**8.1 - Shrinkage Methods (“Automatic” Variable Selection Methods)**

In our review of MLR in Section 2 we reviewed classic stepwise model selection methods: forward, backward, and mixed. All three of these methods will either include or exclude terms starting from an appropriate base model.

Other model selection methods have been developed that are viable alternatives to these in or out strategies. These include ridge regression (old one, but has new found life), LASSO (newer one), and Elastic Net (newest).

Aside from the model selection these methods have also been used extensively in high dimensional regression problems. A high dimensional problem is one in which n < p or n << p. The text authors Hastie & Tibshirani present two examples where this might be the case, but there are certainly many others.

* Predicting blood pressure – rather than use standard predictors such as age, gender, and BMI, one might also collect measurements for half a million *single nucleotide polymorphisms* (SNP’s) for inclusion in the model. Thus we might have and
* Predicting purchasing behavior of online shoppers - using a table of 50,000 key words (coded 0/1) potential customers might use in the process of searching for products (i.e. Amazon.com) we might try to predict their purchasing behavior. We might gather information from 5,000 randomly selected visitors to the website, in which case and

Ridge, Lasso, and Elastic Net regression models will allow us to fit models to these situations where , while OLS mathematically cannot!

**8.2 - Ridge Regression**

Ridge regression chooses parameter estimates,, to minimize the residual sum of squares subject to a penalty on the size of the coefficients. After **standardizing** all potential terms in the model the ridge coefficients minimize

**R Packages**

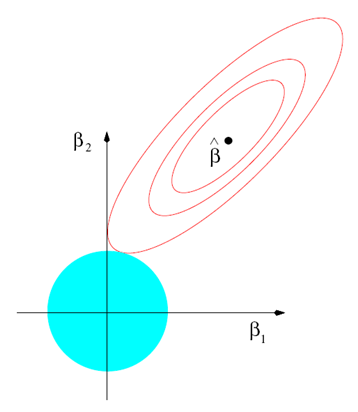
* MASS
* genridge

Here  > 0 is a complexity parameter that controls the amount of shrinkage, the larger  the greater the amount of shrinkage. The intercept is not included in the shrinkage and will be estimated as the mean of the response. An equivalent way to write the ridge regression criterion is

This clearly shows how of the size of the parameter estimates are constrained. Also this formulation of the problem also leads to a nice geometric interpretation of how the penalized least squares estimation works (see figure next page).

**Important Question:** Why is it essential to ***standardize*** the terms in our model?

**Visualization of Ridge Regression**



Usual OLS Estimate =

Contours of the OLS criterion

Ridge regression estimate =

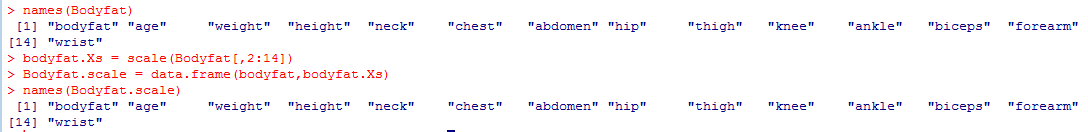
In matrix notation the ridge regression criterion is given by

with the resulting parameter estimates being very similar to those for OLS

*I* is the *k x k* identity matrix.

There are several packages in R that contain functions that perform ridge regression. One we will use is lm.ridge in the package MASS*.* The MASS package actually contains a variety of very useful functions. MASSstands for *Modern Applied Statistics in S-Plus* (old-expensive R) by Venables & Ripley, this is an excellent reference for learning R if you are so inclined. The function call using lm.ridge is very similar to the lm() function. The other function we will use is the function ridge in the **genridge** package. The genridge package contains a number of plotting functions to help visualize the coefficient shrinkage that takes place by using ridge regression.

**Example 8.1 – % Bodyfat**Using the bodyfat dataset we will conduct a ridge regression analysis. In order to fairly compare the parameter estimates obtained via ridge regression to those from OLS we will first run the OLS regression using the ***standardized*** predictors.



> bodyfat.scaled = lm(bodyfat~.,data=Bodyfat.scale)

Comments:

> summary(bodyfat.scaled)

Call:

lm(formula = bodyfat ~ ., data = Bodyfat.scale)

Residuals:

Min 1Q Median 3Q Max

-11.1966 -2.8824 -0.1111 3.1901 9.9979

Coefficients:

Estimate Std. Error t value Pr(>|t|)

(Intercept) 19.15079 0.27147 70.544 < 2e-16 \*\*\*

age 0.81376 0.40570 2.006 0.04601 \*

weight -2.83261 1.81766 -1.558 0.12047

height -0.11466 0.46633 -0.246 0.80599

neck -1.15582 0.57264 -2.018 0.04467 \*

chest -0.14488 0.87021 -0.166 0.86792

abdomen 10.29781 0.97225 10.592 < 2e-16 \*\*\*

hip -1.35104 1.03729 -1.302 0.19401

thigh 1.30382 0.76738 1.699 0.09061 .

knee 0.03364 0.59752 0.056 0.95516

ankle 0.30150 0.37731 0.799 0.42505

biceps 0.55078 0.52117 1.057 0.29166

forearm 0.92091 0.40272 2.287 0.02309 \*

wrist -1.54462 0.49775 -3.103 0.00215 \*\*

---

Signif. codes: 0 ‘\*\*\*’ 0.001 ‘\*\*’ 0.01 ‘\*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 4.309 on 238 degrees of freedom

Multiple R-squared: 0.7486, Adjusted R-squared: 0.7348

F-statistic: 54.5 on 13 and 238 DF, p-value: < 2.2e-16

> mean(Bodyfat.scale$bodyfat)

[1] 19.15079

To run ridge regression, we first need to choose an optimal value for the penalty parameter . The size of reasonable  values varies vastly from one ridge model to the next so using some form of automated selection method like cross-validation to help find one is a good idea. Another approach is use the effective degrees of freedom of the model which is given by the trace (sum of the diagonal elements) of the matrix

which as we can see is a function of . Note that when  = 0, i.e. OLS, this matrix is the *hat matrix* whose trace is always *k*. To fit ridge models and choose an appropriate  we will first use the function lm.ridge from the **MASS**package.

> args(lm.ridge)

function (formula, data, subset, na.action, lambda = 0, model = FALSE, x = FALSE, y = FALSE, contrasts = NULL, ...)

The args command is an easy way to see what arguments a function takes to run. Of course some functions are quite complex so using the command >?lm.ridge will bring up the help file with additional details on the arguments and generally simple examples of the functions use. We first we will use a wide range of  values and let the built-in optimal selection algorithms choose good candidates.

> bodyfat.ridge = lm.ridge(bodyfat~.,data=Bodyfat.scale, lambda=seq(0,1000,.1))

> select(bodyfat.ridge)

modified HKB estimator is 1.664046

modified L-W estimator is 3.91223

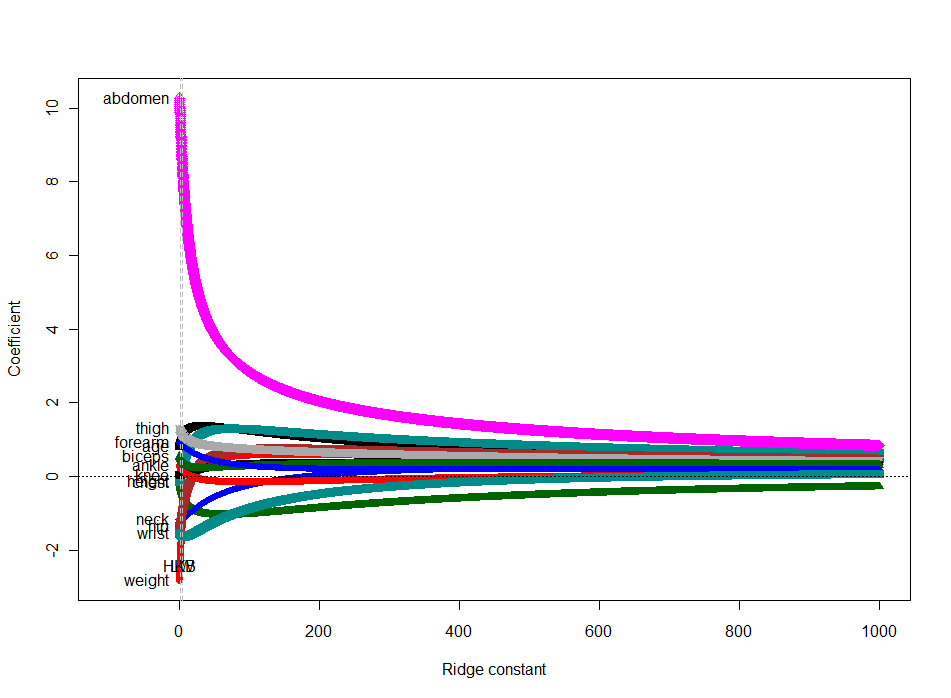
smallest value of GCV at 1.1 🡨 generalized cross-validation choice for 

Using the ridge function from the genridgepackage along with some different plotting features we can see the shrinkage in the parameter estimates.

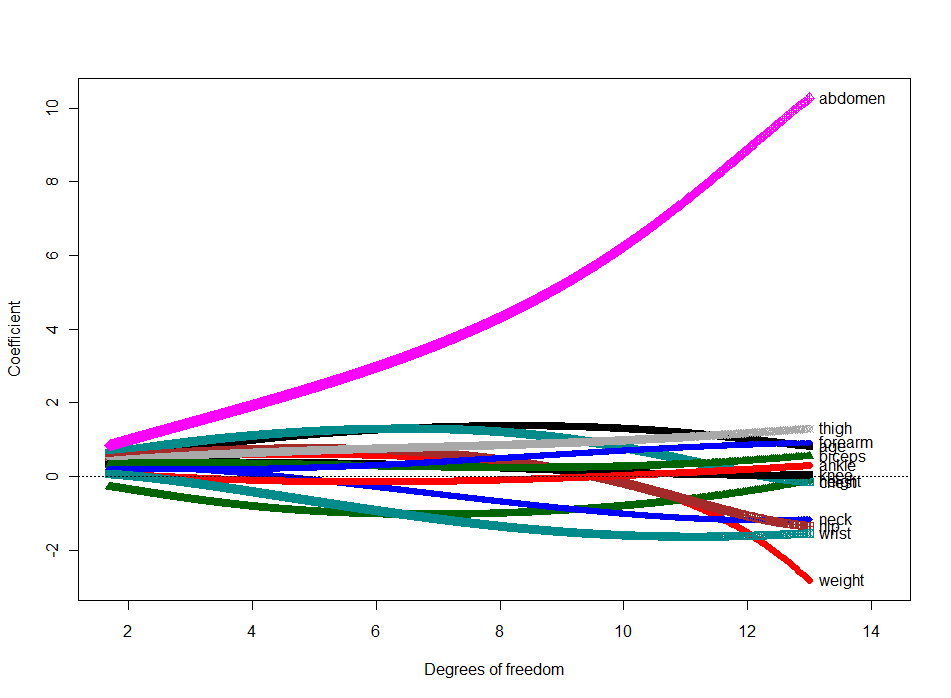
> bodyfat.ridge2 = ridge(Bodyfat.scale$bodyfat,bodyfat.Xs,

lambda=seq(0,1000,.1))

> traceplot(bodyfat.ridge2)



