Regularization - Modern Model Fitting

Ethan Ashby

3/14/2020

Let's think about doing regression on a budget. Suppose that you have m variables, but have a ration on how much β you can use in your model.

The first thing we need to ask is, given a model, how do I think about how much β it uses. Do I get a discount for using negative values of β? How much β is being used in the model:

$y_i = \beta_0 + \beta_1 x_{i1} + \beta x_{i2} + \epsilon_i$

Let's set some ground rules. Let's not charge for intercept. The goal here is to fit a model. AIC penalizes the model by the number of parameters p. We can view this as doing something similar. Certainly, if we set many of the β_j values equal to 0, then we might expect the total amount of β to be small (though not necessarily). But these two ideas seem similar, penalizing large models (which are hard to interpret).

Here are two ideas. We can measure the size of β as either

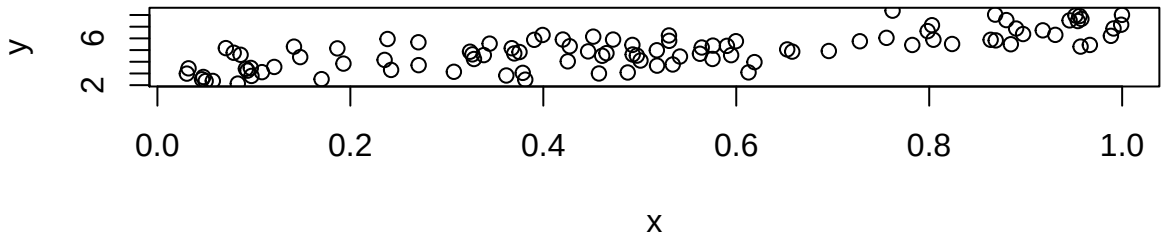
$$||\beta||_1 = \sum_{j=1}^{p-1} |\beta_j| \quad \text{or} \quad ||\beta||_2^2 = \sum_{j=1}^{p-1} \beta_j^2$$

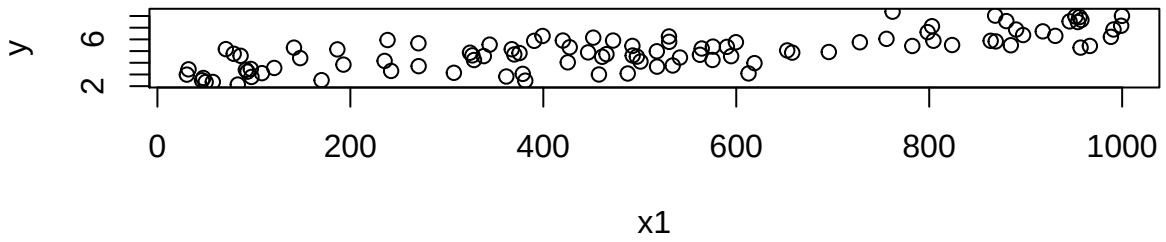
Both of these penalize negative and positive values the same, though the second one cares a lot about very large values. Note, the second of these is the Euclidan norm of the β vector, squared.

So here you go, given some data, here is your allowance. You are going to fit the best model you can without exceeding a total of k, i.e. $||\beta|| \leq k$

First, we need to make it so you can't cheat. If the goal was to establish the relationship between study hours and score on an exam, we could measure time in nano-seconds. Given the interpretation of the slope as the expected change in score for a unit change in *time*, the slope would be exceedingly small. An extra nano-second of studying shouldn't change your performance much at all. How economical of you! But if you measured it in days, well, an extra 24 hours of studying is probably going to be valuable. But these will be the exact same fits! One is a linear transformation of the other. The scatter plot will look identical. Here, see for yourself.

```
x <- runif(100)
y <- 3 + 4 * x + rnorm(100)
par(mfrow=c(2,1))
plot(x,y)
x1 <- 1000*x #transformed data of the original variable
plot(x1,y)
```





We can also compare the SSE values of the respective fits

```
anova(lm(y~x))$Sum[2]
```

```
## [1] 97.29819
```

```
anova(lm(y~x1))$Sum[2]
```

```
## [1] 97.29819
```

How do we keep you from cheating? Let's force every explanatory variable in your data set onto the same scale, so called z-scaling. (associating each value with its z-score.)

This means that for every one of your m x variables, $\mu_x = 0$ and $\sigma_x = 1$

I can do this for a variable using the scale command.

```
#the data we just created
cbind(x[1:10], x1[1:10])
```

```
##            [,1]     [,2]
##  [1,] 0.3992275 399.2275
##  [2,] 0.8636864 863.6864
##  [3,] 0.6959444 695.9444
##  [4,] 0.5996339 599.6339
##  [5,] 0.8845838 884.5838
##  [6,] 0.9883399 988.3399
##  [7,] 0.9455150 945.5150
##  [8,] 0.7824447 782.4447
##  [9,] 0.8901110 890.1110
## [10,] 0.4969829 496.9829
```

```
#and the scaled version
cbind(scale(x)[1:10], scale(x1)[1:10])
```

```
##              [,1]        [,2]
##  [1,] -0.34585517 -0.34585517
```

```
##  [2,]  1.22054756  1.22054756
##  [3,]  0.65483223  0.65483223
##  [4,]  0.33002205  0.33002205
##  [5,]  1.29102489  1.29102489
##  [6,]  1.64094557  1.64094557
##  [7,]  1.49651733  1.49651733
##  [8,]  0.94655729  0.94655729
##  [9,]  1.30966546  1.30966546
## [10,] -0.01617202 -0.01617202
```

The R function for the idea that we are working towards is going to do this automatically (and then scale back!).

So what should we do with rationed β? We know what we'd do otherwise, minize the sum of squares. Let's do the same thing here, but get it as small as we can while still satisfying our constraint.

