CHAPTER 11

Regression: Choosing and Managing Models

This chapter generalizes our understanding of regression in a number of ways. The previous chapter showed we could at least reduce training error, and quite likely improve predictions, by inserting new independent variables into a regression. The difficulty was knowing when to stop. In Sect. 11.1, I will describe some methods to search a family of models (equivalently, a set of subsets of independent variables) to find a good model. In the previous chapter, we saw how to find outlying points and remove them. In Sect. 11.2, I will describe methods to compute a regression that is largely unaffected by outliers. The resulting methods are powerful, but fairly intricate.

To date, we have used regression to predict a number. With a linear model, it is difficult to predict a probability—because linear models can predict negative numbers or numbers bigger than one—or a count—because linear models can predict non-integers. A very clever trick (Sect. 11.3) uses regression to predict the parameter of a carefully chosen probability distribution, and so probabilities, counts, and so on.

Finally, Sect. 11.4 describes methods to force regression models to choose a small set of predictors from a large set, and so produce sparse models. These methods allow us to fit regressions to data where we have more predictors than examples, and often result in significantly improved predictions. Most of the methods in this chapter can be used together to build sophisticated and accurate regressions in quite surprising circumstances.

11.1 Model Selection: Which Model Is Best?

It is usually quite easy to have many explanatory variables in a regression problem. Even if you have only one measurement, you could always compute a variety of non-linear functions of that measurement. As we have seen, inserting variables into a model will reduce the fitting cost, but that doesn't mean that better predictions will result (Sect. 10.4.1). We need to choose which explanatory variables we will use. A linear model with few explanatory variables may make poor predictions because the model itself is incapable of representing the independent variable accurately. A linear model with many explanatory variables may make poor predictions because we can't estimate the coefficients well. Choosing which explanatory variables we will use (and so which model we will use) requires that we balance these effects.

D. Forsyth, *Applied Machine Learning*,
https://doi.org/10.1007/978-3-030-18114-7_11

11.1.1 Bias and Variance

We now look at the process of finding a model in a fairly abstract way. Doing so makes plain three distinct and important effects that cause models to make predictions that are wrong. One is **irreducible error**. Even a perfect choice of model can make mistaken predictions, because more than one prediction could be correct for the same $\mathbf{x}$. Another way to think about this is that there could be many future data items, all of which have the same $\mathbf{x}$, but each of which has a different y. In this case some of our predictions must be wrong, and the effect is unavoidable.

A second effect is **bias**. We must use some collection of models. Even the best model in the collection may not be capable of predicting all the effects that occur in the data. Errors that are caused by the best model still not being able to predict the data accurately are attributed to bias.

The third effect is **variance**. We must choose our model from the collection of models. The model we choose is unlikely to be the best model. This might occur, for example, because our estimates of the parameters aren't exact because we have a limited amount of data. Errors that are caused by our choosing a model that is not the best in the family are attributed to variance.

All this can be written out in symbols. We have a vector of predictors $\mathbf{x}$, and a random variable Y. At any given point $\mathbf{x}$, we have

$$Y = f(\mathbf{x}) + \xi$$

where ξ is noise and f is an unknown function. We have

$$\mathbb{E}[\xi] = 0 \text{ and } \mathbb{E}\left[\xi^2\right] = \mathsf{var}\left(\{\xi\}\right) = \sigma_\xi^2.$$

The noise ξ is independent of X. We have some procedure that takes a selection of training data, consisting of pairs $(\mathbf{x}_i, y_i)$, and selects a model $\hat{f}$. We will use this model to predict values for future $\mathbf{x}$. It is highly unlikely that $\hat{f}$ is the same as f; assuming that it is involves assuming that we can perfectly estimate the best model with a finite dataset, which doesn't happen.

We need to understand the error that will occur when we use $\hat{f}$ to predict for some data item that isn't in the training set. This is the error that we will encounter in practice. The error at any point $\mathbf{x}$ is

$$\mathbb{E}\left[(Y - \hat{f}(\mathbf{x}))^2\right]$$

where the expectation is taken over $P(Y, \text{training data}|\mathbf{x})$. But the new query point $\mathbf{x}$ does not depend on the training data and the value Y does not depend on the training data either, so the distribution is $P(Y|\mathbf{x}) \times P(\text{training data})$.

The expectation can be written in an extremely useful form. Recall $\mathsf{var}[U] = \mathbb{E}\left[U^2\right] - \mathbb{E}[U]^2$. This means we have

$$\begin{aligned}\mathbb{E}\left[(Y - \hat{f}(\mathbf{x}))^2\right] &= \mathbb{E}\left[Y^2\right] - 2\mathbb{E}\left[Y\hat{f}\right] + \mathbb{E}\left[\hat{f}^2\right] \\ &= \mathsf{var}[Y] + \mathbb{E}[Y]^2 - 2\mathbb{E}\left[Y\hat{f}\right] + \mathsf{var}\left[\hat{f}\right] + \mathbb{E}\left[\hat{f}\right]^2.\end{aligned}$$

Now $Y = f(X) + \xi$, $\mathbb{E}[\xi] = 0$, and ξ is independent of X so we have $\mathbb{E}[Y] = \mathbb{E}[f]$, $\mathbb{E}\left[Y\hat{f}\right] = \mathbb{E}\left[(f+\xi)\hat{f}\right] = \mathbb{E}\left[f\hat{f}\right]$, and $\mathsf{var}[Y] = \mathsf{var}[\xi] = \sigma_\xi^2$. This yields

$$\begin{aligned}
\mathbb{E}\left[(Y - \hat{f}(\mathbf{x}))^2\right] &= \mathsf{var}[Y] + \mathbb{E}[f]^2 - 2\mathbb{E}\left[f\hat{f}\right] + \mathsf{var}\left[\hat{f}\right] + \mathbb{E}\left[\hat{f}\right]^2 \\
&= \mathsf{var}[Y] + f^2 - 2f\mathbb{E}\left[\hat{f}\right] + \mathsf{var}\left[\hat{f}\right] + \mathbb{E}\left[\hat{f}\right]^2 (f \text{ isn't random}) \\
&= \sigma_\xi^2 + (f - \mathbb{E}\left[\hat{f}\right])^2 + \mathsf{var}\left[\hat{f}\right]
\end{aligned}$$

The expected error on all future data is the sum of three terms.

- The irreducible error is σ_ξ^2; even the true model must produce this error, on average. There is nothing we can do about this error.
- The bias is $(f - \mathbb{E}\left[\hat{f}\right])^2$. This term reflects the fact that even the best choice of model ($\mathbb{E}\left[\hat{f}\right]$) may not be the same as the true source of data (f).
- The variance is $\mathsf{var}\left[\hat{f}\right] = \mathbb{E}\left[(\hat{f} - \mathbb{E}\left[\hat{f}\right])^2\right]$. To interpret this term, notice the best model to choose would be $\mathbb{E}\left[\hat{f}\right]$ (remember, the expectation is over choices of training data; this model would be the one that best represented all possible attempts to train). Then the variance represents the fact that the model we chose ($\hat{f}$) is different from the best model ($\mathbb{E}\left[\hat{f}\right]$). The difference arises because our training data is a subset of all data, and our model is chosen to be good on the training data, rather than on every possible training set.

Irreducible error is easily dealt with; nothing we do will improve this error, so there is no need to do anything. But there is an important practical trade-off between bias and variance. Generally, when a model comes from a "small" or "simple" family, we expect that (a) we can estimate the best model in the family reasonably accurately (so the variance will be low) but (b) the model may have real difficulty reproducing the data (meaning the bias is large). Similarly, if the model comes from a "large" or "complex" family, the variance is likely to be high (because it will be hard to estimate the best model in the family accurately) but the bias will be low (because the model can more accurately reproduce the data). All modelling involves managing this trade-off between bias and variance. I am avoiding being precise about the complexity of a model because it can be tricky to do. One reasonable proxy is the number of parameters we have to estimate to determine the model.

You can see a crude version of this trade-off in the perch example of Sect. 10.4.1 and Fig. 10.11. Recall that, as I added monomials to the regression of weight against length, the fitting error went down; but the model that uses length^{10} as an explanatory variable makes very odd predictions away from the training data. When I use low degree monomials, the dominant source of error is bias; and when I use high degree monomials, the dominant source of error is variance. A common mistake is to feel that the major difficulty is bias, and so to use extremely complex models. Usually the result is poor estimates of model parameters, leading to huge

errors from variance. Experienced modellers fear variance far more than they fear bias.

The bias–variance discussion suggests it isn't a good idea simply to use all the explanatory variables that you can obtain (or think of). Doing so might lead to a model with serious variance problems. Instead, we must choose a model that uses a subset of the explanatory variables that is small enough to control variance, and large enough that the bias isn't a problem. We need some strategy to choose explanatory variables. The simplest (but by no means the best; we'll see better in this chapter) approach is to search sets of explanatory variables for a good set. The main difficulty is knowing when you have a good set.

