Practical Work 3 - Regularised Regression Methods

Adib Habbou - Alae Khidour

07-11-2022

Data Analysis

Data Import

```
diabetes_data <- read.table(file = "diabetes.txt", header = TRUE)
```

Data Conversion

```
YBin <- as.numeric(diabetes_data$Y > median(diabetes_data$Y))
diabetes_data <- diabetes_data[,-11]
diabetes_data <- cbind(diabetes_data, YBin)
```

```
head(diabetes_data)
```

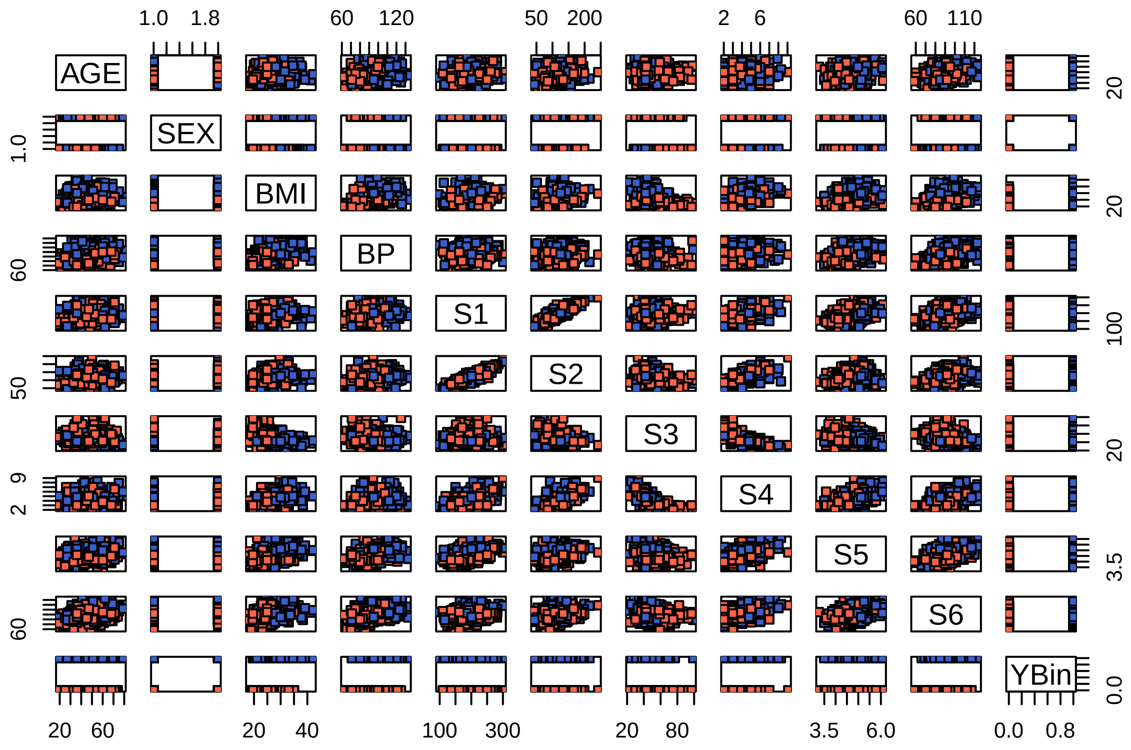
```
  AGE SEX  BMI  BP  S1    S2 S3 S4     S5 S6 YBin
1  59   2 32.1 101 157  93.2 38  4 4.8598 87    1
2  48   1 21.6  87 183 103.2 70  3 3.8918 69    0
3  72   2 30.5  93 156  93.6 41  4 4.6728 85    1
4  24   1 25.3  84 198 131.4 40  5 4.8903 89    1
5  50   1 23.0 101 192 125.4 52  4 4.2905 80  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

```
tail(diabetes_data)
```

```
    AGE SEX  BMI     BP  S1    S2 S3   S4     S5  S6 YBin
437  33   1 19.5  80.00 171  85.4 75 2.00 3.9703  80    0
438  60   2 28.2 112.00 185 113.8 42 4.00 4.9836  93    1
439  47   2 24.9  75.00 225 166.0 42 5.00 4.4427 102    0
440  60   2 24.9  99.67 162 106.6 43 3.77 4.1271  95    0
441  36   1 30.0  95.00 201 125.2 42 4.79 5.1299  85    1
442  36   1 19.6  71.00 250 133.2 97 3.00 4.5951  92    0
```

Data Visualization

```
pairs(diabetes_data, pch = 22,
      bg = c("tomato1","royalblue3")[unclass(factor(diabetes_data[,"YBin"]))])
```

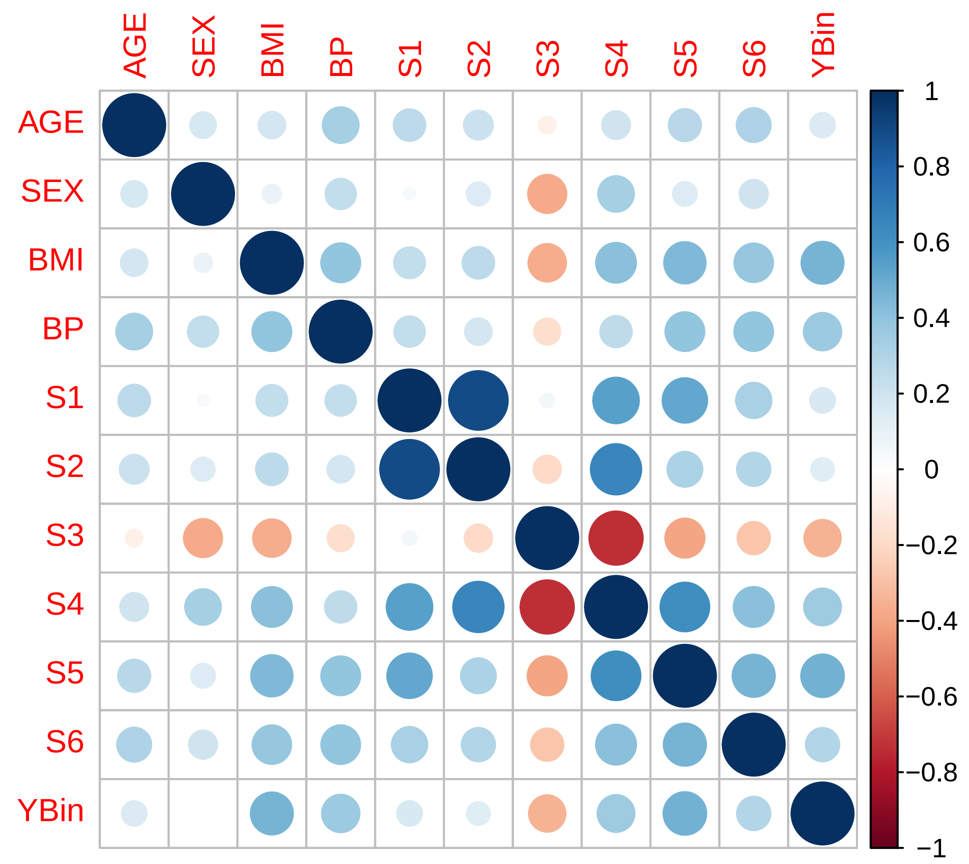


Red squares correspond to observations where $YBin$ is equal to 0 which means Y is lower than the median. Blue squares correspond to observations where $YBin$ is equal to 1 which means Y is greater than the median. From the above plot, we observe the following:

- Colinearity between $S1$ and $S2$
- Above a certain value for BMI and BP we only find blue squares

Study of correlation

```r
corr <- cor(diabetes_data)
corrplot(corr, method = "circle")
```



The correlation plot provides us with a lot of information such as:

- S1 and S2 are highly positively correlated;
- S3 and S4 are highly negatively correlated.

We need to keep in mind the correlation between our variables.

The potential colinearity between variables can have an impact on the Standard Error.

More than that, it means that the co-variable signifacitivity test is useless.

Logistic Regression

Data Partitionning

```r
sample <- sample(c(TRUE, FALSE), nrow(diabetes_data), replace = TRUE, prob = c(0.8, 0.2))
train_data <- diabetes_data[sample, ]
test_data <- diabetes_data[!sample, ]
```

Application of logistic regression

```r
reg_log = glm(formula = YBin ~ ., family = binomial, data = train_data)
summary(reg_log)
```

```

Call:
glm(formula = YBin ~ ., family = binomial, data = train_data)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.4429  -0.7059  -0.1318   0.7664   2.2188

