8 *Logistic regression*

8.1 Introduction

One way to build a probabilistic classifier is to create a joint model of the form $p(y, \mathbf{x})$ and then to condition on $\mathbf{x}$, thereby deriving $p(y|\mathbf{x})$. This is called the generative approach. An alternative approach is to fit a model of the form $p(y|\mathbf{x})$ directly. This is called the **discriminative** approach, and is the approach we adopt in this chapter. In particular, we will assume discriminative models which are linear in the parameters. This will turn out to significantly simplify model fitting, as we will see. In Section 8.6, we compare the generative and discriminative approaches, and in later chapters, we will consider non-linear and non-parametric discriminative models.

8.2 Model specification

As we discussed in Section 1.4.6, logistic regression corresponds to the following binary classification model:

$$p(y|\mathbf{x}, \mathbf{w}) = \mathrm{Ber}(y|\mathrm{sigm}(\mathbf{w}^T\mathbf{x})) \tag{8.1}$$

A 1d example is shown in Figure 1.19(b). Logistic regression can easily be extended to higher-dimensional inputs. For example, Figure 8.1 shows plots of $p(y = 1|\mathbf{x}, \mathbf{w}) = \mathrm{sigm}(\mathbf{w}^T\mathbf{x})$ for 2d input and different weight vectors $\mathbf{w}$. If we threshold these probabilities at 0.5, we induce a linear decision boundary, whose normal (perpendicular) is given by $\mathbf{w}$.

8.3 Model fitting

In this section, we discuss algorithms for estimating the parameters of a logistic regression model.

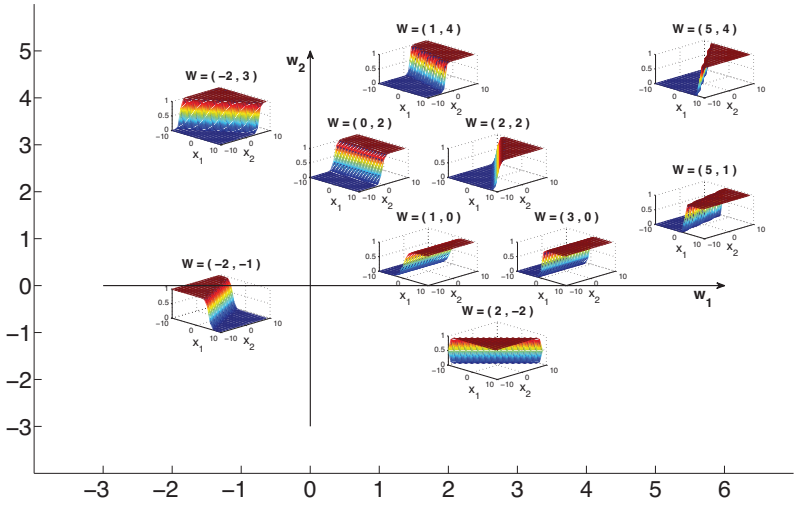


Figure 8.1 Plots of $\text{sigm}(w_1 x_1 + w_2 x_2)$. Here $\mathbf{w} = (w_1, w_2)$ defines the normal to the decision boundary. Points to the right of this have $\text{sigm}(\mathbf{w}^T \mathbf{x}) > 0.5$, and points to the left have $\text{sigm}(\mathbf{w}^T \mathbf{x}) < 0.5$. Based on Figure 39.3 of (MacKay 2003). Figure generated by `sigmoidplot2D`.

8.3.1 MLE

The negative log-likelihood for logistic regression is given by

$$\text{NLL}(\mathbf{w}) = -\sum_{i=1}^{N} \log[\mu_i^{\mathbb{I}(y_i=1)} \times (1-\mu_i)^{\mathbb{I}(y_i=0)}] \tag{8.2}$$

$$= -\sum_{i=1}^{N} [y_i \log \mu_i + (1-y_i) \log(1-\mu_i)] \tag{8.3}$$

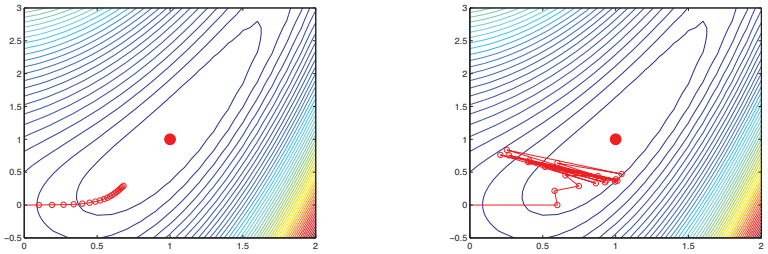
This is also called the **cross-entropy** error function (see Section 2.8.2).

Another way of writing this is as follows. Suppose $\tilde{y}_i \in \{-1, +1\}$ instead of $y_i \in \{0, 1\}$. We have $p(y=1) = \frac{1}{1+\exp(-\mathbf{w}^T \mathbf{x})}$ and $p(y=1) = \frac{1}{1+\exp(+\mathbf{w}^T \mathbf{x})}$. Hence

$$NLL(\mathbf{w}) = \sum_{i=1}^{N} \log(1 + \exp(-\tilde{y}_i \mathbf{w}^T \mathbf{x}_i)) \tag{8.4}$$

Unlike linear regression, we can no longer write down the MLE in closed form. Instead, we need to use an optimization algorithm to compute it. For this, we need to derive the gradient and Hessian.

In the case of logistic regression, one can show (Exercise 8.3) that the gradient and Hessian



(a) (b)

Figure 8.2 Gradient descent on a simple function, starting from $(0, 0)$, for 20 steps, using a fixed learning rate (step size) η. The global minimum is at $(1, 1)$. (a) $\eta = 0.1$. (b) $\eta = 0.6$. Figure generated by `steepestDescentDemo`.

of this are given by the following

$$\mathbf{g} = \frac{d}{d\mathbf{w}} f(\mathbf{w}) = \sum_i (\mu_i - y_i)\mathbf{x}_i = \mathbf{X}^T(\boldsymbol{\mu} - \mathbf{y}) \tag{8.5}$$

$$\mathbf{H} = \frac{d}{d\mathbf{w}} \mathbf{g}(\mathbf{w})^T = \sum_i (\nabla_{\mathbf{w}} \mu_i)\mathbf{x}_i^T = \sum_i \mu_i(1 - \mu_i)\mathbf{x}_i\mathbf{x}_i^T \tag{8.6}$$

$$= \mathbf{X}^T\mathbf{S}\mathbf{X} \tag{8.7}$$

where $\mathbf{S} \triangleq \text{diag}(\mu_i(1 - \mu_i))$. One can also show (Exercise 8.3) that $\mathbf{H}$ is positive definite. Hence the NLL is convex and has a unique global minimum. Below we discuss some methods for finding this minimum.

8.3.2 Steepest descent

Perhaps the simplest algorithm for unconstrained optimization is **gradient descent**, also known as **steepest descent**. This can be written as follows:

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \eta_k \mathbf{g}_k \tag{8.8}$$

where η_k is the **step size** or **learning rate**. The main issue in gradient descent is: how should we set the step size? This turns out to be quite tricky. If we use a constant learning rate, but make it too small, convergence will be very slow, but if we make it too large, the method can fail to converge at all. This is illustrated in Figure 8.2. where we plot the following (convex) function

$$f(\boldsymbol{\theta}) = 0.5(\theta_1^2 - \theta_2)^2 + 0.5(\theta_1 - 1)^2, \tag{8.9}$$

We arbitrarily decide to start from $(0, 0)$. In Figure 8.2(a), we use a fixed step size of $\eta = 0.1$; we see that it moves slowly along the valley. In Figure 8.2(b), we use a fixed step size of $\eta = 0.6$; we see that the algorithm starts oscillating up and down the sides of the valley and never converges to the optimum.

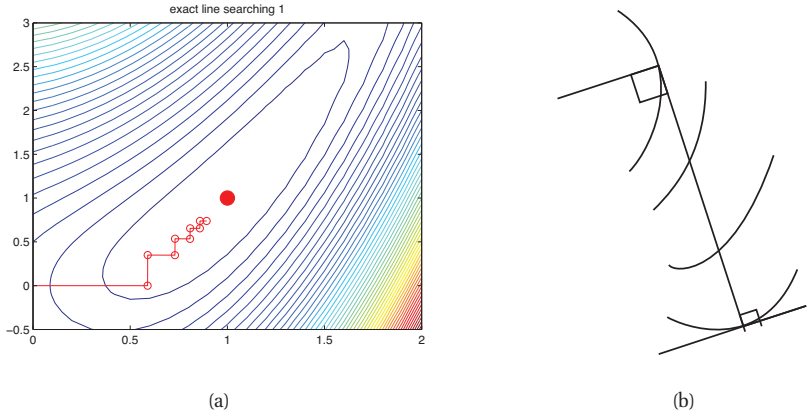


Figure 8.3 (a) Steepest descent on the same function as Figure 8.2, starting from $(0, 0)$, using line search. Figure generated by `steepestDescentDemo`. (b) Illustration of the fact that at the end of a line search (top of picture), the local gradient of the function will be perpendicular to the search direction. Based on Figure 10.6.1 of (Press et al. 1988).

Let us develop a more stable method for picking the step size, so that the method is guaranteed to converge to a local optimum no matter where we start. (This property is called **global convergence**, which should not be confused with convergence to the global optimum!) By Taylor's theorem, we have

$$f(\boldsymbol{\theta} + \eta \mathbf{d}) \approx f(\boldsymbol{\theta}) + \eta \mathbf{g}^T \mathbf{d} \tag{8.10}$$

where $\mathbf{d}$ is our descent direction. So if η is chosen small enough, then $f(\boldsymbol{\theta}+\eta\mathbf{d}) < f(\boldsymbol{\theta})$, since the gradient will be negative. But we don't want to choose the step size η too small, or we will move very slowly and may not reach the minimum. So let us pick η to minimize

$$\phi(\eta) = f(\boldsymbol{\theta}_k + \eta \mathbf{d}_k) \tag{8.11}$$

This is called **line minimization** or **line search**. There are various methods for solving this 1d optimization problem; see (Nocedal and Wright 2006) for details.

Figure 8.3(a) demonstrates that line search does indeed work for our simple problem. However, we see that the steepest descent path with exact line searches exhibits a characteristic **zig-zag** behavior. To see why, note that an exact line search satisfies $\eta_k = \arg\min_{\eta>0} \phi(\eta)$. A necessary condition for the optimum is $\phi'(\eta) = 0$. By the chain rule, $\phi'(\eta) = \mathbf{d}^T\mathbf{g}$, where $\mathbf{g} = f'(\boldsymbol{\theta} + \eta\mathbf{d})$ is the gradient at the end of the step. So we either have $\mathbf{g} = \mathbf{0}$, which means we have found a stationary point, or $\mathbf{g} \perp \mathbf{d}$, which means that exact search stops at a point where the local gradient is perpendicular to the search direction. Hence consecutive directions will be orthogonal (see Figure 8.3(b)). This explains the zig-zag behavior.

One simple heuristic to reduce the effect of zig-zagging is to add a **momentum** term, $(\boldsymbol{\theta}_k - \boldsymbol{\theta}_{k-1})$, as follows:

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \eta_k \mathbf{g}_k + \mu_k(\boldsymbol{\theta}_k - \boldsymbol{\theta}_{k-1}) \tag{8.12}$$

where $0 \leq \mu_k \leq 1$ controls the importance of the momentum term. In the optimization community, this is known as the **heavy ball method** (see e.g., (Bertsekas 1999)).

An alternative way to minimize "zig-zagging" is to use the method of **conjugate gradients** (see e.g., (Nocedal and Wright 2006, ch 5) or (Golub and van Loan 1996, Sec 10.2)). This is the method of choice for quadratic objectives of the form $f(\boldsymbol{\theta}) = \boldsymbol{\theta}^T \mathbf{A} \boldsymbol{\theta}$, which arise when solving linear systems. However, non-linear CG is less popular.

8.3.3 Newton's method

Algorithm 8.1: Newton's method for minimizing a strictly convex function

1 Initialize $\boldsymbol{\theta}_0$;
2 **for** $k = 1, 2, \ldots$ *until convergence* **do**
3 Evaluate $\mathbf{g}_k = \nabla f(\boldsymbol{\theta}_k)$;
4 Evaluate $\mathbf{H}_k = \nabla^2 f(\boldsymbol{\theta}_k)$;
5 Solve $\mathbf{H}_k \mathbf{d}_k = -\mathbf{g}_k$ for $\mathbf{d}_k$;
6 Use line search to find stepsize η_k along $\mathbf{d}_k$;
7 $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \eta_k \mathbf{d}_k$;

One can derive faster optimization methods by taking the curvature of the space (i.e., the Hessian) into account. These are called **second order** optimization metods. The primary example is **Newton's algorithm**. This is an iterative algorithm which consists of updates of the form

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \eta_k \mathbf{H}_k^{-1} \mathbf{g}_k \tag{8.13}$$

The full pseudo-code is given in Algorithm 2.

This algorithm can be derived as follows. Consider making a second-order Taylor series approximation of $f(\boldsymbol{\theta})$ around $\boldsymbol{\theta}_k$:

$$f_{quad}(\boldsymbol{\theta}) = f_k + \mathbf{g}_k^T (\boldsymbol{\theta} - \boldsymbol{\theta}_k) + \frac{1}{2} (\boldsymbol{\theta} - \boldsymbol{\theta}_k)^T \mathbf{H}_k (\boldsymbol{\theta} - \boldsymbol{\theta}_k) \tag{8.14}$$

Let us rewrite this as

$$f_{quad}(\boldsymbol{\theta}) = \boldsymbol{\theta}^T \mathbf{A} \boldsymbol{\theta} + \mathbf{b}^T \boldsymbol{\theta} + c \tag{8.15}$$

where

$$\mathbf{A} = \frac{1}{2} \mathbf{H}_k, \;\; \mathbf{b} = \mathbf{g}_k - \mathbf{H}_k \boldsymbol{\theta}_k, \;\; c = f_k - \mathbf{g}_k^T \boldsymbol{\theta}_k + \frac{1}{2} \boldsymbol{\theta}_k^T \mathbf{H}_k \boldsymbol{\theta}_k \tag{8.16}$$

The minimum of f_{quad} is at

$$\boldsymbol{\theta} = -\frac{1}{2} \mathbf{A}^{-1} \mathbf{b} = \boldsymbol{\theta}_k - \mathbf{H}_k^{-1} \mathbf{g}_k \tag{8.17}$$

Thus the Newton step $\mathbf{d}_k = -\mathbf{H}_k^{-1} \mathbf{g}_k$ is what should be added to $\boldsymbol{\theta}_k$ to minimize the second order approximation of f around $\boldsymbol{\theta}_k$. See Figure 8.4(a) for an illustration.

