Ridge regression

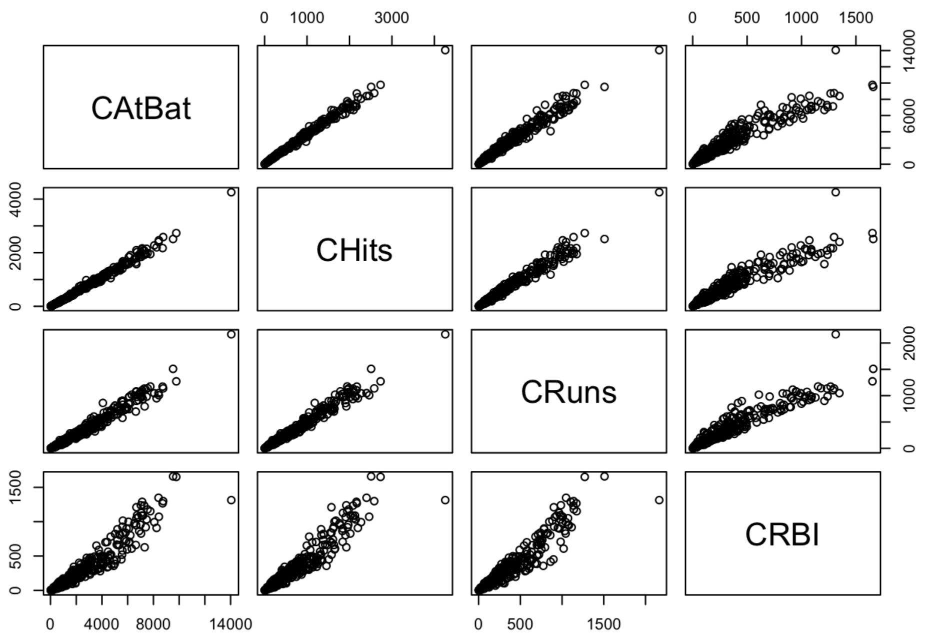
```
library(alr4)
library(glmnet)
library(ISLR)
```

Collinear data

The “Hitters” dataset, with data on player statistics and salaries in Major League Baseball. (Based on material from a course by on statistical modeling by Arturo Valdivia at Indiana University, Bloomington.

What do we do when our predictors are highly correlated (collinear)?

```
pairs(~CAtBat+CHits+CRuns+CRBI,data=Hitters)
```



```
# summary(Hitters)
```

Before we try to model the Hitters data, there’s some preparation to do. We’ll scale the data (adjust so that mean=0 and variance = 1). Some of the data is categorical, so for multiple regression we’ll also need it as a model matrix:

```
x <- scale(model.matrix(Salary~.,Hitters))[,-1]
# head(x)
```

We also need to check for NA values, and make a new data frame with just the data for which salary is available.

```
summary(is.na(Hitters$Salary))
```

```
##    Mode   FALSE    TRUE    NA's
## logical     263      59       0
```

```
y <- scale(with(Hitters,Salary[is.na(Salary)==FALSE]))
yx <- as.data.frame(cbind(y,x))
```

Now, a straightforward Ordinary Least Squares model:

```
ols.lm <- lm(y~.,yx)
ols.glmnet <- glmnet(x,y,alpha=0,lambda=0,thresh=1e-14) # equivalent - see below
# ols.lm$coef
# predict(ols.glmnet,type='coef')
summary(ols.lm)
```

```
##
## Call:
## lm(formula = y ~ ., data = yx)
##
## Residuals:
##        Min         1Q     Median         3Q        Max
## -1.694e-14 -1.290e-16  5.080e-17  2.283e-16  1.276e-15
##
## Coefficients:
##               Estimate Std. Error    t value Pr(>|t|)
## (Intercept) -2.189e-31  6.951e-17  0.000e+00    1.000
## V1           1.000e+00  1.034e-16  9.674e+15   <2e-16 ***
## AtBat        2.480e-16  3.402e-16  7.290e-01    0.467
## Hits        -2.607e-16  3.910e-16 -6.670e-01    0.506
## HmRun       -5.577e-17  1.942e-16 -2.870e-01    0.774
## Runs         2.461e-16  2.723e-16  9.040e-01    0.367
## RBI         -8.448e-18  2.405e-16 -3.500e-02    0.972
## Walks       -2.348e-16  1.452e-16 -1.617e+00    0.107
## Years        2.834e-16  2.126e-16  1.333e+00    0.184
## CAtBat      -2.128e-16  1.108e-15 -1.920e-01    0.848
## CHits       -1.919e-17  1.562e-15 -1.200e-02    0.990
## CHmRun       1.104e-16  4.748e-16  2.320e-01    0.816
## CRuns        1.189e-16  8.947e-16  1.330e-01    0.894
## CRBI        -1.805e-16  8.023e-16 -2.250e-01    0.822
## CWalks      -2.834e-17  3.133e-16 -9.000e-02    0.928
## LeagueN      7.793e-17  1.418e-16  5.500e-01    0.583
## DivisionW   -4.181e-17  7.345e-17 -5.690e-01    0.570
## PutOuts     -1.301e-16  7.952e-17 -1.636e+00    0.103
## Assists      7.228e-17  1.153e-16  6.270e-01    0.531
## Errors      -1.077e-16  1.038e-16 -1.037e+00    0.301
## NewLeagueN   3.940e-17  1.410e-16  2.790e-01    0.780
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.127e-15 on 242 degrees of freedom
## Multiple R-squared:      1,  Adjusted R-squared:      1
## F-statistic: 1.031e+31 on 20 and 242 DF,  p-value: < 2.2e-16
```