Coefficients:
              Estimate Std. Error z value Pr(>|z|)
(Intercept) -7.9900967  4.7479201  -1.683 0.092402 .
AGE          0.0056811  0.0117240   0.485 0.627980
SEX         -1.1953437  0.3356120  -3.562 0.000368 ***
BMI          0.1408921  0.0396128   3.557 0.000375 ***
BP           0.0284984  0.0119963   2.376 0.017521 *
S1           0.0112509  0.0474511   0.237 0.812576
S2          -0.0235857  0.0471034  -0.501 0.616567
S3          -0.0755260  0.0578409  -1.306 0.191636
S4          -0.0023359  0.3633716  -0.006 0.994871
S5           1.6177768  1.1781280   1.373 0.169698
S6          -0.0009758  0.0153589  -0.064 0.949341
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 482.42  on 347  degrees of freedom
Residual deviance: 322.75  on 337  degrees of freedom
AIC: 344.75

Number of Fisher Scoring iterations: 5
```

Interpretation of the results

- For the Deviance Residuals we observe that they are close to be centered on 0 and roughly symmetrical.
- We can make assumptions on the significativity of the co-variables by looking at their p-values. For instance BMI and $S5$ are highly significant co-variables for our model meanwhile AGE and $S4$ are less significant.
- The dispersion parameter in our case is equal to 1, but we can adjust it if we want too. Since we are not estimating the variance from the data instead we are just deriving it from the mean, it is possible that the variance is underestimated.
- The Akaike Information Criterion (AIC) will help us to compare between different models.
- The number of Fisher Scoring iterations tells us how quickly the function converges to the maximum likelihood estimated for the coefficients.

Study of the coefficients

```
reg_log$coefficients
```

```
  (Intercept)           AGE           SEX           BMI            BP 
-7.9900966794  0.0056811233 -1.1953437486  0.1408921394  0.0284983749 
           S1            S2            S3            S4            S5 
 0.0112508931 -0.0235857284 -0.0755260460 -0.0023359257  1.6177767505 
           S6 
-0.0009758189 
```

We should also not only look at the estimated value of the coefficient to determine co-variable significativity because even a very low estimated coefficient can become bigger at the end depending on the co-variable unit and magnitude.

-> The most significant co-variable is $S5$, the less significant co-variable is AGE.

But we need to keep in mind that it does not mean that AGE is not significant in reality, it is just the least significant in our model according to the computed p-values for our data set.

Predictions type response

```r
predict_response <- predict.glm(reg_log, newdata = test_data, type = "response")
predict_response
```

```
        10         12         14         17         18         19         59 
0.94368200 0.14816924 0.58901218 0.81300489 0.71755897 0.57865236 0.06534977 
        60         62         63         65         71         74         76 
0.28440924 0.77701566 0.02490906 0.25034394 0.05220693 0.20698489 0.30798449 
        80         81         82         83         85         97        103 
0.11507647 0.58331661 0.23717402 0.03178885 0.06432964 0.89087369 0.34062404 
       106        109        111        118        124        140        145 
0.23206270 0.94596593 0.02101791 0.96525571 0.39774408 0.97925217 0.64308084 
       150        156        165        167        169        186        194 
0.81430630 0.65341847 0.32159309 0.07297506 0.88682831 0.86205940 0.40022993 
       199        200        201        220        223        225        230 
0.50315181 0.85068189 0.10423184 0.53809688 0.51916762 0.15259770 0.08645594 
       233        238        240        241        242        243        247 
0.49627704 0.07009210 0.67049569 0.91694482 0.16042649 0.12016256 0.31884745 
       248        257        259        260        266        277        278 
0.09207751 0.97620816 0.23393675 0.47849027 0.26373068 0.69239687 0.21830626 
       279        282        287        304        305        308        309 
0.28168542 0.16517691 0.04517958 0.94883083 0.20601715 0.43720950 0.15827339 
       310        318        323        325        331        334        339 
0.21463602 0.64275366 0.99658186 0.90834148 0.61381483 0.76013877 0.38356020 
       340        342        346        356        365        369        370 
0.48121388 0.66473506 0.24060972 0.26157232 0.50543566 0.86211128 0.79478631 
       382        383        386        389        392        403        406 
0.11453807 0.99239198 0.35763011 0.62953089 0.06862223 0.52549878 0.98883153 
       410        411        415        417        419        420        422 
0.57710899 0.65776826 0.57545891 0.87120482 0.08253818 0.13413883 0.77392503 
       423        427        428 
0.63298005 0.50722764 0.27024723 
```

Using the type *response* we obtain values between 0 and 1 which correspond to the probability of the variable $YBin$ being equal to 1 computed from the area under the link function.

Predictions type link

```r
predict_link <- predict.glm(reg_log, newdata = test_data, type = "link")
predict_link
```

```
         10          12          14          17          18          19 
 2.81877508 -1.74903275  0.35988330  1.46965466  0.93238534  0.31724363 
         59          60          62          63          65          71 
-2.66041849 -0.92269424  1.24835897 -3.66729928 -1.09677876 -2.89892103 
         74          76          80          81          82          83 
-1.34319646 -0.80955896 -2.03990435  0.33640343 -1.16823579 -3.41633474 
         85          97         103         106         109         111 
-2.67724281  2.09969660 -0.66051450 -1.19670050  2.86259169 -3.84113830 
        118         124         140         145         150         156 
 3.32437769 -0.41487369  3.85434746  0.58876113  1.47823802  0.63409967 
        165         167         169         186         194         199 
-0.74646024 -2.54186278  2.05874535  1.83250102 -0.40450715  0.01260742 
        200         201         220         223         225         230 
 1.73995928 -2.15106398  0.15268347  0.07670809 -1.71437054 -2.35769674 
        233         238         240         241         242         243 
-0.01489213 -2.58527543  0.71042786  2.40154214 -1.65505822 -1.99089168 
        247         248         257         259         260         266 
-0.75907351 -2.28852832  3.71433304 -1.18621397 -0.08609207 -1.02666753 
        277         278         279         282         287         304 
 0.81134887 -1.27556408 -0.93611669 -1.62020276 -3.05087805  2.92009328 
        305         308         309         310         318         323 
-1.34910247 -0.25249494 -1.67113142 -1.29720361  0.58733598  5.67523573 
        325         331         334         339         340         342 
 2.29355045  0.46337635  1.15344048 -0.47446407 -0.07517987  0.68446747 
        346         356         365         369         370         382 
-1.14933966 -1.03781235  0.02174349  1.83293738  1.35402147 -2.04520216 
        383         386         389         392         403         406 
 4.87091485 -0.58566506  0.53020486 -2.60804849  0.10208367  4.48342907 
        410         411         415         417         419         420 
 0.31091666  0.65336458  0.30415904  1.91165372 -2.40835001 -1.86484926 
        422         423         427         428 
 1.23060832  0.54502269  0.02891255 -0.99336862 
```

Using the type *link* we obtain the values of the link function.

Odd-Ratios

```r
exp(coef(reg_log))
```

```
 (Intercept)          AGE          SEX          BMI           BP           S1 
0.0003388013 1.0056972915 0.3025999180 1.1513004614 1.0289083388 1.0113144224 
          S2           S3           S4           S5           S6 
0.9766902409 0.9272555790 0.9976668005 5.0418685153 0.9990246570 
```

From the odd-ratios obtained for each co-variable we can evaluate the influence of the co-variable on the target knowing that:

- When the Odd-Ratio is lower than 1 it means that the co-variable had a negative influence on the target, for instance AGE, SEX, $S1$, $S3$, $S4$ and $S6$.
- When the Odd-Ratio is greater than 1 it means that the co-variable had a positive influence on the target, for instance BMI, BP and $S2$.

The limits of this approach is that if we change the binary labels we had (0 and 1 for our case) we will obtain different values for the estimated coefficients.

Performance

MAP

Using the Maximum A Posteriori criteria we can make predictions for our binary variable $YBin$:

```r
prediction <- as.numeric(predict.glm(reg_log, diabetes_data, type = "response") > 0.5)
```

```r
table(prediction)
```

```
prediction
  0   1 
226 216 
```

Knowing that our target variable have the following values:

```r
table(diabetes_data$YBin)
```

```

  0   1 
221 221 
```

By comparing the both tables we can tell that our predictions are quite good, since our model has nearly the same count for 0 and 1 than the target variable in our data set.

Confusion Matrix

```
confusion_matrix <- table(diabetes_data$YBin, prediction)
confusion_matrix
```

```
   prediction
      0   1
  0 171  50
  1  55 166
```

Here we just computed the confusion matrix for our model. This matrix has 4 values which corresponds respectively to the number of True Negative, False Negative, False Positive and True Positive.

- True Negative is the specificity which is the ability to predict $Y\hat{B}in = 0$ for $YBin = 0$
- True Positive is the sensitivity which is ability to predict $Y\hat{B}in = 1$ for $YBin = 1$

Accuracy

```
accuracy <- (confusion_matrix[1,1] + confusion_matrix[2,2]) / nrow(diabetes_data)
accuracy
```

