16

Sparse Modeling and the Lasso

The amount of data we are faced with keeps growing. From around the late 1990s we started to see *wide* data sets, where the number of variables far exceeds the number of observations. This was largely due to our increasing ability to measure a large amount of information automatically. In genomics, for example, we can use a high-throughput experiment to automatically measure the expression of tens of thousands of genes in a sample in a short amount of time. Similarly, sequencing equipment allows us to genotype millions of SNPs (single-nucleotide polymorphisms) cheaply and quickly. In document retrieval and modeling, we represent a document by the presence or count of each word in the dictionary. This easily leads to a feature vector with 20,000 components, one for each distinct vocabulary word, although most would be zero for a small document. If we move to bi-grams or higher, the feature space gets really large.

In even more modest situations, we can be faced with hundreds of variables. If these variables are to be predictors in a regression or logistic regression model, we probably do not want to use them all. It is likely that a subset will do the job well, and including all the redundant variables will degrade our fit. Hence we are often interested in identifying a good subset of variables. Note also that in these wide-data situations, even linear models are over-parametrized, so some form of reduction or regularization is essential.

In this chapter we will discuss some of the popular methods for model selection, starting with the time-tested and worthy forward-stepwise approach. We then look at the lasso, a popular modern method that does selection and shrinkage via convex optimization. The LARs algorithm ties these two approaches together, and leads to methods that can deliver paths of solutions.

Finally, we discuss some connections with other modern big- and wide-data approaches, and mention some extensions.

16.1 Forward Stepwise Regression

Stepwise procedures have been around for a very long time. They were originally devised in times when data sets were quite modest in size, in particular in terms of the number of variables. Originally thought of as the poor cousins of "best-subset" selection, they had the advantage of being much cheaper to compute (and in fact *possible* to compute for large p). We will review best-subset regression first.

Suppose we have a set of n observations on a response y_i and a vector of p predictors $x_i' = (x_{i1}, x_{i2}, \ldots, x_{ip})$, and we plan to fit a linear regression model. The response could be quantitative, so we can think of fitting a linear model by least squares. It could also be binary, leading to a linear logistic regression model fit by maximum likelihood. Although we will focus on these two cases, the same ideas transfer exactly to other generalized linear models, the Cox model, and so on. The idea is to build a model using a subset of the variables; in fact the smallest subset that adequately explains the variation in the response is what we are after, both for inference and for prediction purposes. Suppose our loss function for fitting the linear model is L (e.g. sum of squares, negative log-likelihood). The method of best-subset regression is simple to describe, and is given in Algorithm 16.1. Step 3 is easy to state, but requires a lot of computation. For

Algorithm 16.1 BEST-SUBSET REGRESSION.

1. Start with $m = 0$ and the null model $\hat{\eta}_0(x) = \hat{\beta}_0$, estimated by the mean of the y_i.
2. At step $m = 1$, pick the single variable j that fits the response best, in terms of the loss L evaluated on the training data, in a univariate regression $\hat{\eta}_1(x) = \hat{\beta}_0 + x_j'\hat{\beta}_j$. Set $\mathcal{A}_1 = \{j\}$.
3. For each subset size $m \in \{2, 3, \ldots, M\}$ (with $M \leq \min(n-1, p)$) identify the best subset $\mathcal{A}_m$ of size m when fitting a linear model $\hat{\eta}_m(x) = \hat{\beta}_0 + x_{\mathcal{A}_m}'\hat{\beta}_{\mathcal{A}_m}$ with m of the p variables, in terms of the loss L.
4. Use some external data or other means to select the "best" amongst these M models.

p much larger than about 40 it becomes prohibitively expensive to perform exactly—a so-called "N-P complete" problem because of its combinatorial complexity (there are 2^p subsets). Note that the subsets need not be nested:

the best subset of size $m = 3$, say, need not include both or any of the variables in the best subset of size $m = 2$.

In step 4 there are a number of methods for selecting m. Originally the C_p criterion of Chapter 12 was proposed for this purpose. Here we will favor K-fold cross-validation, since it is applicable to all the methods discussed in this chapter.

It is interesting to digress for a moment on how cross-validation works here. We are using it to select the subset size m on the basis of prediction performance (on future data). With $K = 10$, we divide the n training observations randomly into 10 equal size groups. Leaving out say group $k = 1$, we perform steps 1–3 on the 9/10ths, and for each of the chosen models, we summarize the prediction performance on the group-1 data. We do this $K = 10$ times, each time with group k left out. We then average the 10 performance measures for each m, and select the value of m corresponding to the best performance. Notice that for each m, the 10 models $\hat{\eta}_m(x)$ might involve different subsets of variables! This is not a concern, since we are trying to find a good value of m for the method. Having identified $\hat{m}$, we rerun steps 1–3 on the entire training set, and deliver the chosen model $\hat{\eta}_{\hat{m}}(x)$.

As hinted above, there are problems with best-subset regression. A primary issue is that it works exactly only for relatively small p. For example, we cannot run it on the **spam** data with 57 variables (at least not in 2015 on a Macbook Pro!). We may also think that even if we could do the computations, with such a large search space the variance of the procedure might be too high.

As a result, more manageable stepwise procedures were invented. Forward stepwise regression, Algorithm 16.2, is a simple modification of best-subset, with the modification occurring in step 3. Forward stepwise regression produces a *nested* sequence of models $\emptyset \ldots \subset \mathcal{A}_{m-1} \subset \mathcal{A}_m \subset \mathcal{A}_{m+1} \ldots$. It starts with the null model, here an intercept, and adds variables one at a time. Even with large p, identifying the best variable to add at each step is manageable, and can be distributed if clusters of machines are available. Most importantly, it is feasible for large p. Figure 16.1 shows the coefficient profiles for forward-stepwise linear regression on the **spam** training data. Here there are 57 input variables (relative prevalence of particular words in the document), and an "official" (train, test) split of (3065, 1536) observations. The response is coded as +1 if the email was spam, else -1. The figure caption gives the details. We saw the **spam** data earlier, in Table 8.3, Figure 8.7 and Figure 12.2.

Fitting the entire forward-stepwise linear regression path as in the figure

Algorithm 16.2 FORWARD STEPWISE REGRESSION.

1 Start with $m = 0$ and the null model $\hat{\eta}_0(x) = \hat{\beta}_0$, estimated by the mean of the y_i.
2 At step $m = 1$, pick the single variable j that fits the response best, in terms of the loss L evaluated on the training data, in a univariate regression $\hat{\eta}_1(x) = \hat{\beta}_0 + x_j'\hat{\beta}_j$. Set $\mathcal{A}_1 = \{j\}$.
3 For each subset size $m \in \{2, 3, \ldots, M\}$ (with $M \leq \min(n-1, p)$) identify the variable k that when augmented with $\mathcal{A}_{m-1}$ to form $\mathcal{A}_m$, leads to the model $\hat{\eta}_m(x) = \hat{\beta}_0 + x'_{\mathcal{A}_m}\hat{\beta}_{\mathcal{A}_m}$ that performs best in terms of the loss L.
4 Use some external data or other means to select the "best" amongst these M models.

(when $n > p$) has essentially the same cost as a single least squares fit on all the variables. This is because the sequence of models can be updated each time a variable is added.† However, this is a consequence of the linear model and squared-error loss. †1

Suppose instead we run a forward stepwise logistic regression. Here updating does not work, and the entire fit has to be recomputed by maximum likelihood each time a variable is added. Identifying which variable to add in step 3 in principle requires fitting an $(m+1)$-variable model $p - m$ times, and seeing which one reduces the deviance the most. In practice, we can use score tests which are much cheaper to evaluate.† These amount to using the quadratic approximation to the log-likelihood from the final iteratively reweighted least-squares (IRLS) iteration for fitting the model with m terms. The score test for a variable not in the model is equivalent to testing for the inclusion of this variable in the weighted least-squares fit. Hence identifying the next variable is almost back to the previous cases, requiring $p - m$ simple regression updates.† Figure 16.2 shows the test misclassification error for forward-stepwise linear regression and logistic regression on the **spam** data, as a function of the number of steps. They both level off at around 25 steps, and have a similar shape. However, the logistic regression gives more accurate classifications.[1] †2 †3

Although forward-stepwise methods are possible for large p, they get tedious for very large p (in the thousands), especially if the data could sup-

[1] For this example we can halve the gap between the curves by optimizing the prediction threshold for linear regression.

