understanding how lasso regression works

This is a simple example of how the lasso regression model works.

```
save.image("backup.RData")
rm(list=ls())
library("glmnet")
```

```
## Loading required package: Matrix
##
## Attaching package: 'Matrix'
##
## The following objects are masked from 'package:base':
##
##     crossprod, tcrossprod
##
## Loading required package: foreach
## Loaded glmnet 2.0-5
```

The lasso regression model was originally developed in 1989. It is an alterative to the classic least squares estimate that avoids many of the problems with overfitting when you have a large number of indepednent variables.

You can't understand the lasso fully without understanding some of the context of other regression models. The examples will look, by necessity, at a case with only two independent variables, even though the lasso only makes sense for settings where the number of independent variables are larger by several orders of magnitutde.

Start with a built in data set with the R package, mtcars (https://stat.ethz.ch/R-manual/R-devel/library/datasets/html/mtcars.html). You can get details about this data set by typing help("mtcars").

```
head(mtcars)
```

```
##                    mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

```
summary(mtcars)
```

```
##       mpg             cyl             disp             hp
##  Min.   :10.40   Min.   :4.000   Min.   : 71.1   Min.   : 52.0
##  1st Qu.:15.43   1st Qu.:4.000   1st Qu.:120.8   1st Qu.: 96.5
##  Median :19.20   Median :6.000   Median :196.3   Median :123.0
##  Mean   :20.09   Mean   :6.188   Mean   :230.7   Mean   :146.7
##  3rd Qu.:22.80   3rd Qu.:8.000   3rd Qu.:326.0   3rd Qu.:180.0
##  Max.   :33.90   Max.   :8.000   Max.   :472.0   Max.   :335.0
##       drat             wt             qsec             vs
##  Min.   :2.760   Min.   :1.513   Min.   :14.50   Min.   :0.0000
##  1st Qu.:3.080   1st Qu.:2.581   1st Qu.:16.89   1st Qu.:0.0000
##  Median :3.695   Median :3.325   Median :17.71   Median :0.0000
##  Mean   :3.597   Mean   :3.217   Mean   :17.85   Mean   :0.4375
##  3rd Qu.:3.920   3rd Qu.:3.610   3rd Qu.:18.90   3rd Qu.:1.0000
##  Max.   :4.930   Max.   :5.424   Max.   :22.90   Max.   :1.0000
##        am              gear            carb
##  Min.   :0.0000   Min.   :3.000   Min.   :1.000
##  1st Qu.:0.0000   1st Qu.:3.000   1st Qu.:2.000
##  Median :0.0000   Median :4.000   Median :2.000
##  Mean   :0.4062   Mean   :3.688   Mean   :2.812
##  3rd Qu.:1.0000   3rd Qu.:4.000   3rd Qu.:4.000
##  Max.   :1.0000   Max.   :5.000   Max.   :8.000
```

You can pick any three variables in the data set. The first variable (v0) will be the dependent variable and the other two variables (v1, v2) will be the independent variables.

```
v0 <- "mpg"
v1 <- "disp"
v2 <- "hp"
```

Examine the correlations among these variables.

```
round(cor(mtcars[, c(v0,v1,v2)]), 2)
```

```
##        mpg  disp    hp
## mpg   1.00 -0.85 -0.78
## disp -0.85  1.00  0.79
## hp   -0.78  0.79  1.00
```

The units of measurement for these two independent variables are quite different, so to make comparisons more fair, you should standardize them. There’s less of a need to standardize the dependent variable, but let’s do that also, just for simplicity.

```
standardize <- function(x) {(x-mean(x))/sd(x)}
z0 <- standardize(mtcars[, v0])
z1 <- standardize(mtcars[, v1])
z2 <- standardize(mtcars[, v2])
lstsq <- lm(z0~z1+z2-1)
lstsq_beta <- coef(lstsq)
```

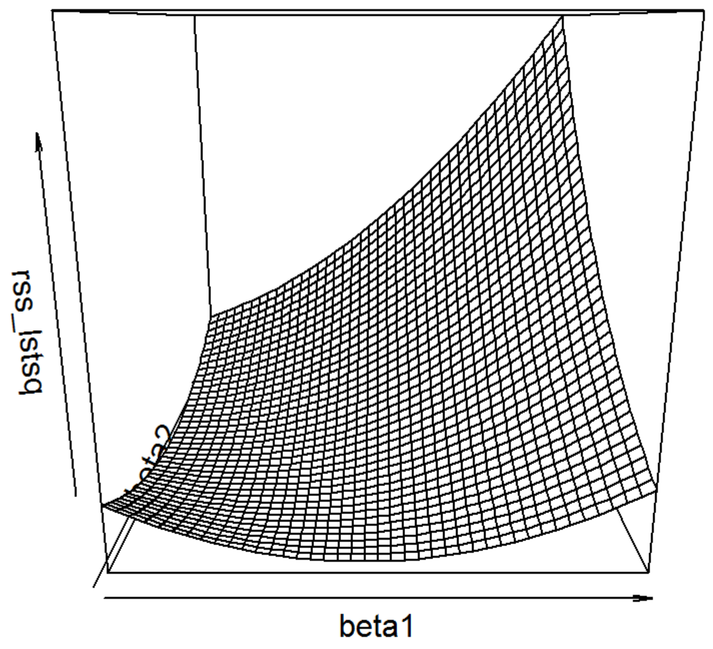
The lm function uses the traditional linear regression approach.

Just a technical point here: the linear regression model fit here does not include an intercept, because all the variables have already been standardized to a mean of zero. The simple least squares estimates are -0.62 for disp and -0.28 for hp.

The traditional linear regression model is sometimes called least squares regression, because it minimizes (least) the sum of squared deviations (squares) of the residuals.

TO understand this better, compute the residual sum of squared deviations (rss) for a range of possible regression coefficients.

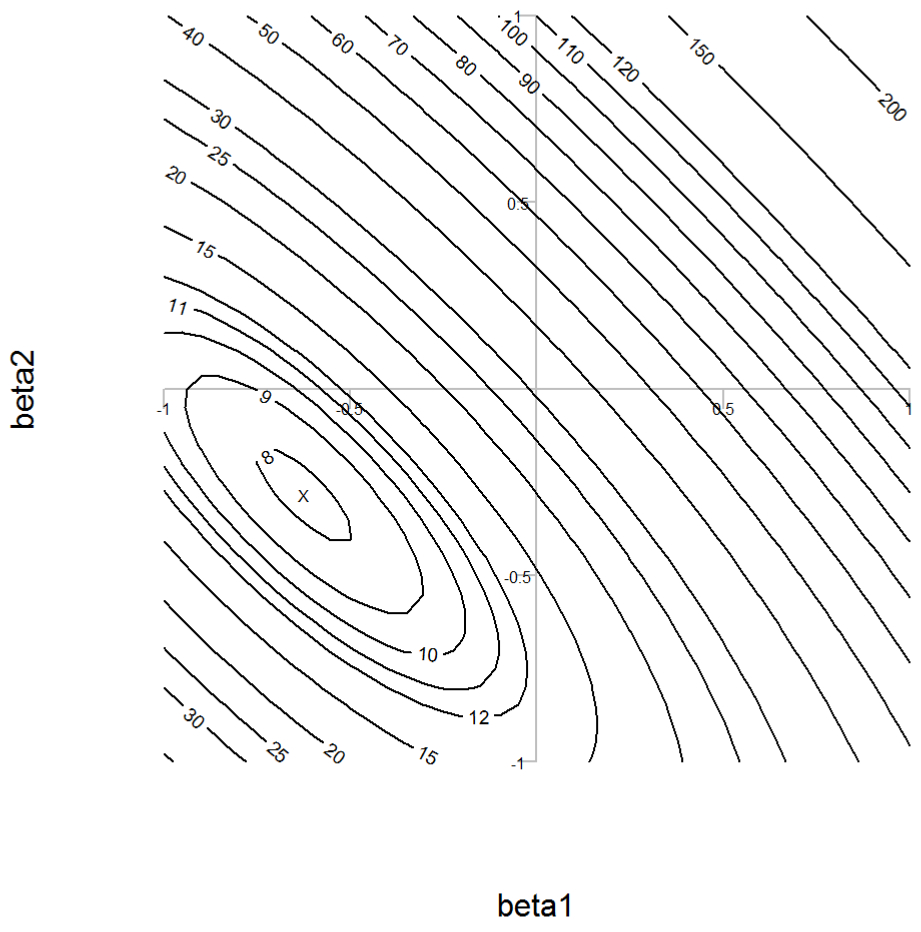
```
n_lstsq <- 41
s <- seq(-1, 1, length=n_lstsq)
rss_lstsq <- matrix(NA, nrow=n_lstsq, ncol=n_lstsq)
for (i in 1:n_lstsq) {
  for (j in 1:n_lstsq) {
    rss_lstsq[i, j] <- sum((z0-s[i]*z1-s[j]*z2)^2)
  }
}
persp(s, s, rss_lstsq, xlab="beta1", ylab="beta2", zlab="rss_lstsq")
```



You may find the contour plot of this three dimensional surface to be easier to follow.

```
draw_axes <- function() {
  k2 <- seq(-1, 1, length=5)
  par(mar=c(4.6, 4.6, 0.6, 0.6), xaxs="i", yaxs="i")
  plot(1.02*range(s), 1.02*range(s), type="n", xlab="beta1", ylab="beta2", axes=FALSE
)
  axis(side=1, pos=0, col="gray", at=k2, labels=rep(" ", length(k2)))
  axis(side=2, pos=0, col="gray", at=k2, labels=rep(" ", length(k2)))
  text(k2[-3], -0.05, k2[-3], cex=0.5, col="black")
  text(-0.05, k2[-3], k2[-3], cex=0.5, col="black")
}
k1 <- c(1, 1.1, 1.2, 1.5, 2, 2.5, 3:9)
k1 <- c(0.1*k1, k1, 10*k1, 100*k1, 1000*k1)

draw_axes()
contour(s, s, matrix(rss_lstsq,nrow=n_lstsq), levels=k1, add=TRUE, col="black")
text(lstsq_beta[1], lstsq_beta[2], "X", cex=0.5)
```



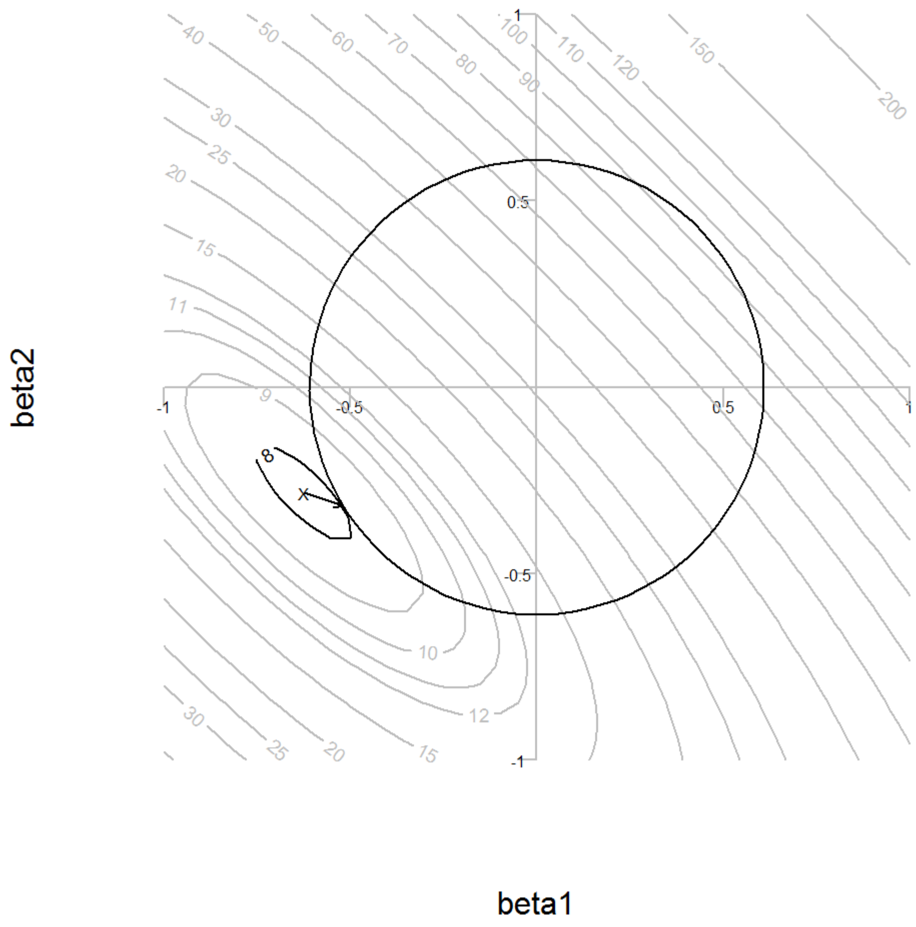
```
k1 <- k1[k1>min(rss_lstsq)]
```

The level curves (the values where the thre dimension surface is constant) are elliptical, which reflects the correlation in $\hat{\beta}_1$ and $\hat{\beta}_2$ that is induced by the correlation between z1 and z2.

The small “X” represents the minimum value, or the least squares solution. It corresponds to height of 7.81 units.

Now suppose that you were willing to sacrifice a bit on the residual sum of squares. You'd be willing to settle for a value of $\hat{\beta}_1$ and $\hat{\beta}_2$ that produced a residual sum of squares of 8 instead of 7.8. In exchange, you'd get a solution that was a bit closer to zero. What would that value be? Any value on the ellipse labelled 8 would be equally desirable from the least squares perspective. But the point on the ellipse closest to (0, 0) has the most simplicity.

