# Practical session 3 - Modèles de régression régularisée

Piseth KHENG, Borachhun YOU

24 October 2022

## II. Application to diabetes medical data

Firstly, we load the data into a data frame. We then convert the values of `Y` into binary values (`YBin`) by using the median of `Y` as the threshold. The value of `YBin` equals:

- 0 if `Y` is smaller or equal to its median
- 1 if `Y` is bigger than its median.

```
tab <- read.table("diabetes.txt", header=TRUE, sep="\t")

# Add YBin to data frame
tab$YBin <- as.numeric(tab$Y > median(tab$Y))

# Drop Y from data frame
tab <- tab[, -11]

dim(tab)
```

```
## [1] 442  11
```

```
head(tab)
```

```
##   AGE SEX  BMI  BP  S1    S2 S3 S4     S5 S6 YBin
## 1  59   2 32.1 101 157  93.2 38  4 4.8598 87    1
## 2  48   1 21.6  87 183 103.2 70  3 3.8918 69    0
## 3  72   2 30.5  93 156  93.6 41  4 4.6728 85    1
## 4  24   1 25.3  84 198 131.4 40  5 4.8903 89    1
## 5  50   1 23.0 101 192 125.4 52  4 4.2905 80    0
## 6  23   1 22.6  89 139  64.8 61  2 4.1897 68    0
```

Now, we randomly split the data into 2 sets: a training data set (80% of the data) to calibrate the models, and a testing data set (the remaining 20% of the data) to evaluate the models after the calibration.

```
test_index <- sample(1:442, size=round(442/5))

train_data <- tab[-test_index, ]
test_data <- tab[test_index, ]
```

Given observations and predictions, the function below is used to compute the confusion matrix of a model along with the value of the accuracy, the precision, the recall and the specificity of the model.

```
performance <- function(Y, Y_predict) {
  confusion_matrix <- table(Prediction=Y_predict, Observation=Y)

  cat("Confusion matrix:\n\n")
  print(confusion_matrix)
```

```
  confusion_matrix["0","0"] -> TN
  confusion_matrix["0","1"] -> FN
  confusion_matrix["1","0"] -> FP
  confusion_matrix["1","1"] -> TP

  accuracy_ <- (TP+TN)/(TP+TN+FP+FN)
  precision_ <- TP/(TP+FP)
  recall_ <- TP/(TP+FN)
  specificity_ <- TN/(TN+FP)

  cat("\nAccuracy:", accuracy_, "\n")
  cat("Precision:", precision_, "\n")
  cat("Recall:", recall_, "\n")
  cat("Specificity:", specificity_, "\n")

  return(list(accuracy=accuracy_, precision=precision_,
              recall=recall_, specificity=specificity_))
}
```

## 1. Logistic regression

### With a training data set and a testing data set

Here, we are using the 80%-20% split data above to train and test the logistic model.

```
res_logit <- glm(YBin~., family=binomial, data=train_data)
summary(res_logit)
```

```
##
## Call:
## glm(formula = YBin ~ ., family = binomial, data = train_data)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.3163  -0.7648  -0.2470   0.7965   2.2795
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -12.860093   3.951590  -3.254 0.001136 **
## AGE          -0.006953   0.011182  -0.622 0.534079
## SEX          -1.036893   0.314967  -3.292 0.000995 ***
## BMI           0.123483   0.037911   3.257 0.001125 **
## BP            0.041308   0.011482   3.598 0.000321 ***
## S1           -0.046609   0.033470  -1.393 0.163756
## S2            0.042608   0.032049   1.329 0.183697
## S3           -0.008942   0.044833  -0.199 0.841913
## S4           -0.105224   0.328844  -0.320 0.748983
## S5            2.568264   0.943027   2.723 0.006461 **
## S6            0.003362   0.014560   0.231 0.817377
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
```

```
##     Null deviance: 490.47  on 353  degrees of freedom
## Residual deviance: 349.31  on 343  degrees of freedom
## AIC: 371.31
##
## Number of Fisher Scoring iterations: 5
```

