7 *Linear regression*

7.1 Introduction

Linear regression is the "work horse" of statistics and (supervised) machine learning. When augmented with kernels or other forms of basis function expansion, it can model also non-linear relationships. And when the Gaussian output is replaced with a Bernoulli or multinoulli distribution, it can be used for classification, as we will see below. So it pays to study this model in detail.

7.2 Model specification

As we discussed in Section 1.4.5, linear regression is a model of the form

$$p(y|\mathbf{x}, \boldsymbol{\theta}) = \mathcal{N}(y|\mathbf{w}^T\mathbf{x}, \sigma^2) \tag{7.1}$$

Linear regression can be made to model non-linear relationships by replacing $\mathbf{x}$ with some non-linear function of the inputs, $\boldsymbol{\phi}(\mathbf{x})$. That is, we use

$$p(y|\mathbf{x}, \boldsymbol{\theta}) = \mathcal{N}(y|\mathbf{w}^T\boldsymbol{\phi}(\mathbf{x}), \sigma^2) \tag{7.2}$$

This is known as **basis function expansion**. (Note that the model is still linear in the parameters $\mathbf{w}$, so it is still called linear regression; the importance of this will become clear below.) A simple example are polynomial basis functions, where the model has the form

$$\boldsymbol{\phi}(x) = [1, x, x^2, \ldots, x^d] \tag{7.3}$$

Figure 1.18 illustrates the effect of changing d: increasing the degree d allows us to create increasingly complex functions.

We can also apply linear regression to more than 1 input. For example, consider modeling temperature as a function of location. Figure 7.1(a) plots $\mathbb{E}[y|\mathbf{x}] = w_0 + w_1x_1 + w_2x_2$, and Figure 7.1(b) plots $\mathbb{E}[y|\mathbf{x}] = w_0 + w_1x_1 + w_2x_2 + w_3x_1^2 + w_4x_2^2$.

7.3 Maximum likelihood estimation (least squares)

A common way to esitmate the parameters of a statistical model is to compute the MLE, which is defined as

$$\hat{\boldsymbol{\theta}} \triangleq \arg\max_{\boldsymbol{\theta}} \log p(\mathcal{D}|\boldsymbol{\theta}) \tag{7.4}$$

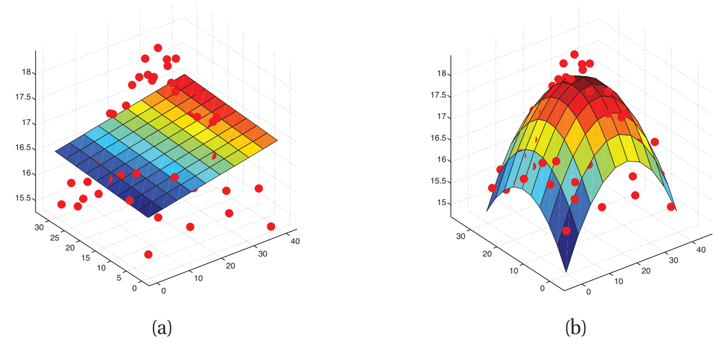

Figure 7.1 Linear regression applied to 2d data. Vertical axis is temperature, horizontal axes are location within a room. Data was collected by some remote sensing motes at Intel's lab in Berkeley, CA (data courtesy of Romain Thibaux). (a) The fitted plane has the form $\hat{f}(\mathbf{x}) = w_0 + w_1 x_1 + w_2 x_2$. (b) Temperature data is fitted with a quadratic of the form $\hat{f}(\mathbf{x}) = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_1^2 + w_4 x_2^2$. Produced by `surfaceFitDemo`.

It is common to assume the training examples are independent and identically distributed, commonly abbreviated to **iid**. This means we can write the log-likelihood as follows:

$$\ell(\boldsymbol{\theta}) \triangleq \log p(\mathcal{D}|\boldsymbol{\theta}) = \sum_{i=1}^{N} \log p(y_i|\mathbf{x}_i, \boldsymbol{\theta}) \tag{7.5}$$

Instead of maximizing the log-likelihood, we can equivalently minimize the **negative log likelihood** or **NLL**:

$$\text{NLL}(\boldsymbol{\theta}) \triangleq -\sum_{i=1}^{N} \log p(y_i|\mathbf{x}_i, \boldsymbol{\theta}) \tag{7.6}$$

The NLL formulation is sometimes more convenient, since many optimization software packages are designed to find the minima of functions, rather than maxima.

Now let us apply the method of MLE to the linear regression setting. Inserting the definition of the Gaussian into the above, we find that the log likelihood is given by

$$\begin{aligned} \ell(\boldsymbol{\theta}) &= \sum_{i=1}^{N} \log \left[\left(\frac{1}{2\pi\sigma^2} \right)^{\frac{1}{2}} \exp \left(-\frac{1}{2\sigma^2} (y_i - \mathbf{w}^T \mathbf{x}_i)^2 \right) \right] && (7.7) \\ &= \frac{-1}{2\sigma^2} RSS(\mathbf{w}) - \frac{N}{2} \log(2\pi\sigma^2) && (7.8) \end{aligned}$$

RSS stands for **residual sum of squares** and is defined by

$$\text{RSS}(\mathbf{w}) \triangleq \sum_{i=1}^{N} (y_i - \mathbf{w}^T \mathbf{x}_i)^2 \tag{7.9}$$

