title

Matt Richey

03/11/2020

1 Set up

Load the librarys

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

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

```
## Loading required package: foreach
```

```
## Loaded glmnet 2.0-16
```

```
library(tidyverse)
```

```
## -- Attaching packages ------------ tidyverse 1.2.1 --
```

```
## v ggplot2 3.3.0     v purrr   0.3.3
## v tibble  2.1.3     v dplyr   0.8.4
## v tidyr   1.0.2     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.3.0
```

```
## -- Conflicts --------------- tidyverse_conflicts() --
## x purrr::accumulate() masks foreach::accumulate()
## x tidyr::expand()     masks Matrix::expand()
## x dplyr::filter()     masks stats::filter()
## x dplyr::lag()        masks stats::lag()
## x tidyr::pack()       masks Matrix::pack()
## x tidyr::unpack()     masks Matrix::unpack()
## x purrr::when()       masks foreach::when()
```

1 Introduction

A comparison of OLS, Ridge Regression, and Lasso On Prostate data set

https://web.stanford.edu/~hastie/glmnet/glmnet_alpha.html

```
prostate.df <- read_csv("Prostate.csv")
```

```
## Parsed with column specification:
## cols(
##   lcavol = col_double(),
##   lweight = col_double(),
```

```
##   age = col_double(),
##   lbph = col_double(),
##   svi = col_double(),
##   lcp = col_double(),
##   gleason = col_double(),
##   pgg45 = col_double(),
##   lpsa = col_double()
## )
```

```
nrow(prostate.df)
```

```
## [1] 97
```

```
names(prostate.df)
```

```
## [1] "lcavol"  "lweight" "age"     "lbph"    "svi"     "lcp"     "gleason"
## [8] "pgg45"   "lpsa"
```

```
N <- nrow(prostate.df)
train <- sample(1:N,N/2,rep=F)

train.df <- prostate.df[train,]
test.df <-  prostate.df[-train,]

nrow(train.df)
```

```
## [1] 48
```

```
nrow(test.df)
```

```
## [1] 49
```

Ordinary Least Squares model

Train/test combo

Predict lpsa using all the the predictors

```
mod.lm <- lm(lpsa ~ .,data=train.df)
```

The model and coefficients

```
summary(mod.lm)
```

```
##
## Call:
## lm(formula = lpsa ~ ., data = train.df)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
```

```
## -1.59958 -0.40863  0.04425  0.29214  1.58881
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.486036   2.188648   0.679  0.50116
## lcavol       0.630255   0.185260   3.402  0.00156 **
## lweight      0.545589   0.326714   1.670  0.10294
## age         -0.027275   0.016916  -1.612  0.11494
## lbph         0.095848   0.096757   0.991  0.32798
## svi          0.377641   0.469601   0.804  0.42617
## lcp         -0.066108   0.178395  -0.371  0.71296
## gleason     -0.049731   0.308751  -0.161  0.87287
## pgg45        0.006744   0.006359   1.061  0.29541
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.8185 on 39 degrees of freedom
## Multiple R-squared:  0.6421, Adjusted R-squared:  0.5687
## F-statistic: 8.747 on 8 and 39 DF,  p-value: 9.435e-07
```

```
(coef.lm <- cbind(coef(mod.lm)))
```

```
##                     [,1]
## (Intercept)  1.486036254
## lcavol       0.630254747
## lweight      0.545589017
## age         -0.027274640
## lbph         0.095848464
## svi          0.377640666
## lcp         -0.066108379
## gleason     -0.049731252
## pgg45        0.006744203
```

Comput the MSE on test set

```
pred.lm <- predict(mod.lm,newdata=test.df)
(mse.lm <- with(test.df,mean( (pred.lm-lpsa)^2)))
```

```
## [1] 0.3832044
```

Ridge Regression with glmnet

```
prostate.mat <- data.matrix(prostate.df)
dim(prostate.mat)
```

```
## [1] 97  9
```

Build full,train, and test data matrices. This is the form that glmnet uses.

```
data.x <- prostate.mat[,-9]
data.y <- prostate.mat[,9]
```

Train

```
train.x <- data.x[train,]
train.y <- data.y[train]
```

Test

```
test.x <- data.x[-train,]
test.y <- data.y[-train]
```

Quick ridge regression test with a single lambda using the training data set

```
mod.ridge <- glmnet(train.x,
                    train.y,
                    alpha=0,
                    lambda=1,
                    intercept=TRUE)
coef.ridge <- coef(mod.ridge)
```

Compare ridge coefficients with linear model coefficients

```
cbind(coef.ridge,coef.lm)
```

```
## 9 x 2 sparse Matrix of class "dgCMatrix"
##                         s0
## (Intercept)  0.582593852  1.486036254
## lcavol       0.246926915  0.630254747
## lweight      0.367868120  0.545589017
## age         -0.009436463 -0.027274640
## lbph         0.060213732  0.095848464
## svi          0.403254431  0.377640666
## lcp          0.107534531 -0.066108379
## gleason      0.089324166 -0.049731252
## pgg45        0.003552688  0.006744203
```

Make a prediction

```
pred.ridge <- predict(mod.ridge,newx=test.x)
```

How did we do??

```
mse.rid0 <- mean( (test.y-pred.ridge)^2)
```

Not so bad, but not much better than OLS

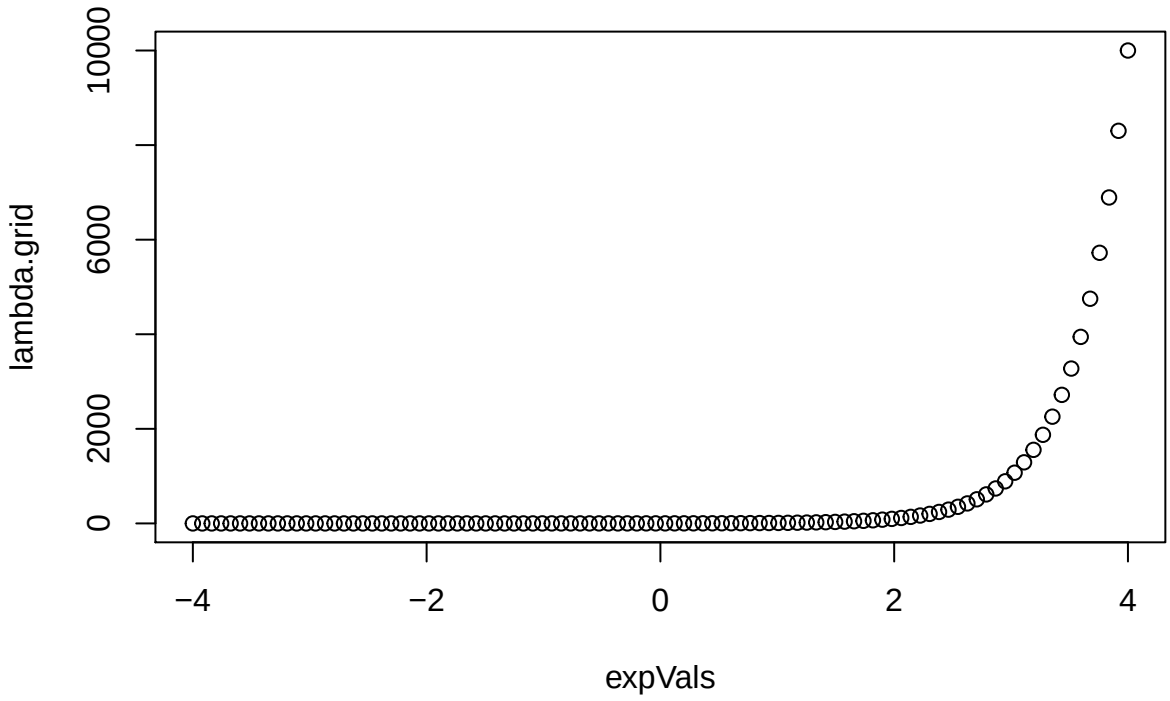
```r
c(mse.lm,mse.rid0)
```

```
## [1] 0.3832044 0.4925891
```

