HDS 6.1 Breast cancer metastasis example

Matti Pirinen, University of Helsinki

9.1.2024

Let's look at a study of breast cancer metastasis published by Van't Veer et al. in 2002. Data (15 MB) were downloaded from http://web.as.uky.edu/statistics/users/pbreheny/603/ and can now be found from https://www.mv.helsinki.fi/home/mjxpirin/HDS_course/material/vantveer.txt.

In the study, biological samples were obtained from tumors of women with breast cancer. These samples were scanned with a microarray, that measures the expression of 10,000s of genes simultaneously, i.e, how much of each gene product is being produced by the cells in the sample. The patients were followed up to see how long it took for the cancer to metastasize (spread elsewhere, which is bad news). Clinically, the goal is to identify patients with poor prognosis in order to administer more aggressive follow-up treatment for them. Scientifically, knowledge of the genes related to the worse outcomes can help understand the disease better and to help develop some therapeutics in the future.

```
set.seed(21)
D = read.table("vantveer.txt", header = T) # add your path here
dim(D) # rows patients, cols outcome + gene expressions
```

```
## [1]    98 24189
```

```
anyNA(D)
```

```
## [1] FALSE
```

```
y = as.numeric(D[,1]) # survival time
X = as.matrix(D[,2:ncol(D)]) # gene expression measurements
rm(D)
n = length(y)
p = ncol(X)
summary(y)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    1.00   25.25   56.50   63.83   97.00  161.00
```

```
summary(apply(X, 2, mean)) # not mean centered
```

```
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## -1.409847 -0.023194  0.006776 -0.003500  0.032255  0.313347
```

```
summary(apply(X, 2, sd)) # not standardized
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.03919 0.11772 0.16494 0.19807 0.24711 1.08625
```

A possible filtering step could be to remove the predictors that have small variance, because those are less likely to be informative in statistical sense. Of course, any gene could be important biologically even if its expression varies only a little between individuals but we have less statistical power to find such effects, and therefore, if we needed to filter out predictors, those uninformative genes could be candidates. Our methods are efficient enough so, for now, let's keep all predictors in and standardize the columns.

```
X = scale(X) # now mean = 0 and var = 1 for each column
y = y - mean(y) # mean-center, but do not scale to keep the interpretation of time units.
# we can now ignore the intercept because everything is mean-centered
```

Let's first do a quick version of the ordinary least squares univariately for each gene j using the model $y = x_j\beta_j + \varepsilon$. We do not want to do a for-loop to apply `lm()` for 24,000+ times but instead we use formulas from the Exercise 1.5.

After standardization within the sample $\boldsymbol{x}_j^T\boldsymbol{x}_j = n-1$ for each column j and thus

$$\widehat{\beta}_j = \frac{\boldsymbol{x}_j^T\boldsymbol{y}}{\boldsymbol{x}_j^T\boldsymbol{x}_j} = \frac{\boldsymbol{x}_j^T\boldsymbol{y}}{n-1}.$$

Thus, the vector $\widehat{\boldsymbol{\beta}}^{(UNI)} = \boldsymbol{X}^T\boldsymbol{y}/(n-1)$ has the univariate least squares estimates, and this can be computed by a single matrix-by-vector operation, so it is as quick as it can get.

Similarly, we can compute the univariate estimate of σ_j^2 for each gene as

$$\widehat{\sigma^2}_j = \frac{1}{n-2}\left(\boldsymbol{y} - \boldsymbol{x}_j\widehat{\beta}_j\right)^T\left(\boldsymbol{y} - \boldsymbol{x}_j\widehat{\beta}_j\right)$$