```
find_closest <- function(x, target) {
  d <- abs(x-target)
  return(which(d==min(d))[1])
}
draw_circle <- function(r) {
  radians <- seq(0, 2*pi, length=100)
  lines(r*sin(radians), r*cos(radians))
}
ridge <- glmnet(cbind(z1, z2), z0, alpha=0, intercept=FALSE, nlambda=1000)
m_ridge <- dim(ridge$beta)[2]
rss_ridge <- rep(NA,m_ridge)
for (i in 1:m_ridge) {
  rss_ridge[i] <- sum((z0 - ridge$beta[1, i]*z1 -ridge$beta[2, i]*z2)^2)
}
r1 <- find_closest(rss_ridge, k1[1])
draw_axes()
contour(s, s, matrix(rss_lstsq, nrow=n_lstsq), levels=k1, add=TRUE, col="gray")
contour(s, s, matrix(rss_lstsq, nrow=n_lstsq), levels=k1[1], add=TRUE, col="black")
draw_circle(sqrt(ridge$beta[1, r1]^2+ridge$beta[2, r1]^2))
text(lstsq_beta[1], lstsq_beta[2], "X", cex=0.5)
arrows(lstsq_beta[1], lstsq_beta[2], ridge$beta[1, r1], ridge$beta[2, r1], len=0.05)
```



For ridge regression, find the circle which just barely touches the ellipse corresponding to a level surface of 8. These values are $\hat{\beta}_1$=-0.52 and $\hat{\beta}_2$=-0.32.

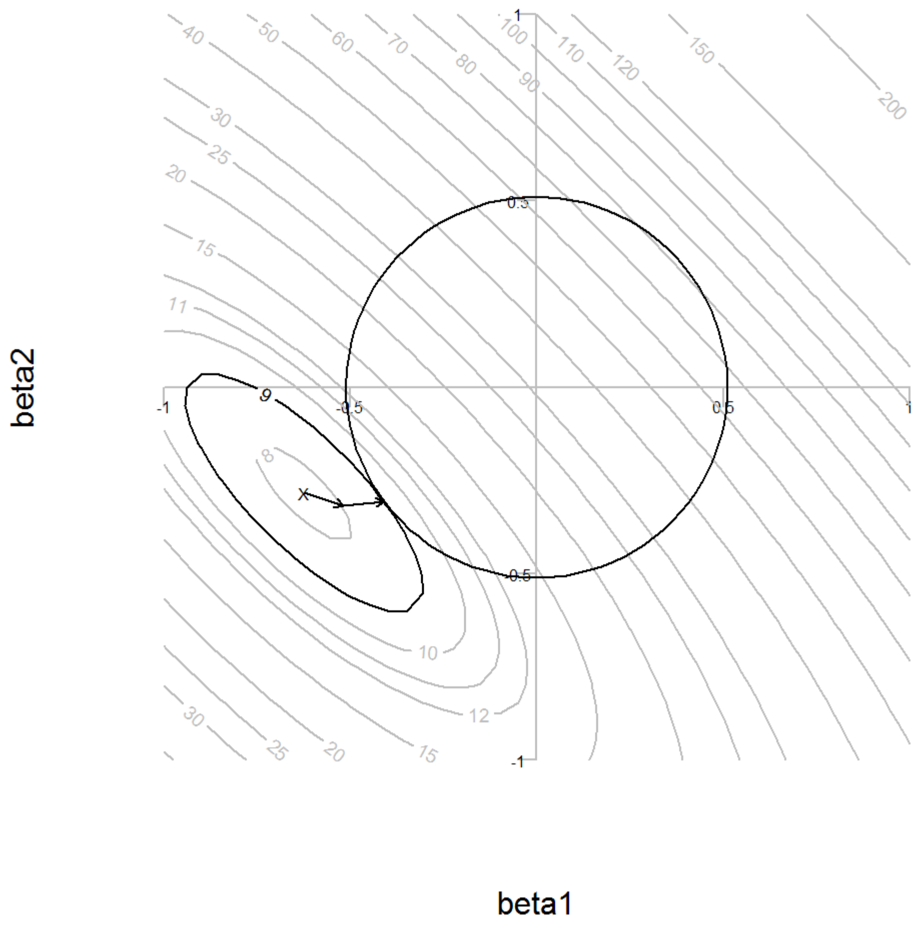
Now what do I mean by “simplicity”? In this case, I mean less of a tendency to produce extreme predictions. Regression coefficients that are flatter, that is, closer to zero, have less of a tendency to produce extreme predictions.

Extreme predictions are sometimes okay, but they are often a symptom of overfitting.

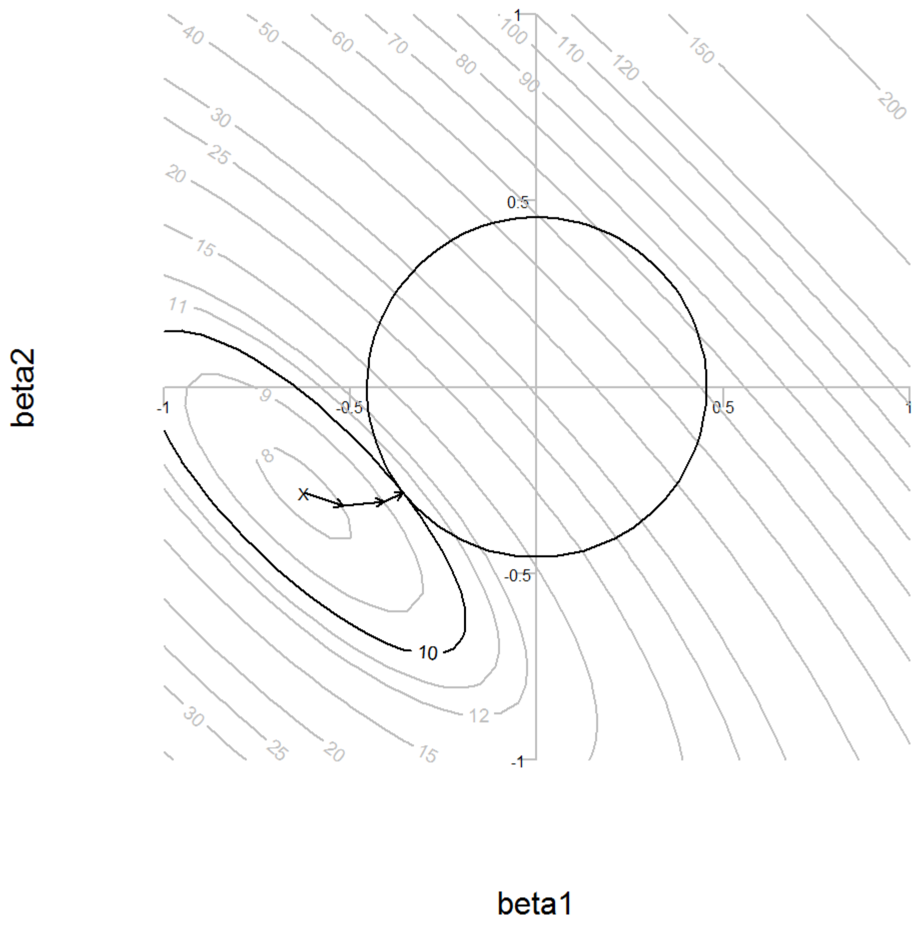
Now ridge regression offers you a multitude of choices, depending on the trade-offs you are willing to make between efficiency (small rss) and simplicity (regression coefficients close to zero).

If you wanted a bit more simplicity and could suffer a bit more on the residual sums of squares end of things, you could find the point on the level surface 9 or 10.

```
r2 <- find_closest(rss_ridge, k1[2])
r3 <- find_closest(rss_ridge, k1[3])
draw_axes()
contour(s, s, matrix(rss_lstsq,nrow=n_lstsq), levels=k1, add=TRUE, col="gray")
contour(s, s, matrix(rss_lstsq,nrow=n_lstsq), levels=k1[2], add=TRUE, col="black")
draw_circle(sqrt(ridge$beta[1, r2]^2+ridge$beta[2, r2]^2))
text(lstsq_beta[1], lstsq_beta[2], "X", cex=0.5)
arrows(lstsq_beta[1], lstsq_beta[2], ridge$beta[1, r1], ridge$beta[2, r1], len=0.05)
arrows(ridge$beta[1, r1], ridge$beta[2, r1], ridge$beta[1, r2], ridge$beta[2, r2], le
n=0.05)
```

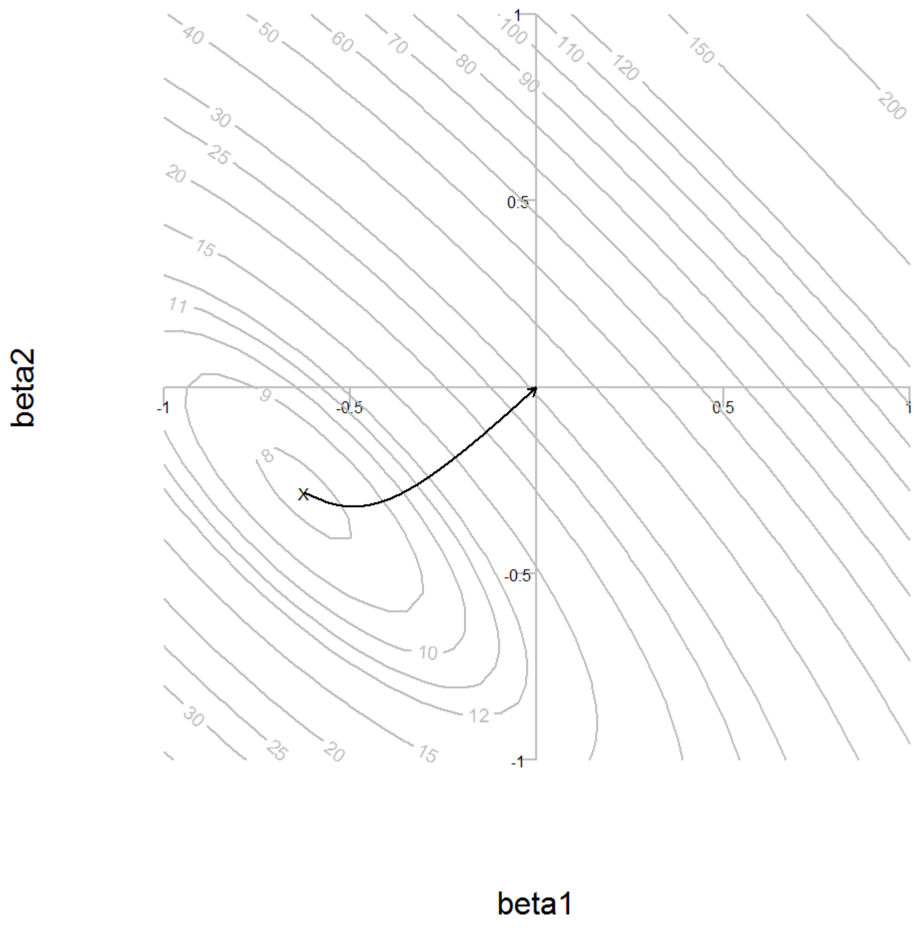


```
draw_axes()
contour(s, s, matrix(rss_lstsq,nrow=n_lstsq), levels=k1, add=TRUE, col="gray")
contour(s, s, matrix(rss_lstsq,nrow=n_lstsq), levels=k1[3], add=TRUE, col="black")
draw_circle(sqrt(ridge$beta[1, r3]^2+ridge$beta[2, r3]^2))
text(lstsq_beta[1], lstsq_beta[2], "X", cex=0.5)
arrows(lstsq_beta[1], lstsq_beta[2], ridge$beta[1, r1], ridge$beta[2, r1], len=0.05)
arrows(ridge$beta[1, r1], ridge$beta[2, r1], ridge$beta[1, r2], ridge$beta[2, r2], le
n=0.05)
arrows(ridge$beta[1, r2], ridge$beta[2, r2], ridge$beta[1, r3], ridge$beta[2, r3], le
n=0.05)
```



You could do this for any level surface, including those level surfaces in between the ones shown here.

```
draw_axes()
contour(s, s, matrix(rss_lstsq,nrow=n_lstsq), levels=k1, add=TRUE, col="gray")
text(lstsq_beta[1], lstsq_beta[2], "X", cex=0.5)
segments(lstsq_beta[1], lstsq_beta[2], ridge$beta[1, m_ridge], ridge$beta[2, m_ridge]
)
lines(ridge$beta[1, ], ridge$beta[2, ])
arrows(ridge$beta[1, 2], ridge$beta[2, 2], ridge$beta[1, 1], ridge$beta[2, 1], len=0.
05)
```



The black curve represents all the ridge regression solutions.

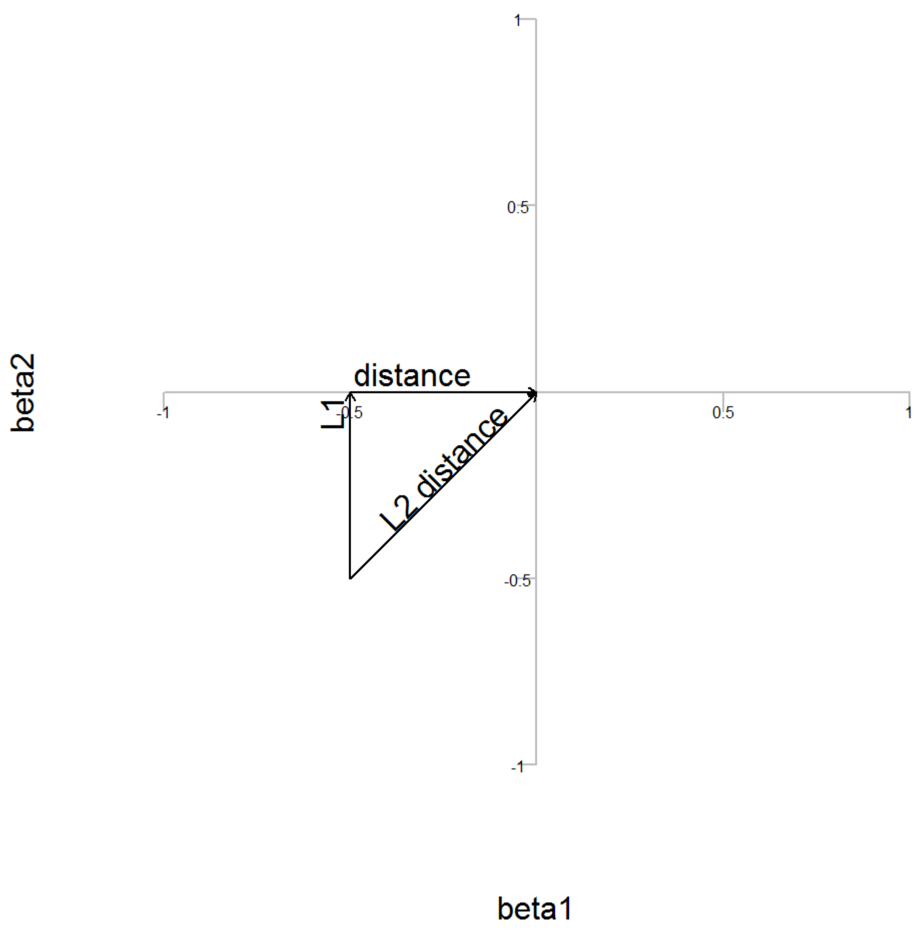
Ridge regression measures simplicity as the straight line distance between (0, 0) and ($\hat{\beta}_1$, $\hat{\beta}_2$). This is known as Euclidean distance or L^2 distance. The formula for L^2 distance is $\sqrt{\hat{\beta}_1{}^2 + \hat{\beta}_2{}^2}$

Values that are at the same L^2 distance from the origin correspond to circles.

The lasso regression model works much like ridge regression, except it use L^1 or absolute value distance. The formula for L^1 distance is $|\hat{\beta}_1| + |\hat{\beta}_2|$.

While L^2 distance represents the length of a diagonal path from ($\hat{\beta}_1$, $\hat{\beta}_2$) to (0, 0), the L^1 represents the length of the path that goes vertically to the X-axis and the horizontally to the Y-axis.