The RSS is also called the **sum of squared errors**, or SSE, and SSE$/N$ is called the **mean squared error** or **MSE**. It can also be written as the square of the ℓ_2 **norm** of the vector of

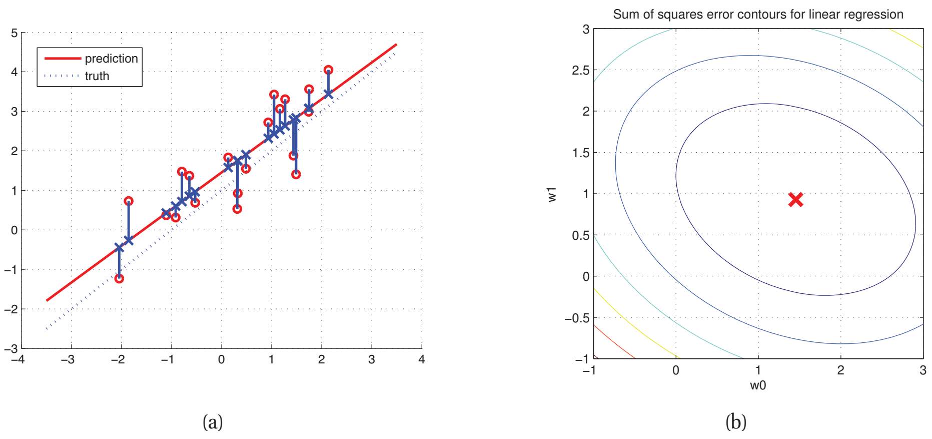

Figure 7.2 (a) In linear least squares, we try to minimize the sum of squared distances from each training point (denoted by a red circle) to its approximation (denoted by a blue cross), that is, we minimize the sum of the lengths of the little vertical blue lines. The red diagonal line represents $\hat{y}(x) = w_0 + w_1 x$, which is the least squares regression line. Note that these residual lines are not perpendicular to the least squares line, in contrast to Figure 12.5. Figure generated by `residualsDemo`. (b) Contours of the RSS error surface for the same example. The red cross represents the MLE, $\mathbf{w} = (1.45, 0.93)$. Figure generated by `contoursSSEdemo`.

residual errors:

$$\text{RSS}(\mathbf{w}) = ||\boldsymbol{\epsilon}||_2^2 = \sum_{i=1}^{N} \epsilon_i^2 \tag{7.10}$$

where $\epsilon_i = (y_i - \mathbf{w}^T \mathbf{x}_i)$.

We see that the MLE for $\mathbf{w}$ is the one that minimizes the RSS, so this method is known as **least squares**. This method is illustrated in Figure 7.2(a). The training data (x_i, y_i) are shown as red circles, the estimated values $(x_i, \hat{y}_i)$ are shown as blue crosses, and the residuals $\epsilon_i = y_i - \hat{y}_i$ are shown as vertical blue lines. The goal is to find the setting of the parameters (the slope w_1 and intercept w_0) such that the resulting red line minimizes the sum of squared residuals (the lengths of the vertical blue lines).

In Figure 7.2(b), we plot the NLL surface for our linear regression example. We see that it is a quadratic "bowl" with a unique minimum, which we now derive. (Importantly, this is true even if we use basis function expansion, such as polynomials, because the NLL is still *linear in the parameters* $\mathbf{w}$, even if it is not linear in the inputs $\mathbf{x}$.)

7.3.1 Derivation of the MLE

First, we rewrite the objective in a form that is more amenable to differentiation:

$$\text{NLL}(\mathbf{w}) = \frac{1}{2}(\mathbf{y} - \mathbf{X}\mathbf{w})^T(\mathbf{y} - \mathbf{X}\mathbf{w}) = \frac{1}{2}\mathbf{w}^T(\mathbf{X}^T\mathbf{X})\mathbf{w} - \mathbf{w}^T(\mathbf{X}^T\mathbf{y}) \tag{7.11}$$

where

$$\mathbf{X}^T\mathbf{X} = \sum_{i=1}^{N} \mathbf{x}_i\mathbf{x}_i^T = \sum_{i=1}^{N} \begin{pmatrix} x_{i,1}^2 & \cdots & x_{i,1}x_{i,D} \\ & \ddots & \\ x_{i,D}x_{i,1} & \cdots & x_{i,D}^2 \end{pmatrix} \tag{7.12}$$

is the **sum of squares** matrix and

$$\mathbf{X}^T\mathbf{y} = \sum_{i=1}^{N} \mathbf{x}_i y_i. \tag{7.13}$$

Using results from Equation 4.10, we see that the gradient of this is given by

$$\mathbf{g}(\mathbf{w}) \quad = \quad [\mathbf{X}^T\mathbf{X}\mathbf{w} - \mathbf{X}^T\mathbf{y}] = \sum_{i=1}^{N} \mathbf{x}_i(\mathbf{w}^T\mathbf{x}_i - y_i) \tag{7.14}$$

Equating to zero we get

$$\mathbf{X}^T\mathbf{X}\mathbf{w} \quad = \quad \mathbf{X}^T\mathbf{y} \tag{7.15}$$

This is known as the **normal equation**. The corresponding solution $\hat{\mathbf{w}}$ to this linear system of equations is called the **ordinary least squares** or **OLS** solution, which is given by

$$\boxed{\hat{\mathbf{w}}_{OLS} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}} \tag{7.16}$$

