8 Regression and Model Fitting

"Why are you trying so hard to fit in when you were born to stand out?" (Ian Wallace)

Regression is a special case of the general model fitting and selection procedures discussed in chapters 4 and 5. It can be defined as the relation between a dependent variable, y, and a set of independent variables, x, that describes the expectation value of y given x: $E[y|x]$. The purpose of obtaining a "best-fit" model ranges from scientific interest in the values of model parameters (e.g., the properties of dark energy, or of a newly discovered planet) to the predictive power of the resulting model (e.g., predicting solar activity). The usage of the word *regression* for this relationship dates back to Francis Galton, who discovered that the difference between a child and its parents for some characteristic is proportional to its parents' deviation from typical people in the whole population,[1] or that children "regress" toward the population mean. Therefore, modern usage of the word in a statistical context is somewhat different.

As we will describe below, regression can be formulated in a way that is very general. The solution to this generalized problem of regression is, however, quite elusive. Techniques used in regression tend, therefore, to make a number of simplifying assumptions about the nature of the data, the uncertainties of the measurements, and the complexity of the models. In the following sections we start with a general formulation for regression, list various simplified cases, and then discuss methods that can be used to address them, such as regression for linear models, kernel regression, robust regression, and nonlinear regression.

8.1. Formulation of the Regression Problem

Given a multidimensional data set drawn from some pdf and the full error covariance matrix for each data point, we can attempt to infer the underlying pdf using either parametric or nonparametric models. In its most general incarnation, this is a very hard problem to solve. Even with a restrictive assumption that the errors are

[1] If your parents have very high IQs, you are more likely to have a lower IQ than them, than a higher one. The expected probability distribution for your IQ if you also have a sister whose IQ exceeds your parents' IQs is left as an exercise for the reader. Hint: this is related to regression toward the mean discussed in §4.7.1.

Gaussian, incorporating the error covariance matrix within the posterior distribution is not trivial (cf. §5.6.1). Furthermore, accounting for any selection function applied to the data can increase the computational complexity significantly (e.g., recall §4.2.7 for the one-dimensional case), and non-Gaussian error behavior, if not accounted for, can produce biased results.

Regression addresses a slightly simpler problem: instead of determining the multidimensional pdf, we wish to infer the expectation value of y given x (i.e., the conditional expectation value). If we have a model for the conditional distribution (described by parameters $\boldsymbol{\theta}$) we can write this function[2] as $y = f(x|\boldsymbol{\theta})$. We refer to y as a scalar dependent variable and x as an independent vector. Here x does not need to be a random variable (e.g., x could correspond to deterministic sampling times for a time series). For a given model class (i.e., the function f can be an analytic function such as a polynomial, or a nonparametric estimator), we have k model parameters θ_p, $p = 1, \ldots, k$.

Figure 8.1 illustrates how the constraints on the model parameters, $\boldsymbol{\theta}$, respond to the observations x_i and y_i. In this example, we assume a simple straight-line model with $y_i = \theta_0 + \theta_1 x_i$. Each point provides a joint constraint on θ_0 and θ_1. If there were no uncertainties on the variables then this constraint would be a straight line in the (θ_0, θ_1) plane ($\theta_0 = y_i - \theta_1 x_i$). As the number of points is increased the best estimate of the model parameters would then be the intersection of all lines. Uncertainties within the data will transform the constraint from a line to a distribution (represented by the region shown as a gray band in figure 8.1). The best estimate of the model parameters is now given by the posterior distribution. This is simply the multiplication of the probability distributions (constraints) for all points and is shown by the error ellipses in the lower panel of figure 8.1. Measurements with upper limits (e.g., point x_4) manifest as half planes within the parameter space. Priors are also accommodated naturally within this picture as additional multiplicative constraints applied to the likelihood distribution (see §8.2).

Computationally, the cost of this general approach to regression can be prohibitive (particularly for large data sets). In order to make the analysis tractable, we will, therefore, define several types of regression using three *classification axes*:

- *Linearity.* When a parametric model is linear in all model parameters, that is, $f(x|\theta) = \sum_{p=1}^{k} \theta_p g_p(x)$, where the functions $g_p(x)$ do not depend on any free model parameters (but can be nonlinear functions of x), regression becomes a significantly simpler problem, called *linear regression*. Examples of this include polynomial regression, and radial basis function regression. Regression of models that include nonlinear dependence on θ_p, such as $f(x|\theta) = \theta_1 + \theta_2 \sin(\theta_3 x)$, is called nonlinear regression.

- *Problem complexity.* A large number of independent variables increases the complexity of the error covariance matrix, and can become a limiting factor in nonlinear regression. The most common regression case found in practice is the $M = 1$ case with only a single independent variable (i.e., fitting a straight line to data). For linear models and negligible errors on the independent variables, the problem of dimensionality is not (too) important.

[2] Sometimes $f(x; \boldsymbol{\theta})$ is used instead of $f(x|\boldsymbol{\theta})$ to emphasize that here f is a function rather than a pdf.

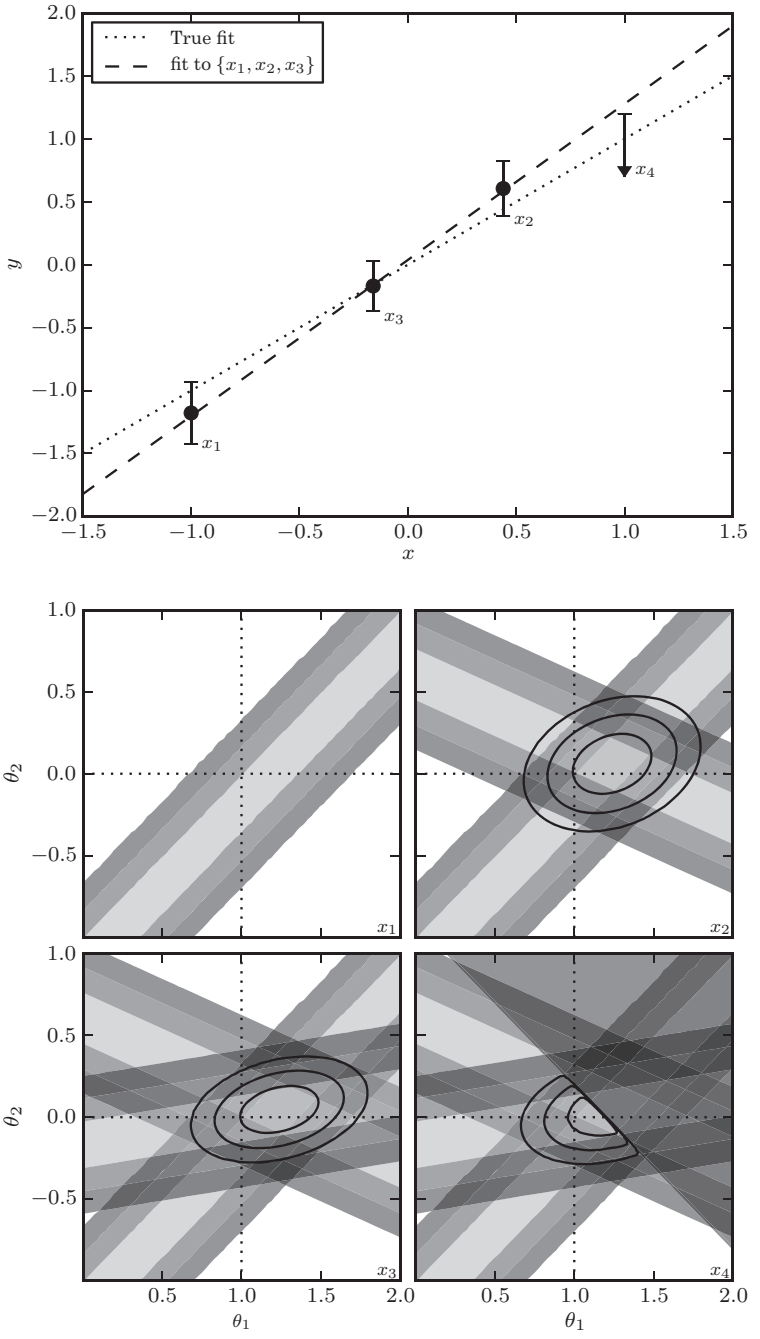

Figure 8.1. An example showing the online nature of Bayesian regression. The upper panel shows the four points used in regression, drawn from the line $y = \theta_1 x + \theta_0$ with $\theta_1 = 1$ and $\theta_0 = 0$. The lower panel shows the posterior pdf in the (θ_0, θ_1) plane as each point is added in sequence. For clarity, the implied dark regions for $\sigma > 3$ have been removed. The fourth point is an upper-limit measurement of y, and the resulting posterior cuts off half the parameter space.

- *Error behavior.* The uncertainties in the values of independent and dependent variables, and their correlations, are the primary factor that determines which regression method to use. The structure of the error covariance matrix, and deviations from Gaussian error behavior, can turn seemingly simple problems into complex computational undertakings. Here we will separately discuss the following cases:

    1. Both independent and dependent variables have negligible errors (compared to the intrinsic spread of data values); this is the simplest and most common "y vs. x" case, and can be relatively easily solved even for nonlinear models and multidimensional data.
    2. Only errors for the dependent variable (y) are important, and their distribution is Gaussian and homoscedastic (with σ either known or unknown).
    3. Errors for the dependent variable are Gaussian and known, but heteroscedastic.
    4. Errors for the dependent variable are non-Gaussian, and their behavior is known.
    5. Errors for the dependent variable are non-Gaussian, but their exact behavior is unknown.
    6. Errors for independent variables (x) are not negligible, but the full covariance matrix can be treated as Gaussian. This case is relatively straightforward when fitting a straight line, but can become cumbersome for more complex models.
    7. All variables have non-Gaussian errors. This is the hardest case and there is no ready-to-use general solution. In practice, the problem is solved on a case-by-case basis, typically using various approximations that depend on the problem specifics.

For the first four cases, when error behavior for the dependent variable is known, and errors for independent variables are negligible, we can easily use the Bayesian methodology developed in chapter 5 to write the posterior pdf for the model parameters,

$$p(\boldsymbol{\theta}|\{x_i, y_i\}, I) \propto p(\{x_i, y_i\}|\boldsymbol{\theta}, I)\, p(\boldsymbol{\theta}, I). \tag{8.1}$$

Here the information I describes the error behavior for the dependent variable. The data likelihood is the product of likelihoods for the individual points, each of which can be expressed as

$$p(y_i|x_i, \boldsymbol{\theta}, I) = e(y_i|y), \tag{8.2}$$

where $y = f(x|\boldsymbol{\theta})$ is the adopted model class, and $e(y_i|y)$ is the probability of observing y_i given the true value (or the model prediction) y. For example, if the y error distribution is Gaussian, with the width for the ith data point given by σ_i, and the errors on x are negligible, then

$$p(y_i|x_i, \boldsymbol{\theta}, I) = \frac{1}{\sigma_i\sqrt{2\pi}} \exp\left(\frac{-[y_i - f(x_i|\boldsymbol{\theta})]^2}{2\sigma_i^2}\right). \tag{8.3}$$

8.1.1. Data Sets Used in This Chapter

For regression and its application to astrophysics we focus on the relation between the redshifts of supernovas and their luminosity distance (i.e., a cosmological parametrization of the expansion of the universe [1]). To accomplish this we generate a set of synthetic supernova data assuming a cosmological model given by

$$\mu(z) = -5\log_{10}\left((1+z)\frac{c}{\mathrm{H}_0}\int\frac{dz}{(\Omega_\mathrm{m}(1+z)^3+\Omega_\Lambda)^{1/2}}\right), \tag{8.4}$$

where $\mu(z)$ is the distance modulus to the supernova, H_0 is the Hubble constant, Ω_m is the cosmological matter density and Ω_Λ is the energy density from a cosmological constant. For our fiducial cosmology we choose $\Omega_\mathrm{m} = 0.27$, $\Omega_\Lambda = 0.73$ and $\mathrm{H}_0 = 71$ km s^{-1} Mpc^{-1}, and add heteroscedastic Gaussian noise that increases linearly with redshift. The resulting $\mu(z)$ cannot be expressed as a sum of simple closed-form analytic functions, such as low-order polynomials. This example addresses many of the challenges we face when working with observational data sets: we do not know the intrinsic complexity of the model (e.g., the form of dark energy), the dependent variables can have heteroscedastic uncertainties, there can be missing or incomplete data, and the dependent variables can be correlated. For the majority of techniques described in this chapter we will assume that uncertainties in the independent variables are small (relative to the range of data and relative to the dependent variables). In real-world applications we do not get to make this choice (the observations themselves define the distribution in uncertainties irrespective of the models we assume). For the supernova data, an example of such a case would be if we estimated the supernova redshifts using broadband photometry (i.e., photometric redshifts). Techniques for addressing such a case are described in §8.8.1. We also note that this toy model data set is a simplification in that it does not account for the effect of K-corrections on the observed colors and magnitudes; see [7].

8.2. Regression for Linear Models

Given an independent variable x and a dependent variable y, we will start by considering the simplest case, a linear model with

$$y_i = \theta_0 + \theta_1 x_i + \epsilon_i. \tag{8.5}$$

Here θ_0 and θ_1 are the coefficients that describe the regression (or objective) function that we are trying to estimate (i.e., the slope and intercept for a straight line $f(x) = \theta_0 + \theta_1 x_i$), and ϵ_i represents an additive noise term.

The assumptions that underlie our linear regression model include the uncertainties on the independent variables that are considered to be negligible, and the dependent variables have known heteroscedastic uncertainties, $\epsilon_i = \mathcal{N}(0, \sigma_i)$. From eq. 8.3 we can write the data likelihood as

$$p(\{y_i\}|\{x_i\}, \boldsymbol{\theta}, I) = \prod_{i=1}^{N}\frac{1}{\sqrt{2\pi}\sigma_i}\exp\left(\frac{-(y_i-(\theta_0+\theta_1 x_i))^2}{2\sigma_i^2}\right). \tag{8.6}$$

For a flat or uninformative prior pdf, $p(\theta|I)$, where we have no knowledge about the distribution of the parameters θ, the posterior will be directly proportional to the likelihood function (which is also known as the error function). If we take the logarithm of the posterior then we arrive at the classic definition of regression in terms of the log-likelihood:

$$\ln(L) \equiv \ln(p(\boldsymbol{\theta}|\{x_i, y_i\}, I)) \propto \sum_{i=1}^{N} \left(\frac{-(y_i - (\theta_0 + \theta_1 x_i))^2}{2\sigma_i^2} \right). \tag{8.7}$$

Maximizing the log-likelihood as a function of the model parameters, $\boldsymbol{\theta}$, is achieved by minimizing the sum of the square errors. This observation dates back to the earliest applications of regression with the work of Gauss [6] and Legendre [14], when the technique was introduced as the *method of least squares*.