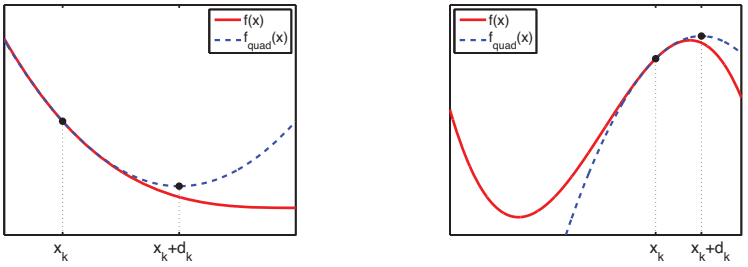


Figure 8.4 Illustration of Newton's method for minimizing a 1d function. (a) The solid curve is the function $f(x)$. The dotted line $f_{quad}(x)$ is its second order approximation at x_k. The Newton step d_k is what must be added to x_k to get to the minimum of $f_{quad}(x)$. Based on Figure 13.4 of (Vandenberghe 2006). Figure generated by `newtonsMethodMinQuad`. (b) Illustration of Newton's method applied to a nonconvex function. We fit a quadratic around the current point x_k and move to its stationary point, $x_{k+1} = x_k + d_k$. Unfortunately, this is a local maximum, not minimum. This means we need to be careful about the extent of our quadratic approximation. Based on Figure 13.11 of (Vandenberghe 2006). Figure generated by `newtonsMethodNonConvex`.

In its simplest form (as listed), Newton's method requires that $\mathbf{H}_k$ be positive definite, which will hold if the function is strictly convex. If not, the objective function is not convex, then $\mathbf{H}_k$ may not be positive definite, so $\mathbf{d}_k = -\mathbf{H}_k^{-1}\mathbf{g}_k$ may not be a descent direction (see Figure 8.4(b) for an example). In this case, one simple strategy is to revert to steepest descent, $\mathbf{d}_k = -\mathbf{g}_k$. The **Levenberg Marquardt** algorithm is an adaptive way to blend between Newton steps and steepest descent steps. This method is widely used when solving nonlinear least squares problems. An alternative approach is this: Rather than computing $\mathbf{d}_k = -\mathbf{H}_k^{-1}\mathbf{g}_k$ directly, we can solve the linear system of equations $\mathbf{H}_k\mathbf{d}_k = -\mathbf{g}_k$ for $\mathbf{d}_k$ using conjugate gradient (CG). If $\mathbf{H}_k$ is not positive definite, we can simply truncate the CG iterations as soon as negative curvature is detected; this is called **truncated Newton**.

8.3.4 Iteratively reweighted least squares (IRLS)

Let us now apply Newton's algorithm to find the MLE for binary logistic regression. The Newton update at iteration $k+1$ for this model is as follows (using $\eta_k = 1$, since the Hessian is exact):

$$\begin{aligned}
\mathbf{w}_{k+1} &= \mathbf{w}_k - \mathbf{H}^{-1}\mathbf{g}_k && (8.18)\\
&= \mathbf{w}_k + (\mathbf{X}^T\mathbf{S}_k\mathbf{X})^{-1}\mathbf{X}^T(\mathbf{y} - \boldsymbol{\mu}_k) && (8.19)\\
&= (\mathbf{X}^T\mathbf{S}_k\mathbf{X})^{-1}\left[(\mathbf{X}^T\mathbf{S}_k\mathbf{X})\mathbf{w}_k + \mathbf{X}^T(\mathbf{y} - \boldsymbol{\mu}_k)\right] && (8.20)\\
&= (\mathbf{X}^T\mathbf{S}_k\mathbf{X})^{-1}\mathbf{X}^T\left[\mathbf{S}_k\mathbf{X}\mathbf{w}_k + \mathbf{y} - \boldsymbol{\mu}_k\right] && (8.21)\\
&= (\mathbf{X}^T\mathbf{S}_k\mathbf{X})^{-1}\mathbf{X}^T\mathbf{S}_k\mathbf{z}_k && (8.22)
\end{aligned}$$

where we have defined the **working response** as

$$\mathbf{z}_k \triangleq \mathbf{X}\mathbf{w}_k + \mathbf{S}_k^{-1}(\mathbf{y} - \boldsymbol{\mu}_k) \quad (8.23)$$

Equation 8.22 is an example of a **weighted least squares problem**, which is a minimizer of

$$\sum_{i=1}^{N} S_{ki}(z_{ki} - \mathbf{w}^T\mathbf{x}_i)^2 \tag{8.24}$$

Since $\mathbf{S}_k$ is a diagonal matrix, we can rewrite the targets in component form (for each case $i = 1 : N$) as

$$z_{ki} = \mathbf{w}_k^T\mathbf{x}_i + \frac{y_i - \mu_{ki}}{\mu_{ki}(1 - \mu_{ki})} \tag{8.25}$$

This algorithm is known as **iteratively reweighted least squares** or **IRLS** for short, since at each iteration, we solve a weighted least squares problem, where the weight matrix $\mathbf{S}_k$ changes at each iteration. See Algorithm 10 for some pseudocode.

Algorithm 8.2: Iteratively reweighted least squares (IRLS)

$\mathbf{w} = \mathbf{0}_D$;
$w_0 = \log(\overline{y}/(1 - \overline{y}))$;
repeat
  $\eta_i = w_0 + \mathbf{w}^T\mathbf{x}_i$;
  $\mu_i = \text{sigm}(\eta_i)$;
  $s_i = \mu_i(1 - \mu_i)$;
  $z_i = \eta_i + \frac{y_i - \mu_i}{s_i}$;
  $\mathbf{S} = \text{diag}(s_{1:N})$;
  $\mathbf{w} = (\mathbf{X}^T\mathbf{S}\mathbf{X})^{-1}\mathbf{X}^T\mathbf{S}\mathbf{z}$;
until *converged*;

8.3.5 Quasi-Newton (variable metric) methods

The mother of all second-order optimization algorithm is Newton's algorithm, which we discussed in Section 8.3.3. Unfortunately, it may be too expensive to compute $\mathbf{H}$ explicitly. **Quasi-Newton** methods iteratively build up an approximation to the Hessian using information gleaned from the gradient vector at each step. The most common method is called **BFGS** (named after its inventors, Broyden, Fletcher, Goldfarb and Shanno), which updates the approximation to the Hessian $\mathbf{B}_k \approx \mathbf{H}_k$ as follows:

$$\mathbf{B}_{k+1} = \mathbf{B}_k + \frac{\mathbf{y}_k\mathbf{y}_k^T}{\mathbf{y}_k^T\mathbf{s}_k} - \frac{(\mathbf{B}_k\mathbf{s}_k)(\mathbf{B}_k\mathbf{s}_k)^T}{\mathbf{s}_k^T\mathbf{B}_k\mathbf{s}_k} \tag{8.26}$$

$$\mathbf{s}_k = \boldsymbol{\theta}_k - \boldsymbol{\theta}_{k-1} \tag{8.27}$$

$$\mathbf{y}_k = \mathbf{g}_k - \mathbf{g}_{k-1} \tag{8.28}$$

This is a rank-two update to the matrix, and ensures that the matrix remains positive definite (under certain restrictions on the step size). We typically start with a diagonal approximation, $\mathbf{B}_0 = \mathbf{I}$. Thus BFGS can be thought of as a "diagonal plus low-rank" approximation to the Hessian.

Alternatively, BFGS can iteratively update an approximation to the inverse Hessian, $\mathbf{C}_k \approx \mathbf{H}_k^{-1}$, as follows:

$$\mathbf{C}_{k+1} = \left(\mathbf{I} - \frac{\mathbf{s}_k \mathbf{y}_k^T}{\mathbf{y}_k^T \mathbf{s}_k}\right) \mathbf{C}_k \left(\mathbf{I} - \frac{\mathbf{y}_k \mathbf{s}_k^T}{\mathbf{y}_k^T \mathbf{s}_k}\right) + \frac{\mathbf{s}_k \mathbf{s}_k^T}{\mathbf{y}_k^T \mathbf{s}_k} \tag{8.29}$$

Since storing the Hessian takes $O(D^2)$ space, for very large problems, one can use **limited memory BFGS**, or **L-BFGS**, where $\mathbf{H}_k$ or $\mathbf{H}_k^{-1}$ is approximated by a diagonal plus low rank matrix. In particular, the product $\mathbf{H}_k^{-1}\mathbf{g}_k$ can be obtained by performing a sequence of inner products with $\mathbf{s}_k$ and $\mathbf{y}_k$, using only the m most recent $(\mathbf{s}_k, \mathbf{y}_k)$ pairs, and ignoring older information. The storage requirements are therefore $O(mD)$. Typically $m \sim 20$ suffices for good performance. See (Nocedal and Wright 2006, p177) for more information. L-BFGS is often the method of choice for most unconstrained smooth optimization problems that arise in machine learning (although see Section 8.5).

8.3.6 ℓ_2 regularization

Just as we prefer ridge regression to linear regression, so we should prefer MAP estimation for logistic regression to computing the MLE. In fact, regularization is important in the classification setting even if we have lots of data. To see why, suppose the data is linearly separable. In this case, the MLE is obtained when $||\mathbf{w}|| \to \infty$, corresponding to an infinitely steep sigmoid function, $\mathbb{I}(\mathbf{w}^T\mathbf{x} > w_0)$, also known as a **linear threshold unit**. This assigns the maximal amount of probability mass to the training data. However, such a solution is very brittle and will not generalize well.

To prevent this, we can use ℓ_2 regularization, just as we did with ridge regression. We note that the new objective, gradient and Hessian have the following forms:

$$f'(\mathbf{w}) = \mathrm{NLL}(\mathbf{w}) + \lambda \mathbf{w}^T\mathbf{w} \tag{8.30}$$

$$\mathbf{g}'(\mathbf{w}) = \mathbf{g}(\mathbf{w}) + \lambda \mathbf{w} \tag{8.31}$$

$$\mathbf{H}'(\mathbf{w}) = \mathbf{H}(\mathbf{w}) + \lambda \mathbf{I} \tag{8.32}$$

It is a simple matter to pass these modified equations into any gradient-based optimizer.

8.3.7 Multi-class logistic regression

Now we consider **multinomial logistic regression**, sometimes called a **maximum entropy classifier**. This is a model of the form

$$p(y = c|\mathbf{x}, \mathbf{W}) = \frac{\exp(\mathbf{w}_c^T\mathbf{x})}{\sum_{c'=1}^{C} \exp(\mathbf{w}_{c'}^T\mathbf{x})} \tag{8.33}$$

A slight variant, known as a **conditional logit model**, normalizes over a different set of classes for each data case; this can be useful for modeling choices that users make between different sets of items that are offered to them.

Let us now introduce some notation. Let $\mu_{ic} = p(y_i = c|\mathbf{x}_i, \mathbf{W}) = \mathcal{S}(\boldsymbol{\eta}_i)_c$, where $\boldsymbol{\eta}_i = \mathbf{W}^T\mathbf{x}_i$ is a $C \times 1$ vector. Also, let $y_{ic} = \mathbb{I}(y_i = c)$ be the one-of-C encoding of y_i; thus $\mathbf{y}_i$ is a bit vector, in which the c'th bit turns on iff $y_i = c$. Following (Krishnapuram et al. 2005), let us

set $\mathbf{w}_C = \mathbf{0}$, to ensure identifiability, and define $\mathbf{w} = \text{vec}(\mathbf{W}(:, 1:C-1))$ to be a $D \times (C-1)$ column vector.

With this, the log-likelihood can be written as

$$\ell(\mathbf{W}) = \log \prod_{i=1}^{N} \prod_{c=1}^{C} \mu_{ic}^{y_{ic}} = \sum_{i=1}^{N} \sum_{c=1}^{C} y_{ic} \log \mu_{ic} \tag{8.34}$$

$$= \sum_{i=1}^{N} \left[\left(\sum_{c=1}^{C} y_{ic} \mathbf{w}_c^T \mathbf{x}_i \right) - \log \left(\sum_{c'=1}^{C} \exp(\mathbf{w}_{c'}^T \mathbf{x}_i) \right) \right] \tag{8.35}$$

Define the NLL as

$$f(\mathbf{w}) = -\ell(\mathbf{w}) \tag{8.36}$$

We now proceed to compute the gradient and Hessian of this expression. Since $\mathbf{w}$ is block-structured, the notation gets a bit heavy, but the ideas are simple. It helps to define $\mathbf{A} \otimes \mathbf{B}$ be the **kronecker product** of matrices $\mathbf{A}$ and $\mathbf{B}$. If $\mathbf{A}$ is an $m \times n$ matrix and $\mathbf{B}$ is a $p \times q$ matrix, then $\mathbf{A} \times \mathbf{B}$ is the $mp \times nq$ block matrix

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{11}\mathbf{B} & \cdots & a_{1n}\mathbf{B} \\ \vdots & \ddots & \vdots \\ a_{m1}\mathbf{B} & \cdots & a_{mn}\mathbf{B} \end{bmatrix} \tag{8.37}$$

Returning to the task at hand, one can show (Exercise 8.4) that the gradient is given by

$$\mathbf{g}(\mathbf{W}) = \nabla f(\mathbf{w}) = \sum_{i=1}^{N} (\boldsymbol{\mu}_i - \mathbf{y}_i) \otimes \mathbf{x}_i \tag{8.38}$$

where $\mathbf{y}_i = (\mathbb{I}(y_i = 1), \ldots, \mathbb{I}(y_i = C-1))$ and $\boldsymbol{\mu}_i(\mathbf{W}) = [p(y_i = 1|\mathbf{x}_i, \mathbf{W}), \ldots, p(y_i = C-1|\mathbf{x}_i, \mathbf{W})]$ are column vectors of length $C-1$, For example, if we have $D = 3$ feature dimensions and $C = 3$ classes, this becomes

$$\mathbf{g}(\mathbf{W}) = \sum_i \begin{pmatrix} (\mu_{i1} - y_{i1})x_{i1} \\ (\mu_{i1} - y_{i1})x_{i2} \\ (\mu_{i1} - y_{i1})x_{i3} \\ (\mu_{i2} - y_{i2})x_{i1} \\ (\mu_{i2} - y_{i2})x_{i2} \\ (\mu_{i2} - y_{i2})x_{i3} \end{pmatrix} \tag{8.39}$$

In other words, for each class c, the derivative for the weights in the c'th column is

$$\nabla_{\mathbf{w}_c} f(\mathbf{W}) = \sum_i (\mu_{ic} - y_{ic}) \mathbf{x}_i \tag{8.40}$$

This has the same form as in the binary logistic regression case, namely an error term times $\mathbf{x}_i$. (This turns out to be a general property of distributions in the exponential family, as we will see in Section 9.3.2.)

