# LAR and LASSO

Alexander McLain

## LASSO and RR with the Bodyfat Data

This example will use the same bodyfat data from the MMST textbook.

```
bf_dat <- read.csv("bodyfat2.csv")
bf_df <- data.frame(bf_dat)
```

Let's recall the fit of the full model

```
bf_mod <- lm(bodyfat~age + weight + height + neck + chest +
               abdomen + hip + thigh + knee + ankle + biceps +
               forearm + wrist,data = bf_df)
round(coef(summary(bf_mod)),4)
```

| | Estimate | Std. Error | t value | Pr(>\|t\|) |
|---|---|---|---|---|
| (Intercept) | -21.3532 | 22.1862 | -0.9625 | 0.3368 |
| age | 0.0646 | 0.0322 | 2.0058 | 0.0460 |
| weight | -0.0964 | 0.0618 | -1.5584 | 0.1205 |
| height | -0.0439 | 0.1787 | -0.2459 | 0.8060 |
| neck | -0.4755 | 0.2356 | -2.0184 | 0.0447 |
| chest | -0.0172 | 0.1032 | -0.1665 | 0.8679 |
| abdomen | 0.9550 | 0.0902 | 10.5917 | 0.0000 |
| hip | -0.1886 | 0.1448 | -1.3025 | 0.1940 |
| thigh | 0.2483 | 0.1462 | 1.6991 | 0.0906 |
| knee | 0.0139 | 0.2477 | 0.0563 | 0.9552 |
| ankle | 0.1779 | 0.2226 | 0.7991 | 0.4251 |
| biceps | 0.1823 | 0.1725 | 1.0568 | 0.2917 |
| forearm | 0.4557 | 0.1993 | 2.2867 | 0.0231 |
| wrist | -1.6545 | 0.5332 | -3.1032 | 0.0021 |

Now we'll take a look a

```
library(MASS)
```

```
##
## Attaching package: 'MASS'
```

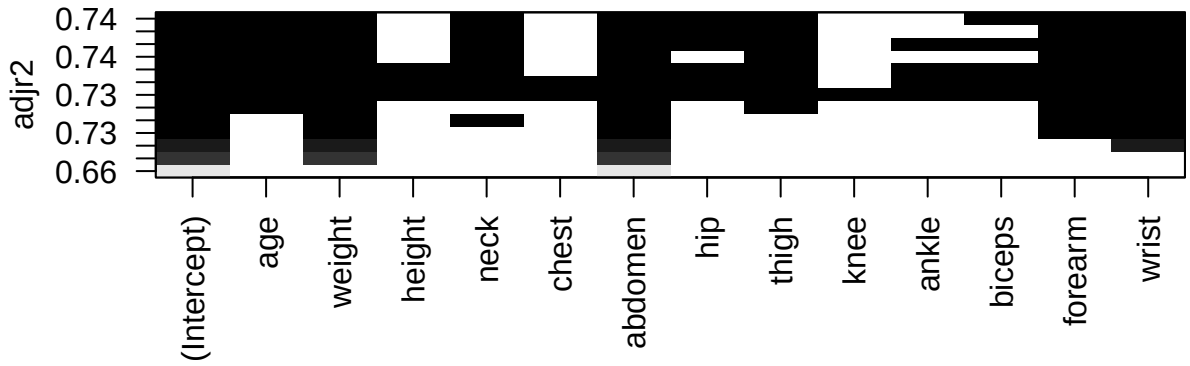
```
## The following object is masked from 'package:dplyr':
##
##     select
```

```
library(leaps)
leaps <- regsubsets(bodyfat~ age + weight + height + neck +
                      chest + abdomen + hip + thigh + knee +
                      ankle + biceps + forearm + wrist, data =
```

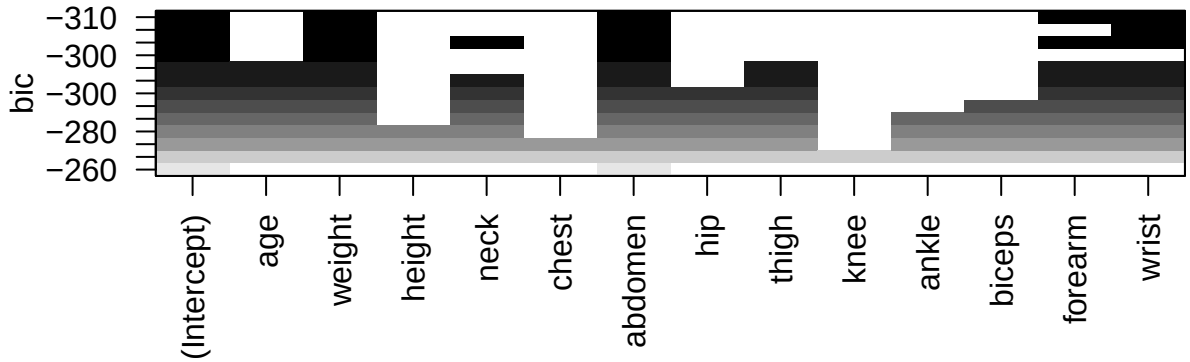
```
                      bf_df, method="exhaustive", nvmax=13)
summary(leaps)
```

```
## Subset selection object
## Call: regsubsets.formula(bodyfat ~ age + weight + height + neck + chest +
##     abdomen + hip + thigh + knee + ankle + biceps + forearm +
##     wrist, data = bf_df, method = "exhaustive", nvmax = 13)
## 13 Variables  (and intercept)
##         Forced in Forced out
## age         FALSE      FALSE
## weight      FALSE      FALSE
## height      FALSE      FALSE
## neck        FALSE      FALSE
## chest       FALSE      FALSE
## abdomen     FALSE      FALSE
## hip         FALSE      FALSE
## thigh       FALSE      FALSE
## knee        FALSE      FALSE
## ankle       FALSE      FALSE
## biceps      FALSE      FALSE
## forearm     FALSE      FALSE
## wrist       FALSE      FALSE
## 1 subsets of each size up to 13
## Selection Algorithm: exhaustive
##           age weight height neck chest abdomen hip thigh knee ankle biceps
## 1  ( 1 )  " " " "    " "    " "  " "   "*"     " " " "  " "  " "   " "
## 2  ( 1 )  " " "*"    " "    " "  " "   "*"     " " " "  " "  " "   " "
## 3  ( 1 )  " " "*"    " "    " "  " "   "*"     " " " "  " "  " "   " "
## 4  ( 1 )  " " "*"    " "    " "  " "   "*"     " " " "  " "  " "   " "
## 5  ( 1 )  " " "*"    " "    "*"  " "   "*"     " " " "  " "  " "   " "
## 6  ( 1 )  "*" "*"    " "    " "  " "   "*"     " " "*"  " "  " "   " "
## 7  ( 1 )  "*" "*"    " "    "*"  " "   "*"     " " "*"  " "  " "   " "
## 8  ( 1 )  "*" "*"    " "    "*"  " "   "*"     "*" "*"  " "  " "   " "
## 9  ( 1 )  "*" "*"    " "    "*"  " "   "*"     "*" "*"  " "  " "   "*"
## 10  ( 1 ) "*" "*"    " "    "*"  " "   "*"     "*" "*"  " "  "*"   "*"
## 11  ( 1 ) "*" "*"    "*"    "*"  " "   "*"     "*" "*"  " "  "*"   "*"
## 12  ( 1 ) "*" "*"    "*"    "*"  "*"   "*"     "*" "*"  " "  "*"   "*"
## 13  ( 1 ) "*" "*"    "*"    "*"  "*"   "*"     "*" "*"  "*"  "*"   "*"
##           forearm wrist
## 1  ( 1 )  " "     " "
## 2  ( 1 )  " "     " "
## 3  ( 1 )  " "     "*"
## 4  ( 1 )  "*"     "*"
## 5  ( 1 )  "*"     "*"
## 6  ( 1 )  "*"     "*"
## 7  ( 1 )  "*"     "*"
## 8  ( 1 )  "*"     "*"
## 9  ( 1 )  "*"     "*"
## 10  ( 1 ) "*"     "*"
## 11  ( 1 ) "*"     "*"
## 12  ( 1 ) "*"     "*"
## 13  ( 1 ) "*"     "*"
```

```
# plot a table of models showing variables in each model.
# models are ordered by the selection statistic.
plot(leaps,scale="adjr2") #plot options are scale=c("bic", "Cp", "adjr2", "r2").
```




```
plot(leaps,scale="bic")
```




Next we'll fit the data using the Least Angle Regression (LAR) algorithm.