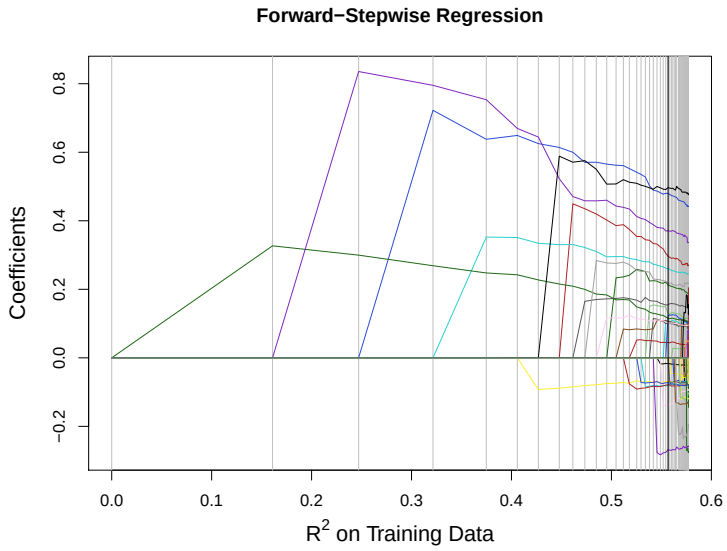

Figure 16.1 Forward stepwise linear regression on the `spam` data. Each curve corresponds to a particular variable, and shows the progression of its coefficient as the model grows. These are plotted against the training R^2, and the vertical gray bars correspond to each step. Starting at the left at step 1, the first selected variable explains $R^2 = 0.16$; adding the second increases R^2 to 0.25, etc. What we see is that early steps have a big impact on the R^2, while later steps hardly have any at all. The vertical black line corresponds to step 25 (see Figure 16.2), and we see that after that the step-wise improvements in R^2 are negligible.

port a model with many variables. However, if the ideal active set is fairly small, even with many thousands of variables forward-stepwise selection is a viable option.

Forward-stepwise selection delivers a sequence of models, as seen in the previous figures. One would generally want to select a single model, and as discussed earlier, we often use cross-validation for this purpose. Figure 16.3 illustrates using stepwise linear regression on the `spam` data. Here the sequence of models are fit using squared-error loss on the binary response variable. However, cross-validation scores each model for *misclassification error*, the ultimate goal of this modeling exercise. This highlights one of the advantages of cross-validation in this context. A con-

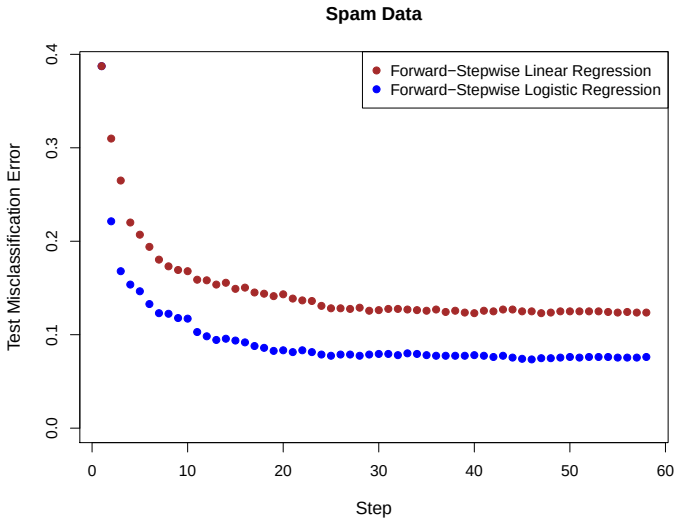

Figure 16.2 Forward-stepwise regression on the `spam` data. Shown is the misclassification error on the test data, as a function of the number of steps. The brown dots correspond to linear regression, with the response coded as -1 and +1; a prediction greater than zero is classified as +1, one less than zero as -1. The blue dots correspond to logistic regression, which performs better. We see that both curves essentially reach their minima after 25 steps.

venient (differentiable and smooth) loss function is used to fit the sequence of models. However, we can use any performance measure to evaluate the sequence of models; here misclassification error is used. In terms of the parameters of the linear model, misclassification error would be a difficult and discontinuous loss function to use for parameter estimation. All we need to use it for here is pick the best model size. There appears to be little benefit in going beyond 25–30 terms.

16.2 The Lasso

The stepwise model-selection methods of the previous section are useful if we anticipate a model using a relatively small number of variables, even if the pool of available variables is very large. If we expect a moderate number of variables to play a role, these methods become cumbersome.

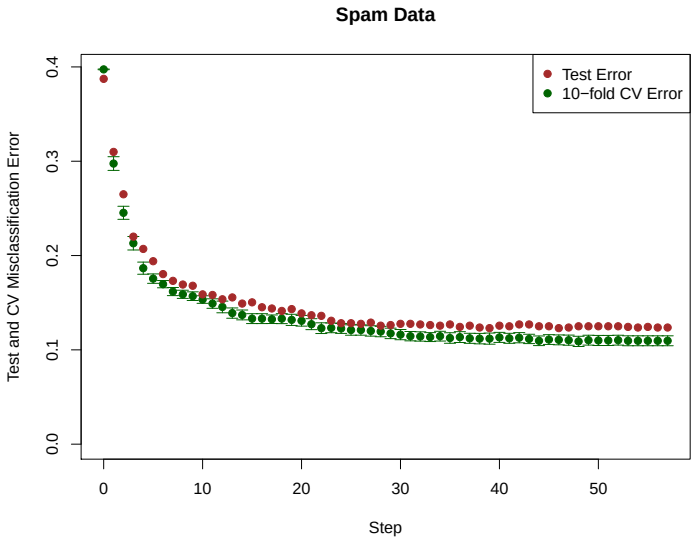

Figure 16.3 Ten-fold cross-validated misclassification errors (green) for forward-stepwise regression on the `spam` data, as a function of the step number. Since each error is an average of 10 numbers, we can compute a (crude) standard error; included in the plot are pointwise standard-error bands. The brown curve is the misclassification error on the test data.

Another black mark against forward-stepwise methods is that the sequence of models is derived in a *greedy* fashion, without any claimed optimality. The methods we describe here are derived from a more principled procedure; indeed they solve a *convex optimization*, as defined below.

We will first present the lasso for squared-error loss, and then the more general case later. Consider the constrained linear regression problem

$$\underset{\beta_0\in\mathbb{R},\,\beta\in\mathbb{R}^p}{\text{minimize}}\ \frac{1}{n}\sum_{i=1}^{n}(y_i-\beta_0-x_i'\beta)^2 \text{ subject to } \|\beta\|_1\le t, \tag{16.1}$$

where $\|\beta\|_1=\sum_{j=1}^{p}|\beta_j|$, the ℓ_1 norm of the coefficient vector. Since both the loss and the constraint are convex in β, this is a convex optimization problem, and it is known as the *lasso*. The constraint $\|\beta\|_1\le t$ restricts the coefficients of the model by pulling them toward zero; this has the effect of reducing their variance, and prevents overfitting. Ridge regression is an earlier great uncle of the lasso, and solves a similar problem to (16.1), ex-

