Practical 5: The epigenetic clock: Predicting biological age on new data

Verena Zuber, Jelena Besevic, Alpha Forna, and Saredo Said

14/2/2019

Part 1: The epigenetic clock: Non-parametric prediction using random forests

In this practical we consider again the same data on $n = 409$ healthy mice and methylation of $p = 3,663$ conserved methylation sites as in the last two weeks. This time we are interested in random forests and see if random forests will provide a better prediction rule than penalised regression techniques. Install and load the package

```
library("randomForest")
```

Load the dataset, that contains the methylation matrix as predictor matrix and the age of the mice (in months) stored in the vector y. Familiarise yourself with the dataset using the following commands

```
load("data_epigenetic_clock_control")
#alternatively try load("data_epigenetic_clock_control.dms")
y = control_mice$y_control
x = control_mice$x_control
dim(x)
```

```
## [1]  409 3663
```

Question 1.1

First compute a random forest with the options 'ntree=100' and 'importance = TRUE' and save the output to an object call rf.out or similar.

Reply: It is very easy to build a random forest: all we need to specify is x and y. Set 'ntree=100' for now and importance =TRUE allows us to assess which variables are important. See next question.

```
rf.out = randomForest(x=x, y=y, ntree=100, importance =TRUE)
```

Question 1.2

Random forests have an intrinsic way of assigning variable importance to the predictors. How do random forests with quantitative outcome rank variables according to their importance? Rank the variables according to their importance and display the 10 most important methylation sites for ageing in the random forest algorithm. Use the varImpPlot() function to visualise the variable importance.

*Reply: Random forests offer two different measures for importance that are returned when setting .importance = TRUE":

-Permutation: Mean decrease in accuracy (%IncMSE)

First the random forest is generated and the prediction accuracy (mse) is evaluated on the out-of-bag samples. Then for a particular variable j the values of the out-of-bag samples are permuted and the prediction accuracy is evaluated on the permuted out-of-bag samples. The mean decrease of mse after permuting variable j is used to indicate the importance of a variable.

-Gini-index: Mean decrease in impurity (IncNodePurity)