The form of the likelihood function and the method of least squares optimization arises from our assumption of Gaussianity for the distribution of uncertainties in the dependent variables. Other forms for the likelihoods can be assumed (e.g., using the L_1 norm, see §4.2.8, which actually precedes the use of the L_2 norm [2, 13], but this is usually at the cost of increased computational complexity). If it is known that measurement errors follow an exponential distribution (see §3.3.6) instead of a Gaussian distribution, then the L_1 norm should be used instead of the L_2 norm and eq. 8.7 should be replaced by

$$\ln(L) \propto \sum_{i=1}^{N} \left(\frac{-|y_i - (\theta_0 + \theta_1 x_i)|}{\Delta_i} \right). \tag{8.8}$$

For the case of Gaussian homoscedastic uncertainties, the minimization of eq. 8.7 simplifies to

$$\theta_1 = \frac{\sum_i^N x_i y_i - \bar{x}\bar{y}}{\sum_i^N (x_i - \bar{x})^2}, \tag{8.9}$$

$$\theta_0 = \bar{y} - \theta_1 \bar{x}, \tag{8.10}$$

where $\bar{x}$ is the mean value of x and $\bar{y}$ is the mean value of y. As an illustration, these estimates of θ_0 and θ_1 correspond to the center of the ellipse shown in the bottom-left panel in Fig. 8.1. An estimate of the variance associated with this regression and the standard errors on the estimated parameters are given by

$$\sigma^2 = \sum_{i=1}^{N} (y_i - \theta_0 + \theta_1 x_i)^2, \tag{8.11}$$

$$\sigma_{\theta_1}^2 = \sigma^2 \frac{1}{\sum_i^N (x_i - \bar{x})^2}, \tag{8.12}$$

$$\sigma_{\theta_0}^2 = \sigma^2 \left(\frac{1}{N} + \frac{\bar{x}^2}{\sum_i^N (x_i - \bar{x})^2} \right). \tag{8.13}$$

For heteroscedastic errors, and in general for more complex regression functions, it is easier and more compact to generalize regression in terms of matrix

notation. We, therefore, define regression in terms of a design matrix, M, such that

$$Y = M\boldsymbol{\theta}, \tag{8.14}$$

where Y is an N-dimensional vector of values y_i,

$$Y = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ . \\ y_{N-1} \end{bmatrix}. \tag{8.15}$$

For our straight-line regression function, $\boldsymbol{\theta}$ is a two-dimensional vector of regression coefficients,

$$\boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix}, \tag{8.16}$$

and M is a $2 \times N$ matrix,

$$M = \begin{bmatrix} 1 & x_0 \\ 1 & x_1 \\ 1 & x_2 \\ . & . \\ 1 & x_{N-1} \end{bmatrix}, \tag{8.17}$$

where the constant value in the first column captures the θ_0 term in the regression.

For the case of heteroscedastic uncertainties, we define a covariance matrix, C, as an $N \times N$ matrix,

$$C = \begin{bmatrix} \sigma_0^2 & 0 & . & 0 \\ 0 & \sigma_1^2 & . & 0 \\ . & . & . & . \\ 0 & 0 & . & \sigma_{N-1}^2 \end{bmatrix} \tag{8.18}$$

with the diagonals of this matrix containing the uncertainties, σ_i, on the dependent variable, Y.

The maximum likelihood solution for this regression is

$$\boldsymbol{\theta} = (M^T C^{-1} M)^{-1} (M^T C^{-1} Y), \tag{8.19}$$

which again minimizes the sum of the square errors, $(Y - \boldsymbol{\theta} M)^T C^{-1} (Y - \boldsymbol{\theta} M)$, as we did explicitly in eq. 8.9. The uncertainties on the regression coefficients, $\boldsymbol{\theta}$, can now be expressed as the symmetric matrix

$$\Sigma_\theta = \begin{bmatrix} \sigma_{\theta_0}^2 & \sigma_{\theta_0\theta_1} \\ \sigma_{\theta_0\theta_1} & \sigma_{\theta_1}^2 \end{bmatrix} = [M^T C^{-1} M]^{-1}. \tag{8.20}$$

Whether we have sufficient data to constrain the regression (i.e., sufficient degrees of freedom) is defined by whether $M^T M$ is an invertible matrix.

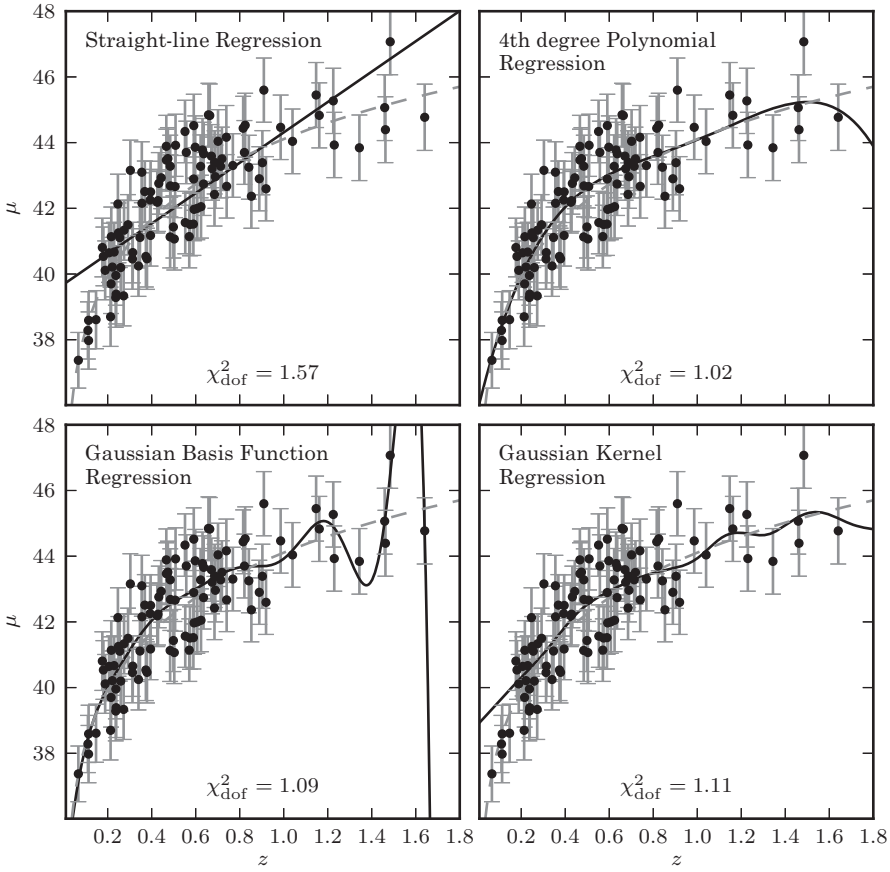

Figure 8.2. Various regression fits to the distance modulus vs. redshift relation for a simulated set of 100 supernovas, selected from a distribution $p(z) \propto (z/z_0)^2 \exp[(z/z_0)^{1.5}]$ with $z_0 = 0.3$. Gaussian basis functions have 15 Gaussians evenly spaced between $z = 0$ and 2, with widths of 0.14. Kernel regression uses a Gaussian kernel with width 0.1.

The top-left panel of figure 8.2 illustrates a simple linear regression of redshift, z, against the distance modulus, μ, for the set of 100 supernovas described in §8.1.1. The solid line shows the regression function for the straight-line model and the dashed line the underlying cosmological model from which the data were drawn (which of course cannot be described by a straight line). It is immediately apparent that the chosen regression model does not capture the structure within the data at the high- and low-redshift limits—the model does not have sufficient flexibility to reproduce the correlation displayed by the data. This is reflected in the $\chi^2_{\rm dof}$ for this fit which is 1.57 (see §4.3.1 for a discussion of the interpretation of $\chi^2_{\rm dof}$).

We now relax the assumptions we made at the start of this section, allowing not just for heteroscedastic uncertainties but also for correlations between the measures of the dependent variables. With no loss in generality, eq. 8.19 can be extended to allow for covariant data through the off-diagonal elements of the covariance matrix C.

8.2.1. Multivariate Regression

For multivariate data (where we fit a hyperplane rather than a straight line) we simply extend the description of the regression function to multiple dimensions, with $y = f(x|\boldsymbol{\theta})$ given by

$$y_i = \theta_0 + \theta_1 x_{i1} + \theta_2 x_{i2} + \cdots + \theta_k x_{ik} + \epsilon_i \tag{8.21}$$

with θ_i the regression parameters and x_{ik} the kth component of the ith data entry within a multivariate data set. This multivariate regression follows naturally from the definition of the design matrix with

$$M = \begin{pmatrix} 1 & x_{01} & x_{02} & . & x_{0k} \\ 1 & x_{11} & x_{12} & . & x_{1k} \\ . & . & . & . & . \\ 1 & x_{N1} & x_{N2} & . & x_{Nk} \end{pmatrix}. \tag{8.22}$$

The regression coefficients (which are estimates of $\boldsymbol{\theta}$ and are often differentiated from the true values by writing them as $\hat{\boldsymbol{\theta}}$) and their uncertainties are, as before,

$$\boldsymbol{\theta} = (M^T C^{-1} M)^{-1} (M^T C^{-1} Y) \tag{8.23}$$

and

$$\Sigma_{\boldsymbol{\theta}} = [M^T C^{-1} M]^{-1}. \tag{8.24}$$

Multivariate linear regression with homoscedastic errors on dependent variables can be performed using the routine `sklearn.linear_model.LinearRegression`. For data with heteroscedastic errors, AstroML implements a similar routine:

```
>>> import numpy as np
>>> from astroML.linear_model import LinearRegression
>>> X = np.random.random((100, 2))  # 100 points in 2 dimensions
>>> dy = np.random.random(100)  # heteroscedastic errors
>>> y = np.random.normal(X[:, 0] + X[:, 1], dy)

>>> model = LinearRegression()
>>> model.fit(X, y, dy)
>>> y_pred = model.predict(X)
```

`LinearRegression` in Scikit-learn has a similar interface, but does not explicitly account for heteroscedastic errors. For a more realistic example, see the source code of figure 8.2.

8.2.2. Polynomial and Basis Function Regression

Due to its simplicity, the derivation of regression in most textbooks is undertaken using a straight-line fit to the data. However, the straight line can simply be interpreted as a first-order expansion of the regression function $y = f(x|\boldsymbol{\theta})$. In general we can express $f(x|\boldsymbol{\theta})$ as the sum of arbitrary (often nonlinear) functions as long as the model is linear in terms of the regression parameters, $\boldsymbol{\theta}$. Examples of these general linear models include a Taylor expansion of $f(x)$ as a series of polynomials where we solve for

the amplitudes of the polynomials, or a linear sum of Gaussians with fixed positions and variances where we fit for the amplitudes of the Gaussians.

Let us initially consider polynomial regression and write $f(x|\boldsymbol{\theta})$ as

$$y_i = \theta_0 + \theta_1 x_i + \theta_2 x_i^2 + \theta_3 x_i^3 + \cdots . \tag{8.25}$$

The design matrix for this expansion becomes

$$M = \begin{pmatrix} 1 & x_0 & x_0^2 & x_0^3 \\ 1 & x_1 & x_1^2 & x_1^3 \\ . & . & . & . \\ 1 & x_N & x_N^2 & x_N^3 \end{pmatrix}, \tag{8.26}$$

where the terms in the design matrix are $1, x, x^2$, and x^3, respectively. The solution for the regression coefficients and the associated uncertainties are again given by eqs. 8.19 and 8.20.

A fourth-degree polynomial fit to the supernova data is shown in the top-right panel of figure 8.2. The increase in flexibility of the model improves the fit (note that we have to be aware of overfitting the data if we just arbitrarily increase the degree of the polynomial; see §8.11). The $\chi^2_{\rm dof}$ of the regression is 1.02, which indicates a much better fit than the straight-line case. At high redshift, however, there is a systematic deviation between the polynomial regression and the underlying generative model (shown by the dashed line), which illustrates the danger of extrapolating this model beyond the range probed by the data.

Polynomial regression with heteroscedastic errors can be performed using the `PolynomialRegression` function in AstroML:

```
>>> import numpy as np
>>> from astroML.linear_model import PolynomialRegression

>>> X = np.random.random((100, 2))  # 100 points in 2 dims
>>> y = X[:, 0] ** 2 + X[:, 1] ** 3
>>> model = PolynomialRegression(3)  # fit 3rd degree polynomial
>>> model.fit(X, y)
>>> y_pred = model.predict(X)
```

Here we have used homoscedastic errors for simplicity. Heteroscedastic errors in `y` can be used in a similar way to `LinearRegression`, above. For a more realistic example, see the source code of figure 8.2.

The number of terms in the polynomial regression grows exponentially with order. Given a data set with k dimensions to which we fit a p-dimensional polynomial, the number of parameters in the model we are fitting is given by

$$m = \frac{(p+k)!}{p!\,k!}, \tag{8.27}$$

including the intercept or offset. The number of degrees of freedom for the regression model is then $\nu = N - m$ and the probability of that model is given by a χ^2 distribution with ν degrees of freedom. As the number of polynomial terms increases, we face the question of overfitting of the data (see §8.11).

We can generalize the polynomial model to a basis function representation by noting that each row of the design matrix can be replaced with any series of linear or nonlinear functions of the variables x_i. Despite the use of arbitrary basis functions, the resulting problem remains linear, because we're fitting only the coefficients multiplying these terms. Examples of commonly used basis functions include Gaussians, trigonometric functions, inverse quadratic functions, and splines.

Basis function regression can be performed using the routine `BasisFunctionRegression` in AstroML. For example, Gaussian basis function regression is as follows:

```
>>> import numpy as np
>>> from astroML.linear_model import BasisFunctionRegression

>>> X = np.random.random((100, 1))  # 100 points in 1 dimension
>>> dy = 0.1
>>> y = np.random.normal(X[:, 0], dy)
>>> mu = np.linspace(0, 1, 10)[:, np.newaxis]  # 10 x 1 array of mu
>>> sigma = 0.1

>>> model = BasisFunctionRegression('gaussian', mu=mu, sigma=sigma)
>>> model.fit(X, y, dy)
>>> y_pred = model.predict(X)
```

For a further example, see the source code of figure 8.2.

The application of Gaussian basis functions to our example regression problem is shown in figure 8.2. In the lower-left panel, 15 Gaussians, evenly spaced between redshifts $0 < z < 2$ with widths of $\sigma_z = 0.14$, are fit to the supernova data. The $\chi^2_{\rm dof}$ for this fit is 1.09, comparable to that for polynomial regression.

8.3. Regularization and Penalizing the Likelihood

All regression examples so far have sought to minimize the mean square errors between a model and data with known uncertainties. The Gauss–Markov theorem states that this least-squares approach results in the minimum variance unbiased estimator (see §3.2.2) for the linear model. In some cases, however, the regression problem may be ill posed and the best unbiased estimator is not the most appropriate regression. Instead, we can trade an increase in bias for a reduction in variance. Examples of such cases include data that are highly correlated (which results in ill-conditioned matrices), or when the number of terms in the regression model decreases the number of degrees of freedom such that we must worry about overfitting of the data.

