Chapter 9

Classification models

This chapter concerns discriminative models for classification. The goal is to directly model the posterior probability distribution $Pr(w|\mathbf{x})$ over a discrete world state $w \in \{1, \ldots K\}$ given the continuous observed data vector $\mathbf{x}$. Models for classification are very closely related to those for regression and the reader should be familiar with the contents of chapter 8 before proceeding.

To motivate the models in this chapter, we will consider *gender classification*: here we observe a 60×60 RGB image containing a face (figure 9.1) and concatenate the RGB values to form the 10800×1 vector $\mathbf{x}$. Our goal is to take the vector $\mathbf{x}$ and return the probability distribution $Pr(w|\mathbf{x})$ over a label $w \in \{0, 1\}$ indicating whether the face is male ($w = 0$) or female ($w = 1$).

Gender classification is a *binary classification* task as there are only two possible values of the world state. Throughout most of this chapter, we will restrict our discussion to binary classification. We discuss how to extend these models to cope with an arbitrary number of classes in section 9.9.

9.1 Logistic regression

We will start by considering *logistic regression*, which despite its name is a model that can be applied to classification. Logistic regression (figure 9.2) is a discriminative model; we select a probability distribution over the world state $w \in \{0, 1\}$ and make its parameters contingent on the observed data $\mathbf{x}$. Since the world state is binary, we will describe it with a Bernoulli distribution and we will make the single Bernoulli parameter λ (indicating the probability that the world state takes the value $w = 1$) a function of the measurements $\mathbf{x}$.

In contrast to the regression model, we cannot simply make the parameter λ a linear function $\phi_0 + \boldsymbol{\phi}^T\mathbf{x}$ of the measurements; a linear function can return any value, but the parameter λ must lie between 0 and 1. Consequently, we first compute the linear function and then pass this through the *logistic sigmoid function* $\text{sig}[\bullet]$ that maps the range $[-\infty, \infty]$ to $[0, 1]$. The final model is hence

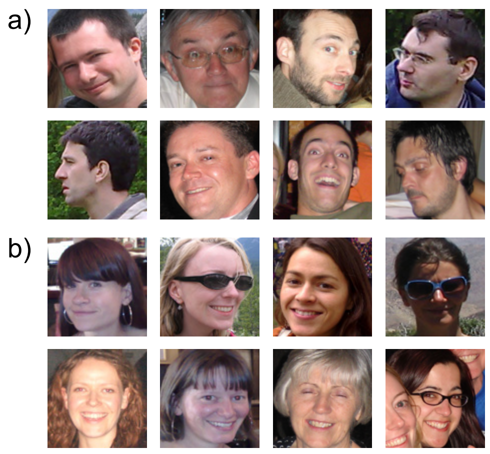

Figure 9.1 Gender classification. Consider a 60×60 pixel image of a face. We concatenate the RGB values to make a 10800×1 data vector $\mathbf{x}$. The goal of gender classification is to use the data $\mathbf{x}$ to infer a label $w \in \{0,1\}$ indicating whether the window contains a) a male or b) a female face. This is challenging because the differences are subtle and there is image variation due to changes in pose, lighting, and expression. Note that real systems would preprocess the image before classification by registering the faces more closely and compensating in some way for lighting variation (see chapter 13).

$$Pr(w|\phi_0, \boldsymbol{\phi}, \mathbf{x}) \quad = \quad \text{Bern}_w\left[\text{sig}[a]\right], \tag{9.1}$$

where a is termed the *activation* and is given by the linear function

$$a = \phi_0 + \boldsymbol{\phi}^T\mathbf{x}. \tag{9.2}$$

The logistic sigmoid function sig[•] is given by

$$\text{sig}[a] = \frac{1}{1+\exp[-a]}. \tag{9.3}$$

Problem 9.1

As the activation a tends to ∞ this function tends to one. As a tends to $-\infty$ it tends to zero. When a is zero, the logistic sigmoid function returns a value of one half. For 1D data x, the overall effect of this transformation is to describe a sigmoid curve relating x to λ (figures 9.2c and 9.3a). The horizontal position of the sigmoid is determined by the place that the linear function a crosses zero (i.e., the x-intercept) and the steepness of the sigmoid depends on the gradient ϕ_1.

In more than one dimension, the relationship between $\mathbf{x}$ and λ is more complex (figure 9.3b). The predicted parameter λ has a sigmoid profile in the direction of the gradient vector $\boldsymbol{\phi}$ but is constant in all perpendicular directions. This induces a linear *decision boundary*. This is the set of positions in data space $\{\mathbf{x} : Pr(w = 1|\mathbf{x}) = 0.5\}$ where the posterior probability is 0.5; the decision boundary separates the region where the world state w is more likely to be 0 from the region where it is more likely to be 1. For logistic regression, the decision boundary takes the form of a hyperplane with the normal vector in the direction of $\boldsymbol{\phi}$.

As for regression, we can simplify the notation by attaching the y-intercept ϕ_0 to the start of the parameter vector $\boldsymbol{\phi}$ so that $\boldsymbol{\phi} \leftarrow [\phi_0 \quad \boldsymbol{\phi}^T]^T$ and attaching 1 to the start of the data vector $\mathbf{x}$ so that $\mathbf{x} \leftarrow [1 \quad \mathbf{x}^T]^T$. After these changes, the activation is now $a = \boldsymbol{\phi}^T\mathbf{x}$, and the final model becomes

$$Pr(w|\boldsymbol{\phi}, \mathbf{x}) = \text{Bern}_w\left[\frac{1}{1+\exp[-\boldsymbol{\phi}^T\mathbf{x}]}\right]. \tag{9.4}$$

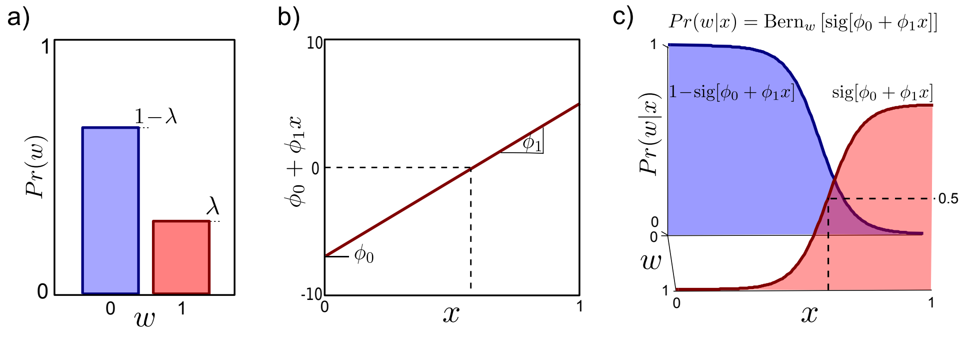

Figure 9.2 Logistic regression. a) We represent the world state w as a Bernoulli distribution and make the Bernoulli parameter λ a function of the observations x. b) We compute the activation a as a linear sum $a = \phi_0 + \phi_1 x$ of the observations. c) The Bernoulli parameter λ is formed by passing the activation through a logistic sigmoid function sig[•] to constrain the value to lie between 0 and 1, giving the characteristic sigmoid shape. In learning, we fit parameters $\boldsymbol{\theta} = \{\phi_0, \phi_1\}$ using training pairs $\{x_i, w_i\}$. In inference, we take a new datum x^* and evaluate the posterior $Pr(w^*|x^*)$ over the state.

Notice that this is very similar to the linear regression model (section 8.1) except for the introduction of the nonlinear logistic sigmoid function sig[•] (explaining the unfortunate name 'logistic regression'). However, this small change has serious implications: maximum likelihood learning of the parameters $\boldsymbol{\phi}$ is considerably harder than for linear regression, and to adopt the Bayesian approach we will be forced to make approximations.

9.1.1 Learning: maximum likelihood

In maximum likelihood learning, we consider fitting the parameters $\boldsymbol{\phi}$ using I paired examples of training data $\{\mathbf{x}_i, w_i\}_{i=1}^{I}$ (figure 9.3). Assuming independence of the training pairs we have

Algorithm 9.1

$$\begin{aligned} Pr(\mathbf{w}|\mathbf{X}, \boldsymbol{\phi}) &= \prod_{i=1}^{I} \lambda^{w_i}(1-\lambda)^{1-w_i} \qquad (9.5) \\ &= \prod_{i=1}^{I} \left(\frac{1}{1+\exp[-\boldsymbol{\phi}^T\mathbf{x}_i]}\right)^{w_i} \left(\frac{\exp[-\boldsymbol{\phi}^T\mathbf{x}_i]}{1+\exp[-\boldsymbol{\phi}^T\mathbf{x}_i]}\right)^{1-w_i}, \end{aligned}$$

where $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_I]$ is a matrix containing the measurements and $\mathbf{w} = [w_1, w_2, \ldots, w_I]^T$ is a vector containing all of the binary world states. The maximum likelihood method finds parameters $\boldsymbol{\phi}$, which maximize this expression.

As usual, however, it is simpler to maximize the logarithm L of this expression. Since the logarithm is a monotonic transformation, it does not change the position

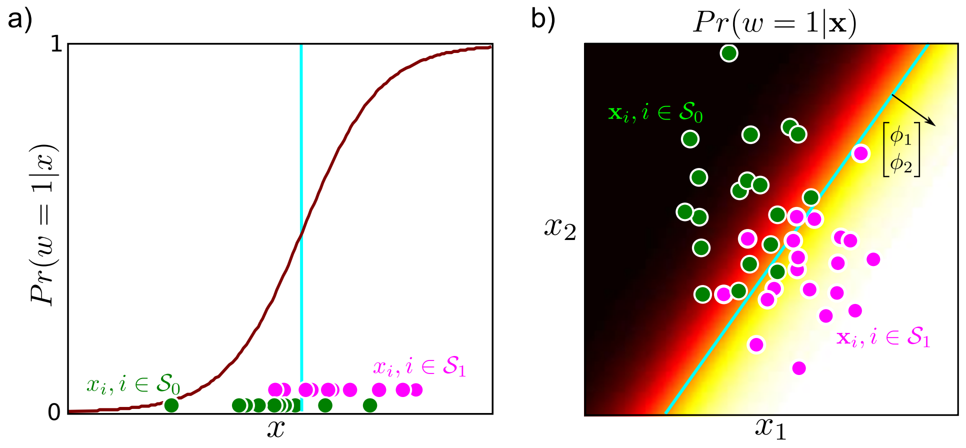

Figure 9.3 Logistic regression model fitted to two different datasets. a) One dimensional data. Green points denote set of examples $\mathcal{S}_0$ where $w = 0$. Pink points denote $\mathcal{S}_1$ where $w = 1$. Note that in this (and all future figures in this chapter) we have only plotted the probability $Pr(w = 1|x)$ (compare to figure 9.2c). The probability $Pr(w = 0|x)$ can be computed as $1 - Pr(w = 1|\mathbf{x})$. b) Two-dimensional data. Here, the model has a sigmoid profile in the direction of the gradient $\boldsymbol{\phi}$ and $Pr(w = 1|\mathbf{x})$ is constant in the orthogonal directions. The decision boundary (cyan line) is linear.

of the maximum with respect to $\boldsymbol{\phi}$. Applying the logarithm replaces the product with a sum so that

$$L = \sum_{i=1}^{I} w_i \log\left[\frac{1}{1+\exp[-\boldsymbol{\phi}^T\mathbf{x}_i]}\right] + \sum_{i=1}^{I}(1-w_i)\log\left[\frac{\exp[-\boldsymbol{\phi}^T\mathbf{x}_i]}{1+\exp[-\boldsymbol{\phi}^T\mathbf{x}_i]}\right]. \tag{9.6}$$

Problem 9.2

The derivative of the log likelihood L with respect to the parameters $\boldsymbol{\phi}$ is

$$\frac{\partial L}{\partial \boldsymbol{\phi}} = -\sum_{i=1}^{I}\left(\frac{1}{1+\exp[-\boldsymbol{\phi}^T\mathbf{x}_i]} - w_i\right)\mathbf{x}_i = -\sum_{i=1}^{I}\left(\text{sig}[a_i] - w_i\right)\mathbf{x}_i. \tag{9.7}$$

Unfortunately, when we equate this expression to zero, there is no way to re-arrange to get a closed form solution for the parameters $\boldsymbol{\phi}$. Instead, we must rely on a *nonlinear optimization* technique to find the maximum of this objective function. Optimization techniques are discussed in detail in appendix B. In brief, we start with an initial estimate of the solution $\boldsymbol{\phi}$ and iteratively improve it until no more progress can be made.

Here, we will apply the *Newton method*, in which we base the update of the parameters on the first and second derivatives of the function at the current position so that

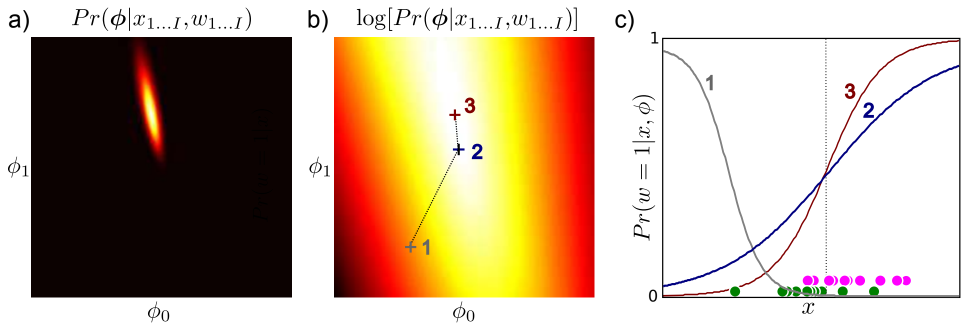

Figure 9.4 Parameter estimation for logistic regression with 1D data. a) In maximum likelihood learning, we seek the maximum of $Pr(\mathbf{w}|\mathbf{X}, \phi)$ with respect to ϕ. b) In practice, we instead maximize log likelihood: notice that the peak is in the same place. Crosses show results of two iterations of optimization using Newton's method. c) The logistic sigmoid functions associated with the parameters at each optimization step. As the log likelihood increases, the model fits the data more closely: the green points represent data where $w = 0$ and the purple points represent data where $w = 1$, so we expect the best-fitting model to increase from left to right just like curve 3.

$$\boldsymbol{\phi}^{[t]} = \boldsymbol{\phi}^{[t-1]} + \alpha \left(\frac{\partial^2 L}{\partial \boldsymbol{\phi}^2}\right)^{-1} \frac{\partial L}{\partial \boldsymbol{\phi}}, \tag{9.8}$$

where $\boldsymbol{\phi}^{[t]}$ denotes the estimate of the parameters $\boldsymbol{\phi}$ at iteration t and α determines how much we change this estimate and is usually chosen by an explicit search at each iteration.

For the logistic regression model, the $D \times 1$ vector of first derivatives and the $D \times D$ matrix of second derivatives are given by Problem 9.3

$$\begin{aligned}\frac{\partial L}{\partial \boldsymbol{\phi}} &= -\sum_{i=1}^{I} (\text{sig}[a_i] - w_i)\mathbf{x}_i \\ \frac{\partial^2 L}{\partial \boldsymbol{\phi}^2} &= -\sum_{i=1}^{I} \text{sig}[a_i](1 - \text{sig}[a_i])\mathbf{x}_i\mathbf{x}_i^T.\end{aligned} \tag{9.9}$$

These are known as the *gradient vector* and the *Hessian matrix*, respectively[1].

The expression for the gradient vector has an intuitive explanation. The contribution of each data point depends on the difference between the actual class w_i and the predicted probability $\lambda = \text{sig}[a_i]$ of being in class 1; points that are classified incorrectly contribute more to this expression and hence have more influence

[1]Note that these are the gradient and Hessian of the log likelihood that we aim to *maximize*. If this is implemented using a nonlinear minimization algorithm, we should multiply the objective function, gradient and Hessian by -1

on the parameter values. Figure 9.4 shows maximum likelihood learning of the parameters $\boldsymbol{\phi}$ for 1D data using a series of Newton steps.

For general functions, the Newton method only finds *local maxima.* At the end of the procedure we cannot be certain that there is not a taller peak in the likelihood function elsewhere. However, the log likelihood for logistic regression has a special property. It is a *concave* function of the parameters $\boldsymbol{\phi}$. For concave functions there are never multiple maxima and gradient-based approaches are guaranteed to find the global maximum. It is possible to establish whether a function is concave or not by examining the Hessian matrix. If this is negative definite for all $\boldsymbol{\phi}$, then the function is concave. This is the case for logistic regression as the Hessian (equation 9.9) consists of a negative weighted sum of outer products[2].