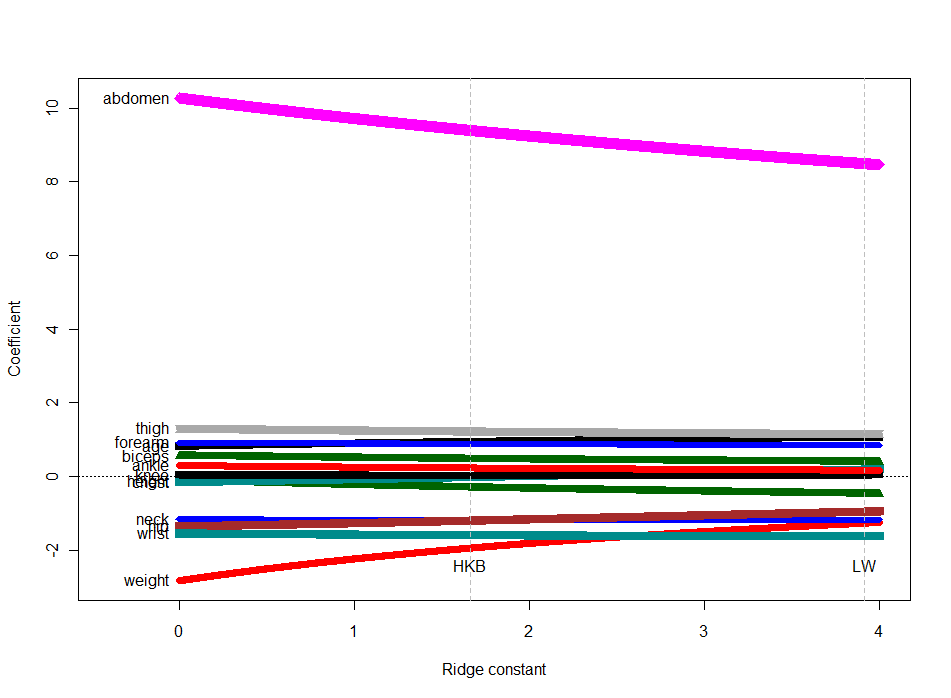
> traceplot(bodyfat.ridge2,X=”df”)



We can narrow the range on the  choices to take a closer look at the optimal shrinkage parameter values.

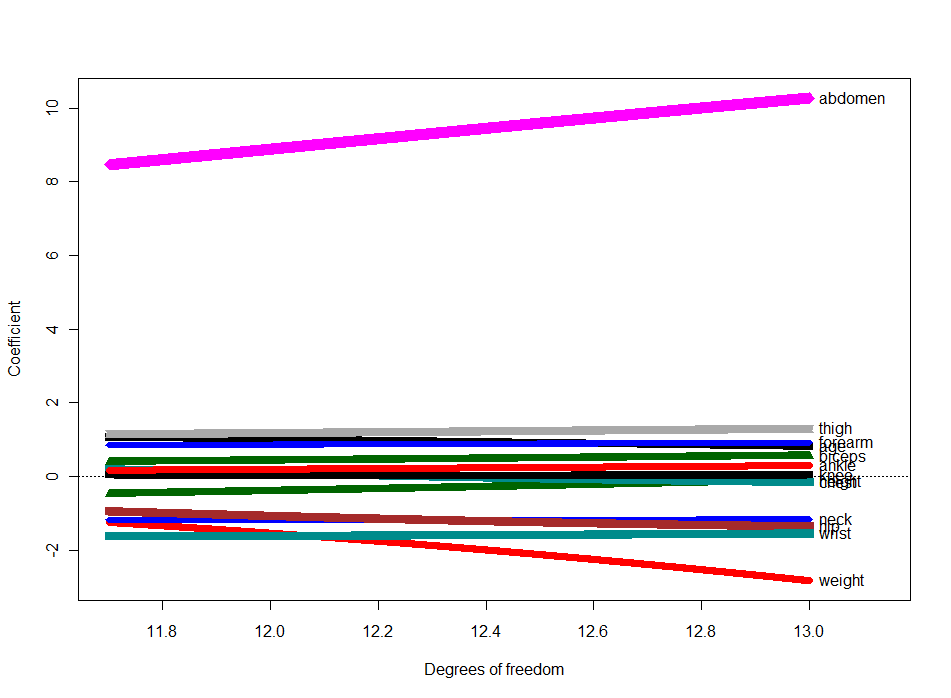
> bodyfat.ridge3 = ridge(bodyfat,bodyfat.Xs,lambda=seq(0,4,.001))

> traceplot(bodyfat.ridge3)



There will not be much shrinkage with values this small.

> traceplot(bodyfat.ridge3,X=”df”)

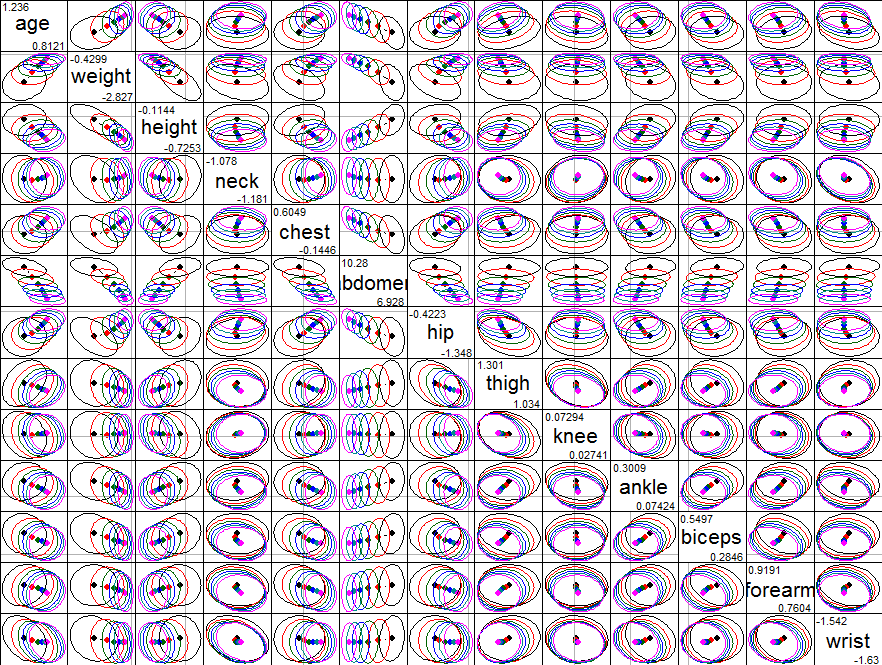


> bodyfat.xs = Bodyfat.scale[,-1]

> bodyfat.y = Bodyfat.scale[,1]

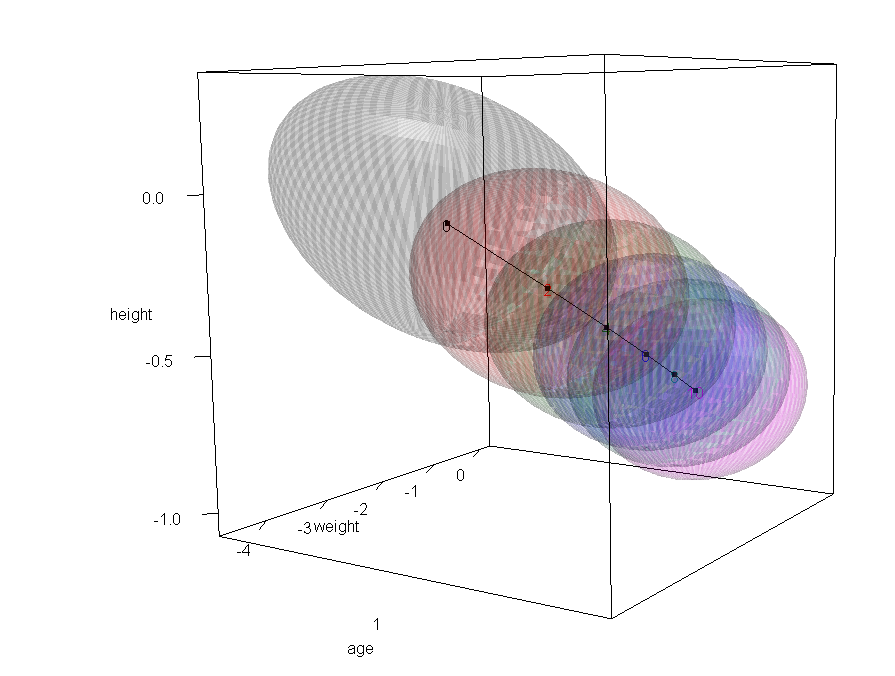
> bodyfat.ridge = ridge(bodyfat.y,bodyfat.xs,lambda=seq(0,10,2))

> pairs(bodyfat.ridge)



This plot shows the shrinkage in the estimated coefficients occurring as lambda increases from 0 to 10 by increments of 2. Most of the “shrinkage” occurs in the terms: age, weight, height, and abdomen.

> plot3d(bodyfat.ridge,variables=1:3)



A 3-D look at the shrinkage of the coefficients of age, weight, and height. This plot requires you download and load the package **rgl**.

> plot3d(bodyfat.ridge,variables=c(2,3,6)) # this is also interesting

Fit ridge regression model using the HKB optimal value for 

> bodyfat.ridge4 = lm.ridge(bodyfat~.,data=Bodyfat.scale,lambda=1.66)

> attributes(bodyfat.ridge4)

$names

[1] "coef" "scales" "Inter" "lambda" "ym" "xm" "GCV"   
[8] "kHKB" "kLW"

$class

[1] "ridgelm"

Compare the OLS coefficients to the ridge coefficients side-by-side.

> cbind(coef(bodyfat.scaled),coef(bodyfat.ridge4))

[,1] [,2]

(Intercept) 19.15079365 19.150793651

age 0.81375776 0.941990017

weight -2.83261161 -1.944588412

height -0.11466232 -0.313666216

neck -1.15582043 -1.182543415

chest -0.14487500 -0.009673795

abdomen 10.29780784 9.416114940

hip -1.35104126 -1.197685531

thigh 1.30382219 1.227323244

knee 0.03363573 0.027303926

ankle 0.30149592 0.235719800

biceps 0.55078084 0.461889816

forearm 0.92090523 0.891302127

wrist -1.54461619 -1.592169696

The decreases in the parameter estimates for most notably abdomen and weight allow for nominal increases in some of the parameter estimates for the other predictors.