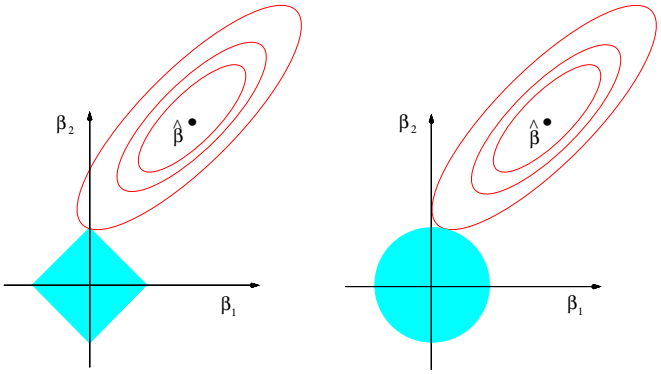

Figure 16.4 An example with $\beta \in \mathbb{R}^2$ to illustrate the difference between ridge regression and the lasso. In both plots, the red contours correspond to the squared-error loss function, with the unrestricted least-squares estimate $\hat{\beta}$ in the center. The blue regions show the constraints, with the lasso on the left and ridge on the right. The solution to the constrained problem corresponds to the value of β where the expanding loss contours first touch the constraint region. Due to the shape of the lasso constraint, this will often be at a corner (or an edge more generally), as here, which means in this case that the minimizing β has $\beta_1 = 0$. For the ridge constraint, this is unlikely to happen.

cept the constraint is $\|\beta\|_2 \leq t$; ridge regression bounds the quadratic ℓ_2 norm of the coefficient vector. It also has the effect of pulling the coefficients toward zero, in an apparently very similar way. Ridge regression is discussed in Section 7.3.[2] Both the lasso and ridge regression are shrinkage methods, in the spirit of the James–Stein estimator of Chapter 7.

A big difference, however, is that for the lasso, the solution typically has many of the β_j equal to zero, while for ridge they are all nonzero. Hence the lasso does variable selection and shrinkage, while ridge only shrinks. Figure 16.4 illustrates this for $\beta \in \mathbb{R}^2$. In higher dimensions, the ℓ_1 norm has sharp edges and corners, which correspond to coefficient estimates zero in β.

Since the constraint in the lasso treats all the coefficients equally, it usually makes sense for all the elements of x to be in the same units. If not, we

[2] Here we use the "bound" form of ridge regression, while in Section 7.3 we use the "Lagrange" form. They are equivalent, in that for every "Lagrange" solution, there is a corresponding bound solution.

typically standardize the predictors beforehand so that each has variance one.

Two natural *boundary* values for t in (16.1) are $t = 0$ and $t = \infty$. The former corresponds to the constant model (the fit is the mean of the y_i,)[3] and the latter corresponds to the unrestricted least-squares fit. In fact, if $n > p$, and $\hat{\beta}$ is the least-squares estimate, then we can replace ∞ by
†4 $\|\hat{\beta}\|_1$, and any value of $t \geq \|\hat{\beta}\|_1$ is a non-binding constraint.† Figure 16.5

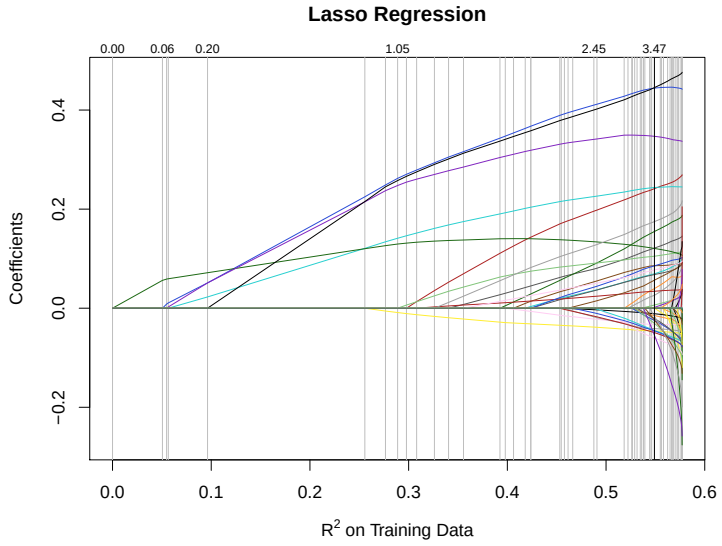

Figure 16.5 The lasso linear regression regularization path on the `spam` data. Each curve corresponds to a particular variable, and shows the progression of its coefficient as the regularization bound t grows. These curves are plotted against the training R^2 rather than t, to make the curves comparable with the forward-stepwise curves in Figure 16.1. Some values of t are indicated at the top. The vertical gray bars indicate changes in the active set of nonzero coefficients, typically an inclusion. Here we see clearly the role of the ℓ_1 penalty; as t is relaxed, coefficients become nonzero, but in a smoother fashion than in forward stepwise.

shows the regularization path[4] for the lasso linear regression problem on

[3] We typically do not restrict the intercept in the model.

[4] Also known as the *homotopy* path.

the **spam** data; that is, the solution path for all values of t. This can be computed exactly, as we will see in Section 16.4, because the coefficient profiles are piecewise linear in t. It is natural to compare this coefficient profile with the analogous one in Figure 16.1 for forward-stepwise regression. Because of the control of $\|\hat{\beta}(t)\|_1$, we don't see the same range as in forward stepwise, and observe somewhat smoother behavior. Figure 16.6 contrasts

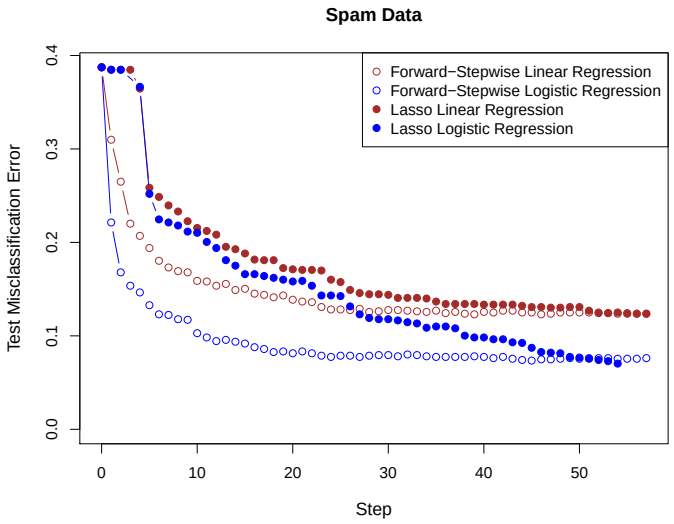

Figure 16.6 Lasso versus forward-stepwise regression on the **spam** data. Shown is the misclassification error on the test data, as a function of the number of variables in the model. Linear regression is coded brown, logistic regression blue; hollow dots forward stepwise, solid dots lasso. In this case it appears stepwise and lasso achieve the same performance, but lasso takes longer to get there, because of the shrinkage.

the prediction performance on the **spam** data for lasso regularized models (linear regression and logistic regression) versus forward-stepwise models. The results are rather similar at the end of the path; here forward stepwise can achieve classification performance similar to that of lasso regularized logistic regression with about half the terms. Lasso logistic regression (and indeed any likelihood-based linear model) is fit by penalized maximum

likelihood:

$$\underset{\beta_0 \in \mathbb{R},\, \beta \in \mathbb{R}^p}{\text{minimize}} \frac{1}{n} \sum_{i=1}^{n} L(y_i, \beta_0 + \beta' x_i) \text{ subject to } \|\beta\|_1 \leq t. \tag{16.2}$$

Here L is the negative of the log-likelihood function for the response distribution.

