12

Regression And Classification

Overview

In this chapter we delve deeper into the problems of regression and classification, where the goal is to estimate a certain quantity of interest (the response) from observed features, as illustrated by the examples in Figure 12.1. In regression, the response is modeled as a numerical variable. In classification, the response belongs to a finite set of predetermined classes. In previous chapters, we derive optimal estimators for both problems: Theorem 7.59 establishes that the conditional mean is optimal for regression, and Theorems 4.30 and 6.11 show that the maximum-a-posteriori estimator is optimal for classification. However, it is usually intractable to compute these estimators from data in practical scenarios due to the curse of dimensionality (see Sections 4.7 and 4.8, and the end of Section 7.8.3).

The chapter presents the main approaches used to perform regression and classification in practice. Section 12.1 provides a comprehensive description of linear regression models. Section 12.2 discusses generalization to held-out data, explaining when linear models can be expected to generalize or to overfit. Section 12.3 describes ridge regression and introduces the concept of regularization. Section 12.3 considers the problem of sparse regression, where the goal is to fit a linear model that uses a small subset of the available features. In Sections 12.5 and 12.6, we study linear models for binary and multiclass classification. Sections 12.7 and 12.8 describe nonlinear models based on trees and neural networks, respectively, which are widely used in machine learning. Finally, Section 12.9 discusses how to evaluate classification performance.

12.1 Linear Regression

Linear regression models are ubiquitous in data science and statistics, because they are simple, interpretable, and often surprisingly effective in practice. In Section 12.1.1 we derive an optimal linear model, under the assumption that the joint statistics between the features and the response are known. In Section 12.1.2 we consider a more realistic scenario, where the linear model is computed from the available data. Section 12.1.3 discusses how to evaluate linear regression models. Finally, Section 12.1.4 explains how to use linear regression to perform causal inference.

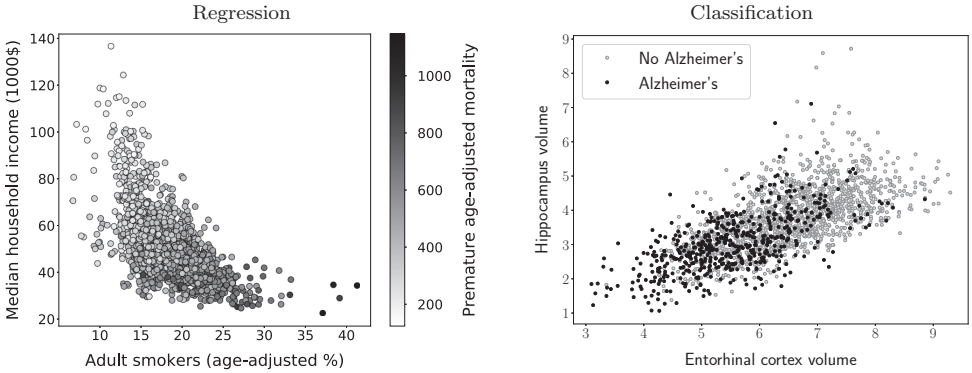


Figure 12.1 Regression and classification problems. The left scatterplot shows the median household income and the fraction of adult smokers (adjusted by age) for each county in the United States in 2019. The response associated to these features is the premature mortality in each county (also adjusted by age), represented by the color of each marker. Since the response is a number, estimating it from the features is a regression problem. The right scatterplot shows the normalized volume of the hippocampus and entorhinal cortex of a group of people. The data are divided into two classes, indicating whether each person has Alzheimer's disease (black) or not (light gray). Estimating the class from the features is a classification problem.

12.1.1 Linear Minimum Mean-Squared-Error Estimation

A popular way to avoid the curse of dimensionality in regression is to assume that the dependence between the response and the features is linear. In this section, we derive the optimal linear estimator when the joint statistics of the response and the features are known. Let us represent the response by a random variable $\tilde{y}$ and the features as entries of a random vector $\tilde{x}$. A linear estimator approximates the response as a linear combination of the features,

$$\tilde{y} \approx \ell(\tilde{x}) := \sum_{j=1}^{d} \beta[j]\tilde{x}[j] + \alpha \tag{12.1}$$

$$= \beta^T \tilde{x} + \alpha, \tag{12.2}$$

where d is the number of features, and the vector of linear coefficients $\beta \in \mathbb{R}^d$ determines the contribution of each feature to the estimate. The additive constant or intercept $\alpha \in \mathbb{R}$ accounts for the mean of the response and the features (see Theorem 12.2). Strictly speaking, this makes the estimator affine instead of linear, but such models are usually called *linear* nevertheless.

In order to fit a linear estimator, we need to find the linear coefficients β and the intercept α that best approximate the response. A popular fitting criterion is to minimize the mean squared error (MSE) (see Definition 7.29) between the

response and the estimator. The corresponding estimator is known as the linear minimum MSE (MMSE) estimator.

To gain some geometric intuition about the linear MMSE estimator, we consider the case where the features and response are centered to have zero mean. In that case, the intercept is zero, so the linear MMSE does not include an additive constant (see (12.15) in Theorem 12.2). As explained in Section 8.7, we can interpret the zero-mean random variables representing the response and the features as vectors in a vector space, where the covariance is a valid inner product and the norm of a random variable is its variance. Within that vector space, the set of linear combinations of the features,

$$\beta^T \tilde{x} = \sum_{j=1}^{d} \beta[j] \tilde{x}[j] \tag{12.3}$$

form a d-dimensional subspace or hyperplane of the vector space spanned by the vectors $\tilde{x}[1]$, ..., $\tilde{x}[d]$ (see Figure 12.2). The squared distance between any point in the subspace and the response is equal to the MSE of the corresponding linear estimator,

$$\left|\left|\tilde{y} - \beta^T \tilde{x}\right|\right|^2 = \text{Var}\left[\tilde{y} - \beta^T \tilde{x}\right] = \text{E}\left[\left(\tilde{y} - \beta^T \tilde{x}\right)^2\right]. \tag{12.4}$$

The linear MMSE estimator $\ell_{\text{MMSE}}(\tilde{x})$ of $\tilde{y}$ given $\tilde{x}$ is the vector in the subspace of linear estimators that is *closest* to the response according to this distance. The closest vector is the *orthogonal projection* of $\tilde{y}$ onto the subspace, selected so that the residual $\tilde{y} - \ell_{\text{MMSE}}(\tilde{x})$ is orthogonal to the subspace, as depicted in Figure 12.2. This yields the closest vector on the subspace, because $\ell_{\text{MMSE}}(\tilde{x}) - \beta^T \tilde{x}$ belongs to the subspace (it is a linear combination of vectors in the subspace), and is consequently orthogonal to the residual. Consequently, by the Pythagorean theorem, for any $\beta \in \mathbb{R}^d$,

$$\begin{aligned} \left|\left|\tilde{y} - \beta^T \tilde{x}\right|\right|^2 &= \left|\left|\tilde{y} - \ell_{\text{MMSE}}(\tilde{x})\right|\right|^2 + \left|\left|\ell_{\text{MMSE}}(\tilde{x}) - \beta^T \tilde{x}\right|\right|^2 && (12.5) \\ &\geq \left|\left|\tilde{y} - \ell_{\text{MMSE}}(\tilde{x})\right|\right|^2. && (12.6) \end{aligned}$$

Enforcing orthogonality between the subspace and the residual yields the following equation, which we can solve to derive the linear MMSE estimator:

$$\begin{aligned} 0 &= \left\langle \beta^T \tilde{x}, \tilde{y} - \ell_{\text{MMSE}}(\tilde{x}) \right\rangle && (12.7) \\ &= \text{E}\left[\beta^T \tilde{x} \left(\tilde{y} - \tilde{x}^T \beta_{\text{MMSE}}\right)\right] && (12.8) \\ &= \beta^T \left(\text{E}\left[\tilde{x}\tilde{y}\right] - \text{E}\left[\tilde{x}\tilde{x}^T\right] \beta_{\text{MMSE}}\right). && (12.9) \end{aligned}$$

Crucially, we do not need to know the whole joint distribution of the features and the response to solve this equation. The equation only involves the covariance matrix of the features $\Sigma_{\tilde{x}} := \text{E}\left[\tilde{x}\tilde{x}^T\right]$ and the *cross-covariance* between the features and the response.

Definition 12.1 (Cross-covariance)**.** *The cross-covariance vector $\Sigma_{\tilde{x}\tilde{y}}$ between a*

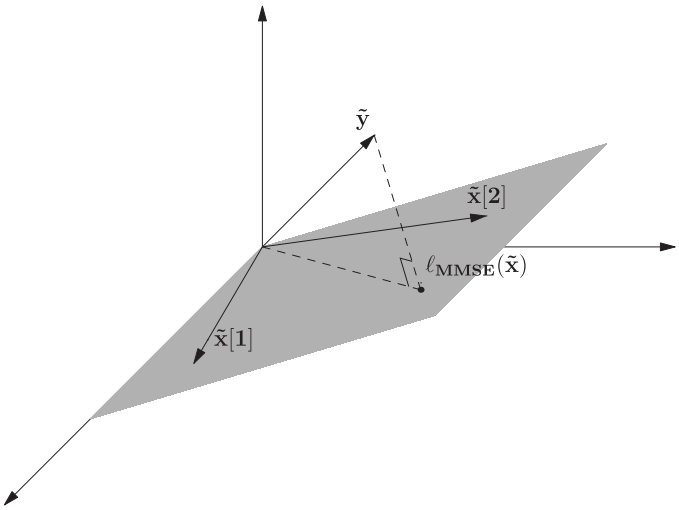


Figure 12.2 The linear MMSE estimator as an orthogonal projection. Following the geometric interpretation of zero-mean random variables as vectors from Section 8.7, we represent the response $\tilde{y}$ and the two features $\tilde{x}[1]$ and $\tilde{x}[2]$ in a regression problem as three-dimensional vectors. Any linear combination of the features must lie in the plane spanned by the corresponding vectors. In this vector space, the squared distance between vectors is the MSE between the corresponding random variables, so the linear MMSE estimator $\ell_{\text{MMSE}}(\tilde{x})$ is the vector in the plane that is closest to $\tilde{y}$, i.e. the orthogonal projection of $\tilde{y}$ onto the plane.

d-dimensional random vector $\tilde{x}$ and a random variable $\tilde{y}$ is a vector with entries equal to the covariances between $\tilde{y}$ and the corresponding entry of $\tilde{x}$:

$$\Sigma_{\tilde{x}\tilde{y}} := \mathrm{E}\left[\mathrm{ct}\left(\tilde{x}\right)\mathrm{ct}\left(\tilde{y}\right)\right] = \begin{bmatrix} \mathrm{Cov}\left[\tilde{x}[1], \tilde{y}\right] \\ \mathrm{Cov}\left[\tilde{x}[2], \tilde{y}\right] \\ \cdots \\ \mathrm{Cov}\left[\tilde{x}[d], \tilde{y}\right] \end{bmatrix}, \tag{12.10}$$

where $\mathrm{ct}\left(\tilde{x}\right) := \tilde{x} - \mathrm{E}\left[\tilde{x}\right]$ and $\mathrm{ct}\left(\tilde{y}\right) := \tilde{y} - \mathrm{E}\left[\tilde{y}\right]$ are obtained by centering $\tilde{x}$ and $\tilde{y}$ using their respective means.

We conclude that the optimal linear estimator satisfies

$$\beta^T\left(\Sigma_{\tilde{x}\tilde{y}} - \Sigma_{\tilde{x}}\beta_{\text{MMSE}}\right) = 0, \tag{12.11}$$

for any $\beta \in \mathbb{R}^d$, by (12.9). As a result, $\Sigma_{\tilde{x}\tilde{y}} = \Sigma_{\tilde{x}}\beta_{\text{MMSE}}$, or equivalently $\beta_{\text{MMSE}} = \Sigma_{\tilde{x}}^{-1}\Sigma_{\tilde{x}\tilde{y}}$, assuming the covariance matrix of the features is invertible. The following theorem establishes that these linear coefficients are indeed optimal, and generalizes the result to features and responses with nonzero means. As promised, when $\tilde{x}$ and $\tilde{y}$ have zero mean, the intercept α_{MMSE} is zero.

Theorem 12.2 (Linear MMSE estimator)**.** *Let $\tilde{x}$ and $\tilde{y}$ be a d-dimensional random vector and a random variable, representing the features and response in a regression problem. We denote the means of $\tilde{x}$ and $\tilde{y}$ by $\mu_{\tilde{x}}$ and $\mu_{\tilde{y}}$ respectively. If the covariance matrix $\Sigma_{\tilde{x}}$ of $\tilde{x}$ is invertible, the linear MMSE estimator of $\tilde{y}$ given $\tilde{x}$ is*

$$\ell_{\text{MMSE}}(\tilde{x}) := \beta_{\text{MMSE}}^T \tilde{x} + \alpha_{\text{MMSE}} \tag{12.12}$$

$$= \Sigma_{\tilde{x}\tilde{y}}^T \Sigma_{\tilde{x}}^{-1} (\tilde{x} - \mu_{\tilde{x}}) + \mu_{\tilde{y}}, \tag{12.13}$$

where $\Sigma_{\tilde{x}\tilde{y}}$ is the cross-covariance between $\tilde{x}$ and $\tilde{y}$. The model parameters

$$\beta_{\text{MMSE}} := \Sigma_{\tilde{x}}^{-1} \Sigma_{\tilde{x}\tilde{y}}, \tag{12.14}$$

$$\alpha_{\text{MMSE}} := \mu_{\tilde{y}} - \mu_{\tilde{x}}^T \beta_{\text{MMSE}} \tag{12.15}$$

are optimal, in the sense that they minimize the MSE cost function or loss

$$(\beta_{\text{MMSE}}, \alpha_{\text{MMSE}}) = \arg \min_{\alpha \in \mathbb{R}, \beta \in \mathbb{R}^d} \mathrm{E}\left[\left(\tilde{y} - \beta^T \tilde{x} - \alpha \right)^2 \right]. \tag{12.16}$$

Proof To prove the result we generalize the argument in the proof of Theorem 8.14 to multiple features. We denote the MSE of an affine estimator, as a function of the parameters β and α, by

$$\text{MSE}(\alpha, \beta) := \mathrm{E}\left[(\tilde{y} - \beta^T \tilde{x} - \alpha)^2 \right]. \tag{12.17}$$

Let us denote by $\alpha^*(\beta)$ the value of α that minimizes the MSE for a fixed $\beta \in \mathbb{R}^p$. Equivalently, $\alpha^*(\beta)$ is the best constant estimate of the random variable $\tilde{y} - \beta^T \tilde{x}$. By Theorem 7.30 and linearity of expectation,

$$\alpha^*(\beta) = \arg \min_{\alpha \in \mathbb{R}} \text{MSE}(\alpha, \beta) \tag{12.18}$$

$$= \mathrm{E}\left[\tilde{y} - \beta^T \tilde{x} \right] \tag{12.19}$$

$$= \mu_{\tilde{y}} - \beta^T \mu_{\tilde{x}}. \tag{12.20}$$

For any $\beta \in \mathbb{R}^d$ and any α, $\text{MSE}(\alpha, \beta) \geq \text{MSE}(\alpha, \beta^*(\beta))$. To obtain the optimal coefficient vector β_{MMSE} we can therefore set α equal to $\alpha^*(\beta)$ and minimize the resulting MSE,

$$\beta_{\text{MMSE}} = \arg \min_{\beta \in \mathbb{R}^d} \text{MSE}(\alpha, \beta^*(\beta)). \tag{12.21}$$

To alleviate notation, we denote the centered response and features by $\text{ct}(\tilde{y}) := \tilde{y} - \mu_{\tilde{y}}$ and $\text{ct}(\tilde{x}) := \tilde{x} - \mu_{\tilde{x}}$, respectively. By linearity of expectation,

$$\text{MSE}(\alpha, \beta^*(\beta)) = \mathrm{E}\left[(\tilde{y} - \beta^T \tilde{x} - \alpha^*(\beta))^2 \right] \tag{12.22}$$

$$= \mathrm{E}\left[(\text{ct}(\tilde{y}) - \beta^T \text{ct}(\tilde{x}))^2 \right] \tag{12.23}$$

$$= \mathrm{E}\left[\text{ct}(\tilde{y})^2 \right] + \beta^T \mathrm{E}\left[\text{ct}(\tilde{x}) \, \text{ct}(\tilde{x})^T \right] \beta - 2\beta^T \mathrm{E}\left[\text{ct}(\tilde{x}) \, \text{ct}(\tilde{y}) \right]$$

$$= \sigma_{\tilde{y}}^2 + \beta^T \Sigma_{\tilde{x}} \beta - 2\beta^T \Sigma_{\tilde{x}\tilde{y}}. \tag{12.24}$$

As a function of β, the MSE is a quadratic form. Its gradient and Hessian with respect to β equal

$$\nabla_\beta \operatorname{MSE}(\alpha, \beta^*(\beta)) = 2\Sigma_{\tilde{x}}\beta - 2\Sigma_{\tilde{x}\tilde{y}}, \tag{12.25}$$

$$\nabla^2_\beta \operatorname{MSE}(\alpha, \beta^*(\beta)) = 2\Sigma_{\tilde{x}}. \tag{12.26}$$

Covariance matrices are positive semidefinite, because by Theorem 11.11 for any vector $a \in \mathbb{R}^d$

$$a^T \Sigma_{\tilde{x}} a = \operatorname{Var}\left[a^T \tilde{x}\right] \geq 0. \tag{12.27}$$

Since $\Sigma_{\tilde{x}}$ is invertible, there cannot be a nonzero vector such that $\Sigma_{\tilde{x}} a$ equals the zero vector, so the inequality is strict as long as a is not the zero vector. This means that the quadratic function is strictly convex and we can set its gradient to zero to find the value of β that achieves the unique minimum:

$$\beta_{\text{MMSE}} = \Sigma_{\tilde{x}}^{-1} \Sigma_{\tilde{x}\tilde{y}}. \tag{12.28}$$

The corresponding value of α is

$$\alpha_{\text{MMSE}} = \alpha^*(\beta_{\text{MMSE}}) \tag{12.29}$$

$$= \mu_{\tilde{y}} - \beta_{\text{MMSE}}^T \mu_{\tilde{x}}. \tag{12.30}$$

We conclude that the optimal linear estimator is

$$\ell_{\text{MMSE}}(\tilde{x}) := \beta_{\text{MMSE}}^T \tilde{x} + \alpha_{\text{MMSE}} \tag{12.31}$$

$$= \Sigma_{\tilde{x}\tilde{y}}^T \Sigma_{\tilde{x}}^{-1} \left(\tilde{x} - \mu_{\tilde{x}}\right) + \mu_{\tilde{y}}. \tag{12.32}$$

■

When there is a single feature ($d := 1$), $\Sigma_{\tilde{x}\tilde{y}}$ is the covariance between the feature and the response, and $\Sigma_{\tilde{x}}$ is the variance of the feature, so

$$\ell_{\text{MMSE}}(\tilde{x}) = \frac{\operatorname{Cov}[\tilde{x}, \tilde{y}]}{\operatorname{Var}[\tilde{x}]} \left(\tilde{x} - \mu_{\tilde{x}}\right) + \mu_{\tilde{y}} \tag{12.33}$$

$$= \sigma_{\tilde{y}} \, \rho_{\tilde{x},\tilde{y}} \left(\frac{\tilde{x} - \mu_{\tilde{x}}}{\sigma_{\tilde{x}}}\right) + \mu_{\tilde{y}}, \tag{12.34}$$

where $\sigma_{\tilde{x}}$ and $\sigma_{\tilde{y}}$ are the standard deviations of $\tilde{x}$ and $\tilde{y}$ respectively, and $\rho_{\tilde{x},\tilde{y}}$ is the correlation coefficient between $\tilde{x}$ and $\tilde{y}$. Reassuringly, we recover the expression for the simple-linear-regression MMSE estimator derived in Theorem 8.14 (set $\tilde{a} := \tilde{x}$ and $\tilde{b} := \tilde{y}$).

Example 12.3 (Uncorrelated features)**.** Consider a regression problem where the d features are all uncorrelated. For simplicity, we assume that the features and the response $\tilde{y}$ are centered so that their mean is zero. We denote the variance of each feature $\tilde{x}[i]$ by $\sigma^2_{\tilde{x}[i]}$ for $1 \leq i \leq d$, and the covariance between $\tilde{x}[i]$ and $\tilde{y}$ by $\sigma_{\tilde{x}[i],\tilde{y}}$. By Theorem 12.2, the linear MMSE estimator of the response $\tilde{y}$ given

the d-dimensional feature vector $\tilde{x}$ equals

$$\ell_{\text{MMSE}}(\tilde{x}) = \Sigma_{\tilde{x}\tilde{y}}^T \Sigma_{\tilde{x}}^{-1} \tilde{x} \tag{12.35}$$

$$= \begin{bmatrix} \sigma_{\tilde{x}[1],\tilde{y}} \\ \sigma_{\tilde{x}[2],\tilde{y}} \\ \cdots \\ \sigma_{\tilde{x}[d],\tilde{y}} \end{bmatrix}^T \begin{bmatrix} \sigma^2_{\tilde{x}[1]} & 0 & \cdots & 0 \\ 0 & \sigma^2_{\tilde{x}[2]} & \cdots & 0 \\ \cdots & \cdots & \ddots & \cdots \\ 0 & 0 & \cdots & \sigma^2_{\tilde{x}[d]} \end{bmatrix}^{-1} \tilde{x} \tag{12.36}$$

$$= \sum_{i=1}^{d} \frac{\sigma_{\tilde{x}[i],\tilde{y}}}{\sigma^2_{\tilde{x}[i]}} \tilde{x}[i] \tag{12.37}$$

$$= \sum_{i=1}^{d} \ell_{\text{MMSE}}(\tilde{x}[i]). \tag{12.38}$$

The estimator is simply the sum of the linear MMSE estimators of the response given each individual feature, which makes sense because the features are uncorrelated, and hence not linearly dependent. This has a geometric interpretation: the features are orthogonal in the vector space of zero-mean random variables endowed with the covariance inner product. Consequently, the projection of the response onto their span can be obtained by summing the projections onto each of the features.

Example 12.4 (Noise cancellation)**.** We are interested in recording the voice of a pilot in a helicopter. To this end we place a microphone inside her helmet and another microphone outside. We model the measurements as

$$\tilde{x}[1] = \tilde{y} + h\tilde{z} \tag{12.39}$$

$$\tilde{x}[2] = h\tilde{y} + \tilde{z}, \tag{12.40}$$

where $\tilde{y}$ and $\tilde{z}$ are random variables representing the voice of the pilot and the noise in the helicopter respectively. The constant $h \in [0, 1]$ models the effect of the helmet. We assume that $\tilde{y}$ and $\tilde{z}$ are zero mean and uncorrelated with each other. The variances of $\tilde{y}$ and $\tilde{z}$ are equal to 1 and 100 respectively (the helicopter is much louder than the pilot).

Our goal is to estimate the voice of the pilot from the measurements. By Theorem 12.2, the linear MMSE estimator of $\tilde{y}$ given $\tilde{x}$ equals

$$\ell_{\text{MMSE}}(\tilde{x}) = \Sigma_{\tilde{x}\tilde{y}}^T \Sigma_{\tilde{x}}^{-1} \tilde{x}. \tag{12.41}$$

By the independence assumptions, Corollary 8.23 and linearity of expectation,

$$\begin{aligned}
\mathrm{Var}[\tilde{x}[1]] &= \mathrm{Var}[\tilde{y}] + h^2\mathrm{Var}[\tilde{z}] && (12.42)\\
&= 1 + 100h^2, && (12.43)\\
\mathrm{Var}[\tilde{x}[2]] &= h^2\mathrm{Var}[\tilde{y}] + \mathrm{Var}[\tilde{z}] && (12.44)\\
&= h^2 + 100, && (12.45)\\
\mathrm{Cov}[\tilde{x}[1]\tilde{x}[2]] &= \mathrm{E}\left[h\tilde{y}^2 + h\tilde{z}^2 + (1+h^2)\tilde{y}\tilde{z}\right] && (12.46)\\
&= h\mathrm{E}\left[\tilde{y}^2\right] + h\mathrm{E}\left[\tilde{z}^2\right] + (1+h^2)\mathrm{E}\left[\tilde{y}\right]\mathrm{E}\left[\tilde{z}\right] && (12.47)\\
&= 101h, && (12.48)
\end{aligned}$$

so the covariance matrix of the features equals

$$\Sigma_{\tilde{x}} = \begin{bmatrix} 1 + 100h^2 & 101h \\ 101h & h^2 + 100 \end{bmatrix}. \tag{12.49}$$

Similarly

$$\begin{aligned}
\mathrm{Cov}[\tilde{x}[1]\tilde{y}] &= \mathrm{E}\left[\tilde{y}^2 + h\tilde{y}\tilde{z}\right] && (12.50)\\
&= \mathrm{E}\left[\tilde{y}^2\right] + h\mathrm{E}\left[\tilde{y}\right]\mathrm{E}\left[\tilde{z}\right] && (12.51)\\
&= 1, && (12.52)\\
\mathrm{Cov}[\tilde{x}[2]\tilde{y}] &= \mathrm{E}\left[h\tilde{y}^2 + \tilde{y}\tilde{z}\right] && (12.53)\\
&= h\mathrm{E}\left[\tilde{y}^2\right] + \mathrm{E}\left[\tilde{y}\right]\mathrm{E}\left[\tilde{z}\right] && (12.54)\\
&= h, && (12.55)
\end{aligned}$$

so the cross-covariance equals

$$\Sigma_{\tilde{x}\tilde{y}} = \begin{bmatrix} 1 \\ h \end{bmatrix}. \tag{12.56}$$

Therefore

$$\begin{aligned}
\ell_{\mathrm{MMSE}}(\tilde{x}) &= \begin{bmatrix} 1 & h \end{bmatrix} \begin{bmatrix} 1 + 100h^2 & 101h \\ 101h & h^2 + 100 \end{bmatrix}^{-1} \tilde{x} && (12.57)\\
&= \begin{bmatrix} 1 & h \end{bmatrix} \frac{1}{100(1-h^2)^2} \begin{bmatrix} h^2 + 100 & -101h \\ -101h & 1 + 100h^2 \end{bmatrix} \tilde{x} && (12.58)\\
&= \frac{1}{100(1-h^2)^2} \begin{bmatrix} 100(1-h^2) & -100h(1-h^2) \end{bmatrix} \tilde{x} && (12.59)\\
&= \frac{\tilde{x}[1] - h\tilde{x}[2]}{1-h^2}. && (12.60)
\end{aligned}$$

In order to evaluate the estimate, we express it in terms of $\tilde{y}$ and $\tilde{z}$:

$$\begin{aligned}
\ell_{\mathrm{MMSE}}(\tilde{x}) &= \frac{\tilde{y} + h\tilde{z} - h(h\tilde{y} + \tilde{z})}{1-h^2} && (12.61)\\
&= \tilde{y}. && (12.62)
\end{aligned}$$

It's perfect! The linear estimator cancels out the noise completely by scaling the second measurement and subtracting it from the first one.

Linear estimation is optimal when the response and the features are jointly Gaussian, as established in the following theorem, which is a generalization of Theorem 8.16 to multiple features.

Theorem 12.5 (Linear estimation is optimal for Gaussian random vectors)**.** *Let $\tilde{x}$ and $\tilde{y}$ be a d-dimensional random vector and a random variable, representing the features and response in a regression problem. We assume that $\tilde{x}$ and $\tilde{y}$ are jointly Gaussian, meaning that*

$$\begin{bmatrix} \tilde{x} \\ \tilde{y} \end{bmatrix} \tag{12.63}$$

is a Gaussian random vector with mean and covariance-matrix parameters

$$\mu := \begin{bmatrix} \mu_{\tilde{x}} \\ \mu_{\tilde{y}} \end{bmatrix}, \qquad \Sigma := \begin{bmatrix} \Sigma_{\tilde{x}} & \Sigma_{\tilde{x}\tilde{y}} \\ \Sigma_{\tilde{x}\tilde{y}}^T & \sigma_{\tilde{y}}^2 \end{bmatrix}, \tag{12.64}$$

where $\mu_{\tilde{x}}$ and $\mu_{\tilde{y}}$ are the means of $\tilde{x}$ and $\tilde{y}$ respectively, $\Sigma_{\tilde{x}}$ is the covariance matrix of $\tilde{x}$, $\sigma_{\tilde{y}}^2$ is the variance of $\tilde{y}$ and $\Sigma_{\tilde{x}\tilde{y}}$ is the cross-covariance between $\tilde{y}$ and $\tilde{x}$. In that case, the minimum MSE (MMSE) estimator of $\tilde{y}$ given $\tilde{x}$ is the linear MMSE estimator

$$\ell_{\text{MMSE}}(\tilde{x}) := \beta_{\text{MMSE}}^T \tilde{x} + \alpha_{\text{MMSE}} \tag{12.65}$$

$$= \Sigma_{\tilde{x}\tilde{y}}^T \Sigma_{\tilde{x}}^{-1} \left(\tilde{x} - \mu_{\tilde{x}}\right) + \mu_{\tilde{y}}. \tag{12.66}$$

Proof By Theorem 5.25, the conditional distribution of $\tilde{y}$ given $\tilde{x} = x$ is Gaussian with mean parameter equal to

$$\mu_{\text{cond}} = \Sigma_{\tilde{x}\tilde{y}}^T \Sigma_{\tilde{x}}^{-1} \left(x - \mu_{\tilde{x}}\right) + \mu_{\tilde{y}}. \tag{12.67}$$

By Lemma 11.3 this is the conditional mean function $\mu_{\tilde{y}\,|\,\tilde{x}}$ of $\tilde{y}$ given $\tilde{x} = x$. Recall that by Theorem 7.59, the conditional mean, obtained by plugging $\tilde{x}$ into $\mu_{\tilde{y}\,|\,\tilde{x}}$, is the MMSE estimator of $\tilde{y}$ given $\tilde{x}$. Since the conditional mean is equal to the linear MMSE estimator, the proof is complete. ■

In general, the linear MMSE estimator is not necessarily equal to the MMSE estimator, because it cannot capture nonlinear dependence between the features and the response. This is illustrated by Example 8.18 and Section 12.7.1 (see Figure 12.26).

12.1.2 Ordinary Least Squares

In this section, we explain how to perform linear regression using data. Our strategy is to approximate the linear minimum MSE estimator derived in Theorem 12.2. The estimator only depends on the covariance matrix of the features,

the cross-covariance between the features and the response, and the means of the features and the response. Estimating the cross-covariance from data is straightforward. Since each entry is equal to the covariance between a feature and the response, we approximate each entry using the corresponding sample covariance.

Definition 12.6 (Sample cross-covariance)**.** *Given n pairs (x_1, y_1), (x_2, y_2), $\ldots$, (x_n, y_n), each consisting of a response y_i and a corresponding d-dimensional feature vector x_i, we denote the bag of responses by $Y := \{y_1, y_2, \ldots, y_n\}$, the bag of features by $X := \{x_1, x_2, \ldots, x_n\}$, and the bag of entries corresponding to the jth feature by $X[j] := \{x_1[j], \ldots, x_n[j]\}$, for $1 \leq j \leq d$. The sample cross-covariance between the response and the feature is*

$$\Sigma_{XY} := \frac{1}{n-1} \sum_{i=1}^{n} \operatorname{ct}(x_i) \operatorname{ct}(y_i) = \begin{bmatrix} c(X[1], Y) \\ c(X[2], Y) \\ \cdots \\ c(X[d], Y) \end{bmatrix}, \tag{12.68}$$

where $\operatorname{ct}(x_i) := x_i - m(X)$ and $\operatorname{ct}(y_i) - m(Y)$ are the result of centering the features and the response using the corresponding sample means $m(X)$ and $m(Y)$ for $1 \leq i \leq n$, and $c(X[j], Y)$ is the sample covariance of $X[j]$ and Y for $1 \leq j \leq d$.

To estimate the linear minimum MSE estimator defined in (12.13), we replace the means of the response and features with their respective sample means $m(Y)$ and $m(X)$, the covariance matrix of the features with the sample covariance matrix Σ_X, and the cross-covariance of the features and the response with the sample cross-covariance Σ_{XY}:

$$\ell_{\text{MMSE}}(x_i) \approx \Sigma_{XY}^T \Sigma_X^{-1} (x_i - m(X)) + m(Y). \tag{12.69}$$

This is known as the *ordinary-least-squares* (OLS) estimator, because it can also be derived as the minimizer of the residual sum of squares, as established in the following theorem.

Theorem 12.7 (Linear regression via ordinary least squares)**.** *Given a dataset formed by n pairs of a response y_i and a corresponding vector x_i containing d features $(1 \leq i \leq n)$, we denote the bag of responses by $Y := \{y_1, y_2, \ldots, y_n\}$ and the bag of features by $X := \{x_1, x_2, \ldots, x_n\}$. If the sample covariance matrix Σ_X of the features is invertible, the ordinary least-squares estimator (OLS) of the response given the features is*

$$\ell_{\text{OLS}}(x_i) := \beta_{\text{OLS}}^T x_i + \alpha_{\text{OLS}}, \quad 1 \leq i \leq n, \tag{12.70}$$

$$\beta_{\text{OLS}} := \Sigma_X^{-1} \Sigma_{XY}, \tag{12.71}$$

$$\alpha_{\text{OLS}} := m(Y) - \Sigma_{XY}^T \Sigma_X^{-1} m(X), \tag{12.72}$$

where Σ_{XY} is the sample cross-covariance of X and Y, and $m(X)$ and $m(Y)$ are the sample means of X and Y.

The OLS estimator produces an optimal affine estimate, in the sense that it minimizes the residual sum of squares,

$$(\beta_{\text{OLS}}, \alpha_{\text{OLS}}) = \arg \min_{\alpha \in \mathbb{R}, \beta \in \mathbb{R}^d} \sum_{i=1}^{n} \left(y_i - \beta^T x_i - \alpha \right)^2. \tag{12.73}$$

Proof The result can be established by generalizing the proof of Theorem 8.17 to multiple features, following a similar argument to the proof of Theorem 12.2 (see Exercise 12.4). ■

For large-scale regression problems with many features, it may not be possible to obtain the OLS estimator using the formula in Theorem 12.7, due to the computational cost of inverting the sample covariance matrix of the features. In such cases, iterative optimization methods, such as conjugate gradients (Shewchuk *et al.*, 1994), can be applied to directly minimize the residual sum of squares.

It is often convenient to express the OLS estimator in terms of a *design matrix*, containing the features, and vector representing the response.

Lemma 12.8 (OLS estimator in matrix-vector form)**.** *Let* (x_1, y_1), (x_2, y_2), $\ldots$, (x_n, y_n) *be a dataset formed by* n *pairs of a response* $y_i \in \mathbb{R}$ *and a corresponding vector* $x_i \in \mathbb{R}^d$ *containing* d *features (where* $1 \leq i \leq n$*). We arrange the feature vectors to form the design matrix*

$$X := \begin{bmatrix} x_1 & x_2 & \cdots & x_n \end{bmatrix} \tag{12.74}$$

and the response values to form the response vector

$$y = \begin{bmatrix} y_1 \\ y_2 \\ \cdots \\ y_n \end{bmatrix}. \tag{12.75}$$

If the features and responses are centered to have zero sample mean, then the OLS estimator of the response given the features equals

$$\beta_{\text{OLS}} = \arg \min_{\beta \in \mathbb{R}^d} ||X^T \beta - y||_2^2 \tag{12.76}$$

$$= \left(X X^T \right)^{-1} X y. \tag{12.77}$$

Proof If the features have zero sample mean, the covariance matrix of the features equals

$$\Sigma_X = \frac{1}{n-1} \sum_{i=1}^{n} x_i x_i^T \tag{12.78}$$

$$= \frac{1}{n-1} X X^T, \tag{12.79}$$

so its inverse is $(n-1)\left(X X^T\right)^{-1}$. If the sample mean of the response values is

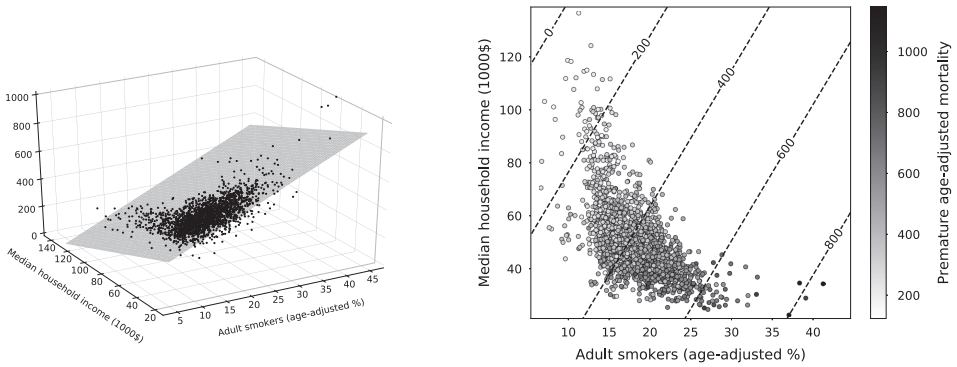


Figure 12.3 Linear model for premature mortality. The figure provides two visualizations of the linear-regression model described in Example 12.9, which estimates premature mortality from tobacco use and income in United States counties. On the left, the data points are represented in 3D, with the features as the horizontal coordinates and the response as the vertical coordinate. The OLS model is the plane that minimizes the sum of squared vertical distances between the plane and the data. On the right, a scatterplot of the data points, color coded to show the corresponding response, is superposed onto the contour lines of the OLS model.

also zero, the cross-covariance between the features and the response equals

$$\Sigma_{XY} = \frac{1}{n-1}\sum_{i=1}^{n} x_i y_i \tag{12.80}$$

$$= \frac{1}{n-1} Xy. \tag{12.81}$$

The result then follows from Theorem 12.7. ■

Example 12.9 (Premature mortality: OLS model)**.** We consider the problem of estimating premature mortality in 2,091 United States counties based on tobacco use and income, using data extracted from Dataset 22. Our response of interest is the number of deaths among residents under age 75 per 100,000 people, which is a common measure of premature mortality. The features are the percentage of adult smokers and the median household income in each county. Both premature mortality and tobacco use depend highly on the age distribution of the population (older people are more likely to die and also more likely to smoke), so the corresponding data are adjusted to correct for age.

The sample mean of the premature-mortality measure is 408 and the sample standard deviation is 116. The sample mean and sample covariance matrix of tobacco use and income are

$$m(X) = \begin{bmatrix} 18 \\ 50.9 \end{bmatrix}, \qquad \Sigma_X = \begin{bmatrix} 13.6 & -30.6 \\ -30.6 & 190 \end{bmatrix}. \tag{12.82}$$

The sample cross-covariance between the response and the two features is

$$\Sigma_{XY} = \begin{bmatrix} 306 \\ -1057 \end{bmatrix}. \tag{12.83}$$

By Theorem 12.7, the coefficients and additive constant of the OLS estimator of premature mortality given tobacco use and income are

$$\beta_{\text{OLS}} = \Sigma_X^{-1}\Sigma_{XY} = \begin{bmatrix} 15.7 \\ -3.04 \end{bmatrix}, \tag{12.84}$$

$$\alpha_{\text{OLS}} = m(Y) - \Sigma_{XY}^T\Sigma_X^{-1}m(X) = 281, \tag{12.85}$$

which yields the OLS model

$$\ell_{\text{OLS}}\left(x_{\text{tobacco}}, x_{\text{income}}\right) = 15.7\, x_{\text{tobacco}} - 3.04\, x_{\text{income}} + 281. \tag{12.86}$$

According to the model, counties with higher tobacco use and lower household income have higher premature mortality. Figure 12.3 provides two different visualizations of the model.