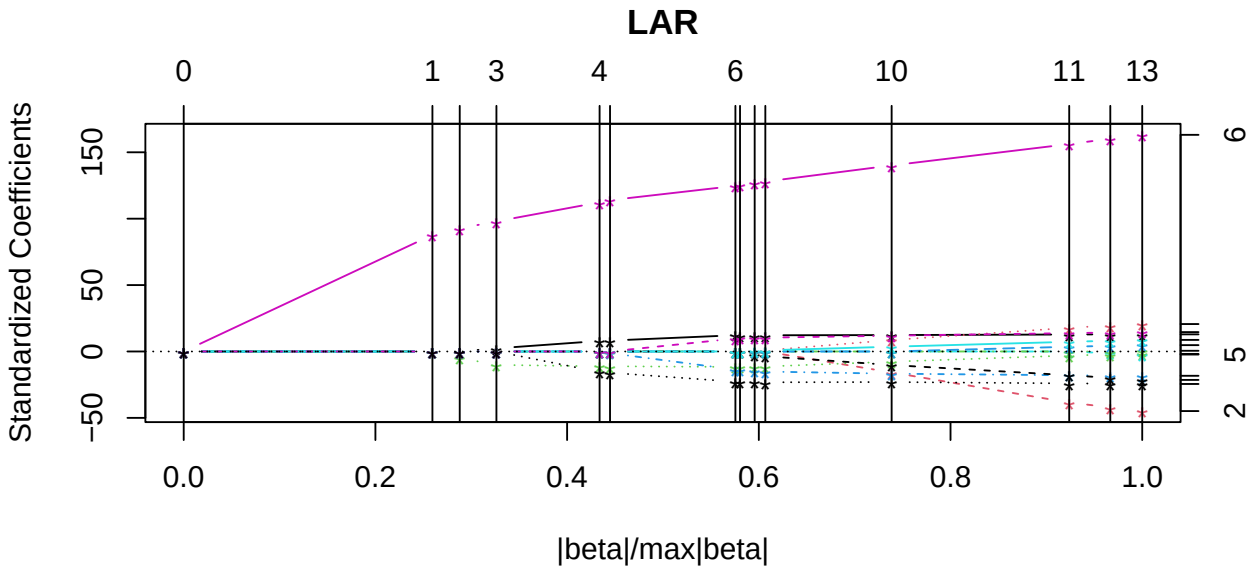
```
library(lars)
```

```
## Loaded lars 1.3
```

```
lars_mod <- lars( as.matrix(bf_df[,3:15]), bf_df$bodyfat,
                  type = "lar")
print(lars_mod)
```

```
##
## Call:
## lars(x = as.matrix(bf_df[, 3:15]), y = bf_df$bodyfat, type = "lar")
## R-squared: 0.749
## Sequence of LAR moves:
##      abdomen height age wrist neck forearm hip biceps thigh weight ankle chest
## Var        6      3   1    13    4      12   7     11     8      2    10     5
## Step       1      2   3     4    5       6   7      8     9     10    11    12
##      knee
## Var     9
## Step   13
```

```
plot(lars_mod)
```




The last fit of the LAR procedure is equivalent to the OLS estimates:

```
cbind(round(lars_mod$beta[14,],4),round(bf_mod$coefficients[-1],4))
```

| | | |
|---|---|---|
| age | 0.0646 | 0.0646 |
| weight | -0.0964 | -0.0964 |
| height | -0.0439 | -0.0439 |
| neck | -0.4755 | -0.4755 |
| chest | -0.0172 | -0.0172 |
| abdomen | 0.9550 | 0.9550 |
| hip | -0.1886 | -0.1886 |
| thigh | 0.2483 | 0.2483 |
| knee | 0.0139 | 0.0139 |
| ankle | 0.1779 | 0.1779 |
| biceps | 0.1823 | 0.1823 |
| forearm | 0.4557 | 0.4557 |
| wrist | -1.6545 | -1.6545 |

Let's see the lasso and ridge procedures.

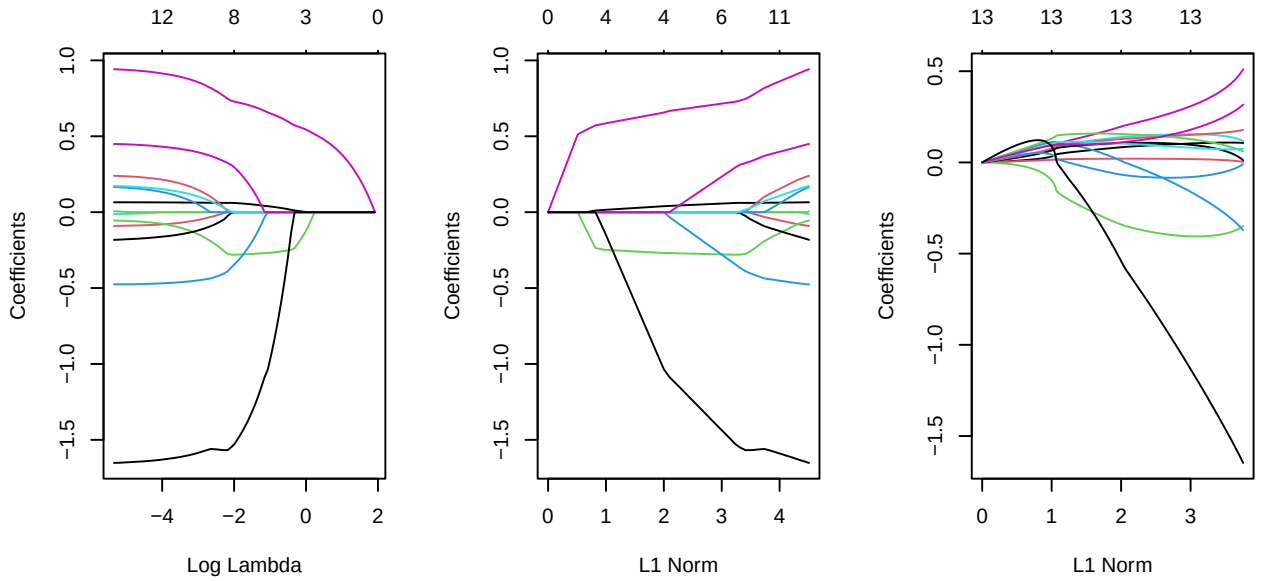
```
library(glmnet)
##Here we fit the LASSO model to our data.  We'll look at this for the
##default nlambda = 100, and lambda.min.ratio = 0.0001.
names(bf_df)
```

```
##  [1] "density" "bodyfat" "age"     "weight"  "height"  "neck"    "chest"
##  [8] "abdomen" "hip"     "thigh"   "knee"    "ankle"   "biceps"  "forearm"
## [15] "wrist"
```

```
lasso_mod <- glmnet(as.matrix(bf_df[,3:15]), bf_df$bodyfat,
                    family="gaussian",alpha=1)
summary(lasso_mod)
```

| | Length | Class | Mode |
|---|---|---|---|
| a0 | 79 | -none- | numeric |
| beta | 1027 | dgCMatrix | S4 |
| df | 79 | -none- | numeric |
| dim | 2 | -none- | numeric |
| lambda | 79 | -none- | numeric |
| dev.ratio | 79 | -none- | numeric |
| nulldev | 1 | -none- | numeric |
| npasses | 1 | -none- | numeric |
| jerr | 1 | -none- | numeric |
| offset | 1 | -none- | logical |
| call | 5 | -none- | call |
| nobs | 1 | -none- | numeric |

```
ridge_mod <- glmnet(as.matrix(bf_df[,3:15]), bf_df$bodyfat,
                    family="gaussian",alpha=0)
par(mfrow=c(1,3))
plot(lasso_mod,xvar="lambda",label=TRUE)
plot(lasso_mod,xvar="norm",label=TRUE)
###Not lets fit the ridge regression model.
plot(ridge_mod,xvar="norm",label=TRUE)
```




```
round(lasso_mod$beta[,32],3) #log(lambda)=-1
```

```
##     age  weight  height    neck   chest abdomen     hip   thigh    knee   ankle
##   0.037   0.000  -0.267   0.000   0.000   0.649   0.000   0.000   0.000   0.000
##  biceps forearm   wrist
##   0.000   0.000  -0.939
```

```
round(lasso_mod$beta[,43],3) #log(lambda)=-2
```

