations do not yield an analytic solution, they can be used to obtain further insight into the nature of logistic regression. Concentrating on t, we first need to do some algebraic groundwork.

$$\begin{aligned}
\ln \hat{p}(\mathbf{x}) &= \ln \frac{\exp(\mathbf{w}\cdot\mathbf{x}-t)}{\exp(\mathbf{w}\cdot\mathbf{x}-t)+1} \\
&= \mathbf{w}\cdot\mathbf{x} - t - \ln(\exp(\mathbf{w}\cdot\mathbf{x}-t)+1) \\
\frac{\partial}{\partial t}\ln \hat{p}(\mathbf{x}) &= -1 - \frac{\partial}{\partial t}\ln(\exp(\mathbf{w}\cdot\mathbf{x}-t)+1) \\
&= -1 - \frac{1}{\exp(\mathbf{w}\cdot\mathbf{x}-t)+1}\exp(\mathbf{w}\cdot\mathbf{x}-t)\cdot(-1) \\
&= \hat{p}(\mathbf{x}) - 1
\end{aligned}$$

Similarly for the negatives:

$$\begin{aligned}
\ln(1-\hat{p}(\mathbf{x})) &= \ln \frac{1}{\exp(\mathbf{w}\cdot\mathbf{x}-t)+1} \\
&= -\ln(\exp(\mathbf{w}\cdot\mathbf{x}-t)+1) \\
\frac{\partial}{\partial t}\ln(1-\hat{p}(\mathbf{x})) &= \frac{\partial}{\partial t} -\ln(\exp(\mathbf{w}\cdot\mathbf{x}-t)+1) \\
&= \frac{-1}{\exp(\mathbf{w}\cdot\mathbf{x}-t)+1}\exp(\mathbf{w}\cdot\mathbf{x}-t)\cdot(-1) \\
&= \hat{p}(\mathbf{x})
\end{aligned}$$

It follows that the partial derivative of LCL with respect to t has a simple form:

$$\begin{aligned}
\frac{\partial}{\partial t}\mathrm{LCL}(\mathbf{w},t) &= \sum_{\mathbf{x}^{\oplus}\in Tr^{\oplus}} (\hat{p}(\mathbf{x}) - 1) + \sum_{\mathbf{x}^{\ominus}\in Tr^{\ominus}} \hat{p}(\mathbf{x}^{\ominus}) \\
&= \sum_{\mathbf{x}_i\in Tr} (\hat{p}(\mathbf{x}_i) - y_i)
\end{aligned}$$

For the optimal solution this partial derivative is zero. What this means is that, on average, the predicted probability should be equal to the proportion of positives *pos*. This is a satisfying result, as it is clearly a desirable global property of a calibrated classifier.

Notice that grouping models such as probability estimating trees have this property by construction, as they set the predicted probability equal to the empirical probability in a segment.

A very similar derivation leads to the partial derivative of the log-conditional likelihood with respect to the j-th weight w_j. The point to note here is that, whereas $\frac{\partial}{\partial t}(\mathbf{w}\cdot\mathbf{x}-t) = -1$, we have $\frac{\partial}{\partial w_j}(\mathbf{w}\cdot\mathbf{x}-t) = \frac{\partial}{\partial w_j}\left(\sum_j w_j x_j - t\right) = x_j$, the instance's j-th feature value. This then leads to

$$\frac{\partial}{\partial w_j}\mathrm{LCL}(\mathbf{w},t) = \sum_{\mathbf{x}_i\in Tr}(y_i - \hat{p}(\mathbf{x}_i))x_{ij} \tag{9.5}$$

Setting this partial derivative to zero expresses another, feature-wise calibration property. For example, if the j-th feature is a sparse Boolean feature that is mostly zero, then this calibration property only involves the instances $\mathbf{x}_i$ for which $x_{ij} = 1$: on average, those instances should have their predicted probability equal the proportion of positives among them.

Example 9.6 (Univariate logistic regression). Consider the data in Figure 9.7 with 20 points in each class. Although both classes were generated from normal distributions, class overlap in this particular sample is less than what could be expected on the basis of the class means. Logistic regression is able to take advantage of this and gives a much steeper sigmoid than the basic linear classifier with logistic calibration (explained in Example 7.7 on p.222), which is entirely formulated in terms of class means and variance. Also shown are the probability estimates obtained from the convex hull of the ROC curve (see Figure 7.13 on p.224); this calibration procedure is non-parametric and hence better able to detect the limited class overlap.

In terms of statistics, logistic regression has better mean squared error (0.040) than the logistically calibrated classifier (0.057). Isotonic calibration leads to the lowest error (0.021), but note that no probability smoothing has been applied to mitigate the risk of overfitting. The sum of predicted probabilities is 18.7 for the logistically calibrated classifier and 20 for the other two – i.e., equal to the number of examples, which is a necessary condition for full calibration. Finally, $\sum_{\mathbf{x}_i\in Tr}(y_i - \hat{p}(\mathbf{x}_i))x_i$ is 2.6 for the logistically calibrated classifier, 4.7 for the ROC-calibrated classifier, and 0 for logistic regression as expected from Equation 9.5.

In order to train a logistic regression model we need to find

$$\mathbf{w}^*, t^* = \arg\max_{\mathbf{w},t}\mathrm{CL}(\mathbf{w},t) = \arg\max_{\mathbf{w},t}\mathrm{LCL}(\mathbf{w},t)$$

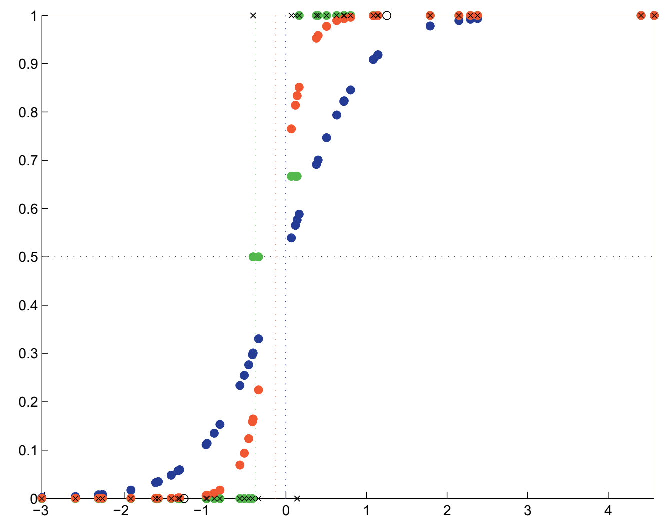


Figure 9.7. Logistic regression (in red) compared with probability estimates obtained by logistic calibration (in blue) and isotonic calibration (in green); the latter two are applied to the basic linear classifier (estimated class means are indicated by circles). The corresponding three decision boundaries are shown as vertical dotted lines.

This can be shown to be a convex optimisation problem, which means that there is only one maximum. A range of optimisation techniques can be applied. One simple approach is inspired by the perceptron algorithm and iterates over examples, using the following update rule:

$$\mathbf{w} = \mathbf{w} + \eta(y_i - \hat{p}_i)\mathbf{x}_i$$

where η is the learning rate. Notice the relationship with the partial derivative in Equation 9.5. Essentially, we are using single examples to approximate the direction of steepest ascent.

9.4 Probabilistic models with hidden variables

Suppose you are dealing with a four-class classification problem with classes A, B, C and D. If you have a sufficiently large and representative training sample of size n, you can use the relative frequencies in the sample $n_A, \dots, n_D$ to estimate the class prior $\hat{p}_A = n_A/n, \dots, \hat{p}_D = n_D/n$, as we have done many times before.[6] Conversely, if you know the prior and want to know the most likely class distribution in a random

[6]Of course, if you're not sure whether the sample is large enough it is better to smooth these relative frequency estimates by, e.g., the ☞*Laplace correction* (Section 2.3).

sample of n instances, you would use the prior to calculate expected values $\mathbb{E}[n_A] = p_A \cdot n, \dots, \mathbb{E}[n_D] = p_D \cdot n$. So, complete knowledge of one allows us to estimate or infer the other. However, sometimes we have a bit of knowledge about both. For example, we may know that $p_A = 1/2$ and that C is twice as likely as B, without knowing the complete prior. And we may know that the sample we saw last week was evenly split between $A \cup B$ and $C \cup D$, and that C and D were equally large, but we can't remember the size of A and B separately. What should we do?

Formalising what we know about the prior, we have $p_A = 1/2$; $p_B = \beta$, as yet unknown; $p_C = 2\beta$, since it is twice p_B; and $p_D = 1/2 - 3\beta$, since the four cases need to add up to 1. Furthermore: $n_A + n_B = a + b = s$, $n_C = c$ and $n_D = d$, with s, c and d known. We want to infer a, b and β: however, it seems we are stuck in a chicken-and-egg problem. If we knew β we would have full knowledge about the prior and we could use that to infer expected values for a and b:

$$\frac{\mathbb{E}[a]}{\mathbb{E}[b]} = \frac{1/2}{\beta} \qquad\qquad \mathbb{E}[a] + \mathbb{E}[b] = s$$

from which we could derive

$$\mathbb{E}[a] = \frac{1}{1+2\beta}s \qquad\qquad \mathbb{E}[b] = \frac{2\beta}{1+2\beta}s \tag{9.6}$$

So, for example, if $s = 20$ and $\beta = 1/10$, then $\mathbb{E}[a] = 16\frac{2}{3}$ and $\mathbb{E}[b] = 3\frac{1}{3}$.

Conversely, if we knew a and b, then we could estimate β by maximum-likelihood estimation, using a multinomial distribution for a, b, c and d:

$$P(a,b,c,d|\beta) = K(1/2)^a \beta^b (2\beta)^c (1/2 - 3\beta)^d$$
$$\ln P(a,b,c,d|\beta) = \ln K + a\ln(1/2) + b\ln\beta + c\ln(2\beta) + d\ln(1/2 - 3\beta)$$

Here, K is a combinatorial constant that doesn't affect the value of β which maximises the likelihood. Taking the partial derivative with respect to β gives

$$\frac{\partial}{\partial\beta}\ln P(a,b,c,d|\beta) = \frac{b}{\beta} + \frac{2c}{2\beta} - \frac{3d}{1/2 - 3\beta}$$

Setting to 0 and solving for β finally gives

$$\hat{\beta} = \frac{b+c}{6(b+c+d)} \tag{9.7}$$

So, for example, if $b = 5$ and $c = d = 10$, then $\hat{\beta} = 1/10$.

The way out of this chicken-and-egg problem is to iterate the following two steps: (*i*) calculate an expected value of the missing frequencies a and b from an assumed or previously estimated value of the parameter β; and (*ii*) calculate a maximum-likelihood estimate of the parameter β from assumed or expected values of the missing frequencies a and b. These two steps are iterated until a stationary configuration is reached.

So, if we start with $a = 15$, $b = 5$ and $c = d = 10$, then we have just seen that $\hat{\beta} = 1/10$. Plugging this value of β into Equation 9.6 gives us $\mathbb{E}[a] = 16\frac{2}{3}$ and $\mathbb{E}[b] = 3\frac{1}{3}$. Plugging these values back into Equation 9.7 yields $\hat{\beta} = 2/21$, which in turn gives $\mathbb{E}[a] = 16.8$ and $\mathbb{E}[b] = 3.2$, and so on. A stationary configuration with $\beta = 0.0948$, $a = 16.813$ and $b = 3.187$ is reached in fewer than 10 iterations. In this simple case this is a global optimum that is reached regardless of the starting point, essentially because the relationship between b and β is monotonic ($\mathbb{E}[b]$ increases with β according to Equation 9.6 and $\hat{\beta}$ increases with b according to Equation 9.7). However, this is not normally the case: we will return to this point later.

Expectation-Maximisation

The problem that we have just discussed is an example of a problem with missing data, where the full data Y separates into observed variables X and *hidden variables* Z (also called *latent variables*). In the example, the observed variables are c, d and s, and the hidden variables are a and b. We also have model parameter(s) θ, which is β in the example.[7] Denote the estimate of θ in the t-th iteration as θ^t. We have two relevant quantities:

- ☞ the expectation $\mathbb{E}\left[Z|X,\theta^t\right]$ of the hidden variables given the observed variables and the current estimate of the parameters (so in Equation 9.6 the expectations of a and b depend on s and β);
- ☞ the likelihood $P(Y|\theta)$, which is used to find the maximising value of θ.

In the likelihood function we need values for $Y = X \cup Z$. We obviously use the observed values for X, but we need to use previously calculated expectations for Z. This means that we really want to maximise $P(X \cup \mathbb{E}\left[Z|X,\theta^t\right]|\theta)$, or equivalently, the logarithm of that function. We now make the assumption that the logarithm of the likelihood function is linear in Y: notice that this assumption is valid in the example above. For any linear function f, $f(\mathbb{E}[Z]) = \mathbb{E}\left[f(Z)\right]$ and thus we can bring the expectation outside in our objective function:

$$\ln P(X \cup \mathbb{E}\left[Z|X,\theta^t\right]|\theta) = \mathbb{E}\left[\ln P(X \cup Z|\theta)|X,\theta^t\right] = \mathbb{E}\left[\ln P(Y|\theta)|X,\theta^t\right] \tag{9.8}$$

This last expression is usually denoted as $Q(\theta|\theta^t)$, as it essentially tells us how to calculate the next value of θ from the current one:

$$\theta^{t+1} = \arg\max_{\theta} Q(\theta|\theta^t) = \arg\max_{\theta} \mathbb{E}\left[\ln P(Y|\theta)|X,\theta^t\right] \tag{9.9}$$

[7]Model parameters are also 'hidden' in a sense, but they are different from hidden variables in that you would never expect to observe the value of a parameter (e.g., a class mean), whereas a hidden variable could be observed in principle but happens to be unobserved in the case at hand.

This, then, is the general form of the celebrated *Expectation-Maximisation* (*EM*) algorithm, which is a powerful approach to probabilistic modelling with hidden variables or missing data. Similar to the example above, we iterate over assigning an expected value to the hidden variables given our current estimates of the parameters, and re-estimating the parameters from these updated expectations, until a stationary configuration is reached. We can start the iteration by initialising either the parameters or the hidden variables in some way. The algorithm bears a striking resemblance to the ☞*K-means* algorithm (Algorithm 8.1 on p.248), which also iterates over assigning data points to current cluster means, and re-estimating the cluster means from the new assignments. This resemblance is not accidental, as we shall see in a moment. Like the K-means algorithm, EM can be proved to always converge to a stationary configuration for a wide class of probabilistic models. However, EM can get trapped in a local optimum that is dependent on the initial configuration.