The linear coefficient associated to each feature in a linear regression model predicts the rate of change of the response with respect to the feature *when the rest of the features are fixed*. For instance, in Example 12.9 the linear model predicts a difference of 15.7 premature deaths (per 100,000 people) between two counties where the fraction of adult smokers differs by 1%, as long as they have the same median household income. Similarly, it predicts a difference of -3.04 premature deaths between two counties with a difference of $1,000 in median household income, as long as the tobacco use is the same.

The magnitude of the coefficients in OLS models depends on the unit used to measure the corresponding feature. Consequently, we cannot use them to compare the relative importance of the features. In Example 12.9, tobacco use is represented as a percentage; if it were a fraction instead, the corresponding coefficient would be 100 times larger. Similarly, income is measured in thousands of dollars; if it were measured in dollars, the coefficient would be 1,000 times smaller.

12.1.3 Explained Variance

In this section, we derive a decomposition of variance, which generalizes the one derived in Section 8.5.3 for simple linear regression, and leverage it to perform model evaluation and measure feature importance. The key insight is that, just as in simple linear regression, the linear MMSE estimator is uncorrelated with its corresponding residual. This enables us to decompose the variance of the response as the sum of the variance of the estimator and the variance of the residual.

Theorem 12.10 (Linear MMSE estimator and residual are uncorrelated)**.** *Let $\tilde{x}$ and $\tilde{y}$ be a d-dimensional random vector and a random variable, representing*

the features and response in a regression problem. The linear MMSE estimator $\ell_{\text{MMSE}}(\tilde{x})$ *of* $\tilde{y}$ *given* $\tilde{x}$ *is uncorrelated with the residual* $\tilde{y} - \ell_{\text{MMSE}}(\tilde{x})$.

Proof The centered linear MMSE estimator equals

$$\begin{aligned} \text{ct}\left(\ell_{\text{MMSE}}\left(\tilde{x}\right)\right) &= \beta_{\text{MMSE}}^T \tilde{x} + \alpha_{\text{MMSE}} - \text{E}\left[\beta_{\text{MMSE}}^T \tilde{x} + \alpha_{\text{MMSE}}\right] && (12.87)\\ &= \beta_{\text{MMSE}}^T \,\text{ct}\left(\tilde{x}\right). && (12.88) \end{aligned}$$

Let us express the corresponding residual in terms of the centered response and features,

$$\begin{aligned} \tilde{y} - \ell_{\text{MMSE}}\left(\tilde{x}\right) &= \tilde{y} - \beta_{\text{MMSE}}^T \left(\tilde{x} - \text{E}\left[\tilde{x}\right]\right) - \text{E}\left[\tilde{y}\right] && (12.89)\\ &= \text{ct}\left(\tilde{y}\right) - \beta_{\text{MMSE}}^T \,\text{ct}\left(\tilde{x}\right), && (12.90) \end{aligned}$$

where $\text{ct}\left(\tilde{y}\right) := \tilde{y} - \text{E}\left[\text{ct}\left(\tilde{y}\right)\right]$ and $\text{ct}\left(\tilde{x}\right) := \tilde{x} - \text{E}\left[\text{ct}\left(\tilde{x}\right)\right]$. This immediately implies that the mean of the residual is zero by linearity of expectation,

$$\text{E}\left[\tilde{y} - \ell_{\text{MMSE}}\left(\tilde{x}\right)\right] = \text{E}\left[\text{ct}\left(\tilde{y}\right)\right] - \Sigma_{\tilde{x}\tilde{y}}^T \Sigma_{\tilde{x}}^{-1} \text{E}\left[\text{ct}\left(\tilde{x}\right)\right] = 0. \tag{12.91}$$

Consequently, the covariance between the estimator and the residual equals

$$\begin{aligned} &\text{Cov}\left[\ell_{\text{MMSE}}\left(\tilde{x}\right), \tilde{y} - \ell_{\text{MMSE}}\left(\tilde{x}\right)\right] && (12.92)\\ &= \text{E}\left[\text{ct}\left(\ell_{\text{MMSE}}\left(\tilde{x}\right)\right) \text{ct}\left(\tilde{y} - \ell_{\text{MMSE}}\left(\tilde{x}\right)\right)\right] && (12.93)\\ &= \text{E}\left[\beta_{\text{MMSE}}^T \,\text{ct}\left(\tilde{x}\right) \left(\text{ct}\left(\tilde{y}\right) - \text{ct}\left(\tilde{x}\right)^T \beta_{\text{MMSE}}\right)\right] && (12.94)\\ &= \beta_{\text{MMSE}}^T \text{E}\left[\text{ct}\left(\tilde{x}\right) \text{ct}\left(\tilde{y}\right)\right] - \beta_{\text{MMSE}}^T \text{E}\left[\text{ct}\left(\tilde{x}\right) \text{ct}\left(\tilde{x}\right)^T\right] \beta_{\text{MMSE}} && (12.95)\\ &= \Sigma_{\tilde{x}\tilde{y}}^T \Sigma_{\tilde{x}}^{-1} \Sigma_{\tilde{x}\tilde{y}} - \Sigma_{\tilde{x}\tilde{y}}^T \Sigma_{\tilde{x}}^{-1} \Sigma_{\tilde{x}} \Sigma_{\tilde{x}}^{-1} \Sigma_{\tilde{x}\tilde{y}} && (12.96)\\ &= \Sigma_{\tilde{x}\tilde{y}}^T \Sigma_{\tilde{x}}^{-1} \Sigma_{\tilde{x}\tilde{y}} - \Sigma_{\tilde{x}\tilde{y}}^T \Sigma_{\tilde{x}}^{-1} \Sigma_{\tilde{x}\tilde{y}} = 0. && (12.97) \end{aligned}$$

■

Theorem 12.11 (Decomposition of variance)**.** *Let* $\tilde{x}$ *and* $\tilde{y}$ *be a* d*-dimensional random vector and a random variable, representing the features and response in a regression problem. The variance of* $\tilde{y}$ *can be decomposed into the sum of the variance of the linear MMSE estimator* $\ell_{\text{MMSE}}(\tilde{x})$ *of* $\tilde{y}$ *given* $\tilde{x}$ *and the MSE incurred by the estimator,*

$$\text{Var}\left[\tilde{y}\right] = \text{Var}\left[\ell_{\text{MMSE}}(\tilde{x})\right] + \text{MSE}, \qquad \text{MSE} := \text{E}\left[\left(\tilde{y} - \ell_{\text{MMSE}}(\tilde{x})\right)^2\right]. \tag{12.98}$$

Proof By Theorem 12.10 the linear MMSE estimator $\ell_{\text{MMSE}}(\tilde{x})$ and the residual $\tilde{y} - \ell_{\text{MMSE}}(\tilde{x})$ are uncorrelated. By (12.91) the mean of the residual is zero, so its variance is equal to the MSE. The result then follows from Corollary 8.23. ■

The decomposition of variance in Theorem 12.11 has an intuitive geometric explanation, if we interpret the features and response as vectors in the vector space of zero-mean random variables defined in Section 8.7 (assuming they are centered). As discussed in Section 12.1.1 and depicted in Figure 12.2, the residual $\tilde{y} - \ell_{\text{MMSE}}(\tilde{x})$ of the linear MMSE estimator $\ell_{\text{MMSE}}(\tilde{x})$ is orthogonal to the subspace spanned by the features, and consequently also to $\ell_{\text{MMSE}}(\tilde{x})$ itself. The sum

of the residual and the estimator is equal to the response $\tilde{y}$, so by the Pythagorean theorem, the squared length of $\tilde{y}$ (the variance of the response) is equal to the sum of the squared lengths of the residual (the MSE) and the estimator (the variance of $\ell_{\text{MMSE}}(\tilde{x})$).

A useful evaluation metric for the linear approximation provided by the linear MMSE estimator is the fraction of variance *explained* by the estimator. The fraction is called the coefficient of determination R^2, as in simple linear regression (see Definition 8.26).

Definition 12.12 (Coefficient of determination)**.** *Let $\tilde{x}$ and $\tilde{y}$ be a d-dimensional random vector and a random variable, representing the features and response in a regression problem. The coefficient of determination is the ratio between the variance of the linear MMSE estimator $\ell_{\text{MMSE}}(\tilde{x})$ of $\tilde{y}$ given $\tilde{x}$ and the variance of $\tilde{y}$,*

$$R^2 := \frac{\text{Var}\left[\ell_{\text{MMSE}}(\tilde{x})\right]}{\text{Var}\left[\tilde{y}\right]}. \tag{12.99}$$

As in the case of simple linear regression (see Theorem 8.27), the coefficient of determination can be expressed in terms of the MSE, and is bounded between zero and one. When it equals one, the MSE is zero, so the estimator is perfect. When it equals zero, the MSE is equal to the variance of the response.

Theorem 12.13 (Properties of the coefficient of determination)**.** *Let $\tilde{x}$ and $\tilde{y}$ be a d-dimensional random vector and a random variable, representing the features and response in a regression problem. The coefficient of determination equals*

$$R^2 := 1 - \frac{\text{MSE}}{\text{Var}\left[\tilde{y}\right]}, \qquad \text{MSE} := \text{E}\left[(\tilde{y} - \ell_{\text{MMSE}}(\tilde{x}))^2\right]. \tag{12.100}$$

In addition, R^2 is bounded between zero and one:

$$0 \leq R^2 \leq 1. \tag{12.101}$$

Proof By Theorem 12.11, $\text{Var}\left[\tilde{y}\right] = \text{Var}\left[\ell_{\text{MMSE}}(\tilde{x})\right] + \text{MSE}$, which directly implies

$$R^2 := \frac{\text{Var}\left[\ell_{\text{MMSE}}(\tilde{x})\right]}{\text{Var}\left[\tilde{y}\right]} \tag{12.102}$$

$$= \frac{\text{Var}\left[\tilde{y}\right] - \text{MSE}}{\text{Var}\left[\tilde{y}\right]}, \tag{12.103}$$

and also $\text{Var}\left[\ell_{\text{MMSE}}(\tilde{x})\right] \leq \text{Var}\left[\tilde{y}\right]$ (because the MSE cannot be negative), so that $0 \leq R^2 \leq 1$. ■

The following example uses the coefficient of determination to compare different models. It also shows that we can measure the importance of a feature in the linear model by comparing the coefficient of determination with and without the feature.

Example 12.14 (Premature mortality: Model evaluation)**.** In Example 12.9 we

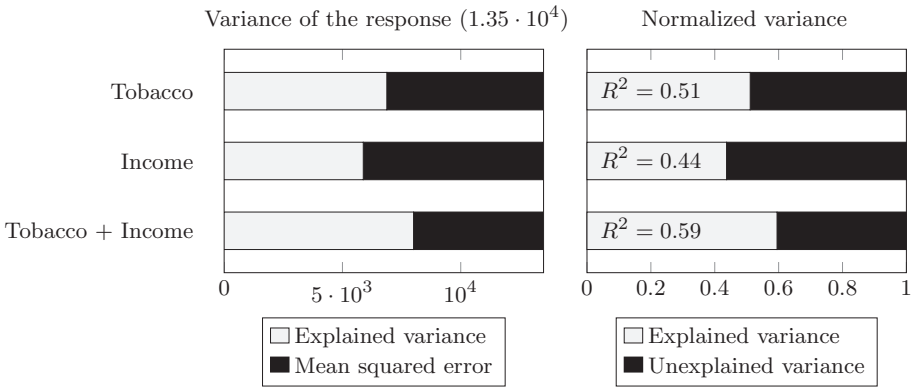


Figure 12.4 Explained variance and coefficient of determination. The left graph depicts the variance decomposition in Theorem 12.11 for the three linear models described in Example 12.14. For each model, the variance of the response is equal to the sum of the variance of the OLS estimator and the MSE. The right graph shows the result of normalizing the decomposition to obtain the corresponding coefficients of determination following Definition 12.12.

build a linear model to estimate premature mortality using tobacco use and income. The sample variance of the response is $1.35 \cdot 10^4$. The sample variance of the estimate produced by the model is $8.00 \cdot 10^3$. The coefficient of determination R^2 is the ratio between these sample variances, which equals 0.59. The linear model explains 59% of the variance in the data.

We can use the coefficient of determination to compare different models. Consider the OLS models based on only one of the features,

$$\ell_{\text{OLS}}\left(x_{\text{tobacco}}\right) = 22.5\, x_{\text{tobacco}} + 2, \tag{12.104}$$

$$\ell_{\text{OLS}}\left(x_{\text{income}}\right) = -5.57\, x_{\text{income}} + 692. \tag{12.105}$$

As depicted in Figure 12.4, the coefficient of determination for the tobacco model ($R^2 = 0.51$) is larger than for the income model ($R^2 = 0.44$). The linear association of premature mortality with tobacco is therefore stronger than with income.

We can also use the coefficient of determination to measure feature importance, by quantifying the benefit of incorporating each feature into the model. If we add income to the tobacco model, the R^2 increases by 8%. The gain is much smaller than the R^2 of the income model, because the two features are correlated (their correlation coefficient equals -0.6). If we add tobacco to the income model, the R^2 increases by 15%. This further confirms that tobacco use has a stronger linear association with premature mortality than household income.

12.1.4 Estimation Of Linear Causal Effects

Each coefficient in a linear regression model indicates the rate of change of the response with respect to the corresponding feature, if the rest of the features are fixed. However, this does *not* necessarily mean that the feature has a *causal effect* on the response, in the sense that the response would actually change as predicted by the linear model, if we were to modify the feature. This is explained in Section 8.8, where we consider the problem of estimating the causal effect of a treatment on an outcome of interest. If the treatment is randomized, Theorem 8.36 establishes that linear causal effects are encoded in the correlation between the treatment and the outcome. However, in the absence of randomization we cannot trust the correlation, because it can be distorted by confounding variables (see Theorem 8.38 and Examples 8.37 and 8.39). In this section, we show that under certain assumptions, fitting a linear regression model automatically adjusts for confounders, *as long as we include them in the model.*

Let us denote the treatment variable $\tilde{t}$ and the observed outcome by $\tilde{y}$. For simplicity, we consider a single confounder $\tilde{c}$. To define the causal relationship between the treatment, the confounder and the outcome formally, we define the potential outcome $\widetilde{\text{po}}_{t,c}$, which represents the value of the outcome in a hypothetical situation where the treatment is set to equal t and the confounder is set to equal c. The observed outcome $\tilde{y}$ is equal to $\widetilde{\text{po}}_{t,c}$ if $\tilde{t} = t$ and $\tilde{c} = c$. For other values of c and t, the potential outcome is an unobserved counterfactual.

We assume that both the treatment and the confounder have a linear effect on the potential outcome, so that the mean of $\widetilde{\text{po}}_{t,c}$ is a linear function of t and c:

$$\text{E}\left[\widetilde{\text{po}}_{t,c}\right] = \beta_{\text{true}} t + \gamma c, \tag{12.106}$$

for some real constants β_{true} and γ. We also assume that the distribution of $\widetilde{\text{po}}_{t,c}$ is the same as the distribution of $\widetilde{\text{po}}_{t,c}$ conditioned on the intersection of the events $\tilde{t} = t$ and $\tilde{c} = c$. This is similar to the independence assumption in Theorem 8.36, with the important exception that we also condition on the confounder. As a result, the dependence between the treatment and the potential outcome is completely governed by the confounder.

The following theorem shows that under these assumptions, we can estimate the coefficient β_{true}, which encodes the true causal effect of the treatment on the potential outcome, by fitting a linear regression model where the confounder is included as a feature.

Theorem 12.15 (Estimation of linear causal effects via linear regression)**.** *Let $\widetilde{\text{po}}_{t,c}$ denote the potential outcomes associated to a treatment $\tilde{t}$ and a confounding variable $\tilde{c}$. The observed outcome $\tilde{y}$ is equal to $\widetilde{\text{po}}_{t,c}$, if $\tilde{t} = t$ and $\tilde{c} = c$. We assume that the treatment and confounder have a linear effect on the mean of $\widetilde{\text{po}}_{t,c}$:*

$$\text{E}\left[\widetilde{\text{po}}_{t,c}\right] = \beta_{\text{true}} t + \gamma c, \tag{12.107}$$

for some real constants β_{true} and γ, where $\tilde{t}$ and $\tilde{c}$ are standardized. In addition,

we assume that the distribution of $\widetilde{\mathrm{po}}_{t,c}$ *is the same as the distribution of* $\widetilde{\mathrm{po}}_{t,c}$ *conditioned on the intersection of the events* $\tilde{t} = c$ *and* $\tilde{c} = c$*. In that case, the coefficients of the linear MMSE estimate of the observed outcome* $\tilde{y}$ *given* $\tilde{t}$ *and* $\tilde{c}$ *equal*

$$\beta_{\mathrm{MMSE}} = \begin{bmatrix} \beta_{\mathrm{true}} \\ \gamma \end{bmatrix}. \tag{12.108}$$

Proof Without loss of generality, we assume that all variables are continuous and centered to have zero mean. We define the random vector of features

$$\tilde{x} := \begin{bmatrix} \tilde{t} \\ \tilde{c} \end{bmatrix}. \tag{12.109}$$

The covariance matrix of the features equals

$$\Sigma_{\tilde{x}} = \begin{bmatrix} \sigma_{\tilde{t}}^2 & \sigma_{\tilde{t},\tilde{c}} \\ \sigma_{\tilde{t},\tilde{c}} & \sigma_{\tilde{c}}^2 \end{bmatrix}, \tag{12.110}$$

where $\sigma_{\tilde{t}}^2$ and $\sigma_{\tilde{c}}^2$ are the variances of $\tilde{t}$ and $\tilde{c}$, and $\sigma_{\tilde{t},\tilde{c}}$ is the covariance between $\tilde{t}$ and $\tilde{c}$.

To derive the coefficients of the linear MMSE estimator, we need to compute the cross-covariance between the features and the observed outcome. By (8.218) in the proof of Theorem 8.38, under our assumptions,

$$\mu_{\tilde{y}\tilde{t}\,|\,\tilde{t},\tilde{c}}(t,c) = \beta_{\mathrm{true}} t^2 + \gamma c t, \tag{12.111}$$

and similarly (exchanging the roles of $\tilde{t}$ and $\tilde{c}$)

$$\mu_{\tilde{y}\tilde{c}\,|\,\tilde{t},\tilde{c}}(t,c) = \gamma c^2 + \beta_{\mathrm{true}} c t. \tag{12.112}$$

By iterated expectation, this implies

$$\begin{aligned} \mathrm{Cov}\left[\tilde{y},\tilde{t}\right] = \mathrm{E}\left[\tilde{y}\tilde{t}\right] &= \mathrm{E}\left[\mu_{\tilde{y}\tilde{t}\,|\,\tilde{t},\tilde{c}}(\tilde{t},\tilde{c})\right] && (12.113)\\ &= \mathrm{E}\left[\beta_{\mathrm{true}}\tilde{t}^2 + \gamma\tilde{t}\tilde{c}\right] && (12.114)\\ &= \beta_{\mathrm{true}}\mathrm{E}\left[\tilde{t}^2\right] + \gamma\mathrm{E}\left[\tilde{t}\tilde{c}\right], && (12.115) \end{aligned}$$

which generalizes Theorem 8.38 to the case where the variables are not standardized. By the same argument,

$$\mathrm{Cov}\left[\tilde{y},\tilde{c}\right] = \gamma\mathrm{E}\left[\tilde{c}^2\right] + \beta_{\mathrm{true}}\mathrm{E}\left[\tilde{t}\tilde{c}\right]. \tag{12.116}$$

The cross-covariance therefore equals

$$\Sigma_{\tilde{x}\tilde{y}} = \begin{bmatrix} \beta_{\mathrm{true}}\sigma_{\tilde{t}}^2 + \gamma\sigma_{\tilde{t},\tilde{c}} \\ \gamma\sigma_{\tilde{c}}^2 + \beta_{\mathrm{true}}\sigma_{\tilde{t},\tilde{c}} \end{bmatrix}. \tag{12.117}$$

By Theorem 12.2, the coefficients of the linear MMSE estimator equal

$$\beta_{\text{MMSE}} = \Sigma_{\tilde{x}}^{-1}\Sigma_{\tilde{x}\tilde{y}} = \begin{bmatrix} \sigma_{\tilde{t}}^2 & \sigma_{\tilde{t},\tilde{c}} \\ \sigma_{\tilde{t},\tilde{c}} & \sigma_{\tilde{c}}^2 \end{bmatrix}^{-1} \begin{bmatrix} \beta_{\text{true}}\sigma_{\tilde{t}}^2 + \gamma\sigma_{\tilde{t},\tilde{c}} \\ \gamma\sigma_{\tilde{c}}^2 + \beta_{\text{true}}\sigma_{\tilde{t},\tilde{c}} \end{bmatrix} \tag{12.118}$$

$$= \frac{1}{\sigma_{\tilde{t}}^2\sigma_{\tilde{c}}^2 - \sigma_{\tilde{t},\tilde{c}}^2} \begin{bmatrix} \sigma_{\tilde{c}}^2 & -\sigma_{\tilde{t},\tilde{c}} \\ -\sigma_{\tilde{t},\tilde{c}} & \sigma_{\tilde{t}}^2 \end{bmatrix} \begin{bmatrix} \beta_{\text{true}}\sigma_{\tilde{t}}^2 + \gamma\sigma_{\tilde{t},\tilde{c}} \\ \gamma\sigma_{\tilde{c}}^2 + \beta_{\text{true}}\sigma_{\tilde{t},\tilde{c}} \end{bmatrix} \tag{12.119}$$

$$= \frac{1}{\sigma_{\tilde{t}}^2\sigma_{\tilde{c}}^2 - \sigma_{\tilde{t},\tilde{c}}^2} \begin{bmatrix} \beta_{\text{true}}\sigma_{\tilde{t}}^2\sigma_{\tilde{c}}^2 - \beta_{\text{true}}\sigma_{\tilde{t},\tilde{c}}^2 \\ \gamma\sigma_{\tilde{t}}^2\sigma_{\tilde{c}}^2 - \gamma\sigma_{\tilde{t},\tilde{c}}^2 \end{bmatrix} \tag{12.120}$$

$$= \begin{bmatrix} \beta_{\text{true}} \\ \gamma \end{bmatrix}, \tag{12.121}$$

revealing the average causal effect of the treatment (and the confounder). ■

Theorem 12.15 shows that (as long as the assumptions of the theorem hold) we can automatically adjust for a confounder by fitting a *long* regression model, which includes the confounder as a feature. In contrast, a *short* regression model without the confounder does *not* reveal the correct average causal effect, as established in Theorem 8.38. Figure 12.5 illustrates this phenomenon for the scenario described in Examples 8.37 and 8.39. The coefficient corresponding to the treatment in the short regression depends on the correlation between the confounder and the treatment, and is different to the true causal effect (unless the treatment is randomized to neutralize the confounder) which equals zero. In contrast, in the long regression that includes the confounder, the coefficient always correctly identifies that there is no true causal effect.

Example 12.16 (Unemployment in Spain)**.** We consider the data in Figure 8.12, which shows that unemployment in Spain is negatively correlated with temperature. As explained in Section 8.8, we suspect that this correlation is due to a confounder: the number of tourists visiting Spain. Inspired by Theorem 12.15, we fit a long regression model where the response is unemployment and the features are temperature and number of tourists. Figure 12.6 compares this linear model to a short linear regression model where the only feature is the temperature. The coefficient corresponding to temperature in the long regression model is very small (and positive, instead of negative). This suggests that, if we adjust for the number of tourists, there is no linear causal effect of temperature on unemployment.

To further analyze the linear relationship between temperature and unemployment, we compute the coefficient of determination of the short regression model ($R^2 = 0.042$), the long regression model ($R^2 = 0.12026$), and a simple linear regression model where the only feature is the number of tourists ($R^2 = 0.12024$). The coefficients of determination show that the number of tourists explains much more variance than the temperature. More importantly, if we incorporate temperature to the model that only relies on the number of tourists, the increase in explained variance is minuscule ($2 \cdot 10^{-5}$!). This is conclusive evidence that the

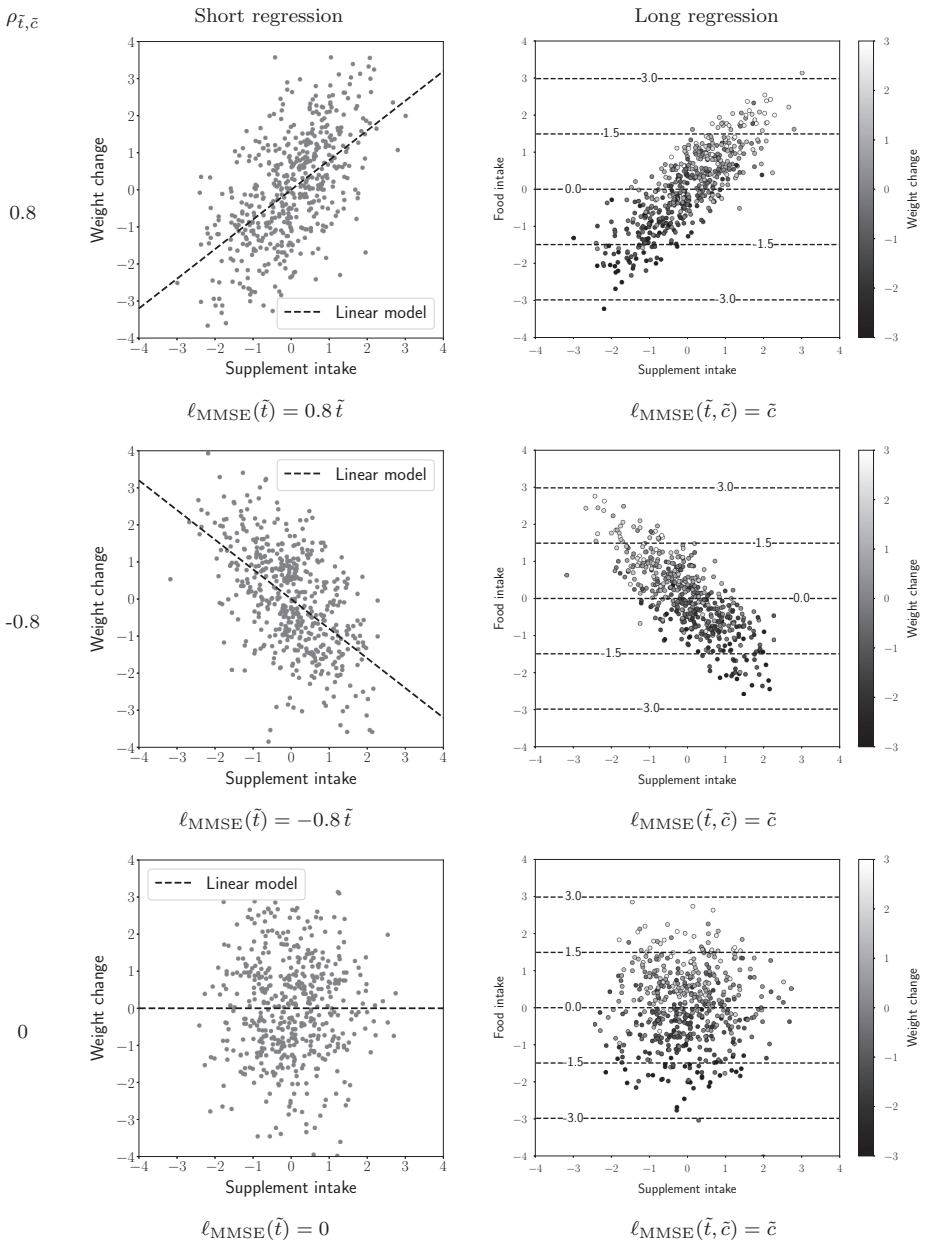


Figure 12.5 Adjusting for a confounder via long regression. The left column depicts a *short* linear regression model for the scenario described in Examples 8.37 and 8.39, where the response is the observed outcome $\tilde{y}$ and the only feature is the treatment $\tilde{t}$. The coefficient corresponding to $\tilde{t}$ changes depending on the correlation between $\tilde{t}$ and the confounder $\tilde{c}$. The right column depicts a *long* linear regression model, where the response is the observed outcome $\tilde{y}$ and the features are $\tilde{t}$ and $\tilde{c}$. The coefficient corresponding to $\tilde{t}$ is zero, regardless of the correlation between $\tilde{t}$ and $\tilde{c}$, correctly revealing that the supplement is useless.

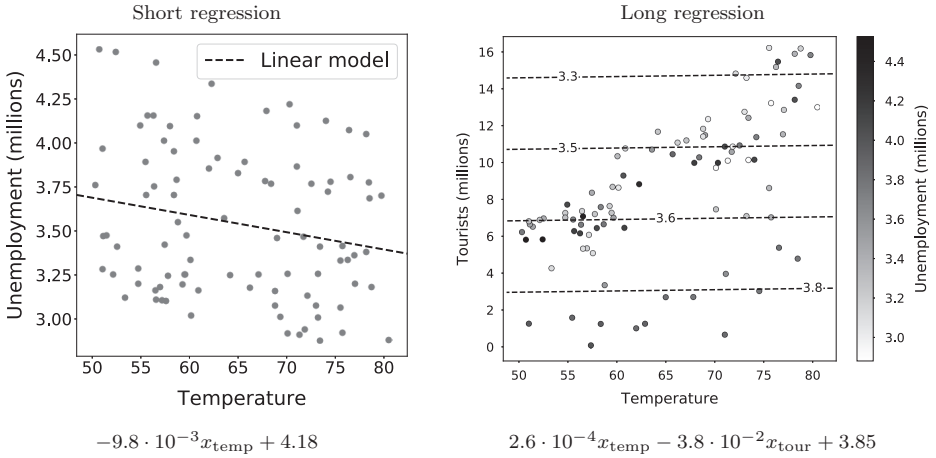


Figure 12.6 Unemployment, temperature and tourists. The left graph shows a simple linear regression model where the response is the unemployment in Spain between 2015 and 2022, and the feature is the corresponding average temperature. This short regression model indicates a negative association between unemployment and temperature. The right graph shows a long linear regression model, which also includes the number of tourists visiting Spain. In the long regression model, the coefficient corresponding to temperature is much smaller and positive, which suggests that the negative association observed in the short regression is an artifact of the correlation between the temperature and the number of tourists.

observed linear dependence between unemployment and temperature is due to the confounder.

12.2 Generalization and Overfitting In Linear Regression

One of the main goals in regression and classification is to predict the response of interest from new test data, which is distinct from the data used to fit or train the models. A prediction model that fits the training data well, but fails to *generalize* to the test data, is said to *overfit*. Figure 12.7 provides a cartoon illustration of generalization and overfitting. In this section, we study under what conditions we can expect linear-regression models to generalize effectively, or to overfit.

12.2.1 Linear-Response Model With Additive Noise

In order to study the generalization behavior of linear-regression models, we assume that the data satisfy the fundamental premise of these models: the response is approximately equal to a linear function of the features. More precisely, we consider a response that can be decomposed into a *signal* component, which is

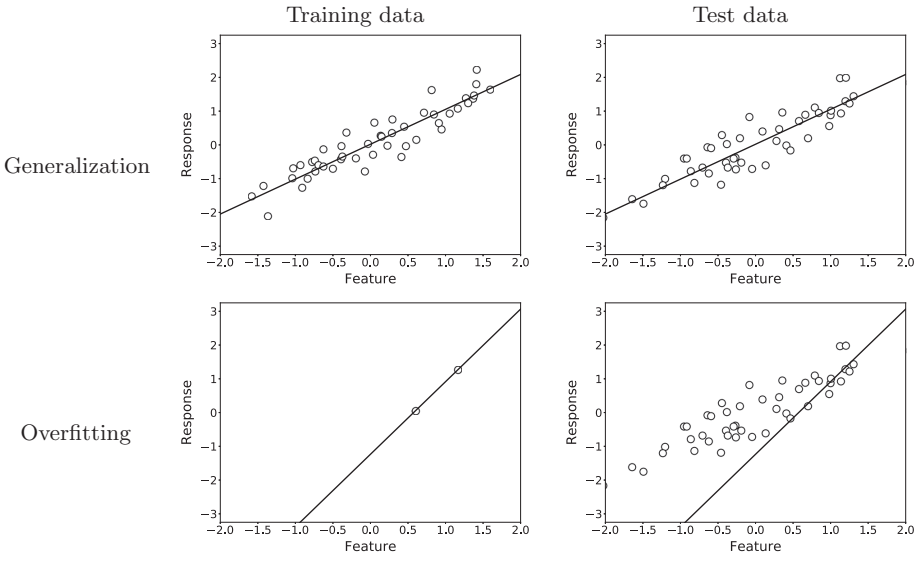


Figure 12.7 Generalization and overfitting. The left column depicts two training sets, consisting of a single feature and a response. The straight lines represent linear-regression models fit to the data. The right column shows held-out test data, not used to train the models. The model in the top row generalizes well: it approximates the test data as well as the training data. In contrast, the model in the bottom row overfits: it yields a perfect estimate of the training data, but has terrible performance on the test data.

a linear function of the features, and a *noise* component independent from the features. If we represent the features as a d-dimensional random vector $\tilde{x}$, the response equals

$$\tilde{y} := \tilde{x}^T \beta_{\text{true}} + \tilde{z} \tag{12.122}$$

where $\beta_{\text{true}} \in \mathbb{R}^d$ is a vector of *true* linear coefficients, and $\tilde{z}$ is a random variable representing the noise, which is independent from $\tilde{x}$. The following theorem shows that the linear MMSE estimator of $\tilde{y}$ given $\tilde{x}$ recovers the true coefficients perfectly. The corresponding mean square error of the estimator is equal to the variance of the noise.

Theorem 12.17 (Linear MMSE estimator for linear-response model with additive noise)**.** *Let $\tilde{x}$ be a d-dimensional random vector of features, and let the corresponding response $\tilde{y}$ be defined as in (12.122), where $\tilde{z}$ has variance equal to σ^2 and is independent of $\tilde{x}$. We assume that the mean of the features and the noise is zero. Under these assumptions, the coefficients of the linear MMSE estimator of $\tilde{y}$ given $\tilde{x}$ are equal to the true linear coefficients,*

$$\beta_{\text{MMSE}} = \beta_{\text{true}}, \tag{12.123}$$

and the corresponding MSE is equal to the variance of $\tilde{z}$,

$$\text{MSE} = \sigma^2. \tag{12.124}$$

Proof Since the mean of the features is zero, the mean of the response is also zero by linearity of expectation. The cross-covariance between the features and the response therefore equals

$$\begin{aligned}
\Sigma_{\tilde{x}\tilde{y}} &= \text{E}\left[\tilde{x}\tilde{y}\right] && (12.125)\\
&= \text{E}\left[\tilde{x}\left(\tilde{x}^T\beta_{\text{true}} + \tilde{z}\right)\right] && (12.126)\\
&= \text{E}\left[\tilde{x}\tilde{x}^T\right]\beta_{\text{true}} + \text{E}\left[\tilde{x}\tilde{z}\right] && (12.127)\\
&= \Sigma_{\tilde{x}}\beta_{\text{true}}, && (12.128)
\end{aligned}$$

because $\tilde{z}$ and the entries of $\tilde{x}$ are uncorrelated and have zero mean, so

$$\text{E}\left[\tilde{x}\tilde{z}\right] = \text{E}\left[\tilde{x}\right]\text{E}\left[\tilde{z}\right] = 0. \tag{12.129}$$

Consequently, by Theorem 12.2

$$\begin{aligned}
\beta_{\text{MMSE}} &= \Sigma_{\tilde{x}}^{-1}\Sigma_{\tilde{x}\tilde{y}} && (12.130)\\
&= \Sigma_{\tilde{x}}^{-1}\Sigma_{\tilde{x}}\beta_{\text{true}} = \beta_{\text{true}}. && (12.131)
\end{aligned}$$

The linear MMSE estimator $\ell_{\text{MMSE}}(\tilde{x}) := \beta_{\text{MMSE}}^T\tilde{x}$ equals $\beta_{\text{true}}^T\tilde{x}$, so the MSE is

$$\begin{aligned}
\text{MSE} &= \text{E}\left[\tilde{y} - \beta_{\text{OLS}}^T\tilde{x}\right] && (12.132)\\
&= \text{E}\left[\left(\beta_{\text{true}}^T\tilde{x} + \tilde{z} - \beta_{\text{true}}^T\tilde{x}\right)^2\right] && (12.133)\\
&= \text{E}\left[\tilde{z}^2\right] = \sigma^2. && (12.134)
\end{aligned}$$

■

Theorem 12.17 establishes that the linear MMSE estimator recovers the true linear coefficients, as long as we have access to the true joint distribution of the features and the response (or to be more precise, to their true covariance matrix and cross-covariance). However, in practice, we perform linear regression by computing the ordinary-least-squares (OLS) estimator from finite data, as explained in Section 12.1.2. In order to study the effect of finite data on the generalization properties of linear-regression models, we introduce a data model where the features in the training set are represented by deterministic vectors.

Definition 12.18 (Finite-data linear-response model with additive noise)**.** *Given n deterministic d-dimensional feature vectors $x_1, \ldots, x_n$, and a d-dimensional true coefficient vector β_{true}, we define the training design matrix as*

$$X_{\text{train}} := \begin{bmatrix} x_1 & x_2 & \cdots & x_n \end{bmatrix}. \tag{12.135}$$

and the corresponding response as a linear function of the features perturbed by additive noise. If the noise is modeled as a deterministic n-dimensional vector z, then the training response is the deterministic vector

$$y_{\text{train}} = X_{\text{train}}^T\beta_{\text{true}} + z_{\text{train}}. \tag{12.136}$$

If the noise is modeled as an n-dimensional random vector $\tilde{z}$, then the response is the n-dimensional random vector

$$\tilde{y}_{\text{train}} = X_{\text{train}}^T \beta_{\text{true}} + \tilde{z}_{\text{train}}. \tag{12.137}$$

Our finite-data model yields an alternative interpretation for the OLS estimator introduced in Section 12.1.2. If we assume that the entries of the noise vector $\tilde{z}_{\text{train}}$ in (12.137) are i.i.d. Gaussian random variables, then the OLS estimator is the corresponding maximum-likelihood estimator of the true linear coefficients (see Exercise 12.6).

The following sections provide an analysis of the OLS estimator for the finite-data model in Definition 12.18, which reveal when it generalizes well to test data, and when it overfits the training data. Section 12.2.2 studies the OLS coefficient estimate, and Sections 12.2.3 and 12.2.4 characterize the OLS training and test error, respectively.