Unfortunately the ridge regression routine in this package does not allow for easy extraction of the fitted values and residuals from the fit. It is not hard to write to a simple function that will return the fitted values from the lm.ridge fit.

ridgefitted = function(fit,xmat) {

p = length(coef(fit))

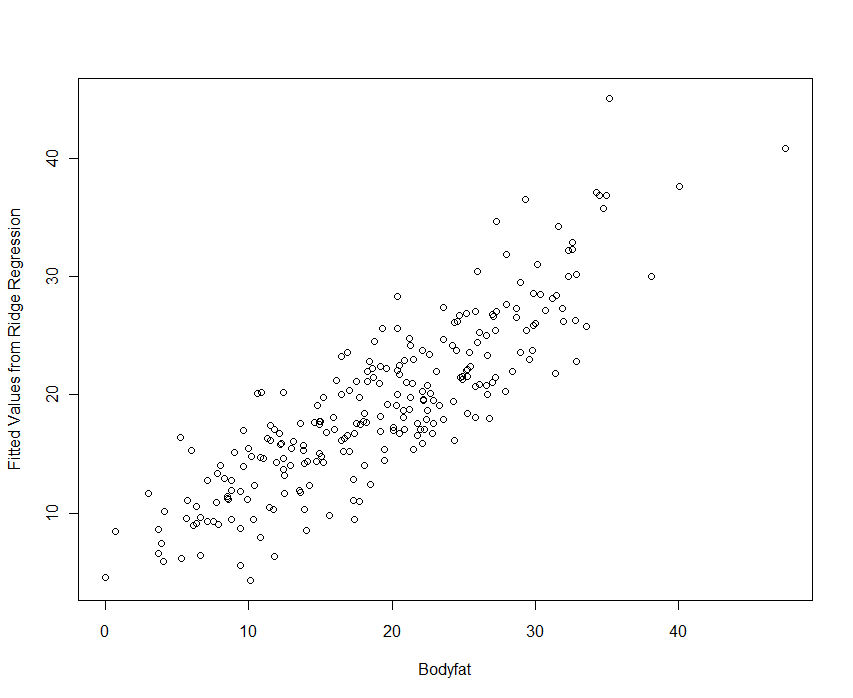
fitted = coef(fit)[1] + xmat%\*%coef(fit)[2:p]

fitted

}

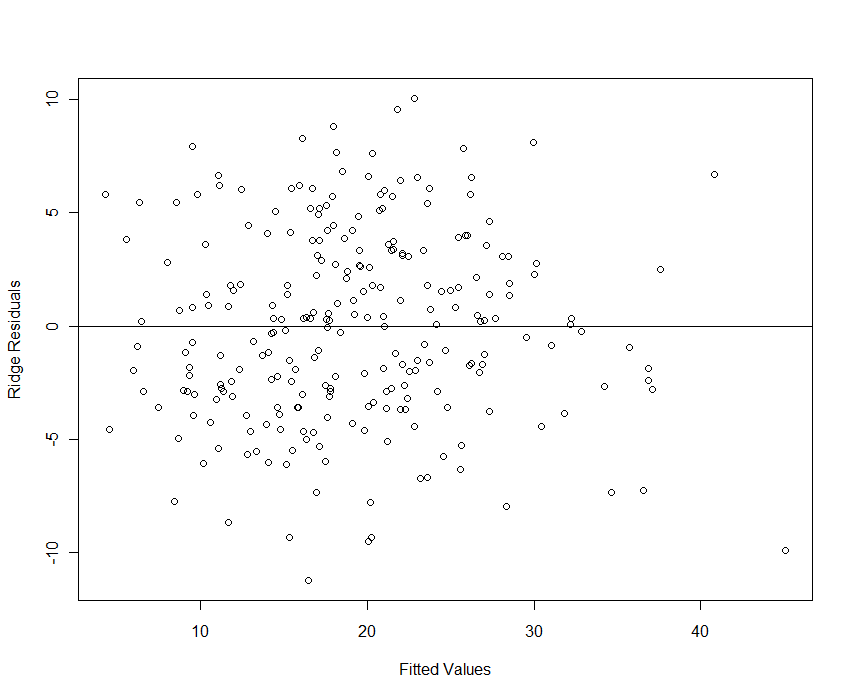
> ridge4fit = ridgefitted(bodyfat.ridge4,bodyfat.Xs)

> plot(Bodyfat$bodyfat,ridge4fit,xlab="Bodyfat",  
ylab="Fitted Values from Ridge Regression")



> ridge4resid = Bodyfat$bodyfat - ridge4fit

> plot(ridge4fit,ridge4resid,xlab="Fitted Values",ylab="Ridge Residuals")



**Ridge Regression using glmnet() (Friedman, Hastie, Tibshirani 2013)**

The glmnet package contains the function glmnet()which can be used to fit both the ridge regression and the Lasso model discussed in the next section. This function has a natural predict() function so obtaining fitted values and making predictions is easier than in the functions used above.

We again return to the body fat example. The author’s also present another example of ridge regression in Lab 2 of Chapter 6 beginning on pg. 251 using data on baseball hitters and their salaries.

The function glmnet()does not use standard formula conventions for developing models. Instead we form the a model matrix () that contains the predictors/terms as columns and the response vector , and use them as arguments to the function. The columns of must be numeric, so any categorical variables will need to converted to dummy variables 1st. This is easily achieved by using the model.matrix()function. For example for the Saratoga, NY homes data with nominal predictors such as Waterfront, Heat Type, etc. already converted to factors we would need to create the model matrix using the command below:

> X = model.matrix(Price~.,data=Saratoga)

**Example 8.2 – Seat Position**  
Car drivers like to adjust the seat position for their own comfort. Car designers would find it helpful to know where different drivers will position the seat depending on their size and age. Researchers at the HuMoSim laboratory at the University of Michigan collected data on 38 drivers. The dataset contains the following variables:

* Age - age in years
* Weight - weight in lbs
* HtShoes - height in shoes in cm
* Ht - height bare foot in cm, Seated - seated height in cm
* Arm - lower arm length in cm
* Thigh - thigh length in cm
* Leg – lower leg length in cm
* Hipcenter - horizontal distance of the midpoint of the hips from a fixed location in the car in mm.

> library(faraway) 🡨 you need to install it first!

> names(seatpos)

[1] "Age" "Weight" "HtShoes" "Ht" "Seated" "Arm" "Thigh" "Leg"

[9] "hipcenter"

> summary(seatpos)

Age Weight HtShoes Ht Seated Arm

Min. :19.00 Min. :100.0 Min. :152.8 Min. :150.2 Min. : 79.40 Min. :26.00

1st Qu.:22.25 1st Qu.:131.8 1st Qu.:165.7 1st Qu.:163.6 1st Qu.: 85.20 1st Qu.:29.50

Median :30.00 Median :153.5 Median :171.9 Median :169.5 Median : 89.40 Median :32.00

Mean :35.26 Mean :155.6 Mean :171.4 Mean :169.1 Mean : 88.95 Mean :32.22

3rd Qu.:46.75 3rd Qu.:174.0 3rd Qu.:177.6 3rd Qu.:175.7 3rd Qu.: 91.62 3rd Qu.:34.48

Max. :72.00 Max. :293.0 Max. :201.2 Max. :198.4 Max. :101.60 Max. :39.60

Thigh Leg hipcenter

Min. :31.00 Min. :30.20 Min. :-279.15

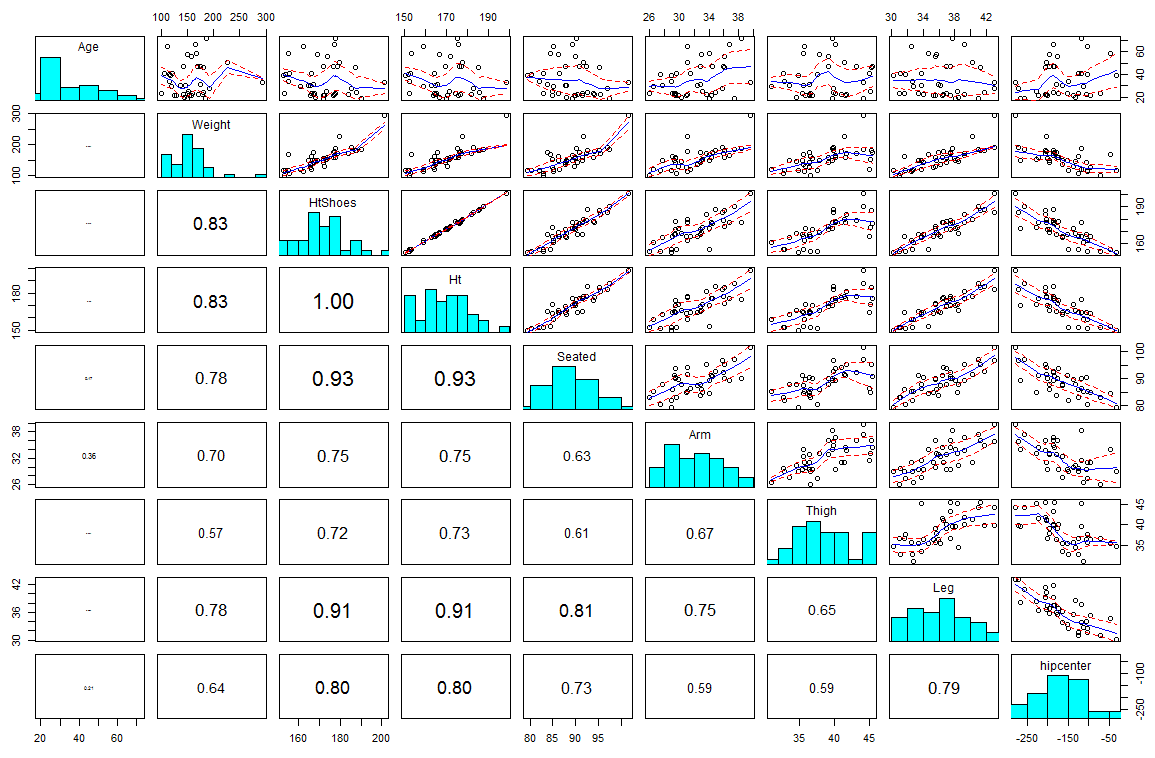
1st Qu.:35.73 1st Qu.:33.80 1st Qu.:-203.09

Median :38.55 Median :36.30 Median :-174.84

Mean :38.66 Mean :36.26 Mean :-164.88

3rd Qu.:41.30 3rd Qu.:38.33 3rd Qu.:-119.92

Max. :45.50 Max. :43.10 Max. : -30.95

All variables are numeric so we do not need to worry about using the model.matrix command before fitting models using the glmnet function.  
> pairs.plus(seatpos)  


**Comments:**

> hip.ols = lm(hipcenter~.,data=seatpos)

> summary(hip.ols)

lm(formula = hipcenter ~ ., data = seatpos)

Residuals:

Min 1Q Median 3Q Max

-73.827 -22.833 -3.678 25.017 62.337

Coefficients:

Estimate Std. Error t value Pr(>|t|)

(Intercept) 436.43213 166.57162 2.620 0.0138 \*

Age 0.77572 0.57033 1.360 0.1843

Weight 0.02631 0.33097 0.080 0.9372

HtShoes -2.69241 9.75304 -0.276 0.7845

Ht 0.60134 10.12987 0.059 0.9531

Seated 0.53375 3.76189 0.142 0.8882

Arm -1.32807 3.90020 -0.341 0.7359

Thigh -1.14312 2.66002 -0.430 0.6706

Leg -6.43905 4.71386 -1.366 0.1824

---

Signif. codes: 0 ‘\*\*\*’ 0.001 ‘\*\*’ 0.01 ‘\*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 37.72 on 29 degrees of freedom

Multiple R-squared: 0.6866, Adjusted R-squared: 0.6001

F-statistic: 7.94 on 8 and 29 DF, p-value: 1.306e-05

> VIF(hip.ols)

Variance Inflation Factor Table

Variable VIF Rsquared

Age Age 1.997931 0.4994823

Weight Weight 3.647030 0.7258043

HtShoes HtShoes 307.429378 0.9967472

Ht Ht 333.137832 0.9969982

Seated Seated 8.951054 0.8882813

Arm Arm 4.496368 0.7775983

Thigh Thigh 2.762886 0.6380596

Leg Leg 6.694291 0.8506190

As stated above it is imperative when performing ridge regression (or any other *regularized* regression method) that we scale the terms to have mean 0 and variance 1. We will form a new data frame in R containing the seat position data with the predictors scaled.