Remember This: *There are three kinds of error. Nothing can be done about irreducible error. Bias is the result of a family of models none of which can fit the data exactly. Variance is the result of difficulty estimating which model in the family to use. Generally, there is a payoff between bias and variance—using simpler model families causes more bias and less variance, and so on.*

11.1.2 Choosing a Model Using Penalties: AIC and BIC

We would like to choose one of a set of models. We cannot do so using just the training error, because more complex models will tend to have lower training error, and so the model with the lowest training error will tend to be the most complex model. Training error is a poor guide to test error, because lower training error is evidence of lower bias on the models part; but with lower bias, we expect to see greater variance, and the training error doesn't take that into account.

One strategy is to penalize the model for complexity. We add some penalty, reflecting the complexity of the model, to the training error. We then expect to see the general behavior of Fig. 11.1. The training error goes down, and the penalty goes up as the model gets more complex, so we expect to see a point where the sum is at a minimum.

There are a variety of ways of constructing penalties. **AIC** (short for an information criterion) is a method due originally to H. Akaike, described in "A new look at the statistical model identification," IEEE Transactions on Automatic Control, 1974. Rather than using the training error, AIC uses the maximum value of the log-likelihood of the model. Write $\mathcal{L}$ for this value. Write k for the number of parameters estimated to fit the model. Then the AIC is

$$2k - 2\mathcal{L}$$

and a better model has a smaller value of AIC (remember this by remembering that a larger log-likelihood corresponds to a better model). Estimating AIC is

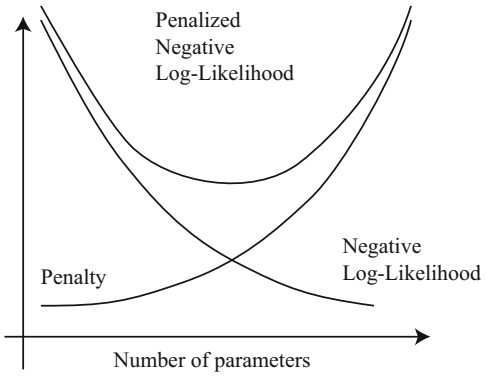

Figure 11.1: This is a standard abstract picture of a family of models. As we add explanatory variables (and so parameters) to produce a more complex model, the value of the negative log-likelihood of the best model can't go up, and usually goes down. This means that we cannot use the value as a guide to how many explanatory variables there should be. Instead, we add a penalty that increases as a function of the complexity of the model, and search for the model that minimizes the sum of negative log-likelihood and penalty. AIC and BIC penalize complexity with a penalty that is linear in the number of parameters, but there are other possible penalties. In this figure, I am following the usual convention of plotting the penalty as a curve rather than a straight line

straightforward for regression models if you assume that the noise is a zero mean normal random variable. You estimate the mean-squared error, which gives the variance of the noise, and so the log-likelihood of the model. You do have to keep track of two points. First, k is the total number of parameters estimated to fit the model. For example, in a linear regression model, where you model y as $\mathbf{x}^T\beta + \xi$, you need to estimate d parameters to estimate $\hat{\beta}$ *and* the variance of ξ (to get the log-likelihood). So in this case $k = d + 1$. Second, log-likelihood is usually only known up to a constant, so that different software implementations often use different constants. This is wildly confusing when you don't know about it (why would `AIC` and `extractAIC` produce different numbers on the same model?) but of no real significance—you're looking for the smallest value of the number, and the actual value doesn't mean anything. Just be careful to compare only numbers computed with the same routine.

An alternative is **BIC** (Bayes' information criterion), given by

$$2k \log N - 2\mathcal{L}$$

(where N is the size of the training dataset). You will often see this written as $2\mathcal{L} - 2k \log N$; I have given the form above so that one always wants the smaller value as with AIC. There is a considerable literature comparing AIC and BIC. AIC has a mild reputation for overestimating the number of parameters required, but is often argued to have firmer theoretical foundations.

Worked Example 11.1 *AIC and BIC*

Write M_d for the model that predicts weight from length for the perch dataset as $\sum_{j=0}^{j=d} \beta_j \text{length}^j$. Choose an appropriate value of $d \in [1, 10]$ using AIC and BIC.

Solution: I used the R functions `AIC` and `BIC`, and got the table below.

	1	2	3	4	5	6	7	8	9	10
AIC	677	617	617	613	615	617	617	612	613	614
BIC	683	625	627	625	629	633	635	633	635	638

The best model by AIC has (rather startlingly!) $d = 8$. One should not take small differences in AIC too seriously, so models with $d = 4$ and $d = 9$ are fairly plausible, too. BIC suggests $d = 2$.

Remember This: *AIC and BIC are methods for computing a penalty that increases as the complexity of the model increases. We choose a model that gets a low value of penalized negative log-likelihood.*

11.1.3 Choosing a Model Using Cross-Validation

AIC and BIC are estimates of error on future data. An alternative is to measure this error on held-out data, using a cross-validation strategy (as in Sect. 1.1.3). One splits the training data into F **folds**, where each data item lies in exactly one fold. The case $F = N$ is sometimes called "leave-one-out" cross-validation. One then sets aside one fold in turn, fitting the model to the remaining data, and evaluating the model error on the left-out fold. The model error is then averaged. This process gives us an estimate of the performance of a model on held-out data. Numerous variants are available, particularly when lots of computation and lots of data are available. For example, one might not average over all folds; one might use fewer or more folds; and so on.

Worked Example 11.2 *Cross-Validation*

Write M_d for the model that predicts weight from length for the perch dataset as $\sum_{j=0}^{j=d} \beta_j \text{length}^j$. Choose an appropriate value of $d \in [1, 10]$ using leave-one-out cross-validation.

Solution: I used the R functions `CVlm`, which takes a bit of getting used to. I found:

1	2	3	4	5	6	7	8	9	10
1.9e4	4.0e3	7.2e3	4.5e3	6.0e3	5.6e4	1.2e6	4.0e6	3.9e6	1.9e8

where the best model is $d = 2$.

11.1.4 Greedy Search with Stagewise Regression

Assume we have a set of explanatory variables and we wish to build a model, choosing some of those variables for our model. Our explanatory variables could be many distinct measurements, or they could be different non-linear functions of the same measurement, or a combination of both. We can evaluate models relative to one another fairly easily (AIC, BIC, or cross-validation, your choice). However, choosing which set of explanatory variables to use can be quite difficult, because there are so many sets. The problem is that you cannot predict easily what adding or removing an explanatory variable will do. Instead, when you add (or remove) an explanatory variable, the errors that the model makes change, and so the usefulness of all other variables changes too. This means that (at least in principle) you have to look at every subset of the explanatory variables. Imagine you start with a set of F possible explanatory variables (including the original measurement, and a constant). You don't know how many to use, so you might have to try every different group, of each size, and there are far too many groups to try. There are two useful alternatives.

In **forward stagewise regression**, you start with an empty working set of explanatory variables. You then iterate the following process. For each of the explanatory variables not in the working set, you construct a new model using the working set and that explanatory variable, and compute the model evaluation score. If the best of these models has a better score than the model based on the working set, you insert the appropriate variable into the working set and iterate. If no variable improves the working set, you decide you have the best model and stop. This is fairly obviously a greedy algorithm.

Backward stagewise regression is pretty similar, but you start with a working set containing all the variables, and remove variables one-by-one and greedily. As usual, greedy algorithms are very helpful but not capable of exact optimization. Each of these strategies can produce rather good models, but neither is guaranteed to produce the best model.

Remember This: *Forward and backward stagewise regression are greedy searches for sets of independent variables that predict effectively. In forward stagewise regression, one adds variables to a regression; in backward, one removes variables from the regression. Success can be checked with AIC, BIC, or cross-validation. The search stops when adding (resp. removing) a variable makes the regression worse.*

11.1.5 What Variables Are Important?

Imagine you regress some measure of risk of death against blood pressure, whether someone smokes or not, and the length of their thumb. Because high blood pressure and smoking tend to increase risk of death, you would expect to see "large" coefficients for these explanatory variables. Since changes in the thumb length have no effect, you would expect to see "small" coefficients for these explanatory variables. You might think that this suggests a regression can be used to determine what effects are important in building a model. It can, but doing so correctly involves serious difficulties that I will not deal with in detail. Instead, I will sketch what can go wrong so that you're discouraged from doing this without learning quite a lot more.

One difficulty is the result of variable scale. If you measure thumb length in kilometers, the coefficient is likely small; if you measure thumb length in micrometers, the coefficient is likely large. But this change has nothing to do with how important the variable is for prediction. This means that interpreting the coefficient is tricky.

Another difficulty is the result of sampling variance. Imagine that we have an explanatory variable that has absolutely no relationship to the dependent variable. If we had an arbitrarily large amount of data, and could exactly identify the correct model, we'd find that, in the correct model, the coefficient of that variable was zero. But we don't have an arbitrarily large amount of data. Instead, we have a sample of data. Hopefully, our sample is random so that (with some work) our estimate of the coefficient is the value of a random variable whose expected value is zero, but whose variance isn't. This means we are very unlikely to see a zero, but should see a value which is a small number of standard deviations away from zero. Dealing with this requires a way to tell whether the difference between a coefficient and zero is meaningful, or is just the result of random effects. There is a theory of **statistical significance** for regression coefficients, but we have other things to do.