7.3.2 Geometric interpretation

This equation has an elegant geometrical intrepretation, as we now explain. We assume $N > D$, so we have more examples than features. The columns of $\mathbf{X}$ define a linear subspace of dimensionality D which is embedded in N dimensions. Let the j'th column be $\tilde{\mathbf{x}}_j$, which is a vector in $\mathbb{R}^N$. (This should not be confused with $\mathbf{x}_i \in \mathbb{R}^D$, which represents the i'th data case.) Similarly, $\mathbf{y}$ is a vector in $\mathbb{R}^N$. For example, suppose we have $N = 3$ examples in $D = 2$ dimensions:

$$\mathbf{X} = \begin{pmatrix} 1 & 2 \\ 1 & -2 \\ 1 & 2 \end{pmatrix}, \quad \mathbf{y} = \begin{pmatrix} 8.8957 \\ 0.6130 \\ 1.7761 \end{pmatrix} \tag{7.17}$$

These vectors are illustrated in Figure 7.3.

We seek a vector $\hat{\mathbf{y}} \in \mathbb{R}^N$ that lies in this linear subspace and is as close as possible to $\mathbf{y}$, i.e., we want to find

$$\underset{\hat{\mathbf{y}} \in \text{span}(\{\tilde{\mathbf{x}}_1,\ldots,\tilde{\mathbf{x}}_D\})}{\text{argmin}} \quad \|\mathbf{y} - \hat{\mathbf{y}}\|_2. \tag{7.18}$$

Since $\hat{\mathbf{y}} \in \text{span}(\mathbf{X})$, there exists some weight vector $\mathbf{w}$ such that

$$\hat{\mathbf{y}} = w_1\tilde{\mathbf{x}}_1 + \cdots + w_D\tilde{\mathbf{x}}_D = \mathbf{X}\mathbf{w} \tag{7.19}$$

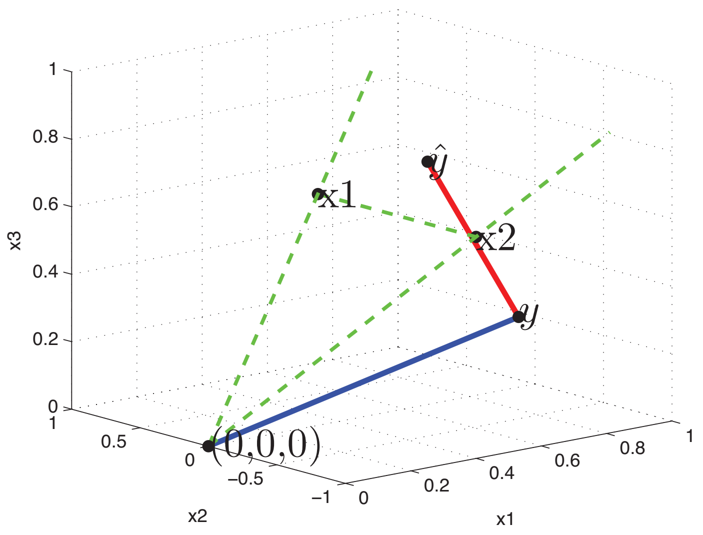

Figure 7.3 Graphical interpretation of least squares for $N = 3$ examples and $D = 2$ features. $\tilde{\mathbf{x}}_1$ and $\tilde{\mathbf{x}}_2$ are vectors in $\mathbb{R}^3$; together they define a 2D plane. $\mathbf{y}$ is also a vector in $\mathbb{R}^3$ but does not lie on this 2D plane. The orthogonal projection of $\mathbf{y}$ onto this plane is denoted $\hat{\mathbf{y}}$. The red line from $\mathbf{y}$ to $\hat{\mathbf{y}}$ is the residual, whose norm we want to minimize. For visual clarity, all vectors have been converted to unit norm. Figure generated by `leastSquaresProjection`.

To minimize the norm of the residual, $\mathbf{y} - \hat{\mathbf{y}}$, we want the residual vector to be orthogonal to every column of $\mathbf{X}$, so $\tilde{\mathbf{x}}_j^T(\mathbf{y} - \hat{\mathbf{y}}) = 0$ for $j = 1 : D$. Hence

$$\tilde{\mathbf{x}}_j^T(\mathbf{y} - \hat{\mathbf{y}}) = 0 \Rightarrow \mathbf{X}^T(\mathbf{y} - \mathbf{X}\mathbf{w}) = \mathbf{0} \Rightarrow \mathbf{w} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} \tag{7.20}$$

Hence our projected value of $\mathbf{y}$ is given by

$$\hat{\mathbf{y}} = \mathbf{X}\hat{\mathbf{w}} = \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} \tag{7.21}$$

This corresponds to an **orthogonal projection** of $\mathbf{y}$ onto the column space of $\mathbf{X}$. The projection matrix $\mathbf{P} \triangleq \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T$ is called the **hat matrix**, since it "puts the hat on $\mathbf{y}$".

7.3.3 Convexity

When discussing least squares, we noted that the NLL had a bowl shape with a unique minimum. The technical term for functions like this is **convex**. Convex functions play a very important role in machine learning.

Let us define this concept more precisely. We say a *set* $\mathcal{S}$ is **convex** if for any $\boldsymbol{\theta}, \boldsymbol{\theta}' \in \mathcal{S}$, we have

$$\lambda\boldsymbol{\theta} + (1 - \lambda)\boldsymbol{\theta}' \in \mathcal{S}, \quad \forall\, \lambda \in [0, 1] \tag{7.22}$$

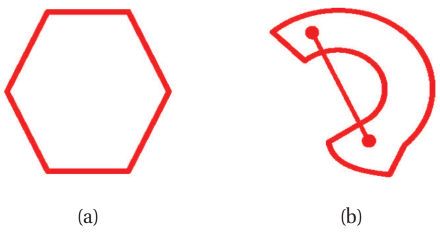

Figure 7.4 (a) Illustration of a convex set. (b) Illustration of a nonconvex set.

