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.

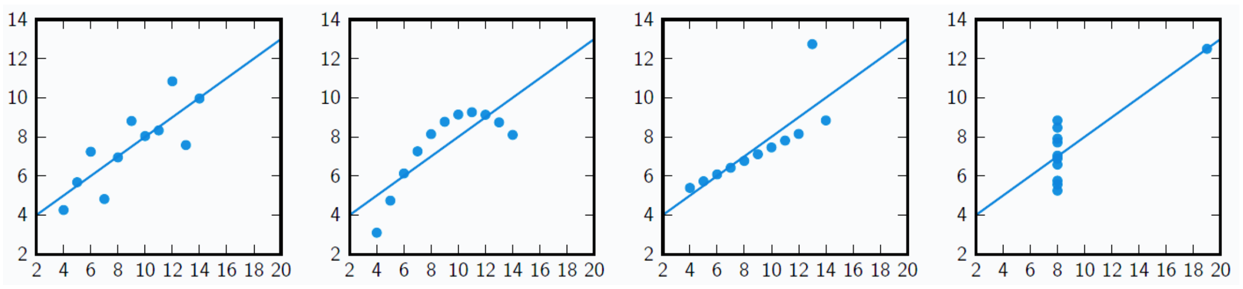
4.1 Residual Analysis

Remember that we defined the residual for data point i as $\hat{\epsilon}_i = y_i - \hat{y}_i$: 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).

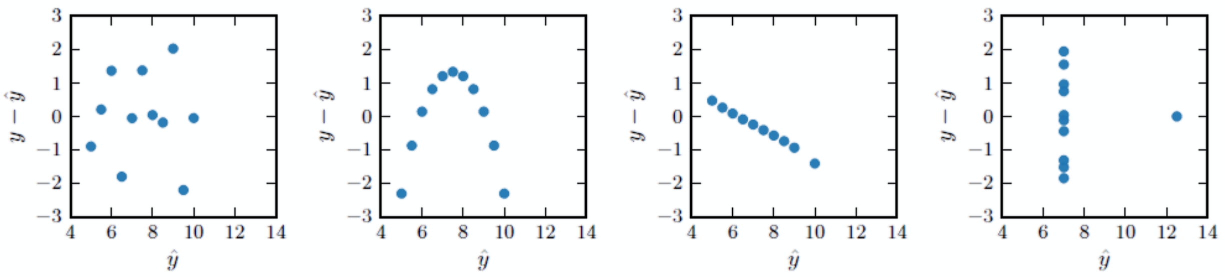
Anscombe's quartet can once again provide us with some intuition.

Anscombe's Quartet Revisited

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



Below, we plot the residuals $y_i - \hat{y}_i$ vs. $\hat{y}$ (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 $y_i = X_i\beta + \epsilon_i$, 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:

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

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

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

This shows how the actual noise ϵ(that we don't get to observe) relates to the residuals $\hat{\epsilon} = y_i - \hat{y}$. In fact, in general the residuals are correlated with each other! Furthermore, the variance of the i-th residual $\hat{\epsilon}_i$ 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 $H = X(X^TX)^{-1}X^T$, which is called the hat matrix (or projection matrix), we can define the standardized residuals as $\hat{\epsilon}_i/\sqrt{1 - H_{ii}}$, 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. (Obviously, there are known variance cases, assuming that we have data for entire population.)

In order to analyze residuals even further, many packages will go one step further and compute ~~standardized~~ "Studentizing" 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) $\chi 2$ distribution with $n-m-1$ degrees of freedom, and the standardized residual $\hat{\epsilon}_i/\sqrt{1 - H_{ii}}$ has a normal distribution, so the result has a Student t distribution. Generally this process is called standardizing with estimated variance, and the term "Studentizing" is used only for a few fraction of statisticians.

