Chapter 4

Regression Diagnostics and Advanced Regression Topics

We continue our discussion of regression by talking about residuals and outliers, and then look at some more advanced approaches for linear regression, including nonlinear models and sparsity- and robustness-oriented approaches.

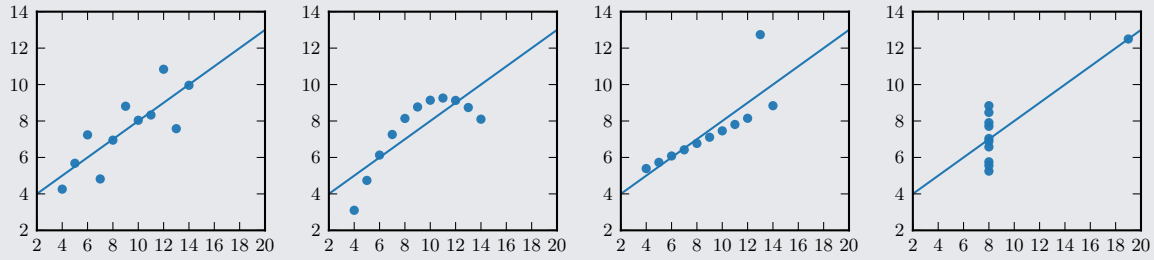
4.1 Residual Analysis

Remember that we defined the **residual** for data point i as $hat\varepsilon_i = y_i - \hat{y}_i$: the difference between the observed value and the predicted value (in contrast, the error ε_i is the difference between the observed value and the true prediction from the correct line). We can often learn a lot about how well our model did by analyzing them. Intuitively, if the model is good, then a plot of the residuals $(y_i - \hat{y}_i)$ against the fitted values $(\hat{y}_i)$ should look like noise (i.e., there shouldn't be any visible patterns).

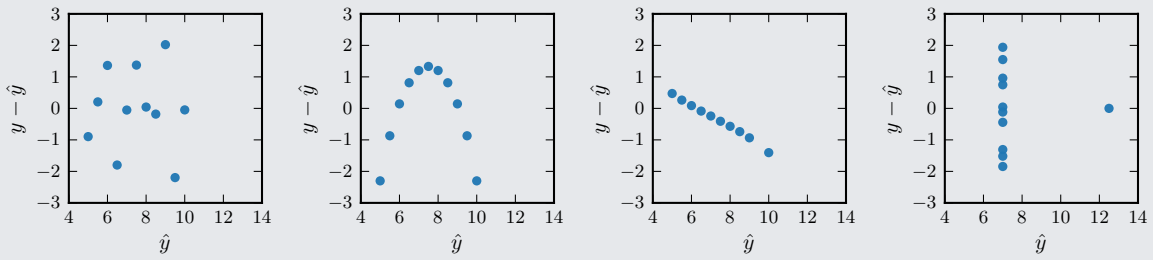
Anscombe's quartet can once again provide us with some intuition.

Example: Anscombe's Quartet Revisited Again

Recall Anscombe's Quartet: 4 datasets with very similar statistical properties under a simple quantitative analysis, but that look very different when inspected visually. Here they are again with linear regression lines fitted to each one:



Below, we plot the residuals $y_i - \hat{y}_i$ vs. $\hat{y}_i$ (note that unlike the previous plot, we're not plotting against x!), which shows how much above and below the fitted line the data points are.



We immediately see that in panels 2, 3, and 4, the residuals don't look anything like random noise as there are specific patterns to them! This suggests that a linear fit is not appropriate for datasets 2, 3, and 4.

While the raw residuals should look like noise, in general their distribution isn't as simple as the i.i.d. Gaussian distribution that we assume for the error. Recalling the probabilistic model from the previous chapter, the data are assumed to be generated by $y_i = X_i\beta + \varepsilon_i$, where each ε_i is a zero-mean Gaussian with variance σ^2. But $\hat{y}_i$ doesn't come directly from this process! Instead, it comes from putting y through the linear regression equations we saw last time. Recall our regression equation solution:

$$\hat{\beta} = (X^T X)^{-1} X^T y$$

Disclaimer: The rest of this section relies heavily on basic linear algebra. If you're uncomfortable or unfamiliar with linear algebra, feel free to skip ahead to the summary at the end of this section.