16.3 Fitting Lasso Models

The lasso objectives (16.1) or (16.2) are differentiable and convex in β and β_0, and the constraint is convex in β. Hence solving these problems is a convex optimization problem, for which standard packages are available. It turns out these problems have special structure that can be exploited to yield efficient algorithms for fitting the entire path of solutions as in Figures 16.1 and 16.5. We will start with problem (16.1), which we rewrite in the more convenient *Lagrange* form:

$$\underset{\beta \in \mathbb{R}^p}{\text{minimize}} \frac{1}{2n} \|\boldsymbol{y} - \boldsymbol{X}\beta\|^2 + \lambda \|\beta\|_1. \tag{16.3}$$

Here we have centered $\boldsymbol{y}$ and the columns of $\boldsymbol{X}$ beforehand, and hence the intercept has been omitted. The Lagrange and constraint versions are equivalent, in the sense that any solution $\hat{\beta}(\lambda)$ to (16.3) with $\lambda \geq 0$ corresponds to a solution to (16.1) with $t = \|\hat{\beta}(\lambda)\|_1$. Here large values of λ will encourage solutions with small ℓ_1 norm coefficient vectors, and vice-versa; $\lambda = 0$ corresponds to the ordinary least squares fit.

The solution to (16.3) satisfies the subgradient condition

$$-\frac{1}{n} \langle \boldsymbol{x}_j, \boldsymbol{y} - \boldsymbol{X}\hat{\beta} \rangle + \lambda s_j = 0, \quad j = 1, \ldots, p, \tag{16.4}$$

where $s_j \in \text{sign}(\hat{\beta}_j)$, $j = 1, \ldots, p$. This notation means $s_j = \text{sign}(\hat{\beta}_j)$ if $\hat{\beta}_j \neq 0$, and $s_j \in [-1, 1]$ if $\hat{\beta}_j = 0$.) We use the inner-product notation $\langle a, b \rangle = a'b$ in (16.4), which leads to more evocative expressions. These subgradient conditions are the modern way of characterizing solutions to problems of this kind, and are equivalent to the Karush–Kuhn–Tucker optimality conditions. From these conditions we can immediately learn some properties of a lasso solution.

- $\frac{1}{n} |\langle \boldsymbol{x}_j, \boldsymbol{y} - \boldsymbol{X}\hat{\beta} \rangle| = \lambda$ for all members of the *active set*; i.e., each of the variables in the model (with nonzero coefficient) has the same covariance with the residuals (in absolute value).

- $\frac{1}{n}|\langle \boldsymbol{x}_k, \boldsymbol{y} - \boldsymbol{X}\hat{\beta}\rangle| \leq \lambda$ for all variables not in the active set (i.e. with coefficients zero).

These conditions are interesting and have a big impact on computation. Suppose we have the solution $\hat{\beta}(\lambda_1)$ at λ_1, and we decrease λ by a small amount to $\lambda_2 < \lambda_1$. The coefficients and hence the residuals change, in such a way that the covariances all remain tied at the smaller value λ_2. If in the process the active set has not changed, and nor have the signs of their coefficients, then we get an important consequence: $\hat{\beta}(\lambda)$ is *linear* for $\lambda \in [\lambda_2, \lambda_1]$. To see this, suppose $\mathcal{A}$ indexes the active set, which is the same at λ_1 and λ_2, and let $s_{\mathcal{A}}$ be the constant sign vector. Then we have

$$\begin{aligned} \boldsymbol{X}'_{\mathcal{A}}(\boldsymbol{y} - \boldsymbol{X}\hat{\beta}(\lambda_1)) &= n s_{\mathcal{A}} \lambda_1, \\ \boldsymbol{X}'_{\mathcal{A}}(\boldsymbol{y} - \boldsymbol{X}\hat{\beta}(\lambda_2)) &= n s_{\mathcal{A}} \lambda_2. \end{aligned}$$

By subtracting and solving we get

$$\hat{\beta}_{\mathcal{A}}(\lambda_2) - \hat{\beta}_{\mathcal{A}}(\lambda_1) = n(\lambda_1 - \lambda_2)(\boldsymbol{X}'_{\mathcal{A}}\boldsymbol{X}_{\mathcal{A}})^{-1} s_{\mathcal{A}}, \tag{16.5}$$

and the remaining coefficients (with indices not in $\mathcal{A}$) are all zero. This shows that the full coefficient vector $\hat{\beta}(\lambda)$ is linear for $\lambda \in [\lambda_2, \lambda_1]$. In fact, the coefficient profiles for the lasso are continuous and piecewise linear over the entire range of λ, with *knots* occurring whenever the active set changes, or the signs of the coefficients change.

Another consequence is that we can easily determine λ_{max}, the smallest value for λ such that the solution $\hat{\beta}(\lambda_{max}) = 0$. From (16.4) this can be seen to be $\lambda_{max} = \max_j \frac{1}{n}|\langle \boldsymbol{x}_j, \boldsymbol{y}\rangle|$.

These two facts plus a few more details enable us to compute the exact solution path for the squared-error-loss lasso; that is the topic of the next section.

16.4 Least-Angle Regression

We have just seen that the lasso coefficient profile $\hat{\beta}(\lambda)$ is piecewise linear in λ, and that the elements of the active set are tied in their absolute covariance with the residuals. With $\boldsymbol{r}(\lambda) = \boldsymbol{y} - \boldsymbol{X}\hat{\beta}(\lambda)$, the covariance between $\boldsymbol{x}_j$ and the evolving residual is $c_j(\lambda) = \frac{1}{n}|\langle x_j, \boldsymbol{r}(\lambda)\rangle|$. Hence these also change in a piecewise linear fashion, with $c_j(\lambda) = \lambda$ for $j \in \mathcal{A}$, and $c_j(\lambda) \leq \lambda$ for $j \notin \mathcal{A}$. This inspires the *Least-Angle Regression* algorithm, given in Algorithm 16.3, which exploits this linearity to fit the entire lasso regularization path.

Algorithm 16.3 LEAST-ANGLE REGRESSION.

1 Standardize the predictors to have mean zero and unit ℓ_2 norm. Start with the residual $\boldsymbol{r}_0 = \boldsymbol{y} - \bar{\boldsymbol{y}}$, $\beta^0 = (\beta_1, \beta_2, \ldots, \beta_p) = 0$.

2 Find the predictor $\boldsymbol{x}_j$ most correlated with $\boldsymbol{r}_0$; i.e., with largest value for $\frac{1}{n}|\langle \boldsymbol{x}_j, \boldsymbol{r}_0\rangle|$. Call this value λ_0, define the active set $\mathcal{A} = \{j\}$, and $\boldsymbol{X}_\mathcal{A}$, the matrix consisting of this single variable.

3 For $k = 1, 2, \ldots, K = \min(n-1, p)$ do:

(a) Define the least-squares direction $\delta = \frac{1}{n\lambda_{k-1}}(\boldsymbol{X}'_\mathcal{A}\boldsymbol{X}_\mathcal{A})^{-1}\boldsymbol{X}'_\mathcal{A}\boldsymbol{r}_{k-1}$, and define the p-vector Δ such that $\Delta_\mathcal{A} = \delta$, and the remaining elements are zero.

(b) Move the coefficients β from β^{k-1} in the direction Δ toward their least-squares solution on $\boldsymbol{X}_\mathcal{A}$: $\beta(\lambda) = \beta^{k-1} + (\lambda_{k-1} - \lambda)\Delta$ for $0 < \lambda \le \lambda_{k-1}$, keeping track of the evolving residuals $\boldsymbol{r}(\lambda) = \boldsymbol{y} - \boldsymbol{X}\beta(\lambda) = \boldsymbol{r}_{k-1} - (\lambda_{k-1} - \lambda)\boldsymbol{X}_\mathcal{A}\delta$.

(c) Keeping track of $\frac{1}{n}|\langle \boldsymbol{x}_\ell, \boldsymbol{r}(\lambda)\rangle|$ for $\ell \notin \mathcal{A}$, identify the largest value of λ at which a variable "catches up" with the active set; if the variable has index ℓ, that means $\frac{1}{n}|\langle \boldsymbol{x}_\ell, \boldsymbol{r}(\lambda)\rangle| = \lambda$. This defines the next "knot" λ_k.

(d) Set $\mathcal{A} = \mathcal{A} \cup \ell$, $\beta^k = \beta(\lambda_k) = \beta^{k-1} + (\lambda_{k-1} - \lambda_k)\Delta$, and $\boldsymbol{r}_k = \boldsymbol{y} - \boldsymbol{X}\beta^k$.