$*$

9.1.2 Problems with the logistic regression model

Problem 9.4

The logistic regression model works well for simple data sets, but for more complex visual data it will not generally suffice. It is limited in the following ways.

1. It is overconfident as it was learned using maximum likelihood.
2. It can only describe linear decision boundaries.
3. It is inefficient and prone to over-fitting in high dimensions.

In the remaining part of this chapter, we will extend this model to cope with these problems (figure 9.5).

9.2 Bayesian logistic regression

In the Bayesian approach, we learn a distribution $Pr(\boldsymbol{\phi}|\mathbf{X}, \mathbf{w})$ over the possible parameter values $\boldsymbol{\phi}$ that are compatible with the training data. In inference, we observe a new data example $\mathbf{x}^*$ and use this distribution to weight the predictions for the world state w^* given by each possible estimate of $\boldsymbol{\phi}$. In linear regression (section 8.2), there were closed form expressions for both of these steps. However, the nonlinear function sig[$\bullet$] in logistic regression means that this is no longer the case. To get around this, we will approximate both steps so that we retain neat closed form expressions and the algorithm is tractable.

9.2.1 Learning

Algorithm 9.2

We start by defining a prior over the parameters $\boldsymbol{\phi}$. Unfortunately, there is no

[2]When we are concerned with minimizing functions we equivalently consider whether the function is *convex* and so has only a single minimum. If the Hessian matrix is positive definite everywhere then the function is convex

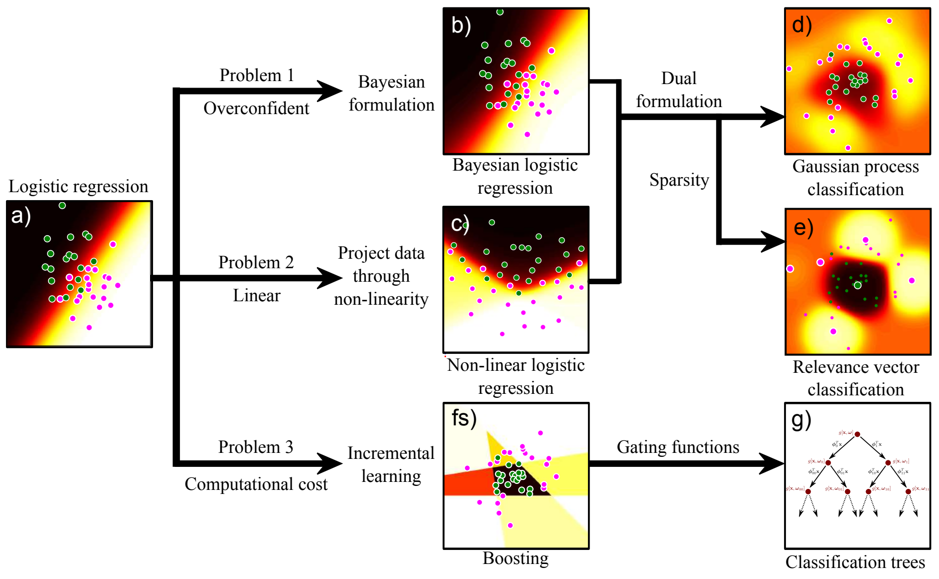

Figure 9.5 Family of classification models. a) In the remaining part of the chapter, we will address several of the limitations of logistic regression for binary classification. b) The logistic regression model with maximum likelihood learning is overconfident, and hence we develop a Bayesian version (section 9.2). c) It is unrealistic to always assume a linear relationship between the data and the world and to this end we introduce a nonlinear version (section 9.3). d) Combining the Bayesian and nonlinear versions of regression leads to Gaussian process classification. e) The logistic regression model also has many parameters and may require considerable resources to learn when the data dimension is high and so we develop relevance vector classification which encourages sparsity. f) We can also build a sparse model by incrementally adding parameters in a boosting model. g) Finally, we consider a very fast classification model based on a tree structure.

conjugate prior for the likelihood in the logistic regression model (equation 9.1); this is why there won't be closed form expressions for the likelihood and predictive distribution. With nothing else to guide us, a reasonable choice for the prior over the continuous parameters $\boldsymbol{\phi}$ is a multivariate normal distribution with zero mean and a large spherical covariance so that

$$Pr(\boldsymbol{\phi}) = \text{Norm}_{\boldsymbol{\phi}}[\mathbf{0}, \sigma_p^2\mathbf{I}]. \tag{9.10}$$

To compute the posterior probability distribution $Pr(\boldsymbol{\phi}|\mathbf{X},\mathbf{w})$ over the parameters $\boldsymbol{\phi}$ given the training data pairs $\{\mathbf{x}_i, w_i\}$ we apply Bayes' rule,

$$Pr(\boldsymbol{\phi}|\mathbf{X},\mathbf{w}) = \frac{Pr(\mathbf{w}|\mathbf{X},\boldsymbol{\phi})Pr(\boldsymbol{\phi})}{Pr(\mathbf{w}|\mathbf{X})}, \tag{9.11}$$

where the likelihood and prior are given by equations 9.5 and 9.10, respectively.

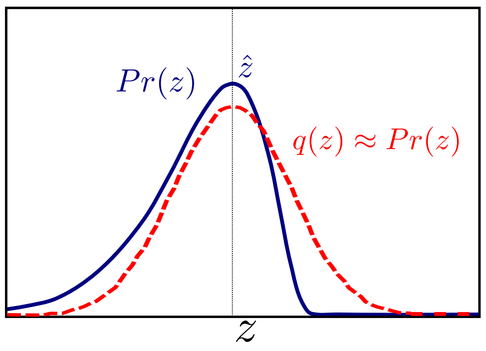

Figure 9.6 Laplace approximation. A probability density (blue curve) is approximated by a normal distribution (red curve). The mean of the normal (and hence the peak) is chosen to coincide with the peak of the original pdf. The variance of the normal is chosen so that its second derivatives at the mean match the second derivatives of the original pdf at the peak.

Since we are not using a conjugate prior, there is no simple closed form expression for this posterior and so we are forced to make an approximation of some kind.

Problem 9.5
Problem 9.6

One possibility is to use the *Laplace approximation* (figure 9.6) which is a general method for approximating complex probability distributions. The goal is to approximate the posterior distribution by a multivariate normal. We select the parameters of this normal so that (i) the mean is at the peak of the posterior distribution (i.e., at the MAP estimate) and (ii) the covariance is such that the second derivatives at the peak match the second derivatives of the true posterior distribution at its peak.

Hence, to make the Laplace approximation, we first find the MAP estimate of the parameters $\hat{\boldsymbol{\phi}}$, and to this end we use a nonlinear optimization technique such as Newton's method to maximize the criterion

$$L = \sum_{i=1}^{I} \log[Pr(w_i|\mathbf{x}_i, \boldsymbol{\phi})] + \log[Pr(\boldsymbol{\phi})]. \tag{9.12}$$

Newton's method needs the derivatives of the log posterior which are

$$\begin{aligned} \frac{\partial L}{\partial \boldsymbol{\phi}} &= -\sum_{i=1}^{I}(\text{sig}[a_i] - w_i)\mathbf{x}_i - \frac{\boldsymbol{\phi}}{\sigma_p^2} \\ \frac{\partial^2 L}{\partial \boldsymbol{\phi}^2} &= -\sum_{i=1}^{I}\text{sig}[a_i](1-\text{sig}[a_i])\mathbf{x}_i\mathbf{x}_i^T - \frac{1}{\sigma_p^2}. \end{aligned} \tag{9.13}$$

We then approximate the posterior by a multivariate normal so that

$$Pr(\boldsymbol{\phi}|\mathbf{X}, \mathbf{w}) \approx q(\boldsymbol{\phi}) = \text{Norm}_{\boldsymbol{\phi}}[\boldsymbol{\mu}, \boldsymbol{\Sigma}], \tag{9.14}$$

Problem 9.7

where the mean $\boldsymbol{\mu}$ is set to the MAP estimate $\hat{\boldsymbol{\phi}}$, and the covariance $\boldsymbol{\Sigma}$ is chosen so that the second derivatives of the normal match those of the posterior distribution at the MAP estimate (figure 9.7) so that

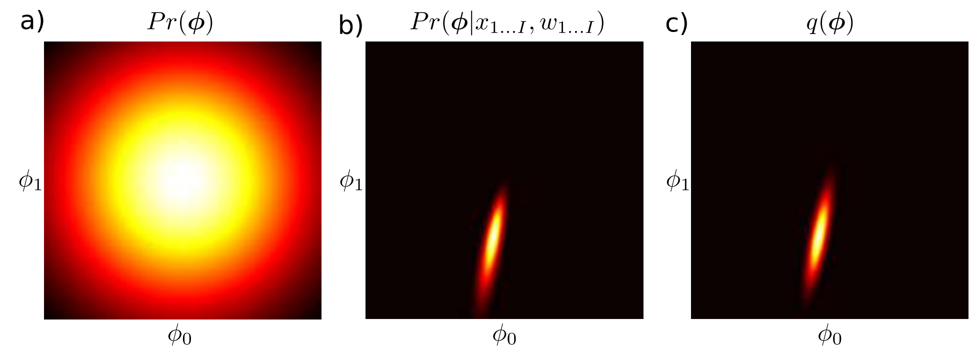

Figure 9.7 Laplace approximation for logistic regression. a) The prior $Pr(\boldsymbol{\phi})$ over the parameters is a normal distribution with mean zero and a large spherical covariance. b) The posterior distribution $Pr(\boldsymbol{\phi}|\mathbf{X}, \mathbf{w})$ represents the refined state of our knowledge after observing the data. Unfortunately, this posterior cannot be expressed in closed form. c) We approximate the true posterior with a normal distribution $q(\boldsymbol{\phi}) = \text{Norm}_{\boldsymbol{\phi}}[\boldsymbol{\mu}, \boldsymbol{\Sigma}]$ whose mean is at the peak of the posterior and whose covariance is chosen so that the second derivatives at the peak of the true posterior match the second derivatives at the peak of the normal. This is termed the Laplace approximation.

$$\begin{aligned}\boldsymbol{\mu} &= \hat{\boldsymbol{\phi}} \\ \boldsymbol{\Sigma} &= -\left(\frac{\partial^2 L}{\partial \boldsymbol{\phi}^2}\right)^{-1}\Bigg|_{\boldsymbol{\phi}=\hat{\boldsymbol{\phi}}}.\end{aligned} \tag{9.15}$$

9.2.2 Inference

In inference we aim to compute a posterior distribution $Pr(w^*|\mathbf{x}^*, \mathbf{X}, \mathbf{w})$ over the world state w^* given new observed data $\mathbf{x}^*$. To this end, we compute an infinite weighted sum (i.e., an integral) of the predictions $Pr(w^*|\mathbf{x}^*, \boldsymbol{\phi})$ given by each possible value of the parameters $\boldsymbol{\phi}$,

$$\begin{aligned}Pr(w^*|\mathbf{x}^*, \mathbf{X}, \mathbf{w}) &= \int Pr(w^*|\mathbf{x}^*, \boldsymbol{\phi}) Pr(\boldsymbol{\phi}|\mathbf{X}, \mathbf{w}) d\boldsymbol{\phi} \\ &\approx \int Pr(w^*|\mathbf{x}^*, \boldsymbol{\phi}) q(\boldsymbol{\phi}) d\boldsymbol{\phi}.\end{aligned} \tag{9.16}$$

where the weights $q(\boldsymbol{\phi})$ are given by the approximated posterior distribution over the parameters from the learning stage. Unfortunately, this integral cannot be computed in closed form either, and so we must make a further approximation.

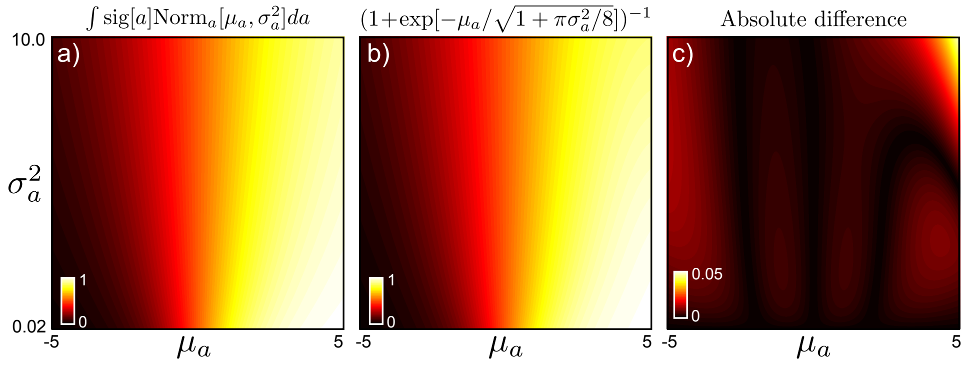

Figure 9.8 Approximation of activation integral (equation 9.19). a) Actual result of integral as a function of μ_a and σ_a^2. b) The (non-obvious) approximation from equation 9.19. c) The absolute difference between the actual result and the approximation is very small over a range of reasonable values.

We first note that the prediction $Pr(w^*|\mathbf{x}^*, \boldsymbol{\phi})$ depends only on a linear projection $a = \boldsymbol{\phi}^T\mathbf{x}^*$ of the parameters (see equation 9.4). Hence we could re-express the prediction as

$$Pr(w^*|\mathbf{x}^*, \mathbf{X}, \mathbf{w}) \approx \int Pr(w^*|a)Pr(a)da. \tag{9.17}$$

The probability distribution $Pr(a)$ can be computed using the transformation property of the normal distribution (section 5.3) and is given by

$$\begin{aligned} Pr(a) = Pr(\boldsymbol{\phi}^T\mathbf{x}^*) &= \text{Norm}_a[\boldsymbol{\mu}^T\mathbf{x}^*, \mathbf{x}^{*T}\boldsymbol{\Sigma}\mathbf{x}] \\ &= \text{Norm}_a[\mu_a, \sigma_a^2], \end{aligned} \tag{9.18}$$

where we have denoted the mean and variance of the activation by μ_a and σ_a^2, respectively. The one dimensional integration in equation 9.17 can now be computed using numerical integration over a, or we can approximate the result with a similar function such as

$$\int Pr(w^*|a)\text{Norm}_a[\mu_a, \sigma_a^2]da \approx \frac{1}{1+\exp[-\mu_a/\sqrt{1+\pi\sigma_a^2/8}]}. \tag{9.19}$$

It is not obvious by inspection that this function should approximate the integral well; however, figure 9.8 demonstrates that the approximation is quite accurate.

Figure 9.9 compares the classification predictions $Pr(w^*|\mathbf{x}^*)$ for the maximum likelihood and Bayesian approaches for logistic regression. The Bayesian approach makes more moderate predictions for the final class. This is particularly the case in regions of data space that are far from the mean.

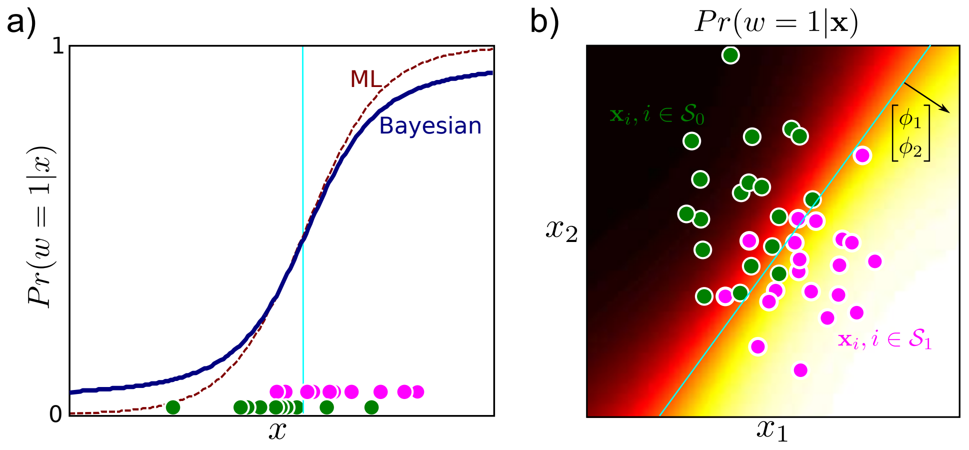

Figure 9.9 Bayesian logistic regression predictions. a) The Bayesian prediction for the class w is more moderate than the maximum likelihood prediction. b) In 2D the decision boundary in the Bayesian case (blue line) is still linear but iso-probability contours at levels other than 0.5 are curved (compare to maximum likelihood case in figure 9.3b). Here too, the Bayesian solution makes more moderate predictions than the maximum likelihood model.