Gaussian mixture models

A common application of Expectation-Maximisation is to estimate the parameters of a *Gaussian mixture model* from data. In such a model the data points are generated by K normal distributions, each with their own mean $\boldsymbol{\mu}_j$ and covariance matrix $\boldsymbol{\Sigma}_j$, and the proportion of points coming from each Gaussian is governed by a prior $\boldsymbol{\tau} = (\tau_1, \ldots, \tau_K)$. If each data point in a sample were labelled with the index of the Gaussian it came from this would be a straightforward classification problem, which could be solved easily by estimating each Gaussian's $\boldsymbol{\mu}_j$ and $\boldsymbol{\Sigma}_j$ separately from the data points belonging to class j. However, we are now considering the much harder predictive clustering problem in which the class labels are hidden and need to be reconstructed from the observed feature values.

A convenient way to model this is to have for each data point $\mathbf{x}_i$ a Boolean vector $\mathbf{z}_i = (z_{i1}, \ldots, z_{iK})$ such that exactly one bit z_{ij} is set to 1 and the rest set to 0, signalling that the i-th data point comes from the j-th Gaussian. Using this notation we can adapt the expression for the ☞*multivariate normal distribution* (Equation 9.2 on p.267) to obtain a general expression for a Gaussian mixture model:

$$P(\mathbf{x}_i, \mathbf{z}_i | \theta) = \sum_{j=1}^{K} z_{ij} \tau_j \frac{1}{(2\pi)^{d/2}\sqrt{|\boldsymbol{\Sigma}_j|}} \exp\left(-\frac{1}{2}(\mathbf{x}_i - \boldsymbol{\mu}_j)^{\mathrm{T}} \boldsymbol{\Sigma}_j^{-1} (\mathbf{x}_i - \boldsymbol{\mu}_j)\right) \tag{9.10}$$

Here, θ collects all the parameters τ, $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K$ and $\boldsymbol{\Sigma}_1, \ldots, \boldsymbol{\Sigma}_K$. The interpretation as a generative model is as follows: we first randomly select a Gaussian using the prior $\boldsymbol{\tau}$, and then we invoke the corresponding Gaussian using the indicator variables z_{ij}.

In order to apply Expectation-Maximisation we form the Q function:

$$
\begin{aligned}
Q(\theta|\theta^t) &= \mathbb{E}\left[\ln P(\mathbf{X}\cup\mathbf{Z}|\theta)|\mathbf{X},\theta^t\right]\\
&= \mathbb{E}\left[\ln\prod_{i=1}^{n} P(\mathbf{x}_i\cup\mathbf{z}_i|\theta)\middle|\mathbf{X},\theta^t\right]\\
&= \mathbb{E}\left[\sum_{i=1}^{n}\ln P(\mathbf{x}_i\cup\mathbf{z}_i|\theta)\middle|\mathbf{X},\theta^t\right]\\
&= \mathbb{E}\left[\sum_{i=1}^{n}\ln\sum_{j=1}^{K} z_{ij}\tau_j\frac{1}{(2\pi)^{d/2}\sqrt{|\mathbf{\Sigma}_j|}}\exp\left(-\frac{1}{2}(\mathbf{x}_i-\boldsymbol{\mu}_j)^{\mathrm{T}}\mathbf{\Sigma}_j^{-1}(\mathbf{x}_i-\boldsymbol{\mu}_j)\right)\middle|\mathbf{X},\theta^t\right]\\
&= \mathbb{E}\left[\sum_{i=1}^{n}\sum_{j=1}^{K} z_{ij}\ln\left(\tau_j\frac{1}{(2\pi)^{d/2}\sqrt{|\mathbf{\Sigma}_j|}}\exp\left(-\frac{1}{2}(\mathbf{x}_i-\boldsymbol{\mu}_j)^{\mathrm{T}}\mathbf{\Sigma}_j^{-1}(\mathbf{x}_i-\boldsymbol{\mu}_j)\right)\right)\middle|\mathbf{X},\theta^t\right] \qquad (*)\\
&= \mathbb{E}\left[\sum_{i=1}^{n}\sum_{j=1}^{K} z_{ij}\left(\ln\tau_j-\frac{d}{2}\ln(2\pi)-\frac{1}{2}\ln|\mathbf{\Sigma}_j|-\frac{1}{2}(\mathbf{x}_i-\boldsymbol{\mu}_j)^{\mathrm{T}}\mathbf{\Sigma}_j^{-1}(\mathbf{x}_i-\boldsymbol{\mu}_j)\right)\middle|\mathbf{X},\theta^t\right]\\
&= \sum_{i=1}^{n}\sum_{j=1}^{K}\mathbb{E}\left[z_{ij}\middle|\mathbf{X},\theta^t\right]\left(\ln\tau_j-\frac{d}{2}\ln(2\pi)-\frac{1}{2}\ln|\mathbf{\Sigma}_j|-\frac{1}{2}(\mathbf{x}_i-\boldsymbol{\mu}_j)^{\mathrm{T}}\mathbf{\Sigma}_j^{-1}(\mathbf{x}_i-\boldsymbol{\mu}_j)\right)
\end{aligned}
\tag{9.11}
$$

The step marked (*) is possible because for a given i only one z_{ij} is switched on, hence we can bring the indicator variables outside the logarithm. The last line shows the Q function in the desired form, involving on the one hand expectations over the hidden variables conditioned on the observable data $\mathbf{X}$ and the previously estimated parameters θ^t, and on the other hand expressions in θ that allow us to find θ^{t+1} by maximisation.

The Expectation step of the EM algorithm is thus the calculation of the expected values of the indicator variables $\mathbb{E}\left[z_{ij}\middle|\mathbf{X},\theta^t\right]$. Notice that expectations of Boolean variables take values on the entire interval $[0,1]$, under the constraint that $\sum_{j=1}^{K} z_{ij} = 1$ for all i. In effect, the hard cluster assignment of K-means is changed into a soft assignment – one of the ways in which Gaussian mixture models generalise K-means. Now, suppose that $K = 2$ and we expect both clusters to be of equal size and with equal covariances. If a given data point $\mathbf{x}_i$ is equidistant from the two cluster means (or rather, our current estimates of these), then clearly $\mathbb{E}\left[z_{i1}|\mathbf{X},\theta^t\right] = \mathbb{E}\left[z_{i2}|\mathbf{X},\theta^t\right] = 1/2$. In the general case these expectations are apportioned proportionally to the probability mass assigned to the point by each Gaussian:

$$
\mathbb{E}\left[z_{ij}\middle|\mathbf{X},\theta^t\right] = \frac{\tau_j^t f(\mathbf{x}_i|\boldsymbol{\mu}_j^t,\mathbf{\Sigma}_j^t)}{\sum_{k=1}^{K}\tau_k^t f(\mathbf{x}_i|\boldsymbol{\mu}_k^t,\mathbf{\Sigma}_k^t)}
\tag{9.12}
$$

where $f(\mathbf{x}|\boldsymbol{\mu},\mathbf{\Sigma})$ stands for the multivariate Gaussian density function.

For the Maximisation step we optimise the parameters in Equation 9.11. Notice there is no interaction between the terms containing τ_j and the terms containing the other parameters, and so the prior distribution $\boldsymbol{\tau}$ can be optimised separately:

$$\begin{aligned}\boldsymbol{\tau}^{t+1} &= \underset{\boldsymbol{\tau}}{\arg\max} \sum_{i=1}^{n}\sum_{j=1}^{K} \mathbb{E}\left[z_{ij}\middle|\mathbf{X},\theta^{t}\right]\ln\tau_j \\ &= \underset{\boldsymbol{\tau}}{\arg\max} \sum_{j=1}^{K} E_j \ln\tau_j \qquad\qquad \text{under the constraint } \sum_{j=1}^{K}\tau_j = 1\end{aligned}$$

where I have written E_j for $\sum_{i=1}^{n}\mathbb{E}\left[z_{ij}\middle|\mathbf{X},\theta^{t}\right]$, which is the total (partial) membership of the j-th cluster – notice that $\sum_{j=1}^{K} E_j = n$. For simplicity we assume $K = 2$, so that $\tau_2 = 1-\tau_1$: then

$$\tau_1^{t+1} = \underset{\tau_1}{\arg\max}\, E_1\ln\tau_1 + E_2\ln(1-\tau_1)$$

Setting the derivative with respect to τ_1 to zero and solving for τ_1, it can be easily verified that $\tau_1^{t+1} = E_1/(E_1+E_2) = E_1/n$ and thus $\tau_2^{t+1} = E_2/n$. In the general case of K clusters we have analogously

$$\tau_j^{t+1} = \frac{E_j}{\sum_{k=1}^{K} E_k} = \frac{1}{n}\sum_{i=1}^{n}\mathbb{E}\left[z_{ij}\middle|\mathbf{X},\theta^{t}\right] \tag{9.13}$$

The means and covariance matrices can be optimised for each cluster separately:

$$\begin{aligned}\boldsymbol{\mu}_j^{t+1},\boldsymbol{\Sigma}_j^{t+1} &= \underset{\boldsymbol{\mu}_j,\boldsymbol{\Sigma}_j}{\arg\max}\sum_{i=1}^{n}\mathbb{E}\left[z_{ij}\middle|\mathbf{X},\theta^{t}\right]\left(-\frac{1}{2}\ln|\boldsymbol{\Sigma}_j| - \frac{1}{2}(\mathbf{x}_i-\boldsymbol{\mu}_j)^{\mathrm{T}}\boldsymbol{\Sigma}_j^{-1}(\mathbf{x}_i-\boldsymbol{\mu}_j)\right) \\ &= \underset{\boldsymbol{\mu}_j,\boldsymbol{\Sigma}_j}{\arg\min}\sum_{i=1}^{n}\mathbb{E}\left[z_{ij}\middle|\mathbf{X},\theta^{t}\right]\left(\frac{1}{2}\ln|\boldsymbol{\Sigma}_j| + \frac{1}{2}(\mathbf{x}_i-\boldsymbol{\mu}_j)^{\mathrm{T}}\boldsymbol{\Sigma}_j^{-1}(\mathbf{x}_i-\boldsymbol{\mu}_j)\right)\end{aligned}$$

Notice that the term between brackets is a squared-distance term with the expectations functioning as instance weights on each instance. This describes a generalised version of the problem of finding the point that ☞*minimises the sum of squared Euclidean distances* to a set of points (Theorem 8.1 on p.238). While that problem is solved by the arithmetic mean, here we simply take the *weighted* average over all the points:

$$\boldsymbol{\mu}_j^{t+1} = \frac{1}{E_j}\sum_{i=1}^{n}\mathbb{E}\left[z_{ij}\middle|\mathbf{X},\theta^{t}\right]\mathbf{x}_i = \frac{\sum_{i=1}^{n}\mathbb{E}\left[z_{ij}\middle|\mathbf{X},\theta^{t}\right]\mathbf{x}_i}{\sum_{i=1}^{n}\mathbb{E}\left[z_{ij}\middle|\mathbf{X},\theta^{t}\right]} \tag{9.14}$$

Similarly, the covariance matrix is computed as a weighted average of covariance matrices obtained from each data point, taking into account the newly estimated mean:

$$\begin{aligned}\boldsymbol{\Sigma}_j^{t+1} &= \frac{1}{E_j}\sum_{i=1}^{n}\mathbb{E}\left[z_{ij}\middle|\mathbf{X},\theta^{t}\right](\mathbf{x}_i-\boldsymbol{\mu}_j^{t+1})(\mathbf{x}_i-\boldsymbol{\mu}_j^{t+1})^{\mathrm{T}} \\ &= \frac{\sum_{i=1}^{n}\mathbb{E}\left[z_{ij}\middle|\mathbf{X},\theta^{t}\right](\mathbf{x}_i-\boldsymbol{\mu}_j^{t+1})(\mathbf{x}_i-\boldsymbol{\mu}_j^{t+1})^{\mathrm{T}}}{\sum_{i=1}^{n}\mathbb{E}\left[z_{ij}\middle|\mathbf{X},\theta^{t}\right]}\end{aligned} \tag{9.15}$$

Equations 9.12–9.15, then, constitute the EM solution to learning a Gaussian mixture model from an unlabelled sample. I have presented it here in its most general

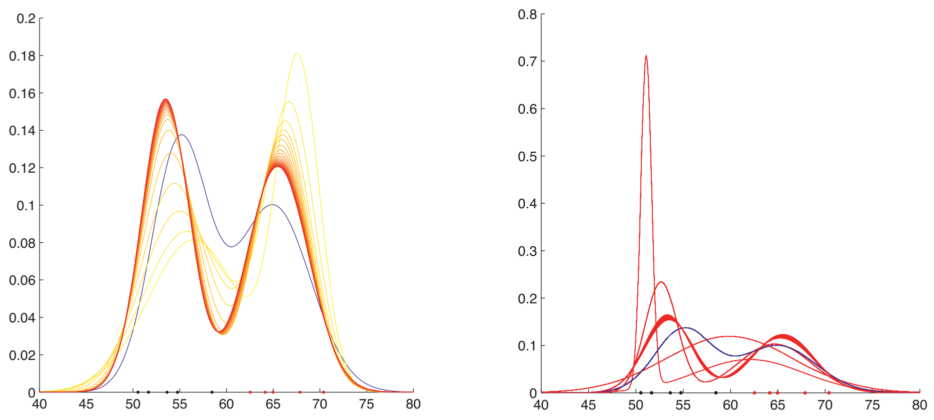


Figure 9.8. (left) The blue line shows the true Gaussian mixture model from which the 10 points on the x-axis were sampled; the colour of the points indicates whether they came from the left or the right Gaussian. The other lines show convergence of Expectation-Maximisation to a stationary configuration from a random initialisation. **(right)** This plot shows four stationary configurations for the same set of points. The EM algorithm was run for 20 iterations; the thickness of one of the lines demonstrates that this configuration takes longer to converge.

form, explicitly modelling unequal cluster sizes and covariance matrices. The latter is important as it allows for clusters of different shapes, unlike the K-means algorithm which assumes that all clusters have the same spherical shape. Consequently, the boundaries between clusters will not be linear, as they are in the clusterings learned by K-means. Figure 9.8 demonstrates the convergence of EM on a simple univariate data set, as well as the existence of multiple stationary configurations.

