I’m writing a series of posts on various function options of the glmnet function (from the package of the same name), hoping to give more detail and insight beyond R’s documentation.

In this post, we will focus on the **standardize** option.

For reference, here is the full signature of the glmnet function:

glmnet(x, y, family=c("gaussian","binomial","poisson","multinomial","cox","mgaussian"),

weights, offset=NULL, alpha = 1, nlambda = 100,

lambda.min.ratio = ifelse(nobs

Unless otherwise stated, n will denote the number of observations, p will denote the number of features, and fit will denote the output/result of the glmnet call. The data matrix is denoted by X \in \mathbb{R}^{n \times p} and the response is denoted by y \in \mathbb{R}^n.

**standardize**

When standardize = TRUE (default), columns of the data matrix x are standardized, i.e. each column of x has mean 0 and standard deviation 1. More specifically, we have that for each j = 1, \dots, p,

\displaystyle\sum_{i=1}^n X_{ij} = 0, and \sqrt{\displaystyle\sum_{i=1}^n \frac{X_{ij}^2}{n}} = 1.

Why might we want to do this? Standardizing our features before model fitting is common practice in statistical learning. This is because if our features are on vastly different scales, the features with larger scales will tend to dominate the action. (One instance where we might not want to standardize our features is if they are already all measured along the same scale, e.g. meters or kilograms.)

Notice that the standardization here is slightly different from that offered by the scale function: scale(x, center = TRUE, scale = TRUE) gives the standardization

\displaystyle\sum_{i=1}^n X_{ij} = 0, and \sqrt{\displaystyle\sum_{i=1}^n \frac{X_{ij}^2}{n-1}} = 1.

We verify this with a small data example. Generate data according to the following code:

n <- 100; p <- 5; true\_p <- 2

set.seed(950)

X <- matrix(rnorm(n \* p), nrow = n)

beta <- matrix(c(rep(1, true\_p), rep(0, p - true\_p)), ncol = 1)

y <- X %\*% beta + 3 \* rnorm(n)

Create a version of the data matrix which has standardized columns:

X\_centered <- apply(X, 2, function(x) x - mean(x))

Xs <- apply(X\_centered, 2, function(x) x / sqrt(sum(x^2) / n))

Next, we run glmnet on Xs and y with both possible options for standardize:

library(glmnet)

fit <- glmnet(Xs, y, standardize = TRUE)

fit2 <- glmnet(Xs, y, standardize = FALSE)

We can check that we get the same fit in both cases (modulo numerical precision):

sum(fit$lambda != fit2$lambda)

# 0

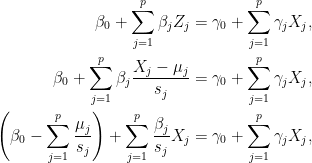
max(abs(fit$beta - fit2$beta))

# 6.661338e-16

The documentation notes that the coefficients returned are on the original scale. Let’s confirm that with our small data set. Run glmnet with the original data matrix and standardize = TRUE:

fit3 <- glmnet(X, y, standardize = TRUE)

For each column j, our standardized variables are Z_j = \dfrac{X_j - \mu_j}{s_j}, where \mu_j and s_j are the mean and standard deviation of column j respectively. If \beta_j and \gamma_j represent the model coefficients of fit2 and fit3 respectively, then we should have



i.e. we should have \gamma_0 = \beta_0 - \sum_{j=1}^p \frac{\mu_j}{s_j} and \gamma_j = \frac{\beta_j}{s_j} for j = 1, \dots, p. The code below checks that this is indeed the case (modulo numerical precision):

# get column means and SDs

X\_mean <- colMeans(X)

X\_sd <- apply(X\_centered, 2, function(x) sqrt(sum(x^2) / n))

# check difference for intercepts

fit2\_int <- coefficients(fit2)[1,]

fit3\_int <- coefficients(fit3)[1,]

temp <- fit2\_int - colSums(diag(X\_mean / X\_sd) %\*% fit2$beta)

max(abs(temp - fit3\_int))

# 1.110223e-16

# check difference for feature coefficients

temp <- diag(1 / X\_sd) %\*% fit2$beta

max(abs(temp - fit3$beta))

# 1.110223e-15

The discussion above has been for the standardization of x. What about standardization for y? The documentation notes that when family = "gaussian", y is automatically standardized, and the coefficients are unstandardized at the end of the procedure.

More concretely, let the mean and standard deviation of y be denoted by \mu_y and s_y respectively. If running glmnet on standardized y gives intercept \beta_0 and coefficients \beta_1, \dots, \beta_p, then glmnet on unstandardized y will give intercept \mu_y + s_y\beta_0 and coefficients s_y\beta_1, \dots, s_y\beta_p.

Again, this can be verified empirically:

# get mean and SD of y

y\_mean <- mean(y)

y\_sd <- sqrt(sum((y - y\_mean)^2) / n)

# fit model with standardized y

fit4 <- glmnet(X, (y - y\_mean) / y\_sd, standardize = TRUE)

# check difference for intercepts

fit4\_int <- coefficients(fit4)[1,]

temp <- fit4\_int \* y\_sd + y\_mean

max(abs(temp - fit3\_int))

# 1.110223e-16

# check difference for feature coefficients

max(abs(y\_sd \* fit4$beta - fit3$beta))

# 8.881784e-16