and then we have estimates for the standard errors as

$$s_j = \sqrt{\frac{\widehat{\sigma^2}_j}{(n-1)}}.$$

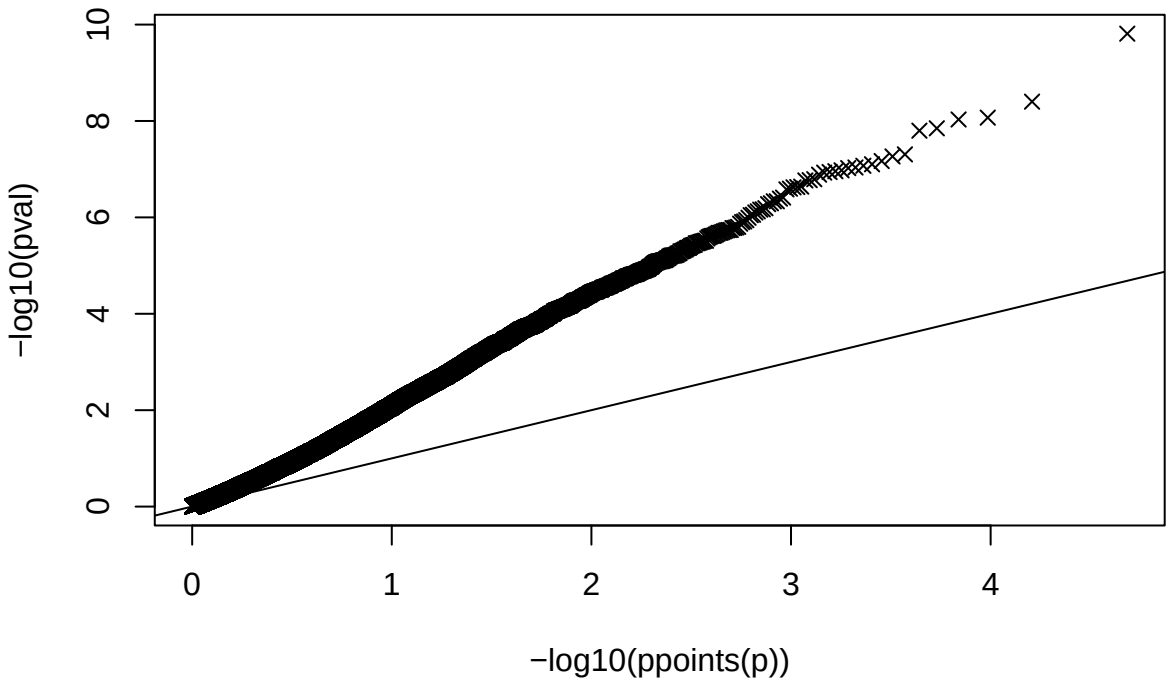
```
beta.uni = as.vector((t(X)%*%y)/(n-1))
sigma2.uni = colSums(( y - t( t(X)*beta.uni) )^2)/(n-2) #this is sigma2 formula above written in R
se = sqrt(sigma2.uni/(n-1))
# Now we have fit > 24,000 linear models.
# Check an arbitrary column against lm() output:
i = 10625
summary(lm(y ~ X[,i]))$coefficients[2,1:2]
```

```
##   Estimate Std. Error
##  -5.682152   4.539231
```

```
c(beta.uni[i], se[i])
```

```
##           Gene10625
## -5.682152  4.539231
```

```r
# OK.
pval = pchisq( (beta.uni/se)^2, df = 1, lower = F)
qqplot(-log10(ppoints(p)), -log10(pval), pch = 4) # huge deviation from the null
abline(0,1)
```




```r
summary(pval)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.00000 0.07208 0.29467 0.36428 0.62452 0.99990
```

