ECON 950 — Winter 2019
Prof. James MacKinnon

1. Supervised Learning

The objective of supervised learning is typically prediction, broadly defined. From the point of view of econometrics, it involves estimating a sort of **reduced form**.

We have a **training set** of data, with N observations on **inputs** or **predictors** or **features**, together with one or more **outcomes** or **outputs** or **responses**. Often, there is just one output.

Some outputs are quantitative, often approximately continuous. The prediction task is then often called **regression**.

Some outputs are categorical or qualitative, in which case the prediction task is usually called **classification**.

The distinction between regression and classification is not hard and fast. Linear regression can be used for classification.

If y_i is binary, we can regress it on $\boldsymbol{x}_i$ to obtain fitted values $\boldsymbol{x}_i\hat{\boldsymbol{\beta}}$. Then, given a new vector $\boldsymbol{x}$, we can classify that observation as 1 if $\boldsymbol{x}\hat{\boldsymbol{\beta}} \geq 0.5$ and as 0 otherwise.

Of course, we do not have to use 0.5 here, and we could use a logit or probit model instead of a linear regression model.

Some methods are designed for a small number of predictors, which are allowed to affect the outcomes in a very general way. Smoothing methods such as kernel regression fall into this category.

Other methods are designed to handle a large number of predictors, most of which will be discarded.

These are called **high-dimensional** methods. The best known example is the **lasso**. Such mehods can handle problems with far more predictors than observations.

Econometricians have studied nonparametric, especially kernel, regression for a long time, although they have largely ignored other smoothing methods.

Recently, econometricians have begun to study high-dimensional methods. Prominent names include Athey, Belloni, Chernozhukov, and Imbens.

1.1. $\boldsymbol{k}$-Nearest-Neighbour Methods

One simplistic approach to regression and classification is ***k*-nearest-neighbour** averaging. For the former, it works as follows:

1. For any observation with predictors $\boldsymbol{x}$, find the k observations with predictors $\boldsymbol{x}_i$ that are closest to $\boldsymbol{x}$.

   This could be based on Euclidean distance or on some other metric. Note that we may need to rescale some or all of the inputs so that distance is not dominated by one or a few of them.

   Call the set of the k closest observations $N_k(\boldsymbol{x})$.

   When $k = 1$, this set just contains the very closest observation.

2. Compute the average of the y_i over all members of the set $N_k(\boldsymbol{x})$. Call it $\hat{y}(\boldsymbol{x})$. This is our prediction.

kNN with $k = 1$ has no bias when $\boldsymbol{x}$ is part of the training set, but it must surely have high variance.

As k increases, bias goes up but variance goes down.

This is a lot like kernel regression, but with a uniform kernel where the bandwidth varies with the density of the $\boldsymbol{x}_i$.

In parts of the data space where the $\boldsymbol{x}_i$ are dense, the bandwidth is small. In parts where the $\boldsymbol{x}_i$ are sparse, it is big.

We can use kNN for classification instead of regression. We simply classify an observation with predictors $\boldsymbol{x}$ as 1 whenever $\hat{y}(\boldsymbol{x}) \geq 0.5$.

If $k = 1$, this procedure always classifies every observation in the training sample correctly!

There is no reason always to use 0.5. If the cost of one type of misclassification is higher than the cost of another type, we must want to use a different number.

1.2. Statistical Decision Theory

We need a **loss function**, of which the most common is **squared error loss**:

$$L\big(Y, f(\boldsymbol{X})\big) = \big(Y - f(\boldsymbol{X})\big)^2. \tag{1}$$

Conditional on $\boldsymbol{X} = \boldsymbol{x}$, this becomes

$$\mathrm{E}_{Y\,|\,\boldsymbol{X}=\boldsymbol{x}}\big(Y - f(\boldsymbol{x})\big)^2, \tag{2}$$

which is minimized at

$$\mu(\boldsymbol{x}) \equiv \mathrm{E}(Y\,|\,\boldsymbol{X} = \boldsymbol{x}) = f(\boldsymbol{x}). \tag{3}$$

If we had many observations with $\boldsymbol{X} = \boldsymbol{x}$, we could simply average them, and we would get something that estimates $\mu(\boldsymbol{x})$ very well. But this is rarely the case.

If k is large, and the k nearest neighbours are all very close to $\boldsymbol{x}$, then we should also get something that estimates $\mu(\boldsymbol{x})$ very well.

In practice, however, making k large often means that we are averaging points that are not close to $\boldsymbol{x}$.

The larger k is, the more we are smoothing the data.

Formally, we need $N \to \infty$, $k \to \infty$, and $k/N \to 0$. So k has to increase more slowly than N.

We can see how well a particular value of k works by using a **test dataset**, or **test sample**, with M observations. The idea is to estimate the loss function by using the test dataset:

$$\mathrm{MSE}(k) = \sum_{i=1}^{M} \big(y_i - \hat{y}(\boldsymbol{x}_i)\big), \tag{4}$$

where $\hat{y}(\boldsymbol{x}_i)$ is computed from the training set using k nearest neighours. We can evaluate (4) for various values of k to see which one works best.

Depending on how the data are actually generated, kNN may work much better or much worse than regression methods.

- kNN assumes that $f(x)$ is well approximated by a locally constant function.
- Linear regression assumes that $\mu(x)$ is well approximated by a globally linear function.
- Polynomial regression assumes that $\mu(x)$ is well approximated by a globally polynomial function.
- For samples where there are plenty of observations near the values of $\boldsymbol{x}$ that interest us, kNN can work well.

- It may work better than polynomial regression if the function cannot be fit well using a low-order polynomial.
- It can work well if $f(x)$ contains both steep and flat segments, which would be hard to approximate using a polynomial.

See ISLR-fig-3.17-19.pdf.

1.3. Restricted Models

In principle, we could minimize

$$\mathrm{SSR}(f) = \sum_{i=1}^{N} \big(y_i - f(\boldsymbol{x}_i)\big)^2 \tag{5}$$

with respect to the function $f(\cdot)$.

But any function that passes through all training points would fit perfectly.

We have to impose restrictions on $f(\boldsymbol{x})$. Various methods differ in how they do this.

We can either limit the ways in which $f(\boldsymbol{x})$ varies within small neighbourhoods of $\boldsymbol{x}$, or we can limit the size of the neighbourhoods.

The larger is the neighbourhood, the stronger are the constraints.

The less $f(\boldsymbol{x})$ is allowed to vary near $\boldsymbol{x}$, or the more restrictive the ways in which it can vary, the stronger are the constraints.

