Problem 1

Vivian De La O Mellum

10 november 2020

Problem 1

In 2014 Cassotti et al. published a study on aquatic toxicity. The motivation of the study was to predict the aquatic toxicitys effect towards Daphina Magna, a small planktonic crustacean that lives in fresh water. Aquatic toxicity was measured and stored in the variable Lc50, namely the concentration that causes death in half of the crustacean population over the test duration of 48 hours.

Importing libraries

```
rm(list = ls())

library(MASS)
library(caret)
library(Metrics)       # to compute RMSE
library(gam)           # gam models
library(rsample)       # data splitting
library(dplyr)         # data wrangling
library(rpart)         # performing regression trees
library(rpart.plot)    # plotting regression trees
library(ipred)         # bagging
library(caret)         # bagging
library(randomForest)  # random forest
library(party)         # aiding random forest
library(mlbench)       # aid
```

Loading the data

```
regression.mydata<- read.csv("https://archive.ics.uci.edu/ml/machine-learning-databases/00505/qsar_aqua

colnames(regression.mydata) <- c("TPSA","SAacc","H050","MLOGP","RDCHI", "GATS1p", "nN", "C040", "LC50")
col.names <- colnames(regression.mydata)

dim(regression.mydata)
```

```
## [1] 546   9
```

The dataset has 546 observations and nine explanatory variables, including its response. The response variable for this analysis is Lc50 which was previously stated to be the concentration that causes death in half of the crustacean population over the test duration of 48 hours.

Describing the data

Response variable

Lc50:

the concentration that causes death in half of the crustacean population over the test duration of 48 hours.

Predictors

For the prediction, the following 8 variables have been considered:

TPSA:

the topological polar surface area calculated by means of a contribution method that takes into account nitrogen, oxygen, potassium and sulphur;

SAacc:

the Van der Waals surface area (VSA) of atoms that are acceptors of hydrogen bonds;

H050:

the number of hydrogen atoms bonded to heteroatoms;

MLOGP:

expresses the lipophilicity of a molecule, this being the driving force of narcosis;

RDCHI:

a topological index that encodes information about molecular size and branching;

GATS1p:

information on molecular polarisability;

nN:

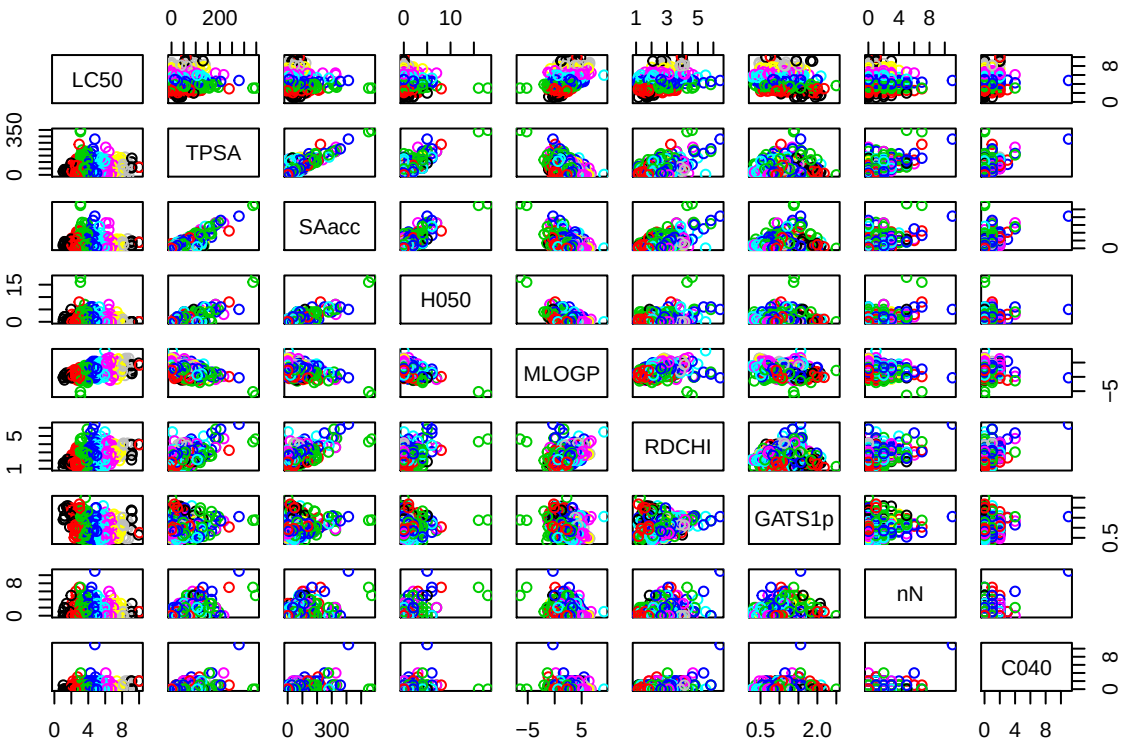
the number of nitrogen atoms present in the molecule.;

C040:

the number of carbon atoms of a certain type, including esters, carboxylic acids, thioesters, carbamic acids, nitriles, etc.

Visualizing the data

```
pairs(LC50~., data=regression.mydata, col=regression.mydata$LC50)
```



Splitting the data into train and test set

```
## 2/3 of the sample size
smp_size <- floor(0.66 * nrow(regression.mydata))

## Setting seed to make the partition reproducible
set.seed(123)
train_ind <- sample(seq_len(nrow(regression.mydata)), size = smp_size)

regression.train <- regression.mydata[train_ind, ]
regression.test <- regression.mydata[-train_ind, ]

## Creating a function for the RMSE
rmse_func <- function(model, data, response) {
  y_pred <- predict(model, data)
  y <- data[,response]
  return(sqrt(mean((y - y_pred)^2)))
}
```

Dichotomizing the training- and test data

```
## Dichotomizing training set
dichotomize_train <- data.frame(regression.train)
dichotomize_train$H050 <- regression.train$H050 > 0
dichotomize_train$nN <- regression.train$nN > 0
dichotomize_train$C040 <- regression.train$C040 > 0


## Dichotomizing test set
dichotomize_test <- data.frame(regression.train)
dichotomize_test$H050 <- regression.train$H050 > 0
dichotomize_test$nN <- regression.train$nN > 0
dichotomize_test$C040 <- regression.train$C040 > 0
```

1 Fittig the models

By inspection we observe that three of the eight variables are count variables; the count variables record the presence of specific atoms. The purpous of this fit is to determine if there is a difference between before and after recoding the countvariables to be dichotomous. The relevant count variables are; **H050, GATS1p and C040**.

Model; not Dichotomized

```
fit.regression = lm(LC50 ~., data = regression.train)

summary(fit.regression)
```

```
##
## Call:
## lm(formula = LC50 ~ ., data = regression.train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.8062 -0.7971 -0.0865  0.6331  4.7432
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2.482988   0.313675   7.916 3.26e-14 ***
## TPSA         0.031036   0.003377   9.190  < 2e-16 ***
## SAacc       -0.016078   0.002713  -5.927 7.36e-09 ***
## H050         0.044978   0.074891   0.601  0.54851
## MLOGP        0.449210   0.081305   5.525 6.43e-08 ***
## RDCHI        0.566456   0.173307   3.269  0.00119 **
## GATS1p      -0.481369   0.192153  -2.505  0.01269 *
## nN          -0.320288   0.062389  -5.134 4.72e-07 ***
## C040         0.014536   0.093870   0.155  0.87703
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.218 on 351 degrees of freedom
## Multiple R-squared:  0.5035, Adjusted R-squared:  0.4922
## F-statistic: 44.49 on 8 and 351 DF,  p-value: < 2.2e-16
```

Model; Dichotomized

```
fit.dichotomized = lm(LC50 ~., data = dichotomize_test)

summary(fit.dichotomized)
```

```
##
## Call:
## lm(formula = LC50 ~ ., data = dichotomize_test)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.5575 -0.8083 -0.1343  0.6705  4.9257
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2.766795   0.326715   8.469 6.89e-16 ***
## TPSA         0.025269   0.003433   7.362 1.30e-12 ***
## SAacc       -0.013156   0.002349  -5.600 4.32e-08 ***
## H050TRUE    -0.156512   0.161134  -0.971   0.3321
## MLOGP        0.470390   0.082843   5.678 2.86e-08 ***
## RDCHI        0.417095   0.177713   2.347   0.0195 *
## GATS1p      -0.474053   0.192208  -2.466   0.0141 *
## nNTRUE      -0.049435   0.158367  -0.312   0.7551
## C040TRUE    -0.211904   0.176732  -1.199   0.2313
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.26 on 351 degrees of freedom
## Multiple R-squared:  0.4691, Adjusted R-squared:  0.457
## F-statistic: 38.77 on 8 and 351 DF,  p-value: < 2.2e-16
```

Adjusted R-squared Adjusted R-squared:0.4922 before dichotomizing. Adjusted R-squared:0.457 after dichotomizing. We also observe that after dichotomizig the significance levels of the different regressors shrink.

Computing the training and test errors Looking at two measures to assess how well the models (before and after dichotomizing the data) perform given the training- and test RMSE. The RMSE is def as

$$RMSE_{data} = RMSE(\hat{f}, data) = \sqrt{\frac{1}{n_{data}} \sum_{i=1}^{n_{data}} (y_i - \hat{f}(x_i))^2}$$

where $data = \{train, test\}$ and n_{data} is the number of observations given the data.

```
## Train error
rmse(actual = regression.train$LC50, predicted = predict(fit.regression, regression.train))
```

```
## [1] 1.202939
```

```
## Test error
rmse(actual = regression.test$LC50, predicted = predict(fit.regression, regression.test))
```

```
## [1] 1.199307
```

```
## Training error; dichotomized
rmse(actual = dichotomize_train$LC50, predicted = predict(fit.dichotomized, dichotomize_train))
```

```
## [1] 1.243876
```

```
## Train error; dichotomized
rmse(actual = dichotomize_test$LC50, predicted = predict(fit.dichotomized, dichotomize_test))
```

```
## [1] 1.243876
```

Results of the training and test error before and after dichotomizing

	$Notdichotomized$	$dichotomized$
$RMSE_{train}$	1.199307	1.131458
$RMSE_{test}$	1.202939	1.28392

We observe that the train RMSE is lower for the categorical effect compared to the linear effect, but this is just used as a guide thorugh the model selection and we are only intersted in knowing the relationship between the test RMSE. We observe that there is a sligtly higher RMSE for the test and could be due to a non-linear relationship and that the data is not evenly distributed across the categorical levels. The reasonalbe thing would be to further study the levels and see if we could uncover some linear/non-linear relationship, or adjust the assumptions we have of the data for a better model-fit.