9.3 Non-linear logistic regression

The logistic regression model described previously can only create linear decision boundaries between classes. To create nonlinear decision boundaries, we adopt the same approach as we did for regression (section 8.3): we compute a nonlinear transformation $\mathbf{z} = \mathbf{f}[\mathbf{x}]$ of the observed data and then build the logistic regression model substituting the original data $\mathbf{x}$ for the transformed data $\mathbf{z}$, so that

$$Pr(w=1|\mathbf{x},\boldsymbol{\phi}) = \text{Bern}_w\left[\text{sig}[\boldsymbol{\phi}^T\mathbf{z}]\right] = \text{Bern}_w\left[\text{sig}[\boldsymbol{\phi}^T\mathbf{f}[\mathbf{x}]]\right]. \tag{9.20}$$

The logic of this approach is that arbitrary nonlinear activations can be built as a linear sum of nonlinear basis functions. Typical nonlinear transformations include

- Heaviside step functions of projections: $z_k = \text{heaviside}[\boldsymbol{\alpha}_k^T\mathbf{x}]$,
- arc tangent functions of projections: $z_k = \arctan[\boldsymbol{\alpha}_k^T\mathbf{x}]$, and
- radial basis functions: $z_k = \exp[-\frac{1}{\lambda_0}(\mathbf{x}-\boldsymbol{\alpha}_k)^T(\mathbf{x}-\boldsymbol{\alpha}_k)]$

where z_k denotes the k^{th} element of the transformed vector $\mathbf{z}$ and the function heaviside[$\bullet$] returns zero if its argument is less than zero and one otherwise. In the first two cases we have attached a 1 to the start of the observed data $\mathbf{x}$ where we use projections $\boldsymbol{\alpha}^T\mathbf{x}$ to avoid having a separate offset parameter. Figures 9.10 and 9.11 show examples of nonlinear classification using arc tangent functions for one- and two-dimensional data, respectively.

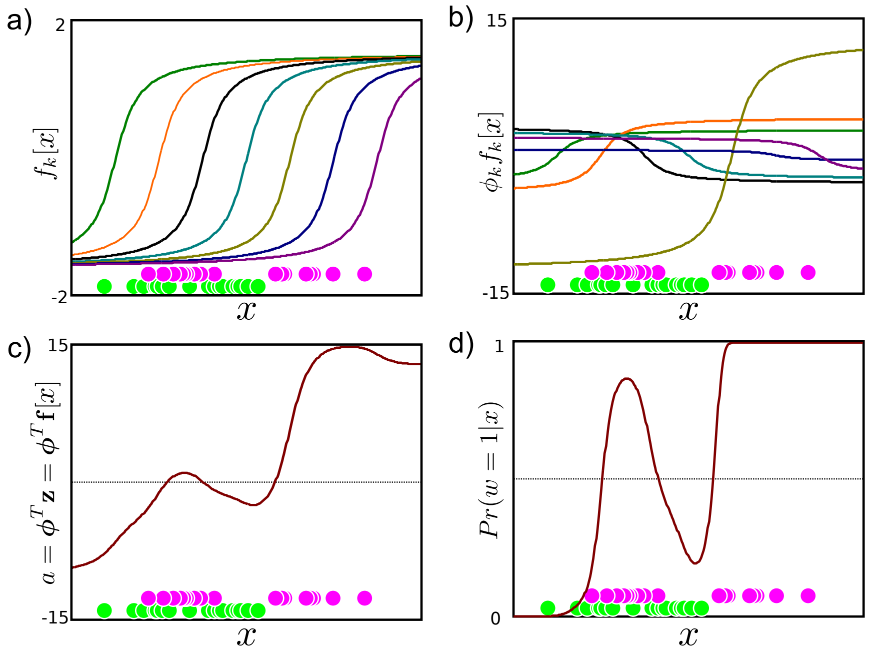

Figure 9.10 Non-linear classification in 1D using arc tangent transformation. We consider a complex 1D data set (bottom of all panels) where the posterior $Pr(w = 1|x)$ cannot easily be described by a single sigmoid. Green circles represent data x_i where $w_i = 0$. Pink circles represent data x_i where $w_i = 1$. a) The seven dimensional transformed data vectors $z_1 \ldots z_I$ are computed by evaluating each data example against seven predefined arc tangent functions $z_{ik} = f_k[x_i] = \arctan[\alpha_{0k} + \alpha_{1k}x_i]$. b) When we learn the parameters $\boldsymbol{\phi}$ we are learning weights for these nonlinear arc tangent functions. The functions are shown after applying the maximum likelihood weights $\hat{\boldsymbol{\phi}}$. c) The final activation $a = \boldsymbol{\phi}^T\mathbf{z}$ is a weighted sum of the nonlinear functions. d) The probability $Pr(w = 1|x)$ is computed by passing the activation a through the logistic sigmoid function.

Note that the basis functions also have parameters. For example, in the arc tangent example, there are the projection directions $\{\boldsymbol{\alpha}_k\}_{k=1}^{K}$ each of which contains an offset and a set of gradients. These can also be optimized during the fitting procedure together with the weights $\boldsymbol{\phi}$. We form a new vector of unknowns $\boldsymbol{\theta} = [\boldsymbol{\phi}^T, \boldsymbol{\alpha}_1^T, \boldsymbol{\alpha}_2^T, \ldots, \boldsymbol{\alpha}_K^T]^T$ and optimize the model with respect to all of these unknowns together. The gradient vector and the Hessian matrix depend on the chosen transformation $\mathbf{f}[\bullet]$ but can be computed using the expressions

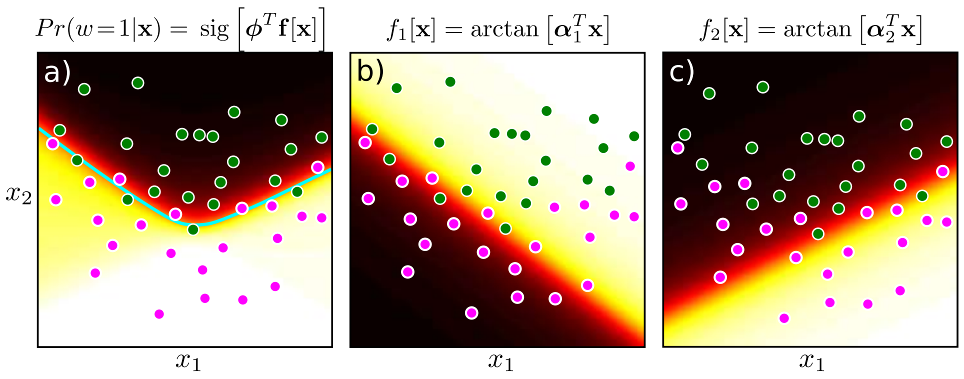

Figure 9.11 Non-linear classification in 2D using arc tangent transform. a) These data have been successfully classified with nonlinear logistic regression. Note the nonlinear decision boundary (cyan line). To compute the posterior $Pr(w = 1|\mathbf{x})$, we transform the data to a new two-dimensional space $\mathbf{z} = \mathbf{f}[\mathbf{x}]$ where the elements of $\mathbf{z}$ are computed by evaluating $\mathbf{x}$ against the 1D arc tangent functions in b) and c). The arc tangent activations are weighted (the first by a negative number) and summed and the result is put through the logistic sigmoid to compute $Pr(w = 1|\mathbf{x})$.

$$\begin{aligned}\frac{\partial L}{\partial \boldsymbol\theta} &= -\sum_{i=1}^{I}(w_i - \text{sig}[a_i])\frac{\partial a_i}{\partial \boldsymbol\theta}\\ \frac{\partial^2 L}{\partial \boldsymbol\theta^2} &= -\sum_{i=1}^{I}\text{sig}[a_i](\text{sig}[a_i]-1)\frac{\partial a_i}{\partial \boldsymbol\theta}\frac{\partial a_i}{\partial \boldsymbol\theta}^T - (w_i - \text{sig}[a_i])\frac{\partial^2 a_i}{\partial \boldsymbol\theta^2},\end{aligned} \tag{9.21}$$

where $a_i = \boldsymbol\phi^T\mathbf{f}[\mathbf{x}_i]$. These relations were established using the chain rule for derivatives. Unfortunately, this joint optimization problem is generally not convex and will be prone to terminating in local maxima. In the Bayesian case, it would be typical to marginalize over the parameters $\boldsymbol\phi$ but maximize over the function parameters.

9.4 Dual logistic regression

Algorithm 9.3

There is a potential problem with the logistic regression models as described earlier: in the original linear model, there is one element of the gradient vector $\boldsymbol\phi$ corresponding to each dimension of the observed data $\mathbf{x}$, and in the nonlinear extension there is one element corresponding to each transformed data dimension $\mathbf{z}$. If the relevant data $\mathbf{x}$ (or $\mathbf{z}$) is very high-dimensional, then the model will have a large number of parameters: this will render the Newton update slow or even intractable.

To solve this problem, we switch to the *dual* representation. For simplicity, we will develop this model using the original data $\mathbf{x}$, but all of the ideas transfer directly to the nonlinear case where we use transformed data $\mathbf{z}$.

In the *dual* parameterization, we express the gradient parameters $\boldsymbol{\phi}$ as a weighted sum of the observed data (see figure 8.12) so that

$$\boldsymbol{\phi} = \mathbf{X}\boldsymbol{\psi}, \tag{9.22}$$

where $\boldsymbol{\psi}$ is an $I \times 1$ variable where each element weights one of the data examples. If the number of data points I is less than the dimensionality D of the data $\mathbf{x}$, then the number of parameters has been reduced.

The price that we pay for this reduction is that we can now only choose gradient vectors $\boldsymbol{\phi}$ that are in the space spanned by the data examples. However, the gradient vector represents the direction in which the final probability $Pr(w = 1|\mathbf{x})$ changes fastest, and this should not point in a direction in which there was no variation in the training data anyway, so this is not a limitation.

Substituting equation 9.22 into the original logistic regression model leads to the dual logistic regression model,

$$Pr(\mathbf{w}|\mathbf{X}, \boldsymbol{\psi}) = \prod_{i=1}^{I} \text{Bern}_{w_i}\left[\text{sig}[a_i]\right] = \prod_{i=1}^{I} \text{Bern}_{w_i}\left[\text{sig}[\boldsymbol{\psi}^T \mathbf{X}^T \mathbf{x}_i]\right]. \tag{9.23}$$

The resulting learning and inference algorithms are very similar to those for the original logistic regression model, so we cover them only in brief:

- In the maximum likelihood method, we learn the parameters $\boldsymbol{\psi}$ by nonlinear optimization of the log likelihood $L = \log[Pr(\mathbf{w}|\mathbf{X}, \boldsymbol{\psi})]$ using the Newton method. This optimization requires the derivatives of the log likelihood, which are

$$\begin{aligned}\frac{\partial L}{\partial \boldsymbol{\psi}} &= -\sum_{i=1}^{I} \left(\text{sig}[a_i] - w_i\right) \mathbf{X}^T \mathbf{x}_i \\ \frac{\partial^2 L}{\partial \boldsymbol{\psi}^2} &= -\sum_{i=1}^{I} \text{sig}[a_i]\left(1 - \text{sig}[a_i]\right) \mathbf{X}^T \mathbf{x}_i \mathbf{x}_i^T \mathbf{X}.\end{aligned} \tag{9.24}$$

- In the Bayesian approach, we use a normal prior over the parameters $\boldsymbol{\psi}$,

$$Pr(\boldsymbol{\psi}) = \text{Norm}_{\boldsymbol{\psi}}[\mathbf{0}, \sigma_p^2 \mathbf{I}]. \tag{9.25}$$

  The posterior distribution $Pr(\boldsymbol{\psi}|\mathbf{X}, \mathbf{w})$ over the new parameters is found using Bayes' rule, and once more this cannot be written in closed form, so we apply the Laplace approximation; we find the MAP solution $\hat{\boldsymbol{\psi}}$ using nonlinear optimization, which requires the derivatives of the log posterior $L = \log[Pr(\boldsymbol{\psi}|\mathbf{X}, \mathbf{w})]$:

$$\begin{aligned}
\frac{\partial L}{\partial \boldsymbol{\psi}} &= -\sum_{i=1}^{I}(\text{sig}[a_i]-w_i)\,\mathbf{X}^T\mathbf{x}_i - \frac{\boldsymbol{\psi}}{\sigma_p^2} \\
\frac{\partial^2 L}{\partial \boldsymbol{\psi}^2} &= -\sum_{i=1}^{I}\text{sig}[a_i]\,(1-\text{sig}[a_i])\,\mathbf{X}^T\mathbf{x}_i\mathbf{x}_i^T\mathbf{X} - \frac{1}{\sigma_p^2}.
\end{aligned} \tag{9.26}$$

The posterior is now approximated by a multivariate normal so that

$$Pr(\boldsymbol{\psi}|\mathbf{X},\mathbf{w}) \approx q(\boldsymbol{\psi}) = \text{Norm}_{\boldsymbol{\psi}}[\boldsymbol{\mu},\boldsymbol{\Sigma}], \tag{9.27}$$

where

$$\begin{aligned}
\boldsymbol{\mu} &= \hat{\boldsymbol{\psi}} \\
\boldsymbol{\Sigma} &= -\left(\frac{\partial^2 L}{\partial \boldsymbol{\psi}^2}\right)^{-1}\Bigg|_{\boldsymbol{\psi}=\hat{\boldsymbol{\psi}}}.
\end{aligned} \tag{9.28}$$

In inference, we compute the distribution over the activation

$$\begin{aligned}
Pr(a) = Pr(\boldsymbol{\psi}^T\mathbf{X}^T\mathbf{x}^*) &= \text{Norm}_a[\mu_a,\sigma_a^2] \\
&= \text{Norm}_a[\boldsymbol{\mu}^T\mathbf{X}^T\mathbf{x}^*, \mathbf{x}^{*T}\mathbf{X}\boldsymbol{\Sigma}\mathbf{X}^T\mathbf{x}^*],
\end{aligned} \tag{9.29}$$

and then approximate the predictive distribution using equation 9.19.

Dual logistic regression gives identical results to the original logistic regression algorithm for the maximum likelihood case and very similar results in the Bayesian situation (where the difference results from the slightly different priors). However, the dual classification model is much faster to fit in high dimensions as the parameters are fewer.

9.5 Kernel logistic regression

Algorithm 9.4

We motivated the dual model by the reduction in the number of parameters $\boldsymbol{\psi}$ in the model when the data lies in a high dimensional space. However, now that we have developed the model, a further advantage is easy to identify: both learning and inference in the dual model rely only on inner products $\mathbf{x}_i^T\mathbf{x}_j$ of that data. Equivalently, the nonlinear version of this algorithm depends only on inner products $\mathbf{z}_i^T\mathbf{z}_j$ of the transformed data vectors. This means that the algorithm is suitable for *kernelization* (see section 8.4).

The idea of kernelization is to define a kernel function $k[\bullet,\bullet]$, which computes the quantity

$$\mathrm{k}[\mathbf{x}_i, \mathbf{x}_j] = \mathbf{z}_i^T \mathbf{z}_j, \tag{9.30}$$

where $\mathbf{z}_i = \mathbf{f}[\mathbf{x}_i]$ and $\mathbf{z}_j = \mathbf{f}[\mathbf{x}_j]$ are the nonlinear transformations of the two data vectors. Replacing the inner products with the kernel function means that we do not have to explicitly calculate the transformed vectors $\mathbf{z}$ and hence they may be of very high, or even infinite dimensions. See section 8.4 for a more detailed description of kernel functions.

The kernel logistic regression model (compare to equation 9.23) is hence

$$Pr(\mathbf{w}|\mathbf{X}, \boldsymbol{\psi}) = \prod_{i=1}^{I} \mathrm{Bern}_{w_i}\left[\mathrm{sig}[a_i]\right] = \prod_{i=1}^{I} \mathrm{Bern}_{w_i}\left[\mathrm{sig}[\boldsymbol{\psi}^T \mathbf{K}[\mathbf{X}, \mathbf{x}_i]\right], \tag{9.31}$$

where the notation $\mathbf{K}[\mathbf{X}, \mathbf{x}]_i$ represents a column vector of dot products where element k is given by $\mathrm{k}[\mathbf{x}_k, \mathbf{x}_i]$.