```
draw_axes()
arrows(-0.5, -0.5, 0, 0, len=0.05)
text(-0.25, -0.20, "L2 distance", srt=45)
arrows(-0.5, -0.5, -0.5, 0, len=0.05)
arrows(-0.5, 0, 0, 0, len=0.05)
text(-0.55, -0.01, "L1", srt=90, adj=1)
text(-0.49, 0.05, "distance", adj=0)
```

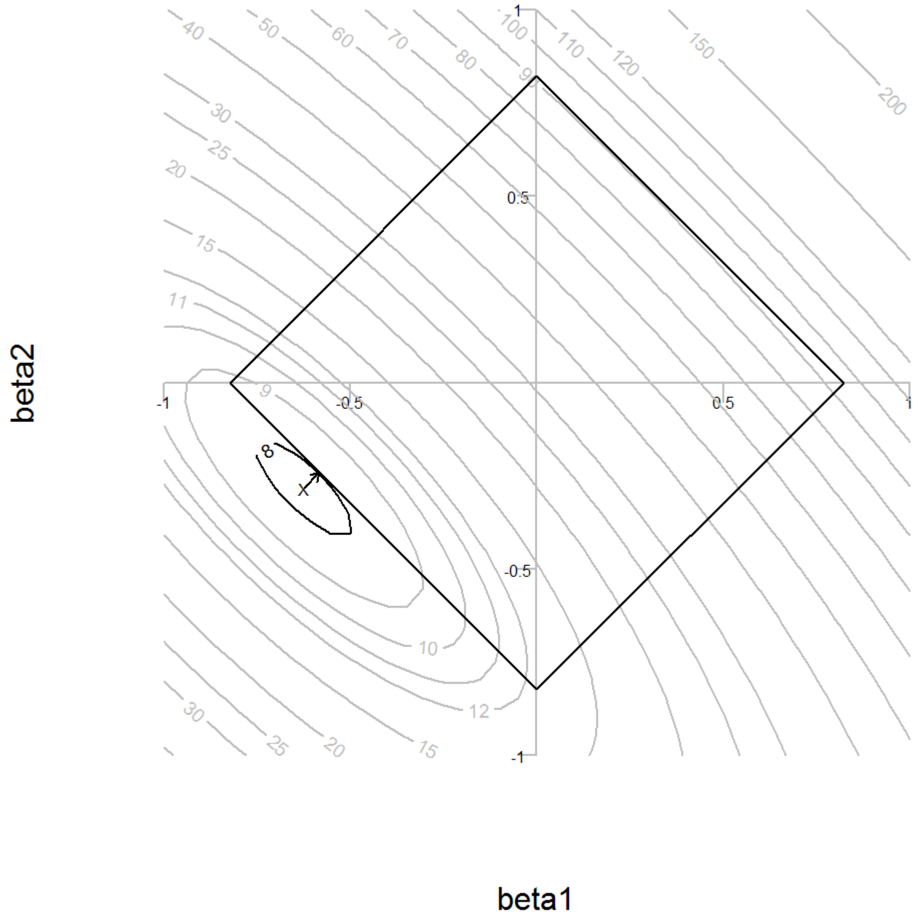


You get the same L^1 distance, of course, if you go first horizontally to the Y-axis and then vertically to the X-axis. A diamond (actually a square rotated to 45 degrees) represents the set of points that are equally distant from the origin using the L^1 concept of distance.

The lasso model finds the largest diamond that just barely touches the level surface.

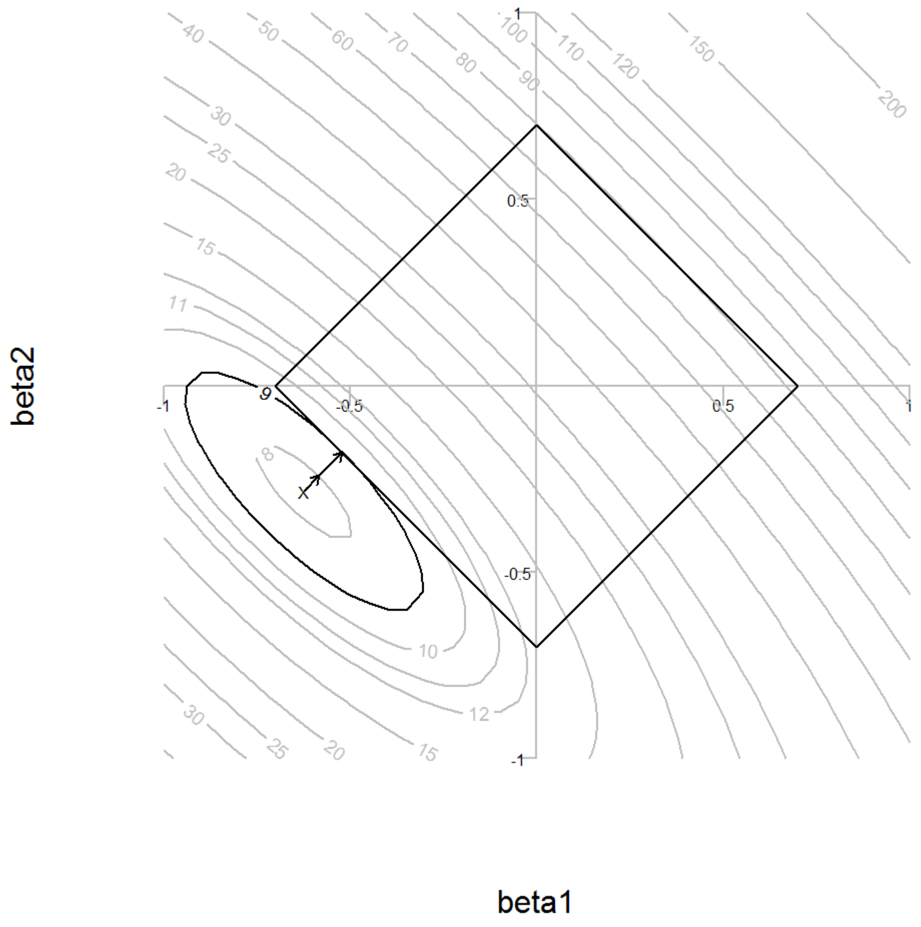
```
lasso <- glmnet(cbind(z1, z2), z0, alpha=1, intercept=FALSE, nlambda=1000)
m_lasso <- dim(lasso$beta)[2]
rss_lasso <- rep(NA,m_lasso)
for (i in 1:m_lasso) {
  rss_lasso[i] <- sum((z0 - lasso$beta[1, i]*z1 -lasso$beta[2, i]*z2)^2)
}
r1 <- find_closest(rss_lasso, k1[1])
r2 <- find_closest(rss_lasso, k1[2])
r3 <- find_closest(rss_lasso, k1[3])

draw_axes()
contour(s, s, matrix(rss_lstsq,nrow=n_lstsq), levels=k1, add=TRUE, col="gray")
contour(s, s, matrix(rss_lstsq,nrow=n_lstsq), levels=k1[1], add=TRUE, col="black")
d <- abs(lasso$beta[1, r1])+abs(lasso$beta[2, r1])
segments( d, 0, 0, d)
segments( 0, d,-d, 0)
segments(-d, 0, 0,-d)
segments( 0,-d, d, 0)
text(lstsq_beta[1], lstsq_beta[2], "X", cex=0.5)
arrows(lstsq_beta[1], lstsq_beta[2], lasso$beta[1, r1], lasso$beta[2, r1], len=0.05)
```

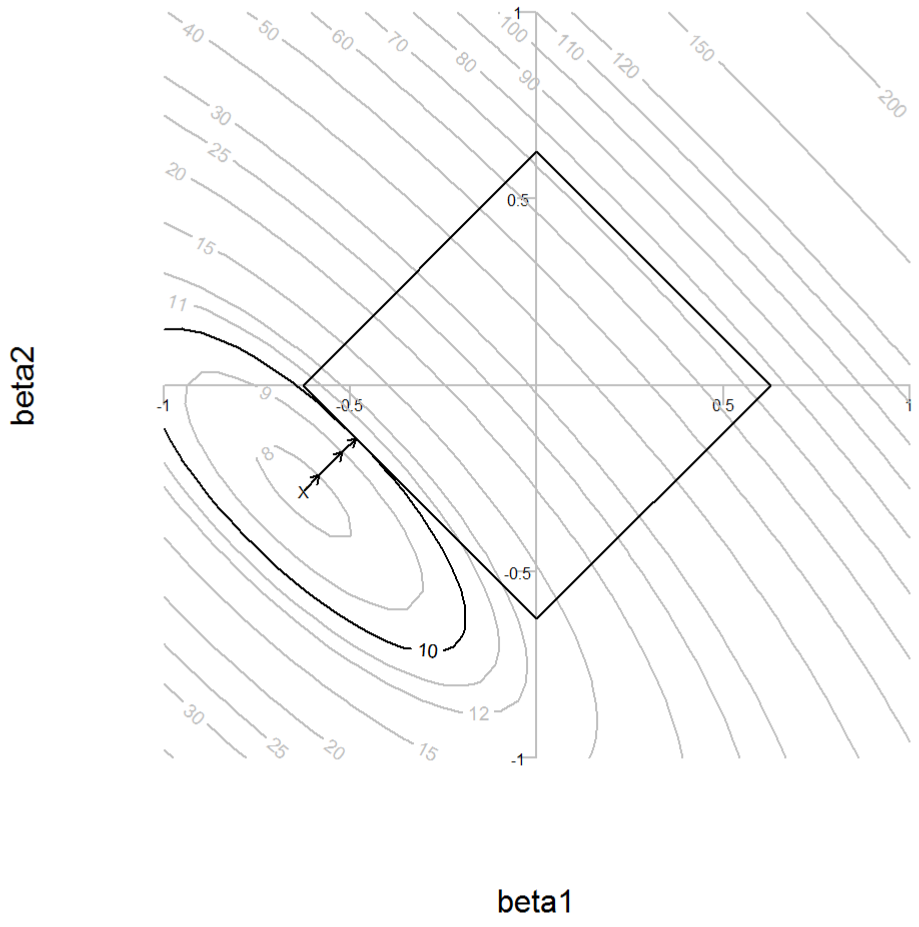


Here are a couple of different lasso fits.

```
draw_axes()
contour(s, s, matrix(rss_lstsq,nrow=n_lstsq), levels=k1, add=TRUE, col="gray")
contour(s, s, matrix(rss_lstsq,nrow=n_lstsq), levels=k1[2], add=TRUE, col="black")
d <- abs(lasso$beta[1, r2])+abs(lasso$beta[2, r2])
segments( d, 0, 0, d)
segments( 0, d,-d, 0)
segments(-d, 0, 0,-d)
segments( 0,-d, d, 0)
text(lstsq_beta[1], lstsq_beta[2], "X", cex=0.5)
arrows(lstsq_beta[1], lstsq_beta[2], lasso$beta[1, r1], lasso$beta[2, r1], len=0.05)
arrows(lasso$beta[1, r1], lasso$beta[2, r1], lasso$beta[1, r2], lasso$beta[2, r2], le
n=0.05)
```

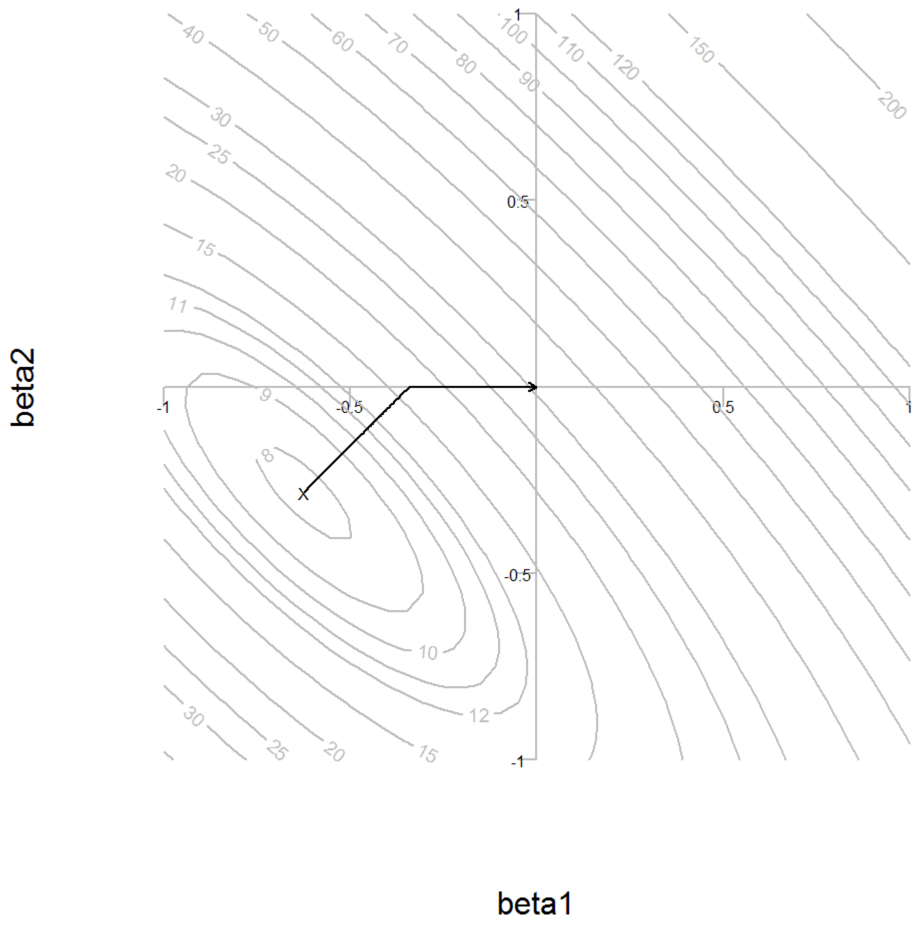


```
draw_axes()
contour(s, s, matrix(rss_lstsq,nrow=n_lstsq), levels=k1, add=TRUE, col="gray")
contour(s, s, matrix(rss_lstsq,nrow=n_lstsq), levels=k1[3], add=TRUE, col="black")
d <- abs(lasso$beta[1, r3])+abs(lasso$beta[2, r3])
segments( d, 0, 0, d)
segments( 0, d,-d, 0)
segments(-d, 0, 0,-d)
segments( 0,-d, d, 0)
text(lstsq_beta[1], lstsq_beta[2], "X", cex=0.5)
arrows(lstsq_beta[1], lstsq_beta[2], lasso$beta[1, r1], lasso$beta[2, r1], len=0.05)
arrows(lasso$beta[1, r1], lasso$beta[2, r1], lasso$beta[1, r2], lasso$beta[2, r2], le
n=0.05)
arrows(lasso$beta[1, r2], lasso$beta[2, r2], lasso$beta[1, r3], lasso$beta[2, r3], le
n=0.05)
```



Here is the set of all lasso fits.