1.4. Kernel Methods and Local Regression

In general,

$$\text{SSR}(f_{\boldsymbol{\theta}}, \boldsymbol{x}_0) = \sum_{i=1}^{N} K_\lambda(\boldsymbol{x}_0, \boldsymbol{x}_i)\big(y_i - f_{\boldsymbol{\theta}}(x_i)\big)^2, \tag{6}$$

where $K_\lambda(\boldsymbol{x}_0, \boldsymbol{x}_i)$ is the kernel function, which depends on a parameter λ (or h), often called the **bandwidth**, and $f_{\boldsymbol{\theta}}(x_i)$ is a (usually simple) function which depends on a parameter vector $\boldsymbol{\theta}$.

One popular kernel function is the standard normal PDF. In that case,

$$K_\lambda(\boldsymbol{x}_0, \boldsymbol{x}_i) = \frac{1}{\lambda}\phi\big(||\boldsymbol{x}_i - \boldsymbol{x}_0||/\lambda\big).$$

So (6) gives more weight to points that are “close” to $\boldsymbol{x}_0$.

In the univariate case, simple examples of $f_{\boldsymbol{\theta}}(x)$ include

- $f_{\boldsymbol{\theta}}(x) = \theta_0$
- $f_{\boldsymbol{\theta}}(x) = \theta_0 + \theta_1 x$
- $f_{\boldsymbol{\theta}}(x) = \theta_0 + \theta_1 x + \theta_2 x^2$

For the one-dimensional case, the simplest type of kernel regression (Nadaraya-Watson) simply estimates $f(x_0)$ as

$$\hat{f}(x_0) = \frac{1}{\lambda n} \sum_{i=1}^{N} \phi\Big(\frac{x_i - x_0}{\lambda}\Big) y_i. \tag{7}$$

This is just a weighted average of the y_i, with more weight given to points near x_0. As x_0 changes, the weights change, and so $\hat{f}(x_0)$ changes.

Nearest-neighbour methods are just kernel methods with a rather naive data-dependent bandwidth:

$$K_k(x_i, x_0) = \mathbb{I}\big(||x_i - x_0|| \leq ||x_{(k)} - x_0||\big), \tag{8}$$

where $x_{(k)}$ is the training observation ranked k^{th} in distance from x_0.

How much weight an observation x_i gets depends on how many observations are nearer to x_0 than it is. The weight is 1 if there are no more than k such observations, and 0 otherwise.

1.5. Roughness Penalty and Bayesian Methods

The idea is to penalize functions that vary too much locally. In general, we have

$$\text{PSSR}(f;\lambda) = \text{SSR}(f) + \lambda J(f), \tag{9}$$

where J will be large for functions that vary too rapidly within small regions of the input space.

An example is the **cubic smoothing spline**

$$\text{PSSR}(f;\lambda) = \sum_{i=1}^{N} \bigl(y_i - f(x_i)\bigr)^2 + \lambda \int \bigl(f''(x)\bigr)^2 dx. \tag{10}$$

The penalty here applies to the second derivative of f.

For $\lambda = 0$, there is no penalty. As $\lambda \to \infty$, the penalty imposed on functions that are not linear becomes prohibitive.

Many types of penalty functions can be devised. For additive models, it would make sense to have

$$f(\boldsymbol{x}) = \sum_{j=1}^{p} f_j(x_j) \quad \text{and} \quad J(f) = \sum_{j=1}^{p} J(f_j). \tag{11}$$

Projection pursuit regression models have

$$f(\boldsymbol{x}) = \sum_{m=1}^{M} g_m({\boldsymbol{\alpha}_m}^{\top}\boldsymbol{x}) \tag{12}$$

for adaptively chosen directions $\boldsymbol{\alpha}_m$, and the g_m functions can each have an associated roughness penalty.

Penalty functions are often equivalent to **regularization methods**, of which the best known is **ridge regression**. It uses ℓ_2-regularization.

For the linear regression model $\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{u}$, the ridge regression estimator is

$$\hat{\boldsymbol{\beta}}_{\text{ridge}} = (\boldsymbol{X}^\top \boldsymbol{X} + \lambda \mathbf{I})^{-1} \boldsymbol{X}^\top \boldsymbol{y}, \tag{13}$$

where λ is a complexity parameter. Observe that $\hat{\boldsymbol{\beta}}_{\text{ridge}}$ is the solution to

$$\min\big((\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})^\top (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}) + \lambda \boldsymbol{\beta}^\top \boldsymbol{\beta}\big). \tag{14}$$

One big advantage of ridge regression and related methods is that $\boldsymbol{X}^\top \boldsymbol{X} + \lambda \mathbf{I}$ is nonsingular, even if $p > N$.

Penalty function methods have a Bayesian interpretation. Our prior belief is that the functions we seek to estimate exhibit a certain type of smooth behaviour.

The penalty J corresponds to a log-prior, and the penalized SSR function corresponds to a log-posterior.

Minimizing the latter corresponds to finding a posterior mode, whereas a fully Bayesian procedure would seek to find a posterior mean.

1.6. Basis Functions and Dictionary Methods

This includes linear and polynomial regression functions, and also a wide variety of more flexible models.

In general,

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sum_{m=1}^{M} \theta_m h_m(\boldsymbol{x}). \tag{15}$$

The **basis functions** $h_m(\boldsymbol{x})$ are typically nonlinear and may include parameters that have to be estimated. The model is linear in the basis functions, with parameters θ_m to be estimated.

Polynomial splines of degree K are represented by a sequence of M **spline basis functions** determined by $M - K - 1$ **knots**.

The functions are piecewise polynomials of degree K between the knots, joined with continuity of degree $K - 1$ at the knots.

For one-dimensional linear splines, the spline basis functions are

$$\begin{aligned} b_1(x) &= 1, \\ b_2(x) &= x, \\ b_3(x) &= (x - t_1)_+, \\ b_m(x) &= (x - t_m)_+, \end{aligned} \tag{16}$$

for $m = 1, \ldots, M - 2$. Here t_m is the m^{th} knot, and

$$(x - t_m)_+ = \max(x - t_m, 0). \tag{17}$$

A single-layer feed-forward **neural network** model with linear output weights can be thought of as an adaptive basis function method.

This model is

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sum_{m=1}^{M} \beta_m \mathit{\Lambda}(b_m + \boldsymbol{\alpha}_m{}^{\top}\boldsymbol{x}), \tag{18}$$

where $\Lambda(z)$ denotes what is called the **activation function** in the NN literature, but we would all the **logistic function**:

$$\Lambda(z) = \frac{1}{1 + \exp(-z)} = \frac{\exp(z)}{1 + \exp(x)}. \tag{19}$$

The hard part here is determining the **directions** $\boldsymbol{\alpha}_m$ and the **bias terms** b_m.

Adaptive basis function methods are also called **dictionary methods**, because we start with a large (perhaps infinite) set of candidate basis functions. This set is called a **dictionary**.