In conclusion, Expectation-Maximisation is a versatile and powerful method to deal with missing variables in a principled way. As we have seen in detail for the Gaussian mixture model, the main ingredient is an expression for the parametric likelihood function $P(X \cup Z|\theta)$, from which the update equations can be derived by means of the Q function. A word of caution is also in order, since – except in the simplest cases – there will be more than one stationary configuration. Like with K-means, the optimisation should therefore be run multiple times with different starting configurations.

9.5 Compression-based models

We end this chapter with a brief discussion of an approach to machine learning that is both closely related to and quite distinct from the probabilistic approach. Consider the maximum a posteriori decision rule again:

$$y_{\text{MAP}} = \arg\max_{y} P(X = x|Y = y)P(Y = y)$$

Y	$P(\text{Viagra}=1\|Y)$	$IC(\text{Viagra}=1\|Y)$	$P(\text{Viagra}=0\|Y)$	$IC(\text{Viagra}=0\|Y)$
spam	0.40	**1.32 bits**	0.60	**0.74 bits**
ham	0.12	**3.06 bits**	0.88	**0.18 bits**

Table 9.2. Example marginal likelihoods.

Taking negative logarithms, we can turn this into an equivalent minimisation:

$$y_{\text{MAP}} = \arg\min_y -\log P(X=x|Y=y) - \log P(Y=y) \tag{9.16}$$

This follows because for any two probabilities $0 < p < p' < 1$ we have $\infty > -\log p > -\log p' > 0$. If an event has probability p of happening, the negative logarithm of p quantifies the *information content* of the message that the event has indeed happened. This makes intuitive sense, as the less expected an event is, the more information an announcement of the event contains. The unit of information depends on the base of the logarithm: it is customary to take logarithms to the base 2, in which case information is measured in bits. For example, if you toss a fair coin once and tell me it came up heads, this contains $-\log_2 1/2 = 1$ bit of information; if you roll a fair die once and let me know it came up six, the information content of your message is $-\log_2 1/6 = 2.6$ bits. Equation 9.16 tells us that the MAP decision rule chooses the least surprising or the most expected class for an instance x given particular prior distributions and likelihoods. We write $IC(X|Y) = -\log_2 P(X|Y)$ and $IC(Y) = -\log_2 P(Y)$.

Example 9.7 (Information-based classification). Table 9.2 reproduces the left table in Table 1.3 on p.29 together with the relevant information content quantities. If Y is uniformly distributed then $IC(Y=\text{spam}) = 1$ bit and $IC(Y=\text{ham}) = 1$ bit. It follows that

$$\arg\min_y \big(IC(\text{Viagra}=1|Y=y) + IC(Y=y)\big) = \text{spam}$$

$$\arg\min_y \big(IC(\text{Viagra}=0|Y=y) + IC(Y=y)\big) = \text{ham}$$

If ham is four times as likely as spam then $IC(Y=\text{spam}) = 2.32$ bit and $IC(Y=\text{ham}) = 0.32$ bit, and $\arg\min_y \big(IC(\text{Viagra}=1|Y=y) + IC(Y=y)\big) = \text{ham}$.

Clearly, for a uniform distribution over k outcomes, each outcome has the same information content $-\log_2 1/k = \log_2 k$. For a non-uniform distribution these information contents differ, and hence it makes sense to compute the average information content or *entropy* $\sum_{i=1}^{k} -p_i \log_2 p_i$. We have encountered entropy before as an ☞*impurity measure* in Section 5.1.

So far I have not really told you anything new, other than that there is a one-to-one relationship between probability and information content. What really kicks things off in compression-based learning is a fundamental result from information theory proved by Claude Shannon in 1948. Shannon's result says – loosely speaking – that we cannot transmit information at a rate that surpasses entropy, but we can get arbitrarily close to the optimal rate by designing clever binary codes. Some well-known codes include the Shannon–Fanon code and the Huffman code, which are worth looking up as they employ a simple tree structure to build the code from empirical probabilities. Even more efficient codes, such as arithmetic coding, combine multiple messages into a single code word.

Assuming the availability of a near-optimal code, we can now turn the tables and use information content – or 'description length' as it is more commonly called – as a proxy for probability. One simplified version of the minimum description length (MDL) principle runs as follows.

Definition 9.1 (Minimum description length principle). *Let $L(m)$ denote the length in bits of a description of model m, and let $L(D|m)$ denote the length in bits of a description of data D given model m. According to the minimum description length principle, the preferred model is the one minimising the description length of model and data given model:*

$$m_{\mathrm{MDL}} = \operatorname*{arg\,min}_{m \in M} \left(L(m) + L(D|m)\right) \tag{9.17}$$

In a predictive learning context, 'description of data given model' refers to whatever information we need, in addition to the model and the feature values of the data, to infer the target labels. If the model is 100% accurate no further information is needed, so this term essentially quantifies the extent to which the model is incorrect. For example, in a uniform two-class setting we need one bit for every data point incorrectly classified by the model. The term $L(m)$ quantifies the complexity of the model. For instance, if we are fitting a polynomial to the data we need to encode the degree of the polynomial as well as its roots, up to a certain resolution. MDL learning thus trades off accuracy and complexity of a model: the complexity term serves to avoid overfitting in a similar way to the ☞*regularisation* term in ridge regression in Section 7.1 and the ☞*slack variable* term in soft-margin SVMs in Section 7.3.

What encoding to use in order to determine the model complexity $L(m)$ is often not straightforward and to some extent subjective. This is similar to the Bayesian perspective, where we need to define a prior distribution on models. The MDL viewpoint offers a concrete way of defining model priors by means of codes.

9.6 Probabilistic models: Summary and further reading

In this chapter we covered a range of machine learning models that are all based on the idea that features and target variables can be modelled as random variables, giving the opportunity to explicitly represent and manipulate the level of certainty we have about those variables. Such models are usually predictive in that they result in a conditional distribution $P(Y|X)$ with which Y can be predicted from X. Generative models estimate the joint distribution $P(Y,X)$ – often through the likelihood $P(X|Y)$ and the prior $P(Y)$ – from which the posterior $P(Y|X)$ can be obtained, while conditional models learn the posterior $P(Y|X)$ directly without spending resources on learning $P(X)$. The 'Bayesian' approach to machine learning is characterised by concentrating on the full posterior distribution wherever this is feasible, rather than just deriving a maximising value.

☞ In Section 9.1 we saw that the normal or Gaussian distribution supports many useful geometric intuitions, essentially because the negative logarithm of the Gaussian likelihood can be interpreted as a squared distance. Straight decision boundaries result from having the same per-class covariance matrices, which means that models resulting in such linear boundaries, including linear classifiers, linear regression and K-means clustering, can be interpreted from a probabilistic viewpoint that makes their inherent assumptions explicit. Two examples of this are that the basic linear classifier is Bayes-optimal for uncorrelated, unit-variance Gaussian features; and least-squares regression is optimal for linear functions contaminated by Gaussian noise on the target variable.

☞ Section 9.2 was devoted to different versions of the naive Bayes classifier, which makes the simplifying assumption that features are independent within each class. Lewis (1998) gives an overview and history. This model is widely used in information retrieval and text classification as it is often a good ranker if not a good probability estimator. While the model that is usually understood as naive Bayes treats features as categorical or Bernoulli random variables, variants employing a multinomial model tend to better model the number of occurrences of words in a document (McCallum and Nigam, 1998). Real-valued features can be taken into account by either modelling them as normally distributed within each class, or by non-parametric density estimation – John and Langley (1995) suggest that the latter gives better empirical results. Webb, Boughton and Wang (2005) discuss ways of relaxing the strong independence assumptions made by naive Bayes. Probability smoothing by means of the m-estimate was introduced by Cestnik (1990).

☞ Perhaps paradoxically, I don't think there is anything particularly 'Bayesian' about the naive Bayes classifier. While it is a generative probabilistic model estimating the posterior $P(Y|X)$ through the joint $P(Y,X)$, in practice the posterior is very poorly calibrated owing to the unrealistic independence assumptions. The reason naive Bayes is often successful is because of the quality of $\arg\max_Y P(Y|X)$ rather than the quality of the posterior as such, as analysed by Domingos and Pazzani (1997). Furthermore, even the use of Bayes' rule in determining the maximising Y can be avoided, as it only serves to transform uncalibrated likelihoods into uncalibrated posteriors. So my recommendation is to use naive Bayes likelihoods as scores on an unknown scale whose decision threshold needs to be calibrated by means of ROC analysis, as has been discussed several times before.

☞ In Section 9.3 we looked at the widely used logistic regression model. The basic idea is to combine a linear decision boundary with logistic calibration, but to train this in a discriminative fashion by optimising conditional likelihood. So, rather than modelling the classes as clouds of points and deriving a decision boundary from those clouds, logistic regression concentrates on areas of class overlap. It is an instance of the larger class of generalised linear models (Nelder and Wedderburn, 1972). Jebara (2004) discusses the advantages of discriminative learning in comparison with generative models. Discriminative learning can also be applied to sequential data in the form of conditional random fields (Lafferty *et al.*, 2001)

☞ Section 9.4 presented the Expectation-Maximisation algorithm as a general way of learning models involving unobserved variables. This general form of EM was proposed by Dempster, Laird and Rubin (1977) based on a variety of earlier work. We have seen how it can be applied to Gaussian mixture models to obtain a more general version of K-means predictive clustering, which is also able to estimate cluster shapes and sizes. However, this increases the number of parameters of the model and thus the risk of getting stuck in a non-optimal stationary configuration. (Little and Rubin, 1987) is a standard reference for dealing with missing data.

☞ Finally, in Section 9.5 we briefly discussed some ideas related to learning as compression. The link with probabilistic modelling is that both seek to model and exploit the non-random aspects of the data. In a simplified setting, the minimum description length principle can be derived from Bayes' rule by taking the negative logarithm, and states that models minimising the description length of the model and of the data given the model should be preferred. The first term quantifies the complexity of the model, and the second term quantifies its accuracy (as only the model's errors need to be encoded explicitly). The advantage of the MDL principle is that encoding schemes are often more tangible and easier to define than prior distributions. However, not just any encoding will do: as with their probabilistic counterparts, these schemes need to be justified in the domain being modelled. Pioneering work in this area has been done by Solomonoff (1964*a*,*b*); Wallace and Boulton (1968); Rissanen (1978), among others. An excellent introduction and overview is provided by Grünwald (2007).

CHAPTER *10*

Features

PREVIOUSLY I REFERRED to features as 'the workhorses of machine learning' – it is therefore high time to consider them in more detail. Features, also called attributes, are defined as mappings $f_i : \mathscr{X} \rightarrow \mathscr{F}_i$ from the instance space $\mathscr{X}$ to the feature domain $\mathscr{F}_i$. We can distinguish features by their domain: common feature domains include real and integer numbers, but also discrete sets such as colours, the Booleans, and so on. We can also distinguish features by the range of permissible operations. For example, we can calculate a group of people's average age but not their average blood type, so taking the average value is an operation that is permissible on some features but not on others. We will take a closer look at different kinds of feature in Section 10.1.

Although many data sets come with pre-defined features, they can be manipulated in many ways. For example, we can change the domain of a feature by rescaling or discretisation; we can select the best features from a larger set and only work with the selected ones; or we can combine two or more features into a new feature. In fact, a model itself is a way of constructing a new feature that solves the task at hand. Feature transformations will be investigated in Section 10.2, while feature construction and selection is the topic of Section 10.3

10.1 Kinds of feature

Consider two features, one describing a person's age and the other their house number. Both features map into the integers, but the way we use those features can be quite different. Calculating the average age of a group of people is meaningful, but an average house number is probably not very useful! In other words, what matters is not just the domain of a feature, but also the range of permissible operations. These, in turn, depend on whether the feature values are expressed on a meaningful *scale*. Despite appearances, house numbers are not really integers but *ordinals*: we can use them to determine that number 10's neighbours are number 8 and number 12, but we cannot assume that the distance between 8 and 10 is the same as the distance between 10 and 12. Because of the absence of a linear scale it is not meaningful to add or subtract house numbers, which precludes operations such as averaging.

Calculations on features

Let's take a closer look at the range of possible calculations on features, often referred to as aggregates or statistics. Three main categories are *statistics of central tendency*, *statistics of dispersion* and *shape statistics*. Each of these can be interpreted either as a theoretical property of an unknown population or a concrete property of a given sample – here we will concentrate on sample statistics.

Starting with statistics of central tendency, the most important ones are

- the *mean* or average value;
- the *median*, which is the middle value if we order the instances from lowest to highest feature value; and
- the *mode*, which is the majority value or values.

Of these statistics, the mode is the one we can calculate whatever the domain of the feature: so, for example, we can say that the most frequent blood type in a group of people is O+. In order to calculate the median, we need to have an ordering on the feature values: so we can calculate both the mode and the median house number in a set of addresses.[1] In order to calculate the mean, we need a feature expressed on some scale: most often this will be a linear scale for which we calculate the familiar arithmetic mean, but Background 10.1 discusses means for some other scales. It is often suggested that the median tends to lie between the mode and the mean, but there are plenty of exceptions to this 'rule'. The famous statistician Karl Pearson suggested a

[1] If our sample contains an even number of instances, there are two middle values. If the feature has a scale it is customary to take the mean of those two values as the median; if the feature doesn't have a scale, or if it is important that we select a value actually occurring in the sample, we can either select both as the lower and upper median, or we can make a random choice.

Imagine a swimmer who swims the same distance d on two different days, taking a seconds one day and b seconds the next. On average, it took her therefore $c = (a+b)/2$ seconds, with an average speed of $d/c = 2d/(a+b)$. Notice how this average speed is *not* calculated as the normal or *arithmetic mean* of the speeds, which would yield $(d/a + d/b)/2$: to calculate average speed over a fixed distance we use a different mean called the *harmonic mean*. Given two numbers x and y (in our swimming example these are the speeds on either day, d/a and d/b), the harmonic mean h is defined as

$$h(x, y) = \frac{2}{1/x + 1/y} = \frac{2xy}{x+y}$$

Since $1/h(x, y) = (1/x + 1/y)/2$, we observe that calculating the harmonic mean on a scale with unit u corresponds to calculating the arithmetic mean on the *reciprocal scale* with unit $1/u$. In the example, speed with fixed distance is expressed on a scale reciprocal to the time scale, and since we use the arithmetic mean to average time, we use the harmonic mean to average speed. (If we average speed over a fixed *time* interval this is expressed on the same scale as distance and thus we would use the arithmetic mean.)