```
draw_axes()
contour(s, s, matrix(rss_lstsq,nrow=n_lstsq), levels=k1, add=TRUE, col="gray")
text(lstsq_beta[1], lstsq_beta[2], "X", cex=0.5)
segments(lstsq_beta[1], lstsq_beta[2], lasso$beta[1, m_lasso], lasso$beta[2, m_lasso]
)
lines(lasso$beta[1, ], lasso$beta[2, ])
arrows(lasso$beta[1, 2], lasso$beta[2, 2], lasso$beta[1, 1], lasso$beta[2, 1], len=0.
05)
```



While both ridge regression and lasso regression find solutions that are closer to the origin, the lasso regression, at least in this simple case, heads off a 45 degree angle. Once the lasso regression meets one of the axes, it heads along that axis towards zero.

While the behavior is more complicated with more than two independent variables, the general tendency of the lasso regression model is to shrink towards the various axes, zeroing out one coefficient after another.

This is a bonus of lasso; It avoids overfitting not just by flattening out all the slopes. It also avoids overfitting by preferentially removes some of the slopes entirely.

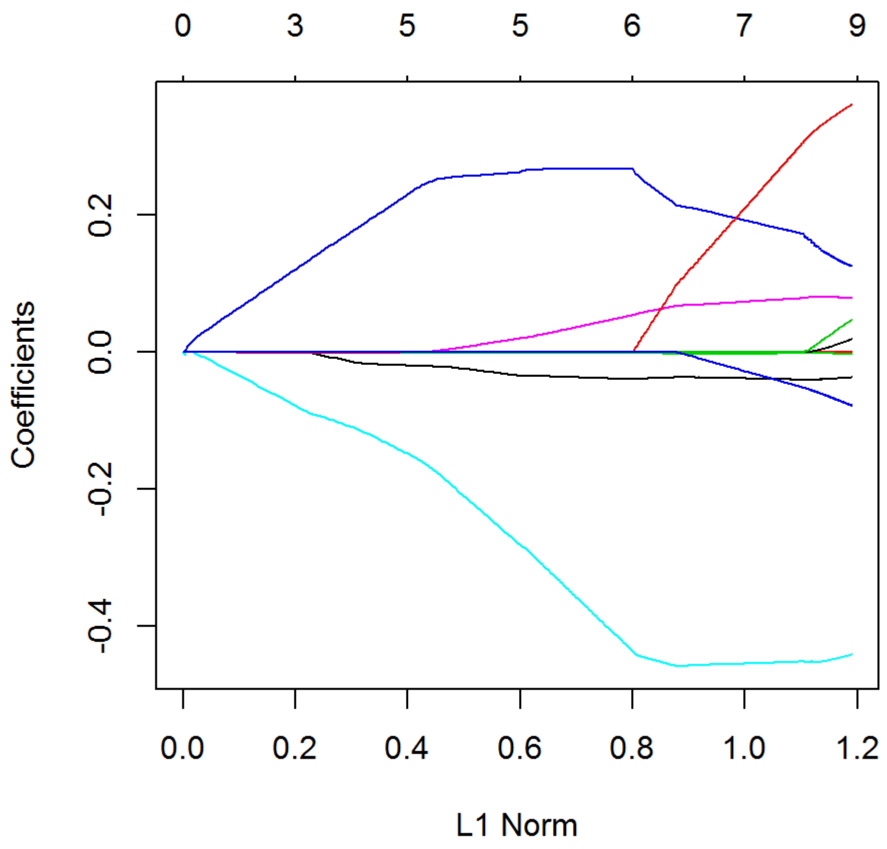
This makes the lasso regression model a feature selection technique, and it avoids all of the problems of overfitting that a simpler feature selection technique, stepwise regression, suffers from.

The lasso acronym (least absolute shrinkage and selection operator) is an description of the dual nature of this model.

Let’s see how the lasso works for a slightly more complex setting. The mtcars data set has ten variables. Let’s use the last nine of these variables to predict the first variable, mpg.

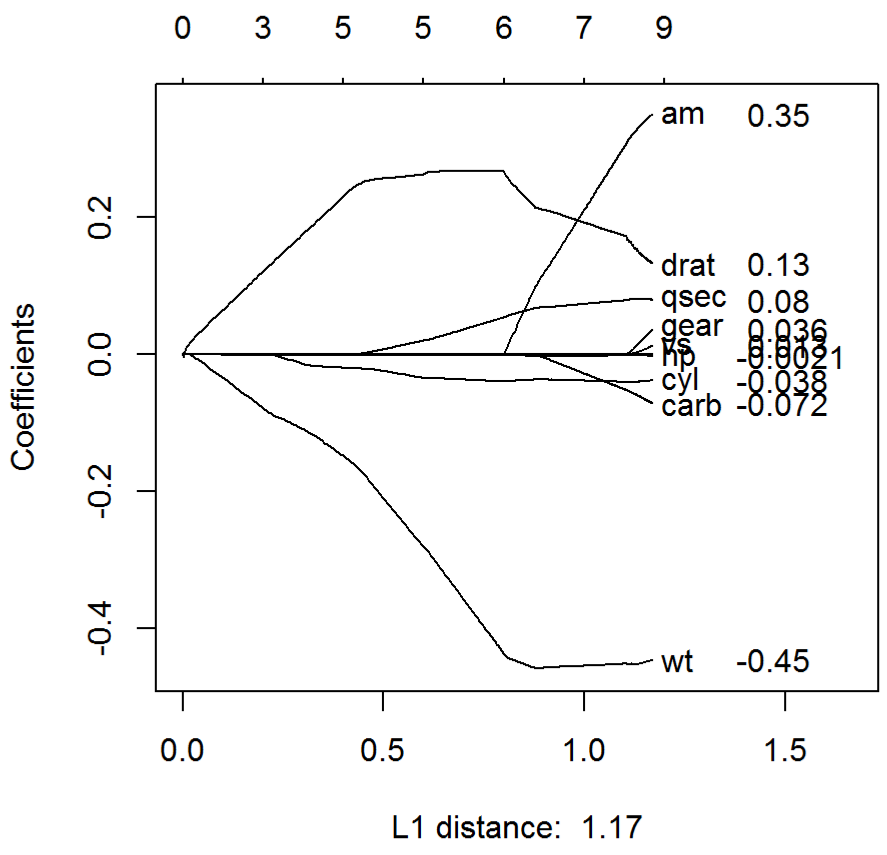
The lasso is really intended for much bigger cases than this, but you can still learn a lot from this small example.