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 $\hat{\epsilon}$ 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

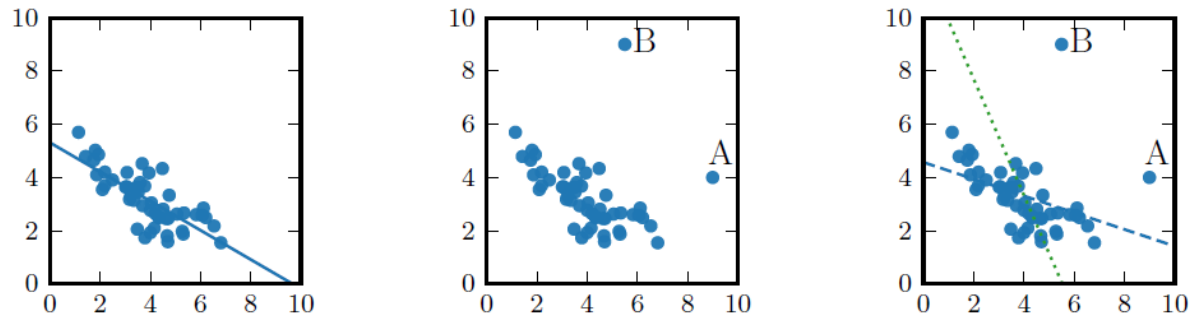
Here are the most important points from our analysis of residuals:

- The residuals themselves, $\hat{\epsilon}_i$, 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 $\hat{\epsilon}$ 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 (outlier) 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.

The leverage of any particular point is formally defined as the corresponding diagonal element of the hat matrix (or the projection 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:

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

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, $\frac{1}{n}\sum_{j=1}^{n}(\hat{y}_f - y_j)^2$.

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

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:

$$\underset{\beta}{min}\sum_{i=1}^{n}(y_i - X\beta)^2 = \sum_{i=1}^{n}\rho(r_i),$$

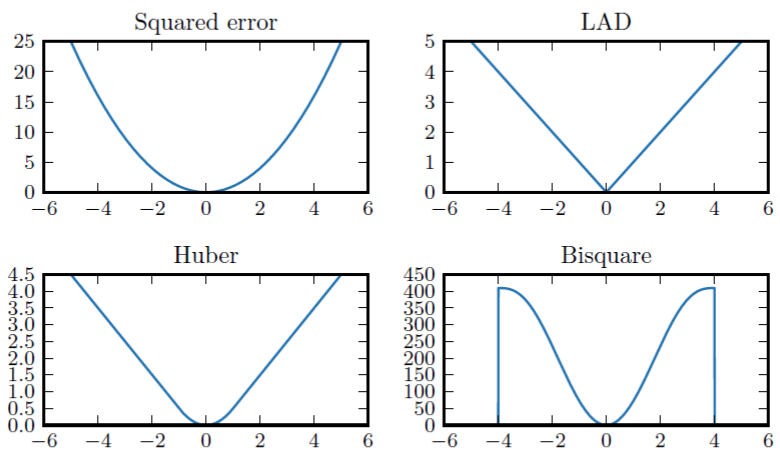


Figure 4.2: Graphs showing the squared-error, LAD, Huber, and bi-square loss functions. Note the different y-axis scales!

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.

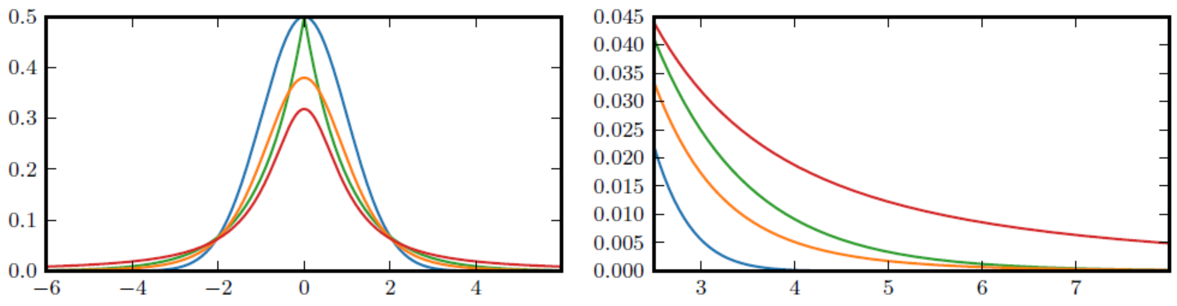


Figure 4.3: Three different distributions and their tails: Gaussian with variance 1 (blue), Laplacian (green), Cauchy (red), and Student's t with 5 degrees of freedom (orange).

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.

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.