A good example of where the harmonic mean is used in machine learning arises when we average precision and recall of a classifier. Remember that precision is the proportion of positive predictions that is correct ($prec = TP/(TP+FP)$), and recall is the proportion of positives that is correctly predicted ($rec = TP/(TP+FN)$). Suppose we first calculate the number of mistakes averaged over the classes: this is the arithmetic mean $Fm = (FP + FN)/2$. We can then derive

$$\frac{TP}{TP+Fm} = \frac{TP}{TP+(FP+FN)/2} = \frac{2TP}{(TP+FP)+(TP+FN)} = \frac{2}{1/prec + 1/rec}$$

We recognise the last term as the harmonic mean of precision and recall. Since the enumerator of both precision and recall is fixed, taking the arithmetic mean of the denominators corresponds to taking the harmonic mean of the ratios. In information retrieval this harmonic mean of precision and recall is very often used and called the *F-measure*.

Yet other means exist for other scales. In music, going from one note to a note one octave higher corresponds to doubling the frequency. So frequencies f and $4f$ are two octaves apart, and it makes sense to take the octave in between with frequency $2f$ as their mean. This is achieved by the *geometric mean*, which is defined as $g(x, y) = \sqrt{xy}$. Since $\log\sqrt{xy} = (\log xy)/2 = (\log x + \log y)/2$ it follows that the geometric mean corresponds to the arithmetic mean on a logarithmic scale. All these means have in common that the mean of two values is an intermediate value, and that they can easily be extended to more than two values.

Background 10.1. On scales and means.

more specific rule of thumb (with therefore even more exceptions): the median tends to fall one-third of the way from mean to mode.

The second kind of calculation on features are statistics of dispersion or 'spread'. Two well-known statistics of dispersion are the *variance* or average squared deviation from the (arithmetic) mean, and its square root, the *standard deviation*. Variance and standard deviation essentially measure the same thing, but the latter has the advantage that it is expressed on the same scale as the feature itself. For example, the variance of the body weight in kilograms of a group of people is measured in kg^2 (kilograms-squared), whereas the standard deviation is measured in kilograms. The absolute difference between the mean and the median is never larger than the standard deviation – this is a consequence of *Chebyshev's inequality*, which states that at most $1/k^2$ of the values are more than k standard deviations away from the mean.

A simpler dispersion statistic is the difference between maximum and minimum value, which is called the *range*. A natural statistic of central tendency to be used with the range is the *midrange point*, which is the mean of the two extreme values. These definitions assume a linear scale but can be adapted to other scales using suitable transformations. For example, for a feature expressed on a logarithmic scale, such as frequency, we would take the ratio of the highest and lowest frequency as the range, and the harmonic mean of these two extremes as the midrange point.

Other statistics of dispersion include *percentiles*. The p-th percentile is the value such that p per cent of the instances fall below it. If we have 100 instances, the 80th percentile is the value of the 81st instance in a list of increasing values.[2] If p is a multiple of 25 the percentiles are also called *quartiles*, and if it is a multiple of 10 the percentiles are also called *deciles*. Note that the 50th percentile, the 5th decile and the second quartile are all the same as the median. Percentiles, deciles and quartiles are special cases of *quantiles*. Once we have quantiles we can measure dispersion as the distance between different quantiles. For instance, the *interquartile range* is the difference between the third and first quartile (i.e., the 75th and 25th percentile).

Example 10.1 (Percentile plot). Suppose you are learning a model over an instance space of countries, and one of the features you are considering is the gross domestic product (GDP) per capita. Figure 10.1 shows a so-called *percentile plot* of this feature. In order to obtain the p-th percentile, you intersect the line $y = p$ with the dotted curve and read off the corresponding percentile on the x-axis. Indicated in the figure are the 25th, 50th and 75th percentile. Also indicated is the

[2] Similar to the median there are issues with non-integer ranks, and they can be dealt with in different ways; however, significant differences do not arise unless the sample size is very small.

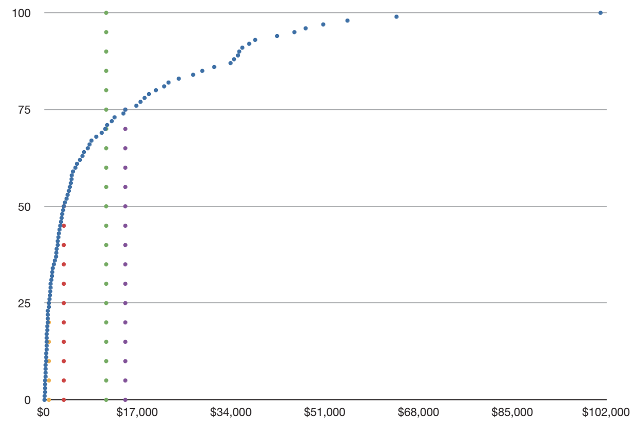


Figure 10.1. Percentile plot of GDP per capita for 231 countries (data obtained from `www.wolframalpha.com` by means of the query 'GDP per capita').The vertical dotted lines indicate, from left to right: the first quartile ($900); the median ($3600); the mean ($11 284); andthe third quartile ($14 750). The interquartile range is $13 850, while the standard deviation is $16 189.

mean (which has to be calculated from the raw data). As you can see, the mean is considerably higher than the median; this is mainly because of a few countries with very high GDP per capita. In other words, the mean is more sensitive to *outliers* than the median, which is why the median is often preferred to the mean for skewed distributions like this one.

You might think that the way I drew the percentile plot is the wrong way around: surely it would make more sense to have p on the x-axis and the percentiles on the y-axis? One advantage of drawing the plot this way is that, by interpreting the y-axis as probabilities, the plot can be read as a *cumulative probability distribution*: a plot of $P(X \leq x)$ against x for a random variable X. For example, the plot shows that $P(X \leq \mu)$ is approximately 0.70, where $\mu = \$11\,284$ is the mean GDP per capita. In other words, if you choose a random country the probability that its GDP per capita is less than the average is about 0.70.

Since GDP per capita is a real-valued feature, it doesn't necessarily make sense to talk about its mode, since if you measure the feature precisely enough every country will have a different value. We can get around this by means of a *histogram*, which counts the number of feature values in a particular interval or bin.

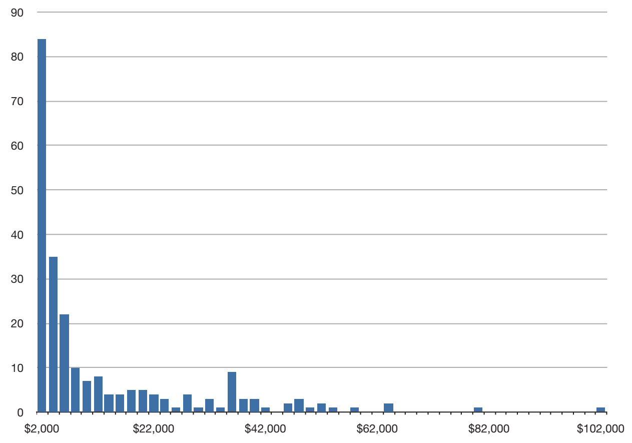


Figure 10.2. Histogram of the data from Figure 10.1, with bins of \$2000 wide.

Example 10.2 (Histogram). A histogram of the data from Example 10.1 is shown in Figure 10.2. The left-most bin is the mode, with well over a third of the countries having a GDP per capita of not more than \$2000. This demonstrates that the distribution is extremely *right-skewed* (i.e., has a long right tail), resulting in a mean that is considerably higher than the median.

The skew and 'peakedness' of a distribution can be measured by shape statistics such as skewness and kurtosis. The main idea is to calculate the third and fourth *central moment* of the sample. In general, the k-th central moment of a sample $\{x_i, \ldots, x_n\}$ is defined as $m_k = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^k$, where μ is the sample mean. Clearly, the first central moment is the average deviation from the mean – this is always zero, as the positive and negative deviations cancel each other out – and the second central moment is the average squared deviation from the mean, otherwise known as the variance. The third central moment m_3 can again be positive or negative. *Skewness* is then defined as m_3/σ^3, where σ is the sample's standard deviation. A positive value of skewness means that the distribution is right-skewed, which means that the right tail is longer than the left tail. Negative skewness indicates the opposite, left-skewed case. *Kurtosis* is defined as m_4/σ^4. As it can be shown that a normal distribution has kurtosis 3, people often use *excess kurtosis* $m_4/\sigma^4 - 3$ as the statistic of interest. Briefly, positive excess kurtosis means that the distribution is more sharply peaked than the normal distribution.

Kind	*Order*	*Scale*	*Tendency*	*Dispersion*	*Shape*
Categorical	×	×	mode	n/a	n/a
Ordinal	√	×	median	quantiles	n/a
Quantitative	√	√	mean	range, interquartile range, variance, standard deviation	skewness, kurtosis

Table 10.1. Kinds of feature, their properties and allowable statistics. Each kind inherits the statistics from the kinds above it in the table. For instance, the mode is a statistic of central tendency that can be computed for any kind of feature.

Example 10.3 (Skewness and kurtosis). In the GDP per capita example we can calculate skewness as 2.12 and excess kurtosis as 2.53. This confirms that the distribution is heavily right-skewed, and also more sharply peaked than the normal distribution.

Categorical, ordinal and quantitative features

Given these various statistics we can distinguish three main kinds of feature: those with a meaningful numerical scale, those without a scale but with an ordering, and those without either. We will call features of the first type *quantitative*; they most often involve a mapping into the reals (another term in common use is 'continuous'). Even if a feature maps into a subset of the reals, such as age expressed in years, the various statistics such as mean or standard deviation still require the full scale of the reals.

Features with an ordering but without scale are called *ordinal features*. The domain of an ordinal feature is some totally ordered set, such as the set of characters or strings. Even if the domain of a feature is the set of integers, denoting the feature as ordinal means that we have to dispense with the scale, as we did with house numbers. Another common example are features that express a rank order: first, second, third, and so on. Ordinal features allow the mode and median as central tendency statistics, and quantiles as dispersion statistics.

Features without ordering or scale are called *categorical features* (or sometimes 'nominal' features). They do not allow any statistical summary except the mode. One subspecies of the categorical features is the *Boolean feature*, which maps into the truth values true and false. The situation is summarised in Table 10.1.

Models treat these different kinds of feature in distinct ways. First, consider tree models such as decision trees. A split on a categorical feature will have as many children as there are feature values. Ordinal and quantitative features, on the other hand, give rise to a binary split, by selecting a value v_0 such that all instances with a feature value less than or equal to v_0 go to one child, and the remaining instances to the other child. It follows that tree models are insensitive to the scale of quantitative features. For example, whether a temperature feature is measured on the Celsius scale or on the Fahrenheit scale will not affect the learned tree. Neither will switching from a linear scale to a logarithmic scale have any effect: the split threshold will simply be $\log v_0$ instead of v_0. In general, tree models are insensitive to *monotonic* transformations on the scale of a feature, which are those transformations that do not affect the relative order of the feature values. In effect, *tree models ignore the scale of quantitative features, treating them as ordinal*. The same holds for rule models.

Now let's consider the naive Bayes classifier. We have seen that this model works by estimating a likelihood function $P(X|Y)$ for each feature X given the class Y. For categorical and ordinal features with k values this involves estimating $P(X = v_1|Y), \ldots, P(X = v_k|Y)$. In effect, ordinal features are treated as categorical ones, ignoring the order. Quantitative features cannot be handled at all, unless they are discretised into a finite number of bins and thus converted to categoricals. Alternatively, we could assume a parametric form for $P(X|Y)$, for instance a normal distribution. We will return to this later in this chapter when we discuss feature calibration.

While naive Bayes only really handles categorical features, many geometric models go in the other direction: they can only handle quantitative features. Linear models are a case in point: the very notion of linearity assumes a Euclidean instance space in which features act as Cartesian coordinates, and thus need to be quantitative. Distance-based models such as k-nearest neighbour and K-means require quantitative features if their distance metric is Euclidean distance, but we can adapt the distance metric to incorporate categorical features by setting the distance to 0 for equal values and 1 for unequal values (the ☞*Hamming distance* as defined in Section 8.1). In a similar vein, for ordinal features we can count the number of values between two feature values (if we encode the ordinal feature by means of integers, this would simply be their difference). This means that distance-based methods can accommodate all feature types by using an appropriate distance metric. Similar techniques can be used to extend support vector machines and other kernel-based methods to categorical and ordinal features.

Structured features

It is usually tacitly assumed that an instance is a vector of feature values. In other words, the instance space is a Cartesian product of d feature domains: $\mathscr{X} = \mathscr{F}_1 \times \ldots \times \mathscr{F}_d$. This means that there is no other information available about an instance apart from the information conveyed by its feature values. Identifying an instance with its vector of feature values is what computer scientists call an *abstraction*, which is the result of filtering out unnecessary information. Representing an e-mail as a vector of word frequencies is an example of an abstraction.

However, sometimes it is necessary to avoid such abstractions, and to keep more information about an instance than can be captured by a finite vector of feature values. For example, we could represent an e-mail as a long string; or as a sequence of words and punctuation marks; or as a tree that captures the HTML mark-up; and so on. Features that operate on such structured instance spaces are called *structured features*.

Example 10.4 (Structured features). Suppose an e-mail is represented as a sequence of words. This allows us to define, apart from the usual word frequency features, a host of other features, including:

☞ whether the phrase 'machine learning' – or any other set of consecutive words – occurs in the e-mail;
☞ whether the e-mail contains at least eight consecutive words in a language other than English;
☞ whether the e-mail is palindromic, as in 'Degas, are we not drawn onward, we freer few, drawn onward to new eras aged?'

Furthermore, we could go beyond properties of single e-mails and express relations such as whether one e-mail is quoted in another e-mail, or whether two e-mails have one or more passages in common.