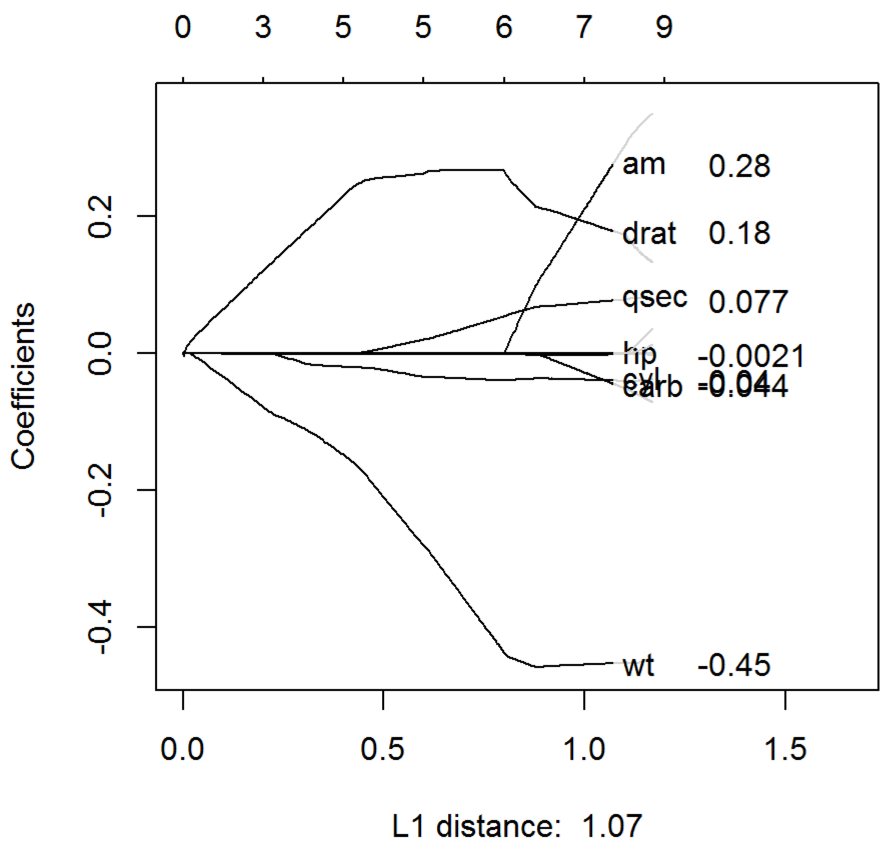
```
# By default, the glmnet function standardizes all the independent variables,
# but I also wanted to standardize the dependent variable for consistency
# with the earlier examples.
lasso <- glmnet(as.matrix(mtcars[, -1]), standardize(mtcars[, 1]), alpha=1, intercept
=FALSE, nlambda=1000)
plot(lasso)
```

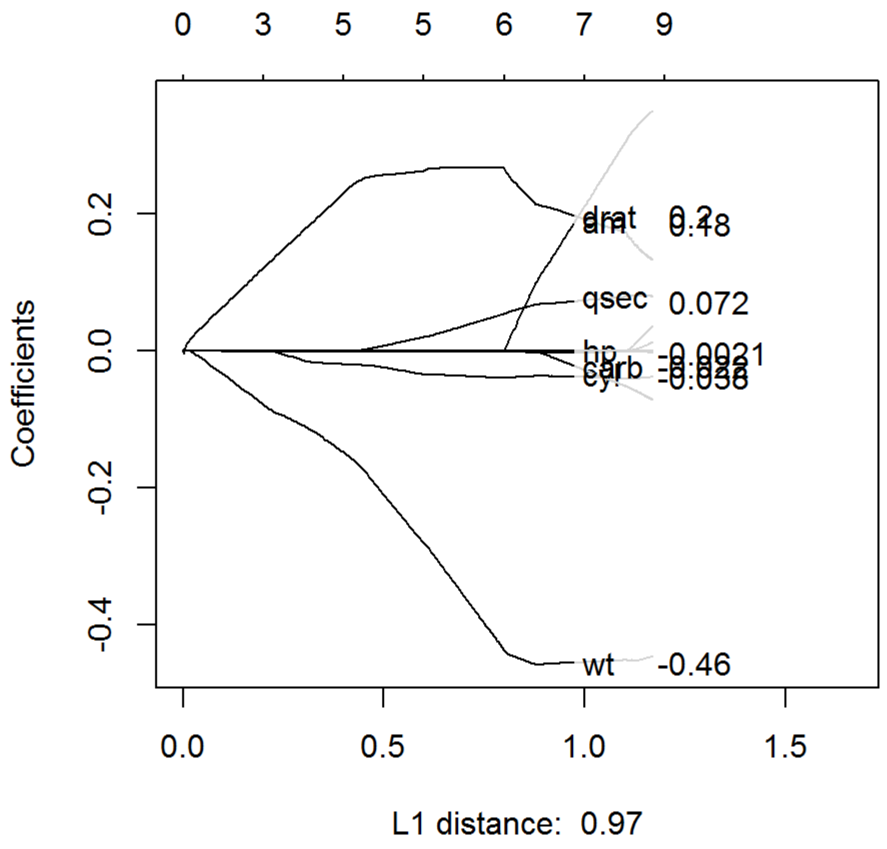


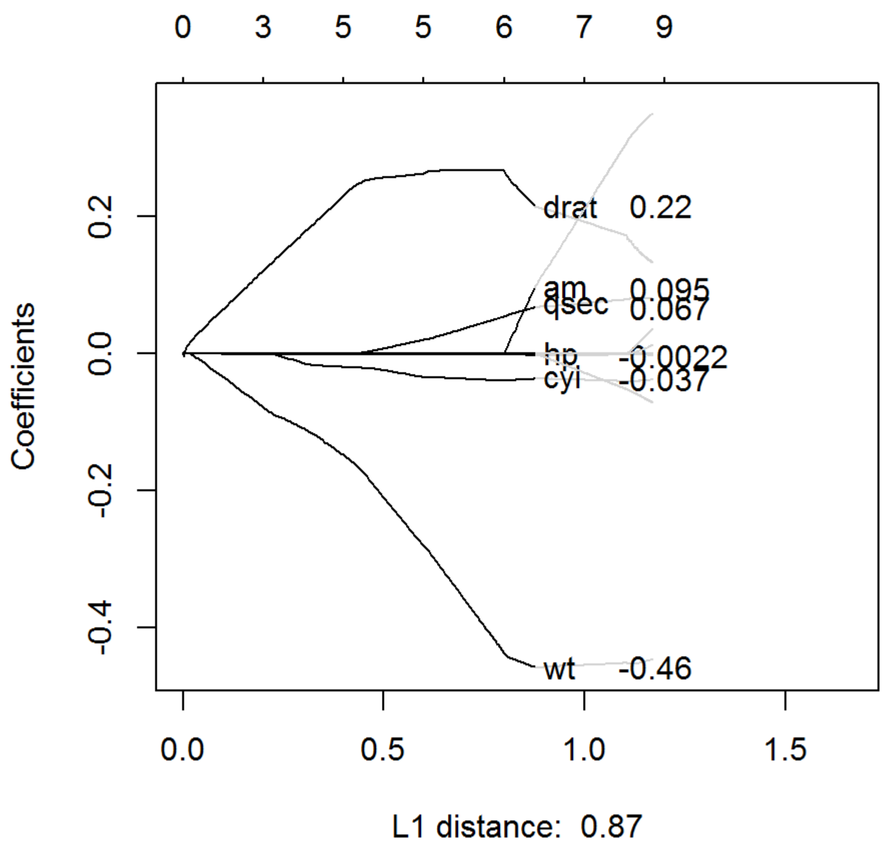
The plot above shows the shrinkage of the lasso coefficients as you move from the right to the left, but unfortunately, it is not clearly labelled. Here is a series of plots with better labels showing how the lasso coefficients change.

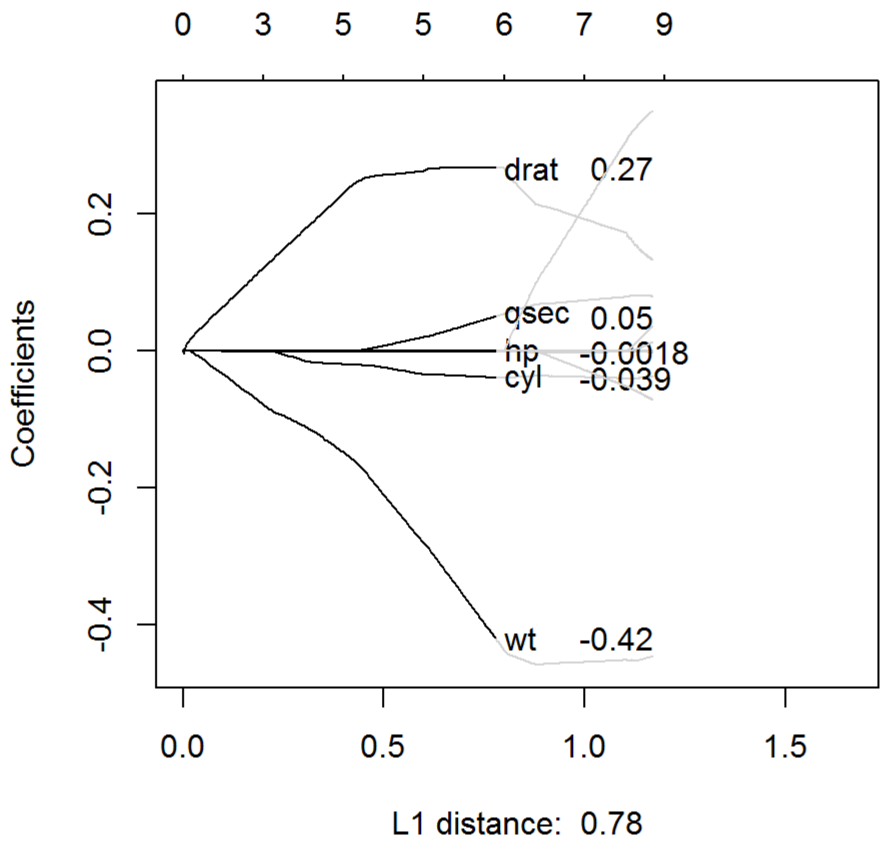
```r
n <- dim(lasso$beta)[1]
l1_dist <- apply(lasso$beta, 2, function(x) sum(abs(x)))
d1 <- 0.02*l1_dist[m_lasso]
d2 <- 0.18*l1_dist[m_lasso]
for (l in 12:1) {
  fraction <- l/12
  j <- max(which(l1_dist <= fraction*l1_dist[m_lasso]))
  xl <- paste("L1 distance: ", round(l1_dist[j], 2))
  plot(lasso, xlim=c(0, 1.4*max(l1_dist)),type="n", xlab=xl)
  offset <- strwidth("-", units="user")
  for (i in 1:n) {
    lines(l1_dist[1:m_lasso], lasso$beta[i, 1:m_lasso], col="lightgray")
    lines(l1_dist[1:j], lasso$beta[i, 1:j])
    if (abs(lasso$beta[i,j]) > 0) {
      text(d1+l1_dist[j], lasso$beta[i, j], names(mtcars[i+1]), adj=0)
      text(d2+l1_dist[j]+(lasso$beta[i, j]>0)*offset, lasso$beta[i, j], signif(lasso$
beta[i, j], 2), adj=0)
    }
  }
}
```

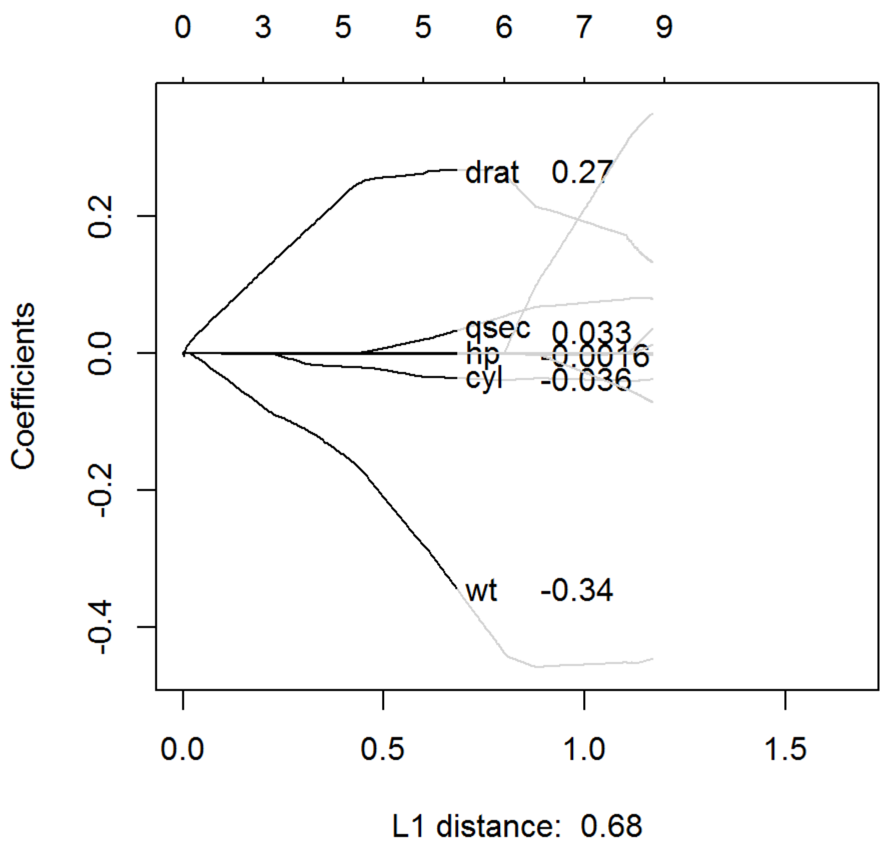


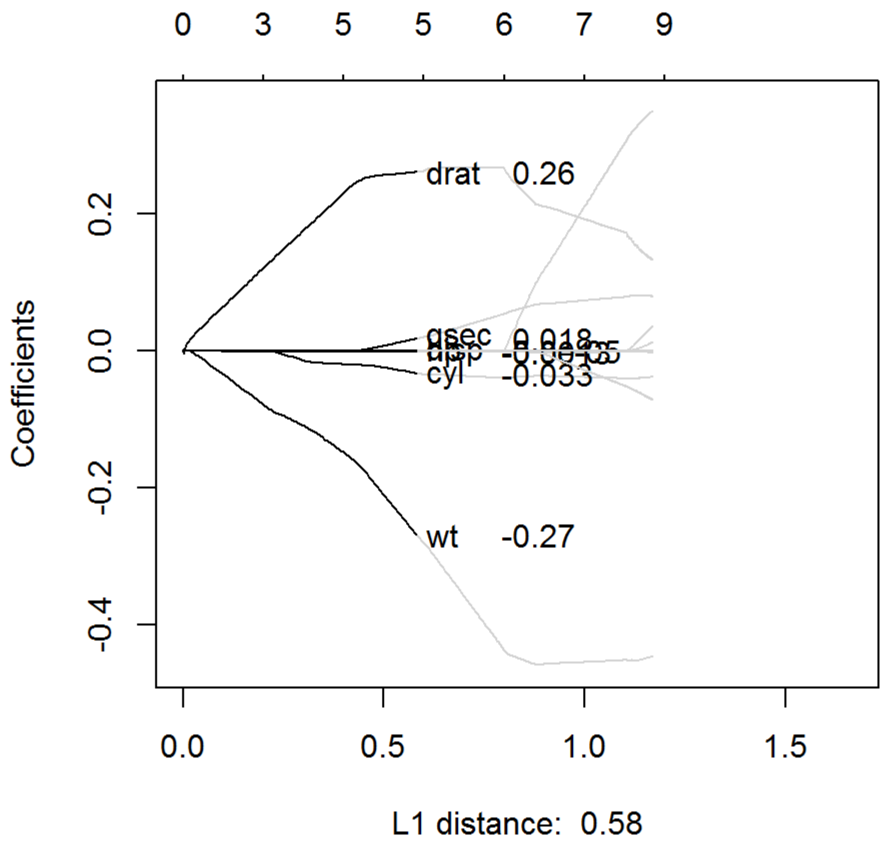

0 3 5 5 6 7 9
Coefficients
0.2
0.0
-0.2
-0.4
am 0.28
drat 0.18
qsec 0.077
hp -0.0021
cyl -0.04