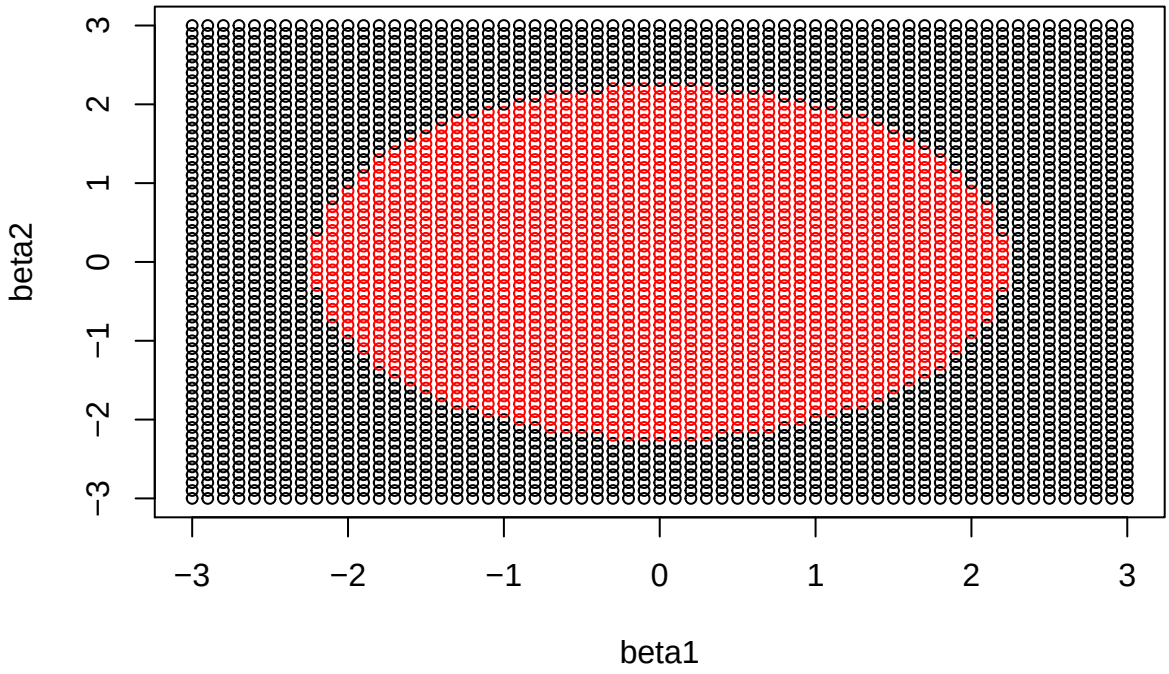
Let's draw some pictures, and we'll do it supposing that we only have two predictor variables (assumed to be on the same scale): X_1 and X_2.

Suppose that you are alloted a total of k beta to spend (i.e. $||\beta||_1 \leq k$ or $||\beta||_2^2 \leq k$). Let's see what we can buy.

I'll create a grid of points, and see which ones are acceptable solutions.

First, with the "2-norm"

```
k <- 5
plot(c(-3,3), c(-3,3), t='n', xlab="beta1", ylab="beta2")
x <- seq(-3,3,.1)
for (i in 1:length(x)) {
  for (j in 1:length(x)) {
    points(x[i], x[j], col=1+((x[i]^2+x[j]^2)<k), cex=.75)
  }
}
```

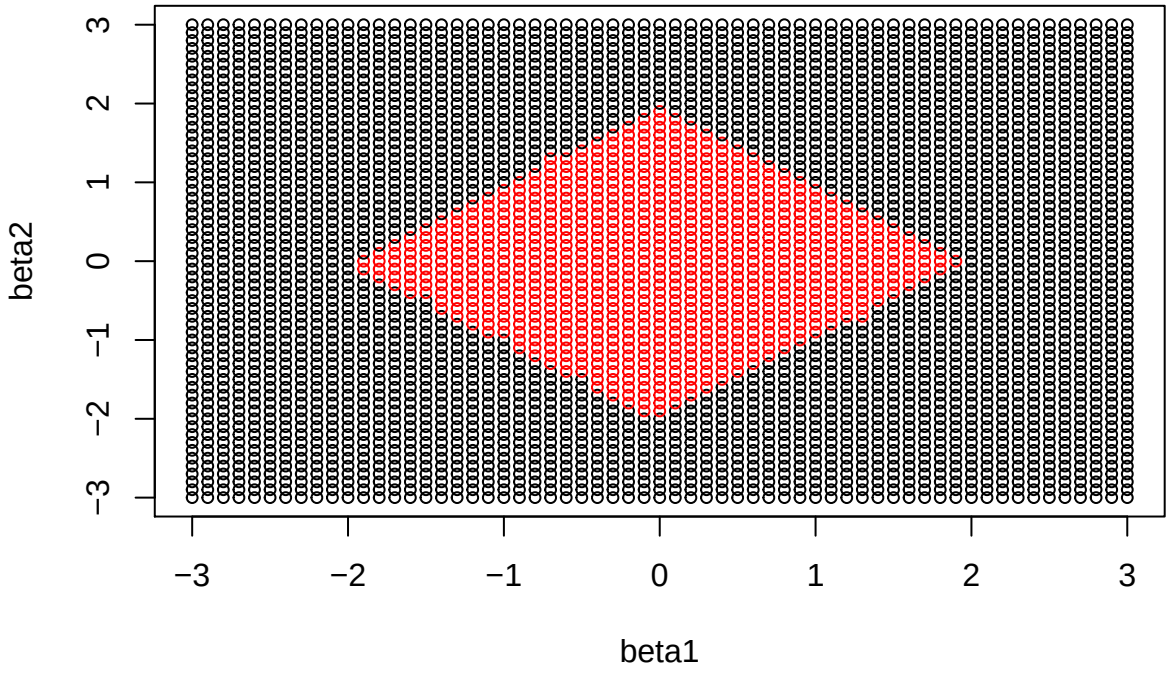


You maybe realize that you didn't need the computer to realize what set was allowed. The equation $\beta_1^2+\beta_2^2 \leq k$

is the ball with radius $\sqrt{k}$ (the circle and the interior), and more generally $\sum_j^p \beta_j^2 \leq k$ is the p-dimensional ball (the hypersphere and everything inside).