After the model is trained, we then use it for predictions. 0.5 is taken as the threshold for the binary predictions.

```
YBin_logit <- round(predict.glm(res_logit, newdata=test_data[, -11], type="response"))
```

The result below is the performance of the model.

```
perf_logit <- performance(test_data$YBin, YBin_logit)
```

```
## Confusion matrix:
##
##           Observation
## Prediction  0  1
##          0 31  8
##          1  8 41
##
## Accuracy: 0.8181818
## Precision: 0.8367347
## Recall: 0.8367347
## Specificity: 0.7948718
```

### K-fold cross-validation

Now, we want to use the whole data for both training and testing the model by performing k-fold cross-validation. We choose $k = 5$ and thus we divide the data set of 442 observations into 5 subsets: 4 data sets of 88 observations and 1 data set of 90 observations.

```
fold_index <- split(
  1:442,
  f = sample(rep(1:5, times=c(88, 88, 88, 88, 90)))
)

sum_accuracy <- 0
sum_precision <- 0
sum_recall <- 0
sum_specificity <- 0

for (i in 1:5) {
  cat("========= FOLD", i, "=========\n\n")

  test_fold_index <- fold_index[[i]]
  train_fold <- tab[-test_fold_index, ]
  test_fold <- tab[test_fold_index, ]

  res_fold <- glm(YBin~., family=binomial, data=train_fold)

  YBin_fold <- round(predict.glm(res_fold, newdata=test_fold[, -11], type="response"))

  perf_fold <- performance(test_fold$YBin, YBin_fold)
  cat("\n")
```

```
  sum_accuracy <- sum_accuracy + perf_fold$accuracy
  sum_precision <- sum_precision + perf_fold$precision
  sum_recall <- sum_recall + perf_fold$recall
  sum_specificity <- sum_specificity + perf_fold$specificity
}
```

```
## ========= FOLD 1 =========
##
## Confusion matrix:
##
##           Observation
## Prediction  0  1
##          0 35 12
##          1 13 28
##
## Accuracy: 0.7159091
## Precision: 0.6829268
## Recall: 0.7
## Specificity: 0.7291667
##
## ========= FOLD 2 =========
##
## Confusion matrix:
##
##           Observation
## Prediction  0  1
##          0 35  9
##          1 16 28
##
## Accuracy: 0.7159091
## Precision: 0.6363636
## Recall: 0.7567568
## Specificity: 0.6862745
##
## ========= FOLD 3 =========
##
## Confusion matrix:
##
##           Observation
## Prediction  0  1
##          0 34 11
##          1  7 36
##
## Accuracy: 0.7954545
## Precision: 0.8372093
## Recall: 0.7659574
## Specificity: 0.8292683
##
## ========= FOLD 4 =========
##
## Confusion matrix:
##
##           Observation
## Prediction  0  1
```

```
##           0 26 15
##           1 16 31
##
## Accuracy: 0.6477273
## Precision: 0.6595745
## Recall: 0.673913
## Specificity: 0.6190476
##
## ========= FOLD 5 =========
##
## Confusion matrix:
##
##           Observation
## Prediction  0  1
##           0 33 10
##           1  6 41
##
## Accuracy: 0.8222222
## Precision: 0.8723404
## Recall: 0.8039216
## Specificity: 0.8461538
```

```
cat("Average accuracy:", sum_accuracy/5, "\n")
```

```
## Average accuracy: 0.7394444
```

```
cat("Average precision:", sum_precision/5, "\n")
```

```
## Average precision: 0.7376829
```

```
cat("Average recall:", sum_recall/5, "\n")
```

```
## Average recall: 0.7401098
```

```
cat("Average specificity:", sum_specificity/5, "\n")
```

```
## Average specificity: 0.7419822
```

The results obtained above are values related to the performance of the method for each fold, as well as the average values of the 5 folds.

