Mandatory Assignment 2 - STK-IN4300

Alessia Sanfelici

# Problem 1: Regression

The first problem of the assignment consists in the analysis of data coming from a study on data toxicity. The dataset is composed of 546 observations and 9 variables. The aim of the analysis is to predict acute aquatic toxicity, which is measured through a variable called (the concentration that causes death in 50% of the planktonic crustacean over a test duration of 48 hours). The other 8 variables, used for the prediction, are:

* : the topological polar surface area calculated by means of a contribution method that takes into account nitrogen, oxygen, potassium and sulphur;
* : the Van der Waals surface area (VSA) of atoms that are acceptors of hydrogen bonds;
* : the number of hydrogen atoms bonded to heteroatoms;
* : the lipophilicity of a molecule, this being the driving force of narcosis;
* : a topological index that encodes information about molecular size and branching;
* : information on molecular polarisability;
* : the number of nitrogen atoms present in the molecule;
* : the number of carbon atoms of a certain type, including esters, carboxylic acids, thioesters, carbamic acids, nitriles, etc.;

df\_toxicity <- read.csv("aquatic\_toxicity.csv", sep = ";", header = FALSE)  
colnames(df\_toxicity) <- c("TPSA", "SAacc", "H050", "MLOGP", "RDCHI", "GATS1p","nN", "C040", "LC50")  
# create the dichotomic version of the dataset (where the count variables are transformed using 0/1 dummy encoding)  
  
df\_toxicity\_dummy <- df\_toxicity  
df\_toxicity\_dummy["H050"] <- c(as.integer(df\_toxicity\_dummy ["H050"]>0))  
df\_toxicity\_dummy ["nN"] <- c(as.integer(df\_toxicity\_dummy ["nN"]>0))  
df\_toxicity\_dummy ["C040"] <- c(as.integer(df\_toxicity\_dummy ["C040"]>0))

## Linear Regression

Firstly, I want to compare the results of the application of linear regression with both the original dataset and a modified dataset, where all the count variables have been transformed using a 0/1 dummy encoding where 0 represents absence of the specific atom and 1 represents presence of the specific atoms. For this reason, I have already created (above) a dicothomic version of the dataset.

Now, I can divide both the datasets into train and test sets, containing 70% and 30% of data, respectively, and then fit linear regression in the two cases and compare the obtained results.

set.seed(1)  
  
# create train and test data  
sample <- sample(c(TRUE, FALSE), nrow(df\_toxicity), replace = TRUE, prob = c(0.7, 0.3))  
train <- df\_toxicity[sample, ]  
test <- df\_toxicity[!sample, ]  
train\_dummy <- df\_toxicity\_dummy[sample, ]  
test\_dummy <- df\_toxicity\_dummy[!sample, ]

# linear regression model with original dataset  
linear\_model <- lm(LC50 ~ . , data = train)  
summary(linear\_model)

##   
## Call:  
## lm(formula = LC50 ~ ., data = train)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -2.8627 -0.7752 -0.1066 0.5407 4.9911   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 2.610045 0.290472 8.986 < 2e-16 \*\*\*  
## TPSA 0.029013 0.003301 8.790 < 2e-16 \*\*\*  
## SAacc -0.014806 0.002531 -5.850 1.06e-08 \*\*\*  
## H050 0.036867 0.070795 0.521 0.602843   
## MLOGP 0.491479 0.073323 6.703 7.45e-11 \*\*\*  
## RDCHI 0.486313 0.160547 3.029 0.002621 \*\*   
## GATS1p -0.589207 0.176999 -3.329 0.000958 \*\*\*  
## nN -0.246422 0.060137 -4.098 5.11e-05 \*\*\*  
## C040 0.007219 0.092060 0.078 0.937535   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 1.191 on 378 degrees of freedom  
## Multiple R-squared: 0.5135, Adjusted R-squared: 0.5032   
## F-statistic: 49.87 on 8 and 378 DF, p-value: < 2.2e-16

prediction <- predict.lm(linear\_model, test)  
train\_error <- mean(linear\_model$residuals^2)  
test\_error <- mean((test$LC50 - prediction)^2)  
sprintf("Train error: %s", train\_error)

## [1] "Train error: 1.38535362970798"

sprintf("Test error: %s", test\_error)

## [1] "Test error: 1.53100346835813"

# linear regression model with dummy dataset  
linear\_model\_dummy <- lm(LC50 ~ ., data = train\_dummy)  
summary(linear\_model\_dummy)

##   
## Call:  
## lm(formula = LC50 ~ ., data = train\_dummy)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -3.1405 -0.7672 -0.0989 0.5426 5.2845   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 2.632220 0.296208 8.886 < 2e-16 \*\*\*  
## TPSA 0.023088 0.003317 6.961 1.50e-11 \*\*\*  
## SAacc -0.012457 0.002308 -5.397 1.20e-07 \*\*\*  
## H050 -0.016850 0.153343 -0.110 0.91256   
## MLOGP 0.536630 0.074574 7.196 3.36e-12 \*\*\*  
## RDCHI 0.362229 0.162016 2.236 0.02595 \*   
## GATS1p -0.528398 0.174199 -3.033 0.00259 \*\*   
## nN 0.174520 0.151643 1.151 0.25052   
## C040 -0.103912 0.163190 -0.637 0.52467   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 1.215 on 378 degrees of freedom  
## Multiple R-squared: 0.4933, Adjusted R-squared: 0.4826   
## F-statistic: 46 on 8 and 378 DF, p-value: < 2.2e-16

prediction\_dummy <- predict.lm(linear\_model\_dummy, test\_dummy)  
train\_error\_dummy <- mean(linear\_model\_dummy$residuals^2)  
test\_error\_dummy <- mean((test\_dummy$LC50 - prediction\_dummy)^2)  
sprintf("Train error dummy: %s", train\_error\_dummy)

## [1] "Train error dummy: 1.44282928244437"

sprintf("Test error dummy: %s", test\_error\_dummy)

## [1] "Test error dummy: 1.61148941476223"

In the two cases, the coefficients have different meanings. If for dummy variables they represent the effect that the presence of the atoms has on the target variable, when dealing with linear values, they represent the change in the dependent variable associated with a one-unit change in that continuous predictor while holding all other predictors constant.

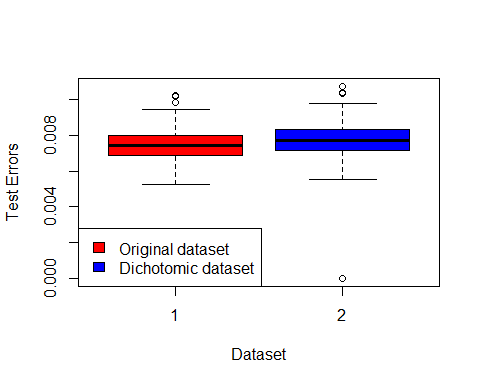
By comparing the summary of the two models, we can notice that the significance of the coefficients is similar. The two main differences can be seen for and coefficients. The former is significant for the first model but not for the second one. On the contrary, the latter has a smaller p-value in the first model than in the second one.

Focusing on the errors, we can easily notice that the train and the test errors associated to the original dataset are smaller than the ones related to the dichotomic dataset.

I then repeat the same procedure 200 times, such that each time I create a new training/test split and I apply linear regression to both the versions of the dataset. The idea behind the repetition is the following: it allows to assess the stability and robustness of the models across different data samples. This repeated process helps in accounting for the potential variability in model performance due to the randomness in the split of data. In addition, it also provides more reliable estimates of the average performance of the model.