2. Repeating the procedure 200 times

```
n <- 200
test_errors <- data.frame(
  Original = numeric(n),
  Dichotomized = numeric(n)
)

train_size <- floor(nrow(regression.mydata)*2/3)
data_dichotomized <- data.frame(regression.mydata)

for (i in 1:n) {
  # repeating the procedure of point 1 for 200 times; each time changing the split in training & test d
  train_i <- sample(seq_len(nrow(regression.mydata)), size = train_size)
  data_train <- regression.mydata[train_i,]
  data_test <- regression.mydata[-train_i,]
  dichotomize_train_new <- data_dichotomized[train_i,]
  dichotomize_test_new <- data_dichotomized[-train_i,]

  # Fit lm on the original data
  fit.regression <- lm(LC50 ~ ., data = data_train)
  test_errors[i, 'Original'] <- rmse_func(fit.regression, data_test, "LC50")
```

```
  # Fit lm on data with dichotomized count variables
  fit.regression.dichotomized <- lm(LC50 ~ ., data = dichotomize_train_new)
  test_errors[i, 'Dichotomized'] <- rmse_func(fit.regression.dichotomized, dichotomize_test_new, "LC50")
}

# Mean test errors
colMeans(test_errors)
```

```
##     Original Dichotomized
##     1.218164     1.218164
```

Compared to the previous results the RMSE for this is higher and could be a result of being the average of several splits as each split is different and therefor assigning the observations differently. If we refer to the colorful figure provided in the lecturenotes with the true and realized value we know that when the realized value is itterated towards ∞ the expected value of the relaized value goes towards the true value of y; hence this being a preview of how it will behave in the long run (if we did a lot more than 200 iterations). The difference in results when dichotomizing the data vs using the original could be due to an underlying effect such as the data not being uniformly spred accross the categories, hence taking into account the importance of each level as to just the precence of atoms.

3. Different variable selection

Comparing the results of different variable selection prosedures using the first training/test split with different stopping criteria.

```
### Fit the full model ###
full.model = lm(LC50 ~., data = regression.train)

### Fit the null model ###
null.model = lm(LC50 ~1, data = regression.train)

### Stepwise regression ###


## AIC ##
model.backward.aic <- step(object = full.model, scope = null.model, direction = 'backward', k=2)
model.stepback.aic <- step(object = full.model, scope = null.model, direction = 'both', k=2)
model.forward.aic <- step(object = null.model, scope = full.model$terms, direction = 'forward', k=2)
model.stepwise.aic <- step(object = null.model, scope = full.model$terms, direction = 'both', k=2)

## BIC ##
model.backward.aic <- step(object = full.model, scope = null.model, direction = 'backward', k=log(nrow(
model.stepback.aic <- step(object = full.model, scope = null.model, direction = 'both',  k=log(nrow(reg
model.forward.aic <- step(object = null.model, scope = full.model$terms, direction = 'forward',  k=log(
model.stepwise.aic <- step(object = null.model, scope = full.model$terms, direction = 'both',  k=log(nr
```

To save space I reccorded only the relevant best models from each selection.

```
### Best from model.backward.aic ###
## Step: AIC=147.41
## LC50 ~ TPSA + SAacc + MLOGP + RDCHI + GATS1p + nN
```

```
##
## Df Sum of Sq RSS AIC
## <none> 521.49 147.41
## - GATS1p 1 11.634 533.12 153.35
## - RDCHI 1 16.229 537.72 156.44
## - nN 1 39.610 561.10 171.77
## - MLOGP 1 45.912 567.40 175.79
## - SAacc 1 77.575 599.06 195.34
## - TPSA 1 126.569 648.06 223.64


### Best: model.stepback.aic ###
## Step: AIC=147.41
## LC50 ~ TPSA + SAacc + MLOGP + RDCHI + GATS1p + nN
##
## Df Sum of Sq RSS AIC
## <none> 521.49 147.41
## - GATS1p 1 11.634 533.12 153.35
## - RDCHI 1 16.229 537.72 156.44
## - nN 1 39.610 561.10 171.77
## - MLOGP 1 45.912 567.40 175.79
## - SAacc 1 77.575 599.06 195.34
## - TPSA 1 126.569 648.06 223.64

### Best: model.forward.aic ###
## Step: AIC=147.41
## LC50 ~ MLOGP + TPSA + SAacc + nN + RDCHI + GATS1p
##
## Df Sum of Sq RSS AIC
## <none> 521.49 147.41
## + H050 1 0.51171 520.98 149.06
## + C040 1 0.01197 521.48 149.40


### Best: model.stepwise.aic ###
## Step: AIC=147.41
## LC50 ~ MLOGP + TPSA + SAacc + nN + RDCHI + GATS1p
##
## Df Sum of Sq RSS AIC
## <none> 521.49 147.41
## + H050 1 0.512 520.98 149.06
## + C040 1 0.012 521.48 149.40
## - GATS1p 1 11.634 533.12 153.35
## - RDCHI 1 16.229 537.72 156.44
## - nN 1 39.610 561.10 171.77
## - MLOGP 1 45.912 567.40 175.79
## - SAacc 1 77.575 599.06 195.34
## - TPSA 1 126.569 648.06 223.64

### Best: model.backward.bic ###
## Step: AIC=174.61
## LC50 ~ TPSA + SAacc + MLOGP + RDCHI + GATS1p + nN
##
```

```
## Df Sum of Sq RSS AIC
## <none> 521.49 174.61
## - GATS1p 1 11.634 533.12 176.67
##
## - RDCHI 1 16.229 537.72 179.76
## - nN 1 39.610 561.10 195.08
## - MLOGP 1 45.912 567.40 199.10
## - SAacc 1 77.575 599.06 218.65
## - TPSA 1 126.569 648.06 246.95

### Best.model.stepback.bic ###
## Step: AIC=174.61
## LC50 ~ MLOGP + TPSA + SAacc + nN + RDCHI + GATS1p
##
## Df Sum of Sq RSS AIC
## <none> 521.49 174.61
## + H050 1 0.51171 520.98 180.15
## + C040 1 0.01197 521.48 180.49


### Best: model.stepback.bic ###
## Step: AIC=174.61
## LC50 ~ TPSA + SAacc + MLOGP + RDCHI + GATS1p + nN
##
## Df Sum of Sq RSS AIC
## <none> 521.49 174.61
## - GATS1p 1 11.634 533.12 176.67
## - RDCHI 1 16.229 537.72 179.76
## - nN 1 39.610 561.10 195.08
## - MLOGP 1 45.912 567.40 199.10
## - SAacc 1 77.575 599.06 218.65
## - TPSA 1 126.569 648.06 246.95


### Best: model.forward.bic ###
## Step: AIC=174.61
## LC50 ~ MLOGP + TPSA + SAacc + nN + RDCHI + GATS1p
##
## Df Sum of Sq RSS AIC
## <none> 521.49 174.61
## + H050 1 0.51171 520.98 180.15
## + C040 1 0.01197 521.48 180.49


### Best: model.stepwise.bic ###
## Step: AIC=174.61
## LC50 ~ MLOGP + TPSA + SAacc + nN + RDCHI + GATS1p
##
##
## Df Sum of Sq RSS AIC
## <none> 521.49 174.61
## - GATS1p 1 11.634 533.12 176.67
## - RDCHI 1 16.229 537.72 179.76
```

```
## + H050 1 0.512 520.98 180.15
## + C040 1 0.012 521.48 180.49
## - nN 1 39.610 561.10 195.08
## - MLOGP 1 45.912 567.40 199.10
## - SAacc 1 77.575 599.06 218.65
## - TPSA 1 126.569 648.06 246.95
```

We observe that for backward, stepback, forward and stepwise all lead to the same AIC= 147.41 or BIC=174.61 value.