12.2.2 Coefficient Estimate

In this section we study the linear coefficient estimate produced by the OLS estimator. The following example illustrates the behavior of OLS coefficients computed from real data, as we vary the number of training examples.

Example 12.19 (Temperature prediction: OLS coefficients)**.** We consider the problem of estimating the temperature in Versailles (Kansas) from the temperatures of 133 other weather stations in the United States, using hourly temperature data extracted from Dataset 9. To solve the regression problem, we fit an OLS model, where the response is the temperature in Versailles and the features are the rest of the temperatures ($d := 133$).

Figure 12.8 shows the OLS coefficients for different values of the number of training data n. When n is large with respect to the number of features d, three of the coefficients are clearly larger than the rest. They correspond to stations that are in geographical proximity to Versailles: Bowling Green (Kentucky), Bedford (Indiana) and Elkins (West Virginia). These coefficients are positive, so the OLS estimator for large n is approximately equal to a weighted average of the temperatures at these locations. This seems like a very reasonable approach to estimate the temperature at Versailles. In contrast, for small n, many of the remaining coefficients have wildly-fluctuating amplitudes, and the geographically-meaningful coefficients are no longer the most prominent. As depicted in Figure 12.10 the resulting models overfit the training data and do not generalize well to held-out data.

Figure 12.8 illustrates an important phenomenon in linear regression: when the number of training data is small, OLS coefficients can be very noisy. In order to understand why, we derive a closed-form expression for the OLS coefficients, when the data are generated according to our finite-data model.

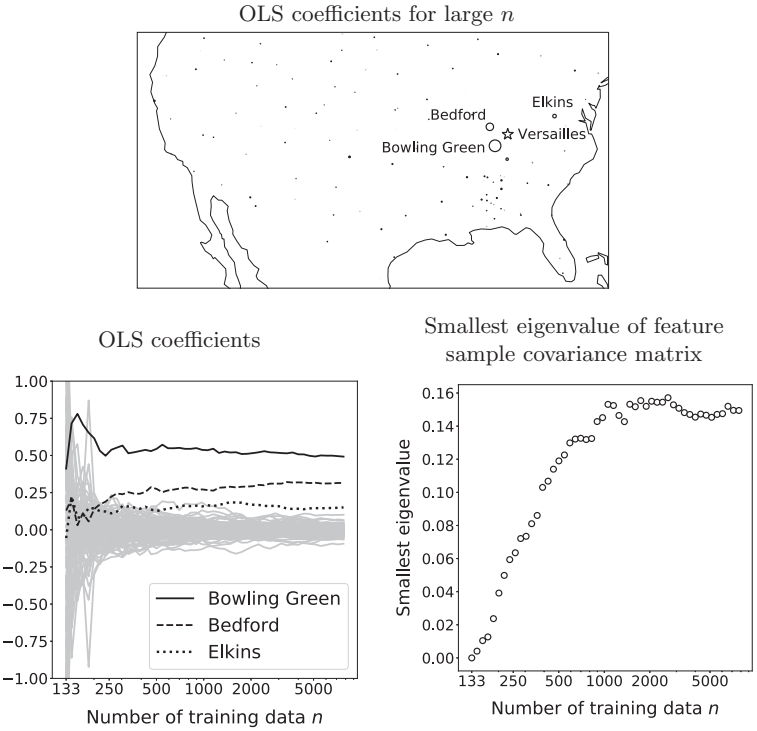


Figure 12.8 OLS coefficients for temperature estimation. The map at the top depicts the OLS coefficients in Example 12.19 when the number of training data n is large. The radius of the circular marker at each location is proportional to the corresponding OLS coefficient. The star indicates the location associated with the response. The plot on the bottom left shows the OLS coefficients for different values of n. For large n, the largest coefficients correspond to locations close to Versailles. For small n the coefficients fluctuate wildly. The plot on the bottom right shows the smallest eigenvalue of the sample covariance matrix of the features as a function of n. The wild fluctuations of the OLS coefficients occur when the smallest eigenvalue is small.

Theorem 12.20 (OLS coefficients for finite-data linear-response model with additive noise)**.** *Let the features and the response in a regression problem satisfy Definition 12.18 for a deterministic noise vector $z_{\text{train}} \in \mathbb{R}^n$. If the features and the noise have zero sample mean and the sample covariance matrix of the features Σ_X is invertible, the OLS coefficients equal*

$$\beta_{\text{OLS}} = \beta_{\text{true}} + \Sigma_X^{-1} \Sigma_{XZ}, \tag{12.138}$$

where Σ_{XZ} denotes the sample cross-covariance between the features and the noise

$$\Sigma_{XZ} := \frac{1}{n-1}\sum_{i=1}^{n} x_i z_{\text{train}}[i]. \tag{12.139}$$

Proof The sample cross-covariance between the features and the response equals

$$\Sigma_{XY} = \frac{1}{n-1}\sum_{i=1}^{n} x_i (x_i^T \beta_{\text{true}} + z_{\text{train}}[i]) \tag{12.140}$$

$$= \left(\frac{1}{n-1}\sum_{i=1}^{n} x_i x_i^T\right)\beta_{\text{true}} + \frac{1}{n-1}\sum_{i=1}^{n} x_i z_{\text{train}}[i] \tag{12.141}$$

$$= \Sigma_X \beta_{\text{true}} + \Sigma_{XZ}. \tag{12.142}$$

The result then follows from Theorem 12.7,

$$\beta_{\text{OLS}} = \Sigma_X^{-1}\Sigma_{XY} \tag{12.143}$$

$$= \beta_{\text{true}} + \Sigma_X^{-1}\Sigma_{XZ}. \tag{12.144}$$

■

The key difference between the coefficient estimates in Theorems 12.17 and 12.20 is that the cross-covariance between the features and the noise is not necessarily zero for the finite-data model. In fact, we cannot expect it to be exactly zero even if the features are samples from a random vector $\tilde{x}$ that is independent from $\tilde{z}$ (this is the same phenomenon as in Example 1.30). The following example shows that the spurious correlation between the features and the noise can be amplified dramatically by *multicollinearity* in the design matrix. Multicollinearity occurs if a subset of the features are close to being linearly dependent.

Example 12.21 (Ordinary least squares and feature multicollinearity)**.** Consider the following data model, satisfying the assumptions in Theorem 12.20,

$$\underbrace{\begin{bmatrix} 0.33 \\ 0.91 \\ -1.51 \\ -0.10 \end{bmatrix}}_{y_{\text{train}}} := \underbrace{\begin{bmatrix} 0.46 & 0.44 \\ 0.97 & 1.03 \\ -1.52 & -1.51 \\ 0.09 & 0.04 \end{bmatrix}}_{X_{\text{train}}} \underbrace{\begin{bmatrix} 0.75 \\ 0.25 \end{bmatrix}}_{\beta_{\text{true}}} + \underbrace{\begin{bmatrix} -0.13 \\ -0.08 \\ 0.01 \\ -0.18 \end{bmatrix}}_{z_{\text{train}}} \tag{12.145}$$

The features and noise are samples from independent random vectors. The sample cross-covariance between them is very small, but not exactly zero because the number of data is finite,

$$\Sigma_{XZ} = \begin{bmatrix} -0.055 \\ -0.053 \end{bmatrix}. \tag{12.146}$$

The sample covariance matrix of the features equals

$$\Sigma_X = \begin{bmatrix} 1.15 & 1.16 \\ 1.16 & 1.17 \end{bmatrix} = \underbrace{\begin{bmatrix} 0.70 & -0.71 \\ 0.71 & 0.70 \end{bmatrix}}_{U} \underbrace{\begin{bmatrix} 2.33 & 0 \\ 0 & \mathbf{9.68 \cdot 10^{-4}} \end{bmatrix}}_{\Lambda} \underbrace{\begin{bmatrix} 0.70 & 0.71 \\ -0.71 & 0.70 \end{bmatrix}}_{U^T}, \tag{12.147}$$

where $U\Lambda U^T$ is the eigendecomposition of Σ_X, which encodes the principal directions of the features and their corresponding sample variance in those directions (see Theorem 11.25). The two features are almost collinear. Consequently, the variance in the second principal direction (equal to the second eigenvalue highlighted in bold) is very small.

By Theorem 12.20, the OLS coefficients equal

$$\begin{aligned} \beta_{\text{OLS}} &= \beta_{\text{true}} + \Sigma_X^{-1}\Sigma_{XZ} && (12.148)\\ &= \beta_{\text{true}} + U\Lambda^{-1}U^T\Sigma_{XZ} && (12.149)\\ &= \begin{bmatrix} 0.75 \\ 0.25 \end{bmatrix} + \begin{bmatrix} 0.70 & -0.71 \\ 0.71 & 0.70 \end{bmatrix} \begin{bmatrix} 0.43 & 0 \\ 0 & \mathbf{1033} \end{bmatrix} \begin{bmatrix} 0.70 & 0.71 \\ -0.71 & 0.70 \end{bmatrix} \begin{bmatrix} -0.35 \\ -0.33 \end{bmatrix} && (12.150)\\ &= \begin{bmatrix} -0.71 \\ 1.65 \end{bmatrix}. && (12.151) \end{aligned}$$

The contribution of the noise is amplified by the inverse of the second eigenvalue (in bold), resulting in an estimate that is completely different from the true linear coefficients. This noise amplification allows the OLS estimator to overfit the component of the noise in the direction of least variance of the features. As a result, the resulting response estimate $y_{\text{OLS}} := X\beta_{\text{OLS}}$ is a closer approximation to the observed response y_{train} than an ideal response estimate $y_{\text{ideal}} := X\beta_{\text{true}}$ based on the true coefficients,

$$||y_{\text{OLS}} - y_{\text{train}}||_2^2 = 0.036 < 0.047 = ||y_{\text{ideal}} - y_{\text{train}}||_2^2. \tag{12.152}$$

In words, the linear model overfits the training data.

Theorem 12.20 and Example 12.21 shed light onto the behavior of the OLS coefficients in Figure 12.8. As the number of training data n decreases, so does the smallest eigenvalue in the sample covariance matrix of the features (bottom right), indicating that there is multicollinearity among the features. This results in large noisy fluctuations of the OLS coefficients (bottom left), which enable the OLS model to overfit the training data.

In order to gain further insight into the typical behavior of OLS coefficients, we study their distribution for our finite-data linear-response model when the additive noise is random. Figure 12.9 shows that the distribution of the resulting OLS coefficients is centered at the true coefficients. This is no coincidence: if

the noise is independent and zero mean, then the OLS coefficient estimate is guaranteed to be unbiased.

Theorem 12.22 (OLS coefficient estimate is unbiased)**.** *Let $x_1, \ldots, x_n$ be d-dimensional feature vectors with an invertible sample covariance matrix, and let $\beta_{\text{true}} \in \mathbb{R}^d$ be a fixed vector of linear coefficients. If the response $\tilde{y}_{\text{train}} \in \mathbb{R}^n$ is defined as in (12.137) for a noise vector $\tilde{z}_{\text{train}}$ with n independent entries that have zero mean, then OLS yields an unbiased estimate of the true linear coefficients,*

$$\mathrm{E}\left[\tilde{\beta}_{\text{OLS}}\right] = \beta_{\text{true}}. \tag{12.153}$$

Proof Under the assumptions in Definition 12.18, the sample cross-covariance between the features and the noise is a random vector that equals

$$\widetilde{\Sigma}_{XZ} := \frac{1}{n-1}\sum_{i=1}^{n} x_i \tilde{z}_{\text{train}}[i]. \tag{12.154}$$

By Theorem 12.20,

$$\tilde{\beta}_{\text{OLS}} = \beta_{\text{true}} + \Sigma_X^{-1}\widetilde{\Sigma}_{XZ}, \tag{12.155}$$

so by linearity of expectation and the assumption that each noise entry has zero mean,

$$\begin{aligned}
\mathrm{E}\left[\tilde{\beta}_{\text{OLS}}\right] &= \beta_{\text{true}} + \Sigma_X^{-1}\mathrm{E}\left[\widetilde{\Sigma}_{XZ}\right] && (12.156)\\
&= \beta_{\text{true}} + \Sigma_X^{-1}\frac{1}{n-1}\sum_{i=1}^{n} x_i \mathrm{E}\left[\tilde{z}_{\text{train}}[i]\right] && (12.157)\\
&= \beta_{\text{true}}. && (12.158)
\end{aligned}$$

■

By Theorem 12.22, the OLS coefficients are centered at the true coefficients, but this does not necessarily imply that they are *close* to them. The following theorem derives the covariance matrix of the OLS coefficients, which quantifies to what extent they fluctuate around the true coefficients.

Theorem 12.23 (Covariance matrix of OLS coefficients)**.** *Let $x_1, \ldots, x_n$ be d-dimensional feature vectors, and let $\beta_{\text{true}} \in \mathbb{R}^d$ be a fixed vector of linear coefficients. If the response $\tilde{y}_{\text{train}} \in \mathbb{R}^n$ is defined as in (12.137) for a noise vector $\tilde{z}_{\text{train}}$ with n independent entries that have zero mean and variance σ^2, then the covariance matrix of the OLS coefficients equals*

$$\Sigma_{\tilde{\beta}_{\text{OLS}}} = \frac{\sigma^2}{n-1}\Sigma_X^{-1}, \tag{12.159}$$

where Σ_X is the sample covariance matrix of the features, which we assume to be invertible.

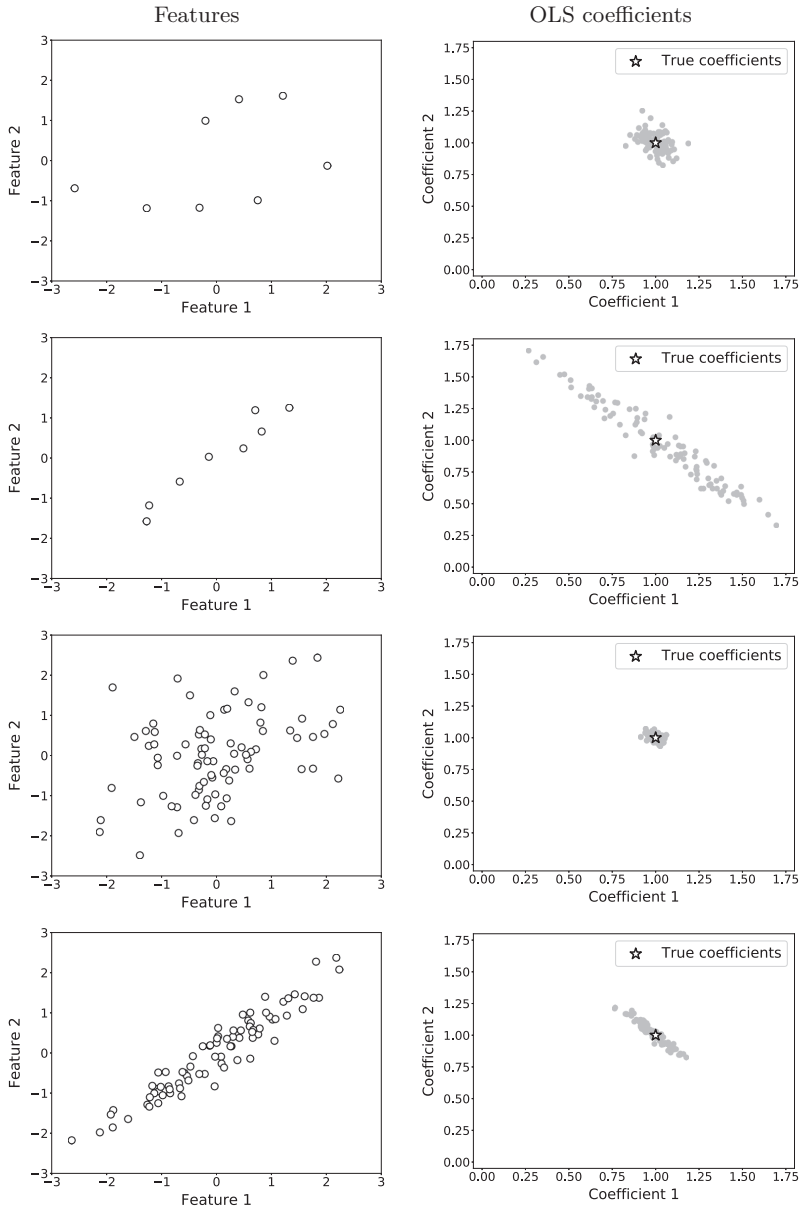


Figure 12.9 OLS coefficients for finite-data linear-response model with additive noise. The left column shows the two-dimensional feature vectors corresponding to $n := 8$ (top two rows) and $n := 80$ (bottom two rows) training examples. The right column shows 100 OLS coefficient estimates corresponding to 100 different realizations of the model in Definition 12.18, where the true linear coefficients (represented by a star) are fixed and the noise is sampled from a fixed-variance i.i.d. Gaussian distribution. As established in Theorem 12.22, the estimates are unbiased: their distribution is centered at the true coefficients. As established in Theorem 12.23, the variance of the OLS coefficients decreases if we increase the number of data, and is highest in the direction of lowest variance of the features.

Proof By Theorems 12.20 and 12.22, the centered OLS coefficients equal

$$\operatorname{ct}\left(\tilde{\beta}_{\mathrm{OLS}}\right) := \tilde{\beta}_{\mathrm{OLS}} - \mathrm{E}\left[\tilde{\beta}_{\mathrm{OLS}}\right] \tag{12.160}$$

$$= \Sigma_X^{-1}\widetilde{\Sigma}_{XZ}, \tag{12.161}$$

where the sample cross-covariance is defined as in (12.154). Since the noise entries are uncorrelated and have zero mean, the covariance matrix of the sample cross-covariance equals

$$\mathrm{E}\left[\widetilde{\Sigma}_{XZ}\widetilde{\Sigma}_{XZ}^T\right] = \mathrm{E}\left[\frac{1}{n-1}\sum_{i=1}^{n} x_i\tilde{z}_{\mathrm{train}}[i]\frac{1}{n-1}\sum_{j=1}^{n}\tilde{z}_{\mathrm{train}}[j]x_j^T\right] \tag{12.162}$$

$$= \frac{1}{(n-1)^2}\sum_{i=1}^{n}\sum_{j=1}^{n} x_i\mathrm{E}\left[\tilde{z}_{\mathrm{train}}[i]\tilde{z}_{\mathrm{train}}[j]\right]x_j^T \tag{12.163}$$

$$= \frac{\sigma^2}{(n-1)^2}\sum_{i=1}^{n} x_i x_i^T \tag{12.164}$$

$$= \frac{\sigma^2}{n-1}\Sigma_X, \tag{12.165}$$

because $\mathrm{E}\left[\tilde{z}_{\mathrm{train}}[i]\tilde{z}_{\mathrm{train}}[j]\right]$ equals σ^2, if $i = j$, and zero otherwise. Consequently, by linearity of expectation, the covariance matrix of the OLS coefficients is

$$\Sigma_{\tilde{\beta}_{\mathrm{OLS}}} = \mathrm{E}\left[\operatorname{ct}\left(\tilde{\beta}_{\mathrm{OLS}}\right)\operatorname{ct}\left(\tilde{\beta}_{\mathrm{OLS}}\right)^T\right] \tag{12.166}$$

$$= \mathrm{E}\left[\Sigma_X^{-1}\widetilde{\Sigma}_{XZ}\widetilde{\Sigma}_{XZ}^T\Sigma_X^{-1}\right] \tag{12.167}$$

$$= \Sigma_X^{-1}\mathrm{E}\left[\widetilde{\Sigma}_{XZ}\widetilde{\Sigma}_{XZ}^T\right]\Sigma_X^{-1} \tag{12.168}$$

$$= \frac{\sigma^2}{n-1}\Sigma_X^{-1}\Sigma_X\Sigma_X^{-1} \tag{12.169}$$

$$= \frac{\sigma^2}{n-1}\Sigma_X^{-1}. \tag{12.170}$$

■

By Theorem 11.11, the variance of the coefficients in the direction of a normalized vector a is equal to $a^T\Sigma_{\tilde{\beta}_{\mathrm{OLS}}}a$. Consequently, Theorem 12.23 implies that the variance of the coefficients in every direction is proportional to the noise variance σ^2 and inversely proportional to the number of data n, as we observe in Figure 12.9. As $n \to \infty$, the variance tends to zero, which means that the OLS coefficients converge to the true coefficients.

Theorem 12.23 reveals the effect of the covariance structure of the features on the OLS estimate. By Theorem 11.20, the eigendecomposition of the sample covariance matrix of the features $\Sigma_X = U\Lambda U^T$ encodes the principal directions of the data (columns of U) and their corresponding sample variance in those directions (diagonal entries of Λ). The eigendecomposition of the covariance matrix of

the OLS coefficients equals

$$\Sigma_{\tilde{\beta}_{\text{OLS}}} = \frac{\sigma^2}{n-1}\Sigma_X^{-1} \tag{12.171}$$

$$= \frac{\sigma^2}{n-1}U\Lambda^{-1}U^T. \tag{12.172}$$

By Theorem 11.20, the principal directions of the coefficients are equal to the columns of U, and hence are the same as the principal directions of the features. The sample variance in each direction is equal to the corresponding eigenvalue, stored in the respective diagonal entry of Λ^{-1}, and is inversely proportional to the sample variance of the features in that direction. When there is multicollinearity in the feature design matrix, some of the eigenvalues in Λ are small, so in the corresponding directions, the features have low variance and the OLS coefficients have high variance. This is what we observe in Figure 12.9 (compare the first and second row, or the third and fourth row).

12.2.3 Training Error

The training error of a model evaluates how closely it approximates the data used to fit it. In the case of regression, it is common to quantify the error using the sum of squares. Given a dataset formed by n feature vectors $x_1, \ldots, x_n$ with d features and a corresponding response vector $y_{\text{train}} \in \mathbb{R}^n$, we define the training error of the OLS estimator as

$$\text{Training error} := \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(y_{\text{train}}[i] - \ell_{\text{OLS}}(x_i)\right)^2}, \tag{12.173}$$

where $\ell_{\text{OLS}}(x_i) := \beta_{\text{OLS}}^T x_i + \alpha_{\text{OLS}}$. We normalize the sum of squares, and apply the square root, so that we can express the error in the same units as the response. Figure 12.10 shows the OLS training error for the weather data in Example 12.19. When the number of training data n is close to the number of features d, the training error is very small, which indicates overfitting. As n increases, so does the training error, converging to a fixed value. The goal of this section is to provide a theoretical explanation of this behavior.

In order to analyze the training error, we consider a regression problem with d features and n training examples, where the features and responses are centered to have zero sample mean. Any linear estimate of the response given the features

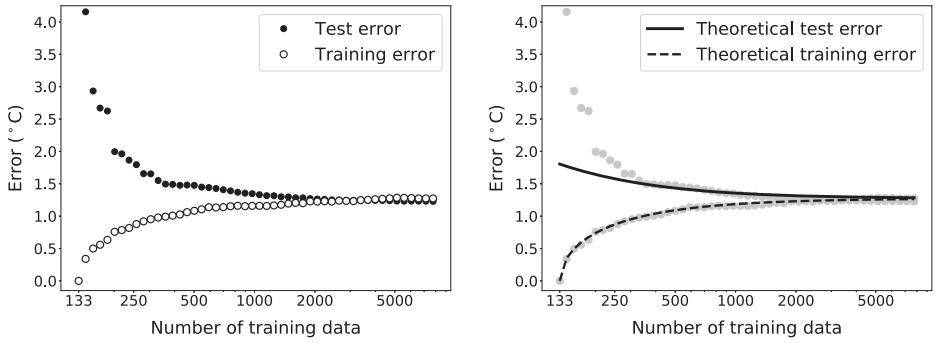


Figure 12.10 OLS training and test error for weather data. The left plot shows the performance of OLS models for the temperature-estimation problem described in Example 12.19 as a function of the number of training data. The test data is fixed and consists of 1,000 held-out data points from the same year (2015). The error metric is the square root of the average squared error, defined as in (12.173), which quantifies the average error in temperature prediction. The right plot shows the theoretical expressions for the training and test error derived in Theorems 12.26 and 12.27, superposed onto the observed errors (gray markers). The variance of the noise σ in the theoretical expressions is set equal to the training RMSE for large n.

can be expressed in matrix-vector form:

$$X_{\text{train}}^T \beta = \begin{bmatrix} x_1[1] & x_1[2] & \cdots & x_1[d] \\ x_2[1] & x_2[2] & \cdots & x_2[d] \\ \cdots & \cdots & \cdots & \cdots \\ x_n[1] & x_n[2] & \cdots & x_n[d] \end{bmatrix} \beta \tag{12.174}$$

$$= \begin{bmatrix} X_1 & X_2 & \cdots & X_d \end{bmatrix} \beta \tag{12.175}$$

$$= \sum_{j=1}^{d} X_j \beta[j], \tag{12.176}$$

where $\beta \in \mathbb{R}^d$ is a vector of coefficients, and $x_i[j]$ denotes the jth feature in the ith example. The jth row X_j of the training design matrix X_{train} is a vector containing the n examples of the jth feature. The set of all possible linear estimates is equal to the row space $\text{row}(X_{\text{train}})$ of X_{train}, because it is precisely the subspace or hyperplane spanned by the rows of X_{train}.

The training error of each linear estimate is proportional to the Euclidean distance between the estimate $X_{\text{train}}^T \beta$ and the response vector y_{train},

$$\text{Training error} := \sqrt{\frac{1}{n} \sum_{i=1}^{n} \left(y_{\text{train}}[i] - \sum_{j=1}^{d} x_i[j] \beta[j] \right)^2} = \frac{1}{\sqrt{n}} \left|\left| y_{\text{train}} - X_{\text{train}}^T \beta \right|\right|_2 .$$

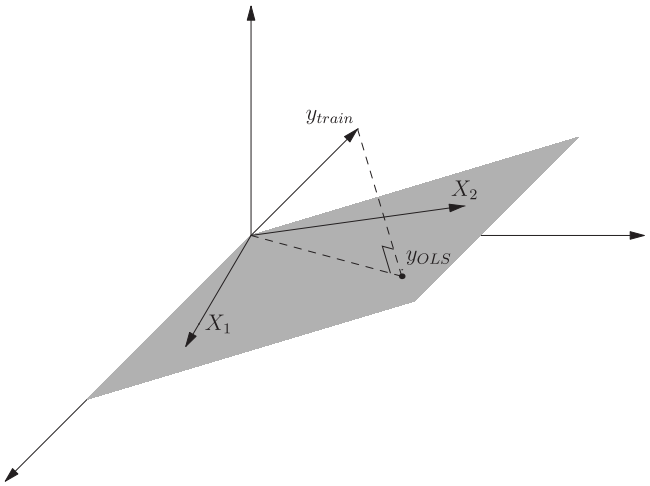


Figure 12.11 OLS response estimate as a projection. Illustration of Theorem 12.24 for a regression problem with $n := 3$ examples and $d := 2$ features. The OLS response estimate is the orthogonal projection of the three-dimensional training response y_{train} onto the two-dimensional subspace spanned by the rows X_1 and X_2 of the design matrix X_{train}.

By Theorem 12.7 the OLS estimator minimizes the training error. Therefore, the OLS response estimate is the vector in $\text{row}(X_{\text{train}})$ that is *closest* to y_{train}. Geometrically, it is the orthogonal projection of y_{train} onto $\text{row}(X_{\text{train}})$, as illustrated by Figure 12.11. The following theorem formalizes this reasoning, which is analogous to our derivation of the linear MMSE estimator in Section 12.1.1. The only difference is that here the features and the response are represented by n-dimensional vectors, instead of random variables (compare Figures 12.2 and 12.11).

Theorem 12.24 (The OLS estimator is a projection)**.** *Let $X_{\text{train}} \in \mathbb{R}^{d \times n}$ be a design matrix containing n d-dimensional vectors corresponding to the features in a regression problem, and let $y_{\text{train}} \in \mathbb{R}^n$ be the response. Assuming that the features and the response are centered to have zero sample mean and the sample covariance matrix of the features is invertible, the OLS response estimate is*

$$y_{\text{OLS}} := X_{\text{train}}^T \beta_{\text{OLS}} \tag{12.177}$$

$$= \mathcal{P}_{\text{row}(X_{\text{train}})}\, y_{\text{train}}, \tag{12.178}$$

where $\text{row}(X_{\text{train}})$ is the row space of X_{train}.

Proof Let USV^T be the singular-value decomposition of X_{train}. The sample

cross-covariance between the features and the response equals

$$\Sigma_{XY} = \frac{1}{n-1} X_{\text{train}} y_{\text{train}} \tag{12.179}$$

$$= \frac{1}{n-1} USV^T y_{\text{train}}. \tag{12.180}$$

In terms of the SVD, the covariance matrix of the features equals

$$\Sigma_X = \frac{1}{n-1} X_{\text{train}} X_{\text{train}}^T \tag{12.181}$$

$$= \frac{1}{n-1} USV^T VSU^T \tag{12.182}$$

$$= \frac{1}{n-1} USSU^T, \tag{12.183}$$

so the inverse equals

$$\Sigma_X^{-1} = (n-1) US^{-1} S^{-1} U^T. \tag{12.184}$$

Consequently, by Theorem 12.7 the OLS response estimate equals

$$y_{\text{OLS}} := X_{\text{train}}^T \beta_{\text{OLS}} \tag{12.185}$$

$$= X_{\text{train}}^T \Sigma_X^{-1} \Sigma_{XY} \tag{12.186}$$

$$= VSU^T US^{-1} S^{-1} U^T USV^T y_{\text{train}} \tag{12.187}$$

$$= VV^T y_{\text{train}}, \tag{12.188}$$

because $U^T U = I$. By Theorem 11.31, V is a matrix with d orthonormal rows that span the row space of X_{train}, so VV^T is a projection matrix onto this subspace. ■

Theorem 12.24 provides an intuitive characterization of the training residual of the OLS response estimate for our finite-data linear-response model with additive noise: it is the projection of the noise component onto the orthogonal complement of the row space of the feature design matrix.

Theorem 12.25 (Training residual for finite-data linear-response model with additive noise)**.** *Let the feature design matrix $X_{\text{train}} \in \mathbb{R}^{d \times n}$ and the response $y_{\text{train}} \in \mathbb{R}^n$ in a regression problem satisfy Definition 12.18 for a deterministic noise vector $z_{\text{train}} \in \mathbb{R}^n$. Assuming that the features and the response are centered to have zero sample mean and the sample covariance matrix of the features is invertible, the training residual of the OLS coefficient estimate $\beta_{\text{OLS}} \in \mathbb{R}^d$ is equal to the projection of the noise onto the orthogonal complement of the row space of X_{train},*

$$y_{\text{train}} - X_{\text{train}}^T \beta_{\text{OLS}} = \mathcal{P}_{\text{row}(X_{\text{train}})^\perp} z_{\text{train}}. \tag{12.189}$$

Proof By Theorem 12.24

$$\begin{aligned} y_{\text{train}} - X_{\text{train}}^T \tilde{\beta}_{\text{OLS}} &= y_{\text{train}} - \mathcal{P}_{\text{row}(X_{\text{train}})}\, y_{\text{train}} && (12.190)\\ &= \mathcal{P}_{\text{row}(X_{\text{train}})^\perp}\, y_{\text{train}} && (12.191)\\ &= \mathcal{P}_{\text{row}(X_{\text{train}})^\perp} \left(X^T \beta_{\text{true}} + z_{\text{train}} \right) && (12.192)\\ &= \mathcal{P}_{\text{row}(X_{\text{train}})^\perp}\, z_{\text{train}}, && (12.193) \end{aligned}$$

because $X^T \beta_{\text{true}}$ belongs to the row space of X_{train} and, hence, is orthogonal to its orthogonal complement. ■

Theorem 12.25 enables us to characterize the training error of the OLS estimator for our finite-data model in terms of the number of features d and the number of examples n. The feature design matrix has n rows of dimension d. Assuming these rows are linearly independent, the row space is a subspace of dimension d, so its orthogonal complement has dimension $n - d$. If the noise component is *isotropic*, meaning that it has the same variance in every direction, then projecting it onto the orthogonal complement should (on average) preserve a fraction of variance proportional to the dimension $n - d$ of the orthogonal complement. The following theorem makes this intuition precise, showing that our back-of-the-envelope calculation is correct.

Theorem 12.26 (Training error of the OLS estimator for finite-data linear-response model with additive noise)**.** *Let $x_1, \ldots, x_n$ be d-dimensional feature vectors with an invertible sample covariance matrix, and let $\beta_{\text{true}} \in \mathbb{R}^d$ be a fixed vector of linear coefficients. We assume that the response $\tilde{y}_{\text{train}}$ is defined as in (12.137) for a noise vector $\tilde{z}_{\text{train}}$ with independent entries that have zero mean and variance σ^2, and define the residual sum of squares as the random variable,*

$$\widetilde{\text{RSS}}_{\text{train}} := \sum_{i=1}^{n} \left(\tilde{y}_{\text{train}}[i] - x_i^T \tilde{\beta}_{\text{OLS}} \right)^2. \tag{12.194}$$

The mean of the residual sum of squares per example equals

$$\text{E}\left[\frac{1}{n} \widetilde{\text{RSS}}_{\text{train}} \right] = \left(1 - \frac{d}{n} \right) \sigma^2. \tag{12.195}$$

Proof Let $V_\perp$ be a matrix with $n - d$ orthonormal columns $v_{\perp i}$, $1 \leq i \leq n - d$, spanning the orthogonal complement of $\text{row}(X_{\text{train}})$, so that $V_\perp^T V_\perp = I$. By

Theorem 12.25,

$$\widetilde{\text{RSS}} = \left|\left|\mathcal{P}_{\text{row}(X_{\text{train}})^\perp}\, \tilde{z}_{\text{train}}\right|\right|_2^2 \tag{12.196}$$

$$= \left|\left|V_\perp V_\perp^T \tilde{z}_{\text{train}}\right|\right|_2^2 \tag{12.197}$$

$$= \tilde{z}_{\text{train}}^T V_\perp V_\perp^T V_\perp V_\perp^T \tilde{z}_{\text{train}} \tag{12.198}$$

$$= \tilde{z}_{\text{train}}^T V_\perp V_\perp^T \tilde{z}_{\text{train}} \tag{12.199}$$

$$= \sum_{i=1}^{n-d} \left(v_{\perp i}^T \tilde{z}_{\text{train}}\right)^2 \tag{12.200}$$

$$= \sum_{i=1}^{n-d} v_{\perp i}^T \tilde{z}_{\text{train}} \tilde{z}_{\text{train}}^T v_{\perp i}. \tag{12.201}$$

Since the entries of $\tilde{z}_{\text{train}}$ are uncorrelated and have variance σ^2, its covariance matrix equals $\Sigma_{\tilde{z}_{\text{train}}} = \sigma^2 I$. Consequently, by linearity of expectation

$$\text{E}\left[\widetilde{\text{RSS}}\right] = \sum_{i=1}^{n-d} v_{\perp i}^T \text{E}\left[\tilde{z}_{\text{train}} \tilde{z}_{\text{train}}^T\right] v_{\perp i} \tag{12.202}$$

$$= \sum_{i=1}^{n-d} v_{\perp i}^T \Sigma_{\tilde{z}_{\text{train}}} v_{\perp i} \tag{12.203}$$

$$= \sigma^2 \sum_{i=1}^{n-d} v_{\perp i}^T v_{\perp i} \tag{12.204}$$

$$= (n-d)\sigma^2. \tag{12.205}$$

■

Figure 12.10 shows that Theorem 12.26 provides a very precise characterization of the dependence between the training error and the number of training data for the real-world dataset in Example 12.19. The training error is proportional to $\sqrt{1 - d/n}$, as predicted by the theorem.

The residual sum of squares per example in Theorem 12.26 is an approximation of the training MSE. Under the assumptions of the theorem, when $n \gg d$, it converges to the variance of the noise σ^2. Thus, when the number of examples is large with respect to the number of features, the OLS estimator behaves like the linear MMSE estimator, which has an MSE equal to σ^2 when the response is linear with additive independent noise (see Theorem 12.17). However, when $n \approx d$, the training error of the OLS estimator approaches zero. Since the features are independent from the noisy component, this can only be achieved by approximating the noise. We conclude that the OLS model overfits the noise in the training data, when the number of data is close to the number of data.

12.2.4 Test Error

The test error of an estimator quantifies its performance on held-out data, which have not been used to fit the model. To study the test error of the OLS estimator, we consider a training set, where the response follows the linear finite-data model with additive noise in Definition 12.18,

$$\tilde{y}_{\text{train}} = X_{\text{train}}^T \beta_{\text{true}} + \tilde{z}_{\text{train}}. \tag{12.206}$$

The estimator is evaluated on a test example, where the features are represented by a d-dimensional random vector $\tilde{x}_{\text{test}}$, and the response follows a linear model with the same coefficients as the training data,

$$\tilde{y}_{\text{test}} := \tilde{x}_{\text{test}}^T \beta_{\text{true}} + \tilde{z}_{\text{test}}. \tag{12.207}$$

The linear coefficients are shared by the training and test set, but the features and noise are not. The OLS response estimate for the test data $\tilde{x}_{\text{test}}^T \tilde{\beta}_{\text{OLS}}$ is computed using the OLS coefficient estimator $\tilde{\beta}_{\text{OLS}}$, which *only depends on the training data*. Under our assumptions, by Theorem 12.22 $\mathrm{E}[\tilde{\beta}_{\text{OLS}}] = \beta_{\text{true}}$, so the residual of the estimator on the test example equals

$$\begin{aligned} \tilde{y}_{\text{test}} - \tilde{x}_{\text{test}}^T \tilde{\beta}_{\text{OLS}} &= \tilde{x}_{\text{test}}^T \beta_{\text{true}} + \tilde{z}_{\text{test}} - \tilde{x}_{\text{test}}^T \beta_{\text{true}} - \tilde{x}_{\text{test}}^T \operatorname{ct}(\tilde{\beta}_{\text{OLS}}) \\ &= \tilde{z}_{\text{test}} - \tilde{x}_{\text{test}}^T \operatorname{ct}(\tilde{\beta}_{\text{OLS}}), \qquad \operatorname{ct}(\tilde{\beta}_{\text{OLS}}) := \tilde{\beta}_{\text{OLS}} - \mathrm{E}[\tilde{\beta}_{\text{OLS}}]. \end{aligned} \tag{12.208}$$

If the test noise $\tilde{z}_{\text{test}}$ is zero mean and independent from the other variables, then the test MSE is equal to the sum of the variance of $\tilde{z}_{\text{test}}$ and the variance of $\tilde{x}_{\text{test}}^T \operatorname{ct}(\tilde{\beta}_{\text{OLS}})$. This second term is crucial to understand the behavior of the test MSE. It equals the mean squared inner product between the test feature vector and the centered OLS coefficients. By Theorem 12.23, the OLS coefficients may have high variance if there is multicollinearity between the features. However, this does *not* necessarily translate to a high test error, as long as the test features have the same covariance matrix as the training features. In that case, the test features have low variance precisely in the directions in which the coefficients have high variance, which neutralizes the coefficient error.

