Probabilistic Models

1. Introduction:

1. Probability estimation tree 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.
2. Similarly, a calibrated linear model translates the distance from the decision boundary into a class probability. 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.
3. A. 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.

   B. In particular, since $P(X) = \sum_y P(Y = y, X)$ follows that the posterior distribution can be obtained as

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

   C. 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 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.

   D. Alternatively, we can use P(Y) to sample a class and P(X|Y) to sample an instance for that class.

   E. 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.

4. One of the most attractive features of the probabilistic perspective is that it allow 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.

5. 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. we can attach a probability distribution to almost anything: not just features and targets, but also model parameters and even models.

6. An important concept related to probabilistic models is Bayes-optimality. A classifier is Bayes-optimal if it always assigns argmax_y P∗(Y = y|X = x) to an instance x, where P∗ denotes the true posterior distribution.

Ex: 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.

Note: 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.

2. The normal distribution and its geometric interpretations:

1. 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".**

2. We start by considering the univariate, two-class case. Suppose the values of $x \in R$ follow a mixture model: i.e., each class has its own probability distribution (a component of the mixture model).

3. 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,

μ - and σ- are the mean and standard deviation for the negative class.

4. This gives the following likelihood ratio:

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

Points to Remember:

a. The normal distribution describes a special class of such distribution that are symmetric and can be described by the distribution mean, μ and the standard deviation, σ

b. A continuous random variable is said to be normally distributed with mean and variance if its probability density function is:

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

c. 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.

d. $z = (x - \mu)/\sigma$ 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 $P(x|\mu,\sigma) = 1\sigma P(z|0, 1)$, where $P(z|0,1)$ denotes the *standard normal distribution*.

e. The multivariate normal distribution over d-vectors $x = (x1, \ldots, xd)^T \in R^d$ is

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

f. Lets first consider the case that both components have the same standard deviation, i.e., $\sigma \oplus = \sigma\_ = \sigma$.

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

g. The general form of the likelihood ratio can be derived from

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

where $\mu\oplus$ and $\mu\_$ are the class means, and $\Sigma\oplus$ and $\Sigma\_$ are the covariance matrices for each class.

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.

4. The multivariate normal distribution essentially translates distances into probabilities. This becomes obvious when we plug the definition of *Mahalanobis distance*

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

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

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

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

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

Case Study1: suppose we want to estimate the mean μ of a multivariate Gaussian distribution with given covariance matrix Σ from a set of data points X. The principle of maximum-likelihood estimation states that we should find the value of μ that maximises the joint likelihood of X.

1. 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:

$$\begin{aligned}\hat{\mu} &= \arg\max_{\mu} \prod_{\mathbf{x}\in X} P(\mathbf{x}|\mu,\boldsymbol{\Sigma}) \\ &= \arg\max_{\mu} \prod_{\mathbf{x}\in X} \frac{1}{E_d} \exp\left(-\frac{1}{2}\left(\mathrm{Dis}_M(\mathbf{x},\mu|\boldsymbol{\Sigma})\right)^2\right) \\ &= \arg\min_{\mu} \sum_{\mathbf{x}\in X} \left[\ln E_d + \frac{1}{2}\left(\mathrm{Dis}_M(\mathbf{x},\mu|\boldsymbol{\Sigma})\right)^2\right] \\ &= \arg\min_{\mu} \sum_{\mathbf{x}\in X} \left(\mathrm{Dis}_M(\mathbf{x},\mu|\boldsymbol{\Sigma})\right)^2\end{aligned}$$

2. 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.

Case Study2: (least-squares solution to a linear regression problem):

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., $y_i = f(x_i) + €_i$, where the €i are independently and identically distributed errors.

2. We want to derive the maximum-likelihood estimates ˆ 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

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

3. Taking the partial derivatives with respect to a, b and σ2 and setting to zero in order to maximise the negative log likelihood gives the following three equations:

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

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.

3. Probabilistic models for categorical data:

1. Categorical variables or features (also called discrete or nominal) are ubiquitous in machine learning.

2. 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 Xi governed by a Bernoulli distribution. The joint distribution over the *bit vector* $X = (X1, \ldots ,Xk)$ is called a *multivariate Bernoulli distribution*.

3. Variables with more than two outcomes are also common: for example, every word position in an e-mail corresponds to a categorical variable with k outcomes, where k is the size of the vocabulary.

4. The multinomial distribution manifests itself as a count vector: a histogram of the number of occurrences of all vocabulary words in a document.