Structured features are not unlike *queries* in a database query language such as SQL or a declarative programming language such as Prolog. In fact, we have already seen examples of structured features in Section 6.4 when we looked at learning Prolog clauses such as the following:

```
fish(X):-bodyPart(X,Y).
fish(X):-bodyPart(X,pairOf(Z)).
```

The first clause has a single structured feature in the body which tests for the existence of some unspecified body part, while the second clause has another structured feature testing for the existence of a pair of unspecified body parts. The defining characteristic of structured features is that they involve *local variables* that refer to objects other than the instance itself. In a logical language such as Prolog it is natural to interpret local variables as existentially quantified, as we just did. However, it is equally possible to use other forms of *aggregation* over local variables: e.g., we can count the number of body parts (or pairs of body parts) an instance has.

↓ to, from →	*Quantitative*	*Ordinal*	*Categorical*	*Boolean*
Quantitative	**normalisation**	**calibration**	**calibration**	**calibration**
Ordinal	**discretisation**	**ordering**	**ordering**	**ordering**
Categorical	**discretisation**	**unordering**	**grouping**	
Boolean	**thresholding**	**thresholding**	**binarisation**	

Table 10.2. An overview of possible feature transformations. **Normalisation and calibration** adapt the scale of quantitative features, or add a scale to features that don't have one. **Ordering** adds or adapts the order of feature values without reference to a scale. The other operations abstract away from unnecessary detail, either in a deductive way (**unordering**, **binarisation**) or by introducing new information (**thresholding**, **discretisation**).

Structured features can be constructed either prior to learning a model, or simultaneously with it. The first scenario is often called *propositionalisation* because the features can be seen as a translation from first-order logic to propositional logic without local variables. The main challenge with propositionalisation approaches is how to deal with combinatorial explosion of the number of potential features. Notice that features can be logically related: e.g., the second clause above covers a subset of the instances covered by the first one. It is possible to exploit this if structured feature construction is integrated with model building, as in inductive logic programming.

10.2 Feature transformations

Feature transformations aim at improving the utility of a feature by removing, changing or adding information. We could order feature types by the amount of detail they convey: quantitative features are more detailed than ordinal ones, followed by categorical features, and finally Boolean features. The best-known feature transformations are those that turn a feature of one type into another of the next type down this list. But there are also transformations that change the scale of quantitative features, or add a scale (or order) to ordinal, categorical and Boolean features. Table 10.2 introduces the terminology we will be using.

The simplest feature transformations are entirely deductive, in the sense that they achieve a well-defined result that doesn't require making any choices. *Binarisation* transforms a categorical feature into a set of Boolean features, one for each value of the categorical feature. This loses information since the values of a single categorical feature are mutually exclusive, but is sometimes needed if a model cannot handle more than two feature values. *Unordering* trivially turns an ordinal feature into a categorical one by discarding the ordering of the feature values. This is often required since most learning models cannot handle ordinal features directly. An interesting alternative that

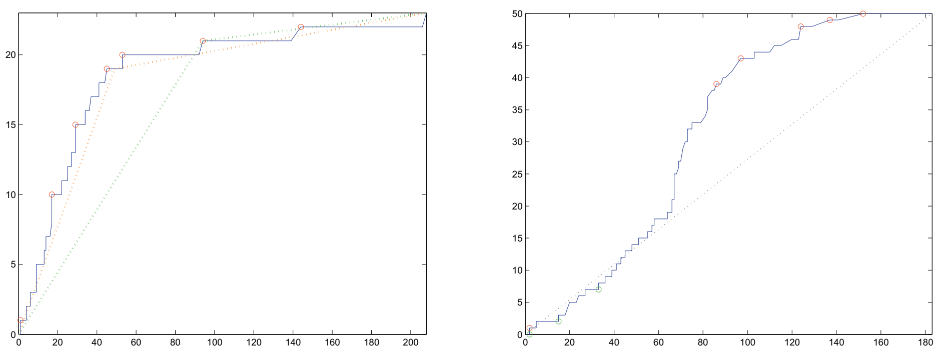


Figure 10.3. (left) Coverage curve obtained by ranking countries on decreasing GDP per capita, using 23 Euro countries as the positive class. The orange split sets the threshold equal to the mean, selecting 19 Euro countries and 49 non-Euro countries. The green split sets the threshold equal to the median, selecting 21 Euro countries and 94 non-Euro countries. The red points are on the convex hull of the coverage curve and indicate potentially optimal splits when the class label is taken into account. **(right)** Coverage curve of the same feature, using 50 countries in the Americas as the positive class. The red splits indicate potentially optimal thresholds with relatively many positives above the threshold, while the green splits indicate potentially optimal thresholds with relatively many positives below the threshold.

we will explore below is to add a scale to the feature by means of calibration.

In the remainder of this section we consider feature transformations that add information, the most important of which are discretisation and calibration.

Thresholding and discretisation

Thresholding transforms a quantitative or an ordinal feature into a Boolean feature by finding a feature value to split on. I briefly alluded to this in Chapter 5 as a way to split on quantitative features in decision trees. Concretely, let $f : \mathscr{X} \to \mathbb{R}$ be a quantitative feature and let $t \in \mathbb{R}$ be a threshold, then $f_t : \mathscr{X} \to \{\mathsf{true}, \mathsf{false}\}$ is a Boolean feature defined by $f_t(x) = \mathsf{true}$ if $f(x) \geq t$ and $f_t(x) = \mathsf{false}$ if $f(x) < t$. We can choose such thresholds in an unsupervised or a supervised way.

Example 10.5 (Unsupervised and supervised thresholding). Consider the GDP per capita feature plotted in Figure 10.1 again. Without knowing how this feature is to be used in a model, the most sensible thresholds are the statistics of central tendency such as the mean and the median. This is referred to as *unsupervised thresholding*.

In a *supervised* learning setting we can do more. For example, suppose we want to use the GDP per capita as a feature in a decision tree to predict whether a country is one of the 23 countries that use the Euro as their official currency (or as one of their currencies). Using the feature as a ranker, we can construct a coverage curve (Figure 10.3 (left)). We see that for this feature the mean is not the most obvious threshold, as it splits right in the middle of a run of negatives. A better split is obtained at the start of that run of negatives, or at the end of the following run of positives, indicated by the red points at either end of the mean split. More generally, any point on the convex hull of the coverage curve represents a candidate threshold; which one to choose is informed by whether we put more value on picking out positives or negatives. As it happens in this example, the median threshold is on the convex hull, but this cannot be guaranteed in general as, by definition, unsupervised thresholding methods select the threshold independently from the target.

Figure 10.3 (right) shows the same feature with a different target: whether a country is in the Americas. We see that part of the curve is below the ascending diagonal, indicating that, in comparison with the whole data set, the initial segment of the ranking contains a smaller proportion of American countries. This means that potentially useful thresholds can also be found on the *lower convex hull*.

In summary, unsupervised thresholding typically involves calculating some statistic over the data, whereas supervised thresholding requires sorting the data on the feature value and traversing down this ordering to optimise a particular objective function such as information gain. Non-optimal split points could be filtered out by means of constructing the upper and lower convex hull, but in practice this is unlikely to be more efficient computationally than a straightforward sweep over the sorted instances.

If we generalise thresholding to multiple thresholds we arrive at one of the most commonly used non-deductive feature transformations. *Discretisation* transforms a quantitative feature into an ordinal feature. Each ordinal value is referred to as a *bin* and corresponds to an interval of the original quantitative feature. Again, we can distinguish between supervised and unsupervised approaches. *Unsupervised discretisation* methods typically require one to decide the number of bins beforehand. A simple method that often works reasonably well is to choose the bins so that each bin has approximately the same number of instances: this is referred to as *equal-frequency discretisation*. If we choose two bins then this method coincides with thresholding on the median. More generally, the bin boundaries are quantiles: for instance, with 10 bins the bin boundaries of equal-width discretisation are deciles. Another unsupervised discretisation method is *equal-width discretisation*, which chooses the bin boundaries so that each interval has the same width. The interval width can be established by dividing the feature range by the number of bins if the feature has upper and lower limits; alternatively, we can take the bin boundaries at an integer number of standard deviations above and below the mean. An interesting alternative is to treat feature discretisation as a univariate clustering problem. For example, in order to generate K bins we can uniformly sample K initial bin centres and run K-means until convergence. We can alternatively use any of the other clustering methods discussed in Chapter 8: K-medoids, partitioning around medoids and hierarchical agglomerative clustering.

Switching now to *supervised discretisation* methods, we can distinguish between *top–down* or *divisive* discretisation methods on the one hand, and *bottom–up* or *agglomerative* discretisation methods on the other. Divisive methods work by progressively splitting bins, whereas agglomerative methods proceed by initially assigning each instance to its own bin and successively merging bins. In either case an important role is played by the *stopping criterion*, which decides whether a further split or merge is worthwhile. We give an example of each strategy. A natural generalisation of thresholding leads to a top–down *recursive partitioning* algorithm (Algorithm 10.1). This discretisation algorithm finds the best threshold according to some scoring function Q, and proceeds to recursively split the left and right bins. One scoring function that is often used is information gain.

Example 10.6 (Recursive partitioning using information gain). Consider the following feature values, which are ordered on increasing value for convenience.

Instance	*Value*	*Class*
e_1	−5.0	⊖
e_2	−3.1	⊕
e_3	−2.7	⊖
e_4	0.0	⊖
e_5	7.0	⊖
e_6	7.1	⊕
e_7	8.5	⊕
e_8	9.0	⊖
e_9	9.0	⊕
e_{10}	13.7	⊖
e_{11}	15.1	⊖
e_{12}	20.1	⊖

This feature gives rise to the following ranking: ⊖⊕⊖⊖⊖⊕⊕[⊖⊕]⊖⊖⊖, where the square brackets indicate a tie between instances e_8 and e_9. The corresponding

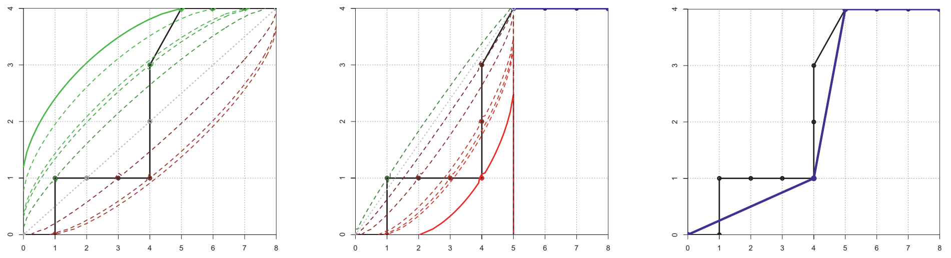


Figure 10.4. (left) A coverage curve visualising the ranking of four positive and eight negative examples by a feature to be discretised. The curved lines are information gain isometrics through possible split points; the solid isometric indicates the best split [4+,5−][0+,3−] according to information gain. **(middle)** Recursive partitioning proceeds to split the segment [4+,5−] into [1+,4−][3+,1−]. **(right)** If we stop here, the blue curve visualises the discretised (but still ordinal) feature.

coverage curve is depicted in Figure 10.4. Tracing information gain isometrics through each possible split, we see that the best split is ⊖⊕⊖⊖⊖⊕⊕[⊖⊕]|⊖⊖⊖. Repeating the process once more gives the discretisation ⊖⊕⊖⊖⊖|⊕⊕[⊖⊕]|⊖⊖⊖.

Clearly, we can stop the recursive partitioning algorithm when the empirical probabilities are the same across the ranking; this has pure bins and bins with a constant feature value as special cases. With this stopping criterion, the algorithm will successfully identify all straight line segments in the ranking. In fact, it is not hard to see that this holds true even if we change the scoring function – the split points may be found in a different order, but the end result will be the same. In practice more aggressive stopping criteria are used, which does mean that the end result depends on the

Algorithm 10.1: RecPart(S, f, Q) – supervised discretisation by means of recursive partitioning.

```
Input  : set of labelled instances $S$ ranked on feature values $f(x)$; scoring
         function $Q$.
Output : sequence of thresholds $t_1, \dots, t_{k-1}$.
1 if stopping criterion applies then return $\varnothing$;
2 Split $S$ into $S_l$ and $S_r$ using threshold $t$ that optimises $Q$ ;
3 $T_l$ = RecPart($S_l, f, Q$);
4 $T_r$ = RecPart($S_r, f, Q$);
5 return $T_l \cup \{t\} \cup T_r$;
```

Algorithm 10.2: AggloMerge(S, f, Q) – supervised discretisation by means of agglomerative merging.

```
Input  : set of labelled instances $S$ ranked on feature values $f(x)$; scoring
         function $Q$.
Output : sequence of thresholds.
1 initialise bins to data points with the same scores;
2 merge consecutive pure bins ;                    // optional optimisation
3 repeat
4     evaluate $Q$ on consecutive bin pairs;
5     merge the pairs with best $Q$ (unless they invoke the stopping criterion);
6 until no further merges are possible;
7 return thresholds between bins;
```

scoring function. For example, in Figure 10.4 we see that the split ⊖|[⊕⊖]⊕⊕⊖⊖⊖⊕⊕ has the second-highest information gain but ends up not being chosen at all, while with a different scoring function it might have been chosen in the first round. One of the most popular stopping criteria applies a minimum description length argument to decide whether a given bin should be split further.

It should be noted that the data set in Example 10.6 is probably so small that the stopping criterion will kick in straight away and recursive partitioning will be unable to go beyond a single bin. More generally, this kind of discretisation tends to be fairly conservative. For example, on the Euro data in Figure 10.3 (left) recursive partitioning produces two bins, selecting 20 Euro countries and 53 non-Euro countries (the red point in between the mean and median splits). On the American countries data in Figure 10.3 (right) we again obtain two bins, corresponding to the third red point from the right.

An algorithm for bottom–up *agglomerative merging* is given in Algorithm 10.2. Again the algorithm can take various choices for the scoring function and the stopping criterion: a popular choice is to use the χ^2 *statistic* for both.