carb -0.044
wt -0.45
0.0 0.5 1.0 1.5
L1 distance: 1.07

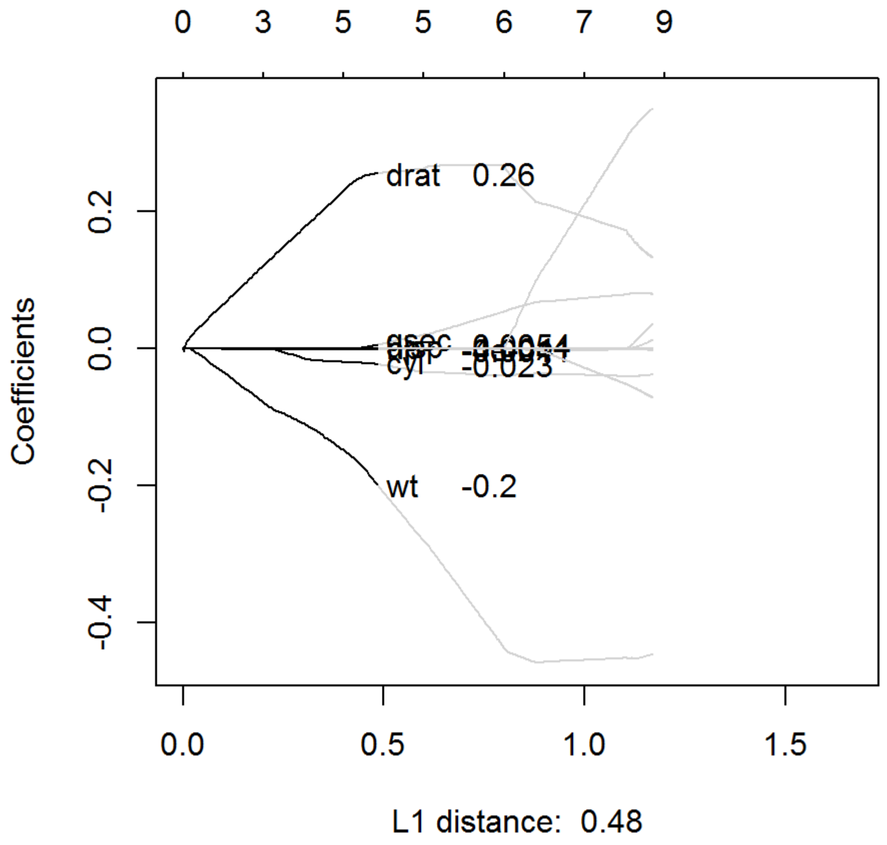

0 3 5 5 6 7 9
Coefficients
0.2
0.0
-0.2
-0.4
drat 0.2
am 0.18
qsec 0.072
hp -0.0021
carb -0.028
cyl -0.038
wt -0.46
0.0 0.5 1.0 1.5
L1 distance: 0.97

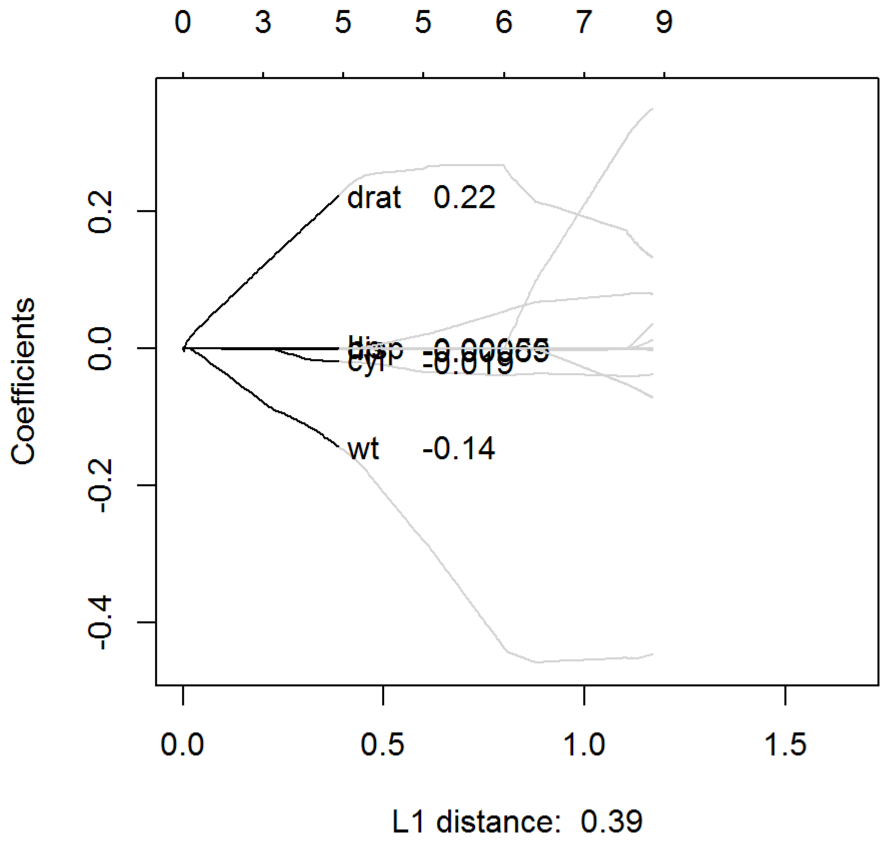

0 3 5 5 6 7 9
Coefficients
0.2
0.0
-0.2
-0.4
drat 0.22
am 0.095
qsec 0.067
hp -0.0022
cyl -0.037
wt -0.46
0.0 0.5 1.0 1.5
L1 distance: 0.87

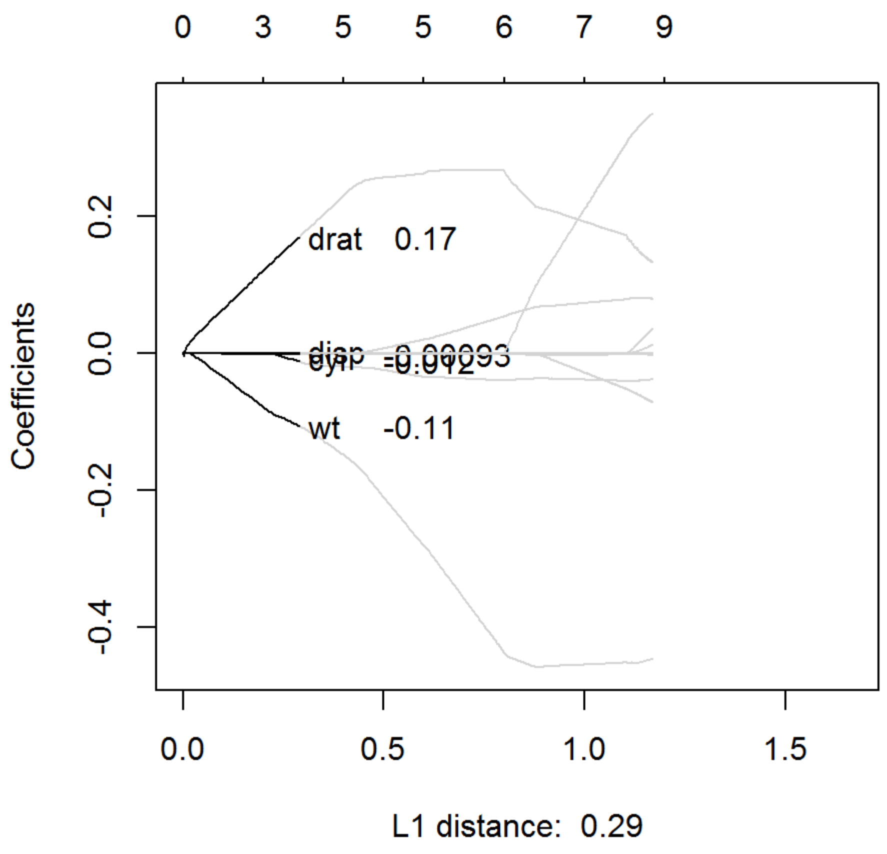

0 3 5 5 6 7 9
Coefficients
0.2
0.0
-0.2
-0.4
drat 0.27
qsec 0.05
hp -0.0018
cyl -0.039
wt -0.42
0.0 0.5 1.0 1.5
L1 distance: 0.78

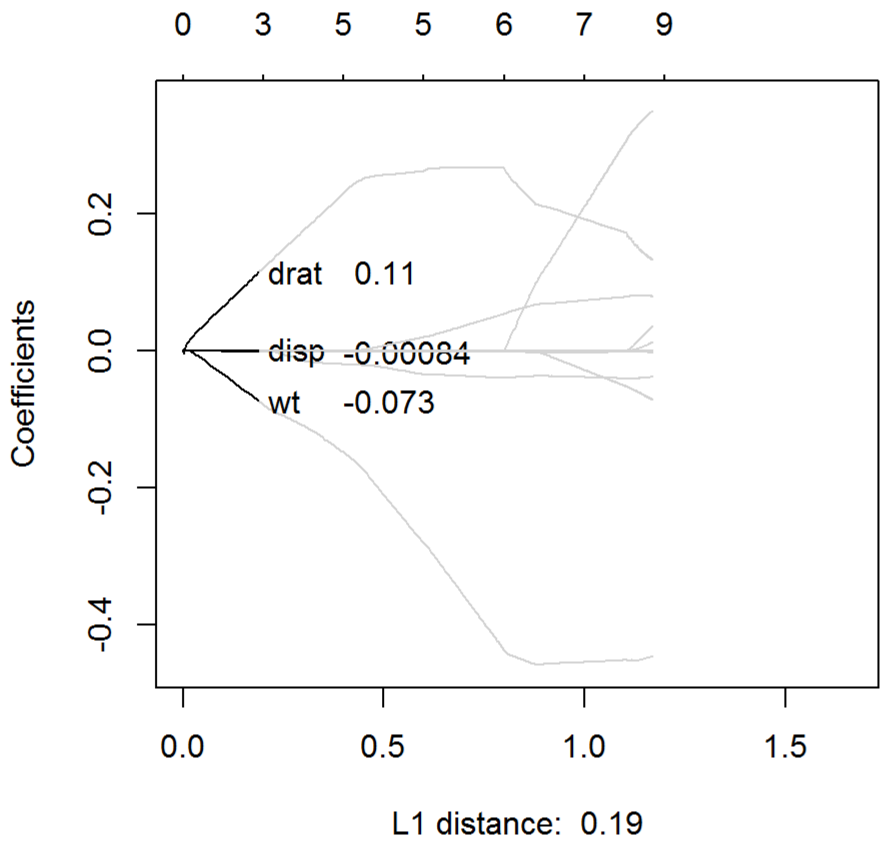

0 3 5 5 6 7 9
Coefficients
drat 0.27
qsec 0.033
hp -0.0016
cyl -0.036
wt -0.34
0.0 0.5 1.0 1.5
-0.4 -0.2 0.0 0.2
L1 distance: 0.68

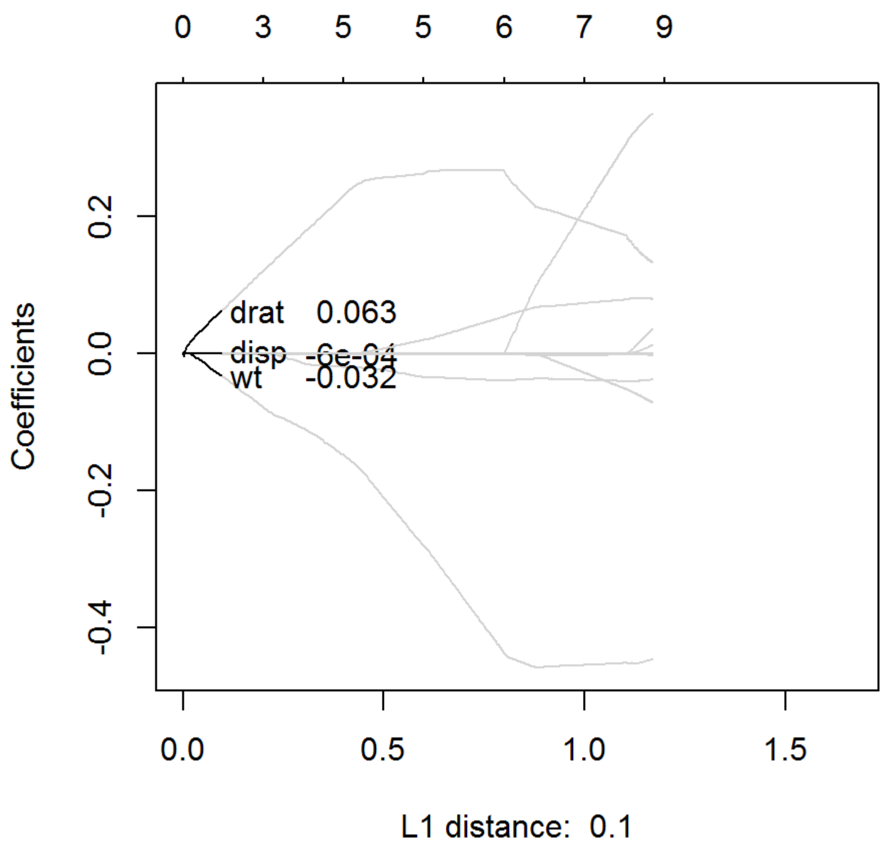

0 3 5 5 6 7 9
Coefficients
drat 0.26
qsec 0.018
cyl -0.033
wt -0.27
0.0 0.5 1.0 1.5
-0.4 -0.2 0.0 0.2
L1 distance: 0.58


0 3 5 5 6 7 9
Coefficients
0.2
0.0
-0.2
-0.4
drat 0.26
cyl -0.023
wt -0.2
0.0 0.5 1.0 1.5
L1 distance: 0.48


0 3 5 5 6 7 9
Coefficients
0.2
0.0
-0.2
-0.4
drat 0.22
cyl -0.019
wt -0.14
0.0 0.5 1.0 1.5
L1 distance: 0.39

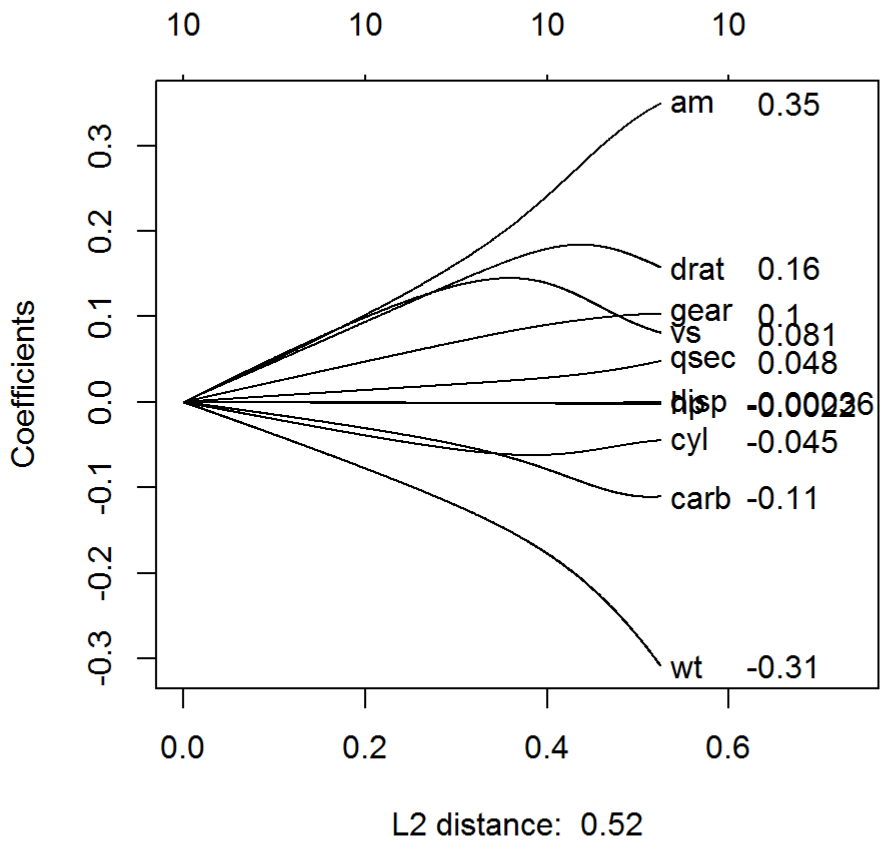

0 3 5 5 6 7 9
Coefficients
0.2
0.0
-0.2
-0.4
drat 0.17
disp -0.00093
cyl -0.012
wt -0.11
0.0 0.5 1.0 1.5
L1 distance: 0.29

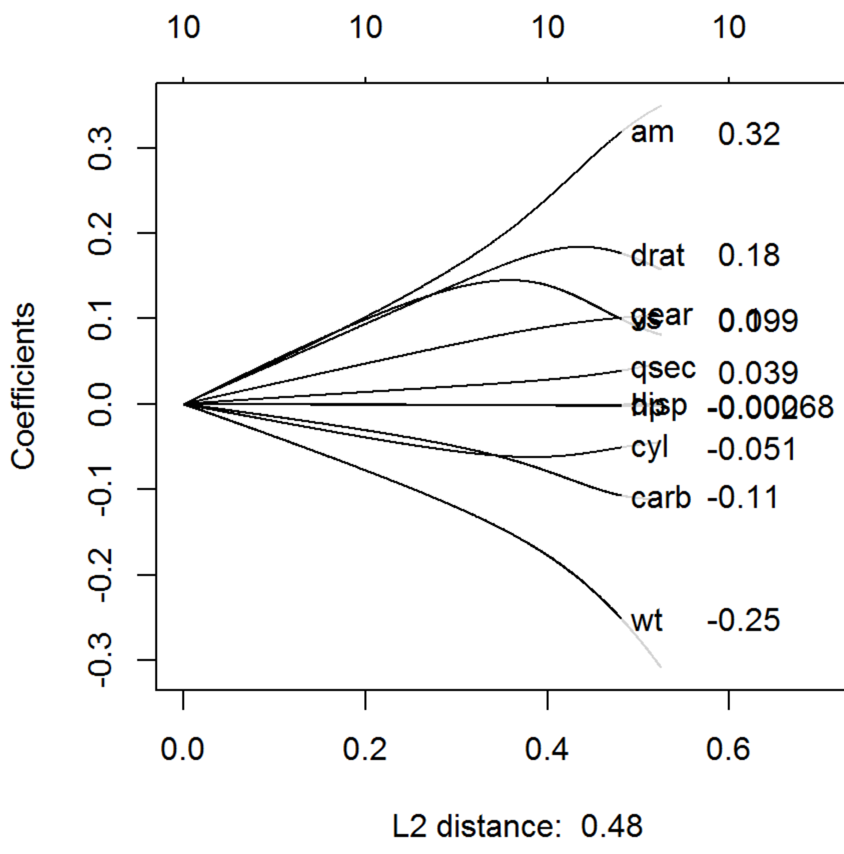

0 3 5 5 6 7 9
Coefficients
0.2
0.0
-0.2
-0.4
drat 0.11
disp -0.00084
wt -0.073
0.0 0.5 1.0 1.5
L1 distance: 0.19

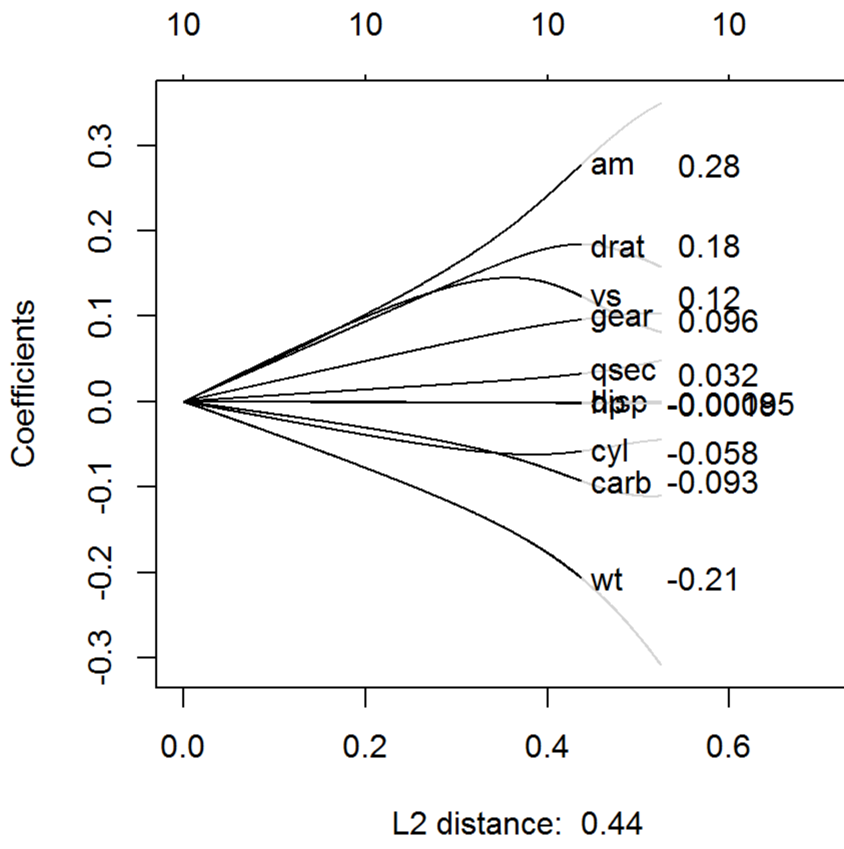


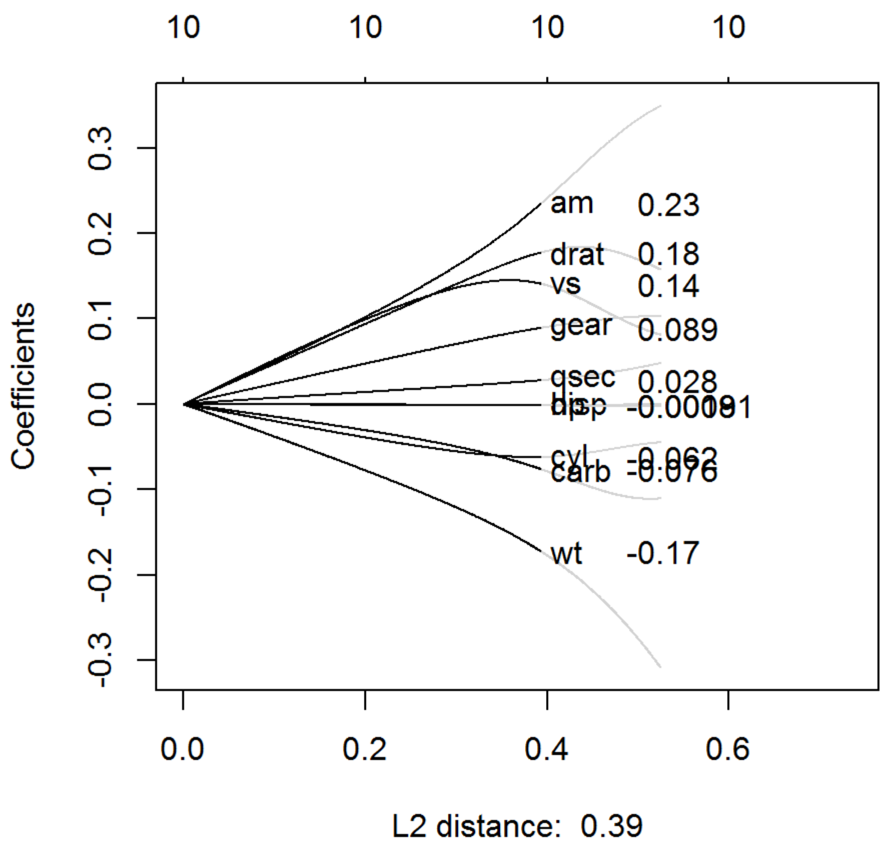
Contrast this with ridge regression, which flattens out everything, but doesn’t zero out any of the regression coefficients.