Using multiple lambda values

Now repeat with a grid of lambdas Build ridge regression models for a grid of lambda values a good way to do this is to size the lambda values geometrically (vs arithmetically)

```r
numLambda <- 100
expVals <- seq(-4,4,length=numLambda)
lambda.grid <- 10^expVals
plot(expVals,lambda.grid)
```

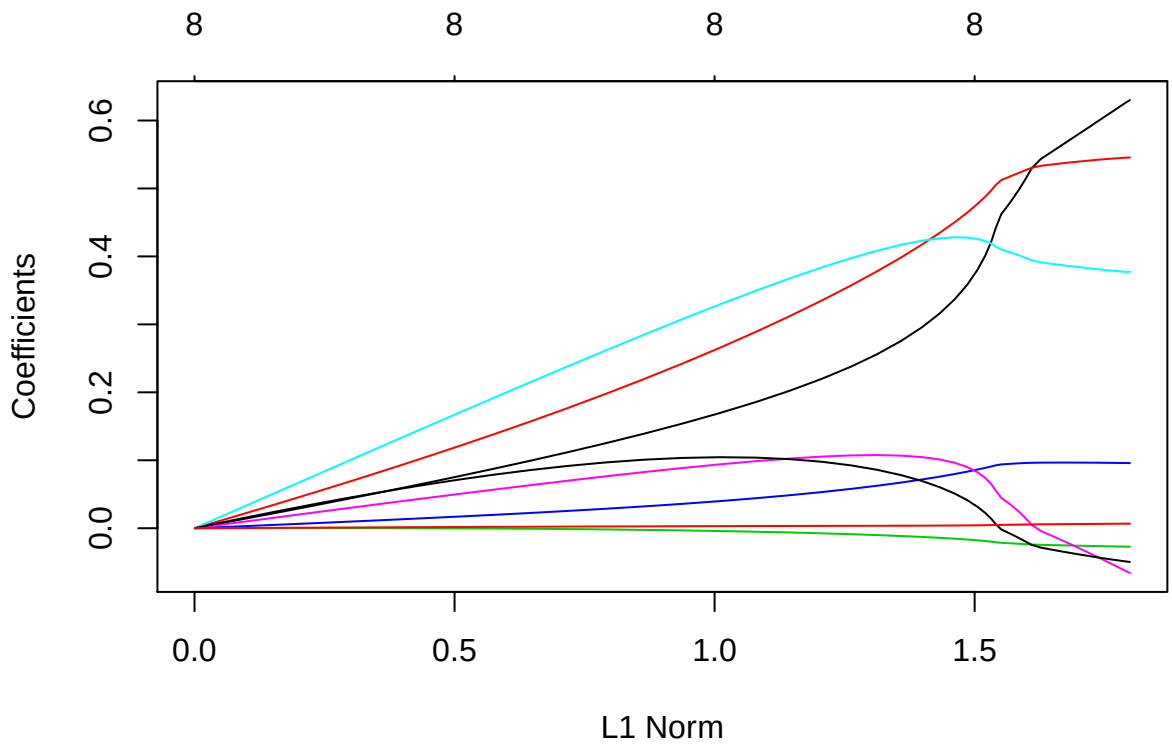


Now apply glmnet, alpha=0 means use ridge (For Lasso, alpha=1)

```r
mod.ridge <-
    glmnet(train.x,
           train.y,
           alpha=0,
           lambda=lambda.grid,
           intercept=TRUE)
```

glmnet has a built-in plot function. This shows the values of the coefficients (no intercept) as the the overall size of the coefficients shrinks to 0.