One can also show (Exercise 8.4) that the Hessian is the following block structured $D(C-1) \times D(C-1)$ matrix:

$$\mathbf{H}(\mathbf{W}) = \nabla^2 f(\mathbf{w}) = \sum_{i=1}^{N} (\text{diag}(\boldsymbol{\mu}_i) - \boldsymbol{\mu}_i \boldsymbol{\mu}_i^T) \otimes (\mathbf{x}_i \mathbf{x}_i^T) \tag{8.41}$$

For example, if we have 3 features and 3 classes, this becomes

$$\mathbf{H}(\mathbf{W}) = \sum_i \begin{pmatrix} \mu_{i1} - \mu_{i1}^2 & -\mu_{i1}\mu_{i2} \\ -\mu_{i1}\mu_{i2} & \mu_{i2} - \mu_{i2}^2 \end{pmatrix} \otimes \begin{pmatrix} x_{i1}x_{i1} & x_{i1}x_{i2} & x_{i1}x_{i3} \\ x_{i2}x_{i1} & x_{i2}x_{i2} & x_{i2}x_{i3} \\ x_{i3}x_{i1} & x_{i3}x_{i2} & x_{i3}x_{i3} \end{pmatrix} \tag{8.42}$$

$$= \sum_i \begin{pmatrix} (\mu_{i1} - \mu_{i1}^2)\mathbf{X}_i & -\mu_{i1}\mu_{i2}\mathbf{X}_i \\ -\mu_{i1}\mu_{i2}\mathbf{X}_i & (\mu_{i2} - \mu_{i2}^2)\mathbf{X}_i \end{pmatrix} \tag{8.43}$$

where $\mathbf{X}_i = \mathbf{x}_i \mathbf{x}_i^T$. In other words, the block c, c' submatrix is given by

$$\mathbf{H}_{c,c'}(\mathbf{W}) = \sum_i \mu_{ic}(\delta_{c,c'} - \mu_{i,c'})\mathbf{x}_i \mathbf{x}_i^T \tag{8.44}$$

This is also a positive definite matrix, so there is a unique MLE.

Now consider minimizing

$$f'(\mathbf{W}) \triangleq -\log p(\mathcal{D}|\mathbf{w}) - \log p(\mathbf{W}) \tag{8.45}$$

where $p(\mathbf{W}) = \prod_c \mathcal{N}(\mathbf{w}_c|\mathbf{0}, \mathbf{V}_0)$. The new objective, its gradient and Hessian are given by

$$f'(\mathbf{W}) = f(\mathbf{W}) + \frac{1}{2}\sum_c \mathbf{w}_c \mathbf{V}_0^{-1} \mathbf{w}_c \tag{8.46}$$

$$\mathbf{g}'(\mathbf{W}) = \mathbf{g}(\mathbf{W}) + \mathbf{V}_0^{-1}(\sum_c \mathbf{w}_c) \tag{8.47}$$

$$\mathbf{H}'(\mathbf{W}) = \mathbf{H}(\mathbf{W}) + \mathbf{I}_C \otimes \mathbf{V}_0^{-1} \tag{8.48}$$

This can be passed to any gradient-based optimizer to find the MAP estimate. Note, however, that the Hessian has size $O((CD) \times (CD))$, which is C times more row and columns than in the binary case, so limited memory BFGS is more appropriate than Newton's method. See `logregFit` for some Matlab code.

8.4 Bayesian logistic regression

It is natural to want to compute the full posterior over the parameters, $p(\mathbf{w}|\mathcal{D})$, for logistic regression models. This can be useful for any situation where we want to associate confidence intervals with our predictions (e.g., this is necessary when solving contextual bandit problems, discussed in Section 5.7.3.1).

Unfortunately, unlike the linear regression case, this cannot be done exactly, since there is no convenient conjugate prior for logistic regression. We discuss one simple approximation below; some other approaches include MCMC (Section 24.3.3.1), variational inference (Section 21.8.1.1), expectation propagation (Kuss and Rasmussen 2005), etc. For notational simplicity, we stick to binary logistic regression.

8.4.1 Laplace approximation

In this section, we discuss how to make a Gaussian approximation to a posterior distribution. The approximation works as follows. Suppose $\boldsymbol{\theta} \in \mathbb{R}^D$. Let

$$p(\boldsymbol{\theta}|\mathcal{D}) = \frac{1}{Z} e^{-E(\boldsymbol{\theta})} \tag{8.49}$$

where $E(\boldsymbol{\theta})$ is called an **energy function**, and is equal to the negative log of the unnormalized log posterior, $E(\boldsymbol{\theta}) = -\log p(\boldsymbol{\theta}, \mathcal{D})$, with $Z = p(\mathcal{D})$ being the normalization constant. Performing a Taylor series expansion around the mode $\boldsymbol{\theta}^*$ (i.e., the lowest energy state) we get

$$E(\boldsymbol{\theta}) \approx E(\boldsymbol{\theta}^*) + (\boldsymbol{\theta} - \boldsymbol{\theta}^*)^T \mathbf{g} + \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}^*)^T \mathbf{H} (\boldsymbol{\theta} - \boldsymbol{\theta}^*) \tag{8.50}$$

where $\mathbf{g}$ is the gradient and $\mathbf{H}$ is the Hessian of the energy function evaluated at the mode:

$$\mathbf{g} \triangleq \nabla E(\boldsymbol{\theta})\big|_{\boldsymbol{\theta}^*}, \; \mathbf{H} \triangleq \frac{\partial^2 E(\boldsymbol{\theta})}{\partial \boldsymbol{\theta} \partial \boldsymbol{\theta}^T} \big|_{\boldsymbol{\theta}^*} \tag{8.51}$$

Since $\boldsymbol{\theta}^*$ is the mode, the gradient term is zero. Hence

$$\hat{p}(\boldsymbol{\theta}|\mathcal{D}) \approx \frac{1}{Z} e^{-E(\boldsymbol{\theta}^*)} \exp\left[-\frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}^*)^T \mathbf{H} (\boldsymbol{\theta} - \boldsymbol{\theta}^*)\right] \tag{8.52}$$

$$= \mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\theta}^*, \mathbf{H}^{-1}) \tag{8.53}$$

$$Z = p(\mathcal{D}) \approx \int \hat{p}(\boldsymbol{\theta}|\mathcal{D}) d\boldsymbol{\theta} = e^{-E(\boldsymbol{\theta}^*)} (2\pi)^{D/2} |\mathbf{H}|^{-\frac{1}{2}} \tag{8.54}$$

The last line follows from normalization constant of the multivariate Gaussian.

Equation 8.54 is known as the **Laplace approximation** to the marginal likelihood. Therefore Equation 8.52 is sometimes called the the **Laplace approximation** to the posterior. However, in the statistics community, the term "Laplace approximation" refers to a more sophisticated method (see e.g. (Rue et al. 2009) for details). It may therefore be better to use the term "**Gaussian approximation**" to refer to Equation 8.52. A Gaussian approximation is often a reasonable approximation, since posteriors often become more "Gaussian-like" as the sample size increases, for reasons analogous to the central limit theorem. (In physics, there is an analogous technique known as a **saddle point approximation**.)

8.4.2 Derivation of the BIC

We can use the Gaussian approximation to write the log marginal likelihood as follows, dropping irrelevant constants:

$$\log p(\mathcal{D}) \approx \log p(\mathcal{D}|\boldsymbol{\theta}^*) + \log p(\boldsymbol{\theta}^*) - \frac{1}{2} \log |\mathbf{H}| \tag{8.55}$$

The penalization terms which are added to the $\log p(\mathcal{D}|\boldsymbol{\theta}^*)$ are sometimes called the **Occam factor**, and are a measure of model complexity. If we have a uniform prior, $p(\boldsymbol{\theta}) \propto 1$, we can drop the second term, and replace $\boldsymbol{\theta}^*$ with the MLE, $\hat{\boldsymbol{\theta}}$.

We now focus on approximating the third term. We have $\mathbf{H} = \sum_{i=1}^{N} \mathbf{H}_i$, where $\mathbf{H}_i = \nabla\nabla \log p(\mathcal{D}_i|\boldsymbol{\theta})$. Let us approximate each $\mathbf{H}_i$ by a fixed matrix $\hat{\mathbf{H}}$. Then we have

$$\log|\mathbf{H}| = \log|N\hat{\mathbf{H}}| = \log(N^d|\hat{\mathbf{H}}|) = D\log N + \log|\hat{\mathbf{H}}| \tag{8.56}$$

where $D = \dim(\boldsymbol{\theta})$ and we have assumed $\mathbf{H}$ is full rank. We can drop the $\log|\hat{\mathbf{H}}|$ term, since it is independent of N, and thus will get overwhelmed by the likelihood. Putting all the pieces together, we recover the BIC score (Section 5.3.2.4):

$$\log p(\mathcal{D}) \approx \log p(\mathcal{D}|\hat{\boldsymbol{\theta}}) - \frac{D}{2}\log N \tag{8.57}$$

8.4.3 Gaussian approximation for logistic regression

Now let us apply the Gaussian approximation to logistic regression. We will use a a Gaussian prior of the form $p(\mathbf{w}) = \mathcal{N}(\mathbf{w}|\mathbf{0}, \mathbf{V}_0)$, just as we did in MAP estimation. The approximate posterior is given by

$$p(\mathbf{w}|\mathcal{D}) \approx \mathcal{N}(\mathbf{w}|\hat{\mathbf{w}}, \mathbf{H}^{-1}) \tag{8.58}$$

where $\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} E(\mathbf{w})$, $E(\mathbf{w}) = -(\log p(\mathcal{D}|\mathbf{w}) + \log p(\mathbf{w}))$, and $\mathbf{H} = \nabla^2 E(\mathbf{w})|_{\hat{\mathbf{w}}}$.

As an example, consider the linearly separable 2D data in Figure 8.5(a). There are many parameter settings that correspond to lines that perfectly separate the training data; we show 4 examples. The likelihood surface is shown in Figure 8.5(b), where we see that the likelihood is unbounded as we move up and to the right in parameter space, along a ridge where $w_2/w_1 = 2.35$ (this is indicated by the diagonal line). The reasons for this is that we can maximize the likelihood by driving $||\mathbf{w}||$ to infinity (subject to being on this line), since large regression weights make the sigmoid function very steep, turning it into a step function. Consequently the MLE is not well defined when the data is linearly separable.

To regularize the problem, let us use a vague spherical prior centered at the origin, $\mathcal{N}(\mathbf{w}|\mathbf{0}, 100\mathbf{I})$. Multiplying this spherical prior by the likelihood surface results in a highly skewed posterior, shown in Figure 8.5(c). (The posterior is skewed because the likelihood function "chops off" regions of parameter space (in a "soft" fashion) which disagree with the data.) The MAP estimate is shown by the blue dot. Unlike the MLE, this is not at infinity.

The Gaussian approximation to this posterior is shown in Figure 8.5(d). We see that this is a symmetric distribution, and therefore not a great approximation. Of course, it gets the mode correct (by construction), and it at least represents the fact that there is more uncertainty along the southwest-northeast direction (which corresponds to uncertainty about the orientation of separating lines) than perpendicular to this. Although a crude approximation, this is surely better than approximating the posterior by a delta function, which is what MAP estimation does.

8.4.4 Approximating the posterior predictive

Given the posterior, we can compute credible intervals, perform hypothesis tests, etc., just as we did in Section 7.6.3.3 in the case of linear regression. But in machine learning, interest usually focusses on prediction. The posterior predictive distribution has the form

$$p(y|\mathbf{x}, \mathcal{D}) = \int p(y|\mathbf{x}, \mathbf{w}) p(\mathbf{w}|\mathcal{D}) d\mathbf{w} \tag{8.59}$$

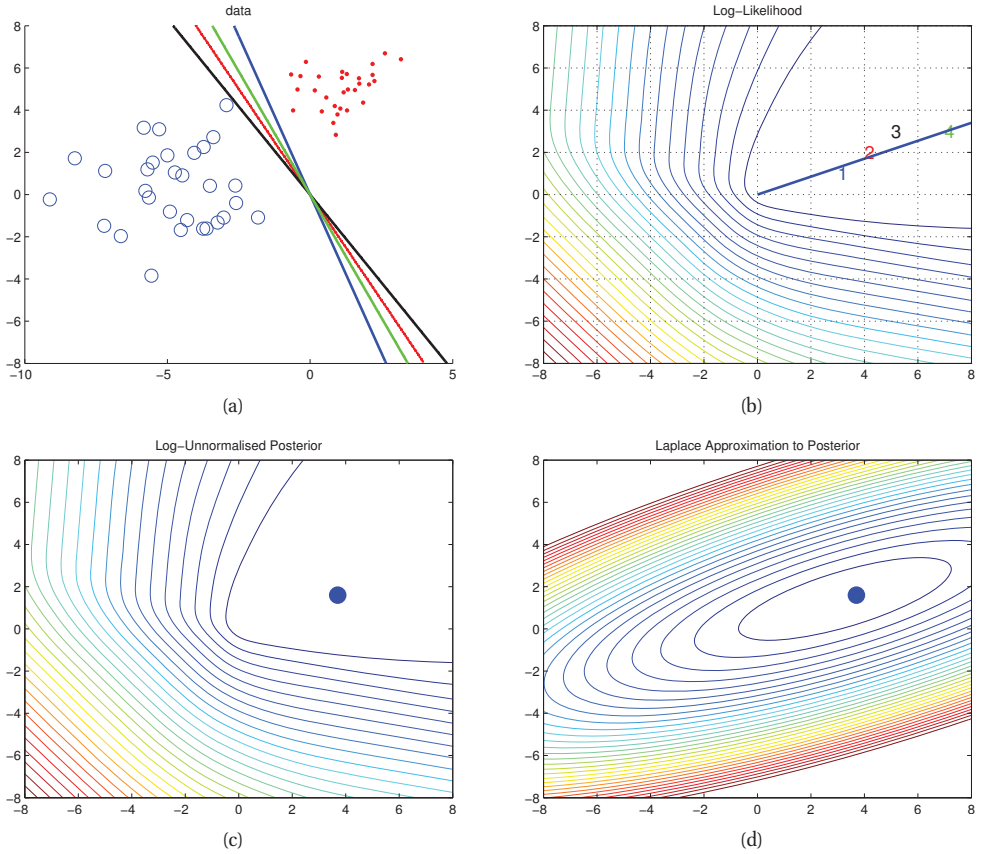


Figure 8.5 (a) Two-class data in 2d. (b) Log-likelihood for a logistic regression model. The line is drawn from the origin in the direction of the MLE (which is at infinity). The numbers correspond to 4 points in parameter space, corresponding to the lines in (a). (c) Unnormalized log posterior (assuming vague spherical prior). (d) Laplace approximation to posterior. Based on a figure by Mark Girolami. Figure generated by `logregLaplaceGirolamiDemo`.