4. Ridge regression

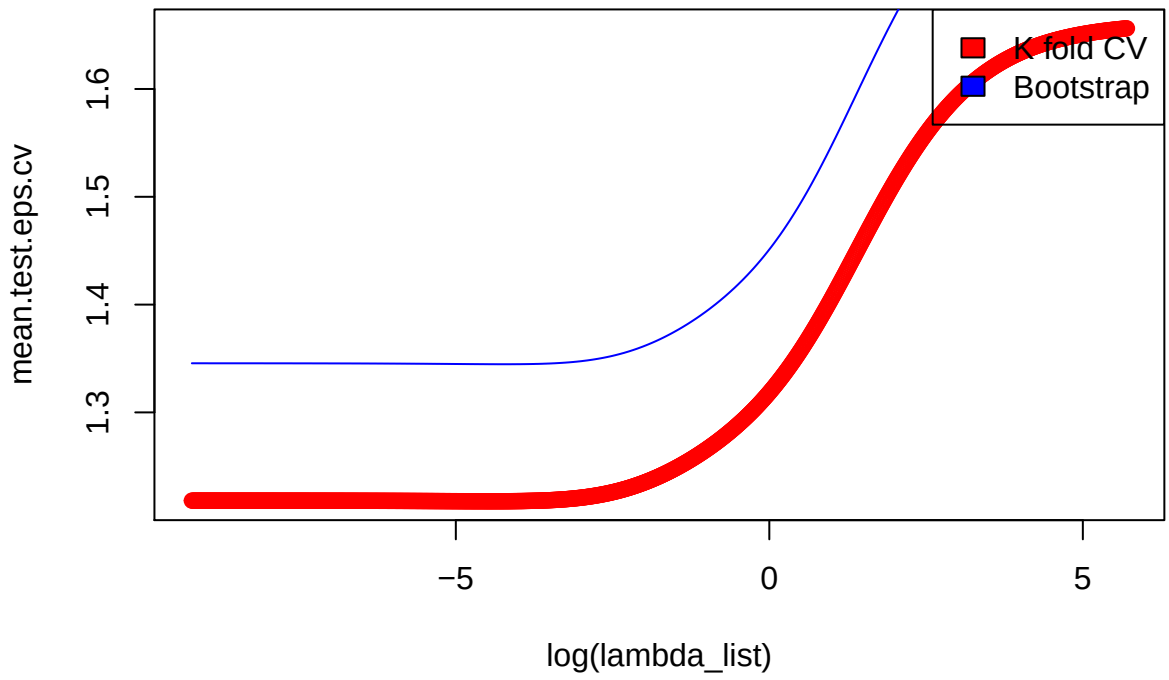
```
library(glmnet)
```

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

```
## Loaded glmnet 4.0
```

```
rmse_func_ridge <- function(X_train, Y_train, X_test, Y_test, lambdas) {
  fit <- glmnet(X_train, Y_train, alpha = 0, lambda = lambdas)
  Y_pred <- predict(fit, newx = X_test)
  return(sqrt(colMeans((Y_test - Y_pred)^2)))
}


# CV; function with k-folds
get.mean.cv.eps <- function(X, y, n_folds, error_function) {
  mean.test.eps <- numeric(length = n_lambdas)
  n_rows <- nrow(X)
  ind_shuffled <- sample(n_rows)
  for (i in 1:n_folds) {
    fold_i <- (floor(n_rows/n_folds*(i - 1)) + 1):floor(n_rows/n_folds*i)
    test_i <- ind_shuffled[fold_i]

    X_train <- X[-test_i, ]
    X_test <- X[test_i, ]
    y_train <- y[-test_i]
    y_test <- y[test_i]

    test_error <- error_function(X_train, y_train, X_test, y_test, lambda_list)
    mean.test.eps <- mean.test.eps + test_error/n_folds
  }
  return(mean.test.eps)
}


# Bootstrapping; function with k-folds
get.mean.boot.eps <- function(X, y, n_draws, error_function) {
  mean.test.eps <- numeric(length = n_lambdas)
  n_rows <- nrow(X)
```

```
  for (i in 1:n_draws) {
    boot.sample.i <- sample(n_rows, replace = TRUE)
    sample.i <- (1:n_rows)[-boot.sample.i]

    X_train <- X[boot.sample.i, ]
    X_test <- X[sample.i, ]
    y_train <- y[boot.sample.i]
    y_test <- y[sample.i]

    test.error <- error_function(X_train, y_train, X_test, y_test, lambda_list)
    mean.test.eps <- mean.test.eps.cv + test.error/n_draws
  }
  return(mean.test.eps)
}

# Splitting data
X_data <- as.matrix(regression.mydata[, col.names != "LC50"])
y_data <- regression.mydata[, "LC50"]

# lambda sequence from 0.1 to 100 = 100 values
n_lambdas <- 1000
lambda_list <- exp(seq(log(300), log(0.0001), length.out = n_lambdas))

# Creating plot
mean.test.eps.cv <- get.mean.cv.eps(X = X_data, y = y_data,
                                    n_folds = 10,
                                    error_function = rmse_func_ridge)


mean.test.eps.boot <- get.mean.boot.eps(X = X_data, y = y_data,
                                     n_draws = 10,
                                     error_function = rmse_func_ridge)



plot_data <- data.frame(Lambda = lambda_list,cv = mean.test.eps.cv)


plot(log(lambda_list), mean.test.eps.cv,  col = "red")
lines(log(lambda_list), mean.test.eps.boot, col = "blue")
legend("topright", c("K fold CV", "Bootstrap"), fill=c("red", "blue"))
```



The plot shows the estimated ridge regression model using the bootstrap and k-fold cross validation. We observe that the rmse computed by the bootstrap was systematically less than the rmse computed by the cross validation; but interestingly both rmse give the same functional form when plotted.

5. Non linear effect; gam

Fitting a generalized additive model with splines and using a variable selection to find the best model.

```
### Preparing variable names ###
LC50 <- regression.train$LC50

TPSA <- regression.train$TPSA
SAacc <- regression.train$SAacc
H050 <- as.numeric(regression.train$H050)
MLOGP <- regression.train$MLOGP
RDCHI <- regression.train$RDCHI
GATS1p <- regression.train$GATS1p
nN <- as.numeric(regression.train$nN)
C040 <- as.numeric(regression.train$C040)


### Considering splines up to five degrees of freedom ###
scope_list = list(
  "TPSA" = ~1 + TPSA + s(TPSA, df=2) + s(TPSA, df=3) + s(TPSA, df =4) + s(TPSA, df=5),
  "SAacc" = ~1 + SAacc + s(SAacc, df=2) + s(SAacc, df=3) + s(SAacc, df=4) + s(SAacc, df=5),
```

```
  "H050" = ~1 + H050 + s(H050, df=2) + s(H050, df=3) + s(H050, df=4) + s(H050, df=5),
  "MLOGP" = ~1 + MLOGP + s(MLOGP, df=2) + s(MLOGP, df=3) + s(MLOGP, df=4) + s(MLOGP, df=5),
  "RDCHI" = ~1 + RDCHI + s(RDCHI, df=2) + s(RDCHI, df=3) + s(RDCHI, df=4) + s(RDCHI, df=5),
  "GATS1p" = ~1 + GATS1p + s(GATS1p, df=2) + s(GATS1p, df=3) + s(GATS1p, df=4) + s(GATS1p, df=5),
  "nN" = ~1 + nN + s(nN, df=2) + s(nN, df=3) + s(nN, df=4) + s(nN, df=5),
  "C040" = ~1 + C040 + s(C040, df=2) + s(C040, df=3) + s(C040, df=4) + s(C040, df=5)
)


### forward stepwise selection ###
start_model = gam(LC50~ TPSA + SAacc + H050 + MLOGP + RDCHI + GATS1p + nN + C040, data = regression.tra
step.Gam(start_model, scope_list, direction = "both")
```

```
## Start:  LC50 ~ TPSA + SAacc + H050 + MLOGP + RDCHI + GATS1p + nN + C040; AIC= 1174.669
## Step:1 LC50 ~ TPSA + SAacc + H050 + s(MLOGP, df = 2) + RDCHI + GATS1p +      nN + C040 ; AIC= 1170.9
## Step:2 LC50 ~ TPSA + SAacc + H050 + s(MLOGP, df = 3) + RDCHI + GATS1p +      nN + C040 ; AIC= 1168.0
## Step:3 LC50 ~ TPSA + SAacc + H050 + s(MLOGP, df = 3) + RDCHI + GATS1p +      s(nN, df = 2) + C040 ;
## Step:4 LC50 ~ TPSA + SAacc + H050 + s(MLOGP, df = 3) + RDCHI + GATS1p +      s(nN, df = 3) + C040 ;
## Step:5 LC50 ~ TPSA + SAacc + H050 + s(MLOGP, df = 4) + RDCHI + GATS1p +      s(nN, df = 3) + C040 ;
## Step:6 LC50 ~ TPSA + SAacc + H050 + s(MLOGP, df = 4) + RDCHI + GATS1p +      s(nN, df = 3) ; AIC= 115
## Step:7 LC50 ~ TPSA + SAacc + s(MLOGP, df = 4) + RDCHI + GATS1p + s(nN,      df = 3) ; AIC= 1157.135
## Step:8 LC50 ~ TPSA + SAacc + s(MLOGP, df = 4) + RDCHI + GATS1p + s(nN,      df = 4) ; AIC= 1155.755
## Step:9 LC50 ~ TPSA + SAacc + s(MLOGP, df = 5) + RDCHI + GATS1p + s(nN,      df = 4) ; AIC= 1154.866
## Step:10 LC50 ~ TPSA + SAacc + s(MLOGP, df = 5) + RDCHI + s(GATS1p, df = 2) +      s(nN, df = 4) ; AI
## Step:11 LC50 ~ TPSA + SAacc + s(MLOGP, df = 5) + RDCHI + s(GATS1p, df = 2) +      s(nN, df = 5) ; AI
```

```
## Call:
## gam(formula = LC50 ~ TPSA + SAacc + s(MLOGP, df = 5) + RDCHI +
##     s(GATS1p, df = 2) + s(nN, df = 5), data = regression.train,
##     trace = FALSE)
##
## Degrees of Freedom: 359 total; 343.9999 Residual
## Residual Deviance: 473.8129
```

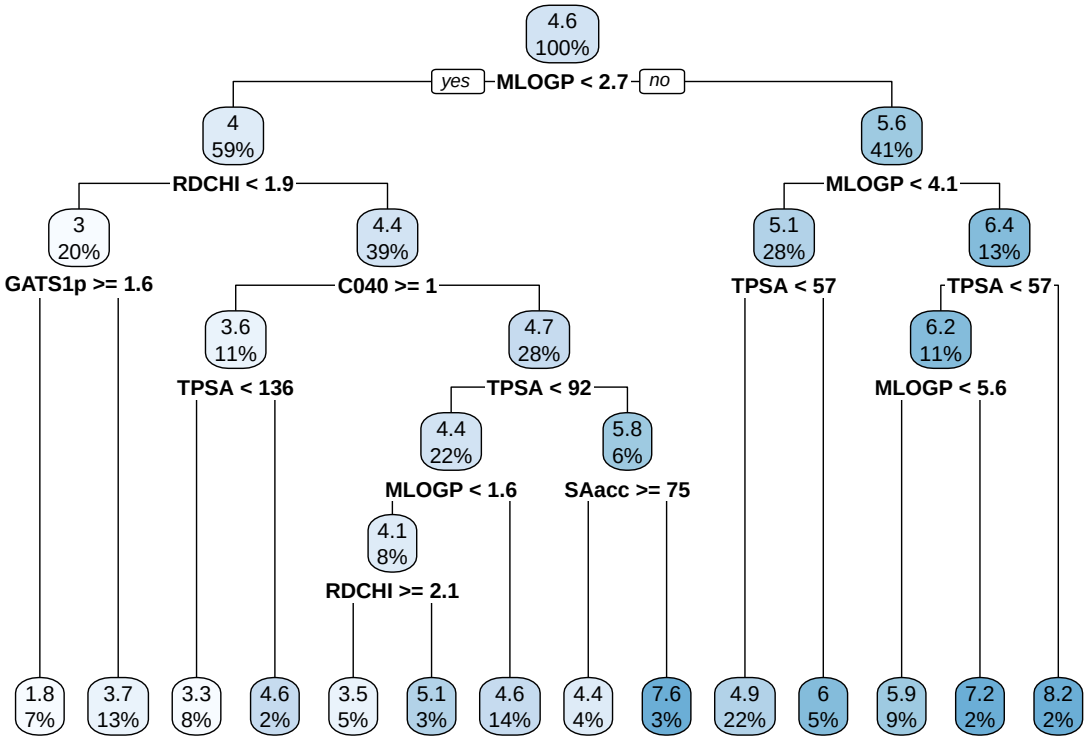
We observe that the best model is given by; LC50 ~ TPSA + SAacc + s(MLOGP, df = 5) + RDCHI + s(GATS1p, df = 2) + s(nN, df = 5).

All models above have empoyed different degrees of complexity when modeling the effects of the independet variables on the dependent variable. The splines adding to the model complexity are given by a generalization of polynomials that give more room for flexibility, hence if we give the model a high degree of complexity the bias will decrease and the variance increase from realization to realisation of the model (ref. fig in STK-IN4300 notes).