For maximum likelihood learning, we simply optimize the log posterior probability L with respect to the parameters, which requires the derivatives:

$$\begin{aligned}\frac{\partial L}{\partial \boldsymbol{\psi}} &= -\sum_{i=1}^{I} (\mathrm{sig}[a_i] - w_i)\, \mathbf{K}[\mathbf{X}, \mathbf{x}_i] \\ \frac{\partial^2 L}{\partial \boldsymbol{\psi}^2} &= -\sum_{i=1}^{I} \mathrm{sig}[a_i]\,(1 - \mathrm{sig}[a_i])\, \mathbf{K}[\mathbf{X}, \mathbf{x}_i]\mathbf{K}[\mathbf{x}_i, \mathbf{X}].\end{aligned} \tag{9.32}$$

The Bayesian formulation of kernel logistic regression, which is sometimes known as *Gaussian process classification*, proceeds along similar lines; we follow the dual formulation, replacing each of the dot products between data examples with the kernel function.

Problem 9.8

A very common example of a kernel function is the radial basis kernel in which the nonlinear transformation and inner product operations are replaced by

$$\mathrm{k}[\mathbf{x}_i, \mathbf{x}_j] = \exp\left[-0.5\left(\frac{(\mathbf{x}_i - \mathbf{x}_j)^T(\mathbf{x}_i - \mathbf{x}_j)}{\lambda^2}\right)\right]. \tag{9.33}$$

This is equivalent to computing transformed vectors $\mathbf{z}_i$ and $\mathbf{z}_j$ of infinite length, where each entry evaluates the data $\mathbf{x}$ against a radial basis function at a different position, and then computing the inner product $\mathbf{z}_i^T\mathbf{z}_j$. Examples of the kernel logistic regression with a radial basis kernel are shown in figures 9.12 and 9.13.

9.6 Relevance vector classification

The Bayesian version of the kernel logistic regression model is powerful, but computationally expensive as it requires us to compute dot products between the new data example and the all of the training examples (in the kernel function in equation 9.31). It would be more efficient if the model depended only sparsely on the

Algorithm 9.5

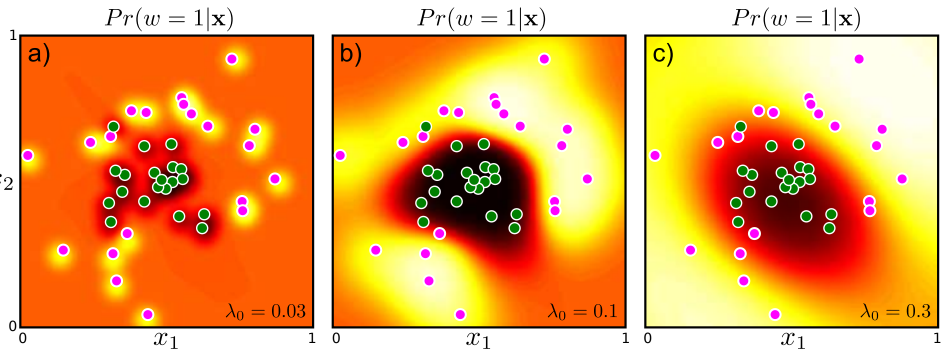

Figure 9.12 Kernel logistic regression using RBF kernel and maximum likelihood learning. a) With a small length scale λ, the model does not interpolate much from the data examples. b) With a reasonable length scale, the classifier does a good job of modeling the posterior $Pr(w=1|\mathbf{x})$. c) With a large length scale, the estimated posterior is very smooth and the model interpolates confident decisions into regions such as the top-left where there is no data.

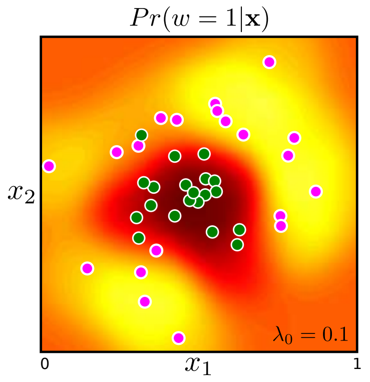

Figure 9.13 Kernel logistic regression with RBF kernel in a Bayesian setting: we now take account of our uncertainty in the dual parameters $\boldsymbol{\psi}$ by approximating their posterior distribution using Laplace's method and marginalizing them out of the model. This produces a very similar result to the maximum likelihood case with the same length scale (figure 9.12b). However, as is typical with Bayesian implementations, the confidence is (appropriately) somewhat lower.

training data. To achieve this, we impose a penalty for every non-zero weighted training example. As in the relevance regression model (section 8.8), we replace the normal prior over the dual parameters $\boldsymbol{\psi}$ (equation 9.25) with a product of one-dimensional t-distributions so that

$$Pr(\boldsymbol{\psi}) = \prod_{i=1}^{I} \text{Stud}_{\psi_i}\left[0, 1, \nu\right]. \tag{9.34}$$

Applying the Bayesian approach to this model with respect to the parameters $\boldsymbol{\Psi}$ is known as *relevance vector classification*.

Following the argument of section 8.6, we re-write each student t-distribution as a marginalization of a joint distribution $Pr(\psi_i, h_i)$

$$\begin{aligned} Pr(\boldsymbol{\psi}) &= \prod_{i=1}^{I}\int \text{Norm}_{\psi_i}\left[0,\frac{1}{h_i}\right]\text{Gam}_{h_i}\left[\frac{\nu}{2},\frac{\nu}{2}\right]\,dh_i \\ &= \int \text{Norm}_{\boldsymbol{\psi}}[0,\mathbf{H}^{-1}]\prod_{d=1}^{D}\text{Gam}_{h_d}[\nu/2,\nu/2]\,d\mathbf{H}, \end{aligned} \tag{9.35}$$

where the matrix $\mathbf{H}$ contains the hidden variables $\{h_i\}_{i=1}^{I}$ on its diagonal and zeros elsewhere. Now we can write the model likelihood as

$$\begin{aligned} &Pr(\mathbf{w}|\mathbf{X}) \\ &\quad= \int Pr(\mathbf{w}|\mathbf{X},\boldsymbol{\psi})Pr(\boldsymbol{\psi})\,d\boldsymbol{\psi} \\ &\quad= \iint\prod_{i=1}^{I}\text{Bern}_{w_i}\Big[\text{sig}[\boldsymbol{\psi}^T\mathbf{K}[\mathbf{X},\mathbf{x}_i]\Big]\text{Norm}_{\boldsymbol{\psi}}[0,\mathbf{H}^{-1}]\prod_{d=1}^{D}\text{Gam}_{h_d}[\nu/2,\nu/2]\,d\mathbf{H}d\boldsymbol{\psi}. \end{aligned} \tag{9.36}$$

Now we make two approximations. First, we use the Laplace approximation to describe the first two terms in this integral as a normal distribution with mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$ centered at the MAP parameters, and use the following result for the integral over $\boldsymbol{\psi}$:

$$\begin{aligned} \int q(\boldsymbol{\psi})\,d\boldsymbol{\psi} &\approx q(\boldsymbol{\mu})\int\exp\left[-\frac{1}{2}(\boldsymbol{\psi}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{\psi}-\boldsymbol{\mu})\right]\,d\boldsymbol{\psi} \\ &= q(\boldsymbol{\mu})(2\pi)^{D/2}|\boldsymbol{\Sigma}|^{1/2}. \end{aligned} \tag{9.37}$$

This yields the expression

$$\begin{aligned} &Pr(\mathbf{w}|\mathbf{X}) \approx \\ &\quad\int\prod_{i=1}^{I}(2\pi)^{I/2}|\boldsymbol{\Sigma}|^{0.5}\text{Bern}_{w_i}\big[\text{sig}[\boldsymbol{\mu}^T\mathbf{K}[\mathbf{X},\mathbf{x}_i]]\text{Norm}_{\boldsymbol{\mu}}\big[0,\mathbf{H}^{-1}\big]\text{Gam}_{h_i}\Big[\frac{\nu}{2},\frac{\nu}{2}\Big]\,d\mathbf{H}, \end{aligned} \tag{9.38}$$

where the matrix $\mathbf{H}$ contains the hidden variables $\{h_i\}_{i=1}^{I}$ on the diagonal, and we have used the general result for the Laplace approximation.

In the second approximation, we maximize over the hidden variables, rather than integrate over them. This yields the expression:

$$\begin{aligned} &Pr(\mathbf{w}|\mathbf{X}) \approx \\ &\quad\max_{\mathbf{H}}\left[\prod_{i=1}^{I}(2\pi)^{I/2}|\boldsymbol{\Sigma}|^{0.5}\text{Bern}_{w_i}\big[\text{sig}[\boldsymbol{\mu}^T\mathbf{K}[\mathbf{X},\mathbf{x}_i]]\text{Norm}_{\boldsymbol{\mu}}\big[0,\mathbf{H}^{-1}\big]\text{Gam}_{h_i}\Big[\frac{\nu}{2},\frac{\nu}{2}\Big]\right]. \end{aligned} \tag{9.39}$$

To learn the model, we now alternate between updating the mean and variance $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ of the posterior distribution and updating the hidden variables $\{h_i\}$. To update the mean and variance parameters, we find the solution $\hat{\boldsymbol{\psi}}$ that maximizes

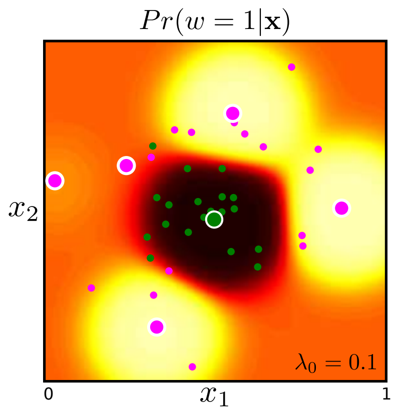

Figure 9.14 Relevance vector regression with RBF kernel. We place a prior over the dual parameters $\boldsymbol{\psi}$ that encourages sparsity. After learning, the posterior distribution over most of the parameters is tightly centered around zero and they can be dropped from the model. Large points indicate data examples associated with non-zero dual parameters. The solution here can be computed from just 6 of the 40 data points but nonetheless classifies the data almost as well as the full kernel approach (figure 9.13).

$$L = \sum_{i=1}^{I} \log\left[\text{Bern}_{w_i}\left[\text{sig}[\boldsymbol{\psi}^T \mathbf{K}[\mathbf{X}, \mathbf{x}_i]]\right]\right] + \log\left[\text{Norm}_{\boldsymbol{\psi}}[0, \mathbf{H}^{-1}]\right] \tag{9.40}$$

using the derivatives

$$\begin{aligned} \frac{\partial L}{\partial \boldsymbol{\psi}} &= -\sum_{i=1}^{I} (\text{sig}[a_i] - w_i)\, \mathbf{K}[\mathbf{X}, \mathbf{x}_i] - \mathbf{H}\psi \\ \frac{\partial^2 L}{\partial \boldsymbol{\psi}^2} &= -\sum_{i=1}^{I} \text{sig}[a_i]\,(1 - \text{sig}[a_i])\, \mathbf{K}[\mathbf{X}, \mathbf{x}_i]\mathbf{K}[\mathbf{x}_i \mathbf{X}] - \mathbf{H}, \end{aligned} \tag{9.41}$$

and then set

$$\begin{aligned} \boldsymbol{\mu} &= \hat{\boldsymbol{\psi}} \\ \boldsymbol{\Sigma} &= -\left(\frac{\partial^2 L}{\partial \boldsymbol{\psi}^2}\right)^{-1}\Bigg|_{\boldsymbol{\psi}=\hat{\boldsymbol{\psi}}}. \end{aligned} \tag{9.42}$$

To update the hidden variables h_i we use the same expression as for relevance vector regression:

$$h_i^{new} = \frac{1 - h_i \Sigma_{ii} + \nu}{\mu_i^2 + \nu}. \tag{9.43}$$

As this optimization proceeds, some of the hidden variables h_i will become very large. This means that the prior over the relevant parameter becomes very concentrated around zero and that the associated data points contribute nothing to the final solution. These can be removed, leaving a kernelized classifier that depends only sparsely on the data and can hence be evaluated very efficiently.

In inference, we aim to compute the distribution over the world state w^* given a new data example $\mathbf{x}^*$. We take the familiar strategy of approximating the posterior distribution over the activation as

$$
\begin{aligned}
Pr(a) = Pr(\boldsymbol{\psi}^T\mathbf{K}[\mathbf{X}^T,\mathbf{x}^*]) &= \text{Norm}_a[\mu_a,\sigma_a^2]r \qquad (9.44)\\
&= \text{Norm}_a[\boldsymbol{\mu}^T\mathbf{K}[\mathbf{X},\mathbf{x}^*],\mathbf{K}[\mathbf{x}^*,\mathbf{X}]\boldsymbol{\Sigma}\mathbf{K}[\mathbf{X},\mathbf{x}^*]],
\end{aligned}
$$

and then approximate the predictive distribution using equation 9.19.

An example of relevance vector classification is shown in figure 9.14, which shows that the data set can be discriminated based on 6 of the original 40 data points. This results in a considerable computational saving and the simpler solution guards against over-fitting of the training set.

9.7 Incremental fitting and boosting

In the previous section, we developed the *relevance vector classification* model in which we applied a prior that encourages sparsity in the dual logistic regression parameters $\boldsymbol{\psi}$ and hence encouraged the model to depend on only a subset of the training data. It is similarly possible to develop a *sparse logistic regression method* by placing a prior that encourages sparsity in the original parameters $\boldsymbol{\phi}$ and hence encourages the classifier to depend only on a subset of the data dimensions. This is left as an exercise to the reader. Algorithm 9.6

In this section we will investigate a different approach to inducing sparsity; we will add one parameter at a time to the model in a greedy fashion; in other words, we add the parameter that improves the objective function most at each stage and then consider this fixed. As the most discriminative parts of the model are added first, it is possible to truncate this process after only a small fraction of the parameters are added and still achieve good results. The remaining, unused parameters can be considered as having a value of zero and so this model also provides a sparse solution. We term this approach *incremental fitting*. We will work with the original formulation (so that the sparsity is over the data dimensions), although these ideas can equally be adapted to the dual case.

To describe the incremental fitting procedure, let us work with the nonlinear formulation of logistic regression (section 9.3) where the probability of the class given the data was described as

$$Pr(w_i|\mathbf{x}_i) = \text{Bern}_{w_i}[\text{sig}[a_i]], \qquad (9.45)$$

where sig[•] is the logistic sigmoid function and the activation a_i is given by

$$a_i = \boldsymbol{\phi}^T\mathbf{z}_i = \boldsymbol{\phi}^T\mathbf{f}[\mathbf{x}_i], \qquad (9.46)$$

and $\mathbf{f}[\bullet]$ is a nonlinear transformation that returns the transformed vector $\mathbf{z}_i$.

To simplify the subsequent description we will now write the activation term in a slightly different way so that the dot product is described explicitly as a weighted sum of individual nonlinear functions of the data

$$a_i = \phi_0 + \sum_{k=1}^{K}\phi_k f[\mathbf{x}_i,\boldsymbol{\xi}_k]. \qquad (9.47)$$

Here $f[\bullet, \bullet]$ is a fixed nonlinear function that takes the data vector $\mathbf{x}_i$ and some parameters $\boldsymbol{\xi}_k$ and returns a scalar value. In other words the k^{th} entry of the transformed vector $\mathbf{z}$ arises by passing the data $\mathbf{x}$ through the function with the k^{th} parameters $\boldsymbol{\xi}_k$. Example functions $f[\bullet, \bullet]$ might include

- radial basis functions, $\boldsymbol{\xi} = \{\boldsymbol{\alpha}, \lambda_0\}$

$$f[\mathbf{x}, \boldsymbol{\xi}] = \exp\left[-\frac{(\mathbf{x}-\boldsymbol{\alpha})^T(\mathbf{x}-\boldsymbol{\alpha})}{\lambda_0^2}\right], \quad \text{and} \tag{9.48}$$

- arc tan functions, $\boldsymbol{\xi} = \{\boldsymbol{\alpha}\}$

$$f[\mathbf{x}, \boldsymbol{\xi}] = \arctan[\boldsymbol{\alpha}^T\mathbf{x}]. \tag{9.49}$$

In incremental learning, we construct the activation term in equation 9.47 piece-wise. At each stage we add a new term, leaving all of the previous terms unchanged except the additive constant ϕ_0. So, at the first stage we use the activation

$$a_i = \phi_0 + \phi_1 f[\mathbf{x}_i, \boldsymbol{\xi}_1] \tag{9.50}$$

and learn the parameters ϕ_0, ϕ_1 and $\boldsymbol{\xi}_1$ using the maximum likelihood approach. At the second stage, we fit the function

$$a_i = \phi_0 + \phi_1 f[\mathbf{x}_i, \boldsymbol{\xi}_1] + \phi_2 f[\mathbf{x}_i, \boldsymbol{\xi}_2] \tag{9.51}$$

and learn the parameters ϕ_0, ϕ_2 and $\boldsymbol{\xi}_2$, while keeping the remaining parameters ϕ_1 and $\boldsymbol{\xi}_1$ constant. At the K^{th} stage, we fit a model with activation

$$a_i = \phi_0 + \sum_{k=1}^{K} \phi_k f[\mathbf{x}_i, \boldsymbol{\xi}_k] \tag{9.52}$$

and learn the parameters ϕ_0, ϕ_K and $\boldsymbol{\xi}_K$, while keeping the remaining parameters $\phi_1 \ldots \phi_{k-1}$ and $\boldsymbol{\xi}_1 \ldots \boldsymbol{\xi}_{k-1}$ constant.