Unfortunately this integral is intractable.

The simplest approximation is the plug-in approximation, which, in the binary case, takes the form

$$p(y = 1|\mathbf{x}, \mathcal{D}) \quad \approx \quad p(y = 1|\mathbf{x}, \mathbb{E}\left[\mathbf{w}\right]) \tag{8.60}$$

where $\mathbb{E}\left[\mathbf{w}\right]$ is the posterior mean. In this context, $\mathbb{E}\left[\mathbf{w}\right]$ is called the **Bayes point**. Of course, such a plug-in estimate underestimates the uncertainty. We discuss some better approximations below.

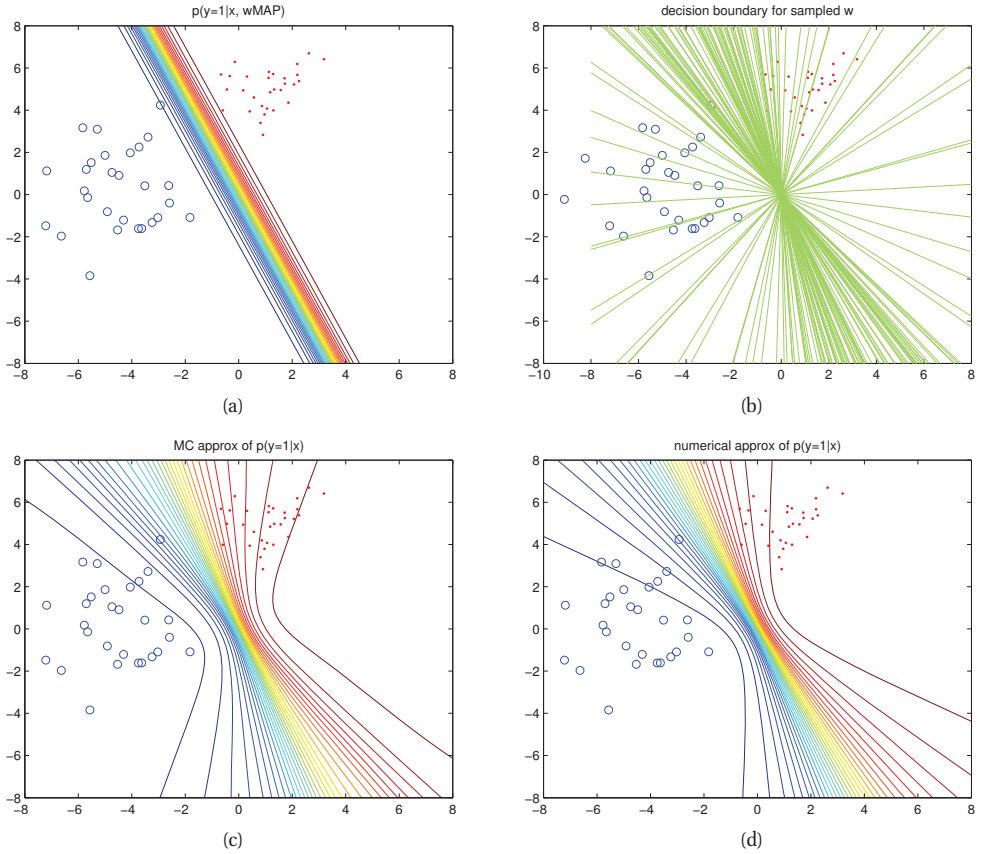


Figure 8.6 Posterior predictive distribution for a logistic regression model in 2d. Top left: contours of $p(y = 1|\mathbf{x}, \hat{\mathbf{w}}_{map})$. Top right: samples from the posterior predictive distribution. Bottom left: Averaging over these samples. Bottom right: moderated output (probit approximation). Based on a figure by Mark Girolami. Figure generated by `logregLaplaceGirolamiDemo`.

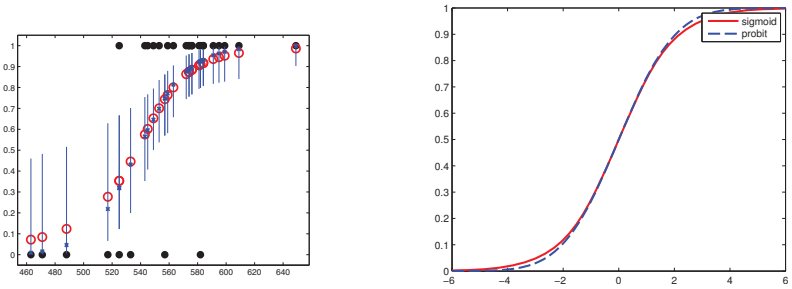
8.4.4.1 Monte Carlo approximation

A better approach is to use a Monte Carlo approximation, as follows:

$$p(y = 1|\mathbf{x}, \mathcal{D}) \approx \frac{1}{S}\sum_{s=1}^{S} \text{sigm}((\mathbf{w}^s)^T\mathbf{x}) \tag{8.61}$$

where $\mathbf{w}^s \sim p(\mathbf{w}|\mathcal{D})$ are samples from the posterior. (This technique can be trivially extended to the multi-class case.) If we have approximated the posterior using Monte Carlo, we can reuse these samples for prediction. If we made a Gaussian approximation to the posterior, we can draw *independent* samples from the Gaussian using standard methods.

Figure 8.6(b) shows samples from the posteiror predictive for our 2d example. Figure 8.6(c)



(a) (b)

Figure 8.7 (a) Posterior predictive density for SAT data. The red circle denotes the posterior mean, the blue cross the posterior median, and the blue lines denote the 5th and 95th percentiles of the predictive distribution. Figure generated by `logregSATdemoBayes`. (b) The logistic (sigmoid) function $\text{sigm}(x)$ in solid red, with the rescaled probit function $\Phi(\lambda x)$ in dotted blue superimposed. Here $\lambda = \sqrt{\pi/8}$, which was chosen so that the derivatives of the two curves match at $x = 0$. Based on Figure 4.9 of (Bishop 2006b). Figure generated by `probitPlot`. Figure generated by `probitRegDemo`.

shows the average of these samples. By averaging over multiple predictions, we see that the uncertainty in the decision boundary "splays out" as we move further from the training data. So although the decision boundary is linear, the posterior predictive density is not linear. Note also that the posterior mean decision boundary is roughly equally far from both classes; this is the Bayesian analog of the large margin principle discussed in Section 14.5.2.2.

Figure 8.7(a) shows an example in 1d. The red dots denote the mean of the posterior predictive evaluated at the training data. The vertical blue lines denote 95% credible intervals for the posterior predictive; the small blue star is the median. We see that, with the Bayesian approach, we are able to model our uncertainty about the probability a student will pass the exam based on his SAT score, rather than just getting a point estimate.

8.4.4.2 Probit approximation (moderated output) *

If we have a Gaussian approximation to the posterior $p(\mathbf{w}|\mathcal{D}) \approx \mathcal{N}(\mathbf{w}|\mathbf{m}_N, \mathbf{V}_N)$, we can also compute a deterministic approximation to the posterior predictive distribution, at least in the binary case. We proceed as follows:

$$p(y=1|\mathbf{x},\mathcal{D}) \approx \int \text{sigm}(\mathbf{w}^T\mathbf{x}) p(\mathbf{w}|\mathcal{D}) d\mathbf{w} = \int \text{sigm}(a)\mathcal{N}(a|\mu_a, \sigma_a^2) da \tag{8.62}$$

$$a \triangleq \mathbf{w}^T\mathbf{x} \tag{8.63}$$

$$\mu_a \triangleq \mathbb{E}[a] = \mathbf{m}_N^T\mathbf{x} \tag{8.64}$$

$$\sigma_a^2 \triangleq \text{var}[a] = \int p(a|\mathcal{D})[a^2 - \mathbb{E}[a^2]] da \tag{8.65}$$

$$= \int p(\mathbf{w}|\mathcal{D})[(\mathbf{w}^T\mathbf{x})^2 - (\mathbf{m}_N^T\mathbf{x})^2] d\mathbf{w} = \mathbf{x}^T\mathbf{V}_N\mathbf{x} \tag{8.66}$$

Thus we see that we need to evaluate the expectation of a sigmoid with respect to a Gaussian. This can be approximated by exploiting the fact that the sigmoid function is similar to the **probit** function, which is given by the cdf of the standard normal:

$$\Phi(a) \triangleq \int_{-\infty}^{a} \mathcal{N}(x|0,1)dx \tag{8.67}$$

Figure 8.7(b) plots the sigmoid and probit functions. We have rescaled the axes so that $\text{sigm}(a)$ has the same slope as $\Phi(\lambda a)$ at the origin, where $\lambda^2 = \pi/8$.

The advantage of using the probit is that one can convolve it with a Gaussian analytically:

$$\int \Phi(\lambda a)\mathcal{N}(a|\mu,\sigma^2)da = \Phi\left(\frac{a}{(\lambda^{-2}+\sigma^2)^{\frac{1}{2}}}\right) \tag{8.68}$$

We now plug in the approximation $\text{sigm}(a) \approx \Phi(\lambda a)$ to both sides of this equation to get

$$\int \text{sigm}(a)\mathcal{N}(a|\mu,\sigma^2)da \approx \text{sigm}(\kappa(\sigma^2)\mu) \tag{8.69}$$

$$\kappa(\sigma^2) \triangleq (1+\pi\sigma^2/8)^{-\frac{1}{2}} \tag{8.70}$$

Applying this to the logistic regression model we get the following expression (first suggested in (Spiegelhalter and Lauritzen 1990)):

$$p(y=1|\mathbf{x},\mathcal{D}) \approx \text{sigm}(\kappa(\sigma_a^2)\mu_a) \tag{8.71}$$

Figure 8.6(d) indicates that this gives very similar results to the Monte Carlo approximation.

Using Equation 8.71 is sometimes called a **moderated output**, since it is less extreme than the plug-in estimate. To see this, note that $0 \leq \kappa(\sigma^2) \leq 1$ and hence

$$\text{sigm}(\kappa(\sigma^2)\mu) \leq \text{sigm}(\mu) = p(y=1|\mathbf{x},\hat{\mathbf{w}}) \tag{8.72}$$

where the inequality is strict if $\mu \neq 0$. If $\mu > 0$, we have $p(y=1|\mathbf{x},\hat{\mathbf{w}}) > 0.5$, but the moderated prediction is always closer to 0.5, so it is less confident. However, the decision boundary occurs whenever $p(y=1|\mathbf{x},\mathcal{D}) = \text{sigm}(\kappa(\sigma^2)\mu) = 0.5$, which implies $\mu = \hat{\mathbf{w}}^T\mathbf{x} = 0$. Hence the decision boundary for the moderated approximation is the same as for the plug-in approximation. So the number of misclassifications will be the same for the two methods, but the log-likelihood will not. (Note that in the multiclass case, taking into account posterior covariance gives different answers than the plug-in approach: see Exercise 3.10.3 of (Rasmussen and Williams 2006).)

8.4.5 Residual analysis (outlier detection) *

It is sometimes useful to detect data cases which are "outliers". This is called **residual analysis** or **case analysis**. In a regression setting, this can be performed by computing $r_i = y_i - \hat{y}_i$, where $\hat{y}_i = \hat{\mathbf{w}}^T\mathbf{x}_i$. These values should follow a $\mathcal{N}(0,\sigma^2)$ distribution, if the modelling assumptions are correct. This can be assessed by creating a **qq-plot**, where we plot the N theoretical quantiles of a Gaussian distribution against the N empirical quantiles of the r_i. Points that deviate from the straightline are potential outliers.

Classical methods, based on residuals, do not work well for binary data, because they rely on asymptotic normality of the test statistics. However, adopting a Bayesian approach, we can just define outliers to be points which which $p(y_i|\hat{y}_i)$ is small, where we typically use $\hat{y}_i = \text{sigm}(\hat{\mathbf{w}}^T\mathbf{x}_i)$. Note that $\hat{\mathbf{w}}$ was estimated from all the data. A better method is to exclude $(\mathbf{x}_i, y_i)$ from the estimate of $\mathbf{w}$ when predicting y_i. That is, we define outliers to be points which have low probability under the cross-validated posterior predictive distribution, defined by

$$p(y_i|\mathbf{x}_i, \mathbf{x}_{-i}, \mathbf{y}_{-i}) = \int p(y_i|\mathbf{x}_i, \mathbf{w}) \prod_{i' \neq i} p(y_{i'}|\mathbf{x}_{i'}, \mathbf{w}) p(\mathbf{w}) d\mathbf{w} \tag{8.73}$$

This can be efficiently approximated by sampling methods (Gelfand 1996). For further discussion of residual analysis in logistic regression models, see e.g.,(Johnson and Albert 1999, Sec 3.4).