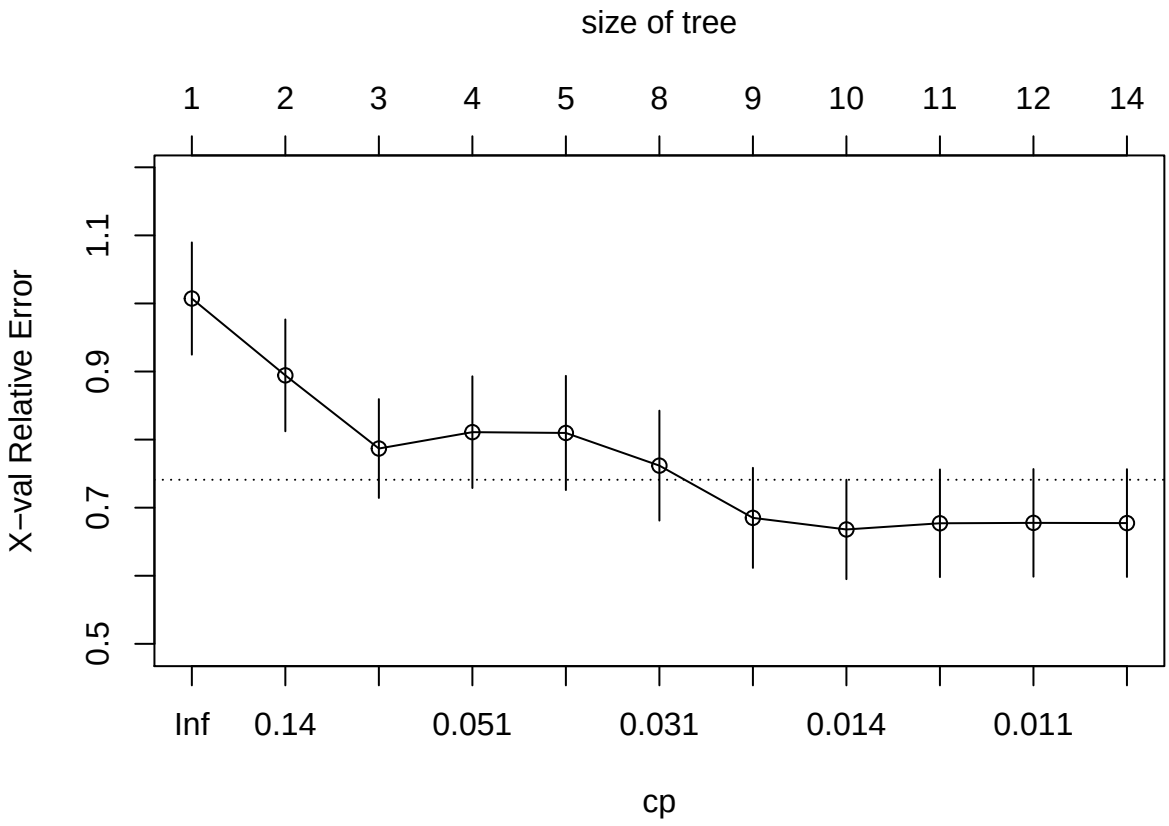
6. Regression tree

Fitting a model using a regression tree.

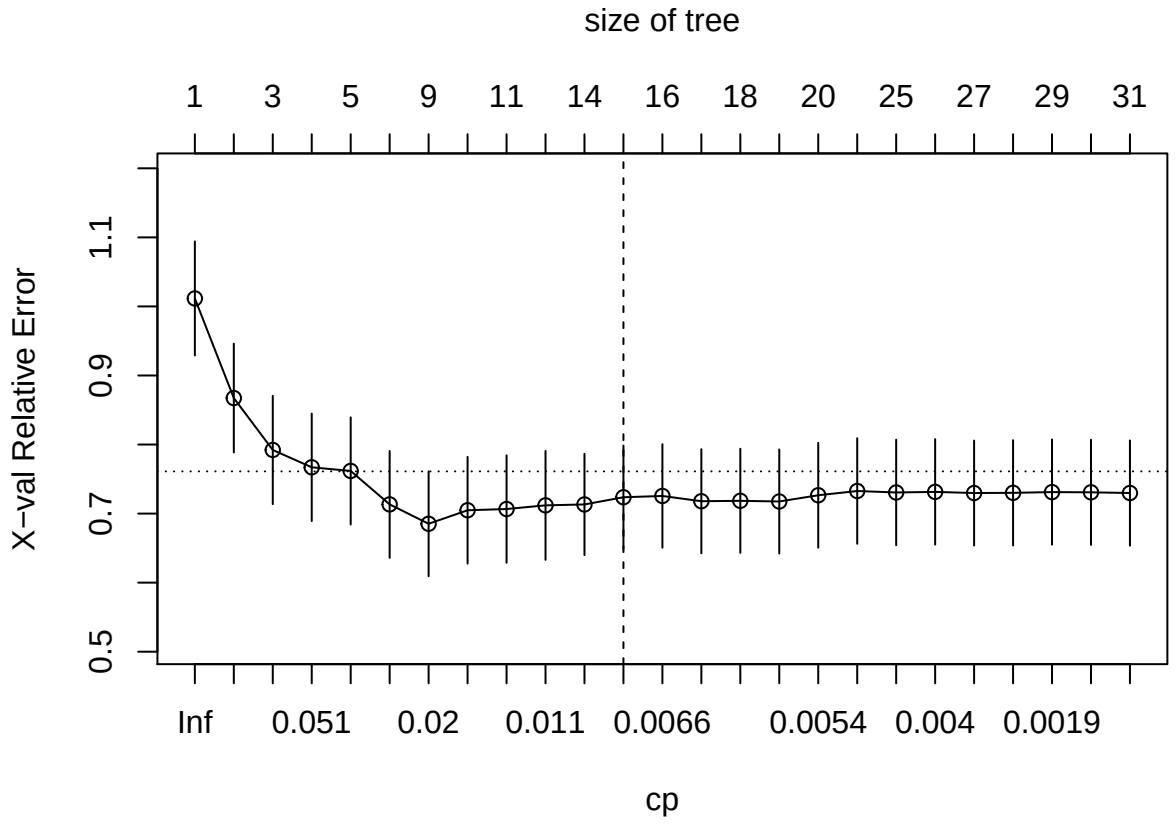
```
tree1 <- rpart(formula = LC50 ~ ., data=regression.train, method="anova")
rpart.plot(tree1)
```




```
plotcp(tree1)
```




```
tree2 <- rpart(formula = LC50 ~ .,data= regression.train, method = "anova", control = list(cp = 0, xval
plotcp(tree2)
abline(v = 12, lty = "dashed")
```



7. Comparing and contrasting

Problem 1.1.; We observed that the train RMSE was lower for the categorical effect compared to the linear effect, but this is just used as a guide thorugh the model selection and we are only intersted in knowing the relationship between the test RMSE. But; the RMSE was noticably higher for the dichotomized than for the original dataset used to fit the model. Hence, the original dataset was better at predicting than the dichotomized.

Problem 1.2.; The repeated procedure of 200 times produced a higher RMSE than the one observed in 1.1.. As mentioned this could be due to many factors but the most sound to assume would be that of variability in the split and as the mean is a sensitice measue it may vary highly between splits, thus increasing with each split. So far the un-dichotomized method is performing better, but the repeated procedure might follow the logic of CLT and approach its true value with an increase in iterations (Splits).

Problem 1.3.; The AIC and BIC in problem 1.3. was the same for each selection method. The reason why is probably because the number of predictors equal 7, which is when AIC = BIC.

Problem 1.4.; The Ridge regression is a smooth method when its being tuned since it follows a L2-norm which is differentiable. The Ridge regression method based on the estimates seams to be (this far) the most reasonable method to choose from the previous methods applied. The boostrap ridge regression did outperform the cv-method in terms of RMSE.

Problem 1.5.; Gam is a non-linear model and could easily fall victim to an overfit if we choose too many splines and add to its complexity. Hence, one schould be mindful of its usage and influence over highly complex models, but as this is not a complex model the gam-method should not be prefered to other models (unless there is a very valid reason to).

Problem 1.6; Regression trees are good visualizers and make it much easier for the person that interprets the analysis to understand the mechanisms and what is happening, this works well for transportability of the model.

The models perfor on average very similar but the one I would prefere to use from all of them would be ridge as it builds on the theory of principal components and find it easier to interpret the results and compexity of the method.

Problem 2

Vivian De La O Mellum

10 november 2020

Problem 2; Classification

The Pima Dataset is a publicly available dataset analysed many times in the literature (Royston & Sauerbrei, 2008). It contains information about 768 women of the population (Pima) particularrly susceptible to diabetes. The response **diabetes** identifies which of the persons involved in the study (268 women, **diabetes = 'pos'**) developed the disease.

Importing libraries

```
rm(list = ls()) ## Removing preexisting data so that there will be no overlap
library(MASS)
library(caret)
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

```
library(Metrics)       # to compute RMSE
```

```
##
## Attaching package: 'Metrics'
```

```
## The following objects are masked from 'package:caret':
##
##     precision, recall
```

```
library(gam)           # gam models
```

```
## Loading required package: splines
```

```
## Loading required package: foreach
```

```
## Loaded gam 1.16.1
```

```
library(rsample)       # data splitting
library(dplyr)         # data wrangling
```

```
##
## Attaching package: 'dplyr'

## The following object is masked from 'package:MASS':
##
##     select

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(rpart)         # performing regression trees
library(rpart.plot)    # plotting regression trees
library(ipred)         # bagging
library(caret)         # bagging
library(randomForest)  # random forest
```

```
## randomForest 4.6-14

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

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:dplyr':
##
##     combine

## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
library(party)         # aiding random forest
```