However, the collinearity in this data means that the OLS model is likely to suffer from high variance. The variance inflation factor is a measure of the collinearity of the data (it's 1/(1-r2) for each model derived from regressing one of the predictor variables on the others). Checking for inflation factors greater than 10, we can see that several variables are highly collinear.

```
vif(ols.lm)[vif(ols.lm)>100]
```

```
##   CAtBat    CHits    CRuns     CRBI
## 253.2229 503.0360 165.0325 132.7044
```

(Note that the variance inflation factor is scale-invariant.)

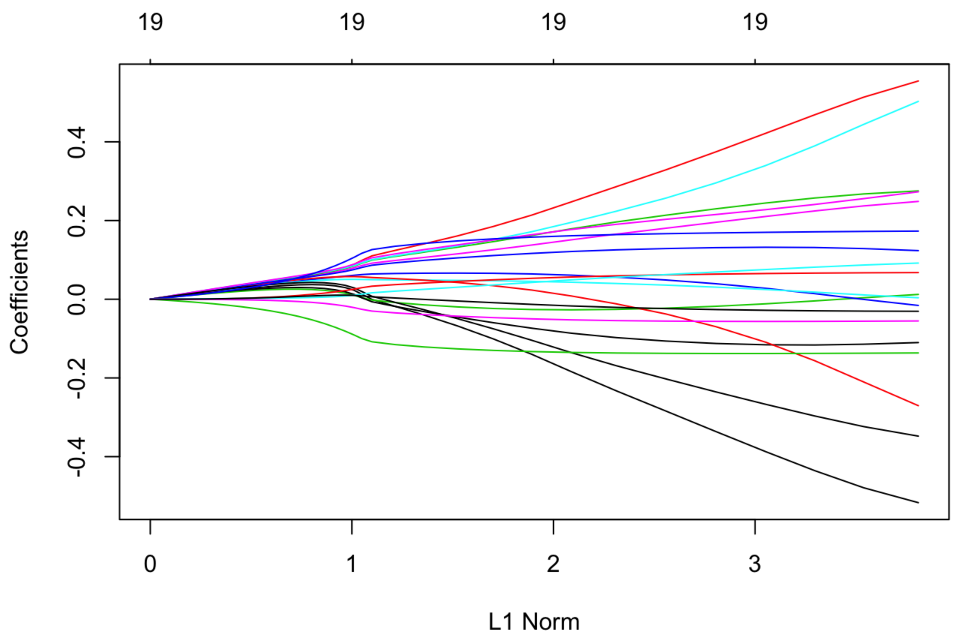
Ridge regression is a technique for constraining the coefficients to avoid overfitting the model and increasing the variance. Rather than finding coefficients that minimize the residual sum of squares, ridge regression minimizes RSS + lambda*(sum of squared coefficients), where lambda is the constraint or regularization parameter. Ridge regression estimates will have a certain amount of bias, but less variance, and better performance when generalized to new data. We'll generate a logarithmic sequence of lambdas and then generate a set of models, one model per value of lambda.

```
grid <- 10^seq(10,-2,length=100) # logarithmic sequence of lambdas
# grid
ridge <- glmnet(x,y,alpha=0,lambda=grid,thresh=1e-14)
dim(coef(ridge))  # a set of coefficients for each value of lambda
```

```
## [1]  20 100
```

We can plot the values the coefficients take for each value of lambda.

```
plot(ridge)
```



For example, the fiftieth value of lambda and the coefficients associated with it:

```
# E.g., for the 50th value of lambda...
ridge$lambda[50]
```

```
## [1] 11497.57
```

```
coef(ridge)[,50]
```

```
##   (Intercept)         AtBat          Hits         HmRun          Runs
##  1.196732e-16  3.424868e-05  3.806017e-05  2.975764e-05  3.642658e-05
##           RBI         Walks         Years        CAtBat         CHits
##  3.899305e-05  3.851037e-05  3.475586e-05  4.564351e-05  4.762041e-05
##        CHmRun        CRuns          CRBI        CWalks       LeagueN
##  4.553973e-05  4.881487e-05  4.918691e-05  4.249240e-05 -1.235463e-06
##     DivisionW       PutOuts       Assists        Errors    NewLeagueN
## -1.670831e-05  2.607515e-05  2.207196e-06 -4.692877e-07 -2.430287e-07
```