In recent years, there has been a great deal of work on **deep learning**, i.e., neural networks that have a great many layers and potentially millions (!) of parameters.

These work extraordinarily well in certain contexts, such as identifying objects in photographs and generating fake photographs. They are a key part of recent work on **artificial intelligence**.

1.7. Model Selection

All of the methods we have discussed involve some kind of **smoothing parameter** or **complexity parameter**. For example:

- k for kNN regression;
- the bandwidth for kernel regression;
- the multiplier of the penalty term in penalty methods;
- the number of basis functions.

We cannot determine this parameter on the basis of the fit for the training sample, because we can always find values that make the model fit perfectly (e.g. $k = 1$ for kNN), or at least fit extremely well within the sample.

The **expected prediction error** for kNN is

$$\mathrm{EPE}_k(\boldsymbol{x}_0) = \mathrm{E}\Big(\big(y - \hat{f}_k(\boldsymbol{x}_0)\big)^2 \,\Big|\, \boldsymbol{x} = \boldsymbol{x}_0\Big). \tag{20}$$

This is equal to

$$\mathrm{E}\big(y - \mu(\boldsymbol{x}_0)\big)^2 + \mathrm{E}\big(\hat{f}_k(\boldsymbol{x}_0) - f_k(\boldsymbol{x}_0)\big)^2 + \mathrm{E}\big(f_k(\boldsymbol{x}_0) - \mu(\boldsymbol{x}_0)\big)^2, \tag{21}$$

where of course everything is conditional on $\boldsymbol{x} = \boldsymbol{x}_0$.

The first term in (21) is the **irreducible error**. Even if we knew $\mu(\boldsymbol{x})$, we would make mistakes, because the realization of y is random.

The second term in (21) is the variance of the prediction around its mean. The k subscript indicates the number of nearest neighbours. For other methods, we would index $f(\boldsymbol{x}_0)$ and $\hat{f}(\boldsymbol{x}_0)$ differently.

For kNN, the prediction is simply an average of k values of y_i. Therefore, under the probably unrealistic assumption of independent and homoskedastic disturbances, the second term in (21) would reduce to σ^2/k.

The last term in (21) is the squared bias. For kNN it becomes

$$\left(\frac{1}{k}\sum_{\ell=1}^{k} f(x_{(\ell)}) - \mu(\boldsymbol{x}_0)\right)^2, \tag{22}$$

where ℓ indexes the nearest neighbours to $\boldsymbol{x}_0$.

This term can be expected to increase with k if $\mu(\boldsymbol{x})$ is reasonably smooth, because we are averaging over points that are further away from $\boldsymbol{x}_0$.

In general, the bias term declines with model complexity, and the variance term increases. Note that, for kNN, the model becomes *more* complex as k diminishes.

Averaging over fewer neighbours is equivalent to imposing less stringent smoothness penalties or fewer restrictions. This leads to **overfitting**.

If we graph prediction error for both the training sample and the test sample as a function of complexity, we should see that:

- The prediction error for the training sample declines monotonically as complexity increases;
- The prediction error for the test sample initially declines and then increases as complexity increases.

We are evidently going to have to use some procedure that penalizes complexity in order to avoid overfitting.

In principle, we could use a separate **validation sample**, like the test sample. Unless data are very plentiful, however, it is usually better to employ **cross-validation**. This uses the training sample in an ingenious way.

1.8. Cross-Validation

The idea of cross-validation is to estimate the MSE of a nonparametric estimator by using the training sample, but omitting the observations we are evaluating.

Consider the **leave-one-out estimator** for a kernel regression:

$$\hat{f}_{-i}(x_i) = \frac{1}{\lambda(n-1)} \sum_{j \neq i}^{n} \frac{1}{\lambda} \phi\left(\frac{x_j - x_i}{\lambda}\right). \tag{23}$$

This is just the kernel estimator of $f(x_i)$ using every observation except the i^{th}. It is normally computed at the point $x = x_i$, so as to get an estimate of $f(x_i)$ that does not depend on x_i.

For any choice of λ, we can compute the MSE using the $\hat{f}_{-i}(x_i)$ instead of the $\hat{f}(x_i)$. The result is

$$\text{MSE}_{\text{CV}}(\lambda) = \sum_{i=1}^{N} \big(y_i - \hat{f}_{-i}(x_i)\big)^2. \tag{24}$$

Then we can see which value of λ gives us the lowest value.

If λ is too small, bias will be small but variance will be large. If it is too large, bias will be large but variance will be small. Ideally, cross-validation will allow us to find the optimal value of λ (or k).

2. Methods Based on Linear Regression

The methods discussed here are primarily intended for high-dimensional situations, where p is large relative to N, perhaps much larger than N.

We can write

$$f(\boldsymbol{x}) = \beta_0 + \boldsymbol{x}\boldsymbol{\beta} = \beta_0 + \sum_{j=1}^{p} \beta_j x_j, \tag{25}$$

treating the constant and other regressors separately. In the notation of ETM, $p = k - 1$, and the number of observations is N instead of n.

Of course, the x_j can include transformations of inputs, such as square roots, logs, squares, or higher powers, various types of dummy variables, and products of inputs or of transformations of them.

So a “linear” regression model can actually allow for very nonlinear relationships.

2.1. Regression by Successive Orthogonalization

When the $\boldsymbol{x}_j$ are orthogonal, the $\hat{\beta}_j = \boldsymbol{x}_j^\top \boldsymbol{y} / \boldsymbol{x}_j^\top \boldsymbol{x}_j$ obtained by univariate regressions of $\boldsymbol{y}$ on each of the $\boldsymbol{x}_j$ are equal to the multiple regression least squares estimates.

We can accomplish something similar even when the data are not orthogonal. This is called **Gram-Schmidt Orthogonalization**. It is related to the FWL Theorem.

1. Set $\boldsymbol{z}_0 = \boldsymbol{x}_0 = \boldsymbol{\iota}$.
2. Regress $\boldsymbol{x}_1$ on $\boldsymbol{z}_0$ to obtain a residual vector $\boldsymbol{z}_1$ and coefficient $\hat{\gamma}_{01}$.
3. For $j = 2, \ldots, p$, regress $\boldsymbol{x}_j$ on $\boldsymbol{z}_0, \ldots, \boldsymbol{z}_{j-1}$ to obtain a residual vector $\boldsymbol{z}_j$ and coefficients $\hat{\gamma}_{\ell j}$ for $\ell = 0, \ldots, j-1$.
4. Regress $\boldsymbol{y}$ on $\boldsymbol{z}_p$ to obtain $\hat{\beta}_p = \boldsymbol{z}_p^\top \boldsymbol{y} / \boldsymbol{z}_p^\top \boldsymbol{z}_p$.