```
## Loading required package: grid

## Loading required package: mvtnorm

## Loading required package: modeltools

## Loading required package: stats4
```

```
## Loading required package: strucchange

## Loading required package: zoo

##
## Attaching package: 'zoo'

## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric

## Loading required package: sandwich
```

```
library(mlbench)       # aid PimaIndiansDiabetes
```

Loading the data

```
## Importing data
data(PimaIndiansDiabetes)
```

Describing the data

Eight Continuous independent variables contain information on:

pregnant: number of pregnancies;

glucose: plasma glucose concentration at 2 h in an oral glucose tolerance test;

pressure: diastolic blod pressure (mmHg);

triceps: triceps skin fold thickness (mm);

insulun: 2-h serum insulin (μU/mL);

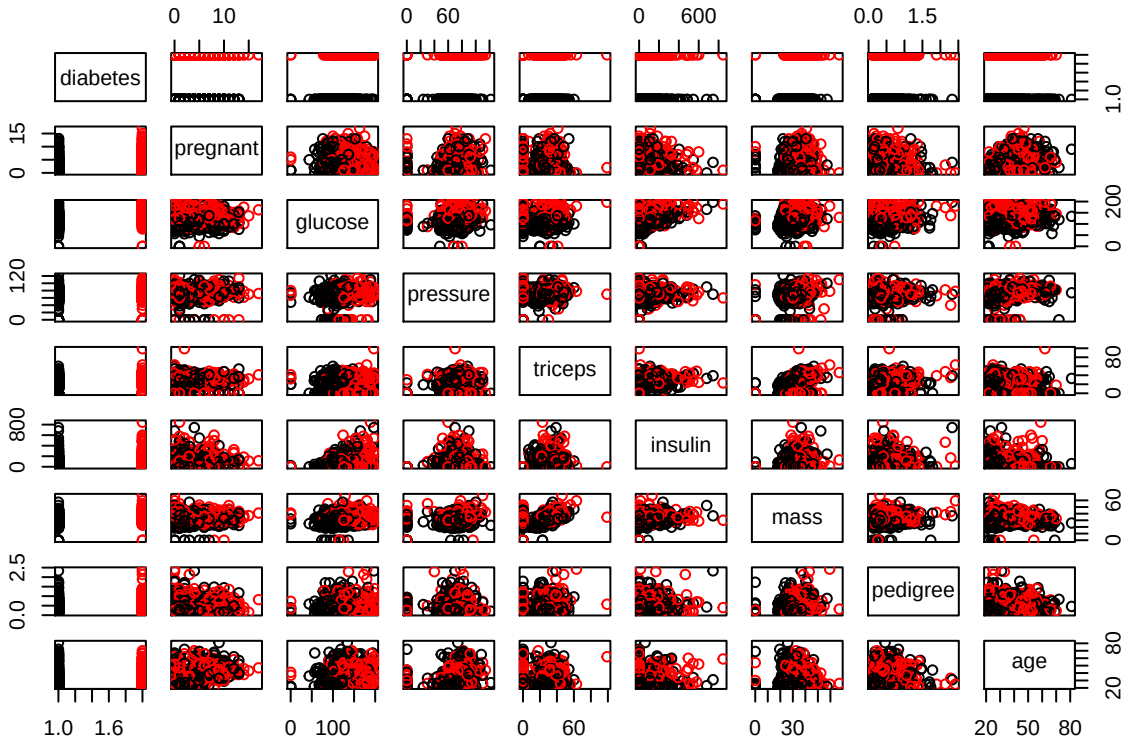
mass: body mass index (kg/m^2);

pedigree: diabetes pedigree function;

age: age(years);

Visualizing the data

```
pairs(diabetes~., data=PimaIndiansDiabetes, col=PimaIndiansDiabetes$diabetes)
```



Splitting the data into train and test set

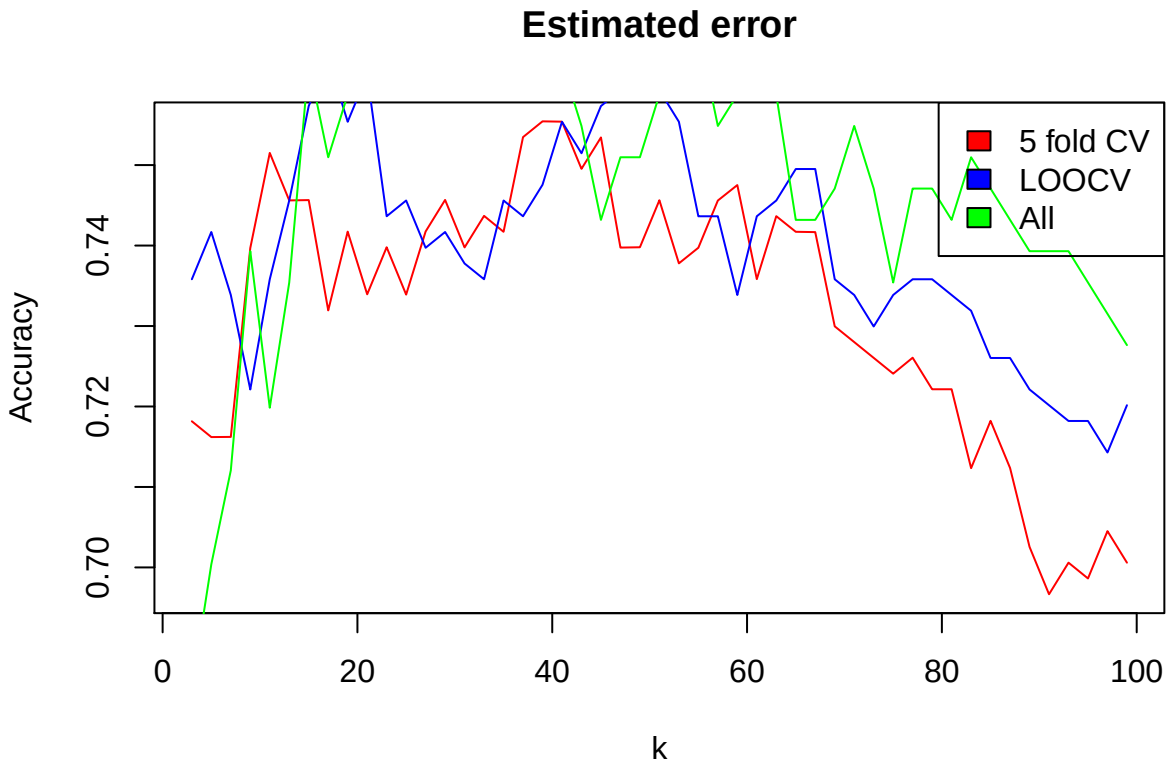
```
## Splitting into train and test set while preserving the proportion of women
## with diabetes and without.
set.seed(1234)


pos_diabetes <- which(PimaIndiansDiabetes$diabetes == 'pos')
neg_diabetes <- length(pos_diabetes)
train_ind_pos <- sample(pos_diabetes, size = floor(neg_diabetes*2/3))

neg_ind <- which(PimaIndiansDiabetes$diabetes == 'neg')
length_neg <- length(neg_ind)
train_ind_neg <- sample(neg_ind, size = floor(length_neg*2/3))

train_ind <- c(train_ind_pos, train_ind_neg)
classification.train <- PimaIndiansDiabetes[train_ind,]
classification.test <- PimaIndiansDiabetes[-train_ind,]
```

Classify using KNN; CV and LOOCV

We now wish to classify the patients using k-NN, selecting the best number of neighbouts both via 5-fold and a LOOCV procedure and plotting the estimated error for each possible value of k. We also wish to add

to the plot the corresponding test errors.

```
## data frame of k-values
k <- data.frame(k = seq(3, 99, 2))

## Cross-validation, K=5
set.seed(1234)
train_control_cv <- trainControl(method = "cv", number=5)

## kNN; k neares neighbours: procedure CV
fit.knn <- train(diabetes~., data=classification.train, method="knn", trControl=train_control_cv, prePr



## LOOCV
train_control_loocv <- trainControl(method="LOOCV")

## kNN; k neares neighbours: procedure LOOCV
fit.LOOCV <- train(diabetes~., data=classification.train, method="knn", trControl = train_control_loocv

## Fitting with all possible k with no CV
train_control <- trainControl(method = 'cv', number = 1, index = list(train_ind))
fit.knnAll <- train(diabetes~., data = PimaIndiansDiabetes, method = 'knn', trControl = train_control,



## Plot
plot(k$k, fit.knn$results$Accuracy, main = "Estimated error", ylab="Accuracy", xlab ="k", type = "l", c
lines(k$k, fit.LOOCV$results$Accuracy, type = "l", col = "blue")
lines(k$k, fit.knnAll$results$Accuracy, type = "l", col = "green")
legend("topright", c("5 fold CV", "LOOCV", "All"), fill=c("red", "blue", "green"))
```



2. Fitting a GAM with splines

```
### Preparing variable names ###
diabetes <- classification.train$diabetes
pregnant <- classification.train$pregnant
glucose <- classification.train$glucose
pressure <- classification.train$pressure
triceps <- classification.train$triceps
insulin <- classification.train$insulin
mass <- classification.train$mass
pedigree <- classification.train$pedigree
age <- as.numeric(classification.train$age)


### Considering splines up to five degrees of freedom ###
scope_list = list(
  "pregnant" = ~1 + pregnant + s(pregnant, df=2) + s(pregnant, df=3) + s(pregnant, df =4) + s(pregnant,
  "glucose" = ~1 + glucose + s(glucose, df=2) + s(glucose, df=3) + s(glucose, df=4) + s(glucose, df=5),
  "pressure" = ~1 + pressure + s(pressure, df=2) + s(pressure, df=3) + s(pressure, df=4) + s(pressure, 
  "triceps" = ~1 + triceps + s(triceps, df=2) + s(triceps, df=3) + s(triceps, df=4) + s(triceps, df=5),
  "insulin" = ~1 + insulin + s(insulin, df=2) + s(insulin, df=3) + s(insulin, df=4) + s(insulin, df=5),
  "mass" = ~1 + mass + s(mass, df=2) + s(mass, df=3) + s(mass, df=4) + s(mass, df=5),
  "pedigree" = ~1 + pedigree + s(pedigree, df=2) + s(pedigree, df=3) + s(pedigree, df=4) + s(pedigree, 
  "age" = ~1 + age + s(age, df=2) + s(age, df=3) + s(age, df=4) + s(age, df=5)
```

```
)


