Exercise report on model selection

Mélanie FOURNIER

2024-10-07

Model selection

In the following lab, we look at the Fathead Minnow Acute Aquatic Toxicity dataset, compiled and described by He and Jurs (2005).

The label variable is the "activity" variable, which varies from values between 0.04 and 6.72. There are 229 samples in the training dataset. All the 49 predictor variables are numerical. A linear regression model seems well suited for the prediction task. There is, however, quite a large amount of variables, so a model selection process must be used in order to avoid overfitting.

We choose the Lasso regression as a method of shrinkage. I already worked with forward and backward selection in the Linear Regression course, but not the shrinkage method, hence the choice.

One problem of having many covariates is that we run into the risk of some of them being correlated to each other, leading to large positive and negative regression coefficients even if the system is not itself underdetermined. This results in overfitting.

The shrinkage methods works by adding a penalty term in the minimization problem which scales with the coefficients. Lasso regression penalty scales with the sum of the absolute value of the coefficients. Other methods can have quadratic scaling in their penalty.

The main defining characteristic of the Lasso regression is its faculty to force some coefficients to be zero, effectively removing the associated variables from the model.

The minimization problem for the Lasso is then the following problem:

$$\text{argmin}_{\beta}(\text{RSS} + \lambda \sum_{i} |\beta_i|)$$

The λ term is then an hyperparameter controlling the strength of the penalization. A value of λ set too low will simply not shrink the model enough to combat overfitting, but setting it too high risk losing out on the data too much.

In order to set this parameter, we can derive a simple method. Whichever values of λ leads to the best performance using K-fold cross validation will be the one chosen. (This is essentially every what every tutorial on lasso regression does, but if it works, it works.)

We first load the library and the data.

```
library(dplyr)
library(ggplot2)
library(glmnet)
```

```
data_x <- read.csv("AquaticTox_train_x.csv")
data_y <- read.csv("AquaticTox_train_y.csv")
```

```
#Remove the molecule variable, as it is not a predictor
X <- as.matrix(data_x[,-1])
y <- data_y[,-1]
```

R is a very nice statistical software in the sense that we barely have to do anything to set up the model with the glmnet packages, which handles the lasso regression and the hyperparameter tuning via k-fold cross validation.

All that is left is to make sure the parameters passed to the function are the one we want.

- alpha regulates the kind of penalty to use, set to 1, this is a Lasso regression.
- The function can calculate automatically appropriates values for lambda depending on the data. For more control, we can also pass values of lambda explicitely.
- We can standardize or not the variables before fitting the model. It makes sense to do so to make sure every variable is equally as important.
- The number of folds to use can also be used, a value of 20 seems reasonable, as this leaves about 10% of the data for validation for each fold. It should be noted that this is another hyperparameter, and changing this also changes the best value of lambda.

```
#alpha = 1 for Lasso
#standardize = TRUE by default, not changing

set.seed(1235812) #Fibonacci, for luck
fit <- cv.glmnet(
    as.matrix(X),
    y,
```

```
    alpha = 1,
    nlambda = 150,
    nfolds = 20
    )
```

The MSE is plotted for different values of λ in the figure below. There is quite a plateau for values of $\log(\lambda)$ between -5 and -2 for which the MSE does not move that much, making the choice of any value of lambda in this region a good choice. If smaller models are prefered for some reason, higher values of λ could be chosen.

```
plot(fit)
```

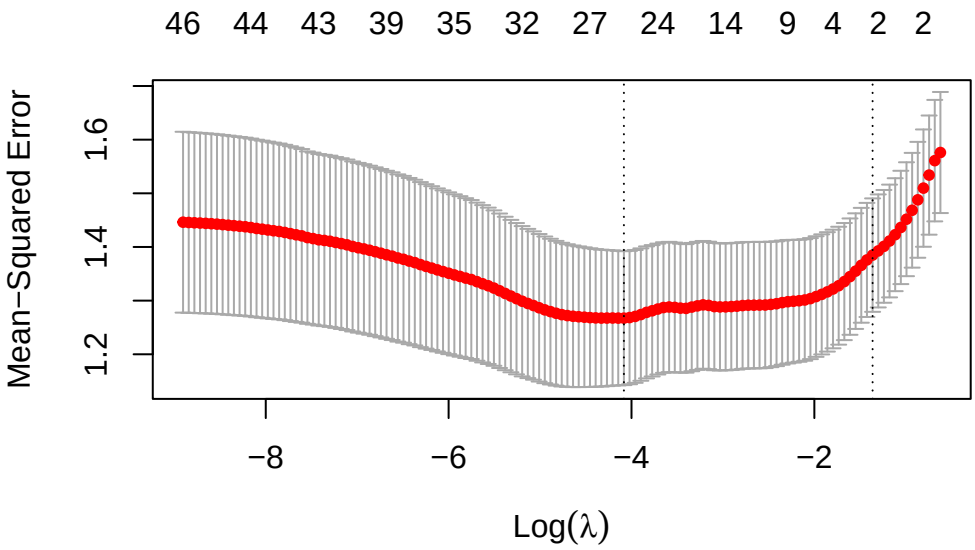

Figure 1: Cross validated mean square error for different values of lambda in the Lasso regression. The values on top are the number of coefficients that are not zero.