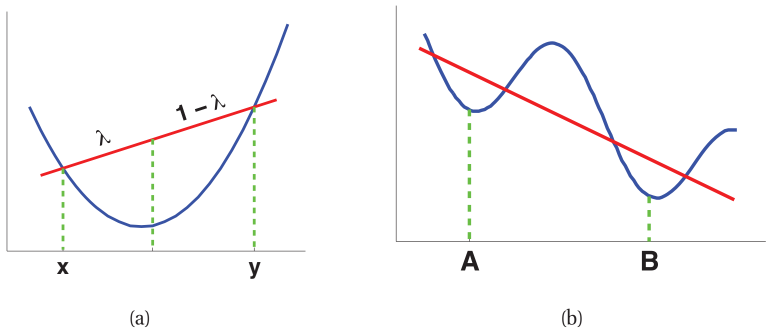

Figure 7.5 (a) Illustration of a convex function. We see that the chord joining $(x, f(x))$ to $(y, f(y))$ lies above the function. (b) A function that is neither convex nor concave. **A** is a local minimum, **B** is a global minimum. Figure generated by `convexFnHand`.

That is, if we draw a line from $\boldsymbol{\theta}$ to $\boldsymbol{\theta}'$, all points on the line lie inside the set. See Figure 7.4(a) for an illustration of a convex set, and Figure 7.4(b) for an illustration of a non-convex set.

A *function* $f(\boldsymbol{\theta})$ is called convex if its **epigraph** (the set of points above the function) defines a convex set. Equivalently, a function $f(\boldsymbol{\theta})$ is called convex if it is defined on a convex set and if, for any $\boldsymbol{\theta}, \boldsymbol{\theta}' \in \mathcal{S}$, and for any $0 \leq \lambda \leq 1$, we have

$$f(\lambda\boldsymbol{\theta} + (1-\lambda)\boldsymbol{\theta}') \leq \lambda f(\boldsymbol{\theta}) + (1-\lambda) f(\boldsymbol{\theta}') \tag{7.23}$$

See Figure 7.5 for a 1d example. A function is called **strictly convex** if the inequality is strict. A function $f(\boldsymbol{\theta})$ is **concave** if $-f(\boldsymbol{\theta})$ is convex. Examples of scalar convex functions include θ^2, e^θ, and $\theta \log \theta$ (for $\theta > 0$). Examples of scalar concave functions include $\log(\theta)$ and $\sqrt{\theta}$.

Intuitively, a (strictly) convex function has a "bowl shape", and hence has a unique global minimum θ^* corresponding to the bottom of the bowl. Hence its second derivative must be positive everywhere, $\frac{d}{d\theta} f(\theta) > 0$. A twice-continuously differentiable, multivariate function f is convex iff its Hessian is positive definite for all $\boldsymbol{\theta}$.[1] In the machine learning context, the function f often corresponds to the NLL.

1. Recall that the Hessian is the matrix of second partial derivatives, defined by $H_{jk} = \frac{\partial f^2(\theta)}{\partial \theta_j \partial \theta_k}$. Also, recall that a matrix $\mathbf{H}$ is **positive definite** iff $\mathbf{v}^T \mathbf{H} \mathbf{v} > 0$ for any non-zero vector $\mathbf{v}$.

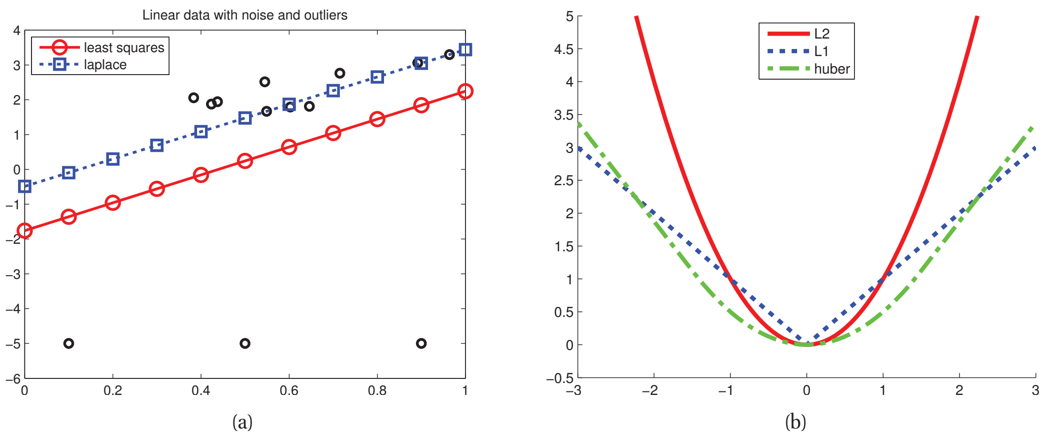

Figure 7.6 (a) Illustration of robust linear regression. Figure generated by `linregRobustDemoCombined`. (b) Illustration of ℓ_2, ℓ_1, and Huber loss functions. Figure generated by `huberLossDemo`.