8.5 Online learning and stochastic optimization

Traditionally machine learning is performed **offline**, which means we have a **batch** of data, and we optimize an equation of the following form

$$f(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^{N} f(\boldsymbol{\theta}, \mathbf{z}_i) \tag{8.74}$$

where $\mathbf{z}_i = (\mathbf{x}_i, y_i)$ in the supervised case, or just $\mathbf{x}_i$ in the unsupervised case, and $f(\boldsymbol{\theta}, \mathbf{z}_i)$ is some kind of loss function. For example, we might use

$$f(\boldsymbol{\theta}, \mathbf{z}_i) = -\log p(y_i|\mathbf{x}_i, \boldsymbol{\theta}) \tag{8.75}$$

in which case we are trying to maximize the likelihood. Alternatively, we might use

$$f(\boldsymbol{\theta}, \mathbf{z}_i) = L(y_i, h(\mathbf{x}_i, \boldsymbol{\theta})) \tag{8.76}$$

where $h(\mathbf{x}_i, \boldsymbol{\theta})$ is a prediction function, and $L(y, \hat{y})$ is some other loss function such as squared error or the Huber loss. In frequentist decision theory, the average loss is called the risk (see Section 6.3), so this overall approach is called empirical risk minimization or ERM (see Section 6.5 for details).

However, if we have **streaming data**, we need to perform **online learning**, so we can update our estimates as each new data point arrives rather than waiting until "the end" (which may never occur). And even if we have a batch of data, we might want to treat it like a stream if it is too large to hold in main memory. Below we discuss learning methods for this kind of scenario.[1]

1. A simple implementation trick can be used to speed up batch learning algorithms when applied to data sets that are too large to hold in memory. First note that the naive implementation makes a pass over the data file, from the beginning to end, accumulating the sufficient statistics and gradients as it goes; then an update is performed and the process repeats. Unfortunately, at the end of each pass, the data from the beginning of the file will have been evicted from the cache (since are are assuming it cannot all fit into memory). Rather than going back to the beginning of the file and reloading it, we can simply work backwards from the end of the file, which is already in memory. We then repeat this forwards-backwards pattern over the data. This simple trick is known as **rocking**.

8.5.1 Online learning and regret minimization

Suppose that at each step, "nature" presents a sample $\mathbf{z}_k$ and the "learner" must respond with a parameter estimate $\boldsymbol{\theta}_k$. In the theoretical machine learning community, the objective used in online learning is the **regret**, which is the averaged loss incurred relative to the best we could have gotten in hindsight using a single fixed parameter value:

$$\text{regret}_k \triangleq \frac{1}{k}\sum_{t=1}^{k} f(\boldsymbol{\theta}_t, \mathbf{z}_t) - \min_{\boldsymbol{\theta}^* \in \Theta} \frac{1}{k}\sum_{t=1}^{k} f(\boldsymbol{\theta}_*, \mathbf{z}_t) \tag{8.77}$$

For example, imagine we are investing in the stock-market. Let θ_j be the amount we invest in stock j, and let z_j be the return on this stock. Our loss function is $f(\boldsymbol{\theta}, \mathbf{z}) = -\boldsymbol{\theta}^T\mathbf{z}$. The regret is how much better (or worse) we did by trading at each step, rather than adopting a "buy and hold" strategy using an oracle to choose which stocks to buy.

One simple algorithm for online learning is **online gradient descent** (Zinkevich 2003), which is as follows: at each step k, update the parameters using

$$\boldsymbol{\theta}_{k+1} = \text{proj}_\Theta(\boldsymbol{\theta}_k - \eta_k \mathbf{g}_k) \tag{8.78}$$

where $\text{proj}_\mathcal{V}(\mathbf{v}) = \text{argmin}_{\mathbf{w}\in\mathbf{V}} ||\mathbf{w} - \mathbf{v}||_2$ is the projection of vector $\mathbf{v}$ onto space $\mathcal{V}$, $\mathbf{g}_k = \nabla f(\boldsymbol{\theta}_k, \mathbf{z}_k)$ is the gradient, and η_k is the step size. (The projection step is only needed if the parameter must be constrained to live in a certain subset of $\mathbb{R}^D$. See Section 13.4.3 for details.) Below we will see how this approach to regret minimization relates to more traditional objectives, such as MLE.

There are a variety of other approaches to regret minimization which are beyond the scope of this book (see e.g., Cesa-Bianchi and Lugosi (2006) for details).

8.5.2 Stochastic optimization and risk minimization

Now suppose that instead of minimizing regret with respect to the past, we want to minimize expected loss in the future, as is more common in (frequentist) statistical learning theory. That is, we want to minimize

$$f(\boldsymbol{\theta}) = \mathbb{E}\left[f(\boldsymbol{\theta}, \mathbf{z})\right] \tag{8.79}$$

where the expectation is taken over future data. Optimizing functions where some of the variables in the objective are random is called **stochastic optimization**.[2]

Suppose we receive an infinite stream of samples from the distribution. One way to optimize stochastic objectives such as Equation 8.79 is to perform the update in Equation 8.78 at each step. This is called **stochastic gradient descent** or **SGD** (Nemirovski and Yudin 1978). Since we typically want a single parameter estimate, we can use a running average:

$$\overline{\boldsymbol{\theta}}_k = \frac{1}{k}\sum_{t=1}^{k} \boldsymbol{\theta}_t \tag{8.80}$$

2. Note that in stochastic optimization, the objective is stochastic, and therefore the algorithms will be, too. However, it is also possible to apply stochastic optimization algorithms to deterministic objectives. Examples include simulated annealing (Section 24.6.1) and stochastic gradient descent applied to the empirical risk minimization problem. There are some interesting theoretical connections between online learning and stochastic optimization (Cesa-Bianchi and Lugosi 2006), but this is beyond the scope of this book.

This is called **Polyak-Ruppert averaging**, and can be implemented recursively as follows:

$$\overline{\boldsymbol{\theta}}_k = \overline{\boldsymbol{\theta}}_{k-1} - \frac{1}{k}(\overline{\boldsymbol{\theta}}_{k-1} - \boldsymbol{\theta}_k) \tag{8.81}$$

See e.g., (Spall 2003; Kushner and Yin 2003) for details.

8.5.2.1 Setting the step size

We now discuss some sufficient conditions on the learning rate to guarantee convergence of SGD. These are known as the **Robbins-Monro** conditions:

$$\sum_{k=1}^{\infty} \eta_k = \infty, \ \sum_{k=1}^{\infty} \eta_k^2 < \infty. \tag{8.82}$$

The set of values of η_k over time is called the learning rate **schedule**. Various formulas are used, such as $\eta_k = 1/k$, or the following (Bottou 1998; Bach and Moulines 2011):

$$\eta_k = (\tau_0 + k)^{-\kappa} \tag{8.83}$$

where $\tau_0 \geq 0$ slows down early iterations of the algorithm, and $\kappa \in (0.5, 1]$ controls the rate at which old values of are forgotten.

The need to adjust these tuning parameters is one of the main drawback of stochastic optimization. One simple heuristic (Bottou 2007) is as follows: store an initial subset of the data, and try a range of η values on this subset; then choose the one that results in the fastest decrease in the objective and apply it to all the rest of the data. Note that this may not result in convergence, but the algorithm can be terminated when the performance improvement on a hold-out set plateaus (this is called **early stopping**).

8.5.2.2 Per-parameter step sizes

One drawback of SGD is that it uses the same step size for all parameters. We now briefly present a method known as **adagrad** (short for adaptive gradient) (Duchi et al. 2010), which is similar in spirit to a diagonal Hessian approximation. (See also (Schaul et al. 2012) for a similar approach.) In particular, if $\theta_i(k)$ is parameter i at time k, and $g_i(k)$ is its gradient, then we make an update as follows:

$$\theta_i(k+1) = \theta_i(k) - \eta \frac{g_i(k)}{\tau_0 + \sqrt{s_i(k)}} \tag{8.84}$$

where the diagonal step size vector is the gradient vector squared, summed over all time steps. This can be recursively updated as follows:

$$s_i(k) = s_i(k-1) + g_i(k)^2 \tag{8.85}$$

The result is a per-parameter step size that adapts to the curvature of the loss function. This method was original derived for the regret minimization case, but it can be applied more generally.

8.5.2.3 SGD compared to batch learning

If we don't have an infinite data stream, we can "simulate" one by sampling data points at random from our training set. Essentially we are optimizing Equation 8.74 by treating it as an expectation with respect to the empirical distribution.

Algorithm 8.3: Stochastic gradient descent

```
Initialize θ, η;
repeat
    Randomly permute data;
    for i = 1 : N do
        g = ∇f(θ, z_i);
        θ ← proj_Θ(θ − ηg);
        Update η;
until converged;
```

In theory, we should sample with replacement, although in practice it is usually better to randomly permute the data and sample without replacement, and then to repeat. A single such pass over the entire data set is called an **epoch**. See Algorithm 8 for some pseudocode.

In this offline case, it is often better to compute the gradient of a **mini-batch** of B data cases. If $B = 1$, this is standard SGD, and if $B = N$, this is standard **steepest descent**. Typically $B \sim 100$ is used.

Although a simple first-order method, SGD performs surprisingly well on some problems, especially ones with large data sets (Bottou 2007). The intuitive reason for this is that one can get a fairly good estimate of the gradient by looking at just a few examples. Carefully evaluating precise gradients using large datasets is often a waste of time, since the algorithm will have to recompute the gradient again anyway at the next step. It is often a better use of computer time to have a noisy estimate and to move rapidly through parameter space. As an extreme example, suppose we double the training set by duplicating every example. Batch methods will take twice as long, but online methods will be unaffected, since the direction of the gradient has not changed (doubling the size of the data changes the magnitude of the gradient, but that is irrelevant, since the gradient is being scaled by the step size anyway).

In addition to enhanced speed, SGD is often less prone to getting stuck in shallow local minima, because it adds a certain amount of "noise". Consequently it is quite popular in the machine learning community for fitting models with non-convex objectives, such as neural networks (Section 16.5) and deep belief networks (Section 28.1).

8.5.3 The LMS algorithm

As an example of SGD, let us consider how to compute the MLE for linear regression in an online fashion. We derived the batch gradient in Equation 7.14. The online gradient at iteration k is given by

$$\mathbf{g}_k = \mathbf{x}_i(\boldsymbol{\theta}_k^T \mathbf{x}_i - y_i) \tag{8.86}$$

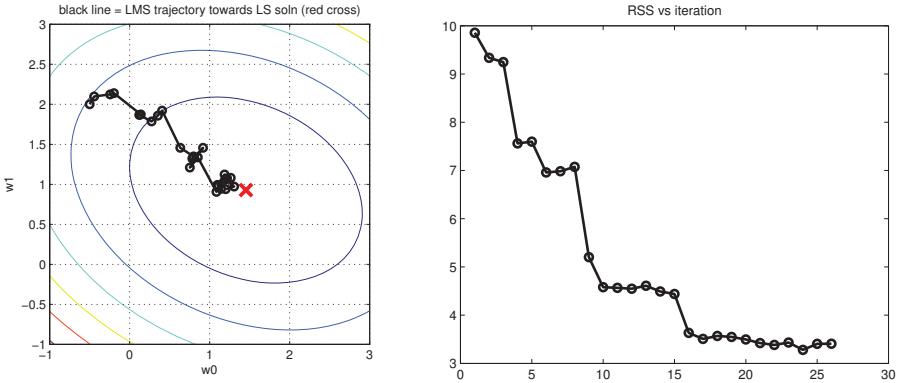


Figure 8.8 Illustration of the LMS algorithm. Left: we start from $\boldsymbol{\theta} = (-0.5, 2)$ and slowly converging to the least squares solution of $\hat{\boldsymbol{\theta}} = (1.45, 0.92)$ (red cross). Right: plot of objective function over time. Note that it does not decrease monotonically. Figure generated by `LMSdemo`.

where $i = i(k)$ is the training example to use at iteration k. If the data set is streaming, we use $i(k) = k$; we shall assume this from now on, for notational simplicity. Equation 8.86 is easy to interpret: it is the feature vector $\mathbf{x}_k$ weighted by the difference between what we predicted, $\hat{y}_k = \boldsymbol{\theta}_k^T \mathbf{x}_k$, and the true response, y_k; hence the gradient acts like an error signal.

After computing the gradient, we take a step along it as follows:

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \eta_k(\hat{y}_k - y_k)\mathbf{x}_k \tag{8.87}$$

(There is no need for a projection step, since this is an unconstrained optimization problem.) This algorithm is called the **least mean squares** or **LMS** algorithm, and is also known as the **delta rule**, or the **Widrow-Hoff rule**.

Figure 8.8 shows the results of applying this algorithm to the data shown in Figure 7.2. We start at $\boldsymbol{\theta} = (-0.5, 2)$ and converge (in the sense that $||\boldsymbol{\theta}_k - \boldsymbol{\theta}_{k-1}||_2^2$ drops below a threshold of 10^{-2}) in about 26 iterations.

Note that LMS may require multiple passes through the data to find the optimum. By contrast, the recursive least squares algorithm, which is based on the Kalman filter and which uses second-order information, finds the optimum in a single pass (see Section 18.2.3). See also Exercise 7.7.

8.5.4 The perceptron algorithm

Now let us consider how to fit a binary logistic regression model in an online manner. The batch gradient was given in Equation 8.5. In the online case, the weight update has the simple form

$$\boldsymbol{\theta}_k = \boldsymbol{\theta}_{k-1} - \eta_k \mathbf{g}_i = \boldsymbol{\theta}_{k-1} - \eta_k(\mu_i - y_i)\mathbf{x}_i \tag{8.88}$$

where $\mu_i = p(y_i = 1|\mathbf{x}_i, \boldsymbol{\theta}_k) = \mathbb{E}[y_i|\mathbf{x}_i, \boldsymbol{\theta}_k]$. We see that this has exactly the same form as the LMS algorithm. Indeed, this property holds for all generalized linear models (Section 9.3).

We now consider an approximation to this algorithm. Specifically, let

$$\hat{y}_i = \arg\max_{y \in \{0,1\}} p(y|\mathbf{x}_i, \boldsymbol{\theta}) \tag{8.89}$$

represent the most probable class label. We replace $\mu_i = p(y = 1|\mathbf{x}_i, \boldsymbol{\theta}) = \text{sigm}(\boldsymbol{\theta}^T \mathbf{x}_i)$ in the gradient expression with $\hat{y}_i$. Thus the approximate gradient becomes

$$\mathbf{g}_i \approx (\hat{y}_i - y_i)\mathbf{x}_i \tag{8.90}$$

It will make the algebra simpler if we assume $y \in \{-1, +1\}$ rather than $y \in \{0, 1\}$. In this case, our prediction becomes

$$\hat{y}_i = \text{sign}(\boldsymbol{\theta}^T \mathbf{x}_i) \tag{8.91}$$

Then if $\hat{y}_i y_i = -1$, we have made an error, but if $\hat{y}_i y_i = +1$, we guessed the right label.