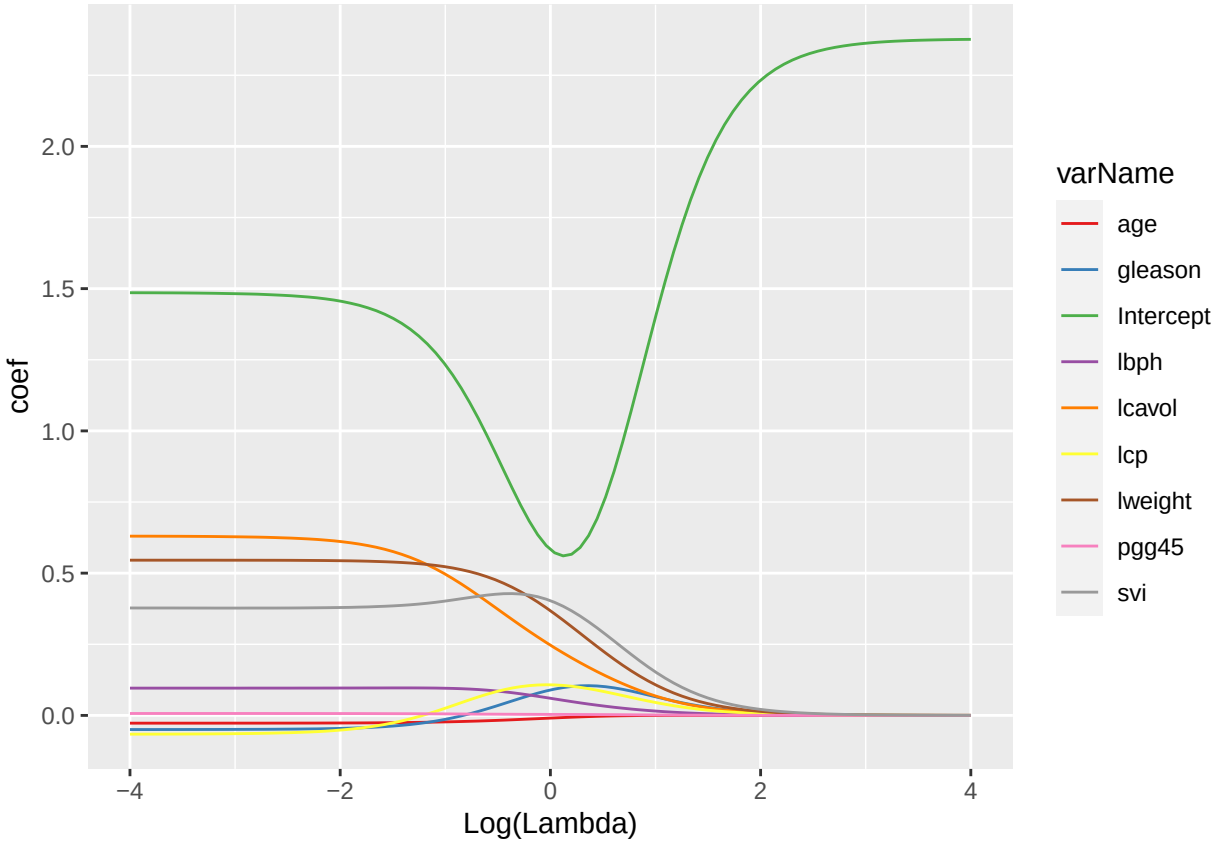
```
plot(mod.ridge)
```



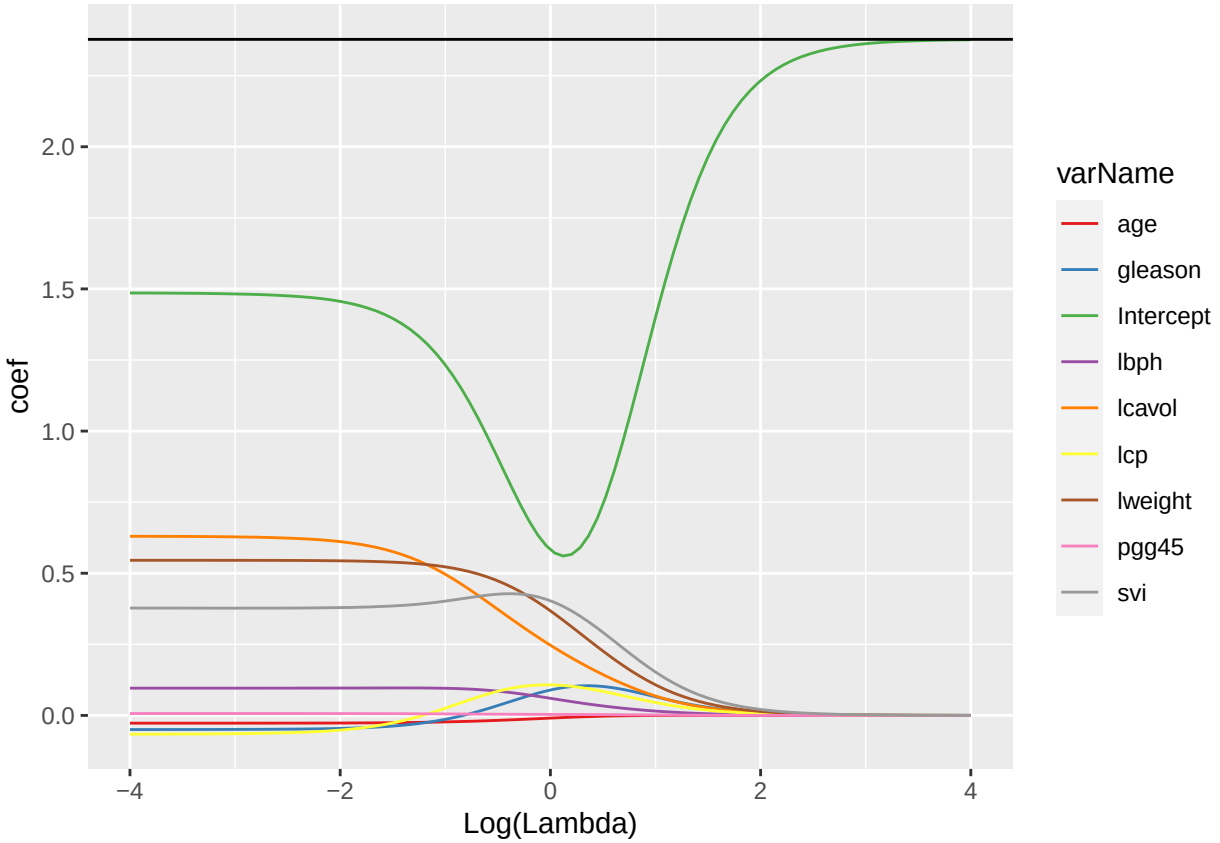
Another look at the shrinkage of the coefficients In this case, we see the shrinkage as the (log of) lambda increases also include the intercept, which eventually approaches he mean of the response variable.

```
coefs.ridge <- coef(mod.ridge)
nms <- names(train.df)[1:8]
coef.df <- data.frame(coef=matrix(coefs.ridge,ncol=1),
                      varName=rep(c("Intercept",nms),numLambda),
                      loglambda=rep(rev(expVals),each=9))

ridge.gg <- ggplot(coef.df,aes(loglambda,coef,color=varName))+
    geom_line() +
    scale_x_continuous("Log(Lambda)")+
    scale_color_brewer(palette="Set1")
ridge.gg
```




```
ridge.gg+
    geom_hline(yintercept=mean(train.y),color="black")
```



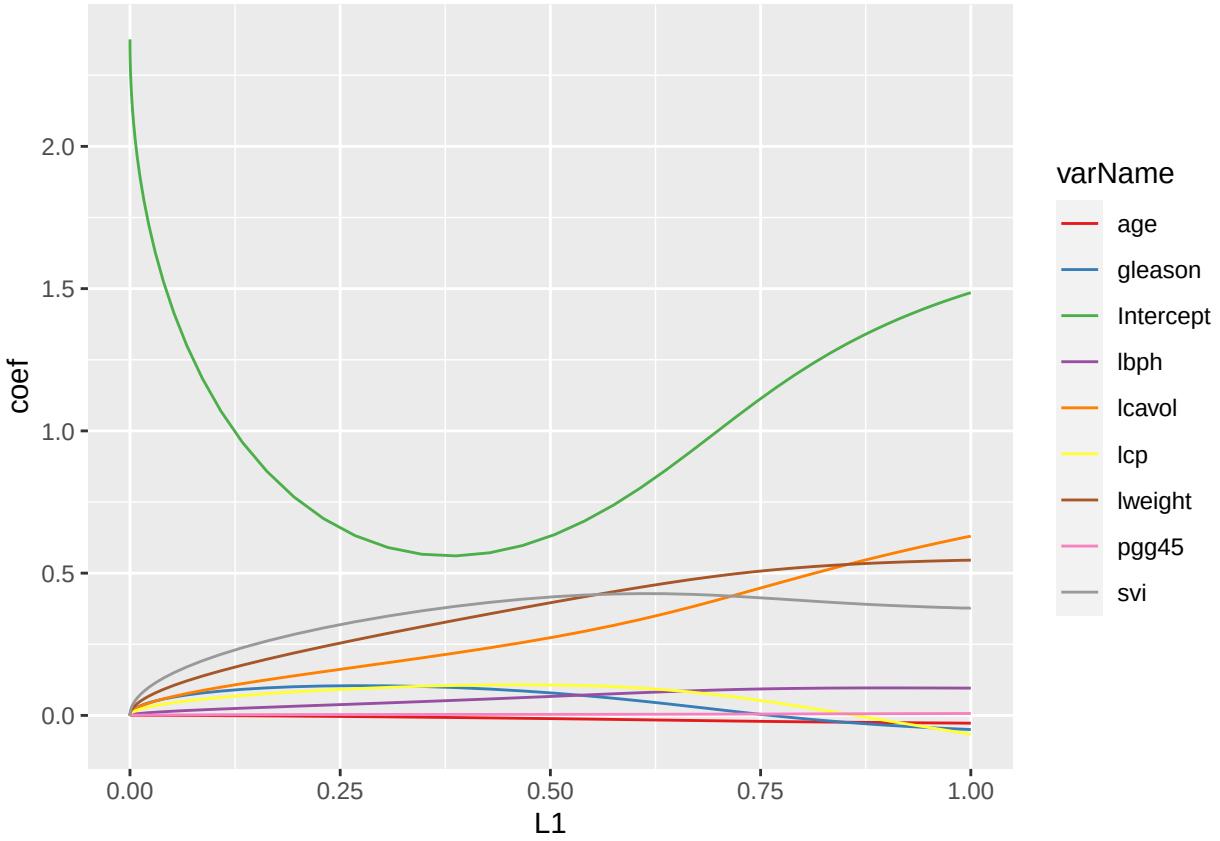
Here's another version, closer to the structure of the built-in glmnet plot command

```
c.df <- coef.df%>%
    group_by(loglambda)%>%
    summarize(bsum=sum((varName!="Intercept")*coef^2))

bsum0 <- with(c.df,max(bsum))
c.df <- mutate(c.df,bsum=bsum/bsum0)

coef.df%>%
    inner_join(c.df)%>%
    ggplot(aes(bsum,coef,color=varName))+
    geom_line() +
    scale_x_continuous("L1")+
    scale_color_brewer(palette="Set1")
```