Plotting the logarithm of λ against the number of coefficient put to zero shows the almost linear growth between the two. With a sufficiently high value of lambda, almost all coefficients are set to zero.

```
plot(log(rev(fit$lambda)),fit$nzero)
```

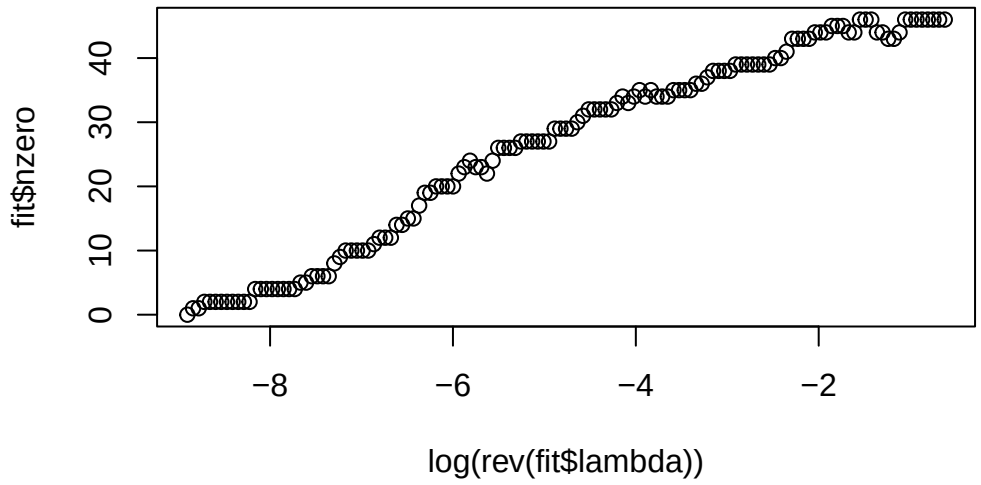

Figure 2: Number of zero coefficient as a function of the logarithm of lambda.

The value of lambda that minimize the MSE is automatically calculated.

```
lambda_best <- fit$lambda.min
lambda_best
```

```
[1] 0.01685814
```

This can be used to define the final model:

```
fit_final <- glmnet(X,y,alpha = 1,lambda = lambda_best)
```

```
sum(coef(fit_final) != 0)
```

```
[1] 26
```

The final model has 25 coefficients that are not zero + intercept, which is a shrinkage of half the available variables.

As a final check, we do a quick residual analysis by plotting the normal QQ-plot for the residual, and the residual turns out to be well behaved.

```
pred <- predict(fit_final,X)
resid <- pred-y
qqnorm(resid)
qqline(resid)
```

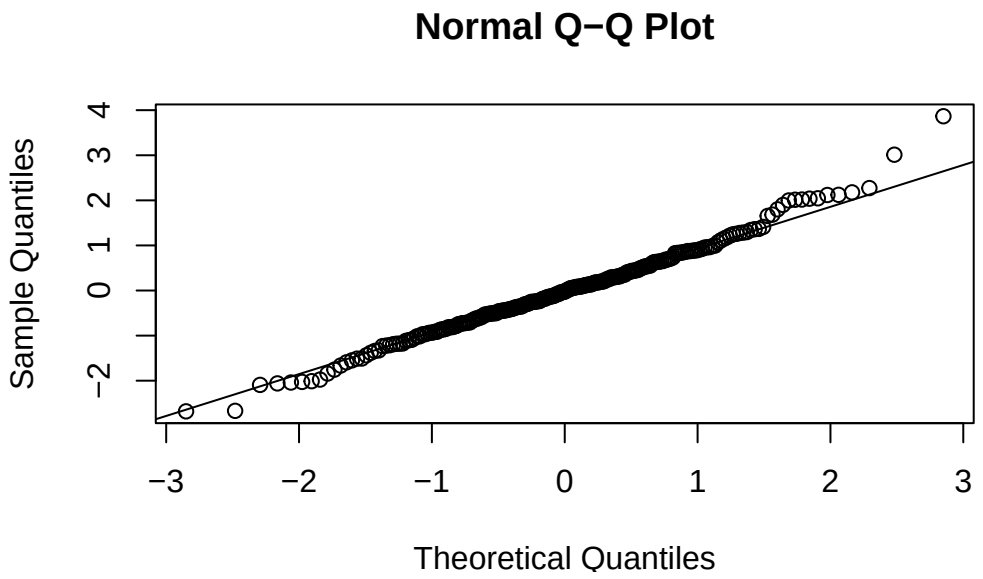

Figure 3: Residual qq plot for the final model. No problem to be seen.