One solution to these problems is to penalize or limit the complexity of the underlying regression model. This is often referred to as regularization, or shrinkage, and works by applying a penalty to the likelihood function. Regularization can come in many forms, but usually imposes smoothness on the model, or limits the numbers of, or the values of, the regression coefficients.

In §8.2 we showed that regression minimizes the least-squares equation,

$$(Y - M\boldsymbol{\theta})^T (Y - M\boldsymbol{\theta}). \tag{8.28}$$

We can impose a penalty on this minimization if we include a regularization term,

$$(Y - M\boldsymbol{\theta})^T C^{-1}(Y - M\boldsymbol{\theta}) - \lambda|\boldsymbol{\theta}^T\boldsymbol{\theta}|^2, \tag{8.29}$$

where λ is the regularization or smoothing parameter and $|\boldsymbol{\theta}^T\boldsymbol{\theta}|^2$ is an example of the penalty function. In this example, we penalize the size of the regression coefficients (which is known as ridge regression as we will discuss in the next section). Solving for $\boldsymbol{\theta}$ we arrive at a modification of eq. 8.19,

$$\boldsymbol{\theta} = (M^T C^{-1} M + \lambda I)^{-1}(M^T C^{-1} Y), \tag{8.30}$$

where I is the identity matrix. One aspect worth noting about robustness through regularization is that, even if $M^T C^{-1} M$ is singular, solutions can still exist for $(M^T C^{-1} M + \lambda I)$.

A Bayesian implementation of regularization would use the prior to impose constraints on the probability distribution of the regression coefficients. If, for example, we assumed that the prior on the regression coefficients was Gaussian with the width of this Gaussian governed by the regularization parameter λ then we could write it as

$$p(\boldsymbol{\theta}|I) \propto \exp\left(\frac{-(\lambda\boldsymbol{\theta}^T\boldsymbol{\theta})^2}{2}\right). \tag{8.31}$$

Multiplying the likelihood function by this prior results in a posterior distribution with an exponent $(Y - M\boldsymbol{\theta})^T(Y - M\boldsymbol{\theta}) - \lambda|\boldsymbol{\theta}^T\boldsymbol{\theta}|^2$, equivalent to the MLE regularized regression described above.

8.3.1. Ridge Regression

The regularization example above is often referred to as *ridge regression* or *Tikhonov regularization* [22]. It provides a penalty on the sum of the squares of the regression coefficients such that

$$|\boldsymbol{\theta}|^2 < s, \tag{8.32}$$

where s controls the complexity of the model in the same way as the regularization parameter λ in eq. 8.29. By suppressing large regression coefficients this penalty limits the variance of the system at the expense of an increase in the bias of the derived coefficients.

A geometric interpretation of ridge regression is shown in figure 8.3. The solid elliptical contours are the likelihood surface for the regression with no regularization. The circle illustrates the constraint on the regression coefficients ($|\boldsymbol{\theta}|^2 < s$) imposed by the regularization. The penalty on the likelihood function, based on the squared norm of the regression coefficients, drives the solution to small values of $\boldsymbol{\theta}$. The smaller the value of s (or the larger the regularization parameter λ) the more the regression coefficients are driven toward zero.

The regularized regression coefficients can be derived through matrix inversion as before. Applying an SVD to the $N \times m$ design matrix (where m is the number of terms in the model; see §8.2.2) we get $M = U\Sigma V^T$, with U an $N \times m$ matrix, V^T the $m \times m$ matrix of eigenvectors and Σ the $m \times m$ matrix of eigenvalues. We can now write the regularized regression coefficients as

$$\boldsymbol{\theta} = V\Sigma' U^T Y, \tag{8.33}$$

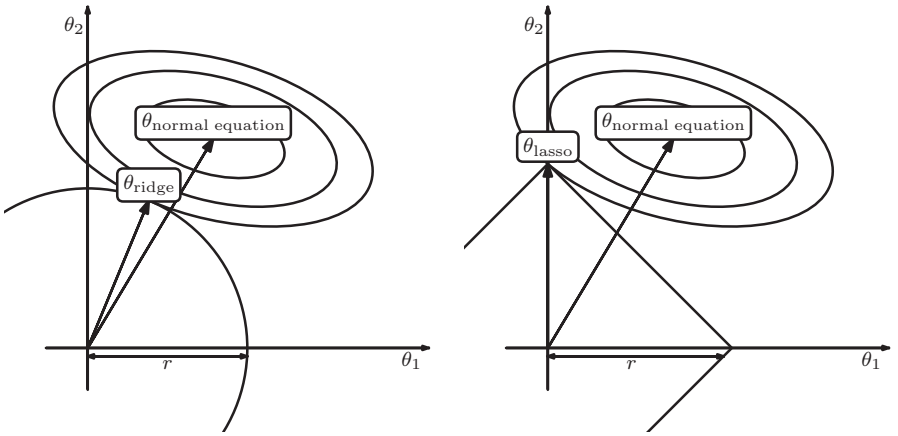

Figure 8.3. A geometric interpretation of regularization. The right panel shows L_1 regularization (LASSO regression) and the left panel L_2 regularization (ridge regularization). The ellipses indicate the posterior distribution for no prior or regularization. The solid lines show the constraints due to regularization (limiting $\boldsymbol{\theta}^2$ for ridge regression and $|\boldsymbol{\theta}|$ for LASSO regression). The corners of the L_1 regularization create more opportunities for the solution to have zeros for some of the weights.

where Σ' is a diagonal matrix with elements $d_i/(d_i^2 + \lambda)$, with d_i the eigenvalues of MM^T.

As λ increases, the diagonal components are down weighted so that only those components with the highest eigenvalues will contribute to the regression. This relates directly to the PCA analysis we described in §7.3. Projecting the variables onto the eigenvectors of MM^T such that

$$Z = MV, \tag{8.34}$$

with z_i the ith eigenvector of M, ridge regression shrinks the regression coefficients for any component for which its eigenvalues (and therefore the associated variance) are small.

The effective goodness of fit for a ridge regression can be derived from the response of the regression function,

$$\hat{y} = M(M^T M + \lambda I)^{-1} M^T y, \tag{8.35}$$

and the number of degrees of freedom,

$$\text{DOF} = \text{Trace}[M(M^T M + \lambda I)^{-1} M^T] = \sum_i \frac{d_i^2}{d_i^2 + \lambda}. \tag{8.36}$$

Figure 8.4 uses the Gaussian basis function regression of §8.2.2 to illustrate how ridge regression will constrain the regression coefficients. The left panel shows the general linear regression for the supernovas (using 100 evenly spaced Gaussians with $\sigma = 0.2$). As we noted in §8.2.2, an increase in the number of model parameters results in an overfitting of the data (the lower panel in figure 8.4 shows how the regression

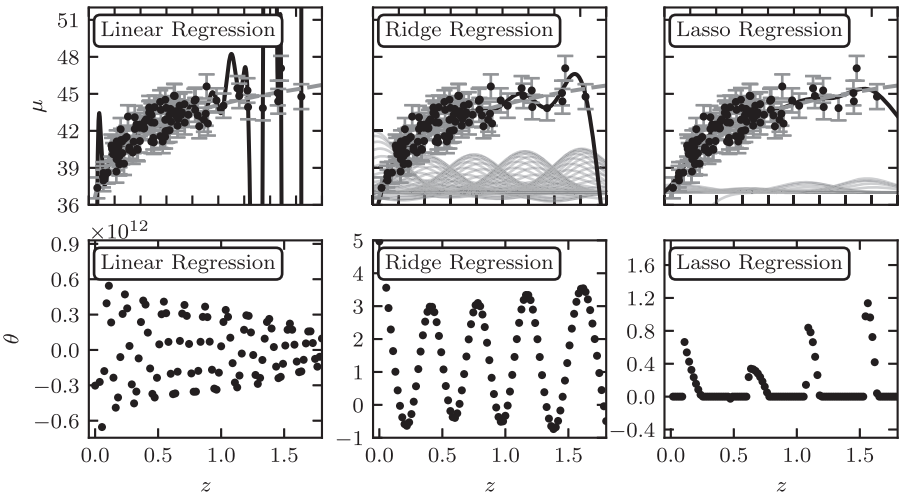

Figure 8.4. Regularized regression for the same sample as Fig 8.2. Here we use Gaussian basis function regression with a Gaussian of width $\sigma = 0.2$ centered at 100 regular intervals between $0 \leq z \leq 2$. The lower panels show the best-fit weights as a function of basis function position. The left column shows the results with no regularization: the basis function weights w are on the order of 10^8, and overfitting is evident. The middle column shows ridge regression (L_2 regularization) with $\alpha = 0.005$, and the right column shows LASSO regression (L_1 regularization) with $\alpha = 0.005$. All three methods are fit without the bias term (intercept).

coefficients for this fit are on the order of 10^8). The central panel demonstrates how ridge regression (with $\lambda = 0.005$) suppresses the amplitudes of the regression coefficients and the resulting fluctuations in the modeled response.

Ridge regression can be accomplished with the `Ridge` class in Scikit-learn:

```
>>> import numpy as np
>>> from sklearn.linear_model import Ridge

>>> X = np.random.random((100, 10))  # 100 points in 10 dims
>>> y = np.dot(X, np.random.random(10))  # random combination of X
>>> model = Ridge(alpha = 0.05)  # alpha controls regularization
>>> model.fit(X, y)
>>> y_pred = model.predict(X)
```

For more information, see the Scikit-learn documentation.

8.3.2. LASSO Regression

Ridge regression uses the square of the regression coefficients to regularize the fits (i.e., the L_2 norm). A modification of this approach is to use the L_1 norm [2] to subset the variables within a model as well as applying shrinkage. This technique is known as *LASSO* (least absolute shrinkage and selection; see [21]). LASSO penalizes the likelihood as

$$(Y - M\boldsymbol{\theta})^T (Y - M\boldsymbol{\theta}) - \lambda |\boldsymbol{\theta}|, \tag{8.37}$$

where $|\boldsymbol{\theta}|$ penalizes the absolute value of $\boldsymbol{\theta}$. LASSO regularization is equivalent to least-squares regression with a penalty on the absolute value of the regression coefficients,

$$|\boldsymbol{\theta}| < s. \tag{8.38}$$

The most interesting aspect of LASSO is that it not only weights the regression coefficients, it also imposes sparsity on the regression model. Figure 8.3 illustrates the impact of the L_1 norm on the regression coefficients from a geometric perspective. The $\lambda|\boldsymbol{\theta}|$ penalty preferentially selects regions of likelihood space that coincide with one of the vertices within the region defined by the regularization. This corresponds to setting one (or more if we are working in higher dimensions) of the model attributes to zero. This subsetting of the model attributes reduces the underlying complexity of the model (i.e., we make zeroing of weights, or feature selection, more aggressive). As λ increases, the size of the region encompassed within the constraint decreases.

LASSO regression can be accomplished with the `Lasso` class in Scikit-learn:

```
>>> import numpy as np
>>> from sklearn.linear_model import Lasso

>>> X = np.random.random((100, 10))  # 100 points in 10 dims
>>> y = np.dot(X, np.random.random(10))  # random comb. of X
>>> model = Lasso(alpha = 0.05)  # alpha controls regularization
>>> model.fit(X, y)
>>> y_pred = model.predict(X)
```

For more information, see the Scikit-learn documentation.

Figure 8.4 shows this effect for the supernova data. Of the 100 Gaussians in the input model, with $\lambda = 0.005$, only 32 are selected by LASSO (note the regression coefficients in the lower panel). This reduction in model complexity suppresses the overfitting of the data.

A disadvantage of LASSO is that, unlike ridge regression, there is no closed-form solution. The optimization becomes a quadratic programming problem (though it is still a convex optimization). There are a number of numerical techniques that have been developed to address these issues including coordinate-gradient descent [12] and least angle regression [5].

8.3.3. How Do We Choose the Regularization Parameter λ?

In each of the regularization examples above we defined a shrinkage parameter that we refer to as the *regularization parameter*. The natural question then is how do we set λ? So far we have only noted that as we increase λ we increase the constraints on the range regression coefficients (with $\lambda = 0$ returning the standard least-squares regression). We can, however, evaluate its impact on the regression as a function of its amplitude.

Applying the k-fold cross-validation techniques described in §8.11 we can define an error (for a specified value of λ) as

$$\text{Error}(\lambda) = k^{-1} \sum_k N_k^{-1} \sum_i^{N_k} \frac{\left[y_i - f(x_i|\boldsymbol{\theta})\right]^2}{\sigma_i^2}, \tag{8.39}$$

where N_k^{-1} is the number of data points in the kth cross-validation sample, and the summation over N_k represents the sum of the squares of the residuals of the fit. Estimating λ is then simply a case of finding the λ that minimizes the cross-validation error.

8.4. Principal Component Regression

For the case of high-dimensional data or data sets where the variables are collinear, the relation between ridge regression and principal component analysis can be exploited to define a regression based on the principal components. For centered data (i.e., zero mean) we recall from §7.3 that we can define the principal components of a system from the data covariance matrix, $X^T X$, by applying an eigenvalue decomposition (EVD) or singular value decomposition (SVD),

$$X^T X = V \Sigma V^T, \tag{8.40}$$

with V^T the eigenvectors and Σ the eigenvalues.

Projecting the data matrix onto these eigenvectors we define a set of projected data points,

$$Z = XV^T, \tag{8.41}$$

and truncate this expansion to exclude those components with small eigenvalues. A standard linear regression can now be applied to the data transposed to this principal component space with

$$Y = M_z \boldsymbol{\theta} + \epsilon, \tag{8.42}$$

with M_z the design matrix for the projected components z_i. The PCA analysis (including truncation) and the regression are undertaken as separate steps in this procedure. The distinction between principal component regression (PCR) and ridge regression is that the number of model components in PCR is ordered by their eigenvalues and is absolute (i.e., we weight the regression coefficients by 1 or 0). For ridge regression the weighting of the regression coefficients is continuous.

The advantages of PCR over ridge regression arise primarily for data containing independent variables that are collinear (i.e., where the correlation between the variables is almost unity). For these cases, the regression coefficients have large variance and their solutions can become unstable. Excluding those principal components with small eigenvalues alleviates this issue. At what level to truncate the set of eigenvectors is, however, an open question (see §7.3.2). A simple approach to take is to truncate based on the eigenvalue with common cutoffs ranging between 1% and 10% of the average eigenvalue (see §8.2 of [10] for a detailed discussion of truncation levels for PCR). The disadvantage of such an approach is that an eigenvalue does not always correlate with the ability of a given principal component to predict the dependent variable. Other techniques, including cross-validation [17], have been proposed yet there is no well-adopted solution to this problem.

Finally, we note that in the case of ill-conditioned regression problems (e.g., those with collinear data), most implementations of linear regression will implicitly perform a form of principal component regression when inverting the singular matrix $M^T M$. This comes through the use of the robust *pseudoinverse*, which truncates small singular values to prevent numerical overflow in ill-conditioned problems.

8.5. Kernel Regression

The previous sections found the regression or objective function that best fits a set of data assuming a linear model. Before we address the question of nonlinear optimization we will describe a number of techniques that make use of locality within the data (i.e., local regression techniques).