Models where the NLL is convex are desirable, since this means we can always find the globally optimal MLE. We will see many examples of this later in the book. However, many models of interest will not have concave likelihoods. In such cases, we will discuss ways to derive locally optimal parameter estimates.

7.4 Robust linear regression *

It is very common to model the noise in regression models using a Gaussian distribution with zero mean and constant variance, $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$, where $\epsilon_i = y_i - \mathbf{w}^T\mathbf{x}_i$. In this case, maximizing likelihood is equivalent to minimizing the sum of squared residuals, as we have seen. However, if we have **outliers** in our data, this can result in a poor fit, as illustrated in Figure 7.6(a). (The outliers are the points on the bottom of the figure.) This is because squared error penalizes deviations quadratically, so points far from the line have more affect on the fit than points near to the line.

One way to achieve **robustness** to outliers is to replace the Gaussian distribution for the response variable with a distribution that has **heavy tails**. Such a distribution will assign higher likelihood to outliers, without having to perturb the straight line to "explain" them.

One possibility is to use the Laplace distribution, introduced in Section 2.4.3. If we use this as our observation model for regression, we get the following likelihood:

$$p(y|\mathbf{x}, \mathbf{w}, b) = \text{Lap}(y|\mathbf{w}^T\mathbf{x}, b) \propto \exp(-\frac{1}{b}|y - \mathbf{w}^T\mathbf{x}|) \tag{7.24}$$

The robustness arises from the use of $|y - \mathbf{w}^T\mathbf{x}|$ instead of $(y - \mathbf{w}^T\mathbf{x})^2$. For simplicity, we will assume b is fixed. Let $r_i \triangleq y_i - \mathbf{w}^T\mathbf{x}_i$ be the i'th residual. The NLL has the form

$$\ell(\mathbf{w}) = \sum_i |r_i(\mathbf{w})| \tag{7.25}$$

Likelihood	Prior	Name	Section
Gaussian	Uniform	Least squares	7.3
Gaussian	Gaussian	Ridge	7.5
Gaussian	Laplace	Lasso	13.3
Laplace	Uniform	Robust regression	7.4
Student	Uniform	Robust regression	Exercise 11.12

Table 7.1 Summary of various likelihoods and priors used for linear regression. The likelihood refers to the distributional form of $p(y|\mathbf{x}, \mathbf{w}, \sigma^2)$, and the prior refers to the distributional form of $p(\mathbf{w})$. MAP estimation with a uniform distribution corresponds to MLE.

Unfortunately, this is a non-linear objective function, which is hard to optimize. Fortunately, we can convert the NLL to a linear objective, subject to linear constraints, using the following **split variable** trick. First we define

$$r_i \triangleq r_i^+ - r_i^- \tag{7.26}$$

and then we impose the linear inequality constraints that $r_i^+ \geq 0$ and $r_i^- \geq 0$. Now the constrained objective becomes

$$\min_{\mathbf{w},\mathbf{r}^+,\mathbf{r}^-} \sum_i (r_i^+ - r_i^-) \qquad \text{s.t.} \quad r_i^+ \geq 0, r_i^- \geq 0, \mathbf{w}^T\mathbf{x}_i + r_i^+ + r_i^- = y_i \tag{7.27}$$

This is an example of a **linear program** with $D + 2N$ unknowns and $3N$ constraints.

Since this is a convex optimization problem, it has a unique solution. To solve an LP, we must first write it in standard form, which as follows:

$$\min_{\boldsymbol{\theta}} \mathbf{f}^T\boldsymbol{\theta} \;\; \text{s.t.} \;\; \mathbf{A}\boldsymbol{\theta} \leq \mathbf{b}, \; \mathbf{A}_{eq}\boldsymbol{\theta} = \mathbf{b}_{eq}, \; \mathbf{l} \leq \boldsymbol{\theta} \leq \mathbf{u} \tag{7.28}$$

In our current example, $\boldsymbol{\theta} = (\mathbf{w}, \mathbf{r}^+, \mathbf{r}^-)$, $\mathbf{f} = [\mathbf{0}, \mathbf{1}, \mathbf{1}]$, $\mathbf{A} = []$, $\mathbf{b} = []$, $\mathbf{A}_{eq} = [\mathbf{X}, \mathbf{I}, -\mathbf{I}]$, $\mathbf{b}_{eq} = \mathbf{y}$, $\mathbf{l} = [-\infty\mathbf{1}, \mathbf{0}, \mathbf{0}]$, $\mathbf{u} = []$. This can be solved by any LP solver (see e.g., (Boyd and Vandenberghe 2004)). See Figure 7.6(a) for an example of the method in action.

An alternative to using NLL under a Laplace likelihood is to minimize the **Huber loss** function (Huber 1964), defined as follows:

$$L_H(r, \delta) = \begin{cases} r^2/2 & \text{if } |r| \leq \delta \\ \delta|r| - \delta^2/2 & \text{if } |r| > \delta \end{cases} \tag{7.29}$$

This is equivalent to ℓ_2 for errors that are smaller than δ, and is equivalent to ℓ_1 for larger errors. See Figure 7.6(b). The advantage of this loss function is that it is everywhere differentiable, using the fact that $\frac{d}{dr}|r| = \text{sign}(r)$ if $r \neq 0$. We can also check that the function is C_1 continuous, since the gradients of the two parts of the function match at $r = \pm\delta$, namely $\frac{d}{dr}L_H(r, \delta)|_{r=\delta} = \delta$. Consequently optimizing the Huber loss is much faster than using the Laplace likelihood, since we can use standard smooth optimization methods (such as quasi-Newton) instead of linear programming.