> X = scale(seatpos[,-9])

> summary(X)

Age Weight HtShoes Ht Seated

Min. :-1.0582 Min. :-1.55477 Min. :-1.66748 Min. :-1.69012 Min. :-1.93695

1st Qu.:-0.8467 1st Qu.:-0.66743 1st Qu.:-0.50810 1st Qu.:-0.49307 1st Qu.:-0.76091

Median :-0.3425 Median :-0.05957 Median : 0.05028 Median : 0.03721 Median : 0.09071

Mean : 0.0000 Mean : 0.00000 Mean : 0.00000 Mean : 0.00000 Mean : 0.00000

3rd Qu.: 0.7474 3rd Qu.: 0.51335 3rd Qu.: 0.55484 3rd Qu.: 0.59434 3rd Qu.: 0.54187

Max. : 2.3904 Max. : 3.83913 Max. : 2.67401 Max. : 2.62373 Max. : 2.56446

Arm Thigh Leg

Min. :-1.8436 Min. :-1.97556 Min. :-1.78135

1st Qu.:-0.8055 1st Qu.:-0.75620 1st Qu.:-0.72367

Median :-0.0640 Median :-0.02716 Median : 0.01082

Mean : 0.0000 Mean : 0.00000 Mean : 0.00000

3rd Qu.: 0.6701 3rd Qu.: 0.68252 3rd Qu.: 0.60577

Max. : 2.1902 Max. : 1.76639 Max. : 2.00866

> var(X) # Variance-covariance matrix of scaled data is the correlation matrix

Age Weight HtShoes Ht Seated Arm Thigh Leg

Age 1.00000000 0.08068523 -0.07929694 -0.09012812 -0.1702040 0.3595111 0.09128584 -0.04233121

Weight 0.08068523 1.00000000 0.82817733 0.82852568 0.7756271 0.6975524 0.57261442 0.78425706

HtShoes -0.07929694 0.82817733 1.00000000 0.99814750 0.9296751 0.7519530 0.72486225 0.90843341

Ht -0.09012812 0.82852568 0.99814750 1.00000000 0.9282281 0.7521416 0.73496041 0.90975238

Seated -0.17020403 0.77562705 0.92967507 0.92822805 1.0000000 0.6251964 0.60709067 0.81191429

Arm 0.35951115 0.69755240 0.75195305 0.75214156 0.6251964 1.0000000 0.67109849 0.75381405

Thigh 0.09128584 0.57261442 0.72486225 0.73496041 0.6070907 0.6710985 1.00000000 0.64954120

Leg -0.04233121 0.78425706 0.90843341 0.90975238 0.8119143 0.7538140 0.64954120 1.00000000

> seatpos.scale = data.frame(cbind(hip=seatpos$hipcenter,X))

> names(seatpos.scale)

[1] "hip" "Age" "Weight" "HtShoes" "Ht" "Seated" "Arm" "Thigh" "Leg"

> hip.ols = lm(hip~.,data=seatpos.scale)

> summary(hip.ols)

Call:

lm(formula = hip ~ ., data = seatpos.scale)

Residuals:

Min 1Q Median 3Q Max

-73.827 -22.833 -3.678 25.017 62.337

Coefficients:

Estimate Std. Error t value Pr(>|t|)

(Intercept) -164.8849 6.1190 -26.946 <2e-16 \*\*\*

Age 11.9218 8.7653 1.360 0.184

Weight 0.9415 11.8425 0.080 0.937

HtShoes -30.0157 108.7294 -0.276 0.784

Ht 6.7190 113.1843 0.059 0.953

Seated 2.6324 18.5529 0.142 0.888

Arm -4.4775 13.1494 -0.341 0.736

Thigh -4.4296 10.3076 -0.430 0.671

Leg -21.9165 16.0445 -1.366 0.182

---

Signif. codes: 0 ‘\*\*\*’ 0.001 ‘\*\*’ 0.01 ‘\*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 37.72 on 29 degrees of freedom

Multiple R-squared: 0.6866, Adjusted R-squared: 0.6001

F-statistic: 7.94 on 8 and 29 DF, p-value: 1.306e-05

> VIF(hip.ols)

Variance Inflation Factor Table

Variable VIF Rsquared

Age Age 1.997931 0.4994823

Weight Weight 3.647030 0.7258043

HtShoes HtShoes 307.429378 0.9967472

Ht Ht 333.137832 0.9969982

Seated Seated 8.951054 0.8882813

Arm Arm 4.496368 0.7775983

Thigh Thigh 2.762886 0.6380596

Leg Leg 6.694291 0.8506190

Rescaling the *X’s* does not change the model performance in any way. The p-values, R2, RSS, VIF’s, etc. are all the same. The only changes are the estimated regression coefficients. We now consider fitting a ridge regression model to these data.

> X = model.matrix(hip~.,data=seatpos.scale)[,-1] 🡨 terms without intercept.

> y = seatpos.scale$hip

> library(glmnet)

> grid = 10^seq(10,-2,length=100) <- set up a wide range of 100  values

> grid  
 [1] 10000000000.00000000000 7564633275.54629039764 5722367659.35021972656 4328761281.08306121826

[5] 3274549162.87773180008 2477076355.99171400070 1873817422.86038684845 1417474162.92680764198

[9] 1072267222.01032531261 811130830.78968894482 613590727.34131634235 464158883.36127728224

[13] 351119173.42151272297 265608778.29466840625 200923300.25650459528 151991108.29529330134

[17] 114975699.53977356851 86974900.26177833974 65793322.46575682610 49770235.64332114160

[21] 37649358.06792471558 28480358.68435804918 21544346.90031882375 16297508.34620643407

[25] 12328467.39442065917 9326033.46883219853 7054802.31071864534 5336699.23120630160

[29] 4037017.25859654974 3053855.50883341255 2310129.70008315798 1747528.40000768285

[33] 1321941.14846602874 1000000.00000000000 756463.32755462907 572236.76593502203

[37] 432876.12810830615 327454.91628777247 247707.63559917090 187381.74228603829

[41] 141747.41629268049 107226.72220103232 81113.08307896872 61359.07273413175

[45] 46415.88833612773 35111.91734215128 26560.87782946684 20092.33002565046

[49] 15199.11082952933 11497.56995397736 8697.49002617783 6579.33224657568

[53] 4977.02356433211 3764.93580679246 2848.03586843580 2154.43469003188

[57] 1629.75083462064 1232.84673944207 932.60334688322 705.48023107186

[61] 533.66992312063 403.70172585965 305.38555088334 231.01297000832

[65] 174.75284000077 132.19411484660 100.00000000000 75.64633275546

[69] 57.22367659350 43.28761281083 32.74549162878 24.77076355992

[73] 18.73817422860 14.17474162927 10.72267222010 8.11130830790

[77] 6.13590727341 4.64158883361 3.51119173422 2.65608778295

[81] 2.00923300257 1.51991108295 1.14975699540 0.86974900262

[85] 0.65793322466 0.49770235643 0.37649358068 0.28480358684

[89] 0.21544346900 0.16297508346 0.12328467394 0.09326033469

[93] 0.07054802311 0.05336699231 0.04037017259 0.03053855509

[97] 0.02310129700 0.01747528400 0.01321941148 0.01000000000

> ridge.mods = glmnet(X,y,alpha=0,lambda=grid) 🡨 alpha = 0 for ridge  
 alpha = 1 for Lasso

> dim(coef(ridge.mods))🡨 has 100 columns of parameter estimates, one for   
[1] 9 100 each lambda in our sequence.

> coef(ridge.mods)[,1] 🡨

> coef(ridge.mods)[,1]

(Intercept) Age Weight HtShoes

-164.88486842105254482 0.00000007202948846 -0.00000022480074310 -0.00000027965991110

Ht Seated Arm Thigh

-0.00000028047825668 -0.00000025672013536 -0.00000020540841817 -0.00000020755224027

Leg

-0.00000027635006516  
  
When lambda is very large we see that the parameter estimates are near 0 and the intercept estimate is approximately equal to the mean of the response .  
  
> mean(y)   
[1] -164.8849

> coef(ridge.mods)[,100] 🡨 

(Intercept) Age Weight HtShoes Ht Seated Arm

-164.8848684 11.7788069 0.9953047 -23.0463404 -0.2996308 2.4814234 -4.4305306

Thigh Leg

-4.2792243 -21.9175012

When lambda is near 0, we see that the coefficients do not differ much from the OLS regression parameter estimates which are shown below.

> coef(hip.ols)

(Intercept) Age Weight HtShoes Ht Seated Arm

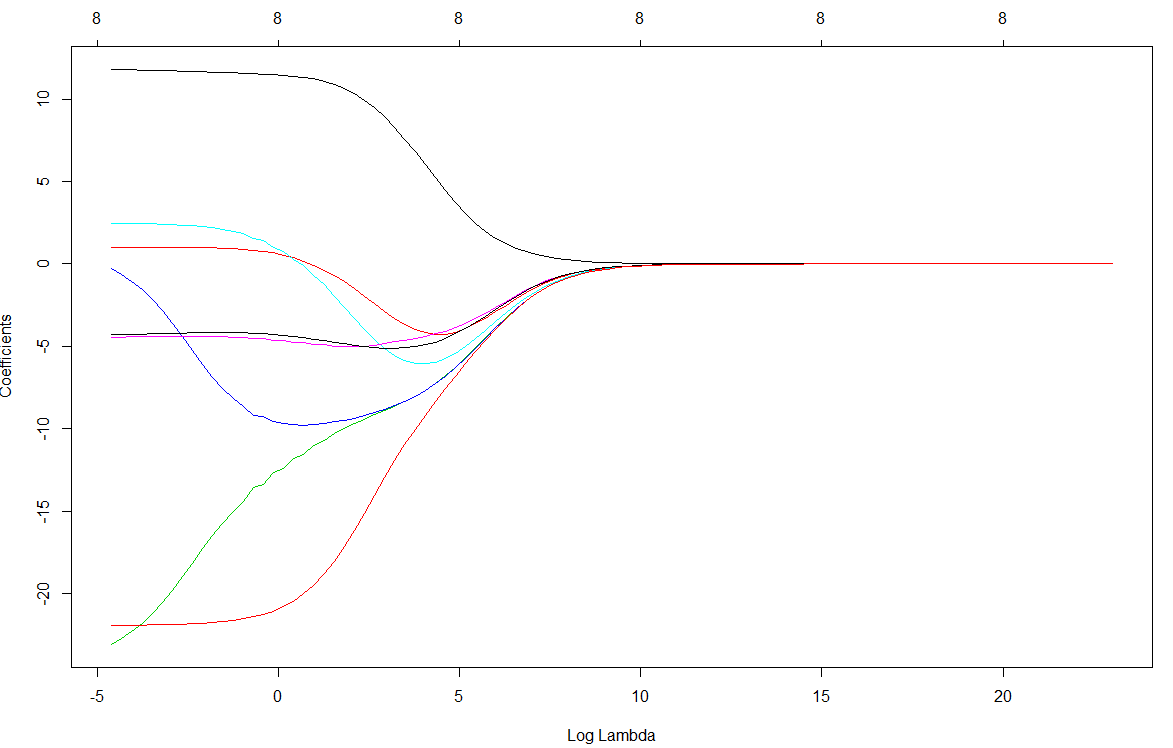
-164.8848684 11.9218052 0.9415132 -30.0156578 6.7190129 2.6323517 -4.4775359

Thigh Leg

-4.4295690 -21.9165049

We can see this shrinkage of the coefficients graphically by plotting the results.