4 Return the sequence $\{\lambda_k,\ \beta^k\}_0^K$.

In step 3(a) $\delta = (\boldsymbol{X}'_\mathcal{A}\boldsymbol{X}_\mathcal{A})^{-1}s_\mathcal{A}$ as in (16.5). We can think of the LAR algorithm as a democratic version of forward-stepwise regression. In forward-stepwise regression, we identify the variable that will improve the fit the most, and then move all the coefficients toward the new least-squares fit. As described in endnotes †1 and †3, this is sometimes done by computing the inner products of each (unadjusted) variable with the residual, and picking the largest in absolute value. In step 3 of Algorithm 16.3, we move the coefficients for the variables in the active set $\mathcal{A}$ *toward* their least-squares fit (keeping their inner products tied), but stop when a variable not in $\mathcal{A}$ catches up in inner product. At that point, it is invited into the club, and the process continues.

Step 3(c) can be performed efficiently because of the linearity of the evolving inner products; for each variable not in $\mathcal{A}$, we can determine exactly when (in λ time) it would catch up, and hence which catches up first and when. Since the path is piecewise linear, and we know the slopes, this

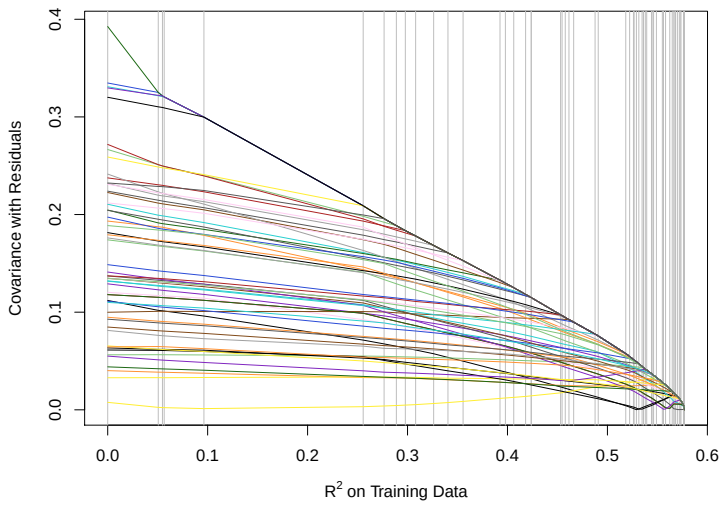

Figure 16.7 Covariance evolution on the `spam` data. As variables tie for maximal covariance, they become part of the active set. These occasions are indicated by the vertical gray bars, again plotted against the training R^2 as in Figure 16.5.

means we know the path *exactly* without further computation between λ_{k-1} and the newly found λ_k.

The name "least-angle regression" derives from the fact that in step 3(b) the fitted vector evolves in the direction $\boldsymbol{X}\Delta = \boldsymbol{X}_{\mathcal{A}}\delta$, and its inner product with each active vector is given by $\boldsymbol{X}'_{\mathcal{A}}\boldsymbol{X}_{\mathcal{A}}\delta = s_{\mathcal{A}}$. Since all the columns of $\boldsymbol{X}$ have unit norm, this means the angles between each active vector and the evolving fitted vector are equal and hence minimal.

The main computational burden in Algorithm 16.3 is in step 3(a), computing the new direction, each time the active set is updated. However, this is easily performed using standard updating of a QR decomposition, and hence the computations for the entire path are of the same order as that of a single least-squares fit using all the variables.

The vertical gray lines in Figure 16.5 show when the active set changes. We see the slopes change at each of these transitions. Compare with the corresponding Figure 16.1 for forward-stepwise regression.

Figure 16.7 shows the the decreasing covariance during the steps of the

LAR algorithm. As each variable joins the active set, the covariances become tied. At the end of the path, the covariances are all zero, because this is the unregularized ordinary least-squares solution.

It turns out that the LAR algorithm is not quite the lasso path; variables can *drop out* of the active set as the path evolves. This happens when a coefficient curve passes through zero. The subgradient equations (16.4) imply that the sign of each active coefficient matches the sign of the gradient. However, a simple addition to step 3(c) in Algorithm 16.3 takes care of the issue:

3(c)+ *lasso modification*: If a nonzero coefficient crosses zero before the next variable enters, drop it from $\mathcal{A}$ and recompute the joint least-squares direction Δ using the reduced set.

Figure 16.5 was computed using the `lars` package in `R`, with the `lasso` option set to accommodate step 3(c)+; in this instance there was no need for dropping. Dropping tends to occur when some of the variables are highly correlated.

Lasso and Degrees of Freedom

We see in Figure 16.6 (left panel) that forward-stepwise regression is more aggressive than the lasso, in that it brings down the training MSE faster. We can use the covariance formula for df from Chapter 12 to quantify the amount of fitting at each step.

In the right panel we show the results of a simulation for estimating the df of forward-stepwise regression and the lasso for the `spam` data. Recall the covariance formula

$$df = \frac{1}{\sigma^2}\sum_{i=1}^{n}\text{cov}(y_i, \hat{y}_i). \tag{16.6}$$

These covariances are of course with respect to the sampling distribution of the y_i, which we do not have access to since these are real data. So instead we simulate from fitted values from the full least-squares fit, by adding Gaussian errors with the appropriate (estimated) standard deviation. (This is the parametric bootstrap calculation (12.64).)

It turns out that each step of the LAR algorithm spends one df, as is evidenced by the brown curve in the right plot of Figure 16.8. Forward stepwise spends more df in the earlier stages, and can be erratic.

Under some technical conditions on the $\boldsymbol{X}$ matrix (that guarantee that

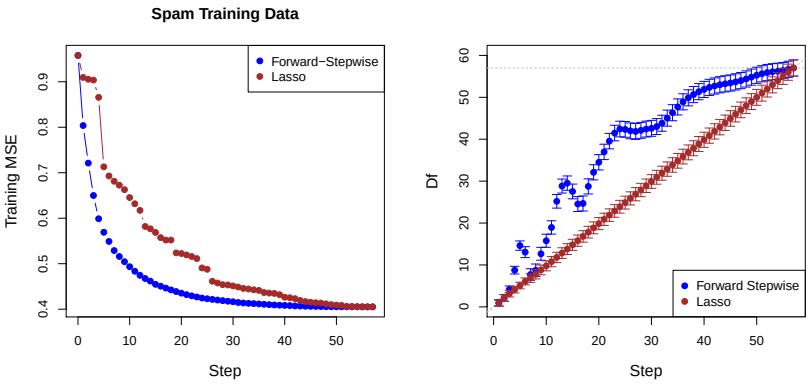

Figure 16.8 Left: Training mean-squared error (MSE) on the `spam` data, for forward-stepwise regression and the lasso, as a function of the size of the active set. Forward stepwise is more aggressive than the lasso, in that it (over-)fits the training data more quickly. Right: Simulation showing the degrees of freedom or df of forward-stepwise regression versus lasso. The lasso uses one df per step, while forward stepwise is greedier and uses more, especially in the early steps. Since these df were computed using 5000 random simulated data sets, we include standard-error bands on the estimates.

LAR delivers the lasso path), one can show that the df is *exactly* one per step. More generally, for the lasso, if we define $\widehat{df}(\lambda) = |\mathcal{A}(\lambda)|$ (the size of the active set at λ), we have that $E[\widehat{df}(\lambda)] = df(\lambda)$. In other words, the size of the active set is an unbiased estimate of df.

Ordinary least squares with a predetermined sequence of variables spends one df per variable. Intuitively forward stepwise spends more, because it pays a price (in some extra df) for searching.† Although the lasso does †5
search for the next variable, it does not fit the new model all the way, but just until the next variable enters. At this point, one new df has been spent.