# repeat the process 200 times   
repetitions <- 200  
train\_error <- 0  
test\_error <- 0  
test\_errors\_lr <- numeric(repetitions)  
train\_error\_dummy <- 0  
test\_error\_dummy <- 0  
test\_errors\_dummy\_lr <- numeric(repetitions)  
  
for (i in 1:repetitions){  
 sample <- sample(c(TRUE, FALSE), nrow(df\_toxicity), replace = TRUE, prob = c(0.7, 0.3))  
 train <- df\_toxicity[sample, ]  
 test <- df\_toxicity[!sample, ]  
 train\_dummy <- df\_toxicity\_dummy[sample, ]  
 test\_dummy <- df\_toxicity\_dummy[!sample, ]  
  
 linear\_model <- lm(LC50 ~ . , data = train)  
 prediction <- predict.lm(linear\_model, test)  
 train\_error <- train\_error + mean(linear\_model$residuals^2)/repetitions  
 test\_error <- test\_error + mean((test$LC50 - prediction)^2)/repetitions  
 test\_errors\_lr[i-1] <- mean((test$LC50 - prediction)^2)/repetitions  
  
 linear\_model\_dummy <- lm(LC50 ~ ., data = train\_dummy)  
 prediction\_dummy <- predict.lm(linear\_model\_dummy, test\_dummy)  
 train\_error\_dummy <- train\_error\_dummy + mean(linear\_model\_dummy$residuals^2)/repetitions  
 test\_error\_dummy <- test\_error\_dummy + mean((test\_dummy$LC50 -  
 prediction\_dummy)^2)/repetitions  
 test\_errors\_dummy\_lr[i-1] <- mean((test\_dummy$LC50 - prediction\_dummy)^2)/repetitions  
  
}

boxplot(list(test\_errors\_lr, test\_errors\_dummy\_lr), col = c("red", "blue"), xlab = "Dataset", ylab = "Test Errors")  
  
legend("bottomleft", legend = c("Original dataset", "Dichotomic dataset"), fill = c("red", "blue"))



sprintf("Train error: %s", train\_error)

## [1] "Train error: 1.41220194179306"

sprintf("Test error: %s", test\_error)

## [1] "Test error: 1.48743822132675"

sprintf("Train error dummy: %s", train\_error\_dummy)

## [1] "Train error dummy: 1.46293967145805"

sprintf("Test error dummy: %s", test\_error\_dummy)

## [1] "Test error dummy: 1.54502914579201"

As it is possible to see from both the plot and the results of the errors, in general, the use of the dataset with dicothomic count variables produces worse results. This is due to the fact that using dummy variables is an oversimplification and thus valuable information is ignored.

Due to this result, in the next steps I’m going to focus only on the original dataset, without taking into account the dichotomic dataset.

## Variable selection

The next step consists in comparing different variable selection procedures with different stopping criteria. In particular, I’m focusing on Backward Elimination and Forward Selection, both tested with AIC and BIC.

set.seed(1)  
  
sample <- sample(c(TRUE, FALSE), nrow(df\_toxicity), replace = TRUE, prob = c(0.7, 0.3))  
train <- df\_toxicity[sample, ]  
test <- df\_toxicity[!sample, ]  
  
full.model <- lm(LC50 ~ . , data = train)  
null.model <- lm(LC50 ~ 1, data = train)

# Backward elimination with BIC stopping criterion  
backward\_bic <- step(full.model, direction = "backward", k = log(nrow(train)), trace = FALSE)  
print("Backward elimination with BIC stopping criterion")

## [1] "Backward elimination with BIC stopping criterion"

backward\_bic

##   
## Call:  
## lm(formula = LC50 ~ TPSA + SAacc + MLOGP + RDCHI + GATS1p + nN,   
## data = train)  
##   
## Coefficients:  
## (Intercept) TPSA SAacc MLOGP RDCHI GATS1p   
## 2.66681 0.02875 -0.01405 0.48283 0.48380 -0.61811   
## nN   
## -0.24072

# Backward elimination with AIC stopping criterion  
backward\_aic <- step(full.model, direction = "backward", k = 2, trace = FALSE)  
print("Backward elimination with AIC stopping criterion")

## [1] "Backward elimination with AIC stopping criterion"

backward\_aic

##   
## Call:  
## lm(formula = LC50 ~ TPSA + SAacc + MLOGP + RDCHI + GATS1p + nN,   
## data = train)  
##   
## Coefficients:  
## (Intercept) TPSA SAacc MLOGP RDCHI GATS1p   
## 2.66681 0.02875 -0.01405 0.48283 0.48380 -0.61811   
## nN   
## -0.24072

# Forward selection with BIC stopping criterion  
forward\_bic <- step(null.model, direction = "forward", scope = formula(full.model), k = log(nrow(train)), trace = FALSE)  
print("Forward selection with BIC stopping criterion")

## [1] "Forward selection with BIC stopping criterion"

forward\_bic

##   
## Call:  
## lm(formula = LC50 ~ MLOGP + TPSA + SAacc + nN + GATS1p + RDCHI,   
## data = train)  
##   
## Coefficients:  
## (Intercept) MLOGP TPSA SAacc nN GATS1p   
## 2.66681 0.48283 0.02875 -0.01405 -0.24072 -0.61811   
## RDCHI   
## 0.48380

# Forward selection with AIC stopping criterion  
forward\_aic <- step(null.model, direction = "forward", scope = formula(full.model), k = 2, trace = FALSE)  
print("Forward selection with AIC stopping criterion")

## [1] "Forward selection with AIC stopping criterion"

forward\_aic

##   
## Call:  
## lm(formula = LC50 ~ MLOGP + TPSA + SAacc + nN + GATS1p + RDCHI,   
## data = train)  
##   
## Coefficients:  
## (Intercept) MLOGP TPSA SAacc nN GATS1p   
## 2.66681 0.48283 0.02875 -0.01405 -0.24072 -0.61811   
## RDCHI   
## 0.48380

As shown from the results of the 4 methods, the resulting models are the same in all the cases. All of them contain the same 6 variables (the coefficients are different, but the variables are the same): TPSA, SAacc, ML0GP, RDCHI, GATS1p and nN. In particular, focusing on the same variable selection technique, the results obtained with AIC and BIC are the same.

## Ridge Regression

Now, I study the performance of ridge regression, using both 5-fold cross-validation and a bootstrap procedure (considering 100 bootstrap iterations) . The idea is to find the optimal complexity parameter among a grid of lambda values.

library(glmnet)

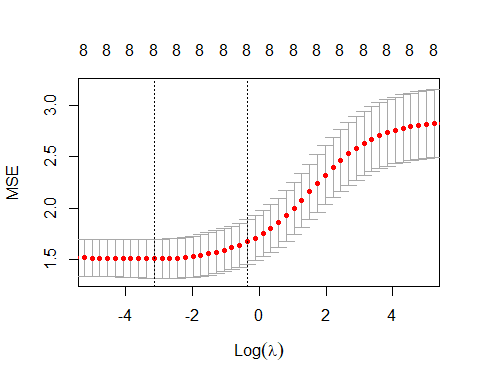
## Caricamento del pacchetto richiesto: Matrix

## Loaded glmnet 4.1-8

x <- as.matrix(subset(train, select = -LC50))  
y <- as.vector(train$LC50)  
# Create a sequence of lambda values to try  
lambda\_seq <- 10^seq(-5, 5, length = 100)  
  
# 5-fold cross-validation  
ridge\_cv\_model <- cv.glmnet(x, y, alpha = 0, lambda = lambda\_seq, nfolds = 5)  
  
print(ridge\_cv\_model)

##   
## Call: cv.glmnet(x = x, y = y, lambda = lambda\_seq, nfolds = 5, alpha = 0)   
##   
## Measure: Mean-Squared Error   
##   
## Lambda Index Measure SE Nonzero  
## min 0.0433 64 1.505 0.1881 8  
## 1se 0.7055 52 1.671 0.2175 8