Yet another difficulty has to do with practical significance, and is rather harder. We could have explanatory variables that are genuinely linked to the independent variable, but might not matter very much. This is a common phenomenon, particularly in medical statistics. It requires considerable care to disentangle some of these issues. Here is an example. Bowel cancer is a nasty disease, which could kill you. Being screened for bowel cancer is at best embarrassing and unpleasant, and involves some startling risks. There is considerable doubt, from reasonable sources, about whether screening has value and if so, how much (as a start point,

you could look at Ransohoff DF. "How Much Does Colonoscopy Reduce Colon Cancer Mortality?" which appears in Ann. Intern. Med. 2009). There is some evidence linking eating red or processed meat to incidence of bowel cancer. A good practical question is: should one abstain from eating red or processed meat based on increased bowel cancer risk?

Coming to an answer is tough; the coefficient in any regression is clearly not zero, but it's pretty small. Here are some numbers. The UK population in 2012 was 63.7 million (this is a summary figure from Google, using World Bank data; there's no reason to believe that it's significantly wrong). I obtained the following figures from the UK cancer research institute website, at http://www.cancerresearchuk.org/health-professional/cancer-statistics/statistics-by-cancer-type/bowel-cancer. There were 41,900 new cases of bowel cancer in the UK in 2012. Of these cases, 43% occurred in people aged 75 or over. Fifty-seven percent of people diagnosed with bowel cancer survive for 10 years or more after diagnosis. Of diagnosed cases, an estimated 21% is linked to eating red or processed meat, and the best current estimate is that the risk of incidence is between 17 and 30% higher per 100 g of red meat eaten per day (i.e., if you eat 100 g of red meat per day, your risk increases by some number between 17 and 30%; 200 g a day gets you twice that number; and—rather roughly—so on). These numbers are enough to confirm that there is a non-zero coefficient linking the amount of red or processed meat in your diet with your risk of bowel cancer (though you'd have a tough time estimating the exact value of that coefficient from the information here). If you eat more red meat, your risk of dying of bowel cancer really will go up. But the numbers I gave above suggest that (a) it won't go up much and (b) you might well die rather late in life, where the chances of dying of something are quite strong. The coefficient linking eating red meat and bowel cancer is clearly pretty small, because the incidence of the disease is about 1 in 1500 per year. Does the effect of this link matter enough to (say) stop eating red or processed meat? you get to choose, and your choice has consequences.

Remember This: *There are serious pitfalls in trying to interpret the coefficients of a regression. A small coefficient might come from a choice of scale for the associated variable. A large coefficient might still be the result of random effects, and assessing whether it requires a model of statistical significance. Worse, a coefficient might be clearly non-zero, but have little practical significance. It's tempting to look at the coefficients and try and come to conclusions, but you should not do this without much more theory.*

11.2 Robust Regression

We have seen that outlying data points can result in a poor model. This is caused by the squared error cost function: squaring a large error yields an enormous number. One way to resolve this problem is to identify and remove outliers before fitting a

model. This can be difficult, because it can be hard to specify precisely when a point is an outlier. Worse, in high dimensions most points will look somewhat like outliers, and we may end up removing almost all the data. The alternative solution I offer here is to come up with a cost function that is less susceptible to problems with outliers. The general term for a regression that can ignore some outliers is a **robust regression**.

11.2.1 M-Estimators and Iteratively Reweighted Least Squares

One way to reduce the effect of outliers on a least squares solution would be to weight each point in the cost function. We need some method to estimate an appropriate set of weights. This would use a large weight for errors at points that are "trustworthy," and a low weight for errors at "suspicious" points.

We can obtain such weights using an **M-estimator**, which estimates parameters by replacing the negative log-likelihood with a term that is better behaved. In our examples, the negative log-likelihood has always been squared error. Write β for the parameters of the model being fitted, and $r_i(\mathbf{x}_i, \beta)$ for the residual error of the model on the ith data point. For us, r_i will always be $y_i - \mathbf{x}_i^T\beta$. So rather than minimizing

$$\sum_i (r_i(\mathbf{x}_i, \beta))^2$$

as a function of β, we will minimize an expression of the form

$$\sum_i \rho(r_i(\mathbf{x}_i, \beta); \sigma),$$

for some appropriately chosen function ρ. Clearly, our negative log-likelihood is one such estimator (use $\rho(u; \sigma) = u^2$). The trick to M-estimators is to make $\rho(u; \sigma)$ look like u^2 for smaller values of u, but ensure that it grows more slowly than u^2 for larger values of u (Fig. 11.2).

The **Huber loss** is one important M-estimator. We use

$$\rho(u; \sigma) = \begin{cases} \frac{u^2}{2} & |u| < \sigma \\ \sigma|u| - \frac{\sigma^2}{2} & \end{cases}$$

which is the same as u^2 for $-\sigma \leq u \leq \sigma$, and then switches to $|u|$ for larger (or smaller) σ. The Huber loss is convex (meaning that there will be a unique minimum for our models) and differentiable, but its derivative is not continuous. The choice of the parameter σ (which is known as **scale**) has an effect on the estimate. You should interpret this parameter as the distance that a point can lie from the fitted function while still being seen as an **inlier** (anything that isn't even partially an outlier) (Fig. 11.3).

Generally, M-estimators are discussed in terms of their **influence function**. This is

$$\frac{\partial \rho}{\partial u}.$$

Its importance becomes evidence when we consider algorithms to fit $\hat{\beta}$ using an

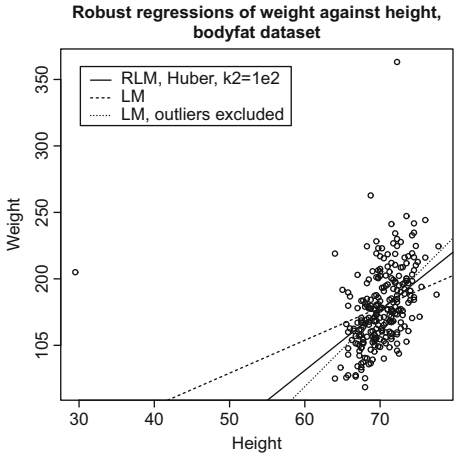

Figure 11.2: Comparing three different linear regression strategies on the bodyfat data, regressing weight against height. Notice that using an M-estimator gives an answer very like that obtained by rejecting outliers by hand. The answer may well be "better" because it isn't certain that each of the four points rejected is an outlier, and the robust method may benefit from some of the information in these points. I tried a range of scales for the Huber loss (the "k2" parameter), but found no difference in the line resulting over scales varying by a factor of 1e4, which is why I plot only one scale

M-estimator. Our minimization criterion is

$$\begin{aligned}\nabla_\beta \left(\sum_i \rho(y_i - \mathbf{x}_i^T \beta; \sigma) \right) &= \sum_i \left[\frac{\partial \rho}{\partial u} \right] (-\mathbf{x}_i) \\ &= 0.\end{aligned}$$

Here the derivative $\frac{\partial \rho}{\partial u}$ is evaluated at $y_i - \mathbf{x}_i^T \beta$, so it is a function of β. Now write $w_i(\beta)$ for

$$\frac{\frac{\partial \rho}{\partial u}}{y_i - \mathbf{x}_i^T \beta}$$

(again, where the derivative is evaluated at $y_i - \mathbf{x}_i^T \beta$, and so w_i is a function of β). We can write the minimization criterion as

$$\sum_i \left[w_i(\beta) \right] \left[y_i - \mathbf{x}_i^T \beta \right] \left[-\mathbf{x}_i \right] = 0.$$

Now write $\mathcal{W}(\beta)$ for the diagonal matrix whose i'th diagonal entry is $w_i(\beta)$. Then our fitting criterion is equivalent to

$$\mathcal{X}^T \left[\mathcal{W}(\beta) \right] \mathbf{y} = \mathcal{X}^T \left[\mathcal{W}(\beta) \right] \mathcal{X} \beta.$$

The difficulty in solving this is that $w_i(\beta)$ depend on β, so we can't just solve a linear system in β. We could use the following strategy. Find some initial $\hat{\beta}^{(1)}$.