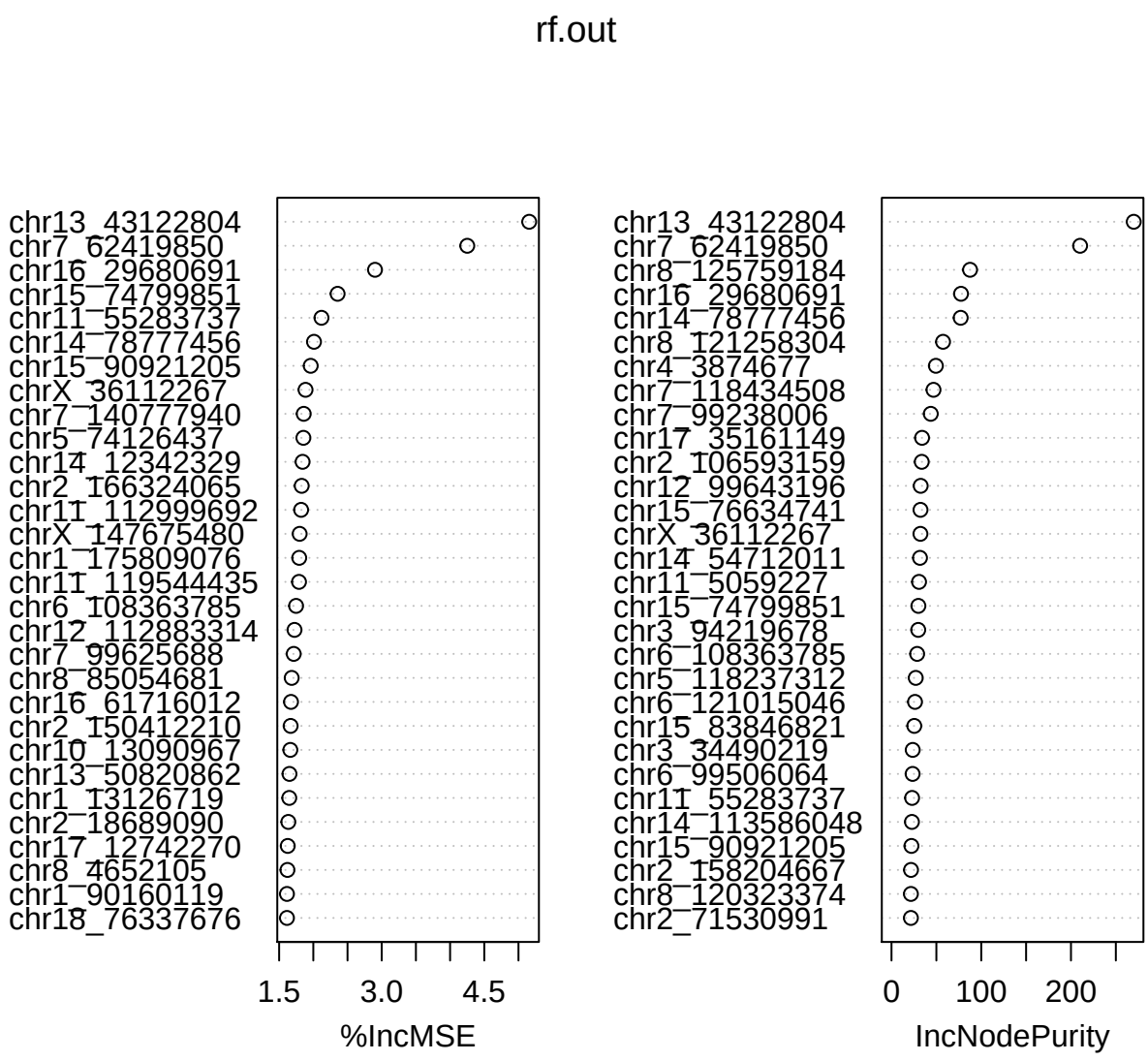
Alternatively, the mean decrease of impurity can be used to measure variable importance. For quantitative traits, impurity can be measured using residual sum of squares. It is computed as the decrease of impurity

every time a tree is split at variable j and averaged over all trees. *

```
importance=rf.out$importance
importance[order(importance[,2], decreasing =TRUE),] [1:10,]
```

```
##                     %IncMSE IncNodePurity
## chr13_43122804 1.031013814     269.96346
## chr7_62419850  0.694810134     210.18757
## chr8_125759184 0.145005190      87.53192
## chr16_29680691 0.339100400      77.48248
## chr14_78777456 0.274559062      77.10739
## chr8_121258304 0.013848863      57.42944
## chr4_3874677   0.013222970      49.45386
## chr7_118434508 0.086355596      46.79695
## chr7_99238006  0.104335224      43.75049
## chr17_35161149 0.005612514      34.04893
```

```
varImpPlot(rf.out)
```



Question 1.3

How would you run bagging using the randomForest function? What is the interpretation of the mtry option?

Reply: mtry indicates how many predictors of x should be considered to split the tree. Setting mtry to the number of columns of the predictor matrix is basically a bagging approach and not a random forest.

```
bagging.out = randomForest(x=x, y=y, ntree=100, importance =TRUE, mtry=ncol(x))
```

Question 1.4

Next step is to write a prediction function for the random forest algorithm. Re-use your code from last week (Practical 3 Question 2) and see the prediction function for a linear regression model below. Set the number of trees as an open parameter.

```
predfun.lm = function(train.x, train.y, test.x, test.y){
    #fit the model and build a prediction rule
    lm.fit = lm(train.y ~ ., data=train.x)
    #predict the new observation based on the test data and the prediction rule
    ynew = predict(lm.fit, test.x )

    #compute mse as squared difference between predicted and observed outcome
    out = mean( (ynew - test.y)^2 )
    return( out )

}
```