> plot(ridge.mods,xvar="lambda")



What value of  should we use to obtain the “best” ridge regression model? In the code below we form a train data set consisting of 75% of the original data set and use the remaining cases as test cases. We then look at the mean MSEP for various choices of  by setting the parameter s =  in the glmnet()function call.

> train = sample(1:n,floor(n\*.75),replace=F)

> train

[1] 15 23 4 28 20 2 9 35 21 25 22 31 34 18 32 7 16 27 26 36 29 5 8 19 12 13 17 11

> test = (-train)

> ridge.mod = glmnet(X[train,],y[train],alpha=0,lambda=grid)

> ridge.pred = predict(ridge.mod,s=1000,newx=X[test,]) 🡨 

> MSEP = mean((ridge.pred-y[test])^2)

> MSEP

[1] 1991.91455  
  
> ridge.pred = predict(ridge.mod,s=100,newx=X[test,])🡨

> MSEP = mean((ridge.pred-y[test])^2)

> MSEP

[1] 1446.587399  
  
  
> ridge.pred = predict(ridge.mod,s=10,newx=X[test,])🡨 

> MSEP = mean((ridge.pred-y[test])^2)

> MSEP

[1] 1993.2063  
> ridge.pred = predict(ridge.mod,s=150,newx=X[test,]) 🡨 

> MSEP = mean((ridge.pred-y[test])^2)

> MSEP

[1] 1440.651072

> ridge.pred = predict(ridge.mod,s=200,newx=X[test,]) 🡨

> MSEP = mean((ridge.pred-y[test])^2)

> MSEP

[1] 1464.634606

> ridge.pred = predict(ridge.mod,s=125,newx=X[test,]) 🡨

> mean((ridge.pred-y[test])^2)

[1] 1438.089147

It appears a value between 125 and 150 appears optimal for **this** particular train/test set combination. What if we use different train/test sets?

> set.seed(1)

> train = sample(n,floor(n\*p))

> ridge.mod glmnet(X[train,],y[train],lambda=grid,alpha=0)  
  
> ridge.pred = predict(ridge.mod,s=200,newx=X[test,])

> mean((ridge.pred-y[test])^2)

[1] 1001.191857

> ridge.pred = predict(ridge.mod,s=100,newx=X[test,])

> mean((ridge.pred-y[test])^2)

[1] 883.9791728

> ridge.pred = predict(ridge.mod,s=50,newx=X[test,])

> mean((ridge.pred-y[test])^2)

[1] 849.9181088

> ridge.pred = predict(ridge.mod,s=25,newx=X[test,])

> mean((ridge.pred-y[test])^2)

[1] 861.3195607

Now it appears that the “optimal”  is somewhere between 25 and 100!

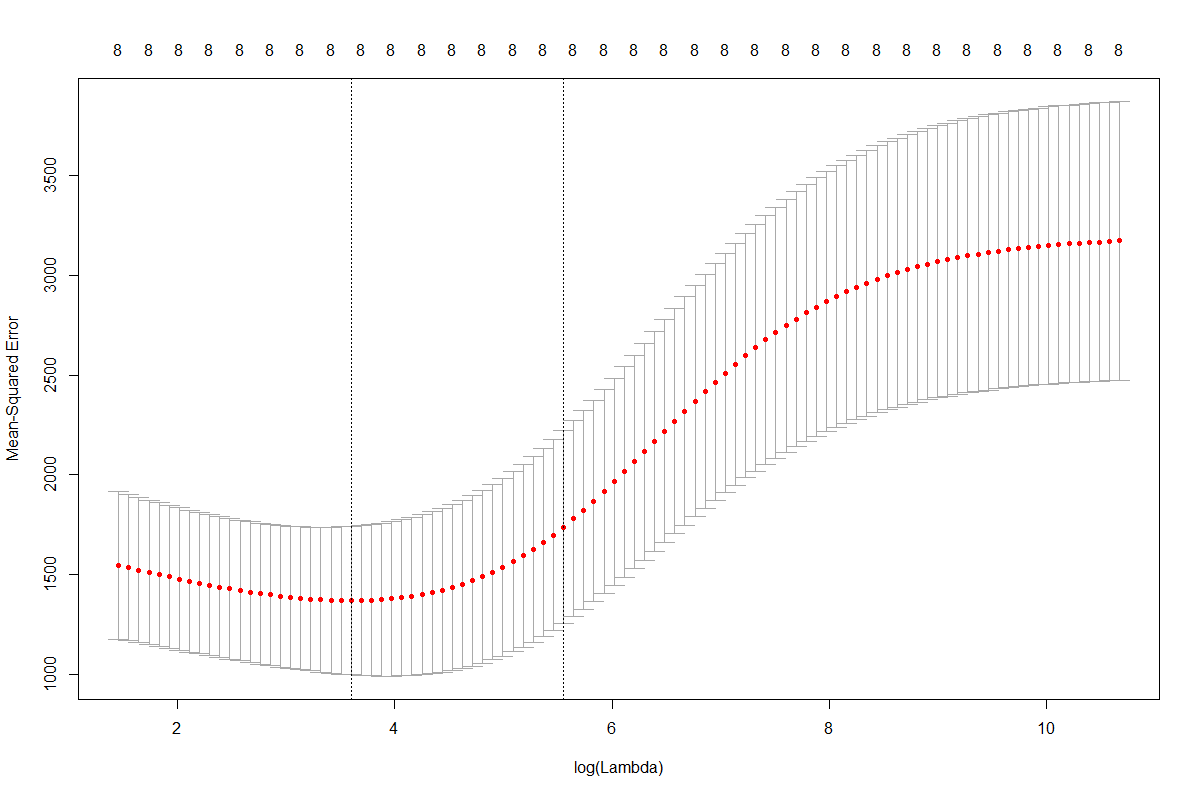
We can use cross-validation to choose an “optimal”  for prediction purposes. The function cv.glmnet()uses 10-fold cross-validation to find an optimal value for .

> cv.out = cv.glmnet(X[train,],y[train],alpha=0)

Warning message:

Option grouped=FALSE enforced in cv.glmnet, since < 3 observations per fold

This dataset is too small to use 10-fold cross-validation as the sample size is *n = 38*!

> plot(cv.out)  


> cv.out$lambda.min

[1] 50.74707557  
> bestlam = cv.out$lambda.min

> ridge.best = glmnet(X[train,],y[train],alpha=0,lambda=bestlam)

> ridge.pred = predict(ridge.best,newx=X[test,])

> mean((ridge.pred-y[test])^2)

[1] 850.2535287

> hip.ols = lm(hip~.,data=seatpos.scale[train,])}

> cbind(coef(ridge.best),coef(hip.ols))

9 x 2 sparse Matrix of class "dgCMatrix"

s0

(Intercept) -166.104009234 -164.933702436

Age 6.529599123 13.992278127

Weight -3.339435798 7.855210947

HtShoes -8.367553205 -31.447579652

Ht -8.372073998 -1.527922822

Seated -5.541424573 8.084809404

Arm -6.227976511 -13.720781271

Thigh -4.379715015 -0.836875912

Leg -10.781880429 -20.495083999

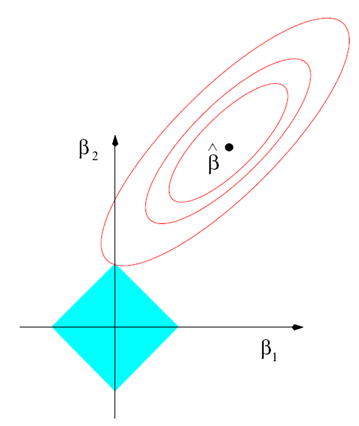
We can see that the coefficients for the highly correlated height related predictors change substantially when using ridge regression vs. OLS regression. Other coefficients change substantially as well.

**8.2 - The Lasso**

The lasso is another shrinkage method like ridge, but uses an L1-norm based penalty.

The parameter estimates are chosen according to the following

Here *t* > 0 is the complexity parameter that controls the amount of shrinkage, the smaller *t* the greater the amount of shrinkage. As with ridge regression, the intercept is not included in the shrinkage and will be estimated as the mean of the response. If *t* is chosen larger than then there will be no shrinkage and the lasso estimates will be the same as the OLS estimates. If then the OLS estimates will be shrunk by about 50%, however this is not to say that . The shrinkage can result in some parameters being zeroed (which ridge will not ever do), essentially dropping the associated predictor from the model as the figure below shows. In the diagram below, the lasso estimate for .



Usual OLS Estimate =

Contours of the OLS criterion

Lasso regression estimate =

We return again to the body fat example and look at the use of the Lasso to build a model for the percent body fat.

> X = model.matrix(bodyfat~.,data=Bodyfat.scale)[,-1]

> y = Bodyfat.scale$bodyfat

> n = nrow(Bodyfat.scale)

> p = .667

> set.seed(1)

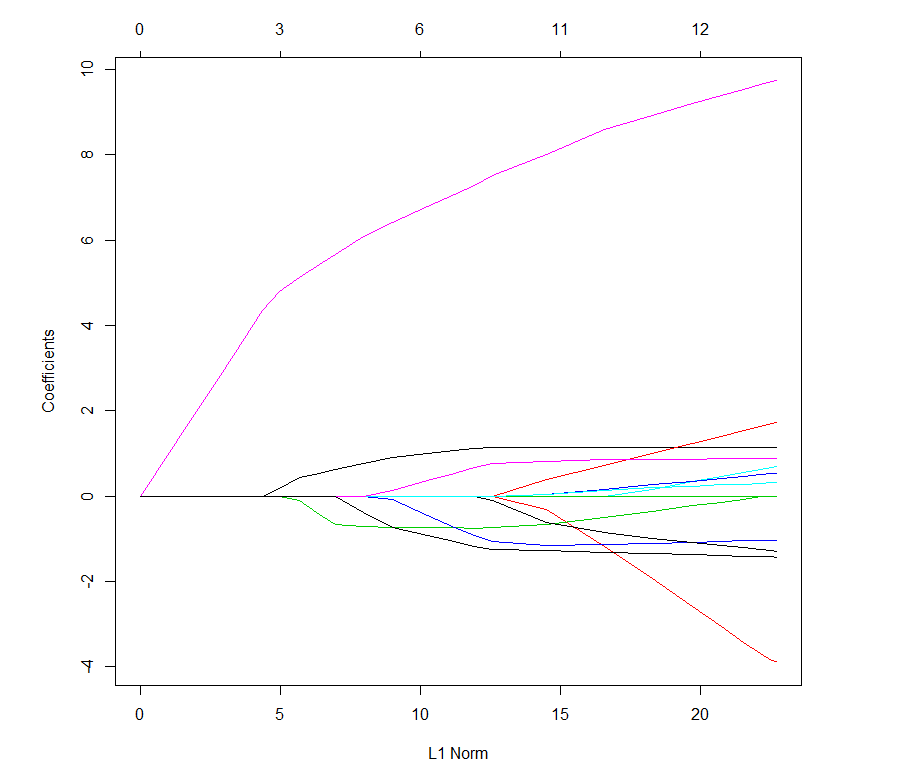
> train = sample(n,floor(n\*p))