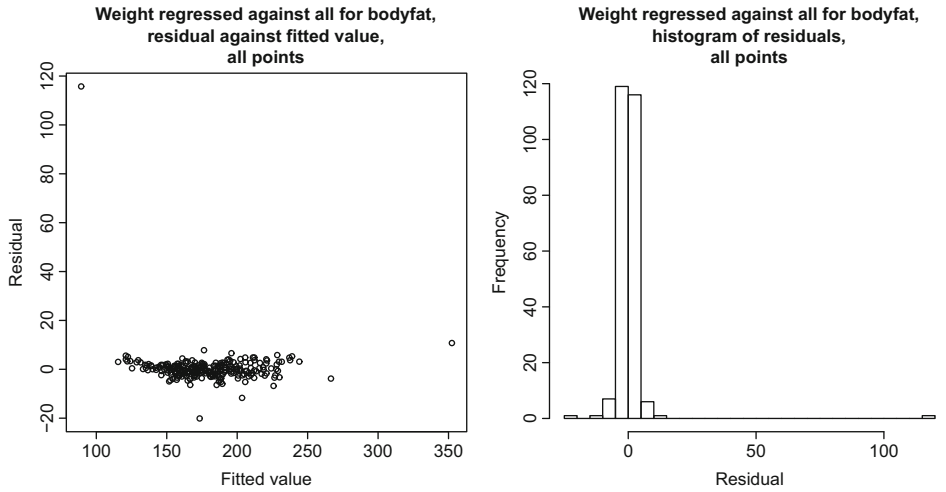

Figure 11.3: A robust linear regression of weight against all variables for the bodyfat dataset, using the Huber loss and all data points. On the **left**, residual plotted against fitted value (the residual is not standardized). Notice that there are some points with very large residual, but most have much smaller residual; this wouldn't happen with a squared error. On the **right**, a histogram of the residual. If one ignores the extreme residual values, this looks normal. The robust process has been able to discount the effect of the outliers, without us needing to identify and reject outliers by hand

Now evaluate $\mathcal{W}$ using that estimate, and re-estimate by solving the linear system. Iterate this until it settles down. This process uses $\mathcal{W}$ to downweight points that are suspiciously inconsistent with our current estimate of β, then update β using those weights. The strategy is known as **iteratively reweighted least squares**, and is very effective.

We assume we have an estimate of the correct parameters $\hat{\beta}^{(n)}$, and consider updating it to $\hat{\beta}^{(n+1)}$. We compute

$$w_i^{(n)} = w_i(\hat{\beta}^{(n)}) = \frac{\frac{\partial \rho}{\partial u}(y_i - \mathbf{x}_i^T \beta^{(n)}; \sigma)}{y_i - \mathbf{x}_i^T \hat{\beta}^{(n)}}.$$

We then estimate $\hat{\beta}^{(n+1)}$ by solving

$$\mathcal{X}^T \mathcal{W}^{(n)} \mathbf{y} = \mathcal{X}^T \mathcal{W}^{(n)} \mathcal{X} \hat{\beta}^{(n+1)}.$$

The key to this algorithm is finding good start points for the iteration. One strategy is randomized search. We select a small subset of points uniformly at random, and fit some $\hat{\beta}$ to these points, then use the result as a start point. If we do this often enough, one of the start points will be an estimate that is not contaminated by outliers.

Procedure: 11.1 *Fitting a Regression with Iteratively Reweighted Least Squares*

Write r_i for the residual at the i'th point, $y_i - \mathbf{x}_i^T\beta$. Choose an M-estimator ρ, likely the Huber loss; write $w_i(\beta)$ for

$$\frac{\frac{\partial \rho}{\partial u}}{y_i - \mathbf{x}_i^T\beta}.$$

We will minimize

$$\sum_i \rho(r_i(\mathbf{x}_i, \beta); \sigma)$$

by repeatedly

- finding some initial $\hat{\beta}^{(1)}$ by selecting a small subset of points uniformly at random and fitting a regression to those points;
- iterating the following procedure until the update is very small
  1. compute $\mathcal{W}^{(n)} = \text{diag}(w_i(\hat{\beta}^{(n)}))$;
  2. solve
  $$\mathcal{X}^T\mathcal{W}^{(n)}\mathbf{y} = \mathcal{X}^T\mathcal{W}^{(n)}\mathcal{X}\hat{\beta}^{(n+1)}$$
  for $\hat{\beta}^{(n+1)}$;
- keep the resulting $\hat{\beta}$ if $\sum_i \rho(r_i(\mathbf{x}_i, \hat{\beta}); \sigma)$ is smaller than any seen so far.

11.2.2 Scale for M-Estimators

The estimators require a sensible estimate of σ, which is often referred to as **scale**. Typically, the scale estimate is supplied at each iteration of the solution method. One reasonable estimate is the **MAD** or **median absolute deviation**, given by

$$\sigma^{(n)} = 1.4826 \text{ median}_i |\, r_i^{(n)}(x_i; \hat{\beta}^{(n-1)}) \,|.$$

Another popular estimate of scale is obtained with **Huber's proposal 2** (that is what everyone calls it!). Choose some constant $k_1 > 0$, and define $\Xi(u) = \min(|\, u \,|, k_1)^2$. Now solve the following equation for σ:

$$\sum_i \Xi\left(\frac{r_i^{(n)}(x_i; \hat{\beta}^{(n-1)})}{\sigma}\right) = Nk_2$$

where k_2 is another constant, usually chosen so that the estimator gives the right answer for a normal distribution (exercises). This equation needs to be solved with an iterative method; the MAD estimate is the usual start point. R provides `hubers`, which will compute this estimate of scale (and figures out k_2 for itself). The choice of k_1 depends somewhat on how contaminated you expect your data to be. As $k_1 \to \infty$, this estimate becomes more like the standard deviation of the data.

11.3 Generalized Linear Models

We have used a linear regression to predict a value from a feature vector, but implicitly have assumed that this value is a real number. Other cases are important, and some of them can be dealt with using quite simple generalizations of linear regression. When we derived linear regression, I said one way to think about the model was

$$y = \mathbf{x}^T\beta + \xi$$

where ξ was a normal random variable with zero mean and variance σ_ξ^2. Another way to write this is to think of y as the value of a random variable Y. In this case, Y has mean $\mathbf{x}^T\beta$ and variance σ_ξ^2. This can be written as

$$Y \sim N(\mathbf{x}^T\beta, \sigma_\xi^2).$$

This offers a fruitful way to generalize: we replace the normal distribution with some other parametric distribution, and predict the parameter of that distribution using $\mathbf{x}^T\beta$. This is a **generalized linear model** or **GLM**. Three examples are particularly important.

11.3.1 Logistic Regression

Assume the y values can be either 0 or 1. You could think of this as a two-class classification problem, and deal with it using an SVM. There are sometimes advantages to seeing it as a regression problem. One is that we get to see a new classification method that explicitly models class posteriors, which an SVM doesn't do.

We build the model by asserting that the y values represent a draw from a Bernoulli random variable (definition below, for those who have forgotten). The parameter of this random variable is θ, the probability of getting a one. But $0 \leq \theta \leq 1$, so we can't just model θ as $\mathbf{x}^T\beta$. We will choose some **link function** g so that we can model $g(\theta)$ as $\mathbf{x}^T\beta$. This means that, in this case, g must map the interval between 0 and 1 to the whole line, and must be 1–1. The link function maps θ to $\mathbf{x}^T\beta$; the direction of the map is chosen by convention. We build our model by asserting that $g(\theta) = \mathbf{x}^T\beta$.

Remember This: *A generalized linear model predicts the parameter of a probability distribution from a regression. The link function ensures that the prediction of the regression meets the constraints required by the distribution.*

Useful Fact: 11.1 *Definition: Bernoulli Random Variable*

A Bernoulli random variable with parameter θ takes the value 1 with probability θ and 0 with probability $1 - \theta$. This is a model for a coin toss, among other things.

Notice that, for a Bernoulli random variable, we have that

$$\log\left[\frac{P(y=1|\theta)}{P(y=0|\theta)}\right] = \log\left[\frac{\theta}{1-\theta}\right]$$

and the **logit function** $g(u) = \log\left[\frac{u}{1-u}\right]$ meets our needs for a link function (it maps the interval between 0 and 1 to the whole line, and is 1–1). This means we can build our model by asserting that

$$\log\left[\frac{P(y=1|\mathbf{x})}{P(y=0|\mathbf{x})}\right] = \mathbf{x}^T\beta$$

then solving for the β that maximizes the log-likelihood of the data. Simple manipulation yields

$$P(y=1|\mathbf{x}) = \frac{e^{\mathbf{x}^T\beta}}{1+e^{\mathbf{x}^T\beta}} \text{ and } P(y=0|\mathbf{x}) = \frac{1}{1+e^{\mathbf{x}^T\beta}}.$$

In turn, this means the log-likelihood of a dataset will be

$$\mathcal{L}(\beta) = \sum_i \left[\mathbb{I}_{[y=1]}(y_i)\mathbf{x}_i^T\beta - \log\left(1+e^{\mathbf{x}_i^T\beta}\right)\right].$$

You can obtain β from this log-likelihood by gradient ascent (or rather a lot faster by Newton's method, if you know that).