Reply: The following code can be used as a prediction function. Adding ntree as open parameter allows us to compare the performance for different training length of the random forest algorithm.

```
predfun.rf = function(train.x, train.y, test.x, test.y, ntree){

    #fit the model and build a prediction rule
    rf.fit = randomForest(x=train.x, y=train.y, ntree=ntree)
    #predict the new observation based on the test data and the prediction rule
    ynew = predict(rf.fit , newdata=test.x)

    # compute squared error risk (MSE)
    out = mean( (ynew - test.y)^2 )
    return( out )

}
```

Question 1.5

Compare the prediction performance using ntree = 10 and ntree = 100 random trees to train the random forest. To this end use the

```
library(crossval)
```

package and compare the two parameters using k-fold cross-validation (cv) using $k = 5$ folds and B=10 repetitions (Ideally you would want to run this with more repetitions (B=100 to B=1000), but for this practical 10 will do).

Reply: We find that the random forest that was trained using more random trees performs better. It is recommended to use at least 500 or more trees.

```
set.seed(14)
cv.out.rf10  =  crossval(predfun.rf, x, y, ntree=10, K=5, B=10, verbose=FALSE)
cv.out.rf10$stat
```

```
## [1] 17.93867
```

```
cv.out.rf10$stat.se
```

```
## [1] 0.4388459
```

```
cv.out.rf100  =  crossval(predfun.rf, x, y, ntree=100, K=5, B=10, verbose=FALSE)
cv.out.rf100$stat
```

```
## [1] 16.30819
```

```
cv.out.rf100$stat.se
```

```
## [1] 0.3803941
```

Question 1.6

Finally, we want to know if random forests outperform regularised regression with respect to prediction performance. Re-use the following code from last week (Practical 3 Question 2) as prediction rule for glmnet() and use cv as implemented in crossval() to determine if random forests (ntree = 100) is better than lasso and elastic net as implemented in the glmnet package.

```
library(glmnet)
predfun.glmnet = function(train.x, train.y, test.x, test.y, lambda = lambda, alpha=alpha){

    #fit glmnet prediction rule
    glmnet.fit = glmnet(x=train.x, y=train.y, lambda = lambda, alpha=alpha)
    #predict the new observation based on the test data and the prediction rule
    ynew = predict(glmnet.fit , newx=test.x)

    # compute squared error risk (MSE)
    out = mean( (ynew - test.y)^2 )
    return( out )

}
```

Reply: First we repeat the cv to tune the regularised regression.

```
#cv
set.seed(12)
lasso.cv = cv.glmnet(x,y,family="gaussian",alpha=1, type.measure="mse")
set.seed(123)
ridge.cv = cv.glmnet(x,y,family="gaussian",alpha=0, type.measure="mse")
set.seed(1234)
a = seq(0.05, 0.95, 0.05)
search = foreach(i = a, .combine = rbind)%do%{
            cv = cv.glmnet(x,y,family = "gaussian", type.measure = "mse", alpha = i)
            data.frame(cvm = cv$cvm[cv$lambda == cv$lambda.1se],
             lambda.1se = cv$lambda.1se, alpha = i)
}
elasticnet.cv = search[search$cvm == min(search$cvm), ]
```

Then we run the crossval function to compare different methods. We find that elastic net has the lowest cross-validation error, followed by lasso. This is a surprising results because random forests are generally known to outperform other methods. See for example the Article 'Do we Need Hundreds of Classifiers to Solve Real World Classification Problems? here:

http://jmlr.org/papers/volume15/delgado14a/delgado14a.pdf

```
set.seed(15)
cv.out.ridge =  crossval(predfun.glmnet, x, y, lambda=ridge.cv$lambda.1se,
```

```
 alpha=0, K=5, B=10, verbose=FALSE)
cv.out.lasso =  crossval(predfun.glmnet, x, y, lambda=lasso.cv$lambda.1se,
 alpha=1, K=5, B=10, verbose=FALSE)
cv.out.enet  =  crossval(predfun.glmnet, x, y, lambda = elasticnet.cv$lambda.1se,
 alpha = elasticnet.cv$alpha, K=5, B=10, verbose=FALSE)
table.out = rbind(c(cv.out.ridge$stat, cv.out.ridge$stat.se), c(cv.out.lasso$stat, cv.out.lasso$stat.se
 c(cv.out.enet$stat, cv.out.enet$stat.se), c(cv.out.rf10$stat, cv.out.rf10$stat.se), c(cv.out.rf100$sta
rownames(table.out) = c("ridge", "lasso", "elastic net", "rf 10", "rf 100")
colnames(table.out) = c("mse", "se")
table.out
```

```
##                  mse        se
## ridge       15.87487 0.4215827
## lasso       13.20459 0.3796007
## elastic net 12.56983 0.4282344
## rf 10       17.93867 0.4388459
## rf 100      16.30819 0.3803941
```