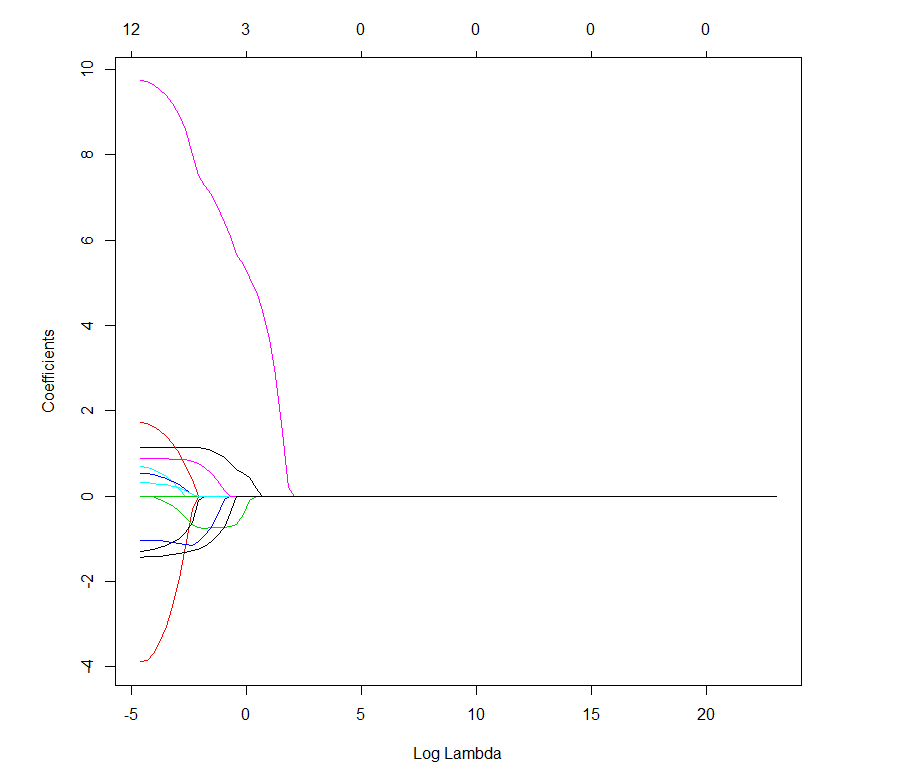
> test = (-train)

> grid = 10^seq(10,-2,length=100)

> lasso.mod = glmnet(X[train,],y[train],alpha=1,lambda=grid)

> plot(lasso.mod)

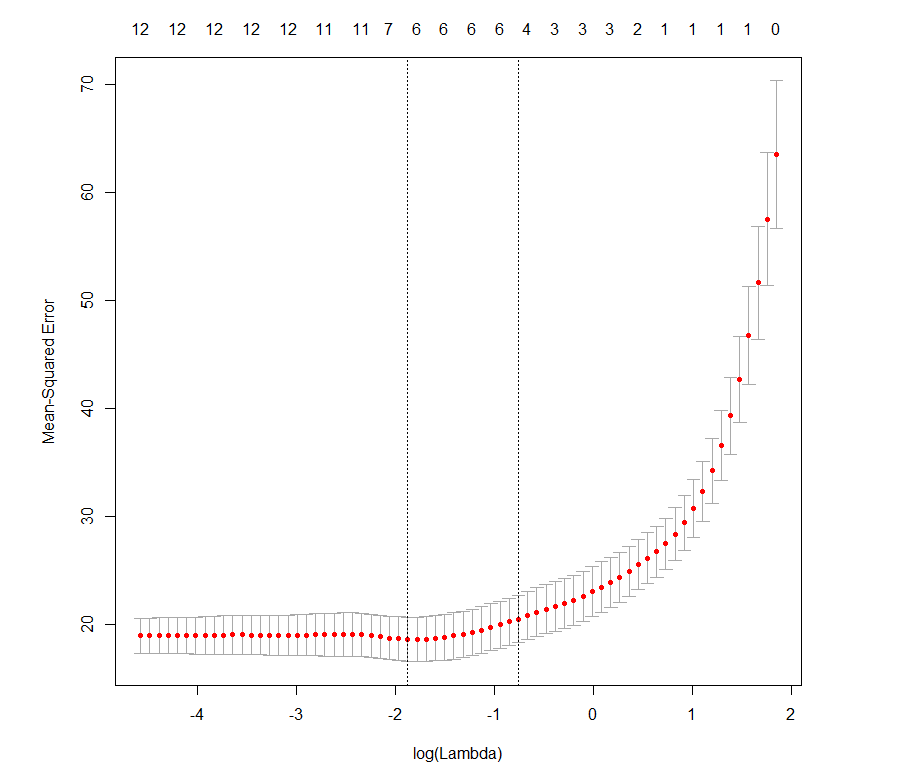
  
> plot(lasso.mod,xvar=”lambda”)



> set.seed(1)

> cv.out = cv.glmnet(X[train,],y[train],alpha=1)

> plot(cv.out)



> bestlam.lasso = cv.out$lambda.min

> bestlam.lasso

[1] 0.1533247

Use test set to obtain and estimate of the MSEP for the Lasso  
===========================================================================================

> lasso.mod = glmnet(X[train,],y[train],alpha=1,lambda=bestlam.lasso)

> lasso.pred = predict(lasso.mod,newx=X[test,])

> MSEP = mean((lasso.pred-y[test])^2)

> MSEP

[1] 23.72193

Use the same 10-fold cross-validation to estimate optimal  for ridge regression. Then estimate the MSEP using the same test data as for the Lasso. Compare the mean MSEP values.

============================================================================================

> set.seed(1)

> cv.out = cv.glmnet(X[train,],y[train],alpha=0)

> bestlam.ridge = cv.out$lambda.min  
> bestlam.ridge

[1] 0.6335397

> ridge.mod = glmnet(X[train,],y[train],alpha=0,lambda=bestlam.ridge)

> ridge.pred = predict(ridge.mod,newx=X[test,])

> MSEP = mean((ridge.pred - y[test])^2)

> MSEP

[1] 26.11665

Comparing the coefficient estimates from Lasso, ridge regression, and OLS. Also compare MSEP for test data.

===============================================================================================

> coef(lasso.mod)

s0

(Intercept) 1.27888249

age 0.08947089

weight .

height -0.28803077

neck -0.39922361

chest .

abdomen 0.67740803

hip .

thigh .

knee .

ankle .

biceps .

forearm 0.34448133

wrist -1.27946216

> coef(ridge.mod)

s0

(Intercept) -4.956145707

age 0.130174182

weight -0.005247158

height -0.310172813

neck -0.452885891

chest 0.159678718

abdomen 0.467277929

hip 0.003963329

thigh 0.189565205

knee 0.057918646

ankle 0.043846187

biceps 0.022254664

forearm 0.348491036

wrist -1.470360498

> temp = data.frame(bodyfat = y[train],X[train,])

> head(temp)

bodyfat age weight height neck chest abdomen hip thigh knee ankle biceps forearm wrist

67 21.5 54 151.50 70.75 35.6 90.0 83.9 93.9 55.0 36.1 21.7 29.6 27.4 17.4

94 24.9 46 192.50 71.75 38.0 106.6 97.5 100.6 58.9 40.5 24.5 33.3 29.6 19.1

144 9.4 23 159.75 72.25 35.5 92.1 77.1 93.9 56.1 36.1 22.7 30.5 27.2 18.2

227 14.8 55 169.50 68.25 37.2 101.7 91.1 97.1 56.6 38.5 22.6 33.4 29.3 18.8

51 10.2 47 158.25 72.25 34.9 90.2 86.7 98.3 52.6 37.2 22.4 26.0 25.8 17.3

222 26.0 54 230.00 72.25 42.5 119.9 110.4 105.5 64.2 42.7 27.0 38.4 32.0 19.6

> ols.mod = lm(bodyfat~.,data=temp)

> coef(ols.mod)

(Intercept) age weight height neck chest abdomen hip thigh

-40.65178764 0.09175572 -0.15569221 0.06515098 -0.41393595 0.10173785 0.92607342 -0.18562568 0.37387418

knee ankle biceps forearm wrist

-0.03849608 0.36585984 0.11606918 0.44247339 -1.54993981

> ols.step = step(ols.mod)

> coef(ols.step)

(Intercept) age weight neck abdomen hip thigh forearm wrist

-24.91558645 0.09187304 -0.10466396 -0.46132959 0.93852739 -0.24119508 0.38608812 0.51997961 -1.33498663

> ols.pred = predict(ols.mod,newdata=Bodyfat[test,])

> MSEP = mean((ols.pred-y[test])^2)

[1] 23.39602

> ols.pred2 = predict(ols.step,newdata=Bodyfat[test,])

> MSEP = mean((ols.pred2-y[test])^2)

[1] 22.8308

For these data we see the three approaches differ in their results. The LASSO zeroes out some coefficients, thus does completely eliminate some terms from the model. Ridge will shrink coefficients down to very near zero, effectively eliminating them, but technically will zero none of them. Stepwise selection in OLS is either in or out, so some get zeroed some don’t, however there is no shrinkage of the estimated coefficients. A good question to ask would be “how do these methods cross-validate for making future predictions?” We can use cross-validation methods to compare these competing models via estimates of the MSEP.

**Monte Carlo Cross-Validation of OLS Regression Models**

> MLR.ssmc = function(fit,p=.667,M=100) {

RMSEP = rep(0,M)

MAEP = rep(0,M)

MAPEP = rep(0,M)

y = fit$model[,1]

x = fit$model[,-1]

data = fit$model

n = nrow(data)

for (i in 1:M) {

ss = floor(n\*p)

sam = sample(1:n,ss,replace=F)

fit2 = lm(formula(fit),data=data[sam,])

ypred = predict(fit2,newdata=x[-sam,])

RMSEP[i] = sqrt(mean((y[-sam]-ypred)^2))

MAEP[i] = mean(abs(y[-sam]-ypred))

yp = ypred[y[-sam]!=0]

As one of the response values in these data is y=0, we need to exclude it to prevent the MAPE from becoming .

ya = y[-sam][y[-sam]!=0]

MAPEP[i]=mean(abs(yp-ya)/ya)

}

cat("RMSEP =",mean(RMSEP)," MAEP=",mean(MAEP)," MAPEP=",mean(MAPEP))

cv = return(data.frame(RMSEP=RMSEP,MAEP=MAEP,MAPEP=MAPEP))

}

**Monte Carlo Cross-Validation of Ridge and Lasso Regression**

> glmnet.ssmc = function(X,y,p=.667,M=100,alpha=1,lambda=1) {

RMSEP = rep(0,M)

MAEP = rep(0,M)

MAPEP = rep(0,M)

n = nrow(X)

for (i in 1:M) {

ss = floor(n\*p)

sam = sample(1:n,ss,replace=F)

fit = glmnet(X[sam,],y[sam],lambda=lambda,alpha=alpha)

ypred = predict(fit,newx=X[-sam,])

RMSEP[i] = sqrt(mean((y[-sam]-ypred)^2))

MAEP[i] = mean(abs(y[-sam]-ypred))

yp = ypred[y[-sam]!=0]

ya = y[-sam][y[-sam]!=0]

MAPEP[i]=mean(abs(yp-ya)/ya)

}

cat("RMSEP =",mean(RMSEP)," MAEP=",mean(MAEP)," MAPEP=",mean(MAPEP))

cv = return(data.frame(RMSEP=RMSEP,MAEP=MAEP,MAPEP=MAPEP))

}

We can run these functions on the both the body fat data (Example 8.1) and the seat position data (Example 8.2).