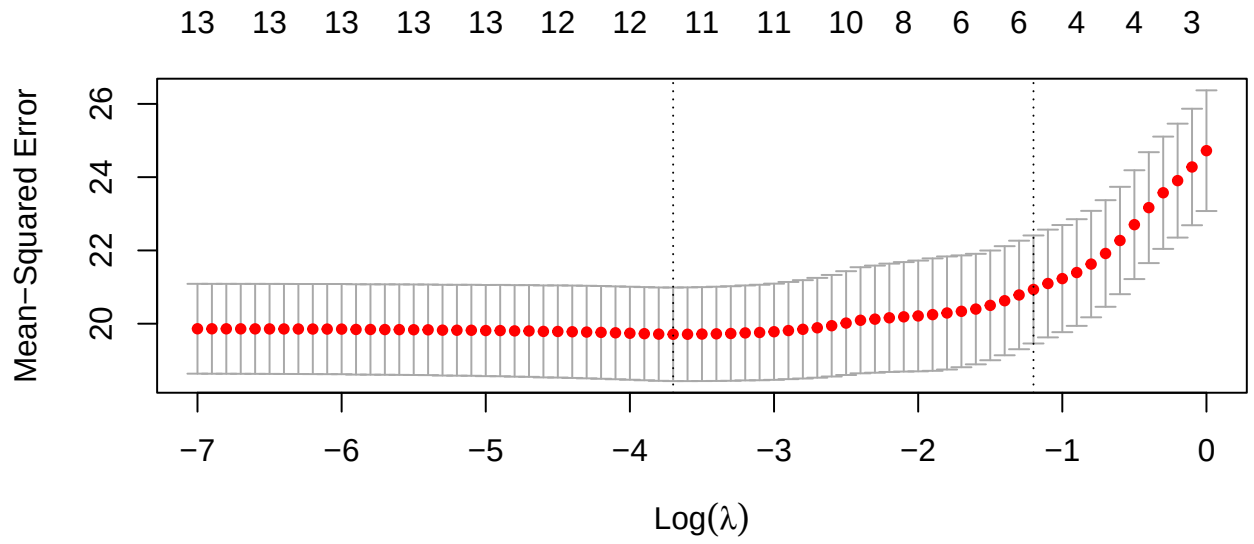
```
##     age  weight  height    neck   chest abdomen     hip   thigh    knee   ankle
##   0.061   0.000  -0.280  -0.343   0.000   0.728   0.000   0.000   0.000   0.000
##  biceps forearm   wrist
##   0.000   0.295  -1.528
```

```
round(lasso_mod$beta[,65],4) #log(lambda)=-4
```

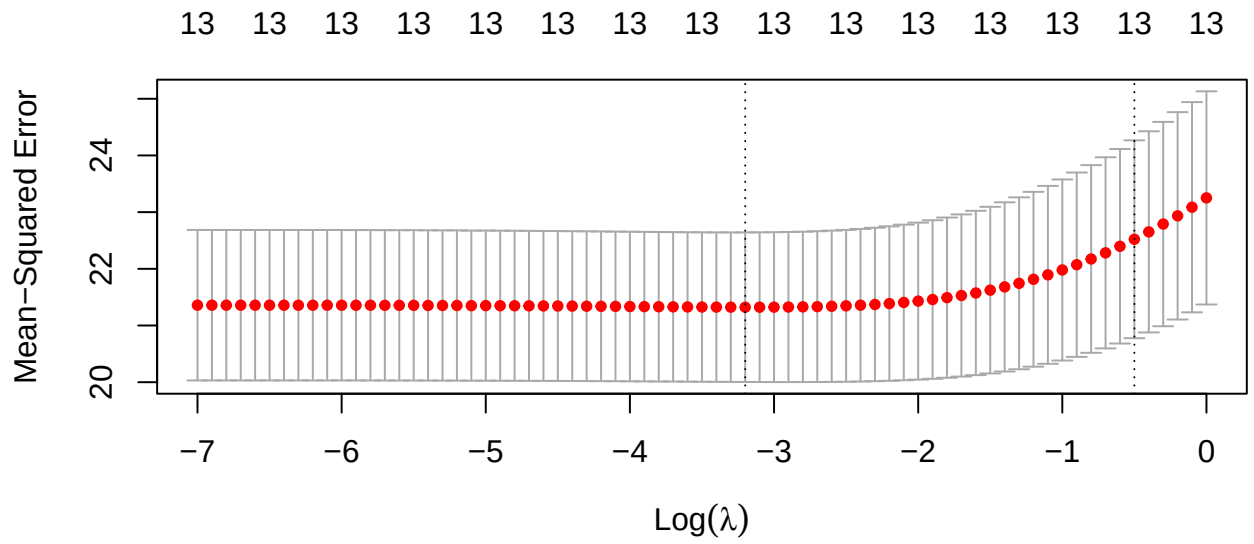
```
##     age  weight  height    neck   chest abdomen     hip   thigh    knee   ankle
##  0.0644 -0.0814 -0.0752 -0.4692 -0.0015  0.9152 -0.1624  0.2158  0.0000  0.1343
##  biceps forearm   wrist
##  0.1528  0.4321 -1.6312
```

Which value of $\lambda$ should we use? To determine that we can use cross-validation.

```
lasso_cv_mod <- cv.glmnet(as.matrix(bf_df[,3:15]), bf_df$bodyfat,
                          family="gaussian",alpha=1, lambda = exp( seq(-7, 0, 0.1)) )
ridge_cv_mod <- cv.glmnet(as.matrix(bf_df[,3:15]), bf_df$bodyfat,
                          family="gaussian",alpha=0, lambda = exp( seq(-7, 0, 0.1)) )

plot(lasso_cv_mod)
```




```
plot(ridge_cv_mod)
```




```
lasso_coef_1se <- as.matrix( coef.glmnet( lasso_cv_mod, s = c("lambda.1se")) )
lasso_coef_min <- as.matrix( coef.glmnet( lasso_cv_mod, s = c("lambda.min")) )
ridge_coef_1se <- as.matrix( coef.glmnet( ridge_cv_mod, s = c("lambda.1se")) )
```

```
ridge_coef_min <- as.matrix( coef.glmnet( ridge_cv_mod, s = c("lambda.min")) )
coef <- data.frame( lasso_coef_1se, lasso_coef_min, ridge_coef_1se, ridge_coef_min)
colnames(coef) <- c("lasso_1se", "lasso_min", "ridge_1se", "ridge_min")
round(coef,3)
```

| | lasso_1se | lasso_min | ridge_1se | ridge_min |
|---|---|---|---|---|
| (Intercept) | -3.978 | -16.301 | -2.042 | -14.687 |
| age | 0.043 | 0.064 | 0.106 | 0.073 |
| weight | 0.000 | -0.075 | 0.000 | -0.071 |
| height | -0.271 | -0.090 | -0.338 | -0.105 |
| neck | -0.057 | -0.465 | -0.386 | -0.485 |
| chest | 0.000 | 0.000 | 0.111 | -0.005 |
| abdomen | 0.672 | 0.902 | 0.532 | 0.891 |
| hip | 0.000 | -0.153 | 0.001 | -0.175 |
| thigh | 0.000 | 0.201 | 0.181 | 0.237 |
| knee | 0.000 | 0.000 | 0.054 | 0.011 |
| ankle | 0.000 | 0.115 | -0.001 | 0.147 |
| biceps | 0.000 | 0.142 | 0.077 | 0.158 |
| forearm | 0.025 | 0.423 | 0.328 | 0.444 |
| wrist | -1.123 | -1.620 | -1.671 | -1.700 |