Let's do it for the "1-norm" (absolute values as measuring the size).

```
k <- 2
plot(c(-3,3), c(-3,3), t='n', xlab="beta1", ylab="beta2")
x <- seq(-3,3,.1)
for (i in 1:length(x)) {
  for (j in 1:length(x)) {
    points(x[i], x[j], col=1+(abs(x[i])+abs(x[j])<k), cex=.75)
  }
}
```



It's a diamond. Higher dimensional analogs continue to be "pointy" (this is crucial).

So these are the regions I'm allowed, and this is what they look like. Now, what should we do? Where in there should we choose? We decided to continue to minimize the sum of squares. Two things could happen.

1. The least squares solution is in your budget. Then, we walk away with our old answer.
   This is as if it was your birthday, and you wanted a friend to take you out for dinner. Your friend agrees, but admits to only having 100 dollars in their bank account. If the restaurant you wanted to go to was Taco Bell, then you get Taco Bell. But if you wanted to go to a 3-star Michelin restaurant, well. That's the second situation.

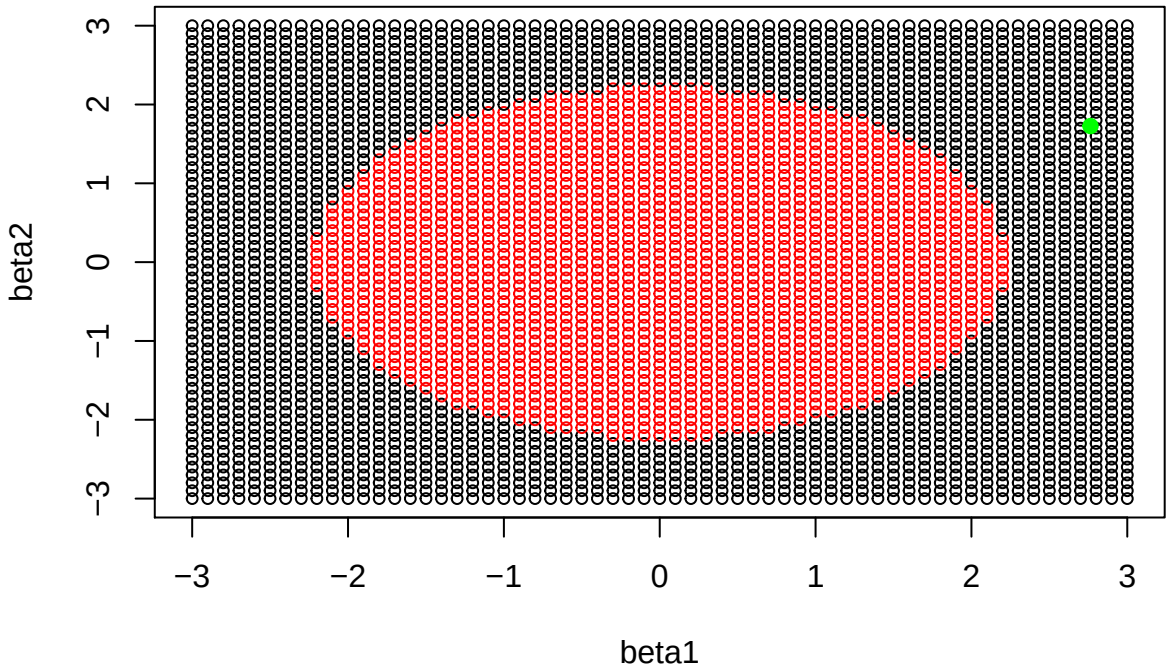
Let's generate some regression data.

```
set.seed(47)
n <- 100
x1 <- rnorm(n)
x2 <- rnorm(n)
y <- 3 + 2.5 *x1 + 2 * x2 + rnorm(n, 0, 2)
data.lab <- data.frame(y,x1,x2)
fit <- lm(y~x1+x2)
fit  #the OLS solutions, including the intercept that we aren't worried about
```

```
##
## Call:
## lm(formula = y ~ x1 + x2)
##
## Coefficients:
## (Intercept)           x1           x2
##       3.310        2.764        1.727
```

Let's add this to the plot above, we'll use the one with the squares defining the distance.

```
k <- 5
plot(c(-3,3), c(-3,3), t='n', xlab="beta1", ylab="beta2")
x <- seq(-3,3,.1)
for (i in 1:length(x)) {
  for (j in 1:length(x)) {
    points(x[i], x[j], col=1+((x[i]^2+x[j]^2)<k), cex=.75)
  }
}
points(fit$coef[2], fit$coef[3], col='green', pch=19)
```



Darn it, we can't afford it. (green dot) Currently, the sum of squares (SSE) is

```
anova(fit)$Sum[3]
```