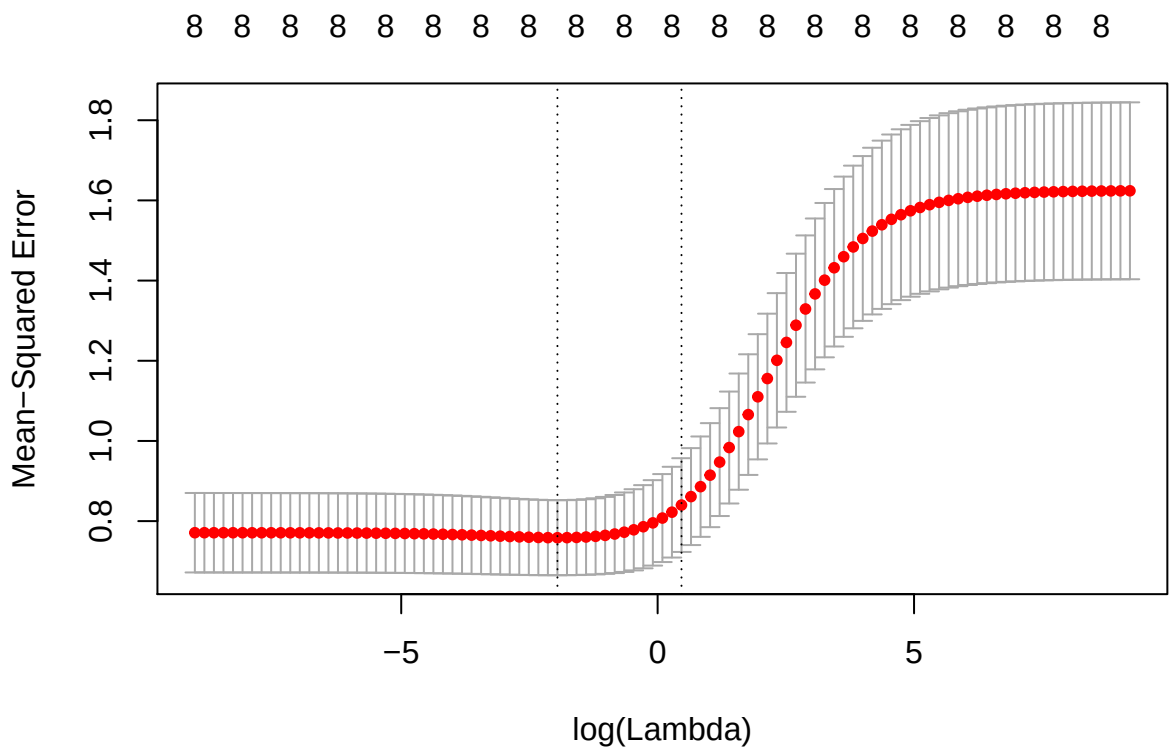
```
## Joining, by = "loglambda"
```



We need to find the optimal lambda. To do so, we cross-validate the mse across all the values of lambda. glmnet will do this automatically

Use grid of lambda values with cv.glmnet. The plot shows cross-validated with error bars as a function of log(lambda) base=2.

```
cv.ridge <- cv.glmnet(train.x,
                      train.y,
                      alpha=0,
                      lambda=lambda.grid,
                      intercept=TRUE)
plot(cv.ridge)
```



Note the vertical lines correspond to these

```
(lamb0 <- cv.ridge$lambda.min)
```

```
## [1] 0.1417474
```

```
(lamb1 <- cv.ridge$lambda.1se)
```

```
## [1] 1.592283
```

```
log(lamb0)
```

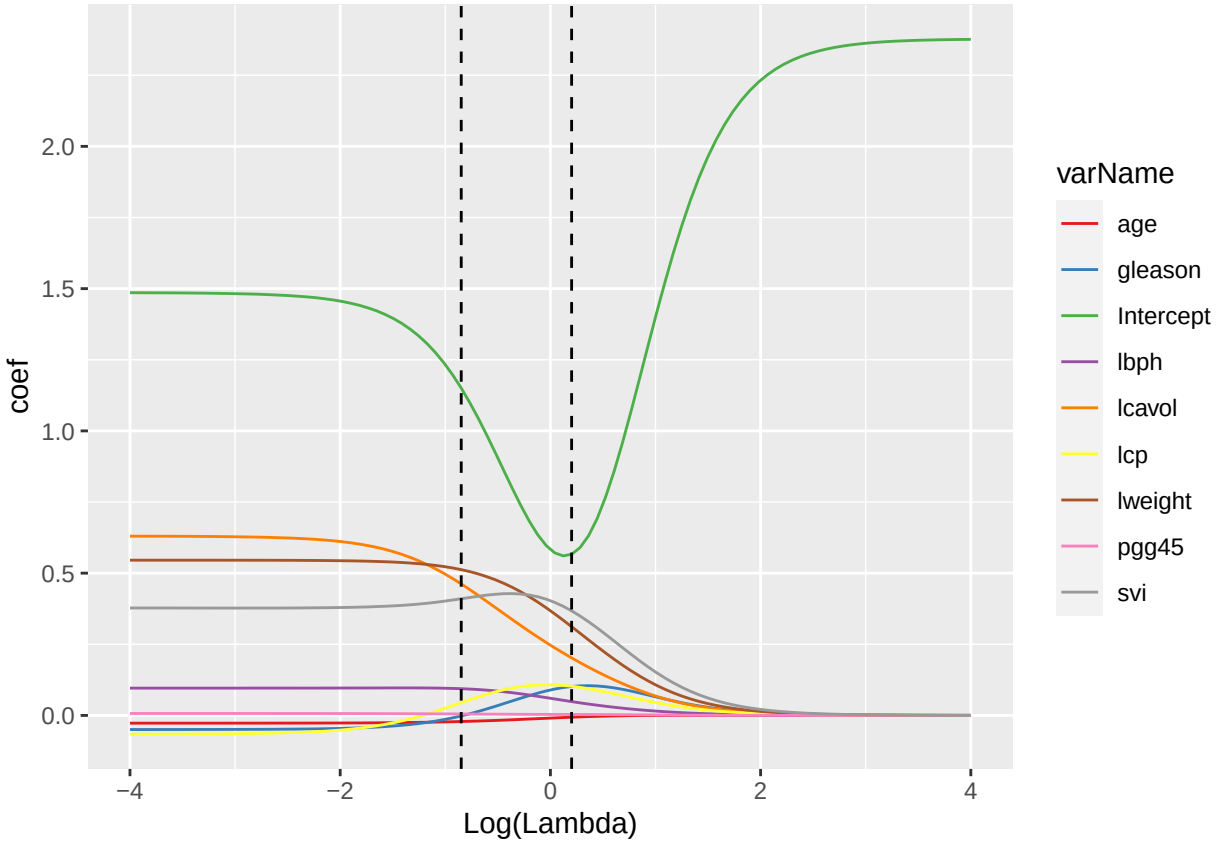
```
## [1] -1.953709
```

```
log(lamb1)
```

```
## [1] 0.4651687
```

add info on lambda opt to our previous plot

```
ridge.gg+
    geom_vline(xintercept=log(lamb0,10),size=.5,linetype="dashed")+
    geom_vline(xintercept=log(lamb1,10),size=.5,linetype="dashed")
```



Now we can see how the shrinkage worked here. lcp svi and gleason seem to be affected the most.