```
[1] 0.7624434
```

The accuracy of our model correspond to it's ability to predict the right value (0 or 1) for all the observations of our data set. In our case it's equal to 76 %.

Gloabl Error

```
global_error <- (confusion_matrix[1,2] + confusion_matrix[2,1]) / nrow(diabetes_data)
global_error
```

```
[1] 0.2375566
```

The global error of our model correspond to it's inability to predict the right value (0 or 1) for all the observations of our data set. In our case it's equal to 24 %.

Recall

```
recall <- confusion_matrix[2,2] / (confusion_matrix[1,2] + confusion_matrix[2,2])
recall
```

```
[1] 0.7685185
```

The recall of our model correspond to the correctly predicted positive rate. In our case it's equal to 75 %.

Precision

```r
precision <- confusion_matrix[2,2] / (confusion_matrix[2,1] + confusion_matrix[2,2])
precision
```

```
[1] 0.7511312
```

The precision correspond to the rate of correct positive predictions. In our case it's equal to 78 %.

F1-Score

```r
f1_score <- (2 * precision * recall) / (precision + recall)
f1_score
```

```
[1] 0.7597254
```

The F1-score correspond to the ability to predict positive individuals well. In our case it's equal to 76 %.

The F_β-score uses a more general formula where β is chosen such that the recall is considered β times as important as the precision:

$$F_\beta = \frac{(1+\beta^2) * precision * recall}{(\beta^2 * precision) + recall}$$

False Positive Rate

```r
confusion_matrix[2,1] / nrow(diabetes_data)
```

```
[1] 0.1244344
```

The false positive is equal to 11 %.

False Negative Rate

```r
confusion_matrix[1,2] / nrow(diabetes_data)
```

```
[1] 0.1131222
```

The false negative is equal to 13 %.

K-Fold

The Cross-Validation is a technique which simply reserves a part of the training data and uses it to test the model while the remaining non-reserved data is used to train the model.

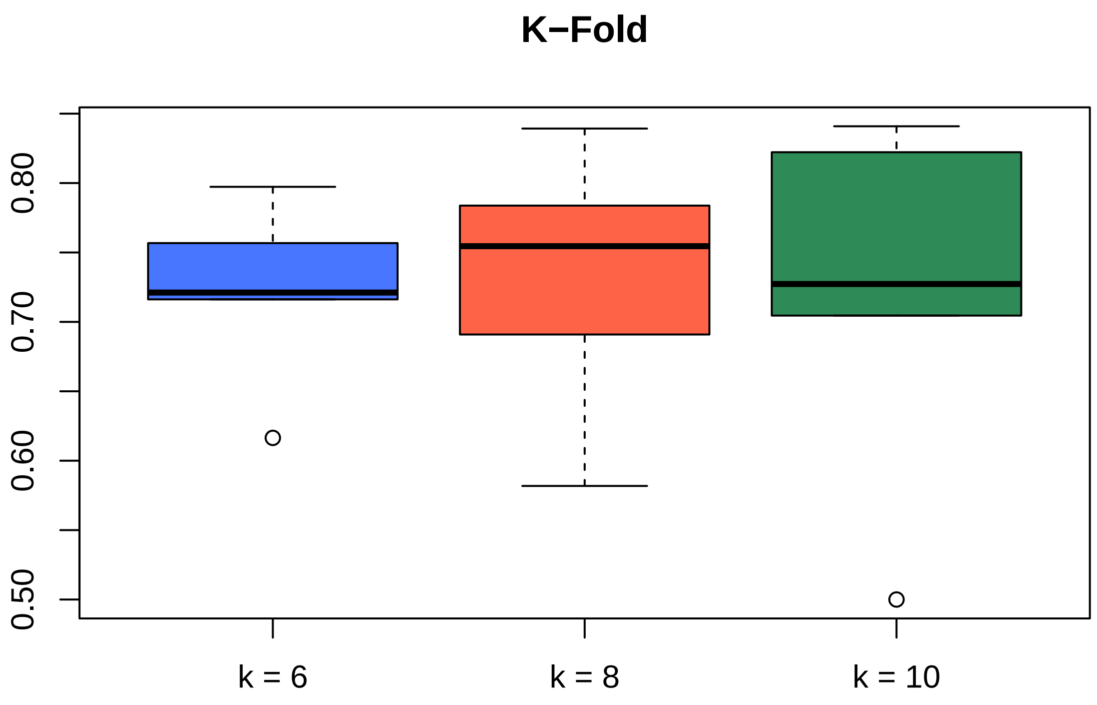
The principle behind K-Fold cross validation is that we start by dividing our data set into K equal parts. Then we will train our model on the K-1 first parts and use the last part to test the model. Then we will use another combination of parts to train and test our model until we computed all the possible combinations. In the end, every part of the data set is used for testing and we can then have an idea of the performance of our model on new data.

This technique is used to avoid overfitting and to know the performance of our model on new data.

Here we are coding a function that takes the number of folds and returns a vector of the performance computed at each iteration. The model used here is logistic regression and we are computing the performance by making predictions and computing the confusion matrix

```r
kfold_all <- function(k) # k is here the number of folds
{
  # create a vector of length number of folds
  performance <- vector(length = k)

  # create a sequence from 1 to k
  folds <- cut(seq(1,nrow(diabetes_data)), breaks = k, labels = FALSE)

  # perform k fold cross validation
  for(i in 1:k)
  {
    # split data by fold
    index <- which(folds == i, arr.ind = TRUE)
    test_data <- diabetes_data[index,]
    train_data <- diabetes_data[-index,]

    # train the logistic regression on the train data set
    reg_log <- glm(YBin ~ ., family = binomial, data = train_data)

    # make predictions
    prediction <- predict(reg_log, test_data)

    # compute confusion matrix
    confusion_matrix <- table(as.numeric(prediction > 0.5), diabetes_data[index,]$YBin)

    # compute the performance
    performance[i] <- (confusion_matrix[1,1] + confusion_matrix[2,2]) / nrow(test_data)
  }

  # returning the vector of performances
  return (performance)
}
```

```
boxplot(kfold_all(6), kfold_all(8), kfold_all(10),
        main = "K-Fold", names = c("k = 6", "k = 8", "k = 10"),
        col = c("royalblue1", "tomato1", "seagreen"))
```



In our case we did 3 different K-Folds for K equal to 6, 8 and 10. By looking at the boxplots we can see that the median of our accuracy is around 75 % with maximum and minimum values that can go from 60 % to 80 % for the 10-fold.

Variable Selection

Statistical Approach

We start by setting up a model with all the variables and another with only an intercept:

```
reg_all <- glm(YBin ~., data = diabetes_data, family = binomial)
reg_none <- glm(YBin ~ 1, data = diabetes_data, family = binomial)
```

Forward Logistic Regression

```
reg_forward <- step(reg_none, list(upper = reg_all), direction = 'forward')
```

```
Start:  AIC=614.74
YBin ~ 1

       Df Deviance    AIC
+ S5    1   500.61 504.61
+ BMI   1   507.06 511.06
+ BP    1   549.95 553.95
+ S4    1   552.60 556.60
+ S3    1   555.06 559.06
+ S6    1   573.71 577.71
+ S1    1   601.14 605.14
+ AGE   1   601.63 605.63
+ S2    1   604.41 608.41
<none>      612.74 614.74
+ SEX   1   612.73 616.73

Step:  AIC=504.61
YBin ~ S5

       Df Deviance    AIC
+ BMI   1   457.80 463.80
+ BP    1   480.20 486.20
+ S3    1   485.31 491.31
+ S1    1   494.40 500.40
+ SEX   1   497.09 503.09
+ S6    1   497.33 503.33
+ S4    1   497.75 503.75
<none>      500.61 504.61
+ S2    1   499.98 505.98
+ AGE   1   500.18 506.18

Step:  AIC=463.8
YBin ~ S5 + BMI

       Df Deviance    AIC
+ BP    1   448.52 456.52
+ S1    1   449.40 457.40
+ S3    1   450.90 458.90
+ SEX   1   454.15 462.15
+ S2    1   454.55 462.55
<none>      457.80 463.80
+ S4    1   457.56 465.56
+ S6    1   457.56 465.56
+ AGE   1   457.78 465.78

Step:  AIC=456.52
YBin ~ S5 + BMI + BP

       Df Deviance    AIC
```

```
+ S1    1   438.95 448.95
+ S3    1   440.93 450.93
+ SEX   1   441.96 451.96
+ S2    1   444.62 454.62
<none>      448.52 456.52
+ AGE   1   448.10 458.10
+ S4    1   448.24 458.24
+ S6    1   448.48 458.48