Figure 7.6(a) gives an illustration of the Huber loss function. The results are qualitatively similiar to the probabilistic methods. (In fact, it turns out that the Huber method also has a probabilistic interpretation, although it is rather unnatural (Pontil et al. 1998).)

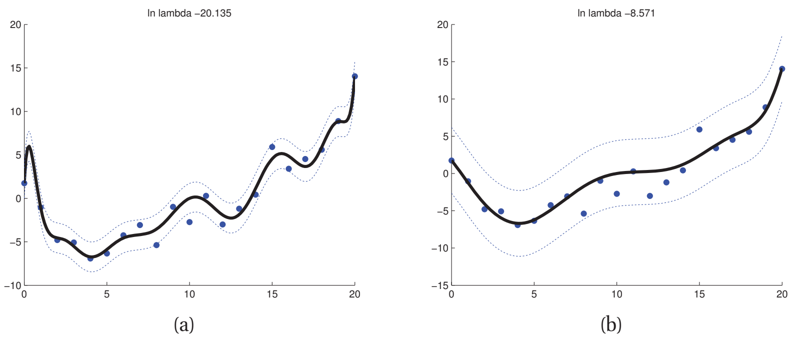

Figure 7.7 Degree 14 Polynomial fit to $N = 21$ data points with increasing amounts of ℓ_2 regularization. Data was generated from noise with variance $\sigma^2 = 4$. The error bars, representing the noise variance σ^2, get wider as the fit gets smoother, since we are ascribing more of the data variation to the noise. Figure generated by `linregPolyVsRegDemo`.

7.5 Ridge regression

One problem with ML estimation is that it can result in overfitting. In this section, we discuss a way to ameliorate this problem by using MAP estimation with a Gaussian prior. For simplicity, we assume a Gaussian likelihood, rather than a robust likelihood.

7.5.1 Basic idea

The reason that the MLE can overfit is that it is picking the parameter values that are the best for modeling the training data; but if the data is noisy, such parameters often result in complex functions. As a simple example, suppose we fit a degree 14 polynomial to $N = 21$ data points using least squares. The resulting curve is very "wiggly", as shown in Figure 7.7(a). The corresponding least squares coefficients (excluding w_0) are as follows:

```
6.560, -36.934, -109.255, 543.452, 1022.561, -3046.224, -3768.013,
8524.540, 6607.897, -12640.058, -5530.188, 9479.730, 1774.639, -2821.526
```

We see that there are many large positive and negative numbers. These balance out exactly to make the curve "wiggle" in just the right way so that it almost perfectly interpolates the data. But this situation is unstable: if we changed the data a little, the coefficients would change a lot.

We can encourage the parameters to be small, thus resulting in a smoother curve, by using a zero-mean Gaussian prior:

$$p(\mathbf{w}) = \prod_j \mathcal{N}(w_j|0, \tau^2) \tag{7.30}$$

where $1/\tau^2$ controls the strength of the prior. The corresponding MAP estimation problem becomes

$$\underset{\mathbf{w}}{\operatorname{argmax}} \sum_{i=1}^{N} \log \mathcal{N}(y_i|w_0 + \mathbf{w}^T\mathbf{x}_i, \sigma^2) + \sum_{j=1}^{D} \log \mathcal{N}(w_j|0, \tau^2) \tag{7.31}$$

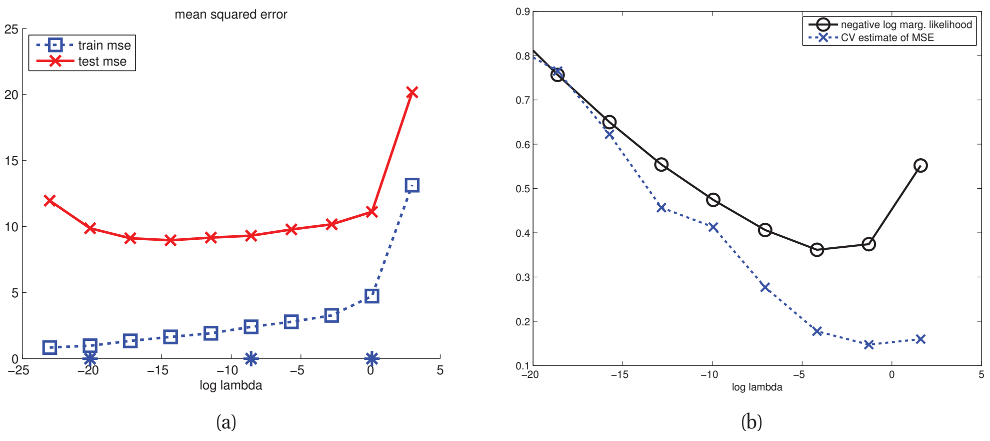

Figure 7.8 (a) Training error (dotted blue) and test error (solid red) for a degree 14 polynomial fit by ridge regression, plotted vs $\log(\lambda)$. Data was generated from noise with variance $\sigma^2 = 4$ (training set has size $N = 21$). Note: Models are ordered from complex (small regularizer) on the left to simple (large regularizer) on the right. The stars correspond to the values used to plot the functions in Figure 7.7. (b) Estimate of performance using training set. Dotted blue: 5-fold cross-validation estimate of future MSE. Solid black: negative log marginal likelihood, $-\log p(\mathcal{D}|\lambda)$. Both curves have been vertically rescaled to [0,1] to make them comparable. Figure generated by `linregPolyVsRegDemo`.