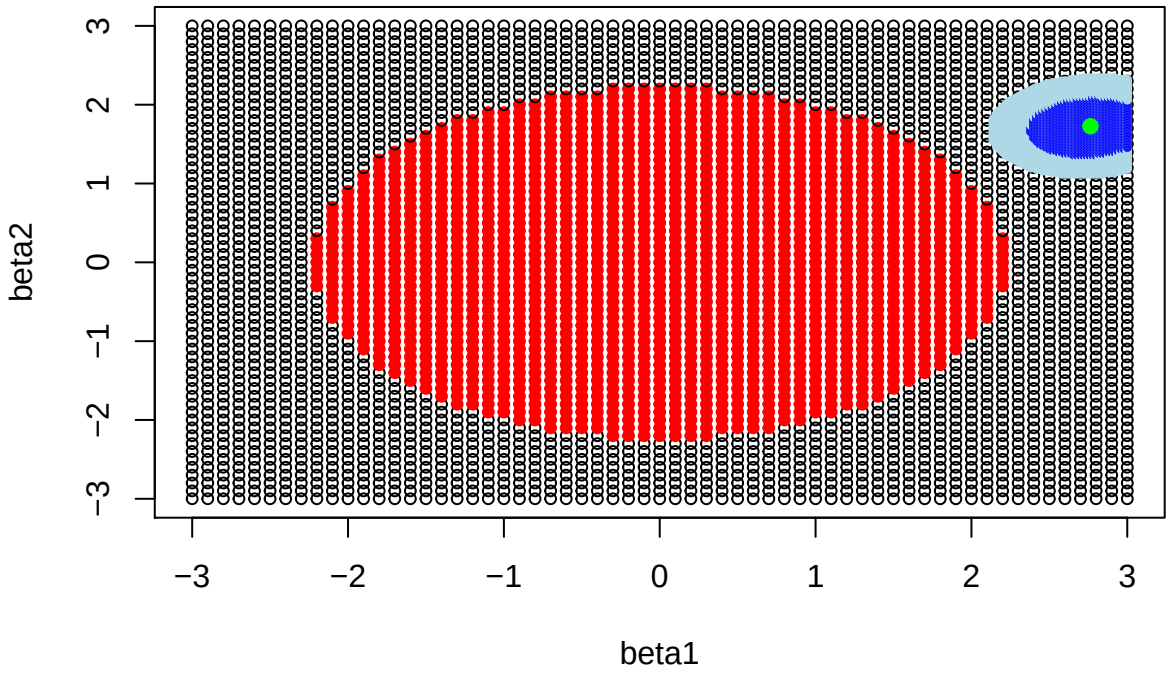
```
## [1] 410.8021
```

So we know we won't be able to get the sum of squares that small. It's outside of our price range. Let's look at the collection of β values that make the sum of squares no bigger than 425 and 450 respectively.

```
k <- 5
plot(c(-3,3), c(-3,3), t='n', xlab="beta1", ylab="beta2")
x <- seq(-3,3,.1)
for (i in 1:length(x)) {
  for (j in 1:length(x)) {
    points(x[i], x[j], col=1+((x[i]^2+x[j]^2)<k), cex=.75, pch=1+18*((x[i]^2+x[j]^2)<k))
  }
```

```
}


betas <- seq(1,3,.02)
for (i in 1:length(betas)) {
  for (j in 1:length(betas)) {
    sse <- sum((y-fit$coef[1]-betas[i]*x1-betas[j]*x2)^2)
    points(betas[i], betas[j], col="lightblue", cex=.5*(sse<450))
    points(betas[i], betas[j], col="blue", cex=.5*(sse<425))
  }
}
points(fit$coef[2], fit$coef[3], col='green', pch=19)
```



They are (nested) ellipses! The boundaries of which have the property that SSE=425 (dark blue) or 450 (light blue) respectively. (In higher dimensions, they will be ellipsoids). The orientation of this ellipsoid is related to how the two variables relate to each other.

To find the smallest sum of squares we can afford, we simply increase SSE (which "inflates" the ellipse), until the ellipse touches the red circle. This is then our model (the smallest SSE that we can afford).

Rather than budget before hand, this is equivalent to penalizing least squares, and minimizing

$$\text{SSE}(\beta) + \lambda \sum \beta_j^2$$

where λ needs to be chosen but there is a one-to-one relationship between a λ here and a k value in your budgeted problem. This is called ridge regression, the solution is straight forward using linear algebra, namely (if you are interested)

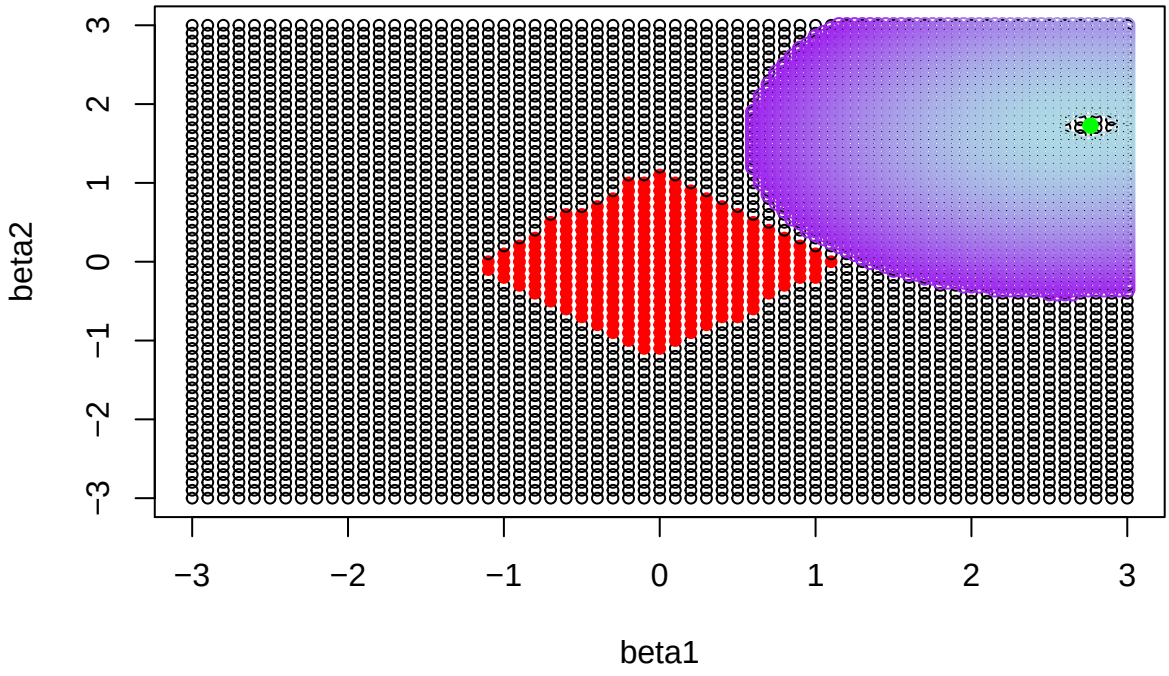
$$\hat{\beta}_{RR} = (X^T X + \lambda I)^{-1} X^T Y$$