At each stage, the learning is carried out using the maximum likelihood approach. We use a nonlinear optimization procedure to maximize the log posterior probability L with respect to the relevant parameters. The derivatives required by the optimization procedure depend on the choice of nonlinear function, but can be computed using the chain rule relations (equation 9.21).

This procedure is obviously sub-optimal as we do not learn the parameters together or even revisit early parameters once they have been set. However, it has three nice properties.

1. It creates sparse models: the weights ϕ_k tend to decrease as we move through the sequence and each subsequent basis function tends to have less influence on the model. Consequently, the series can be truncated to the desired length and the associated performance is likely to remain good.

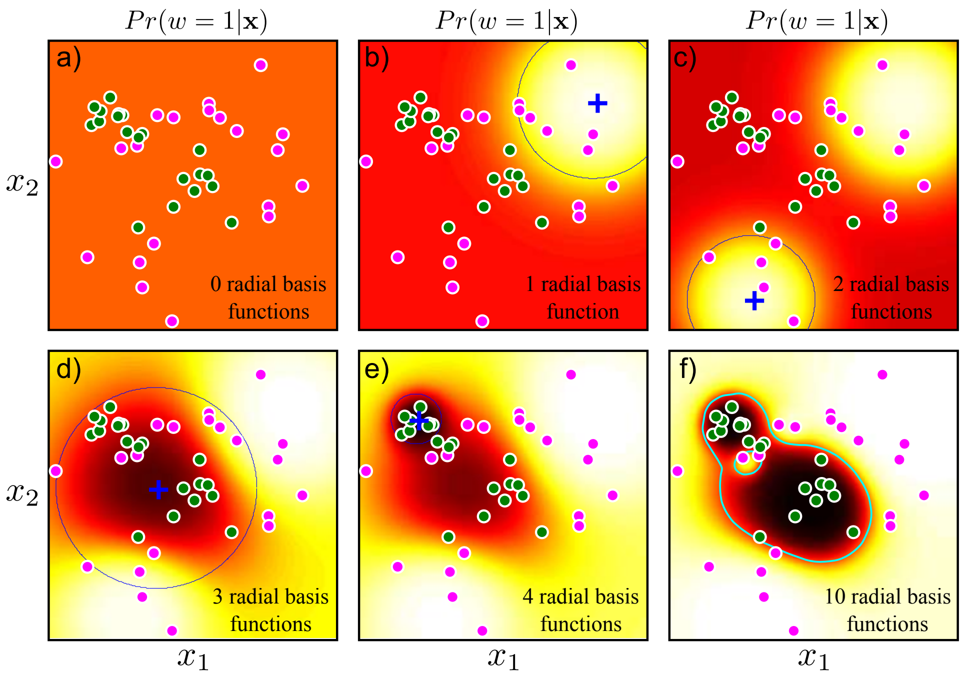

Figure 9.15 Incremental approach to fitting nonlinear logistic regression model with RBF functions. a) Before fitting, the activation (and hence the posterior probability) is uniform. b) Posterior probability after fitting one function (mean and scale of RBF shown in blue). c-e) After fitting two, three and four RBFs. f) After fitting ten RBFs. The data are now all classified correctly as can be seen from the decision boundary (cyan line).

2. The previous logistic regression models have been suited to cases where either the dimensionality D of the data is small (original formulation) or the number of training examples I is small (dual formulation). However, it is quite possible that neither of these things is true. A strong advantage of incremental fitting is that it is still practical when the data are high dimensional *and* there are a large number of training examples. During training, we do not need to hold all of the transformed vectors $\mathbf{z}$ in memory at once: at the K^{th} stage, we need only the K^{th} dimension of the transformed parameters $z_K = f[\mathbf{x}, \boldsymbol{\xi}_K]$ and the aggregate of the previous contributions to activation term $\sum_{k=1}^{K-1} \phi_k f[\mathbf{x}_i, \boldsymbol{\xi}_k]$.

3. Learning is relatively inexpensive because we only optimize a few parameters at each stage.

Figure 9.15 illustrates the incremental approach to learning a 2D data set using radial basis functions. Notice that even after only a few functions have been added to the sequence, the classification is substantially correct. Nonetheless, it is worth

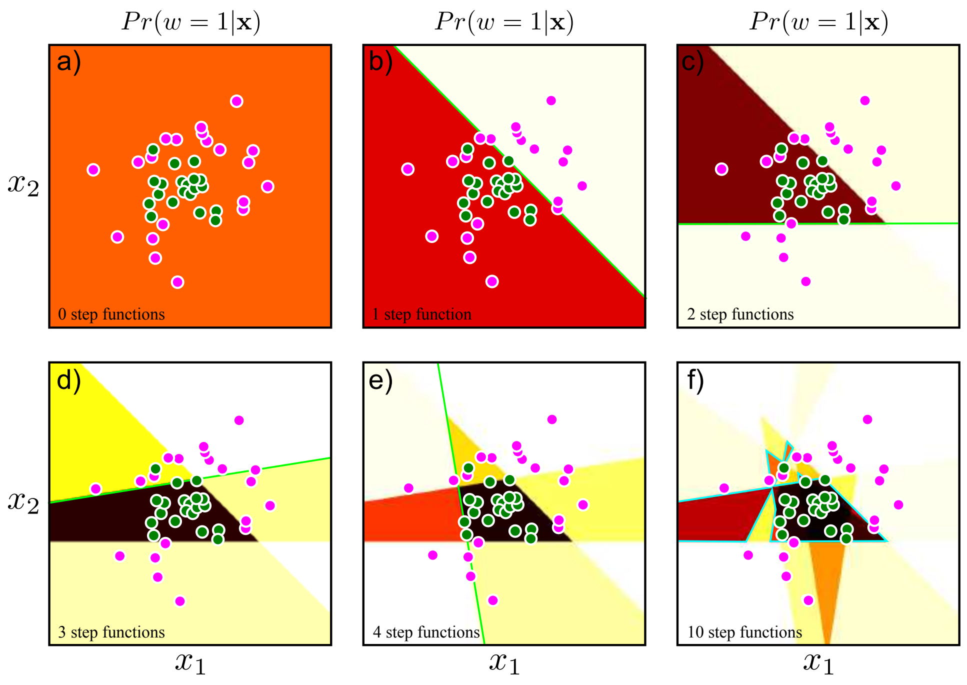

Figure 9.16 Boosting. a) We start with a uniform prediction $Pr(w = 1|\mathbf{x})$ and b) incrementally add a step function to the activation (green line indicates position of step). In this case the parameters of the step function were chosen greedily from a pre-determined set containing 20 angles each with 40 offsets. c)-e) As subsequent functions are added the overall classification improves. f) However, the final decision surface (cyan line) is complex and does not interpolate smoothly between regions of high confidence.

continuing to train this model even after the training data are classified correctly. Usually the model continues to improve and the classification performance on test data will continue to increase for some time.

9.7.1 Boosting

There is a special case of the incremental approach to fitting nonlinear logistic regression that is commonly used in vision applications. Consider a logistic regression model based on a sum of step functions Algorithm 9.7

$$a_i = \phi_0 + \sum_{k=1}^{K} \phi_k \text{heaviside}[\boldsymbol{\alpha}_k^T \mathbf{x}_i], \tag{9.53}$$

where the function heaviside[•] returns 0 if its argument is less than 0 and 1 otherwise. As usual, we have attached a 1 to the start of the data $\mathbf{x}$ so that the parameters $\boldsymbol{\alpha}_k$ contain both a direction $[\alpha_{k1}, \alpha_{k2}, \ldots, \alpha_{kD}]$ in the D dimensional space (which determines the direction of the step function) and an offset α_{k0} (that determines where the step occurs).

One way to think about the step functions is as *weak classifiers*; they return 0 or 1 depending on the value of x_i so each classifies the data. The model combines these weak classifiers to compute a final *strong classifier*. Schemes for combining weak classifiers in this way are generically known as *boosting* and this particular model is called *logitboost*.

Unfortunately, we cannot simply fit this model using a gradient-based optimization approach because the derivative of the heaviside step function with respect to the parameters $\boldsymbol{\alpha}_k$ is not smooth. Consequently it is usual to predefine a large set of J weak classifiers and assume that each parameter vector $\boldsymbol{\alpha}_k$ is taken from this set so that $\boldsymbol{\alpha}_k \in \{\boldsymbol{\alpha}^{(1)} \ldots \boldsymbol{\alpha}^{(J)}\}$.

As before, we learn the logitboost model incrementally by adding one term at a time to the activation (equation 9.53). However, now we exhaustively search over the weak classifiers $\{\boldsymbol{\alpha}^{(1)} \ldots \boldsymbol{\alpha}^{(J)}\}$ and for each, we use nonlinear optimization to estimate the weights ϕ_0 and ϕ_k. We choose the combination $\{\boldsymbol{\alpha}_k, \phi_0, \phi_k\}$ that improves the log likelihood the most. This procedure may be made even more efficient (but more approximate) by choosing the weak classifier based on the log likelihood after just a single Newton or gradient descent step in the nonlinear optimization stage. When we have selected the best weak classifier $\boldsymbol{\alpha}_k$ we can return and perform the full optimization over the offset ϕ_0 and weight ϕ_k.

Note that after each classifier is added, the relative importance of each data point is effectively changed: the data points contribute to the derivative according to how well they are currently predicted (equation 9.9). Consequently, the later weak classifiers become more specialized to the more difficult parts of the data set that are not well classified by the early ones. Usually, these are close to the final decision boundary.

Figure 9.16 shows several iterations of the boosting procedure. Because the model is composed from step functions, the final classification boundary is irregular and does not interpolate smoothly between the data examples. This is a potential disadvantage of this approach. In general, a classifier based on arc tangent functions (which are roughly smooth step functions) will have superior generalization and can also be fit using continuous optimization.

9.8 Classification trees

In the nonlinear logistic regression model, we created complex decision boundaries using an activation function that is a linear combination $\boldsymbol{\phi}^T\mathbf{z}$ of nonlinear functions $\mathbf{z} = \mathbf{f}[\mathbf{x}]$ of the data $\mathbf{x}$. We now investigate an alternative method to induce complex decision boundaries: we partition data space into distinct regions and apply a different classifier in each region.

The *branching logistic regression model* has activation,

$$a_i = (1 - g[\mathbf{x}_i, \boldsymbol{\omega}])\boldsymbol{\phi}_0^T\mathbf{x}_i + g[\mathbf{x}_i, \boldsymbol{\omega}]\boldsymbol{\phi}_1^T\mathbf{x}_i. \tag{9.54}$$

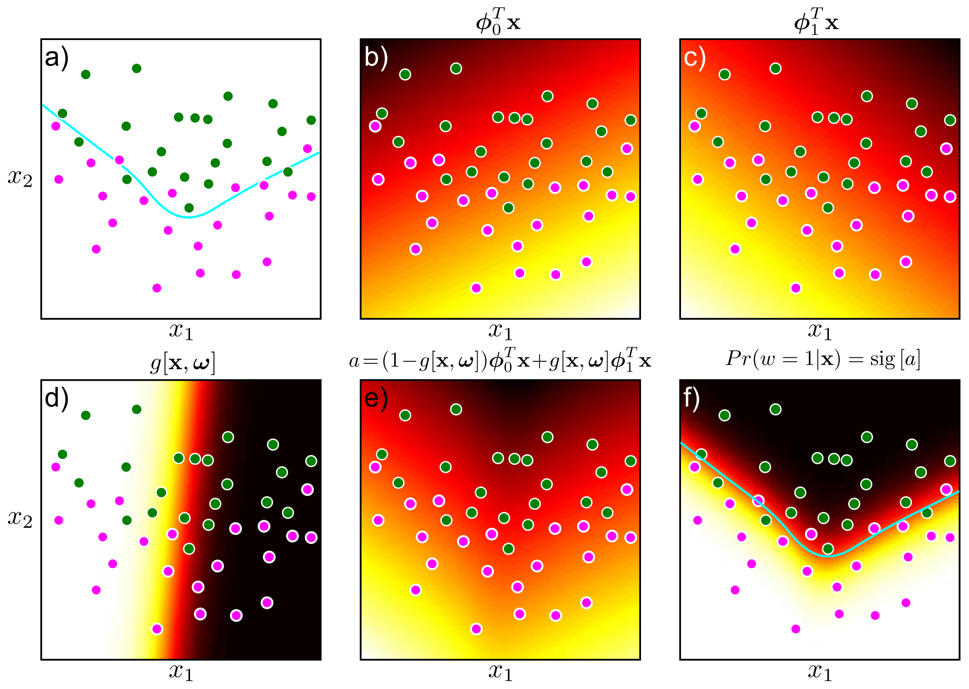

Figure 9.17 Branching logistic regression. a) This data set needs a nonlinear decision surface (cyan line) to classify the data reasonably. b) This linear activation is an *expert* that is specialized to describing the right-hand side of the data. c) This linear activation is an expert that describes the left-hand side of the data. d) A gating function takes the data vector $\mathbf{x}$ and returns a number between 0 and 1, which we will use to decide which expert contributes at each decision. e) The final activation consists of a weighted sum of the activation indicated by the two experts where the weight comes from the gating function. f) The final classifier predictions $Pr(w=1|\mathbf{x})$ are generated by passing this activation through the logistic sigmoid function.

The term $g[\bullet,\bullet]$ is a *gating function* that returns a number between 0 and 1. If this gating function returns 0, then the activation will be $\boldsymbol{\phi}_0\mathbf{x}_i$, whereas if it returns 1, the activation will be $\boldsymbol{\phi}_1\mathbf{x}_i$. If the gating returns an intermediate value, then the activation will be a weighted sum of these two components. The gating function itself depends on the data $\mathbf{x}_i$ and takes parameters $\boldsymbol{\omega}$. This model induces a complex nonlinear decision boundary (figure 9.17) where the two linear functions $\boldsymbol{\phi}_0\mathbf{x}_i$ and $\boldsymbol{\phi}_1\mathbf{x}_i$ are specialized to different regions of the data space. In this context, they are sometimes referred to as *experts*.

The gating function could take many forms, but an obvious possibility is to use a second logistic regression model. In other words, we compute a linear function $\boldsymbol{\omega}^T\mathbf{x}_i$ of the data that is passed through a logistic sigmoid so that

$$g[\mathbf{x}_i,\boldsymbol{\omega}] = \text{sig}[\boldsymbol{\omega}^T\mathbf{x}_i]. \tag{9.55}$$

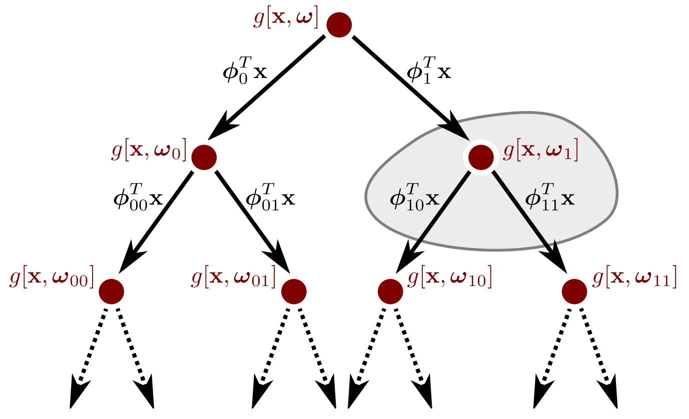
$g[\mathbf{x},\boldsymbol{\omega}]$
$\boldsymbol{\phi}_0^T\mathbf{x}$
$\boldsymbol{\phi}_1^T\mathbf{x}$
$g[\mathbf{x},\boldsymbol{\omega}_0]$
$g[\mathbf{x},\boldsymbol{\omega}_1]$
$\boldsymbol{\phi}_{00}^T\mathbf{x}$
$\boldsymbol{\phi}_{01}^T\mathbf{x}$
$\boldsymbol{\phi}_{10}^T\mathbf{x}$
$\boldsymbol{\phi}_{11}^T\mathbf{x}$
$g[\mathbf{x},\boldsymbol{\omega}_{00}]$
$g[\mathbf{x},\boldsymbol{\omega}_{01}]$
$g[\mathbf{x},\boldsymbol{\omega}_{10}]$
$g[\mathbf{x},\boldsymbol{\omega}_{11}]$

Figure 9.18 Logistic classification tree. Data flows from the root to the leaves. Each node is a gating function that weights the contributions of terms in the sub-branches in the final activation. The gray region indicates variables that would be learned together in an incremental training approach.

