Ridge and Lasso

L. Insolia, F. Chiaromonte (special thanks to J. Di Iorio)

May 10th 2022

Contents

Introduction

Libraries

We are going to use:

- **glmnet**: *Lasso and Elastic-Net Regularized Generalized Linear Models*
- **tidyverse**: *Easily Install and Load the 'Tidyverse'*
- **caret**: *Classification and Regression Training*
- **ggplot2**: *Create Elegant Data Visualisations Using the Grammar of Graphics*

```
library(glmnet)     # ridge and lasso for GLMs
library(tidyverse)  # data manipulation and visualization
library(caret)      # statistical learning techniques
library(ggplot2)    # plots
```

Data

We will use the **Body Fat dataset** (which is available in the Datasets folder of our course).

The data concerns a sample of 252 men, and contains 15 variables:

1. Density of the body, determined from underwater weighing
2. Percentage of body fat, calculated as a function of the Density according to Siri's equation: (495/Density) – 450.
3. Indicator for Age group (binary; 0: up to 45 years, 1: over 45)
4. Weight (lbs)
5. Height (inches)
6. Neck circumference (cm)

7. Chest circumference (cm)
8. Abdomen circumference (cm)
9. Hip circumference (cm)
10. Thigh circumference (cm)
11. Knee circumference (cm)
12. Ankle circumference (cm)
13. Biceps circumference (cm)
14. Forearm circumference (cm)
15. Wrist circumference (cm)

We want to understand if we can reliably describe and predict body fat percentage on the basis of these variables, using regression. For age, we only have a binary indicator separating men below and above 45 years. The body measurements, on the other hand, are all continuous variables. Please see the data description file for more details.

```
df <- read.table('BODY_FAT.TXT', header=TRUE)
names(df)
```

```
##  [1] "Density"  "SiriBF."  "Over45"   "Weight"   "Height"   "NeckC"
##  [7] "ChestC"   "AbdomenC" "HipC"     "ThighC"   "KneeC"    "AnkleC"
## [13] "BicepsC"  "ForearmC" "WristC"
```

We want to predict "SiriBF." using the other features, aside from "Density". So we drop the "Density" column.

```
df <- df[,-1]
```

Penalized regression

We will perform ridge/lasso penalization through the **glmnet** package. Let us identify predictors and response variable

```
# getting the predictors
x_var <- data.matrix(df[,-1])     # NOTE: glmnet requires a matrix structure
# getting the response variable
y_var <- df[,"SiriBF."]
```

Let's have a look a the glmnet function:

```
help(glmnet)
```

Note that:

- x: input matrix
- y: response variable
- α is the elastic-net mixing parameter with range [0, 1]. Namely. $\alpha = 1$ is the lasso (default) and $\alpha = 0$ is the ridge.
- standardize is a logical flag for x variable standardization, prior to fitting the model sequence. The coefficients are always returned on the original scale. Default is standardize=TRUE.

Ridge

To perform ridge regression, we run glmnet with $\alpha = 0$. The λ's sequence is internally computed by the package itself – although a user-defined sequence can be provided as a *lambda* argument.

```
ridge <- glmnet(x_var, y_var, alpha=0)
summary(ridge)
```

```
##           Length Class     Mode
## a0         100   -none-    numeric
## beta      1300   dgCMatrix S4
## df         100   -none-    numeric
## dim          2   -none-    numeric
## lambda     100   -none-    numeric
## dev.ratio  100   -none-    numeric
## nulldev      1   -none-    numeric
## npasses      1   -none-    numeric
## jerr         1   -none-    numeric
## offset       1   -none-    logical
## call         4   -none-    call
## nobs         1   -none-    numeric
```