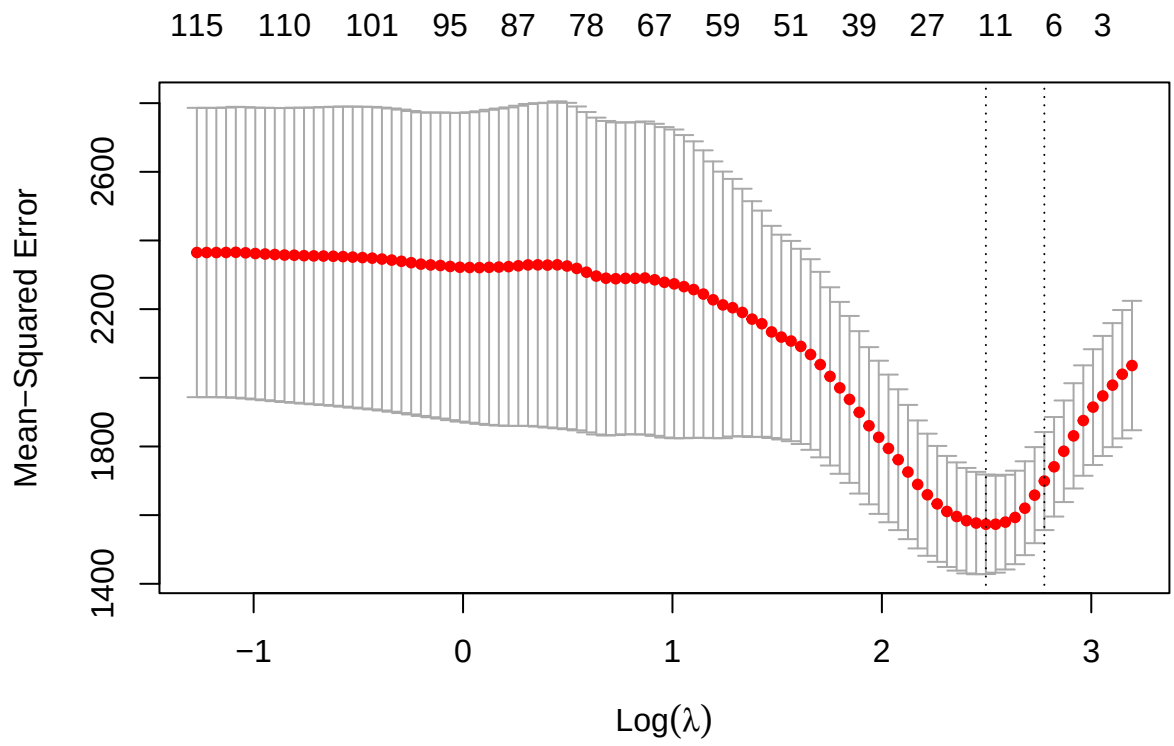
Let's fit a LASSO model to the data.

```r
library(glmnet)
```

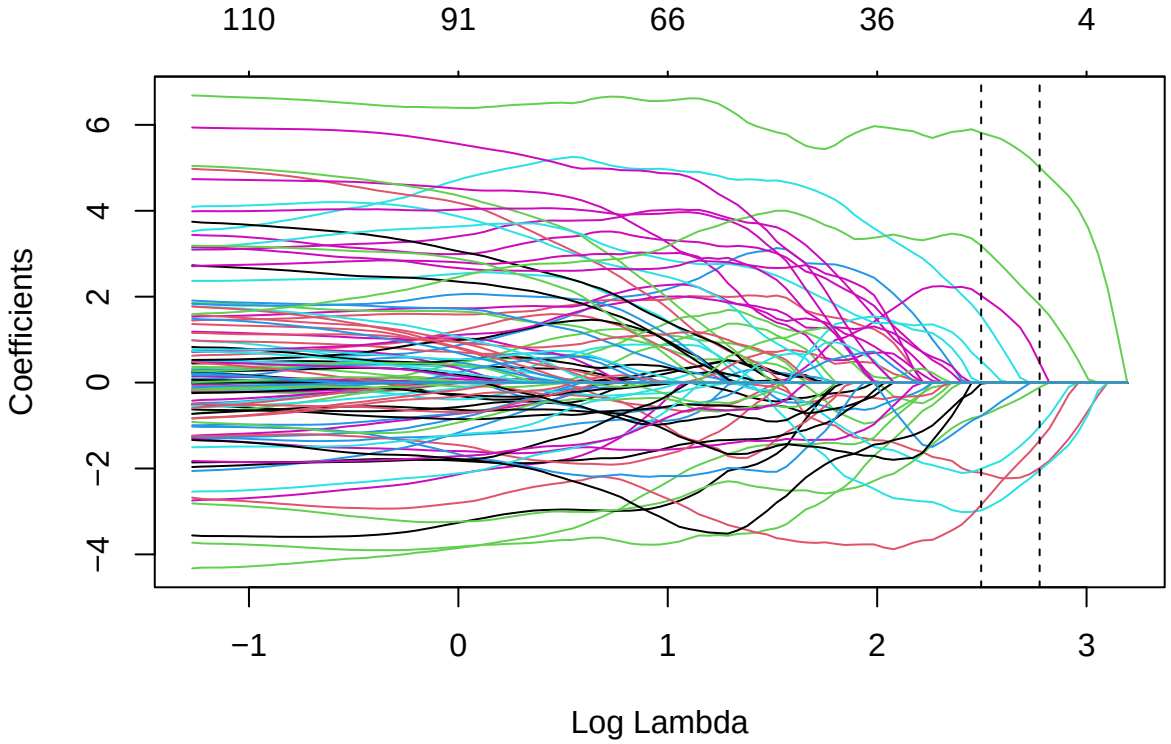
```
## Loading required package: Matrix
```

```
## Loaded glmnet 4.1-8
```

```r
cv.lasso = cv.glmnet(X, y, alpha = 1) # takes < 5 seconds even with p > 24,000
plot(cv.lasso)
```




```
plot(cv.lasso$glmnet.fit, xvar = "lambda")
abline( v = log(cv.lasso$lambda.min), lty = 2 )
abline( v = log(cv.lasso$lambda.1se), lty = 2 )
```