### forward stepwise selection ###
start_model = gam(diabetes~ pregnant + glucose + pressure + triceps + insulin + mass + pedigree + age, f

step.Gam(start_model, scope_list, direction = "both")
```

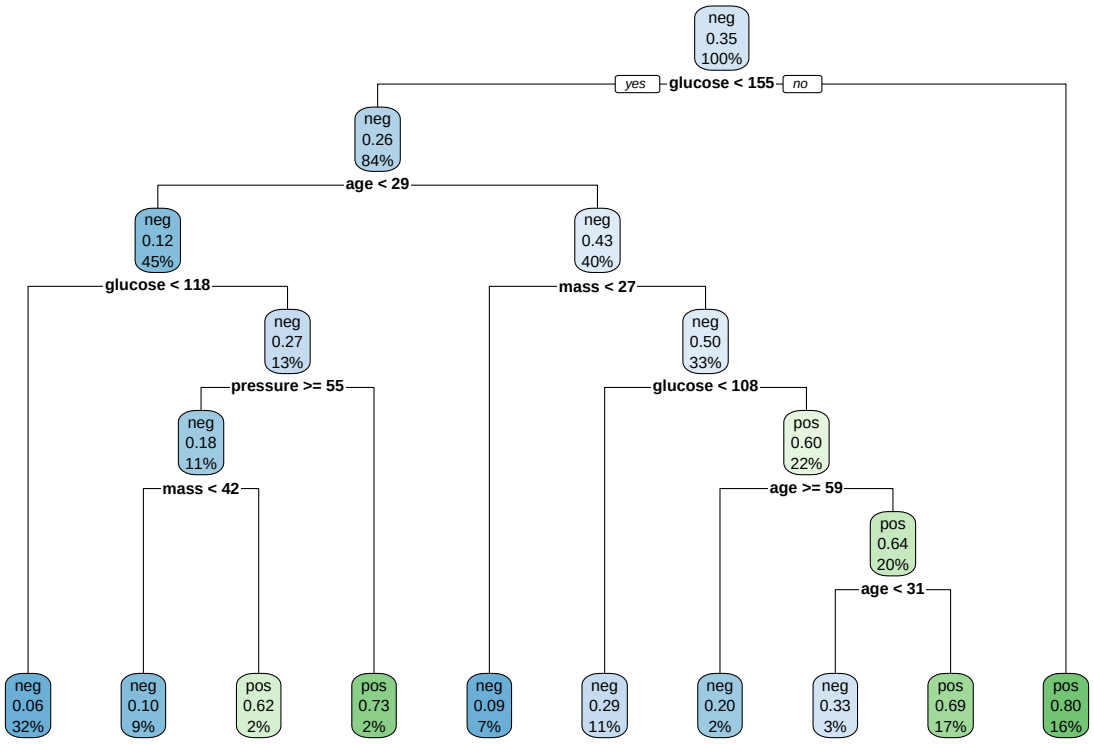
```
## Start:  diabetes ~ pregnant + glucose + pressure + triceps + insulin +      mass + pedigree + age; AI
## Step:1 diabetes ~ pregnant + glucose + pressure + triceps + insulin +      mass + pedigree + s(age, d
## Step:2 diabetes ~ pregnant + s(glucose, df = 2) + pressure + triceps +      insulin + mass + pedigree
## Step:3 diabetes ~ pregnant + s(glucose, df = 2) + pressure + triceps +      insulin + mass + s(pedigr
## Step:4 diabetes ~ pregnant + s(glucose, df = 3) + pressure + triceps +      insulin + mass + s(pedigr
## Step:5 diabetes ~ pregnant + s(glucose, df = 3) + pressure + triceps +      insulin + mass + s(pedigr
## Step:6 diabetes ~ pregnant + s(glucose, df = 3) + pressure + triceps +      insulin + s(mass, df = 2)
## Step:7 diabetes ~ pregnant + s(glucose, df = 3) + pressure + triceps +      insulin + s(mass, df = 3)
## Step:8 diabetes ~ pregnant + s(glucose, df = 3) + pressure + triceps +      insulin + s(mass, df = 4)
## Step:9 diabetes ~ pregnant + s(glucose, df = 4) + pressure + triceps +      insulin + s(mass, df = 4)
## Step:10 diabetes ~ pregnant + s(glucose, df = 4) + pressure + triceps +      insulin + s(mass, df = 5
## Step:11 diabetes ~ pregnant + s(glucose, df = 5) + pressure + triceps +      insulin + s(mass, df = 5
## Step:12 diabetes ~ pregnant + s(glucose, df = 5) + pressure + insulin +      s(mass, df = 5) + s(pedi
## Step:13 diabetes ~ s(glucose, df = 5) + pressure + insulin + s(mass,      df = 5) + s(pedigree, df =
## Step:14 diabetes ~ s(glucose, df = 5) + pressure + insulin + s(mass,      df = 5) + s(pedigree, df =
## Step:15 diabetes ~ s(glucose, df = 5) + pressure + insulin + s(mass,      df = 5) + s(pedigree, df =
```

```
## Call:
## gam(formula = diabetes ~ s(glucose, df = 5) + pressure + insulin +
##     s(mass, df = 5) + s(pedigree, df = 3) + s(age, df = 4), family = "binomial",
##     data = classification.train, trace = FALSE)
##
## Degrees of Freedom: 510 total; 491.0006 Residual
## Residual Deviance: 407.2912
```

Oddly enough the code does not want to work this time, eventhough the code was not touched. I tried to fix it but had no luck – therefore I will just ref. to the first attempt for the assignment where the code infact did work and was successful.

3 Classification tree, bagging (prob and consensus vote), random forest, neural network, AdaBoost

```
library(rpart)
library(randomForest)
library(ada)


diabetes <- classification.test$diabetes

cl.tree <- rpart(formula = diabetes ~ ., data=classification.train, method="class")
rpart.plot(cl.tree)
```




```
rmse_func <- function(model, data, response) {
  y_pred <- predict(model, data)
  y <- data[,response]
  return(sqrt(mean((y - y_pred)^2)))
}

# Bagging
bagCtrl <- bagControl(fit = ctreeBag$fit,
                      predict = ctreeBag$pred,
                      aggregate = ctreeBag$aggregate)

fit_bagging <- bag(diabetes~., data = classification.train, bagControl = bagCtrl)
```

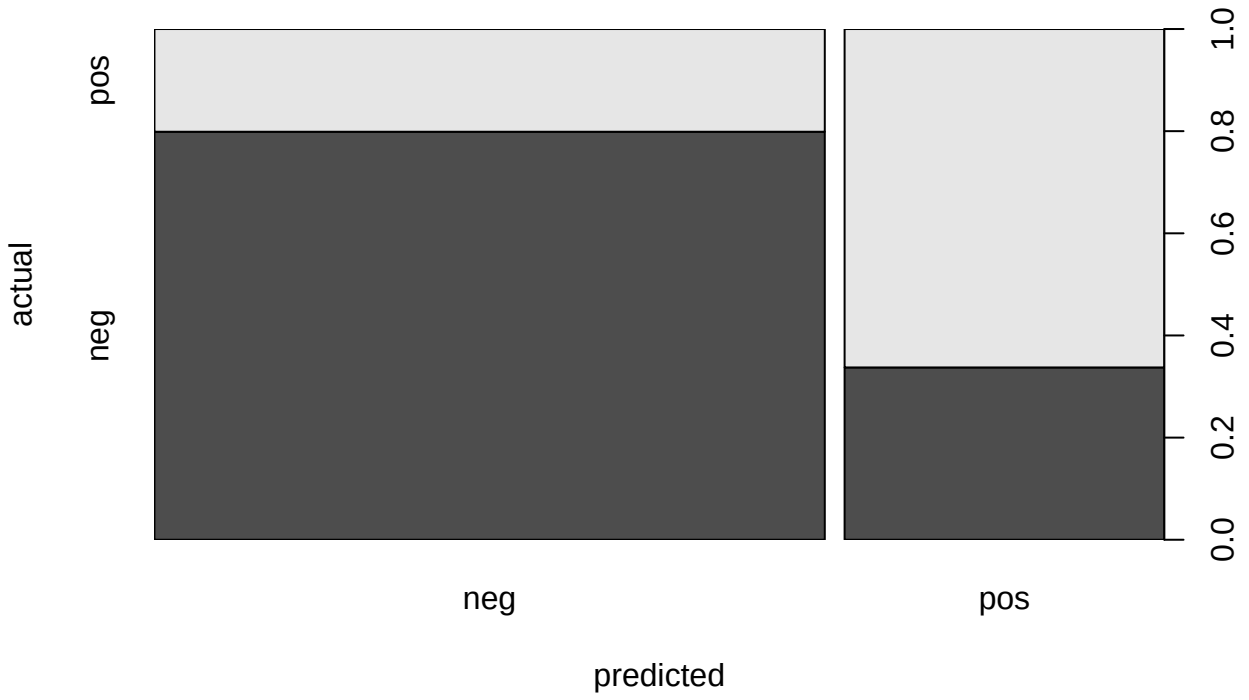
```
## Warning: executing %dopar% sequentially: no parallel backend registered
```

```
print(fit_bagging)
```

```
##
## Call:
## bag.formula(formula = diabetes ~ ., data = classification.train, bagControl
##  = bagCtrl)
##
##
## B: 10
```

```
## Training data: 8 variables and 511 samples
## All variables were used in each model
```

```
pred <- predict(fit_bagging, classification.test)
res <- data.frame(predicted = pred, actual = classification.test$diabetes)

plot(res)
```




```
# Creating random forest.
output.forest <- randomForest(diabetes ~ ., data = classification.train)
print(output.forest)
```

```
##
## Call:
##  randomForest(formula = diabetes ~ ., data = classification.train)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 22.7%
## Confusion matrix:
##     neg pos class.error
## neg 286  47   0.1411411
## pos  69 109   0.3876404
```

```r
# Adaboost
library(adabag)
```

```
## Loading required package: doParallel
```

```
## Loading required package: iterators
```

```
## Loading required package: parallel
```

```
##
## Attaching package: 'adabag'
```

```
## The following object is masked from 'package:ipred':
##
##     bagging
```

```r
library(caret)

model = boosting(diabetes~., data=classification.train, boos=TRUE, mfinal=50)
pred = predict(model, classification.test)
print(pred$confusion)
```

```
##                Observed Class
## Predicted Class neg pos
##             neg 138  34
##             pos  29  56
```

```r
print(pred$error)
```

```
## [1] 0.2451362
```

```r
cv.model = boosting.cv(diabetes~., data=classification.train, boos=TRUE, mfinal=10, v=5)
```

```
## i:  1 Tue Nov 10 21:20:46 2020
## i:  2 Tue Nov 10 21:20:48 2020
## i:  3 Tue Nov 10 21:20:50 2020
## i:  4 Tue Nov 10 21:20:53 2020
## i:  5 Tue Nov 10 21:20:55 2020
```

```r
print(cv.model[-1])
```

```
## $confusion
##                Observed Class
## Predicted Class neg pos
##             neg 278  68
##             pos  55 110
##
## $error
## [1] 0.2407045
```

4. Which method to choose

The purpose of this analysis is to create a model that can be interpreted in a simple way without compromising its ability of predicting. Therefor it is important to choose a method that is easy to interpret, predicts well and is trasferable to other problems (not just this data set). Therefore I wouls not reccomend the boosting algorithms as this takes the weight of several weak models and could be prone to outliers which in turn could lower the variance much more (this would in turn increase the bias). I would by preferences choose to go for the k-NN with a CV-procedure as this can be further improved upon by splitting the training into training and trainin-test set which could be used to tune nesessary parameters (such as shrinkage or number neighbours). The controll which we have over cv-prosedure driven methods is much more enjoyable to work with in terms of interpretability, predictability and transferability.

Problem 2.5

Vivian De La O Mellum

10 november 2020

By some reason the code did not comply in R-markdown but did execute in regular R-studio, therefor I include the last but of the assignment as a "separate" file.

R- code; Problem 2.5