Theorem 12.27 (Test error of the OLS estimator for linear data model with additive noise)**.** *Let $x_1, \ldots, x_n$ be d-dimensional feature vectors with an invertible sample covariance matrix Σ_X, and let $\beta_{\text{true}} \in \mathbb{R}^d$ be a fixed vector of linear coefficients. We assume that the response $\tilde{y}_{\text{train}}$ is defined as in (12.137) for a noise vector $\tilde{z}_{\text{train}}$ with independent entries that have zero mean and variance σ^2.*

We consider a d-dimensional test feature vector $\tilde{x}_{\text{test}}$ and a corresponding test response $\tilde{y}_{\text{test}}$, defined as in (12.207), where the test noise has zero mean and variance σ^2. The test features $\tilde{x}_{\text{test}}$ and the training and test noise $\tilde{z}_{\text{train}}$ and $\tilde{z}_{\text{test}}$ are all independent. If $\Sigma_{\tilde{x}_{\text{test}}} = \Sigma_X$, then the mean squared test error of the OLS estimator equals

$$\mathrm{E}\left[\left(\tilde{y}_{\text{test}} - \tilde{x}_{\text{test}}^T \tilde{\beta}_{\text{OLS}}\right)^2\right] = \sigma^2 \left(1 + \frac{d}{n-1}\right), \tag{12.209}$$

where $\tilde{\beta}_{\text{OLS}}$ denotes the OLS coefficient estimate obtained using the training data.

Proof By (12.208) and Theorem 9.11, under the assumption that $\tilde{z}_{\text{test}}$ has zero mean and is independent from $\tilde{x}_{\text{test}}$ and $\tilde{\beta}_{\text{OLS}}$ (because it is independent from $\tilde{z}_{\text{train}}$),

$$\mathrm{E}\left[\left(\tilde{y}_{\text{test}} - \tilde{x}_{\text{test}}^T \tilde{\beta}_{\text{OLS}}\right)^2\right] = \mathrm{Var}\left[\tilde{z}_{\text{test}} - \tilde{x}_{\text{test}}^T \,\mathrm{ct}(\tilde{\beta}_{\text{OLS}})\right] \tag{12.210}$$

$$= \mathrm{Var}\left[\tilde{z}_{\text{test}}\right] + \mathrm{E}\left[\left(\tilde{x}_{\text{test}}^T \,\mathrm{ct}(\tilde{\beta}_{\text{OLS}})\right)^2\right]. \tag{12.211}$$

In order to analyze the second term, we use the fact that for any matrices $A \in \mathbb{R}^{n_1 \times n_2}$ and $B \in \mathbb{R}^{n_1 \times n_2}$, $\mathrm{Trace}\,(A^T B) = \mathrm{Trace}\,(B A^T)$, and also that for any random matrix $\widetilde{M}$, $\mathrm{E}[\mathrm{Trace}(\widetilde{M})] = \mathrm{Trace}(\mathrm{E}[\widetilde{M}])$, by linearity of expectation. This allows us to decouple $\tilde{x}_{\text{test}}$ and $\mathrm{ct}(\tilde{\beta}_{\text{OLS}})$, and exploit that they are independent (by independence of $\tilde{x}_{\text{test}}$ and $\tilde{z}_{\text{train}}$):

$$\mathrm{E}\left[\left(\tilde{x}_{\text{test}}^T \,\mathrm{ct}(\tilde{\beta}_{\text{OLS}})\right)^2\right] = \mathrm{E}\left[\mathrm{Trace}\left(\tilde{x}_{\text{test}}^T \,\mathrm{ct}(\tilde{\beta}_{\text{OLS}})\,\mathrm{ct}(\tilde{\beta}_{\text{OLS}})^T \tilde{x}_{\text{test}}\right)\right] \tag{12.212}$$

$$= \mathrm{E}\left[\mathrm{Trace}\left(\tilde{x}_{\text{test}} \tilde{x}_{\text{test}}^T \,\mathrm{ct}(\tilde{\beta}_{\text{OLS}})\,\mathrm{ct}(\tilde{\beta}_{\text{OLS}})^T\right)\right] \tag{12.213}$$

$$= \mathrm{Trace}\left(\mathrm{E}\left[\tilde{x}_{\text{test}} \tilde{x}_{\text{test}}^T \,\mathrm{ct}(\tilde{\beta}_{\text{OLS}})\,\mathrm{ct}(\tilde{\beta}_{\text{OLS}})^T\right]\right) \tag{12.214}$$

$$= \mathrm{Trace}\left(\mathrm{E}\left[\tilde{x}_{\text{test}} \tilde{x}_{\text{test}}^T\right] \mathrm{E}\left[\mathrm{ct}(\tilde{\beta}_{\text{OLS}})\,\mathrm{ct}(\tilde{\beta}_{\text{OLS}})^T\right]\right) \tag{12.215}$$

$$= \mathrm{Trace}\left(\Sigma_{\tilde{x}_{\text{test}}} \Sigma_{\tilde{\beta}_{\text{OLS}}}\right) \tag{12.216}$$

$$= \mathrm{Trace}\left(\Sigma_X \frac{\sigma^2}{n-1} \Sigma_X^{-1}\right) \tag{12.217}$$

$$= \frac{\sigma^2}{n-1} \mathrm{Trace}\,(I) \tag{12.218}$$

$$= \frac{\sigma^2 d}{n-1}. \tag{12.219}$$

Since $\mathrm{Var}\left[\tilde{z}_{\text{test}}\right] = \sigma^2$, we conclude that

$$\mathrm{E}\left[\left(\tilde{y}_{\text{test}} - \tilde{x}_{\text{test}}^T \tilde{\beta}_{\text{OLS}}\right)^2\right] = \sigma^2 + \frac{\sigma^2 d}{n-1}. \tag{12.220}$$

■

For a test dataset formed by n_{test} feature vectors $x_1, \ldots, x_{n_{\text{test}}}$ and a response vector $y_{\text{test}} \in \mathbb{R}^{n_{\text{test}}}$, we set

$$\text{Test error} := \sqrt{\frac{1}{n_{\text{test}}} \sum_{i=1}^{n} \left(y_{\text{test}}[i] - \ell_{\text{OLS}}(x_i)\right)^2}, \tag{12.221}$$

which is a finite-sample approximation to the test MSE. Consequently, if the data are sampled according to the assumptions of Theorem 12.27, the error should scale proportionally to $\sqrt{1 + \frac{d}{n-1}}$. This is consistent with the behavior of the test error

in Figure 12.10, when the number of training data n is large with respect to the number of features d.

When n is close to d, the test error in Figure 12.10 is much larger than the theoretical prediction in Theorem 12.27. This is not surprising: we cannot expect Theorem 12.27 to provide an accurate approximation for small n, even if the response is sampled from the linear-response model in (12.137). The theorem assumes that the sample covariance matrix of the training data is equal to the true covariance matrix of the test data. For large n this is approximately true, but for small n it is not. When the assumption fails, the directions of high variance of the OLS coefficients no longer coincide with directions of low variance in the test data, which translates into a larger test error. In this regime, the OLS estimator overfits the training data (as established in Theorem 12.25) generalizing very poorly on held-out test data.

For large n, Theorem 12.27 shows that the test MSE converges to the noise variance σ^2. In that regime, the test error matches the training error (see Theorem 12.26 and also Figure 12.10) and the error of the linear MMSE estimator in Theorem 12.17. Consequently, the OLS estimator generalizes robustly to test data. This justifies our approach in Examples 12.9 and 12.16, where we evaluate the coefficient of determination using the same data used to train the OLS models. In both cases, the number of training data is large and there are only two features, so there is no overfitting.

12.3 Ridge Regression

As described in Section 12.2.2, the OLS coefficient estimator is prone to suffer from noise amplification, when there is multicollinearity among the features. In such cases, the OLS model overfits the training set and generalizes poorly to held-out test data, as explained in Sections 12.2.3 and 12.2.4. Ridge regression addresses this issue by adding a *regularization* term to the OLS cost function, which penalizes large coefficients.

Definition 12.28 (Ridge regression)**.** *Let (x_1, y_1), (x_2, y_2), $\ldots$, (x_n, y_n) be a dataset formed by n pairs of a response y_i and a corresponding vector x_i containing d features $(1 \leq i \leq n)$. We assume that the features and the response so that their sample mean is zero. The ridge-regression estimator of the response given the features is*

$$\ell_{\mathrm{RR}}(x_i) := \beta_{\mathrm{RR}}^T x_i, \quad 1 \leq i \leq n, \tag{12.222}$$

where the linear coefficients minimize the regularized least-squares cost function,

$$\beta_{\mathrm{RR}} = \arg\min_{\beta \in \mathbb{R}} \sum_{i=1}^{n} \left(y_i - \beta^T x_i\right)^2 + \lambda \sum_{j=1}^{d} \beta[j]^2. \tag{12.223}$$

and λ is a nonnegative regularization parameter.

The regularization parameter λ governs the trade-off between the data-fidelity

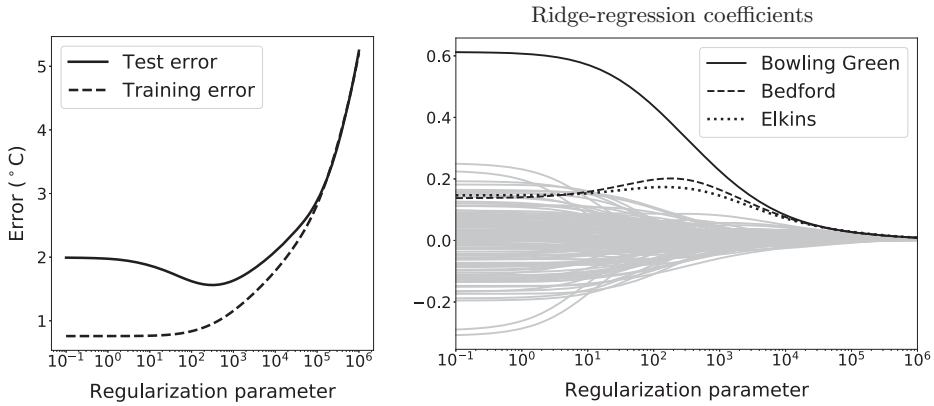


Figure 12.12 Effect of ridge-regression regularization. The left graph shows the training and test errors of the ridge-regression estimator applied to the temperature-estimation problem in Example 12.19 for different values of the regularization parameter λ. The number of training data is fixed to $n := 200$. The right figure shows the ridge-regression coefficients as a function of λ. All coefficients are depicted in gray except the three that have the largest magnitudes for large n, which correspond to locations close to Versailles (see Figure 12.8).

term and the regularization term in the ridge-regression cost function. When λ is small, the data-fidelity term dominates, so the ridge-regression estimator is close to the OLS estimator. When λ is large, the regularization term dominates, resulting in very small coefficients. This is illustrated in Figure 12.12, which shows an application of ridge regression to the weather regression problem from Example 12.19 in a regime where the OLS estimator overfits the training data (small n). For small λ, the ridge-regression estimator overfits just like the OLS estimator: the training error is very small, the test error is substantially larger, and many of the coefficients have large amplitudes, which drowns out the geographically-meaningful coefficients associated with Bedford and Elkins (see Figure 12.8). As we increase λ, the magnitudes of the noisy coefficients decrease, the meaningful coefficients rise above the rest, and the test error improves. However, if we increase λ too much, the coefficients eventually become very small, causing the model to underfit the training data, and the test error to deteriorate.

A key challenge is how to select the parameter λ in order to strike the right balance between data fidelity and regularization. Our ultimate goal is to minimize the test error, but we cannot use the test data to choose λ. The test set should be used *only* for evaluation. Instead, we separate the training data into a training set and a held-out *validation* set. The training set is used to fit ridge-regression models with multiple values of λ (typically on a logarithmic grid). The validation set is used to evaluate these models and find the best λ value. This approach can

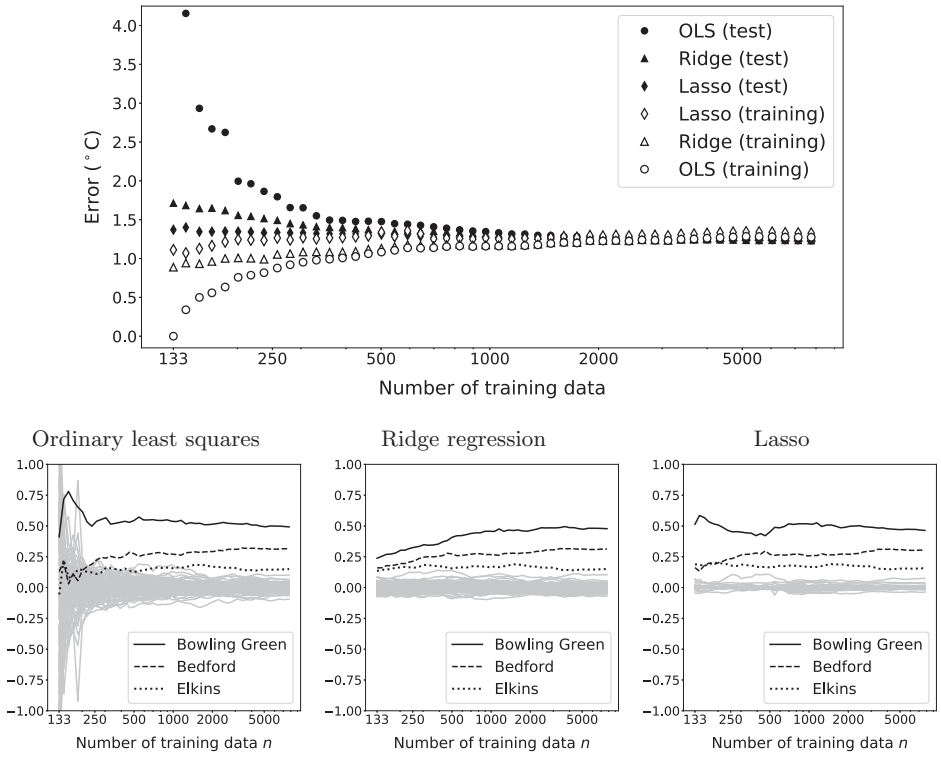


Figure 12.13 Comparison between ordinary least squares, ridge regression and the lasso. The top graph shows the training and test errors of OLS, ridge regression and the lasso applied to the temperature-estimation problem in Example 12.19, as a function of the number of training data n. The linear coefficients of the three models are depicted in the graphs below. For small n, OLS overfits the training data and has many large noisy coefficients, which drown out the three geographically-meaningful coefficients corresponding to Bowling Green, Bedford and Elkins. Ridge regression controls noise amplification in the coefficients, so that the three meaningful coefficients have the largest magnitude, resulting in better generalization. The lasso yields sparse coefficients, making the meaningful coefficients even more prominent, which further improves generalization. For large n, the three models yield similar results.

be repeated multiple times, using different training-validation splits, a technique known as *cross-validation*.

Figure 12.13 compares the performance of OLS and ridge regression on the temperature-estimation task from Example 12.19 for different values of the number of training data n. The regularization parameter is set via cross-validation for each n. When n is small, ridge-regression regularization prevents overfitting by neutralizing noise amplification in the coefficients. This results in a larger train-

ing error, and a much better test error. For large n both methods yield similar results.

As in the case of OLS (see Theorem 12.7), we can derive an explicit formula for the ridge-regression coefficients that only depends on the sample covariance matrix of the features, and on the sample cross-covariance between the features and the response.

Theorem 12.29 (Ridge-regression estimator)**.** *Let (x_1, y_1), (x_2, y_2), ..., (x_n, y_n) be a dataset formed by n pairs of a response y_i and a corresponding vector x_i containing d features $(1 \leq i \leq n)$. If the features and the response are centered to have zero sample mean, the linear coefficients of the ridge-regression estimator equal*

$$\beta_{\mathrm{RR}} = \left(\Sigma_X + \frac{\lambda}{n-1} I\right)^{-1} \Sigma_{XY}, \tag{12.224}$$

where Σ_X is the covariance matrix of the features, Σ_{XY} the sample cross-covariance of the features and the response, and λ the ridge-regression regularization parameter.

Proof Let us denote the feature design matrix by $X_{\mathrm{train}} \in \mathbb{R}^{d \times n}$ and the corresponding response vector by $y_{\mathrm{train}} \in \mathbb{R}^n$. The ridge-regression cost function is equivalent to the cost function of a modified OLS estimator,

$$\sum_{i=1}^{n} \left(y_i - \beta^T x_i\right)^2 + \lambda \sum_{j=1}^{d} \beta[j]^2 = \left|\left|y_{\mathrm{train}} - X_{\mathrm{train}}^T \beta\right|\right|_2^2 + \left|\left|0 - \sqrt{\lambda} I \beta\right|\right|_2^2 \tag{12.225}$$

$$= \left|\left| \begin{bmatrix} y_{\mathrm{train}} \\ 0 \end{bmatrix} - \begin{bmatrix} X_{\mathrm{train}}^T \\ \sqrt{\lambda} I \end{bmatrix} \beta \right|\right|_2^2 \tag{12.226}$$

$$= \left|\left|y_{\mathrm{RR}} - X_{\mathrm{RR}}^T \beta\right|\right|_2^2, \tag{12.227}$$

with design matrix

$$X_{\mathrm{RR}} := \begin{bmatrix} X_{\mathrm{train}} & \sqrt{\lambda} I \end{bmatrix} \tag{12.228}$$

and response vector

$$y_{\mathrm{RR}} := \begin{bmatrix} y_{\mathrm{train}} \\ 0 \end{bmatrix}. \tag{12.229}$$

Here 0 denotes a vector of d zeros and I is an $d \times d$ identity matrix. By Lemma 12.8,

the coefficients of the modified OLS estimator equal

$$\beta_{\text{RR}} := \arg\min_{\beta\in\mathbb{R}^d} \left|\left|y_{\text{RR}} - X_{\text{RR}}^T\beta\right|\right|_2^2 \tag{12.230}$$

$$= \left(X_{\text{RR}}X_{\text{RR}}^T\right)^{-1} X_{\text{RR}}y_{\text{RR}} \tag{12.231}$$

$$= \left(\begin{bmatrix} X_{\text{train}} & \sqrt{\lambda}I \end{bmatrix}\begin{bmatrix} X_{\text{train}} & \sqrt{\lambda}I \end{bmatrix}^T\right)^{-1}\begin{bmatrix} X_{\text{train}} & \sqrt{\lambda}I \end{bmatrix}\begin{bmatrix} y_{\text{train}} \\ 0 \end{bmatrix} \tag{12.232}$$

$$= \left(X_{\text{train}}X_{\text{train}}^T + \lambda I\right)^{-1} X_{\text{train}}y_{\text{train}} \tag{12.233}$$

$$= \left(\Sigma_X + \frac{\lambda}{n-1}I\right)^{-1}\Sigma_{XY}. \tag{12.234}$$

■

Comparing Theorems 12.29 and 12.7, the difference between the ridge-regression and the OLS coefficients is that the matrix multiplying the cross-covariance Σ_{XY} is the inverse of $\Sigma_X + \frac{\lambda}{n-1}I$, instead of the inverse of the feature covariance matrix Σ_X. This results in a simple relationship between the OLS and ridge-regression coefficients, which becomes apparent when we express them in the basis of principal directions of the features. The component of the ridge-regression coefficients in each principal direction is equal to the corresponding OLS component shrunk by a factor that depends on the regularization parameter λ and the sample variance of the features in that direction.

Theorem 12.30 (Relationship between ridge-regression and OLS coefficients)**.** *Let (x_1, y_1), (x_2, y_2), ..., (x_n, y_n) be a dataset formed by n pairs of a response y_i and a d-dimensional feature vector x_i $(1 \leq i \leq n)$, with an invertible sample covariance matrix Σ_X. We assume that the features and the response are centered to have zero sample mean. Let*

$$\beta_{\text{OLS}} = \sum_{j=1}^{d} c_{\text{OLS}}[j]u_j \tag{12.235}$$

be the representation of the OLS coefficients in the basis of principal directions of the features (the eigenvectors of Σ_X) u_1, u_2, ..., u_d. The ridge-regression coefficients equal

$$\beta_{\text{RR}} = \sum_{j=1}^{d} c_{\text{RR}}[j]u_j, \tag{12.236}$$

$$c_{\text{RR}}[j] = \frac{c_{\text{OLS}}[j]}{1 + \lambda_n/\xi_j}, \tag{12.237}$$

where $\lambda_n := \frac{\lambda}{n-1}$, λ is the ridge-regression regularization parameter, and $\xi_1 \geq \xi_2 \geq \cdots \geq \xi_d$ are the sample variances of the features in each of their principal directions (the eigenvalues of Σ_X).

Proof Let $U := \begin{bmatrix} u_1 & u_2 & \cdots & u_d \end{bmatrix}$ and let Λ be a diagonal matrix with $\xi_1 \geq$

$\xi_2 \geq \cdots \geq \xi_d$ in the diagonal, so that $\Sigma_X = U\Lambda U^T$. The principal directions are orthonormal by Theorem 11.19, so the inverse of U is U^T and vice versa. Consequently, the inverse of Σ_X equals

$$\Sigma_X^{-1} = U\Lambda^{-1}U^T = \sum_{j=1}^{d} \frac{1}{\xi_j} u_j u_j^T, \tag{12.238}$$

so by Theorem 12.7

$$\beta_{\text{OLS}} = \Sigma_X^{-1}\Sigma_{XY} \tag{12.239}$$

$$= \sum_{j=1}^{d} \frac{1}{\xi_j} u_j u_j^T \Sigma_{XY}, \tag{12.240}$$

which implies

$$c_{\text{OLS}}[j] = \frac{u_j^T \Sigma_{XY}}{\xi_j}. \tag{12.241}$$

Since $UU^T = I$,

$$\Sigma_X + \lambda_n I = U\left(\Lambda + \lambda_n I\right) U^T, \tag{12.242}$$

so

$$\left(\Sigma_X + \lambda_n I\right)^{-1} = U\left(\Lambda + \lambda_n I\right)^{-1} U^T = \sum_{j=1}^{d} \frac{1}{\xi_j + \lambda_n} u_j u_j^T. \tag{12.243}$$

By Theorem 12.29

$$\beta_{\text{RR}} = \left(\Sigma_X + \lambda_n I\right)^{-1} \Sigma_{XY} \tag{12.244}$$

$$= \sum_{j=1}^{d} \frac{1}{\xi_j + \lambda_n} u_j u_j^T \Sigma_{XY}, \tag{12.245}$$

which implies

$$c_{\text{RR}}[j] = \frac{u_j^T \Sigma_{XY}}{\xi_j + \lambda_n} \tag{12.246}$$

$$= \frac{\xi_j}{\xi_j + \lambda_n} \frac{u_j^T \Sigma_{XY}}{\xi_j} \tag{12.247}$$

$$= \frac{c_{\text{OLS}}[j]}{1 + \lambda_n / \xi_j}. \tag{12.248}$$

■

Theorem 12.30 reveals why ridge regularization is able to alleviate noise amplification in OLS coefficients. As explained in Section 12.2.2, noise amplification occurs when the features have low variance in certain directions, because there is multicollinearity in the design matrix. Consider a regression problem where the features have high variance ξ_1 in a certain principal direction u_1 and very low

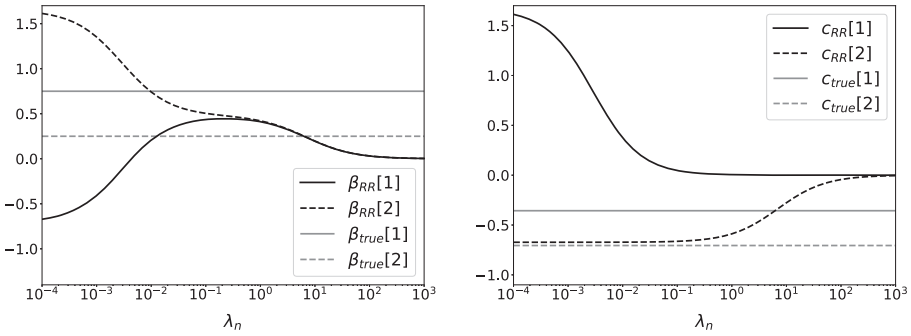


Figure 12.14 Effect of regularization in ridge regression. The left plot shows the ridge-regression coefficients (in black) as a function of the scaled regularization parameter λ_n for the regression problem in Example 12.31, as well as the true coefficients (in gray). The right plot shows the two components of the ridge-regression coefficients (in black) and the true coefficients (in gray) represented in the basis of principal directions of the features. The first component is close to the truth, whereas the second is completely distorted by the noise. The magnitude of both ridge-regression components decreases monotonically as λ_n increases, but the decrease is much faster for the noisy component. Consequently, for intermediate values of λ_n, this component is suppressed, while the first one is preserved, resulting in a better approximation to the true coefficients.

variance ξ_2 in another principal direction u_2. As a result, the component of the OLS coefficient in the first direction $c_{\text{OLS}}[1] := u_1^T \beta_{\text{OLS}}$ is mostly unaffected by the noise, whereas the component in the second direction $c_{\text{OLS}}[2] := u_2^T \beta_{\text{OLS}}$ is noisy and unreliable. A concrete illustration of this is provided in Example 12.31 and Figure 12.16. To reduce noise amplification, we would like to suppress $c_{\text{OLS}}[2]$, but not $c_{\text{OLS}}[1]$. Ridge regression achieves this if λ_n is selected so that $\lambda_n/\xi_2 \gg 1$ and $\lambda_n/\xi_1 \ll 1$, which is possible since by assumption ξ_1 is much larger than ξ_2. By Theorem 12.30, for such λ_n

$$c_{\text{RR}}[1] = \frac{c_{\text{OLS}}[1]}{1 + \lambda_n/\xi_1} \approx c_{\text{OLS}}[1], \tag{12.249}$$

$$c_{\text{RR}}[2] = \frac{c_{\text{OLS}}[2]}{1 + \lambda_n/\xi_2} \approx 0, \tag{12.250}$$

so ridge regression selectively inhibits the noisier component, while preserving the cleaner component. The following example shows that this can substantially improve the estimate of the true linear coefficients for the finite-data linear-response model with additive noise in Definition 12.18.

Example 12.31 (Ridge regression and feature multicollinearity)**.** Example 12.21 illustrates the effect of multicollinearity in the OLS coefficients for a regression problem with two features, where the response follows Definition 12.18. Inspired by Theorem 12.30, we represent the true coefficients β_{true} in the orthonormal

basis of principal directions u_1 and u_2 of the features, which are the eigenvectors of the sample covariance matrix Σ_X:

$$\beta_{\text{true}} := \begin{bmatrix} 0.75 \\ 0.25 \end{bmatrix} = c_{\text{true}}[1]u_1 + c_{\text{true}}[2]u_2, \tag{12.251}$$

$$c_{\text{true}} = U^T \beta_{\text{true}} = \begin{bmatrix} -0.71 \\ -0.36 \end{bmatrix}. \tag{12.252}$$

In this basis, the components of the cross-covariance between the features and the noise in the basis of principal directions are

$$u_1^T \Sigma_{XZ} := -0.055, \qquad u_2^T \Sigma_{XZ} := -0.053. \tag{12.253}$$

The variance of the features in the two principal directions are very different:

$$\xi_1 := 2.33, \qquad \xi_2 := 9.68 \cdot 10^{-4}. \tag{12.254}$$

By Theorem 12.20, the OLS components equal

$$c_{\text{OLS}} = U^T \beta_{\text{OLS}} \tag{12.255}$$

$$= U^T \left(\beta_{\text{true}} + \Sigma_X^{-1} \Sigma_{XZ} \right) \tag{12.256}$$

$$= c_{\text{true}} + \begin{bmatrix} \frac{u_1^T \Sigma_{XZ}}{\xi_1} \\ \frac{u_2^T \Sigma_{XZ}}{\xi_2} \end{bmatrix} = c_{\text{true}} + \begin{bmatrix} 0.03 \\ 2.03 \end{bmatrix}. \tag{12.257}$$

Due to the disparity between the corresponding feature variances, the first coefficient $c_{\text{OLS}}[1]$ is very close to $c_{\text{true}}[1]$, but the second one $c_{\text{OLS}}[2]$ is completely distorted by the noise. By Theorem 12.30 the corresponding components of the ridge-regression coefficients equal

$$c_{\text{RR}}[1] = \frac{c_{\text{true}}[1] + 0.03}{1 + 0.43\lambda_n}, \tag{12.258}$$

$$c_{\text{RR}}[2] = \frac{c_{\text{true}}[2] + 2.03}{1 + 1033\lambda_n}, \tag{12.259}$$

so they are a monotonically-decreasing function of λ_n, as depicted in the right plot of Figure 12.16. In contrast, the relationship between the coefficients β_{RR} and λ_n is much more complicated, as evinced by the trajectory of $\beta_{\text{RR}}[2]$ in the left plot of Figure 12.16. This highlights the importance of analyzing the ridge-regression coefficients in the basis of principal directions.

Crucially, $c_{\text{RR}}[2]$ decreases much faster than $c_{\text{RR}}[1]$ as a function of λ_n. This makes it possible to choose λ_n in order to neutralize noise amplification and better approximate the true linear coefficients. Setting $\lambda_n := 0.1$ so that

$$\frac{\lambda_n}{\xi_2} = 103.3 \gg 1 \gg 0.043 = \frac{\lambda_n}{\xi_1}, \tag{12.260}$$

suppresses the noisier coefficient while preserving the cleaner one:

$$c_{\text{RR}}[1] = \frac{c_{\text{true}}[1] + 0.03}{1.043} = -0.65 \approx c_{\text{true}}[1], \tag{12.261}$$

$$c_{\text{RR}}[2] = \frac{c_{\text{true}}[2] + 2.03}{103.3} = 0.016 \approx 0. \tag{12.262}$$

The resulting ridge-regression coefficients equal

$$\beta_{\text{RR}} = \begin{bmatrix} 0.44 \\ 0.47 \end{bmatrix}, \tag{12.263}$$

which are much closer to $\beta_{\text{true}} := \left[\begin{smallmatrix} 0.75 \\ 0.25 \end{smallmatrix}\right]$ than the OLS coefficients $\beta_{\text{OLS}} = \left[\begin{smallmatrix} -0.71 \\ 1.65 \end{smallmatrix}\right]$. Figure 12.15 shows the contour lines of the OLS cost function (top left), the ridge-regression regularization term (top right) and the ridge-regression cost function (bottom left) for $\lambda_n := 0.1$. Regularization shifts the ridge-regression minimum towards the origin, closer to the true coefficients. Consistent with our analysis, the shift occurs mostly in the direction of u_2 (bottom right), canceling out the corresponding component.

To gain further insight into the effect of regularization in ridge regression, we consider the finite-data linear-response model in Definition 12.18, when the additive noise is random. Under this model, the component $\tilde{c}_{\text{OLS}}[j]$ of the OLS coefficients in the jth principal direction of the features u_j is unbiased, since, by Theorem 12.22,

$$\text{E}\left[\tilde{c}_{\text{OLS}}[j]\right] = \text{E}\left[u_j^T \tilde{\beta}_{\text{OLS}}\right] \tag{12.264}$$

$$= u_j^T \text{E}\left[\tilde{\beta}_{\text{OLS}}\right] \tag{12.265}$$

$$= u_j^T \beta_{\text{true}} := c_{\text{true}}[j]. \tag{12.266}$$

By Theorem 12.23 the variance of the component equals

$$\text{Var}\left[\tilde{c}_{\text{OLS}}[j]\right] = \text{Var}\left[u_j^T \tilde{\beta}_{\text{OLS}}\right] \tag{12.267}$$

$$= u_j^T \Sigma_{\tilde{\beta}_{\text{OLS}}} u_j \tag{12.268}$$

$$= \frac{\sigma^2}{n-1} u_j^T \Sigma_X^{-1} u_j \tag{12.269}$$

$$= \frac{\sigma^2}{n-1} \sum_{l=1}^{d} \frac{1}{\xi_l} u_j^T u_l u_l^T u_j \tag{12.270}$$

$$= \frac{\sigma^2}{(n-1)\xi_j}, \tag{12.271}$$

because $\Sigma_X^{-1} = \sum_{l=1}^{d} \frac{1}{\xi_l} u_l u_l^T$, and $u_j^T u_l$ equals one when $j = l$ (and zero otherwise). Consequently, the component suffers from noise amplification if the feature variance ξ_j in that direction is small, as depicted in Figure 12.9.

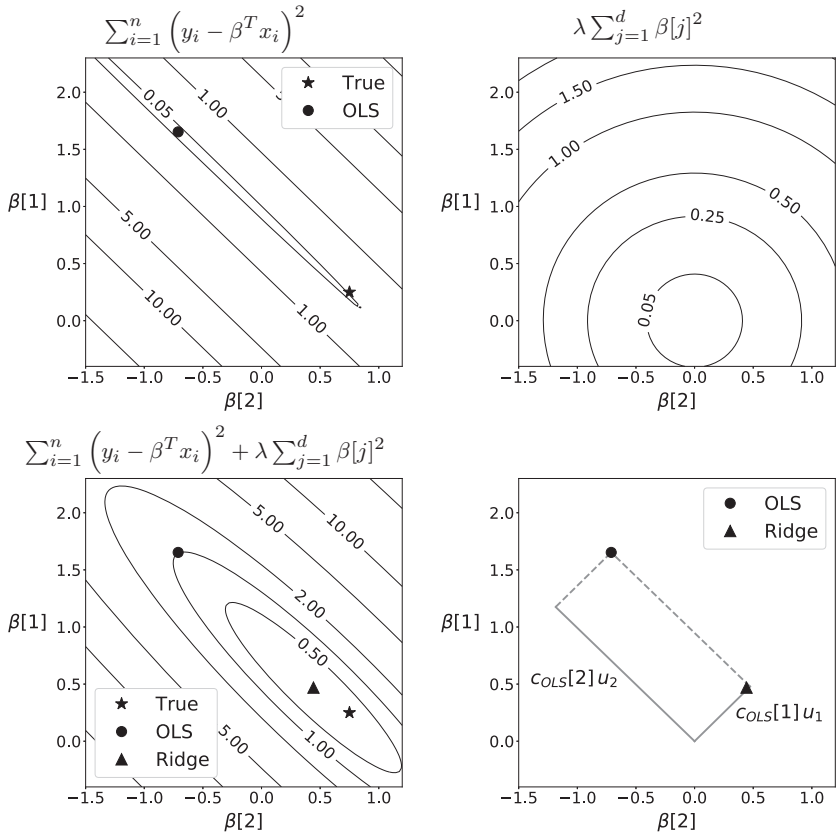


Figure 12.15 Optimization landscape of ridge regression. The top left plot shows the contour lines of the OLS cost function for the data in Examples 12.21 and 12.31. Due to multicollinearity in the feature design matrix, the minimum of the OLS cost function (circle marker) is far from the true coefficients (star marker). The top right plot shows the contour lines of the regularization term in ridge regression for $\lambda := \frac{0.1}{n-1}$. The bottom left plot shows the contour lines of the ridge-regression cost function, resulting from summing the two terms in the top row. Regularization shifts the minimum towards the origin, closer to the true coefficients (triangle marker). The bottom right plot depicts the components of the OLS minimum in the basis of principal directions of the features, revealing that this shift takes place almost entirely in the direction of the second principal direction, as described in Example 12.31.

Figure 12.16 shows the distribution of the ridge-regression coefficients for different values of the regularization parameter λ. As λ increases, the center of the distribution moves towards the origin, resulting in a biased estimate that is not centered at the true coefficients (unlike the OLS estimate). However, this is accompanied by a decrease in the variance of the distribution when compared to the

OLS coefficients. As a result, most realizations of the ridge-regression estimator provide a better approximation of the true coefficients than the OLS estimator for intermediate values of λ. The following theorem provides an explanation for this. It establishes that the mean and variance of the component of the ridge-regression coefficients in the jth principal direction equal

$$\mathrm{E}\left[\tilde{c}_{\mathrm{RR}}[j]\right] = \frac{c_{\mathrm{true}}[j]}{1 + \lambda_n/\xi_j}, \tag{12.272}$$

$$\mathrm{Var}\left[\tilde{c}_{\mathrm{RR}}[j]\right] = \frac{\mathrm{Var}\left[\tilde{c}_{\mathrm{OLS}}[j]\right]}{\left(1 + \lambda_n/\xi_j\right)^2}, \tag{12.273}$$

where $\lambda_n := \frac{\lambda}{n-1}$ and λ is the ridge-regression regularization parameter. Consider two principal directions u_1 and u_2 in which the feature variance are very high and very low, respectively ($\xi_1 \gg \xi_2$). Then, the OLS coefficient has low variance in the first direction and high variance in the second direction. If we select λ_n, so that

$$\lambda_n/\xi_2 \gg 1 \gg \lambda_n/\xi_1, \tag{12.274}$$

we suppress the variance in the second direction, while remaining unbiased in the first direction, because

$$\mathrm{E}\left[\tilde{c}_{\mathrm{RR}}[1]\right] \approx c_{\mathrm{true}}[1], \tag{12.275}$$

$$\mathrm{Var}\left[\tilde{c}_{\mathrm{RR}}[2]\right] \approx 0. \tag{12.276}$$

The price to pay is bias in the second direction, because

$$\mathrm{E}\left[\tilde{c}_{\mathrm{RR}}[2]\right] \approx 0. \tag{12.277}$$

The ridge-regression estimator is thus able to selectively neutralize the variance in the directions suffering from noise amplification, by biasing the corresponding components towards zero.