## 2. Logistic regression with model selection

Here, we are using the **forward selection** method. We start the process with no variable and we try adding a variable at each iteration. Also, we use the 80% training data set to train the model.

```
res0 <- glm(YBin~1, family=binomial, data=train_data)
res_forward <- step(res0, list(upper=res_logit), direction="forward")
```

```
## Start:  AIC=492.47
## YBin ~ 1
##
##        Df Deviance    AIC
## + BMI   1   414.18 418.18
## + S5    1   414.34 418.34
## + BP    1   438.86 442.86
## + S3    1   445.30 449.30
## + S4    1   446.68 450.68
```

```
## + S6    1   462.77 466.77
## + S1    1   482.85 486.85
## + S2    1   484.15 488.15
## + AGE   1   485.82 489.82
## <none>      490.47 492.47
## + SEX   1   490.46 494.46
##
## Step:  AIC=418.18
## YBin ~ BMI
##
##        Df Deviance    AIC
## + S5    1   381.94 387.94
## + BP    1   394.25 400.25
## + S3    1   398.16 404.16
## + S4    1   400.64 406.64
## + S6    1   408.91 414.91
## <none>      414.18 418.18
## + AGE   1   413.24 419.24
## + S1    1   413.51 419.51
## + SEX   1   413.82 419.82
## + S2    1   414.02 420.02
##
## Step:  AIC=387.94
## YBin ~ BMI + S5
##
##        Df Deviance    AIC
## + BP    1   372.10 380.10
## + S3    1   375.30 383.30
## + S1    1   377.35 385.35
## + SEX   1   379.54 387.54
## <none>      381.94 387.94
## + S2    1   380.88 388.88
## + S4    1   381.20 389.20
## + S6    1   381.63 389.63
## + AGE   1   381.90 389.90
##
## Step:  AIC=380.1
## YBin ~ BMI + S5 + BP
##
##        Df Deviance    AIC
## + S3    1   365.24 375.24
## + S1    1   366.51 376.51
## + SEX   1   367.79 377.79
## <none>      372.10 380.10
## + S2    1   370.50 380.50
## + AGE   1   370.88 380.88
## + S4    1   371.50 381.50
## + S6    1   372.09 382.09
##
## Step:  AIC=375.24
## YBin ~ BMI + S5 + BP + S3
##
##        Df Deviance    AIC
## + SEX   1   353.02 365.02
```

```
## + S1    1   362.69 374.69
## + S4    1   362.95 374.95
## + S2    1   363.10 375.10
## <none>      365.24 375.24
## + AGE   1   364.17 376.17
## + S6    1   365.14 377.14
##
## Step:  AIC=365.02
## YBin ~ BMI + S5 + BP + S3 + SEX
##
##        Df Deviance    AIC
## <none>      353.02 365.02
## + S1    1   351.29 365.29
## + S4    1   351.45 365.45
## + S2    1   351.72 365.72
## + AGE   1   352.67 366.67
## + S6    1   353.02 367.02
```

```
summary(res_forward)
```

```
##
## Call:
## glm(formula = YBin ~ BMI + S5 + BP + S3 + SEX, family = binomial,
##     data = train_data)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.3965  -0.7849  -0.2711   0.7807   2.2020
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -8.70380    1.93813  -4.491  7.1e-06 ***
## BMI          0.12269    0.03656   3.356 0.000791 ***
## S5           1.21241    0.32187   3.767 0.000165 ***
## BP           0.03971    0.01092   3.637 0.000276 ***
## S3          -0.04924    0.01355  -3.633 0.000280 ***
## SEX         -1.04499    0.30941  -3.377 0.000732 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 490.47  on 353  degrees of freedom
## Residual deviance: 353.02  on 348  degrees of freedom
## AIC: 365.02
##
## Number of Fisher Scoring iterations: 4
```

