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 $f_i : \mathcal{X} \to \mathcal{F}_i$ from the instance space $\mathcal{X}$ to the feature domain $\mathcal{F}_i$. 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.

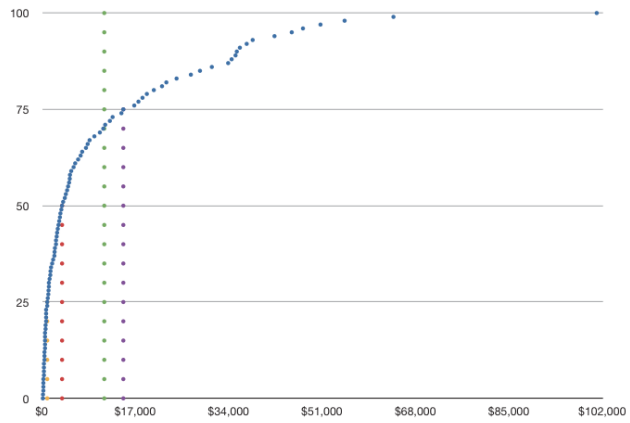

Figure 10.1. Percentile plot of GDP per capita for 231 countries (data obtained from `www.wolframalpha.com` by means of the query 'GDP per capita').The vertical dotted lines indicate, from left to right: the first quartile ($900); the median ($3600); the mean ($11 284); andthe third quartile ($14 750).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

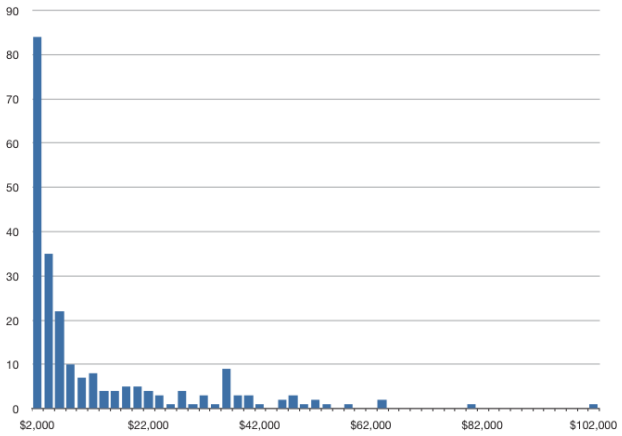

Figure 10.2. Histogram of the data from Figure 10.1, with bins of $2000 wide.

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: $\mathcal{X} = \mathcal{F}_1 \times \ldots \times \mathcal{F}_d$. 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

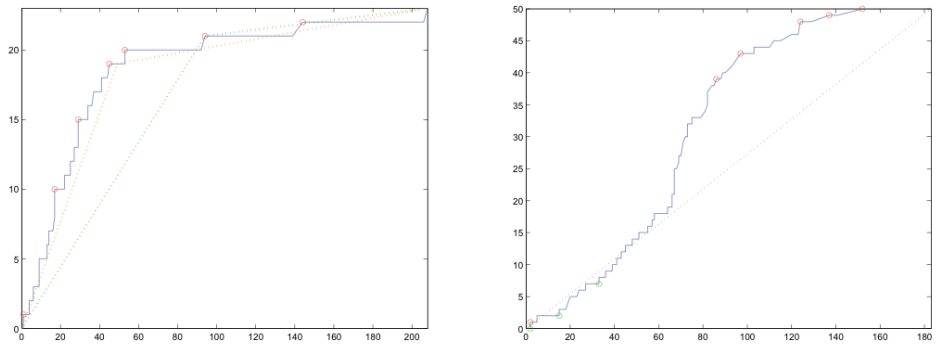

Figure 10.3. (left) Coverage curve obtained by ranking countries on decreasing GDP per capita, using 23 Euro countries as the positive class. The orange split sets the threshold equal to the mean, selecting 19 Euro countries and 49 non-Euro countries. The green split sets the threshold equal to the median, 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

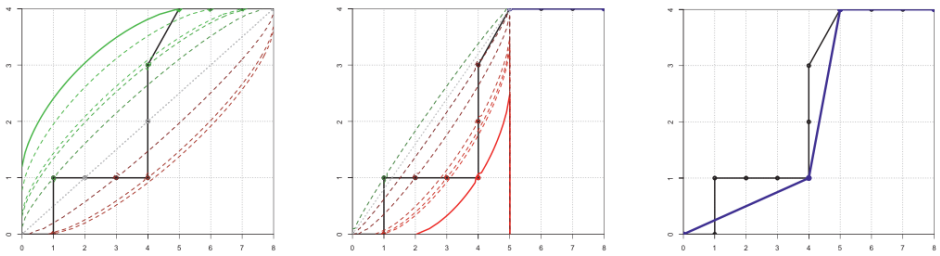

Figure 10.4. (left) A coverage curve visualising the ranking of four positive and eight negative examples by a feature to be discretised. The curved lines are information gain isometrics through possible split points; the solid isometric indicates the best split [4+,5−][0+,3−] according to information 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 $t_1, \ldots, t_{k-1}$.

```
if stopping criterion applies then return ∅;
Split S into S_l and S_r using threshold t that optimises Q;
T_l = RecPart(S_l, f, Q);
T_r = RecPart(S_r, f, Q);
return T_l ∪ {t} ∪ 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.