A regression of this form is known as a **logistic regression**. It has the attractive property that it produces estimates of posterior probabilities. Another interesting property is that a logistic regression is a lot like an SVM. To see this, we replace the labels with new ones. Write $\hat{y}_i = 2y_i - 1$; this means that $\hat{y}_i$ takes the values -1 and 1, rather than 0 and 1. Now $\mathbb{I}_{[y=1]}(y_i) = \frac{\hat{y}_i+1}{2}$, so we can write

$$\begin{aligned}
-\mathcal{L}(\beta) &= -\sum_i \left[\frac{\hat{y}_i+1}{2}\mathbf{x}_i^T\beta - \log\left(1+e^{\mathbf{x}_i^T\beta}\right)\right] \\
&= \sum_i \left[-\left(\frac{\hat{y}_i+1}{2}\mathbf{x}_i^T\beta\right) + \log\left(1+e^{\mathbf{x}_i^T\beta}\right)\right] \\
&= \sum_i \left[\log\left(\frac{1+e^{\mathbf{x}_i^T\beta}}{e^{\frac{\hat{y}_i+1}{2}\mathbf{x}_i^T\beta}}\right)\right] \\
&= \sum_i \left[\log\left(e^{\frac{-(\hat{y}_i+1)}{2}\mathbf{x}_i^T\beta} + e^{\frac{1-\hat{y}_i}{2}\mathbf{x}_i^T\beta}\right)\right]
\end{aligned}$$

and we can interpret the term in square brackets as a loss function. If you plot it, you will notice that it behaves rather like the hinge loss. When $\hat{y}_i = 1$, if $\mathbf{x}^T\beta$ is positive, the loss is very small, but if $\mathbf{x}^T\beta$ is strongly negative, the loss grows linearly in $\mathbf{x}^T\beta$. There is similar behavior when $\hat{y}_i = -1$. The transition is smooth, unlike the hinge loss. Logistic regression should (and does) behave well for the same reasons the SVM behaves well.

Be aware that logistic regression has one annoying quirk. When the data are linearly separable (i.e., there exists some β such that $y_i \mathbf{x}_i^T \beta > 0$ for all data items), logistic regression will behave badly. To see the problem, choose the β that separates the data. Now it is easy to show that increasing the magnitude of β will increase the log-likelihood of the data; there isn't any limit. These situations arise fairly seldom in practical data.

Remember This: *Logistic regression predicts the probability that a Bernoulli random variable is one using a logit link function. The result is a binary classifier whose loss is very similar to a hinge loss.*

11.3.2 Multiclass Logistic Regression

Imagine $y \in [0, 1, \ldots, C-1]$. Then it is natural to model $p(y|\mathbf{x})$ with a discrete probability distribution on these values. This can be specified by choosing $(\theta_0, \theta_1, \ldots, \theta_{C-1})$ where each term is between 0 and 1 and $\sum_i \theta_i = 1$. Our link function will need to map this constrained vector of θ values to a $\Re^{C-1}$. We can do this with a fairly straightforward variant of the logit function, too. Notice that there are $C-1$ probabilities we need to model (the C'th comes from the constraint $\sum_i \theta_i = 1$). We choose one vector β for each probability, and write β_i for the vector used to model θ_i. Then we can write

$$\mathbf{x}^T \beta_i = \log \left(\frac{\theta_i}{1 - \sum_u \theta_u} \right)$$

and this yields the model

$$\begin{aligned} P(y = 0|\mathbf{x}, \beta) &= \frac{e^{\mathbf{x}^T \beta_0}}{1 + \sum_i e^{\mathbf{x}^T \beta_i}} \\ P(y = 1|\mathbf{x}, \beta) &= \frac{e^{\mathbf{x}^T \beta_1}}{1 + \sum_i e^{\mathbf{x}^T \beta_i}} \\ &\cdots \\ P(y = C-1|\mathbf{x}, \beta) &= \frac{1}{1 + \sum_i e^{\mathbf{x}^T \beta_i}} \end{aligned}$$

and we would fit this model using maximum likelihood. The likelihood is easy to write out, and gradient descent is a good strategy for actually fitting models.

Remember This: *Multiclass logistic regression predicts a multinomial distribution using a logit link function. The result is an important multiclass classifier.*

11.3.3 Regressing Count Data

Now imagine that the y_i values are counts. For example, y_i might have the count of the number of animals caught in a small square centered on $\mathbf{x}_i$ in a study region. As another example, $\mathbf{x}_i$ might be a set of features that represent a customer, and y_i might be the number of times that customer bought a particular product. The natural model for count data is a Poisson model, with parameter θ representing the intensity (reminder below).

Useful Fact: 11.2 *Definition: Poisson Distribution*

A non-negative, integer valued random variable X has a Poisson distribution when its probability distribution takes the form

$$P(\{X = k\}) = \frac{\theta^k e^{-\theta}}{k!},$$

where $\theta > 0$ is a parameter often known as the **intensity** of the distribution.

Now we need $\theta > 0$. A natural link function is to use

$$\mathbf{x}^T\beta = \log\theta$$

yielding a model

$$P(\{X = k\}) = \frac{e^{k\mathbf{x}^T\beta}e^{-e^{\mathbf{x}^T\beta}}}{k!}.$$

Now assume we have a dataset. The negative log-likelihood can be written as

$$\begin{aligned} -\mathcal{L}(\beta) &= -\sum_i \log\left(\frac{e^{y_i\mathbf{x}_i^T\beta}e^{-e^{\mathbf{x}_i^T\beta}}}{y_i!}\right) \\ &= -\sum_i \left(y_i\mathbf{x}_i^T\beta - e^{\mathbf{x}_i^T\beta} - \log(y_i!)\right). \end{aligned}$$

There isn't a closed form minimum available, but the log-likelihood is convex, and gradient descent (or Newton's method) is enough to find a minimum. Notice that the $\log(y_i!)$ term isn't relevant to the minimization, and is usually dropped.

Remember This: *You can predict count data with a GLM by predicting the parameter of a Poisson distribution with an exponential link function.*

11.3.4 Deviance

Cross-validating a model is done by repeatedly splitting a dataset into two pieces, training on one, evaluating some score on the other, and averaging the score. But we need to keep track of *what* to score. For earlier linear regression models (e.g., Sect. 11.1), we have used the squared error of predictions. This doesn't really make sense for a generalized linear model, because predictions are of quite different form. It is usual to use the **deviance** of the model. Write y_t for the true prediction at a point, $\mathbf{x}_p$ for the independent variables we want to obtain a prediction for, $\hat{\beta}$ for our estimated parameters; a generalized linear model yields $P(y|\mathbf{x}_p, \hat{\beta})$. For our purposes, you should think of the deviance as

$$-2 \log P(y_t|\mathbf{x}_p, \hat{\beta})$$

(this expression is sometimes adjusted in software to deal with extreme cases, etc.). Notice that this is quite like the least squares error for the linear regression case, because there

$$-2 \log P(y|\mathbf{x}_p, \hat{\beta}) = (\mathbf{x}_p^T \hat{\beta} - y_t)^2/\sigma^2 + K$$

for K some constant.

Remember This: *Evaluate a GLM with the model's deviance.*

11.4 L1 Regularization and Sparse Models

Forward and backward stagewise regression were strategies for adding independent variables to, or removing independent variables from, a model. An alternative, and very powerful, strategy is to construct a model with a method that forces some coefficients to be zero. The resulting model ignores the corresponding independent variables. Models built this way are often called **sparse models**, because (one hopes) that many independent variables will have zero coefficients, and so the model is using a sparse subset of the possible predictors.

In some situations, we are forced to use a sparse model. For example, imagine there are more independent variables than there are examples. In this case, the matrix $\mathcal{X}^T\mathcal{X}$ will be rank deficient. We could use a ridge regression (Sect. 10.4.2) and the rank deficiency problem will go away, but it would be hard to trust the resulting model, because it will likely use all the predictors (more detail below). We really want a model that uses a small subset of the predictors. Then, because the model ignores the other predictors, there will be more examples than there are predictors *that we use*.

There is now quite a strong belief among practitioners that using sparse models is the best way to deal with high dimensional problems (although there are lively debates about *which* sparse model to use, etc.). This is sometimes called the "bet on sparsity" principle: use a sparse model for high dimensional data, because dense models don't work well for such problems.

11.4.1 Dropping Variables with L1 Regularization

We have a large set of explanatory variables, and we would like to choose a small set that explains most of the variance in the independent variable. We could do this by encouraging β to have many zero entries. In Sect. 10.4.2, we saw we could regularize a regression by adding a term to the cost function that discouraged large values of β. Instead of solving for the value of β that minimized $\sum_i (y_i - \mathbf{x}_i^T\beta)^2 = (\mathbf{y} - \mathcal{X}\beta)^T(\mathbf{y} - \mathcal{X}\beta)$ (which I shall call the **error cost**), we minimized

$$\sum_i (y_i - \mathbf{x}_i^T\beta)^2 + \frac{\lambda}{2}\beta^T\beta = (\mathbf{y} - \mathcal{X}\beta)^T(\mathbf{y} - \mathcal{X}\beta) + \frac{\lambda}{2}\beta^T\beta$$

(which I shall call the **L2 regularized error**). Here $\lambda > 0$ was a constant chosen by cross-validation. Larger values of λ encourage entries of β to be small, but do not force them to be zero. The reason is worth understanding.