We now use the trained model for predictions and we use the 20% testing data set as new data. We then compare the predictions and the observations in order to compute the performance of the model.

```
YBin_forward <- round(predict.glm(res_forward, newdata=test_data[, -11], type="response"))
perf_forward <- performance(test_data$YBin, YBin_forward)
```

```
## Confusion matrix:
##
```

```
##            Observation
## Prediction  0  1
##          0 32 10
##          1  7 39
##
## Accuracy: 0.8068182
## Precision: 0.8478261
## Recall: 0.7959184
## Specificity: 0.8205128
```

### 3. Logistic regression with $l_2$ penalization (Ridge regression)

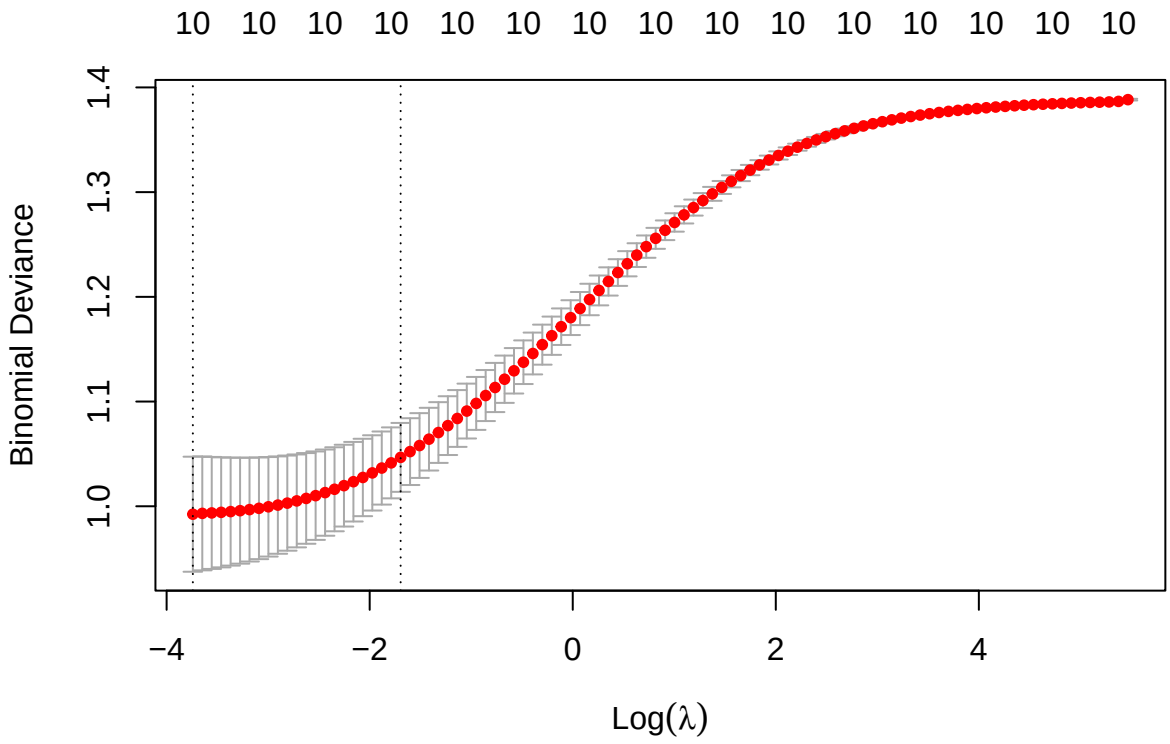
With Ridge regression, we start by finding the best value of $\lambda$, by performing k-fold cross-validation ($k = 10$ in this case) and find the value of $\lambda$ that minimizes the cross-validated error.

```
library(glmnet)
```

```
## Loading required package: Matrix
```

```
## Loaded glmnet 4.1-4
```

```
cv_ridge <- cv.glmnet(as.matrix(tab[,-11]), tab$YBin, family="binomial", alpha=0)
plot(cv_ridge)
```




```
best_ridge_lambda <- cv_ridge$lambda.min
cat("Best Ridge lambda:", best_ridge_lambda, "\n")
```

```
## Best Ridge lambda: 0.02371246
```