> bf.ols = lm(bodyfat~.,data=Bodyfat.scale)

> bf.step = step(bf.ols)

> ols.results = MLR.ssmc(bf.step,M=1000)

RMSEP = 4.412897 MAEP= 3.603178 MAPEP= 0.2906514

> bf.ridge = glmnet(X,y,alpha=0)

> cv.ridge = cv.glmnet(X,y,alpha=0)

> plot(cv.ridge)

> bestlam.ridge = cv.ridge$lambda.min

> bestlam.ridge

[1] 0.7456279

> ridge.results = glmnet.ssmc(X,y,M=1000,alpha=0,lambda=bestlam.ridge)

RMSEP = 4.724888 MAEP= 3.831887 MAPEP= 0.3302092

> bf.lasso = glmnet(X,y,alpha=1)

> cv.lasso = cv.glmnet(X,y,alpha=1)

> bestlam.lasso = cv.lasso$lambda.min

> bestlam.lasso

[1] 0.01463698  
> lasso.results = glmnet.ssmc(X,y,M=1000,alpha=1,lambda=bestlam.lasso)

RMSEP = 4.546967 MAEP= 3.705931 MAPEP= 0.3010099

Here we see that the OLS MLR outperforms both the ridge and LASSO regression results for the body fat data.

Comparing the results for the seat position data we obtain the following.

> sp.ols = lm(hipcenter~.,data=seatpos)

> sp.step = step(sp.ols)

> ols.results = MLR.ssmc(sp.step,M=1000)

RMSEP = 38.32659 MAEP= 31.09246 MAPEP= -0.2697277 🡨 response

**Ridge Regression**  
> X = scale(model.matrix(hipcenter~.,data=seatpos)[,-1])

> y = seatpos$hipcenter

> sp.ridge = glmnet(X,y,alpha=0)

> plot(sp.ridge)

> cv.ridge = cv.glmnet(X,y,alpha=0)

> plot(cv.ridge)

> bestlam.ridge = cv.ridge$lambda.min

> ridge.results = glmnet.ssmc(X,y,M=1000,alpha=0,lambda=bestlam.ridge)

RMSEP = 38.0442 MAEP= 30.74938 MAPEP= -0.2804524

**LASSO Regression**

> sp.lasso = glmnet(X,y,alpha=1)

> plot(sp.lasso)

> cv.lasso = cv.glmnet(X,y,alpha=1)

> plot(cv.lasso)

> bestlam.lasso = cv.lasso$lambda.min

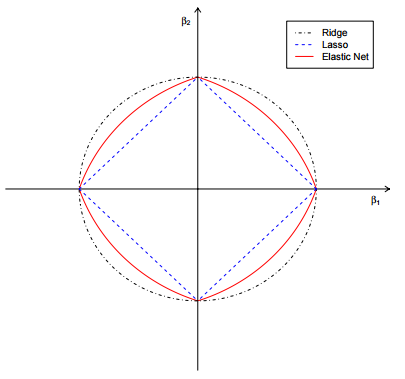
> lasso.results = glmnet.ssmc(X,y,M=1000,alpha=1,lambda=bestlam.lasso)

RMSEP = 40.95677 MAEP= 33.32161 MAPEP= -0.2946756

**8.3 – Elastic Net** (**glmnet** and elasticnet packages in R)

The Elastic Net is another shrinkage method like ridge and LASSO, but uses both the -norm and -norm based penalties on the parameter estimates. Thus the parameter estimates are chosen by minimizing the expression:

It should be clear that is equivalent to ridge regression and is equivalent to the LASSO. If both then we have an Elastic Net. Below is a figure illustrating the differences between the ridge, LASSO, and Elastic Net penalties where and .



The glmnet function allows fitting an elastic net model by choosing alpha . The elastic net criteria above then has and . There is also a package called elasticnet which allows fitting these models. We will continue with our use of the glmnet package for fitting elastic net models in R.

**Elastic Net for Bodyfat Data (Example 8.1)**

> bf.en = glmnet(X,y,alpha=0.5)  
> plot(bf.en)

> cv.en = cv.glmnet(X,y,alpha=0.5)

> plot(cv.en)

> bestlam.en = cv.en$lambda.min

> bestlam.en

[1] 0.06161887

> en.results = glmnet.ssmc(X,y,M=1000,alpha=0.5,lambda=bestlam.en)

RMSEP = 4.535962 MAEP= 3.693045 MAPEP= 0.3062816

We could consider varying the choice for alpha to see we could improve these results. I tried several other choices and found none that would outperform the OLS results for these data or even outperform the LASSO results.

**Elastic Nets for Seat Position Data (Example 8.2)**

> sp.en = glmnet(X,y,alpha=0.5)

> plot(sp.en)

> cv.en = cv.glmnet(X,y,alpha=0.5)

> plot(cv.en)

> bestlam.en = cv.en$lambda.min

> en.results = glmnet.ssmc(X,y,M=1000,alpha=0.5,lambda=bestlam.en)

RMSEP = 39.65151 MAEP= 31.87481 MAPEP= -0.2879419

> sp.en = glmnet(X,y,alpha=0.25)

> cv.en = cv.glmnet(X,y,alpha=0.25)

> bestlam.en = cv.en$lambda.min

> en.results = glmnet.ssmc(X,y,M=1000,alpha=0.25,lambda=bestlam.en)

RMSEP = 39.27006 MAEP= 31.83208 MAPEP= -0.2861834

> cv.en = cv.glmnet(X,y,alpha=0.1)

> bestlam.en = cv.en$lambda.min

> en.results = glmnet.ssmc(X,y,M=1000,alpha=0.1,lambda=bestlam.en)

RMSEP = 38.25156 MAEP= 30.93727 MAPEP= -0.276889

For these data the ridge regression seems best outperforming MLR, LASSO, and Elastic Net regressions.

**Example 8.3 – Permeability of Chemical Compounds (permeability.csv)**

This pharmaceutical data set was used to develop a model for predicting compounds' permeability. In short, permeability is the measure of a molecule's ability to cross a membrane. The body, for example, has notable membranes between the body and brain, known as the blood-brain barrier, and between the gut and body in the intestines. These membranes help the body guard critical regions from receiving undesirable or detrimental substances. For an orally taken drug to be effective in the brain, it first must pass through the intestinal wall and then must pass through the blood-brain barrier in order to be present for the desired neurological target. Therefore, a compound's ability to permeate relevant biological membranes is critically important to understand early in the drug discovery process. Compounds that appear to be effective for a particular disease in research screening experiments, but appear to be poorly permeable may need to be altered in order improve permeability, and thus the compound's ability to reach the desired target. Identifying permeability problems can help guide chemists towards better molecules.

Permeability assays such as PAMPA and Caco-2 have been developed to help measure compounds' permeability (Kansy et al, 1998). These screens are effective at quantifying a compound's permeability, but the assay is expensive labor intensive. Given a sufficient number of compounds that have been screened, we could develop a predictive model for permeability in an attempt to potentially reduce the need for the assay. In this project there were 165 unique compounds; 1107 molecular fingerprints were determined for each. A molecular fingerprint is a binary sequence of numbers that represents the presence or absence of a specific molecular sub-structure. The response is highly skewed, the predictors are sparse (15.5 percent are present), and many predictors are strongly associated.

**IMPORTANT NOTE: For these data and , i.e. , thus we cannot use OLS MLR regression to develop a model for the permeability of these substances.**

> perm = read.table(file.choose(),header=T,sep=”,”)  
> names(perm)

[1] "permeability" "X1" "X2" "X3" "X4" "X5" "X6"

[8] "X7" "X8" "X9" "X10" "X11" "X12" "X13"

[15] "X14" "X15" "X16" "X17" "X18" "X19" "X20"

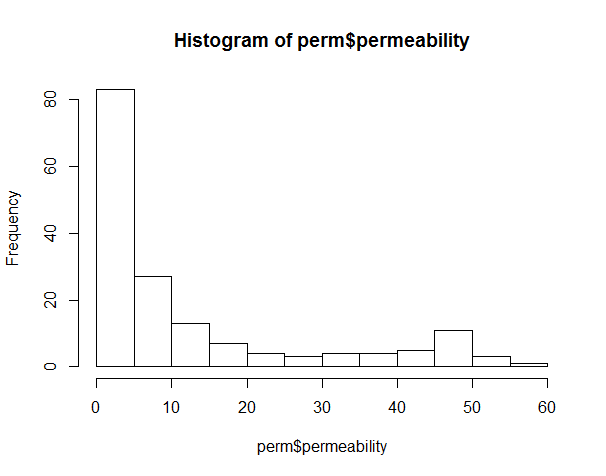
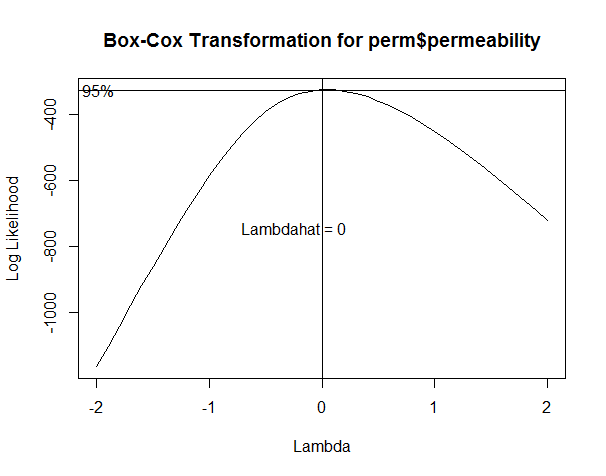
[22] "X21" "X22" "X23" "X24" "X25" "X26" "X27"

[29] "X28" "X29" "X30" "X31" "X32" "X33" "X34"

[36] "X35" "X36" "X37" "X38" "X39" "X40" "X41"

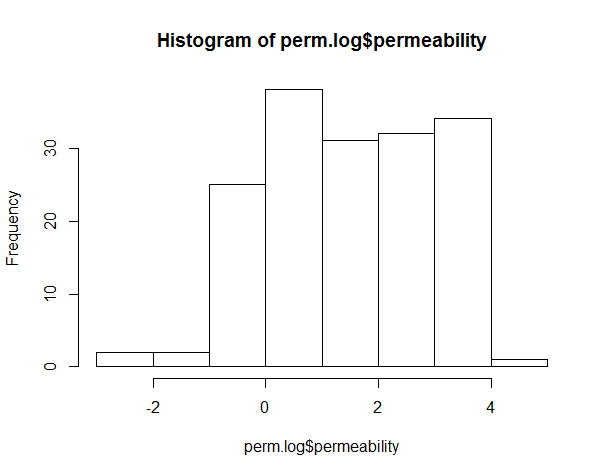
[43] "X42" "X43" "X44" "X45" "X46" "X47" "X48"

[50] "X49" "X50" "X51" "X52" "X53" "X54" "X55"   
 **ETC…**

> hist(perm$permeability) > BCtran(perm$permeability)  
 