The scalar $\hat{\beta}_p$ is the coefficient on $\boldsymbol{x}_p$ in the multiple regression of $\boldsymbol{y}$ on a constant and all the $\boldsymbol{x}_j$.

Note that step 3 simply involves j univariate regressions.

If we did this $p+1$ times, changing the ordering so that each of the $\boldsymbol{x}_j$ came last, we could obtain all the OLS coefficients.

The **QR decomposition** of $\boldsymbol{X}$ is

$$\boldsymbol{X} = \boldsymbol{Z}\boldsymbol{D}^{-1}\boldsymbol{D}\boldsymbol{\Gamma} = \boldsymbol{Q}\boldsymbol{R}, \tag{26}$$

where $\boldsymbol{Z}$ contains the columns $\boldsymbol{z}_0$ to $\boldsymbol{z}_p$ (in order), $\boldsymbol{\Gamma}$ is an upper triangular matrix containing the $\hat{\gamma}_{\ell j}$, and $\boldsymbol{D}$ is a diagonal matrix with typical diagonal element $||\boldsymbol{z}_j||$.

It can be seen that $\boldsymbol{Q}$ is an $N \times (p+1)$ orthogonal matrix with the property that $\boldsymbol{Q}^\top\boldsymbol{Q} = \mathbf{I}$, and $\boldsymbol{R}$ is a $(p+1) \times (p+1)$ upper triangular matrix.

The QR decomposition provides a convenient orthogonal basis for $\mathcal{S}(\boldsymbol{X})$.

It is easy to see that

$$\hat{\boldsymbol{\beta}} = (\boldsymbol{X}^\top\boldsymbol{X})^{-1}\boldsymbol{X}^\top\boldsymbol{y} = \boldsymbol{R}^{-1}\boldsymbol{Q}^\top\boldsymbol{y}, \tag{27}$$

because

$$(\boldsymbol{X}^\top\boldsymbol{X})^{-1} = (\boldsymbol{R}^\top\boldsymbol{Q}^\top\boldsymbol{Q}\boldsymbol{R})^{-1} = (\boldsymbol{R}^\top\boldsymbol{R})^{-1} = \boldsymbol{R}^{-1}(\boldsymbol{R}^\top)^{-1},$$

and

$$\boldsymbol{X}^\top\boldsymbol{y} = \boldsymbol{R}^\top\boldsymbol{Q}^\top\boldsymbol{y}.$$

Since $\boldsymbol{R}$ is triangular, it is very easy to invert. Also,

$$\hat{\boldsymbol{y}} = \boldsymbol{P}_{\boldsymbol{X}}\boldsymbol{y} = \boldsymbol{Q}\boldsymbol{Q}^\top\boldsymbol{y}. \tag{28}$$

so that

$$\mathrm{SSR}(\hat{\boldsymbol{\beta}}) = (\boldsymbol{y} - \boldsymbol{Q}\boldsymbol{Q}^\top\boldsymbol{y})^\top(\boldsymbol{y} - \boldsymbol{Q}\boldsymbol{Q}^\top\boldsymbol{y}) = \boldsymbol{y}^\top\boldsymbol{y} - \boldsymbol{y}^\top\boldsymbol{Q}\boldsymbol{Q}^\top\boldsymbol{y}. \tag{29}$$

The diagonals of the hat matrix are the diagonals of $\boldsymbol{Q}\boldsymbol{Q}^\top$.

The QR decomposition is easy to implement and numerically stable. It requires $O(Np^2)$ operations.

An alternative is to form $\boldsymbol{X}^\top\boldsymbol{X}$ and $\boldsymbol{X}^\top\boldsymbol{y}$ efficiently, use the Cholesky decomposition to invert $\boldsymbol{X}^\top\boldsymbol{X}$, and then multiply $\boldsymbol{X}^\top\boldsymbol{y}$ by $(\boldsymbol{X}^\top\boldsymbol{X})^{-1}$.

The Cholesky approach requires $O(p^3 + Np^2/2)$ operations. For $p << N$, Cholesky wins, but for large p QR does. Never use a general matrix inversion routine.

There are cases where QR yields reliable results and Cholesky does not. If $\boldsymbol{X}^\top\boldsymbol{X}$ is not well-conditioned (i.e., if the ratio of the largest to the smallest eigenvalue is large), then it is much safer to use QR than Cholesky.

2.2. Subset Selection

When p is large, OLS tends to have low bias but large variance.

By setting some coefficients to zero, we can often reduce variance substantially while increasing bias only modestly.

It can be hard to make sense of models with a great many coefficients.

Best-subset regression tries every subset of size $k = 1, \ldots, p$ to find the one that, for each k, has the smallest SSR.

There is an efficient algorithm, but it becomes infeasible once p becomes even moderately large (say much over 40).

Forward-stepwise selection and **backward-stepwise selection** are feasible even when p is large. The former works even if $p > N$.

To begin the forward-stepwise selection procedure, we regress $\boldsymbol{y}$ on each of the $\boldsymbol{x}_j$ and pick the one that yields the smallest SSR.

Next, we add each of the remaining $p - 1$ regressors, one at a time, and pick the one that yields the smallest SSR.

Then we add each of the remaining $p-2$ regressors, one at a time, and pick the one that yields the smallest SSR.

If we are using the QR decomposition, we can do this efficiently. We are just adding one column to $\boldsymbol{Z}$ and hence $\boldsymbol{Q}$, and one more to $\boldsymbol{\Gamma}$ and hence $\boldsymbol{R}$.

We do this for each of the candidate regressors and calculate the SSR from (29) for each of them. We can reuse most of the elements of the old $\boldsymbol{Q}$ matrix.

For backward-stepwise regression, we start with the full model (assuming that $p < N$) and then successively drop regressors. At each step, we drop the one with the smallest t statistic.

Smart stepwise programs will recognize that regressors often come in groups. It generally does not make sense to

- Drop a regressor X if the model includes X^2 or XZ;
- Drop some dummy variables that are part of a set (e.g. some province dummies but not all, or some categorical dummies but not all).

In cases like this, variables should be added, or subtracted, in groups.

If we use either forward-stepwise or backward-stepwise selection, how do we decide which model to choose?

The model with all p regressors will always fit best, in terms of SSR.

We either need to penalize the number of parameters (d) or use cross-validation.

$$C_p\colon \quad \frac{1}{N}(\text{SSR} + 2d\hat{\sigma}^2) = (1 + 2d/N)\hat{\sigma}^2. \tag{30}$$

Thus C_p adds a penalty of $2d\hat{\sigma}^2$ to SSR.

The **Akaike information criterion**, or **AIC**, is defined for all models estimated by maximum likelihood. For linear regression models with Gaussian errors,

$$\text{AIC} = \frac{1}{N\hat{\sigma}^2}(\text{SSR} + 2d\hat{\sigma}^2), \tag{31}$$

which is proportional to C_p.