After obtaining the value of $\lambda$, we then use it to compute the Ridge regression.

```
res_ridge <- glmnet(as.matrix(train_data[,-11]), train_data$YBin, family="binomial",
                    alpha=0, lambda=best_ridge_lambda)
coef(res_ridge)
```

```
## 11 x 1 sparse Matrix of class "dgCMatrix"
##                        s0
## (Intercept) -8.761905377
## AGE         -0.003746300
## SEX         -0.761116004
## BMI          0.115938256
## BP           0.034503687
## S1          -0.003566499
## S2          -0.001817371
## S3          -0.032308182
## S4           0.078025504
## S5           1.131317121
## S6           0.006674394
```

Once again, we use the trained model to predict new data (testing data set), then compare the predictions and the observations to calculate the performance of the model.

```
YBin_ridge <- round(predict(res_ridge, as.matrix(test_data[, -11]), type="response"))
perf_ridge <- performance(test_data$YBin, YBin_ridge)
```

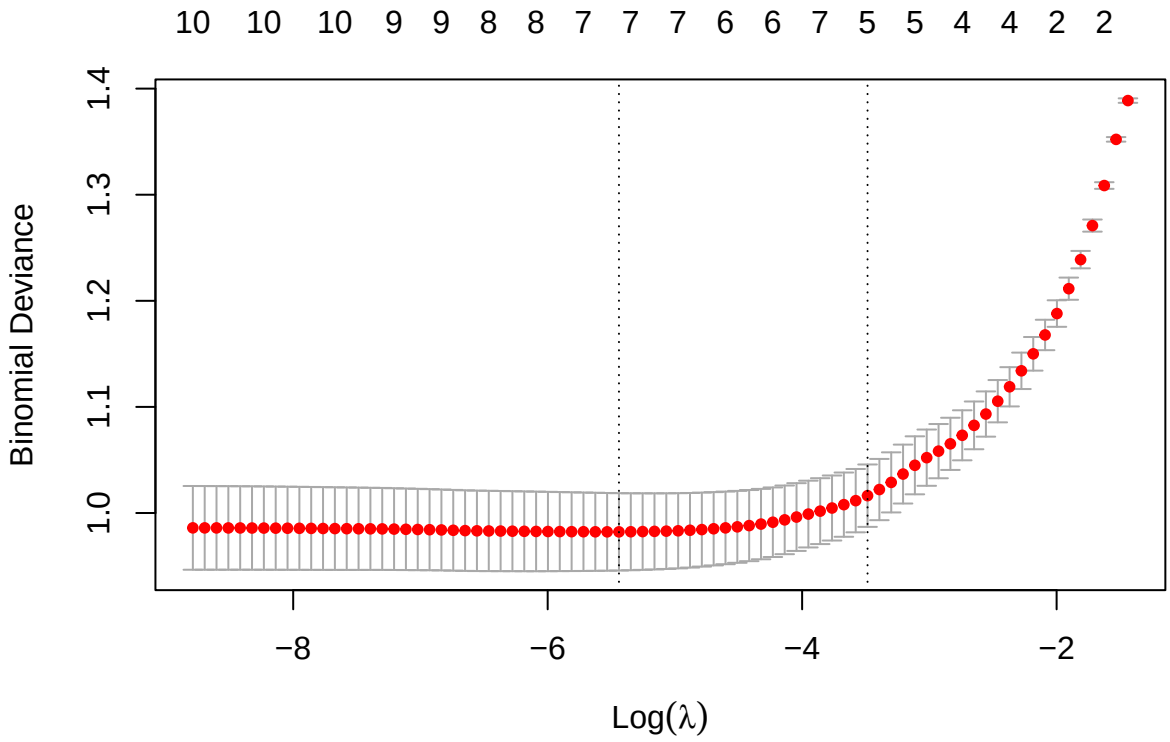
```
## Confusion matrix:
##
##           Observation
## Prediction  0  1
##          0 33  9
##          1  6 40
##
## Accuracy: 0.8295455
## Precision: 0.8695652
## Recall: 0.8163265
## Specificity: 0.8461538
```