## Grouped LASSO

This will be a quick example that uses grouped lasso with the `gglasso` package.

```
library(gglasso)
`?`(gglasso)
```

```
Fits the regularization paths for group-lasso penalized learning
problems

Description:

     Fits regularization paths for group-lasso penalized learning
     problems at a sequence of regularization parameters lambda.

Usage:

     gglasso(
       x,
       y,
       group = NULL,
       loss = c("ls", "logit", "sqsvm", "hsvm", "wls"),
       nlambda = 100,
       lambda.factor = ifelse(nobs < nvars, 0.05, 0.001),
       lambda = NULL,
       pf = sqrt(bs),
       weight = NULL,
       dfmax = as.integer(max(group)) + 1,
       pmax = min(dfmax * 1.2, as.integer(max(group))),
       eps = 1e-08,
       maxit = 3e+08,
```

```
       delta,
       intercept = TRUE
     )

Arguments:

       x: matrix of predictors, of dimension n*p; each row is an
          observation vector.

       y: response variable. This argument should be quantitative for
          regression (least squares), and a two-level factor for
          classification (logistic model, huberized SVM, squared SVM).

   group: a vector of consecutive integers describing the grouping of
          the coefficients (see example below).

    loss: a character string specifying the loss function to use, valid
          options are:

            • '"ls"' least squares loss (regression),

            • '"logit"' logistic loss (classification).

            • '"hsvm"' Huberized squared hinge loss (classification),

            • '"sqsvm"' Squared hinge loss (classification),

          Default is '"ls"'.

 nlambda: the number of 'lambda' values - default is 100.

lambda.factor: the factor for getting the minimal lambda in 'lambda'
          sequence, where 'min(lambda)' = 'lambda.factor' *
          'max(lambda)'.  'max(lambda)' is the smallest value of
          'lambda' for which all coefficients are zero. The default
          depends on the relationship between n (the number of rows in
          the matrix of predictors) and p (the number of predictors).
          If n >= p, the default is '0.001', close to zero.  If n<p,
          the default is '0.05'.  A very small value of 'lambda.factor'
          will lead to a saturated fit. It takes no effect if there is
          user-defined 'lambda' sequence.

  lambda: a user supplied 'lambda' sequence. Typically, by leaving this
          option unspecified users can have the program compute its own
          'lambda' sequence based on 'nlambda' and 'lambda.factor'.
          Supplying a value of 'lambda' overrides this. It is better to
          supply a decreasing sequence of 'lambda' values than a single
          (small) value, if not, the program will sort user-defined
          'lambda' sequence in decreasing order automatically.

      pf: penalty factor, a vector in length of bn (bn is the total
          number of groups). Separate penalty weights can be applied to
          each group of beta'ss to allow differential shrinkage. Can be
          0 for some groups, which implies no shrinkage, and results in
```

```
          that group always being included in the model. Default value
          for each entry is the square-root of the corresponding size
          of each group.

  weight: a nxn observation weight matrix in the where n is the number
          of observations. Only used if 'loss='wls'' is specified. Note
          that cross-validation is NOT IMPLEMENTED for 'loss='wls''.

   dfmax: limit the maximum number of groups in the model. Useful for
          very large 'bs' (group size), if a partial path is desired.
          Default is 'bs+1'.

    pmax: limit the maximum number of groups ever to be nonzero. For
          example once a group enters the model, no matter how many
          times it exits or re-enters model through the path, it will
          be counted only once. Default is 'min(dfmax*1.2,bs)'.

     eps: convergence termination tolerance. Defaults value is '1e-8'.

   maxit: maximum number of outer-loop iterations allowed at fixed
          lambda value. Default is 3e8. If models do not converge,
          consider increasing 'maxit'.

   delta: the parameter delta in '"hsvm"' (Huberized squared hinge
          loss). Default is 1.

intercept: Whether to include intercept in the model. Default is TRUE.
```

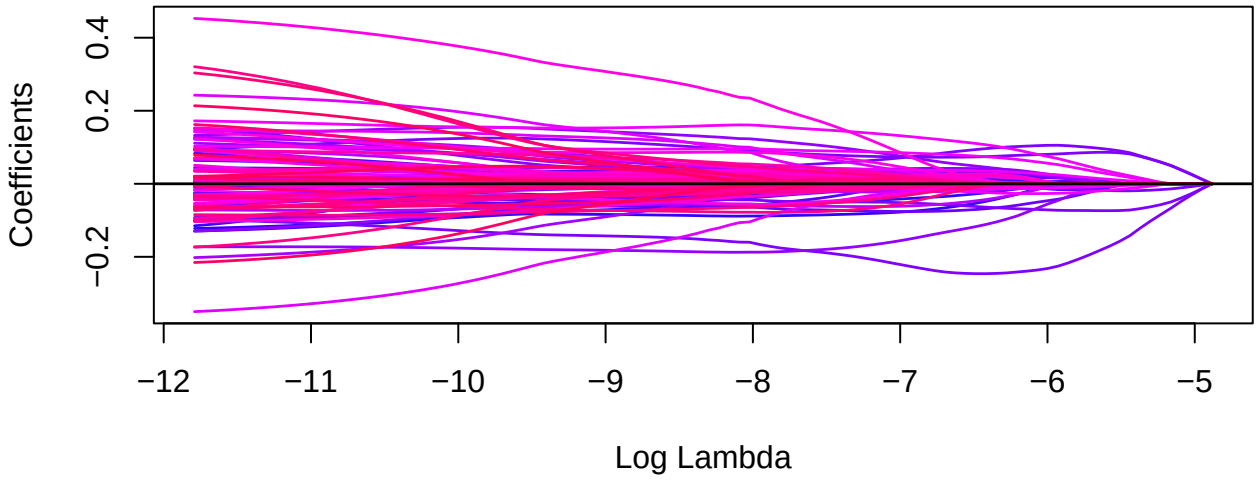
```
data(bardet)
`?`(bardet)
```

```
Simplified gene expression data from Scheetz et al. (2006)

Description:

     Gene expression data (20 genes for 120 samples) from the
     microarray experiments of mammalian eye tissue samples of Scheetz
     et al. (2006).

Details:

     This data set contains 120 samples with 100 predictors (expanded
     from 20 genes using 5 basis B-splines, as described in Yang, Y.
     and Zou, H. (2015)).

Value:

     A list with the following elements:

       x: a [120 x 100] matrix (expanded from a [120 x 20] matrix)
          giving the expression levels of 20 filtered genes for the 120
          samples. Each row corresponds to a subject, each 5
          consecutive columns to a grouped gene.

       y: a numeric vector of length 120 giving expression level of
```

```
gene TRIM32, which causes Bardet-Biedl syndrome.
```

Now we'll fit the group lasso to the bardet data

```
# define group index
group1 <- rep(1:20, each = 5)

# fit group lasso penalized least squares
m1 <- gglasso(x = bardet$x, y = bardet$y, group = group1, loss = "ls")
plot(m1)
```




Now we'll do some CV to properly choose the value of $\lambda$.

```
cv <- cv.gglasso(x = bardet$x, y = bardet$y, group = group1, loss = "ls", pred.loss = "L2",
    lambda.factor = 0.05, nfolds = 5)
cv$lambda.min
```

```
## [1] 0.0004681541
```

```
cv$lambda.1se
```

```
## [1] 0.007575771
```

```
cbind(c(0, group1), c(coef(cv, s = "lambda.1se")), c(coef(cv, s = "lambda.min")))
```