Kernel or Nadaraya–Watson [18, 23] regression defines a kernel, $K(x_i, x)$, local to each data point, with the amplitude of the kernel depending only on the distance from the local point to all other points in the sample. The properties of the kernel are such that it is positive for all values and approaches zero asymptotically as the distance approaches infinity. The influence of the kernel (i.e., the region of parameter space over which we weight the data) is determined by its width or bandwidth, h. Common forms of the kernel include the top-hat function, and the Gaussian distribution.

The Nadaraya–Watson estimate of the regression function is given by

$$f(x|K) = \frac{\sum_{i=1}^{N} K\left(\frac{||x_i - x||}{h}\right) y_i}{\sum_{i=1}^{N} K\left(\frac{||x_i - x||}{h}\right)}, \tag{8.43}$$

which can be viewed as taking a weighted average of the dependent variable, y. This gives higher weight to points near x with a weighting function,

$$w_i(x) = \frac{K\left(\frac{||x_i - x||}{h}\right)}{\sum_{i=1}^{N} K\left(\frac{||x_i - x||}{h}\right)}. \tag{8.44}$$

Nadaraya–Watson kernel regression can be performed using AstroML in the following way:

```
>>> import numpy as np
>>> from astroML.linear_model import NadarayaWatson

>>> X = np.random.random((100, 2))  # 100 points in 2 dims
>>> y = X[:, 0] + X[:, 1]
>>> model = NadarayaWatson('gaussian', 0.05)
>>> model.fit(X, y)
>>> y_pred = model.predict(X)
```

Figure 8.2 shows the application of Gaussian kernel regression to the synthetic supernova data compared to the standard linear regression techniques introduced in §8.2. For this example we use a Gaussian kernel with $h = 0.1$ that is constant across the redshift interval. At high redshift, where the data provide limited support for the model, we see that the weight function drives the predicted value of y to that of the nearest neighbor. This prevents the extrapolation problems common when fitting polynomials that are not constrained at the edges of the data (as we saw in the top panels of figure 8.2). The width of the kernel acts as a smoothing function. At low redshift, the increase in the density of points at $z \approx 0.25$ biases the weighted estimate of $\hat{y}$ for $z < 0.25$. This is because, while the kernel amplitude is small for the higher redshift

points, the increase in the sample size results in a higher than average weighting of points at $z \approx 0.25$. Varying the bandwidth as a function of the independent variable can correct for this. The rule of thumb for kernel-based regression (as with kernel density estimation described in §6.3) is that the bandwidth is more important than the exact shape of the kernel.

Estimation of the optimal bandwidth of the kernel is straightforward using cross-validation (see §8.11). We define the CV error as

$$\mathrm{CV}_{L_2}(h) = \frac{1}{N}\sum_{i=1}^{N}\left(y_i - f\left(x_i|K\left(\frac{||x_i - x_j||}{h}\right)\right)\right)^2. \tag{8.45}$$

We actually do not need to compute separate estimates for each leave-one-out cross-validation subsample if we rewrite this as

$$\mathrm{CV}_{L_2}(h) = \frac{1}{N}\frac{\sum_{i=1}^{N}\left(y_i - f(x_i|K)\right)^2}{\left(1 - \frac{K(0)}{\sum_{j=1}^{N}K\left(\frac{||x_i - x_j||}{h}\right)}\right)^2}. \tag{8.46}$$

It can be shown that, as for kernel density estimation, the optimal bandwidth decreases with sample size at a rate of $N^{-1/5}$.

8.6. Locally Linear Regression

Related to kernel regression is *locally linear regression*, where we solve a separate weighted least-squares problem at each point x, finding the $w(x)$ which minimizes

$$\sum_{i=1}^{N} K\left(\frac{||x - x_i||}{h}\right)\left(y_i - w(x)\,x_i\right)^2. \tag{8.47}$$

The assumption for locally linear regression is that the regression function can be approximated by a Taylor series expansion about any local point. If we truncated this expansion at the first term (i.e., a locally constant solution) we recover kernel regression. For locally linear regression the function estimate is

$$f(x|K) = \theta(x)x \tag{8.48}$$

$$= x^T\left(\mathbf{X}^T W(x)\mathbf{X}\right)^{-1}\mathbf{X}^T W(x)\mathbf{Y} \tag{8.49}$$

$$= \sum_{i=1}^{N} w_i(x)y_i, \tag{8.50}$$

where $W(x)$ is an $N \times N$ diagonal matrix with the ith diagonal element given by $K||x_i - x||/h$.

A common form for $K(x)$ is the tricubic kernel,

$$K(x_i, x) = \left(1 - \left(\frac{|x - x_i|}{h}\right)^3\right)^3 \tag{8.51}$$

for $|x_i - x| < h$, which is often referred to as lowess (or loess; locally weighted scatter plot smoothing); see [3].

There are further extensions possible for local linear regression:

- *Local polynomial regression*. We can consider any polynomial order. However there is a bias–variance (complexity) trade-off, as usual. The general consensus is that going past linear increases variance without decreasing bias much, since local linear regression captures most of the boundary bias.
- *Variable-bandwidth kernels*. Let the bandwidth for each training point be inversely proportional to its kth nearest neighbor's distance. Generally a good idea in practice, though there is less theoretical consensus on how to choose the parameters in this framework.

None of these modifications improves the convergence rate.

8.7. Nonlinear Regression

Forcing data to correspond to a linear model through the use of coordinate transformations is a well-used trick in astronomy (e.g., the extensive use of logarithms in the astronomical literature to linearize complex relations between attributes; fitting $y = A\exp(Bx)$ becomes a linear problem with $z = K + Bx$, where $z = \log y$ and $K = \log A$). These simplifications, while often effective, introduce other complications (including the non-Gaussian nature of the uncertainties for low signal-to-noise data). We must, eventually, consider the case of nonlinear regression and model fitting.

In the cosmological examples described previously we have fit a series of parametric and nonparametric models to the supernova data. Given that we know the theoretical form of the underlying cosmological model, these models are somewhat ad hoc (e.g., using a series of polynomials to parameterize the dependence of distance modulus on cosmological redshift). In the following we consider directly fitting the cosmological model described in eq. 8.4. Solving for Ω_m and Ω_Λ is a nonlinear optimization problem requiring that we maximize the posterior,

$$p(\Omega_m, \Omega_\Lambda | z, I) \propto \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}\sigma_i} \exp\left(\frac{-(\mu_i - \mu(z_i|\Omega_m, \Omega_\Lambda))^2}{2\sigma_i^2}\right) p(\Omega_m, \Omega_\Lambda) \quad (8.52)$$

with μ_i the distance modulus for the supernova and z_i the redshift.

In §5.8 we introduced Markov chain Monte Carlo as a sampling technique that can be used for searching through parameter space. Figure 8.5 shows the resulting likelihood contours for our cosmological model after applying the Metropolis–Hastings algorithm to generate the MCMC chains and integrating the chains over the parameter space.

An alternate approach is to use the Levenberg–Marquardt (LM) algorithm [15, 16] to optimize the maximum likelihood estimation. LM searches for the sum-of-squares minima of a multivariate distribution through a combination of gradient descent and Gauss–Newton optimization. Assuming that we can express our regression function as a Taylor series expansion, then, to first order, we can write

$$f(x_i|\boldsymbol{\theta}) = f(x_i|\boldsymbol{\theta}_0) + Jd\boldsymbol{\theta}, \quad (8.53)$$

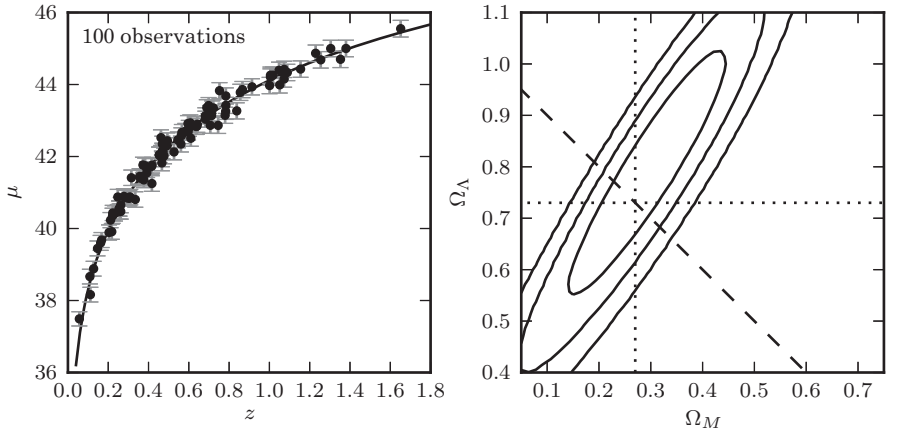

Figure 8.5. Cosmology fit to the standard cosmological integral. Errors in μ are a factor of ten smaller than for the sample used in figure 8.2. Contours are 1-, 2-, and 3-σ for the posterior (uniform prior in Ω_M and Ω_Λ). The dashed line shows flat cosmology. The dotted lines show the input values.

where $\boldsymbol{\theta}_0$ is an initial guess for the regression parameters, J is the Jacobian about this point ($J = \partial f(x_i|\boldsymbol{\theta})/\partial\boldsymbol{\theta}$), and $d\boldsymbol{\theta}$ is a small change in the regression parameters. LM minimizes the sum of square errors,

$$\sum_i (y_i - f(x_i|\boldsymbol{\theta}_0) - J_i d\boldsymbol{\theta})^2 \tag{8.54}$$

for the perturbation $d\boldsymbol{\theta}$. This results in an update relation for $d\boldsymbol{\theta}$ of

$$(J^T C^{-1} J + \lambda \,\mathrm{diag}(J^T C^{-1} J))\, d\boldsymbol{\theta} = J^T C^{-1}(Y - f(X|\boldsymbol{\theta})), \tag{8.55}$$

with C the standard covariance matrix introduced in eq. 8.18. In this expression the λ term acts as a damping parameter (in a manner similar to ridge regression regularization discussed in §8.3.1). For small λ the relation approximates a Gauss–Newton method (i.e., it minimizes the parameters assuming the function is quadratic). For large λ the perturbation $d\boldsymbol{\theta}$ follows the direction of steepest descent. The $\mathrm{diag}(J^T C^{-1} J)$ term, as opposed to the identity matrix used in ridge regression, ensures that the update of $d\boldsymbol{\theta}$ is largest along directions where the gradient is smallest (which improves convergence).

LM is an iterative process. At each iteration LM searches for the step $d\boldsymbol{\theta}$ that minimizes eq. 8.54 and then updates the regression model. The iterations cease when the step size, or change in likelihood values, reaches a predefined value. Throughout the iteration process the damping parameter, λ, is adaptively varied (decreasing as the minimum is approached). A common approach involves decreasing (or increasing) λ by ν^k, where k is the number of the iteration and ν is a damping factor (with $\nu > 1$). Upon successful convergence the regression parameter covariances are given by $[J^T \boldsymbol{\theta} J]^{-1}$. It should be noted that the success of the LM algorithm often relies on the initial guesses for the regression parameters being close to the maximum likelihood solution (in the presence of nonlocal minima), or the likelihood function surface being unimodal.

The submodule `scipy.optimize` includes several routines for optimizing linear and nonlinear equations. The Levenberg–Marquardt algorithm is used in `scipy.optimize.leastsq`. Below is a brief example of using the routine to estimate the first six terms of the Taylor series for the function $y = \sin x \approx \sum_n a_n x^n$:

```
>>> import numpy as np
>>> from scipy import optimize
>>> x = np.linspace(-3, 3, 100) # 100 values between -3 and 3
>>> def taylor_err(a, x, f):
...     p = np.arange(len(a))[:, np.newaxis]  # column vector
...     return f(x) - np.dot(a, x ** p)
>>> a_start = np.zeros(6)  # starting guess
>>> a_best, flag = optimize.leastsq(taylor_err, a_start,
...                                 args=(x, np.sin))
```

8.8. Uncertainties in the Data

In the opening section we introduced the problem of regression in its most general form. Computational complexity (particularly for the case of multivariate data) led us through a series of approximations that can be used in optimizing the likelihood and the prior (e.g., that the uncertainties have a Gaussian distribution, that the independent variables are error-free, that we can control complexity of the model, and that we can express the likelihood in terms of linear functions). Eventually we have to face the problem that many of these assumptions no longer hold for data in the wild. We have addressed the question of model complexity; working from linear to nonlinear regression. We now return to the question of *error behavior* and consider the uncertainty that is inherent in any analysis.

8.8.1. Uncertainties in the Dependent and Independent Axes

In almost all real-world applications, the assumption that one variable (the independent variable) is essentially free from any uncertainty is not valid. Both the dependent and independent variables will have measurement uncertainties. For our example data set we have assumed that the redshifts of the supernovas are known to a very high level of accuracy (i.e., that we measure these redshifts spectroscopically). If, for example, the redshifts of the supernovas were estimated based on the colors of the supernova (or host galaxy) then the errors on the redshift estimate can be significant. For our synthetic data we assume fractional uncertainties on the independent variable of 10%.

The impact of errors on the independent variables is a bias in the derived regression coefficients. This is straightforward to show if we consider a linear model with a dependent and an independent variable, y^* and x^*, respectively. We can write the objective function as before,

$$y_i^* = \theta_0 + \theta_1 x_i^*. \tag{8.56}$$

Now let us assume that we observe y and x, which are noisy representations of y^* and x^*, i.e.,

$$x_i = x_i^* + \delta_i, \tag{8.57}$$

$$y_i = y^* + \epsilon_i, \tag{8.58}$$

with δ and ϵ centered normal distributions.

Solving for y we get

$$y = \theta_0 + \theta_1(x_i - \delta_i) + \epsilon_i. \tag{8.59}$$

The uncertainty in x is now part of the regression equation and scales with the regression coefficients (biasing the regression coefficient). This problem is known in the statistics literature as *total least squares* and belongs to the class of *errors-in-variables* problems; see [4]. A very detailed discussion of regression and the problem of uncertainties from an astronomical perspective can be found in [8, 11].

How can we account for the measurement uncertainties in both the independent and dependent variables? Let us start with a simple example where we assume the errors are Gaussian so we can write the covariance matrix as

$$\Sigma_i = \begin{bmatrix} \sigma_{x_i}^2 & \sigma_{xy_i} \\ \sigma_{xy_i} & \sigma_{y_i}^2 \end{bmatrix}. \tag{8.60}$$

For a straight-line regression we express the slope of the line, θ_1, in terms of its normal vector,

$$\mathbf{n} = \begin{bmatrix} -\sin\alpha \\ \cos\alpha \end{bmatrix}, \tag{8.61}$$

where $\theta_1 = \arctan(\alpha)$ and α is the angle between the line and the x-axis. The covariance matrix projects onto this space as

$$S_i^2 = \mathbf{n}^T \Sigma_i \mathbf{n} \tag{8.62}$$

and the distance between a point and the line is given by (see [8])

$$\Delta_i = \mathbf{n}^T z_i - \theta_0 \cos\alpha, \tag{8.63}$$

where z_i represents the data point (x_i, y_i). The log-likelihood is then

$$\ln L = -\sum_i \frac{\Delta_i^2}{2S_i^2}. \tag{8.64}$$

Maximizing this likelihood for the regression parameters θ_0 and θ_1 is shown in figure 8.6, where we use the data from [8] with correlated uncertainties on the x and y components, and recover the underlying linear relation. For a single-parameter search (θ_1) the regression can be undertaken in a brute-force manner. As we increase the complexity of the model or the dimensionality of the data, the computational cost will grow and techniques such as MCMC must be employed (see [4]).