## 4. Logistic regression with $l_1$ penalization (Lasso regression)

Similar to Ridge regression, with Lasso regression, we begin by selecting the best value of $\lambda$ by performing k-fold cross-validation.

```
library(glmnet)

cv_lasso <- cv.glmnet(as.matrix(tab[,-11]), tab$YBin, family="binomial", alpha=1)
plot(cv_lasso)
```




```
best_lasso_lambda <- cv_lasso$lambda.min
cat("Best Lasso lambda:", best_lasso_lambda, "\n")
```

```
## Best Lasso lambda: 0.004341131
```

We now compute the Lasso regression with the obtained value of $\lambda$.

```
res_lasso <- glmnet(as.matrix(train_data[,-11]), train_data$YBin, family="binomial",
                    alpha=1, lambda=best_lasso_lambda)
coef(res_lasso)
```

```
## 11 x 1 sparse Matrix of class "dgCMatrix"
##                        s0
## (Intercept) -8.998920723
## AGE         -0.002458228
## SEX         -0.889712104
## BMI          0.124169835
## BP           0.038034065
## S1          -0.004426680
## S2           .
## S3          -0.040558439
## S4           .
## S5           1.347831299
## S6           0.001029583
```

Now, we use the trained model for predictions and compute the performance of the model.

```
YBin_lasso <- round(predict(res_lasso, as.matrix(test_data[, -11]), type="response"))
perf_lasso <- performance(test_data$YBin, YBin_lasso)
```

```
## Confusion matrix:
##
##            Observation
## Prediction  0  1
##          0 32  9
##          1  7 40
##
## Accuracy: 0.8181818
## Precision: 0.8510638
## Recall: 0.8163265
## Specificity: 0.8205128
```

## 5. Comparison of the model performances

Below are the sums of squared of errors of the 4 models from above.

```
error_logit <- sum((test_data$YBin - YBin_logit)^2)
error_forward <- sum((test_data$YBin - YBin_forward)^2)
error_ridge <- sum((test_data$YBin - YBin_ridge)^2)
error_lasso <- sum((test_data$YBin - YBin_lasso)^2)

cat("Sum of squared of errors:\n",
    "- Logit:", error_logit, "\n",
    "- Forward:", error_forward, "\n",
    "- Ridge:", error_ridge, "\n",
    "- Lasso:", error_lasso, "\n"
)
```

```
## Sum of squared of errors:
##  - Logit: 16
##  - Forward: 17
##  - Ridge: 15
##  - Lasso: 16
```

We can see that the model of Ridge regression makes the least errors. We can also say that with a data set of 85 observations (the 20% training data set), the model of full logistic regression, forward method, Ridge regression and Lasso regression make 16, 17, 15 and 16 mistakes, respectively.

We now compare the performances of the 4 models.

```
data.frame(
  Model=c("Logit", "Forward", "Ridge", "Lasso"),
  Accuracy=c(perf_logit$accuracy, perf_forward$accuracy,
             perf_ridge$accuracy, perf_lasso$accuracy),
  Precision=c(perf_logit$precision, perf_forward$precision,
              perf_ridge$precision, perf_lasso$precision),
  Recall=c(perf_logit$recall, perf_forward$recall,
           perf_ridge$recall, perf_lasso$recall),
  Specificity=c(perf_logit$specificity, perf_forward$specificity,
                perf_ridge$specificity, perf_lasso$specificity)
)
```

```
##     Model  Accuracy Precision    Recall Specificity
## 1   Logit 0.8181818 0.8367347 0.8367347   0.7948718
## 2 Forward 0.8068182 0.8478261 0.7959184   0.8205128
```

```
## 3   Ridge 0.8295455 0.8695652 0.8163265   0.8461538
## 4   Lasso 0.8181818 0.8510638 0.8163265   0.8205128
```

Based on the comparison above, we can see that the model of Ridge regression is the best in terms of accuracy. It is also worth noting that the performances of the 4 models are not too different from each other.