The **Bayesian information criterion**, or **BIC**, for linear regression models is

$$\text{BIC} = \frac{1}{N}\big(\text{SSR} + \log(N)d\hat{\sigma}^2\big). \tag{32}$$

When N is large, this places a much heavier penalty on d than does AIC, because $\log(N) >> 2$. Therefore, the chosen model is very likely to have lower complexity.

If the correct model is among the ones we estimate, BIC chooses it with probability 1 as $N \to \infty$. The other two criteria sometimes choose excessively complex models.

2.3. Ridge Regression

As we have seen, ridge regression minimizes the objective function

$$\sum_{i=1}^{N}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2 + \lambda \sum_{j=1}^{p}\beta_j^2, \tag{33}$$

where λ is a complexity parameter that controls the amount of shrinkage.

Minimizing (33) is equivalent to solving the constrained minimization problem

$$\min \sum_{i=1}^{N}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2 \text{ subject to } \sum_{j=1}^{p}\beta_j^2 \le t. \tag{34}$$

In (33), λ plays the role of a Lagrange multiplier. For any value of t, there is an associated value of λ, which might be 0 if t is sufficiently large.

Notice that the intercept is not included in the penalty term. The simplest way to accomplish this is to recenter all variables before running the regression.

It would seem very strange to impose the same penalty on each of the β_j^2 if the associated regressors were not scaled similarly. Therefore, it is usual to rescale all the regressors to have variance 1. They already have mean 0 from the recentering.

We have seen that

$$\hat{\boldsymbol{\beta}}_{\text{ridge}} = (\boldsymbol{X}^\top\boldsymbol{X} + \lambda\mathbf{I}_p)^{-1}\boldsymbol{X}^\top\boldsymbol{y}, \tag{35}$$

where now every column of $\boldsymbol{X}$ has sample mean 0 and sample variance 1.

A number that is often more interesting than λ is the **effective degrees of freedom** of the ridge regression fit. If the d_j are the **singular values** of $\boldsymbol{X}$ (see below), then

$$\text{df}(\lambda) \equiv \text{Tr}\big(\boldsymbol{X}(\boldsymbol{X}^\top\boldsymbol{X} + \lambda\mathbf{I}_p)^{-1}\boldsymbol{X}^\top\big) = \sum_{j=1}^{p} \frac{d_j^2}{d_j^2 + \lambda}\,. \tag{36}$$

As $\lambda \to 0$, $\text{df}(\lambda) \to p$.

Thus the effective degrees of freedom for the ridge regression is never greater than the actual degrees of freedom for the corresponding OLS regression.

The ridge regression estimator (35) can be obtained as the mode (and also the mean) of a Bayesian posterior. Suppose that

$$y_i \sim \text{N}(\beta_0 + \boldsymbol{x}_i^\top \boldsymbol{\beta}, \sigma^2), \tag{37}$$

and the β_j are believed to follow independent $\text{N}(0, \tau^2)$ distributions. Then expression (33) is minus the log of the posterior, with $\lambda = \sigma^2/\tau^2$.

Thus minimizing (33) yields the posterior mode, which is $\hat{\boldsymbol{\beta}}_{\text{ridge}}$.

Evidently, as τ increases, λ becomes smaller. The more diffuse our prior about the β_j, the less it affects the posterior mode.

Similarly, as σ^2 increases, so that the data become noisier, the more our prior influences the posterior mode.

A simple way to compute ridge regression for any given λ is to form the augmented dataset

$$\boldsymbol{X}_a = \begin{bmatrix} \boldsymbol{X} \\ \sqrt{\lambda}\mathbf{I}_p \end{bmatrix} \quad \text{and} \quad \boldsymbol{y}_a = \begin{bmatrix} \boldsymbol{y} \\ \mathbf{0} \end{bmatrix}, \tag{38}$$

which has $N + p$ observations. Then OLS estimation of the regression

$$\boldsymbol{y}_a = \boldsymbol{X}_a\boldsymbol{\beta} + \boldsymbol{u}_a \tag{39}$$

yields the ridge regression estimator (35), because

$$\boldsymbol{X}_a^\top\boldsymbol{X}_a = \boldsymbol{X}^\top\boldsymbol{X} + \lambda\mathbf{I}_p \quad \text{and} \quad \boldsymbol{X}_a^\top\boldsymbol{y}_a = \boldsymbol{X}^\top\boldsymbol{y}. \tag{40}$$

2.4. The Singular Value Decomposition

The **singular value decomposition** of $\boldsymbol{X}$ is

$$\boldsymbol{X} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^\top, \tag{41}$$

where $\boldsymbol{U}$ and $\boldsymbol{V}$ are $N \times p$ and $p \times p$ orthogonal matrices, with $\mathcal{S}(\boldsymbol{U}) = \mathcal{S}(\boldsymbol{X})$. Thus $\boldsymbol{U}^\top\boldsymbol{U} = \boldsymbol{V}^\top\boldsymbol{V} = \mathbf{I}_p$. The columns of $\boldsymbol{V}$ are the **eigenvectors** of $\boldsymbol{X}^\top\boldsymbol{X}$.

Recall that λ is an **eigenvalue** of a matrix $\boldsymbol{A}$ if there exists a nonzero vector $\boldsymbol{\xi}$, called an **eigenvector**, such that

$$\boldsymbol{A\xi} = \lambda\boldsymbol{\xi}. \tag{42}$$

Thus the action of $\boldsymbol{A}$ on $\boldsymbol{\xi}$ produces a vector with the same direction as $\boldsymbol{\xi}$, but a different length unless $\lambda = 1$.

In (41), $\boldsymbol{D}$ is a $p \times p$ diagonal matrix, with diagonal elements d_j that have the property

$$d_1 \geq d_2 \geq \ldots \geq d_p \geq 0. \tag{43}$$

These are the square roots of the eigenvalues of $\boldsymbol{X}^\top\boldsymbol{X}$.

The rank of the matrix $\boldsymbol{D}$, and also of the matrix $\boldsymbol{X}^\top\boldsymbol{X}$, is the number of d_j that are strictly positive.