Part 2: The epigenetic clock: Evaluating the impact of nitrogen dioxide on biological age

An environmental research group gets in contact with us. They have measured the methylation profile of $n = 140$ mice of which 40 mice have been exposed to nitrogen dioxide (NO2) for 10 hours a day for 10 weeks. The remaining 100 mice are healthy controls. Load the data and familiarise yourself with the data structure.

```
load("data_epigenetic_clock_experimental")
#alternatively try load("data_epigenetic_clock_experimental.dms")
exposed = experimental_mice$exposed
table(exposed)
```

```
## exposed
##   0   1
## 100  40
```

```
x.test = experimental_mice$x_exp
dim(x.test)
```

```
## [1]  140 3663
```

The working hypothesis is that NO2 exposure reduces the biological age of mice and makes the exposed mice age more quickly. The environmental research group asks us to predict the biological age of the mice using our algorithms to determine the biological age of a mouse.

Question 2.1

Predict the biological age of the $n = 140$ new mice using the lasso model (Practical 3: Question 1.4) from last week as implemented in the glmnet package. Use again \$lambda.1se as the regularisation parameter and save the predicted age in an object called lasso.hat.

Reply: For completeness we repeat here the analysis from last week and save the lasso predictions of the new data (x.test) to the object lasso.hat.

```
#building the prediction rule
lasso.cv.out = glmnet(x,y,family="gaussian",alpha=1,lambda=lasso.cv$lambda.1se)
#evaluating on the new data
lasso.hat = predict(lasso.cv.out,newx=x.test, s=lasso.cv$lambda.1se)
```

Question 2.2

Predict the biological age of the $n = 140$ new mice using the ridge model (Practical 3: Question 1.5) from last week. Use again $lambda.1se as the regularisation parameter and save the predicted age in an object called ridge.hat.

Reply: In analogy, we perform first cv, then build the ridge prediction rule and finally make the predictions of the new data (x.test) and save them in the object ridge.hat.

```
#building the prediction rule
ridge.cv.out = glmnet(x,y,family="gaussian",alpha=0,lambda=ridge.cv$lambda.1se)
#evaluating on the new data
ridge.hat = predict(ridge.cv.out,newx=x.test, s=ridge.cv$lambda.1se)
```

Question 2.3

Predict the biological age of the $n = 140$ new mice using the elastic net model (Practical 3: Question 1.6) from last week. Use again $lambda.1se as the regularisation parameter and save the predicted age in an object called enet.hat.

Reply: And we repeat this for the elastic net as well and save the predictions in the elasticnet.hat object.

```
#building the prediction rule
enet.out = glmnet(x, y, family = "gaussian",
 lambda = elasticnet.cv$lambda.1se, alpha = elasticnet.cv$alpha)
#evaluating on the new data
enet.hat = predict(enet.out,newx=x.test,
 lambda = elasticnet.cv$lambda.1se, alpha = elasticnet.cv$alpha)
```

Question 2.4

Predict the biological age of the $n = 140$ new mice using the a random forest model. This time use "ntree=500" to build a reliable prediction rule and save the predicted age in an object called rf.hat.