So the residuals $\hat{\varepsilon}$ (length n vector) are:

$$\begin{aligned}
\hat{\varepsilon} &= y - \hat{y} \\
&= y - X\hat{\beta} \\
&= y - (\underbrace{X(X^T X)^{-1} X^T}_{=H})y \\
&= (I - H)y \\
&= (I - H)(X\beta + \varepsilon) \\
&= (I - H)X\beta + (I - H)\varepsilon \\
&= (I - X(X^T X)^{-1} X^T)X\beta + (I - H)\varepsilon \\
&= \underbrace{(X - X(X^T X)^{-1}(X^T X))}_{=0}\beta + (I - H)\varepsilon \\
&= (I - H)\varepsilon.
\end{aligned} \tag{4.1}$$

This shows how the actual noise ε (that we don't get to observe) relates to the residuals $\hat{\varepsilon} = y_i - \hat{y}$. In fact, in general the residuals are correlated with each other! Furthermore, the variance of the i-th residual $\hat{\varepsilon}_i$ may not be the same as that of ε_i. We can standardize the residuals so that each one has the same variance σ^2.

Using the matrix $H = X(X^T X)^{-1} X^T$, which is called the **hat matrix**, we can define the **standardized residuals** as $\hat{\varepsilon}_i / \sqrt{1 - H_{ii}}$, where H_{ii} is the i-th diagonal entry of H.

Thus, our model tells us that the residuals may not have the same distribution and may be correlated, but the standardized residuals have the same, albeit unknown, variance σ^2.

In order to analyze residuals even further, many packages will go one step further and compute **Studentized residuals**. These are computed by estimating the variance of the noise σ^2 and then dividing by it. Why is this process called "Studentizing"? The estimated noise variance has a (scaled) χ^2 distribution with $n - m - 1$ degrees of freedom, and the standardized residual $\hat{\varepsilon}_i/\sqrt{1 - H_{ii}}$ has a normal distribution, so the result has a Student t distribution.

Most packaged software will standardize residuals for you, but if you're writing your own analysis code, don't forget to standardize before analyzing the residuals!

While the residuals may be correlated with each other, an important theoretical result states that under the probabilistic model we've been using for regression, the residuals $\hat{\varepsilon}$ are uncorrelated with the fitted values $\hat{y}$. This means that when we plot the residuals against the fitted values (as we did in the previous example for Anscombe's Quartet), the resulting plot should look like random noise if the fitted linear regression model is any good.

Residual analysis summary

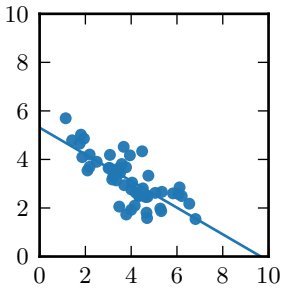
Here are the most important points from our analysis of residuals:

- The residuals themselves, $\hat{\varepsilon}_i$, might have different variances, and furthermore, might be correlated with each other. We can address the first problem by **standardizing** them so that they have the same variance as the residuals, or **Studentizing** them so that they have variance 1. It's important to make sure you visualize standardized or Studentized residuals!
- The residuals $\hat{\varepsilon}$ are uncorrelated with the fitted values $\hat{y}$. This means that if the model we use is in fact a good fit, we shouldn't see any patterns in the standardized residuals.

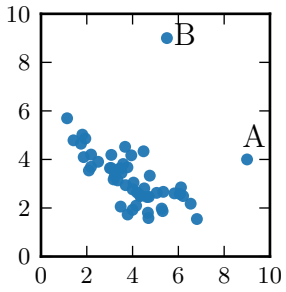
■ 4.2 Outliers

Real life datasets often have some points that are far away from the rest of the data. Here are some informal definitions used to characterize such anomalies:

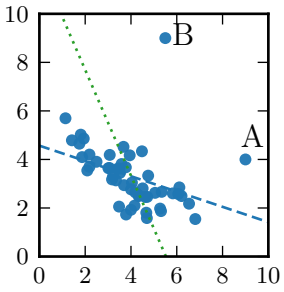
- An **outlier** is any point that's far away from the rest of the data. There are different definitions and ways of quantifying them depending on how you define "far away".
- The **leverage** of a data point is a quantitative description of how far it is from the rest of the points in the x-direction. We'll see a more formal description later, but intuitively, points that are farther away from the average in the x-direction have higher leverage, and points closer to the average have lower leverage.
- An **influential point** is an outlier with high leverage that significantly affects the slope of a regression line. We'll quantify this a little later.