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 $(x_1, y_1), ..., (x_n, y_n)$

```
1: function RANSAC
2:     for iteration $t \in 1, 2, ..., T$ do
3:         Choose a random subset of points to be inliers; call them $I_t$
4:         Fit a model $M_t$ using the selected points
5:         Find all points that have error less than $\alpha$ 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 $\epsilon_t$
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:

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

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 later talk a little more about how to choose it. With some linear algebra, we can get the following solution:

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

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:

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

where the penalty imposes a cost for any non-zero 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 $\beta_k s$ are 0 and which aren't.

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

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

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 prior2 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). 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: earlier, 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.

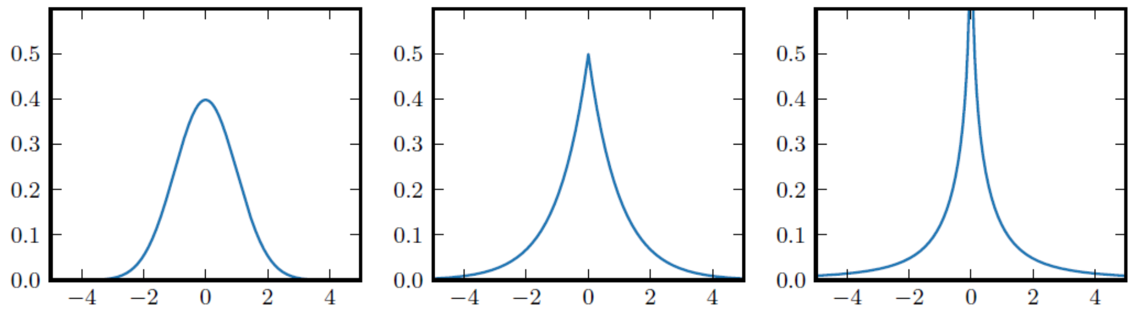


Figure 4.4: Several coefficient priors for sparse regression: Gaussian (left) as in ridge regression, Laplacian (center) as in LASSO, and a horseshoe prior (right). Note that the horseshoe prior approaches 1 as the coefficient becomes smaller.

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 $y = \mu_y + \epsilon$, where $\mu_y = X\beta$.

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

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

where $g(\cdot)$ 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.

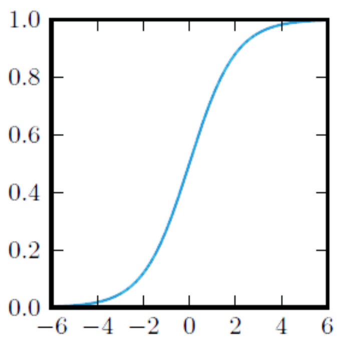


Figure 4.5: A sigmoid function maps a real number to a value between 0 and 1.

Generalized linear models also make fewer assumptions about the form of the error: while before, we assumed that $y = \mu_y + \epsilon$, 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.

5 Nonparametric statistics and model selection

In Chapter 2, we learned about the t-test and its variations. These were designed to compare sample means, and relied heavily on assumptions of normality. We were able to apply them to non-Gaussian populations by using the central limit theorem, but that only really works for the mean (since the central limit theorem holds for averages of samples). Sometimes, we're interested in computing other sample statistics and evaluating their distributions (remember that all statistics computed from samples are random variables, since they're functions of the random samples) so that we can obtain confidence intervals for them. In other situations, we may not be able to use the central limit theorem due to small sample sizes and/or unusual distributions.

In this chapter, we'll focus on techniques that don't require these assumptions. Such methods are usually called nonparametric or distribution-free. We'll first look at some statistical tests, then move to methods outside the testing framework.

5.1 Estimating distributions and distribution-free tests

So far, we've only used "eyeballing" and visual inspection to see if distributions are similar. In this section, we'll look at more quantitative approaches to this problem. Despite this, don't forget that visual inspection is usually an excellent place to start!

We've seen that it's important to pay attention to the assumptions inherent in any test. The methods in this section make fewer assumptions and will help us test whether our assumptions are accurate.

5.1.1 Kolmogorov-Smirnov test

The Kolmogorov-Smirnov test tests whether two arbitrary distributions are the same. It can be used to compare two empirical data distributions, or to compare one empirical data distribution to any reference distribution. It's based on comparing two cumulative distribution functions (CDFs).

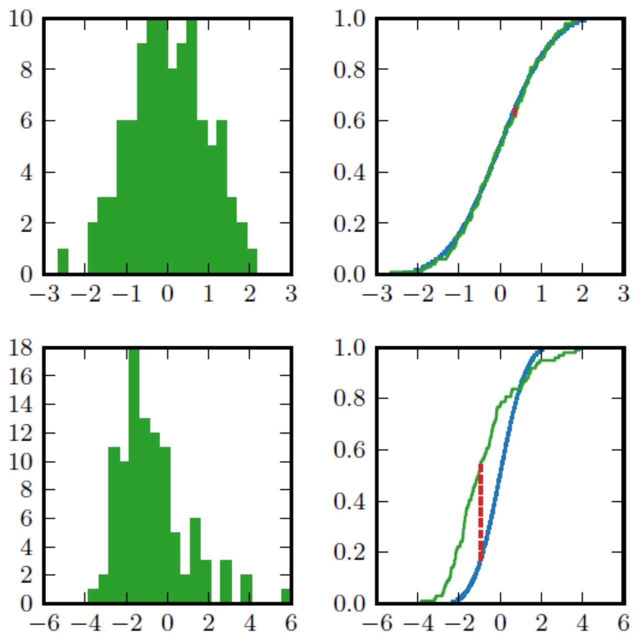