To learn this model we maximize the log probability $L = \sum_i \log[Pr(w_i|\mathbf{x}_i)]$ of the training data pairs $\{\mathbf{x}_i, w_i\}_{i=1}^I$ with respect to all of the parameters $\boldsymbol{\theta} = \{\boldsymbol{\phi}_0, \boldsymbol{\phi}_1, \boldsymbol{\omega}\}$. As usual this can be accomplished using a nonlinear optimization procedure. The parameters can be estimated simultaneously or using a coordinate ascent approach in which we alternately update the three sets of parameters.

Problem 9.9

We can extend this idea to create a hierarchical tree structure by nesting gating functions (figure 9.18). For example, consider the activation

$$\begin{aligned} a_i = \; & (1-g[\mathbf{x}_i,\boldsymbol{\omega}])\left[\boldsymbol{\phi}_0^T\mathbf{x}_i + (1-g[\mathbf{x}_i,\boldsymbol{\omega}_0])\boldsymbol{\phi}_{00}^T\mathbf{x}_i + g[\mathbf{x}_i,\boldsymbol{\omega}_0]\boldsymbol{\phi}_{01}^T\mathbf{x}_i\right] \\ & + g[\mathbf{x}_i,\boldsymbol{\omega}]\left[\boldsymbol{\phi}_1^T\mathbf{x}_i + (1-g[\mathbf{x}_i,\boldsymbol{\omega}_1])\boldsymbol{\phi}_{10}^T\mathbf{x}_i + g[\mathbf{x}_i,\boldsymbol{\omega}_1]\boldsymbol{\phi}_{11}^T\mathbf{x}_i\right]. \end{aligned} \tag{9.56}$$

This is an example of a *classification tree.*

To learn the parameters $\boldsymbol{\theta} = \{\boldsymbol{\phi}_0, \boldsymbol{\phi}_1, \boldsymbol{\phi}_{00}, \boldsymbol{\phi}_{01}, \boldsymbol{\phi}_{10}, \boldsymbol{\phi}_{11}, \boldsymbol{\omega}, \boldsymbol{\omega}_0, \boldsymbol{\omega}_1\}$ we could take an incremental approach. At the first stage we fit the top part of the tree (equation 9.54), setting parameters $\boldsymbol{\omega}, \boldsymbol{\phi}_0, \boldsymbol{\phi}_1$. Then we fit the left branch, setting parameters $\boldsymbol{\omega}_0, \boldsymbol{\phi}_{00}, \boldsymbol{\phi}_{01}$ and subsequently the right branch, setting parameters $\boldsymbol{\omega}_1, \boldsymbol{\phi}_{10}, \boldsymbol{\phi}_{11}$ and so on.

The classification tree has the potential advantage of speed. If each gating function produces a binary output (like the heaviside step function), then each data point passes down just one of the outgoing edges from each node and ends up at a single leaf. When each branch in the tree is a linear operation (as in this example), these operations can be aggregated to a single linear operation at each leaf. Since each data point receives specialized processing, the tree need not usually be deep, and new data can be classified very efficiently.

9.9 Multi-class logistic regression

Algorithm 9.8

Throughout this chapter, we have discussed binary classification. We now discuss how to extend these models to handle $N > 2$ world states. One possibility is to build N *one-against-all* binary classifiers each of which computes the probability that the n^{th} class is present as opposed to any of the other classes. The final label is assigned according to the one-against-all classifier with the highest probability.

The one-against-all approach works in practice but is not very elegant. A more principled way to cope with multi-class classification problems is to describe the the posterior $Pr(w|\mathbf{x})$ as a categorical distribution, where the parameters $\boldsymbol{\lambda} = [\lambda_1 \ldots \lambda_N]$ are functions of the data $\mathbf{x}$

$$Pr(w|\mathbf{x}) = \text{Cat}_w[\boldsymbol{\lambda}[\mathbf{x}]], \tag{9.57}$$

where the parameters are in the range $\lambda_n \in [0, 1]$ and sum to one, $\sum_n \lambda_n = 1$. In constructing the function $\boldsymbol{\lambda}[\mathbf{x}]$ we must ensure that we obey these constraints.

As for the two class logistic regression case, we will base the model on linear functions of the data $\mathbf{x}$ and pass these through a function that enforces the constraints. To this end, we define N activations (one for each class),

$$a_n = \boldsymbol{\phi}_n^T \mathbf{x}, \tag{9.58}$$

where $\boldsymbol{\phi}_1 \ldots \boldsymbol{\phi}_N$ are parameter vectors. We assume that as usual we have prepended a 1 to each of the data vectors $\mathbf{x}_i$ so that the first entry of each parameter vectors $\boldsymbol{\phi}_n$ represents an offset. The n^{th} entry of the final categorical distribution is now defined by

$$\lambda_n = \text{softmax}_n[a_1, a_2 \ldots a_N] = \frac{\exp[a_n]}{\sum_{m=1}^{N} \exp[a_m]}. \tag{9.59}$$

Problem 9.10

The function **softmax**[•] takes the N activations $a_1 \ldots a_N$, which can take any real number, and maps them to the N parameters $\lambda_1 \ldots \lambda_N$ of the categorical distribution, which are constrained to be positive and sum to one (figure 9.19).

To learn the parameters $\boldsymbol{\theta} = \{\boldsymbol{\phi}_1 \ldots \boldsymbol{\phi}_N\}$ given training pairs $(w_i, \mathbf{x}_i)$ we optimize the log likelihood of the training data

$$L = \sum_{i=1}^{I} \log\left[Pr(w_i|\mathbf{x}_i)\right]. \tag{9.60}$$

Problem 9.11

As for the two class case, there is no closed form expression for the maximum likelihood parameters. However, this is a convex function, and the maximum can be found using a nonlinear optimization technique such as the Newton method. These techniques require the first and second derivatives of the log likelihood with respect to the parameters, which are given by:

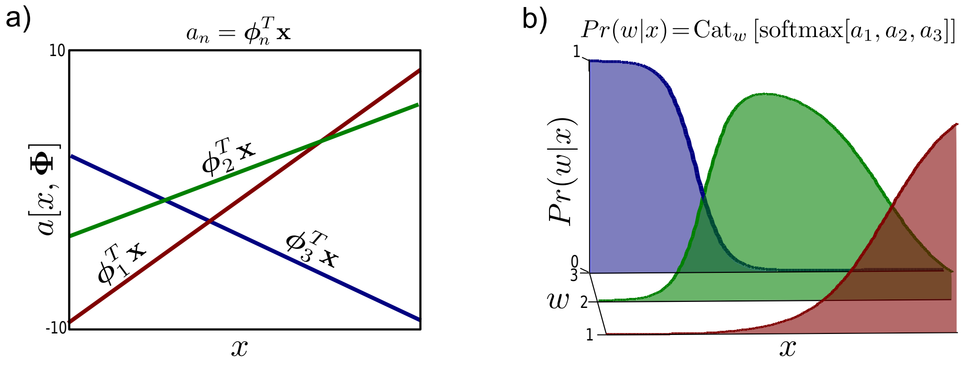

Figure 9.19 Multi-class logistic regression. a) We form one activation for each class based on linear functions of the data. b) We pass these activations through the softmax function to create the distribution $Pr(w|x)$ which is shown here as a function of x. The softmax function takes the three real-valued activations and returns three positive values that sum to one, ensuring that the distribution $Pr(w|x)$ is a valid probability distribution for all x.

$$\begin{aligned}\frac{\partial L}{\partial \boldsymbol{\phi}_n} &= -\sum_{i=1}^{I}(y_{in} - \delta[w_i - n])\mathbf{x}_i \\ \frac{\partial^2 L}{\partial \boldsymbol{\phi}_m \boldsymbol{\phi}_n} &= -\sum_{i=1}^{I} y_{im}(\delta[m-n] - y_{in})\mathbf{x}_i\mathbf{x}_i^T,\end{aligned} \tag{9.61}$$

where we define the term

$$y_{in} = Pr(w_i = n|\mathbf{x}_i) = \text{softmax}_n[a_{i1}, a_{i2} \ldots a_{iN}]. \tag{9.62}$$

It is possible to extend multi-class logistic regression in all of the ways that we extended the two class model. We can construct Bayesian, nonlinear, dual and kernelized versions. It is possible to train incrementally and combine weak classifiers in a boosting framework. Here, we will consider tree-structured models as these are very common in modern vision applications.

9.10 Random trees, forests, and ferns

Algorithm 9.9

In section 9.8 we introduced the idea of tree-structured classifiers, in which the processing for each data example is different and becomes steadily more specialized. This idea has recently become extremely popular for multi-class problems in the form of *random classification trees.*

As for the two-class case, the key idea is to construct a binary tree where at each node, the data are evaluated to determine whether it will pass to the left or the right branch. Unlike in section 9.8, we will assume that each data point passes into just one branch. In a random classification tree, the data are evaluated against a function $q[\mathbf{x}]$ that was randomly chosen from a predefined family of possible functions. For example, this might be the response of a randomly chosen filter. The data proceeds one way in the tree if the response of this function exceeds a threshold τ and the other way if not. While the functions are chosen randomly, the threshold is carefully selected.

We select the threshold that maximizes the log-likelihood L of the data:

$$\begin{aligned} L &= \sum_{i=1}^{I}(1-\text{heaviside}[q[\mathbf{x}_i]-\tau])\log\left[\text{Cat}_{w_i}\left[\boldsymbol{\lambda}^{[l]}\right]\right] \\ &\qquad\qquad +\text{heaviside}[q[\mathbf{x}_i]-\tau]\log\left[\text{Cat}_{w_i}\left[\boldsymbol{\lambda}^{[r]}\right]\right]. \end{aligned} \tag{9.63}$$

Here the first term represents the contribution of the data that passes down the left branch, and the second term represents the contribution of the data that passes down the right branch. In each case, the data are evaluated against a categorical distribution with parameters $\boldsymbol{\lambda}^{[l]}$ and $\boldsymbol{\lambda}^{[r]}$, respectively. These parameters are set using maximum likelihood:

$$\begin{aligned} \lambda_k^{[l]} &= \frac{\sum_{i=1}^{I}\delta[w_i-k](1-\text{heaviside}[q[\mathbf{x}_i]-\tau])}{\sum_{i=1}^{I}(1-\text{heaviside}[q[\mathbf{x}_i]-\tau])} \\ \lambda_k^{[r]} &= \frac{\sum_{i=1}^{I}\delta[w_i-k](\text{heaviside}[q[\mathbf{x}_i]-\tau])}{\sum_{i=1}^{I}(\text{heaviside}[q[\mathbf{x}_i]-\tau])}. \end{aligned} \tag{9.64}$$

The log likelihood is not a smooth function of the threshold τ, and so in practice we maximize the log likelihood by empirically trying a number of different threshold values and choosing the one that gives the best result.

We then perform this same procedure recursively; the data that pass to the left branch has a new randomly chosen classifier applied to them and a new threshold is chosen that splits it again. This can be done without recourse to the data in the right branch. When we classify a new data example $\mathbf{x}^*$, we pass it down the tree until it reaches one of the leaves. The posterior distribution $Pr(w^*|\mathbf{x}^*)$ over the world state w^* is set to $\text{Cat}_{w^*}[\boldsymbol{\lambda}]$ where the parameters $\boldsymbol{\lambda}$ are the categorical parameters associated with this leaf during the training process.

The random classification tree is attractive because it is very fast to train – after all, most of its parameters are chosen randomly. It can also be trained with very large amounts of data as its complexity is linear in the number of data examples.

There are two important variations on this model:

1. A *fern* is a tree where the randomly chosen functions at each level of the tree are constrained to be the same. In other words, the data that pass through the left and right branches at any node are subsequently acted on by the same function (although the threshold level may optionally be different in each branch). In practice, this means that every data point is acted on by the same sequence of functions. This can make implementation extremely efficient when we are evaluating the classifier repeatedly.

2. A *random forest* is a collection of random trees, each of which uses a different randomly chosen set of functions. By averaging together the probabilities $Pr(w^*|\mathbf{x}^*)$ predicted by these trees, a more robust classifier is produced. One way to think of this is as approximating the Bayesian approach; we are constructing the final answer by taking a weighted sum of the predictions suggested by different sets of parameters.

9.11 Relation to non-probabilistic models

In this chapter, we have described a family of probabilistic algorithms for classification. Each is based on maximizing either the log Bernoulli probability of the training class labels given the data (two-class case) or the log categorical probability of the training class labels given the data (multi-class case).

However, it is more common in the computer vision literature to use non-probabilistic classification algorithms such as the multilayer perceptron, adaboost or support vector classification. At their core, these algorithms optimize different objective functions and so are neither directly equivalent to each other, nor to the models in this chapter.

We chose to describe the less common probabilistic algorithms because

- they have no serious disadvantages relative to non-probabilistic techniques,
- they naturally produce estimates of certainty,
- they are easily extensible to the multi-class case whereas non-probabilistic algorithms usually rely on one-against-all formulations, and
- they are more easily related to one another and to the rest of the book.

In short, it can reasonably be argued that the dominance of non-probabilistic approaches to classification is largely for historical reasons. We will now briefly describe the relationship between our models and common non-probabilistic approaches.

The *multi-layer perceptron* or *neural network* is very similar to our nonlinear logistic regression model in the special case where the nonlinear transform consists of a set of sigmoid functions applied to linear projections of data (e.g., $z_k = \arctan[\boldsymbol{\alpha}_k^T\mathbf{x}]$). In the MLP, learning is known as *back propagation* and the transformed variable $\mathbf{z}$ is known as the *hidden layer*.

Adaboost is very closely related to the the logitboost model described in this chapter, but adaboost is not probabilistic. Performance of the two algorithms is similar.

The *support vector machine* (SVM) is similar to relevance vector classification; it is a kernelized classifier that depends sparsely on the data. It has the advantage that its objective function is convex, whereas the objective function in relevance vector classification is non-convex and only guarantees to converge to a local minimum. However, the SVM has several disadvantages: it does not assign certainty to its class predictions, it is not so easily extended to the multi-class case, it produces solutions that are less sparse than relevance vector classification, and it places more restrictions on the form of the kernel function. In practice, classification performance of the two models is again similar.

9.12 Applications

We now present a number of examples of the use of classification in computer vision from the research literature. In many of the examples, the method used was non-probabilistic (e.g., adaboost), but is very closely related to the algorithms in this chapter, and one would not expect the performance to differ significantly if these were substituted.

9.12.1 Gender classification

The algorithms in this chapter were motivated by the problem of gender detection in unconstrained facial images. The goal is to assign a label $w \in \{0, 1\}$ indicating whether a small patch of an image $\mathbf{x}$ contains a male or a female face. Prince & Aghajanian (2009) developed a system of this type. First, a bounding box around the face was identified using a face detector (see next section). The data within this bounding box was resized to 60×60, converted to grayscale and histogram equalized. The resulting image was convolved with a bank of Gabor functions and the filtered images sampled at regular intervals that were proportionate to the wavelength to create a final feature vector of length 1064. Each dimension was whitened to have mean zero and unit standard deviation. Chapter 13 contains information about these and other preprocessing methods.