Theorem 12.32 (Mean and variance of the ridge-regression coefficients)**.** *Let $x_1, \ldots, x_n$ be d-dimensional feature vectors with sample covariance matrix Σ_X, and let $\beta_{\mathrm{true}} \in \mathbb{R}^d$ be a fixed vector of linear coefficients. The response $\tilde{y}_{\mathrm{train}} \in \mathbb{R}^n$ is defined as in (12.137) for a noise vector $\tilde{z}_{\mathrm{train}}$ with independent entries that have zero mean and variance σ^2. We express the true coefficients and the ridge-regression estimator in the basis of principal directions of the features (the eigenvectors of Σ_X):*

$$\beta_{\mathrm{true}} = \sum_{j=1}^{d} c_{\mathrm{true}}[j] u_j, \tag{12.278}$$

$$\tilde{\beta}_{\mathrm{RR}} = \sum_{j=1}^{d} \tilde{c}_{\mathrm{RR}}[j] u_j. \tag{12.279}$$

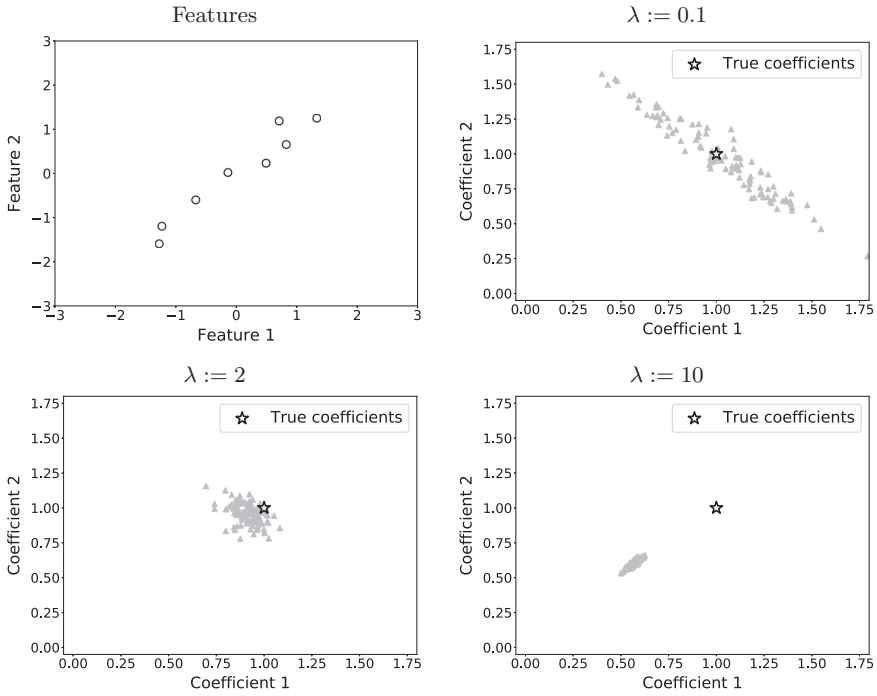


Figure 12.16 Ridge-regression coefficients for finite-data linear-response model with additive noise. The top left plot shows two-dimensional feature vectors corresponding to $n := 8$ training examples. The remaining plots show 100 ridge-regression coefficient estimates corresponding to 100 different responses generated according to Definition 12.18 using fixed-variance i.i.d. Gaussian noise and the same true linear coefficients (represented by a star). Each plot corresponds to a different value of the regularization parameter λ. As λ increases, the mean of the coefficient distribution moves towards the origin and its variance decreases, which is consistent the theoretical analysis in Theorem 12.32. The variance decrease occurs more rapidly in the principal direction of minimum variance of the features, which is the direction in which the OLS coefficients have the highest variance. Consequently, the resulting coefficient estimate tends to be closer to the true coefficients than the OLS estimate for intermediate values of λ (bottom left).

The mean and variance of the ridge-regression coefficients equal

$$\mathrm{E}\left[\tilde{c}_{\mathrm{RR}}[j]\right] = \frac{c_{\mathrm{true}}[j]}{1 + \lambda_n/\xi_j}, \tag{12.280}$$

$$\mathrm{Var}\left[\tilde{c}_{\mathrm{RR}}[j]\right] = \frac{1}{\left(1 + \lambda_n/\xi_j\right)^2} \frac{\sigma^2}{(n-1)\xi_j}, \tag{12.281}$$

where $\lambda_n := \frac{\lambda}{n-1}$, λ *is the ridge-regression regularization parameter, and* $\xi_1 \geq$

$\xi_2 \geq \cdots \geq \xi_d$ are the sample variances of the features in each of their principal directions (the eigenvalues of Σ_X).

Proof The jth OLS coefficient equals

$$\tilde{c}_{\mathrm{OLS}}[j] = u_j^T \tilde{\beta}_{\mathrm{OLS}}, \tag{12.282}$$

so by Theorem 12.30

$$\tilde{c}_{\mathrm{RR}}[j] = \frac{u_j^T \tilde{\beta}_{\mathrm{OLS}}}{1 + \lambda_n / \xi_j}. \tag{12.283}$$

Consequently, by linearity of expectation and Theorem 12.22

$$\mathrm{E}\left[\tilde{c}_{\mathrm{RR}}[j]\right] = \frac{u_j^T \mathrm{E}\left[\tilde{\beta}_{\mathrm{OLS}}\right]}{1 + \lambda_n / \xi_j} \tag{12.284}$$

$$= \frac{u_j^T \beta_{\mathrm{true}}}{1 + \lambda_n / \xi_j} \tag{12.285}$$

$$= \frac{c_{\mathrm{true}}[j]}{1 + \lambda_n / \xi_j}, \tag{12.286}$$

and by (12.271)

$$\mathrm{Var}\left[\tilde{c}_{\mathrm{RR}}[j]\right] = \mathrm{Var}\left[\frac{\tilde{c}_{\mathrm{OLS}}[j]}{1 + \lambda_n / \xi_j}\right] \tag{12.287}$$

$$= \frac{\mathrm{Var}\left[\tilde{c}_{\mathrm{OLS}}[j]\right]}{\left(1 + \lambda_n / \xi_j\right)^2} \tag{12.288}$$

$$= \frac{1}{\left(1 + \lambda_n / \xi_j\right)^2} \frac{\sigma^2}{(n-1)\xi_j}. \tag{12.289}$$

■

12.4 Sparse Regression

As described in Section 12.2, the OLS estimator tends to overfit when the number of data is small with respect to the number of features. In such scenarios, it is often useful to perform *variable selection*, in order to model the response using only a subset of the features. This can be achieved by constraining the vector of coefficients in a linear model to be *sparse*, meaning that most of its entries are zero. In a regression problem with d features, if the linear coefficients $\beta \in \mathbb{R}^d$ have k nonzero coefficients, then the resulting model just depends on the corresponding k features,

$$y \approx \sum_{j=1}^{d} \beta[j] x[j] = \sum_{j \in \mathcal{K}} \beta[j] x[j], \tag{12.290}$$

where y is the response, $x \in \mathbb{R}^d$ is the feature vector, and $\mathcal{K}$ is the set of nonzero coefficients. We assume that the feature and the response are centered, so we don't need an intercept.

The lasso is a technique to obtain linear models with sparse coefficients, based on the insight that sparse vectors tend to have small ℓ_1 norm. Let us illustrate this with an idealized example. Consider a regression problem with d features, where there are several candidate coefficient vectors $\beta_k \in \mathbb{R}^d$ with a number of nonzero entries k ranging from 1 to d. The ℓ_2 norm of the vectors equals one, and the entries all have the same magnitude $(1/\sqrt{k})$. Consequently, the ℓ_1 norm of each vector scales as the square root of the number of nonzero entries k,

$$||\beta_k||_1 = \sum_{j=1}^{k} \left| \frac{1}{\sqrt{k}} \right| = \sqrt{k}. \tag{12.291}$$

The sparser the vector, the smaller its ℓ_1 norm. If several of these coefficient vectors yield a good fit to the data, then we can identify the sparsest one by penalizing the ℓ_1-norm of the coefficients, while fitting the model. This is the strategy leveraged by the lasso estimator, originally introduced in (Tibshirani, 1996) (see also (**?**)).

Definition 12.33 (The lasso)**.** *Let (x_1, y_1), (x_2, y_2), ..., (x_n, y_n) be a dataset formed by n pairs of a response y_i and a corresponding vector x_i containing d features $(1 \leq i \leq n)$. We assume that the features and the response are centered to have zero sample mean. The lasso estimator of the response given the features is*

$$\ell_{\text{lasso}}(x_i) := \beta_{\text{lasso}}^T x_i, \quad 1 \leq i \leq n, \tag{12.292}$$

where the linear coefficients minimize the regularized least-squares cost function,

$$\beta_{\text{lasso}} := \arg\min_{\beta \in \mathbb{R}^d} \sum_{i=1}^{n} \left(y_i - \beta^T x_i \right)^2 + \lambda \sum_{j=1}^{d} |\beta[j]|, \tag{12.293}$$

and λ is a nonnegative regularization parameter. In matrix-vector form

$$\beta_{\text{lasso}} := \arg\min_{\beta \in \mathbb{R}^d} ||y - X\beta||_2 + \lambda ||\beta||_1, \tag{12.294}$$

where $y[i] := y_i$ for $1 \leq i \leq n$ and $X := \begin{bmatrix} x_1 & x_2 & \cdots & x_n \end{bmatrix}$.

The lasso cost function is reminiscent of ridge regression (see Section 12.3), with the crucial difference that the regularization term is designed to promote sparsity of the coefficients. As in ridge-regression, the regularization parameter λ governs the trade-off between the data-fidelity term and the regularization term. When λ is small, the data-fidelity term dominates, so the lasso estimator is close to the OLS estimator. When λ is large, the regularization term dominates, yielding sparse coefficients. This is illustrated in Figure 12.17, which shows the results of applying the lasso to the weather regression problem from Example 12.19 in a regime where the OLS estimator overfits the training data. For small λ, the

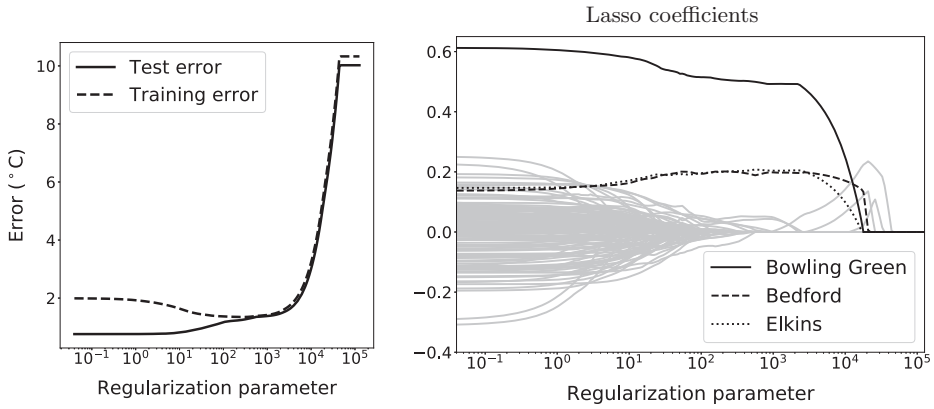


Figure 12.17 Effect of lasso regularization. The left graph shows the training and test errors of the lasso estimator applied to the temperature-estimation problem in Example 12.19 for different values of the regularization parameter λ. The number of training data is fixed to $n := 200$. The right figure shows the lasso coefficients as a function of λ. All coefficients are depicted in gray except the three that have the largest magnitudes for large n, which correspond to locations close to Versailles (see Figure 12.8).

coefficients have large negative and positive amplitudes, which drown out the geographically-meaningful coefficients associated with Bedford and Elkins (see Figure 12.8). This results in overfitting: the training error is small and the test error is large. As λ increases, most of the coefficients are set to zero, except for the meaningful coefficients (Bowling Green, Bedford and Elkins) and a few others. This prevents overfitting and achieves better generalization to the test data.

Comparing Figures 12.17 and 12.12, we can see the difference between the effect of regularization in ridge regression and in the lasso. Increasing the regularization parameter λ shrinks the ridge-regression coefficients gradually towards zero.$*$ In contrast, the lasso coefficients either vanish abruptly, or preserve relatively large amplitudes. This is apparent in Figure 12.13, which compares OLS, ridge regression and the lasso models on the temperature-regression problem from Example 12.19 for different values of the number of training data n. The regularization parameter is set via cross-validation for each n. When n is small, most of the lasso coefficients are exactly zero, as opposed to ridge regression, where most coefficients have small nonzero values. In addition, the coefficients corresponding to geographically-meaningful locations are larger for the lasso than for ridge regression. This results in a better test error, which suggests that the response indeed depends mostly on the few features selected by the lasso.

Figure 12.18 compares ridge regression and the lasso for the finite-data linear-response model in Definition 12.18 when the additive noise is random. One of the

$*$To be precise, the regularization shrinks the components of the coefficients in the principal directions of the features, as established in Theorem 12.30.

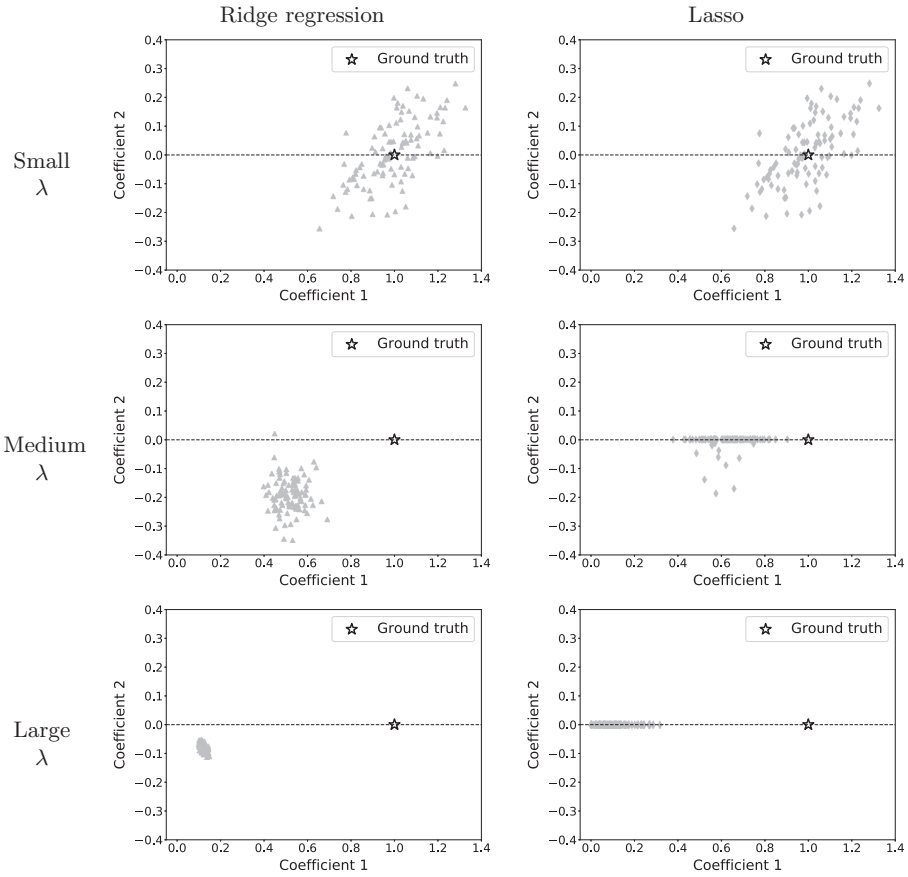


Figure 12.18 Ridge regression vs the lasso. The left column shows ridge-regression coefficients for different values of the regularization parameter λ, computed from responses following Definition 12.18 with fixed-variance i.i.d. Gaussian noise. The right column shows lasso coefficients also for different values of λ. For the lasso, increasing the regularization parameter results in sparse coefficient estimates, but this is not the case for ridge regression.

two true coefficients is zero. If the regularization parameter λ is large enough, the lasso estimator correctly identifies the correct coefficient and sets the other to zero. In contrast, the ridge-regression coefficients are not sparse, no matter how much we increase the regularization parameter. This is consistent with Theorem 12.30: the ridge-regression estimator gradually shrinks the coefficients in the basis of principal directions of the features, which does not induce sparsity (see Exercise 12.12).

The left column of Figure 12.15 shows the contour lines of the OLS cost function (top), the lasso regularization term (center) and the lasso cost function (bottom)

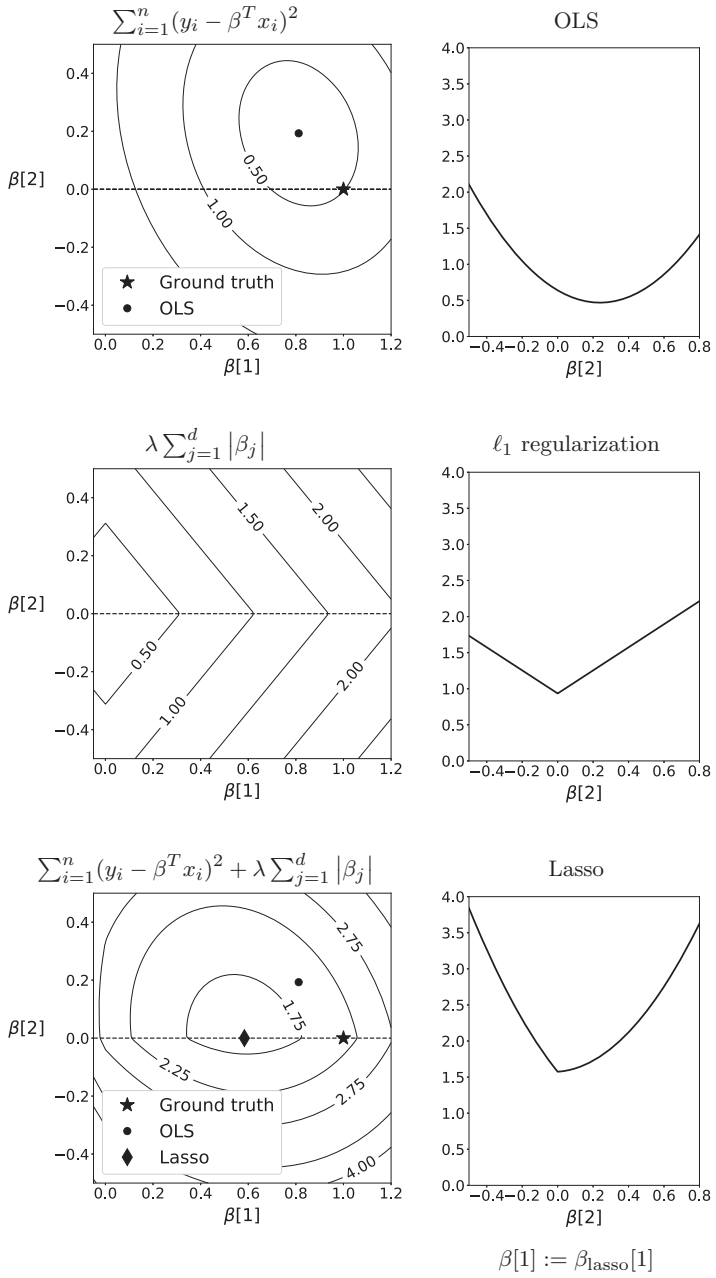


Figure 12.19 Optimization landscape of the lasso. The left columns shows the contour lines of the OLS cost function (top), the lasso regularization term (center) and the lasso cost function (bottom) for a specific instance of the finite-data linear-response model, where the true second coefficient is zero (see Exercise 12.12 for the values of the feature matrix and the response). The regularization parameter is set to $\lambda := 1.6$. Even though the second true coefficient (star marker) is zero, the second OLS coefficient (circle marker) is nonzero due to the noise in the response. The ℓ_1-norm regularization shifts the minimum of the cost function to the horizontal line, so that the second lasso coefficient is zero (diamond marker). This is evident in the right column, which shows the different functions restricted to the line where the first coefficient $\beta[1]$ is equal to the first lasso coefficient $\beta_{\text{lasso}}[1]$.

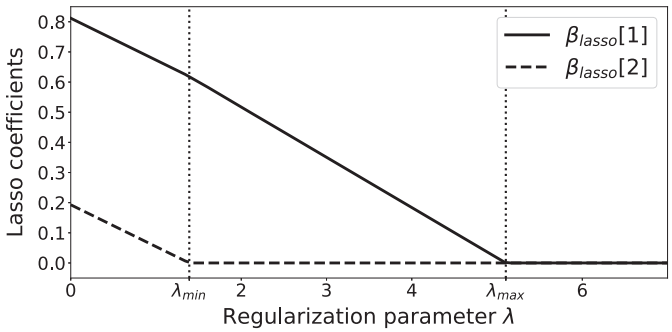


Figure 12.20 Effect of regularization in the lasso. The plot depicts the lasso coefficients for the data in Figure 12.19 as a function of the regularization parameter λ. The second coefficient decreases linearly with λ, until it vanishes at $\lambda := \lambda_{\min}$. From then on, it remains zero, while the first coefficient decreases linearly and eventually vanishes too at $\lambda := \lambda_{\max}$. Precise expressions for $\lambda_{\min}$ and $\lambda_{\max}$ are derived in Exercise 12.11.

for a specific instance of the finite-data linear-response model, where the true second coefficient is zero. The ℓ_1-norm regularization shifts the minimum of the cost function to the horizontal line, resulting in a sparse estimate. The right column shows a 1D slice of the different functions on the line where the first coefficient is fixed to its optimal value. This visualization reveals that, for values close to the origin, the linear regularization term dominates the quadratic data-fidelity term, warping the cost function so that the minimum is exactly at zero.

Figure 12.20 plots the lasso coefficients for the data in Figure 12.19 as a function of the regularization parameter λ. For small λ, both coefficients are nonzero, but when λ is above a certain threshold $\lambda_{\min}$, the second coefficient vanishes. As λ increases further, the first coefficient decreases linearly and eventually vanishes too. Exercise 12.11 establishes that this is a general phenomenon, and derives a precise expression for $\lambda_{\min}$ and $\lambda_{\max}$ in terms of the correlation between the features, and the correlation between the features and the noise.

12.5 Logistic Regression

In this section we describe logistic regression, which is a popular linear model for binary classification problems, where there are just two classes. In Section 12.6 we extend the approach to problems with more classes. Let us represent the features in a binary classification problem as a d-dimensional random vector $\tilde{x}$ and the corresponding class label as a Bernoulli random variable $\tilde{y}$, which equals 0 or 1. Our goal is to use a linear model to estimate the conditional probability that $\tilde{y} = 1$ given $\tilde{x} = x$, for any observed feature vector $x \in \mathbb{R}^d$. Unfortunately, we

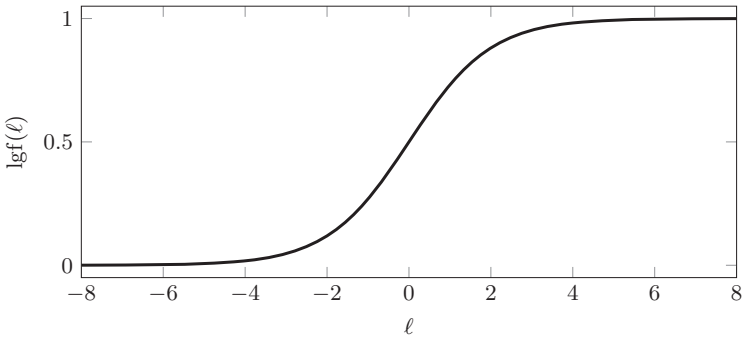


Figure 12.21 Logistic function. The logistic function is a smooth monotone function that maps the real line to the unit interval.

cannot model the probability as an affine function of the features,

$$\beta^T x + \alpha, \tag{12.295}$$

where $\beta \in \mathbb{R}^d$ and $\alpha \in \mathbb{R}$. For many values of x, the affine function is negative or greater than one, so it would not be a valid probability. To address this, we use the *logistic function*,

$$\text{lgf}(\ell) := \frac{\exp(\ell)}{1 + \exp(\ell)} = \frac{1}{1 + \exp(-\ell)}, \tag{12.296}$$

to transform the output of the affine function, so that it remains in the interval $[0, 1]$. The composition of the affine and logistic functions yields the logistic-regression model

$$\text{P}\left(\tilde{y} = 1 \,|\, \tilde{x} = x\right) = \text{lgf}(\beta^T x + \alpha). \tag{12.297}$$

Mapping linear or affine functions of the features to probability estimates using a one-dimensional *link* function, such as the logistic function, is known as *generalized linear modeling*.

Figure 12.21 depicts the logistic function. It is a smooth monotone function that is almost linear at the origin. It tends towards zero on the left and towards one on the right. Theorem 6.7 motivates its use to model probabilities. The inverse of the logistic function is called the logit function, so its input is often referred to as a logit.. If we interpret the output of a logistic function as a probability $p := \text{lgf}(\ell)$, then the associated odds (defined as the ratio between the probability of an event, and the probability of its complement) equal

$$\frac{p}{1-p} = \frac{\text{lgf}(\ell)}{1 - \text{lgf}(\ell)} = \exp\left(\ell\right). \tag{12.298}$$

Consequently, the logit ℓ is the logarithm of the odds, or log odds. In logistic

regression, the logit and hence the log odds are an affine function of the features:

$$\log\left(\frac{\mathrm{P}\left(\tilde{y}=1\,|\,\tilde{x}=x\right)}{1-\mathrm{P}\left(\tilde{y}=1\,|\,\tilde{x}=x\right)}\right)=\beta^T x+\alpha. \tag{12.299}$$

Logistic regression is a *discriminative* classification method, because it seeks to directly approximate the conditional distribution of the class labels given the features. In contrast, naive Bayes (described in Section 4.8) and Gaussian discriminant analysis (described in Section 6.5) are called *generative* approches, because they model the full joint distribution of the labels and the features generating the data.

In order to fit a logistic-regression model to data, we maximize the corresponding likelihood function. In Section 2.4, we define the likelihood of a parametric model as the joint pmf of the data, interpreted as a function of the model parameters. Here we use a different definition, because we are interested in modeling a conditional distribution, as opposed to a marginal distribution as in Section 2.4. We define the likelihood as the *conditional* pmf of the observed labels, given the observed features. The following theorem derives the conditional pmf under the assumption that the labels are conditionally independent given the features, and that the label of each example is conditionally independent of the other feature vectors given its own feature vector.

Theorem 12.34 (Logistic-regression likelihood)**.** *Let* (x_1, y_1), (x_2, y_2), ..., (x_n, y_n) *be a dataset formed by* n *pairs of a binary label* $y_i \in \{0, 1\}$ *and a corresponding feature vector* $x_i \in \mathbb{R}^d$*. We model the data as a realization from the joint distribution of random vectors representing the feature vectors* $\tilde{x}_1$, ..., $\tilde{x}_n$ *and random variables representing the labels* $\tilde{y}_1$, ..., $\tilde{y}_n$*. We define the likelihood of the logistic-regression model*

$$p_{\alpha,\beta}(x) := \mathrm{lgf}(\beta^T x+\alpha), \tag{12.300}$$

with parameters $\alpha \in \mathbb{R}$ *and* $\beta \in \mathbb{R}^d$*, as the conditional pmf of the labels given the features. If the labels are conditionally independent given the features, and each label* $\tilde{y}_i$ *is conditionally independent from the remaining feature vectors* $\{\tilde{x}_m\}_{m\neq i}$ *given* $\tilde{x}_i$*, then the likelihood equals*

$$\mathcal{L}_{XY}(\alpha,\beta)=\prod_{\{i:y_i=0\}}\left(1-p_{\alpha,\beta}(x_i)\right)\prod_{\{l:y_l=1\}}p_{\alpha,\beta}(x_l), \tag{12.301}$$

and the corresponding log-likelihood is

$$\log\mathcal{L}_{XY}(\alpha,\beta)=\sum_{\{i:y_i=0\}}\log\left(1-p_{\alpha,\beta}(x_i)\right)+\sum_{\{l:y_l=1\}}\log p_{\alpha,\beta}(x_l). \tag{12.302}$$

Proof By the conditional independence assumptions,

$$\mathcal{L}_{XY}(\alpha, \beta) := \mathrm{P}\left(\tilde{y}_1 = y_1, \ldots, \tilde{y}_n = y_n \,|\, \tilde{x}_1 = x_1, \ldots, \tilde{x}_n = x_n\right) \tag{12.303}$$

$$= \prod_{i=1}^{n} \mathrm{P}\left(\tilde{y}_i = y_i \,|\, \tilde{x}_1 = x_1, \ldots, \tilde{x}_n = x_n\right) \tag{12.304}$$

$$= \prod_{i=1}^{n} \mathrm{P}\left(\tilde{y}_i = y_i \,|\, \tilde{x}_i = x_i\right) \tag{12.305}$$

$$= \prod_{\{i:y_i=0\}} \left(1 - p_{\alpha,\beta}(x_i)\right) \prod_{\{l:y_l=1\}} p_{\alpha,\beta}(x_l), \tag{12.306}$$

setting $\mathrm{P}\left(\tilde{y}_i = 0 \,|\, \tilde{x}_i = x_i\right) = 1 - p_{\alpha,\beta}(x_i)$ and $\mathrm{P}\left(\tilde{y}_l = 1 \,|\, \tilde{x}_l = x_l\right) = p_{\alpha,\beta}(x_l)$. ■

The log-likelihood function derived in Theorem 12.34 is concave (see Section 7.1 in (Boyd and Vandenberghe, 2004)), but it does not have a closed-form maximum like the likelihood functions in Chapters 2 and 3. Consequently, logistic-regression models must be fit via iterative methods, such as gradient ascent (see Exercise 12.13) or faster alternatives such as Newton's method or reweighted least squares.

Example 12.35 (Sex classification based on height)**.** In this example we analyze the heights of 4,082 men and 1,986 women from the United States army, extracted from Dataset 5. Our goal is to fit a logistic-regression model that predicts the sex of an individual from their height. The top left graph in Figure 12.22 shows the log-likelihood function of the model. The top right plot shows the affine logits corresponding to different choices of the parameters. These affine functions of the feature are mapped to the estimated probabilities by the logistic function. As depicted in the plots below, the maximum-likelihood parameters provide a much better fit to the data than the other two choices. The corresponding conditional probability that an individual is a man given their height h is

$$p_{\alpha_{\mathrm{ML}},\beta_{\mathrm{ML}}}(h) := \frac{1}{1 + \exp(-0.29h + 48.6)}. \tag{12.307}$$

The function for the estimated probability is similar to the one obtained from the generative model in Example 6.3, which is depicted in Figure 6.5, but not the same (see equation (6.30)).

Figure 12.23 shows the result of performing automatic diagnosis of Alzheimer's disease using a logistic-regression model. As in Section 6.5, our goal is to distinguish between healthy subjects and patients with Alzheimer's using the normalized volumes of their hippocampus and entorhinal cortex. The figure shows the two-dimensional logit plane corresponding to the maximum-likelihood estimate of the model parameters. The logits are mapped to the unit interval by the logistic function. The contour lines of the resulting probability estimate are linear and orthogonal to the linear coefficients β_{ML}. The logistic-regression model is very similar to the linear discriminant analysis model in Figure 6.7. However,

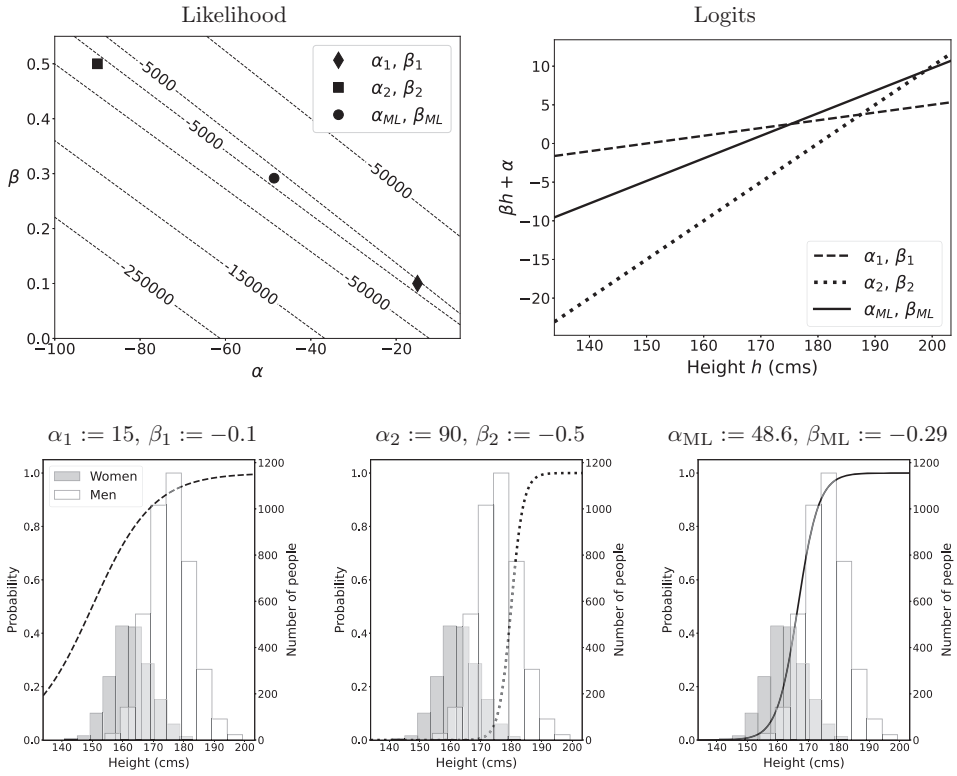


Figure 12.22 Logistic-regression model for sex. The top left graph shows the log-likelihood function of the logistic-regression model in Example 12.35. The top right plot shows the affine logits corresponding to three different choices of the parameters, including the maximum-likelihood values that maximize the log-likelihood. The plots below show the estimated conditional probability that an individual is a man given their height for the three choices of parameters. The maximum-likelihood parameters provide the best discrimination between the histograms of men (gray) and women (white) heights.

the logistic-regression model directly approximates the conditional distribution of the labels given the features, instead of modeling the feature prior and the conditional distribution of the features given the labels. In Section 12.9 we provide a detailed evaluation of the logistic-regression classifier for this dataset.

Just like linear-regression models (see Section 12.2), logistic-regression models tend to overfit when the number of features is close to the number of training data, or when there is multicollinearity among the features. This can be alleviated using an additive regularization term to penalize the ℓ_2 norm of the model coefficients,

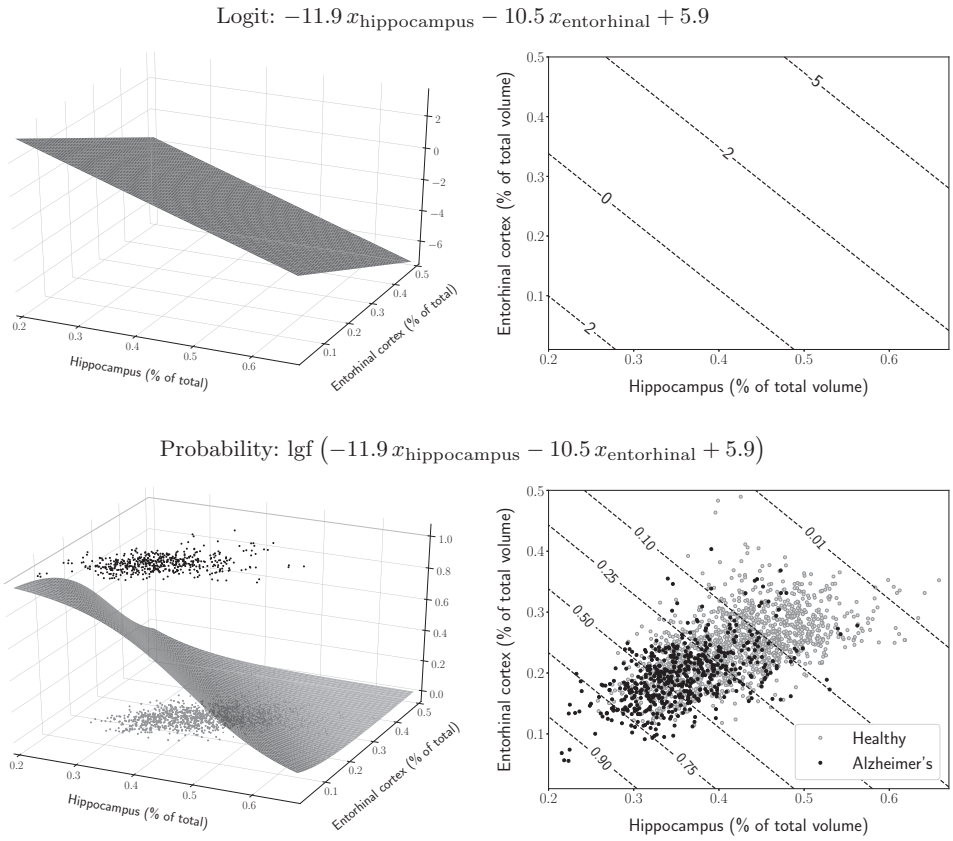


Figure 12.23 Automatic diagnosis of Alzheimer's via logistic regression. The top row shows the 3D plot and contour lines of the logits of a logistic-regression model to perform diagnosis of Alzheimer's from normalized volumes of the hippocampus and the entorhinal cortex. The logistic function maps the logits to the corresponding probability estimates. The bottom row shows a 3D plot and contour lines of these probabilities, superposed onto a scatterplot of the features corresponding to the healthy subjects (in light gray) and Alzheimer's patients (in black).

as in ridge regression (see Section 12.3). The resulting cost function or loss is

$$-\log \mathcal{L}_{XY}(\alpha, \beta) + \lambda \, ||\beta||_2^2 \,, \tag{12.308}$$

where $\log \mathcal{L}_{XY}$ is the log likelihood derived in Theorem 12.34 and $\lambda > 0$ is a regularization parameter, which can be set using validation data. Example 12.38 illustrates the benefits of this approach for softmax regression, a multi-class generalization of logistic regression presented in Section 12.6. Alternatively, the ℓ_2 norm can be replaced by the ℓ_1 norm to promote sparse coefficients, as in the lasso (see Section 12.4).

12.6 Softmax Regression

As explained in Section 12.5, to perform logistic regression we compute an affine function of the input features, and then map the result to a probability by applying a logistic function, which is essentially a normalized exponential. In this section, we generalize this approach to classification problems with more than two classes.

Let us represent the features in a classification problem as a d-dimensional random vector $\tilde{x}$, and the corresponding class label as a random variable $\tilde{y}$ with c possible values, which we arbitrarily select to be the integers from one to c. We define c different affine functions of the input feature vector $x \in \mathbb{R}^d$, one for each class:

$$\beta_k^T x + \alpha_k, \qquad 1 \leq k \leq c. \tag{12.309}$$

As mentioned in Section 12.5, we cannot directly use these affine functions to model the conditional probability that $\tilde{y} = k$ given $\tilde{x} = x$, because they are not between zero and one for most possible values of x. Instead, we apply an exponential function to each of them, and normalize the result so that all the conditional probabilities sum to one. This operation is known as the *softmax*, because it tends to push the input with the largest magnitude towards one, and the rest towards zero (see Example 12.37). The resulting softmax-regression model for the conditional probability is

$$\mathrm{P}\left(\tilde{y} = k \,|\, \tilde{x} = x\right) = \frac{\exp\left(\beta_k^T x + \alpha_k\right)}{\sum_{l=1}^{c} \exp\left(\beta_l^T x + \alpha_l\right)}, \qquad 1 \leq k \leq c. \tag{12.310}$$

The softmax-regression model is overparametrized because the probabilities sum to one. The same expressiveness can be obtained with only $c-1$ coefficient vectors and intercepts. A way to achieve this is to set β_c and α_c to zero, which actually yields the two-class logistic-regression model defined in Section 12.5 for $c := 2$.

In order to fit a softmax-regression model from data, we maximize the corresponding log-likelihood, presented in the following theorem. We omit the proof, as it is identical to that of Theorem 12.34.