It is a simple exercise to show that this is equivalent to minimizing the following:

$$J(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N}(y_i - (w_0 + \mathbf{w}^T\mathbf{x}_i))^2 + \lambda||\mathbf{w}||_2^2 \tag{7.32}$$

where $\lambda \triangleq \sigma^2/\tau^2$ and $||\mathbf{w}||_2^2 = \sum_j w_j^2 = \mathbf{w}^T\mathbf{w}$ is the squared two-norm. Here the first term is the MSE/ NLL as usual, and the second term, $\lambda \geq 0$, is a complexity penalty. The corresponding solution is given by

$$\boxed{\hat{\mathbf{w}}_{ridge} = (\lambda\mathbf{I}_D + \mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}} \tag{7.33}$$

This technique is known as **ridge regression**, or **penalized least squares**. In general, adding a Gaussian prior to the parameters of a model to encourage them to be small is called ℓ_2 **regularization** or **weight decay**. Note that the offset term w_0 is not regularized, since this just affects the height of the function, not its complexity. By penalizing the sum of the magnitudes of the weights, we ensure the function is simple (since $\mathbf{w} = \mathbf{0}$ corresponds to a straight line, which is the simplest possible function, corresponding to a constant.)

We illustrate this idea in Figure 7.7, where we see that increasing λ results in smoother functions. The resulting coefficients also become smaller. For example, using $\lambda = 10^{-3}$, we have

```
2.128, 0.807, 16.457, 3.704, -24.948, -10.472, -2.625, 4.360, 13.711,
10.063, 8.716, 3.966, -9.349, -9.232
```

In Figure 7.8(a), we plot the MSE on the training and test sets vs $\log(\lambda)$. We see that, as we increase λ (so the model becomes more constrained), the error on the training set increases. For the test set, we see the characteristic U-shaped curve, where the model overfits and then underfits. It is common to use cross validation to pick λ, as shown in Figure 7.8(b). In Section 1.4.8, we will discuss a more probabilistic approach.

We will consider a variety of different priors in this book. Each of these corresponds to a different form of **regularization**. This technique is very widely used to prevent overfitting.

7.5.2 Numerically stable computation *

Interestingly, ridge regression, which works better statistically, is also easier to fit numerically, since $(\lambda \mathbf{I}_D + \mathbf{X}^T\mathbf{X})$ is much better conditioned (and hence more likely to be invertible) than $\mathbf{X}^T\mathbf{X}$, at least for suitable largy λ.

Nevertheless, inverting matrices is still best avoided, for reasons of numerical stability. (Indeed, if you write `w=inv(X' * X)*X'*y` in Matlab, it will give you a warning.) We now describe a useful trick for fitting ridge regression models (and hence by extension, computing vanilla OLS estimates) that is more numerically robust. We assume the prior has the form $p(\mathbf{w}) = \mathcal{N}(\mathbf{0}, \mathbf{\Lambda}^{-1})$, where $\mathbf{\Lambda}$ is the precision matrix. In the case of ridge regression, $\mathbf{\Lambda} = (1/\tau^2)\mathbf{I}$. To avoid penalizing the w_0 term, we should center the data first, as explained in Exercise 7.5.

First let us augment the original data with some "virtual data" coming from the prior:

$$\tilde{\mathbf{X}} = \begin{pmatrix} \mathbf{X}/\sigma \\ \sqrt{\mathbf{\Lambda}} \end{pmatrix}, \quad \tilde{\mathbf{y}} = \begin{pmatrix} \mathbf{y}/\sigma \\ \mathbf{0}_{D\times 1} \end{pmatrix} \tag{7.34}$$

where $\mathbf{\Lambda} = \sqrt{\mathbf{\Lambda}}\sqrt{\mathbf{\Lambda}}^T$ is a **Cholesky decomposition** of $\mathbf{\Lambda}$. We see that $\tilde{\mathbf{X}}$ is $(N+D)\times D$, where the extra rows represent pseudo-data from the prior.

We now show that the NLL on this expanded data is equivalent to *penalized* NLL on the original data:

$$\begin{aligned} f(\mathbf{w}) &= (\tilde{\mathbf{y}} - \tilde{\mathbf{X}}\mathbf{w})^T(\tilde{\mathbf{y}} - \tilde{\mathbf{X}}\mathbf{w}) && (7.35) \\ &= \left(\begin{pmatrix} \mathbf{y}/\sigma \\ \mathbf{0} \end{pmatrix} - \begin{pmatrix} \mathbf{X}/\sigma \\ \sqrt{\mathbf{\Lambda}} \end{pmatrix}\mathbf{w}\right)^T \left(\begin{pmatrix} \mathbf{y}/\sigma \\ \mathbf{0} \end{pmatrix} - \begin{pmatrix} \mathbf{X}/\sigma \\ \sqrt{\mathbf{\Lambda}} \end{pmatrix}\mathbf{w}\right) && (7.36) \\ &= \begin{pmatrix} \frac{1}{\sigma}(\mathbf{y} - \mathbf{X}\mathbf{w}) \\ -\sqrt{\mathbf{\Lambda}}\mathbf{w} \end{pmatrix}^T \begin{pmatrix} \frac{1}{\sigma}(\mathbf{y} - \mathbf{X}\mathbf{w}) \\ -\sqrt{\mathbf{\Lambda}}\mathbf{w} \end{pmatrix} && (7.37) \\ &= \frac{1}{\sigma^2}(\mathbf{y} - \mathbf{X}\mathbf{w})^T(\mathbf{y} - \mathbf{X}\mathbf{w}) + (\sqrt{\mathbf{\Lambda}}\mathbf{w})^T(\sqrt{\mathbf{\Lambda}}\mathbf{w}) && (7.38) \\ &= \frac{1}{\sigma^2}(\mathbf{y} - \mathbf{X}\mathbf{w})^T(\mathbf{y} - \mathbf{X}\mathbf{w}) + \mathbf{w}^T\mathbf{\Lambda}\mathbf{w} && (7.39) \end{aligned}$$