At each step, we update the weight vector by adding on the gradient. The key observation is that, if we predicted correctly, then $\hat{y}_i = y_i$, so the (approximate) gradient is zero and we do not change the weight vector. But if $\mathbf{x}_i$ is misclassified, we update the weights as follows: If $\hat{y}_i = 1$ but $y_i = -1$, then the negative gradient is $-(\hat{y}_i - y_i)\mathbf{x}_i = -2\mathbf{x}_i$; and if $\hat{y}_i = -1$ but $y_i = 1$, then the negative gradient is $-(\hat{y}_i - y_i)\mathbf{x}_i = 2\mathbf{x}_i$. We can absorb the factor of 2 into the learning rate η and just write the update, in the case of a misclassification, as

$$\boldsymbol{\theta}_k = \boldsymbol{\theta}_{k-1} + \eta_k y_i \mathbf{x}_i \tag{8.92}$$

Since it is only the sign of the weights that matter, not the magnitude, we will set $\eta_k = 1$. See Algorithm 11 for the pseudocode.

One can show that this method, known as the **perceptron algorithm** (Rosenblatt 1958), will converge, provided the data is linearly separable, i.e., that there exist parameters $\boldsymbol{\theta}$ such that predicting with $\text{sign}(\boldsymbol{\theta}^T \mathbf{x})$ achieves 0 error on the training set. However, if the data is not linearly separable, the algorithm will not converge, and even if it does converge, it may take a long time. There are much better ways to train logistic regression models (such as using proper SGD, without the gradient approximation, or IRLS, discussed in Section 8.3.4). However, the perceptron algorithm is historically important: it was one of the first machine learning algorithms ever derived (by Frank Rosenblatt in 1957), and was even implemented in analog hardware. In addition, the algorithm can be used to fit models where computing marginals $p(y_i|\mathbf{x}, \boldsymbol{\theta})$ is more expensive than computing the MAP output, $\arg\max_{\mathbf{y}} p(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta})$; this arises in some structured-output classification problems. See Section 19.7 for details.

8.5.5 A Bayesian view

Another approach to online learning is to adopt a Bayesian view. This is conceptually quite simple: we just apply Bayes rule recursively:

$$p(\boldsymbol{\theta}|\mathcal{D}_{1:k}) \propto p(\mathcal{D}_k|\boldsymbol{\theta}) p(\boldsymbol{\theta}|\mathcal{D}_{1:k-1}) \tag{8.93}$$

This has the obvious advantage of returning a posterior instead of just a point estimate. It also allows for the online adaptation of hyper-parameters, which is important since cross-validation cannot be used in an online setting. Finally, it has the (less obvious) advantage that it can be

Algorithm 8.4: Perceptron algorithm

Input: linearly separable data set $\mathbf{x}_i \in \mathbb{R}^D$, $y_i \in \{-1, +1\}$ for $i = 1 : N$;
Initialize $\boldsymbol{\theta}_0$;
$k \leftarrow 0$;
repeat
  $k \leftarrow k + 1$;
  $i \leftarrow k \bmod N$;
  if $\hat{y}_i \neq y_i$ **then**
    $\boldsymbol{\theta}_{k+1} \leftarrow \boldsymbol{\theta}_k + y_i \mathbf{x}_i$
  else
    no-op
until *converged*;

quicker than SGD. To see why, note that by modeling the posterior variance of each parameter in addition to its mean, we effectively associate a different learning rate for each parameter (de Freitas et al. 2000), which is a simple way to model the curvature of the space. These variances can then be adapted using the usual rules of probability theory. By contrast, getting second-order optimization methods to work online is more tricky (see e.g., (Schraudolph et al. 2007; Sunehag et al. 2009; Bordes et al. 2009, 2010)).

As a simple example, in Section 18.2.3 we show how to use the Kalman filter to fit a linear regression model online. Unlike the LMS algorithm, this converges to the optimal (offline) answer in a single pass over the data. An extension which can learn a robust non-linear regression model in an online fashion is described in (Ting et al. 2010). For the GLM case, we can use an assumed density filter (Section 18.5.3), where we approximate the posterior by a Gaussian with a diagonal covariance; the variance terms serve as a per-parameter step-size. See Section 18.5.3.2 for details. Another approach is to use particle filtering (Section 23.5); this was used in (Andrieu et al. 2000) for sequentially learning a kernelized linear/logistic regression model.

8.6 Generative vs discriminative classifiers

In Section 4.2.2, we showed that the posterior over class labels induced by Gaussian discriminant analysis (GDA) has exactly the same form as logistic regression, namely $p(y = 1|\mathbf{x}) = \mathrm{sigm}(\mathbf{w}^T\mathbf{x})$. The decision boundary is therefore a linear function of $\mathbf{x}$ in both cases. Note, however, that many generative models can give rise to a logistic regression posterior, e.g., if each class-conditional density is Poisson, $p(x|y = c) = \mathrm{Poi}(x|\lambda_c)$. So the assumptions made by GDA are much stronger than the assumptions made by logistic regression.

A further difference between these models is the way they are trained. When fitting a discriminative model, we usually maximize the conditional log likelihood $\sum_{i=1}^N \log p(y_i|\mathbf{x}_i, \boldsymbol{\theta})$, whereas when fitting a generative model, we usually maximize the joint log likelihood, $\sum_{i=1}^N \log p(y_i, \mathbf{x}_i|\boldsymbol{\theta})$. It is clear that these can, in general, give different results (see Exercise 4.20).

When the Gaussian assumptions made by GDA are correct, the model will need less training data than logistic regression to achieve a certain level of performance, but if the Gaussian

assumptions are incorrect, logistic regression will do better (Ng and Jordan 2002). This is because discriminative models do not need to model the distribution of the features. This is illustrated in Figure 8.10. We see that the class conditional densities are rather complex; in particular, $p(x|y = 1)$ is a multimodal distribution, which might be hard to estimate. However, the class posterior, $p(y = c|x)$, is a simple sigmoidal function, centered on the threshold value of 0.55. This suggests that, in general, discriminative methods will be more accurate, since their "job" is in some sense easier. However, accuracy is not the only important factor when choosing a method. Below we discuss some other advantages and disadvantages of each approach.

8.6.1 Pros and cons of each approach

- **Easy to fit?** As we have seen, it is usually very easy to fit generative classifiers. For example, in Sections 3.5.1.1 and 4.2.4, we show that we can fit a naive Bayes model and an LDA model by simple counting and averaging. By contrast, logistic regression requires solving a convex optimization problem (see Section 8.3.4 for the details), which is much slower.
- **Fit classes separately?** In a generative classifier, we estimate the parameters of each class conditional density independently, so we do not have to retrain the model when we add more classes. In contrast, in discriminative models, all the parameters interact, so the whole model must be retrained if we add a new class. (This is also the case if we train a generative model to maximize a discriminative objective Salojarvi et al. (2005).)
- **Handle missing features easily?** Sometimes some of the inputs (components of $\mathbf{x}$) are not observed. In a generative classifier, there is a simple method for dealing with this, as we discuss in Section 8.6.2. However, in a discriminative classifier, there is no principled solution to this problem, since the model assumes that $\mathbf{x}$ is always available to be conditioned on (although see (Marlin 2008) for some heuristic approaches).
- **Can handle unlabeled training data?** There is much interest in **semi-supervised learning**, which uses unlabeled data to help solve a supervised task. This is fairly easy to do using generative models (see e.g., (Lasserre et al. 2006; Liang et al. 2007)), but is much harder to do with discriminative models.
- **Symmetric in inputs and outputs?** We can run a generative model "backwards", and infer probable inputs given the output by computing $p(\mathbf{x}|y)$. This is not possible with a discriminative model. The reason is that a generative model defines a joint distribution on $\mathbf{x}$ and y, and hence treats both inputs and outputs symmetrically.
- **Can handle feature preprocessing?** A big advantage of discriminative methods is that they allow us to preprocess the input in arbitrary ways, e.g., we can replace $\mathbf{x}$ with $\boldsymbol{\phi}(\mathbf{x})$, which could be some basis function expansion, as illustrated in Figure 8.9. It is often hard to define a generative model on such pre-processed data, since the new features are correlated in complex ways.
- **Well-calibrated probabilities?** Some generative models, such as naive Bayes, make strong independence assumptions which are often not valid. This can result in very extreme posterior class probabilities (very near 0 or 1). Discriminative models, such as logistic regression, are usually better calibrated in terms of their probability estimates.

We see that there are arguments for and against both kinds of models. It is therefore useful to have both kinds in your "toolbox". See Table 8.1 for a summary of the classification and

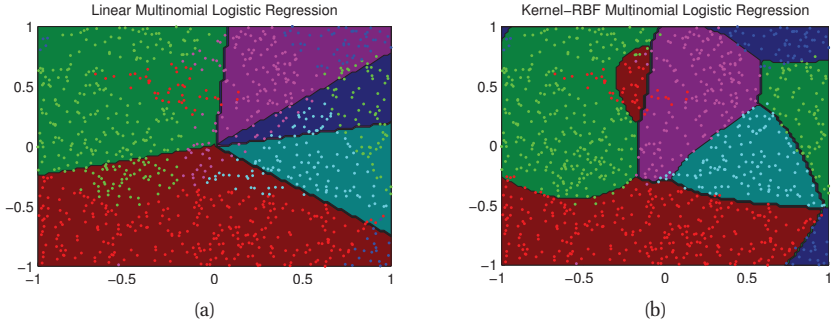


Figure 8.9 (a) Multinomial logistic regression for 5 classes in the original feature space. (b) After basis function expansion, using RBF kernels with a bandwidth of 1, and using all the data points as centers. Figure generated by `logregMultinomKernelDemo`.

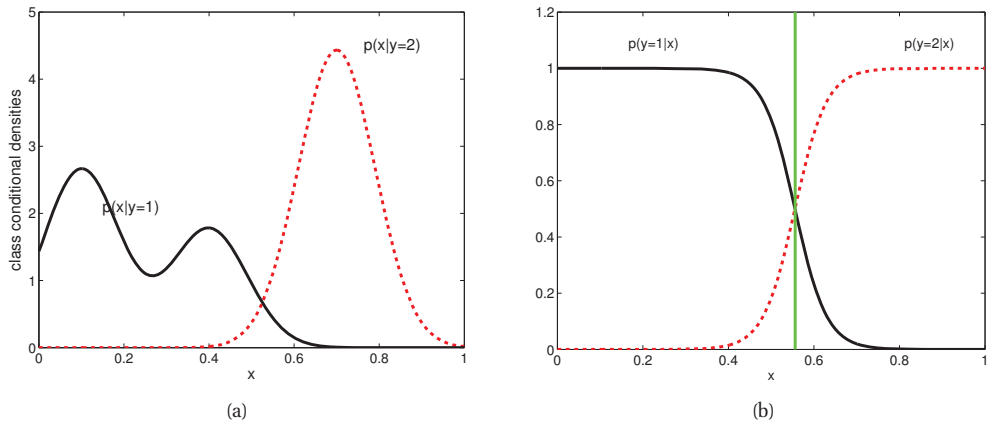


Figure 8.10 The class-conditional densities $p(x|y = c)$ (left) may be more complex than the class posteriors $p(y = c|x)$ (right). Based on Figure 1.27 of (Bishop 2006a). Figure generated by `generativeVsDiscrim`.

regression techniques we cover in this book.

8.6.2 Dealing with missing data

Sometimes some of the inputs (components of $\mathbf{x}$) are not observed; this could be due to a sensor failure, or a failure to complete an entry in a survey, etc. This is called the **missing data problem** (Little. and Rubin 1987). The ability to handle missing data in a principled way is one of the biggest advantages of generative models.

To formalize our assumptions, we can associate a binary response variable $r_i \in \{0, 1\}$, that specifies whether each value $\mathbf{x}_i$ is observed or not. The joint model has the form $p(\mathbf{x}_i, r_i|\boldsymbol{\theta}, \boldsymbol{\phi}) = p(r_i|\mathbf{x}_i, \boldsymbol{\phi})p(\mathbf{x}_i|\boldsymbol{\theta})$, where $\boldsymbol{\phi}$ are the parameters controlling whether the item

Model	Classif/regr	Gen/Discr	Param/Non	Section
Discriminant analysis	Classif	Gen	Param	Sec. 4.2.2, 4.2.4
Naive Bayes classifier	Classif	Gen	Param	Sec. 3.5, 3.5.1.2
Tree-augmented Naive Bayes classifier	Classif	Gen	Param	Sec. 10.2.1
Linear regression	Regr	Discrim	Param	Sec. 1.4.5, 7.3, 7.6,
Logistic regression	Classif	Discrim	Param	Sec. 1.4.6, 8.3.4, 8.4.3, 21.8.1.1
Sparse linear/ logistic regression	Both	Discrim	Param	Ch. 13
Mixture of experts	Both	Discrim	Param	Sec. 11.2.4
Multilayer perceptron (MLP)/ Neural network	Both	Discrim	Param	Ch. 16
Conditional random field (CRF)	Classif	Discrim	Param	Sec. 19.6
K nearest neighbor classifier	Classif	Gen	Non	Sec. 1.4.2, 14.7.3
(Infinite) Mixture Discriminant analysis	Classif	Gen	Non	Sec. 14.7.3
Classification and regression trees (CART)	Both	Discrim	Non	Sec. 16.2
Boosted model	Both	Discrim	Non	Sec. 16.4
Sparse kernelized lin/logreg (SKLR)	Both	Discrim	Non	Sec. 14.3.2
Relevance vector machine (RVM)	Both	Discrim	Non	Sec. 14.3.2
Support vector machine (SVM)	Both	Discrim	Non	Sec. 14.5
Gaussian processes (GP)	Both	Discrim	Non	Ch. 15
Smoothing splines	Regr	Discrim	Non	Section 15.4.6

Table 8.1 List of various models for classification and regression which we discuss in this book. Columns are as follows: Model name; is the model suitable for classification, regression, or both; is the model generative or discriminative; is the model parametric or non-parametric; list of sections in book which discuss the model. See also `http://pmtk3.googlecode.com/svn/trunk/docs/tutorial/html/tutSupervised.html` for the PMTK equivalents of these models. Any generative probabilistic model (e.g., HMMs, Boltzmann machines, Bayesian networks, etc.) can be turned into a classifier by using it as a class conditional density.

is observed or not. If we assume $p(r_i|\mathbf{x}_i, \boldsymbol{\phi}) = p(r_i|\boldsymbol{\phi})$, we say the data is **missing completely at random** or **MCAR**. If we assume $p(r_i|\mathbf{x}_i, \boldsymbol{\phi}) = p(r_i|\mathbf{x}_i^o, \boldsymbol{\phi})$, where $\mathbf{x}_i^o$ is the observed part of $\mathbf{x}_i$, we say the data is **missing at random** or **MAR**. If neither of these assumptions hold, we say the data is **not missing at random** or **NMAR**. In this case, we have to model the missing data mechanism, since the pattern of missingness is informative about the values of the missing data and the corresponding parameters. This is the case in most collaborative filtering problems, for example. See e.g., (Marlin 2008) for further discussion. We will henceforth assume the data is MAR.