Theorem 12.36 (Softmax-regression likelihood)**.** *Let (x_1, y_1), (x_2, y_2), ..., (x_n, y_n) be a dataset formed by n pairs of a label $y_i \in \{1, ..., c\}$ (where $c \geq 2$ is the number of classes) and a corresponding feature vector $x_i \in \mathbb{R}^d$. We model the data as a realization from a joint distribution of feature vectors $\tilde{x}_1$, ..., $\tilde{x}_n$ and labels $\tilde{y}_1$, ..., $\tilde{y}_n$. We define the likelihood of the logistic-regression model*

$$p_\Theta\left(x_i\right)_k := \frac{\exp\left(\beta_k^T x + \alpha_k\right)}{\sum_{l=1}^{c} \exp\left(\beta_l^T x + \alpha_l\right)}, \qquad 1 \leq k \leq c, \tag{12.311}$$

$$\Theta := \left\{\beta_1, ..., \beta_c, \alpha_1, ..., \alpha_c\right\}, \tag{12.312}$$

as the conditional pmf of the observed labels given the observed features. If the labels are conditionally independent given the features, and each label $\tilde{y}_i$ is conditionally independent from the remaining feature vectors $\{\tilde{x}_l\}_{l \neq i}$ given $\tilde{x}_i$, then

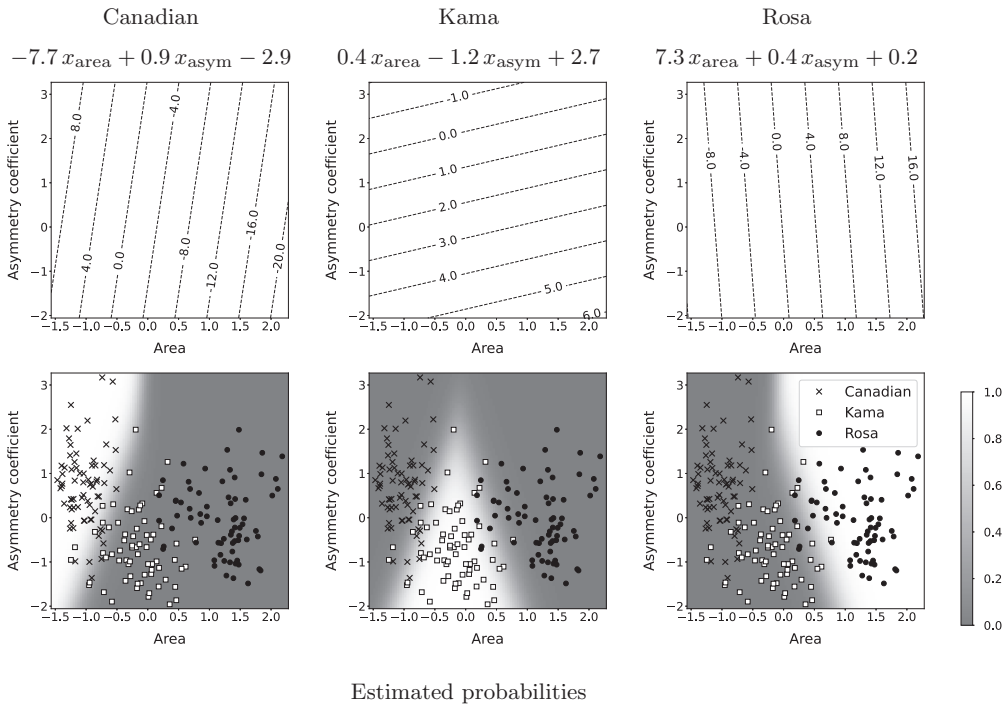


Figure 12.24 Classification of wheat seeds via softmax regression. Visualization of a softmax-regression model to classify wheat seeds into three varieties according to their area and asymmetric coefficient. The top row depicts the affine functions in the model for each class. The affine functions are mapped via the softmax operation to the estimated probabilities, shown in the bottom row superposed onto the training data. The probabilities discriminate effectively between the three wheat varieties.

the likelihood equals

$$\mathcal{L}_{XY}(\Theta) = \prod_{k=1}^{c} \prod_{\{i:y_i=k\}} p_\Theta\left(x_i\right)_k, \tag{12.313}$$

and the corresponding log-likelihood is

$$\log \mathcal{L}_{XY}(\Theta) = \sum_{k=1}^{c} \sum_{\{i:y_i=k\}} \log p_\Theta\left(x_i\right)_k. \tag{12.314}$$

Example 12.37 (Seed classification)**.** We consider the problem of classifying wheat seeds based on their surface area and asymmetric coefficient, using data from Dataset 21. The three classes are the wheat varieties Kama, Rosa and Canadian. We center and normalize the features so that each has zero sample mean and

unit sample variance, and then fit a softmax-regression model. The top row in Figure 12.24 shows the three affine functions corresponding to maximum-likelihood estimates of the coefficient vectors and intercepts. A seed with area $x_{\text{area}} := -1$ and asymmetric coefficient $x_{\text{asym}} := 2$ is assigned the following three values by the affine functions

$$\begin{bmatrix} \text{Canadian} \\ \text{Kama} \\ \text{Rosa} \end{bmatrix} = \begin{bmatrix} -7.7\, x_{\text{area}} + 0.9\, x_{\text{asym}} - 2.9 \\ 0.4\, x_{\text{area}} - 1.2\, x_{\text{asym}} + 2.7 \\ 7.3\, x_{\text{area}} + 0.4\, x_{\text{asym}} + 0.2 \end{bmatrix} = \begin{bmatrix} 6.6 \\ -0.1 \\ -6.3 \end{bmatrix}. \tag{12.315}$$

The exponential function amplifies the positive values and maps negative values to small positive numbers,

$$\exp(6.6) = 735.095, \quad \exp(-0.1) = 0.905, \quad \exp(-6.3) = 0.002. \tag{12.316}$$

Normalizing these values, dividing by their sum, yields the softmax probability estimates

$$\begin{bmatrix} \mathrm{P}(\text{Canadian} \,|\, x_{\text{area}}, x_{\text{asym}}) \\ \mathrm{P}(\text{Kama} \,|\, x_{\text{area}}, x_{\text{asym}}) \\ \mathrm{P}(\text{Rosa} \,|\, x_{\text{area}}, x_{\text{asym}}) \end{bmatrix} = \begin{bmatrix} \frac{735.095}{735.095+0.905+0.002} \\ \frac{0.905}{735.095+0.905+0.002} \\ \frac{0.002}{735.095+0.905+0.002} \end{bmatrix} = \begin{bmatrix} 0.999 \\ 0.001 \\ 0.000 \end{bmatrix}. \tag{12.317}$$

The softmax operation pushes the largest affine value towards one and the rest towards zero. It can thus be interpreted as a *soft maximum*, which motivates its name. The bottom row of Figure 12.24 shows the estimated probabilities as a function of the two features for the three varieties of wheat. The model is able to effectively discriminate between the three varieties.

Example 12.38 (Digit classification)**.** In this example we apply softmax regression to classify 28×28 images of handwritten digits from 0 to 9, extracted from Dataset 23. We separate the dataset into a training set and a test set, each containing 35,000 examples. The number of features is equal to the number of pixels in each image (784). Consequently, the number of parameters in the softmax-regression model is $784 \cdot 10 + 10 = 7850$. We fit these parameters maximizing the log-likelihood function from Theorem 12.36. The training error of the resulting model is just 4.3%, but the test error is 10.4%. The model overfits. In order to gain insight into why, we analyze the model coefficients.

According to the model, the probability that an image $x \in \mathbb{R}^{784}$ corresponds to a digit k for $k \in \{0, 1, ..., 9\}$ equals

$$\mathrm{P}(\tilde{y} = k \,|\, \tilde{x} = x) = \frac{\exp(\beta_k^T x + \alpha_k)}{\sum_{l=0}^{9} \exp(\beta_l^T x + \alpha_l)}, \tag{12.318}$$

where $\beta_k \in \mathbb{R}^{784}$ and $\alpha_k \in \mathbb{R}$ are the coefficient vector and intercept associated with digit k. The model takes inner products between the image and the ten different coefficient vectors, sums the intercepts, and then maps the resulting values to the estimated probabilities via the softmax operation.

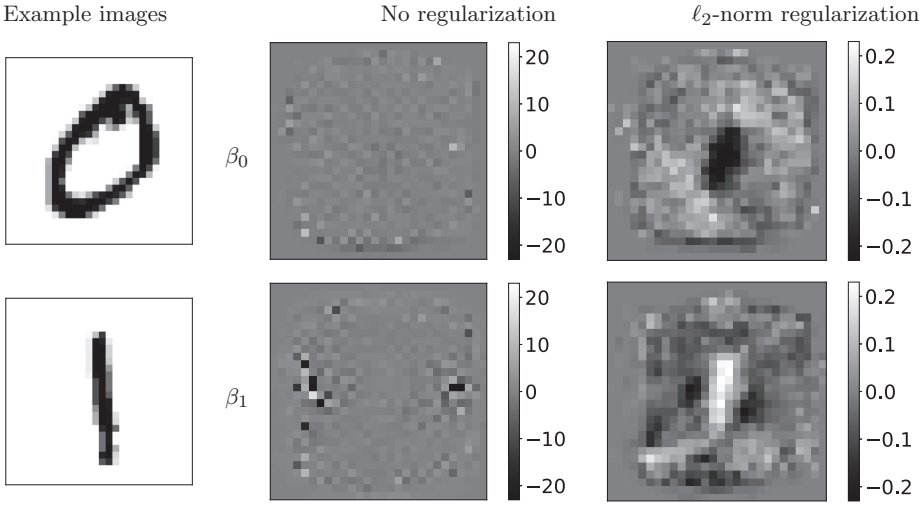


Figure 12.25 Regularization mitigates overfitting in softmax regression. The left column shows two images of handwritten digits from Dataset 23. The central column shows the coefficient vectors corresponding to zero (top) and one (bottom), obtained by fitting a softmax-regression model via maximum likelihood. The model overfits the training data, which results in fluctuating noisy coefficients. The right column shows coefficients obtained by minimizing the cost function in equation (12.319), which provide a worse fit to the training data but generalize better to test data. The regularized coefficients capture meaningful geometric features of both digits.

To determine whether the coefficient vectors capture meaningful discriminative structure, we plot β_0 and β_1 in the center column of Figure 12.25, reshaping them into 28×28 images to visualize what coefficient corresponds to each pixel. Ideally, β_0 should be such that $\beta_0^T x$ is very large if x depicts a zero, and small otherwise. This can be achieved by choosing negative coefficients for pixels that are likely to have small values in zero images (e.g. at the center of the image), and positive coefficients for pixels that are likely to be nonzero (e.g. around the center). However, the estimated coefficients look nothing like this! They take very large negative and positive values around the edges. This is a symptom of overfitting: the noisy coefficients achieve a higher likelihood on the training set by fitting spurious patterns in the edges, but they hinder generalization to the test data, which does not contain the same patterns.

Section 12.2.2 reports a similar overfitting phenomenon in linear regression, which also results in large, fluctuating coefficients. Section 12.3 shows that regularization can effectively control the amplitude of the model coefficients and mitigate overfitting. Motivated by this, we modify our cost function to combine the log likelihood defined in Theorem 12.36 and an additive ℓ_2-norm regulariza-

tion term

$$-\log \mathcal{L}_{XY}(\alpha, \beta) + \lambda \sum_{k=0}^{9} ||\beta_k||_2^2 , \tag{12.319}$$

setting the regularization parameter λ equal to 50 (in practice, we would select it based on the model performance on a validation set). The model obtained by minimizing this cost function has a training error rate equal to 6.2% and a test error equal to 7.8%. The regularized model does not fit the training data as well as our original model, but it generalizes better. This is reflected in the model coefficients, shown in the rightmost column of Figure 12.25. The coefficients look much more reasonable than those of the unregularized model, clearly capturing meaningful geometric structure associated with each digit.

12.7 Tree-Based Models

Tree-based models are popular methods for regression and classification. Sections 12.7.1 and 12.7.2 present regression and classification trees, respectively. Section 12.7.3 describes several strategies to build complex nonlinear models by combining multiple trees.

12.7.1 Regression trees

The linear regression models described in Section 12.1 are very useful, but they have a fundamental limitation: they cannot encode any *nonlinear* dependence between the response and the features. To illustrate this, let us attempt to model the temperature in Manhattan (Kansas) as a function of the day of the year $1 \leq x_{\text{day}} \leq 365$ ($x_{\text{day}} = 1$ is the first of January) and the hour of the day $0 \leq x_{\text{hour}} \leq 23$ ($x_{\text{hour}} = 0$ is midnight). The training data consist of temperatures from 2015, extracted from Dataset 9, annotated with the corresponding day and hour.

The OLS estimator of the temperature (as defined in Theorem 12.7) given the hour x_{hour} and the day x_{day} is

$$\ell_{\text{OLS}}(x_{\text{hour}}, x_{\text{day}}) = 0.25 x_{\text{hour}} + 0.03 x_{\text{day}} + 5.85. \tag{12.320}$$

According to the linear model, if we fix the day x_{day}, then the temperature grows proportionally to the hour with a factor of proportionality equal to the linear coefficient 0.25. As shown in the top left graph of Figure 12.26 the temperature does indeed increase each day up until around noon ($x_{\text{hour}} \approx 12$). However, in the afternoon and evening ($x_{\text{hour}} \geq 12$) the temperature tends to decrease instead. Similarly, for each fixed hour, the temperature increases as x_{day} increases, but only until around the middle of the summer ($x_{\text{day}} \approx 200$); afterwards it decreases during the fall and winter. The linear model (depicted in the bottom left graph of Figure 12.26) is completely unable to capture this nonlinear structure, so its training and test errors are very large.

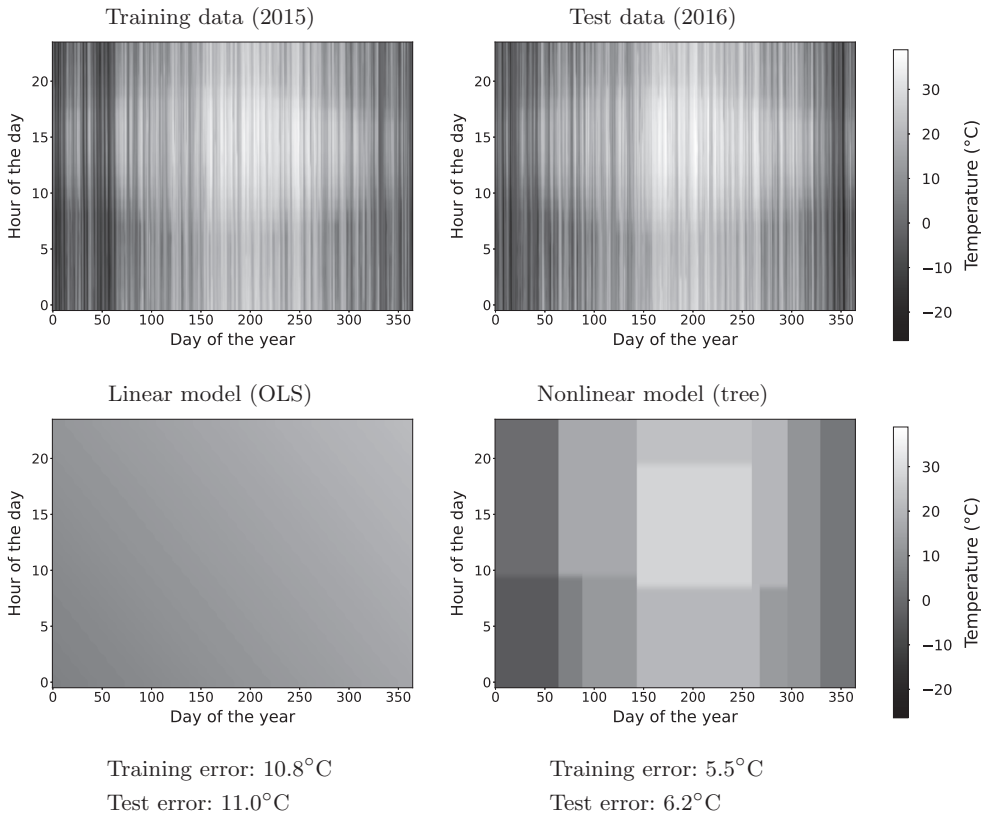


Figure 12.26 Linear vs. nonlinear temperature estimation. The top row shows hourly temperatures for 2015 (left) and 2016 (right) in Manhattan (Kansas) as a function of the day of the year $1 \leq x_{\text{day}} \leq 365$ and the hour of the day $0 \leq x_{\text{hour}} \leq 23$. The bottom row shows the linear OLS model (left) and the nonlinear model (right) corresponding to the regression tree in Figure 12.27, both fitted using the 2015 data. The training and test root average squared error, reported below each graph, indicate that the nonlinear model captures the structure of the data much more effectively.

A simple way to build a nonlinear regression model is to partition the feature space into m non-overlapping regions, and assign a fixed constant estimate $\alpha_r \in \mathbb{R}$ to each region $R_r \subseteq \mathbb{R}^d$ for $1 \leq r \leq m$ (where d is the number of features). If a feature vector x belongs to R_r the estimate is

$$n\ell_{\text{pc}}(x) := \alpha_r. \tag{12.321}$$

The resulting estimator is therefore *piecewise constant*. The following theorem establishes that for a fixed partition, the optimal piecewise-constant estimator in terms of average squared error can be obtained by averaging the responses of the training data in each region.

Theorem 12.39 (Piecewise-constant regression)**.** *Let (x_1, y_1), (x_2, y_2), ..., (x_n, y_n) be a dataset formed by n pairs of a response y_i and a corresponding vector x_i containing d features. For any collection of sets R_1, ..., R_m forming a partition of the feature space (so that each feature vector x_i belongs to only one of the sets), we define the piecewise-constant estimator*

$$n\ell_{\text{pc}}(x) := \alpha_r \qquad \text{if } x \in R_r. \tag{12.322}$$

The value of α_1, ..., α_m that minimizes the residual sum of squares (RSS)

$$\text{RSS} := \sum_{i=1}^{n} \left(y_i - n\ell_{\text{pc}}(x_i)\right)^2 \tag{12.323}$$

is

$$\alpha_r = \frac{1}{n_r} \sum_{\{i:x_i \in R_r\}} y_i, \quad 1 \le r \le m, \tag{12.324}$$

where $n_r := |\{i : x_i \in R_r\}|$ is the number of data points in R_r. In words, the estimate in R_r is equal to the sample mean of the responses associated with the examples whose feature vectors belong to R_r.

Proof Since the regions are disjoint and cover all of $\mathbb{R}^d$, the total RSS decouples into the sum of RSSs in each region,

$$\text{RSS} = \sum_{r=1}^{m} \sum_{\{i:x_i \in R_r\}} \left(y_i - \alpha_r\right)^2. \tag{12.325}$$

Consequently, we can choose the estimates to separately minimize each of the individual RSSs. By Theorem 7.31, the minimum is achieved by simply averaging the data in the corresponding region,

$$\arg\min_{\alpha_r} \text{RSS} = \arg\min_{\alpha_r} \sum_{\{i:x_i \in R_r\}} \left(y_i - \alpha_r\right)^2 \tag{12.326}$$

$$= \frac{1}{n_r} \sum_{\{i:x_i \in R_r\}} y_i. \tag{12.327}$$

■

Regression trees provide a framework to define piecewise-constant estimators that can be learned efficiently from data. The regions in a regression tree are hyperrectangles formed by Cartesian products of intervals, which we denote by

$$R_r := \left(a_r^{[1]}, b_r^{[1]}\right] \times \left(a_r^{[2]}, b_r^{[2]}\right] \times \cdots \times \left(a_r^{[d]}, b_r^{[d]}\right] \tag{12.328}$$

$$= \bigtimes_{j=1}^{d} \left(a_r^{[j]}, b_r^{[j]}\right], \tag{12.329}$$

where $a_r^{[j]} < b_r^{[j]}$ for $1 \le j \le d$ and $1 \le r \le m$. These regions are associated with the leaves (terminal nodes with no descendants) of a binary tree. Each leaf is assigned a constant estimate, obtained by averaging the response of the training

data points within the corresponding region, as prescribed by Theorem 12.39. Figure 12.27 shows the regression tree corresponding to the nonlinear model depicted in the bottom right graph of Figure 12.26. The regression-tree model achieves a much better fit to the training data than the linear OLS model, because it is able to capture nonlinear patterns such as annual seasonality, and days being warmer than nights. It also generalizes better to a test set consisting of temperatures from 2016.

The regions associated with a regression tree are defined recursively. We illustrate this using the regression tree in Figure 12.27, which represents the nonlinear temperature model in Figure 12.26. We begin by assigning the whole feature space to the root node at the top of the tree. For our example this is $[0, 23] \times [1, 365]$, because the hour is between 0 and 23, and the day between 1 and 365. In a regression tree, a node is either a parent node with two children, or a leaf with no descendants. The regions assigned to the two children are obtained by splitting the parent's region *according to a single feature*. In our example, the root-node region is split according to the day feature. The left child is assigned $[10, 23] \times [0, 65]$ (winter), and the right child is assigned $[10, 23] \times [66, 365]$ (spring, summer and fall). The left child has two children. Their regions are obtained by dividing $[10, 23] \times [0, 65]$ according to the hour feature, yielding $[0, 9] \times [1, 65]$ (winter nights) and $[10, 23] \times [0, 65]$ (winter days). These are leaf nodes, so their regions are part of the piecewise-constant model associated with the tree. The temperature estimates assigned to each leaf are -4.3°C and 0.8°C respectively, obtained by averaging the temperature in 2015 within the corresponding regions, following Theorem 12.39.

No matter how many bifurcations there are in a regression tree, the regions associated with the leaf nodes are always guaranteed to form a partition of the feature space (see Exercise 12.14). Consequently, the tree provides a single estimate associated with any specific feature vector, which can be retrieved by traversing the tree from the root to find the corresponding leaf. For instance, at the time of writing it is 3 am in Kansas and the date is August 19, so $x_{\text{hour}} := 3$ and $x_{\text{day}} := 251$. To obtain the corresponding temperature estimate $n\ell_{\text{tree}}(x_{\text{hour}}, x_{\text{day}}) = 20.5$°C, we start at the root and follow the appropriate branches (represented by the thick black lines in Figure 12.27):

$$x_{\text{day}} > 65 \rightarrow x_{\text{day}} < 296 \rightarrow x_{\text{day}} > 144 \rightarrow x_{\text{hour}} \leq 8 \rightarrow x_{\text{day}} \leq 268 \rightarrow 20.5\text{°C}.$$

According to the internet, the temperature in Manhattan (Kansas) is actually 22°C, so the estimate is not too bad! The model provides an interpretable explanation of the estimate in terms of the different features and the corresponding thresholds it depends on: the temperature estimate is 20.5°C because $65 < x_{\text{day}} \leq 268$ and $x_{\text{hour}} \leq 8$ (i.e. it's a summer night). Interpretability is an important advantage of regression trees with respect to more complicated nonlinear models such as the tree ensembles in Section 12.7.3 or the neural networks in Section 12.8.

The splits at each bifurcation in a regression tree should be chosen to fit the training data. For each split, we need to determine what feature to use and how to

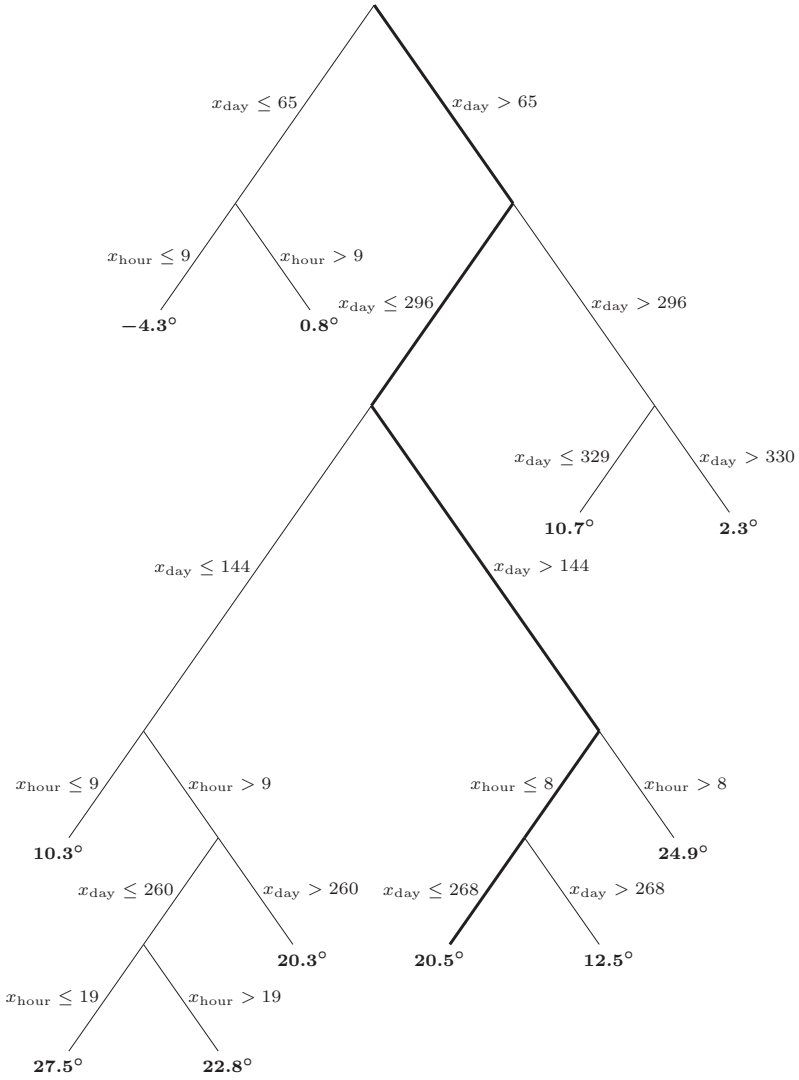


Figure 12.27 Regression tree for temperature estimation. The diagram shows the regression tree associated with the nonlinear temperature model depicted in the bottom right graph of Figure 12.26. Every node represents a region of the feature space. Each bifurcation in the tree represents a split in the parent-node region indicated on the corresponding branches, and each leaf node is associated to a constant temperature estimate for the corresponding region. The thick black lines show the path from the root to the leaf node corresponding to August 19 at 3 am ($x_{\text{day}} := 251$ and $x_{\text{hour}} := 3$).

threshold it. Unfortunately, the number of possible trees quickly explodes as their depth increases. This is yet another manifestation of the curse of dimensionality,

discussed in Section 4.7. For simplicity let us assume that all nodes at depth w are leaves. Then there are $b := 2^{w-1}$ bifurcations in the tree (each associated with a non-leaf node). If the number of features equals d and there are t possible thresholds, then we have $(dt)^b$ different trees to choose from! For $d := 10$ and $t := 100$, which are rather modest values, the number of possible depth-4 trees is astronomical (more than 10^{24}). It is therefore impossible to consider all possible regression trees when fitting most datasets. Instead, regression trees are grown iteratively, adding nodes one by one to optimize the residual sum of squares in a greedy fashion. This technique is known as recursive binary splitting.

Definition 12.40 (Recursive binary splitting)**.** *Let (x_1, y_1), (x_2, y_2), ..., (x_n, y_n) be a dataset formed by n pairs of a response y_i and a corresponding vector x_i containing d features. Consider a partition of the feature space associated with the leaf nodes of a binary regression tree, R_1, ..., R_m, where*

$$R_r := \bigtimes_{j=1}^{k} \left(a_r^{[j]}, b_r^{[j]} \right], \qquad a_r^{[j]} < b_r^{[j]} \quad \text{for } 1 \leq j \leq d,\ 1 \leq r \leq m. \tag{12.330}$$

Each region R_r can be split by thresholding the sth feature using a threshold t. We denote the resulting two regions by

$$\mathcal{A}_r(s,t) := \bigtimes_{j=1}^{k} \left(a_r^{[j]}, c^{[j]} \right], \qquad c^{[s]} := t, \quad c_r^{[j]} := b_r^{[j]} \quad \text{for } j \neq s, \tag{12.331}$$

$$\mathcal{B}_r(s,t) := \bigtimes_{j=1}^{k} \left(d^{[j]}, b_r^{[j]} \right], \qquad d_r^{[s]} := t, \quad d^{[j]} := a_r^{[j]} \quad \text{for } j \neq s. \tag{12.332}$$

$\mathcal{A}_r(s,t)$ contains every data point $x \in R_r$ such that $x[s] \leq t$, and $\mathcal{B}_r(s,t)$ contains every data point $x \in R_r$ such that $x[s] > t$.

The RSS of the tree model equals

$$\text{RSS} = \sum_{r=1}^{m} \sum_{\{i : x_i \in R_r\}} (y_i - \alpha_r)^2, \tag{12.333}$$

where we set

$$\alpha_r := \frac{1}{n_r} \sum_{\{i : x_i \in R_r\}} y_i, \quad 1 \leq r \leq m, \tag{12.334}$$

for $n_r := |\{i : x_i \in R_r\}|$ as suggested by Theorem 12.39. For each possible split, we define

$$\alpha_{\mathcal{A}_r(s,t)} := \frac{1}{n_{\mathcal{A}_r(s,t)}} \sum_{\{i : x_i \in \mathcal{A}_r(s,t)\}} y_i, \tag{12.335}$$

$$\alpha_{\mathcal{B}_r(s,t)} := \frac{1}{n_{\mathcal{B}_r(s,t)}} \sum_{\{i : x_i \in \mathcal{B}_r(s,t)\}} y_i, \tag{12.336}$$

where $n_{\mathcal{A}_r(s,t)} := |\{i : x_i \in \mathcal{A}_r(s,t)\}|$ and $n_{\mathcal{B}_r(s,t)} := |\{i : x_i \in \mathcal{B}_r(s,t)\}|$ are the

number of elements in $\mathcal{A}_r(s,t)$ *and* $\mathcal{B}_r(s,t)$*. The RSS decrease after the split equals*

$$\triangle \,\mathrm{RSS}(r,s,t) = \sum_{\{i:x_i \in R_r\}}^{n} (y_i - \alpha_r)^2 - \sum_{\{i:x_i \in \mathcal{A}_r(s,t)\}}^{n} \left(y_i - \alpha_{\mathcal{A}_r(s,t)}\right)^2 - \sum_{\{i:x_i \in \mathcal{B}_r(s,t)\}}^{n} \left(y_i - \alpha_{\mathcal{B}_r(s,t)}\right)^2, \tag{12.337}$$

which is always nonnegative (see Exercise 12.14). The optimal split can be found by iterating over the regions, features and possible thresholds. For each region and feature, the set of possible thresholds is finite: it equals the observed values of the feature in the training examples that belong to the region. Let

$$(r^*, s^*, t^*) := \arg\max_{r,s,t} \triangle \,\mathrm{RSS}(r,s,t) \tag{12.338}$$

be the region, feature and threshold that achieve the greatest RSS decrease. We replace R_{r^*} *by* $\mathcal{A}_{r^*}(s^*,t^*)$ *and* $\mathcal{B}_{r^*}(s^*,t^*)$ *in the original partition to complete the split and obtain a new partition with* $m+1$ *regions.*

To fit a regression tree using a training dataset, we begin with a single region equal to the whole feature space. Then we apply recursive binary splitting, as described in Definition 12.40 to grow the tree until it has a desired depth or number of leaves, or until another stopping criterion is satisfied. Figure 12.28 illustrates the splitting procedure for our temperature example. The top left diagram shows a 4-leaf tree. The top right diagram shows the four regions associated with the leaves in the tree, which roughly correspond to the four seasons (winter, spring, summer and fall). The middle row shows the RSS decrease achieved by splitting each region by thresholding the hour (left) and day (right) features. The highest decrease is achieved by splitting Region 3 according to the hour feature with a threshold equal to 8. This creates two new regions, roughly corresponding to night ($x_{\text{hour}} \leq 8$) and day ($x_{\text{hour}} > 8$). The resulting tree with five leaves, and the corresponding regions, are depicted in the bottom row of Figure 12.28.

The top graph in Figure 12.29 shows the training and test error of regression trees built via recursive binary splitting, as described in Definition 12.40, for our temperature-estimation example. The training data are temperatures measured in Manhattan (Kansas) in 2015 and the test data are temperatures from 2016. As the number of leaves in the tree increases, the fit to the training data improves and the training error decreases. At first, this is accompanied by a decrease in the test error, indicating that the model learns patterns that generalize to the test set. However, eventually (when there are more than 11 leaves) the test error starts to increase, indicating that beyond this point the model overfits spurious structure in the training set that is not present in the test set.

The second row of Figure 12.29 shows the learned regions for the 11-leaf and 15-leaf models. Notice that the 15-leaf model includes the region $[14, 41] \times [10, 23]$ (after 10 am from mid-January to mid-February), which has an average temperature that is higher than the region $[42, 65] \times [10, 23]$ (after 10 am from mid-

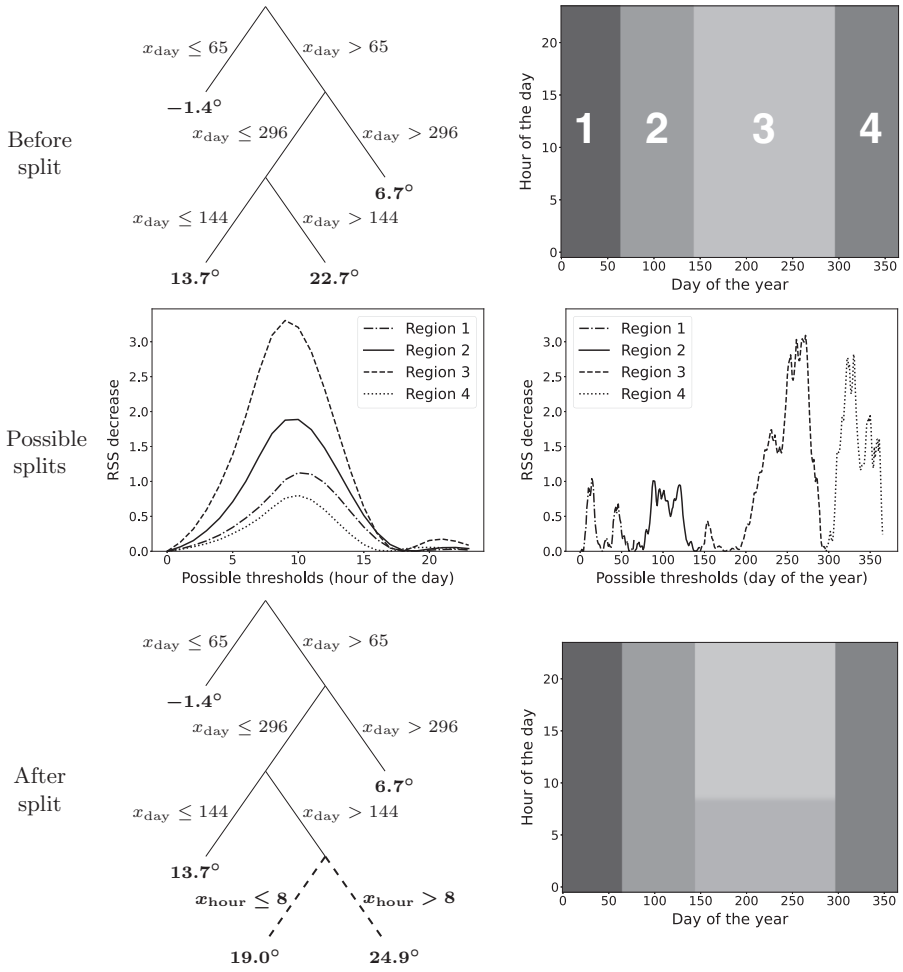


Figure 12.28 Growing a regression tree via recursive binary splitting. The top left diagram shows a tree with four leaves. The top right diagram shows the four regions associated with the leaves in the tree. The middle row shows the RSS decrease achieved by splitting each region by thresholding the hour (left) and day (right) features. The highest decrease is achieved by splitting Region 3 according to the hour feature with a threshold equal to 8. The resulting tree with five leaves (left) and the corresponding regions (right) are depicted in the bottom row.

February to mid-March). This is surprising: one would not expect temperatures to decrease from February to March. Indeed, the pattern is not present in the test data (compare the third and fourth rows of Figure 12.29), so incorporating this region increases the test error of the model. Such spurious patterns arise due to noisy fluctuations of the temperature when we make the regions small enough

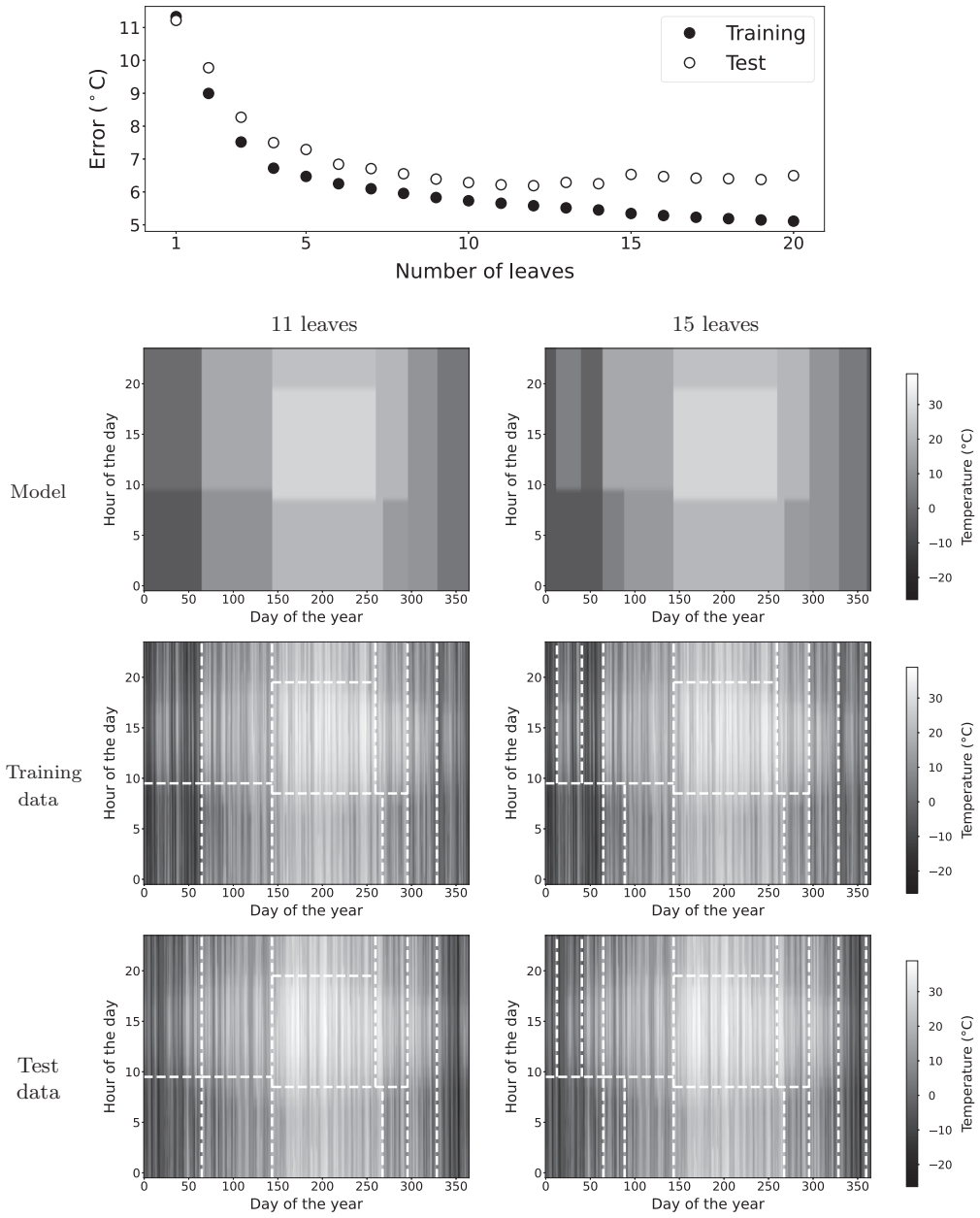


Figure 12.29 Overfitting and generalization in regression trees. The graph at the top shows the training and test root average squared error of regression trees with different number of leaves. The trees are built via recursive binary splitting, as described in Definition 12.40, using the training data in Figure 12.26. We observe overfitting when the number of leaves is greater than 11: the training error continues to decrease, but the test error increases. The second row shows the learned regions for the 11-leaf and 15-leaf models. The third and fourth rows show the regions superposed onto the training and test data, respectively, which reveals that the 11-leaf model generalizes better than the 15-leaf model, because the latter overfits spurious structure in the training data (compare the regions $[14, 41] \times [10, 23]$ and $[42, 65] \times [10, 23]$).