Write β_k for the kth component of β, and write β_{-k} for all the other components. Now we can write the L2 regularized error as a function of β_k:

$$(a + \lambda)\beta_k^2 - 2b(\beta_{-k})\beta_k + c(\beta_{-k})$$

where a is a function of the data and b and c are functions of the data and of β_{-k}. Now notice that the best value of β_k will be

$$\beta_k = \frac{b(\beta_{-k})}{(a + \lambda)}.$$

Notice that λ doesn't appear in the numerator. This means that, to force β_k to zero by increasing λ, we may have to make λ arbitrarily large. This is because the improvement in the penalty obtained by going from a small β_k to $\beta_k = 0$ is tiny—the penalty is proportional to β_k^2.

To force some components of β to zero, we need a penalty that grows linearly around zero rather than quadratically. This means we should use the $\mathbf{L}_1$ **norm** of β, given by

$$\|\beta\|_1 = \sum_k |\beta_k|.$$

To choose β, we must now solve

$$(\mathbf{y} - \mathcal{X}\beta)^T(\mathbf{y} - \mathcal{X}\beta) + \lambda\|\beta\|_1$$

for an appropriate choice of λ. An equivalent problem is to solve a constrained minimization problem, where one minimizes

$$(\mathbf{y} - \mathcal{X}\beta)^T(\mathbf{y} - \mathcal{X}\beta) \text{ subject to } \|\beta\|_1 \leq t$$

where t is some value chosen to get a good result, typically by cross-validation. There is a relationship between the choice of t and the choice of λ (with some thought, a smaller t will correspond to a bigger λ) but it isn't worth investigating in any detail.

Actually solving this system is quite involved, because the cost function is not differentiable. You should *not* attempt to use stochastic gradient descent, because

this will not compel zeros to appear in $\hat{\beta}$. There are several methods, which are beyond our scope. As the value of λ increases, the number of zeros in $\hat{\beta}$ will increase too. We can choose λ in the same way we used for classification; split the training set into a training piece and a validation piece, train for different values of λ, and test the resulting regressions on the validation piece. The family of solutions $\hat{\beta}(\lambda)$ for all values of $\lambda \geq 0$ is known as the **regularization path**. One consequence of modern methods is that we can generate a very good approximation to the **regularization path** about as easily as we can get a solution for a single value of λ. As a result, cross-validation procedures for choosing λ are efficient.

Remember This: *An L_1 regularization penalty encourages models to have zero coefficients. The optimization problem that results is quite specialized. A strong approximation to the regularization path can be produced relatively easily, so cross-validation to choose λ is efficient.*

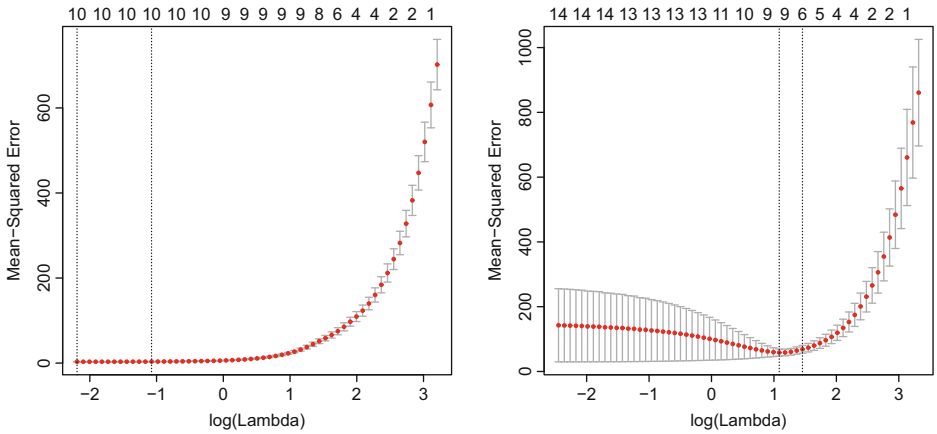

Figure 11.4: Plots of mean-squared error as a function of log regularization parameter (i.e., $\log \lambda$) for a regression of weight against all variables for the bodyfat dataset using an L1 regularizer (i.e., a lasso). These plots show mean-squared error averaged over cross-validation folds with a vertical one standard deviation bar. On the **left**, the plot for the dataset with the six outliers identified in Fig. 10.15 removed. On the **right**, the plot for the whole dataset. Notice how the outliers increase the variability of the error, and the best error. The top row of numbers gives the number of non-zero components in $\hat{\beta}$. Notice how as λ increases, this number falls (there are 15 explanatory variables, so the largest model would have 15 variables). The penalty ensures that explanatory variables with small coefficients are dropped as λ gets bigger

One way to understand the models that result is to look at the behavior of cross-validated error as λ changes. The error is a random variable, random because of the random split. It is a fair model of the error that would occur on a randomly chosen test example (assuming that the training set is "like" the test set, in a way that I do not wish to make precise yet). We could use multiple splits, and average over the splits. Doing so yields both an average error for each value of λ and an estimate of the standard deviation of error. Figure 11.4 shows the result of doing so for two datasets. Again, there is no λ that yields the smallest validation error, because the value of error depends on the random split cross-validation. A reasonable choice of λ lies between the one that yields the smallest error encountered (one vertical line in the plot) and the largest value whose mean error is within one standard deviation of the minimum (the other vertical line in the plot). It is informative to keep track of the number of zeros in $\hat{\beta}$ as a function of λ, and this is shown in Fig. 11.4.

Worked Example 11.3 *Building an L1 Regularized Regression*

Fit a linear regression to the bodyfat dataset, predicting weight as a function of all variables, and using the lasso to regularize. How good are the predictions? Do outliers affect the predictions?

Solution: I used the `glmnet` package, and I benefited a lot from example code by Trevor Hastie and Junyang Qian and published at https://web.stanford.edu/~hastie/glmnet/glmnet_alpha.html. I particularly like the R version; on my computer, the Matlab version occasionally dumps core, which is annoying. You can see from Fig. 11.4 that (a) for the case of outliers removed, the predictions are very good and (b) the outliers create problems. Note the magnitude of the error, and the low variance, for good cross-validated choices.

Another way to understand the models is to look at how $\hat{\beta}$ changes as λ changes. We expect that, as λ gets smaller, more and more coefficients become non-zero. Figure 11.5 shows plots of coefficient values as a function of $\log \lambda$ for a regression of weight against all variables for the bodyfat dataset, penalized using the L_1 norm. For different values of λ, one gets different solutions for $\hat{\beta}$. When λ is very large, the penalty dominates, and so the norm of $\hat{\beta}$ must be small. In turn, most components of $\hat{\beta}$ are zero. As λ gets smaller, the norm of $\hat{\beta}$ falls and some components of become non-zero. At first glance, the variable whose coefficient grows very large seems important. Look more carefully, this is the last component introduced into the model. But Fig. 11.4 implies that the right model has 7 components. This means that the right model has $\log \lambda \approx 1.3$, the vertical line shown in the detailed figure. In the best model, that coefficient is in fact zero.

The L_1 norm can sometimes produce an impressively small model from a large number of variables. In the UC Irvine Machine Learning repository, there is a dataset to do with the geographical origin of music (https://archive.ics.uci.edu/

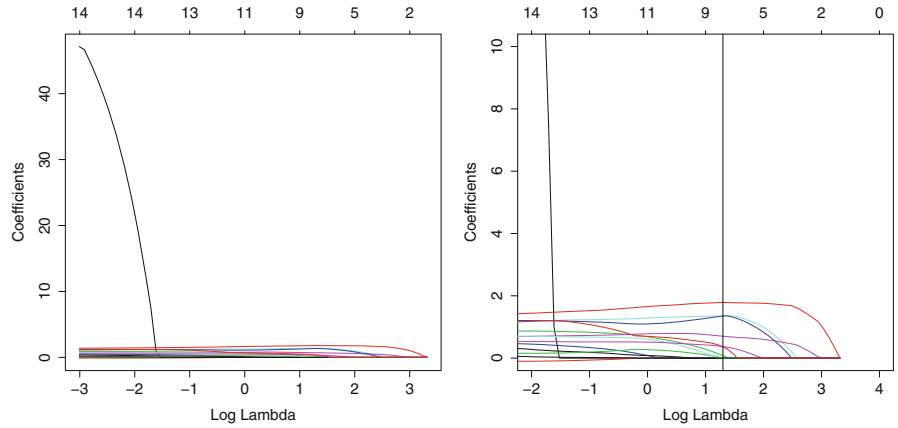

Figure 11.5: Plots of coefficient values as a function of $\log \lambda$ for a regression of weight against all variables for the bodyfat dataset, penalized using the L_1 norm. In each case, the six outliers identified in Fig. 10.15 were removed. On the **left**, the plot of the whole path for each coefficient (each curve is one coefficient). On the **right**, a detailed version of the plot. The vertical line shows the value of $\log \lambda$ that produces the model with smallest cross-validated error (look at Fig. 11.4). Notice that the variable that appears to be important, because it would have a large weight with $\lambda = 0$, does not appear in this model.

ml/datasets/Geographical+Original+of+Music). The dataset was prepared by Fang Zhou, and donors were Fang Zhou, Claire Q, and Ross D. King. Further details appear on that webpage, and in the paper: "Predicting the Geographical Origin of Music" by Fang Zhou, Claire Q, and Ross D. King, which appeared at ICDM in 2014. There are two versions of the dataset. One has 116 explanatory variables (which are various features representing music), and 2 independent variables (the latitude and longitude of the location where the music was collected). Figure 11.6 shows the results of a regression of latitude against the independent variables using L_1 regularization. Notice that the model that achieves the lowest cross-validated prediction error uses only 38 of the 116 variables.