5. 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.

6. 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.

7. 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 $(x1, \ldots ,xk)$ as the product of the probabilities of each component $P(Xi = xi)$.

Using a naive Bayes model for classification:

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 = spam) and P(X = x|Y = ham), and apply one of several possible decision rules:

maximum likelihood (ML)	– predict $\operatorname{argmax}_y P(X = x \mid Y = y)$;
maximum a posteriori (MAP)	– predict $\operatorname{argmax}_y P(X = x \mid Y = y)P(Y = y)$;
recalibrated likelihood	– predict $\operatorname{argmax}_y w_y P(X = x \mid Y = y)$.

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

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

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

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

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.

Observation:

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 $x_i = 0$.

- 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' *xi* .

4. 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 *pseudocounts*, 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 *pi* is a distribution over the categories

Example:

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*2: *b c e b b d d e c c*

*e*3: *a d a d e a e e*

*e*4: *b a d b e d a b*

*e*5: *a b a b a b a e d*

*e*6: *a c a c a c a e d*

*e*7: *e a e d a e a*

*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, 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

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	−

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 $\hat{\theta}_{\oplus}$ = (6/20,10/20,4/20) = (0.3,0.5,0.2) for spam and $\hat{\theta}_{-}$ = (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

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	−

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 $\hat{\theta}_{\oplus}$ = (3/6,4/6,2/6) = (0.5,0.67,0.33) for spam and $\hat{\theta}_{-}$ = (4/6,2/6,2/6) = (0.67,0.33,0.33) for ham.

Observation: '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 $\theta^{(c)}{}_{il}$, 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 $P(Fi, Fj \mid C) = P(Fi \mid C)P(Fj \mid C)$ for all pairs of features and for all classes.

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.

5. Discriminative learning by optimising conditional likelihood:

The easiest way to understand logistic regression is as a linear classifier whose probability estimates have been logistically calibrated, 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.

the likelihood ratio as $\exp\left(\gamma(d(\mathbf{x}) - d_0)\right)$ 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 $d0$ into $\mathbf{w}$ and t . So the logistic regression model is 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 $P(yi|\mathbf{x}i)$ 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. the logarithm of the likelihood function is easier to work with:

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

6. 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 $nA, \ldots, nD$ to estimate the class prior $p\hat{}A = nA/n, \ldots, p\hat{}D = n_D/n$

6.1. Expectation-Maximisation:

- the expectation $\mathbb{E}[Z|X,\theta^t]$ 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.

This means that we really want to maximise $P(X \cup \mathbb{E}[Z|X,\theta^t]|\theta)$, 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, $f(\mathbb{E}[Z])=\mathbb{E}[f(Z)]$ and thus we can bring the expectation outside in our objective function:

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

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:

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

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.

6.2. 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 .

A convenient way to model this is to have for each data point $\mathbf{x}i$ a Boolean vector $\mathbf{z}i = (zi1, \ldots, ziK)$ such that exactly one bit zij 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* to obtain a general expression for a Gaussian mixture model:

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

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

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

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 θt , and on the other hand expressions in θ that allow us to find $\theta t+1$ by maximisation. In the general case these expectations are apportioned proportionally to the probability mass assigned to the point by each Gaussian:

$$\mathbb{E}[z_{ij}|\mathbf{X},\theta^t] = \frac{\tau_j^t f(\mathbf{x}_i|\mu_j^t,\Sigma_j^t)}{\sum_{k=1}^K \tau_k^t f(\mathbf{x}_i|\mu_k^t,\Sigma_k^t)}$$

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

6.3. Compression-based models

Consider the maximum a posteriori decision rule again:

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

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

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

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 logarit;:hms to the base 2, in which case information is measured in bits.

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* $'\sum_{i=1}^k -p_i \log_2 p_i$

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:

$$m_{\text{MDL}} = \arg\min_{m \in M} (L(m) + L(D|m))$$

'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.

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 and the *slack variable* term in soft-margin SVMs

7. Features:

Features are 'the workhorses of machine learning' – it is therefore high time to consider them in more detail. Features, also called attributes, are defined as mappings $fi : X \rightarrow Fi$ from the instance space X to the feature domain, Fi.

7.1.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*

7.2. Calculations on features

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.

A. 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.

B. 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.

> 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.

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).

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.

C. 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 {*xi* , . . . ,*xn*} is defined as $m_k = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^k$, where μ is the sample mean.

Skewness is then defined as $m3/\sigma 3$, 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 $m4/\sigma 4$.