```
initialise bins to data points with the same scores;
merge consecutive pure bins ;                          // optional optimisation
repeat
    evaluate Q on consecutive bin pairs;
    merge the pairs with best Q (unless they invoke the stopping criterion);
until no further merges are possible;
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 $F : \mathcal{X} \to \mathbb{R}$ 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

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

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

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

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 $F^{\mathrm{d}}(x) = d'z(x)$ with $d' = (\mu^{\oplus} - \mu^{\ominus})/\sigma$.
4. Apply a sigmoidal transformation to $F^{\mathrm{d}}(x)$ to give calibrated probabilities $F^{\mathrm{c}}(x) = \frac{\exp(F^{\mathrm{d}}(x))}{1+\exp(F^{\mathrm{d}}(x))}$.

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

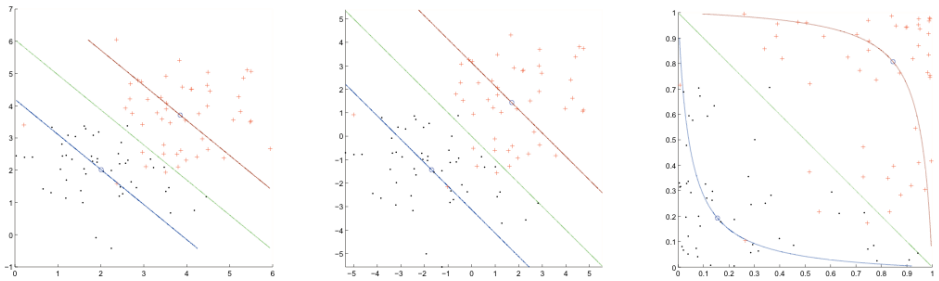

Figure 10.5. (left) Two-class Gaussian data. The middle line is the decision boundary learned by the basic linear classifier; the other two are parallel lines through the class means. **(middle)** Logistic calibration to log-odds space is a linear transformation; assuming unit standard deviations, the basic linear classifier is now the fixed line $F_1^{\mathrm{d}}(x) + F_2^{\mathrm{d}}(x) = 0$. **(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^{c} is multiplicative. Notice that the two are interdefinable as $F^{\mathrm{d}}(x) = \ln \frac{F^{\mathrm{c}}(x)}{1-F^{\mathrm{c}}(x)} = \ln F^{\mathrm{c}}(x) - \ln(1 - F^{\mathrm{c}}(x))$. I will call the feature space spanned by F^{d} *log-odds space*, since $\exp\left(F^{\mathrm{d}}(x)\right) = LR(x)$ and the likelihood ratio is equal to the odds if we're assuming a uniform class prior. Calibrated features F^{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 $F_1^{\mathrm{d}}(x) + F_2^{\mathrm{d}}(x) = 0$ 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 $F_1^c(x) + F_2^c(x) = 1$ can be rewritten as

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

which can be simplified to $\exp\left(F_1^d(x)\right)\exp\left(F_2^d(x)\right) = 1$ and hence to $F_1^d(x) + F_2^d(x) = 0$. 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 $\sum_i w_i F_i^d(x) = t$. Taking natural logarithms this can be rewritten as