> perm.log = perm

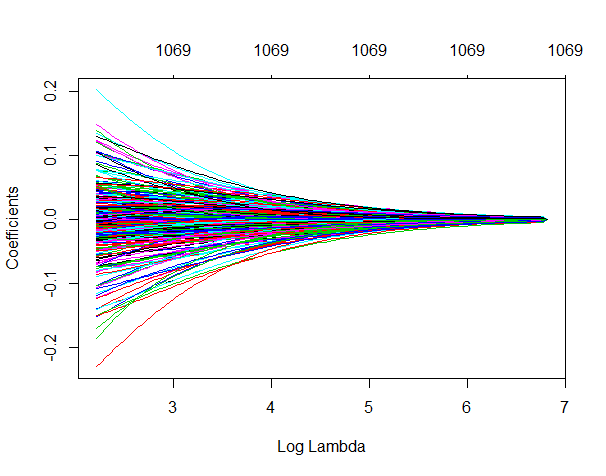
> perm.log$permeability = log(perm$permeability)

> hist(perm.log$permeability)  


> X = model.matrix(permeability~.,data=perm.log)[,-1]

> y = perm.log$permeability

> perm.ridge = glmnet(X,y,alpha=0)

> plot(perm.ridge,xvar=”lambda”)  


Because the response was log transformed we need to back-transform the predictions and the actual response values for the validation sample to the original scale. A simple edit to the glmnet.ssmc function above does this.

glmnet.sslog = function(X,y,p=.667,M=100,alpha=1,lambda=1) {

RMSEP = rep(0,M)

MAEP = rep(0,M)

MAPEP = rep(0,M)

n = nrow(X)

for (i in 1:M) {

ss = floor(n\*p)

sam = sample(1:n,ss,replace=F)

fit = glmnet(X[sam,],y[sam],lambda=lambda,alpha=alpha)

ypred = predict(fit,newx=X[-sam,])

ya = exp(y[-sam])

ypred = exp(ypred)

RMSEP[i] = sqrt(mean((ya-ypred)^2))

MAEP[i] = mean(abs(ya-ypred))

MAPEP[i]=mean(abs(ypred-ya)/ya)

}

cat("RMSEP =",mean(RMSEP)," MAEP=",mean(MAEP)," MAPEP=",mean(MAPEP))

cv = return(data.frame(RMSEP=RMSEP,MAEP=MAEP,MAPEP=MAPEP))

}

**Ridge**

> ridge.results = glmnet.sslog(X,y,M=1000,alpha=0,lambda=bestlam.ridge)

RMSEP = 13.3899 MAEP= 8.332502 MAPEP= 2.130067

> summary(ridge.results)

RMSEP MAEP MAPEP

Min. : 8.102 Min. : 5.229 Min. :0.6487

1st Qu.:12.168 1st Qu.: 7.441 1st Qu.:1.3257

Median :13.349 Median : 8.249 Median :2.0420

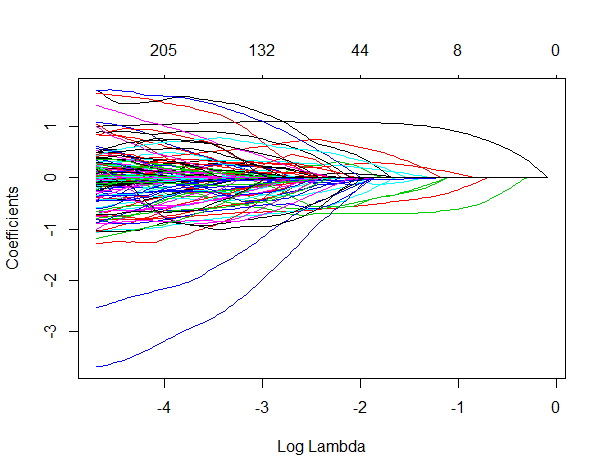
Mean :13.390 Mean : 8.333 Mean :2.1301

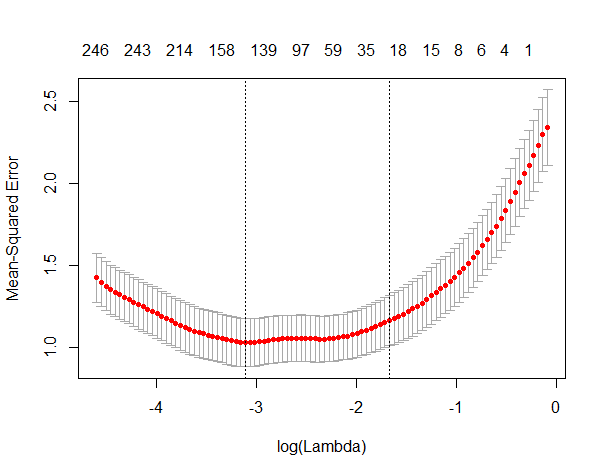
3rd Qu.:14.635 3rd Qu.: 9.202 3rd Qu.:2.7221

Max. :19.913 Max. :13.070 Max. :6.8239   
  
**LASSO**

> perm.lasso = glmnet(X,y,alpha=1)

> plot(perm.lasso,xvar=”lambda”)



> cv.lasso = cv.glmnet(X,y,alpha=1)  
> plot(cv.lasso)  


> lasso.results = glmnet.sslog(X,y,M=1000,alpha=1,lambda=bestlam.lasso)

RMSEP = 14.8487 MAEP= 8.265711 MAPEP= 1.750878  
> summary(lasso.results)

RMSEP MAEP MAPEP

Min. : 7.546 Min. : 4.354 Min. : 0.5694

1st Qu.:12.359 1st Qu.: 7.172 1st Qu.: 1.2238

Median :13.773 Median : 8.059 Median : 1.6533

Mean :14.849 Mean : 8.266 Mean : 1.7509

3rd Qu.:15.506 3rd Qu.: 9.033 3rd Qu.: 2.1580

Max. :55.661 Max. :17.372 Max. :14.0333

> lasso.results = glmnet.sslog(X,y,M=1000,alpha=1,lambda=exp(-1.75))

RMSEP = 13.17988 MAEP= 7.645071 MAPEP= 1.495597

> summary(lasso.results)

RMSEP MAEP MAPEP

Min. : 4.804 Min. : 3.311 Min. :0.5712

1st Qu.:11.909 1st Qu.: 6.726 1st Qu.:1.0492

Median :13.259 Median : 7.667 Median :1.3689

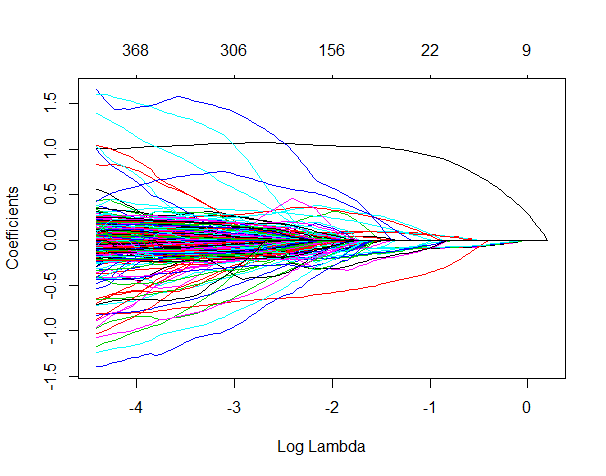
Mean :13.180 Mean : 7.645 Mean :1.4956

3rd Qu.:14.496 3rd Qu.: 8.482 3rd Qu.:1.8837

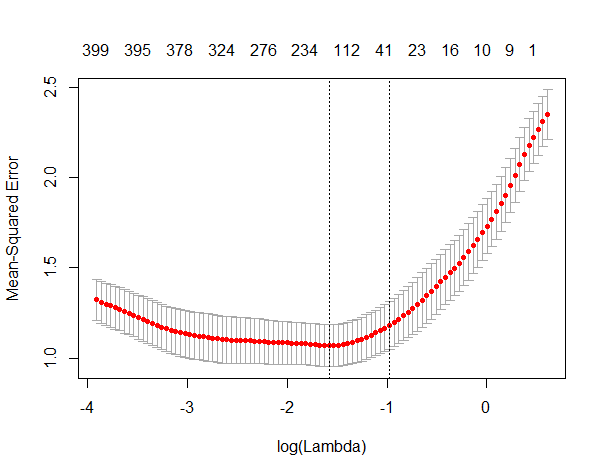
Max. :19.048 Max. :12.017 Max. :3.6300

**Elastic Net**

> perm.en = glmnet(X,y,alpha=0.5)

> plot(perm.en,xvar=”lambda”)  


> cv.en = cv.glmnet(X,y,alpha=0.5)

> plot(cv.en)  


> bestlam.en = cv.en$lambda.min

> bestlam.en

[1] 0.2061238

> cv.en$lambda.min + cv.en$lambda.1se

[1] 0.5834824  
> en.results = glmnet.sslog(X,y,alpha=0.5,lambda=bestlam.en)

RMSEP = 13.10113 MAEP= 7.600295 MAPEP= 1.559329

> summary(en.results)

RMSEP MAEP MAPEP

Min. : 9.166 Min. : 5.259 Min. :0.7139

1st Qu.:11.639 1st Qu.: 6.679 1st Qu.:1.0393

Median :12.765 Median : 7.540 Median :1.4951

Mean :13.101 Mean : 7.600 Mean :1.5593

3rd Qu.:14.514 3rd Qu.: 8.366 3rd Qu.:1.8835

Max. :21.093 Max. :11.082 Max. :5.2646   
  
> en.results = glmnet.sslog(X,y,alpha=0.5,lambda=.5835)

RMSEP = 14.13835 MAEP= 8.18124 MAPEP= 1.671998

We can try other values for (alpha)  
> perm.en = glmnet(X,y,alpha=0.75) 🡨 gives more weight to the LASSO penalty

> cv.en = cv.glmnet(X,y,alpha=0.75)

> bestlam.en = cv.en$lambda.min

> bestlam.en

[1] 0.1508137

> en.results = glmnet.sslog(X,y,alpha=0.75,lambda=bestlam.en)

RMSEP = 12.65367 MAEP= 7.400833 MAPEP= 1.433923  
> summary(en.results)

RMSEP MAEP MAPEP

Min. : 7.259 Min. : 4.366 Min. :0.636

1st Qu.:11.491 1st Qu.: 6.599 1st Qu.:1.018

Median :12.768 Median : 7.420 Median :1.326

Mean :12.654 Mean : 7.401 Mean :1.434

3rd Qu.:13.865 3rd Qu.: 8.318 3rd Qu.:1.823

Max. :17.415 Max. :10.106 Max. :2.866

> cv.en = cv.glmnet(X,y,alpha=0.25) 🡨 gives more weight to the ridge penalty

> bestlam.en = cv.en$lambda.min

> bestlam.en

[1] 0.2589034  
  
> en.results = glmnet.sslog(X,y,alpha=0.25,lambda=bestlam.en)

RMSEP = 13.04625 MAEP= 7.566317 MAPEP= 1.557941

For these data it appears the Elastic Net model outperforms both the LASSO and Ridge Regression models. The Elastic Net that appears best gives more weight to the LASSO () penalty. Though was used we could consider other choices as well.

In the next section we examine other methods for dealing with situations where the number of predictors is large. These methods seek to find linear combinations of the predictors that maximally explain the variation in the response, rather than treat each predictor individually as the regularized/penalized regression methods discussed in this section do.