(a) Some points and a regression line fit to those points.



(b) When viewing y as a function of x, points A and B are both outliers since they're far away from the data, but A is also an influential point, since moving it just a little will have a large effect on the slope of the regression line.



(c) The difference between fitting $y = ax + b$ (blue, dashed) and $x = cy + d$ (green, dotted). While the results would be similar if not for the outliers, A has a big impact on the first fit, while B has a big impact on the second.

Figure 4.1: An illustration of outliers and influential points.

Figure 4.1 illustrates these concepts. Since influential points can really mess up linear regression results, it's important to deal with them. Sometimes, such points indicate the need for more data-gathering: if most of our data is concentrated in one area and our sampling methodology was flawed, we might have missed data points in the region between the main cluster and an outlier. Alternatively, if such points don't indicate a problem in data collection but really just are irrelevant outliers, we can remove such points from the analysis; we'll see a few techniques for identifying them later today.

The leverage of any particular point is formally defined as the corresponding diagonal element of the hat matrix, H_{ii} (as defined in Equation (4.1)). We can see this intuition more clearly in the one-dimensional case, where

$$H_{ii} = \frac{1}{n} + \frac{(x_i - \bar{x})^2}{\sum_{j=1}^{n}(x_j - \bar{x})^2}$$

Here, we can see that relatively large values of x_i's have larger leverage as expected. Leverage only measures how far away from the average x-value a point is: it's a measure of how influential a point has the potential to be, and doesn't depend on the point's y-value.

How might we quantify the influence of a particular point? One way is to take each point and fit the model with and without that point. We can then compare the two results: the more different they are, the more influential that point would be. **Cook's distance** is a measure of how influential each point i is that captures exactly this idea. The definition is based on fitting the full model (i.e., using all of our data), and then fitting the model again,

but this time excluding point i:

$$D_i = \frac{\sum_{j=1}^{n}(\overbrace{\hat{y}_j}^{\text{full prediction}} - \overbrace{\hat{y}_{j(-i)}}^{\text{prediction w/o } i})^2}{p \cdot MSE},$$

where

- $\hat{y}_j$ is the predicted value at x_j based on the full model,
- $\hat{y}_{j(-i)}$ is the predicted value at x_j based on the model fit without point i,
- p is the number of features (and the number of coefficients),
- and MSE is the mean squared error, $\frac{1}{n}\sum_{j=1}^{n}(\hat{y}_j - y_j)^2$.

Unfortunately, it seems from this formula that we have to recompute $\hat{\beta}$ without point i any time we want to compute D_i. But, with a bit of algebra, we can derive the following formula, which gives us an easy way to compute Cook's distance in terms of the leverage:

$$D_i = \frac{1}{p \cdot MSE}\frac{H_{ii}}{(1 - H_{ii})^2}\hat{\varepsilon}_i^2$$

We could thus use Cook's distance to compute how influential a point is and screen for outliers by removing points that are too influential based on a threshold of our choice.

Manually removing outliers can feel rather cumbersome, raising the question of whether there are regression methods that automatically handle outliers more gracefully. This leads us to the topic of **robust regression**.

■ 4.3 Advanced Regression: Robustness

One issue with standard linear regression is that it can be affected by outliers. As we saw with influential points, one inconveniently-placed outlier can dramatically alter the outcome of a regression. In this section, we'll first look at two complementary views of how we can achieve more robust outcomes by tweaking the model we've learned about, and then see a completely different way of achieving robustness using random sampling.