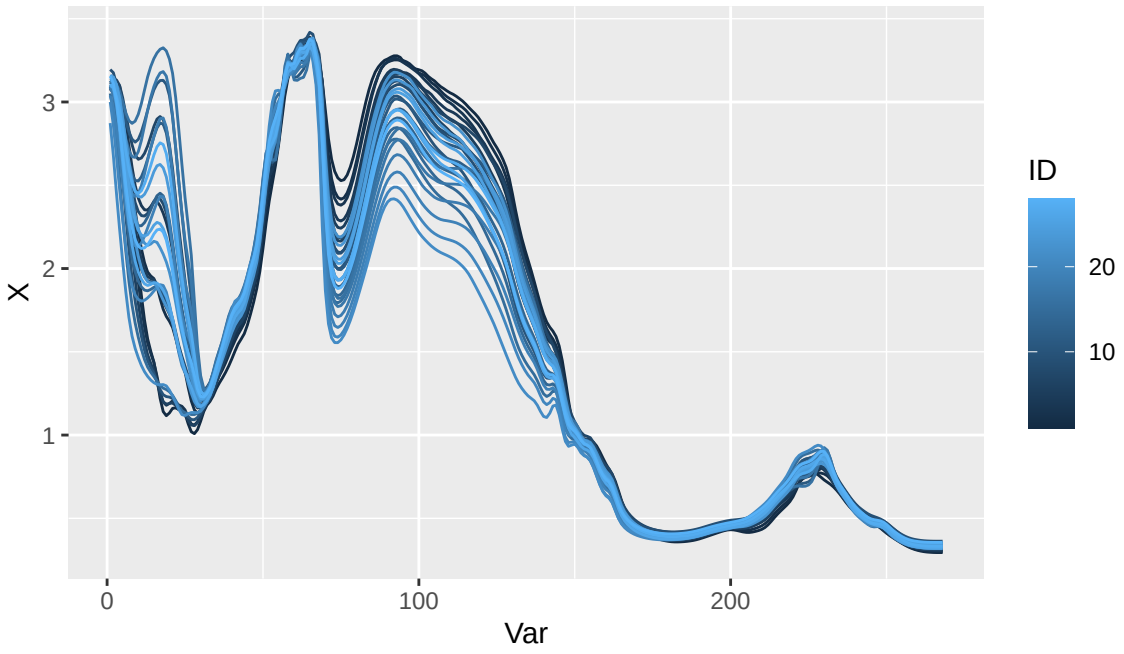
| | | |
|---|---|---|
| 0 | 8.390844 | 8.2173271 |
| 1 | 0.000000 | -0.0137397 |
| 1 | 0.000000 | -0.0523389 |
| 1 | 0.000000 | 0.0356671 |
| 1 | 0.000000 | 0.0090829 |
| 1 | 0.000000 | -0.0838165 |
| 2 | 0.000000 | 0.0002781 |
| 2 | 0.000000 | 0.0017139 |
| 2 | 0.000000 | 0.0012646 |
| 2 | 0.000000 | 0.0001063 |
| 2 | 0.000000 | -0.0028337 |
| 3 | 0.000000 | 0.0220613 |
| 3 | 0.000000 | -0.0254643 |
| 3 | 0.000000 | -0.0007793 |
| 3 | 0.000000 | 0.0028207 |
| 3 | 0.000000 | -0.0452385 |
| 4 | 0.000000 | 0.0046160 |

| | | |
|---|---|---|
| 4 | 0.000000 | 0.0442182 |
| 4 | 0.000000 | -0.0227353 |
| 4 | 0.000000 | -0.0058258 |
| 4 | 0.000000 | -0.0702611 |
| 5 | 0.000000 | 0.0508213 |
| 5 | 0.000000 | 0.0818889 |
| 5 | 0.000000 | -0.0491862 |
| 5 | 0.000000 | -0.0181138 |
| 5 | 0.000000 | -0.1756574 |
| 6 | 0.000000 | 0.1123866 |
| 6 | 0.000000 | 0.0339147 |
| 6 | 0.000000 | -0.0271549 |
| 6 | 0.000000 | 0.0083516 |
| 6 | 0.000000 | -0.1898404 |
| 7 | 0.000000 | -0.0013961 |
| 7 | 0.000000 | 0.0005563 |
| 7 | 0.000000 | 0.0006054 |
| 7 | 0.000000 | 0.0010068 |
| 7 | 0.000000 | 0.0014450 |
| 8 | 0.000000 | 0.0357903 |
| 8 | 0.000000 | -0.0202496 |
| 8 | 0.000000 | 0.0076874 |
| 8 | 0.000000 | 0.0117014 |
| 8 | 0.000000 | -0.0607913 |
| 9 | 0.000000 | -0.0004775 |
| 9 | 0.000000 | -0.0006332 |
| 9 | 0.000000 | 0.0004213 |
| 9 | 0.000000 | 0.0015303 |
| 9 | 0.000000 | 0.0006469 |
| 10 | 0.000000 | -0.0639107 |
| 10 | 0.000000 | 0.0487546 |
| 10 | 0.000000 | 0.0760078 |
| 10 | 0.000000 | 0.0110616 |
| 10 | 0.000000 | 0.0166093 |
| 11 | 0.000000 | -0.0126953 |
| 11 | 0.000000 | 0.0017876 |
| 11 | 0.000000 | 0.0898015 |
| 11 | 0.000000 | 0.1500386 |
| 11 | 0.000000 | 0.0413499 |
| 12 | 0.000000 | 0.0000000 |
| 12 | 0.000000 | 0.0000000 |
| 12 | 0.000000 | 0.0000000 |
| 12 | 0.000000 | 0.0000000 |
| 12 | 0.000000 | 0.0000000 |
| 13 | 0.000000 | -0.0520235 |
| 13 | 0.000000 | 0.0262814 |
| 13 | 0.000000 | -0.0005512 |
| 13 | 0.000000 | 0.0788385 |
| 13 | 0.000000 | 0.1842589 |
| 14 | 0.000000 | -0.0261599 |
| 14 | 0.000000 | 0.0062349 |
| 14 | 0.000000 | 0.0259086 |
| 14 | 0.000000 | 0.0338999 |

| | | |
|---|---|---|
| 14 | 0.000000 | 0.0037641 |
| 15 | 0.000000 | -0.0095728 |
| 15 | 0.000000 | -0.0039897 |
| 15 | 0.000000 | 0.0310495 |
| 15 | 0.000000 | -0.0030660 |
| 15 | 0.000000 | 0.0132002 |
| 16 | 0.000000 | -0.0182053 |
| 16 | 0.000000 | 0.0172314 |
| 16 | 0.000000 | 0.0460860 |
| 16 | 0.000000 | -0.0025930 |
| 16 | 0.000000 | 0.0182979 |
| 17 | 0.000000 | 0.0051751 |
| 17 | 0.000000 | -0.0138591 |
| 17 | 0.000000 | 0.0070085 |
| 17 | 0.000000 | 0.0067323 |
| 17 | 0.000000 | -0.0220390 |
| 18 | 0.000000 | 0.0094923 |
| 18 | 0.000000 | -0.0584266 |
| 18 | 0.000000 | 0.0092960 |
| 18 | 0.000000 | 0.0365321 |
| 18 | 0.000000 | -0.0790914 |
| 19 | 0.000000 | 0.0048644 |
| 19 | 0.000000 | -0.0012055 |
| 19 | 0.000000 | -0.0018575 |
| 19 | 0.000000 | 0.0039680 |
| 19 | 0.000000 | -0.0068155 |
| 20 | 0.000000 | 0.0000000 |
| 20 | 0.000000 | 0.0000000 |
| 20 | 0.000000 | 0.0000000 |
| 20 | 0.000000 | 0.0000000 |
| 20 | 0.000000 | 0.0000000 |