Step:  AIC=448.95
YBin ~ S5 + BMI + BP + S1

       Df Deviance    AIC
+ SEX   1   431.06 443.06
+ S2    1   434.89 446.89
+ S3    1   435.61 447.61
+ S4    1   436.32 448.32
<none>      438.95 448.95
+ AGE   1   438.91 450.91
+ S6    1   438.94 450.94

Step:  AIC=443.06
YBin ~ S5 + BMI + BP + S1 + SEX

       Df Deviance    AIC
+ S2    1   419.13 433.13
+ S3    1   420.41 434.41
+ S4    1   422.48 436.48
<none>      431.06 443.06
+ S6    1   430.85 444.85
+ AGE   1   431.02 445.02

Step:  AIC=433.13
YBin ~ S5 + BMI + BP + S1 + SEX + S2

       Df Deviance    AIC
<none>      419.13 433.13
+ AGE   1   419.00 435.00
+ S4    1   419.12 435.12
+ S3    1   419.12 435.12
+ S6    1   419.13 435.13
```

We can observe that the forward logistic regression give us a model which contains the following variables: $S5$, BMI, BP, $S1$, SEX and $S5$ with an AIC equal to 433.

Backward Logistic Regression

```r
reg_back <- step(reg_all, direction = 'backward')
```

```
Start:  AIC=440.97
YBin ~ AGE + SEX + BMI + BP + S1 + S2 + S3 + S4 + S5 + S6

       Df Deviance    AIC
- S6    1   418.97 438.97
- S3    1   418.97 438.97
- S4    1   418.98 438.98
- AGE   1   419.11 439.11
- S2    1   420.19 440.19
- S1    1   420.79 440.79
<none>      418.97 440.97
- S5    1   427.77 447.77
- BP    1   433.51 453.51
- SEX   1   434.72 454.72
- BMI   1   437.80 457.80

Step:  AIC=438.97
YBin ~ AGE + SEX + BMI + BP + S1 + S2 + S3 + S4 + S5

       Df Deviance    AIC
- S3    1   418.97 436.97
- S4    1   418.98 436.98
- AGE   1   419.12 437.12
- S2    1   420.20 438.20
- S1    1   420.80 438.80
<none>      418.97 438.97
- S5    1   427.88 445.88
- BP    1   434.04 452.04
- SEX   1   434.76 452.76
- BMI   1   438.19 456.19

Step:  AIC=436.97
YBin ~ AGE + SEX + BMI + BP + S1 + S2 + S4 + S5

       Df Deviance    AIC
- S4    1   419.00 435.00
- AGE   1   419.12 435.12
<none>      418.97 436.97
- S2    1   422.28 438.28
- S1    1   427.40 443.40
- BP    1   434.05 450.05
- SEX   1   434.79 450.79
- BMI   1   438.29 454.29
- S5    1   441.41 457.41

Step:  AIC=435
YBin ~ AGE + SEX + BMI + BP + S1 + S2 + S5

       Df Deviance    AIC
```

```
- AGE   1   419.13 433.13
<none>      419.00 435.00
- S2    1   431.02 445.02
- BP    1   434.05 448.05
- SEX   1   434.83 448.83
- BMI   1   438.30 452.30
- S1    1   438.91 452.91
- S5    1   480.74 494.74

Step:  AIC=433.13
YBin ~ SEX + BMI + BP + S1 + S2 + S5

       Df Deviance    AIC
<none>      419.13 433.13
- S2    1   431.06 443.06
- SEX   1   434.89 446.89
- BP    1   435.39 447.39
- BMI   1   438.47 450.47
- S1    1   438.92 450.92
- S5    1   480.94 492.94
```

We can observe that the backward logistic regression give us a model which contains the following variables: $S5$, BMI, BP, $S1$, SEX and $S5$ with an AIC equal to 433.

Stepwise Logistic Regression

```r
reg_both <- step(reg_all, direction = 'both')
```

```
Start:  AIC=440.97
YBin ~ AGE + SEX + BMI + BP + S1 + S2 + S3 + S4 + S5 + S6

       Df Deviance    AIC
- S6    1   418.97 438.97
- S3    1   418.97 438.97
- S4    1   418.98 438.98
- AGE   1   419.11 439.11
- S2    1   420.19 440.19
- S1    1   420.79 440.79
<none>      418.97 440.97
- S5    1   427.77 447.77
- BP    1   433.51 453.51
- SEX   1   434.72 454.72
- BMI   1   437.80 457.80

Step:  AIC=438.97
YBin ~ AGE + SEX + BMI + BP + S1 + S2 + S3 + S4 + S5

       Df Deviance    AIC
- S3    1   418.97 436.97
- S4    1   418.98 436.98
- AGE   1   419.12 437.12
- S2    1   420.20 438.20
- S1    1   420.80 438.80
<none>      418.97 438.97
+ S6    1   418.97 440.97
- S5    1   427.88 445.88
- BP    1   434.04 452.04
- SEX   1   434.76 452.76
- BMI   1   438.19 456.19

Step:  AIC=436.97
YBin ~ AGE + SEX + BMI + BP + S1 + S2 + S4 + S5

       Df Deviance    AIC
- S4    1   419.00 435.00
- AGE   1   419.12 435.12
<none>      418.97 436.97
- S2    1   422.28 438.28
+ S3    1   418.97 438.97
+ S6    1   418.97 438.97
- S1    1   427.40 443.40
- BP    1   434.05 450.05
- SEX   1   434.79 450.79
- BMI   1   438.29 454.29
- S5    1   441.41 457.41

Step:  AIC=435
```

```
YBin ~ AGE + SEX + BMI + BP + S1 + S2 + S5

       Df Deviance    AIC
- AGE   1   419.13 433.13
<none>      419.00 435.00
+ S4    1   418.97 436.97
+ S3    1   418.98 436.98
+ S6    1   419.00 437.00
- S2    1   431.02 445.02
- BP    1   434.05 448.05
- SEX   1   434.83 448.83
- BMI   1   438.30 452.30
- S1    1   438.91 452.91
- S5    1   480.74 494.74

Step:  AIC=433.13
YBin ~ SEX + BMI + BP + S1 + S2 + S5

       Df Deviance    AIC
<none>      419.13 433.13
+ AGE   1   419.00 435.00
+ S4    1   419.12 435.12
+ S3    1   419.12 435.12
+ S6    1   419.13 435.13
- S2    1   431.06 443.06
- SEX   1   434.89 446.89
- BP    1   435.39 447.39
- BMI   1   438.47 450.47
- S1    1   438.92 450.92
- S5    1   480.94 492.94
```

We can observe that the stepwise logistic regression give us a model which contains the following variables: $S5$, BMI, BP, $S1$, SEX and $S5$ with an AIC equal to 433.

Final model

Each time, we obtain the same value for the *AIC* criteria which equals 433. We could choose another criteria for our model selection, such as *BIC* or C_p. It could influence our model because the formula we will minimize changes. This formula will depend differently on the number of co-variables p. So the choice of criteria will always depend on the business constraint behind. More than that, we can customize our criteria as we want by choosing the value of λ in the function *step*.

Our final model will be the following:

```r
diabetes_reg <- lm(formula(reg_both), data = diabetes_data)
summary(diabetes_reg)
```

```
Call:
lm(formula = formula(reg_both), data = diabetes_data)

Residuals:
     Min       1Q   Median       3Q      Max 
-1.02548 -0.29301 -0.00166  0.31694  0.94958 