8.9. Regression That Is Robust to Outliers

A fact of experimental life is that if you can measure an attribute you can also measure it incorrectly. Despite the increase in fidelity of survey data sets, any regression or model fitting must be able to account for outliers from the fit. For the standard

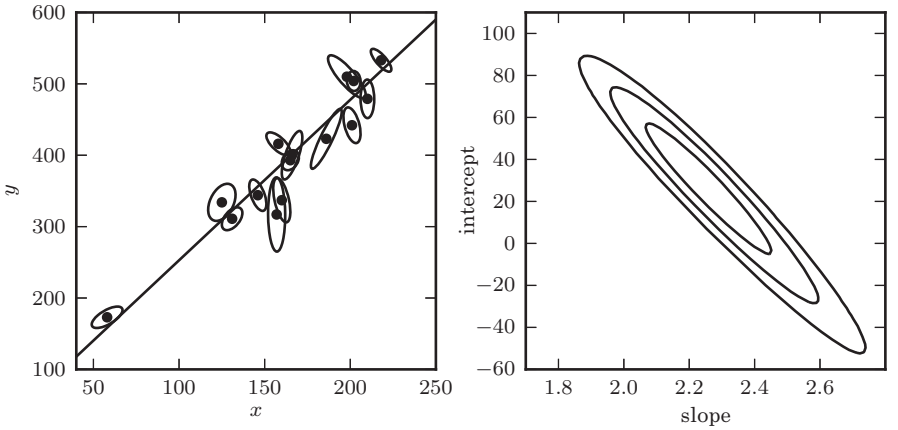

Figure 8.6. A linear fit to data with correlated errors in x and y. In the literature, this is often referred to as *total least squares* or *errors-in-variables* fitting. The left panel shows the lines of best fit; the right panel shows the likelihood contours in slope/intercept space. The points are the same set used for the examples in [8].

least-squares regression the use of an L_2 norm results in outliers that have substantial leverage in any fit (contributing as the square of the systematic deviation). If we knew $e(y_i|y)$ for all of the points in our sample (e.g., they are described by an exponential distribution where we would use the L_1 norm to define the error) then we would simply include the error distribution when defining the likelihood. When we do not have a priori knowledge of $e(y_i|y)$, things become more difficult. We can either model $e(y_i|y)$ as a mixture model (see §5.6.7) or assume a form for $e(y_i|y)$ that is less sensitive to outliers. An example of the latter would be the adoption of the L_1 norm, $\sum_i ||y_i - w_i x_i||$, which we introduced in §8.3, which is less sensitive to outliers than the L_2 norm (and was, in fact, proposed by Rudjer Bošković prior to the development of least-squares regression by Legendre, Gauss, and others [2]). Minimizing the L_1 norm is essentially finding the median. The drawback of this least absolute value regression is that there is no closed-form solution and we must minimize the likelihood space using an iterative approach.

Other approaches to robust regression adopt an approach that seeks to reject outliers. In the astronomical community this is usually referred to as *sigma clipping* and is undertaken in an iterative manner by progressively pruning data points that are not well represented by the model. Least-trimmed squares formalizes this, somewhat ad hoc approach, by searching for the subset of K points which minimize $\sum_i^K (y_i - \theta_i x_i)^2$. For large N the number of combinations makes this search expensive.

Complementary to outlier rejection are the Theil–Sen method [20] (or the Kendall robust line-fit method) and associated techniques. In these cases the regression is determined from the median of the slope, θ_1, calculated from all pairs of points within the data set. Given the slope, the offset or zero point, θ_0, can be defined from the median of $y_i - \theta_1 x_i$. Each of these techniques is simple to estimate and scales to large numbers.

M estimators (M stands for "maximum-likelihood-type") approach the problem of outliers by modifying the underlying likelihood estimator to be less sensitive than the classic L_2 norm. M estimators are a class of estimators that include many

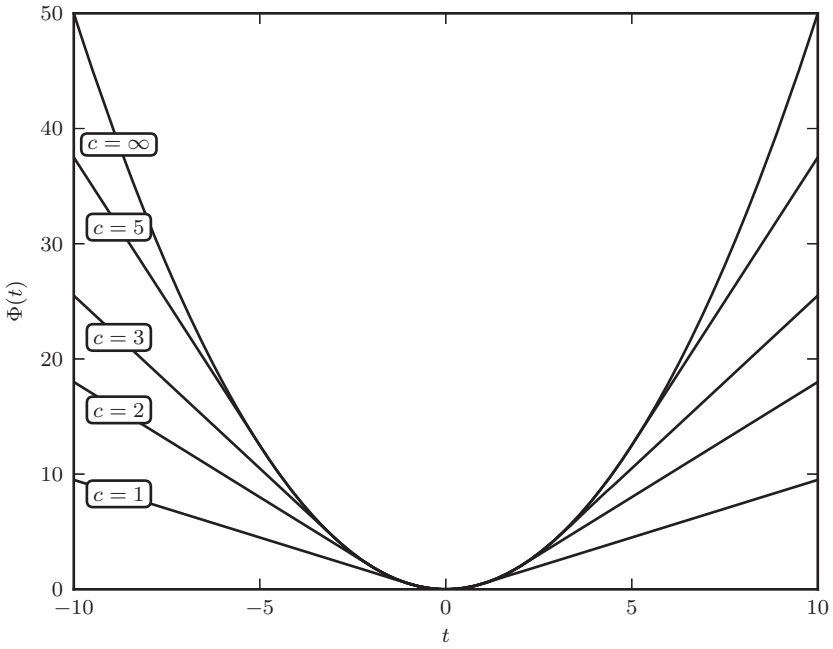

Figure 8.7. The Huber loss function for various values of c.

maximum-likelihood approaches (including least squares). They replace the standard least squares, which minimizes the sum of the squares of the residuals between a data value and the model, with a different function. Ideally the M estimator has the property that it increases less than the square of the residual and has a unique minimum at zero.

Huber Loss Function

An example of an M estimator that is common in robust regression is that of the Huber loss (or cost) function [9]. The Huber estimator minimizes

$$\sum_{i=1}^{N} e(y_i|y), \tag{8.65}$$

where

$$e(t) = \begin{cases} \frac{1}{2}t^2 & \text{if } |t| \le c, \\ c|t| - \frac{1}{2}c^2 & \text{if } |t| \ge c, \end{cases} \tag{8.66}$$

where the constant c must be chosen. Therefore, $e(t)$ is a function which acts like t^2 for $|t| \le c$ and like $|t|$ for $|t| > c$ and is continuous and differentiable (see figure 8.7). The transition in the Huber function is equivalent to assuming a Gaussian error distribution for small excursions from the true value of the function and an exponential distribution for large excursions (its behavior is a compromise between the mean and the median). Figure 8.8 shows an application of the Huber loss function to data with outliers. Outliers do have a small effect, and the slope of the Huber loss fit is closer to that of standard linear regression.

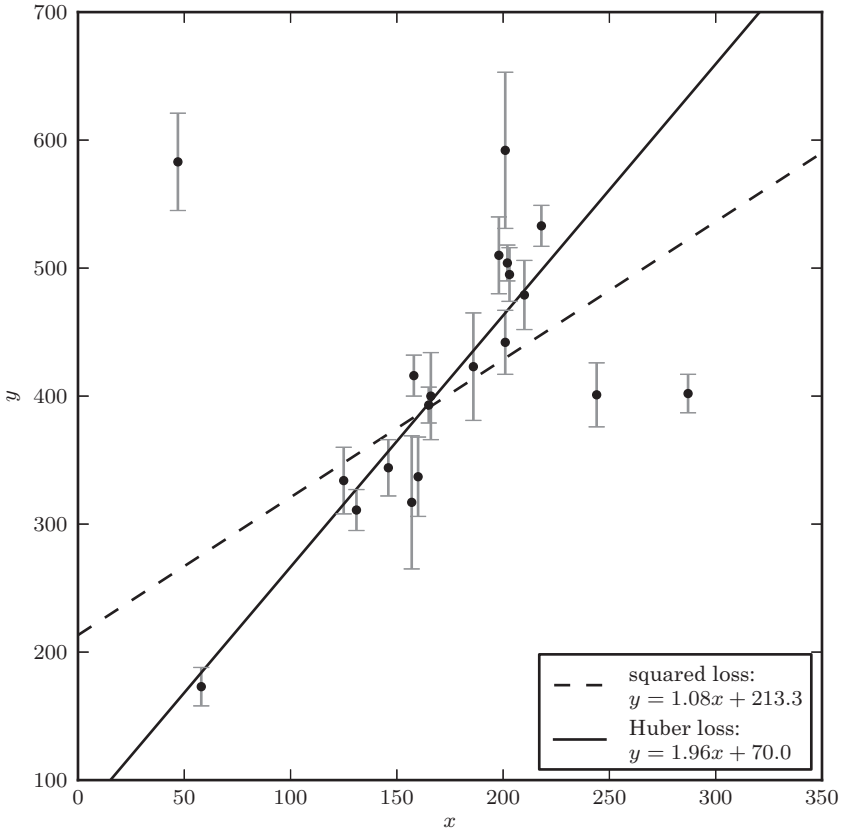

Figure 8.8. An example of fitting a simple linear model to data which includes outliers (data is from table 1 of [8]). A comparison of linear regression using the squared-loss function (equivalent to ordinary least-squares regression) and the Huber loss function, with $c = 1$ (i.e., beyond 1 standard deviation, the loss becomes linear).

8.9.1. Bayesian Outlier Methods

From a Bayesian perspective, one can use the techniques developed in chapter 5 within the context of a regression model in order to account for, and even to individually identify outliers (recall §5.6.7). Figure 8.9 again shows the data set used in figure 8.8, which contains three clear outliers. In a standard straight-line fit to the data, the result is strongly affected by these points. Though this standard linear regression problem is solvable in closed form (as it is in figure 8.8), here we compute the best-fit slope and intercept using MCMC sampling (and show the resulting contours in the upper-right panel).

The remaining two panels show two different Bayesian strategies for accounting for outliers. The main idea is to enhance the model such that it can naturally explain the presence of outliers. In the first model, we account for the outliers through the use of a mixture model, adding a background Gaussian component to our data. This is the regression analog of the model explored in §5.6.5, with the difference that here we are modeling the background as a wide Gaussian rather than a uniform distribution.

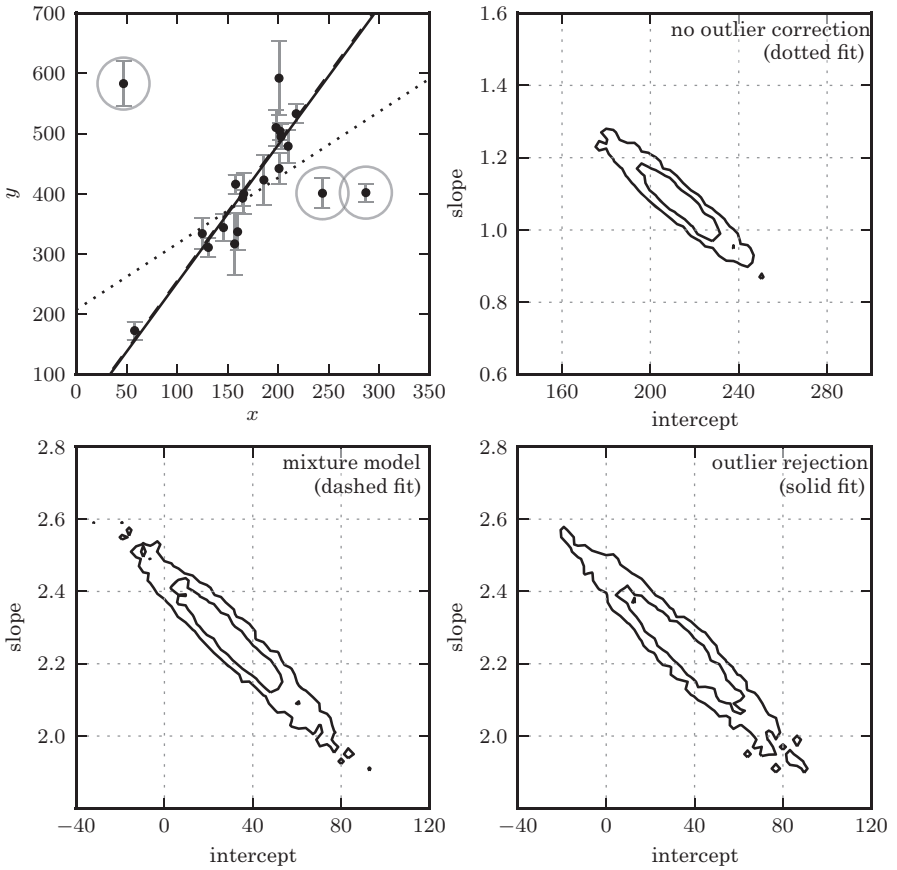

Figure 8.9. Bayesian outlier detection for the same data as shown in figure 8.8. The top-left panel shows the data, with the fits from each model. The top-right panel shows the 1-σ and 2-σ contours for the slope and intercept with no outlier correction: the resulting fit (shown by the dotted line) is clearly highly affected by the presence of outliers. The bottom-left panel shows the marginalized 1-σ and 2-σ contours for a mixture model (eq. 8.67). The bottom-right panel shows the marginalized 1-σ and 2-σ contours for a model in which points are identified individually as *good* or *bad* (eq. 8.68). The points which are identified by this method as bad with a probability greater than 68% are circled in the first panel.

The mixture model includes three additional parameters: μ_b and V_b, the mean and standard deviation of the background, and p_b, the probability that any point is an outlier. V_b implies a source of error in addition to the already existing measurement error for each point. With this model, the likelihood becomes (cf. eq. 5.83; see also [8])

$$p(\{y_i\}|\{x_i\},\{\sigma_i\},\theta_0,\theta_1,\mu_b,V_b,p_b) \propto \prod_{i=1}^{N}\left[\frac{1-p_b}{\sqrt{2\pi\sigma_i^2}}\exp\left(-\frac{(y_i-\theta_1 x_i-\theta_0)^2}{2\sigma_i^2}\right) + \frac{p_b}{\sqrt{2\pi(V_b+\sigma_i^2)}}\exp\left(-\frac{(y_i-\mu_b)^2}{2(V_b+\sigma_i^2)}\right)\right]. \tag{8.67}$$

Using MCMC sampling and marginalizing over the background parameters yields the dashed-line fit in figure 8.9. The marginalized posterior for this model is shown in the lower-left panel. This fit is much less affected by the outliers than is the simple regression model used above.