Now compute mse for the optimal lambda value

```
mod.ridge <-
    glmnet(train.x,train.y,alpha=0,lambda=lamb0,intercept=TRUE)
coef.ridge <- as.matrix(coef(mod.ridge))
```

Compare ridge coefficients to liner model coefficients

```
cbind(coef.lm,coef.ridge)
```

```
##                                s0
## (Intercept)  1.486036254  1.150030050
## lcavol       0.630254747  0.461841975
## lweight      0.545589017  0.512207914
## age         -0.027274640 -0.021483428
## lbph         0.095848464  0.093917948
## svi          0.377640666  0.410227418
## lcp         -0.066108379  0.044746986
## gleason     -0.049731252 -0.001437780
## pgg45        0.006744203  0.004945327
```

Ridge prediction for optimal lambda

```
pred.ridge <- predict(mod.ridge,newx=test.x)
mse.ridge <- with(test.df,mean( (pred.ridge-lpsa)^2))
```

Comparison of mse. . . slight improvement

```
c(mse.lm, mse.ridge)
```

```
## [1] 0.3832044 0.3881722
```

Assignment

Recreate plot(cv.ridge) **without using cv.glmnet**.

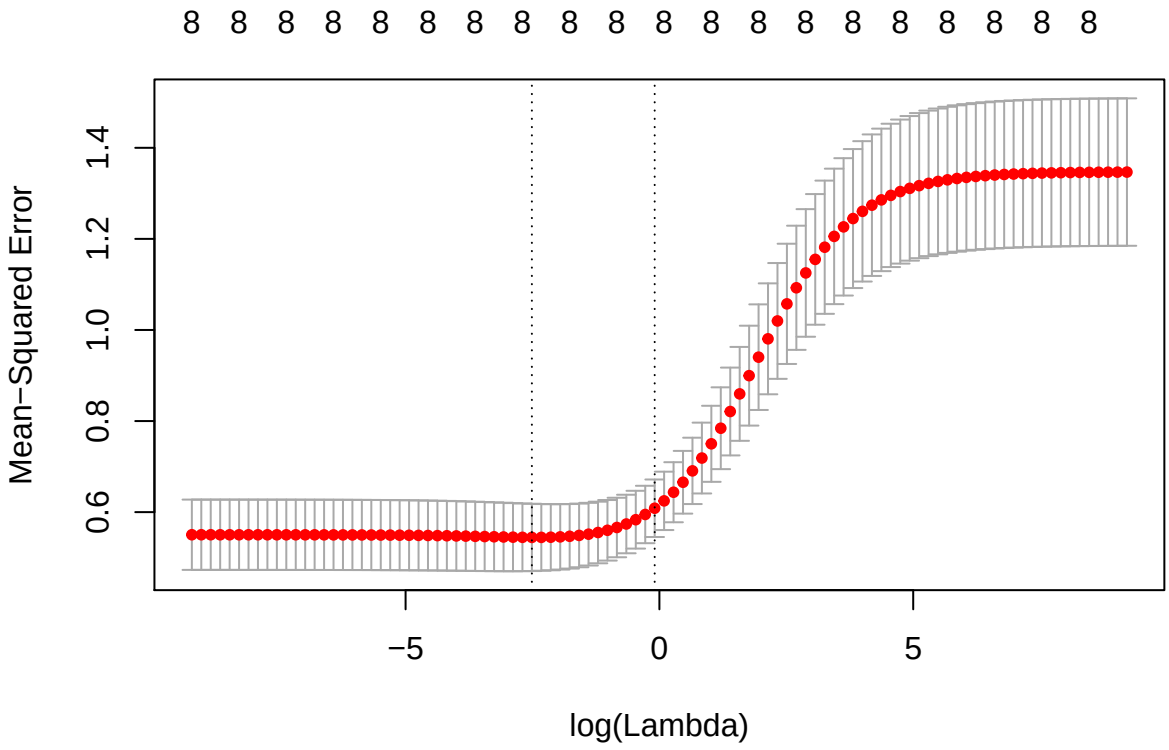
In other words, do your own 10-fold cross-validation.
To do so, loop over glmnet with a single lambda value. Each time, do a 10-fold cross validation (on the training set only) and save both the mean and standard deviation for the mse (for the particular lambda value). From these you should be able to recreate a version of plot(cv.ridge) above.

Note: the standard error of the mean is sd/sqrt(n) where sd is the sample standard deviation and n=sample size. Hence if you have K=10 mse values mse.vals then the se=sd(mse.vals)/sqrt(10). These give the error bars.

Solution

Here's what we're shooting for.

```
cv.ridge <- cv.glmnet(data.x,
                      data.y,
                      alpha=0,
                      lambda=lambda.grid,
                      intercept=TRUE)
plot(cv.ridge)
```



Set up..build everything from scratch just to be sure we have current values.

The data as matrices

```
data.x <- prostate.mat[,-9]
data.y <- prostate.mat[,9]
```

The folds

```
N <- nrow(prostate.df)
numFolds <- 10
```

Create a lambda grid

```
numLambda <- 100
expVals <- seq(-4,4,length=numLambda)
lambda.grid <- 10^expVals
```

Plan: Loop through the lambda grid. At each stop, do a k-fold cross-validation.

Practice: Get started with one value of lambda

```
lambda.val <- lambda.grid[1]
```

Do a 10-fold cross-validation with this lambda value. Compare the predicted values to the test response values to compute the mse. Take the average and standard deviation of these

```
folds <- sample(1:numFolds,N,rep=T)
mseVals <- numeric(numFolds)
```

K-fold....

Note the mean and standard deviation

```
c(mean(mseVals),sd(mseVals))
```

```
## [1] 0.5244482 0.2765014
```

As always, make the process a function.

```
cvLambda <- function(lambda.val){
 #  folds <- sample(1:numFolds,N,rep=T)
 #  mseVals <- numeric(numFolds)
 # .....
 # c(mean(mseVals),sd(mseVals))
}
```

Test it out...just to be sure

```
lambda.val <- lambda.grid[1]
cvLambda(lambda.val)
```

```
## [1] 0.5381973 0.2967458
```

```
lambda.val <- lambda.grid[50]
cvLambda(lambda.val)
```

```
## [1] 0.5898148 0.2885125
```