When dealing with missing data, it is helpful to distinguish the cases when there is missingness only at test time (so the training data is **complete data**), from the harder case when there is missingness also at training time. We will discuss these two cases below. Note that the class label is always missing at test time, by definition; if the class label is also sometimes missing at training time, the problem is called semi-supervised learning.

8.6.2.1 Missing data at test time

In a generative classifier, we can handle features that are MAR by marginalizing them out. For example, if we are missing the value of x_1, we can compute

$$p(y=c|\mathbf{x}_{2:D},\boldsymbol{\theta}) \propto p(y=c|\boldsymbol{\theta})p(\mathbf{x}_{2:D}|y=c,\boldsymbol{\theta}) \tag{8.94}$$

$$= p(y=c|\boldsymbol{\theta})\sum_{x_1} p(x_1,\mathbf{x}_{2:D}|y=c,\boldsymbol{\theta}) \tag{8.95}$$

If we make the naive Bayes assumption, the marginalization can be performed as follows:

$$\sum_{x_1} p(x_1,x_{2:D}|y=c,\boldsymbol{\theta}) = \left[\sum_{x_1} p(x_1|\boldsymbol{\theta}_{1c})\right]\prod_{j=2}^{D} p(x_j|\boldsymbol{\theta}_{jc}) = \prod_{j=2}^{D} p(x_j|\boldsymbol{\theta}_{jc}) \tag{8.96}$$

where we exploited the fact that $\sum_{x_1} p(x_1|y=c,\boldsymbol{\theta})=1$. Hence in a naive Bayes classifier, we can simply ignore missing features at test time. Similarly, in discriminant analysis, no matter what regularization method was used to estimate the parameters, we can always analytically marginalize out the missing variables (see Section 4.3):

$$p(\mathbf{x}_{2:D}|y=c,\boldsymbol{\theta}) = \mathcal{N}(\mathbf{x}_{2:D}|\boldsymbol{\mu}_{c,2:D},\boldsymbol{\Sigma}_{c,2:D,2:D}) \tag{8.97}$$

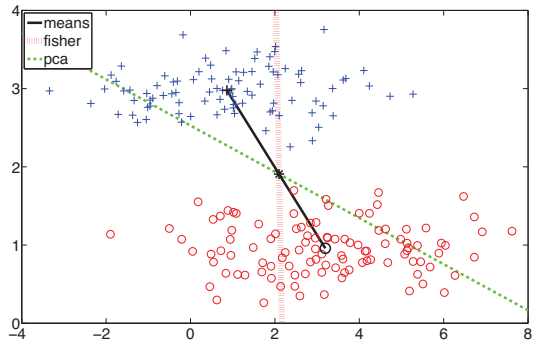
8.6.2.2 Missing data at training time

Missing data at training time is harder to deal with. In particular, computing the MLE or MAP estimate is no longer a simple optimization problem, for reasons discussed in Section 11.3.2. However, soon we will study are a variety of more sophisticated algorithms (such as EM algorithm, in Section 11.4) for finding approximate ML or MAP estimates in such cases.

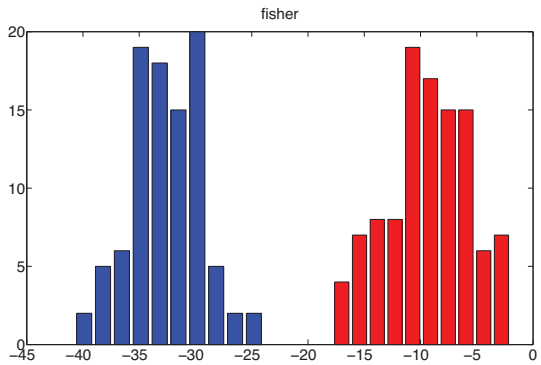
8.6.3 Fisher's linear discriminant analysis (FLDA) *

Discriminant analysis is a generative approach to classification, which requires fitting an MVN to the features. As we have discussed, this can be problematic in high dimensions. An alternative approach is to reduce the dimensionality of the features $\mathbf{x}\in\mathbb{R}^D$ and then fit an MVN to the resulting low-dimensional features $\mathbf{z}\in\mathbb{R}^L$. The simplest approach is to use a linear projection matrix, $\mathbf{z}=\mathbf{W}\mathbf{x}$, where $\mathbf{W}$ is a $L\times D$ matrix. One approach to finding $\mathbf{W}$ would be to use PCA (Section 12.2); the result would be very similar to RDA (Section 4.2.6), since SVD and PCA are essentially equivalent. However, PCA is an unsupervised technique that does not take class labels into account. Thus the resulting low dimensional features are not necessarily optimal for classification, as illustrated in Figure 8.11. An alternative approach is to find the matrix $\mathbf{W}$ such that the low-dimensional data can be classified as well as possible using a Gaussian class-conditional density model. The assumption of Gaussianity is reasonable since we are computing linear combinations of (potentially non-Gaussian) features. This approach is called **Fisher's linear discriminant analysis**, or **FLDA**.

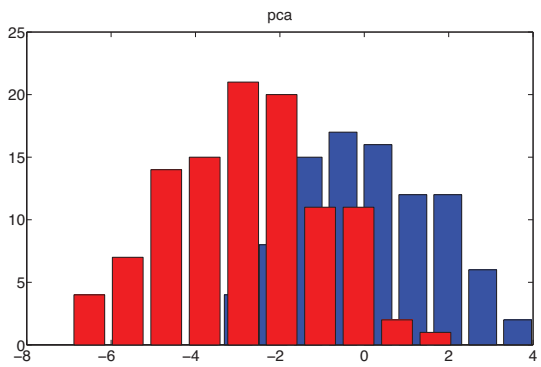
FLDA is an interesting hybrid of discriminative and generative techniques. The drawback of this technique is that it is restricted to using $L\leq C-1$ dimensions, regardless of D, for reasons that we will explain below. In the two-class case, this means we are seeking a single vector $\mathbf{w}$ onto which we can project the data. Below we derive the optimal $\mathbf{w}$ in the two-class case. We



(a)



(b)



(c)

Figure 8.11 Example of Fisher's linear discriminant. (a) Two class data in 2D. Dashed green line = first principal basis vector. Dotted red line = Fisher's linear discriminant vector. Solid black line joins the class-conditional means. (b) Projection of points onto Fisher's vector shows good class separation. (c) Projection of points onto PCA vector shows poor class separation. Figure generated by `fisherLDAdemo`.

then generalize to the multi-class case, and finally we give a probabilistic interpretation of this technique.

8.6.3.1 Derivation of the optimal 1d projection

We now derive this optimal direction $\mathbf{w}$, for the two-class case, following the presentation of (Bishop 2006b, Sec 4.1.4). Define the class-conditional means as

$$\boldsymbol{\mu}_1 = \frac{1}{N_1} \sum_{i:y_i=1} \mathbf{x}_i, \ \boldsymbol{\mu}_2 = \frac{1}{N_2} \sum_{i:y_i=2} \mathbf{x}_i \tag{8.98}$$

Let $m_k = \mathbf{w}^T \boldsymbol{\mu}_k$ be the projection of each mean onto the line $\mathbf{w}$. Also, let $z_i = \mathbf{w}^T \mathbf{x}_i$ be the projection of the data onto the line. The variance of the projected points is proportional to

$$s_k^2 = \sum_{i:y_i=k} (z_i - m_k)^2 \tag{8.99}$$

The goal is to find $\mathbf{w}$ such that we maximize the distance between the means, $m_2 - m_1$, while also ensuring the projected clusters are "tight":

$$J(\mathbf{w}) = \frac{(m_2 - m_1)^2}{s_1^2 + s_2^2} \tag{8.100}$$

We can rewrite the right hand side of the above in terms of $\mathbf{w}$ as follows

$$J(\mathbf{w}) = \frac{\mathbf{w}^T \mathbf{S}_B \mathbf{w}}{\mathbf{w}^T \mathbf{S}_W \mathbf{w}} \tag{8.101}$$

where $\mathbf{S}_B$ is the between-class scatter matrix given by

$$\mathbf{S}_B = (\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)^T \tag{8.102}$$

and $\mathbf{S}_W$ is the within-class scatter matrix, given by

$$\mathbf{S}_W = \sum_{i:y_i=1} (\mathbf{x}_i - \boldsymbol{\mu}_1)(\mathbf{x}_i - \boldsymbol{\mu}_1)^T + \sum_{i:y_i=2} (\mathbf{x}_i - \boldsymbol{\mu}_2)(\mathbf{x}_i - \boldsymbol{\mu}_2)^T \tag{8.103}$$

To see this, note that

$$\mathbf{w}^T \mathbf{S}_B \mathbf{w} = \mathbf{w}^T (\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)^T \mathbf{w} = (m_2 - m_1)(m_2 - m_1) \tag{8.104}$$

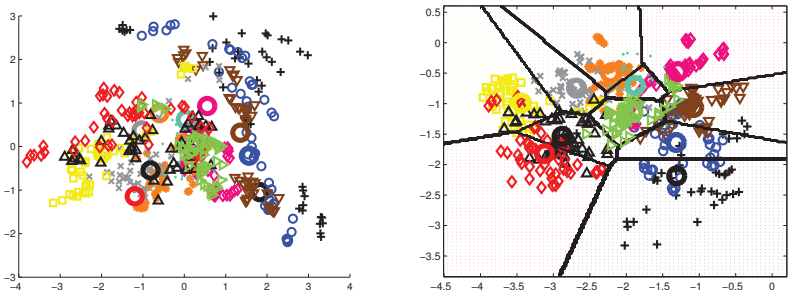
and

$$\mathbf{w}^T \mathbf{S}_W \mathbf{w} = \sum_{i:y_i=1} \mathbf{w}^T (\mathbf{x}_i - \boldsymbol{\mu}_1)(\mathbf{x}_i - \boldsymbol{\mu}_1)^T \mathbf{w} + \sum_{i:y_i=2} \mathbf{w}^T (\mathbf{x}_i - \boldsymbol{\mu}_2)(\mathbf{x}_i - \boldsymbol{\mu}_2)^T \mathbf{w} \tag{8.105}$$

$$= \sum_{i:y_i=1} (z_i - m_1)^2 + \sum_{i:y_i=2} (z_i - m_2)^2 \tag{8.106}$$

Equation 8.101 is a ratio of two scalars; we can take its derivative with respect to $\mathbf{w}$ and equate to zero. One can show (Exercise 12.6) that that $J(\mathbf{w})$ is maximized when

$$\mathbf{S}_B \mathbf{w} = \lambda \mathbf{S}_W \mathbf{w} \tag{8.107}$$



(a) (b)

Figure 8.12 (a) PCA projection of vowel data to 2d. (b) FLDA projection of vowel data to 2d. We see there is better class separation in the FLDA case. Based on Figure 4.11 of (Hastie et al. 2009). Figure generated by `fisherDiscrimVowelDemo`, by Hannes Bretschneider.

where

$$\lambda = \frac{\mathbf{w}^T \mathbf{S}_B \mathbf{w}}{\mathbf{w}^T \mathbf{S}_W \mathbf{w}} \tag{8.108}$$

Equation 8.107 is called a **generalized eigenvalue** problem. If $\mathbf{S}_W$ is invertible, we can convert it to a regular eigenvalue problem:

$$\mathbf{S}_W^{-1} \mathbf{S}_B \mathbf{w} = \lambda \mathbf{w} \tag{8.109}$$

However, in the two class case, there is a simpler solution. In particular, since

$$\mathbf{S}_B \mathbf{w} = (\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)^T \mathbf{w} = (\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)(m_2 - m_1) \tag{8.110}$$

then, from Equation 8.109 we have

$$\lambda \, \mathbf{w} = \mathbf{S}_W^{-1}(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)(m_2 - m_1) \tag{8.111}$$

$$\mathbf{w} \propto \mathbf{S}_W^{-1}(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1) \tag{8.112}$$

Since we only care about the directionality, and not the scale factor, we can just set

$$\mathbf{w} = \mathbf{S}_W^{-1}(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1) \tag{8.113}$$

This is the optimal solution in the two-class case. If $\mathbf{S}_W \propto \mathbf{I}$, meaning the pooled covariance matrix is isotropic, then $\mathbf{w}$ is proportional to the vector that joins the class means. This is an intuitively reasonable direction to project onto, as shown in Figure 8.11.