LASSO seems to choose only 12 predictors at `lambda.min`. Let's look at their statistics.

```
lasso.var = as.matrix(coef(cv.lasso, s = "lambda.min"))
genes = names(which(abs(lasso.var[,1]) > 1e-10)) # names of the chosen genes
lasso.ind = which(abs(lasso.var[,1]) > 1e-10) - 1 # indexes, removing intercept index by -1
data.frame(gene = genes, beta = beta.uni[lasso.ind], se = se[lasso.ind], pval = pval[lasso.ind])
```

```
##                gene      beta       se         pval
## Gene681     Gene681 -23.08704 3.922848 3.973909e-09
## Gene1699   Gene1699  18.03849 4.189450 1.664642e-05
## Gene3812   Gene3812  18.78226 4.155264 6.180659e-06
## Gene8878   Gene8878  21.26679 4.028614 1.299367e-07
## Gene9616   Gene9616  24.52200 3.831071 1.545437e-10
## Gene10755 Gene10755 -21.11582 4.036878 1.688423e-07
## Gene10986 Gene10986 -22.67197 3.947985 9.319588e-09
## Gene12106 Gene12106 -21.63213 4.008301 6.782746e-08
## Gene13143 Gene13143 -21.32241 4.025550 1.178703e-07
## Gene16323 Gene16323 -21.46052 4.017898 9.231780e-08
## Gene20199 Gene20199  22.71551 3.945377 8.536801e-09
## Gene23726 Gene23726 -18.23905 4.180395 1.282930e-05
```