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

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

7.3. Structured features:

The instance space is a Cartesian product of *d* feature domains: $X = F1 \times \ldots \times Fd$. 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*.

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.

8. 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

↓ 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**).

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.

8.1. Thresholding and discretisation

Thresholding transforms a quantitative or an ordinal feature into a Boolean feature by finding a feature value to split on. Concretely, let $f : X \rightarrow$R be a quantitative feature and let t □ R be a threshold, then $ft : X \rightarrow$ {true, false} is a Boolean feature defined by $ft(x)$ = true if $f(x) \geq t$ and $ft(x)$ = false if $f(x) < t$.

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. 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*.

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.

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;
```

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 $\chi 2$ *statistic* for both.

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;
```

In agglomerative discretisation the stopping criterion usually takes the form of a simple threshold on the scoring function. In the case of the $\chi 2$ statistic, the threshold can be derived from the *p*-value associated with the $\chi 2$ distribution, which is the probability of observing a $\chi 2$ value above the threshold if the two variables are actually independent.

8.3 Normalisation and calibration

Thresholding and discretisation are feature transformations that remove the scale of a quantitative feature. *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* by centring on the mean and dividing by the standard deviation. In certain cases it is mathematically more convenient to divide by the variance. 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 $f \_\rightarrow (f - l)/(h - l)$. We sometimes have to guess the value of h or l, and truncate any value outside $[l, h]$.

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. 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. Ordinal and quantitative features can be discretised and then calibrated as categorical features. A calibrated weight feature attaches a probability to every weight, such that these probabilities are non-decreasing with weight. 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; we obtain the calibrated feature value as

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

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 $F^{\mathrm{c}}(x) = \frac{\exp(F^{\mathrm{d}}(x))}{1+\exp(F^{\mathrm{d}}(x))}$.

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. 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 $v_i^{\mathrm{d}} = \ln \frac{v_i^{\mathrm{c}}}{1-v_i^{\mathrm{c}}} = \ln v_i^{\mathrm{c}} - \ln(1 - v_i^{\mathrm{c}})$.

8.4. Incomplete features

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*.

9. Feature construction and selection

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. One attractive possibility is to first apply concept learning or subgroup discovery, and then use these concepts or subgroups as new Boolean features. 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 $\chi 2$ 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 Dis(xi ,hi)2 and increased by Dis(xi ,mi)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 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: $\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\mathrm{T}}$

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 $\mathbf{XV} = \mathbf{U\Sigma VTV} = \mathbf{U\Sigma} =\mathbf{W}$ 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 rowis 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:

$$\mathbf{S} = \mathbf{X}^T\mathbf{X} = (\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T)^1 (\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T) = (\mathbf{V}\mathbf{\Sigma}\mathbf{U}^T)(\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T) = \mathbf{V}\mathbf{\Sigma}^2\mathbf{V}^T$$

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

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

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.

10. Bagging and random forests

Bagging, short for 'bootstrap aggregating', is a simple but highly effective ensemble method that creates diverse models on different random samples of the original data set. These samples are taken uniformly with replacement and are known as *bootstrap samples*. Because samples are taken with replacement the bootstrap sample will in general contain duplicates, and hence some of the original data points will be missing even if the bootstrap sample is of the same size as the original data set.

Algorithm 11.1 gives the basic bagging algorithm, which returns the ensemble as a set of models. We can choose to combine the predictions from the different models by voting – the class predicted by the majority of models wins – or by averaging, which is more appropriate if the base classifiers output scores or probabilities.

Algorithm 11.1: Bagging($D, T, \mathscr{A}$) – train an ensemble of models from bootstrap samples.

Input : data set D; ensemble size T; learning algorithm $\mathscr{A}$.
Output : ensemble of models whose predictions are to be combined by voting or averaging.

```
for t = 1 to T do
    build a bootstrap sample D_t from D by sampling |D| data points with replacement;
    run A on D_t to produce a model M_t;
end
return {M_t | 1 ≤ t ≤ T}
```

Bagging is particularly useful in combination with tree models, which are quite sensitive to variations in the training data. When applied to tree models, bagging is often combined with another idea: to build each tree from a different random subset of the features, a process also referred to as *subspace sampling*. This encourages the diversity in the ensemble even more, and has the additional advantage that the training time of each tree is reduced. The resulting ensemble method is called *random forests*, and the algorithm is given in Algorithm 11.2.

Algorithm 11.2: RandomForest(D, T, d) – train an ensemble of tree models from bootstrap samples and random subspaces.