Hence the MAP estimate is given by

$$\hat{\mathbf{w}}_{ridge} = (\tilde{\mathbf{X}}^T\tilde{\mathbf{X}})^{-1}\tilde{\mathbf{X}}^T\tilde{\mathbf{y}} \tag{7.40}$$

as we claimed.

Now let

$$\tilde{\mathbf{X}} = \mathbf{Q}\mathbf{R} \tag{7.41}$$

be the **QR decomposition** of $\mathbf{X}$, where $\mathbf{Q}$ is orthonormal (meaning $\mathbf{Q}^T\mathbf{Q} = \mathbf{Q}\mathbf{Q}^T = \mathbf{I}$), and $\mathbf{R}$ is upper triangular. Then

$$(\tilde{\mathbf{X}}^T\tilde{\mathbf{X}})^{-1} = (\mathbf{R}^T\mathbf{Q}^T\mathbf{Q}\mathbf{R})^{-1} = (\mathbf{R}^T\mathbf{R})^{-1} = \mathbf{R}^{-1}\mathbf{R}^{-T} \tag{7.42}$$

Hence

$$\hat{\mathbf{w}}_{ridge} = \mathbf{R}^{-1}\mathbf{R}^{-T}\mathbf{R}^T\mathbf{Q}^T\tilde{\mathbf{y}} = \mathbf{R}^{-1}\mathbf{Q}\tilde{\mathbf{y}} \tag{7.43}$$

Note that $\mathbf{R}$ is easy to invert since it is upper triangular. This gives us a way to compute the ridge estimate while avoiding having to invert $(\mathbf{\Lambda} + \mathbf{X}^T\mathbf{X})$.

We can use this technique to find the MLE, by simply computing the QR decomposition of the unaugmented matrix $\mathbf{X}$, and using the original $\mathbf{y}$. This is the method of choice for solving least squares problems. (In fact, it is so sommon that it can be implemented in one line of Matlab, using the **backslash operator**: `w=X\y`.) Note that computing the QR decomposition of an $N \times D$ matrix takes $O(ND^2)$ time, and is numerically very stable.

If $D \gg N$, we should first perform an SVD decomposition. In particular, let $\mathbf{X} = \mathbf{U}\mathbf{S}\mathbf{V}^T$ be the SVD of $\mathbf{X}$, where $\mathbf{V}^T\mathbf{V} = \mathbf{I}_N$, $\mathbf{U}\mathbf{U}^T = \mathbf{U}^T\mathbf{U} = \mathbf{I}_N$, and $\mathbf{S}$ is a diagonal $N \times N$ matrix. Now let $\mathbf{Z} = \mathbf{U}\mathbf{D}$ be an $N \times N$ matrix. Then we can rewrite the ridge estimate thus:

$$\hat{\mathbf{w}}_{ridge} \quad = \quad \mathbf{V}(\mathbf{Z}^T\mathbf{Z} + \lambda\mathbf{I}_N)^{-1}\mathbf{Z}^T\mathbf{y} \tag{7.44}$$

In other words, we can replace the D-dimensional vectors $\mathbf{x}_i$ with the N-dimensional vectors $\mathbf{z}_i$ and perform our penalized fit as before. We then transform the N-dimensional solution to the D-dimensional solution by multiplying by $\mathbf{V}$. Geometrically, we are rotating to a new coordinate system in which all but the first N coordinates are zero. This does not affect the solution since the spherical Gaussian prior is rotationally invariant. The overall time is now $O(DN^2)$ operations.

7.5.3 Connection with PCA *

In this section, we discuss an interesting connection between ridge regression and PCA (Section 12.2), which gives further insight into why ridge regression works well. Our discussion is based on (Hastie et al. 2009, p66).

Let $\mathbf{X} = \mathbf{U}\mathbf{S}\mathbf{V}^T$ be the SVD of $\mathbf{X}$. From Equation 7.44, we have

$$\hat{\mathbf{w}}_{ridge} = \mathbf{V}(\mathbf{S}^2 + \lambda\mathbf{I})^{-1}\mathbf{S}\mathbf{U}^T\mathbf{y} \tag{7.45}$$

Hence the ridge predictions on the training set are given by

$$\begin{aligned}\hat{\mathbf{y}} &= \mathbf{X}\hat{\mathbf{w}}_{ridge} = \mathbf{U}\mathbf{S}\mathbf{V}^T\mathbf{V}(\mathbf{S}^2 + \lambda\mathbf{I})^{-1}\mathbf{S}\mathbf{U}^T\mathbf{y} & (7.46)\\ &= \mathbf{U}\tilde{\mathbf{S}}\mathbf{U}^T\mathbf{y} = \sum_{j=1}^{D}\mathbf{u}_j\tilde{S}_{jj}\mathbf{u}_j^T\mathbf{y} & (7.47)\end{aligned}$$