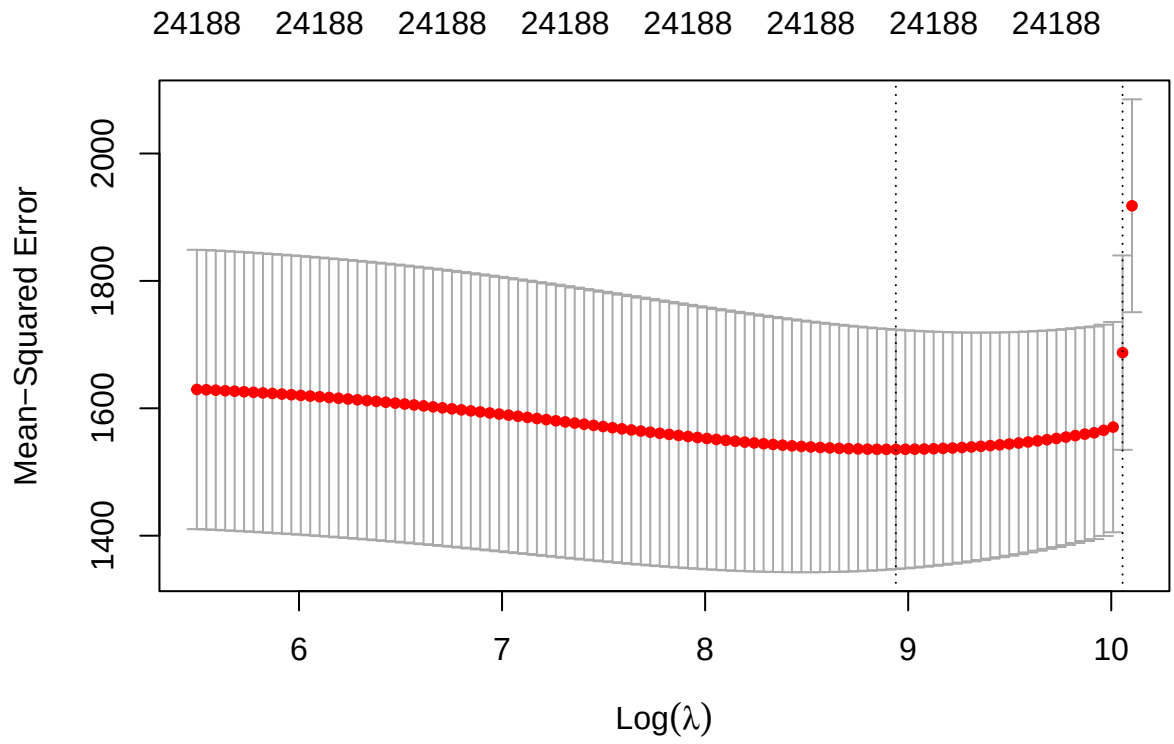
```
sum(pval < 1e-7)
```

```
## [1] 13
```

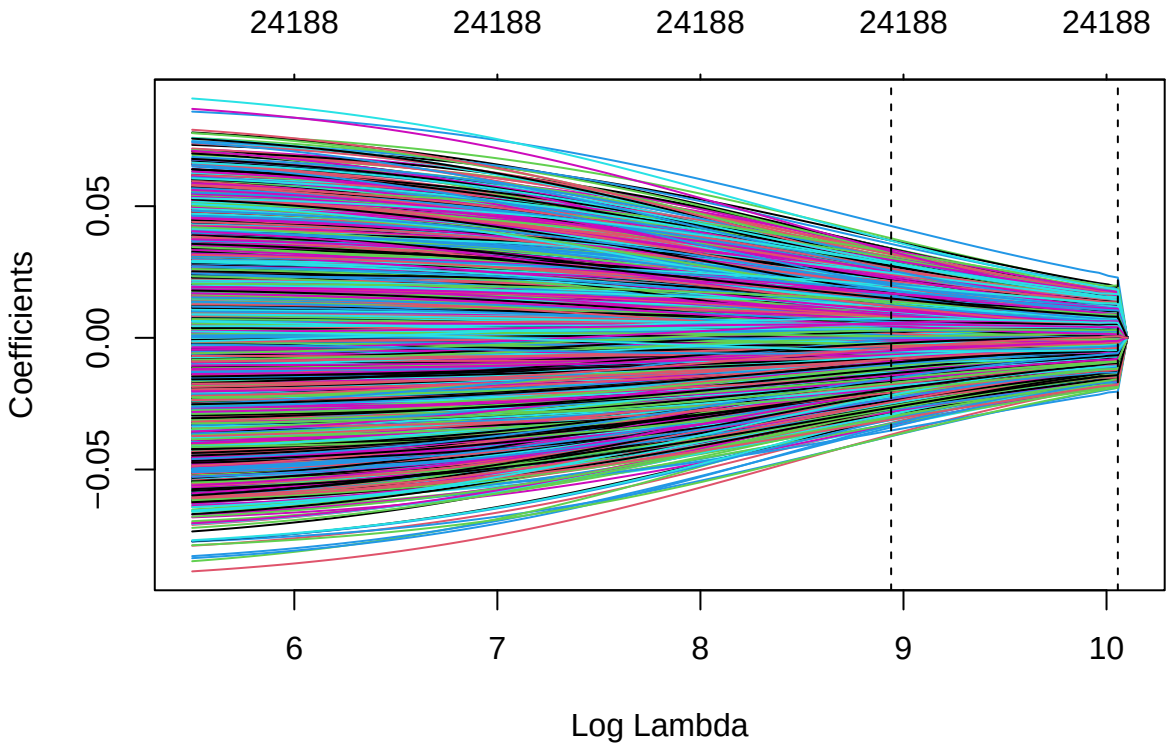
We see that the P-values of variables chosen by LASSO are small in general ($<$ 1e-5) but that there are many other genes with small P-values that LASSO has dropped, e.g., 13 genes had a P-value $<$ 1e-7 and there are only 6 such in the list. So LASSO is not just about sorting P-values.