Finally, we can go further and perform an analysis analogous to that of §5.6.7, in which we attempt to identify bad points individually. In analogy with eq. 5.94 we can fit for nuisance parameters g_i, such that if $g_i = 1$, the point is a "good" point, and if $g_i = 0$ the point is a "bad" point. With this addition our model becomes

$$p(\{y_i\}|\{x_i\},\{\sigma_i\},\{g_i\},\theta_0,\theta_1,\mu_b,V_b) \propto \prod_{i=1}^{N}\left[\frac{g_i}{\sqrt{2\pi\sigma_i^2}}\exp\left(-\frac{(y_i-\theta_1 x_i-\theta_0)^2}{2\sigma_i^2}\right)\right. \\ \left. + \frac{1-g_i}{\sqrt{2\pi(V_b+\sigma_i^2)}}\exp\left(-\frac{(y_i-\mu_b)^2}{2(V_b+\sigma_i^2)}\right)\right]. \tag{8.68}$$

This model is very powerful: by marginalizing over all parameters but a particular g_i, we obtain a posterior estimate of whether point i is an outlier. Using this procedure, the "bad" points have been marked with a circle in the upper-left panel of figure 8.9. If instead we marginalize over the nuisance parameters, we can compute a two-dimensional posterior for the slope and intercept of the straight-line fit. The lower-right panel shows this posterior, after marginalizing over $\{g_i\}$, μ_b, and V_b. In this example, the result is largely consistent with the simpler Bayesian mixture model used above.

8.10. Gaussian Process Regression

Another powerful class of regression algorithms is *Gaussian process regression*. Despite its name, Gaussian process regression is widely applicable to data that are not generated by a Gaussian process, and can lead to very flexible regression models that are more data driven than other parametric approaches. There is a rich literature on the subject, and we will only give a cursory treatment here. An excellent book-length treatment of Gaussian processes can be found in [19].

A Gaussian process is a collection of random variables in parameter space, any subset of which is defined by a joint Gaussian distribution. It can be shown that a Gaussian process can be completely specified by its mean and covariance function. Gaussian processes can be defined in any number of dimensions for any positive covariance function. For simplicity, in this section we will consider one dimension and use a familiar squared-exponential covariance function,

$$\mathrm{Cov}(x_1,x_2;h) = \exp\left(\frac{-(x_1-x_2)^2}{2h^2}\right), \tag{8.69}$$

where h is the bandwidth. Given a chosen value of h, this covariance function specifies the statistics of an infinite set of possible functions $f(x)$. The upper-left panel of figure 8.10 shows three of the possible functions drawn from a zero-mean Gaussian

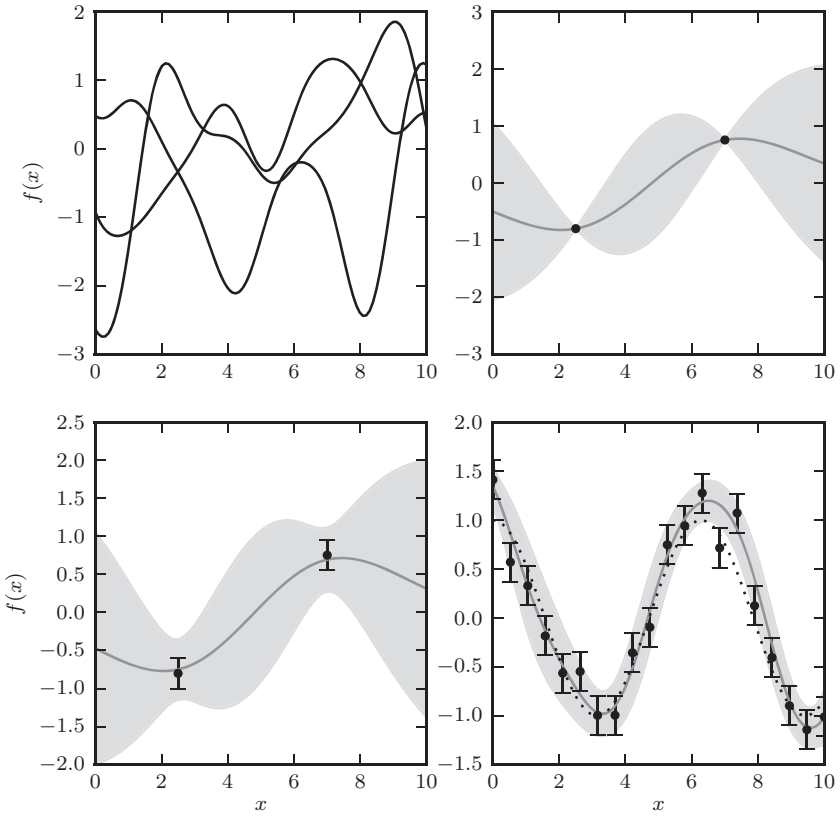

Figure 8.10. An example of Gaussian process regression. The upper-left panel shows three functions drawn from an unconstrained Gaussian process with squared-exponential covariance of bandwidth $h = 1.0$. The upper-right panel adds two constraints, and shows the 2-σ contours of the constrained function space. The lower-left panel shows the function space constrained by the points with error bars. The lower-right panel shows the function space constrained by 20 noisy points drawn from $f(x) = \cos x$.

process[3] with $h = 1.0$. This becomes more interesting when we specify constraints on the Gaussian process: that is, we select only those functions $f(x)$ which pass through particular points in the space. The remaining panels of figure 8.10 show this Gaussian process constrained by points without error (upper-right panel), points with error (lower-left panel), and a set of 20 noisy observations drawn from the function $f_{\text{true}}(x) = \cos x$. In each, the shaded region shows the 2-σ contour in which 95% of all possible functions $f(x)$ lie.

These constrained Gaussian process examples hint at how these ideas could be used for standard regression tasks. The Gaussian process regression problem can be formulated similarly to the other regression problems discussed above. We assume our data are drawn from an underlying model $f(x)$: that is, our observed data are $\{x_i, y_i = f(x_i) + \sigma_i\}$. Given the observed data, we desire an estimate of the mean value $\bar{f}_j^*$ and variance Σ_{jk}^* for a new set of measurements x_j^*. In Bayesian terms, we want to

[3]These functions are drawn by explicitly creating the $N \times N$ covariance matrix C from the functional form in eq. 8.69, and drawing N correlated Gaussian random variables with mean 0 and covariance C.

compute the posterior pdf

$$p(f_j|\{x_i, y_i, \sigma_i\}, x_j^*). \tag{8.70}$$

This amounts to averaging over the *entire set* of possible functions $f(x)$ which pass through our constraints. This seemingly infinite calculation can be made tractable using a "kernel trick," as outlined in [19], transforming from the infinite function space to a finite covariance space. The result of this mathematical exercise is that the Gaussian process regression problem can be formulated as follows.

Using the assumed form of the covariance function (eq. 8.69), we compute the covariance matrix

$$K = \begin{pmatrix} K_{11} & K_{12} \\ K_{12}^T & K_{22} \end{pmatrix}, \tag{8.71}$$

where K_{11} is the covariance between the input points x_i with observational errors σ_i^2 added in quadrature to the diagonal, K_{12} is the cross-covariance between the input points x_i and the unknown points x_j^*, and K_{22} is the covariance between the unknown points x_j^*. Then for observed vectors $\mathbf{x}$ and $\mathbf{y}$, and a vector of unknown points $\mathbf{x}^*$, it can be shown that the posterior in eq. 8.70 is given by

$$p(f_j|\{x_i, y_i, \sigma_i\}, x_j^*) = \mathcal{N}(\mu, \Sigma), \tag{8.72}$$

where

$$\mu = K_{12}^T K_{11}^{-1} \mathbf{y}, \tag{8.73}$$

$$\Sigma = K_{22} - K_{12}^T K_{11}^{-1} K_{12}. \tag{8.74}$$

Then μ_j gives the expected value $\bar{f}_j^*$ of the result, and Σ_{jk} gives the error covariance between any two unknown points. Note that the physics of the underlying process enters through the assumed form of the covariance function (e.g., eq. 8.69; for an analysis of several plausible covariance functions in the astronomical context of quasar variability; see [24]).

A powerful and general framework for Gaussian process regression is available in Scikit-learn. It can be used as follows:

```
>>> import numpy as np
>>> import sklearn.gaussian_process as gp

>>> X = np.random.random((100, 2))  # 100 pts in 2 dims
>>> kernel1 = gp.kernels.RBF()
>>> y = np.sin(10 * X[:, 0] + X[:, 1])
>>> gpr = gp.GaussianProcessRegressor(kernel=kernel1)
>>> gpr.fit(X, y)
>>> y_pred, dy_pred = gpr.predict(X, return_std=True)
```

The `predict` method can optionally return the mean square error, which is the diagonal of the covariance matrix of the fit. For more details, see the source code of the figures in this chapter, as well as the Scikit-learn documentation.

In figure 8.11, we show a Gaussian process regression analysis of the supernova data set used above. The model is very well constrained near $z = 0.6$, where there is a lot of data, but not well constrained at higher redshifts. This is an important feature of Gaussian process regression: the analysis produces not only a best-fit model, but an uncertainty at each point, as well as a full covariance estimate of the result at unknown points.

The results shown in figure 8.11 depend on correctly tuning the correlation function bandwidth h. If h is too large or too small, the particular Gaussian process model will be unsuitable for the data and the results will be biased. Figure 8.11 uses a simple cross-validation approach to decide on the best value of h: we will discuss cross-validation in the following section. Sometimes, the value of h is scientifically an interesting outcome of the analysis; such an example is discussed in the context of the damped random walk model in §10.5.4.

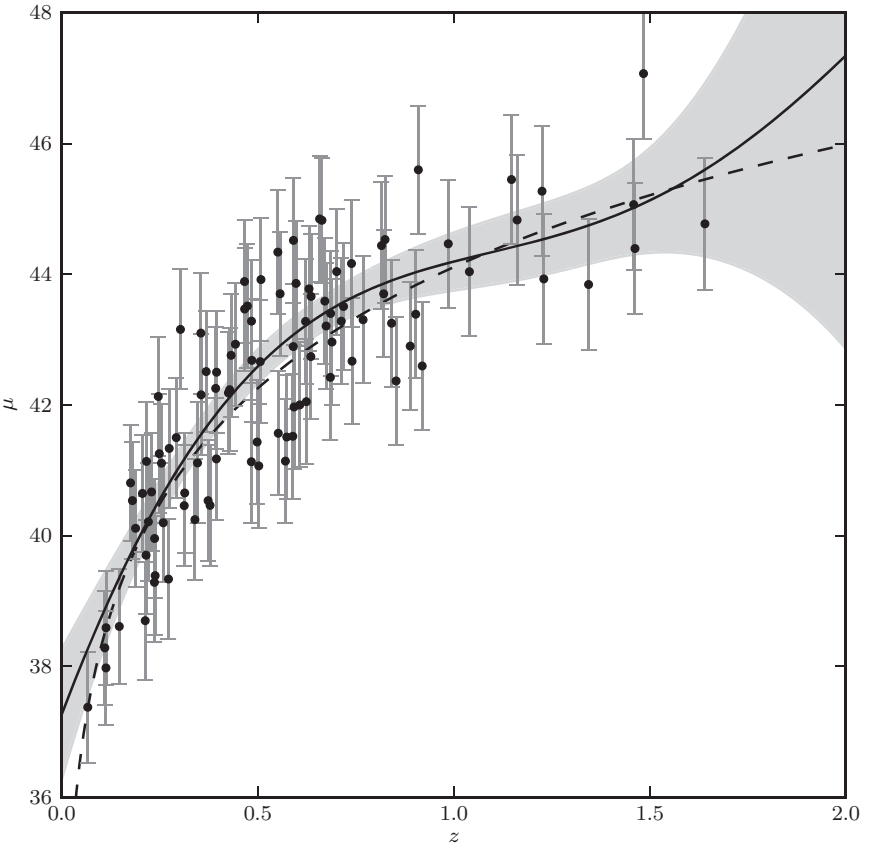

Figure 8.11. A Gaussian process regression analysis of the simulated supernova sample used in figure 8.2. This uses a squared-exponential covariance model, with bandwidth learned through cross-validation.

8.11. Overfitting, Underfitting, and Cross-Validation

When using regression, whether from a Bayesian or maximum likelihood perspective, it is important to recognize some of the potential pitfalls associated with these methods. As noted above, the optimality of the regression is contingent on correct model selection. In this section we explore cross-validation methods which can help determine whether a potential model is a good fit to the data. These techniques are complementary to the model selection techniques such as AIC and BIC discussed in §4.3. This section will introduce the important topics of overfitting and underfitting, bias and variance, and introduces the frequentist tool of cross-validation to understand these.

Here, for simplicity, we will consider the example of a simple one-dimensional model with homoscedastic errors, though the results of this section naturally generalize to more sophisticated models. As above, our observed data is x_i, and we're trying to predict the dependent variable y_i. We have a training sample in which we have observed both x_i and y_i, and an unknown sample for which only x_i is measured. For example, you may be looking at the fundamental plane for elliptical galaxies, and trying to predict a galaxy's central black hole mass given the velocity dispersion and surface brightness of the stars. Here y_i is the mass of the black hole, and x_i is a vector of a length two consisting of velocity dispersion and surface brightness measurements.

Throughout the rest of this section, we will use a simple model where x and y satisfy the following:

$$\begin{aligned} 0 \le x_i \le 3, \\ y_i = x_i \sin(x_i) + \epsilon_i, \end{aligned} \tag{8.75}$$

where the noise is drawn from a normal distribution $\epsilon_i \sim \mathcal{N}(0, 0.1)$. The values for 20 regularly spaced points are shown in figure 8.12.

We will start with a simple straight-line fit to our data. The model is described by two parameters, the slope of the line, θ_1, and the y-axis intercept, θ_0, and is found by minimizing the mean square error,

$$\epsilon = \frac{1}{N} \sum_{i=1}^{N} (y_i - \theta_0 - \theta_1 x_i)^2. \tag{8.76}$$

The resulting best-fit line is shown in figure 8.12. It is clear that a straight line is not a good fit: it does not have enough flexibility to accurately model the data. We say in this case that the model is biased, and that it underfits the data.

What can be done to improve on this? One possibility is to make the model more sophisticated by increasing the degree of the polynomial (see §8.2.2). For example, we could fit a quadratic function, or a cubic function, or in general a d-degree polynomial. A more complicated model with more free parameters should be able to fit the data much more closely. The panels of figure 8.13 show the best-fit polynomial model for three different choices of the polynomial degree d.