Example 10.7 (Agglomerative merging using χ^2). We continue Example 10.6. Algorithm 10.2 initialises the bins to ⊖|⊕|⊖⊖⊖|⊕⊕|[⊖⊕]|⊖⊖⊖. We illustrate the calculation of the χ^2 statistic for the last two bins. We construct the following contingency table:

	Left bin	Right bin	
⊕	**1**	**0**	1
⊖	**1**	**3**	4
	2	3	5

At the basis of the χ^2 statistic lies a comparison of these observed frequencies with expected frequencies obtained from the row and column marginals. For example, the marginals say that the top row contains 20% of the total mass and the left column 40%; so if rows and columns were statistically independent we would expect 8% of the mass – or 0.4 of the five instances – in the top-left cell. Following a clockwise direction, the expected frequencies for the other cells are 0.6, 2.4 and 1.6. If the observed frequencies are close to the expected ones, this suggests that these two bins are candidates for merging since the split appears to have no bearing on the class distribution.

The χ^2 statistic sums the squared differences between the observed and expected frequencies, each term normalised by the expected frequency:

$$\chi^2 = \frac{(1-0.4)^2}{0.4} + \frac{(0-0.6)^2}{0.6} + \frac{(3-2.4)^2}{2.4} + \frac{(1-1.6)^2}{1.6} = 1.88$$

Going left-to-right through the other pairs of consecutive bins, the χ^2 values are 2, 4, 5 and 1.33 (there's an easy way to calculate the χ^2 value for two pure bins, which I'll leave you to discover). This tells us that the fourth and fifth bin are first to be merged, leading to ⊖|⊕|⊖⊖⊖|⊕⊕[⊖⊕]|⊖⊖⊖. We then recompute the χ^2 values (in fact, only those involving the newly merged bin need to be re-computed), yielding 2, 4, 3.94 and 3.94. We now merge the first two bins, giving the partition ⊖⊕|⊖⊖⊖|⊕⊕[⊖⊕]|⊖⊖⊖. This changes the first χ^2 value to 1.88, so we again merge the first two bins, arriving at ⊖⊕⊖⊖⊖|⊕⊕[⊖⊕]|⊖⊖⊖ (the same three bins as in Example 10.6).

In agglomerative discretisation the stopping criterion usually takes the form of a simple threshold on the scoring function. In the case of the χ^2 statistic, the threshold can be derived from the p-value associated with the χ^2 distribution, which is the probability of observing a χ^2 value above the threshold if the two variables are actually independent. For two classes (i.e., one degree of freedom) and a p-value of 0.10 the χ^2 threshold is 2.71, which in our example means that we stop at the above three bins. For a lower p-value of 0.05 the χ^2 threshold is 3.84, which means that we eventually merge all the bins.

Notice that both top–down and bottom–up supervised discretisation bear some resemblance to algorithms we have seen previously: recursive partitioning shares the divide-and-conquer nature of the ☞*decision tree* training algorithm (Algorithm 5.1 on p.132), and agglomerative discretisation by merging consecutive bins is related to ☞*hierarchical agglomerative clustering* (Algorithm 8.4 on p.255). It is also worth mentioning that, although our examples were predominantly drawn from binary classification, most methods can handle more than two classes without complication.

Normalisation and calibration

Thresholding and discretisation are feature transformations that remove the scale of a quantitative feature. We now turn our attention to adapting the scale of a quantitative feature, or adding a scale to an ordinal or categorical feature. If this is done in an unsupervised fashion it is usually called normalisation, whereas calibration refers to supervised approaches taking in the (usually binary) class labels. *Feature normalisation* is often required to neutralise the effect of different quantitative features being measured on different scales. If the features are approximately normally distributed, we can convert them into ☞*z-scores* (Background 9.1 on p.267) by centring on the mean and dividing by the standard deviation. In certain cases it is mathematically more convenient to divide by the variance instead, as we have seen in Section 7.1. If we don't want to assume normality we can centre on the median and divide by the interquartile range.

Sometimes feature normalisation is understood in the stricter sense of expressing the feature on a $[0,1]$ scale. This can be achieved in various ways. If we know the feature's highest and lowest values h and l, then we can simply apply the linear scaling $f \mapsto (f-l)/(h-l)$. We sometimes have to guess the value of h or l, and truncate any value outside $[l,h]$. For example, if the feature measures age in years, we may take $l = 0$ and $h = 100$, and truncate any $f > h$ to 1. If we can assume a particular distribution for the feature, then we can work out a transformation such that almost all feature values fall in a certain range. For instance, we know that more than 99% of the probability mass of a normal distribution falls within $\pm 3\sigma$ of the mean, where σ is the standard deviation, so the linear scaling $f \mapsto (f-\mu)/6\sigma + 1/2$ virtually removes the need for truncation.

Feature calibration is understood as a supervised feature transformation adding a meaningful scale carrying class information to arbitrary features. This has a number of important advantages. For instance, it allows models that require scale, such as linear classifiers, to handle categorical and ordinal features. It also allows the learning algorithm to choose whether to treat a feature as categorical, ordinal or quantitative. We will assume a binary classification context, and so a natural choice for the calibrated feature's scale is the posterior probability of the positive class, conditioned on the feature's value. This has the additional advantage that models that are based on such probabilities, such as naive Bayes, do not require any additional training once the features are calibrated, as we shall see. The problem of feature calibration can thus be stated as follows: given a feature $F : \mathscr{X} \to \mathscr{F}$, construct a calibrated feature $F^c : \mathscr{X} \to [0,1]$ such that $F^c(x)$ estimates the probability $F^c(x) = P(\oplus|v)$, where $v = F(x)$ is the value of the original feature for x.

For categorical features this is as straightforward as collecting relative frequencies from a training set.

Example 10.8 (Calibration of categorical features). Suppose we want to predict whether or not someone has diabetes from categorical features including whether the person is obese or not, whether he or she smokes, and so on. We collect some statistics which tell us that 1 in every 18 obese persons has diabetes while among non-obese people this is 1 in 55 (data obtained from `www.wolframalpha.com` with the query 'diabetes'). If $F(x) = 1$ for person x who is obese and $F(y) = 0$ for person y who isn't, then the calibrated feature values are $F^c(x) = 1/18 = 0.055$ and $F^c(y) = 1/55 = 0.018$.

In fact, it would be better to compensate for the non-uniform class distribution, in order to avoid over-emphasising the class prior, which is better taken into account in the decision rule. This can be achieved as follows. If m of n obese people have diabetes, then this corresponds to a posterior odds of $m/(n-m)$ or a likelihood ratio of $m/c(n-m)$, where c is the prior odds of having diabetes (since posterior odds is likelihood ratio times prior odds). Working with the likelihood ratio is equivalent to assuming a uniform class distribution. Converting the likelihood ratio into a probability gives

$$F^c(x) = \frac{\frac{m}{c(n-m)}}{\frac{m}{c(n-m)} + 1} = \frac{m}{m + c(n-m)}$$

In our example, if the prior odds of having diabetes is $c = 1/48$, then $F^c(x) = 1/(1 + 17/48) = 48/(48 + 17) = 0.74$. The extent to which this probability is more than 1/2 quantifies the extent to which obese people are more likely than average to have diabetes. For non-obese people the probability is $1/(1 + 54/48) = 48/(48 + 54) = 0.47$, so they are slightly less likely than average to have diabetes. Keep in mind also that it is usually a good idea to smooth these probability estimates by means of the Laplace correction, which adds 1 to m and 2 to n. This leads to the final expression for calibrating a categorical feature:

$$F^c(x) = \frac{m+1}{m+1+c(n-m+1)}$$

Ordinal and quantitative features can be discretised and then calibrated as categorical features. In the remainder of this section we look at calibration methods that maintain the ordering of the feature. For example, suppose we want to use body weight as an indicator for diabetes. A calibrated weight feature attaches a probability to every weight, such that these probabilities are non-decreasing with weight. This is related to our discussion of ☞*calibrating classifier scores* in Section 7.4, as those calibrated probabilities should likewise takes the ranking of the classifier's predictions into account. In fact, the two approaches to classifier calibration – by employing the logistic function and by constructing the ROC convex hull – are directly applicable to feature calibration, since a quantitative feature can simply be treated as a univariate scoring classifier.

We briefly reiterate the main points of *logistic calibration*, but with a slight change in notation. Let $F : \mathscr{X} \to \mathbb{R}$ be a quantitative feature with class means $\mu^{\oplus}$ and $\mu^{\ominus}$ and variance σ^2. Assuming the feature is normally distributed within each class with the same variance, we can express the likelihood ratio of a feature value v as

$$LR(v) = \frac{P(v|\oplus)}{P(v|\ominus)} = \exp\left(\frac{-(v-\mu^{\oplus})^2 + (v-\mu^{\ominus})^2}{2\sigma^2}\right)$$
$$= \exp\left(\frac{\mu^{\oplus}-\mu^{\ominus}}{\sigma}\,\frac{v-(\mu^{\oplus}+\mu^{\ominus})/2}{\sigma}\right) = \exp\left(d'z\right)$$

where $d' = (\mu^{\oplus} - \mu^{\ominus})/\sigma$ is the difference between the means in proportion to the standard deviation, which is known as *d-prime* in signal detection theory; and $z = (v-\mu)/\sigma$ is the z-score associated with v (notice we take the mean as $\mu = (\mu^{\oplus} + \mu^{\ominus})/2$ to simulate an equal class distribution). Again we work directly with the likelihood ratio to neutralise the effect of a non-uniform class distribution, and we obtain the calibrated feature value as

$$F^{c}(x) = \frac{LR(F(x))}{1+LR(F(x))} = \frac{\exp\left(d'z(x)\right)}{1+\exp\left(d'z(x)\right)}$$

You may recognise the logistic function we discussed in Chapter 7 (see Figure 7.11 on p.222).

In essence, logistic feature calibration performs the following steps.

1. Estimate the class means $\mu^{\oplus}$ and $\mu^{\ominus}$ and the standard deviation σ.

2. Transform $F(x)$ into z-scores $z(x)$, making sure to use $\mu = (\mu^{\oplus} + \mu^{\ominus}))/2$ as the feature mean.

3. Rescale the z-scores to $F^{d}(x) = d'z(x)$ with $d' = (\mu^{\oplus} - \mu^{\ominus})/\sigma$.

4. Apply a sigmoidal transformation to $F^{d}(x)$ to give calibrated probabilities $F^{c}(x) = \frac{\exp(F^{d}(x))}{1+\exp(F^{d}(x))}$.

Sometimes it is preferred to work directly with $F^{d}(x)$, as it is expressed on a scale linearly related to the original feature's scale, and the Gaussian assumption implies that

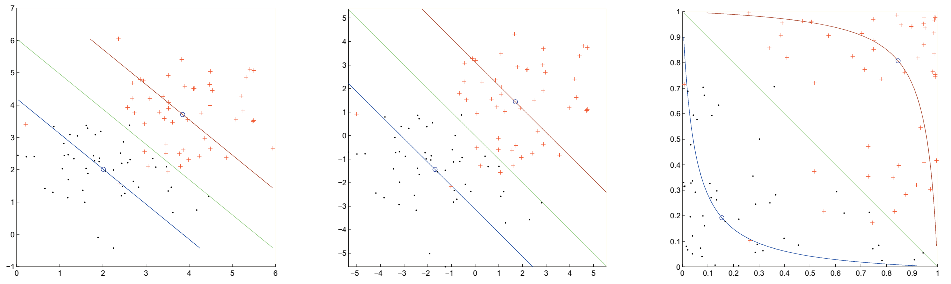


Figure 10.5. (left) Two-class Gaussian data. The middle line is the decision boundary learned by the basic linear classifier; the other two are parallel lines through the class means. **(middle)** Logistic calibration to log-odds space is a linear transformation; assuming unit standard deviations, the basic linear classifier is now the fixed line $F_1^{\rm d}(x) + F_2^{\rm d}(x) = 0$. **(right)** Logistic calibration to probability space is a non-linear transformation that pushes data away from the decision boundary.

we expect that scale to be additive. For example, distance-based models expect additive features in order to calculate Euclidean distance. In contrast, the scale of $F^{\rm c}$ is multiplicative. Notice that the two are interdefinable as $F^{\rm d}(x) = \ln \frac{F^{\rm c}(x)}{1-F^{\rm c}(x)} = \ln F^{\rm c}(x) - \ln(1 - F^{\rm c}(x))$. I will call the feature space spanned by $F^{\rm d}$ *log-odds space*, since $\exp\left(F^{\rm d}(x)\right) = LR(x)$ and the likelihood ratio is equal to the odds if we're assuming a uniform class prior. Calibrated features $F^{\rm c}$ live in *probability space*.

Example 10.9 (Logistic calibration of two features). Logistic feature calibration is illustrated in Figure 10.5. I generated two sets of 50 points by sampling bivariate Gaussians with identity covariance matrix, centred at (2,2) and (4,4). I then constructed the basic linear classifier as well as two parallel decision boundaries through the class means. Tracing these three lines in calibrated space will help us understand feature calibration.

In the middle figure we see the transformed data in log-odds space, which is clearly a linear rescaling of the axes. The basic linear classifier is now the line $F_1^{\rm d}(x) + F_2^{\rm d}(x) = 0$ through the origin. In other words, for this simple classifier feature calibration has removed the need for further training: instead of fitting a decision boundary to the data, we have fitted the data to a fixed decision boundary! (I should add that I cheated very slightly here, as I fixed $\sigma = 1$ in the calibration process – had I estimated each feature's standard deviation from the data, the decision boundary would most likely have had a slightly different slope.)

On the right we see the transformed data in probability space, which clearly has a non-linear relationship with the other two feature spaces. The basic linear classifier is still linear in this space, but actually this is no longer true for more than two features. To see this, note that $F_1^{\mathrm{c}}(x) + F_2^{\mathrm{c}}(x) = 1$ can be rewritten as

$$\frac{\exp\left(F_1^{\mathrm{d}}(x)\right)}{1+\exp\left(F_1^{\mathrm{d}}(x)\right)} + \frac{\exp\left(F_2^{\mathrm{d}}(x)\right)}{1+\exp\left(F_2^{\mathrm{d}}(x)\right)} = 1$$

which can be simplified to $\exp\left(F_1^{\mathrm{d}}(x)\right)\exp\left(F_2^{\mathrm{d}}(x)\right) = 1$ and hence to $F_1^{\mathrm{d}}(x) + F_2^{\mathrm{d}}(x) = 0$. However, if we add a third feature not all cross-terms cancel and we obtain a non-linear boundary .