# Plot the MSE with respect to lambda  
plot(ridge\_cv\_model, ylab = "MSE", xlim = c(-5, 5))



best\_lambda <- ridge\_cv\_model$lambda[which.min(ridge\_cv\_model$cvm)]  
sprintf("The best value of lambda is %s", best\_lambda)

## [1] "The best value of lambda is 0.0432876128108306"

So, I compute the train and test error for the model with the optimal value of lambda:

ridge\_best <- glmnet(x, y, alpha = 0, lambda = best\_lambda)  
ridge\_cv\_train\_err <- mean((predict(ridge\_best, as.matrix(subset(train, select = -LC50))) -  
 as.vector(train$LC50))^2)  
ridge\_cv\_test\_err <- mean((predict(ridge\_best, as.matrix(subset(test, select = -LC50))) -  
 as.vector(test$LC50))^2)  
sprintf("Ridge cv train error: %s", ridge\_cv\_train\_err)

## [1] "Ridge cv train error: 1.39907235413407"

sprintf("Ridge cv test error: %s", ridge\_cv\_test\_err)

## [1] "Ridge cv test error: 1.52200201505737"

library(boot)  
library(pracma)

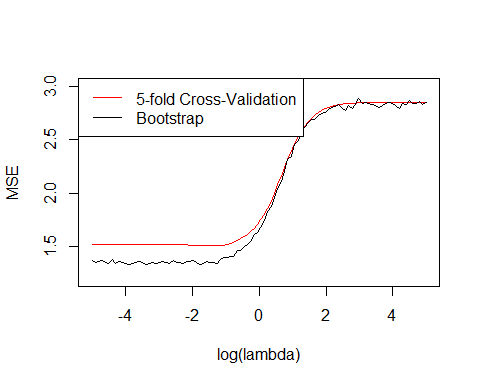
##   
## Caricamento pacchetto: 'pracma'

## Il seguente oggetto è mascherato da 'package:boot':  
##   
## logit

## I seguenti oggetti sono mascherati da 'package:Matrix':  
##   
## expm, lu, tril, triu

# Bootstrap  
ridge\_fit <- function(data, indices, lambda\_seq) {  
 data\_train <- data[indices, ]  
 X\_train <- as.matrix(subset(data\_train, select = -LC50))  
 y\_train <- as.vector(data\_train$LC50)  
 fit <- glmnet(X\_train, y\_train, alpha = 0, lambda = lambda\_seq)  
 y\_pred\_train <- predict(fit, newx = X\_train, s = lambda\_seq)  
 mse <- sum((y\_train - y\_pred\_train)^2) / length(y\_train)  
 return (mse)   
}  
  
mse\_boot <- c()  
for (i in 1:100){  
 lambda = lambda\_seq[i]  
 boot\_results <- boot(data = train, statistic = ridge\_fit, R = 100, lambda = lambda)  
 mse\_boot[i] = mean(boot\_results$t)  
}

plot(log10(ridge\_cv\_model$lambda), ridge\_cv\_model$cvm, ylab = "MSE", type = "l", xlab = "log(lambda)", col = "red", ylim = c(1.2, 3))  
lines(log10(lambda\_seq), mse\_boot, col = "black", type = "l", ylim = c(1, 3))  
legend("topleft", legend = c("5-fold Cross-Validation", "Bootstrap"), col = c("red", "black"), lty = 1)



best\_lambda\_boot <- lambda\_seq[which.min(mse\_boot)]  
sprintf("The best value of lambda is %s", best\_lambda\_boot)

## [1] "The best value of lambda is 0.0170735264747069"

So, I compute the train and test error for the model with the optimal value of lambda:

ridge\_best\_boot <- glmnet(x, y, alpha = 0, lambda = best\_lambda\_boot)  
ridge\_boot\_train\_err <- mean((predict(ridge\_best\_boot, as.matrix(subset(train, select = -LC50))) -  
 as.vector(train$LC50))^2)  
ridge\_boot\_test\_err <- mean((predict(ridge\_best\_boot, as.matrix(subset(test,   
 select = -LC50))) - as.vector(test$LC50))^2)  
sprintf("Ridge bootstrap train error: %s", ridge\_boot\_train\_err)

## [1] "Ridge bootstrap train error: 1.38827674305907"

sprintf("Ridge bootstrap test error: %s", ridge\_boot\_test\_err)

## [1] "Ridge bootstrap test error: 1.52260965421196"

The train and test errors resulting form the two methods are very similar to each other, especially the test error. This means that the two methods work similarly, as it is also shown in the figure above, which allows to compare the MSE w.r.t lambda in the two cases.

## Generalized Additive Model

Next, I consider possible non linear effects, fitting a generalized additive model in which the effects of the variables are fitted using smoothing splines. I decided to compare different levels of complexity for the smoothing splines. In particular, I focused on degrees of freedom from 3 to 10.

library(gam)

## Caricamento del pacchetto richiesto: splines

## Caricamento del pacchetto richiesto: foreach

## Loaded gam 1.22-2

degrees = 3:10  
  
train\_error\_gam <- zeros(length(degrees), 1)  
test\_error\_gam <- zeros(length(degrees), 1)  
  
# Fit GAM and compute the errors for each degree of freedom  
for (k in degrees){  
 gam\_model <- gam(LC50 ~ s(TPSA, df = k) + s(SAacc, df = k) + s(H050, df = k) +   
 s(MLOGP, df = k) + s(RDCHI, df = k) + s(GATS1p, df = k) +   
 s(nN, df = k) + s(C040, df = k), data = train)  
   
 train\_error\_gam[k-2] <- mean((predict(gam\_model, subset(train, select = -LC50)) -  
 train$LC50)^2)  
 test\_error\_gam[k-2] <- mean((predict(gam\_model, subset(test, select = -LC50)) -  
 test$LC50)^2)  
}

# Create a dataframe containing the errors for each degree of freedom  
df\_gam <- data.frame(degrees, train\_error\_gam, test\_error\_gam)  
names(df\_gam) <- c("Degrees of Freedom", "Train Error", "Test Error")  
df\_gam

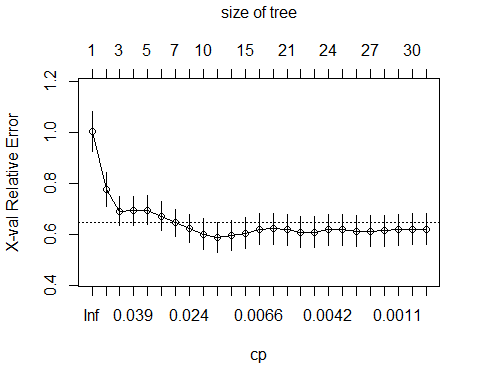
## Degrees of Freedom Train Error Test Error  
## 1 3 1.1986018 1.655150  
## 2 4 1.1278069 1.716900  
## 3 5 1.0721133 1.763671  
## 4 6 1.0303796 1.803536  
## 5 7 0.9951357 1.830276  
## 6 8 0.9616357 1.863732  
## 7 9 0.9296123 2.003940  
## 8 10 0.9005974 2.016821

I can immediately notice, from the above table, that the lowest test errors are related to small degrees of freedom. In particular, for the final comparison, I will use the result related to degree 3, which correspond to the lowest value of test error.