■ 4.3.1 Optimization View of Robustness

Suppose instead that we tried to adjust the optimization problem we're solving. Here's our optimization problem:

$$\min_{\beta} \sum_{i=1}^{n}(y_i - X_i\beta)^2 = \sum_{i=1}^{n}\rho(r_i),$$

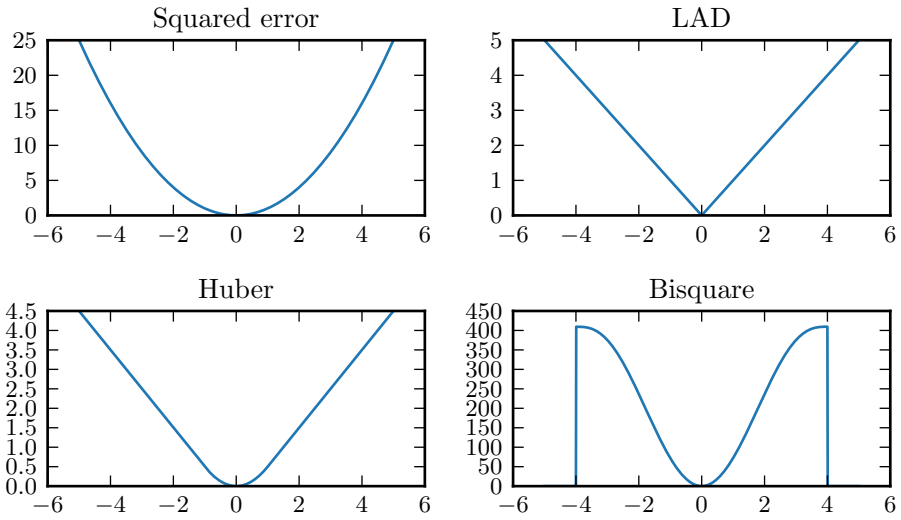


Figure 4.2: Graphs showing the squared-error, LAD, Huber, and bisquare loss functions. Note the different y-axis scales!

where $r_i = (y_i - X_i\beta)$ is the residual and $\rho(r) = r^2$ is the squared error function. Unfortunately, squared error puts very large penalties on large errors: $\rho(2) = 4$, but $\rho(10) = 100$. So, any good solution to this optimization problem will avoid large errors, even if that means fitting to outliers. To get around this, we'll try choosing a different function ρ. The function $\rho(r)$ we choose here is usually called the **loss function**, and any solution to a problem of this form is called an M-estimator. What other loss functions can we try?

- **Least absolute deviations**, or LAD: $\rho(r) = |r|$. While this avoids the problem of excessively penalizing outliers, it leads to a loss function that isn't differentiable at $r = 0$. It also is less *stable* than least-squares: changing the x-value of a point just a little can have a large impact on the solution.
- **Huber**: $\rho(r) = \begin{cases} r^2/2 & |r| < k \\ k(|x| - k/2) & |r| \geq k \end{cases}$.
  This is similar to LAD, but replaces the sharp corner around 0 with a smoothed parabola for differentiability (being differentiable makes it easy to take derivatives when optimizing for the best β).
- **Bisquare**: these functions have a behavior similar to squared loss near 0, but level off after a certain point. Some even assign a cost of 0 to very large deviations, to avoid being sensitive to outliers at all.

These loss functions are shown in Figure 4.2.

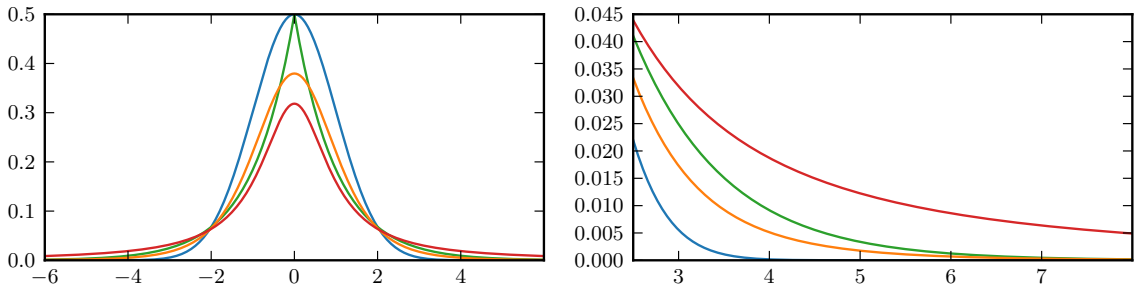


Figure 4.3: Three different distributions and their tails: Gaussian with variance 1 (blue), Laplacian (green), Cauchy (red), and Student's t with 5 degrees of freedom (orange).