The log-odds representation does hold an interest in another respect. An arbitrary linear decision boundary in log-odds space is represented by $\sum_i w_i F_i^{\mathrm{d}}(x) = t$. Taking natural logarithms this can be rewritten as

$$\exp\left(\sum_i w_i F_i^{\mathrm{d}}(x)\right) = \prod_i \exp\left(w_i F_i^{\mathrm{d}}(x)\right) = \prod_i \left(\exp\left(F_i^{\mathrm{d}}(x)\right)\right)^{w_i} = \prod_i LR_i(x)^{w_i} = \exp(t) = t'$$

This exposes a connection with the naive Bayes models discussed in Section 9.2, whose decision boundaries are also defined as products of likelihood ratios for individual features. The basic naive Bayes model has $w_i = 1$ for all i and $t' = 1$, which means that *fitting data to a fixed linear decision boundary in log-odds space by means of feature calibration can be understood as training a naive Bayes model*. Changing the slope of the decision boundary corresponds to introducing non-unit feature weights, which is similar to the way feature weights arose in the multinomial naive Bayes model.

It is instructive to investigate the distribution of the calibrated feature a bit more (I will omit the technical details). Assuming the uncalibrated distributions were two Gaussian bumps, what do the calibrated distributions look like? We have already seen that calibrated data points are pulled away from the decision boundary, so we would expect the peaks of the calibrated distributions to be closer to their extreme values. How much closer depends solely on d'; Figure 10.6 depicts the calibrated distributions for various values of d'.

We move on to *isotonic calibration*, a method that requires order but ignores scale and can be applied to both ordinal and quantitative features. We essentially use the feature as a univariate ranker, and construct its ROC curve and convex hull to obtain a piecewise-constant calibration map. Suppose we have an ROC curve, and suppose the i-th segment of the curve involves n_i training examples, out of which m_i are positives. The corresponding ROC segment has slope $l_i = m_i/(c(n_i - m_i))$, where c is the prior odds. Suppose first the ROC curve is convex: i.e., $i < j$ implies $l_i \geq l_j$. In that case, we can use the same formula as for categorical features to obtain a calibrated feature

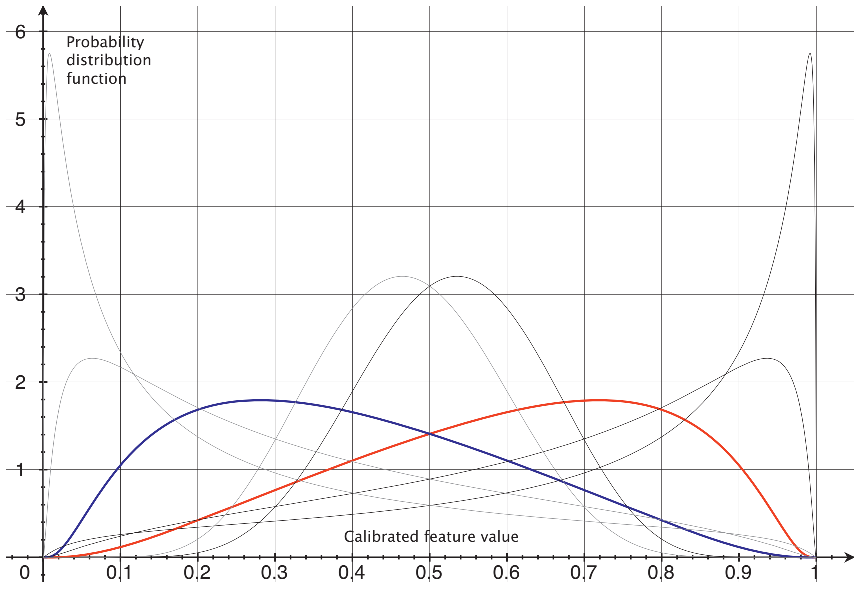


Figure 10.6. Per-class distributions of a logistically calibrated feature for different values of d', the distance between the uncalibrated class means in proportion to the feature's standard deviation. The red and blue curves depict the distributions for the positive and negative class for a feature whose means are one standard deviation apart ($d' = 1$). The other curves are for $d' \in \{0.5, 1.4, 1.8\}$.

value:

$$v_i^c = \frac{m_i + 1}{m_i + 1 + c(n_i - m_i + 1)} \tag{10.1}$$

As before, this achieves both probability smoothing through Laplace correction and compensation for non-uniform class distributions. If the ROC curve is not convex, there exist $i < j$ such that $l_i < l_j$. Assuming we want to maintain the original feature ordering, we first construct the convex hull of the ROC curve. The effect of this is that we join adjacent segments in the ROC curve that are part of a concavity, until no concavities remain. We recalculate the segments and assign calibrated feature values as in Equation 10.1.

Example 10.10 (Isotonic feature calibration). The following table gives sample values of a weight feature in relation to a diabetes classification problem. Figure 10.7 shows the ROC curve and convex hull of the feature and the calibration map obtained by isotonic calibration.

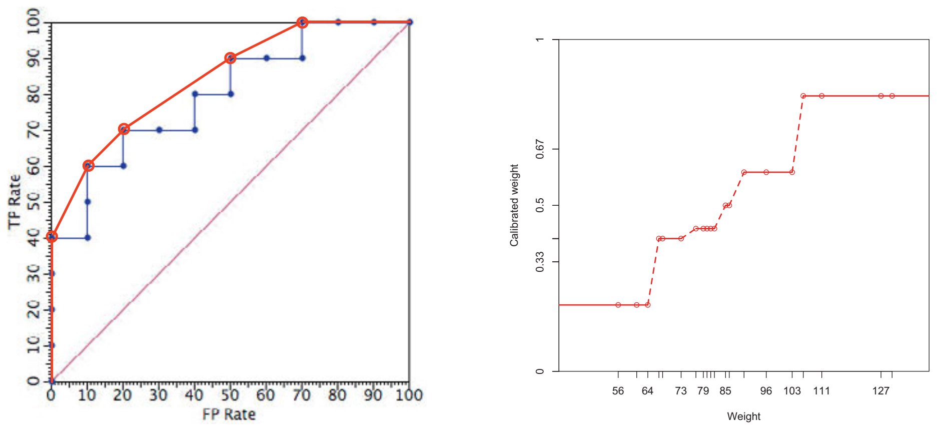


Figure 10.7. (left) ROC curve and convex hull of an uncalibrated feature. Calibrated feature values are obtained from the proportion of positives in each segment of the ROC convex hull. **(right)** The corresponding piecewise-constant calibration map, which maps the uncalibrated feature values on the x-axis to the calibrated feature values on the y-axis.

Weight	*Diabetes?*	*Calibrated weight*	*Weight*	*Diabetes?*	*Calibrated weight*
130	⊕	0.83	81	⊖	0.43
127	⊕	0.83	80	⊕	0.43
111	⊕	0.83	79	⊖	0.43
106	⊕	0.83	77	⊕	0.43
103	⊖	0.60	73	⊖	0.40
96	⊕	0.60	68	⊖	0.40
90	⊕	0.60	67	⊕	0.40
86	⊖	0.50	64	⊖	0.20
85	⊕	0.50	61	⊖	0.20
82	⊖	0.43	56	⊖	0.20

For example, a weight of 80 kilograms is calibrated to 0.43, meaning that three out of seven people in that weight interval have diabetes (after Laplace correction).

Example 10.11 gives a bivariate illustration. As is clearly visible, for quantitative features the process amounts to supervised discretisation of the feature values, which means that many points are mapped to the same point in calibrated space. This is different from logistic calibration, which is invertible.

Example 10.11 (Isotonic calibration of two features). Figure 10.8 shows the result of isotonic calibration on the same data as in Example 10.9, both in log-odds

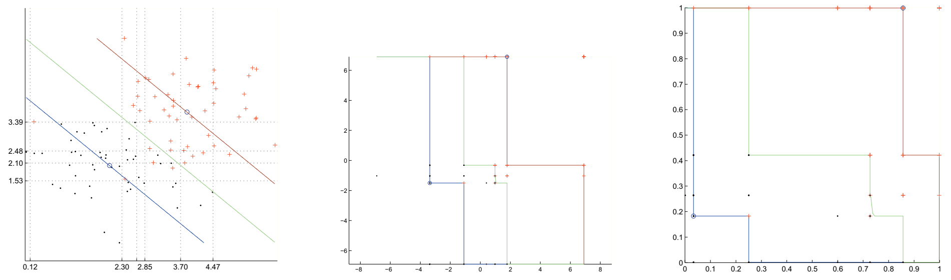


Figure 10.8. **(left)** The data from Figure 10.5, with grid lines indicating the discretisation obtained by isotonic feature calibration. **(middle)** Isotonically calibrated data in log-odds space. **(right)** Isotonically calibrated data in probability space.

space and in probability space. Because of the discrete nature of isotonic calibration, even the transformation to log-odds space is no longer linear: the basic linear classifier becomes a series of axis-parallel line segments. This is also true in the opposite direction: if we imagine a linear decision boundary in log-odds space or in probability space, this maps to a decision boundary following the dotted lines in the original feature space. Effectively, isotonic feature calibration has changed the linear grading model into a grouping model.

In summary, isotonic feature calibration performs the following steps.

1. Sort the training instances on feature value and construct the ROC curve. The sort order is chosen such that the ROC curve has AUC $\geq 1/2$.
2. Construct the convex hull of this curve, and count the number of positives m_i and the total number of instances n_i in each segment of the convex hull.
3. Discretise the feature according to the convex hull segments, and associate a calibrated feature value $v_i^{\text{c}} = \frac{m_i+1}{m_i+1+c(n_i-m_i+1)}$ with each segment.
4. If an additive scale is required, use $v_i^{\text{d}} = \ln \frac{v_i^{\text{c}}}{1-v_i^{\text{c}}} = \ln v_i^{\text{c}} - \ln(1-v_i^{\text{c}})$.

Incomplete features

At the end of this section on feature transformations we briefly consider what to do if we don't know a feature's value for some of the instances. We encountered this situation in Example 1.2 on p.26, where we discussed how to classify an e-mail if we didn't know whether it contained one of the vocabulary words or not. Probabilistic models handle this rather gracefully by taking a weighted average over all possible values of the feature:

$$P(Y|X) = \sum_z P(Y, Z = z|X) = \sum_z P(Y|X, Z = z)P(Z = z)$$

Here, Y is the target variable as usual, X stands for the features that are observed for the instance to be classified, while Z are the features that are unobserved at classification time. The distribution $P(Z)$ can be obtained from the trained model, at least for a generative model – if our model is discriminative we need to estimate $P(Z)$ separately.

Missing feature values at training time are trickier to handle. First of all, the very fact that a feature value is missing may be correlated with the target variable. For example, the range of medical tests carried out on a patient is likely to depend on their medical history. For such features it may be best to have a designated 'missing' value so that, for instance, a tree model can split on it. However, this would not work for, say, a linear model. In such cases we can complete the feature by 'filling in' the missing values, a process known as *imputation*. For instance, in a classification problem we can calculate the per-class means, medians or modes over the observed values of the feature and use this to impute the missing values. A somewhat more sophisticated method takes feature correlation into account by building a predictive model for each incomplete feature and uses that model to 'predict' the missing value. It is also possible to invoke the ☞*Expectation-Maximisation* algorithm (Section 9.4), which goes roughly as follows: assuming a multivariate model over all features, use the observed values for maximum-likelihood estimation of the model parameters, then derive expectations for the unobserved feature values and iterate.

10.3 Feature construction and selection

The previous section on feature transformation makes it clear that there is a lot of scope in machine learning to play around with the original features given in the data. We can take this one step further by constructing new features from several original features. A simple example of this can be used to improve the ☞*naive Bayes* classifier discussed in Section 9.2. Remember that in text classification applications we have a feature for every word in the vocabulary, which disregards not only the order of words but also their adjacency. This means that sentences such as 'they write about machine learning' and 'they are learning to write about a machine' will be virtually indistinguishable, even though the former is about machine learning and the latter is not. It may therefore sometimes be necessary to include phrases consisting of multiple words in the dictionary and treat them as single features. In the information retrieval literature, a multi-word phrase is referred to as an *n-gram* (unigram, bigram, trigram and so on).

Taking this idea one step further, we can construct a new feature from two Boolean or categorical features by forming their Cartesian product. For example, if we have one feature Shape with values Circle, Triangle and Square, and another feature Colour with values Red, Green and Blue, then their Cartesian product would be the feature (Shape, Colour) with values (Circle, Red), (Circle, Green), (Circle, Blue), (Triangle, Red), and so on. The effect that this would have depends on the model being trained. Constructing Cartesian product features for a naive Bayes classifier means that the two original features are no longer treated as independent, and so this reduces the strong bias that naive Bayes models have. This is not the case for tree models, which can already distinguish between all possible pairs of feature values. On the other hand, a newly introduced Cartesian product feature may incur a high information gain, so it can possibly affect the model learned.

There are many other ways of combining features. For instance, we can take arithmetic or polynomial combinations of quantitative features (we saw examples of this in the use of a ☞*kernel* in Example 1.9 on p.43 and Section 7.5). One attractive possibility is to first apply concept learning or subgroup discovery, and then use these concepts or subgroups as new Boolean features. For instance, in the dolphin domain we could first learn subgroups such as $\text{Length} = [3,5] \wedge \text{Gills} = \text{no}$ and use these as Boolean features in a subsequent tree model. Notice that this expands the expressive power of tree models through the use of negation: e.g., $(\text{Length} = [3,5] \wedge \text{Gills} = \text{no}) = \text{false}$ is equivalent to the disjunction $\text{Length} \neq [3,5] \vee \text{Gills} = \text{yes}$, which is not directly expressible by a feature tree.

Once we have constructed new features it is often a good idea to select a suitable subset of them prior to learning. Not only will this speed up learning as fewer candidate features need to be considered, it also helps to guard against overfitting. There are two main approaches to feature selection. The *filter* approach scores features on a particular metric and the top-scoring features are selected. Many of the metrics we have seen so far can be used for feature scoring, including information gain, the χ^2 statistic, the correlation coefficient, to name just a few. An interesting variation is provided by the *Relief* feature selection method, which repeatedly samples a random instance x and finds its nearest hit h (instance of the same class) as well as its nearest miss m (instance of opposite class). The i-th feature's score is then decreased by $\text{Dis}(x_i, h_i)^2$ and increased by $\text{Dis}(x_i, m_i)^2$, where Dis is some distance measure (e.g., Euclidean distance for quantitative features, Hamming distance for categorical features). The intuition is that we want to move closer to the nearest hit while differentiating from the nearest miss.