Regularizing a regression with the L_1 norm is sometimes known as a **lasso**. A nuisance feature of the lasso is that, if several explanatory variables are correlated, it will tend to choose one for the model and omit the others (example in exercises). This can lead to models that have worse predictive error than models chosen using the L_2 penalty. One nice feature of good minimization algorithms for the lasso is that it is easy to use both an L_1 penalty and an L_2 penalty together. One can form

$$\underbrace{\left(\frac{1}{N}\right)\left(\sum_i (y_i - \mathbf{x}_i^T\beta)^2\right)}_{\text{Error}} + \underbrace{\lambda\left(\frac{(1-\alpha)}{2}\|\beta\|_2{}^2 + \alpha\|\beta\|_1\right)}_{\text{Regularizer}}$$

where one usually chooses $0 \leq \alpha \leq 1$ by hand. Doing so can both discourage large values in β and encourage zeros. Penalizing a regression with a mixed norm like this

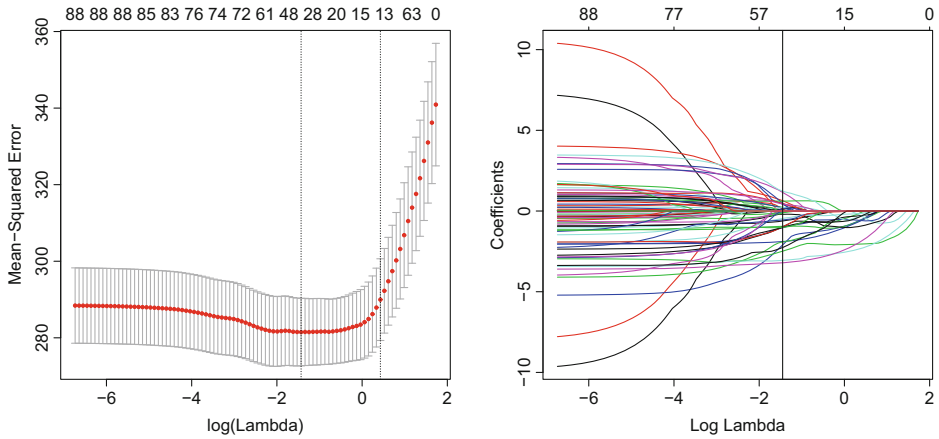

Figure 11.6: Mean-squared error as a function of log regularization parameter (i.e., $\log \lambda$) for a regression of latitude against features describing music (details in text), using the dataset at https://archive.ics.uci.edu/ml/datasets/Geographical+Original+of+Music and penalized with the L_1 norm. The plot on the **left** shows mean-squared error averaged over cross-validation folds with a vertical one standard deviation bar. The top row of numbers gives the number of non-zero components in $\hat{\beta}$. Notice how as λ increases, this number falls. The penalty ensures that explanatory variables with small coefficients are dropped as λ gets bigger. On the **right**, a plot of the coefficient values as a function of $\log \lambda$ for the same regression. The vertical line shows the value of $\log \lambda$ that produces the model with smallest cross-validated error. Only 38 of 116 explanatory variables are used by this model

is sometimes known as **elastic net**. It can be shown that regressions penalized with elastic net tend to produce models with many zero coefficients, while not omitting correlated explanatory variables. All the computation can be done by the `glmnet` package in R (see exercises for details).

11.4.2 Wide Datasets

Now imagine we have more independent variables than examples (this is sometimes referred to as a "wide" dataset). This occurs quite often for a wide range of datasets; it's particularly common for biological datasets and natural language datasets. Unregularized linear regression must fail, because $\mathcal{X}^T\mathcal{X}$ must be rank deficient. Using an L2 (ridge) regularizer will produce an answer that should seem untrustworthy. The estimate of β is constrained by the data in some directions, but in other directions it is constrained only by the regularizer.

An estimate produced by L1 (lasso) regularization should look more reliable to you. Zeros in the estimate of β mean that the corresponding independent variables

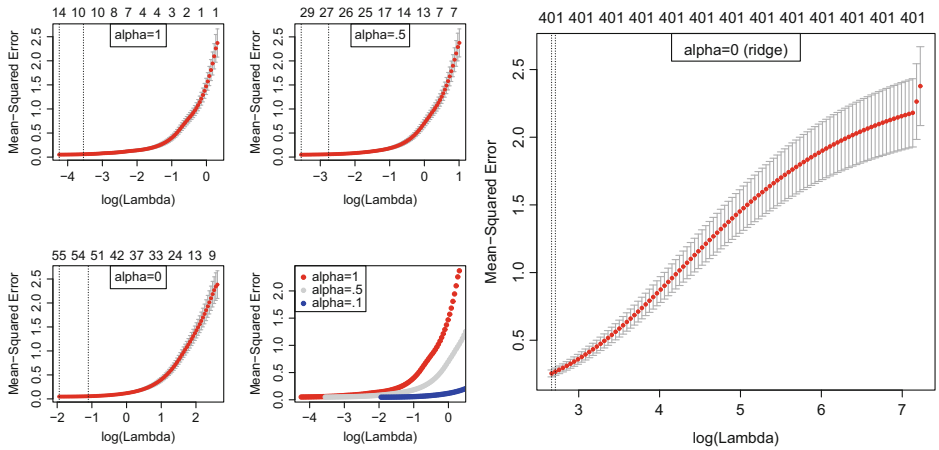

Figure 11.7: On the **left**, a comparison between three values of α in a `glmnet` regression predicting octane from NIR spectra (see Example 11.4). The plots show cross-validated error against log regularization coefficient for $\alpha = 1$ (lasso) and two elastic net cases, $\alpha = 0.5$ and $\alpha = 0.1$. I have plotted these curves separately, with error bars, and on top of each other but without error bars. The values on top of each separate plot show the number of independent variables with non-zero coefficients in the best model with that regularization parameter. On the **right**, a ridge regression for comparison. Notice that the error is considerably larger, even at the best value of the regularization parameter

are ignored. Now if there are many zeros in the estimate of β, the model is being fit with a small subset of the independent variables. If this subset is small enough, then the number of independent variables that are actually being used is smaller than the number of examples. If the model gives low enough error, it should seem trustworthy in this case. There are some hard questions to face here (e.g., does the model choose the "right" set of variables?) that we can't deal with.

Remember This: *The lasso can produce impressively small models, and handles wide datasets very well.*

Worked Example 11.4 *L1 Regularized Regression for a "Wide" Dataset*

The gasoline dataset has 60 examples of near infrared spectra for gasoline of different octane ratings. The dataset is due to John H. Kalivas, and was originally described in the article "Two Data Sets of Near Infrared Spectra," in the journal *Chemometrics and Intelligent Laboratory Systems*, vol. 37, pp. 255–259, 1997. Each example has measurements at 401 wavelengths. I found this dataset in the R library pls. Fit a regression of octane against infrared spectrum using L1 regularized logistic regression.

Solution: I used the `glmnet` package, and I benefited a lot from example code by Trevor Hastie and Junyang Qian and published at https://web.stanford.edu/~hastie/glmnet/glmnet_alpha.html. The package will do ridge, lasso, and elastic net regressions. One adjusts a parameter in the function call, α, that balances the terms; $\alpha = 0$ is ridge and $\alpha = 1$ is lasso. Not surprisingly, the ridge isn't great. I tried $\alpha = 0.1$, $\alpha = 0.5$, and $\alpha = 1$. Results in Fig. 11.7 suggest fairly strongly that very good predictions should be available with the lasso using quite a small regularization constant; there's no reason to believe that the best ridge models are better than the best elastic net models, or vice versa. The models are very sparse (look at the number of variables with non-zero weights, plotted on the top).