```
rm(list = ls()) ## Removing preexisting data so that there will be no overlap


library(mlbench)
data("PimaIndiansDiabetes2")

dependent.variable <- "diabetes"
betas <- names(PimaIndiansDiabetes2)[names(PimaIndiansDiabetes2) != dependent.variable]

# Fix NA
PimaIndiansDiabetes2 <- na.omit(PimaIndiansDiabetes2)
set.seed(4300)
pos_ind <- which(PimaIndiansDiabetes2$diabetes == 'pos')
n_pos <- length(pos_ind)
train_ind_pos <- sample(pos_ind, size = floor(n_pos*2/3))

neg_ind <- which(PimaIndiansDiabetes2$diabetes == 'neg')
n_neg <- length(neg_ind)
train_ind_neg <- sample(neg_ind, size = floor(n_neg*2/3))

train_ind <- c(train_ind_pos, train_ind_neg)
classification.train <- PimaIndiansDiabetes2[train_ind,]
classification.test <- PimaIndiansDiabetes2[-train_ind,]



## 1  KNN; 5-fold & LOOCV


k <- data.frame(k = seq(3, 99, 2))



##  KNN; 5-fold & LOOCV
train.Control.cv <- trainControl(method = "repeatedcv", number = 5, repeats = 3)
```

```r
fit.knn <- train(
  diabetes ~ ., data = classification.train, method = 'knn', trControl = train.Control.cv,
  preProcess = c("center","scale"), tuneGrid = k)

train.Control.loocv <- trainControl(method = "LOOCV")
fit.LOOCV <- train(
  diabetes ~ ., data = classification.train, method = 'knn', trControl = train.Control.loocv,
  preProcess = c("center","scale"), tuneGrid = k)

train.controll.all <- trainControl(method = 'cv', number = 1, index = list(train_ind))
fit.knnAll <- train(
  diabetes ~ ., data = PimaIndiansDiabetes2, method = 'knn',
  trControl = train.controll.all , preProcess = c("center","scale"), tuneGrid = k)




## Plot
plot(k$k, fit.knn$results$Accuracy, main = "Estimated error", ylab="Accuracy", xlab ="k", type = "l", c
lines(k$k, fit.LOOCV$results$Accuracy, type = "l", col = "blue")
lines(k$k, fit.knnAll$results$Accuracy, type = "l", col = "green")
legend("topright", c("5 fold CV", "LOOCV", "All"), fill=c("red", "blue", "green"))



## 2  GAM
## ------------------------------------------------------------------------------------------------------



### Preparing variable names ###
diabetes <- classification.train$diabetes
pregnant <- classification.train$pregnant
glucose <- classification.train$glucose
pressure <- classification.train$pressure
triceps <- classification.train$triceps
insulin <- classification.train$insulin
mass <- classification.train$mass
pedigree <- classification.train$pedigree
age <- classification.train$age


### Considering splines up to five degrees of freedom ###
scope_list = list(
  "pregnant" = ~1 + pregnant + s(pregnant, df=2) + s(pregnant, df=3) + s(pregnant, df =4) + s(pregnant,
  "glucose" = ~1 + glucose + s(glucose, df=2) + s(glucose, df=3) + s(glucose, df=4) + s(glucose, df=5),
  "pressure" = ~1 + pressure + s(pressure, df=2) + s(pressure, df=3) + s(pressure, df=4) + s(pressure, d
  "triceps" = ~1 + triceps + s(triceps, df=2) + s(triceps, df=3) + s(triceps, df=4) + s(triceps, df=5),
  "insulin" = ~1 + insulin + s(insulin, df=2) + s(insulin, df=3) + s(insulin, df=4) + s(insulin, df=5),
  "mass" = ~1 + mass + s(mass, df=2) + s(mass, df=3) + s(mass, df=4) + s(mass, df=5),
  "pedigree" = ~1 + pedigree + s(pedigree, df=2) + s(pedigree, df=3) + s(pedigree, df=4) + s(pedigree, d
  "age" = ~1 + age + s(age, df=2) + s(age, df=3) + s(age, df=4) + s(age, df=5)
```

```
)


### forward stepwise selection ###
start_model = gam(diabetes~ pregnant + glucose + pressure + triceps + insulin + mass + pedigree + as.nu

final_model <- step.Gam(start_model, scope_list, direction = "both")




## 3  Classification tree, bagging, random forest, neural network and AdaBoost

library(rpart)
library(randomForest)
library(ada)

# Classification tree

diabetes <- classification.test$diabetes

cl.tree <- rpart(formula = diabetes ~ ., data=classification.train, method="class")
rpart.plot(cl.tree)




rmse_func <- function(model, data, response) {
  y_pred <- predict(model, data)
  y <- data[,response]
  return(sqrt(mean((y - y_pred)^2)))
}

# Bagging
bagCtrl <- bagControl(fit = ctreeBag$fit,
                      predict = ctreeBag$pred,
                      aggregate = ctreeBag$aggregate)

fit_bagging <- bag(diabetes~., data = classification.train, bagControl = bagCtrl)
print(fit_bagging)

pred <- predict(fit_bagging, classification.test)
res <- data.frame(predicted = pred, actual = classification.test$diabetes)

plot(res)




# Creating random forest.
output.forest <- randomForest(diabetes ~ ., data = classification.train)
print(output.forest)
```

```r
# Adaboost
library(adabag)
library(caret)

model = boosting(diabetes~., data=classification.train, boos=TRUE, mfinal=50)
pred = predict(model, classification.test)
print(pred$confusion)
print(pred$error)
cv.model = boosting.cv(diabetes~., data=classification.train, boos=TRUE, mfinal=10, v=5)
print(cv.model[-1])
```

The code when it runs in R-studio

```r
#
# > rm(list = ls()) ## Removing preexisting data so that there will be no overlap
# >
# >
# > library(mlbench)
# > data("PimaIndiansDiabetes2")
# >
# > dependent.variable <- "diabetes"
# > betas <- names(PimaIndiansDiabetes2)[names(PimaIndiansDiabetes2) != dependent.variable]
# >
# > # Fix NA
# > PimaIndiansDiabetes2 <- na.omit(PimaIndiansDiabetes2)
# > set.seed(4300)
# > pos_ind <- which(PimaIndiansDiabetes2$diabetes == 'pos')
# > n_pos <- length(pos_ind)
# > train_ind_pos <- sample(pos_ind, size = floor(n_pos*2/3))
# >
# > neg_ind <- which(PimaIndiansDiabetes2$diabetes == 'neg')
# > n_neg <- length(neg_ind)
# > train_ind_neg <- sample(neg_ind, size = floor(n_neg*2/3))
# >
# > train_ind <- c(train_ind_pos, train_ind_neg)
# > classification.train <- PimaIndiansDiabetes2[train_ind,]
# > classification.test <- PimaIndiansDiabetes2[-train_ind,]
# > ```{r}
# Error: attempt to use zero-length variable name
# > k <- data.frame(k = seq(3, 99, 2))
# >
# > ##  KNN; 5-fold & LOOCV
# > train.Control.cv <- trainControl(method = "repeatedcv", number = 5, repeats = 3)
# > fit.knn <- train(
# +   diabetes ~ ., data = data_train, method = 'knn', trControl = train.Control.cv,
# +   preProcess = c("center","scale"), tuneGrid = k)
# Error in eval(expr, p) : object 'data_train' not found
# >
# > library(mlbench)
```

```
# > data("PimaIndiansDiabetes2")
# >
# > dependent.variable <- "diabetes"
# > betas <- names(PimaIndiansDiabetes2)[names(PimaIndiansDiabetes2) != dependent.variable]
# >
# > # Fix NA
# > PimaIndiansDiabetes2 <- na.omit(PimaIndiansDiabetes2)
# > set.seed(4300)
# > pos_ind <- which(PimaIndiansDiabetes2$diabetes == 'pos')
# > n_pos <- length(pos_ind)
# > train_ind_pos <- sample(pos_ind, size = floor(n_pos*2/3))
# >
# > neg_ind <- which(PimaIndiansDiabetes2$diabetes == 'neg')
# > n_neg <- length(neg_ind)
# > train_ind_neg <- sample(neg_ind, size = floor(n_neg*2/3))
# >
# > train_ind <- c(train_ind_pos, train_ind_neg)
# > classification.train <- PimaIndiansDiabetes2[train_ind,]
# > classification.test <- PimaIndiansDiabetes2[-train_ind,]
# > k <- data.frame(k = seq(3, 99, 2))
# >
# >
# >
# > ##  KNN; 5-fold & LOOCV
# > train.Control.cv <- trainControl(method = "repeatedcv", number = 5, repeats = 3)
# > fit.knn <- train(
# +   diabetes ~ ., data = data_train, method = 'knn', trControl = train.Control.cv,
# +   preProcess = c("center","scale"), tuneGrid = k)
# Error in eval(expr, p) : object 'data_train' not found
# >
# > train.Control.loocv <- trainControl(method = "LOOCV")
# > fit.LOOCV <- train(
# +   diabetes ~ ., data = data_train, method = 'knn', trControl = train.Control.loocv,
# +   preProcess = c("center","scale"), tuneGrid = k)
# Error in eval(expr, p) : object 'data_train' not found
# >
# > train.controll.all <- trainControl(method = 'cv', number = 1, index = list(train_ind))
# > fit.knnAll <- train(
# +   diabetes ~ ., data = PimaIndiansDiabetes2, method = 'knn',
# +   trControl = train.controll.all , preProcess = c("center","scale"), tuneGrid = k)
# >
# >
# >
# >
# >
# > ## Plot
# > plot(k$k, fit.knn$results$Accuracy, main = "Estimated error", ylab="Accuracy", xlab ="k", type = "l
# Error in plot(k$k, fit.knn$results$Accuracy, main = "Estimated error",  :
#   object 'fit.knn' not found
# > lines(k$k, fit.LOOCV$results$Accuracy, type = "l", col = "blue")
# Error in xy.coords(x, y) : object 'fit.LOOCV' not found
# > lines(k$k, fit.knnAll$results$Accuracy, type = "l", col = "green")
# Error in plot.xy(xy.coords(x, y), type = type, ...) :
```