One drawback of a simple filter approach is that no account is taken of redundancy between features. Imagine, for the sake of the argument, duplicating a promising feature in the data set: both copies score equally high and will be selected, whereas the second one provides no added value in the context of the first one. Secondly, feature filters do not detect dependencies between features as they are solely based on marginal distributions. For example, consider two Boolean features such that half the positives have the value true for both features and the other half have the value false for both, whereas all negatives have opposite values (again distributed half-half over the two possibilities). It follows that each feature in isolation has zero information gain and hence is unlikely to be selected by a feature filter, despite their combination being a perfect classifier. One could say that feature filters are good at picking out possible root features for a decision tree, but not necessarily good at selecting features that are useful further down the tree.

To detect features that are useful in the context of other features we need to evaluate sets of features; this usually goes under the name of *wrapper* approaches. The idea is that feature selection is 'wrapped' in a search procedure that usually involves training and evaluating a model with a candidate set of features. *Forward selection* methods start with an empty set of features and add features to the set one at a time, as long as they improve the performance of the model. *Backward elimination* starts with the full set of features and aims at improving performance by removing features one at a time. Since there are an exponential number of subsets of features it is usually not feasible to search all possible subsets, and most approaches apply a 'greedy' search algorithm that never reconsiders the choices it makes.

Matrix transformations and decompositions

We can also view feature construction and selection from a geometric perspective, assuming quantitative features. To this end we represent our data set as a matrix $\mathbf{X}$ with n data points in rows and d features in columns, which we want to transform into a new matrix $\mathbf{W}$ with n rows and r columns by means of matrix operations. To simplify matters a bit, we assume that $\mathbf{X}$ is zero-centred and that $\mathbf{W} = \mathbf{XT}$ for some d-by-r transformation matrix $\mathbf{T}$. For example, feature scaling corresponds to $\mathbf{T}$ being a d-by-d diagonal matrix; this can be combined with feature selection by removing some of $\mathbf{T}$'s columns. A rotation is achieved by $\mathbf{T}$ being orthogonal, i.e., $\mathbf{TT}^{\mathrm{T}} = \mathbf{I}$. Clearly, several such transformations can be combined (see also Background 1.2 on p.24).

One of the best-known algebraic feature construction methods is *principal component analysis* (*PCA*). Principal components are new features constructed as linear combinations of the given features. The first principal component is given by the direction of maximum variance in the data; the second principal component is the direction of maximum variance orthogonal to the first component, and so on. PCA can be explained in a number of different ways: here, we will derive it by means of the *singular value decomposition* (*SVD*). Any n-by-d matrix can be uniquely written as a product of three matrices with special properties:

$$\mathbf{X} = \mathbf{U\Sigma V}^{\mathrm{T}} \tag{10.2}$$

Here, $\mathbf{U}$ is an n-by-r matrix, $\mathbf{\Sigma}$ is an r-by-r matrix and $\mathbf{V}$ is an d-by-r matrix (for the moment we will assume $r = d < n$). Furthermore, $\mathbf{U}$ and $\mathbf{V}$ are orthogonal (hence rotations) and $\mathbf{\Sigma}$ is diagonal (hence a scaling). The columns of $\mathbf{U}$ and $\mathbf{V}$ are known as the left and right singular vectors, respectively; and the values on the diagonal of $\mathbf{\Sigma}$ are the corresponding singular values. It is customary to order the columns of $\mathbf{V}$ and $\mathbf{U}$ so that the singular values are decreasing from top-left to bottom-right.

Now, consider the n-by-r matrix $\mathbf{W} = \mathbf{U\Sigma}$, and notice that $\mathbf{XV} = \mathbf{U\Sigma V}^{\mathrm{T}}\mathbf{V} = \mathbf{U\Sigma} = \mathbf{W}$ by the orthogonality of $\mathbf{V}$. In other words, we can construct $\mathbf{W}$ from $\mathbf{X}$ by means of the transformation $\mathbf{V}$: this is the reformulation of $\mathbf{X}$ in terms of its principal components. The newly constructed features are found in $\mathbf{U\Sigma}$: the first row is the first principal component, the second row is the second principal component, and so on. These principal components have a geometric interpretation as the directions in which $\mathbf{X}$ has largest, second-largest, ... variance. Assuming the data is zero-centred, these directions can be brought out by a combination of rotation and scaling, which is exactly what PCA does.

We can also use SVD to rewrite the scatter matrix in a standard form:

$$\mathbf{S} = \mathbf{X}^{\mathrm{T}}\mathbf{X} = \left(\mathbf{U\Sigma V}^{\mathrm{T}}\right)^{\mathrm{T}}\left(\mathbf{U\Sigma V}^{\mathrm{T}}\right) = \left(\mathbf{V\Sigma U}^{\mathrm{T}}\right)\left(\mathbf{U\Sigma V}^{\mathrm{T}}\right) = \mathbf{V\Sigma}^2\mathbf{V}^{\mathrm{T}}$$

This is known as the *eigendecomposition* of the matrix $\mathbf{S}$: the columns of $\mathbf{V}$ are the eigenvectors of $\mathbf{S}$, and the elements on the diagonal of $\mathbf{\Sigma}^2$ – which is itself a diagonal matrix – are the eigenvalues. The right singular vectors of the data matrix $\mathbf{X}$ are the eigenvectors of the scatter matrix $\mathbf{S} = \mathbf{X}^{\mathrm{T}}\mathbf{X}$, and the singular values of $\mathbf{X}$ are the square root of the eigenvalues of $\mathbf{S}$. We can derive a similar expression for the Gram matrix $\mathbf{G} = \mathbf{XX}^{\mathrm{T}} = \mathbf{U\Sigma}^2\mathbf{U}^{\mathrm{T}}$, from which we see that the eigenvectors of the Gram matrix are the left singular vectors of $\mathbf{X}$. This demonstrates that in order to perform principal components analysis it is sufficient to perform an eigendecomposition of the scatter or Gram matrices, rather than a full singular value decomposition.

We have seen something resembling SVD in Section 1.1, where we considered the following matrix product:

$$\begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 2 & 2 & 2 \\ 0 & 0 & 0 & 1 \\ 1 & 2 & 3 & 2 \\ 1 & 0 & 1 & 1 \\ 0 & 2 & 2 & 3 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} \times \begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

The matrix on the left expresses people's preferences for films (in columns). The right-hand side decomposes or factorises this into film genres: the first matrix quantifies people's appreciation of genres; the last matrix associates films with genres; and the middle matrix tells us the weight of each genre in determining preferences. This is not actually the decomposition computed by SVD, because the left and right matrices in the product are not orthogonal. However, one could argue that this factorisation better captures the data, because the person-by-genre and the film-by-genre matrices are Boolean and sparse, which they won't be in the SVD. The downside is that adding integer or Boolean constraints makes the decomposition problem non-convex (there are local optima) and computationally harder. Matrix decomposition with additional constraints is a very active research area.

These matrix decomposition techniques are often used for dimensionality reduction. The *rank* of an n-by-d matrix is d (assuming $d < n$ and no columns are linear combinations of other columns). The above decompositions are full-rank because $r = d$, and hence the data matrix is reconstructed exactly. A low-rank approximation of a matrix is a factorisation where r is chosen as small as possible while still sufficiently approximating the original matrix. The reconstruction error is usually measured as the sum of the squared differences of the entries in $\mathbf{X}$ and the corresponding entries in $\mathbf{U\Sigma V}^{\mathrm{T}}$. It can be shown that a truncated singular value decomposition with $r < d$ results in the lowest reconstruction error in this sense among all decompositions of rank up to r. Truncated SVD and PCA are popular ways to combine feature construction and feature selection for quantitative features.

One interesting aspect of matrix decompositions such as SVD is that they expose a previously hidden variable in the data. This can be seen as follows. Consider a decomposition or approximation $\mathbf{U\Sigma V}^{\mathrm{T}}$ with diagonal $\mathbf{\Sigma}$ but not necessarily orthogonal $\mathbf{U}$ and $\mathbf{V}$, and denote the i-the column of $\mathbf{U}$ and $\mathbf{V}$ as $\mathbf{U}_{\cdot i}$ (an n-vector) and $\mathbf{V}_{\cdot i}$ (a d-vector). Then $\mathbf{U}_{\cdot i}\sigma_i(\mathbf{V}_{\cdot i})^{\mathrm{T}}$ is an outer product that produces an n-by-d matrix with rank 1 (σ_i denotes the i-th diagonal value of $\mathbf{\Sigma}$). A rank-1 matrix is such that every column is obtained from a single basis vector multiplied by a scalar (and the same for rows). Assuming $\mathbf{U}$ and $\mathbf{V}$ have rank r these basis vectors are independent and so summing up these rank-1 matrices for all i produces the original matrix:

$$\mathbf{U\Sigma V}^{\mathrm{T}} = \sum_{i=1}^{r} \mathbf{U}_{\cdot i}\sigma_i(\mathbf{V}_{\cdot i})^{\mathrm{T}}$$

For example, the film rating matrix can be written as follows:

$$\begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 2 & 2 & 2 \\ 0 & 0 & 0 & 1 \\ 1 & 2 & 3 & 2 \\ 1 & 0 & 1 & 1 \\ 0 & 2 & 2 & 3 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 2 & 2 & 2 \\ 0 & 0 & 0 & 0 \\ 0 & 2 & 2 & 2 \\ 0 & 0 & 0 & 0 \\ 0 & 2 & 2 & 2 \end{pmatrix} + \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

The matrices on the right can be interpreted as rating models conditioned on genre.

Exposing hidden variables in the data is one of the main applications of matrix decomposition methods. For example, in information retrieval PCA is known under the name *latent semantic indexing* (*LSA*) ('latent' is synonymous with 'hidden'). Instead of film genres, LSA uncovers document topics by decomposing matrices containing word counts per document, under the assumption that the word counts per topic are independent and can thus simply be added up.[3] The other main application of matrix factorisation is *completion* of missing entries in a matrix, the idea being that if we approximate the observed entries in the matrix as closely as possible using a low-rank decomposition, this allows us to infer the missing entries.

10.4 Features: Summary and further reading

In this chapter we have given features some long-overdue attention. Features are the telescopes through which we observe the data universe and therefore an important unifying force in machine learning. Features are related to measurements in science, but there is no widespread consensus on how to formalise and categorise different measurements – I have taken inspiration from Stevens' scales of measurements (Stevens, 1946), but otherwise aimed to stay close to current practice in machine learning.

☞ The main kinds of feature distinguished in Section 10.1 are categorical, ordinal and quantitative features. The latter are expressed on a quantitative scale and admit the calculation of the widest range of statistics of tendency (mean, median, mode; see (von Hippel, 2005) for a discussion of rules of thumb regarding these), dispersion (variance and standard deviation, range, interquartile range) and shape (skewness and kurtosis). In machine learning quantitative features are often referred to as continuous features, but I think this term is inappropriate as it wrongly suggests that their defining feature is somehow an unlimited precision. It is important to realise that quantitative features do not necessarily have an additive scale: e.g., quantitative features expressing a probability are expressed on a multiplicative scale, and the use of Euclidean distance, say, would be inappropriate for non-additive features. Ordinal features have order but not scale; and categorical features (also called nominal or discrete) have neither order nor scale.

☞ Structured features are first-order logical statements that refer to parts of objects by means of local variables and use some kind of aggregation, such as existential quantification or counting, to extract a property of the main object. Constructing first-order features prior to learning is often referred to as propositionalisation;

[3]Other models are possible: e.g., in Boolean matrix decomposition the matrix product is changed to a Boolean product in which integer addition is replaced by Boolean disjunction (so that $1+1=1$), with the effect that additional topics do not provide additional explanatory power for the occurrence of a word in a document.

Kramer *et al.* (2000) and Lachiche (2010) give surveys, and an experimental comparison of different approaches is carried out by Krogel *et al.* (2003).

☞ In Section 10.2 we looked at a number of feature transformations. Discretisation and thresholding are the best-known of these, turning a quantitative feature into a categorical or a Boolean one. One of the most effective discretisation methods is the recursive partitioning algorithm using information gain to find the thresholds and a stopping criterion derived from the minimum description length principle proposed by Fayyad and Irani (1993). Other overviews and proposals are given by Boullé (2004, 2006). The agglomerative merging approach using χ^2 was proposed by Kerber (1992).

☞ We have seen that in a two-class setting, supervised discretisation can be visualised by means of coverage curves. This then naturally leads to the idea of using these coverage curves and their convex hull to calibrate rather than just discretise the features. After all, ordinal and quantitative features are univariate rankers and scoring classifiers and thus the same classifier calibration methods can be applied, in particular logistic and isotonic calibration as discussed in Section 7.4. The calibrated features live in probability space, but we might prefer to work with log-odds space instead as this is additive rather than multiplicative. Fitting data to a fixed linear decision boundary in calibrated log-odds space is closely related to training a naive Bayes model. Isotonic calibration leads to piecewise axis-parallel decision boundaries; owing to the discretising nature of isotonic calibration this can be understood as the constructing of a grouping model, even if the original model in the uncalibrated space was a grading model.

☞ Section 10.3 was devoted to feature construction and selection. Early approaches to feature construction and constructive induction were proposed by Ragavan and Rendell (1993); Donoho and Rendell (1995). The instance-based Relief feature selection method is due to Kira and Rendell (1992) and extended by Robnik-Sikonja and Kononenko (2003). The distinction between filter approaches to feature selection – which evaluate features on their individual merits – and wrapper approaches, which evaluate sets of features, is originally due to Kohavi and John (1997). Hall (1999) proposes a filter approach called correlation-based feature selection that aims at combining the best of both worlds. Guyon and Elisseeff (2003) give an excellent introduction to feature selection.

☞ Finally, we looked at feature construction and selection from a linear algebra perspective. Matrix decomposition and factorisation is an actively researched technique that was instrumental in winning a recent film recommendation challenge worth \$1 million (Koren *et al.*, 2009). Decomposition techniques employing additional constraints include non-negative matrix decomposition (Lee *et al.*, 1999).