Figure 5.1: Two Kolmogorov-Smirnov test plots (right column) with histograms of the data being tested (left column). On the top row, the empirical CDF (green) matches the test CDF (blue) closely, and the largest difference (dotted vertical red line, near 0.5) is very small. On the bottom, the empirical CDF is quite different from the test CDF, and the largest difference is much larger.

Remember that the CDF of a random variable x is the probability that the random variable is less than or equal to some value. To be a bit more precise, it's a function F such that $F(a) = P(x \leq a)$. When talking about data, it's often useful to look at empirical CDFs: $F_n(a) = \frac{1}{n}\sum_i \mathbb{I}(x_i < a)$ is the CDF of n observed data points.

Now suppose we want to compare two CDFs, F1 and F2. They might be empirical CDFs (to compare two different datasets and see whether they're significantly different) or one might be a reference CDF (to see whether a particular distribution is an appropriate choice for a dataset). The Kolmogorov-Smirnov test computes the statistic D_n:

$$D_n = \underset{x}{max}|F_n^1(x) - F_n^2(x)|$$

This compares the two CDFs and looks at the point of maximum discrepancy; see Figure 5.1 for an example. We can theoretically show that if F^1 is the empirical distribution of x and F^2 is the true distribution x was drawn from, then $lim_{n\to\infty} D_n = 0$. Similarly, if the two distributions have no overlap at all, the maximum difference will be 1 (when one CDF is 1 and the other is 0). Therefore, we can test distribution equality by comparing the statistic D_n to 0 (if D_n is significantly larger than 0 and close to 1, then we might conclude that the distributions are not equal).

Notice that this method is only defined for one-dimensional random variables: although there are extensions to multiple random variables; they are more complex than simply comparing joint CDFs.

Also notice that this test is sensitive to any differences at all in two distributions: two distributions with the same mean but significantly different shapes will produce a large value of D_n.

5.1.2 Shapiro-Wilk test

The Shapiro-Wilk test tests whether a distribution is Gaussian using quantiles. It's a bit more specific than the Kolmogorov-Smirnov test, and as a result tends to be more powerful. We won't go into much detail on it in this class, but if you're interested, the Wikipedia page has more detail.

5.1.3 Wilcoxon's signed-rank test

The two-sample t test we discussed in Chapter 2 requires us to use the central limit theorem to approximate the distribution of the sample mean as Gaussian. When we can't make this assumption (i.e., when the number of samples is small and/or the distributions are very skewed or have very high variance), we can use Wilcoxon's signed-rank test for matched pairs.
This is a paired test that compares the medians of two distributions. For example, when comparing the incomes of two different groups (especially groups that span the socioeconomic spectrum), the distributions will likely be highly variable and highly skewed. In such a case, it might be better to use a nonparametric test like Wilcoxon's signed-rank test. In contrast to the Kolmogorov-Smirnov test earlier, this test (like its unpaired cousin the Mann-Whitney U) is only sensitive to changes in the median, and not to changes in the shape.

The null hypothesis in this test states that the median difference between the pairs is zero. We'll compute a test statistic and a corresponding p-value, which give us a sense for how likely our data are under the null hypothesis. We compute this test statistic as follows:

(1) For each pair i, compute the difference, and keep its absolute value d_i and its sign S_i (where $S_i \in \{1, 0, +1\}$). We'll exclude pairs with $S_i = 0$.

(2) Sort the absolute values from smallest to largest, and rank them accordingly. Let R_i be the rank of pair i (for example, if the fifth pair had the third smallest absolute difference, then $R_5 = 3$).

(3) Compute the test statistic:

$$W = |\sum_i S_i R_i|$$

W has a known distribution. In fact, if N is greater than about 10, it's approximately normally distributed (if not, it still has a known form). So, we can evaluate the probability of observing it under a null hypothesis and thereby obtain a significance level.