Can we also fit ridge regression to this data set with p $>$ 24,000 even when it does not produce similar sparsity as LASSO?

```
cv.ridge = cv.glmnet(X, y, alpha = 0) # takes only ~ 20 seconds even with p > 24000
plot(cv.ridge)
```




```
plot(cv.ridge$glmnet.fit, xvar = "lambda")
abline( v = log(cv.ridge$lambda.min), lty = 2 )
abline( v = log(cv.ridge$lambda.1se), lty = 2 )
```

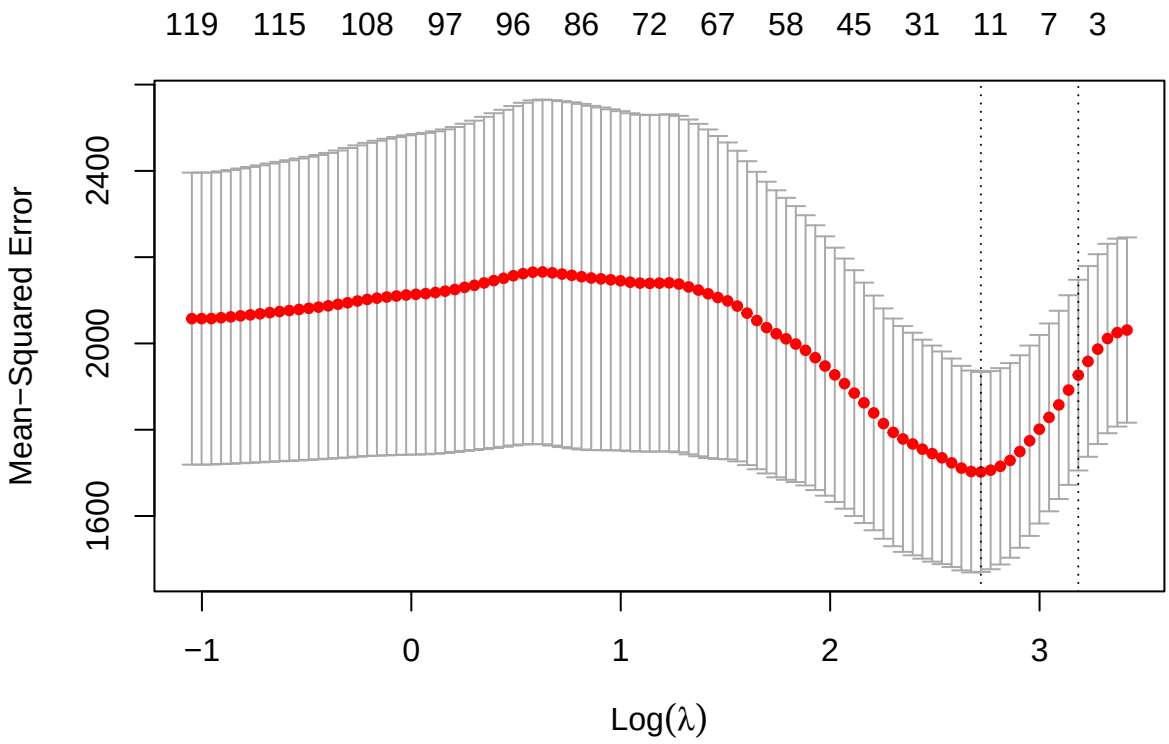


The cross-validated MSEs at minimum seem similar between ridge regression (RR) and LASSO. Note also that λ is large ($> \exp(9) \approx 8100$) so RR heavily penalizes the linear model, as it should when $p >> n$.