16.5 Fitting Generalized Lasso Models

So far we have focused on the lasso for squared-error loss, and exploited the piecewise-linearity of its coefficient profile to efficiently compute the entire path. Unfortunately this is not the case for most other loss functions,

so obtaining the coefficient path is potentially more costly. As a case in point, we will use logistic regression as an example; in this case in (16.2) L represents the negative binomial log-likelihood. Writing the loss explicitly and using the Lagrange form for the penalty, we wish to solve

$$\underset{\beta_0\in\mathbb{R},\,\beta\in\mathbb{R}^p}{\text{minimize}} -\left[\frac{1}{n}\sum_{i=1}^{n} y_i \log \mu_i + (1-y_i)\log(1-\mu_i)\right] + \lambda\|\beta\|_1. \quad (16.7)$$

Here we assume the $y_i \in \{0,1\}$ and μ_i are the fitted probabilities

$$\mu_i = \frac{e^{\beta_0+x_i'\beta}}{1+e^{\beta_0+x_i'\beta}}. \quad (16.8)$$

Similar to (16.4), the solution satisfies the subgradient condition

$$\frac{1}{n}\langle \boldsymbol{x}_j, \boldsymbol{y}-\boldsymbol{\mu}\rangle - \lambda s_j = 0, \quad j = 1,\ldots,p, \quad (16.9)$$

where $s_j \in \text{sign}(\beta_j)$, $j = 1,\ldots,p$, and $\boldsymbol{\mu}' = (\mu_1,\ldots,\mu_n)$.[5] However, the nonlinearity of μ_i in β_j results in piecewise nonlinear coefficient profiles. Instead we settle for a solution path on a sufficiently fine grid of values for λ. It is once again easy to see that the largest value of λ we need consider is

$$\lambda_{max} = \max_j |\langle \boldsymbol{x}_j, \boldsymbol{y}-\bar{y}\mathbf{1}\rangle|, \quad (16.10)$$

since this is the smallest value of λ for which $\hat{\beta}=0$, and $\hat{\beta}_0 = \text{logit}(\bar{y})$. A reasonable sequence is 100 values $\lambda_1 > \lambda_2 > \ldots > \lambda_{100}$ equally spaced on the log-scale from λ_{max} down to $\epsilon\lambda_{max}$, where ϵ is some small fraction such as 0.001.

An approach that has proven to be surprisingly efficient is *path-wise coordinate descent*.

- For each value λ_k, solve the lasso problem for one β_j only, holding all the others fixed. Cycle around until the estimates stabilize.
- By starting at λ_1, where all the parameters are zero, we use *warm starts* in computing the solutions at the decreasing sequence of λ values. The warm starts provide excellent initializations for the sequence of solutions $\hat{\beta}(\lambda_k)$.
- The active set grows slowly as λ decreases. Computational hedges that guess the active set prove to be particularly efficient. If the guess is good (and correct), one iterates coordinate descent using only those variables,

[5] The equation for the intercept is $\frac{1}{n}\sum_{i=1}^n y_i = \frac{1}{n}\sum_{i=1}^n \mu_i$.

until convergence. One more sweep through all the variables confirms the hunch.

The **R** package **glmnet** employs a *proximal-Newton* strategy at each value λ_k.

1 Compute a weighted least squares (quadratic) approximation to the log-likelihood L at the current estimate for the solution vector $\hat{\beta}(\lambda_k)$; This produces a *working response* and observation weights, as in a regular GLM.
2 Solve the weighted least-squares lasso at λ_k by coordinate descent, using warm starts and active-set iterations.

We now give some details, which illustrate why these particular strategies are effective. Consider the weighted least-squares problem

$$\underset{\beta_j}{\text{minimize}} \frac{1}{2n} \sum_{i=1}^{n} w_i (z_i - \beta_0 - x_i' \beta)^2 + \lambda \|\beta\|_1, \tag{16.11}$$

with all but β_j fixed at their current values. Writing $r_i = z_i - \beta_0 - \sum_{\ell \neq j} x_{i\ell} \beta_\ell$, we can recast (16.11) as

$$\underset{\beta_j}{\text{minimize}} \frac{1}{2n} \sum_{i=1}^{n} w_i (r_i - x_{ij} \beta_j)^2 + \lambda |\beta_j|, \tag{16.12}$$

a one-dimensional problem. The subgradient equation is

$$\frac{1}{n} \sum_{i=1}^{n} w_i x_{ij} (r_i - x_{ij} \beta_j) - \lambda \cdot \text{sign}(\beta_j) = 0. \tag{16.13}$$

The simplest form of the solution occurs if each variable is standardized to have weighted mean zero and variance one, and the weights sum to one; in that case we have a two-step solution.

1 Compute the weighted simple least-squares coefficient

$$\tilde{\beta}_j = \langle \boldsymbol{x}_j, \boldsymbol{r} \rangle_w = \sum_{i=1}^{n} w_i x_{ij} r_i. \tag{16.14}$$

2 *Soft-threshold* $\tilde{\beta}_j$ to produce $\hat{\beta}_j$:

$$\hat{\beta}_j = \begin{cases} 0 & \text{if } |\tilde{\beta}| < \lambda; \\ \text{sign}(\tilde{\beta}_j)(|\tilde{\beta}_j| - \lambda) & \text{otherwise.} \end{cases} \tag{16.15}$$

Without the standardization, the solution is almost as simple but less intuitive.

Hence each coordinate-descent update essentially requires an inner product, followed by the soft thresholding operation. This is especially convenient for x_{ij} that are stored in *sparse-matrix* format, since then the inner products need only visit the nonzero values. If the coefficient is zero before the step, and remains zero, one just moves on, otherwise the model is updated.

Moving from the solution at λ_k (for which $|\langle \boldsymbol{x}_j, \boldsymbol{r}\rangle_w| = \lambda_k$ for all the nonzero coefficients $\hat{\beta}_j$), down to the smaller λ_{k+1}, one might expect all variables for which $|\langle \boldsymbol{x}_j, \boldsymbol{r}\rangle_w| \geq \lambda_{k+1}$ would be natural candidates for the new active set. The *strong rules* lower the bar somewhat, and include any variables for which $|\langle \boldsymbol{x}_j, \boldsymbol{r}\rangle_w| \geq \lambda_{k+1} - (\lambda_k - \lambda_{k+1})$; this tends to rarely make mistakes, and still leads to considerable computational savings.

Apart from variations in the loss function, other penalties are of interest as well. In particular, the *elastic net* penalty bridges the gap between the lasso and ridge regression. That penalty is defined as

$$P_\alpha(\beta) = \frac{1}{2}(1-\alpha)\|\beta\|_2^2 + \alpha\|\beta\|_1, \tag{16.16}$$

where the factor $1/2$ in the first term is for mathematical convenience. When the predictors are excessively correlated, the lasso performs somewhat poorly, since it has difficulty in choosing among the correlated cousins. Like ridge regression, the elastic net shrinks the coefficients of correlated variables toward each other, and tends to select correlated variables in groups. In this case the co-ordinate descent update is almost as simple as in (16.15)

$$\hat{\beta}_j = \begin{cases} 0 & \text{if } |\tilde{\beta}| < \alpha\lambda; \\ \frac{\text{sign}(\tilde{\beta}_j)(|\tilde{\beta}_j| - \alpha\lambda)}{1+(1-\alpha)\lambda} & \text{otherwise,} \end{cases} \tag{16.17}$$

again assuming the observations have weighted variance equal to one. When $\alpha = 0$, the update corresponds to a coordinate update for ridge regression.