Intuitively, if the median difference is 0, then half the signs should be positive and half should be negative, and the signs shouldn't be related to the ranks. If the median difference is non-zero, W will be large (the sum will produce a large negative value or a large positive value). Notice that once we constructed the rankings and defined R_i, we never used the actual differences!

5.1.4 Mann-Whitney U Test

The Mann-Whitney U test is similar to the Wilcoxon test, but can be used to compare multiple samples that aren't necessarily paired. The null hypothesis for this test is that the two groups have the same distribution, while the alternative hypothesis is that one group has larger (or smaller) values than the other. Computing the test statistic is simple:

(1) Combine all data points and rank them (largest to smallest or smallest to largest).

(2) Add up the ranks for data points in the first group; call this R_1. Find the number of points in the group; call it n_1. Compute $U_1 = R_1 - n_1(n_1 + 1)/2$. Compute U_2 similarly for the second group.

(3) The test statistic is defined as $U = min(U_1, U_2)$.

As with W from the Wilcoxon test, U has a known distribution. If n_1 and n_2 are reasonably large, it's approximately normally distributed with mean $n_1n_2/2$ under the null hypothesis. If the two medians are very different, U will be close to 0, and if they're similar, U will be close to $n_1n_2/2$. Intuitively, here's why:

- If the values in the first sample were all bigger than the values in the second sample, then $R1 = n_1(n_1+1)/2$: this is the smallest possible value for R_1. U_1 would then be 0.
- If the ranks between the two groups aren't very different, then U_1 will be close to U_2. With a little algebra, you can show that the sum $U_1 + U_2$ will always be n_1n_2. If they're both about the same, then they'll both be near half this value, or $n_1n_2/2$.

5.2 Resampling-based methods

All the approaches we've described involve computing a test statistic from data and measuring how unlikely our data are based on the distribution of that statistic. If we don't know enough about the distribution of our test statistic, we can use the data to tell us about the distribution: this is exactly what resampling-based methods do. Permutation tests "sample" different relabelings of the data in order to give us a sense for how significant the true relabeling's result is. Bootstrap creates "new" datasets by resampling several times from the data itself, and treats those as separate samples. The next example illustrates a real-world example where these methods are useful.

EXAMPLE: CHICAGO TEACHING SCANDAL

In 2002, economists Steven Levitt and Brian Jacob investigated cheating in Chicago public schools, but not in the way you might think: they decided to investigate cheating by teachers, usually by changing student answers after the students had taken standardized tests.

So, how'd they do it? Using statistics! They went through test scores from thousands of classrooms in Chicago schools, and for each classroom, computed two measures:

(1) How unexpected is that classroom's performance? This was computed by looking at every student's performance the year before and the year after. If many students had an unusually high score one year that wasn't sustained the following year, then cheating was likely.

(2) How suspicious are the answer sheets? This was computed by looking at how similar the A-B-C-D patterns on different students' answer sheets were.

Unfortunately, computing measures like performance and answer sheet similarity is tricky, and results in quantities that don't have well-defined distributions! As a result, it isn't easy to determine a null distribution for these quantities, but we still want to evaluate how unexpected or suspicious they are.

To solve this problem, Levitt and Jacob use two nonparametric methods to determine appropriate null distributions as a way of justifying these measures. In particular:

- They assume (reasonably) that most classrooms have teachers who don't cheat, so by looking at the 50th to 75th percentiles of both measures above, they can obtain a null distribution for the correlation between the two.

- In order to test whether the effects they observed are because of cheating teachers, they randomly re-assign all the students to new, hypothetical classrooms and repeat their analysis. As a type of permutation test, this allows them to establish a baseline level for these measures by which they can evaluate the values they observed.