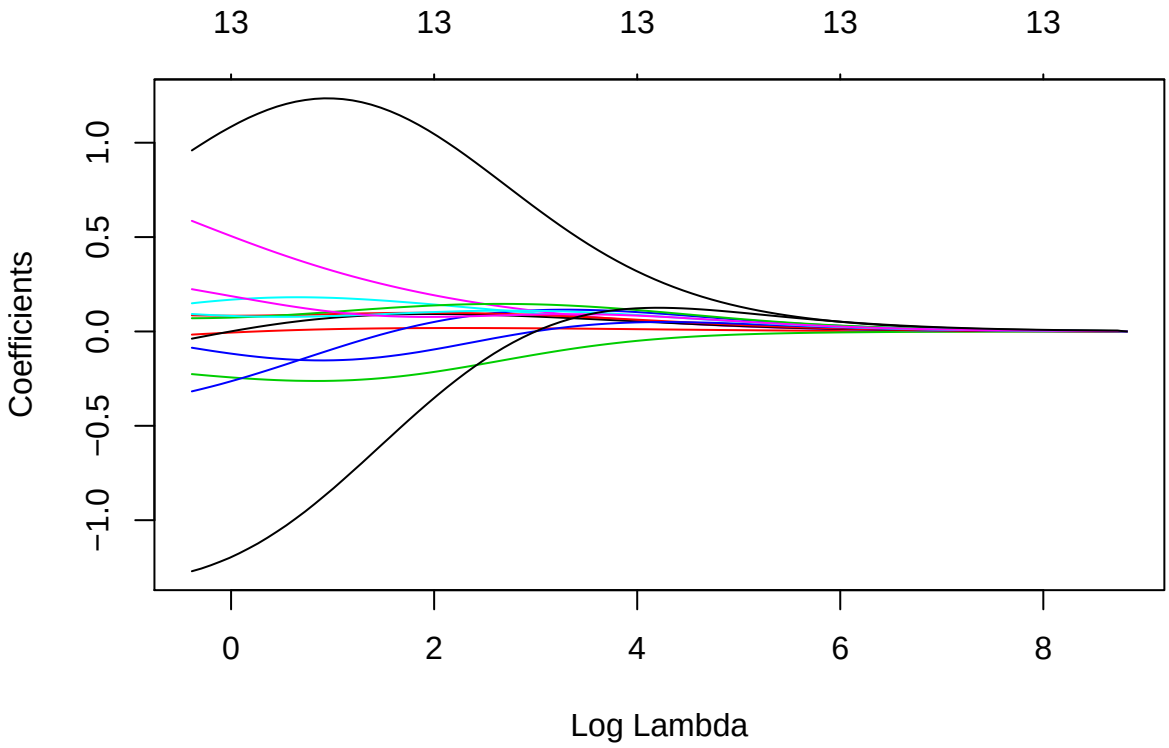
The summary is quite different than the one for linear regression, since ridge regression requires the tuning of λ. The code above fits a ridge regression for each λ value, and we have access to each of these model estimates.

We can plot the panalization path as follows:

```
dim(ridge$beta)
```

```
## [1]  13 100
```

```
plot(ridge, xvar="lambda")
```



We can automate the task of finding the optimal lambda value using the **cv.glmnet** function. This performs a k-fold cross-validation for glmnet, produces a plot, and returns "optimal" λ values.

```
cv_ridge <- cv.glmnet(x_var, y_var, alpha = 0)
cv_ridge
```

```
##
## Call:  cv.glmnet(x = x_var, y = y_var, alpha = 0)
##
## Measure: Mean-Squared Error
##
##     Lambda Measure    SE Nonzero
## min 0.6794   23.40 3.197      13
## 1se 2.0748   26.45 3.492      13
```

Two particular values of λ are highlighted: the minimum (min) and the largest value of lambda such that error is within 1 standard error of the minimum (1se).

```
cv_ridge$lambda.min
```