It gives you back values of $\hat{\beta}$ that are "shrunk" towards 0 from the OLS (ordinary least squares) solution. Not terribly interesting. And doesn't help in model selection. Let's check that other definition of distance, using absolute values.

```
k <- 1.2
plot(c(-3,3), c(-3,3), t='n', xlab="beta1", ylab="beta2")
x <- seq(-3,3,.1)
for (i in 1:length(x)) {
  for (j in 1:length(x)) {
    points(x[i], x[j], col=1+(abs(x[i])+abs(x[j])<k), cex=.75, pch=1+18*(abs(x[i])+abs(x[j])<k))
  }
}

colfunc <- colorRampPalette(c("lightblue", "purple"))
cols <- colfunc(100)

betas <- seq(-3,3,.05)
sse <- c()
for (i in 1:length(betas)) {
  for (j in 1:length(betas)) {
    sse <- sum((y-fit$coef[1]-betas[i]*x1-betas[j]*x2)^2)
    points(betas[i], betas[j], col=cols[100*(sse-410)/(870-410)], cex=(sse<870))
  }
}
points(fit$coef[2], fit$coef[3], col='green', pch=19)
```



What you might believe is happening (and is going to happen much more dramatically in higher dimensions) is that this ellipse is going to first touch the "ball" at the corner. This cooordinate is $\beta_1 = k$ and $\beta_2 = 0$. This results in a fit that has removed one of the variables!
Note: we don't actually want to remove one of the variables here, they are both useful. Let's rerun this with X_2 garbage.

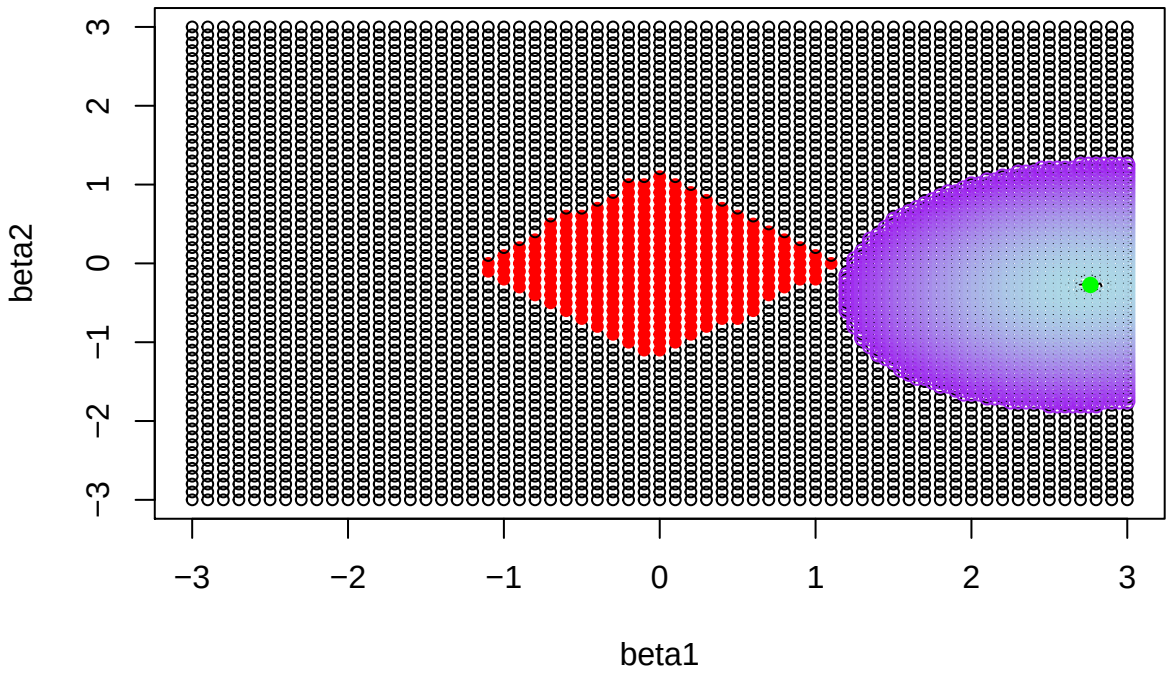
```
set.seed(47)
n <- 100
x1 <- rnorm(n)
x2 <- rnorm(n)
y <- 3 + 2.5 *x1 + rnorm(n, 0, 2)  #x2 isn't in the data generation!!!
```

```
data.lab <- data.frame(y,x1,x2)
fit <- lm(y~x1+x2)
fit
```

```
##
## Call:
## lm(formula = y ~ x1 + x2)
##
## Coefficients:
## (Intercept)           x1           x2
##      3.3101       2.7638      -0.2733
```

```
k <- 1.2
plot(c(-3,3), c(-3,3), t='n', xlab="beta1", ylab="beta2")
x <- seq(-3,3,.1)
for (i in 1:length(x)) {
  for (j in 1:length(x)) {
    points(x[i], x[j], col=1+(abs(x[i])+abs(x[j])<k), cex=.75, pch=1+18*(abs(x[i])+abs(x[j])<k))
  }
}

betas <- seq(-3,3,.05)
for (i in 1:length(betas)) {
  for (j in 1:length(betas)) {
    sse <- sum((y-fit$coef[1]-betas[i]*x1-betas[j]*x2)^2)
    points(betas[i], betas[j], col=cols[100*(sse-410)/(650-410)], cex=(sse<650))
  }
}
points(fit$coef[2], fit$coef[3], col='green', pch=19)
```



The "budgeted" solution is going to remove X_2 from the model. Perfect! This is again equivalent to the optimization problem to minimize

$$SSE(\beta) + \lambda \sum |\beta_j|$$

with a correspondance between k and λ. This will tend to remove variables from your model. Note, it will also tend to shrink the non-zero β values away from the OLS solution. More on this in a bit.