While neither of these methods is exactly like what we'll discuss here, they're both examples of a key idea in nonparametric statistics: using the data to generate a null hypothesis rather than assuming any kind of distribution.

What'd they find? 3.4% of classrooms had teachers who cheated on at least one standardized test when the two measures above were thresholded at the 95th percentile. They also used regression with a variety of classroom demographics to determine that academically poorer classrooms were more likely to have cheating teachers, and that policies that put more weight on test scores correlated with increased teacher cheating.

5.2.1 Permutation tests

Suppose we want to see if a particular complex statistic is significantly different between two groups. If we don't know the distribution of the statistic, then we can't assign any particular probabilities to particular values, and so we can't say anything interesting after computing a statistic from data. The idea behind permutation tests is to use the data to generate the null hypothesis.

The key idea is that if there weren't a significant difference between this statistic across the two groups in our data, then the statistic should have the same value for any other two arbitrarily selected "groups" of the same sizes. We'll make up a bunch of fake "groups" and see if this is the case.

We'll demonstrate the idea with comparing sample means, but note that in practice sample means could be compared using a t-test, and that permutation tests are generally used for more complicated statistics whose distribution can't be deduced or approximated.

Let's call the two groups in our data A and B. Our test statistic of interest will be $w^* = \bar{x}_A - \bar{x}_B$: the difference in the means of the two groups. Our null hypothesis will be $H_0 : \bar{x}_A = \bar{x}_B$. As before, we want to see whether this difference is significantly large or small. In order to get a sense for the null distribution of w, we can try relabeling the points many different times and recomputing the statistic.

Suppose the original set A has n data points, and B has m points. We'll randomly pick n points out of all n+m, assign those to our new "group" A_i, and assign the others to our new "group" B_i (where the subscript i indicates the iteration).
We'll recompute w_i based on these new groups, and repeat the process. We can then see whether our value w^* is unusual based on the collection of w_i, perhaps by looking at what fraction of them are more extreme than w^*.

We could either repeat this for every possible labeling (usually called an exact test), or randomly choose a subset and evaluate only on those (usually called a Monte Carlo approximation). So, this entire procedure requires only a procedure for computing the statistic of interest (in this case, w): the rest is determined by the data.

We can apply permutations elsewhere, too! For example, if we suspect there is some kind of trend in time series data, we can permute the time indices and compare the correlation. For example, if we observe the time series data (8, 12, 15.8, 20.3) and suspect there's a trend, we can permute the order, obtaining permutations like (15.8, 12, 20.3, 8) or (20.3, 8, 15.8, 12), fit linear models to each one, and evaluate how unlikely the observed fit is.

5.2.2 Bootstrap

Suppose we have some complicated statistic y that we computed from our data x. If we want to provide a confidence interval for this statistic, we need to know its variance. When our statistic was simply $\bar{x}$, we could compute the statistic's standard deviation (i.e., the standard error of that statistic) from our estimated standard deviation using $s_x/\sqrt{n}$. But for more complicated statistics, where we don't know the distributions, how do we provide a confidence interval?

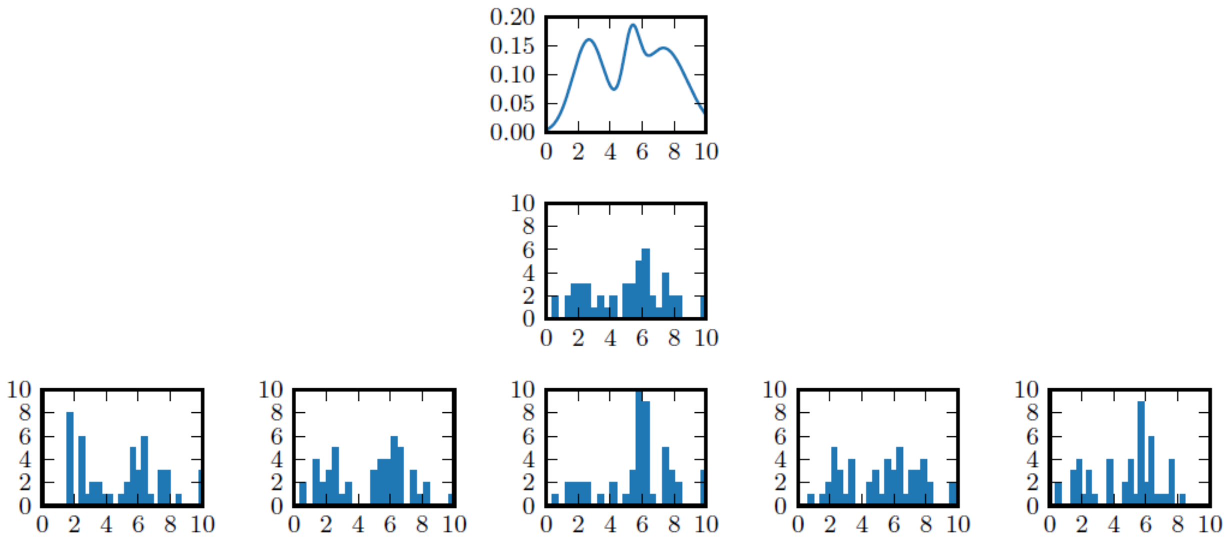


Figure 5.2: An illustration of bootstrap sampling. The top figure shows the true distribution that our data points are drawn from, and the second figure shows a histogram of the particular data points we observed ($N = 50$). The bottom row shows various bootstrap resamplings of our data (with $n = N = 50$). Even though they were obtained from our data, they can be thought of as samples from the true distribution (top).