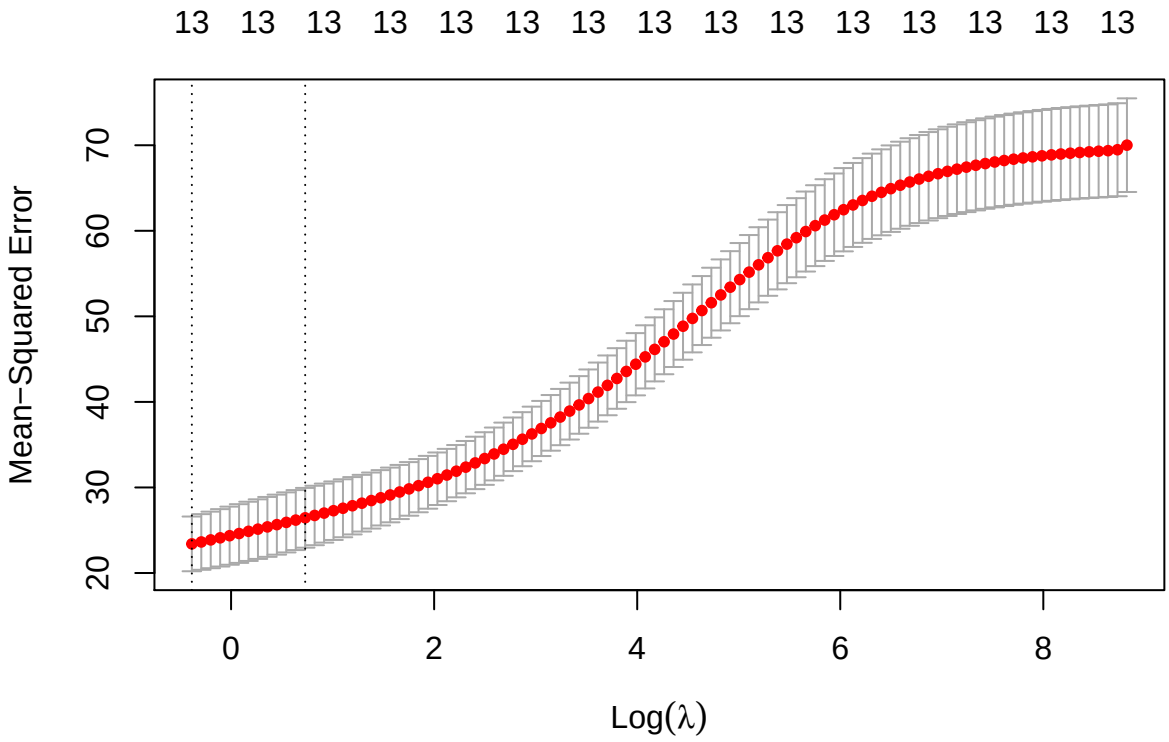
```
## [1] 0.6793884
```

```
cv_ridge$lambda.1se
```

```
## [1] 2.074754
```