Figure 16.9 compares lasso with forward-stepwise logistic regression on the **spam** data, here using all binarized variables and their pairwise interactions. This amounts to 3061 variables in all, once degenerate variables have been excised. Forward stepwise takes a long time to run, since it enters one variable at a time, and after each one has been selected, a new GLM must be fit. The lasso path, as fit by **glmnet**, includes many new variables at each step (λ_k), and is extremely fast (6 s for the entire path). For very large

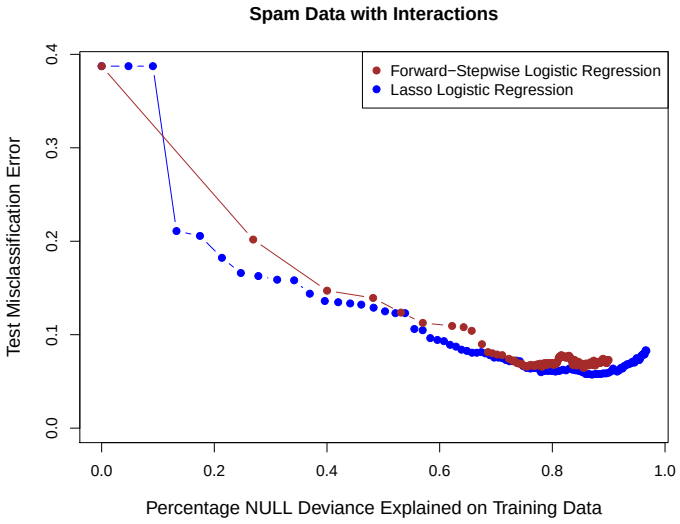

Figure 16.9 Test misclassification error for lasso versus forward-stepwise logistic regression on the `spam` data, where we consider pairwise interactions as well as main effects (3061 predictors in all). Here the minimum error for lasso is 0.057 versus 0.064 for stepwise logistic regression, and 0.071 for the main-effects-only lasso logistic regression model. The stepwise models went up to 134 variables before encountering convergence issues, while the lasso had a largest active set of size 682.

and wide modern data sets (millions of examples and millions of variables), the lasso path algorithm is feasible and attractive.

16.6 Post-Selection Inference for the Lasso

This chapter is mostly about building interpretable models for prediction, with little attention paid to inference; indeed, inference is generally difficult for adaptively selected models.

Suppose we have fit a lasso regression model with a particular value for λ, which ends up selecting a subset $\mathcal{A}$ of size $|\mathcal{A}| = k$ of the p available variables. The question arises as to whether we can assign p-values to these selected variables, and produce confidence intervals for their coefficients. A recent burst of research activity has made progress on these important problems. We give a very brief survey here, with references ap-

†6 pearing in the notes. † We discuss post-selection inference more generally in Chapter 20.

One question that arises is whether we are interested in making inferences about the population regression parameters using the full set of p predictors, or whether interest is restricted to the population regression parameters using only the subset $\mathcal{A}$.

For the first case, it has been proposed that one can view the coefficients of the selected model as an efficient but biased estimate of the full population coefficient vector. The idea is to then *debias* this estimate, allowing inference for the full vector of coefficients. Of course, sharper inference will be available for the stronger variables that were selected in the first place.

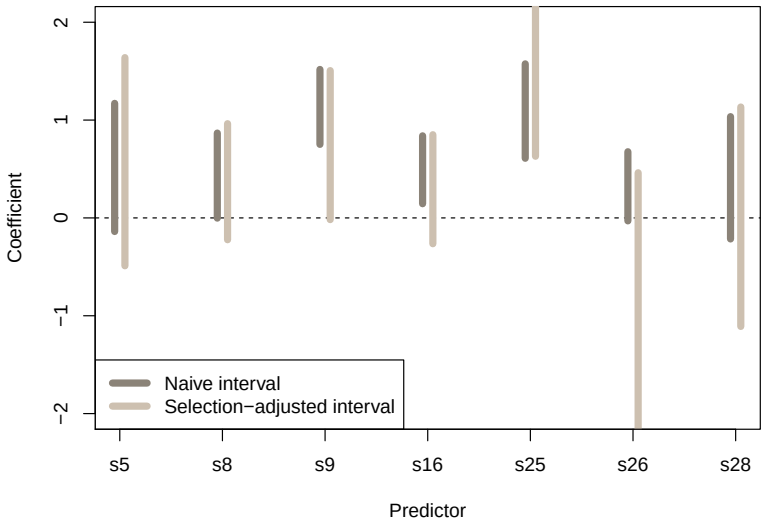

Figure 16.10 HIV data. Linear regression of drug resistance in HIV-positive patients on seven sites, indicators of mutations at particular genomic locations. These seven sites were selected from a total of 30 candidates, using the lasso. The naive 95% confidence intervals (dark) use standard linear-regression inference, ignoring the selection event. The light intervals are 95% confidence intervals, using linear regression, but conditioned on the selection event.

For the second case, the idea is to *condition* on the selection event(s) and hence the set $\mathcal{A}$ itself, and then perform conditional inference on the

unrestricted (i.e. not lasso-shrunk) regression coefficients of the response on only the variables in $\mathcal{A}$. For the case of a lasso with squared-error loss, it turns out that the set of response vectors $\boldsymbol{y} \in \mathbb{R}^N$ that would lead to a particular subset $\mathcal{A}$ of variables in the active set form a convex polytope in $\mathbb{R}^N$ (if we condition on the signs of the coefficients as well; ignoring the signs leads to a finite union of such polytopes). This, along with delicate Gaussian conditioning arguments, leads to truncated Gaussian and t-distrubtions for parameters of interest.

Figure 16.10 shows the results of using the lasso to select variables in an HIV study. The outcome Y is a measure of the resistence to an HIV-1 treatment (nucleoside reverse transcriptase inhibitor), and the 30 predictors are indicators of whether mutations had occurred at particular genomic sites. Lasso regression with 10-fold cross-validation selected a value of $\lambda = 0.003$ and the seven sites indicated in the figure had nonzero coefficients. The dark bars in the figure indicate standard 95% confidence intervals for the coefficients of the selected variables, using linear regression, and ignoring the fact that the lasso was used to select the variables. Three variables are significant, and two more nearly so. The lighter bars are confidence intervals in a similar regression, but conditioned on the selection event. †We see that they are generally wider, and only variable $s25$ remains significant. †7

16.7 Connections and Extensions

There are interesting connections between lasso models and other popular approaches to the prediction problem. We will briefly cover two of these here, namely support-vector machines and boosting.

Lasso Logistic Regression and the SVM

We show in Section 19.3 that ridged logistic regression has a lot in common with the linear support-vector machine. For separable data the limit as $\lambda \downarrow 0$ in ridged logistic regression coincides with the SVM. In addition their loss functions are somewhat similar. The same holds true for ℓ_1 regularized logistic regression versus the ℓ_1 SVM—their end-path limits are the same. In fact, due to the similarity of the loss functions, their solutions are not too different elsewhere along the path. However, the end-path behavior is a little more complex. They both converge to the ℓ_∞ maximizing margin separator—that is, the margin is measured with respect to the ℓ_∞ distance of points to the decision boundary, or maximum absolute coordinate.† †8

Lasso and Boosting

In Chapter 17 we discuss boosting, a general method for building a complex prediction model using simple building components. In its simplest form (regression) boosting amounts to the following simple iteration:

1 Inititialize $b = 0$ and $F^0(x) := 0$.
2 For $b = 1, 2, \ldots, B$:
   (a) compute the residuals $r_i = y_i - F(x_i),\ i = 1, \ldots, n$;
   (b) fit a small regression tree to the observations $(x_i, r_i)_1^n$, which we can think of as estimating a function $g^b(x)$; and
   (c) update $F^b(x) = F^{b-1}(x) + \epsilon \cdot g^b(x)$.

The "smallness" of the tree limits the interaction order of the model (e.g. a tree with only two splits involves at most two variables). The number of terms B and the shrinkage parameter ϵ are both tuning parameters that control the rate of learning (and hence overfitting), and need to be set, for example by cross-validation.

In words this algorithm performs a search in the space of trees for the one most correlated with the residual, and then moves the fitted function F^b a small amount in that direction—a process known as *forward-stagewise fitting*. One can paraphrase this simple algorithm in the context of linear regression, where in step 2(b) the space of small trees is replaced by linear functions.