■ 4.3.2 Distribution View of Robustness

Remember from Chapter 1 that the normal distribution is very concentrated. So, by assuming Gaussian noise in our model, we're effectively assuming that large errors are extremely unlikely. One way to be less sensitive to outliers is to assume distributions with *heavier tails*: that is, distributions that don't assign such low probability to improbable events. For example, the Student's t distribution, which we've already seen, the Laplacian distribution, the Cauchy distribution, and any power-law distribution all have heavier tails than the Gaussian. These are illustrated in Figure 4.3. Assuming the noise ε is Gaussian leads to squared loss. Assuming the noise to be Laplacian leads to LAD.

■ 4.3.3 RANSAC

Another approach to robust regression is RANSAC, or RANdom SAmple Consensus. While RANSAC doesn't have the theoretical justifications of the methods in the last two sections, it nonetheless is widely used and often works well in practice. It's so popular that there's even a song about RANSAC[1]!

The basic assumption of RANSAC is just that the data consists primarily of non-outliers, which we'll call "inliers". The algorithm works as follows: we randomly pick a subset of our points to call inliers and compute a model based on those points. Any other points that fit this computed model well are added to the inliers. We then compute the error for this model, and repeat the entire process several times. Whichever model achieves the smallest error is the one we pick. Here's a description in pseudocode:

Inputs: Points $(x_1, y_1), \ldots, (x_n, y_n)$

1: **function** RANSAC
2: **for** iteration $t \in 1, 2, \ldots, T$ **do**
3: Choose a random subset of points to be inliers; call them I_t

[1] See http://www.youtube.com/watch?v=1YNjMxxXO-E

4: Fit a model M_t using the selected points
5: Find all points that have error less than α and add them to I_t
6: *(Optional)* Re-fit the model M_t with the new I_t
7: Find error for model M_t on points in I_t, call it ε_t
8: Choose model with the smallest error

Notice how imprecise our specification is; the following are all parameters that must be decided upon to use RANSAC:

- The number of iterations, T: this one can be chosen based on a theoretical result.
- How large of a subset to choose
- How to fit the model and evaluate the error
- The error threshold for adding inliers, α
- Whether or not to do the optional refitting

Many of these depend on the specific problem being solved.

4.4 Advanced Regression: Sparsity

Another useful criterion that we often want our model to satisfy is *sparsity*. This means that we want many of the coefficients β_k to be 0. This constraint is useful in many cases, such as when there are more features than data points, or when some features are very similar to others. Adding in a sparsity constraint in these settings often helps prevent overfitting, and leads to simpler, more interpretable models.

4.4.1 A first attempt: ridge regression

Since we want the coefficients to be close to zero (i.e., small), let's try adding a term to our minimization that "encourages" them to be small. In **ridge regression**, we solve the following optimization problem:

$$\min_{\beta} \left[\underbrace{\sum_{i=1}^{n} (y_i - X_i\beta)^2}_{\text{"data term"}} + \underbrace{\lambda \sum_{k=1}^{p} \beta_k^2}_{\text{"regularization term"}} \right],$$

where λ is a nonnegative parameter that trades off between how small we want the coefficients β_k to be and how small we want the error to be: if λ is 0, then the problem is the same as before, but if λ is very large, the second term counts a lot more than the first, and so we'll try to make the coefficients β_k as small as possible. λ is often called a **regularization**

parameter, and we'll talk a little more next week about how to choose it. With some linear algebra, we can get the following solution:

$$\hat{\beta} = (X^T X - \lambda I)^{-1} X^T y.$$

We see that our intuition from before carries over: very large values of λ will give us a solution of $\hat{\beta} = 0$, and if $\lambda = 0$, then our solution is the same as before.

Unfortunately, while ridge regression does give us smaller coefficients (and helps make the problem solvable when $X^T X$ isn't invertible), it doesn't provide us with sparsity. Here's an example illustrating why: Suppose we have only 3 input components, and they're all the same. Then the solutions $\beta_a = (1, 1, 1)$ and $\beta_b = (0, 0, 3)$ will both give us the same y, but the second one is sparse while the first one isn't. But, the second one actually produces a higher ridge penalty than the first! So, rather than produce a sparse solution, ridge regression will actually favor solutions that are not sparse.