Observe that

$$\begin{aligned}\boldsymbol{X}(\boldsymbol{X}^\top\boldsymbol{X})^{-1}\boldsymbol{X}^\top\boldsymbol{y} &= \boldsymbol{UDV}^\top(\boldsymbol{VDU}^\top\boldsymbol{UDV}^\top)^{-1}\boldsymbol{VDU}^\top\boldsymbol{y} \\ &= \boldsymbol{UDV}^\top(\boldsymbol{DV}^\top)^{-1}(\boldsymbol{U}^\top\boldsymbol{U})^{-1}(\boldsymbol{VD})^{-1}\boldsymbol{VDU}^\top\boldsymbol{y} \\ &= \boldsymbol{UU}^\top\boldsymbol{y}.\end{aligned} \tag{44}$$

We can now see that

$$\begin{aligned} \boldsymbol{X}\hat{\boldsymbol{\beta}}_{\text{ridge}} &= (\boldsymbol{X}^\top\boldsymbol{X} + \lambda\mathbf{I})^{-1}\boldsymbol{X}^\top\boldsymbol{y} \\ &= \boldsymbol{U}\boldsymbol{D}(\boldsymbol{D}^2 + \lambda\mathbf{I})^{-1}\boldsymbol{D}\boldsymbol{U}^\top\boldsymbol{y} \\ &= \sum_{j=1}^{p} \boldsymbol{u}_j \frac{d_j^2}{d_j^2 + \lambda}\boldsymbol{u}_j^\top\boldsymbol{y}, \end{aligned} \tag{45}$$

where $\boldsymbol{u}_j$ is the j^{th} column of $\boldsymbol{U}$.

When $\lambda = 0$, the scalars in the last line of (45) collapse to 1, and we see that

$$\boldsymbol{P}_{\boldsymbol{X}}\boldsymbol{y} = \boldsymbol{X}\hat{\boldsymbol{\beta}}_{\text{OLS}} = \sum_{j=1}^{p} \boldsymbol{u}_j\boldsymbol{u}_j^\top\boldsymbol{y}, \tag{46}$$

which is another way of writing the last line of (44).

Observe from (45) that there is more shrinkage applied to the coordinates of basis vectors with small values of d_j.

The **eigen decomposition** of $\boldsymbol{X}^\top\boldsymbol{X}$ is

$$\boldsymbol{X}^\top\boldsymbol{X} = \boldsymbol{V}\boldsymbol{D}^2\boldsymbol{V}^\top. \tag{47}$$

The eigenvectors $\boldsymbol{v}_j$ are the columns of $\boldsymbol{V}$.

The largest eigenvector $\boldsymbol{v}_1$ has the property that $\boldsymbol{z}_1 = \boldsymbol{X}\boldsymbol{v}_1$ has the largest sample variance among all normalized linear combinations of the columns of $\boldsymbol{X}$.

Since $\boldsymbol{z}_1 = \boldsymbol{u}_1 d_1$, this sample variance is

$$\mathrm{Var}(\boldsymbol{X}\boldsymbol{v}_1) = \mathrm{Var}(\boldsymbol{u}_1 d_1) = d_1^2/N. \tag{48}$$

The first **principal component** of $\boldsymbol{X}$ is $\boldsymbol{z}_1$.

Subsequent principal components are orthogonal to $\boldsymbol{z}_1$, and to each other. Their variance declines as j increases.

Thus small singular values d_j correspond to directions in $\mathcal{S}(\boldsymbol{X})$ that have small variance. Ridge regression shrinks these directions the most, because they provide relatively little information, which may easily be swamped by noise.

2.5. Principal Components Regression

Recall from (48) that the principal components of $\boldsymbol{X}$ are $\boldsymbol{z}_m = \boldsymbol{X}\boldsymbol{v}_m$, where the $\boldsymbol{v}_m$ are the eigenvectors, that is, the columns of $\boldsymbol{V}$. These are $p \times 1$.

If we regress $\boldsymbol{y}$ on the first M principal components, we find that

$$\hat{\boldsymbol{y}}^{\text{pc}} = \bar{y}\boldsymbol{\iota} + \sum_{m=1}^{M} \hat{\theta}_m \boldsymbol{z}_m, \tag{49}$$

where, because the $\boldsymbol{z}_m$ are orthogonal,

$$\hat{\theta}_m = \frac{{\boldsymbol{z}_m}^{\top}\boldsymbol{y}}{{\boldsymbol{z}_m}^{\top}\boldsymbol{z}_m}. \tag{50}$$

Because of this, it is extremely easy to calculate $\hat{\boldsymbol{y}}^{\text{pc}}$ for many values of M.

As with other methods, it is a good idea to standardize the $\boldsymbol{x}_j$ so that they have mean 0 and variance 1.

The principal components are ordered in the same way as the eigenvalues:

$$||\boldsymbol{z}_1|| \geq ||\boldsymbol{z}_2|| \geq ||\boldsymbol{z}_3|| \geq \ldots \geq ||\boldsymbol{z}_p||. \tag{51}$$

Thus $\boldsymbol{z}_1$ is the direction in which the columns of $\boldsymbol{X}$ vary the most.

For principal components regression (PCR), we first regress $\boldsymbol{y}$ on $\boldsymbol{z}_1$, then on $\boldsymbol{z}_1$ and $\boldsymbol{z}_2$, and so on.

Because the $\boldsymbol{z}_m$ are orthogonal, adding another regressor, say $\boldsymbol{z}_g$, simply means regressing the current residual

$$\hat{\boldsymbol{\epsilon}}_{g-1} \equiv \boldsymbol{y} - \hat{\theta}_0 - \hat{\theta}_1 \boldsymbol{z}_1 - \hat{\theta}_2 \boldsymbol{z}_2 - \ldots - \hat{\theta}_{g-1} \boldsymbol{z}_{g-1} \tag{52}$$

on $\boldsymbol{z}_g$. Then we set $\hat{\boldsymbol{\epsilon}}_g = \hat{\boldsymbol{\epsilon}}_{g-1} - \hat{\theta}_g \boldsymbol{z}_g$. We keep going in this way until $g = M$.

The raw fit will improve as M increases, but criteria like AIC may get better or worse. As usual, we can use cross-validation to choose M.

Of course, one can recover the implied β coefficients from the $\hat{\theta}_m$:

$$\hat{\boldsymbol{\beta}}^{\text{pc}} = \sum_{m=1}^{M} \hat{\theta}_m \boldsymbol{v}_m. \tag{53}$$

Especially when M is small, these are likely to be shrunk relative to the OLS estimates. When $M = p$, these will be equal to OLS estimates.

There is a close relationship between ridge regression and PCR. Both operate via the principal components of the input matrix.

In both cases, coefficients are shrunk but not set to zero.

For ridge, directions with small eigenvalues get shrunk more than ones with large eigenvalues; recall (45).

For PCR, directions with small eigenvalues are discarded, while ones with large eigenvalues are retained.

2.6. The Lasso

Lasso (Tibshirani, 1996) stands for **least absolute shrinkage and selection operator**.

The lasso can be obtained as the solution to

$$\min \frac{1}{2N} \sum_{i=1}^{N} \Bigl(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \Bigr)^2 \text{ subject to } \sum_{j=1}^{p} |\beta_j| \leq t. \tag{54}$$

This looks almost identical to (34). The main difference is that the constraint involves absolute values instead of squares. Thus the lasso involves ℓ_1-regularization.