## Re-analysis of the PET data

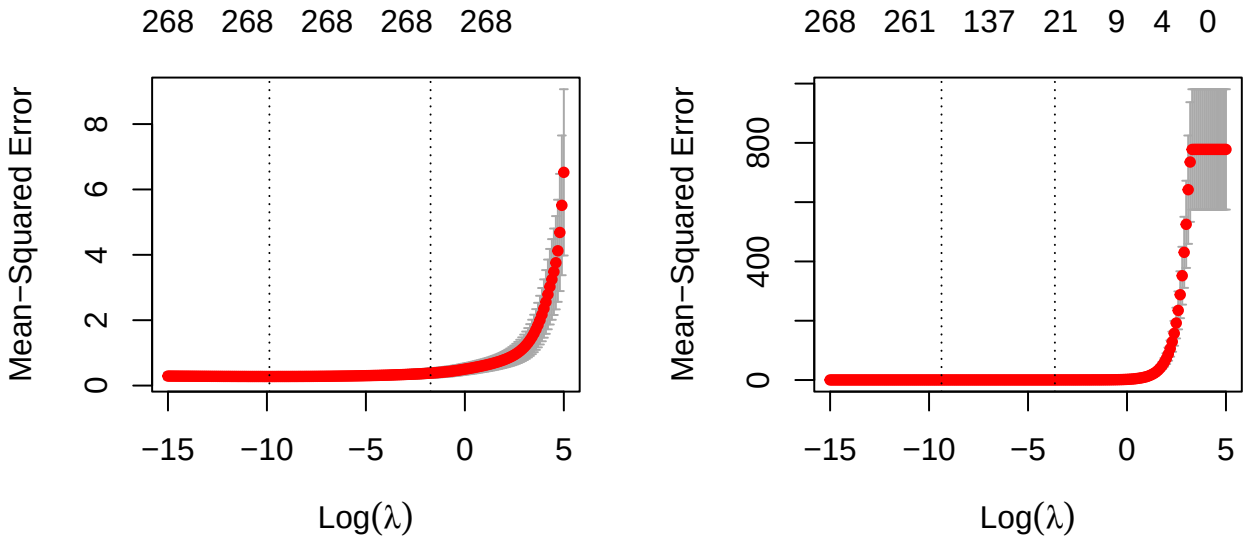
```
## [1]  28 269
```




Now lets use `cv.glmnet` to fit ridge regression and lasso models.

```
ridge_cv_mod <- cv.glmnet(pet_mat[, 1:268], pet_mat[, 269], family = "gaussian",
    alpha = 0, lambda = exp(seq(-15, 5, length.out = 200)), grouped = FALSE)
```

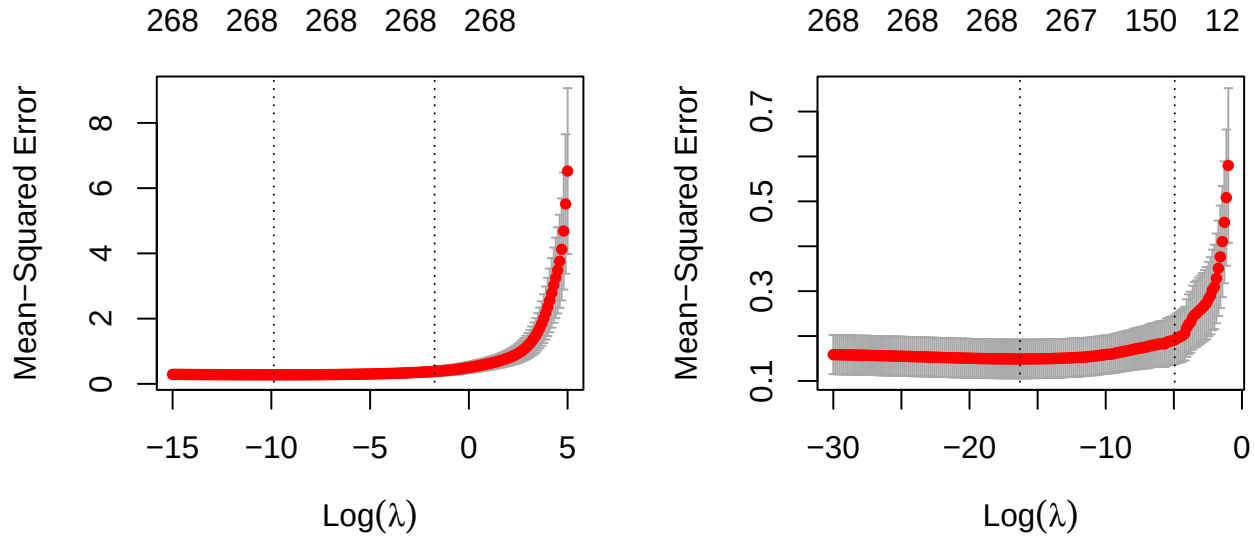
```
lasso_cv_mod <- cv.glmnet(pet_mat[, 1:268], pet_mat[, 269], family = "gaussian",
    alpha = 1, lambda = exp(seq(-15, 5, length.out = 200)), grouped = FALSE)
par(mfrow = c(1, 2))
plot(ridge_cv_mod)
plot(lasso_cv_mod)
```




We can see that the best lasso model had the smallest possible value of $\lambda$. As a result, we'll fit again with some smaller values.

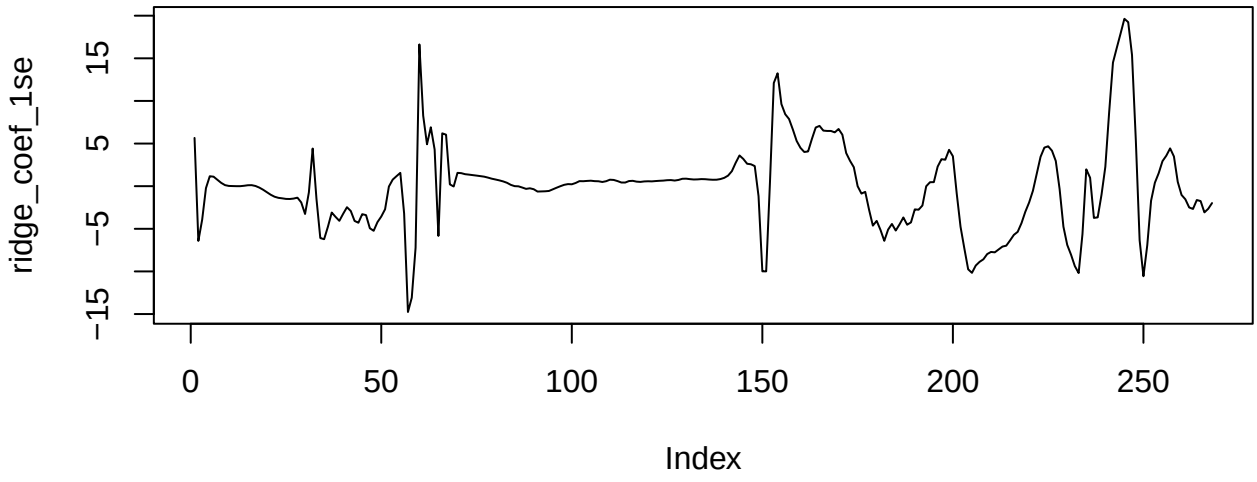
```
lasso_cv_mod <- cv.glmnet(pet_mat[, 1:268], pet_mat[, 269], family = "gaussian",
    alpha = 1, lambda = exp(seq(-30, -1, length.out = 200)), grouped = FALSE)
par(mfrow = c(1, 2))
```

```
plot(ridge_cv_mod)
plot(lasso_cv_mod)
```