The solution to this optimization problem is more complicated, but it is computationally pretty simple. This idea is fairly modern, and is known as the LASSO (Least Absolute Shrinkage and Selection Operator).

Next question: how do I choose λ. We should have asked this when we first saw AIC. How does one balance the quality of fit with the size of the model. Notice that this is very similar, instead of measuring the size of the model by the number of variables (which makes the most sense to me), we measure the size of the model by the size of the coefficients. But very similar.

Well, we didn't ask that question before. What should a good model do. Well, the right model is really good at predicting future data. In fact, the best guess for some new y_h at some collection of x values is μ_{y_h}, i.e. the linear with the correct β values (assuming the data comes from this model). So what we should do is pick the model that does best on future data. Oh, wait. Ain't got that.

Cross Validation

Let's pretend we have future data. We can do this by hiding some of our data from the model fitting, and then seeing how the model guesses it. We'll split our data into a training set and a test set (pseudo-future data).

A good model is going to guess it well.

Too small of a model is going to return too simple of a model, and will leave information on the table.

Too complicated of a model is going to overfit the training data, thinking it's learned signal that is actually just noise, and try to use that noise to predict future observations.

We do this via $k-$fold cross validation. For some value of k, we remove a random $\frac{1}{k}$ proportion of our data, and do that k times, averaging the prediction error each time. The best value of λ is the one that has the lowest average prediciton error.

New R Package - GLMNET

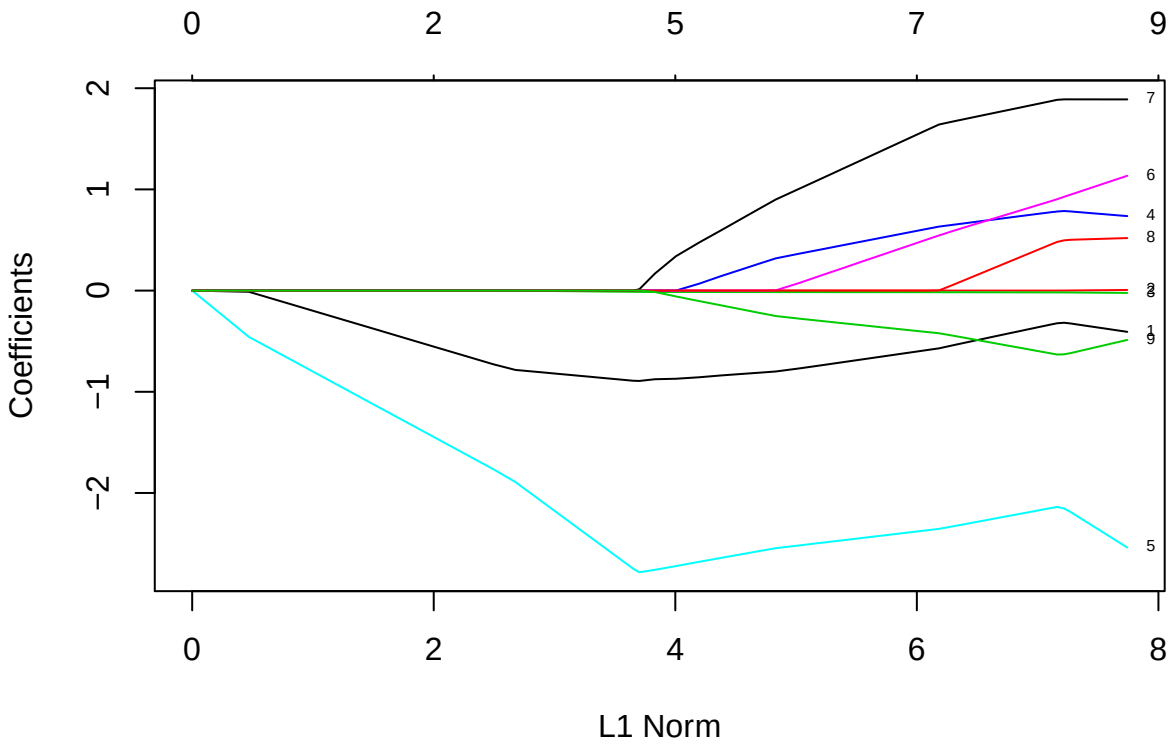
Let's check this out. The package will need to be installed via install.packages('glmnet')

The authors have made a nice vignette for this package, available at https://web.stanford.edu/~hastie/glmnet/glmnet_alpha.html

```
data(mtcars)
y <- mtcars$mpg
x <- as.matrix(mtcars[,-c(1,7)]) #remove mpg and qsec, now this is only predictors
```

Note that x need to be a matrix, not a data frame. I use as.matrix to convert it.

```
fit <- glmnet(x, y,alpha = 1) #alpha=1 means lasso, you can use a mix of ridge (alpha=0) and lasso pena
plot(fit, label=TRUE)
```