As the degree of the polynomial increases, the best-fit curve matches the data points more and more closely. In the extreme of $d = 19$, we have 20 degrees of freedom

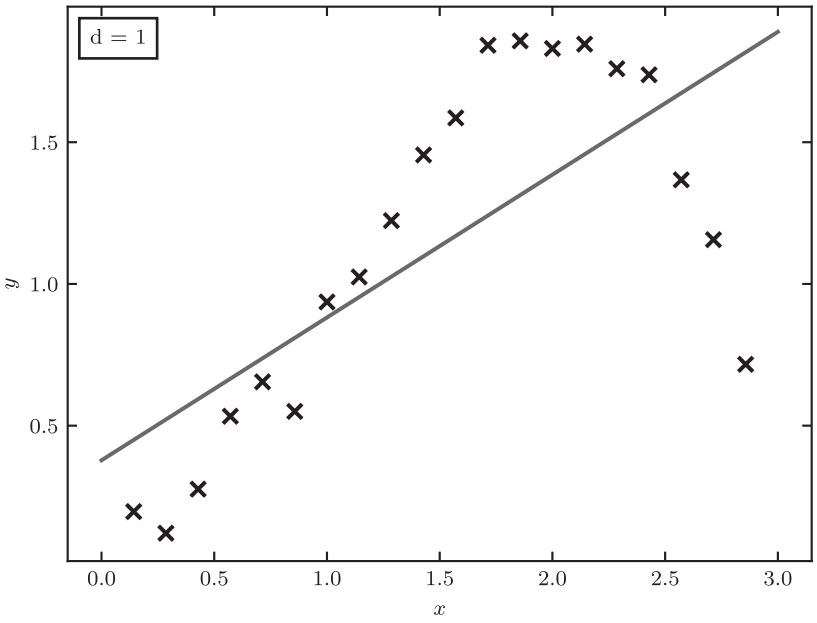

Figure 8.12. Our toy data set described by eq. 8.75. Shown is the line of best fit, which quite clearly underfits the data. In other words, a linear model in this case has high bias.

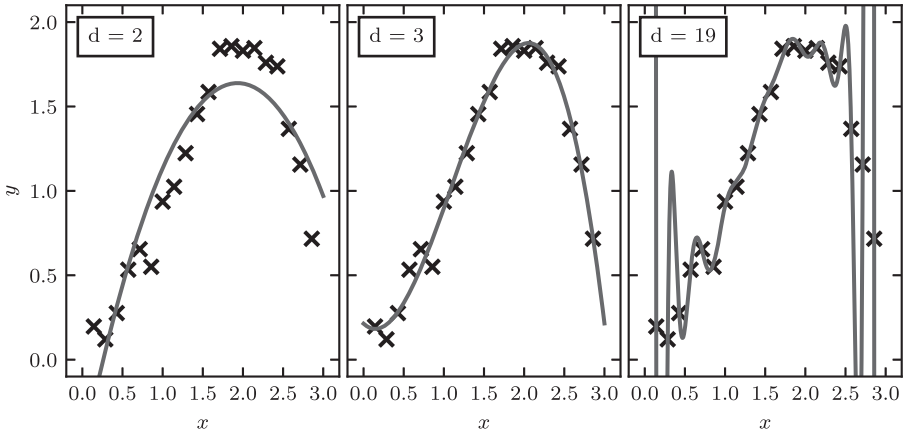

Figure 8.13. Three models of increasing complexity applied to our toy data set (eq. 8.75). The $d = 2$ model, like the linear model in figure 8.12, suffers from high bias, and underfits the data. The $d = 19$ model suffers from high variance, and overfits the data. The $d = 3$ model is a good compromise between these extremes.

with 20 data points, and the training error given by eq. 8.76 can be reduced to zero (though numerical issues can prevent this from being realized in practice). Unfortunately, it is clear that the $d = 19$ polynomial is not a better fit to our data as a whole: the wild swings of the curve in the spaces between the training points are not a good description of the underlying data model. The model suffers from high variance; it overfits the data. The term *variance* is used here because a small perturbation of one

of the training points in the $d = 19$ model can change the best-fit model by a large magnitude. In a high-variance model, the fit varies strongly depending on the exact set or subset of data used to fit it.

The center panel of figure 8.13 shows a $d = 3$ model which balances the trade-off between bias and variance: it does not display the high bias of the $d = 2$ model, and does not display high variance like the $d = 19$ model. For simple two-dimensional data like that seen here, the bias/variance trade-off is easy to visualize by plotting the model along with the input data. But this strategy is not as fruitful as the number of data dimensions grows. What we need is a general measure of the goodness of fit of different models to the training data. As displayed above, the mean square error does not paint the whole picture: increasing the degree of the polynomial in this case can lead to smaller and smaller training errors, but this reflects overfitting of the data rather than an improved approximation of the underlying model.

An important practical aspect of regression analysis lies in addressing this deficiency of the training error as an evaluation of goodness of fit, and finding a model which best compromises between high bias and high variance. To this end, the process of *cross-validation* can be used to quantitatively evaluate the bias and variance of a regression model.

8.11.1. Cross-Validation

There are several possible approaches to cross-validation. We will discuss one approach in detail here, and list some alternative approaches at the end of the section. The simplest approach to cross-validation is to split the training data into three parts: the training set, the cross-validation set, and the test set. As a rule of thumb, the training set should comprise 50%–70% of the original training data, while the remainder is divided equally into the cross-validation set and test set.

The training set is used to determine the parameters of a given model (i.e., the optimal values of θ_j for a given choice of d). Using the training set, we evaluate the training error $\epsilon_{\rm tr}$ using eq. 8.76. The cross-validation set is used to evaluate the cross-validation error $\epsilon_{\rm cv}$ of the model, also via eq. 8.76. Because this cross-validation set was not used to construct the fit, the cross-validation error will be large for a high-bias (overfit) model, and better represents the true goodness of fit of the model. With this in mind, the model which minimizes this cross-validation error is likely to be the best model in practice. Once this model is determined, the test error is evaluated using the test set, again via eq. 8.76. This test error gives an estimate of the reliability of the model for an unlabeled data set.

Why do we need a test set as well as a cross-validation set? In one sense, just as the parameters (in this case, θ_j) are learned from the training set, the so-called hyperparameters—those parameters which describe the complexity of the model (in this case, d)—are learned from the cross-validation set. In the same way that the parameters can be overfit to the training data, the hyperparameters can be overfit to the cross-validation data, and the cross-validation error gives an overly optimistic estimate of the performance of the model on an unlabeled data set. The test error is a better representation of the error expected for a new set of data. This is why it is recommended to use both a cross-validation set and a test set in your analysis.

A useful way to use the training error and cross-validation error to evaluate a model is to look at the results graphically. Figure 8.14 shows the training error and

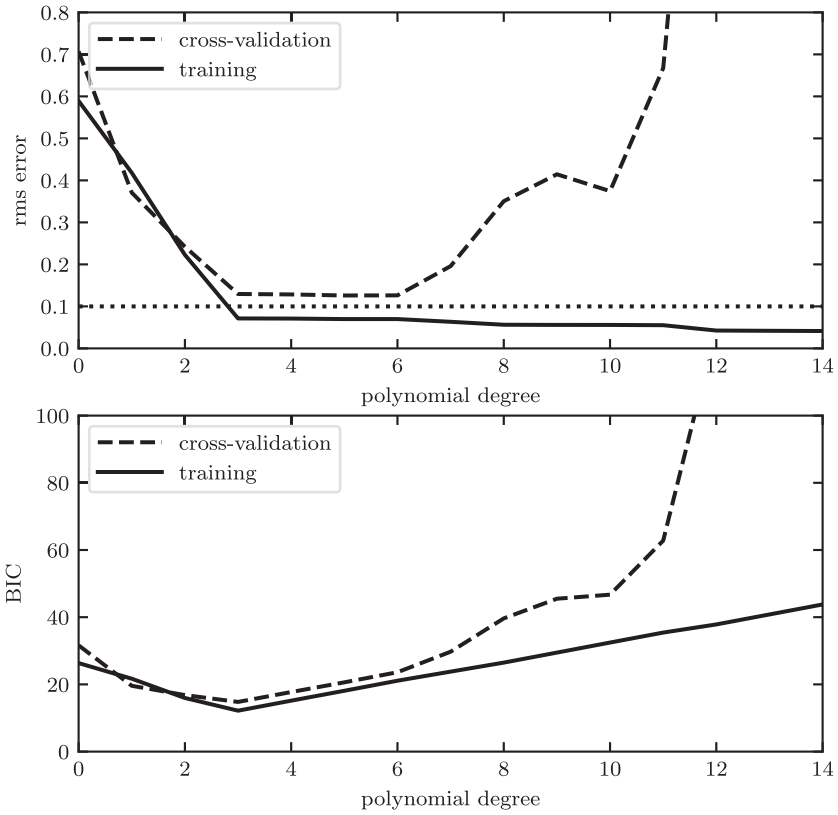

Figure 8.14. The top panel shows the root-mean-square (rms) training error and cross-validation error for our toy model (eq. 8.75) as a function of the polynomial degree d. The horizontal dotted line indicates the level of intrinsic scatter in the data. Models with polynomial degree from 3 to 5 minimize the cross-validation rms error. The bottom panel shows the Bayesian information criterion (BIC) for the training and cross-validation subsamples. According to the BIC, a degree-3 polynomial gives the best fit to this data set.

cross-validation error for the data in figure 8.13 as a function of the polynomial degree d. For reference, the dotted line indicates the level of intrinsic scatter added to our data.

The broad features of this plot reflect what is generally seen as the complexity of a regression model is increased: for small d, we see that both the training error and cross-validation error are very high. This is the tell-tale indication of a high-bias model, in which the model underfits the data. Because the model does not have enough complexity to describe the intrinsic features of the data, it performs poorly for both the training and cross-validation sets.

For large d, we see that the training error becomes very small (smaller than the intrinsic scatter we added to our data) while the cross-validation error becomes very large. This is the tell-tale indication of a high-variance model, in which the model overfits the data. Because the model is overly complex, it can match subtle variations in the training set which do not reflect the underlying distribution. Plotting this sort of information is a very straightforward way to settle on a suitable model. Of course,

AIC and BIC provide another way to choose optimal d. Here both methods would choose the model with the best possible cross-validation error: $d = 3$.

8.11.2. Learning Curves

One question that cross-validation does not directly address is that of how to improve a model that is not giving satisfactory results (e.g., the cross-validation error is much larger than the known errors). There are several possibilities:

1. **Get more training data**. Often, using more data to train a model can lead to better results. Surprisingly, though, this is not always the case.
2. **Use a more/less complicated model**. As we saw above, the complexity of a model should be chosen as a balance between bias and variance.
3. **Use more/less regularization**. Including regularization, as we saw in the discussion of ridge regression (see §8.3.1) and other methods above, can help with the bias/variance trade-off. In general, increasing regularization has a similar effect to decreasing the model complexity.
4. **Increase the number of features**. Adding more observations of each object in your set can lead to a better fit. But this may not always yield the best results.

The choice of which route to take is far beyond a simple philosophical matter: for example, if you desire to improve your photometric redshifts for a large astronomical survey, it is important to evaluate whether you stand to benefit more from increasing the size of the training set (i.e., gathering spectroscopic redshifts for more galaxies) or from increasing the number of observations of each galaxy (i.e., reobserving the galaxies through other passbands). The answer to this question will inform the allocation of limited, and expensive, telescope time.

Note that this is a fundamentally different question than that explored above. There, we had a fixed data set, and were trying to determine the best model. Here, we assume a fixed model, and are asking how to improve the data set. One way to address this question is by plotting learning curves. A learning curve is the plot of the training and cross-validation error as a function of the number of training points. The details are important, so we will write this out explicitly.

Let our model be represented by the set of parameters $\boldsymbol{\theta}$. In the case of our simple example, $\boldsymbol{\theta} = \{\theta_0, \theta_1, \ldots, \theta_d\}$. We will denote by $\boldsymbol{\theta}^{(n)} = \{\theta_0^{(n)}, \theta_1^{(n)}, \ldots, \theta_d^{(n)}\}$ the model parameters which best fit the first n points of the training data: here $n \leq N_{\rm train}$, where $N_{\rm train}$ is the total number of training points. The truncated training error for $\theta^{(n)}$ is given by

$$\epsilon_{\rm tr}^{(n)} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left[y_i - \sum_{m=0}^{d}\theta_0^{(n)}x_i^m\right]^2}. \tag{8.77}$$

Note that the training error $\epsilon_{\rm tr}^{(n)}$ is evaluated using only the n points on which the model parameters $\theta^{(n)}$ were trained, not the full set of $N_{\rm train}$ points. Similarly, the truncated cross-validation error is given by

$$\epsilon_{\rm cv}^{(n)} = \sqrt{\frac{1}{n}\sum_{i=1}^{N_{\rm cv}}\left[y_i - \sum_{m=0}^{d}\theta_0^{(n)}x_i^m\right]^2}, \tag{8.78}$$

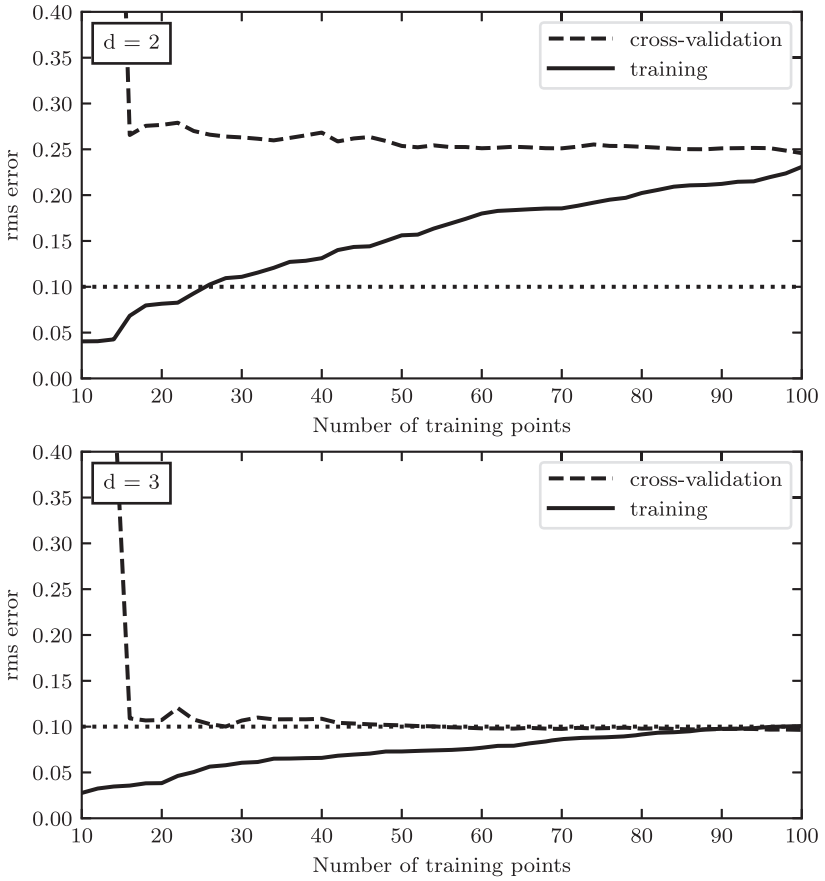

Figure 8.15. The learning curves for the data given by eq. 8.75, with $d = 2$ and $d = 3$. Both models have high variance for a few data points, visible in the spread between training and cross-validation error. As the number of points increases, it is clear that $d = 2$ is a high-bias model which cannot be improved simply by adding training points.

where we sum over *all* of the cross-validation points. A learning curve is the plot of the truncated training error and truncated cross-validation error as a function of the size n of the training set used. For our toy example, this plot is shown in figure 8.15 for models with $d = 2$ and $d = 3$. The dotted line in each panel again shows for reference the intrinsic error added to the data.