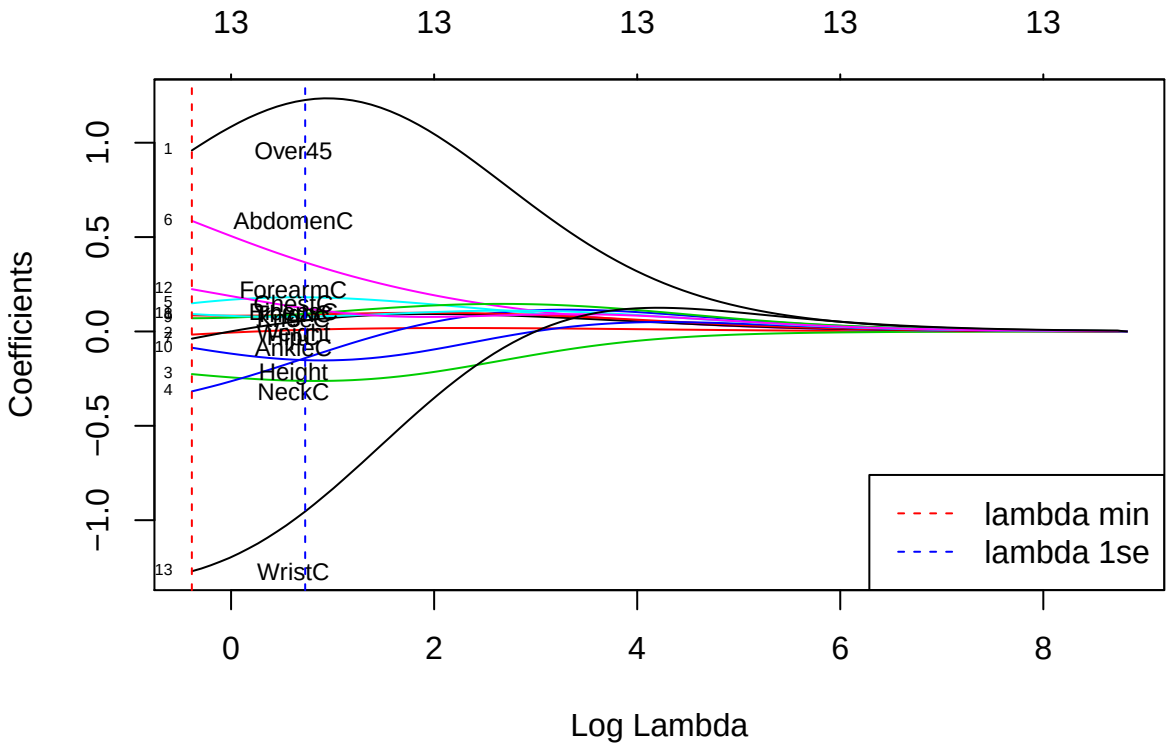
We can visualize them in this way:

```
plot(cv_ridge)
```



Let us see again how the regression coefficients change by modifying λ, highlighting the min and 1se values:

```
lbs_fun <- function(fit, offset_x=1, ...) {
  L <- length(fit$lambda)
  x <- log(fit$lambda[L])+ offset_x
  y <- fit$beta[, L]
  labs <- names(y)
  text(x, y, labels=labs, cex=0.75, ...)
}
plot(ridge, xvar = "lambda", label=T)
lbs_fun(ridge)
abline(v=log(cv_ridge$lambda.min), col = "red", lty=2)
abline(v=log(cv_ridge$lambda.1se), col="blue", lty=2)
legend(x = "bottomright",
       legend = c("lambda min", "lambda 1se"),
       lty = c(2, 2),
       col = c("red", "blue"))
```



Let's re-fit the model and see the estimates associated to the minimum λ.

```
min_ridge <- glmnet(x_var, y_var, alpha=0, lambda= cv_ridge$lambda.min)
coef(min_ridge)
```

```
## 14 x 1 sparse Matrix of class "dgCMatrix"
##                       s0
## (Intercept) -7.71880407
## Over45       0.95953292
## Weight      -0.01619205
## Height      -0.22630209
## NeckC       -0.31617445
## ChestC       0.14889642
## AbdomenC     0.58565111
## HipC        -0.03875702
## ThighC       0.08495378
## KneeC        0.06923480
## AnkleC      -0.08687724
## BicepsC      0.09165335
## ForearmC     0.22347815
## WristC      -1.27272327
```

We can use this model to make predictions on the training set.

```
# Make predictions on the training data
predictions <- min_ridge %>% predict(x_var) %>% as.vector()
# Model performance metrics
data.frame(
```

```
  RMSE = RMSE(predictions, y_var),
  Rsquare = R2(predictions, y_var)
)
```

```
##       RMSE  Rsquare
## 1 4.494532 0.714938
```

Be careful though! We are making predictions and assessing the goodness of fit based on training data. Is it the best choice? Do you have any other suggestions?