## Regression Tree

The next step is to fit a model using a regression tree. The procedure is the following:

library(rpart)  
  
set.seed(1)  
  
# Grow the full tree  
full\_tree <- rpart(LC50 ~ . , data = train, control = rpart.control(cp=0))  
plotcp(full\_tree)

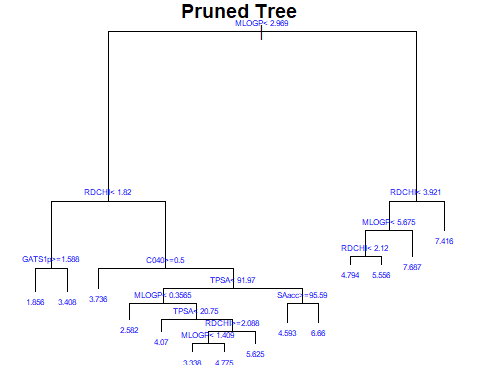


full\_tree$cptable

## CP nsplit rel error xerror xstd  
## 1 0.2389868389 0 1.0000000 1.0053201 0.07733352  
## 2 0.0951625548 1 0.7610132 0.7775182 0.06654780  
## 3 0.0415133719 2 0.6658506 0.6927572 0.05726826  
## 4 0.0368160036 3 0.6243372 0.6945627 0.05757412  
## 5 0.0321428113 4 0.5875212 0.6975016 0.05716882  
## 6 0.0281393375 5 0.5553784 0.6737111 0.05593765  
## 7 0.0244511912 6 0.5272391 0.6465647 0.05352552  
## 8 0.0230221121 8 0.4783367 0.6253484 0.05415741  
## 9 0.0117017714 9 0.4553146 0.6028115 0.05980978  
## 10 0.0116540354 10 0.4436128 0.5885493 0.05952466  
## 11 0.0086193946 13 0.4080555 0.5973646 0.06011120  
## 12 0.0067993671 14 0.3994361 0.6069817 0.06020988  
## 13 0.0064407811 17 0.3790380 0.6219058 0.06142105  
## 14 0.0063752009 18 0.3725972 0.6237564 0.06140710  
## 15 0.0052847317 20 0.3598468 0.6194814 0.06169323  
## 16 0.0051663059 21 0.3545621 0.6105035 0.06108054  
## 17 0.0047295393 22 0.3493958 0.6111279 0.06104307  
## 18 0.0036998853 23 0.3446662 0.6196626 0.06121635  
## 19 0.0030857471 24 0.3409664 0.6192291 0.06149363  
## 20 0.0021174022 25 0.3378806 0.6144837 0.06084740  
## 21 0.0015699559 26 0.3357632 0.6139374 0.06067609  
## 22 0.0011863536 27 0.3341933 0.6161329 0.06071866  
## 23 0.0010271511 28 0.3330069 0.6193909 0.06081903  
## 24 0.0003817938 29 0.3319797 0.6211185 0.06083289  
## 25 0.0000000000 30 0.3315980 0.6214007 0.06082918

The lowest error is given y a cp = 0.0116540354. So, I prune the tree according to this value. This is the obtained tree:

# Prune the tree  
pruned\_tree <- prune(full\_tree, cp = 0.0116540354)  
par(mar = c(1, 1, 1, 1))  
plot(pruned\_tree, main = "Pruned Tree")  
text(pruned\_tree, cex = 0.5, col = "blue")



Then, I compute the train and the test error for the pruned tree:

# Train and test error for the pruned tree  
  
train\_error\_tree <- mean((predict(pruned\_tree, subset(train, select = -LC50)) -  
 train$LC50)^2)  
test\_error\_tree <- mean((predict(pruned\_tree, subset(test, select = -LC50)) -   
 test$LC50)^2)  
sprintf("Train error Regression Tree: %s", train\_error\_tree)

## [1] "Train error Regression Tree: 1.16193347294163"

sprintf("Test error Regression Tree: %s", test\_error\_tree)

## [1] "Test error Regression Tree: 2.03975169136301"

## Comparison of all the methods

Now that I have applied all the methods, I can make a comparison between all of them, in order to draw some conclusions.

methods <- c("Linear Regression", "Dummy Linear Regression", "Variable selection",  
 "Ridge Cross-Validation", "Ridge Bootstrap", "GAM", "Regression Tree")  
train\_errors <- c(train\_error, train\_error\_dummy, mean(backward\_bic$residuals^2),  
 ridge\_cv\_train\_err, ridge\_boot\_train\_err, train\_error\_gam[1],  
 train\_error\_tree)  
test\_errors <- c(test\_error, test\_error\_dummy, mean((test$LC50 -   
 predict.lm(backward\_bic, test))^2),  
 ridge\_cv\_test\_err, ridge\_boot\_test\_err, test\_error\_gam[1],  
 test\_error\_tree)  
# Dataframe containing all the errors related to each method  
df\_comparison <- data.frame(methods, train\_errors, test\_errors)  
names(df\_comparison) <- c("Method", "Train Error", "Test Error")  
df\_comparison

## Method Train Error Test Error  
## 1 Linear Regression 1.412202 1.487438  
## 2 Dummy Linear Regression 1.462940 1.545029  
## 3 Variable selection 1.386374 1.533380  
## 4 Ridge Cross-Validation 1.399072 1.522002  
## 5 Ridge Bootstrap 1.388277 1.522610  
## 6 GAM 1.198602 1.655150  
## 7 Regression Tree 1.161933 2.039752

The above table shows the train and the test errors for all the different methods used by now. If we consider the best method as the one that minimizes the test error, then in this case the best one is Linear Regression. All the other methods give a larger test error.

We can also see that GAM and Regression Tree have a lower train error but a larger test error with respect to the other methods. This means that these techniques work very well with the train sample, but when dealing with new data (the test sample) their performance is worse, thus the test error is larger. This behavior suggest a problem of overfitting for these two methodologies.

# 

# Problem 2: Classification

The second problem of this work is based on the study of the Pima Indians Diabetes Database, containing information about 768 women from a population (Pima indians) that is particularly susceptible to diabetes. The response variable identifies two classes (“pos” and “neg”), depending whether a person involved in the study has developed the disease or not. The dataset contains also other 8 numeric variables:

* : number of pregnancies;
* : plasma glucose concentration at 2 h in an oral glucose tolerance test;
* : diastolic blood pressure (mm Hg);
* : triceps skin fold thickness (mm);
* : 2-h serum insulin (µU/mL);
* : body mass index (kg/m2);
* : diabetes pedigree function;
* : age (years);

library(mlbench)  
  
data(PimaIndiansDiabetes)  
diabetes\_df <- as.data.frame(PimaIndiansDiabetes)  
diabetes\_df["diabetes"] <- c(as.integer(diabetes\_df["diabetes"] == "pos"))  
head(diabetes\_df)

## pregnant glucose pressure triceps insulin mass pedigree age diabetes  
## 1 6 148 72 35 0 33.6 0.627 50 1  
## 2 1 85 66 29 0 26.6 0.351 31 0  
## 3 8 183 64 0 0 23.3 0.672 32 1  
## 4 1 89 66 23 94 28.1 0.167 21 0  
## 5 0 137 40 35 168 43.1 2.288 33 1  
## 6 5 116 74 0 0 25.6 0.201 30 0

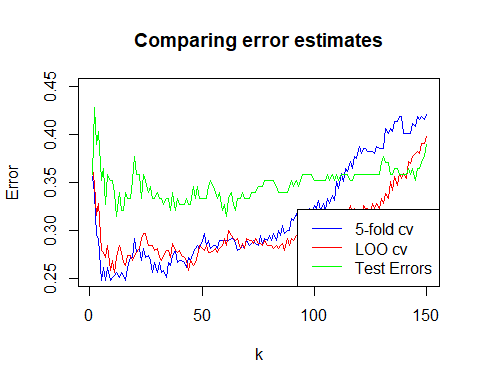
I randomly split the dataset into a training set (70% of the observations) and a test set (remaining 30% of the observations), such that the class distributions (i.e. the empirical distribution of diabetes) is similar in the two sets.