Coefficients:
             Estimate Std. Error t value Pr(>|t|)    
(Intercept) -1.962754   0.188818 -10.395  < 2e-16 ***
SEX         -0.169907   0.042440  -4.003 7.34e-05 ***
BMI          0.024674   0.005261   4.690 3.66e-06 ***
BP           0.006835   0.001605   4.257 2.54e-05 ***
S1          -0.007748   0.001642  -4.719 3.20e-06 ***
S2           0.006534   0.001709   3.823 0.000151 ***
S5           0.457993   0.054360   8.425 5.31e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.4021 on 435 degrees of freedom
Multiple R-squared:  0.3634,    Adjusted R-squared:  0.3546 
F-statistic: 41.38 on 6 and 435 DF,  p-value: < 2.2e-16
```

From the summary of our regression we can tell that:

- The residuals are quite symmetrically distributed around their median;
- The intercept is equal to -1.962754, we also notice the influence of each co-variable on Y;
- The standard error and the t-value are provided to show how the p-values were calculated;
- ALL the p-values are very low which means that all co-variables are significant;
- The R^2 tells us that the p co-variables can explain 36 % of the variation in the target variable YBin;
- The first degree of freedom corresponds to $p - 1$ with $p = 7$ the number of variables of the model;
- The second degree of freedom corresponds to $n - p$ with $n = 442$ the number of data points.

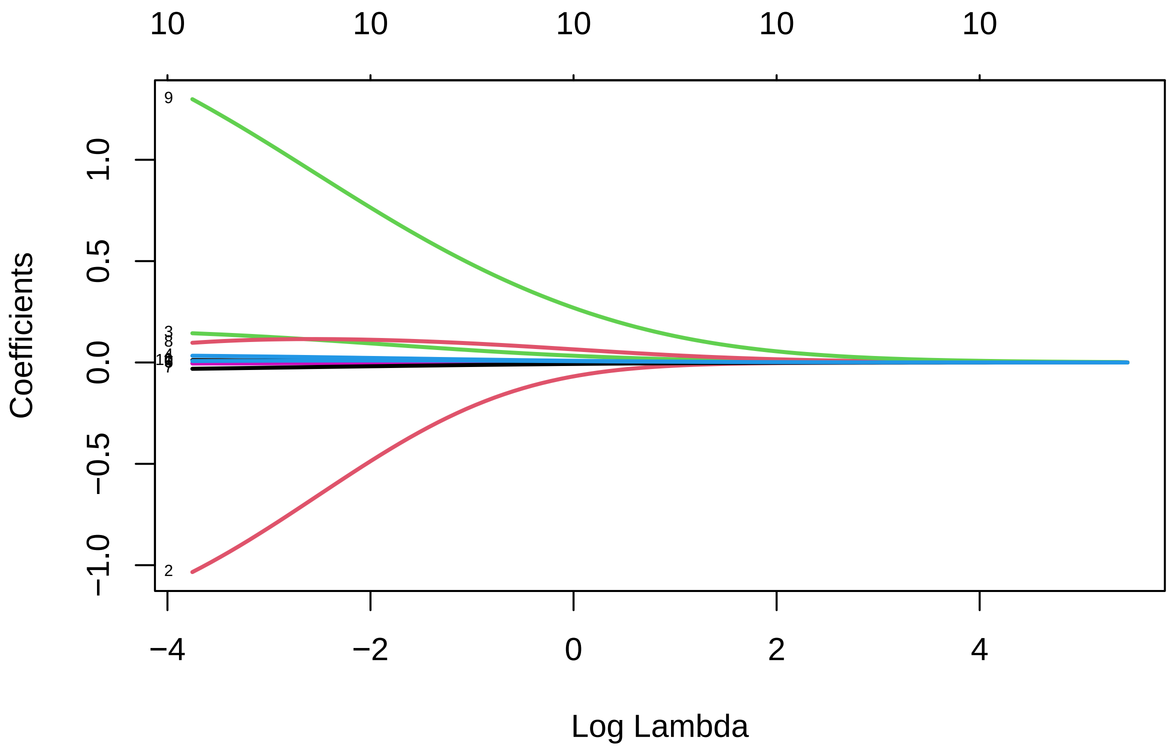
Penalized Methods

```r
sample <- sample(c(TRUE, FALSE), nrow(diabetes_data), replace = TRUE, prob = c(0.8, 0.2))
train_data <- diabetes_data[sample, ]
test_data <- diabetes_data[!sample, ]
```

Ridge Regression

Regularization Path

```r
ridge <- glmnet(x = train_data[,-11], y = train_data$YBin, alpha = 0, family = "binomial")
plot(ridge, xvar = "lambda", label = TRUE, lwd = 2)
```



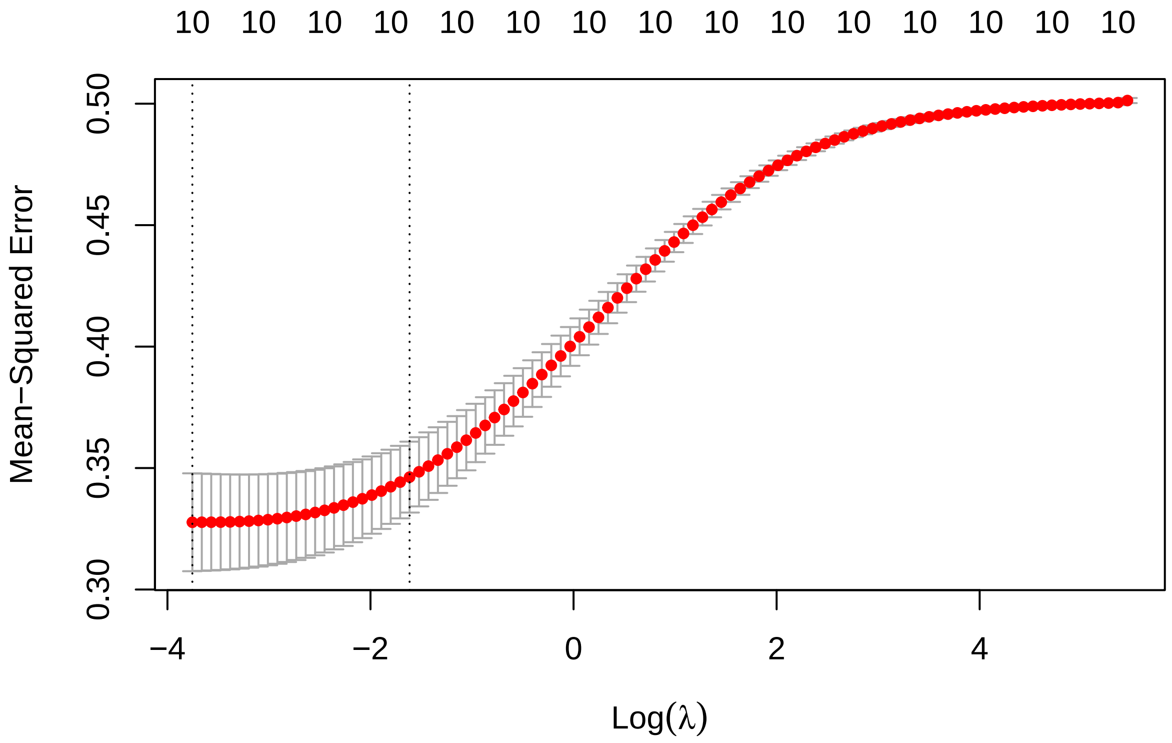
In the plot above, we can see the evolution of the coefficient values depending on λ Knowing that we are doing a ridge regression, we can tell that they will all converge to 0.

However, ridge regression does not perform feature selection and will retain all available features in the final model. Therefore, a ridge model is good if we suppose that there is a need to retain all features in our model yet reduce the noise that less influential variables may create.

If greater interpretation is necessary and many of the features are redundant or irrelevant then a lasso or elastic net penalty may be preferable.

Cross-Validation

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



In the above plot, we visualize the cross validation curve:

- λ_{min} is the value which minimizes the Mean Squared Error in Cross-Validation;
- λ_{1se} is the value which is the largest λ value within 1 standard error;
- The intervals estimate variance of the loss metric (red points) using Cross-Validation;
- The vertical lines show the locations of λ_{min} and λ_{1se};
- The numbers across the top are the number of non-zero coefficients.

We observe a slight improvement in the Mean Squared Error as our penalty $log(\lambda)$ gets larger, suggesting that a regular OLS model likely overfits the training data. But as we constrain it further by continuing to increase the penalty of our Mean Squared Error starts to increase.

Lambda Min

```
lambda_min <- cv_ridge$lambda.min
ridge_min <- glmnet(x = train_data[,-11], y = train_data$YBin,
                    alpha = 0, family = "binomial", lambda = lambda_min)
ridge_min$beta
```

```
10 x 1 sparse Matrix of class "dgCMatrix"
              s0
AGE  0.011780226
SEX -1.034321232
BMI  0.144095420
BP   0.033272898
S1  -0.003418393
S2  -0.004901545
S3  -0.031189107
S4   0.096812586
S5   1.299302188
S6   0.007230844
```

The model obtained using λ_{min} gives us the model with lowest Mean Squared Error, which can seem to be a good choice but in fact it can implies overfitting.

```
prediction_min <- as.numeric(
  predict(ridge_min, as.matrix(diabetes_data[,-11]), type = "response") > 0.5)
```

Using the Maximum A Posteriori criteria we can make predictions for our binary variable $YBin$:

```
table(prediction_min)
```

```
prediction_min
  0   1
218 224
```

By comparing with the table of the target variable we can tell that our predictions are quite consistent, since our model has nearly the same count for 0 and 1 than the target variable in our data set.