Lasso

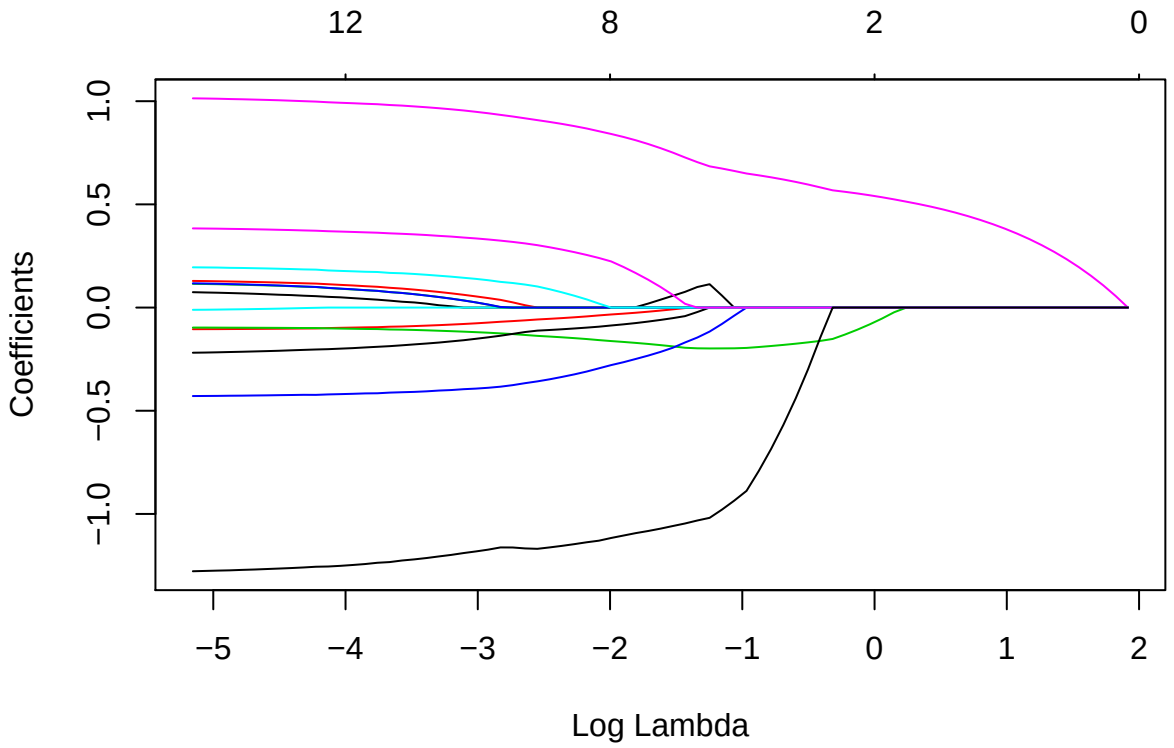
Let us now perform Lasso regression using the **glmnet** package. We follow the same approach as in Ridge regression, but set $\alpha = 1$.

```
lasso <- glmnet(x_var, y_var, alpha=1)
summary(lasso)
```

```
##           Length Class     Mode
## a0          77   -none-    numeric
## beta      1001   dgCMatrix S4
## df          77   -none-    numeric
## dim          2   -none-    numeric
## lambda      77   -none-    numeric
## dev.ratio   77   -none-    numeric
## nulldev      1   -none-    numeric
## npasses      1   -none-    numeric
## jerr         1   -none-    numeric
## offset       1   -none-    logical
## call         4   -none-    call
## nobs         1   -none-    numeric
```