One approach is a method called bootstrap. The key idea here is that we can resample points from our data, compute a statistic, and repeat several times to look at the variance across different resamplings.

Recall that our original data (N points) are randomly generated from some true distribution. If we randomly sample n points ($n \leq N$, and often $n = N$) from our data with replacement, these points will also be random samples from our true distribution, as shown in Figure 5.2. So, we can compute our statistic over this smaller random sample and repeat many times, measuring the variance of the statistic across the different sample runs.

Everything we've talked about so far has been based on the idea of trying to approximating the true distribution of observed data with samples. Bootstrap takes this a step further and samples from the samples to generate more data.

A similar method, known as jackknife, applies a similar process, but looks at $N - 1$ points taken without replacement each time instead of n with replacement. Put more simply, we remove one point at a time and test the model. Notice that we've seen a similar idea before: our initial definition of Cook's distance was based on the idea of removing one point at a time. In practice, boostrap is more widely used than jackknife; jackknife also has very different theoretical properties.

5.3 Model selection

When fitting models (such as regression) to real data, we'll often have a choice to make for model complexity: a more complex model might fit the data better but be harder to interpret, while a simpler model might be more interpretable but produce a larger error. We've seen this before when looking at polynomial regression models and LASSO in Chapters 3 and 4. In this section, we'll learn how to pick the "best" model for some data among several choices.

5.3.1 Minimum generalization error

First, we'll define "best" by how well our model generalizes to new data that we'll see in the future. In particular, we'll define it this way to avoid overfitting. How can we figure out how well our model generalizes? The most common way is to divide our data into three parts:

- The training set is what we'll use to fit the model for each possible value of the manually-set parameters,
- the validation set is what we'll use to determine parameters that require manual settings,
- and the test set is what we use to evaluate our results for reporting, and get a sense for how well our model will do on new data in the real world.

It's critically important to properly separate the test set from the training and validation sets! At this point you may be wondering: why do we need separate test and validation sets? The answer is that we choose a model based on its validation set performance: if we really want to see generalization error, we need to see how it does on some new data, not data that we used to pick it.

A good analogy is to think of model fitting and parameter determination as a student learning and taking a practice exam respectively, and model evaluation as that student taking an actual exam. Using the test data in any way during the training or validation process is like giving the student an early copy of the exam: it's cheating!

Figure 5.3 illustrates a general trend we usually see in this setup: as we increase model complexity, the training error (i.e. the error of the model on the training set) will go down, while the testing error will hit a "sweet spot" and then start increasing due to overfitting.

For example, if we're using LASSO (linear regression with sum-of-absolute-value penalty) as described in Chapter4, we need to choose our regularization parameter λ. Recall that λ controls model sparsity/complexity: small values of λ lead to complex models, while large values lead to simpler models. One approach is:

(a) Choose several possibilities for λ and, for each one, compute coefficients using the training set.

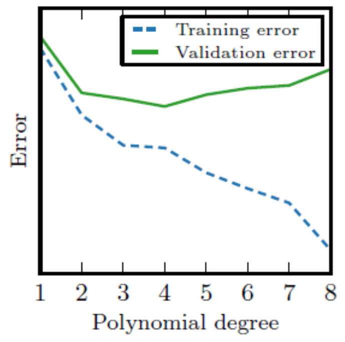


Figure 5.3: Training and validation error from fitting a polynomial to data. The data were generated from a fourth-order polynomial. The validation error is smallest at this level, while the training error continues to decrease as more complex models overfit the training data.