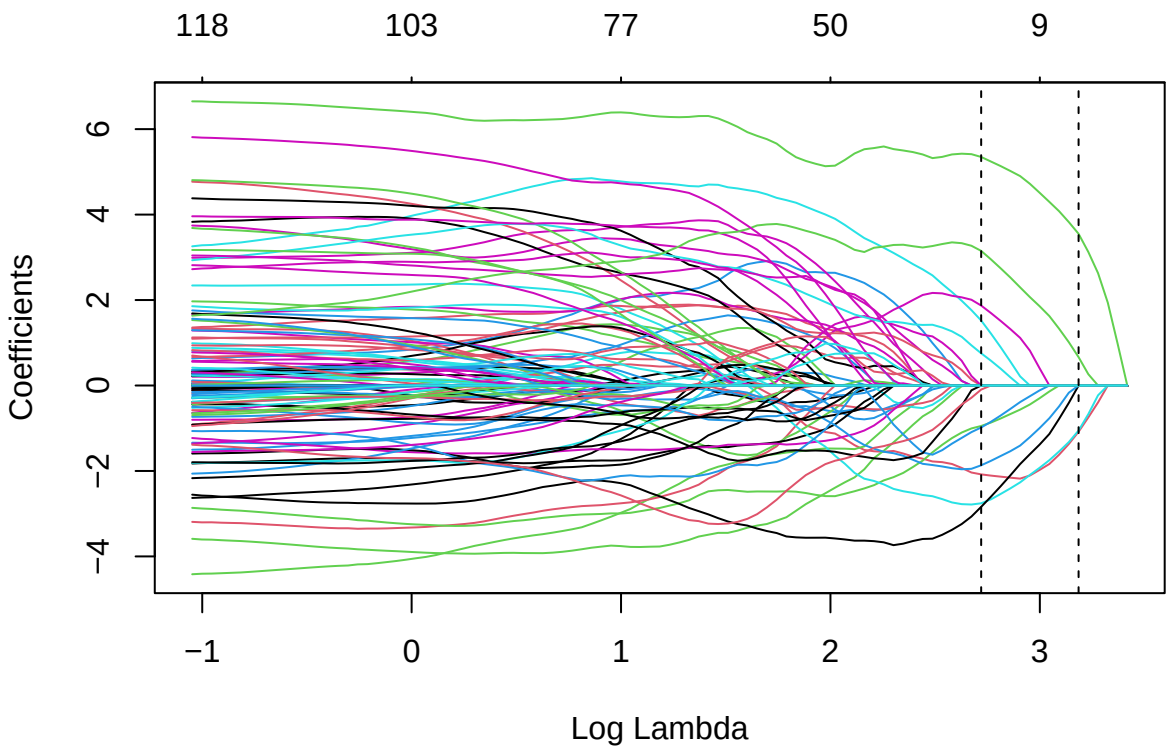
A big benefit of LASSO is its small number of predictors, here only 12 compared to 24,188 that are used by RR. If we can build two prediction models that are almost as good, and the first one has 12 predictors and the second has 24,188 predictors, the first one is more practical in many ways.

What about trying out some elastic net model that is a compromise between the two?

```
cv.enet = cv.glmnet(X, y, alpha = 0.8) # takes only ~ 5 seconds even with p > 24,000
plot(cv.enet)
```




```
plot(cv.enet$glmnet.fit, xvar = "lambda")
abline( v = log(cv.enet$lambda.min), lty = 2 )
abline( v = log(cv.enet$lambda.1se), lty = 2 )
```



Elastic net has a few more predictors than LASSO but still produces a very sparse model compared to original p. CV'd MSEs seem similar between the three methods.