```
confusion_matrix <- table(diabetes_data$YBin, prediction_min)
confusion_matrix
```

```
   prediction_min
      0   1
  0 165  56
  1  53 168
```

```
accuracy <- (confusion_matrix[1,1] + confusion_matrix[2,2]) / nrow(diabetes_data)
accuracy
```

```
[1] 0.7533937
```

The accuracy is equal to 76.2 %.

Lambda 1se

```
lambda_1se <- cv_ridge$lambda.1se
ridge_1se <- glmnet(x = train_data[,-11], y = train_data$YBin,
                    alpha = 0, family = "binomial", lambda = lambda_1se)
ridge_1se$beta
```

```
10 x 1 sparse Matrix of class "dgCMatrix"
               s0
AGE  0.0068028289
SEX -0.3708382040
BMI  0.0807923072
BP   0.0194120409
S1  -0.0002557427
S2  -0.0015573200
S3  -0.0160877961
S4   0.1067847839
S5   0.6495087342
S6   0.0098241418
```

The model obtained using λ_{1se} gives us a simpler model which can avoid overfitting.

```
prediction_1se <- as.numeric(
  predict(ridge_1se, as.matrix(diabetes_data[,-11]), type = "response") > 0.5)
```

Using the Maximum A Posteriori criteria we can make predictions for our binary variable $YBin$:

```
table(prediction_1se)
```

```
prediction_1se
  0   1
220 222
```

By comparing with the table of the target variable we can tell that our predictions are quite consistent, since our model has nearly the same count for 0 and 1 than the target variable in our data set.

```
confusion_matrix <- table(diabetes_data$YBin, prediction_1se)
confusion_matrix
```

```
   prediction_1se
      0   1
  0 166  55
  1  54 167
```

```
accuracy <- (confusion_matrix[1,1] + confusion_matrix[2,2]) / nrow(diabetes_data)
accuracy
```

```
[1] 0.7533937
```

The accuracy is equal to 76.6 %.

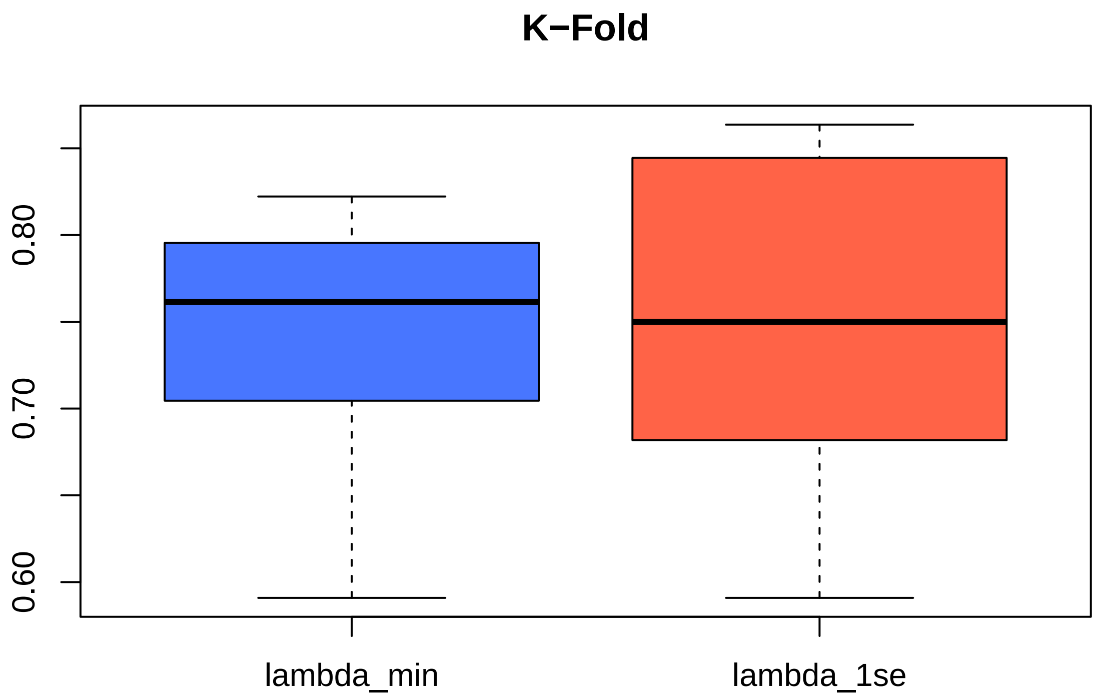
K-Fold function

Let's code a function that will take not only the number of folds but also the lambda so we can use it for both values of λ_{min} and λ_{1se} in our ridge regression model.

```r
kfold_ridge <- function(k, lambda)
{
  # create a vector of length number of folds
  performance <- vector(length = k)

  # create a sequence from 1 to k
  folds <- cut(seq(1,nrow(diabetes_data)), breaks = k, labels = FALSE)

  # perform 10 fold cross validation
  for(i in 1:k)
  {
    # split data by fold
    index <- which(folds == i, arr.ind = TRUE)
    test_data <- diabetes_data[index,]
    train_data <- diabetes_data[-index,]

    # train the logistic regression on the train data set
    reg_log <- glmnet(x = train_data[,-11], y = train_data$YBin,
                      alpha = 0, family = "binomial", lambda = lambda)

    # make predictions
    prediction <- as.numeric(
      predict(reg_log, as.matrix(test_data[,-11]), type = "response") > 0.5)
    confusion_matrix <- table(prediction, test_data$YBin)

    # compute the performance
    performance[i] <- (confusion_matrix[1,1] + confusion_matrix[2,2]) / nrow(test_data)
  }

  # returning the vector of performances
  return (performance)
}
```

K-Fold boxplots

```
boxplot(kfold_ridge(10, lambda_min), kfold_ridge(10, lambda_1se), main = "K-Fold",
        names = c("lambda_min", "lambda_1se"), col = c("royalblue1", "tomato1"))
```

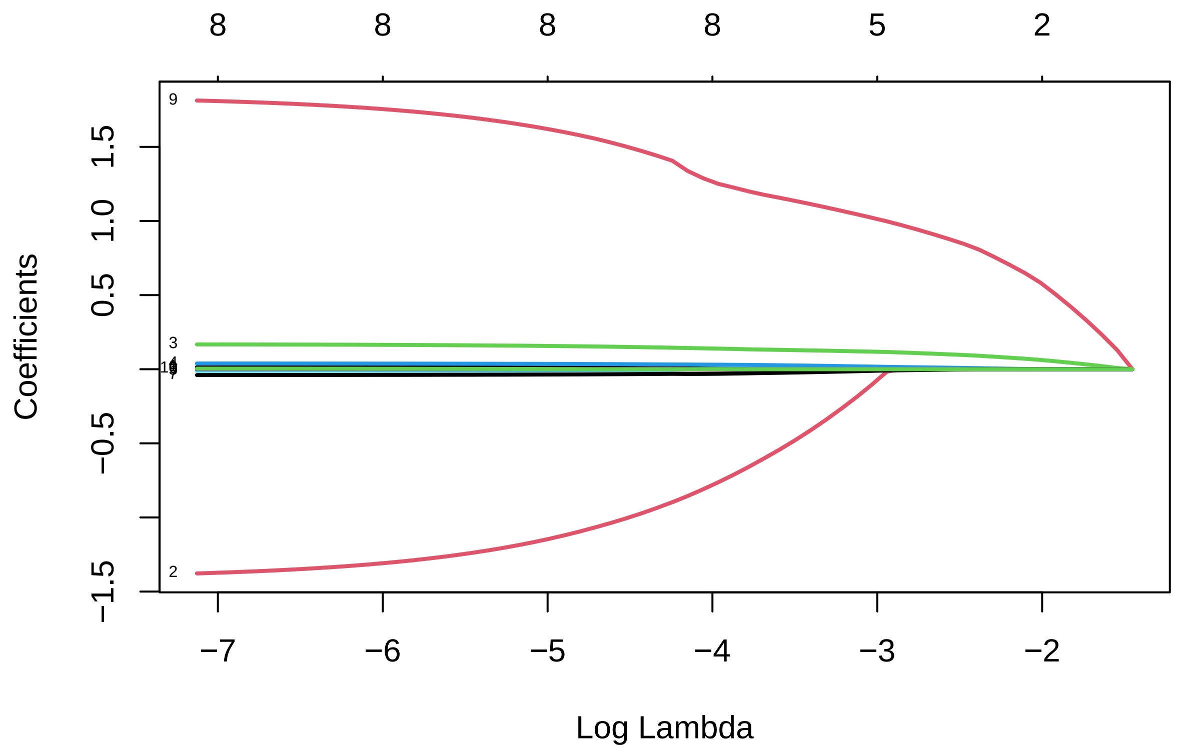


In our case we did two 10-Folds with λ_{min} and λ_{1se}. By looking at the boxplots we can see that the median of our accuracy in between 75 % and 80 % for the two models. But we can clearly observe that the box for λ_{1se} is higher than the one for λ_{min} which means in generally we will have a better accuracy by using λ_{1se}. More than that, we know that with λ_{1se} we are having a larger penalization than with λ_{min} which explains why we have a larger interquartile range and a larger distance between the minmum and the maximum.

Lasso Regression

Regularization Path