Let's have a look at the selection path:

```
plot(lasso, xvar="lambda")
```

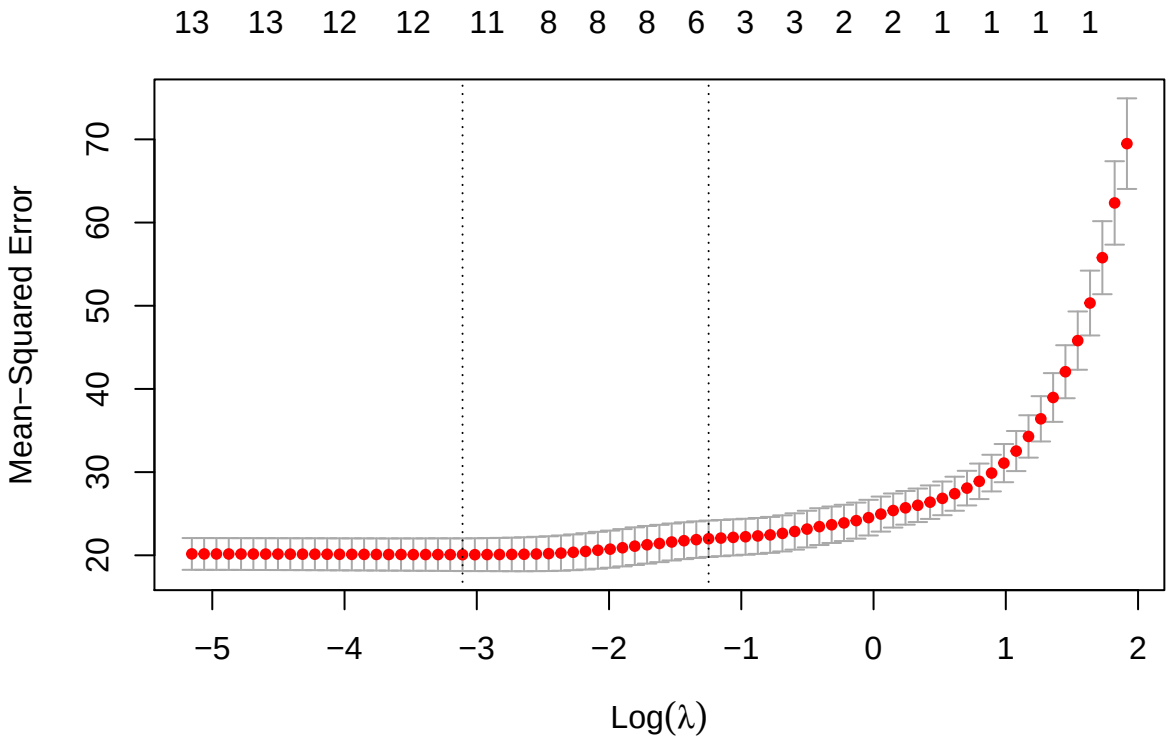


Once again, we need to tune the sparsity parameter λ. We use k-fold cross-validation through the **cv.glmnet** function.

```
cv_lasso <- cv.glmnet(x_var, y_var, alpha = 1)
cv_lasso
```

```
##
## Call:  cv.glmnet(x = x_var, y = y_var, alpha = 1)
##
## Measure: Mean-Squared Error
##
##     Lambda Measure    SE Nonzero
## min 0.0447   20.07 1.966      11
## 1se 0.2873   21.99 2.185       5
```