We can also use the predict method in glmnet to get the actual coefficients for particular values of lambda:

```
# predict(ridge,s=ridge$lambda[50],type='coef')
```

How do we choose the best value of lambda? We can use cross-validation, splitting the data into training and test sets and then determining which value of lambda leads to the lowest mean squared error. For example, for one possible partition of the data into training and test sets, and one value of lambda, we get the following predictions (in standardized units):

```
set.seed(1)
train <- sample(1:nrow(x),nrow(x)/2)
test <- (-train)
y.test <- y[test]
y.train <- y[train]
ridge1 <- glmnet(x[train, ],y[train],alpha=0,lambda=grid,thresh=1e-14)
```

Predictions of the trained model for the test data, with lambda=4:

```
ridge1.pred <- predict(ridge1,s=4,newx=x[test,])
head(ridge1.pred)
```

```
##                            1
## -Alan Ashby       0.03994414
## -Andre Dawson     0.51678145
## -Alfredo Griffin  0.12920569
## -Argenis Salazar -0.55032339
## -Andres Thomas   -0.42857460
## -Andre Thornton   0.45096946
```

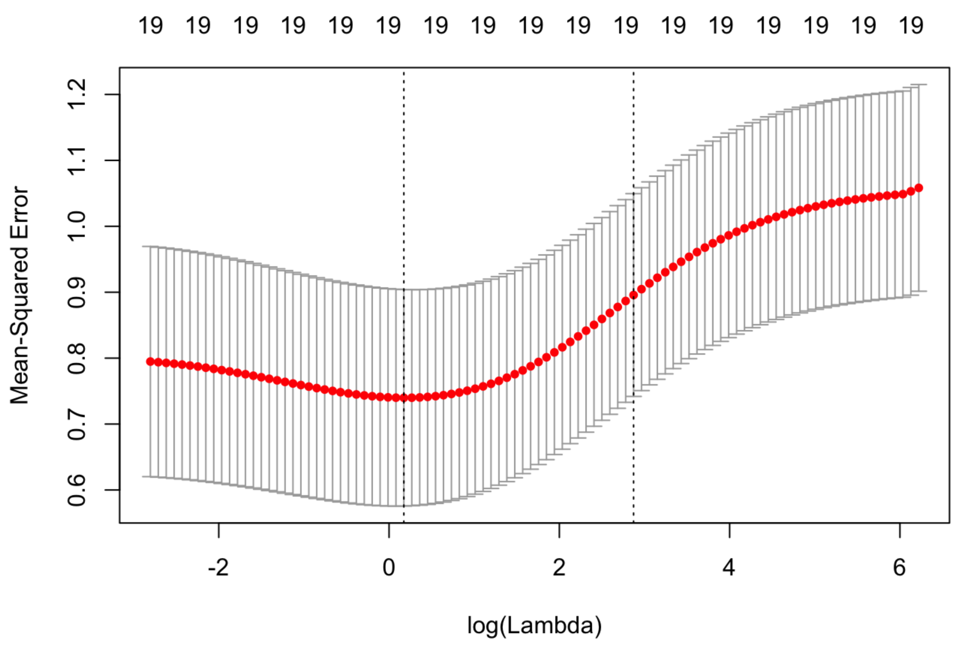
Now, compare the mean squared error of the model with different values of lambda, 0 for OLS. Not all nonzero values of lambda will result in lower MSE than lambda=0!

```
ols.pred <- predict(ridge1, s=0, newx=x[test, ])
ridge.pred <- predict(ridge1,s=1e2,newx=x[test, ],exact=F)
ols.mse <- mean((ols.pred - y[test])^2)
ridge.mse <- mean((ridge.pred - y[test])^2)
cbind(ols.mse,ridge.mse)
```

```
##        ols.mse ridge.mse
## [1,] 0.4934768 0.8941265
```

Choosing a good value for lambda is essential. Use cross-validation to estimate the MSE for different possible values of lambda. (Compare the results above with lambda=0 and lambda=100 to the predicted MSE values in as plotted below.)

```
set.seed(1)
cv.results <- cv.glmnet(x[train,],y[train],alpha=0,nfolds=8)
plot(cv.results)
```



For this optimal value of lambda, the mean squared error is somewhat lower than for the OLS model:

```
prize.lambda <- cv.results$lambda.min
prize.lambda
```

```
## [1] 1.190024
```

```
ridge.pred.prize=predict(ridge,s=prize.lambda ,newx=x[test ,])
ridge.prize.mse <- mean((ridge.pred.prize -y[test])^2)
cbind(ols.mse,ridge.mse,ridge.prize.mse)
```

```
##        ols.mse ridge.mse ridge.prize.mse
## [1,] 0.4934768 0.8941265       0.4206914
```