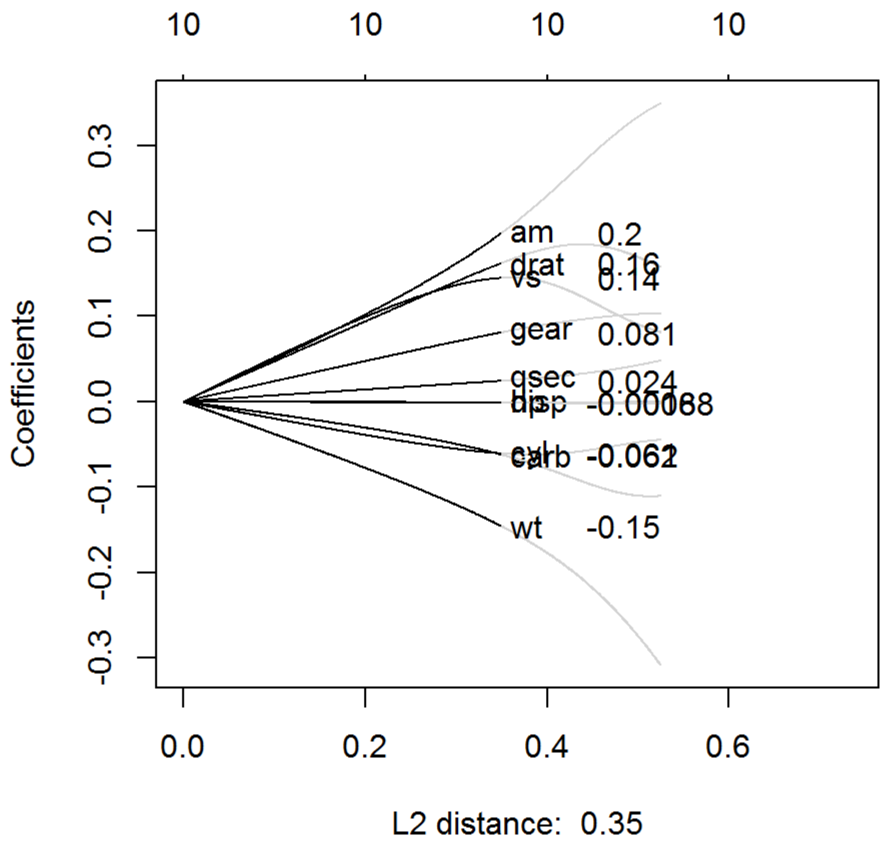
```
ridge <- glmnet(as.matrix(mtcars[, -1]), standardize(mtcars[, 1]), alpha=0, intercept
=FALSE, nlambda=1000)
m_ridge <- dim(ridge$beta)[2]
l2_dist <- apply(ridge$beta, 2, function(x) sqrt(sum(x^2)))
d1 <- 0.02*l2_dist[m_ridge]
d2 <- 0.18*l2_dist[m_ridge]
for (l in 12:1) {
  fraction <- l/12
  j <- max(which(l2_dist <= fraction*l2_dist[m_ridge]))
  xl <- paste("L2 distance: ", round(l2_dist[j], 2))
  plot(ridge, xlim=c(0, 1.4*max(l2_dist)),type="n", xlab=xl)
  offset <- strwidth("-", units="user")
  for (i in 1:n) {
    lines(l2_dist[1:m_ridge], ridge$beta[i, 1:m_ridge], col="lightgray")
    lines(l2_dist[1:j], ridge$beta[i, 1:j])
    if (abs(ridge$beta[i,j]) > 0) {
      text(d1+l2_dist[j], ridge$beta[i, j], names(mtcars[i+1]), adj=0)
      text(d2+l2_dist[j]+(ridge$beta[i, j]>0)*offset, ridge$beta[i, j], signif(ridge$
beta[i, j], 2), adj=0)
    }
  }
}
```

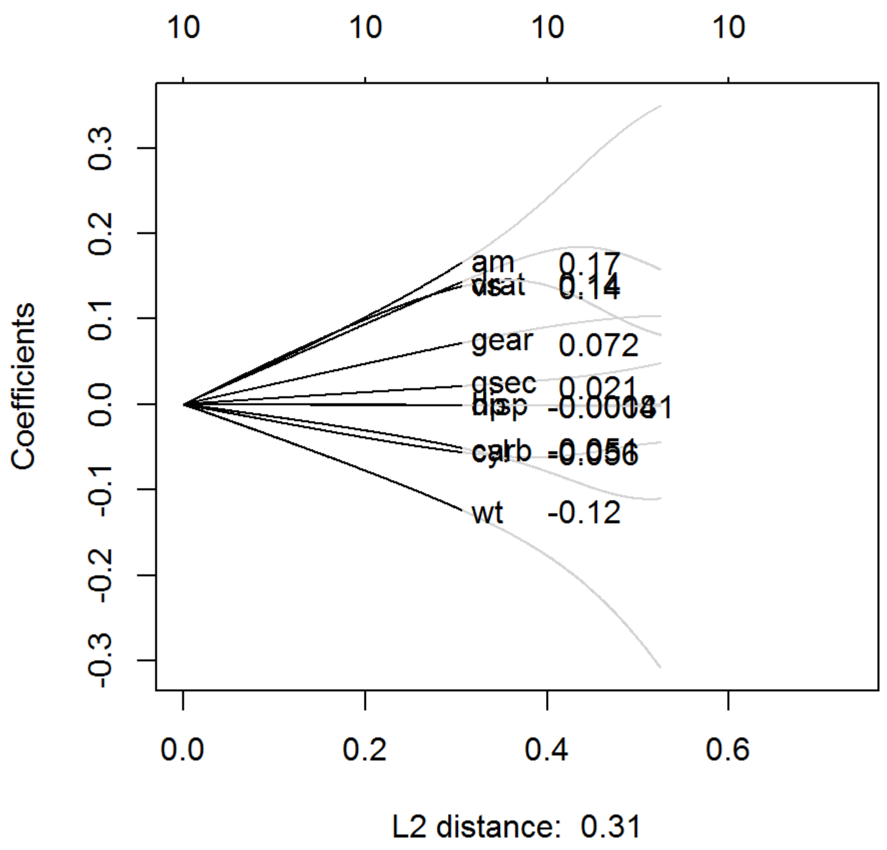


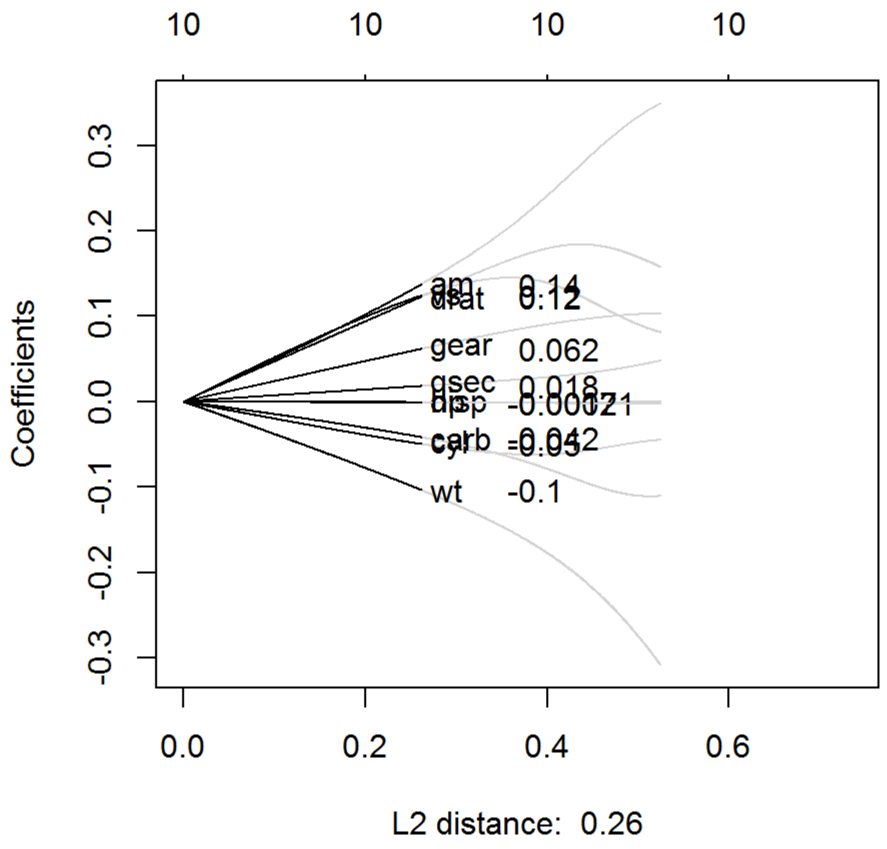

10 10 10 10
Coefficients
am 0.32
drat 0.18
gear
vs 0.1
0.099
qsec 0.039
disp
hp -0.00268
-0.002
cyl -0.051
carb -0.11
wt -0.25
L2 distance: 0.48

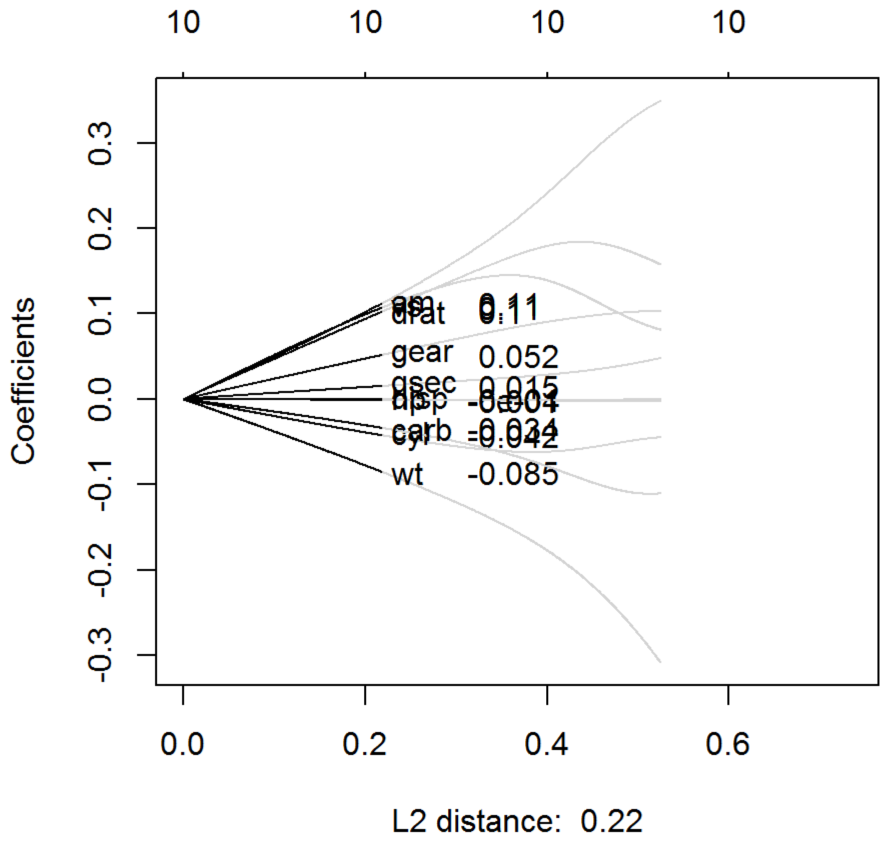

10 10 10 10
Coefficients
am 0.28
drat 0.18
vs 0.12
gear 0.096
qsec 0.032
disp
hp -0.00195
-0.001
cyl -0.058
carb -0.093
wt -0.21
L2 distance: 0.44

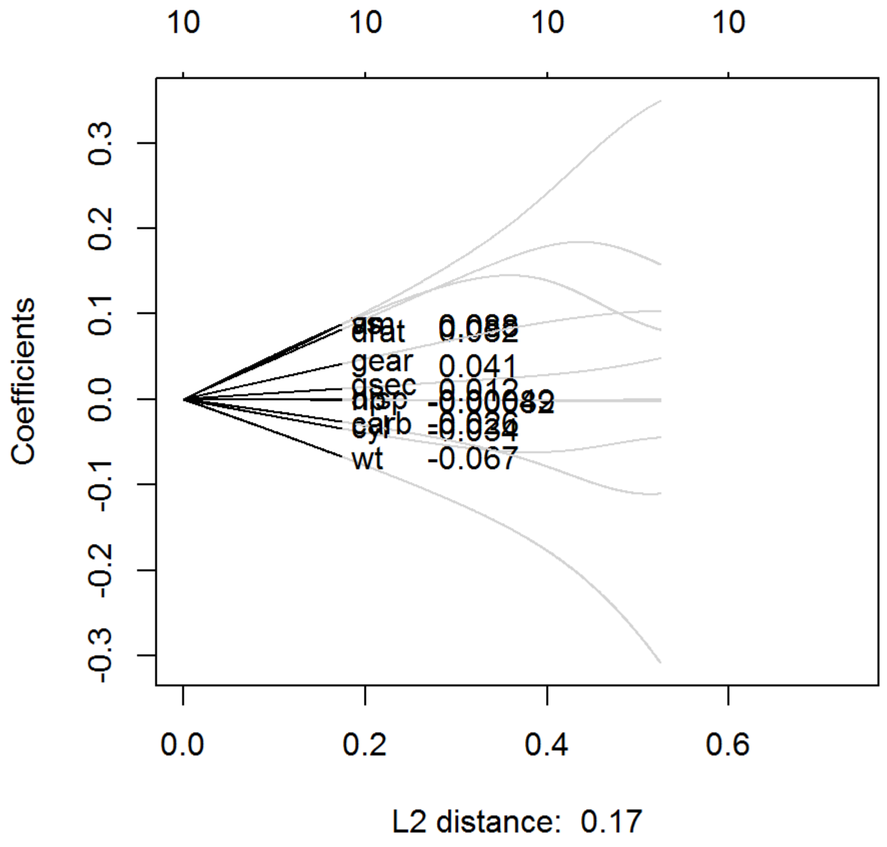

10
10
10
10
Coefficients
0.3
0.2
0.1
0.0
-0.1
-0.2
-0.3
am 0.23
drat 0.18
vs 0.14
gear 0.089
qsec 0.028
disp -0.00091
cyl -0.062
carb -0.076
wt -0.17
0.0
0.2
0.4
0.6
L2 distance: 0.39

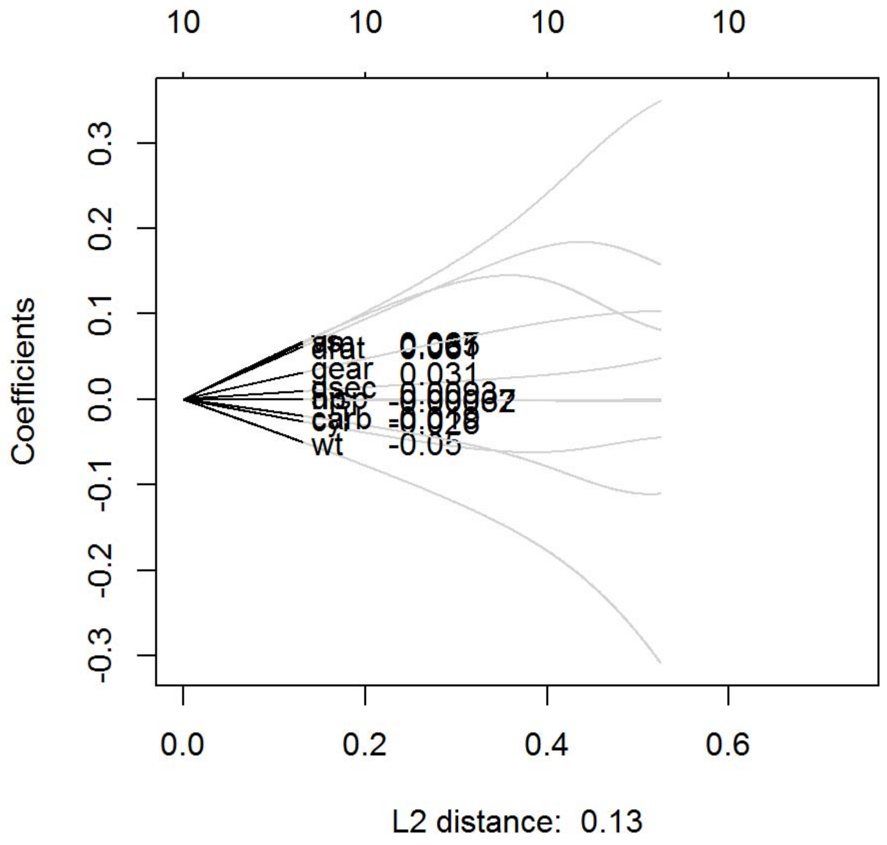

10
10
10
10
Coefficients
0.3
0.2
0.1
0.0
-0.1
-0.2
-0.3
am 0.2
drat 0.16
vs 0.14