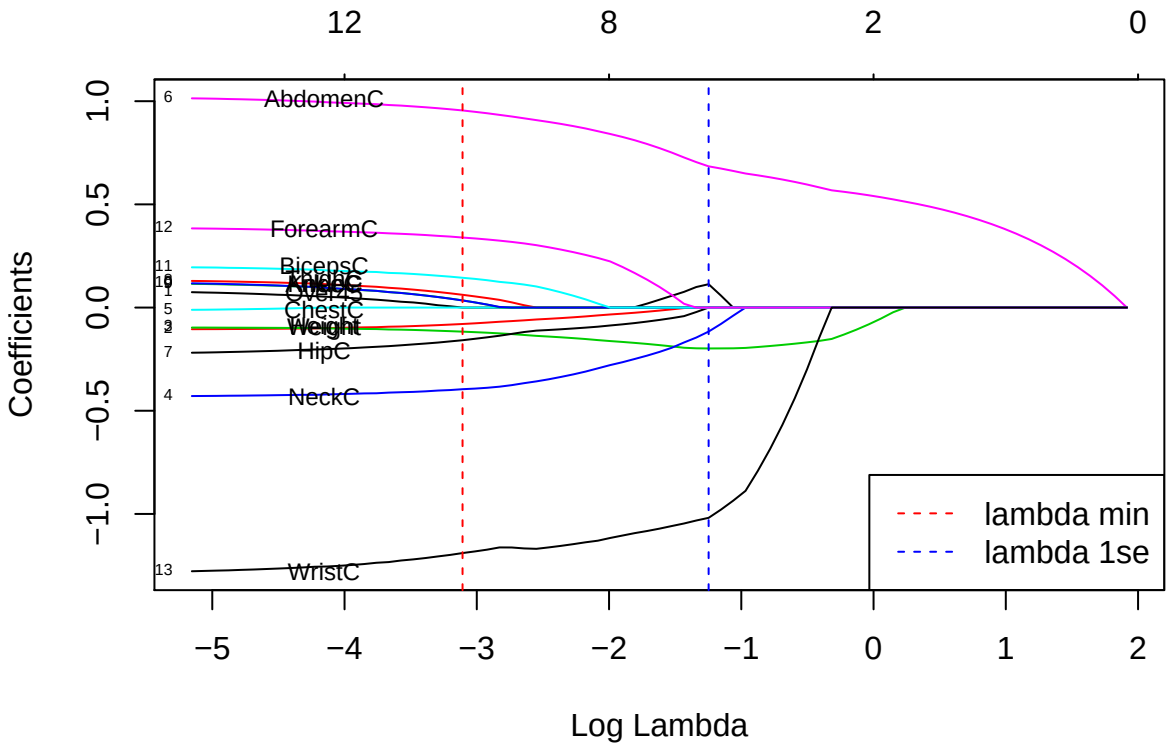
```
plot(cv_lasso)
```



Also here, it outputs the min and the 1se λ. As expected, the number of non-zero coefficients (which is printed on top of the previous plot) is lower than the one for Ridge regression.

Let us see again how the regression coefficients change by modifying λ:

```
lbs_fun <- function(fit, offset_x=1, ...) {
  L <- length(fit$lambda)
  x <- log(fit$lambda[L])+ offset_x
  y <- fit$beta[, L]
  labs <- names(y)
  text(x, y, labels=labs, cex=0.75, ...)
}
plot(lasso, xvar = "lambda", label=T)
lbs_fun(lasso)
abline(v=log(cv_lasso$lambda.min), col = "red", lty=2)
abline(v=log(cv_lasso$lambda.1se), col="blue", lty=2)
legend(x = "bottomright",
       legend = c("lambda min", "lambda 1se"),
       lty = c(2, 2),
       col = c("red", "blue"))
```



Let us rebuilt the model and compare the estimated coefficients for min and 1se λ.

```
min_lasso <- glmnet(x_var, y_var, alpha=1, lambda= cv_lasso$lambda.min)
se_lasso <- glmnet(x_var, y_var, alpha=1, lambda= cv_lasso$lambda.1se)

lasso_mat <- cbind(coef(min_lasso), coef(se_lasso))
colnames(lasso_mat) <- c("min", "1se")
lasso_mat
```

```
## 14 x 2 sparse Matrix of class "dgCMatrix"
##                       min        1se
## (Intercept) -14.40453587 -7.3351790
## Over45        .           0.1121805
## Weight       -0.07939885  .
## Height       -0.11654360 -0.1982956
## NeckC        -0.39602242 -0.1176232
## ChestC        .           .
## AbdomenC      0.95582980  0.6842130
## HipC         -0.16050576  .
## ThighC        0.06428259  .
## KneeC         0.03321073  .
## AnkleC        0.03394848  .
## BicepsC       0.14198097  .
## ForearmC      0.33930802  .
## WristC       -1.18720126 -1.0155818
```

We can use this model to make predictions on the training set.

```r
# Make predictions on the training data
predictions <- se_lasso %>% predict(x_var) %>% as.vector()
# Model performance metrics
data.frame(
  RMSE = RMSE(predictions, y_var),
  Rsquare = R2(predictions, y_var)
)
```

```
##       RMSE   Rsquare
## 1 4.508947 0.7109238
```