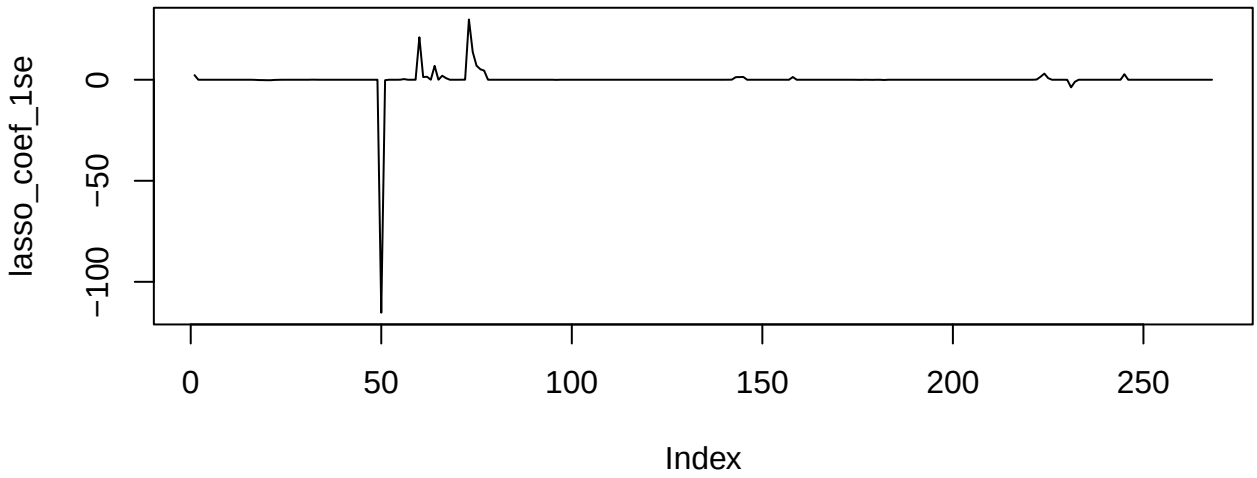



Let's look at the results:

```
ridge_coef_1se <- coef.glmnet( ridge_cv_mod,
                               s = c("lambda.1se"))[-1]
lasso_coef_1se <- coef.glmnet( lasso_cv_mod,
                               s = c("lambda.1se"))[-1]

plot(ridge_coef_1se, type = "l")
```

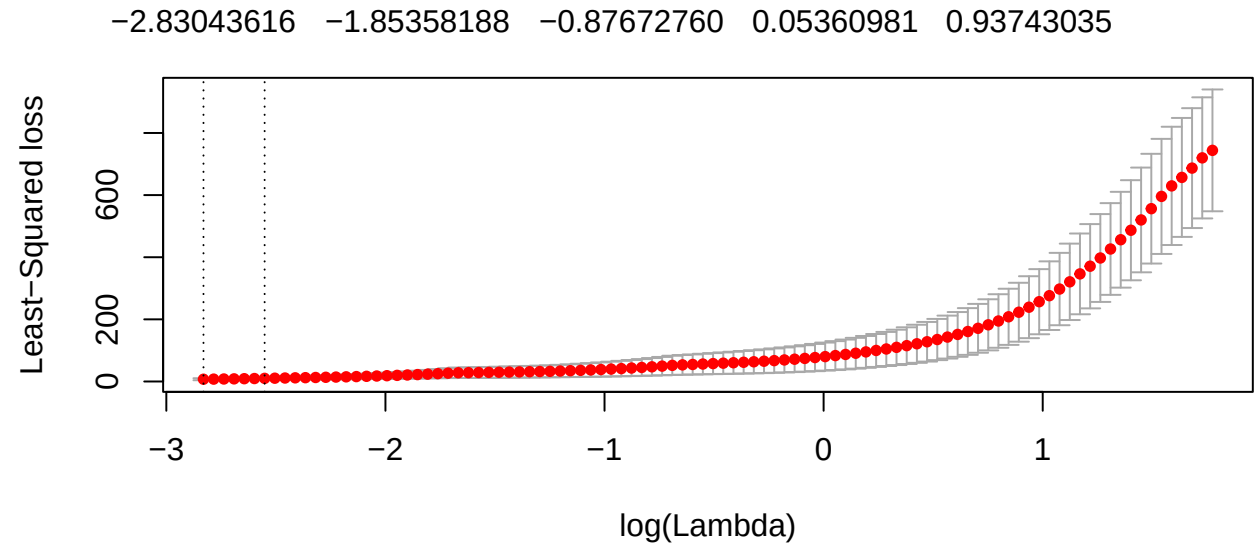



```
plot(lasso_coef_1se, type = "l")
```




While there doesn't appear to be any grouping or similar effect by variable number, it does make sense if the we think about the data. As a result, let's create a grouping variable and analyze this with the grouped lasso.

```
pet_group <- rep(1:14, each = 20)[1:268]
gglasso_cv <- cv.gglasso(x = pet_mat[, 1:268], y = pet_mat[, 269], group = pet_group,
    loss = "ls", pred.loss = "L2", lambda.factor = 0.01, nfolds = 5)
par(mfrow = c(1, 1))
plot(gglasso_cv)
```




Let's look at the results:

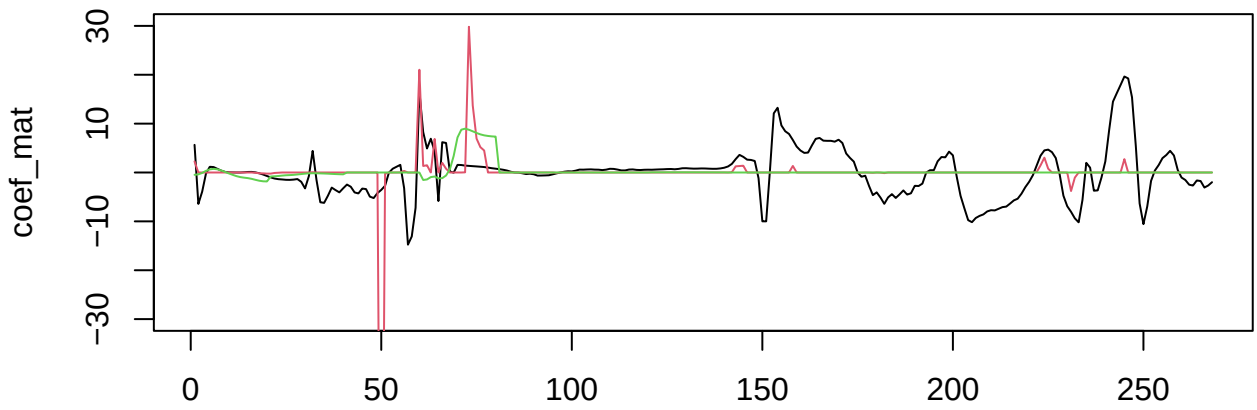
```
gglasso_coef_1se <- coef(gglasso_cv,
                         s="lambda.1se")[-1]
coef_mat <- cbind(ridge_coef_1se,
                  lasso_coef_1se,
                  gglasso_coef_1se)

table(lasso_coef_1se==0)
```

| FALSE | TRUE |
|---|---|
| 55 | 213 |

```r
table(gglasso_coef_1se==0)
```

| FALSE | TRUE |
|---|---|
| 80 | 188 |

```r
matplot(coef_mat, type ="l",
        ylim = c(-30,30), lty = 1)
```




Now let's look at this with the fused LASSO.

```r
library(genlasso)
`?`(fusedlasso)
```

```
Compute the fused lasso solution path for a general graph, or a 1d or
2d grid

Description:

     These functions produce the solution path for a general fused
     lasso problem. The 'fusedlasso' function takes either a penalty
     matrix or a graph object from the 'igraph' package. The
     'fusedlasso1d' and 'fusedlasso2d' functions are convenience
     functions that construct the penalty matrix over a 1d or 2d grid.

Usage:

     fusedlasso(y, X, D, graph, gamma = 0, approx = FALSE, maxsteps = 2000,
                minlam = 0, rtol = 1e-07, btol = 1e-07, eps = 1e-4,
                verbose = FALSE)
     fusedlasso1d(y, pos, X, gamma = 0, approx = FALSE, maxsteps = 2000,
                  minlam = 0, rtol = 1e-07, btol = 1e-07, eps = 1e-4,
                  verbose = FALSE)
     fusedlasso2d(y, X, dim1, dim2, gamma = 0, approx = FALSE, maxsteps = 2000,
                  minlam = 0, rtol = 1e-07, btol = 1e-07, eps = 1e-4,
                  verbose = FALSE)

Arguments:

       y: a numeric response vector. Alternatively, for 'fusedlasso2d'
          with no matrix 'X' passed, 'y' can be a matrix (its
```

dimensions corresponding to the underlying 2d grid). Note that when 'y' is given as a vector in 'fusedlasso2d', with no 'X' passed, it should be in column major order.

pos: only for 'fusedlasso1d', these are the optional positions of the positions in the 1d grid. If missing, the 1d grid is assumed to have unit spacing.