# Split in train and test sets (we want similar class distributions in the two sets)  
  
positives\_df <- diabetes\_df[diabetes\_df["diabetes"] == "1", ]  
negatives\_df <- diabetes\_df[diabetes\_df["diabetes"] == "0", ]  
  
set.seed(1)  
  
sample\_positives <- sample(c(TRUE, FALSE), nrow(positives\_df), replace = TRUE,   
 prob = c(0.7, 0.3))  
train\_positives <- positives\_df[sample\_positives, ]  
test\_positives <- positives\_df[!sample\_positives, ]  
  
sample\_negatives <- sample(c(TRUE, FALSE), nrow(negatives\_df), replace = TRUE,   
 prob = c(0.7, 0.3))  
train\_negatives <- negatives\_df[sample\_negatives, ]  
test\_negatives <- negatives\_df[!sample\_negatives, ]  
  
# Reunite train and test data  
train\_data <- rbind(train\_positives, train\_negatives)  
test\_data <- rbind(test\_positives, test\_negatives)  
  
# Randomly shuffle train and test sets  
train\_data <- train\_data[sample(1:nrow(train)), ]  
test\_data <- test\_data[sample(1:nrow(test)), ]

## kNN

The first step for this classification problem is to apply a k-Nearest Neighbor method. The idea is to compare the error estimates obtained using 5-fold cross-validation, Leave-One-Out cross-validation and the actual test error I would have obtained when fitting k-NN for the different values of k. The following plot shows the obtained results.

fold5\_errors <- knn.cv(x = as.matrix(subset(train\_data,select=-diabetes)), y = train\_data$diabetes,   
 nfolds = 5, stratified = TRUE, seed = TRUE, k = c(1:150),  
 dist.type = "euclidean", type = "C")  
loo\_errors <- knn.cv(x = as.matrix(subset(train\_data,select=-diabetes)), y = train\_data$diabetes,   
 nfolds = nrow(train), stratified = TRUE, seed = TRUE, k = c(1:150),  
 dist.type = "euclidean")  
predictions\_knn <- knn(xnew = as.matrix(subset(test\_data, select = -diabetes)), y = train\_data$diabetes,  
 x = as.matrix(subset(train\_data, select = -diabetes)), k = c(1:150), type = "C")  
test\_errors\_knn <- colMeans(abs(predictions\_knn - test\_data$diabetes))  
  
plot(1:150, 1 - fold5\_errors$crit, type = "l", ylim = c(0.25, 0.45), col = "blue", xlab = "k",   
 ylab = "Error")  
lines(1:150, 1 - loo\_errors$crit, col = "red")  
lines(1:150, test\_errors\_knn, col = "green")  
title("Comparing error estimates")  
legend("bottomright", legend = c("5-fold cv", "LOO cv", "Test Errors"), col =   
 c("blue", "red", "green"), lty = 1)



It can be immediately noticed that cross-validation (both with 5-folds and LOO) tends to underestimate the error, especially for small values of k.

I save the values of the lowest test errors related to each of the three methods.

k\_min\_5fold <- which.min(1 - fold5\_errors$crit)  
k\_min\_loo <- which.min(1 - loo\_errors$crit)  
k\_min\_knn <- which.min(test\_errors\_knn)  
  
sprintf("k with 5-fold cv: %s", k\_min\_5fold)

## [1] "k with 5-fold cv: 5"

sprintf("k with LOO cv: %s", k\_min\_loo)

## [1] "k with LOO cv: 11"

sprintf("k with kNN: %s", k\_min\_knn)

## [1] "k with kNN: 12"

test\_err\_5fold <- 1 - fold5\_errors$crit[k\_min\_5fold]  
test\_err\_loo <- 1- loo\_errors$crit[k\_min\_loo]  
test\_err\_knn <- test\_errors\_knn[k\_min\_knn]  
sprintf('Test error with 5-fold cv: %s', test\_err\_5fold)

## [1] "Test error with 5-fold cv: 0.248062015503876"

sprintf('Test error with LOO cv: %s', test\_err\_loo)

## [1] "Test error with LOO cv: 0.255813953488372"

sprintf('Actual test error with kNN: %s', test\_err\_knn)

## [1] "Actual test error with kNN: 0.314465408805031"

## Generalized Additive Model

Next, I fit a Generalized Additive Model with splines and use a variable selection method to find the best model.

model\_gam <- mgcv::gam(diabetes ~ s(pregnant, k = 5) + s(glucose, k = 5) + s(pressure, k = 5) +   
 s(triceps, k = 5) + s(insulin, k = 5) + s(mass, k = 5) + s(pedigree, k = 5) +   
 s(age, k = 5), data = train\_data, family = binomial())  
summary(model\_gam)

##   
## Family: binomial   
## Link function: logit   
##   
## Formula:  
## diabetes ~ s(pregnant, k = 5) + s(glucose, k = 5) + s(pressure,   
## k = 5) + s(triceps, k = 5) + s(insulin, k = 5) + s(mass,   
## k = 5) + s(pedigree, k = 5) + s(age, k = 5)  
##   
## Parametric coefficients:  
## Estimate Std. Error z value Pr(>|z|)  
## (Intercept) 0.1247 0.1896 0.658 0.511  
##   
## Approximate significance of smooth terms:  
## edf Ref.df Chi.sq p-value   
## s(pregnant) 1.000 1.000 0.033 0.85684   
## s(glucose) 3.868 3.988 41.321 < 2e-16 \*\*\*  
## s(pressure) 1.718 2.087 3.669 0.17192   
## s(triceps) 1.000 1.000 1.042 0.30735   
## s(insulin) 1.000 1.000 0.017 0.89497   
## s(mass) 3.843 3.979 18.075 0.00115 \*\*   
## s(pedigree) 1.000 1.000 8.107 0.00441 \*\*   
## s(age) 2.608 3.135 12.902 0.00568 \*\*   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## R-sq.(adj) = 0.434 Deviance explained = 40.3%  
## UBRE = -0.084607 Scale est. = 1 n = 387

The result shows that only 4 variables are significant, so I have to remove the non significant variables from the model:

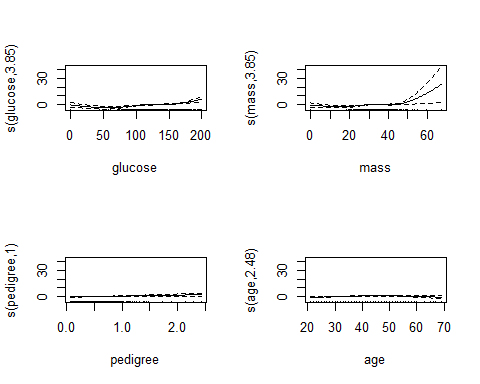
# Remove the non significant variables  
model\_gam <- mgcv::gam(diabetes ~ s(glucose, k = 5) + s(mass, k = 5) + s(pedigree, k = 5) +   
 s(age, k = 5), data = train\_data, family = binomial())  
summary(model\_gam)

##   
## Family: binomial   
## Link function: logit   
##   
## Formula:  
## diabetes ~ s(glucose, k = 5) + s(mass, k = 5) + s(pedigree, k = 5) +   
## s(age, k = 5)  
##   
## Parametric coefficients:  
## Estimate Std. Error z value Pr(>|z|)  
## (Intercept) 0.1091 0.1876 0.581 0.561  
##   
## Approximate significance of smooth terms:  
## edf Ref.df Chi.sq p-value   
## s(glucose) 3.846 3.984 46.809 < 2e-16 \*\*\*  
## s(mass) 3.847 3.980 16.087 0.00175 \*\*   
## s(pedigree) 1.000 1.000 6.771 0.00927 \*\*   
## s(age) 2.484 2.996 22.153 5.43e-05 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## R-sq.(adj) = 0.421 Deviance explained = 38.8%  
## UBRE = -0.089439 Scale est. = 1 n = 387

Now, all the variables are significant.