Input : data set D; ensemble size T; subspace dimension d.
Output : ensemble of tree models whose predictions are to be combined by voting or averaging.

```
for t = 1 to T do
    build a bootstrap sample D_t from D by sampling |D| data points with replacement;
    select d features at random and reduce dimensionality of D_t accordingly;
    train a tree model M_t on D_t without pruning;
end
return {M_t | 1 ≤ t ≤ T}
```

11. Boosting

Boosting is an ensemble technique that is superficially similar to bagging, but uses a more sophisticated technique than bootstrap sampling to create diverse training sets. The basic idea is simple and appealing. Suppose we train a linear classifier on a data set and find that its training error rate is _. We want to add another classifier to the ensemble that does better on the misclassifications of the first classifier. One way to do that is to duplicate the misclassified instances: if our base model is the basic linear classifier, this will shift the class means towards the duplicated instances. A better way to achieve the same thing is to give the

misclassified instances a higher weight, and to modify the classifier to take these weights into account (e.g., the basic linear classifier
can calculate the class means as a weighted average).

Algorithm 11.3: Boosting($D, T, \mathscr{A}$) – train an ensemble of binary classifiers from reweighted training sets.

Input : data set D; ensemble size T; learning algorithm $\mathscr{A}$.
Output : weighted ensemble of models.

$w_{1i} \leftarrow 1/|D|$ for all $x_i \in D$; // start with uniform weights
for $t = 1$ to T **do**
  run $\mathscr{A}$ on D with weights w_{ti} to produce a model M_t;
  calculate weighted error ϵ_t;
  if $\epsilon_t \geq 1/2$ **then**
    set $T \leftarrow t-1$ and break
  end
  $\alpha_t \leftarrow \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$; // confidence for this model
  $w_{(t+1)i} \leftarrow \frac{w_{ti}}{2\epsilon_t}$ for misclassified instances $x_i \in D$; // increase weight
  $w_{(t+1)j} \leftarrow \frac{w_{tj}}{2(1-\epsilon_t)}$ for correctly classified instances $x_j \in D$; // decrease weight
end
return $M(x) = \sum_{t=1}^{T} \alpha_t M_t(x)$

12. Boosted rule learning

An interesting variant of boosting arises when our base models are partial classifiers that sometimes abstain from making a prediction. For example, suppose that our base classifiers are conjunctive rules whose head is fixed to predicting the positive class. An individual rule therefore either predicts the positive class for those instances it covers, or otherwise abstains from making a prediction. We can use boosting to learn an ensemble of such rules that takes a weighted vote among its members.

We need to make some small adjustments to the boosting equations, as follows. Notice that ϵ_t is the weighted error of the t-th base classifier. Since our rules always predict positive for covered instances, these errors only concern covered negatives, which we will indicate by $\epsilon_t^{\ominus}$. Similarly, we indicate the weighted sum of covered positives as $\epsilon_t^{\oplus}$, which will play the role of $1-\epsilon_t$. However, with abstaining rules there is a third component, indicated as ϵ_t^0, which is the weighted sum of instances which the rule doesn't cover ($\epsilon_t^0+\epsilon_t^{\oplus}+\epsilon_t^{\ominus}=1$). We then have

$$Z_t=\epsilon_t^0+\epsilon_t^{\ominus}\exp(\alpha_t)+\epsilon_t^{\oplus}\exp(-\alpha_t)$$

The value of α_t which maximises this is

$$\alpha_t=\frac{1}{2}\ln\frac{\epsilon_t^{\oplus}}{\epsilon_t^{\ominus}}=\ln\sqrt{\frac{\epsilon_t^{\oplus}}{\epsilon_t^{\ominus}}} \tag{11.4}$$

which gives

$$Z_t=\epsilon_t^0+2\sqrt{\epsilon_t^{\oplus}\epsilon_t^{\ominus}}=1-\epsilon_t^{\oplus}-\epsilon_t^{\ominus}+2\sqrt{\epsilon_t^{\oplus}\epsilon_t^{\ominus}}=1-\left(\sqrt{\epsilon_t^{\oplus}}-\sqrt{\epsilon_t^{\ominus}}\right)^2$$

This means that in each boosting round we construct a rule that maximises $\left|\sqrt{\epsilon_t^{\oplus}}-\sqrt{\epsilon_t^{\ominus}}\right|$, and set its confidence factor to α_t as in Equation 11.4. In order to obtain a prediction