X: an optional matrix of predictor variables, with observations along the rows, and variables along the columns. If the passed 'X' has more columns than rows, then a warning is given, and a small ridge penalty is added to the generalized lasso criterion before the path is computed. If 'X' has less columns than rows, then its rank is not checked for efficiency, and (unlike the 'genasso' function) a ridge penalty is not automatically added if it is rank deficient. Therefore, a tall, rank deficient 'X' may cause errors.

D: only for 'fusedlasso', this is the penalty matrix, i.e., the oriented incidence matrix over the underlying graph (the orientation of each edge being arbitrary). Only one of 'D' or 'graph' needs to be specified.

graph: only for 'fusedlasso', this is the underlying graph as an 'igraph' object from the 'igraph' package. Only one of 'D' or 'graph' needs to be specified.

dim1: only for 'fusedlasso2d', this is the number of rows in the underlying 2d grid. If missing and 'y' is given as a matrix, it is assumed to be the number of rows of 'y'.

dim2: only for 'fusedlasso2d', this is the number of columns in the underlying 2d grid. If missing and 'y' is given as a matrix, it is assumed to be the number of columns of 'y'.

gamma: a numeric variable greater than or equal to 0, indicating the ratio of the two tuning parameters, one for the fusion penalty, and the other for the pure \ell_1 penalty. Default is 0. See "Details" for more information.

approx: a logical variable indicating if the approximate solution path should be used (with no dual coordinates leaving the boundary). Default is 'FALSE'. Note that for the 1d fused lasso, with identity predicor matrix, this approximate path is the same as the exact solution path.

maxsteps: an integer specifying the maximum number of steps for the algorithm to take before termination. Default is 2000.

minlam: a numeric variable indicating the value of lambda at which the path should terminate. Default is 0.

rtol: a numeric variable giving the tolerance for determining the rank of a matrix: if a diagonal value in the R factor of a QR

```
          decomposition is less than R, in absolute value, then it is
          considered zero. Hence making rtol larger means being less
          stringent with determination of matrix rank. In general, do
          not change this unless you know what you are getting into!
          Default is 1e-7.

      btol: a numeric variable giving the tolerance for accepting "late"
          hitting and leaving times: future hitting times and leaving
          times should always be less than the current knot in the
          path, but sometimes for numerical reasons they are larger;
          any computed hitting or leaving time larger than the current
          knot + btol is thrown away. Hence making btol larger means
          being less stringent withthe determination of hitting and
          leaving times. Again, in general, do not change this unless
          you know what you are getting into! Default is 1e-7.

       eps: a numeric variable indicating the multiplier for the ridge
          penalty, in the case that 'X' is wide (more columns than
          rows). If numeric problems occur, make 'eps' larger. Default
          is 1e-4.

   verbose: a logical variable indicating if progress should be reported
          after each knot in the path.

Details:

     The fused lasso estimate minimizes the criterion

         1/2 sum_{i=1}^n (y_i - x_i^T beta_i)^2 + lambda sum_{(i,j) \in E}
         |beta_i - beta_j| + gamma \cdot lambda sum_{i=1}^p |beta_i|,

     where x_i is the ith row of the predictor matrix and E is the edge
     set of the underlying graph. The solution \hat{beta} is computed
     as a function of the regularization parameter lambda, for a fixed
     value of gamma. The default is to set gamma=0, which corresponds
     to pure fusion of the coefficient vector beta. A choice gamma>0
     introduces both sparsity and fusion in the coefficient vector,
     with a higher value placing more priority on sparsity.

     If the predictor matrix is the identity, and the primal solution
     path beta is desired at several levels of the ratio parameter
     gamma, it is much more efficient to compute the solution path once
     with gamma=0, and then use soft-thresholding via the 'softthresh'
     function.

     Finally, for the image denoising problem, i.e., the fused lasso
     over a 2d grid with identity predictor matrix, it is easy to
     specify a huge graph with a seemingly small amount of data. For
     instance, running the 2d fused lasso (with identity predictor
     matrix) on an image at standard 1080p HD resolution yields a graph
     with over 2 million edges. Moreover, in image denoising
     problems-somewhat unlike most other applications of the fused
     lasso (and generalized lasso)-a solution is often desired near the
     dense end of the path (lambda=0) as opposed to the regularized end
```

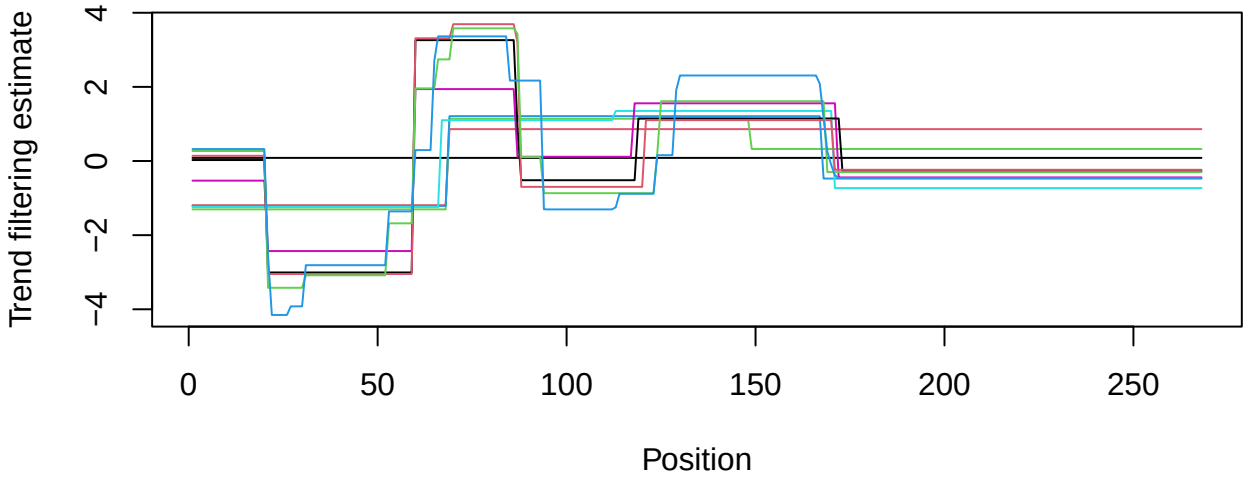
```
(lambda=Inf). The dual path algorithm implemented by the
'fusedlasso2d' function begins at the fully regularized end and
works its way down to the dense end. For a problem with many edges
(dual variables), if a solution at the dense is desired, then it
must usually pass through a huge number knots in the path. Hence
it is not advisable to run 'fusedlasso2d' on image denoising
problems of large scale, as the dual solution path is
computationally infeasible. It should be noted that a faster
algorithm for the 2d fused lasso solution path (when the predictor
matrix is the identity), which begins at the dense end of the
path, is available in the 'flsa' package.
```

Since our $X$ variables lie on a 1-dimensional line, we'll use the `fusedlasso1d` version of the function.

```
fusedlasso <- fusedlasso1d(y = pet_mat[, 269], pos = 1:268, X = pet_mat[, 1:268])
```

```
## Warning in trendfilter(y, pos, X, 0, approx, maxsteps, minlam, rtol, btol, :
## Adding a small ridge penalty (multiplier 0.0001), because X has more columns
## than rows.
```

```
par(mfrow = c(1, 1))
plot(fusedlasso)
```




This package doesn't have a built in CV function and there's really no way to manually do CV because this package doesn't have a way of manually selecting the $\lambda$ values. Possibly other packages can do this.