Reply: Similarly we can use the random forest model from Question 1.1 for prediction and save the predictions in the rf.hat object.

```
#building the prediction rule
rf.out = randomForest(x=x, y=y, importance =TRUE, ntree=500)
#evaluating on the new data
rf.hat = predict(rf.out,newdata=x.test)
```

Question 2.5

Can you confirm the working hypothesis that NO2 exposure reduces the biological age of mice? Perform a t-test to see if the predicted biological age of the mice differs significantly between exposed and healthy mice. Perform a $t-$test for all four predictions done in Question 2.1-2.4. How will you advice the environmental research group?

Reply: We perform a 2 group t-test for each epigenetic clock and .

```
#random forest
t.test(rf.hat~exposed)
```

```
##
##  Welch Two Sample t-test
##
## data:  rf.hat by exposed
## t = -1.2554, df = 73.643, p-value = 0.2133
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
```

```
##  -0.35075760  0.07961571
## sample estimates:
## mean in group 0 mean in group 1
##        31.20033        31.33590
```

```
#ridge
t.test(ridge.hat~exposed)
```

```
##
##  Welch Two Sample t-test
##
## data:  ridge.hat by exposed
## t = 0.6639, df = 74.268, p-value = 0.5088
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.1339463  0.2678200
## sample estimates:
## mean in group 0 mean in group 1
##        30.77101        30.70407
```

```
#lasso
t.test(lasso.hat~as.factor(exposed))
```

```
##
##  Welch Two Sample t-test
##
## data:  lasso.hat by as.factor(exposed)
## t = -2.652, df = 72.476, p-value = 0.009823
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -1.6230142 -0.2301665
## sample estimates:
## mean in group 0 mean in group 1
##        31.36475        32.29134
```

```
#elastic net
t.test(enet.hat~exposed)
```

```
##
##  Welch Two Sample t-test
##
## data:  enet.hat by exposed
## t = -2.9056, df = 75.755, p-value = 0.004803
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -1.8458872 -0.3444328
## sample estimates:
## mean in group 0 mean in group 1
##        31.44297        32.53813
```

Interpretation: We do see a significant difference in biological age using the lasso and elastic net prediction rules. Given the cross-validation we performed in Section 1.5 we know that both lasso and elastic net have the best prediction performance. Since both lasso and elastic net agree that the mice who have been exposed to NO2 did age quicker than the mice in the control group we are confident that NO2 exposure impacts the epigenetic clock of mice.

Part 3 (optional): Non-linear methods and curve-fitting

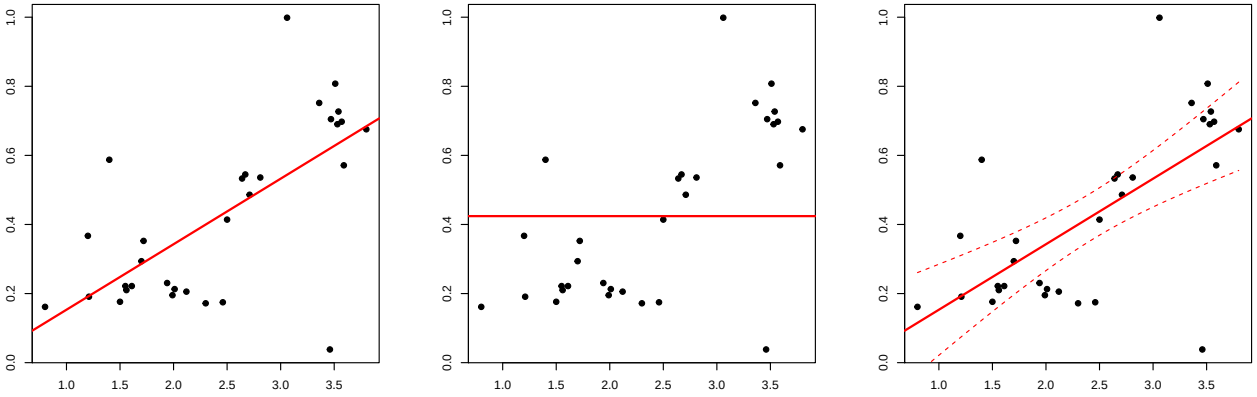
Question 3: How many of the following curve-fitting methods as shown in Figure 1 can you fit in R? Load the data from blackboard into R using the following lines

```
curve.fitting = read.csv("curve-fitting.txt", sep="\t")
x = curve.fitting$x
y = curve.fitting$y
```