(b) Then, look at how well each one does on the validation set. Separating training and validation helps guard against overfitting: if a model is overfit to the training data, then it probably won't do very well on the validation data.

(c) Once we've determined the best value for λ (i.e., the one that achieves minimum error in step (b)), we can fit the model on all the training and validation data, and then see how well it does on the test data. The test data, which the model/parameters have never seen before, should give a measure of how well the model will do on arbitrary new data that it sees.

The procedure described above completely separates our fitting and evaluation processes, but it does so at the cost of preventing us from using much of the data. Recall from last week that using more data for training typically decreases the variance of our estimates, and helps us get more accurate results. We also need to have enough data for validation, since using too little will leave us vulnerable to overfitting.

One widely used solution that lets us use more data for training is cross-validation. Here's how it works:

(1) First, divide the non-test data into K uniformly sized blocks, often referred to as folds. This gives us K training-validation pairs: in each pair, the training set consists of $K-1$ blocks, and the validation set is the remaining block.

(2) For each training/validation pair, repeat steps (a) and (b) above: this gives us K different errors for each value of λ. We can average these together to get an average error for each λ, which we'll then use to select a model.

(3) Repeat step (c) above to obtain the test error as an evaluation of the model.

Although these examples were described with respect to LASSO and the parameter λ, the procedures are much more general: we could have easily replaced "value for λ" above with a different measure of model complexity. Also note that we could use a bootstrap-like approach here too: instead of deterministically dividing our dataset into K parts, we could have randomly subsampled the non-test data K different times and applied the same procedure.

5.3.2 Penalizing complexity

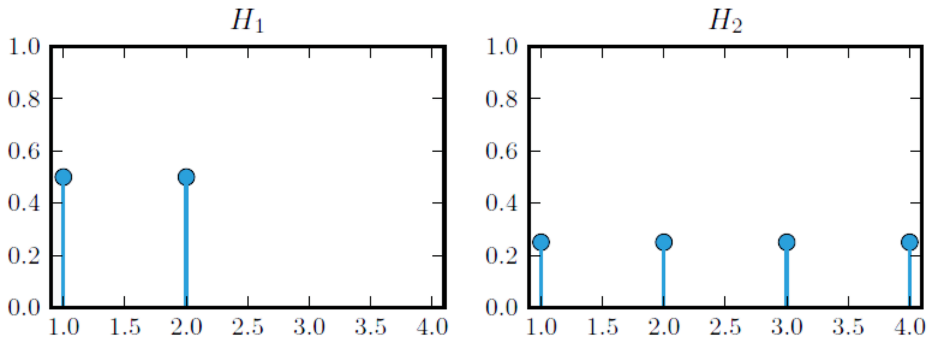
Another way to address the tradeoff between model complexity and fit error is to directly penalize model complexity. That is, we can express the "badness" of a model as the sum of an error term (recall the sum of squared error cost that we used in linear regression) and some penalty term that increases as the model becomes more complex. Here, the penalty term effectively serves as a proxy for minimizing generalization error. Two common penalty terms are AIC, or Akaike Information Criterion, which adds a penalty term equal to $2p$(where p is the number of parameters), and BIC, or Bayesian Information Criterion, which adds a penalty term equal to p ln n (where n is the number of data points).

BIC is closely related to another criterion known as MDL, or minimum description length. This criterion tries to compress the model and the error: if the model and the error are both small and simple, they'll be highly compressible. A complex model that produces low error will be hard to compress, and an overly simplistic model will produce high error that will be hard to compress. This criterion tries to achieve a balance between the two.

EXAMPLE: BAYESIAN OCCAM'S RAZOR

Another idea commonly used is Bayesian Occam's Razor. In Bayesian models, simpler models tend to have higher probability of observing data.

Suppose we observe a number and want to decide which of two models generate it. The simpler model, H_1, says that it's equally likely to be 1 or 2, while the more complex model, H_2, says that it's equally likely to be 1, 2, 3, or 4. The following figure shows the two distributions:



If we observe the number 2, the likelihood of this observation under H_1 is 0.5, while the likelihood under H_2 is 0.25. Therefore, by choosing the model with the highest probability placed on our observation, we're also choosing the simpler model.

Intuitively, the more values that a model allows for, the more it has to spread out its probability for each value.