While using three identical inputs is a contrived example, in practice it's common to have several input dimensions that could have similar effects on the output, and choosing a small (sparse) subset is critical to avoid overfitting.

For the purposes of preventing overfitting and where we don't care about sparsity, ridge regression is actually widely used in practice. However, if we seek a sparse solution, we must turn to a different approach.

Ridge regression: a Bayesian view

Up until now, we've been treating the parameter β as a fixed but unknown quantity. But what if we treat it as a random quantity? Suppose before we know y or X, we have a prior belief about β, which we'll encode in the form of a **prior distribution** $p(\beta)$. Once we observe X and y, we can compute a more informed distribution, called a **posterior distribution**, from that information. We'll write it as $p(\beta|X, y)$, where the "|" means "given" or "having observed". Using Bayes' rule, we can compute the posterior distribution as:

$$p(\beta|X, y) \propto p(\beta)p(X, y|\beta) \tag{4.2}$$

Notice that we've already specified $p(X, y|\beta)$: it's just the model for the errors ε. Once we choose $p(\beta)$, we can try to find the value of β that's most likely according to $p(\beta|X, y)$. If we choose $p(\beta)$ to be a zero-mean Gaussian with variance $\sigma^2 = \frac{1}{2\lambda^2}$, then we'll end up with ridge regression.

■ 4.4.2 Sparse solutions: LASSO

We've seen that introducing a penalty of the form $\sum_{k=1}^{p} \beta_k^2$ doesn't give us sparsity. So why not penalize non-sparsity directly? The ideal alternative might look something like:

$$\min_{\beta} \left[\sum_{i=1}^{n} (y_i - X_i\beta)^2 + \lambda \underbrace{\sum_{k=1}^{p} \mathbb{1}(\beta_k^2 \neq 0)}_{\#\text{ of nonzeros in } \beta} \right], \tag{4.3}$$

where the penalty imposes a cost for any nonzero value of β. Alas, there is no known *efficient* method for computing a solution to this problem. For any moderate to large dataset, the above optimization problem is intractable to solve in any reasonable amount of time. We basically can't do much better than searching over every possible combination of which β_ks are 0 and which aren't.

Instead, we'll opt to solve the following problem:

$$\min_{\beta} \left[\sum_{i=1}^{n} (y_i - X_i\beta)^2 + \lambda \sum_{k=1}^{p} |\beta_k| \right].$$

This problem, also known as ℓ_1-norm minimization or the Least Absolute Shrinkage and Selection Operator (LASSO), is an approximation to (4.3) that can be solved efficiently. Additionally, under certain reasonable conditions, if the true model is in fact sparse, then the solution to this problem will also be sparse. There are numerous other interesting theoretical results about LASSO, and it's an active area of open research! As a result of its popularity, most software packages have an option for LASSO when performing linear regression.

The Bayesian interpretation corresponds to using a *Laplacian prior* over β, so that our prior belief $p(\beta)$ is $e^{-\lambda|\beta|}$ for each coefficient. If we want to perform hypothesis tests or get confidence intervals for coefficients that come out of this method, we'll need to use nonparametric statistics, which we'll talk about in Chapter 5.

■ 4.4.3 Sparsity and shrinkage: a graphical view

Figure 4.4 shows the two Bayesian priors that correspond to ridge regression and LASSO (along with one other prior). Notice that in ridge regression, a value close to zero is almost as likely as a value equal to zero. However, LASSO is a bit better: the gap between zero and near-zero in terms of probability is bigger. We can take this one step further and define other so-called *shrinkage priors*, which even more strongly encourage values to be zero. One such prior, called the horseshoe prior[2] is shown in Figure 4.4. Unfortunately, many of these result in more difficult optimization problems; one reason for LASSO's popularity is that it produces a convex optimization problem where we can always find the global optimum (i.e., the best solution)[3]. Notice that the horseshoe prior also has heavier tails than the other two distributions: this means that in addition to encouraging values near zero, it allows large nonzero values to be more likely.

At this point, we've talked about two different kinds of distributions: in Section **??**, we looked at different distributions for the error ε, which led to more robust methods that were less sensitive to outliers. In this section, we looked at different prior distributions for the coefficients β, which led to sparser solutions.