As with ridge regression, solving (54) is equivalent to minimizing

$$\frac{1}{2N}\sum_{i=1}^{N}\Bigl(y_i-\beta_0-\sum_{j=1}^{p}\beta_j x_{ij}\Bigr)^2+\lambda\sum_{j=1}^{p}|\beta_j|, \tag{55}$$

where λ is a Lagrange multiplier; compare (33). Large values of λ are associated with small values of t.

Following SLS but not ESL, there is a factor of $1/(2N)$ in (54) and (55). This is not needed, but it makes the values of λ comparable across samples of different sizes.

If all variables are recentered to have mean zero, we can dispense with the β_0 parameter. We can recover it if necessary as

$$\hat{\beta}_0=\bar{y}-\sum_{j=1}^{p}\hat{\beta}_j\bar{x}_j. \tag{56}$$

The obvious way to solve (54) is by quadratic programming, but there are better approaches, which we discuss below.

The key difference between ridge and lasso is that, for lasso, some of the coefficients will be 0 when t is sufficiently small (equivalently, when λ is sufficiently large).

If t is sufficiently large — greater than $t_0 = \sum_{j=1}^{p} |\hat{\beta}_j|$ — then $\hat{\boldsymbol{\beta}}^{\text{lasso}} = \hat{\boldsymbol{\beta}}^{\text{ols}}$. In this case, the constraints do not bind.

For $t < t_0$, the lasso coefficients are shrunk, and some of them may well be 0. Ultimately, as $t \to 0$, they will all be 0.

For the tuning parameter, we can use λ, t, or the **standardized tuning parameter**

$$s \equiv t \Big/ \sum_{j=1}^{p} |\hat{\beta}_j|. \tag{57}$$

Whichever one we use, it typically has to be chosen by cross-validation.

ESL and SLS recommend using K-fold cross-validation, where $K = 5$ or $K = 10$. We will discuss these methods in detail later.

For K-fold cross-validation, we divide the training sample into K folds (subsamples) of equal or roughly equal size.

Then we sequentially omit one fold at a time, applying the lasso to the other $K-1$ folds. This gives us K sets of estimates.

For $k = 1, \ldots, K$, the estimates using all folds except the k^{th} are then used to compute fitted values for observations in fold k.

We use these fitted values to compute the mean squared prediction error for all observations. This is then plotted against t, s, or λ to choose the optimal value.

As usual, when the bound is too tight, the bias is large, and when the bound is too loose, the variance is large.

In the former case, there will be few non-zero coefficients, and in the latter case there will be many.

See SLS-fig-2.3.pdf for cross-validated estimate of prediction error.

If $\boldsymbol{X}$ is orthonormal, there is an explicit solution for lasso:

$$\hat{\beta}_j^{\text{lasso}} = \text{sign}(\hat{\beta}_j)\big(|\hat{\beta}_j| - \lambda\big)_+, \tag{58}$$

where $(z)_+ = z$ if $z > 0$ and $(z)_+ = 0$ if $z \leq 0$.

Thus $\hat{\beta}_j^{\text{lasso}}$ either has the same sign as $\hat{\beta}_j$ or equals 0.

Like ridge, lasso is defined even when $p+1 > N$.

The equivalent solution for ridge in the orthonormal case is

$$\hat{\beta}_j^{\text{ridge}} = \hat{\beta}_j/(1+\lambda). \tag{59}$$

So it always has the same sign as $\hat{\beta}_j$ and never equals 0.

Like ridge regression, the lasso has a Bayesian interpretation. The prior for β_j is a Laplace (double exponential) distribution:

$$p(\beta_j) = \frac{\lambda}{2}\exp\bigl(-\lambda|\beta_j|\bigr). \tag{60}$$

The lasso has been generalized in many ways. One important one is the **elastic net**, which is discussed below.

ESL-fig3.10.pdf plots the profiles of lasso coefficients against s — see (57) — for the prostate cancer data.

This may be compared with ESL-fig3.08.pdf, which plots the profiles of ridge coefficients against df(λ) for the same data.

The lasso shrinkage causes the estimates of the non-zero coefficients to be biased towards zero and inconsistent.

This bias can be reduced in various ways.

- Use the lasso to identify the set of predictors with non-zero coefficients. Then run another regression on just that set of predictors. Of course, this will not work if there are too many of them.
- Alternatively, use the lasso twice. This is called the **relaxed lasso**. In the second step, include only the variables with non-zero coefficients in the first step.

If cross-validation is used at each step, the penalty parameter λ in the first step is likely to be larger than in the second step, because there are more noise variables to eliminate in the first step.

Since bias increases with λ, there should be less bias after the second step.

2.7. Properties of the Lasso

The lasso objective function is a special case of

$$\frac{1}{2N}\sum_{i=1}^{N}\Big(y_i-\beta_0-\sum_{j=1}^{p}\beta_j x_{ij}\Big)^2+\lambda\sum_{j=1}^{p}|\beta_j|^q. \tag{61}$$

For lasso, $q = 1$, and for ridge, $q = 2$. When $q = 0$, we have best-subset regression.

The smallest value of q for which the constraint set is convex is $q = 1$. Convexity makes optimization much easier.

The maximum number of non-zero coefficients is $\min(N, p)$.

Various consistency results have been proved for the lasso. The true parameter vector needs to be sparse relative to $N/\log(p)$. So it needs to contain a lot of zeros!

The obvious way to obtain standard errors for lasso is to use the pairs bootstrap. Resample from $(y_i, \boldsymbol{x}_i)$ pairs and apply the procedure to each bootstrap sample.

We can use the standard deviations of the bootstrap lasso estimates around their mean as standard errors.

We can also calculate the proportion of the time that each parameter estimate is non-zero.

2.8. Least Angle Regression

Least angle regression (Efron, Hastie, Johnstone, and Tibshirani, 2004) can be thought of as an alternative to stepwise regression, but it also provides an efficient way to compute the lasso.

Before doing anything else, standardize all predictors to have mean 0 and unit norm, so that $||\boldsymbol{x}_j|| = 1$.

1. Let $\boldsymbol{r} = \boldsymbol{y} - \bar{\boldsymbol{y}}$.
2. Find the predictor $\boldsymbol{x}_j$ most correlated with $\boldsymbol{r}$.
3. Move β_j from 0 towards $\boldsymbol{x}_j^\top \boldsymbol{r} = (\boldsymbol{x}_j^\top \boldsymbol{x}_j)^{-1} \boldsymbol{x}_j^\top \boldsymbol{r}$ until some other $\boldsymbol{x}_k$ has as much correlation with the current residual as does $\boldsymbol{x}_j$.
4. Move both β_j and β_k together in the direction of the least squares coefficient of the current residual on $[\boldsymbol{x}_j, \boldsymbol{x}_k]$ until some other $\boldsymbol{x}_\ell$ has as much correlation with the current residual.