(with respect to the number of training data), leading to overfitting. This can be avoided by using a validation dataset to decide when to stop the recursive splitting procedure.

12.7.2 Classification trees

Classification trees are classification models with the same structure as regression trees. Let us represent the features in a classification problem as a d-dimensional random vector $\tilde{x}$ and the corresponding class label as a random variable $\tilde{y}$ with c possible values, which we arbitrarily select to be the integers from one to c. Classification trees allow us to learn a nonlinear estimator of the conditional probability of the label given the features,

$$\mathrm{P}\left(\tilde{y} = k \,|\, \tilde{x} = x\right) = p_{\mathcal{R},\Theta}\left(x_i\right)_k := \theta_r[k], \qquad \text{if } x \in R_r, \tag{12.339}$$

which is piecewise constant because the estimate is constant within each region. $\mathcal{R} := \{R_1, ..., R_m\}$ is the partition of the feature space containing all the regions, and Θ contains the corresponding probability estimates.

Let (x_1, y_1), (x_2, y_2), ..., (x_n, y_n) be a dataset formed by n pairs of a label $y_i \in \{1, ..., c\}$ and a corresponding feature vector $x_i \in \mathbb{R}^d$. We model the data as a realization from a joint distribution of feature vectors $\tilde{x}_1$, ..., $\tilde{x}_n$ and labels $\tilde{y}_1$, ..., $\tilde{y}_n$. If the labels are conditionally independent given the features, and each label $\tilde{y}_i$ is conditionally independent from the remaining feature vectors $\{\tilde{x}_l\}_{l \neq i}$ given $\tilde{x}_i$, then the log-likelihood of the model equals

$$\log \mathcal{L}_{XY}(\mathcal{R}, \Theta) = \sum_{k=1}^{c} \sum_{\{i:y_i=k\}} \log p_{\mathcal{R},\Theta}\left(x_i\right)_k . \tag{12.340}$$

We omit the derivation, as it is identical to the one in the proof of Theorem 12.34 (see also Theorem 12.36). The following theorem shows that if we fix the partition $\mathcal{R}$, then maximizing the log-likelihood with respect to the estimated probabilities Θ is very simple: we just compute the fraction of data in each region associated with each label.

Theorem 12.41 (Piecewise-constant classification)**.** *For a fixed partition $\mathcal{R}$, the log-likelihood function in equation* (12.340) *is maximized by setting*

$$\theta_r[k] = \frac{n_r^{[k]}}{n_r}, \quad 1 \leq r \leq m, 1 \leq k \leq c, \tag{12.341}$$

where $n_r := |\{i : x_i \in R_r\}|$ is the number of data points in R_r and $n_r^{[k]} := |\{i : x_i \in R_r, y_i = k\}|$ is the number of data points in R_r with labels equal to k. In words, the probability estimate for each region R_r is equal to the fraction of labels equal to k in that region.

Proof The regions form a partition, so they are disjoint and cover all of $\mathbb{R}^d$. Consequently, the log-likelihood decouples into the sum of log-likelihoods in each

region

$$\log \mathcal{L}_{XY}(\mathcal{R}, \Theta) = \sum_{r=1}^{m} \sum_{k=1}^{c} \sum_{\{i: x_i \in R_r, y_i = k\}} \log p_{\mathcal{R},\Theta}(x_i)_k \tag{12.342}$$

$$= \sum_{r=1}^{m} \sum_{k=1}^{c} n_r^{[k]} \log \theta_r[k], \tag{12.343}$$

where we have used the fact that there is a single probability estimate θ_r assigned to each region R_r. Consequently, we can choose these estimates to separately minimize each of the individual log-likelihoods. For a specific region R_r

$$\arg\max_{\theta_r} \log \mathcal{L}_{XY}(\mathcal{R}, \Theta) = \arg\max_{\theta_r} \sum_{k=1}^{c} n_r^{[k]} \log \theta_r[k], \tag{12.344}$$

which is equivalent to maximizing the log-likelihood of a multinoulli parametric model with parameter θ_r using only the data points with feature vectors in R_r. By Exercise 2.11 (see also Example 2.26), for $1 \leq k \leq c$, the kth entry of the maximum-likelihood estimator is equal to the fraction of data in R_r with labels equal to k, which yields (12.341). ■

Figure 12.30 shows a classification tree for the classification problem in Example 12.37, where the goal is to identify wheat seeds based on two features: surface area and asymmetric coefficient. Each node in the tree corresponds to a region of the feature space, exactly as in regression trees (see Section 12.7.1). The only difference is that the estimate associated with the leaf-node regions is a probability vector (instead of a single number, as in regression). Since the leaf nodes form a partition of the feature space (see Exercise 12.14), the tree maps every possible feature vector to a single probability estimate, which can be determined by traversing the tree from the root node. To illustrate this, we use the classification tree in Figure 12.30 to classify a wheat seed with surface area $x_{\text{area}} := 0.5$ and asymmetric coefficient $x_{\text{asym}} := 0$. The path from the root to the appropriate leaf, marked by thick black lines on the diagram, is

$$x_{\text{area}} > 0.18 \to x_{\text{area}} \leq 0.78 \to x_{\text{asym}} > -0.83 \to p_{\mathcal{R},\Theta}(x_{\text{area}}, x_{\text{asym}}) = \begin{bmatrix} 0 \\ 0.25 \\ 0.75 \end{bmatrix}$$

The probability estimate for the region is obtained from the corresponding 20 training data points. 5 are from the Kama variety and 15 from the Rosa variety, so

$$\begin{bmatrix} \mathrm{P}\left(\text{Canadian} \,|\, x_{\text{area}}, x_{\text{asym}}\right) \\ \mathrm{P}\left(\text{Kama} \,|\, x_{\text{area}}, x_{\text{asym}}\right) \\ \mathrm{P}\left(\text{Rosa} \,|\, x_{\text{area}}, x_{\text{asym}}\right) \end{bmatrix} = p_{\mathcal{R},\Theta}(x_{\text{area}}, x_{\text{asym}}) = \begin{bmatrix} \frac{0}{20} \\ \frac{5}{20} \\ \frac{15}{20} \end{bmatrix} = \begin{bmatrix} 0 \\ 0.25 \\ 0.75 \end{bmatrix}. \tag{12.345}$$

As in the case of regression trees, the estimate produced by a classification tree

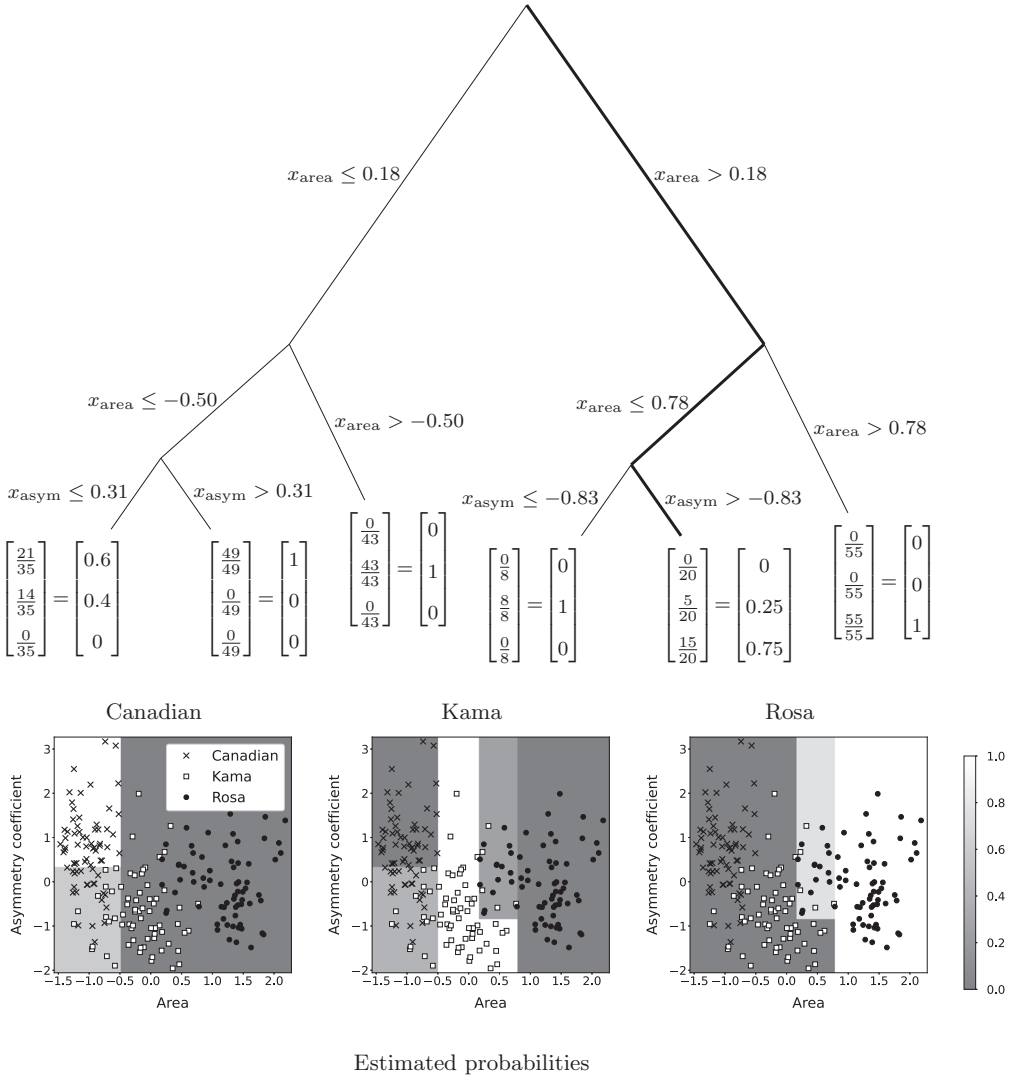


Figure 12.30 Classification tree for wheat varieties. The top diagram shows a classification tree built via recursive binary splitting using the same data as in Figure 12.24. The tree classifies wheat seeds into three varieties according to their area and asymmetric coefficient. The thick black lines show the path from the root to the leaf node corresponding to $x_{\text{area}} := 0.5$ and $x_{\text{asym}} := 0$. The plots below depict the regions in feature space corresponding to each of the leaf nodes, as well as the training data belonging to each region and the corresponding probability estimates.

is accompanied by an interpretable explanation, in terms of the relevant features and thresholds.

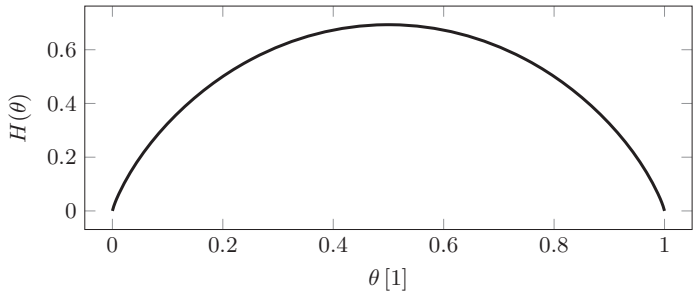


Figure 12.31 Entropy of a binary random variable. The graph shows the entropy of a binary discrete random variable equal to 1 with probability $\theta[1]$ and to 2 with probability $\theta[2] = 1 - \theta[1]$. The entropy is lowest when the probabilities are close to one or zero, and highest when they both equal 0.5.

When fitting a classification tree to a training dataset, we need to learn the partition of regions $\mathcal{R}$. Here we again encounter the challenge discussed in Section 12.7.1: as the depth of the tree model increases, the number of possible trees explodes exponentially. It is therefore intractable to find a global maximum of the tree log-likelihood (12.340) with respect to $\mathcal{R}$. However, recursive binary splitting can be leveraged to optimize the log-likelihood in a greedy fashion, adding nodes one by one. The procedure is exactly as described in Definition 12.40, with the only difference that the splits are chosen to increase the log-likelihood, instead of decreasing the residual sum of squares.

The possible splits for each region R_r in recursive binary splitting are obtained by thresholding each of the d features using a set of thresholds (usually equal to the set of values taken by the feature within the region). We denote by $\mathcal{A}_r(s,t)$ and $\mathcal{B}_r(s,t)$ the two regions resulting from thresholding the sth feature with threshold t. By Theorem 12.41, we should set the probability estimates θ_r, θ_a and θ_b associated with R_r, $\mathcal{A}_r(s,t)$ and $\mathcal{B}_r(s,t)$ equal to:

$$\theta_r[k] := \frac{n_r^{[k]}}{n_r}, \quad n_r := \left|\{i : x_i \in R_r\}\right|, n_r^{[k]} := \left|\{i : x_i \in R_r, y_i = k\}\right|, \tag{12.346}$$

$$\theta_a[k] := \frac{n_a^{[k]}}{n_a}, \quad n_a := \left|\{i : x_i \in \mathcal{A}_r(s,t)\}\right|, n_a^{[k]} := \left|\{i : x_i \in \mathcal{A}_r(s,t), y_i = k\}\right|,$$

$$\theta_b[k] := \frac{n_b^{[k]}}{n_b}, \quad n_b := \left|\{i : x_i \in \mathcal{B}_r(s,t)\}\right|, n_b^{[k]} := \left|\{i : x_i \in \mathcal{B}_r(s,t), y_i = k\}\right|.$$

The increase in log-likelihood provided by the split therefore equals

$$\triangle \mathcal{L}(r,s,t) = \sum_{k=1}^{c} \left(n_a^{[k]} \log \theta_a[k] + n_b^{[k]} \log \theta_b[k] - n_r^{[k]} \log \theta_r[k] \right), \tag{12.347}$$

which replaces the RSS decrease in equation (12.337). The region, feature and threshold for the split are selected to maximize this log-likelihood increase.

In order to gain insight into the implications of maximizing the log-likelihood during recursive binary splitting, we introduce a fundamental quantity in information theory called *entropy*. The entropy of a discrete random variable $\tilde{a}$ with pmf $p_{\tilde{a}}(k) = \theta[k]$ for $1 \leq k \leq c$ and $\theta \in \mathbb{R}^c$ is

$$H(\theta) = -\sum_{k=1}^{c} \theta[k] \log \theta[k]. \tag{12.348}$$

We refer to (Cover, 1999) for an in-depth introduction to the properties of entropy. For our purposes, it is sufficient to notice that the entropy describes to what extent the pmf is concentrated or spread out. The entropy is lower when the random variable equals a small number of values with high probability (the pmf is concentrated), and higher when it can take many values with similar probability (the pmf is spread out). Figure 12.31 shows the entropy of a binary random variable as a function of the pmf entries: the entropy is highest when the two entries have the same probability, and it decreases as the probabilities diverge.

The increase in log-likelihood (12.347) maximized during recursive binary splitting can be expressed in terms of the entropy of the conditional distributions associated with the original region and the two regions resulting from the split:

$$\begin{aligned} \triangle \mathcal{L}(k, s, t) &= -n_r \sum_{k=1}^{c} \frac{n_r^{[k]}}{n_r} \log \theta_r[k] + n_a \sum_{k=1}^{c} \frac{n_a^{[k]}}{n_a} \log \theta_a[k] + n_b \sum_{k=1}^{c} \frac{n_b}{n_b^{[k]}} \log \theta_b[k] \\ &= n_r H(\theta_r) - n_a H(\theta_a) - n_b H(\theta_b). \end{aligned} \tag{12.349}$$

Consequently, the increase in log-likelihood can be interpreted as a decrease in entropy (weighted by the number of points in the corresponding regions). Maximizing this decrease promotes splits for which θ_a and θ_b are highly concentrated in a small number of entries. This occurs when many of the labels within each region are the same, which is exactly what we want in order to achieve high classification accuracy! Example 12.42 and Figure 12.32 illustrate this using a simple classification problem with one feature.

Example 12.42 (Minimizing entropy yields discriminative splits)**.** Figure 12.32 illustrates the process of splitting a region R associated with the node of a classification tree in order to maximize the log-likelihood. Each data point consists of a single feature x and a label y:

x	0.5	1.5	2.5	3.5	4.5	5.5
y	+	-	-	+	+	+

The data are depicted in Figure 12.32. Each label is situated at a position on the horizontal axis equal to its respective feature. The cross and circle markers below the data indicate the probability of the + label in the two regions A and B resulting from splitting R according to each of the possible thresholds. These probabilities, denoted by $\theta_a[+]$ and $\theta_b[+]$, are equal to the fraction of + labels in each region. For example if the threshold is 2, then the labels in A are $\{+, -\}$ and the labels in B are $\{-, +, +, +\}$ so $\theta_a[+] = 0.5$ and $\theta_b[+] = 0.75$.

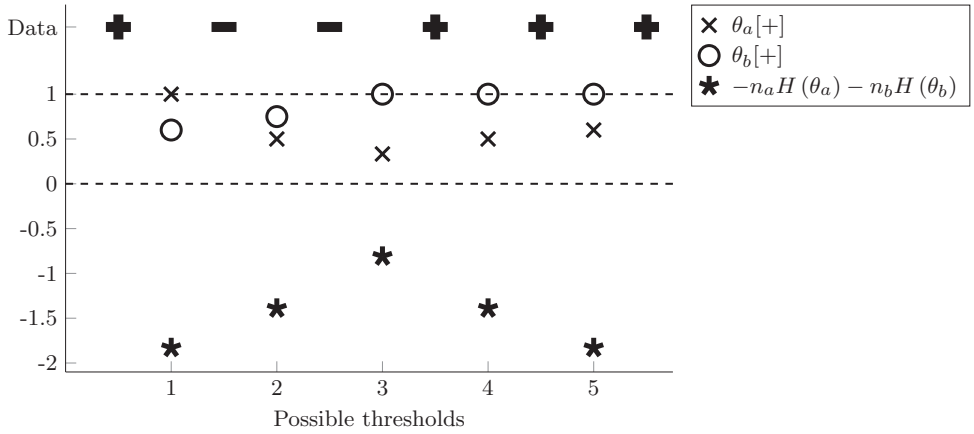


Figure 12.32 Maximizing the decrease in entropy yields discriminative splits. The diagram illustrates the process of splitting a region R associated with the node of a classification tree for a classification problem where there are two labels (+ and -) and a single feature, indexed by the horizontal axis, as described in Example 12.42. The top row shows the labels of the data in R. The cross and circle markers below indicate the probability of the + label in the two regions resulting from the different possible splits. The star markers at the bottom represent the negative weighted sum of entropies corresponding to each split, which is maximized when the threshold is chosen via maximum likelihood. The maximum corresponds to the most discriminative split, which creates one region with mostly - labels on the left and another region with only + labels on the right.

By (12.349) the change in log-likelihood due to the split equals

$$\triangle \mathcal{L}(r, s, t) = n_r H(\theta_r) - n_a H(\theta_a) - n_b H(\theta_b), \tag{12.350}$$

where $n_r = 6$ is the total number of data in the region and θ_r contains the fraction of + and - labels in the region ($\theta_r[+] = 2/3$). Maximizing the log-likelihood with respect to s and t is equivalent to minimizing the weighted sum of entropies $n_a H(\theta_a) + n_b H(\theta_b)$. As shown in Figure 12.31 the entropy is smallest when the probability of + is close to 0 or 1, which occurs if the majority of the labels in the region are the same. Consequently, minimizing the weighted sum promotes homogeneous regions with high label purity, which discriminate between the classes. The star markers at the bottom of Figure 12.32 show the values of $-n_a H(\theta_a) - n_b H(\theta_b)$ for each of the thresholds. The maximum is indeed achieved by the most discriminative split, which creates a region with mostly - labels and another region with only + labels.

An alternative to maximizing the decrease in entropy during recursive binary

splitting, is to minimize the decrease in the Gini index,

$$G(\theta) = \sum_{k=1}^{c} \theta[k]\,(1 - \theta[k])\,, \tag{12.351}$$

which also quantifies the impurity of the labels within a region.

12.7.3 Bagging, Random Forests And Boosting

The tree models described in Sections 12.7.1 and 12.7.2 are simple and interpretable, but their estimation performance on held-out data is often modest, due to their tendency to overfit as the tree depth increases. In this section we explain how to combine multiple trees to build models with improved performance. This is an example of *machine learning*, a general approach to statistical modeling that leverages highly complex models to automatically learn complicated nonlinear dependencies in the data.

Bagging

Bagging, which stands for bootstrap aggregating, combines multiple models via averaging, an approach known as *ensembling*. Intuitively, if the models are different enough for their errors to be approximately independent, then averaging will tend to cancel the errors out, yielding a more accurate estimate. The challenge is to ensure that the models are different to each other; there is no point in averaging models that are the same! Bagging promotes model diversity by fitting each model on a different dataset, obtained by resampling the original training dataset via the bootstrap technique introduced in Section 9.9.

Definition 12.43 (Bagging)**.** *Let (x_1, y_1), (x_2, y_2), ..., (x_n, y_n) be a dataset formed by n pairs of a response y_i and a corresponding vector x_i containing d features. To build a bagging ensemble:*

1 *We generate B bootstrap datasets of size n. To create the bth dataset, we sample bootstrap indices $l_1^{[b]}$, $l_2^{[b]}$, ..., $l_n^{[b]}$ independently and uniformly at random with replacement from the set of possible indices $\{1, ..., n\}$, and then select the bootstrap training data according to those indices,*

$$(x_i^{[b]}, y_i^{[b]}) := (x_{l_i^{[b]}}, y_{l_i^{[b]}}), \qquad 1 \le b \le B, 1 \le i \le n. \tag{12.352}$$

2 *We use each of the B bootstrap datasets to train a model, such as a regression or classification tree:*

$$(x_1^{[b]}, y_1^{[b]}), (x_2^{[b]}, y_2^{[b]}), \ldots, (x_n^{[b]}, y_n^{[b]}) \to t_b, \quad 1 \le b \le B. \tag{12.353}$$

3 *We average the B models to obtain the bagging estimator,*

$$n\ell_{\text{bagging}}(x) := \frac{1}{B} \sum_{b=1}^{B} t_b(x). \tag{12.354}$$

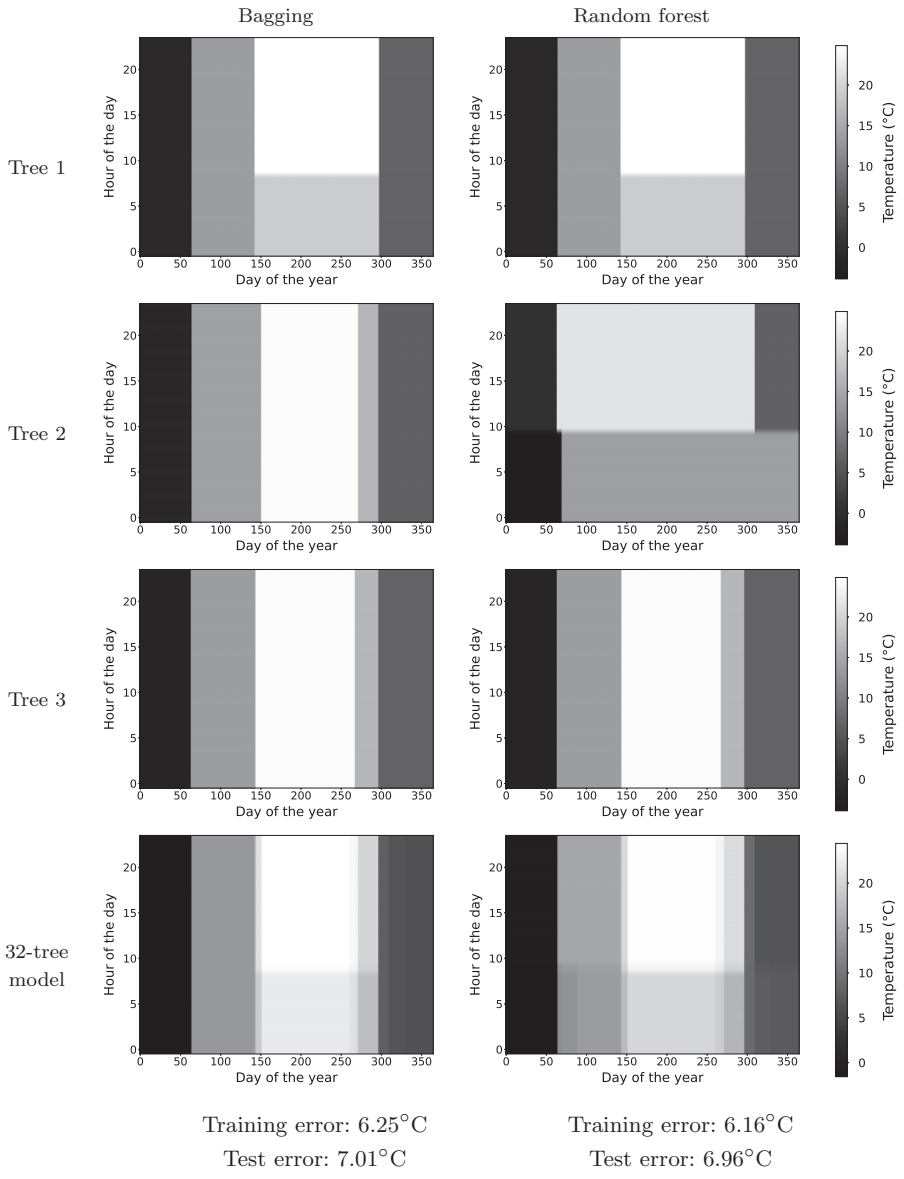


Figure 12.33 Bagging and random forests. The first three rows depict regression-tree estimators based on different bootstrap datasets sampled from the 2015 data in Figure 12.26, as described in Definition 12.43. Each row shows a tree estimator built via standard recursive binary splitting (left) and randomized recursive binary splitting (right) using the same bootstrap dataset. The bottom row shows the bagging (left) and random-forest (right) estimators obtained by averaging 32 such trees. The random-forest model is more complex due to the higher diversity of the individual trees resulting from randomized splitting. This results in a lower root average squared error for the 2015 training dataset and the 2016 test dataset, reported below.

The left column in Figure 12.33 shows bagging in action for the temperature-estimation example from Section 12.7.1. The three first rows show models corresponding to regression trees with five leaves based on different bootstrap training sets. The bagging estimator obtained by averaging the output of 32 such trees is depicted below. Figure 12.34 shows the training and test errors of bagging estimators with different number of trees. Bagging decreases the test error from 7.25°C for a single tree, to around 7°C for an ensemble of 30 trees or more. The price to pay is that the bagging estimate is no longer easily interpretable, as opposed to the estimate of each individual tree.

Random Forests

A key limitation of bagging is that the aggregated models may not be very diverse. Random forests are ensembles of trees that are randomized to increase diversity. This is achieved by modifying the recursive-binary-splitting procedure in Definition 12.40 to choose from a random subset of features at each split. Otherwise, the approach is identical to bagging: B randomized trees are built using bootstrap training datasets, and then combined via averaging.

Definition 12.44 (Random forest)**.** *Let (x_1, y_1), (x_2, y_2), ..., (x_n, y_n) be a dataset formed by n pairs of a response y_i and a corresponding vector x_i containing d features. To build a random forest:*

1 *We generate B bootstrap training sets of size n*

$$(x_i^{[b]}, y_i^{[b]}), \qquad 1 \leq b \leq B, 1 \leq i \leq n, \tag{12.355}$$

as described in Definition 12.43.

2 *We use each of the B datasets to train a tree via randomized recursive binary splitting, where each split is performed using a random subset of $d' < d$ features:*

$$(x_1^{[b]}, y_1^{[b]}), (x_2^{[b]}, y_2^{[b]}), \ldots, (x_n^{[b]}, y_n^{[b]}) \to t_b^{\text{rand}}, \quad 1 \leq b \leq B. \tag{12.356}$$

3 *We average the B models to obtain the random-forest estimator,*

$$n\ell_{\text{rforest}}(x) := \frac{1}{B} \sum_{b=1}^{B} t_b^{\text{rand}}(x). \tag{12.357}$$

The right column of Figure 12.33 shows an application of random forests to our temperature-estimation example. Since there are only two features (hour and day), each split of the randomized recursive-binary-splitting procedure utilizes either just the day, just the hour, or both. Empirically, we observe that using both features 80% of the time, and randomly choosing one of them 20% of the time yields good results. In problems with a larger number of features d, a popular choice is to use $\sqrt{d}$ features in each split.

Each row of Figure 12.33 shows trees constructed by applying the standard and randomized versions of recursive binary splitting to the same bootstrap training set. The first and third trees are the same for bagging and random forest, but the second trees are very different, because the first split in randomized splitting is

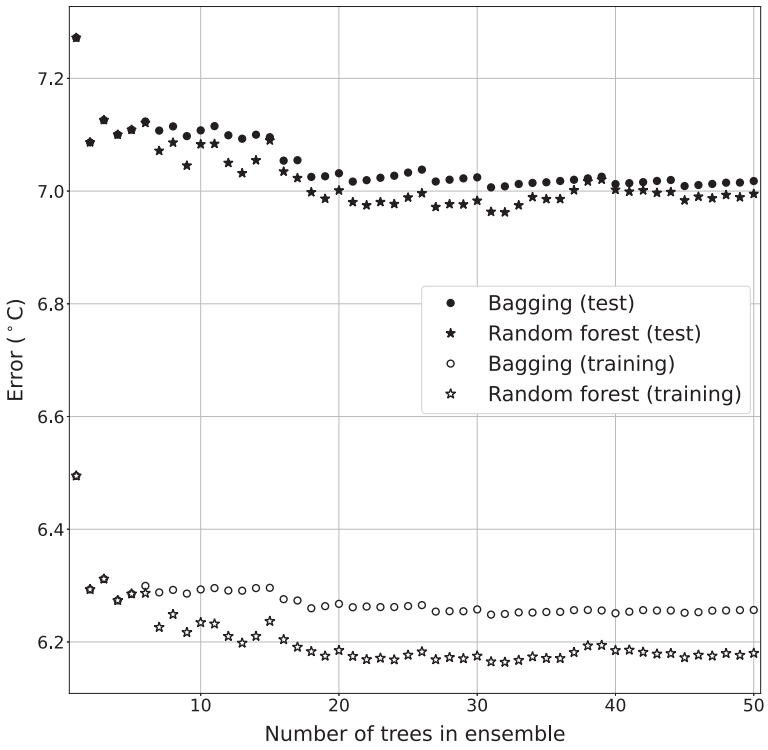


Figure 12.34 Training and test error of bagging and random-forest models. The graph shows the training and test root average squared error of bagging and random-forest ensembles with different number of 5-leaf trees for the temperature-estimation example from Section 12.7.1. As the number of trees increases, the training and test error decrease, eventually plateauing. The random-forest model consistently achieves a lower training and test error than the bagging model, and both models clearly outperform the single-tree model.

randomly chosen to only consider the hour feature. In contrast, standard splitting selects the day feature, resulting in a tree that is identical to the third one. The ensemble models are shown in the bottom row. Due to the higher diversity of its components, the random-forest model is more complex. Figure 12.34 reports the training and test errors of both ensembles for different number of trees. The random forest consistently achieves a better fit to the training data, and also better generalization to the test set.

Boosting

In bagging and random-forest ensembles, each individual model is fitted independently from the rest. Boosting ensembles utilize a different strategy: the models

are fitted sequentially in order to ensure that they complement each other, *boosting* the overall performance.

Let us consider a regression problem, where the data (x_1, y_1), (x_2, y_2), ..., (x_n, y_n) consist of n pairs of a response y_i and a corresponding vector x_i containing d features. To build a boosting ensemble we fit the trees one by one. We begin by fitting the first tree t_1 to the response via recursive binary splitting, which greedily minimizes the residual sum of squares (RSS)

$$\sum_{i=1}^{n} \left(y_i - t_1(x_i)\right)^2, \tag{12.358}$$

as described in Definition 12.40. In bagging or random forests, we would fit the remaining trees in the same way, but using different bootstrap datasets. Boosting instead attempts to build the second tree in a way that complements the first tree. Notice that if the first tree is fixed, minimizing the RSS of the sum of the first and second trees is equivalent to fitting the residual obtained by subtracting the first tree estimate from the response,

$$\sum_{i=1}^{n} \left(\underbrace{y_i - t_1(x_i)}_{r_i^{[2]}} - t_2(x_i) \right)^2. \tag{12.359}$$

Inspired by this observation, we build the second tree to approximate this residual. Applying the same logic, we then fit the third tree to the residual of the sum of the two first trees, the fourth tree to the residual of the sum of the first three trees, and so forth. After fitting L trees, we obtain the *naive* boosting ensemble

$$n\ell_{\text{naiveboost}}(x) := \sum_{l=1}^{L} t_l(x), \tag{12.360}$$

where x denotes an input feature vector.

The left column of Figure 12.35 shows naive boosting estimators built with different numbers of 5-leaf trees for our temperature-estimation example. The training and test errors are reported below each image and also in Figure 12.36. The estimator rapidly achieves very low training error, but does not generalize effectively to the test set. The test error remains relatively high as we increase the number of trees.

Fortunately, the overfitting tendency of our naive boosting estimator can be mitigated using a simple trick. We multiply each tree by a small constant γ to shrink the estimate and prevent it from fitting the training data too well. The resulting (non-naive) boosting estimator, described in the following definition, typically generalizes much better to held-out data.

Definition 12.45 (Boosting)**.** *Let (x_1, y_1), (x_2, y_2), ..., (x_n, y_n) be a dataset formed by n pairs of a response y_i and a corresponding vector x_i containing d features. To build a boosted ensemble we set $r_i^{[1]} := y_i$ for $1 \leq i \leq n$, choose γ to be a small nonnegative number and apply the following steps for $1 \leq l \leq L$, or until a stopping criterion is achieved:*

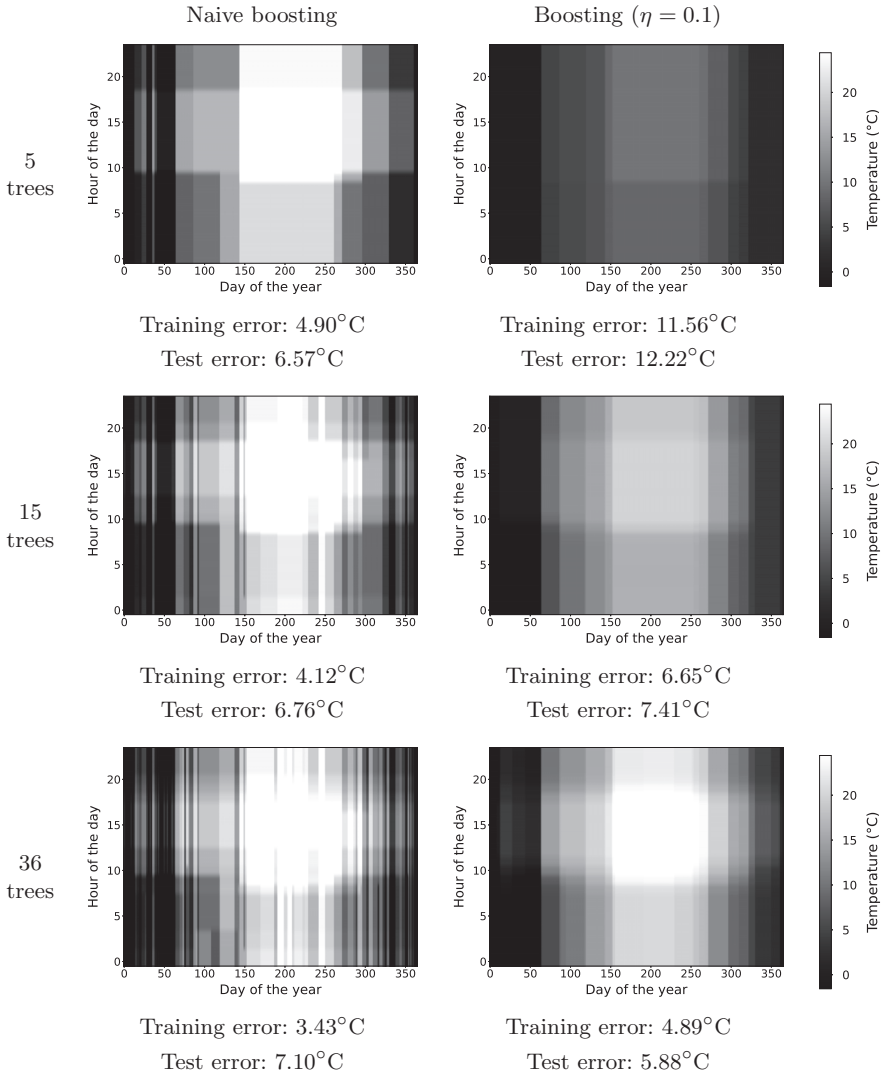


Figure 12.35 Boosting tree ensembles. The images show naive boosting (left) and boosting (right) estimators built using different numbers of 5-leaf trees for the temperature-estimation example from Section 12.7.1. The training and test error of each estimator is reported below the corresponding image.