```r
lasso <- glmnet(x = train_data[,-11], y = train_data$YBin, alpha = 1, family = "binomial")
plot(lasso, xvar = "lambda", label = TRUE, lwd = 2)
```

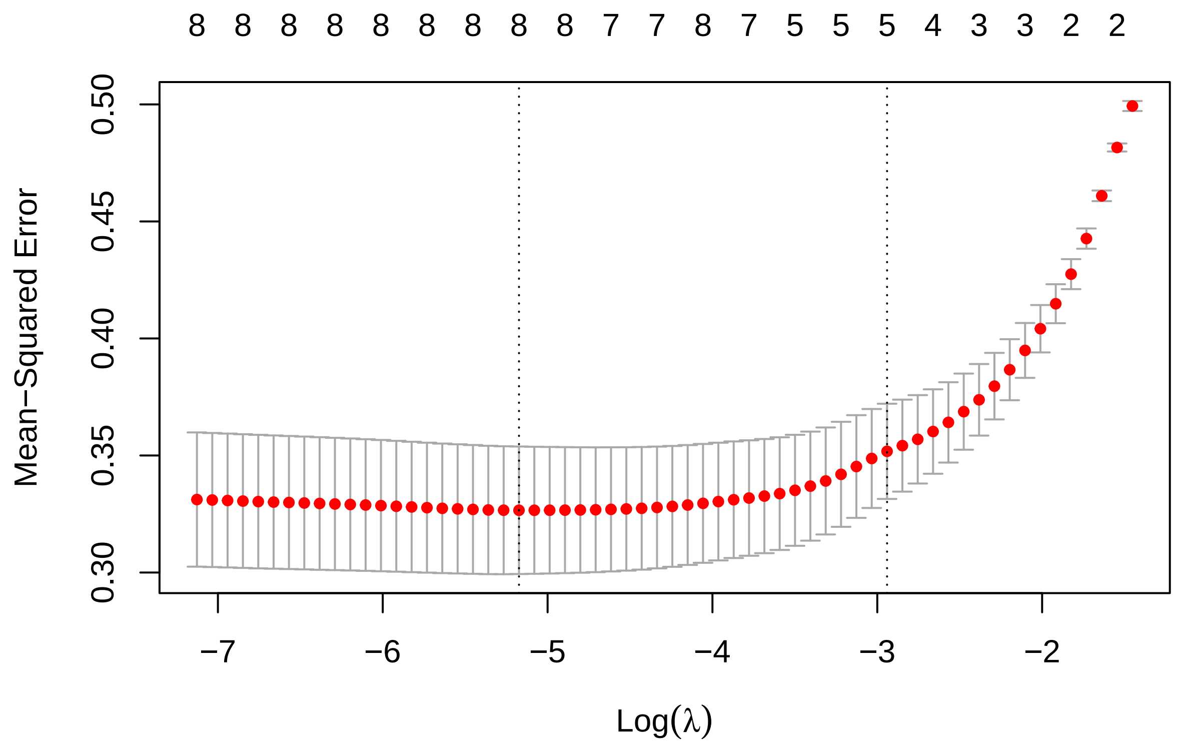


In the plot above, we can see the evolution of the coefficient values depending on λ Knowing that we are doing a lasso regression, we can tell that they will all converge to 0 one by one and not all at the same time.

When a data set has many co-variables, lasso can be used to identify and extract those co-variables which are the most significant. Switching to the lasso penalty also conducts automated variable selection.

Cross-Validation

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



We observe a slight improvement in the Mean Squared Error as our penalty $log(\lambda)$ gets larger, suggesting that a regular OLS model likely overfits the training data. But as we constrain it further by continuing to increase the penalty our Mean Squared Error starts to increase.

Lambda Min

```r
lambda_min <- cv_lasso$lambda.min
lasso_min <- glmnet(x = train_data[,-11], y = train_data$YBin,
                    alpha = 1, family = "binomial", lambda = lambda_min)
lasso_min$beta
```

```
10 x 1 sparse Matrix of class "dgCMatrix"
              s0
AGE  0.011199543
SEX -1.186277595
BMI  0.158657128
BP   0.036306767
S1  -0.007128551
S2   .
S3  -0.036311964
S4   .
S5   1.652994086
S6   0.001486600
```

The model obtained using λ_{min} gives us the model with lowest Mean Squared Error, which can seem to be a good choice but in fact it can implies overfitting. We observe that the model does not contain AGE, $S2$ and $S4$.

```r
prediction_min <- as.numeric(
  predict(lasso_min, as.matrix(diabetes_data[,-11]), type = "response") > 0.5)
```

Using the Maximum A Posteriori criteria we can make predictions for our binary variable $YBin$:

```r
table(prediction_min)
```

```
prediction_min
  0   1
219 223
```

By comparing with the table of the target variable we can tell that our predictions are quite consistent, since our model has nearly the same count for 0 and 1 than the target variable in our data set.

```r
confusion_matrix <- table(diabetes_data$YBin, prediction_min)
confusion_matrix
```

```
   prediction_min
      0   1
  0 163  58
  1  56 165
```

```r
accuracy <- (confusion_matrix[1,1] + confusion_matrix[2,2]) / nrow(diabetes_data)
accuracy
```

```
[1] 0.7420814
```

The accuracy is equal to 76.5 %.

Lambda 1se

```
lambda_1se <- cv_lasso$lambda.1se
lasso_1se <- glmnet(x = train_data[,-11], y = train_data$YBin,
                    alpha = 1, family = "binomial", lambda = lambda_1se)
lasso_1se$beta
```

```
10 x 1 sparse Matrix of class "dgCMatrix"
              s0
AGE  .
SEX -0.015962142
BMI  0.116206311
BP   0.015752329
S1   .
S2   .
S3  -0.008381972
S4   .
S5   0.997527620
S6   .
```

The model obtained using λ_{1se} gives us a simpler model which can avoid overfitting. We observe that the model does not contain AGE, $S1$, $S2$, $S4$ and $S6$, so the model is simpler than the one with λ_{min}.

```
prediction_1se <- as.numeric(
  predict(lasso_1se, as.matrix(diabetes_data[,-11]), type = "response") > 0.5)
```

Using the Maximum A Posteriori criteria we can make predictions for our binary variable $YBin$:

```
table(prediction_1se)
```

```
prediction_1se
  0   1
222 220
```

By comparing with the table of the target variable we can tell that our predictions are quite consistent, since our model has nearly the same count for 0 and 1 than the target variable in our data set.

```
confusion_matrix <- table(diabetes_data$YBin, prediction_1se)
confusion_matrix
```

```
   prediction_1se
      0   1
  0 165  56
  1  57 164
```

```
accuracy <- (confusion_matrix[1,1] + confusion_matrix[2,2]) / nrow(diabetes_data)
accuracy
```

```
[1] 0.7443439
```

The accuracy is equal to 74.8 %.

K-Fold function

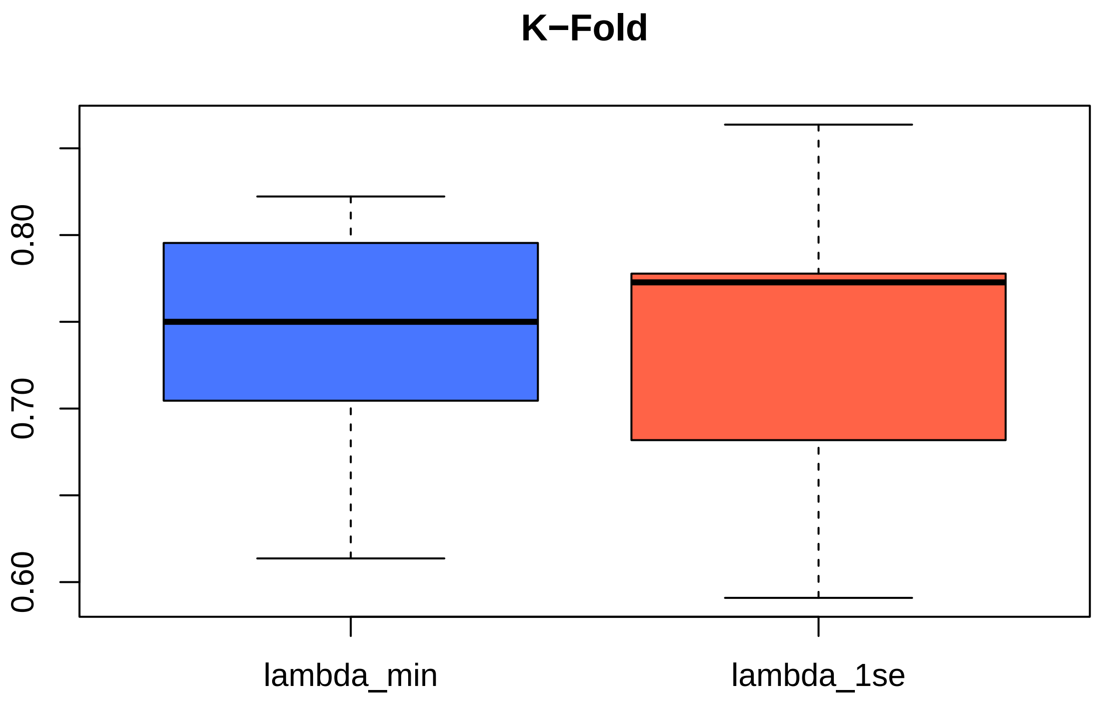
Let's code a function that will take not only the number of folds but also the lambda so we can use it for both values of λ_{min} and λ_{1se} in our ridge lasso model.