A training database of 32000 examples was used to learn a nonlinear logistic regression model of the form

$$Pr(w_i|\mathbf{x}_i) = \text{Bern}_{w_i}\left[\frac{1}{1+\exp\left[-\phi_0 - \sum_{k=1}^{K}\phi_k \text{f}[\mathbf{x}_i, \boldsymbol{\xi}_k]\right]}\right], \tag{9.65}$$

where the nonlinear functions f[•] were arc tangents of linear projections of the data so that

$$\text{f}[\mathbf{x}_i, \boldsymbol{\xi}_k] = \arctan[\boldsymbol{\xi}_k^T \mathbf{x}_i]. \tag{9.66}$$

As usual the data were augmented by prepending a 1 so the projection vectors $\{\boldsymbol{\xi}_k\}$ were of length $D+1$. This model was learned using an incremental approach so that at each stage the parameters $\boldsymbol{\phi}_0, \phi_k$ and $\boldsymbol{\xi}_k$ were modified.

The system achieved 87.5% performance with $K = 300$ arc tangent functions on a challenging real-world database that contained large variations in scale, pose, lighting and expression similar to the faces in figure 9.1. Human observers managed only 95% performance on the same database using the resized face region alone.

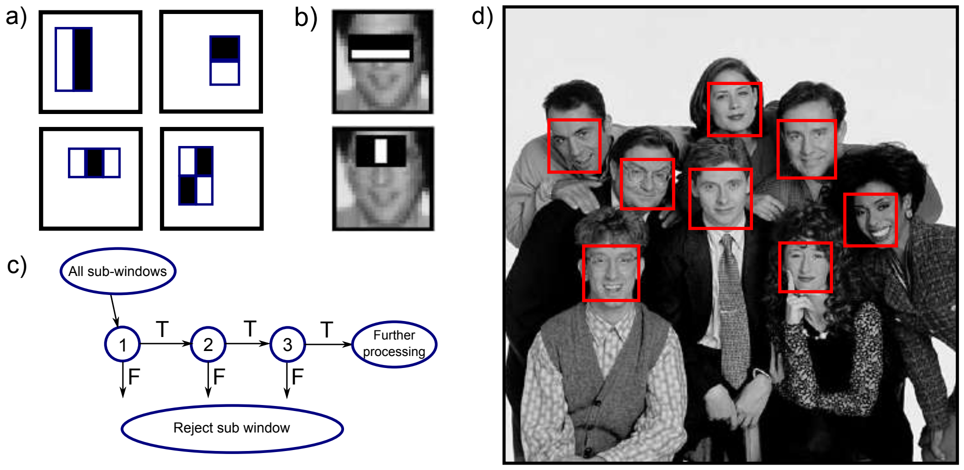

Figure 9.20 Fast face detection using a boosting method (Viola & Jones 2004). a) Each weak classifier consists of the response of the image to a Haar-like filter, which is then passed through a step function. b) The first two weak classifiers learned in this implementation have clear interpretations: the first responds to the dark horizontal region belonging to the eyes and the second responds to the relative brightness of the bridge of the nose. c) The data passes through a cascade: most regions can be quickly rejected after evaluating only a few weak classifiers as they look nothing like faces. More ambiguous regions undergo further preprocessing. d) Example results. Adapted from Viola & Jones (2004).

9.12.2 Face and pedestrian detection

Before we can determine the gender of a face, we must first find it. In face detection (figure 7.1), we assign a label $w \in \{0,1\}$ to a small region of the image $\mathbf{x}$ indicating whether a face is present ($w\!=\!1$) or not ($w\!=\!0$). To ensure that the face is found, this process is repeated at every position and scale in the image and consequently the classifier must be very fast.

Viola & Jones (2004) presented a face detection system based on *adaboost* (figure 9.20). This is a non-probabilistic analogue of the boosting methods described in section 9.7.1. The final classification is based on the sign of a sum of nonlinear functions of the data

$$a = \phi_0 + \sum_{k=1}^{K} \phi_k \mathrm{f}[\mathbf{x}, \boldsymbol{\xi}_k], \tag{9.67}$$

where the nonlinear functions $f[\bullet]$ are heaviside step functions of projections of the data (weak classifiers giving a response of zero or one for each possible data vector $\mathbf{x}$) so that

$$\mathrm{f}[\mathbf{x}, \boldsymbol{\xi}_k] = \mathrm{heaviside}[\boldsymbol{\xi}_k^T \mathbf{x}]. \tag{9.68}$$

As usual, the data vector $\mathbf{x}$ was prepended with a 1 to account for an offset.

The system was trained on 5,000 faces and 10,000 non-face regions, each of which was represented as a 24×24 image patch. Since the model is not smooth (due to the step function), gradient-based optimization is unsuitable, and so Viola & Jones (2004) exhaustively searched through a very large number of pre-defined projections $\boldsymbol{\xi}_k$.

There were two aspects of the design that ensured that the system ran quickly.

1. The structure of the classifier was exploited: training in boosting is incremental – the 'weak classifiers' (nonlinear functions of the data) are incrementally added to create an increasingly sophisticated strong classifier. Viola & Jones (2004) exploited this structure when they ran the classifier: they reject regions that are very unlikely to be faces based on responses of the first few weak classifiers and only subject more ambiguous regions to further processing. This is known as a *cascade* structure. During training, the later stages of the cascade are trained with new negative examples that were not rejected by the earlier stages.

2. The projections $\boldsymbol{\xi}_k$ were carefully chosen so that they were very fast to evaluate: they consisted of Haar-like filters (section 11.4.4) which require only a few operations to compute.

The final system consisted of 4297 weak classifiers divided into a 32 stage cascade. It found 91.1% of 507 frontal faces across 130 images, with a false positive rate of less than 1 per frame, and processed images in fractions of a second.

Viola *et al.* (2005) developed a similar system for detecting pedestrians in video sequences (figure 9.21). The main modification was to extend the set of weak classifiers to encompass features that span more than one frame and hence select for the particular temporal patterns associated with human motion. To this end their system used not only the image data itself, but also the difference image between adjacent frames and similar difference images when taken after offsetting the frames in each of four directions. The final system achieved an 80% detection rate with a false alarm rate of 1/400,000 which corresponds to one false positive for every two frames.

9.12.3 Semantic segmentation

The goal of semantic segmentation is to assign a label $w \in \{1 \ldots M\}$ to each pixel indicating which of M objects is present, based on the local image data $\mathbf{x}$. Shotton *et al.* (2009) developed a system known as *textonboost* that was based on a non-probabilistic boosting algorithm called *jointboost* (Torralba *et al.* 2007). The decision was based on a one-against-all strategy in which M binary classifiers are computed based on the weighted sums

$$a_m = \phi_{0m} + \sum_{k=1}^{K} \phi_{km} \mathrm{f}[\mathbf{x}, \boldsymbol{\xi}_k], \tag{9.69}$$

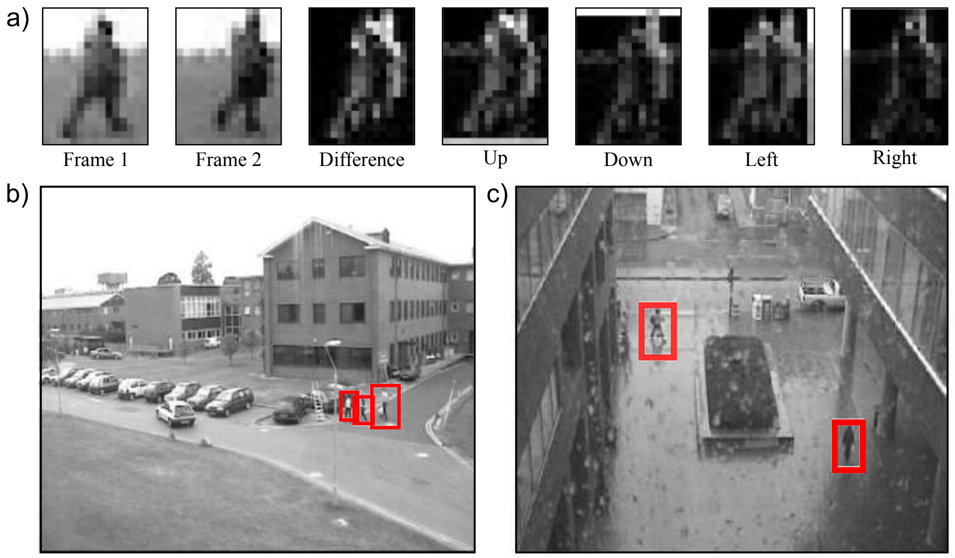

Figure 9.21 Boosting methods based on the thresholded responses of Haar functions have also been used for pedestrian detection in video footage. a) To improve detection rates two subsequent frames are used. The absolute difference between the frames is computed as is the difference when one of the frames is offset in each of four directions. The set of potential weak classifiers consists of Haar functions applied to all six of these representations. b,c) Example results. Adapted from Viola *et al.* (2005). ©2005 Springer.

where the nonlinear functions f[•] were once more based on heaviside step functions. Note that the weighted sums associated with each object class share the same nonlinear functions, but weight them differently. After computing these series, the decision is assigned based on the activation a_m that is the greatest.

Shotton *et al.* (2009) based the nonlinear functions on a *texton* representation of the image: each pixel in the image is replaced by a discrete index indicating the 'type' of texture present at that position (see section 13.1.5). Each nonlinear function considers one of these texton types and computes the number of times that it is found within a rectangular area. This area has a fixed spatial displacement from the pixel under consideration (figure 9.22c-f). If this displacement is zero, then the function provides evidence about the pixel directly (e.g., it looks like grass). If the spatial displacement is larger, then the function provides evidence of the local context (e.g., there is grass nearby, so this pixel may belong to a cow).

For each nonlinear function, an offset is added to the texton count and the result is passed through a step function. The system was learned incrementally by assessing each of a set of randomly chosen classifiers (defined by the choice of texton, rectangular region and offset) and choosing the best at the current stage.

The full system also included a post-processing step in which the result was refined using a conditional random field model (see chapter 12). It achieved 72.2%

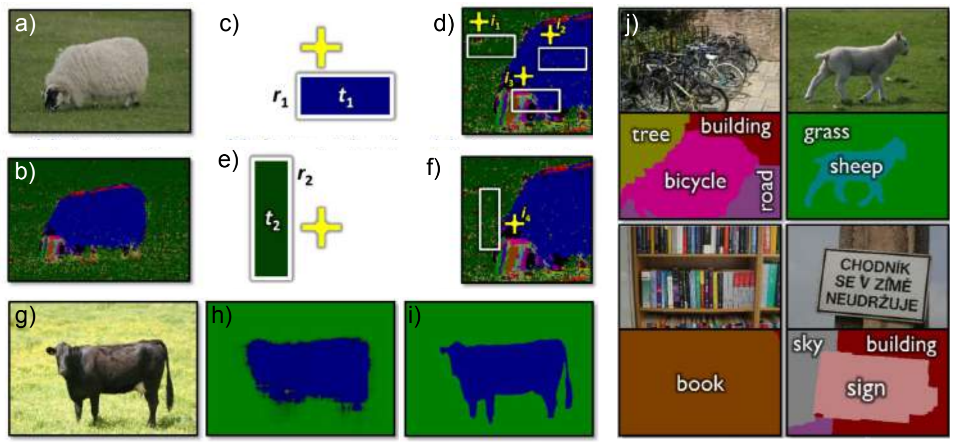

Figure 9.22 Semantic image labeling using "TextonBoost". a) Original image. b) Image converted to textons – a discrete value at each pixel indicating the type of texture that is present. c) The system was based on weak classifiers that count the number of textons of a certain type within a rectangle that is offset from the current position (yellow cross). d) This provides both information about the object itself (contains sheep-like textons) and nearby objects (near to grass-like textons). e,f) Another example of a weak classifier. g) Test image. h) Per-pixel classification is not very precise at the edges of objects and so i) a conditional random field is used to improve the result. j) Examples of results and ground truth. Adapted from Shotton *et al.* (2009) ©2009 Springer.

performance on the challenging MRSC database that includes 21 diverse object classes including wiry objects such as bicycles and objects with a large degree of variation such as dogs.

9.12.4 Recovering surface layout

To recover the *surface layout* of a scene we assign a label $w \in \{1, \ldots 3\}$ to each pixel in the image indicating whether the pixel contains a support object (e.g., floor), a vertical object (e.g., building), or the sky. This decision is based on local image data $\mathbf{x}$. Hoiem *et al.* (2007) constructed a system of this type using a one-against-all principle. Each of the three binary classifiers was based on logitboosted classification trees; different classification trees are treated as weak classifiers and the results are weighted together to compute the final probability.

Hoiem *et al.* (2007) worked with the intermediate representation of *superpixels* – an over-segmentation of the scene into small homogeneous regions, which are assumed to belong to the same object. Each superpixel was assigned a label w using the classifier based on a data vector $\mathbf{x}$, which contained location, appearance, texture and perspective information associated with the superpixel.

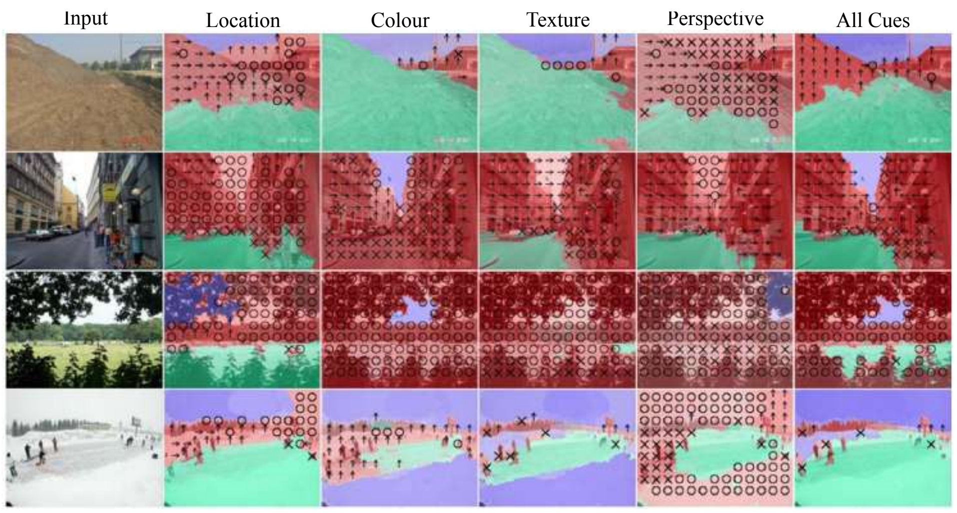

Figure 9.23 Recovering surface layout. The goal is to take an image and return a label indicating whether the pixel is part of a support surface (green pixels) vertical surface (red pixels) or the sky (blue pixels). Vertical surfaces were sub-classified into planar objects at different orientations (left arrows, upward arrows and right arrows denote left-facing, fronto-parrallel and right-facing surfaces) and non-planar objects which can be porous (marked as 'o') or non-porous (marked as 'x'. The final classification was based on (i) location cues (which include position in the image and position relative to the horizon), (ii) color cues, (iii) texture cues, and (iv) perspective cues, which were based on the statistics of line segments in the region. The figure shows example classifications for each of these cues alone and when combined. Adapted from Hoiem *et al.* (2007). ©2007 Springer.

To mitigate against the possibility that the original superpixel segmentation was wrong, multiple segmentations were computed and the results merged to provide a final per-pixel classification (figure 9.23). In the full system, regions that were classified as vertical were sub-classified into left-facing planar surfaces, fronto-parallel planar surfaces, or right-facing planar surfaces, or non-planar surfaces, which may be porous (e.g., trees) or solid. The system was trained and tested on a data set consisting of images collected from the web including diverse environments (forests, cities, roads etc.) and conditions (snowy, sunny, cloudy etc.). The data set was pruned to remove photos where the horizon was not within the image.

The system correctly labeled 88.1% of pixels correctly with respect to the main three classes and 61.5% correctly with respect to the subclasses of the vertical surface. This algorithm was the basis of a remarkable system for creating a 3D model from a single 2D photograph (Hoiem *et al.* 2005).