Other performance metrics can then be calculated, including mse, AIC and adjusted R-squared.

```
mse <- function(fit, X,y){
    pred <- predict(fit,X)
    err_sq <- (pred - y)^2
    mse <- mean(err_sq)
    return(mse)
}
cat("MSE, training data:", mse(fit_final,X,y))
```

```
MSE, training data: 1.004066
```

```
#https://stackoverflow.com/questions/40920051/r-getting-aic-bic-likelihood-from-glmnet

aic <- function(fit){
    tLL <- fit$nulldev - deviance(fit)
    k <- fit$df
    n <- fit$nobs
    AICc <- -tLL+2*k+2*k*(k+1)/(n-k-1)
    return(AICc)
}
cat("AIC, training data:",aic(fit_final))
```

```
AIC, training data: -72.11586
```

```r
adj_rsq <- function(fit,X,y){
    p <- sum(coef(fit) != 0)
    n <- length(X[,1])
    df.res <- n-p-1
    df.tot <- n-1
    pred <- predict(fit,X)
    err.sq <- (pred - y)^2
    SS.res <- sum(err.sq)
    SS.tot <- sum((y - mean(y))^2)
    adj.R2 <- 1 - (SS.res/df.res)/(SS.tot/df.tot)
    return(adj.R2)
}
cat("Adjusted R², training data:", adj_rsq(fit_final,X,y))
```

```
Adjusted R², training data: 0.2759784
```

Performance on test data.

Now that the model is selected, we can check the performance on the test data.

```r
data_x_test <- read.csv("AquaticTox_test_x.csv")
data_y_test <- read.csv("AquaticTox_test_y.csv")
```

```r
#Remove the molecule variable, as it is not a predictor
X_test<- as.matrix(data_x_test[,-1])
y_test <- data_y_test[,-1]
```

```r
cat("MSE, test data:", mse(fit_final,X_test,y_test))
```

```
MSE, test data: 1.646245
```

```r
cat("Adjusted R², test data:", adj_rsq(fit_final,X_test,y_test))
```

```
Adjusted R², test data: -0.2163912
```

The MSE is very high and the adjusted R-squared is appaling. It turns out that the chosen model did not work out with the test data.

Plotting prediction against actual value explains the problem here: the model tends to always predict values between 3 and 5 while the true values are quite uniformly distributed between 0 and 6.

```
pred <- predict(fit_final,X_test)

plot(y_test,pred, xlim = c(0,6),ylim = c(0,6))
segments(x0=0,y0=0,x1=6,y1=6)
```

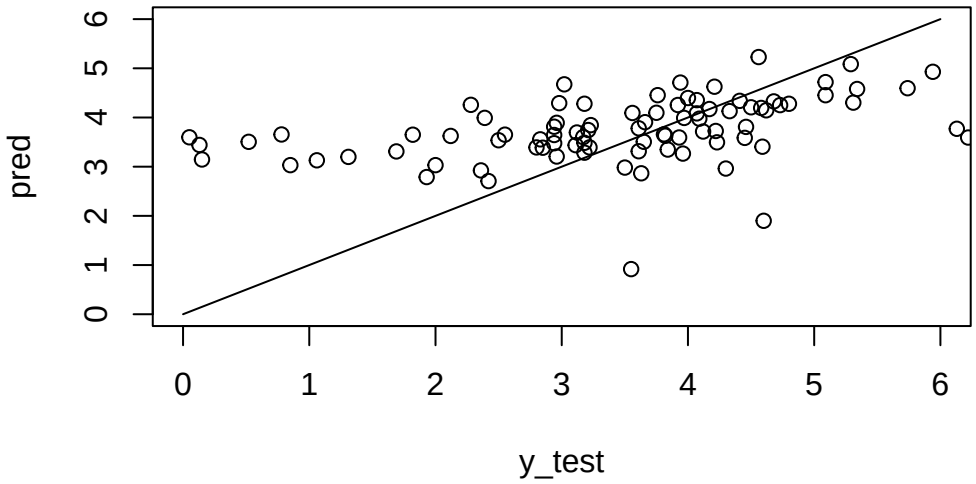

Figure 4: Prediction vs actual value in the test dataset.