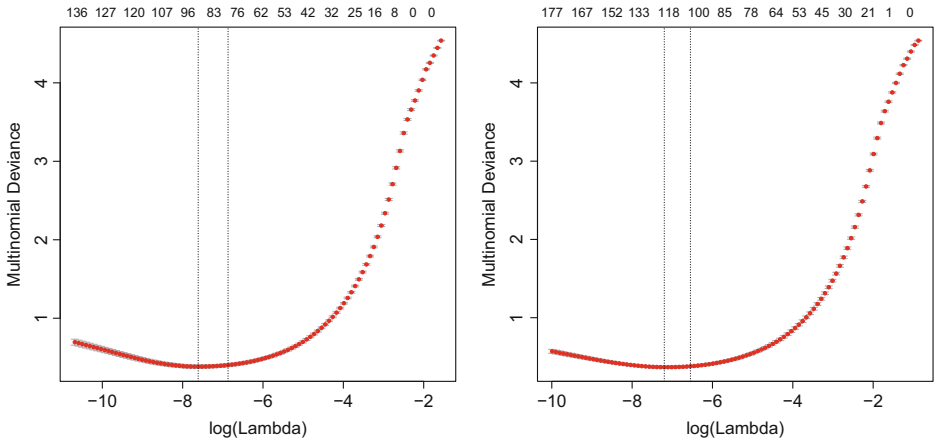

Figure 11.8: Multiclass logistic regression on the MNIST dataset, using a lasso and elastic net regularizers. On the **left**, deviance of held-out data on the digit dataset (Worked Example 11.5), for different values of the log regularization parameter in the lasso case. On the **right**, deviance of held-out data on the digit dataset (Worked Example 11.5), for different values of the log regularization parameter in the elastic net case, $\alpha = 0.5$

11.4.3 Using Sparsity Penalties with Other Models

A really nice feature of using an L1 penalty to enforce sparsity in a model is that it applies to a very wide range of models. For example, we can obtain a sparse SVM by replacing the L2 regularizer with an L1 regularizer. Most SVM packages will do this for you, although I'm not aware of any compelling evidence that this produces an improvement in most cases. All of the generalized linear models I described can be regularized with an L1 regularizer. For these cases, `glmnet` will do the computation required. The worked example shows using a multinomial (i.e., multiclass) logistic regression with an L1 regularizer.

Worked Example 11.5 *Multiclass Logistic Regression with an L1 Regularizer*

The MNIST dataset consists of a collection of handwritten digits, which must be classified into 10 classes $(0, \ldots, 9)$. There is a standard train/test split. This dataset is often called the zip code dataset because the digits come from zip codes, and has been quite widely studied. Yann LeCun keeps a record of the performance of different methods on this dataset at http://yann.lecun.com/exdb/mnist/. Obtain the Zip code dataset from http://statweb.stanford.edu/~tibs/ElemStatLearn/, and use a multiclass logistic regression with an L1 regularizer to classify it.

Solution: The dataset is rather large, and on my computer the fitting process takes a little time. Figure 11.8 shows what happens with the lasso, and with elastic net with $\alpha = 0.5$ on the training set, using `glmnet` to predict and cross-validation to select λ values. For the lasso, I found an error rate on the held-out data of 8.5%, which is OK, but not great compared to other methods. For elastic net, I found a slightly better error rate (8.2%); I believe even lower error rates are possible with these codes.

11.5 You Should

11.5.1 Remember These Terms

11.5.2 Remember These Facts

11.5.3 Remember These Procedures

Problems

Programming Exercises

11.1. *This is an extension of the previous exercise.* At https://archive.ics.uci.edu/ml/machine-learning-databases/housing/housing.data, you will find the famous Boston Housing dataset. This consists of 506 data items. Each is 13 measurements, and a house price. The data was collected by Harrison, D. and Rubinfeld, D.L in the 1970s (a date which explains the very low house prices). The dataset has been widely used in regression exercises, but seems to be waning in popularity. At least one of the independent variables measures the fraction of population nearby that is "Black" (their word, not mine). This variable appears to have had a significant effect on house prices then (and, sadly, may still now). **Hint:** *you really shouldn't write your own code; I used* `rlm` *and* `boxcox` *in R for this.*

- **(a)** Use Huber's robust loss and iteratively reweighted least squares to regress the house price against all other variables. How well does this regression compare to the regression obtained by removing outliers and Box-Coxing, above?
- **(b)** As you should have noticed, the Box-Cox transformation can be quite strongly affected by outliers. Remove up to six outliers from this dataset using a diagnostic plot, then estimate the Box-Cox transformation. Now transform the dependent variable, and use Huber's robust loss and iteratively reweighted least squares to regress the transformed variable against all others *using all data* (i.e., put the outliers you removed to compute a Box-Cox transformation back into the regression). How does this regression compare to the regression in the previous subexercise, and to the regression obtained by removing outliers and Box-Coxing, above?

11.2. UC Irvine hosts a dataset of blog posts at https://archive.ics.uci.edu/ml/datasets/BlogFeedback. There are 280 independent features which measure various properties of the blog post. The dependent variable is the number of comments that the blog post received in the 24 h after a base time. The zip file that you download will have training data in `blogData_train.csv`, and test data in a variety of files named `blogData_test-*.csv`.

- **(a)** Predict the dependent variable using all features, a generalized linear model (I'd use a Poisson model, because these are count variables), and the lasso. For this exercise, you really should use `glmnet` in R. Produce a plot of the cross-validated deviance of the model against the regularization variable (`cv.glmnet` and `plot` will do this for you). Use only the data in `blogData_train.csv`.
- **(b)** Your cross-validated plot of deviance likely doesn't mean all that much to you, because the deviance of a Poisson model takes a bit of getting used to. Choose a value of the regularization constant that yields a strong model, at least by the deviance criterion. Now produce a scatterplot of true values vs predicted values for data in `blogData_train.csv`. How well does this regression work? keep in mind that you are looking at predictions on the training set.
- **(c)** Choose a value of the regularization constant that yields a strong model, at least by the deviance criterion. Now produce a scatterplot of true values vs predicted values for data in `blogData_test-*.csv`. How well does this regression work?
- **(d)** Why is this regression difficult?

11.3. At http://genomics-pubs.princeton.edu/oncology/affydata/index.html, you will find a dataset giving the expression of 2000 genes in tumor and normal colon tissues. Build a logistic regression of the label (normal vs tumor) against the expression levels for those genes. There are a total of 62 tissue samples, so this is a wide regression. For this exercise, you really should use `glmnet` in R. Produce a plot of the classification error of the model against the regularization variable (`cv.glmnet`—look at the `type.measure` argument—and `plot` will do this for you). Compare the prediction of this model with the baseline of predicting the most common class.

11.4. The Jackson lab publishes numerous datasets to do with genetics and phenotypes of mice. At https://phenome.jax.org/projects/Crusio1, you can find a dataset giving the strain of a mouse, its gender, and various observations (click on the "Downloads" button). These observations are of body properties like mass, behavior, and various properties of the mouse's brain.

(a) We will predict the gender of a mouse from the body properties and the behavior. The variables you want are columns 4 through 41 of the dataset (or `bw` to `visit_time_d3_d5`; you shouldn't use the `id` of the mouse). Read the description; I've omitted the later behavioral measurements because there are many N/A's. Drop rows with N/A's (there are relatively few). How accurately can you predict gender using these measurements, using a logistic regression and the lasso? For this exercise, you really should use `glmnet` in R. Produce a plot of the classification error of the model against the regularization variable (`cv.glmnet`—look at the `type.measure` argument—and `plot` will do this for you). Compare the prediction of this model with the baseline of predicting the most common gender for all mice.

(b) We will predict the strain of a mouse from the body properties and the behavior. The variables you want are columns 4 through 41 of the dataset (or `bw` to `visit_time_d3_d5`; you shouldn't use the `id` of the mouse). Read the description; I've omitted the later behavioral measurements because there are many N/A's. Drop rows with N/A's (there are relatively few). This exercise is considerably more elaborate than the previous, because multinomial logistic regression does not like classes with few examples. You should drop strains with fewer than 10 rows. How accurately can you predict strain using these measurements, using multinomial logistic regression and the lasso? For this exercise, you really should use `glmnet` in R. Produce a plot of the classification error of the model against the regularization variable (`cv.glmnet`—look at the `type.measure` argument—and `plot` will do this for you). Compare the prediction of this model with the baseline of predicting a strain at random.

This data was described in a set of papers produced by this laboratory, and they like users to cite the papers. Papers are

- Delprato A, Bonheur B, Algéo MP, Rosay P, Lu L, Williams RW, Crusio WE. Systems genetic analysis of hippocampal neuroanatomy and spatial learning in mice. Genes Brain Behav. 2015 Nov;14(8):591–606.
- Delprato A, Algéo MP, Bonheur B, Bubier JA, Lu L, Williams RW, Chesler EJ, Crusio WE. QTL and systems genetics analysis of mouse grooming and behavioral responses to novelty in an open field. Genes Brain Behav. 2017 Nov;16(8):790–799.
- Delprato A, Bonheur B, Algéo MP, Murillo A, Dhawan E, Lu L, Williams RW, Crusio WE. A QTL on chromosome 1 modulates inter-male aggression in mice. Genes Brain Behav. 2018 Feb 19.