Build the lambda CV values

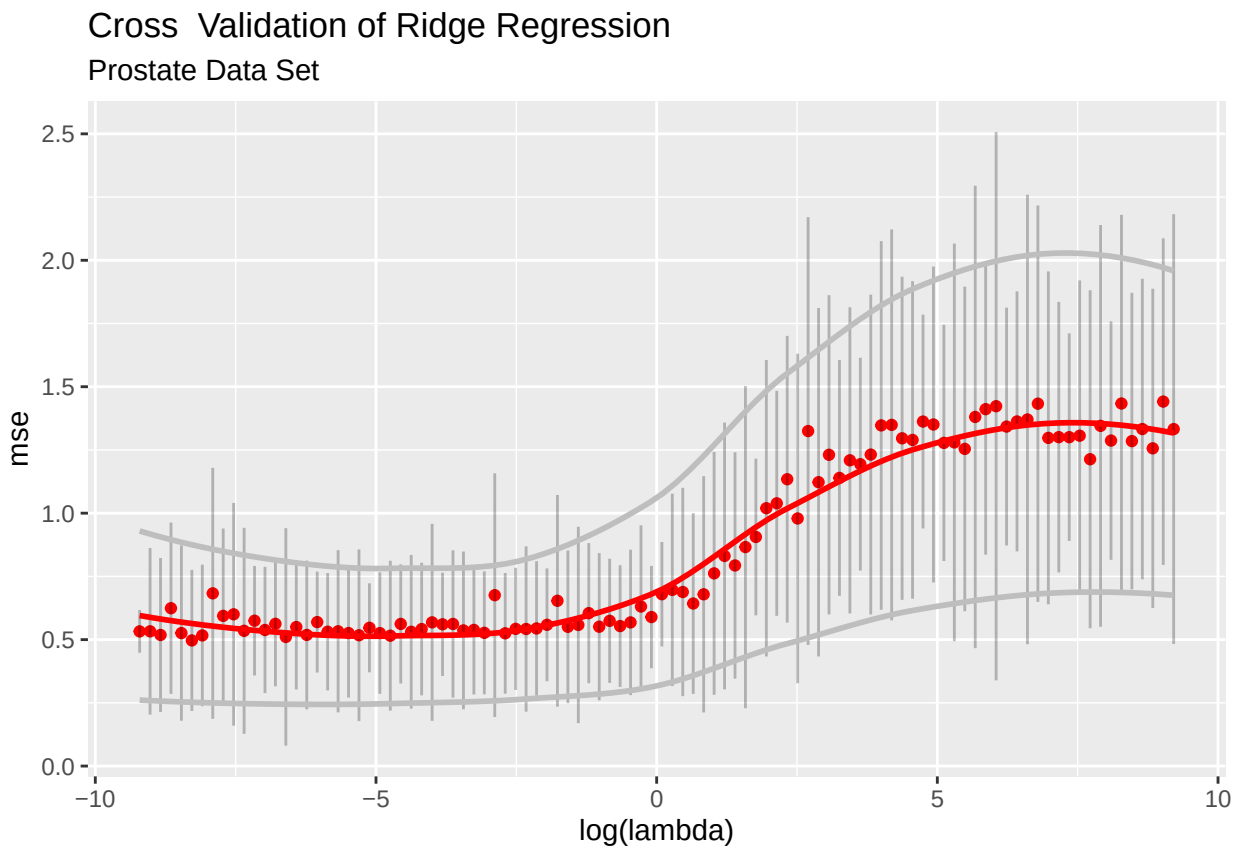
Run this the function over the the lambda grid. Keep track of both the mean and the sd.

Pack into a data frame and plot...

```
cvVals.df <- data.frame(lambda=lambda.grid,
                        mse = cvVals[,1],
                        se =  cvVals[,2])
```

If you can get something that looks like this, you're in good shape

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



Lasso

We can repeat the above analysis using lasso. Now alpha=1

```
mod.lasso <- glmnet(train.x,train.y,alpha=1,lambda=.01,intercept=TRUE)
coef(mod.lasso)
```

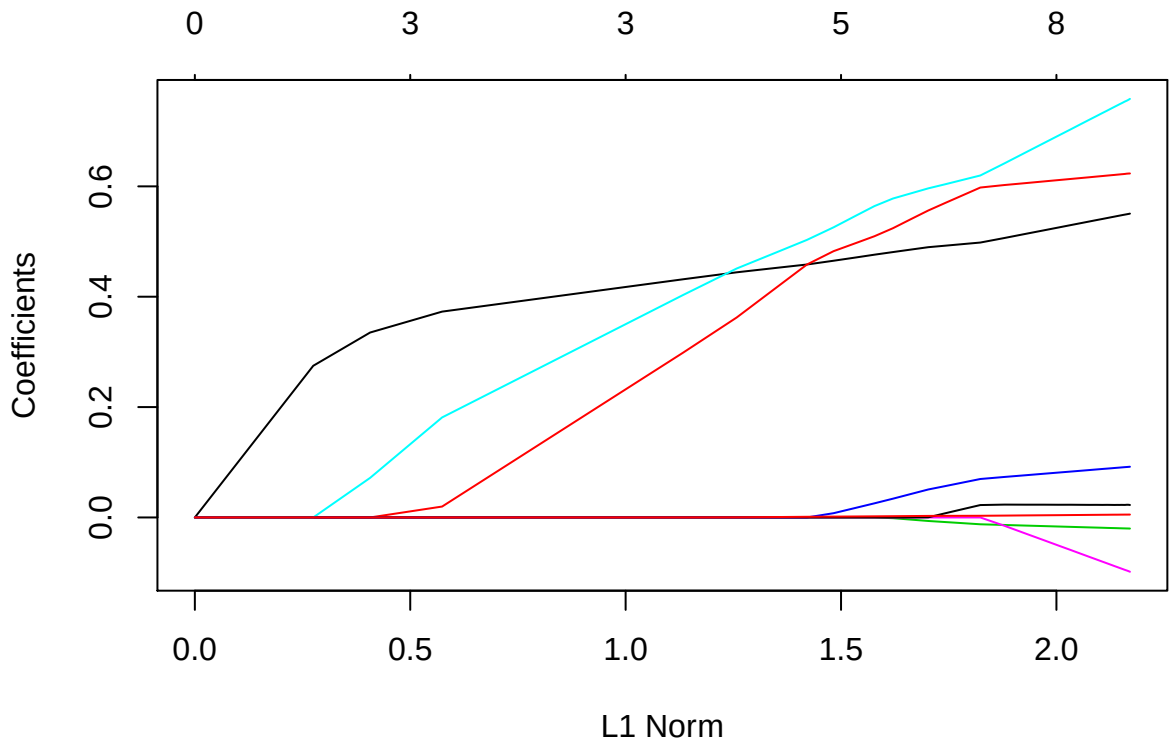
```
## 9 x 1 sparse Matrix of class "dgCMatrix"
##                       s0
## (Intercept)  0.154002508
## lcavol       0.521527080
## lweight      0.609766395
## age         -0.015877899
## lbph         0.079721971
## svi          0.681605493
## lcp         -0.043853021
## gleason      0.023123884
## pgg45        0.004019055
```

Use the grid of lambda values

```
mod.lasso <-
    glmnet(train.x,train.y,alpha=1,lambda=lambda.grid,intercept=TRUE)
```

The built-in plot function. We can see how the coefficients drop out

```
plot(mod.lasso)
```