The two panels show some common features, which are reflective of the features of learning curves for any regression model:

1. As we increase the size of the training set, the training error increases. The reason for this is simple: a model of a given complexity can better fit a small set of data than a large set of data. Moreover, aside from small random fluctuations, we expect this training error to always increase with the size of the training set.
2. As we increase the size of the training set, the cross-validation error decreases. The reason for this is easy to see: a smaller training set leads to overfitting the model, meaning that the model is less representative of the cross-validation

data. As the training set grows, overfitting is reduced and the cross-validation error decreases. Again, aside from random fluctuations, and as long as the training set and cross-validation set are statistically similar, we expect the cross-validation error to always decrease as the training set size grows.

3. The training error is everywhere less than or equal to the cross-validation error, up to small statistical fluctuations. We expect the model on average to better describe the data used to train it.
4. The logical outcome of the above three observations is that as the size N of the training set becomes large, the training and cross-validation curves will converge to the same value.

By plotting these learning curves, we can quickly see the effect of adding more training data. When the curves are separated by a large amount, the model error is dominated by variance, and additional training data will help. For example, in the lower panel of figure 8.15, at $N = 15$, the cross-validation error is very high and the training error is very low. Even if we only had 15 points and could not plot the remainder of the curve, we could infer that adding more data would improve the model.

On the other hand, when the two curves have converged to the same value, the model error is dominated by bias, and adding additional training data cannot improve the results *for that model*. For example, in the top panel of figure 8.15, at $N = 100$ the errors have nearly converged. Even without additional data, we can infer that adding data will not decrease the error below about 0.23. Improving the error in this case requires a more sophisticated model, or perhaps more features measured for each point.

To summarize, plotting learning curves can be very useful for evaluating the efficiency of a model and potential paths to improving your data. There are two possible situations:

1. **The training error and cross-validation error have converged**. In this case, increasing the number of training points under the same model is futile: the error cannot improve further. This indicates a model error dominated by bias (i.e., it is underfitting the data). For a high-bias model, the following approaches may help:
    - Add additional features to the data.
    - Increase the model complexity.
    - Decrease the regularization.
2. **The training error is much smaller than the cross-validation error**. In this case, increasing the number of training points is likely to improve the model. This condition indicates that the model error is dominated by variance (i.e., it is overfitting the data). For a high-variance model, the following approaches may help:
    - Increase the training set size.
    - Decrease the model complexity.
    - Increase the amplitude of the regularization.

Finally, we note a few caveats: first, the learning curves seen in figure 8.15 and the model complexity evaluation seen in figure 8.14 are actually aspects of a three-dimensional space. Changing the data and changing the model go hand in hand, and

one should always combine the two diagnostics to seek the best match between model and data.

Second, this entire discussion assumes that the training data, cross-validation data, and test data are statistically similar. This analysis will fail if the samples are drawn from different distributions, or have different measurement errors or different observational limits.

8.11.3. Other Cross-Validation Techniques

There are numerous cross-validation techniques available which are suitable for different situations. It is easy to generalize from the above discussion to these various cross-validation strategies, so we will just briefly mention them here.

Twofold Cross-Validation

Above, we split the data into a training set d_1, a cross-validation set d_2, and a test set d_0. Our simple tests involved training the model on d_0 and cross-validating the model on d_1. In twofold cross-validation, this process is repeated, training the model on d_1 and cross-validating the model on d_0. The training error and cross-validation error are computed from the mean of the errors in each fold. This leads to more robust determination of the cross-validation error for smaller data sets.

K-fold Cross-Validation

A generalization of twofold cross-validation is K-fold cross-validation. Here we split the data into $K + 1$ sets: the test set d_0, and the cross-validation sets $d_1, d_2, \ldots, d_K$. We train K different models, each time leaving out a single subset to measure the cross-validation error. The final training error and cross-validation error can be computed using the mean or median of the set of results. The median can be a better statistic than the mean in cases where the subsets d_i contain few points.

Leave-One-Out Cross-Validation

At the extreme of K-fold cross-validation is leave-one-out cross-validation. This is essentially the same as K-fold cross-validation, but this time our sets $d_1, d_2, \ldots, d_K$ have only one data point each. That is, we repeatedly train the model, leaving out only a single point to estimate the cross-validation error. Again, the final training error and cross-validation error are estimated using the mean or median of the individual trials. This can be useful when the size of the data set is very small, so that significantly reducing the number of data points leads to much different model characteristics.

Random Subset Cross-Validation

In this approach, the cross-validation set and training set are selected by randomly partitioning the data, and repeating any number of times until the error statistics are well sampled. The disadvantage here is that not every point is guaranteed to be used both for training and for cross-validation. Thus, there is a finite chance that an outlier

can lead to spurious results. For N points and P random samplings of the data, this situation becomes very unlikely for $N/2^P \ll 1$.

8.11.4. Summary of Cross-Validation and Learning Curves

In this section we have shown how to evaluate how well a model fits a data set through cross-validation. This is one practical route to the model selection ideas presented in chapters 4–5. We have covered how to determine the best model given a data set (§8.11.1) and how to address both the model and the data together to improve results (§8.11.2).

Cross-validation is one place where machine learning and data mining may be considered more of an art than a science. The optimal route to improving a model is not always straightforward. We hope that by following the suggestions in this chapter, you can apply this art successfully to your own data.

8.12. Which Regression Method Should I Use?

As we did in the last two chapters, we will use the axes of *accuracy*, *interpretability*, *simplicity*, and *speed* (see §6.6 for a description of these terms) to provide a rough guide to the trade-offs in choosing between the different regression methods described in this chapter.

What are the most *accurate* regression methods? The starting point is basic linear regression. Adding ridge or LASSO regularization in principle increases accuracy, since as long as $\lambda = 0$ is among the options tried for λ, it provides a superset of possible complexity trade-offs to be tried. This goes along with the general principle that models with more parameters have a greater chance of fitting the data well. Adding the capability to incorporate measurement errors should in principle increase accuracy, assuming of course that the error estimates are accurate enough. Principal component regression should generally increase accuracy by treating collinearity and effectively denoising the data. All of these methods are of course linear—the largest leap in accuracy is likely to come when going from linear to nonlinear models. A starting point for increasing accuracy through nonlinearity is a linear model on nonlinear transformations of the original data. The extent to which this approach increases accuracy depends on the sensibility of the transformations done, since the nonlinear functions introduced are chosen manually rather than automatically. The sensibility can be diagnosed in various ways, such as checking the Gaussianity of the resulting errors. Truly nonlinear models, in the sense that the variable interactions can be nonlinear as well, should be more powerful in general. The final significant leap of accuracy will generally come from going to nonparametric methods such as kernel regression, starting with Nadaraya–Watson regression and more generally, local polynomial regression. Gaussian process regression is typically even more powerful in principle, as it effectively includes an aspect similar to that of kernel regression through the covariance matrix, but also learns coefficients on each data point.

What are the most *interpretable* regression methods? Linear methods are easy to interpret in terms of understanding the relative importance of each variable through

the coefficients, as well as reasoning about how the model will react to different inputs. Ridge or LASSO regression in some sense increase interpretability by identifying the most important features. Because feature selection happens as the result of an overall optimization, reasoning deeply about why particular features were kept or eliminated is not necessarily fruitful. Bayesian formulations, as in the case of models that incorporate errors, add to interpretability by making assumptions clear. PCR begins to decrease interpretability in the sense that the columns are no longer identifiable in their original terms. Moving to nonlinear methods, generalized linear models maintain the identity of the original columns, while general nonlinear models tend to become much less interpretable. Kernel regression methods can arguably be relatively interpretable, as their behavior can be reasoned about in terms of distances between points. Gaussian process regression is fairly opaque, as it is relatively difficult to reason about the behavior of the inverse of the data covariance matrix.

What are the *simplest* regression methods? Basic linear regression has no tunable parameters, so it qualifies as the simplest out-of-the-box method. Generalized linear regression is similar, aside from the manual preparation of the nonlinear features. Ridge and LASSO regression require that only one parameter is tuned, and the optimization is convex, so that there is no need for random restarts, and cross-validation can make learning fairly automatic. PCR is similar in that there is only one critical parameter and the optimization is convex. Bayesian formulations of regression models which result in MCMC are subject to our comments in §5.11 regarding fiddliness (as well as computational cost). Nadaraya–Watson regression typically has only one critical parameter, the bandwidth. General local polynomial regression can be regarded as having another parameter, the polynomial order. Basic Gaussian process regression has only one critical parameter, the bandwidth of the covariance kernel, and is convex, though extensions with several additional parameters are often used.

What are the most *scalable* regression methods? Methods based on linear regression are fairly tractable using state-of-the-art linear algebra methods, including basic linear regression, ridge regression, and generalized linear regression, assuming the dimensionality is not excessively high. LASSO requires the solution of a linear program (LP), which becomes expensive as the dimension rises. For PCR, see our comments in §8.4 regarding PCA and SVD. Kernel regression methods are naively $\mathcal{O}(N^2)$ but can be sped up by fast tree-based algorithms in ways similar to those discussed in §2.5.2 and chapter 6. Certain approximate algorithms exist for Gaussian process regression, which is naively $\mathcal{O}(N^3)$, but GP is by far the most expensive among the methods we have discussed, and difficult to speed up algorithmically while maintaining high predictive accuracy.

Other considerations, and taste. Linear methods can be straightforwardly augmented to handle missing values. The Bayesian approach to linear regression incorporates measurement errors, while standard versions of machine learning methods do not incorporate errors. Gaussian process regression typically comes with posterior uncertainty bands, though confidence bands can be obtained for any method, as we have discussed in previous chapters. Regarding taste, LASSO is embedded in much recent work on sparsity and is related to work on compressed sensing, and Gaussian

TABLE 8.1.
A summary of the practical properties of different regression methods.

Method	Accuracy	Interpretability	Simplicity	Speed
Linear regression	L	H	H	H
Linear basis function regression	M	M	M	M
Ridge regression	L	H	M	H
LASSO regression	L	H	M	L
PCA regression	M	M	M	M
Nadaraya–Watson regression	M/H	M	H	L/M
Local linear/polynomial regression	H	M	M	L/M
Nonlinear regression	M	H	L	L/M

process regression has interesting interpretations in terms of priors in function space and in terms of kernelized linear regression.

Simple summary

We summarize our discussion in table 8.1, in terms of *accuracy*, *interpretability*, *simplicity*, and *speed* (see §6.6 for a description of these terms), given in simple terms of *high* (H), *medium* (M), and *low* (L).

References

[1] Astier, P., J. Guy, N. Regnault, and others (2005). The supernova legacy survey: Measurement of ω_m, ω_λ and w from the first year data set. *Astronomy & Astrophysics 447*(1), 24.

[2] Boscovich, R. J. (1757). De litteraria expeditione per pontificiam ditionem, et synopsis amplioris operis, ac habentur plura ejus ex exemplaria etiam sensorum impressa. *Bononiensi Scientarum et Artum Instituto Atque Academia Commentarii IV*, 353–396.

[3] Cleveland, W. S. (1979). Robust locally weighted regression and smoothing scatterplots. *Journal of the American Statistical Association 74*(368), 829–836.

[4] Dellaportas, P. and D. A. Stephens (1995). Bayesian analysis of errors-in-variables regression models. *Biometrics 51*(3), 1085–1095.

[5] Efron, B., T. Hastie, I. Johnstone, and R. Tibshirani (2004). Least angle regression. *Annals of Statistics 32*(2), 407–451.

[6] Gauss, K. F. (1809). *Theoria Motus Corporum Coelestium in Sectionibus Conicis Solem Ambientium*. Hamburg: Sumtibus F. Perthes et I. H. Besser.

[7] Hogg, D. W., I. K. Baldry, M. R. Blanton, and D. J. Eisenstein (2002). The K correction. *ArXiv:astro-ph/0210394*.

[8] Hogg, D. W., J. Bovy, and D. Lang (2010). Data analysis recipes: Fitting a model to data. *ArXiv:astro-ph/1008.4686*.

[9] Huber, P. J. (1964). Robust estimation of a local parameter. *Annals of Mathematical Statistics 35*, 73–101.

[10] Jolliffe, I. T. (1986). *Principal Component Analysis*. Springer.

[11] Kelly, B. C. (2011). Measurement error models in astronomy. *ArXiv:astro-ph/1112.1745*.

[12] Kim, J., Y. Kim, and Y. Kim (2008). A gradient-based optimization algorithm for LASSO. *Journal of Computational and Graphical Statistics 17*(4), 994–1009.

[13] Krajnović, D. (2011). A Jesuit anglophile: Rogerius Boscovich in England. *Astronomy and Geophysics 52*(6), 060000–6.

[14] Legendre, A. M. (1805). *Nouvelles méthodes pour la détermination des orbites des comètes*. Courcier.

[15] Levenberg, K. (1944). A method for the solution of certain non-linear problems in least squares. *Quarterly Applied Mathematics II(2)*, 164–168.

[16] Marquardt, D. W. (1963). An algorithm for least-squares estimation of non-linear parameters. *Journal of the Society of Industrial and Applied Mathematics 11*(2), 431–441.

[17] Mertens, B., T. Fearn, and M. Thompson (1995). The efficient cross-validation of principal components applied to principal component regression. *Statistics and Computing 5*, 227–235. 10.1007/BF00142664.

[18] Nadaraya, E. A. (1964). On estimating regression. *Theory of Probability and its Applications 9*, 141–142.

[19] Rasmussen, C. and C. Williams (2005). *Gaussian Processes for Machine Learning*. Adaptive Computation And Machine Learning. MIT Press.

[20] Theil, H. (1950). A rank invariant method of linear and polynomial regression analysis, I, II, III. *Proceedings of the Koninklijke Nederlandse Akademie Wetenschappen, Series A – Mathematical Sciences 53*, 386–392, 521–525, 1397–1412.

[21] Tibshirani, R. J. (1996). Regression shrinkage and selection via the Lasso. *Journal of the Royal Statistical Society, Series B 58*(1), 267–288.

[22] Tikhonov, A. N. (1995). *Numerical Methods for the Solution of Ill-Posed Problems*, Volume 328 of *Mathematics and Its Applications*. Kluwer.

[23] Watson, G. S. (1964). Smooth regression analysis. *Sankhyā Ser. 26*, 359–372.

[24] Zu, Y., C. S. Kochanek, S. Kozłowski, and A. Udalski (2012). Is quasar variability a damped random walk? *ArXiv:astro-ph/1202.3783*.