The following plots show the relationship between the 4 significant variables and the response.

par(mfrow = c(2,2))  
plot(model\_gam)



Finally, I compute the train error and the test error for GAM:

ypredict\_train <- predict(model\_gam, newdata = subset(train\_data, select = -diabetes), type = "response")  
ypredict\_train <- as.factor(as.integer(ypredict\_train >= 0.5))  
gam\_train\_error = mean((ypredict\_train != train\_data$diabetes))  
sprintf("Train error GAM: %s", gam\_train\_error)

## [1] "Train error GAM: 0.219638242894057"

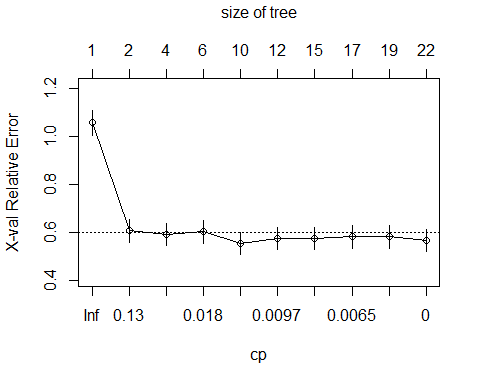
ypredict\_test <- predict(model\_gam, newdata = subset(test\_data, select = -diabetes), type = "response")  
ypredict\_test <- as.factor(as.integer(ypredict\_test >= 0.5))  
gam\_test\_error = mean((ypredict\_test != test\_data$diabetes))  
sprintf("Test error GAM: %s", gam\_test\_error)

## [1] "Test error GAM: 0.289308176100629"

## Classification tree

Now, I start to consider tree-based methods for classifications. The first model I fit is the classification tree, starting from a full tree and then pruning it.

set.seed(1)  
  
# Grow the tree  
classification\_tree <- rpart(diabetes ~ . , data = train\_data, control = rpart.control(cp=0),   
 method = "class")  
plotcp(classification\_tree)

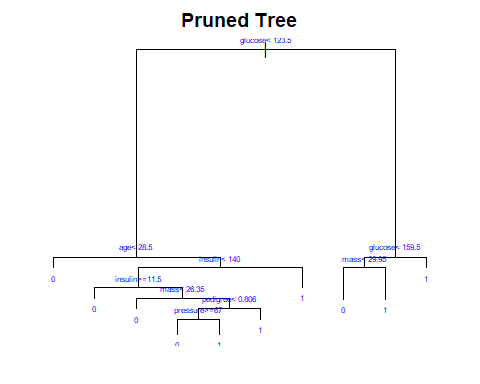


classification\_tree$cptable

## CP nsplit rel error xerror xstd  
## 1 0.455026455 0 1.0000000 1.0582011 0.05201379  
## 2 0.034391534 1 0.5449735 0.6084656 0.04756813  
## 3 0.021164021 3 0.4761905 0.5925926 0.04720175  
## 4 0.014550265 5 0.4338624 0.6031746 0.04744784  
## 5 0.010582011 9 0.3756614 0.5555556 0.04628091  
## 6 0.008818342 11 0.3544974 0.5767196 0.04681859  
## 7 0.007936508 14 0.3280423 0.5767196 0.04681859  
## 8 0.005291005 16 0.3121693 0.5820106 0.04694820  
## 9 0.003527337 18 0.3015873 0.5820106 0.04694820  
## 10 0.000000000 21 0.2910053 0.5661376 0.04655364

The minimum value of cp = 0.010582011, so I prune it at this value.

class\_tree <- prune(classification\_tree, cp = 0.010582011)  
par(mar = c(2, 2, 2, 2))  
plot(class\_tree, main = "Pruned Tree")  
text(class\_tree, cex = 0.5, col = "blue")



The corresponding train and test errors are:

tree\_train\_error <- mean(predict(class\_tree, subset(train\_data, select = -diabetes), type = "class")  
 != train\_data$diabetes)  
sprintf("Train error Classification Tree: %s", tree\_train\_error)

## [1] "Train error Classification Tree: 0.183462532299742"

tree\_test\_error <- mean(predict(class\_tree, subset(test\_data, select = -diabetes), type = "class")   
 != test\_data$diabetes)  
sprintf("Train error Classification Tree: %s", tree\_test\_error)

## [1] "Train error Classification Tree: 0.276729559748428"

## Bagged trees

I now consider another tree-based method, by constructing and evaluating an ensemble of bagged trees. The train and the test errors are then computed.

train\_data$diabetes <- as.factor(train\_data$diabetes)  
test\_data$diabetes <- as.factor(test\_data$diabetes)  
  
bagged\_tree <- ipred::bagging(diabetes ~ . , data = train\_data, nbagg = 100)  
bagged\_train\_error <- mean(predict(bagged\_tree, subset(train\_data, select = -diabetes), type = "class")  
 != train\_data$diabetes)  
sprintf("Train error Bagged Trees: %s", bagged\_train\_error)

## [1] "Train error Bagged Trees: 0.00258397932816537"

bagged\_test\_error <- mean(predict(bagged\_tree, subset(test\_data, select = -diabetes), type = "class")   
 != test\_data$diabetes)  
sprintf("Test error Bagged Trees: %s", bagged\_test\_error)

## [1] "Test error Bagged Trees: 0.283018867924528"

## Random forest

The last tree-based method is a Random Forest, for which I used 100 trees. The train and the test errors are then computed.

library(randomForest)

## randomForest 4.7-1.1

## Type rfNews() to see new features/changes/bug fixes.

##   
## Caricamento pacchetto: 'randomForest'

## Il seguente oggetto è mascherato da 'package:dplyr':  
##   
## combine

## Il seguente oggetto è mascherato da 'package:ggplot2':  
##   
## margin

random\_forest <- randomForest(diabetes ~ . , data = train\_data, ntree = 100)  
forest\_train\_error <- mean(predict(random\_forest, subset(train\_data, select = -diabetes), type = "response")  
 != train\_data$diabetes)  
sprintf("Train error Random Forest: %s", forest\_train\_error)

## [1] "Train error Random Forest: 0"

forest\_test\_error <- mean(predict(random\_forest, subset(test\_data, select = -diabetes), type = "response")   
 != test\_data$diabetes)  
sprintf("Test error Random Forest: %s", forest\_test\_error)

## [1] "Test error Random Forest: 0.320754716981132"

Comparing the results of the errors of the three tree-based methods, it is clear that the results are similar. The test errors of the first two techniques are very close, while for Random Forest the test error is a bit higher.

## Comparison of all the methods

After the fitting of all these classification methods, it is reasonable to make a comparison of all of them, in order to understand which performs better. Since I consider the best method as the one minimizing the test error, I just need the test error values to evaluate the performance of each technique.

methods <- c("kNN with 5-fold cv", "kNN with LOO cv", "GAM", "Classification Tree",  
 "Bagging", "Random Forest")  
  
test\_errors <- c(test\_err\_5fold, test\_err\_loo, gam\_test\_error, tree\_test\_error,  
 bagged\_test\_error, forest\_test\_error)  
df\_comparison <- data.frame(methods, test\_errors)  
names(df\_comparison) <- c("Method", "Test Error")  
df\_comparison

## Method Test Error  
## 1 kNN with 5-fold cv 0.2480620  
## 2 kNN with LOO cv 0.2558140  
## 3 GAM 0.2893082  
## 4 Classification Tree 0.2767296  
## 5 Bagging 0.2830189  
## 6 Random Forest 0.3207547

The above table shows that the lowest test error is obtained with a kNN fitted using 5-fold cross-validation. The result obtained with LOO cross-validation is very good as well. In general, all the test error values shown in the table are similar, so the different models have a similar performance.