1 *We use each a modified dataset where the response is set equal to the residual to train a tree via recursive binary splitting:*

$$(x_1, r_1^{[l]}), (x_2, r_2^{[l]}), \ldots, (x_n, r_n^{[l]}) \to t_l. \tag{12.361}$$

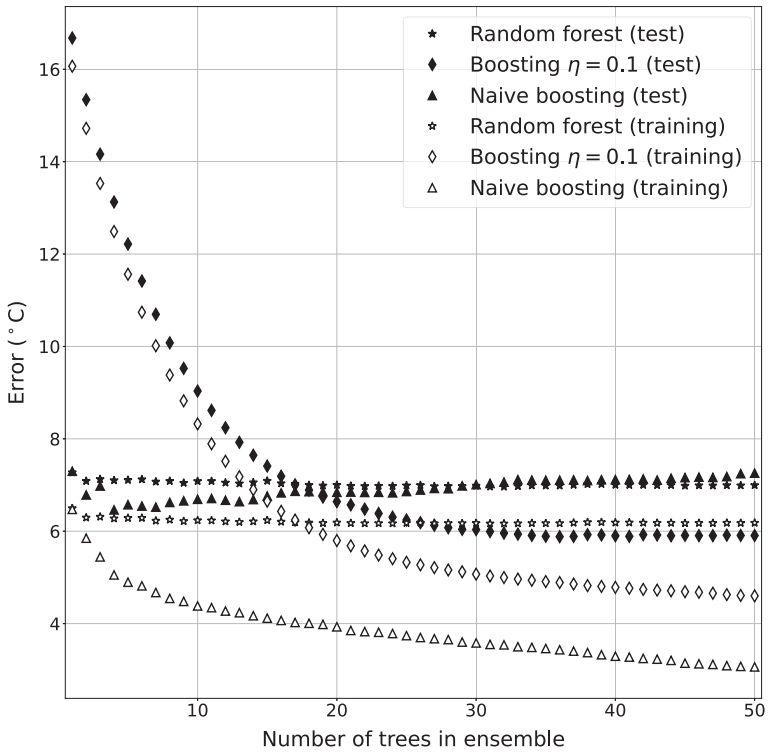


Figure 12.36 Training and test error of boosting models. The graph shows the training and test root average squared error of random-forest, naive boosting and boosting ensembles with different number of 5-leaf trees for the temperature-estimation example from Section 12.7.1. As the number of trees increases, the naive boosting estimator overfits: the training error decreases rapidly and the test error starts increasing after a few boosting iterations. In contrast, the boosting estimator initially underfits the data, but eventually achieves much better generalization. Boosting clearly outperforms random forests in this scenario.

2 We update the estimator and the residual,

$$n\ell_{\text{boost}}(x) := \gamma \sum_{l=1}^{l} t_l(x), \tag{12.362}$$

$$r_i^{[l+1]} := y_i - n\ell_{\text{boost}} t(x_i), \quad 1 \leq l \leq L. \tag{12.363}$$

Figure 12.35 shows boosting estimators built with different numbers of 5-leaf trees for our temperature-estimation example. The training and test errors are reported below each image and also in Figure 12.36. The γ constant is set equal to 0.1 (in practice it should be chosen using validation data). At the beginning of the boosting sequence, multiplying by the γ constant has a shrinking effect. The

resulting estimate has small amplitude, which hinders it from approximating the training data very closely. As a result, the training error is very high, and in fact much higher than that of a single regression tree. However, as more trees join the ensemble, the boosting model is able to gradually approximate the training data, while still improving its performance on the test data.

In this section we have described a simplified version of boosting for regression problems. We refer to Chapter 10 in (Hastie et al., 2009) for a more detailed explanation of boosting and its application to classification. In practice, boosting algorithms often achieve state-of-the-art results for regression and classification problems with a moderate number of features (on the order of hundreds). XGBoost is a particularly popular boosting method (Chen and Guestrin, 2016).

12.8 Neural Networks And Deep Learning

Neural networks or neural nets are perhaps the most renowned models for nonlinear regression and classification. Similarly to the tree-based models in Section 12.7, neural networks learn nonlinear functions directly from data, and are therefore considered machine-learning models. Neural networks are particularly well suited for processing complex high-dimensional data. Fitting large neural networks using massive datasets, an approach known as *deep learning*, has achieved spectacular success in computer vision, natural language processing, reinforcement learning, and many other applications. In Sections 12.8.1 and 12.8.2 we explain how to utilize neural networks to perform regression and classification, respectively.

12.8.1 Regression

In order to explain how to perform regression with a neural network, let us consider the problem of estimating the temperature in Manhattan (Kansas) as a function of the hour x_{hour} and the day x_{day}, as in Section 12.7.1. Recall that for these data, a linear model is unable to capture the nonlinear relationship between the features and the response, which is crucial to produce an accurate estimate (see Figure 12.26).

Neural networks implement nonlinear functions by interleaving affine transformations, known as *layers*, and pointwise nonlinearities. The top left diagram in Figure 12.37 shows a simple two-layer network for our temperature example. The first layer transforms the input into a vector h consisting of two hidden variables, which is typically known as a *feature map* or *activation map* in the deep-learning

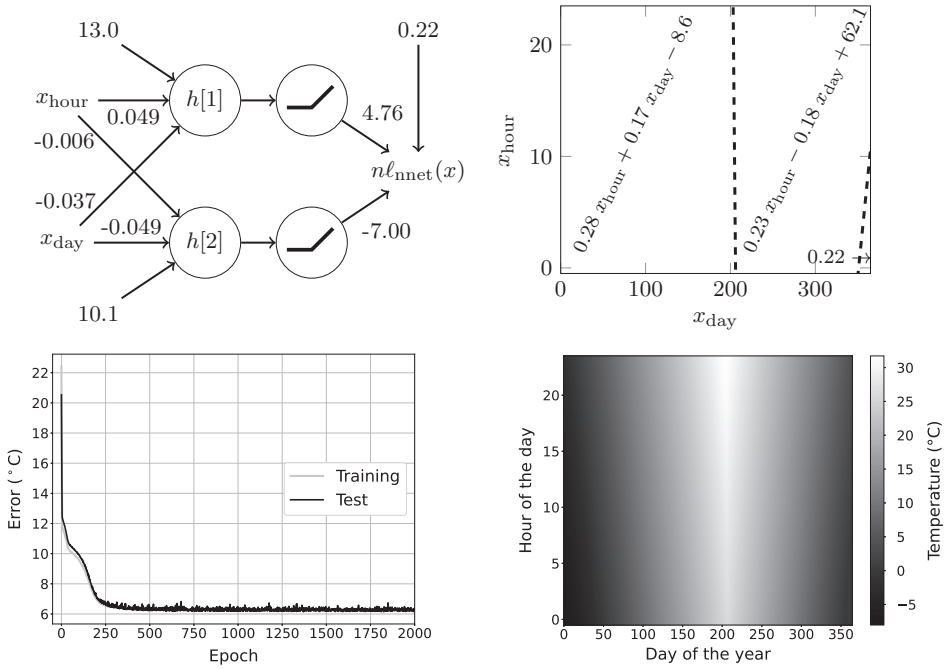


Figure 12.37 Two-layer neural network for temperature estimation. The top left diagram shows a two-layer neural network trained to estimate the temperature in Manhattan (Kansas) as a function of the hour $0 \leq x_{\text{hour}} \leq 23$ and the day $1 \leq x_{\text{day}} \leq 365$. The bottom left graph shows the decrease of the root average squared error during the training epochs on the training set (2015) and test set (2016) depicted in Figure 12.26. The right column shows two visualizations of the function implemented by the neural network, which reveal that the estimate is directly proportional to the day during the first half of the year and inversely proportional during the second half.

literature,

$$h := \underbrace{\begin{bmatrix} 0.049 & -0.037 \\ -0.006 & -0.049 \end{bmatrix}}_{W_1} \begin{bmatrix} x_{\text{hour}} \\ x_{\text{day}} \end{bmatrix} + \underbrace{\begin{bmatrix} 13.0 \\ 10.1 \end{bmatrix}}_{\alpha_1} \tag{12.364}$$

$$= \begin{bmatrix} 0.049x_{\text{hour}} - 0.037x_{\text{day}} + 13.0 \\ -0.006x_{\text{hour}} - 0.049x_{\text{day}} + 10.1 \end{bmatrix}. \tag{12.365}$$

The affine transformation in the first layer is parameterized by the matrix of linear weights W_1 and the additive constant or *bias* α_1. The hidden variables from the first layer are fed into a pointwise nonlinearity or *activation function* called a rectified linear unit (ReLU), which sets negative entries to zero and preserves

nonnegative entries:

$$h_+ := \begin{bmatrix} h[1]_+ \\ h[2]_+ \end{bmatrix}, \tag{12.366}$$

$$a_+ := \begin{cases} a & \text{if} \quad a \geq 0, \\ 0 & \text{otherwise.} \end{cases} \tag{12.367}$$

The rectified hidden variables are combined in the second layer to generate the final regression estimate via another affine transformation, parameterized by a weight matrix W_2 and an additive bias α_2,

$$n\ell_{\text{nnet}}(x) := \underbrace{\begin{bmatrix} 4.76 \\ -7.00 \end{bmatrix}}_{W_2} h_+ + \underbrace{0.22}_{\alpha_2} \tag{12.368}$$

$$= 4.76 h_+[1] - 7.00 h_+[2] + 0.22. \tag{12.369}$$

The presence of the ReLU function between the two layers enables the network to implement a nonlinear function, which applies different affine transformations to different inputs. The first entry of the rectified hidden variable is nonzero when $x_{\text{day}} < 351 + 1.32 x_{\text{hour}}$. In that case, if $x_{\text{day}} < 206 - 0.12 x_{\text{hour}}$, then the second entry is also nonzero, and the function applied to the input equals

$$n\ell_{\text{nnet}}(x) = W_2 (W_1 x + \alpha_1) + \alpha_2 \tag{12.370}$$

$$= 0.28 x_{\text{hour}} + 0.17 x_{\text{day}} - 8.6. \tag{12.371}$$

If instead $351 + 1.32 x_{\text{hour}} > x_{\text{day}} \geq 206 - 0.12 x_{\text{hour}}$, then the second entry of the rectified hidden variable is zero, so the function equals

$$n\ell_{\text{nnet}}(x) = W_2 \left(\begin{bmatrix} 0.049 & -0.037 \\ 0 & 0 \end{bmatrix} x + \begin{bmatrix} 13.0 \\ 0 \end{bmatrix} \right) + \alpha_2 \tag{12.372}$$

$$= 0.23 x_{\text{hour}} - 0.18 x_{\text{day}} + 62.1. \tag{12.373}$$

Finally, if $x_{\text{day}} > 351 + 1.32 x_{\text{hour}}$, then both rectified hidden variables are zero and the output of the network is simply equal to $\alpha_2 = 0.22$. In summary, the estimate $n\ell_{\text{nnet}}(x)$ produced by the neural network equals

$$\begin{cases} 0.28 x_{\text{hour}} + \mathbf{0.17 x_{day}} - 8.6, & \text{if } x_{\text{day}} < 206 - 0.12 x_{\text{hour}}, \\ 0.23 x_{\text{hour}} - \mathbf{0.18 x_{day}} + 62.1, & \text{if } 206 - 0.12 x_{\text{hour}} \leq x_{\text{day}} < 351 + 1.32 x_{\text{hour}}, \\ 0.22, & \text{if } x_{\text{day}} > 351 + 1.32 x_{\text{hour}}. \end{cases} \tag{12.374}$$

The function learned by the network is depicted as a diagram and a heatmap on the right column of Figure 12.37. At the beginning of the year, the temperature estimate is directly proportional to the day. This makes sense: the temperature generally increases between the end of the winter and the beginning of the summer. Then, after day 206 (July 25), the temperature estimate is inversely proportional to the day, reflecting the gradual decrease in temperature between the end of the summer and the beginning of winter.

The parameters of neural networks are learned from data by minimizing a loss or cost function tailored to the task of interest over a training set. For regression, the most popular loss is the residual sum of squares, which approximates the mean squared error of the estimator and is the same cost function optimized by OLS (see Theorem 12.7) and the regression trees in Section 12.7.1. In contrast to OLS, the residual sum of squares is a highly nonconvex function of the parameters in the neural network. As in the case of regression trees, we cannot hope to attain a global minimum. In fact, for large neural networks it is often not feasible to even compute the cost function over all of the training data at once (although it is for our toy problem). Instead, the training set is divided into disjoint sets called *batches*, and the parameters are optimized by iterating through these batches. Consider a batch with n_B pairs of response vectors $x_1, \ldots, x_{n_B}$ and corresponding responses $y_1, \ldots, y_{n_B}$. Let $n\ell_{\text{nnet}}(x)$ denote the output of a neural network with parameters equal to

$$\Theta := \{W_1', \alpha_1', W_2', \alpha_2'\}, \tag{12.375}$$

where W_1' and W_2' are the linear weights of the first and second layer, and α_1' and α_2' the corresponding bias parameters. The batch training error or loss of the network equals

$$\text{RSS}(\Theta) := \sum_{i=1}^{n_B} \left(y_i - n\ell_{\text{nnet}}(x_i)\right)^2. \tag{12.376}$$

Training is performed by taking a step in the direction opposite to the gradient $\nabla_\Theta \text{RSS}(\Theta)$ of the loss with respect to the model parameters,

$$\Theta_b := \Theta_{b-1} - \eta \nabla_\Theta \text{RSS}(\Theta). \tag{12.377}$$

The gradient in (12.377) is obtained via backpropagation, an efficient implementation of the chain rule that propagates derivatives backwards through the network (see Section 6.5 in (Goodfellow et al., 2016)).

This training procedure is known as *stochastic gradient descent*, because the batch RSS can be interpreted as a stochastic approximation to the error over the whole training set, as long as the batches are formed by sampling at random from the training data. Θ_{b-1} and Θ_b denote the model parameters before and after processing the bth batch. The size of the gradient step is scaled by the *learning rate* η, which can depend on b. During training, it is important to tune the value of η correctly. A small value results in very slow progress, whereas a large value can disrupt learning completely. A popular strategy is to utilize methods such as Adam (**?**), which adapt the learning rate based on estimates of lower-order moments of the gradient.

Performing gradient steps over all of the batches in the training set completes an *epoch*. During training, it is advisable to monitor the model error on a separate validation set at each epoch, to evaluate how well the model generalizes to held-out data. This complements the training error, which evaluates the fit to the training set. The bottom left plot of Figure 12.37 shows the evolution of the

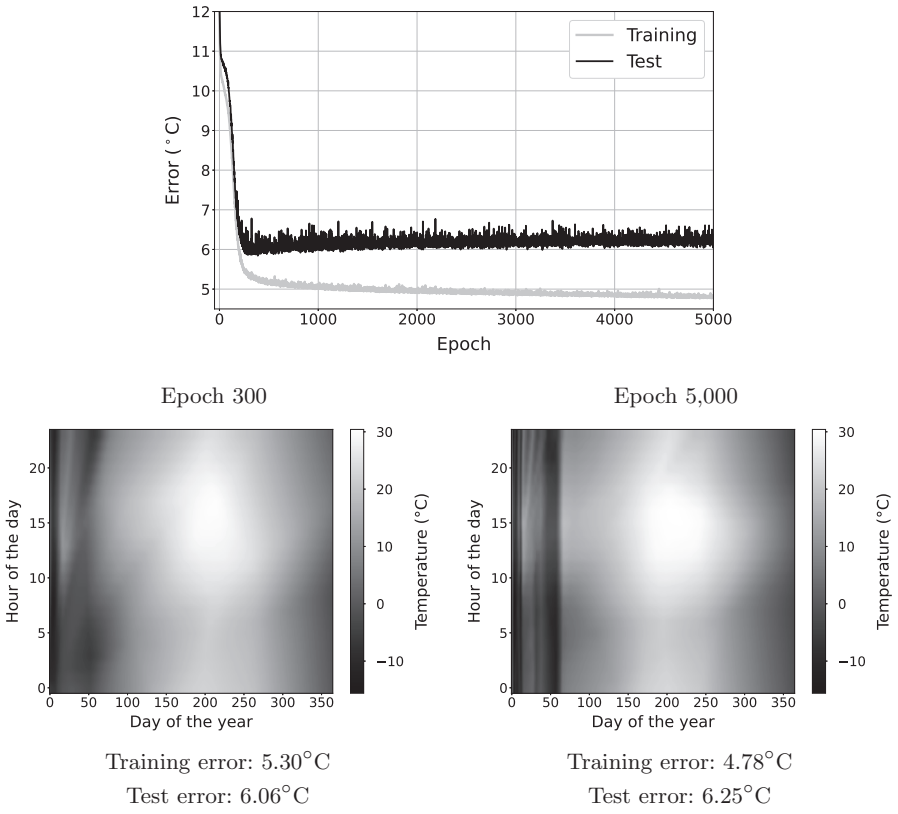


Figure 12.38 Four-layer neural network for temperature estimation. The top diagram shows the evolution of the training and test error of a four-layer neural network trained to estimate the temperature in Manhattan (Kansas) as a function of the the hour $0 \leq x_{\text{hour}} \leq 23$ and the day $1 \leq x_{\text{day}} \leq 365$. The model initially generalizes well to the test set (bottom left), but eventually overfits the training data (bottom right).

training and test error when training our neural network via stochastic gradient descent with a batch size of 100 examples using Adam. Both the training and test error tend to decrease as the epochs proceed. The decrease is not monotonic, which is typical of stochastic gradient descent. The training and test error are equal to the root average error over the whole training set, which consists of data from 2015, and over a test set corresponding to data from 2016, respectively. The training error converges to 6.32°C, and the test error to 6.25°C.

Due to its simplicity, our two-layer network underfits the data. In particular, it does not capture the dependence between the hour of the day and the temperature. In order to improve the prediction, we consider a deeper neural network

with four affine layers interleaved with ReLU nonlinearities:

$$\begin{aligned} h^{[1]} &:= W_1 x + \alpha_1, & W_1 &\in \mathbb{R}^{100\times 2}, \alpha_1 \in \mathbb{R}^{100}, \\ h^{[\ell]} &:= W_\ell h_+^{[\ell-1]} + \alpha_\ell, & W_\ell &\in \mathbb{R}^{100\times 100}, \alpha_\ell \in \mathbb{R}^{100}, \quad 2 \leq \ell \leq 3, \\ n\ell_{\text{nnet}}(x) &:= W_4 h_+^{[3]} + \alpha_4, & W_4 &\in \mathbb{R}^{100\times 1}, \alpha_4 \in \mathbb{R}, \end{aligned} \tag{12.378}$$

where $h^{[\ell]}$ denotes the vector of hidden variables in layer ℓ and $h_+^{[\ell]}$ represents the result of feeding $h^{[\ell]}$ through the ReLU nonlinearity. Each of the intermediate layers has 100 hidden variables, so the total number of parameters in the network is 20,601, which is more than double the number of training examples (8,760)! This is by no means uncommon in deep learning. Neural networks for computer vision tasks have millions of parameters and large language models have billions.

The top graph in Figure 12.38 shows the training and test error of our larger network at each epoch. Just like the smaller network, the model is trained via stochastic gradient descent with a batch size of 100 examples using Adam. The training error experiences a general (nonmonotonic) decrease throughout the training epochs. In contrast, the test error initially decreases, but then starts to increase. At 300 epochs the four-layer model clearly outperforms the simpler two-layer model on the test set (6.06°C vs 6.25°C), but at 5,000 epochs their performance is the same, even though the four-layer model has much smaller training error (4.78°C vs 6.32°C). The resulting models are depicted as heatmaps in the bottom row of Figure 12.38. The behavior of the four-layer network is typical in deep learning: deep networks often exhibit good generalization at the beginning of the training procedure before eventually overfitting the training data. To avoid overfitting, model parameters are usually selected by monitoring performance on a held-out validation set, an approach known as *early-stopping*. Exercise 12.16 shows that, in the case of linear models, early stopping has a regularizing effect on the model coefficients similar to ridge regression.

In contrast to the simpler two-layer model, it is not tractable to derive an analytical expression for the function implemented by our four-layer network, because it contains 300 ReLU nonlinearities (one for each hidden variable in each intermediate layer) with more than 10^{90} (2^{300}) possible states! This is the price to pay for using deep-learning models. They often achieve excellent performance due to their ability to learn highly complex functions, but are notoriously difficult to interpret, precisely because of the complexity of the learned functions.

12.8.2 Classification

In order to perform classification using neural networks, we utilize the softmax function defined in Section 12.6 to transform the network output into probability estimates. Let us consider the seed classification task in Example 12.37, where the goal is to classify wheat seeds based on two features (surface area and asymmetric coefficient). We apply a two-layer neural network with 100 hidden variables to produce a three-dimensional vector that is fed into the softmax function to

generate probability estimates for each of the three classes. For $k \in \{1, 2, 3\}$ the probability estimate equals

$$h := W_1 x + \alpha_1, \qquad W_1 \in \mathbb{R}^{100 \times 2}, \alpha_1 \in \mathbb{R}^{100} \tag{12.379}$$

$$p_\Theta \left(x_i \right)_k := \frac{\exp \left(W_2 h_+[k] + \alpha_2[k] \right)}{\sum_{l=1}^{3} \exp \left(W_2 h_+[l] + \alpha_2[l] \right)}, \qquad W_2 \in \mathbb{R}^{3 \times 100}, \alpha_2 \in \mathbb{R}^3, \tag{12.380}$$

where $\Theta := \{W_1, \alpha_1, W_2, \alpha_2\}$ denotes the network parameters. In order to optimize these parameters, we maximize the log-likelihood of the model on a training set with n examples (x_1, y_1), (x_2, y_2), $\ldots$, (x_n, y_n), where x_i is the ith feature vector and y_i the corresponding label for $1 \leq i \leq n$.

By the same argument used to establish Theorems 12.34 and 12.36, if we assume that the labels are conditionally independent given the features, and each label $\tilde{y}_i$ is conditionally independent from the remaining feature vectors $\{\tilde{x}_l\}_{l \neq i}$ given $\tilde{x}_i$, then the likelihood equals

$$\mathcal{L}_{XY}(\Theta) = \prod_{k=1}^{3} \prod_{\{i : y_i = k\}} p_\Theta \left(x_i \right)_k , \tag{12.381}$$

and the corresponding log-likelihood is

$$\log \mathcal{L}_{XY}(\Theta) = \sum_{k=1}^{3} \sum_{\{i : y_i = k\}} \log p_\Theta \left(x_i \right)_k . \tag{12.382}$$

The negative log-likelihood $-\log \mathcal{L}_{XY}(\Theta)$ is often referred to as the cross-entropy loss in the deep-learning literature, because it can be interpreted as the cross entropy between the estimated probabilities and the labels. To optimize the log-likelihood, we follow the same procedure used to minimize the residual sum of squares in Section 12.8.1. We perform stochastic gradient ascent (or descent if we use the negative log-likelihood) based on the gradient of the log-likelihood with respect to the network parameters Θ, computed via backpropagation on batches of training examples.

The top row in Figure 12.39 shows the softmax inputs for our wheat-classification example, which implement an intricate, nonlinear function of the features. This is in contrast to the softmax-regression model in Figure 12.24, where the softmax inputs are linear. The conditional probabilities estimated by the neural network are shown in the bottom row of Figure 12.39, superposed onto the training data. The flexibility of the model enables it to correctly classify every training example. As in the case of regression, this flexibility can easily lead to overfitting the training set, so it is crucial to evaluate the generalization ability of such models on held-out data.

12.9 Evaluation Of Classification Models

In this section we discuss how to evaluate the performance of classification models, restricting our attention to binary classification problems where there are only

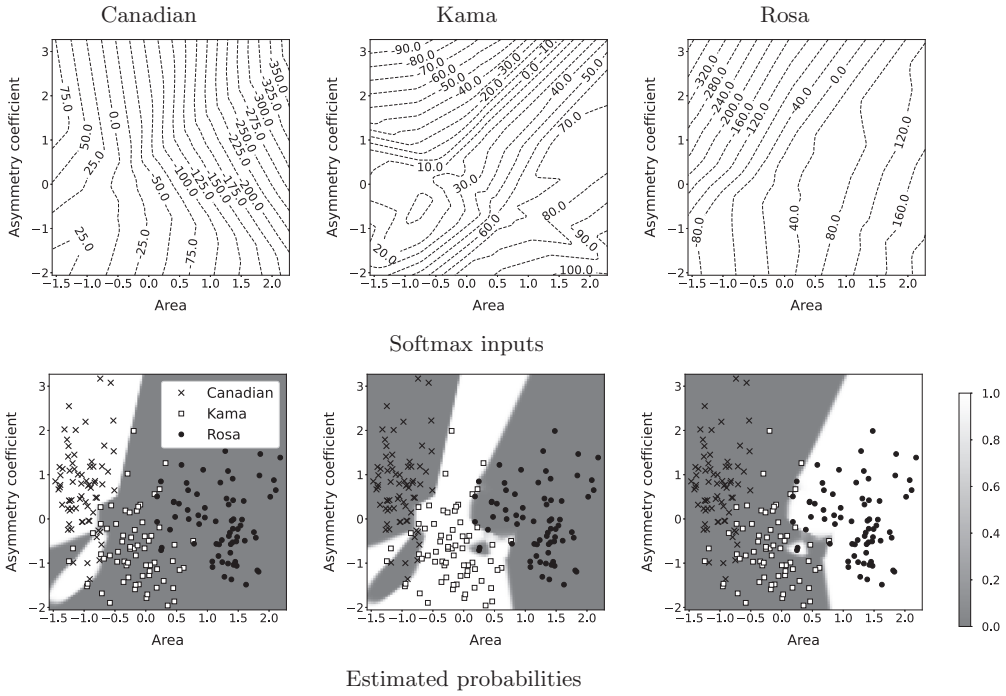


Figure 12.39 Neural-net classification of wheat seeds. Visualization of a two-layer neural-network model trained to classify wheat seeds into three varieties according to their area and asymmetric coefficient. The top row depicts the nonlinear softmax inputs produced by the neural network for each class. These inputs are mapped to the estimated probabilities, shown in the bottom row, by the softmax operation. The network implements a highly intricate function of the features that is able to correctly classify every training example (compare with Figure 12.24).

two classes. Section 12.9.1 describes metrics to measure the discriminative ability of a classifier. Section 12.9.2 explains how to determine whether the probability estimates generated by a classifier are well calibrated.

12.9.1 Measuring Discriminative Performance

In Section 6.5 we use accuracy to evaluate the performance of our model to diagnose Alzheimer's disease. The accuracy of a classifier on a dataset is the fraction of examples that it classifies correctly, which is a very reasonable measure of its discriminative ability. However, in applications such as medical diagnostics, it is often crucial to provide a more nuanced description of classification performance. For instance, we may want to determine what fraction of patients with the disease are being correctly diagnosed. This metric, known as the true positive rate (TPR) or recall, can be very different from the accuracy. In our Alzheimer's example, a

naive baseline that simply classifies everyone as healthy has quite high accuracy on the test set (78.4%), but an abysmal TPR (0%!). The following definition describes several evaluation metrics that quantify different properties of binary classifiers.

Definition 12.46 (Evaluation metrics for binary classification)**.** *We consider a dataset, where there are* P *examples with positive labels and* N *examples with negative labels. A classifier is applied to the data, producing an estimated positive or negative label for each example. Examples with positive labels that are correctly/incorrectly classified are known as true positives* (TP)*/false negatives* (FN)*, respectively (notice that* $\mathrm{TP}+\mathrm{FN}=\mathrm{P}$*). Examples with negative labels that are correctly/incorrectly classified are known as true negatives* (TN)*/false positives* (FP)*, respectively* ($\mathrm{TN}+\mathrm{FP}=\mathrm{N}$)*. This decomposition of the data is often represented in a confusion matrix with the following format:*

True / *Est.*	*Negative*	*Positive*
Negative	***True Negatives*** (TN)	***False Positives*** (FP)
Positive	***False Negatives*** (FN)	***True Positives*** (TP)

The accuracy is the fraction of examples that are correctly classified:

$$\text{Accuracy} := \frac{\mathrm{TN}+\mathrm{TP}}{\mathrm{N}+\mathrm{P}}. \tag{12.383}$$

The true positive rate (TPR), also known as the recall, sensitivity or hit rate, is the fraction of positive examples that are correctly classified:

$$\mathrm{TPR} := \frac{\mathrm{TP}}{\mathrm{P}}. \tag{12.384}$$

The specificity or selectivity is the fraction of negative examples that are correctly classified:

$$\text{Specificity} := \frac{\mathrm{TN}}{\mathrm{N}}. \tag{12.385}$$

The false positive rate (FPR) is the fraction of negative examples that are incorrectly classified:

$$\mathrm{FPR} := \frac{\mathrm{FP}}{\mathrm{N}} = 1 - \text{Specificity}\,. \tag{12.386}$$

The precision or positive predictive value is the fraction of examples predicted as positive that are true positives:

$$\text{Precision} := \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FP}}. \tag{12.387}$$

The F1 score is the harmonic mean∗ of the TPR and the precision:

$$\mathrm{F}_1 := \frac{2 \cdot \mathrm{TPR} \cdot \mathrm{Precision}}{\mathrm{TPR} + \mathrm{Precision}}. \tag{12.388}$$

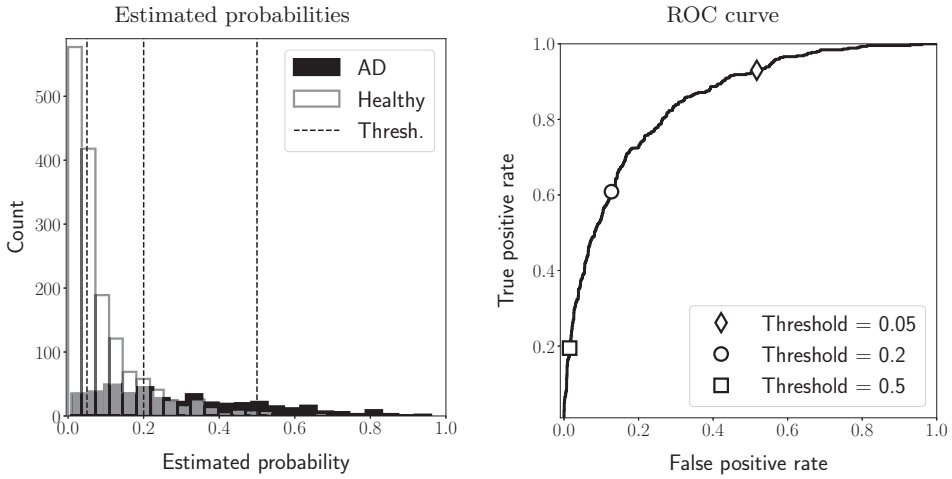
The classification models described in this book generate estimated conditional probabilities of the labels given the input features: naive Bayes (Section 4.8), Gaussian discriminant analysis (Section 6.5), logistic regression (Section 12.5), classification trees (Section 12.7.2) and neural networks (Section 12.8). The estimated probabilities are typically thresholded in order to assign estimated labels to each example. The metrics in Definition 12.46 for the resulting classifier depend heavily on the threshold value.

The top left graph in Figure 12.40 shows histograms of the probability of Alzheimer's estimated by the logistic-regression model from Section 12.5 for the negative (healthy) and positive (Alzheimer's) examples in the test set. The dashed lines represent three different thresholds, which result in different proportions of true/false positives/negatives, as reported in the confusion matrices below. If the threshold is set to 0.05, then the TPR is very high: we correctly identify almost all (93%) Alzheimer's cases! The bad news is that the FPR is also high; more than half (52%) of the healthy subjects are misclassified as having Alzheimer's. The precision is also bad; out of the subjects diagnosed as having Alzheimer's only one third (33%) actually have the disease. Increasing the threshold to 0.2 reduces the FPR dramatically (13%), and increases the precision (57%). The price to pay is a reduction of the TPR to 61%. Further increasing the threshold to 0.5 almost completely eliminates false positives (the FPR is just 1%) and achieves very high precision (78%), but results in a very low TPR (19%). Ultimately, the choice of threshold should be dictated by the application of interest, but among these three options, the middle threshold achieves a reasonable trade-off between TPR and precision, as captured by the F1 score (59%, compared to 49% for the lower threshold and 31% for the higher threshold).

As illustrated by Figure 12.40, there is an inherent trade-off between the TPR and the FPR: as we increase the threshold on the estimated probabilities, the FPR decreases (because we reduce the false positives), but so does the TPR (because we also reduce the true positives). If we plot the FPR-TPR combinations achieved by every possible threshold, we obtain the graph in the top right of Figure 12.40, which is known as the receiver operating characteristic (ROC) curve of the binary classifier. This terminology originated in electrical engineering, as the ROC curve was originally developed to perform detection based on radar signals. The ROC curve enables us to visualize the performance of a classifier for all possible thresholds.

An ROC curve that stretches towards the top left corner is ideal, because it implies that there exist thresholds for which the FPR is low and the TPR is high. Equivalently, we would like for the area under the ROC curve to be as close to one as possible. This is one of the most popular metrics for evaluation of classifiers.

∗Here, the harmonic mean is used, as opposed to the arithmetic mean, because the TPR and the precision are fractions that share the same numerator, see Exercise 12.21.



Threshold = 0.05

Est. \ True	Healthy	AD
Healthy	774	31
AD	829	411

Accuracy = 0.58
TPR = 0.93
FPR = 0.52
Precision = 0.33
F1 score = 0.49

Threshold = 0.2

Est. \ True	Healthy	AD
Healthy	1399	173
AD	204	269

Accuracy = 0.82
TPR = 0.61
FPR = 0.13
Precision = 0.57
F1 score = 0.59

Threshold = 0.5

Est. \ True	Healthy	AD
Healthy	1579	356
AD	24	86

Accuracy = 0.81
TPR = 0.19
FPR = 0.01
Precision = 0.78
F1 score = 0.31

Figure 12.40 Evaluation of a binary classifier. The top left graph shows the probability of Alzheimer's estimated by the logistic-regression model from Section 12.5 for the negative (healthy, in white) and positive (Alzheimer's, in black) examples in the test set. The dashed lines represent three thresholds, which result in different proportions of true/false positives and negatives, reported in the confusion matrices below, and also in different values of the evaluation metrics in Definition 12.46, listed below the confusion matrices. In general, increasing the threshold decreases the FPR and increases the precision, but decreases the TPR. The trade-off between TPR and FPR for this classifier is captured by the receiver operating characteristic curve depicted in the top right graph. The (FPR,TPR) coordinate on the curve corresponding to each of the three thresholds is indicated by a marker.

It can be computed via numerical integration of the ROC curve, and is known as the AUROC or often just AUC, for area under the curve.

Beyond its obvious connection to the TPR and FPR of the classifier, the AUC is equivalent to the concordance or c-statistic of the classifier, which directly

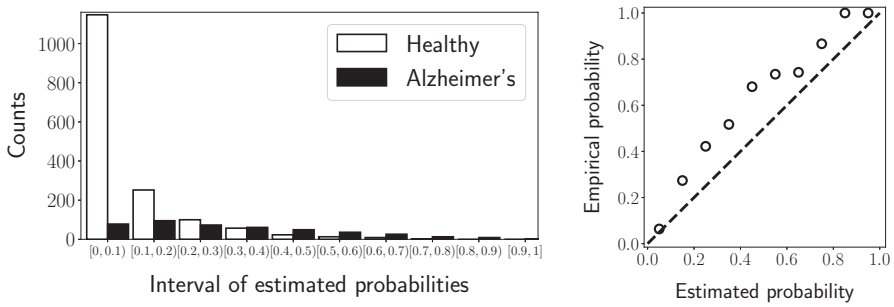


Figure 12.41 Evaluating the calibration of a classification model. In order to evaluate calibration, we separate the probabilities of the positive label estimated by a classifier (in this case, the logistic-regression model from Section 12.5) into bins. The left plot shows the number of negative (healthy, white bars) and positive (Alzheimer's, black bars) test examples in each bin. The reliability diagram on the right plots the empirical probability of Alzheimer's in each bin against the midpoint of the corresponding estimated probabilities. Perfect calibration corresponds to the dashed diagonal line, where the empirical and estimated probabilities are equal.

quantifies its discriminative ability (see Exercise 12.20 for a proof in the limit when the number of examples is large). If we compare the probabilities estimated by the classifier for each possible pair of examples in our dataset, such that one example is positive and the other negative, then the concordance is the proportion of pairs for which the estimated probability of the positive example is higher. When the concordance approaches one, this implies that the classifier is able to completely discriminate between the negative and positive examples. In our Alzheimer's example, the logistic-regression classifier achieves an AUC of 0.847 on the test set.

12.9.2 Calibration of Probability Estimates

A classifier is said to be well calibrated, if it generates probability estimates that adequately reflect the uncertainty associated with each class in a dataset. Highly discriminative classifiers are not necessarily well calibrated. Consider a classifier that assigns a probability of 0.9 to all Alzheimer's patients and a probability of 0.8 to all healthy subjects. The classifier discriminates perfectly between the classes: the estimated probability of all positive examples is higher than that of any negative example, so the AUC equals 1. However, the probability estimates are completely misleading. If we aggregate all examples that are assigned a probability of 0.8 by the classifier, we would expect 80% of them to have Alzheimer's. Instead, none of them do!

Figure 12.41 explains how to evaluate the calibration of a classifier on a dataset. First, we apply the classifier to all the data, in order to obtain a probability estimate for each example. Then, we separate the examples in bins according to

their probability estimate for the positive class, as illustrated by the left graph in Figure 12.41. For example, the first bin in the graph consists of the 1,125 subjects with estimated probabilities between 0 and 0.1. For each bin, we compute the empirical probability of the positive label. In our example, out of the 1,125 subjects in the first bin, 78 have Alzheimer's, so the empirical probability of Alzheimer's in that bin is 6.9%. We then compare the empirical probability to the midpoint of the interval, in this case 5%. The right graph in Figure 12.41 shows a plot of the empirical probabilities against the estimated probabilities for all bins, known as a reliability diagram. Perfect calibration corresponds to a diagonal reliability diagram, where the empirical and estimated probabilities are the same. In our example, the classifier is quite well calibrated, although the estimated probabilities systematically underestimate the empirical probabilities.

Just like a discriminative classifier can be badly calibrated, a well-calibrated classifier can be non-discriminative. Consider a simple classifier that assigns the same estimated probability to every subject in our Alzheimer's example. When we evaluate calibration, all subjects fall into the same bin. If the estimated probability is equal to the prevalence of Alzheimer's in the population (78.4% in our test set), then the empirical and estimated probabilities are the same, so the classifier is perfectly calibrated. However, it has no discriminative ability (in fact, it completely ignores the features!).

The Brier score is a metric that simultaneously evaluates the calibration and discrimination ability of a classifier. It equals the average squared difference between the estimated probabilities and the labels.

Definition 12.47 (Brier score)**.** *Let* (x_1, y_1), (x_2, y_2), ..., (x_n, y_n) *be a dataset formed by* n *pairs of a feature vector* x_i *and a corresponding binary label* y_i *equal to 0 or 1, for* $1 \leq i \leq n$*. The Brier score of a classifier that assigns a probability estimate* $p_{\text{est}}(x_i)$ *to the* i*th example is*

$$\text{Brier score} := \frac{1}{n} \sum_{i=1}^{n} \left(y_i - p_{\text{est}}(x_i) \right)^2 . \tag{12.389}$$

The Brier score of the perfectly discriminative, but uncalibrated, model that assigns 0.9 to all Alzheimer's patients and 0.8 to the healthy subject equals

$$B_{\text{cal}} := \frac{1}{n} \left(\sum_{\{i:y_i=1\}} (1 - 0.9)^2 + \sum_{\{i:y_i=0\}} (0 - 0.8)^2 \right) \tag{12.390}$$

$$= 0.504. \tag{12.391}$$

The Brier score of the perfectly calibrated, but non-discriminative, model that assigns the prevalence to each example is

$$B_{\text{cal}} := \frac{1}{n} \sum_{i=1}^{n} (y_i - 0.784)^2 \tag{12.392}$$

$$= 0.169. \tag{12.393}$$

The Brier score of our logistic-regression model is lower than both, it equals 0.131. This suggests that the probability estimates it produces are quite calibrated and discriminative, which is corroborated by the reliability diagram in Figure 12.41 and its AUC (0.847), respectively.