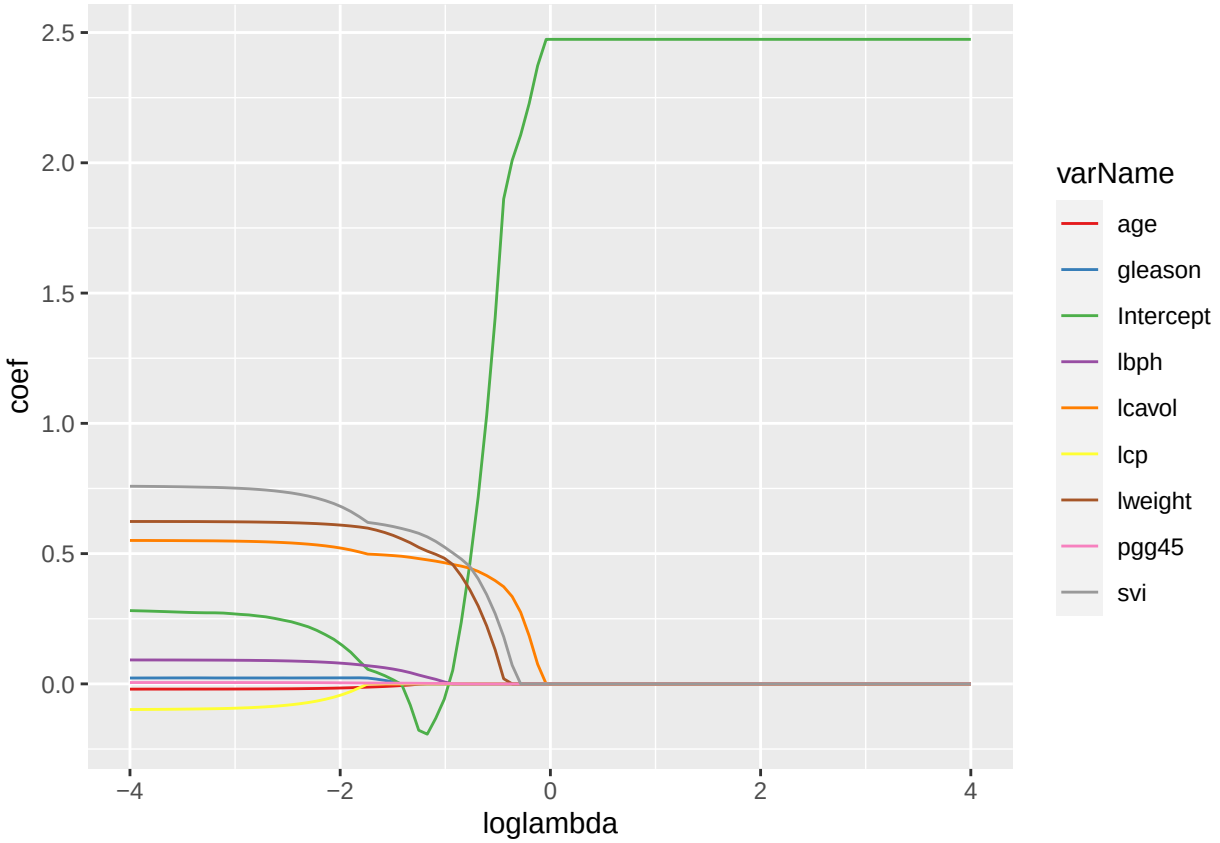


Again, we can add more detail

```
coefs.lasso <- coef(mod.lasso)
coef.df <- data.frame(coef=matrix(coefs.lasso,ncol=1),
                      varName=rep(c("Intercept",nms),numLambda),
                      loglambda=rep(rev(expVals),each=9))
```

We get a similar look at the lasso coefficeints

```
lasso.gg <- ggplot(coef.df,aes(loglambda,coef,color=varName))+geom_line()    +
    scale_color_brewer(palette="Set1")
lasso.gg
```



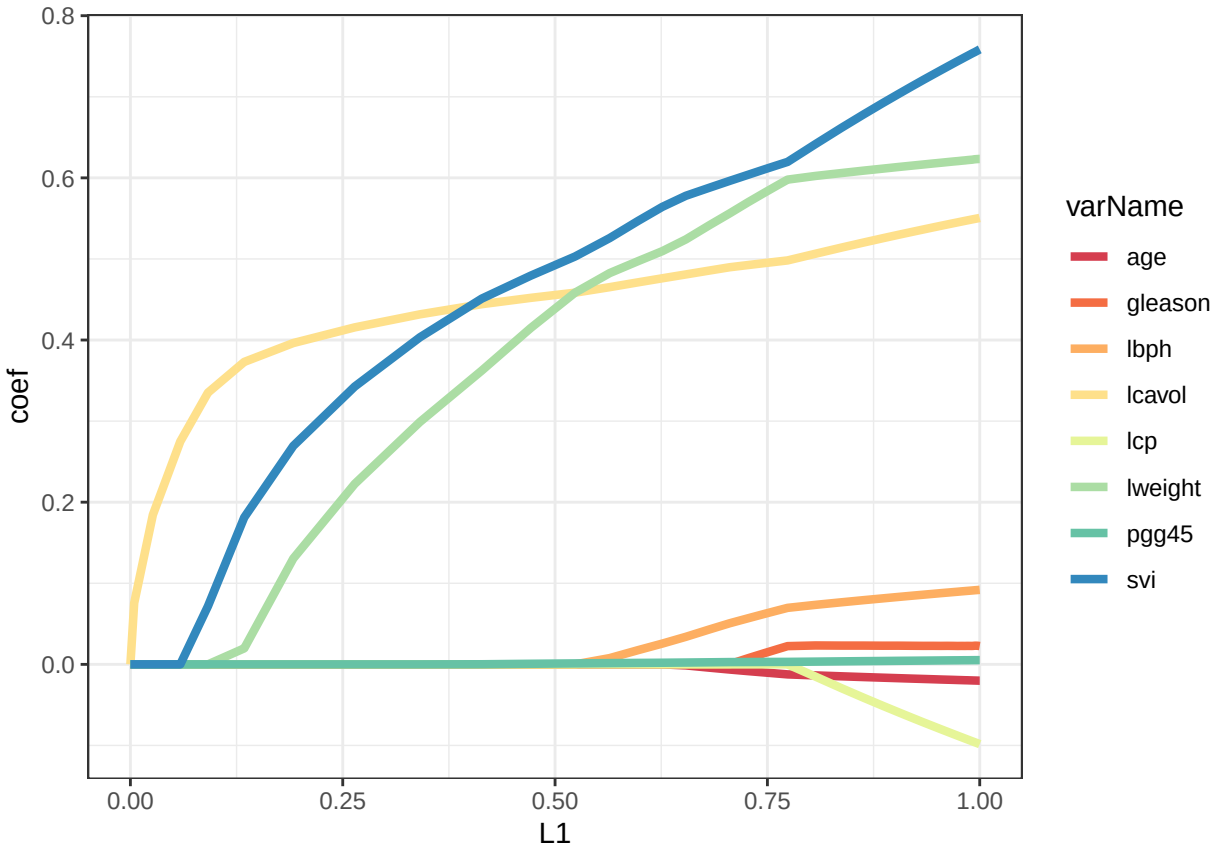

```
c.df <- coef.df%>%
    group_by(loglambda)%>%
    summarize(bsum=sum((varName!="Intercept")*coef^2))

bsum0 <- with(c.df,max(bsum))

lasso.gg1 <- c.df %>%
    mutate(bsum=bsum/bsum0)%>%
    inner_join(coef.df)%>%
    filter(varName!="Intercept")%>%
    ggplot(aes(bsum,coef,color=varName))+
    geom_line(size=1.5) +
    scale_x_continuous("L1")+
    scale_color_brewer(palette="Spectral")+
    theme_bw()
```