## New Dataset

Looking more closely at the data, it has been noted that several values are impossible (e.g., a body mass index of 0). This is due to the fact that some of the observations are actually not zeros, but missing values. So, I now consider the correct dataset and compare the old results of all the methods implemented in the previous points to those obtained after removing all observations that contain missing values.

data(PimaIndiansDiabetes2)  
new\_diabetes\_df <- as.data.frame(PimaIndiansDiabetes2)  
new\_diabetes\_df["diabetes"] <- c(as.integer(new\_diabetes\_df["diabetes"] == "pos"))  
new\_diabetes\_df <- na.omit(new\_diabetes\_df) # remove all the rows containing missing values  
head(new\_diabetes\_df)

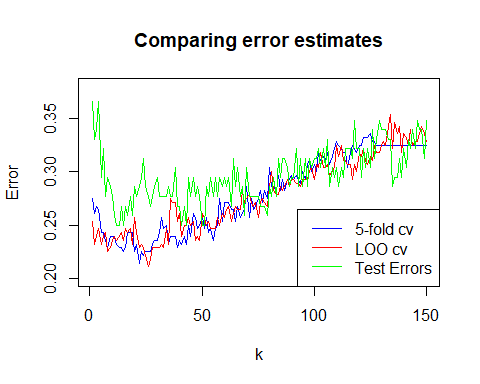
## pregnant glucose pressure triceps insulin mass pedigree age diabetes  
## 4 1 89 66 23 94 28.1 0.167 21 0  
## 5 0 137 40 35 168 43.1 2.288 33 1  
## 7 3 78 50 32 88 31.0 0.248 26 1  
## 9 2 197 70 45 543 30.5 0.158 53 1  
## 14 1 189 60 23 846 30.1 0.398 59 1  
## 15 5 166 72 19 175 25.8 0.587 51 1

After the elimination of all the rows containing missing values I can split the dataset in train and test sets, as I did before.

# Split in train and test sets (we want similar class distributions in the two sets)  
  
positives\_df <- new\_diabetes\_df[new\_diabetes\_df["diabetes"] == 1, ]  
negatives\_df <- new\_diabetes\_df[new\_diabetes\_df["diabetes"] == 0, ]  
  
set.seed(1)  
  
sample\_positives <- sample(c(TRUE, FALSE), nrow(positives\_df), replace = TRUE,   
 prob = c(0.7, 0.3))  
train\_positives <- positives\_df[sample\_positives, ]  
test\_positives <- positives\_df[!sample\_positives, ]  
  
sample\_negatives <- sample(c(TRUE, FALSE), nrow(negatives\_df), replace = TRUE,   
 prob = c(0.7, 0.3))  
train\_negatives <- negatives\_df[sample\_negatives, ]  
test\_negatives <- negatives\_df[!sample\_negatives, ]  
  
# Reunite train and test data  
train\_data <- rbind(train\_positives, train\_negatives)  
test\_data <- rbind(test\_positives, test\_negatives)  
  
# Randomly shuffle train and test sets  
train\_data <- na.omit(train\_data[sample(1:nrow(train)), ])  
test\_data <- na.omit(test\_data[sample(1:nrow(test)), ])

Now, I apply and fit all the methods, as I did with the complete dataset:

fold5\_errors <- knn.cv(x = as.matrix(subset(train\_data, select = -diabetes)), y = train\_data$diabetes,   
 nfolds = 5, stratified = TRUE, seed = TRUE, k = c(1:150),  
 dist.type = "euclidean", type = "C")  
loo\_errors <- knn.cv(x = as.matrix(subset(train\_data, select = -diabetes)), y = train\_data$diabetes,   
 nfolds = nrow(train\_data), stratified = TRUE, seed = TRUE, k = c(1:150),  
 dist.type = "euclidean", type = "C")  
predictions\_knn <- knn(xnew = as.matrix(subset(test\_data, select = -diabetes)), y = train\_data$diabetes,  
 x = as.matrix(subset(train\_data, select = -diabetes)), k = c(1:150), type = "C")  
test\_errors\_knn <- colMeans(abs(predictions\_knn - test\_data$diabetes))  
  
  
# plot the results  
plot(1:150, 1 - fold5\_errors$crit, type = "l", ylim = c(0.20, 0.38), col = "blue", xlab = "k",   
 ylab = "Error")  
lines(1:150, 1 - loo\_errors$crit, col = "red")  
lines(1:150, test\_errors\_knn, col = "green")  
title("Comparing error estimates")  
legend("bottomright", legend = c("5-fold cv", "LOO cv", "Test Errors"), col = c("blue", "red", "green"),   
 lty = 1)



From the graph, we can notice that now, for k equal to 70 onward, the use of cross-validation does not generate an underestimation of the error anymore: the three lines in the plot have similar values.

k\_min\_5fold <- which.min(1 - fold5\_errors$crit)  
k\_min\_loo <- which.min(1 - loo\_errors$crit)  
k\_min\_knn <- which.min(test\_errors\_knn)  
  
sprintf("k with 5-fold cv: %s", k\_min\_5fold)

## [1] "k with 5-fold cv: 22"

sprintf("k with LOO cv: %s", k\_min\_loo)

## [1] "k with LOO cv: 26"

sprintf("k with kNN: %s", k\_min\_knn)

## [1] "k with kNN: 12"

test\_err\_5fold\_2 <- 1 - fold5\_errors$crit[k\_min\_5fold]  
test\_err\_loo\_2 <- 1- loo\_errors$crit[k\_min\_loo]  
test\_err\_knn\_2 <- test\_errors\_knn[k\_min\_knn]  
sprintf('Test error with 5-fold cv: %s', test\_err\_5fold\_2)

## [1] "Test error with 5-fold cv: 0.214285714285714"

sprintf('Test error with LOO cv: %s', test\_err\_loo\_2)

## [1] "Test error with LOO cv: 0.210714285714286"

sprintf('Test error with kNN: %s', test\_err\_knn\_2)

## [1] "Test error with kNN: 0.25"

model\_gam <- mgcv::gam(diabetes ~ s(pregnant, k = 5) + s(glucose, k = 5) + s(pressure, k = 5) +   
 s(triceps, k = 5) + s(insulin, k = 5) + s(mass, k = 5) + s(pedigree, k = 5) +   
 s(age, k = 5), data = train\_data, family = binomial())  
summary(model\_gam)

##   
## Family: binomial   
## Link function: logit   
##   
## Formula:  
## diabetes ~ s(pregnant, k = 5) + s(glucose, k = 5) + s(pressure,   
## k = 5) + s(triceps, k = 5) + s(insulin, k = 5) + s(mass,   
## k = 5) + s(pedigree, k = 5) + s(age, k = 5)  
##   
## Parametric coefficients:  
## Estimate Std. Error z value Pr(>|z|)   
## (Intercept) -1.2115 0.1972 -6.144 8.06e-10 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Approximate significance of smooth terms:  
## edf Ref.df Chi.sq p-value   
## s(pregnant) 1.685 2.069 1.954 0.35756   
## s(glucose) 1.000 1.000 30.589 < 2e-16 \*\*\*  
## s(pressure) 1.220 1.410 0.622 0.70227   
## s(triceps) 1.000 1.000 0.193 0.66052   
## s(insulin) 3.768 3.968 10.614 0.02650 \*   
## s(mass) 1.000 1.000 2.961 0.08531 .   
## s(pedigree) 1.000 1.000 7.766 0.00532 \*\*   
## s(age) 1.930 2.411 4.807 0.11911   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## R-sq.(adj) = 0.435 Deviance explained = 39.6%  
## UBRE = -0.14068 Scale est. = 1 n = 280

# Remove the non significant variables  
model\_gam <- mgcv::gam(diabetes ~ glucose + pedigree + s(insulin, k = 5),   
 data = train\_data, family = binomial())  
summary(model\_gam)