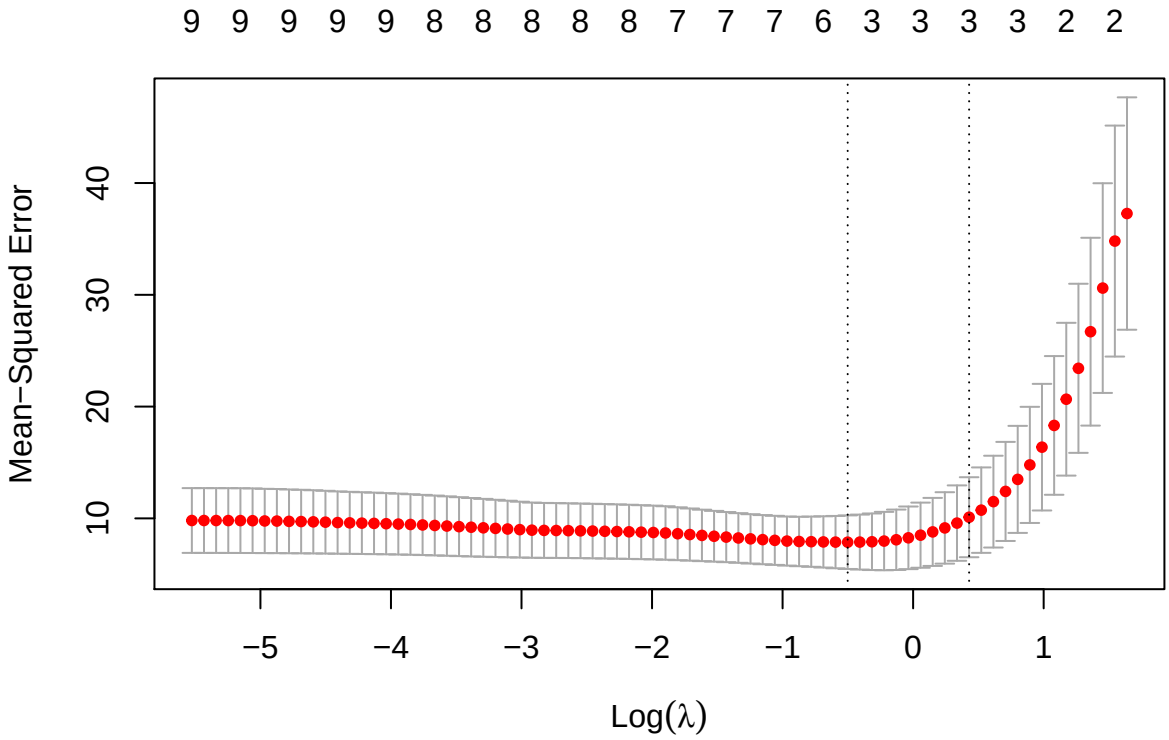


This shows the values of the coefficients as our old budget k increases. As k gets bigger, more variables are added to the model. Eventually, the ball is so big, that it contains the OLS solution.

Notice that for a very long time (lots of values of k), the data prefers to have only two variables in there (which is a model that we've found using a couple of different methods.) It's cylinders and weight.

But which of these is best? Cross-validate.

```
cv1=cv.glmnet(x,y,alpha=1)
plot(cv1)
```



Across the top is the number of variables in the model. We want the mean-square prediction error to be small. The two vertical dashed lines show where it is minimized, as well as the sparsest model that has a similar mean square error that the minimumizer (Occam!).

We can ask for those values.

```
cv1$lambda.min
```

```
## [1] 0.6057029
```

```
cv1$lambda.1se
```

```
## [1] 1.535678
```

We can then fit the model.

```
fit.1se <- glmnet(x, y,alpha = 1, lambda=cv1$lambda.1se)
fit.1se$beta
```

```
## 9 x 1 sparse Matrix of class "dgCMatrix"
##                s0
## cyl  -0.832970031
## disp  .
## hp   -0.005894918
## drat  .
## wt   -2.288075473
## vs    .
## am    .
## gear  .
## carb  .
```

Notice that there are 3 variables in this model, one of which looks pretty close to zero (but we'd have to consider scale to make sense of this). We thought we liked the model with 2 variables.

Here's the problem. The two-variable model doesn't fit great because the coefficients are shrunk from the

two-variable model you know. So we increase λ so they can grow, but by the time they've reached a reasonable size, a 3rd variable has snuck in, if only slightly.

The creators of this idea ran a massive simulation study, comparing LASSO and Ridge and AIC, etc, and found that probably the best thing you can do is called the "relaxed LASSO". You let LASSO choose the variables, and then you use a combination of LASSO and OLS values of β in your model. They found good results for either using $\hat{\beta}_{OLS}$ or $.5\hat{\beta}_{OLS} + .5\hat{\beta}_{LASSO}$.