```
#   plot.new has not been called yet
# > legend("topright", c("5 fold CV", "LOOCV", "All"), fill=c("red", "blue", "green"))
# Error in strwidth(legend, units = "user", cex = cex, font = text.font) :
#   plot.new has not been called yet
# > k <- data.frame(k = seq(3, 99, 2))
# > ##  KNN; 5-fold & LOOCV
# > train.Control.cv <- trainControl(method = "repeatedcv", number = 5, repeats = 3)
# > fit.knn <- train(
# +   diabetes ~ ., data = data_train, method = 'knn', trControl = train.Control.cv,
# +   preProcess = c("center","scale"), tuneGrid = k)
# Error in eval(expr, p) : object 'data_train' not found
# > ##  KNN; 5-fold & LOOCV
# > train.Control.cv <- trainControl(method = "repeatedcv", number = 5, repeats = 3)
# > fit.knn <- train(
# +   diabetes ~ ., data = classification.train, method = 'knn', trControl = train.Control.cv,
# +   preProcess = c("center","scale"), tuneGrid = k)
# > train.Control.loocv <- trainControl(method = "LOOCV")
# > fit.LOOCV <- train(
# +   diabetes ~ ., data = classification.train, method = 'knn', trControl = train.Control.loocv,
# +   preProcess = c("center","scale"), tuneGrid = k)
# > train.controll.all <- trainControl(method = 'cv', number = 1, index = list(train_ind))
# > fit.knnAll <- train(
# +   diabetes ~ ., data = PimaIndiansDiabetes2, method = 'knn',
# +   trControl = train.controll.all , preProcess = c("center","scale"), tuneGrid = k)
# > ## Plot
# > plot(k$k, fit.knn$results$Accuracy, main = "Estimated error", ylab="Accuracy", xlab ="k", type = "l
# > lines(k$k, fit.LOOCV$results$Accuracy, type = "l", col = "blue")
# > lines(k$k, fit.knnAll$results$Accuracy, type = "l", col = "green")
# > legend("topright", c("5 fold CV", "LOOCV", "All"), fill=c("red", "blue", "green"))
# > ### Preparing variable names ###
# > diabetes <- classification.train$diabetes
# > pregnant <- classification.train$pregnant
# > glucose <- classification.train$glucose
# > pressure <- classification.train$pressure
# > triceps <- classification.train$triceps
# > insulin <- classification.train$insulin
# > mass <- classification.train$mass
# > pedigree <- classification.train$pedigree
# > age <- classification.train$age
# > ### Considering splines up to five degrees of freedom ###
# > scope_list = list(
# +   "pregnant" = ~1 + pregnant + s(pregnant, df=2) + s(pregnant, df=3) + s(pregnant, df =4) + s(pregn
# +   "glucose" = ~1 + glucose + s(glucose, df=2) + s(glucose, df=3) + s(glucose, df=4) + s(glucose, df
# +   "pressure" = ~1 + pressure + s(pressure, df=2) + s(pressure, df=3) + s(pressure, df=4) + s(pressu
# +   "triceps" = ~1 + triceps + s(triceps, df=2) + s(triceps, df=3) + s(triceps, df=4) + s(triceps, df
# +   "insulin" = ~1 + insulin + s(insulin, df=2) + s(insulin, df=3) + s(insulin, df=4) + s(insulin, df
# +   "mass" = ~1 + mass + s(mass, df=2) + s(mass, df=3) + s(mass, df=4) + s(mass, df=5),
# +   "pedigree" = ~1 + pedigree + s(pedigree, df=2) + s(pedigree, df=3) + s(pedigree, df=4) + s(pedigr
# +   "age" = ~1 + age + s(age, df=2) + s(age, df=3) + s(age, df=4) + s(age, df=5)
# + )
# > ### forward stepwise selection ###
# > start_model = gam(diabetes~ pregnant + glucose + pressure + triceps + insulin + mass + pedigree + a
# >
```

```
# > final_model <- step.Gam(start_model, scope_list, direction = "both")
# Start:  diabetes ~ pregnant + glucose + pressure + triceps + insulin +      mass + pedigree + as.nume
# Error in terms.formula(reformulate(term[i])) :
#   invalid model formula in ExtractVars
# In addition: Warning message:
# In model.matrix.default(mt, mf, contrasts) :
#   non-list contrasts argument ignored
# > ## 3  Classification tree, bagging, random forest, neural network and AdaBoost
# >
# > library(rpart)
# > library(randomForest)
# > library(ada)
# >
# > # Classification tree
# >
# > diabetes <- classification.test$diabetes
# >
# > cl.tree <- rpart(formula = diabetes ~ ., data=classification.train, method="class")
# > rpart.plot(cl.tree)
# > rmse_func <- function(model, data, response) {
# +   y_pred <- predict(model, data)
# +   y <- data[,response]
# +   return(sqrt(mean((y - y_pred)^2)))
# + }
# > # Bagging
# > bagCtrl <- bagControl(fit = ctreeBag$fit,
# +                       predict = ctreeBag$pred,
# +                       aggregate = ctreeBag$aggregate)
# >
# > fit_bagging <- bag(diabetes~., data = classification.train, bagControl = bagCtrl)
# > print(fit_bagging)
#
# Call:
# bag.formula(formula = diabetes ~ ., data = classification.train, bagControl = bagCtrl)
#
#
# B: 10
# Training data: 8 variables and 260 samples
# All variables were used in each model
# >
# > pred <- predict(fit_bagging, classification.test)
# > res <- data.frame(predicted = pred, actual = classification.test$diabetes)
# >
# > plot(res)
# > # Creating random forest.
# > output.forest <- randomForest(diabetes ~ ., data = classification.train)
# > print(output.forest)
#
# Call:
#  randomForest(formula = diabetes ~ ., data = classification.train)
#                Type of random forest: classification
#                      Number of trees: 500
# No. of variables tried at each split: 2
```

```
#
#         OOB estimate of  error rate: 23.46%
# Confusion matrix:
#     neg pos class.error
# neg 150  24   0.1379310
# pos  37  49   0.4302326
# > # Adaboost
# > library(adabag)
# > library(caret)
# >
# > model = boosting(diabetes~., data=classification.train, boos=TRUE, mfinal=50)
# > pred = predict(model, classification.test)
# > print(pred$confusion)
#                Observed Class
# Predicted Class neg pos
#             neg  75  16
#             pos  13  28
# > print(pred$error)
# [1] 0.219697
# > cv.model = boosting.cv(diabetes~., data=classification.train, boos=TRUE, mfinal=10, v=5)
# i:  1 Tue Nov 10 21:43:48 2020
# i:  2 Tue Nov 10 21:43:51 2020
# i:  3 Tue Nov 10 21:43:53 2020
# i:  4 Tue Nov 10 21:43:55 2020
# i:  5 Tue Nov 10 21:43:57 2020
# > print(cv.model[-1])
# $confusion
#                Observed Class
# Predicted Class neg pos
#             neg 147  30
#             pos  27  56
#
# $error
# [1] 0.2192308
```

Comments:

In general there were few differences between the datasets PimaIndiansDiabetes and PimaIndiansDiabetes2. Somehow I found it more difficult to work with the last dataset as it had its own challenges.

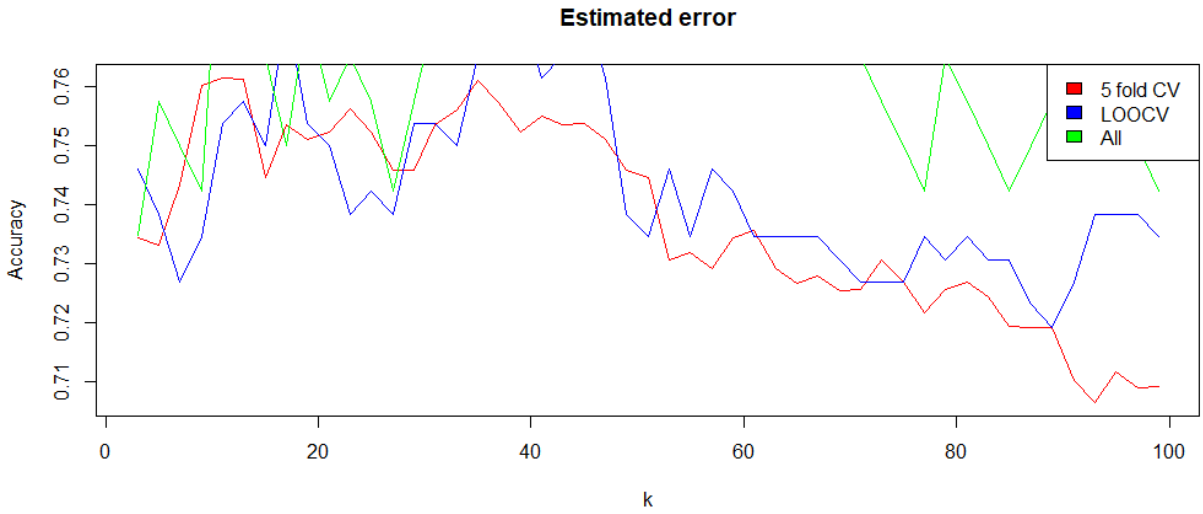


Figure 1: Fig. Classification tree

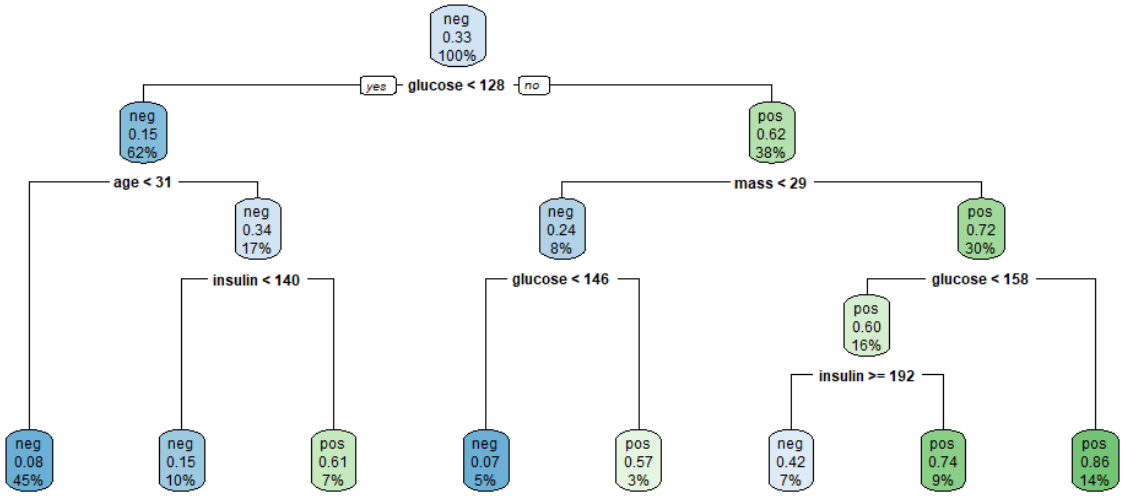


Figure 2: Fig. Random forest