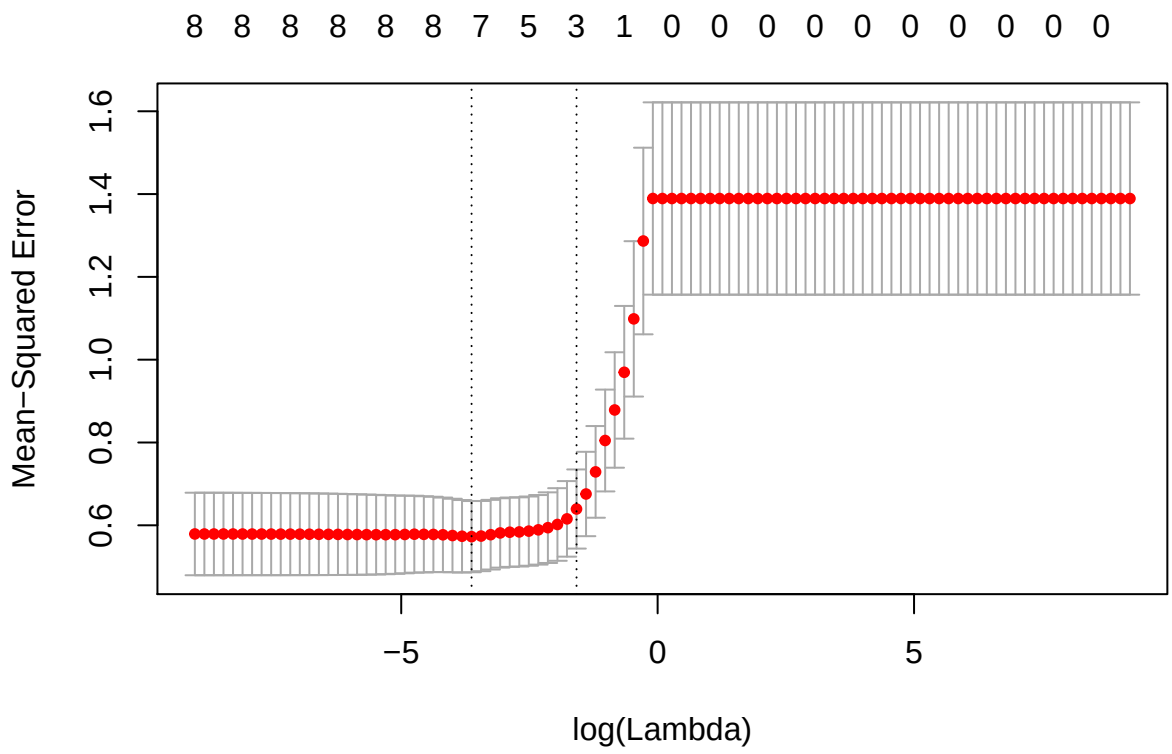
```
## Joining, by = "loglambda"
```

```
lasso.gg1
```



Now we can go full bore and cross-validate

```
cv.lasso <- cv.glmnet(train.x,train.y,alpha=1,lambda=lambda.grid,intercept=TRUE)
plot(cv.lasso)
```



Extract the optimal lambda

```
(lamb0 <- cv.lasso$lambda.min)
```

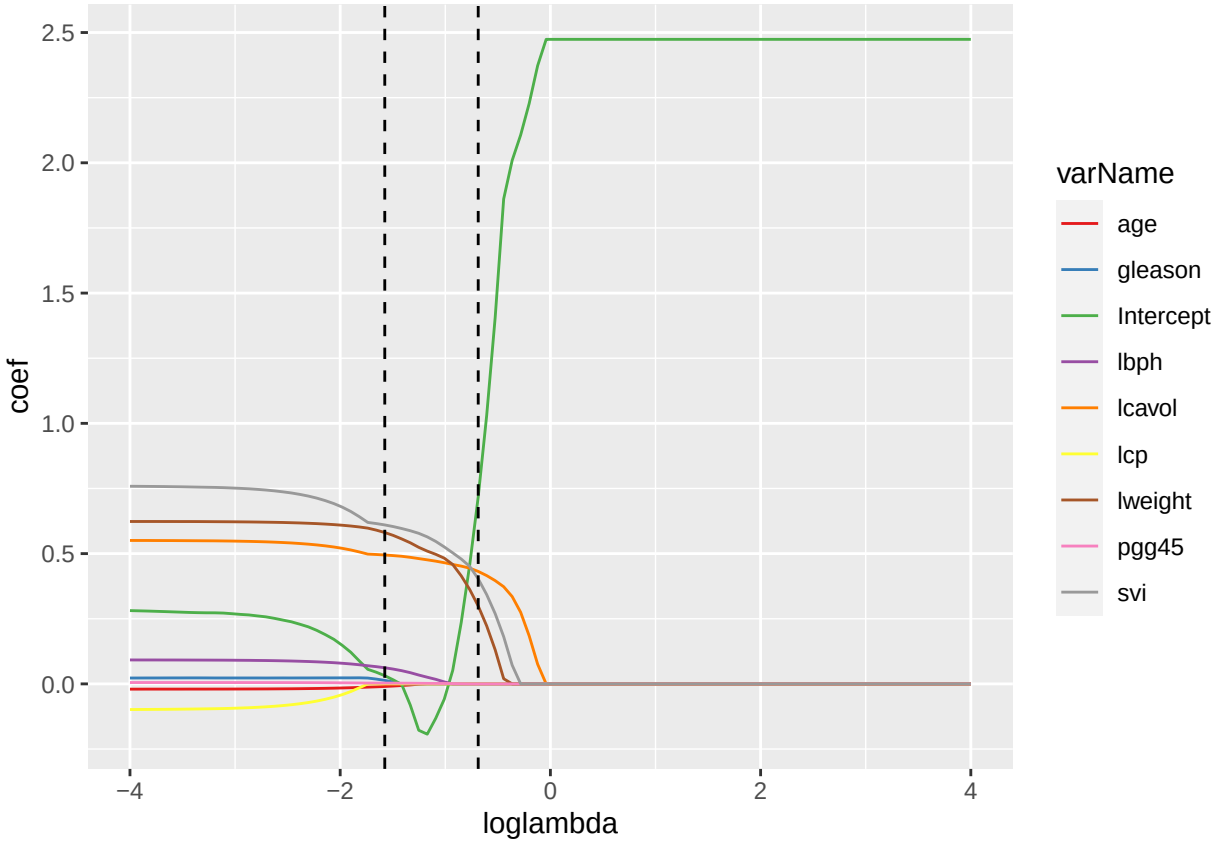
```
## [1] 0.02656088
```

```
(lamb1 <- cv.lasso$lambda.1se)
```

```
## [1] 0.2056512
```

```
mod.lasso <-
    glmnet(train.x,train.y,alpha=1,lambda=lamb0,intercept=TRUE)

lasso.gg +
    geom_vline(xintercept=log(lamb0,10),size=.5,linetype="dashed")+
    geom_vline(xintercept=log(lamb1,10),size=.5,linetype="dashed")
```



Also get the more conservative estimate

```
mod.lasso1 <-
    glmnet(train.x,train.y,alpha=1,lambda=lamb1,intercept=TRUE)
```

Coefficients

```
coef.lasso <- coef(mod.lasso)
coef.lasso1 <- coef(mod.lasso1)
cbind(coef.lm,coef.ridge,coef.lasso1)
```

```
## 9 x 3 sparse Matrix of class "dgCMatrix"
##                                    s0         s0
## (Intercept)  1.486036254  1.150030050 0.7178051
## lcavol       0.630254747  0.461841975 0.4316813
## lweight      0.545589017  0.512207914 0.2989394
## age         -0.027274640 -0.021483428 .
## lbph         0.095848464  0.093917948 .
## svi          0.377640666  0.410227418 0.4033631
## lcp         -0.066108379  0.044746986 .
## gleason     -0.049731252 -0.001437780 .
## pgg45        0.006744203  0.004945327 .
```

And finally, the mse

```
pred.lasso <- predict(mod.lasso,newx=test.x)
pred.lasso1 <- predict(mod.lasso1,newx=test.x)
mse.lasso <- with(test.df,mean( (pred.lasso-test.y)^2))
mse.lasso1 <- with(test.df,mean( (pred.lasso1-test.y)^2))
```

Comparison of errors

```
c(mse.lm,mse.ridge,mse.lasso,mse.lasso1)
```

```
## [1] 0.3832044 0.3881722 0.1367043 0.2062220
```