```r
kfold_lasso <- function(k, lambda)
{
  # create a vector of length number of folds
  performance <- vector(length = k)

  # create a sequence from 1 to k
  folds <- cut(seq(1,nrow(diabetes_data)), breaks = k, labels = FALSE)

  # perform 10 fold cross validation
  for(i in 1:k)
  {
    # split data by fold
    index <- which(folds == i, arr.ind = TRUE)
    test_data <- diabetes_data[index,]
    train_data <- diabetes_data[-index,]

    # train the logistic regression on the train data set
    reg_log <- glmnet(x = train_data[,-11], y = train_data$YBin,
                      alpha = 1, family = "binomial", lambda = lambda)

    # make predictions
    prediction <- as.numeric(
      predict(reg_log, as.matrix(test_data[,-11]), type = "response") > 0.5)

    confusion_matrix <- table(prediction, test_data$YBin)
    # compute the performance
    performance[i] <- (confusion_matrix[1,1] + confusion_matrix[2,2]) / nrow(test_data)
  }
  # returning the vector of performances
  return (performance)
}
```

We could code a function that takes not only the number of folds and the lambda but also the alpha which will be equal to 0 for a ridge regression and 1 for a lasso regression.

K-Fold boxplots

```r
boxplot(kfold_lasso(10, lambda_min), kfold_lasso(10, lambda_1se), main = "K-Fold",
        names = c("lambda_min", "lambda_1se"), col = c("royalblue1", "tomato1"))
```



In our case we did two 10-Folds with λ_{min} and λ_{1se}. By looking at the boxplots we can see that the median of our accuracy in between 75 % and 80 % for the two models. But we can clearly observe that the box for λ_{1se} is higher than the one for λ_{min} which means in generally we will have a better accuracy by using λ_{1se}. More than that, we know that with λ_{1se} we are having a larger penalization than with λ_{min} which explains why we have a larger interquartile range and a larger distance between the minimum and the maximum.

Conclusion

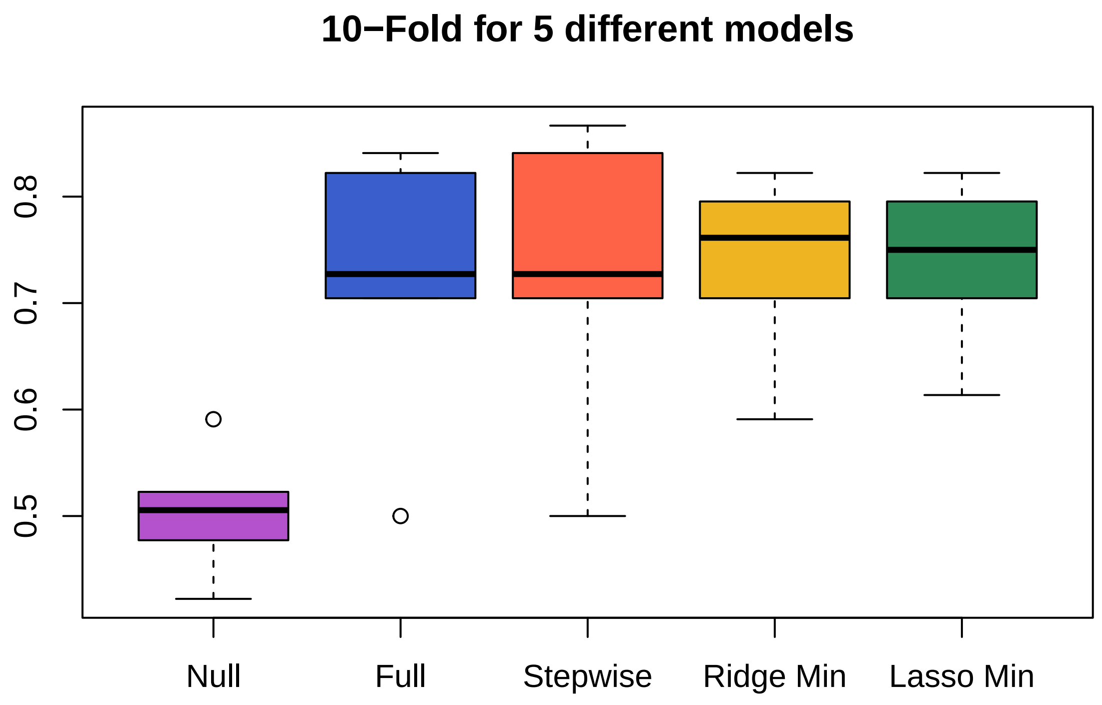
Let's compare the accuracy of all the models obtained using a K-Fold procedure for Cross Validation.

In order to do that let's code a function to make k-fold on the null and the step-wise model.

```
kfold_none <- function(k) # k is here the number of folds
{
  # create a vector of length number of folds
  performance <- vector(length = k)
  # create a sequence from 1 to k
  folds <- cut(seq(1,nrow(diabetes_data)), breaks = k, labels = FALSE)
  # perform k fold cross validation
  for(i in 1:k)
  {
    # split data by fold
    index <- which(folds == i, arr.ind = TRUE)
    test_data <- diabetes_data[index,]
    train_data <- diabetes_data[-index,]
    # train the logistic regression on the train data set
    reg_log <- glm(YBin ~ 1, family = binomial, data = train_data)
    # make predictions
    prediction <- predict(reg_log, test_data)
    # compute confusion matrix and performance
    confusion_matrix <- table(as.numeric(prediction > 0.5), diabetes_data[index,]$YBin)
    performance[i] <- confusion_matrix[1,2] / nrow(test_data)
  }
  return (performance)
}
```

```
kfold_step <- function(k) # k is here the number of folds
{
  # create a vector of length number of folds
  performance <- vector(length = k)
  # create a sequence from 1 to k
  folds <- cut(seq(1,nrow(diabetes_data)), breaks = k, labels = FALSE)
  # perform k fold cross validation
  for(i in 1:k)
  {
    # split data by fold
    index <- which(folds == i, arr.ind = TRUE)
    test_data <- diabetes_data[index,]
    train_data <- diabetes_data[-index,]
    # train the logistic regression on the train data set
    reg_log <- step(glm(YBin ~ ., family = binomial, data = train_data),
                    direction = "both", trace = FALSE)
    # make predictions
    prediction <- predict(reg_log, test_data)
    # compute confusion matrix and performance
    confusion_matrix <- table(as.numeric(prediction > 0.5), diabetes_data[index,]$YBin)
    performance[i] <- (confusion_matrix[1,1] + confusion_matrix[2,2]) / nrow(test_data)
  }
  return (performance)
}
```

```
boxplot(kfold_none(10), kfold_all(10), kfold_step(10),
        kfold_ridge(10, cv_ridge$lambda.min), kfold_lasso(10, cv_lasso$lambda.min),
        main = "10-Fold for 5 different models",
        names = c("Null", "Full", "Stepwise", "Ridge Min", "Lasso Min"),
        col = c("mediumorchid3", "royalblue3", "tomato1", "goldenrod2", "seagreen"))
```



By looking at the boxplots we can compare between all the models we did in this practical work:

- We can notice the presence of outliers;
- The Null model has the lowest accuracy because he doesn't take in count any co-variables;
- The Stepwise model has a very small minimum value which is near to the median of the Null model;
- The Full and Stepwise models have a better accuracy than the Null model but their median are lower than the median for Ridge and Lasso, we can explain that by the fact that Ridge and Lasso are applying penalization in order to avoid overfitting so it has a better predictive power on new data set which is the case when doing K-Fold Cross Validation;
- For Ridge and Lasso we can clearly see that they have the best accuracy but we can observe than the interquartile range and the distance between min and max values are larger for Ridge than for Lasso because Lasso does variable selection. Generally, when we have many small or medium sized effects we should go with Ridge. If we have only a few variables with a medium or large effect, we should go with Lasso;
- "Ridge regression does a proportional shrinkage. Lasso translates each coefficient by a constant factor, truncating at zero." from **The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Hastie, Tibshirani, Friedman**.