Question 3.1

Reply: First, we consider linear regression and linear regression without slope, just intercept and with confidence intervals.

```
par(mfrow=c(1,3)) #How many figures next to each other?
#Linear
lm.object = lm(y~x)
plot(x,y, pch=19, xlab="", ylab="")
abline(lm.object, col="red", lwd = 2)
#Linear, no slope
lm.intercept = lm(y~1)
plot(x,y, pch=19, xlab="", ylab="")
abline(lm.intercept, col="red", lwd = 2)
#Linear with confidence
newx = seq(min(x), max(x), length.out=100)
lm.confidence = predict(lm.object, newdata = data.frame(x=newx), interval = 'confidence')
plot(x,y, pch=19, xlab="", ylab="")
abline(lm.object, col="red", lwd = 2)
lines(newx, lm.confidence[ ,3], lty = 'dashed', col = 'red')
lines(newx, lm.confidence[ ,2], lty = 'dashed', col = 'red')
```



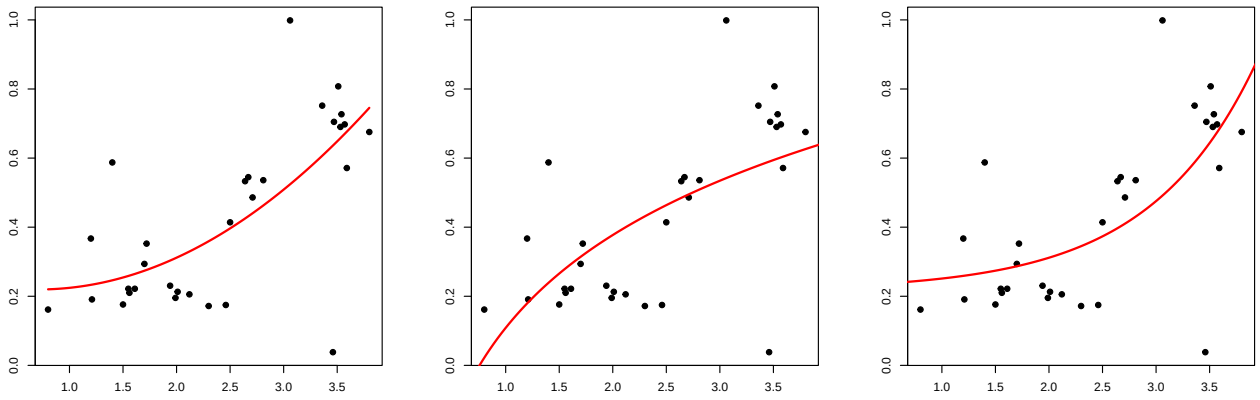
Question 3.2

How can we fit non-linear relationships with the linear model?

Reply: We can also fit non-linear functions of x such as quadratic, log or exponential in the linear model by transforming x accordingly.

```
par(mfrow=c(1,3)) #How many figures next to each other?
#Quadratic
lm.quad = lm(y~x+I(x^2))
plot(x,y, pch=19, xlab="", ylab="")
curve(predict(lm.quad,data.frame(x=x)),col='red',lwd=2,add=TRUE)
#Log
lm.log = lm(y~log(x))
```

```
fitted=predict(lm.log,newdata=list(x=seq(from=0.5,to=4,length.out=100)))
plot(x,y, pch=19, xlab="", ylab="")
matlines(x=seq(from=0.5,to=4,length.out=100),fitted,lwd=2, col='red')
#Exponential
lm.exp = lm(y~exp(x))
fitted=predict(lm.exp,newdata=list(x=seq(from=0.5,to=4,length.out=100)))
plot(x,y, pch=19, xlab="", ylab="")
matlines(x=seq(from=0.5,to=4,length.out=100),fitted,lwd=2, col='red')
```