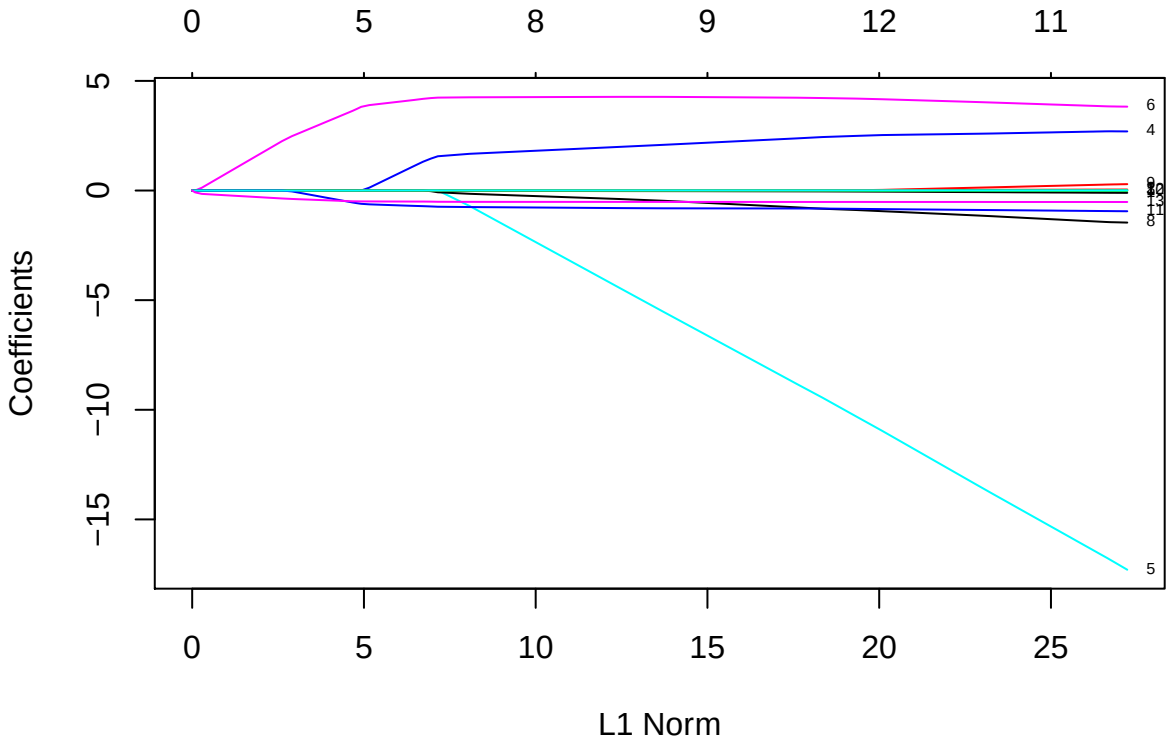
The seems to be implemented in the package RELAXO!
Feel free to check it out. I haven't tried it.

Homework:

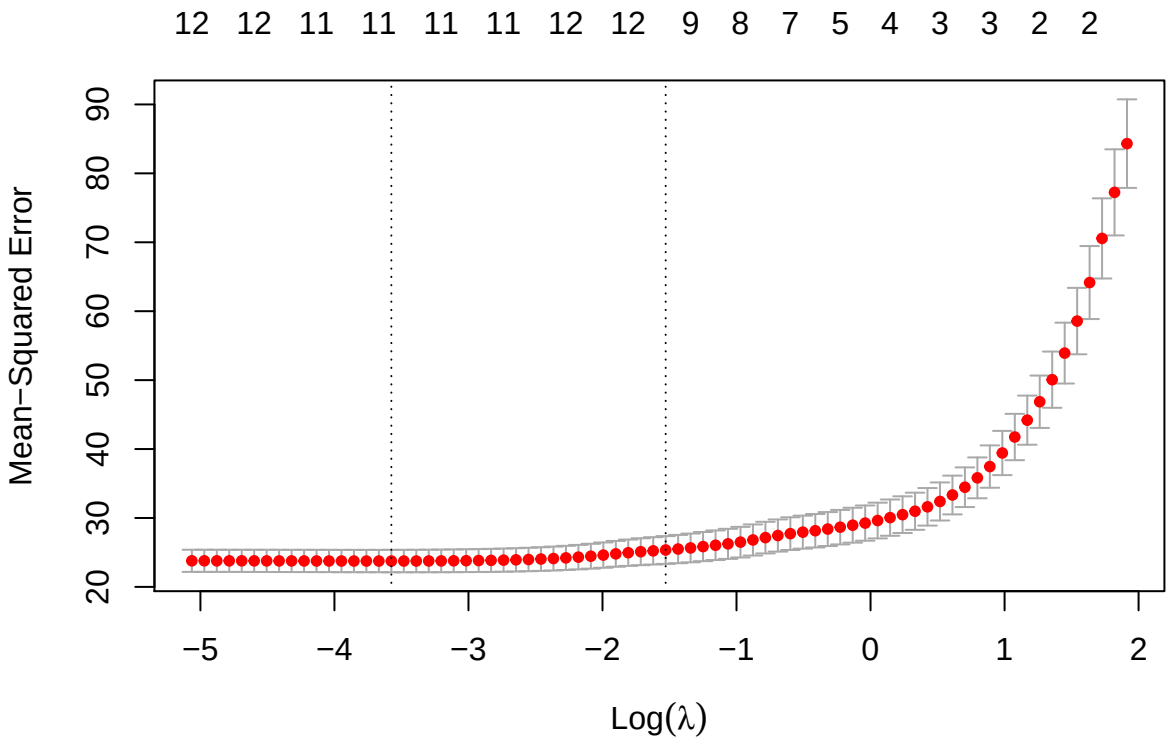
Use LASSO to fit a model to a different data set. Homeworks 3 and 4 both had data sets with multiple variables. The data set Boston (data(Boston)) has a bunch of variables with a response of median home value (medv). Or you could simulate data. See how it does with multicollinearity. Up to you.

```
library(MASS)
data(Boston)
y <- Boston$medv
x<-as.matrix(Boston[,c(1:13)])

#lasso
fit <- glmnet(x, y,alpha = 1)
plot(fit, label=TRUE)
```




```
#k fold cross validation
cv1=cv.glmnet(x,y,alpha=1)
plot(cv1)
```




```
fit.1se <- glmnet(x, y,alpha = 1, lambda=cv1$lambda.1se)

#final model
fit.1se$beta
```

```
## 13 x 1 sparse Matrix of class "dgCMatrix"
##                    s0
## crim    -3.740304e-02
## zn       1.426862e-02
## indus   -7.341294e-06
## chas     2.378364e+00
## nox     -8.847179e+00
## rm       4.230295e+00
## age      .
## dis     -7.700577e-01
## rad      .
## tax      .
## ptratio -8.189468e-01
## black    7.267155e-03
## lstat   -5.209792e-01
```

Lasso works with multicolinearity too!