gear 0.081
qsec 0.024
disp -0.00068
cyl -0.061
carb -0.062
wt -0.15
0.0
0.2
0.4
0.6
L2 distance: 0.35

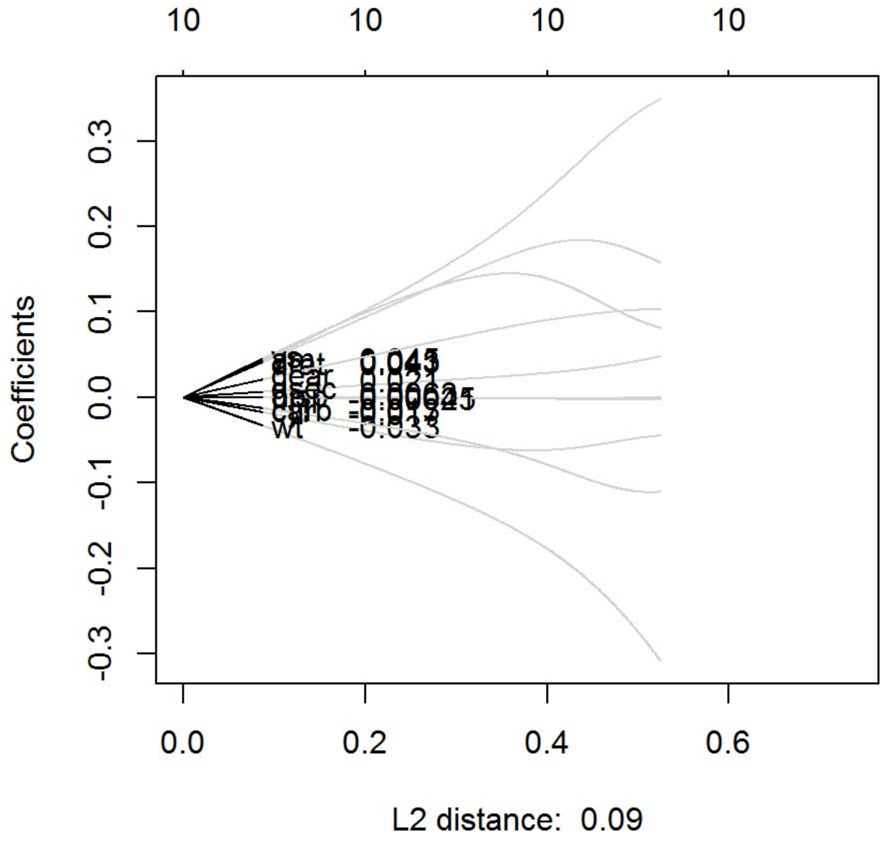

10
10
10
10
Coefficients
0.3
0.2
0.1
0.0
-0.1
-0.2
-0.3
am 0.17
vs 0.14
drat 0.14
gear 0.072
qsec 0.021
disp -0.00031
hp -0.00081
carb -0.051
cyl -0.056
wt -0.12
0.0
0.2
0.4
0.6
L2 distance: 0.31

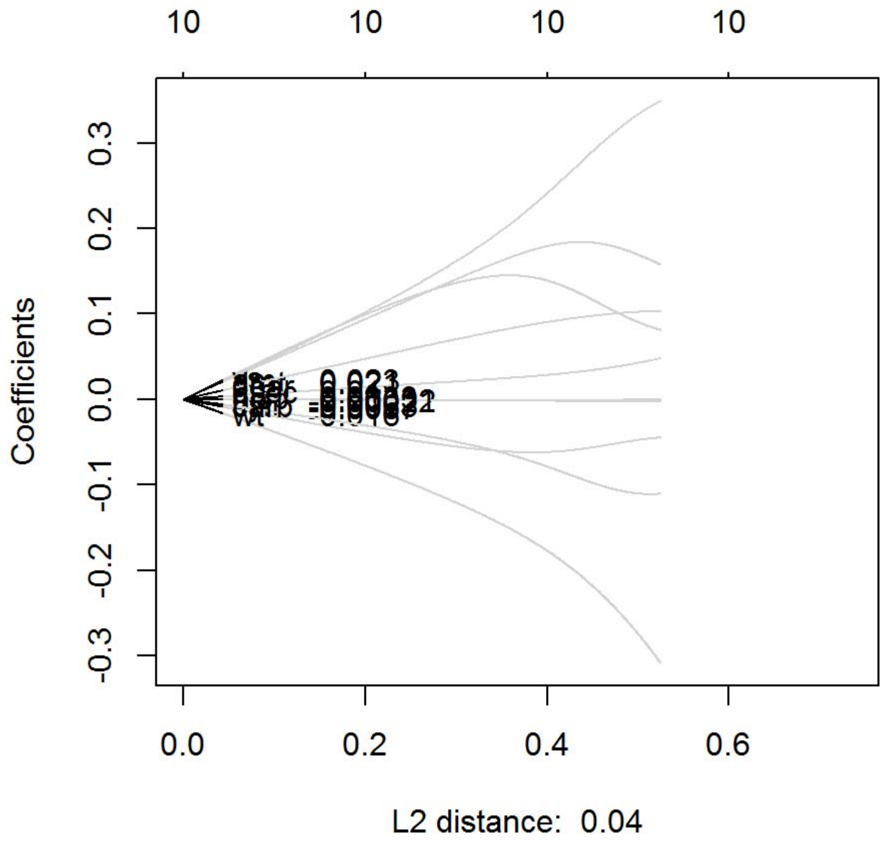

10
10
10
10
Coefficients
0.3
0.2
0.1
0.0
-0.1
-0.2
-0.3
am 0.14
vs 0.12
drat 0.12
gear 0.062
qsec 0.018
disp -0.00027
hp -0.00071
carb -0.042
cyl -0.05
wt -0.1
0.0
0.2
0.4
0.6
L2 distance: 0.26


10
10
10
10
Coefficients
0.3
0.2
0.1
0.0
-0.1
-0.2
-0.3
am 0.11
drat 0.11
gear 0.052
qsec 0.015
disp
hp -0.004
carb -0.034
cyl -0.042
wt -0.085
0.0
0.2
0.4
0.6
L2 distance: 0.22


10
10
10
10
Coefficients
0.3
0.2
0.1
0.0
-0.1
-0.2
-0.3
am 0.088
drat 0.082
gear 0.041
qsec 0.012
disp
hp -0.00082
carb -0.026
cyl -0.034
wt -0.067
0.0
0.2
0.4
0.6
L2 distance: 0.17


10
10
10
10
Coefficients
0.3
0.2
0.1
0.0
-0.1
-0.2
-0.3
0.0
0.2
0.4
0.6
drat
gear 0.031
wt -0.05
L2 distance: 0.13


10
10
10
10
Coefficients
0.3
0.2
0.1
0.0
-0.1
-0.2
-0.3
0.0
0.2
0.4
0.6
wt -0.033
L2 distance: 0.09



You can get more complicated than this. Although the lasso regression model does fairly well, it can sometimes get in a bit of trouble if there are several variables which are highly intercorrelated and which all predict the dependent variable with about the same strength. You might find better performance with Elastic Net regression, a hybrid model that combines some of the features of ridge regression with lasso regression.

Save everything for possible re-use.

```
save.image("lasso.RData")
load("backup.RData")
```