Question 3.3

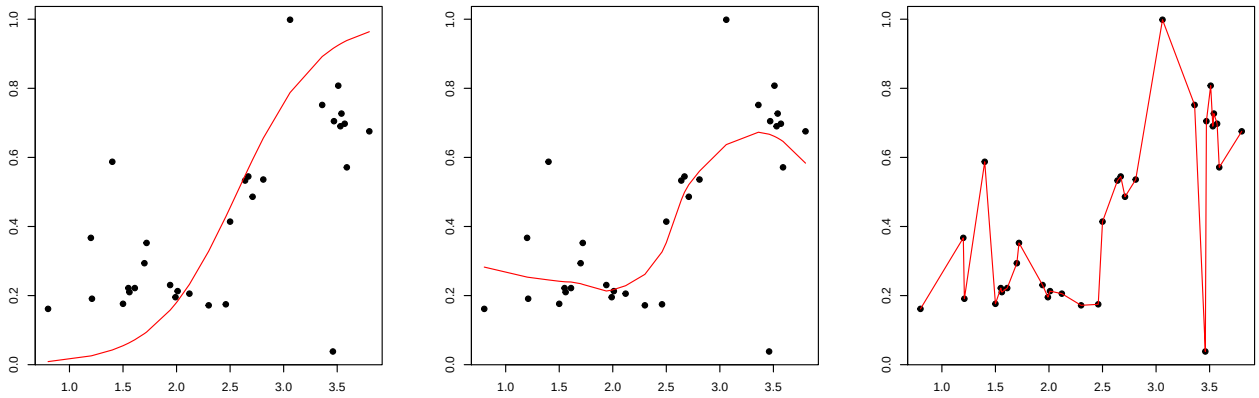
Can we use a glm to fit a logistic function? And how does smoothing using the loess function work?

Reply: By binarising y (which is in the range [0,1]) we can create a new binary outcome y.bin. The binary outcome we can use to fit in a logistic regression using the glm function. LOWESS (Locally Weighted Scatterplot Smoothing), often called LOESS (locally weighted smoothing) is a non-parametric smoothing approach. It has a very important tuning parameter named span, which controls the degree of smoothing. Setting it to span=0.75 gives a smooth line. This smooth line is biased but will generalise well to new observations. If we set span to a small value, e.g span=0.1 we are essentially not smoothing at all and connecting lines. This may not be biased but will certainly not generalise to new data.

```
par(mfrow=c(1,3))
y.bin = as.numeric(y>mean(y))
table(y.bin)
```

```
## y.bin
##  0  1
## 17 14
```

```
glm.object = glm(y.bin~x, family = binomial)
glm.fitted = fitted(glm.object)
plot(x,y, pch=19, xlab="", ylab="")
lines(glm.fitted, x=x, col="red")
#Smooth loess
loess = loess(y~x, span=0.75)
smoothed75 = predict(loess)
plot(x,y, pch=19, xlab="", ylab="")
lines(smoothed75, x=x, col="red")
#Connecting lines
loess = loess(y~x, span=0.1)
connectinglines = predict(loess)
plot(x,y, pch=19, xlab="", ylab="")
lines(connectinglines, x=x, col="red")
```



Question 3.4

The last three curve-fitting methods are advanced and are beyond the scope of the module. House of Cards is a mistery and we are not sure if it is implemented in R ...

```
par(mfrow=c(1,3))
#piecewice non-linear Least Squares in the nls function,
#we fit two sections, for x smaller and for x larger than the median
nls.out = nls(y ~ ifelse(x < median(x),ya+A*x,yb+B*x), data = data.frame(x,y))
plot(x,y, pch=19, xlab="", ylab="")
lines(x[1:15], fitted(nls.out)[1:15], lwd=2, col='red')
lines(x[17:31], fitted(nls.out)[17:31], lwd=2, col='red')
#Filter using runmed function where k: integer width of median window
ymed = runmed(y,5)
plot(x,y, pch=19, xlab="", ylab="")
lines(x, ymed, lwd=2, col='red')
#House of Cards??
plot(x,y, pch=19, xlab="", ylab="")
```