[2]See Carvalho, Polson, Scott. *The Horseshoe Estimator for Sparse Signals*, Biometrika. Since there isn't actually a closed-form expression for the distribution, the version shown in the figure is an approximation.

[3]While this is true in most cases, it isn't strictly true all the time: if the matrix X doesn't have full rank, then the LASSO problem doesn't have a unique solution.

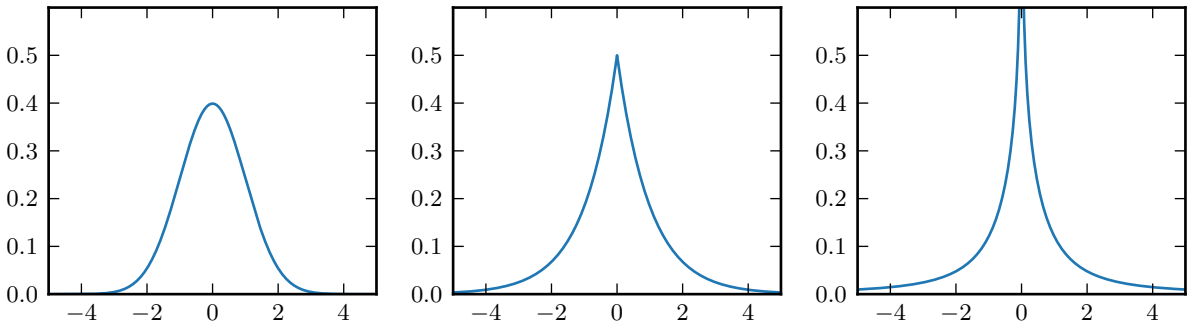


Figure 4.4: Several coefficient priors for sparse regression: Gaussian (left) as in ridge regression, Laplacian (center) as in LASSO, and a horseshoe prior (right). Note that the horseshoe prior approaches ∞ as the coefficient becomes smaller.

■ 4.5 Generalized Linear Models

So far, we've dealt entirely with linear models and linear relationships. Unfortunately, in the real world, we often come across data with nonlinear properties. While we can often get by with cleverly defined features, we often have to deal with output data that has a nonlinear dependence that we can't capture. For example, if our output data is binary, or even between 0 and 1, the techniques we've talked about so far have no way of constraining the predictions to that range.

Recall that before, we assumed that $y = \mu_y + \varepsilon$, where $\mu_y = X\beta$.

Generalized linear models[4] are a family of methods that assume the following:

$$\mu_y = g^{-1}(X\beta),$$

where $g(\cdot)$[5] is called the **link function** and is usually nonlinear. Here, the interaction between the input X and the parameters β remains linear, but the result of that linear interaction is passed through the inverse link function to obtain the output y. We often write $\eta = X\beta$, so that $\mu_y = g^{-1}(\eta)$. The choice of link function typically depends on the problem being solved. Below, we provide an example link function commonly used when y is binary.

Example: Logistic Regression

Suppose that we have data between 0 and 1. Then we can use a sigmoidal link function, which has the form

$$g^{-1}(\eta) = \frac{1}{1 + \exp(-\eta)}.$$

A graph of this function is shown in Figure 4.5. This function maps a real number to a number between 0 and 1. Logistic regression is often used in classification problems, where our output is binary.

[4] These are not to be confused with *general linear models*, which we'll talk more about next week.
[5] Using the inverse of $g(\cdot)$ in the setup is a matter of historical tradition.

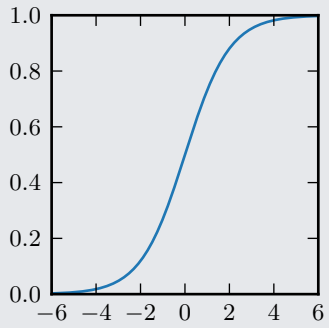


Figure 4.5: A sigmoid function maps a real number to a value between 0 and 1.

Generalized linear models also make fewer assumptions about the form of the error: while before, we assumed that $y = \mu_y + \varepsilon$, generalized linear models give us more freedom in expressing the relationship between y and μ_y by fully specifying the distribution for y.

While generalized linear regression problems usually can't be solved in closed form (in other words, we can't get a simple expression that tells us what β is), there are efficient computational methods to solve such problems, and many software packages have common cases such as logistic regression implemented.