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

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

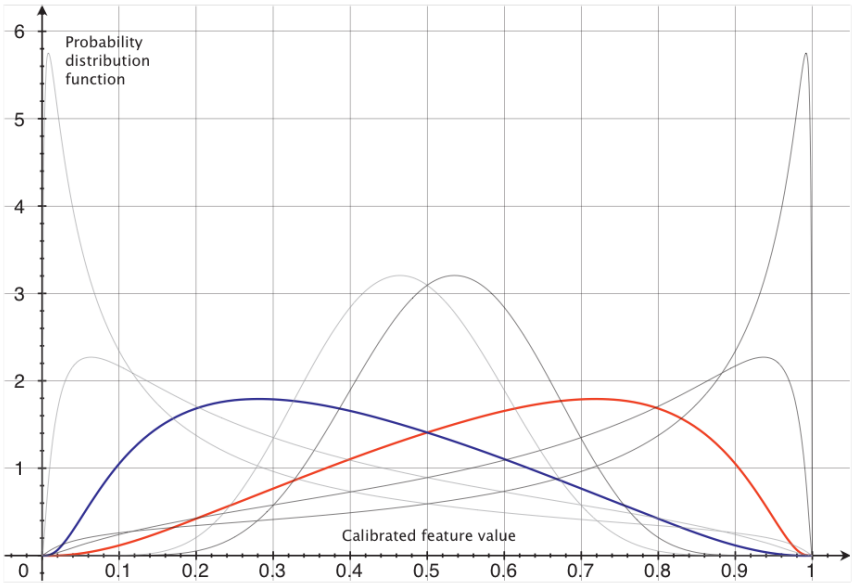

Figure 10.6. Per-class distributions of a logistically calibrated feature for different values of d', the distance between the uncalibrated class means in proportion to the feature's standard deviation. The red and blue curves depict the distributions for the positive and negative class for a feature 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.

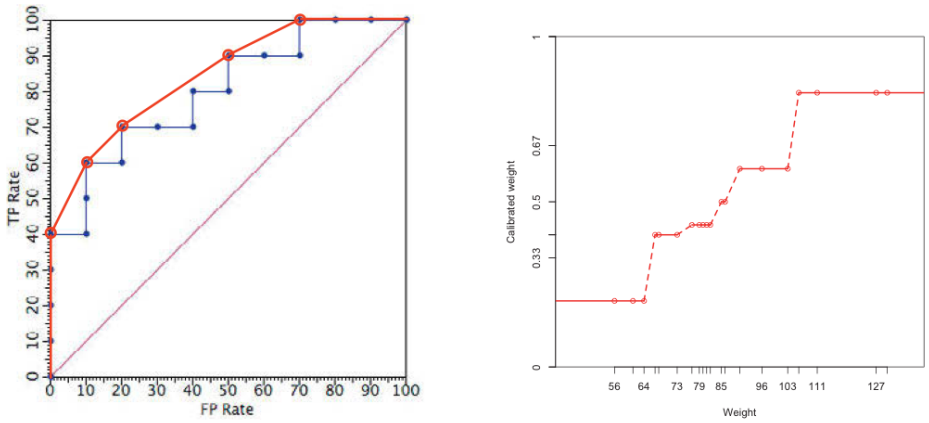

Figure 10.7. (left) ROC curve and convex hull of an uncalibrated feature. Calibrated feature values are obtained from the proportion of positives in each segment of the ROC convex hull. **(right)** The corresponding piecewise-constant calibration map, which maps the uncalibrated feature values on the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

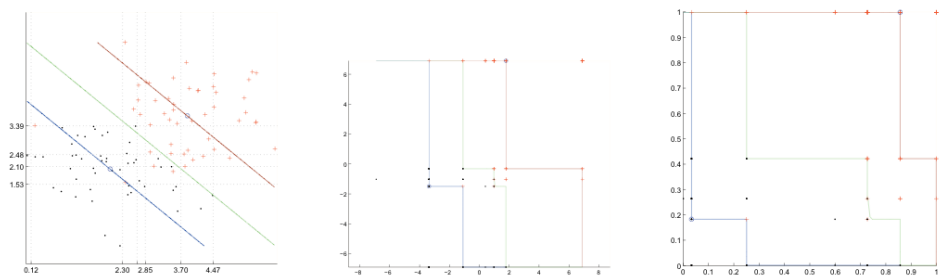

Figure 10.8. (left) The data from Figure 10.5, with grid lines indicating the discretisation obtained by isotonic feature calibration. **(middle)** Isotonically calibrated data in log-odds space. **(right)** Isotonically calibrated data in probability space.

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 $v_i^{\mathrm{c}} = \frac{m_i+1}{m_i+1+c(n_i-m_i+1)}$ with each segment.
4. If an additive scale is required, use $v_i^{\mathrm{d}} = \ln \frac{v_i^{\mathrm{c}}}{1-v_i^{\mathrm{c}}} = \ln v_i^{\mathrm{c}} - \ln(1-v_i^{\mathrm{c}})$.

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 Length = [3,5] ∧ Gills = 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., (Length = [3,5] ∧ Gills = no) = false is equivalent to the disjunction Length ≠ [3,5] ∨ Gills = 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 $\mathrm{Dis}(x_i, h_i)^2$ and increased by $\mathrm{Dis}(x_i, m_i)^2$, 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:

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

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).

Boolean matrix decomposition is studied by Miettinen (2009). Mahoney and Drineas (2009) describe a matrix decomposition technique that uses actual columns and rows of the data matrix to preserve sparsity (unlike SVD which produces dense matrices even if the original matrix is sparse). Latent semantic indexing and a probabilistic extension is described by Hofmann (1999). Ding and He (2004) discuss the relationship between K-means clustering and principal component analysis.