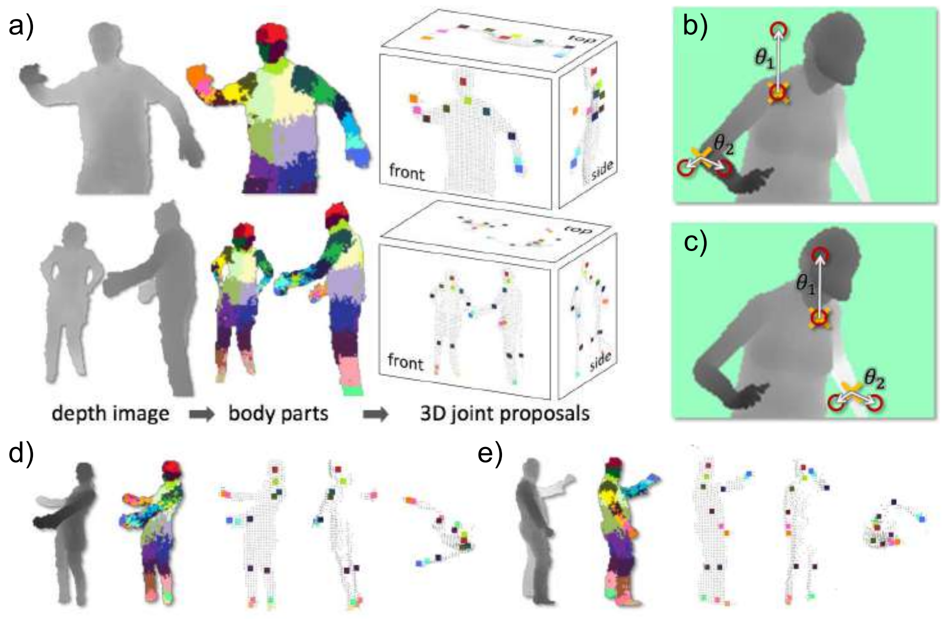

Figure 9.24 Identifying human parts. a) The goal of the system is to take a depth image $\mathbf{x}$ and assign a discrete label w to each pixel indicating which of 31 possible body parts is present. These depth labels are used to form proposals about the position of 3D joints. b) The classification is based on decision trees. At each point in the tree, the data are divided according to the relative depth at two points (red circles) offset relative to the current pixel (yellow crosses). In this example, this difference is large in both cases, whereas in c) this difference is small - hence these differences provide information about the pose. d,e) Two more examples of depth image, labeling and hypothesized pose. Adapted from Shotton *et al.* (2011) ©2011 IEEE

.

9.12.5 Identifying human parts

Shotton *et al.* (2011) describe a system that assigns a discrete label $w \in \{1, \ldots, 31\}$, indicating which of 31 body parts is present at each pixel based on a depth image $\mathbf{x}$. The resulting distribution of labels is an intermediate representation in a system that proposes a possible configuration of the 3D joint positions in the Microsoft Kinect gaming system (figure 9.24).

The classification was based on a *forest* of decision trees: the final probability $Pr(w|\mathbf{x})$ is an average (i.e., a mixture) of the predictions from a number of different classification trees. The goal is to mitigate against biases introduced by the greedy method with which a single tree is trained.

Within each tree, the decision about which branch a data point travels down is based on the difference in measured depths at two points, each of which is spatially offset from the current pixel. The offsets are inversely scaled by the distance to the pixel itself, which ensures that they address the same relative positions on the body when the person moves closer or further away to the depth camera.

The system was trained from a very large data set of 900,000 depth images which were synthesized based on motion capture data and consisted of three trees of depth 20. Remarkably, the system is capable of assigning the correct label 59%, of the time and this provides a very solid basis for the subsequent joint proposals.

Discussion

In this chapter we have considered classification problems. We note that all of the ideas that were applied to regression models in chapter 8 are also applicable to classification problems. However, for classification the model includes a nonlinear mapping between the data $\mathbf{x}$ and the parameters of the distribution $Pr(w|\mathbf{x})$ over the world w. This means that we cannot find the maximum likelihood solution in closed form (although the problem is still convex) and we cannot compute a full Bayesian solution without making approximations.

Classification techniques have many uses in machine vision. Notice though that these models have no domain specific information about the problem other than that provided by the preprocessing of the data. This is both an advantage (they find many applications) and a disadvantage (they cannot take advantage of a priori information about the problem). In the remaining part of the book we will explore models that introduce increasing amounts of domain specific information to the problem.

Notes

Classification in vision: Classification techniques such as those discussed in this chapter have been applied to many problems in vision including face detection (Viola & Jones 2004), surface layout estimation (Hoiem *et al.* 2007), boundary detection (Dollár *et al.* 2006), keypoint matching (Lepetit *et al.* 2005), body part classification (Hoiem *et al.* 2007), semantic segmentation (He *et al.* 2004), object recognition (Csurka *et al.* 2004), and gender classification (Kumar *et al.* 2008).

Probabilistic classification: More information about logistic regression can be found in Bishop (2006) and many other statistics textbooks. Kernel logistic regression (or Gaussian process regression) was presented in Williams & Barber (1998), and more information can be found in Rasmussen & Williams (2006). A sparse version of kernel logistic regression (relevance vector classification) was presented by Tipping (2001) and a sparse multi-class variant was developed by Brishnapuram *et al.* (2005). Probabilistic interpretations of boosting were introduced by Friedman *et al.* (2000). Random forests of multinomial regressors were introduced in Prinzie & Van den Poel (2008).

Other classification schemes: In this chapter, we have presented a family of probabilistic classification models based on logistic regression. There are other non-probabilistic techniques for classification, and these include single and multi-layer perceptrons (Rosenblatt 1958; Rumelhart *et al.* 1986), support vector machines (Vapnik 1995; Cristianini & Shawe-Taylor 2000), and adaboost (Freund & Schapire 1995). A critical difference between these techniques is the underlying objective function. Logistic regression models optimize the log Bernoulli probability, but the other models optimize different criteria, such as the hinge loss (support vector machines) or exponential error (adaboost). It is difficult to make general statements about the relative merits of these approaches, but it is probably fair to say that (i) there is no major disadvantage to using the probabilistic techniques in this chapter and (ii) the choice of classification method is usually less important in vision problems than the preprocessing of the data. Methods based on boosting and classification trees are particularly popular in vision because of their speed.

Boosting: Adaboost was introduced by Freund & Schapire (1995). Since then there have been a large number of variations, most of which have been used in computer vision. These include discrete adaboost (Freund & Schapire 1996), real adaboost (Schapire & Singer 1998), gentleboost (Friedman *et al.* 2000), logitboost (Friedman *et al.* 2000), floatboost (Li *et al.* 2003), KLBoost (Liu & Shum 2003), asymmetric boost (Viola & Jones 2002), and statboost (Pham & Chan 2007a). Boosting has also been applied to the multi-class case (Schapire & Singer 1998; Torralba *et al.* 2007) and for regression (Friedman 1999). A review of boosting approaches can be found in Meir & Mätsch (2003).

Classification trees: Classification trees have a long history in computer vision, dating back to at least Shepherd (1983). Modern interest was stimulated by Amit & Geman (1997) and Breiman (2001) who investigated the use of random forests. Since this time classification trees and forests have been applied to keypoint matching (Lepetit *et al.* 2005), segmentation (Yin *et al.* 2007), human pose detection (Rogez *et al.* 2006; Shotton *et al.* 2011), object detection (Bosch *et al.* 2007), image classification (Moosmann *et al.* 2006; Moosmann *et al.* 2008), deciding image suitability (Mac Aodha *et al.* 2010), detecting occlusions (Humayun *et al.* 2011), and semantic image segmentation (Shotton *et al.* 2009).

Gender classification: Automatic determination of gender from a facial image has variously been tackled with neural networks (Golomb *et al.* 1990), support vector machines (Moghaddam & Yang 2002), linear discriminant analysis (Bekios-Calfa *et al.* 2011) and both adaboost (Baluja & Rowley 2003), and logitboost (Prince & Aghajanian 2009). A review is provided by Mäkinen & Raisamo (2008b) and quantative comparisons are presented in Mäkinen & Raisamo (2008a). Representative examples of the state of the art can be found in Kumar *et al.* (2008) and Shan (in press).

Face detection: The application of boosting to face detection (Viola & Jones 2004) usurped earlier techniques (e.g., Osuna *et al.* 1997; Schneiderman & Kanade 2000). Since then, many boosting variants have been applied to the problem including floatboost (Li *et al.* 2002; Li & Zhang 2004), gentleboost (Lienhart *et al.* 2003), realboost (Huang *et al.* 2007a; Wu *et al.* 2007), asymboost (Pham & Chan 2007b; Viola & Jones 2002) and statboost (Pham & Chan 2007a). A recent review of this area can be found in Zhang & Zhang (2010).

Semantic segmentation: The authors of the system described in the text (Shotton *et al.* 2008b) subsequently presented a much faster system based on classification trees (Shotton *et al.* 2009). A recent comparison of quantitative performance can be found in Ranganathan (2009). Other work has investigated the imposition of prior knowledge such as the co-presence of object classes (He *et al.* 2006) and likely spatial configurations of objects (He *et al.* 2004).

Problems

Problem 9.1 The logistic sigmoid function is defined as

$$\text{sig}[a] = \frac{1}{1+\exp[-a]}.$$

Show that (i) sig$[-\infty] = 0$, (ii) sig$[0] = 0.5$, (iii) sig$[\infty] = 1$.

Problem 9.2 Show that the derivative of the log posterior probability for the logistic regression model

$$L = \sum_{i=1}^{I} w_i \log\left[\frac{1}{1+\exp[-\boldsymbol{\phi}^T\mathbf{x}_i]}\right] + \sum_{i=1}^{I}(1-w_i)\log\left[\frac{\exp[-\boldsymbol{\phi}^T\mathbf{x}_i]}{1+\exp[-\boldsymbol{\phi}^T\mathbf{x}_i]}\right]$$

with respect to the parameters $\boldsymbol{\phi}$ is given by

$$\frac{\partial L}{\partial \boldsymbol{\phi}} = -\sum_{i=1}^{I}\left(\text{sig}[a_i] - w_i\right)\mathbf{x}_i.$$

Problem 9.3 Show that the second derivative of the log likelihood of the logistic regression model is given by

$$\frac{\partial^2 L}{\partial \boldsymbol{\phi}^2} = -\sum_{i=1}^{I}\text{sig}[a_i](1-\text{sig}[a_i])\mathbf{x}_i\mathbf{x}_i^T.$$

Problem 9.4 Consider fitting a logistic regression model to 1D data x where the two classes are perfectly separable. For example, perhaps all the data x where the world state $w = 0$ takes values less than 0 and all the data x where the world state is $w = 1$ takes

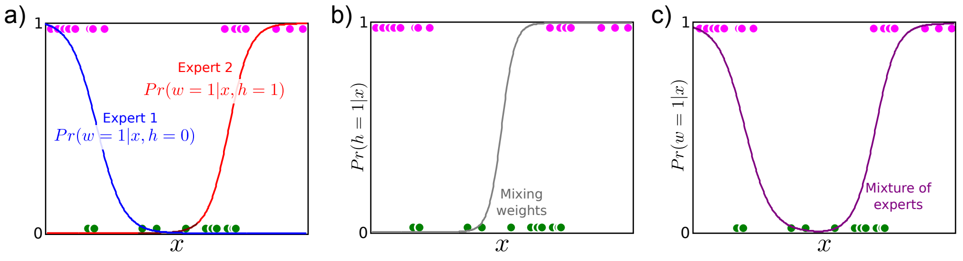

Figure 9.25 Mixture of two experts model for 1D data. Pink circles indicate positive examples. Green circles indicate negative examples. a) Two expert is specialized to model the left and right sides of the data, respectively. b) The mixing weights change as a function of the data. c) The final output of the model is mixture of the two constituent experts and fits the data well.

values greater than 1. Hence it is possible to classify the training data perfectly. What will happen to the parameters of the model during learning? How could you rectify this problem?

Problem 9.5 Compute the Laplace approximation to a beta distribution with parameters $\alpha = 1.0$, $\beta = 1.0$.

Problem 9.6 Show that the Laplace approximation to a univariate normal distribution with mean μ and variance σ^2 is the normal distribution itself.

Problem 9.7 Show that the second derivative of the logarithm $L = \log[\text{Norm}_{\mathbf{x}}[\boldsymbol{\mu}, \boldsymbol{\Sigma}]]$ of a normal distribution evaluated at the mean $\boldsymbol{\mu}$ is given by

$$\left|\frac{\partial L}{\partial \phi^2}\right|_{\boldsymbol{\mu}} = -\boldsymbol{\Sigma}^{-1}.$$

Problem 9.8 Devise a method to choose the scale parameter λ_0 in the radial basis function in kernel logistic regression (equation 9.33).

Problem 9.9 A *mixture of experts* (Jordan & Jacobs 1994) divides space into different regions, each of which receives specialized attention (figure 9.25) . For example, we could describe the data as a mixture of logistic classifiers so that

$$Pr(w_i|\mathbf{x}_i) = \sum_{k=1}^{K} \lambda_k[\mathbf{x}_i]\text{Bern}_{w_i}\left[\text{sig}[\boldsymbol{\phi}_k^T\mathbf{x}_i]\right].$$

Each logistic classifier is considered as an expert and the mixing weights decide the combination of experts that are applied to the data. The mixing weights, which are positive and sum to one, depend on the data $\mathbf{x}$: for a two-component model they could be based on a second logistic regression model with activation $\boldsymbol{\omega}^T\mathbf{x}$. This model can be expressed as the marginalization of a joint distribution between $\mathbf{w}_i$ and a hidden variable h_i so that

$$Pr(w_i|\mathbf{x}_i) = \sum_{k=1}^{K} Pr(w_i, h_i = k|\mathbf{x}_i) = \sum_{k=1}^{K} Pr(w_i|h_i = k, \mathbf{x}_i)Pr(h_i = k|\mathbf{x}_i),$$

where

$$\begin{aligned} Pr(w_i|h_i = k, \mathbf{x}_i) &= \text{Bern}_{w_i}\left[\text{sig}[\boldsymbol{\phi}_k^T \mathbf{x}_i]\right] \\ Pr(h_i = k|\mathbf{x}_i) &= \text{Bern}_{h_i}\left[\text{sig}[\boldsymbol{\omega}^T \mathbf{x}_i]\right]. \end{aligned}$$

How does this model differ from branching logistic regression (section 9.8)? Devise a learning algorithm for this model.

Problem 9.10 The **softmax**$[\bullet, \bullet, \ldots, \bullet]$ function is defined to return a multivariate quantity where the k^{th} element is given by

$$s_k = \text{softmax}_k[a_1, a_2, \ldots a_K] = \frac{\exp[a_k]}{\sum_{j=1}^{K} \exp[a_j]}.$$

Show that $0 < s_k < 1$ and that $\sum_{k=1}^{K} s_k = 1$.

Problem 9.11 Show that the first derivative of the log-probability of the multi-class logistic regression model is given by equation 9.61.

Problem 9.12 The classifiers in this chapter have all been based on continuous data $\mathbf{x}$. Devise a model that can distinguish between M world states $w \in \{1 \ldots M\}$ based on a discrete observation $x \in \{1 \ldots K\}$ and discuss potential learning algorithms.

Part III

Connecting local models

Part III: Connecting local models

The models in chapters 6-9 describe the relationship between a set of measurements and the world state. They work well when the measurements and the world state are both low dimensional. However, there are many situations where this is not the case, and these models are unsuitable.

For example, consider the semantic image labeling problem in which we wish to assign a label to each pixel in the image where the label denotes the object class. For example, in a road scene we might wish to label pixels as 'road', 'sky', 'car', 'tree', 'building' or 'other'. For an image with $N = 10000$ pixels, this means we need to build a model relating the 10000 measured RGB triples to 6^{10000} possible world states. None of the models discussed so far can cope with this challenge: the number of parameters involved (and hence the amount of training data and the computational requirements of the learning and inference algorithms) is far beyond what current machines can handle.

One possible solution to this problem would be to build a set of independent local models: for example, we could build models that relate each pixel label separately to the nearby RGB data. However, this is not ideal as the image may be locally ambiguous. For example, a small blue image patch might result from a variety of semantically different classes: sky, water, a car door or a person's clothing. In general, it is insufficient to build independent local models.

The solution to this problem is to build local models that are *connected* to one another. Consider again the semantic labeling example: given the whole image, we can see that when the image patch is blue and is found above trees and mountains and alongside similar patches across the top of the image, then the correct class is probably sky. Hence, to solve this problem, we still model the relationship between the label and its local image region, but we also connect these models so that nearby elements can help to disambiguate one another.

In chapter 10 we introduce the idea of conditional independence, which is a way of characterizing redundancies in the model (i.e., the lack of direct dependence between variables). We show how conditional independence relations can be visualized with graphical models. We distinguish between directed and undirected graphical models. In chapter 11, we discuss models in which the local units are combined together to form chains or trees. In chapter 12, we extend this to the case where they have more general connections.