5. Continue in this way until all p predictors have been included in the regression. After $\min(N-1, p)$ steps, we arrive at the least squares solution.

The direction of the active step after variable k is entered is

$$\boldsymbol{\delta}_k = (\boldsymbol{X}_{[k]}^\top \boldsymbol{X}_{[k]})^{-1} \boldsymbol{X}_{[k]} (\boldsymbol{y} - \boldsymbol{X}_{[k]} \boldsymbol{\beta}_{[k]}). \tag{62}$$

Here $\boldsymbol{X}_{[k]}$ denotes the columns of $\boldsymbol{X}$ that are now included, and $\boldsymbol{\beta}_{[k]}$ denotes the coefficients, one of which is 0.

The coefficients then evolve as

$$\boldsymbol{\beta}_{[k]}(\alpha) = \boldsymbol{\beta}_{[k]} + \alpha \boldsymbol{\delta}_k, \tag{63}$$

and the fit vector evolves as

$$\hat{\boldsymbol{f}}_k(\alpha) = \hat{\boldsymbol{f}}_k + \boldsymbol{X}_{[k]} \boldsymbol{\delta}_k. \tag{64}$$

The exact step length and the identity of the next variable to add can be calculated analytically, but ESL does not give details.

If we start at $t = 0$, the lasso coefficient(s) look just like the LAR ones until one of the non-zero lasso coefficients hits zero again.

Therefore, a modified version of the LAR algorithm can be used to compute lasso for all values of t or λ.

Add an additional step between 4. and 5.

4a. If a non-zero coefficient hits 0, drop its variable from the active set and recompute the current least-squares directions.

Note that dropped variables will reappear later.

LAR always takes p steps to obtain the OLS estimates. The lasso version may take more than that, but it tends to be similar.

This procedure works even when $p \gg N$. It provides a whole sequence of lasso estimates for $0 \leq \lambda < \infty$.

Chapter 5 of SLS discusses optimization methods for lasso and other convex optimization problems in much greater detail. It suggests that, although LAR is theoretically attractive, it is not the fastest method.

2.9. Pathwise Coordinate Optimization

We can rewrite (55) as

$$\frac{1}{2}\sum_{i=1}^{N}\Big(y_i - \beta_0 - \sum_{k\neq j}\tilde{\beta}_k(\lambda)x_{ik} - \beta_j x_{ij}\Big)^2 + \lambda\sum_{k\neq j}|\tilde{\beta}_k(\lambda)| + \lambda|\beta_j|, \tag{65}$$

where $\tilde{\beta}_k(\lambda)$ denotes the current estimate of β_k at penalty parameter λ.

Minimizing (65) is a one-dimensional problem. It is effectively a univariate lasso problem with response variable

$$\tilde{u}_i^{(j)} = y_i - \sum_{k\neq j}\tilde{\beta}_k(\lambda)x_{ik}. \tag{66}$$

The solution to this problem is

$$\tilde{\beta}_j(\lambda) = \operatorname{sign}(t)\big(|t| - \lambda\big)_+, \quad t = \sum_{i=1}^{N} x_{ij}\tilde{u}_i^{(j)}; \tag{67}$$

recall (58).

Repeated iteration of (66) and (67) over all j yields the lasso estimate $\hat{\boldsymbol{\beta}}(\lambda)$; see Friedman, Hastie, Hoefling, and Tibshirani (2007). This approach is an example of **coordinate descent**.

Pathwise coordinate optimization is extremely simple. But one disadvantage, relative to LAR, is that it just yields estimates for a single value of λ.

However, estimates for a large value of λ can be used as starting points for a smaller value, and so on. When λ is sufficiently large, $\hat{\boldsymbol{\beta}}^{\text{lasso}} = \mathbf{0}$. We can start at the smallest value of λ for which that is true.

We still only get results for a grid of values, rather than all values as with LAR, but that is often sufficient.

Observe that the actual computation of the $\tilde{u}_i^{j)}$ in (66) and of $\tilde{\beta}_j(\lambda)$ in (67) are extremely simple. Moreover, in many cases, when $\tilde{\beta}_j(\lambda) = 0$, it just stays at 0.

SLS reports simulation results which suggest that coordinate descent is the fastest way to obtain lasso estimates, both when $N >> p$ and when $p >> N$.

2.10. Elastic-Net Penalization

The elastic-net penalty (Zou and Hastie, 2005) has the form

$$\sum_{j=1}^{p} \bigl(\alpha|\beta_j| + (1-\alpha)\beta_j^2\bigr). \tag{68}$$

This reduces to the lasso penalty when $\alpha = 1$ and the ridge penalty when $\alpha = 0$.

As usual, expression (68) is either multiplied by a tuning parameter λ and added to the sum of squares, or it is required to be less than an upper limit t.

Since there are two tuning parameters, cross-validation is a lot more work than for lasso or ridge. To reduce effort, people often just choose α and condition on it.

Elastic-net tends to yield more non-zero coefficients than lasso, but they are shrunk more towards zero.

Unlike lasso, it can yield more than N non-zero coefficients when $p > N$.

We can use any program for lasso to compute the elastic net estimator by using the augmentation trick discussed above to compute ridge regression.

If we multiply (68) by λ and then separate the two penalties, the ridge one becomes

$$(1-\alpha)\lambda\sum_{j=1}^{p}\beta_j^2. \tag{69}$$

As in (38), we can form the augmented dataset

$$\boldsymbol{X}_a = \begin{bmatrix} \boldsymbol{X} \\ \sqrt{(1-\alpha)\lambda}\,\mathbf{I}_p \end{bmatrix} \quad \text{and} \quad \boldsymbol{y}_a = \begin{bmatrix} \boldsymbol{y} \\ \mathbf{0} \end{bmatrix}. \tag{70}$$

Then we can obtain the elastic-net estimates by solving the lasso problem

$$\underset{\boldsymbol{\beta}}{\operatorname{argmin}}\left(\frac{1}{2}(\boldsymbol{y}_a-\boldsymbol{X}_a\boldsymbol{\beta})^\top(\boldsymbol{y}_a-\boldsymbol{X}_a\boldsymbol{\beta})+\alpha\lambda\sum_{j=1}^{p}|\beta_j|\right). \tag{71}$$

This just depends on one tuning parameter, $\alpha\lambda$. However, since $\boldsymbol{X}_a$ depends on $(1-\alpha)\lambda$, we ultimately have to choose two tuning parameters.