8.6.3.2 Extension to higher dimensions and multiple classes

We can extend the above idea to multiple classes, and to higher dimensional subspaces, by finding a projection *matrix* $\mathbf{W}$ which maps from D to L so as to maximize

$$J(\mathbf{W}) = \frac{|\mathbf{W}\boldsymbol{\Sigma}_B \mathbf{W}^T|}{|\mathbf{W}\Sigma_W \mathbf{W}^T|} \tag{8.114}$$

where

$$\boldsymbol{\Sigma}_B \triangleq \sum_c \frac{N_c}{N}(\boldsymbol{\mu}_c - \boldsymbol{\mu})(\boldsymbol{\mu}_c - \boldsymbol{\mu})^T \tag{8.115}$$

$$\boldsymbol{\Sigma}_W \triangleq \sum_c \frac{N_c}{N}\boldsymbol{\Sigma}_c \tag{8.116}$$

$$\boldsymbol{\Sigma}_c \triangleq \frac{1}{N_c}\sum_{i:y_i=c}(\mathbf{x}_i - \boldsymbol{\mu}_c)(\mathbf{x}_i - \boldsymbol{\mu}_c)^T \tag{8.117}$$

The solution can be shown to be

$$\mathbf{W} = \boldsymbol{\Sigma}_W^{-\frac{1}{2}}\mathbf{U} \tag{8.118}$$

where $\mathbf{U}$ are the L leading eigenvectors of $\boldsymbol{\Sigma}_W^{-\frac{1}{2}}\boldsymbol{\Sigma}_B\boldsymbol{\Sigma}_W^{-\frac{1}{2}}$, assuming $\boldsymbol{\Sigma}_W$ is non-singular. (If it is singular, we can first perform PCA on all the data.)

Figure 8.12 gives an example of this method applied to some $D = 10$ dimensional speech data, representing $C = 11$ different vowel sounds. We see that FLDA gives better class separation than PCA.

Note that FLDA is restricted to finding at most a $L \leq C - 1$ dimensional linear subspace, no matter how large D, because the rank of the between class covariance matrix $\boldsymbol{\Sigma}_B$ is $C - 1$. (The -1 term arises because of the $\boldsymbol{\mu}$ term, which is a linear function of the $\boldsymbol{\mu}_c$.) This is a rather severe restriction which limits the usefulness of FLDA.

8.6.3.3 Probabilistic interpretation of FLDA *

To find a valid probabilistic interpretation of FLDA, we follow the approach of (Kumar and Andreo 1998; Zhou et al. 2009). They proposed a model known as **heteroscedastic LDA** (HLDA), which works as follows. Let $\mathbf{W}$ be a $D \times D$ invertible matrix, and let $\mathbf{z}_i = \mathbf{W}\mathbf{x}_i$ be a transformed version of the data. We now fit full covariance Gaussians to the transformed data, one per class, but with the constraint that only the first L components will be class-specific; the remaining $H = D - L$ components will be shared across classes, and will thus not be discriminative. That is, we use

$$p(\mathbf{z}_i|\boldsymbol{\theta}, y_i = c) = \mathcal{N}(\mathbf{z}_i|\boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c) \tag{8.119}$$

$$\boldsymbol{\mu}_c \triangleq (\mathbf{m}_c; \mathbf{m}_0) \tag{8.120}$$

$$\boldsymbol{\Sigma}_c \triangleq \begin{pmatrix} \mathbf{S}_c & \mathbf{0} \\ \mathbf{0} & \mathbf{S}_0 \end{pmatrix} \tag{8.121}$$

where $\mathbf{m}_0$ is the shared H dimensional mean and $\mathbf{S}_0$ is the shared $H \times H$ covariace. The pdf of the original (untransformed) data is given by

$$p(\mathbf{x}_i|y_i = c, \mathbf{W}, \boldsymbol{\theta}) = |\mathbf{W}|\,\mathcal{N}(\mathbf{W}\mathbf{x}_i|\boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c) \tag{8.122}$$

$$= |\mathbf{W}|\,\mathcal{N}(\mathbf{W}_L\mathbf{x}_i|\mathbf{m}_c, \mathbf{S}_c)\,\mathcal{N}(\mathbf{W}_H\mathbf{x}_i|\mathbf{m}_0, \mathbf{S}_0) \tag{8.123}$$

where $\mathbf{W} = \begin{pmatrix} \mathbf{W}_L \\ \mathbf{W}_H \end{pmatrix}$. For fixed $\mathbf{W}$, it is easy to derive the MLE for $\boldsymbol{\theta}$. One can then optimize $\mathbf{W}$ using gradient methods.

In the special case that the $\mathbf{\Sigma}_c$ are diagonal, there is a closed-form solution for $\mathbf{W}$ (Gales 1999). And in the special case the $\mathbf{\Sigma}_c$ are all equal, we recover classical LDA (Zhou et al. 2009).

In view of this this result, it should be clear that HLDA will outperform LDA if the class covariances are not equal within the discriminative subspace (i.e., if the assumption that $\mathbf{\Sigma}_c$ is independent of c is a poor assumption). This is easy to demonstrate on synthetic data, and is also the case on more challenging tasks such as speech recognition (Kumar and Andreo 1998). Furthermore, we can extend the model by allowing each class to use its own projection matrix; this is known as **multiple LDA** (Gales 2002).

Exercises

Exercise 8.1 Spam classification using logistic regression

Consider the email spam data set discussed on p300 of (Hastie et al. 2009). This consists of 4601 email messages, from which 57 features have been extracted. These are as follows:

- 48 features, in $[0, 100]$, giving the percentage of words in a given message which match a given word on the list. The list contains words such as "business", "free", "george", etc. (The data was collected by George Forman, so his name occurs quite a lot.)
- 6 features, in $[0, 100]$, giving the percentage of characters in the email that match a given character on the list. The characters are ; ([! $ #
- Feature 55: The average length of an uninterrupted sequence of capital letters (max is 40.3, mean is 4.9)
- Feature 56: The length of the longest uninterrupted sequence of capital letters (max is 45.0, mean is 52.6)
- Feature 57: The sum of the lengts of uninterrupted sequence of capital letters (max is 25.6, mean is 282.2)

Load the data from `spamData.mat`, which contains a training set (of size 3065) and a test set (of size 1536).

One can imagine performing several kinds of preprocessing to this data. Try each of the following separately:

a. Standardize the columns so they all have mean 0 and unit variance.

b. Transform the features using $\log(x_{ij} + 0.1)$.

c. Binarize the features using $\mathbb{I}(x_{ij} > 0)$.

For each version of the data, fit a logistic regression model. Use cross validation to choose the strength of the ℓ_2 regularizer. Report the mean error rate on the training and test sets. You should get numbers similar to this:

method	train	test
stnd	0.082	0.079
log	0.052	0.059
binary	0.065	0.072

(The precise values will depend on what regularization value you choose.) Turn in your code and numerical results.

(See also Exercise 8.2.

Exercise 8.2 Spam classification using naive Bayes

We will re-examine the dataset from Exercise 8.1.

a. Use `naiveBayesFit` and `naiveBayesPredict` on the binarized spam data. What is the training and test error? (You can try different settings of the pseudocount α if you like (this corresponds to the $\text{Beta}(\alpha, \alpha)$ prior each θ_{jc}), although the default of $\alpha = 1$ is probably fine.) Turn in your error rates.

b. Modify the code so it can handle real-valued features. Use a Gaussian density for each feature; fit it with maximum likelihood. What are the training and test error rates on the standardized data and the log transformed data? Turn in your 4 error rates and code.

Exercise 8.3 Gradient and Hessian of log-likelihood for logistic regression

a. Let $\sigma(a) = \frac{1}{1+e^{-a}}$ be the sigmoid function. Show that

$$\frac{d\sigma(a)}{da} = \sigma(a)(1 - \sigma(a)) \tag{8.124}$$

b. Using the previous result and the chain rule of calculus, derive an expression for the gradient of the log likelihood (Equation 8.5).

c. The Hessian can be written as $\mathbf{H} = \mathbf{X}^T\mathbf{S}\mathbf{X}$, where $\mathbf{S} \triangleq \text{diag}(\mu_1(1-\mu_1), \ldots, \mu_n(1-\mu_n))$. Show that $\mathbf{H}$ is positive definite. (You may assume that $0 < \mu_i < 1$, so the elements of $\mathbf{S}$ will be strictly positive, and that $\mathbf{X}$ is full rank.)

Exercise 8.4 Gradient and Hessian of log-likelihood for multinomial logistic regression

a. Let $\mu_{ik} = \mathcal{S}(\boldsymbol{\eta}_i)_k$. Prove that the Jacobian of the softmax is

$$\frac{\partial \mu_{ik}}{\partial \eta_{ij}} = \mu_{ik}(\delta_{kj} - \mu_{ij}) \tag{8.125}$$

where $\delta_{kj} = I(k = j)$.

b. Hence show that

$$\nabla_{\mathbf{w}_c} \ell \quad = \quad \sum_i (y_{ic} - \mu_{ic})\mathbf{x}_i \tag{8.126}$$

Hint: use the chain rule and the fact that $\sum_c y_{ic} = 1$.

c. Show that the block submatrix of the Hessian for classes c and c' is given by

$$\mathbf{H}_{c,c'} \quad = \quad -\sum_i \mu_{ic}(\delta_{c,c'} - \mu_{i,c'})\mathbf{x}_i\mathbf{x}_i^T \tag{8.127}$$

Exercise 8.5 Symmetric version of ℓ_2 regularized multinomial logistic regression

(Source: Ex 18.3 of (Hastie et al. 2009).)

Multiclass logistic regression has the form

$$p(y = c|\mathbf{x}, \mathbf{W}) = \frac{\exp(w_{c0} + \mathbf{w}_c^T\mathbf{x})}{\sum_{k=1}^{C} \exp(w_{k0} + \mathbf{w}_k^T\mathbf{x})} \tag{8.128}$$

where $\mathbf{W}$ is a $(D+1) \times C$ weight matrix. We can arbitrarily define $\mathbf{w}_c = \mathbf{0}$ for one of the classes, say $c = C$, since $p(y = C|\mathbf{x}, \mathbf{W}) = 1 - \sum_{c=1}^{C-1} p(y = c|\mathbf{x}, \mathbf{w})$. In this case, the model has the form

$$p(y = c|\mathbf{x}, \mathbf{W}) = \frac{\exp(w_{c0} + \mathbf{w}_c^T\mathbf{x})}{1 + \sum_{k=1}^{C-1} \exp(w_{k0} + \mathbf{w}_k^T\mathbf{x})} \tag{8.129}$$

If we don't "clamp" one of the vectors to some constant value, the parameters will be unidentifiable. However, suppose we don't clamp $\mathbf{w}_c = \mathbf{0}$, so we are using Equation 8.128, but we add ℓ_2 regularization by optimizing

$$\sum_{i=1}^{N} \log p(y_i|\mathbf{x}_i, \mathbf{W}) - \lambda \sum_{c=1}^{C} ||\mathbf{w}_c||_2^2 \tag{8.130}$$

Show that at the optimum we have $\sum_{c=1}^{C} \hat{w}_{cj} = 0$ for $j = 1 : D$. (For the unregularized $\hat{w}_{c0}$ terms, we still need to enforce that $w_{0C} = 0$ to ensure identifiability of the offset.)

Exercise 8.6 Elementary properties of ℓ_2 regularized logistic regression

(Source: Jaaakkola.). Consider minimizing

$$J(\mathbf{w}) = -\ell(\mathbf{w}, \mathcal{D}_{\text{train}}) + \lambda ||\mathbf{w}||_2^2 \tag{8.131}$$

where

$$\ell(\mathbf{w}, \mathcal{D}) = \frac{1}{|\mathcal{D}|} \sum_{i \in \mathcal{D}} \log \sigma(y_i \mathbf{x}_i^T \mathbf{w}) \tag{8.132}$$

is the average log-likelihood on data set $\mathcal{D}$, for $y_i \in \{-1, +1\}$. Answer the following true/ false questions.

a. $J(\mathbf{w})$ has multiple locally optimal solutions: T/F?
b. Let $\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} J(\mathbf{w})$ be a global optimum. $\hat{\mathbf{w}}$ is sparse (has many zero entries): T/F?
c. If the training data is linearly separable, then some weights w_j might become infinite if $\lambda = 0$: T/F?
d. $\ell(\hat{\mathbf{w}}, \mathcal{D}_{\text{train}})$ always increases as we increase λ: T/F?
e. $\ell(\hat{\mathbf{w}}, \mathcal{D}_{\text{test}})$ always increases as we increase λ: T/F?

Exercise 8.7 Regularizing separate terms in 2d logistic regression

(Source: Jaaakkola.)

a. Consider the data in Figure 8.13, where we fit the model $p(y = 1|\mathbf{x}, \mathbf{w}) = \sigma(w_0 + w_1 x_1 + w_2 x_2)$. Suppose we fit the model by maximum likelihood, i.e., we minimize

$$J(\mathbf{w}) = -\ell(\mathbf{w}, \mathcal{D}_{\text{train}}) \tag{8.133}$$

where $\ell(\mathbf{w}, \mathcal{D}_{\text{train}})$ is the log likelihood on the training set. Sketch a possible decision boundary corresponding to $\hat{\mathbf{w}}$. (Copy the figure first (a rough sketch is enough), and then superimpose your answer on your copy, since you will need multiple versions of this figure). Is your answer (decision boundary) unique? How many classification errors does your method make on the training set?

b. Now suppose we regularize only the w_0 parameter, i.e., we minimize

$$J_0(\mathbf{w}) = -\ell(\mathbf{w}, \mathcal{D}_{\text{train}}) + \lambda w_0^2 \tag{8.134}$$

Suppose λ is a very large number, so we regularize w_0 all the way to 0, but all other parameters are unregularized. Sketch a possible decision boundary. How many classification errors does your method make on the training set? Hint: consider the behavior of simple linear regression, $w_0 + w_1 x_1 + w_2 x_2$ when $x_1 = x_2 = 0$.

c. Now suppose we heavily regularize only the w_1 parameter, i.e., we minimize

$$J_1(\mathbf{w}) = -\ell(\mathbf{w}, \mathcal{D}_{\text{train}}) + \lambda w_1^2 \tag{8.135}$$

Sketch a possible decision boundary. How many classification errors does your method make on the training set?

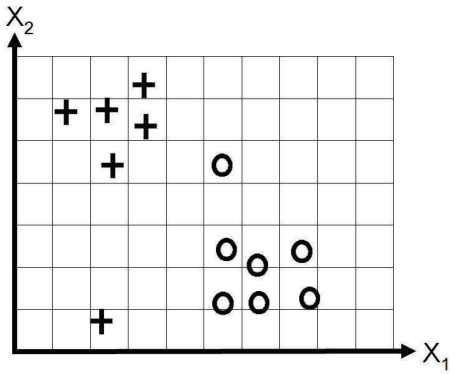


Figure 8.13 Data for logistic regression question.

d. Now suppose we heavily regularize only the w_2 parameter. Sketch a possible decision boundary. How many classification errors does your method make on the training set?