1 Inititialize $\beta^0 = 0$, and standardize all the variables $\boldsymbol{x}_j,\ j = 1, \ldots, p$.
2 For $b = 1, 2, \ldots, B$:
   (a) compute the residuals $\boldsymbol{r} = \boldsymbol{y} - \boldsymbol{X}\beta^b$;
   (b) find the predictor $\boldsymbol{x}_j$ most correlated with the residual vector $\boldsymbol{r}$; and
   (c) update β^b to β^{b+1}, where $\beta_j^{b+1} = \beta_j^b + \epsilon \cdot s_j$ (s_j being the sign of the correlation), leaving all the other components alone.

For small ϵ the solution paths for this least-squares boosting and the lasso are very similar. It is natural to consider the limiting case or *infinitesimal forward stagewise* fitting, which we will abbreviate *i*FS. One can imagine a scenario where a number of variables are vying to win the competition in step 2(b), and once they are tied their coefficients move in concert as they each get incremented. This was in fact the inspiration for the LAR algorithm 16.3, where $\mathcal{A}$ represents the set of tied variables, and δ is the relative number of turns they each have in getting their coefficients updated. It turns out that *i*FS is often but not always exactly the lasso; it can †9 instead be characterized as a type of monotone lasso.†

Not only do these connections inspire new insights and algorithms for the lasso, they also offer insights into boosting. We can think of boosting as fitting a monotone lasso path in the high-dimensional space of variables defined by all possible trees of a certain size.

Extensions of the Lasso

The idea of using ℓ_1 regularization to induce sparsity has taken hold, and variations of these ideas have spread like wildfire in applied statistical modeling. Along with advances in convex optimization, hardly any branch of applied statistics has been left untouched. We don't go into detail here, but refer the reader to the references in the endnotes. Instead we will end this section with a (non-exhaustive) list of such applications, which may entice the reader to venture into this domain.

- The group lasso penalty $\sum_{k=1}^{K} \|\theta_k\|_2$ applies to vectors θ_k of parameters, and selects whole groups at a time. Armed with these penalties, one can derive lasso-like schemes for including multilevel factors in linear models, as well as hierarchical schemes for including low-order interactions.
- The graphical lasso applies ℓ_1 penalties in the problem of edge selection in dependence graphs.
- Sparse principal components employ ℓ_1 penalties to produce components with many loadings zero. The same ideas are applied to discriminant analysis and canonical correlation analysis.
- The nuclear norm of a matrix is the sum of its singular values—a lasso penalty on matrices. Nuclear-norm regularization is popular in matrix completion for estimating missing entries in a matrix.

16.8 Notes and Details

Classical regression theory aimed for an unbiased estimate of each predictor variable's effect. Modern *wide* data sets, often with enormous numbers of predictors p, make that an untenable goal. The methods described here, by necessity, use shrinkage methods, biased estimation, and sparsity.

The lasso was introduced by Tibshirani (1996), and has spawned a great deal of research. The recent monograph by Hastie *et al.* (2015) gives a compact summary of some of the areas where the lasso and sparsity have been applied. The regression version of boosting was given in Hastie *et al.* (2009, Chapter 16), and inspired the *least-angle regression* algorithm (Efron

et al., 2004)—a new and more democratic version of forward-stepwise regression, as well as a fast algorithm for fitting the lasso. These authors showed under some conditions that each step of the LAR algorithm corresponds to one df; Zou *et al.* (2007) show that, with a fixed λ, the size of the active set is unbiased for the df for the lasso. Hastie *et al.* (2009) also view boosting as fitting a lasso regularization path in the high-dimensional space of trees.

Friedman *et al.* (2010) developed the pathwise coordinate-descent algorithm for generalized lasso problems, and provide the `glmnet` package for `R` (Friedman *et al.*, 2009). Strong rules for lasso screening are due to Tibshirani *et al.* (2012). Hastie *et al.* (2015, Chapter 3) show the similarity between the ℓ_1 SVM and lasso logistic regression.

We now give some particular technical details on topics covered in the chapter.

†1 [p. 301] *Forward-stepwise computations.* Building up the forward-stepwise model can be seen as a guided Gram–Schmidt orthogonalization (QR decomposition). After step r, all $p - r$ variables not in the model are orthogonal to the r in the model, and the latter are in QR form. Then the next variable to enter is the one most correlated with the residuals. This is the one that will reduce the residual sum-of-squares the most, and one requires $p - r$ n-vector inner products to identify it. The regression is then updated trivially to accommodate the chosen one, which is then regressed out of the $p - r - 1$ remaining variables.

†2 [p. 301] *Iteratively reweighted least squares (IRLS).* Generalized linear models (Chapter 8) are fit by maximum-likelihood, and since the log-likelihood is differentiable and concave, typically a Newton algorithm is used. The Newton algorithm can be recast as an iteratively reweighted linear regression algorithm (McCullagh and Nelder, 1989). At each iteration one computes a *working response* variable z_i, and a weight per observation w_i (both of which depend on the current parameter vector $\hat{\beta}$). Then the Newton update for $\hat{\beta}$ is obtained by a weighted least-squares fit of the z_i on the x_i with weights w_i (Hastie *et al.*, 2009, Section 4.4.1).

†3 [p. 301] *Forward-stepwise logistic regression computations.* Although the current model is in the form of a weighted least-squares fit, the $p - r$ variables not in the model cannot be kept orthogonal to those in the model (the weights keep changing!). However, since our current model will have performed a weighted QR decomposition (say), this orthogonalization can be obtained without too much cost. We will need $p - r$ multiplications of an $r \times n$ matrix with an n vector—$O((p - r) \cdot r \cdot n$ computations. An even simpler alternative for the selection is to use the size of the gradient of the

log-likelihood, which simply requires an inner product $|\langle \boldsymbol{y} - \hat{\boldsymbol{\mu}}_r, \boldsymbol{x}_j \rangle|$ for each omitted variable $\boldsymbol{x}_j$ (assuming all the variables are standardized to unit variance).

†4 [p. 306] *Best ℓ_1 interpolant.* If $p > n$, then another boundary solution becomes interesting for the lasso. For t sufficiently large, we will be able to achieve a perfect fit to the data, and hence a zero residual. There will be many such solutions, so it becomes interesting to find the perfect-fit solution with smallest value of t: the minimum-ℓ_1-norm perfect-fit solution. This requires solving a separate convex-optimization problem.

†5 [p. 313] *More on df.* When the search is easy in that a variable stands out as far superior, LAR takes a big step, and forward stepwise spends close to a unit df. On the other hand, when there is close competition, the LAR steps are small, and a unit df is spent for little progress, while forward stepwise can spend a fair bit more than a unit df (the price paid for searching). In fact, the df_j curve for forward stepwise can exceed p for $j < p$ (Jansen *et al.*, 2015).

†6 [p. 318] *Post-selection inference.* There has been a lot of activity around post-selection inference for lasso and related methods, all of it since 2012. To a large extent this was inspired by the work of Berk *et al.* (2013), but more tailored to the particular selection process employed by the lasso. For the debiasing approach we look to the work of Zhang and Zhang (2014), van de Geer *et al.* (2014) and Javanmard and Montanari (2014). The conditional inference approach began with Lockhart *et al.* (2014), and then was developed further in a series of papers (Lee *et al.*, 2016; Taylor *et al.*, 2015; Fithian *et al.*, 2014), with many more in the pipeline.

†7 [p. 319] *Selective inference software.* The example in Figure 16.10 was produced using the `R` package `selectiveInference` (Tibshirani *et al.*, 2016). Thanks to Rob Tibshirani for providing this example.

†8 [p. 319] *End-path behavior of ridge and lasso logistic regression for separable data.* The details here are somewhat technical, and rely on dual norms. Details are given in Hastie *et al.* (2015, Section 3.6.1).

†9 [p. 320] *LAR and boosting.* Least-squares boosting moves the "winning" coefficient in the direction of the correlation of its variable with the residual. The direction δ computed in step 3(a) of the LAR algorithm may have some components whose signs do not agree with their correlations, especially if the variables are very correlated. This can be fixed by a particular nonnegative least-squares fit to yield an exact path algorithm for *i*FS; details can be found in Efron *et al.* (2004).