##   
## Family: binomial   
## Link function: logit   
##   
## Formula:  
## diabetes ~ glucose + pedigree + s(insulin, k = 5)  
##   
## Parametric coefficients:  
## Estimate Std. Error z value Pr(>|z|)   
## (Intercept) -7.503205 0.985063 -7.617 2.60e-14 \*\*\*  
## glucose 0.044861 0.006979 6.428 1.29e-10 \*\*\*  
## pedigree 1.622236 0.496454 3.268 0.00108 \*\*   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Approximate significance of smooth terms:  
## edf Ref.df Chi.sq p-value   
## s(insulin) 3.81 3.979 12.97 0.00959 \*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## R-sq.(adj) = 0.364 Deviance explained = 32.1%  
## UBRE = -0.095318 Scale est. = 1 n = 280

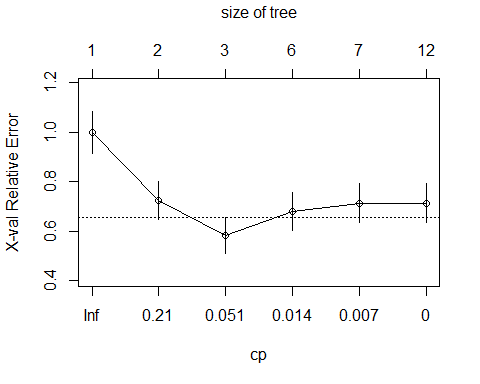
ypredict\_train <- predict(model\_gam, newdata = subset(train\_data, select = -diabetes), type = "response")  
ypredict\_train <- as.factor(as.integer(ypredict\_train >= 0.5))  
gam\_train\_error\_2 = mean((ypredict\_train != train\_data$diabetes))  
sprintf("Train error GAM: %s", gam\_train\_error\_2)

## [1] "Train error GAM: 0.185714285714286"

ypredict\_test <- predict(model\_gam, newdata = subset(test\_data, select = -diabetes), type = "response")  
ypredict\_test <- as.factor(as.integer(ypredict\_test >= 0.5))  
gam\_test\_error\_2 = mean((ypredict\_test != test\_data$diabetes))  
sprintf("Test error GAM: %s", gam\_test\_error\_2)

## [1] "Test error GAM: 0.303571428571429"

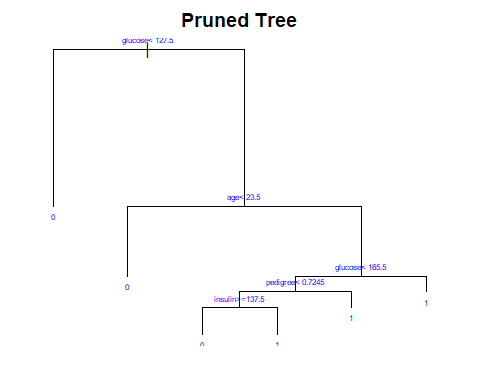
set.seed(2)  
  
# Grow the tree  
classification\_tree <- rpart(diabetes ~ . , data = train\_data, control = rpart.control(cp = 0),   
 method = "class")  
plotcp(classification\_tree)



classification\_tree$cptable

## CP nsplit rel error xerror xstd  
## 1 0.318681319 0 1.0000000 1.0000000 0.08612539  
## 2 0.142857143 1 0.6813187 0.7252747 0.07804740  
## 3 0.018315018 2 0.5384615 0.5824176 0.07203283  
## 4 0.010989011 5 0.4835165 0.6813187 0.07634906  
## 5 0.004395604 6 0.4725275 0.7142857 0.07763463  
## 6 0.000000000 11 0.4505495 0.7142857 0.07763463

class\_tree\_10 <- prune(classification\_tree, cp = 0.018315018)  
par(mar = c(2, 2, 2, 2))  
plot(class\_tree\_10, main = "Pruned Tree")  
text(class\_tree\_10, cex = 0.5, col = "blue")



tree\_train\_error\_2 <- mean(predict(class\_tree\_10, subset(train\_data, select = -diabetes), type = "class")  
 != train\_data$diabetes)  
sprintf("Train error Classification Tree: %s", tree\_train\_error\_2)

## [1] "Train error Classification Tree: 0.157142857142857"

tree\_test\_error\_2 <- mean(predict(class\_tree\_10, subset(test\_data, select = -diabetes), type = "class")   
 != test\_data$diabetes)  
sprintf("Train error Classification Tree: %s", tree\_test\_error\_2)

## [1] "Train error Classification Tree: 0.3125"

train\_data$diabetes <- as.factor(train\_data$diabetes)  
test\_data$diabetes <- as.factor(test\_data$diabetes)  
  
bagged\_tree <- ipred::bagging(diabetes ~ . , data = train\_data, nbagg = 100)  
bagged\_train\_error\_2 <- mean(predict(bagged\_tree, subset(train\_data, select = -diabetes), type = "class")  
 != train\_data$diabetes)  
sprintf("Train error Bagged Trees: %s", bagged\_train\_error\_2)

## [1] "Train error Bagged Trees: 0"

bagged\_test\_error\_2 <- mean(predict(bagged\_tree, subset(test\_data, select = -diabetes), type = "class")   
 != test\_data$diabetes)  
sprintf("Train error Bagged Trees: %s", bagged\_test\_error\_2)

## [1] "Train error Bagged Trees: 0.25"

random\_forest <- randomForest(diabetes ~ . , data = train\_data, ntree = 100, mtry=2)  
forest\_train\_error\_2 <- mean(predict(random\_forest, subset(train\_data, select = -diabetes), type = "response")  
 != train\_data$diabetes)  
sprintf("Train error Random Forest: %s", forest\_train\_error\_2)

## [1] "Train error Random Forest: 0"

forest\_test\_error\_2 <- mean(predict(random\_forest, subset(test\_data, select = -diabetes), type = "response")   
 != test\_data$diabetes)  
sprintf("Train error Random Forest: %s", forest\_test\_error\_2)

## [1] "Train error Random Forest: 0.205357142857143"

After thf fitting of all the methods, it is possible to make a comparison of the new results with the previous ones. The following table show what I obtained.

methods <- c("kNN with 5-fold cv", "kNN with LOO cv", "GAM", "Classification Tree",  
 "Bagging", "Random Forest")  
  
test\_errors <- c(test\_err\_5fold, test\_err\_loo, gam\_test\_error, tree\_test\_error,  
 bagged\_test\_error, forest\_test\_error)  
test\_errors\_2 <- c(test\_err\_5fold\_2, test\_err\_loo\_2, gam\_test\_error\_2, tree\_test\_error\_2,  
 bagged\_test\_error\_2, forest\_test\_error\_2)  
df\_comparison <- data.frame(methods, test\_errors, test\_errors\_2)  
names(df\_comparison) <- c("Method", "Test Error 1", "Test error 2")  
df\_comparison

## Method Test Error 1 Test error 2  
## 1 kNN with 5-fold cv 0.2480620 0.2142857  
## 2 kNN with LOO cv 0.2558140 0.2107143  
## 3 GAM 0.2893082 0.3035714  
## 4 Classification Tree 0.2767296 0.3125000  
## 5 Bagging 0.2830189 0.2500000  
## 6 Random Forest 0.3207547 0.2053571

The results related to the correct dataset, as expected, are better. This is due to the fact that the data I removed were not correct (since they were set to 0 even if they were missing values). Therefore, with the correct data, the performance of almost all the models (except fro GAM and Classification Tree) is improved. In this case, the best model is Random Forest. kNN with LOO and 5-fold cross validation still have a good performance, very close to the one of Random Forest. Bagging has an intermediate position, while GAM and classification trees are the worst ones.