Universidade do Porto

Faculdade de Engenharia

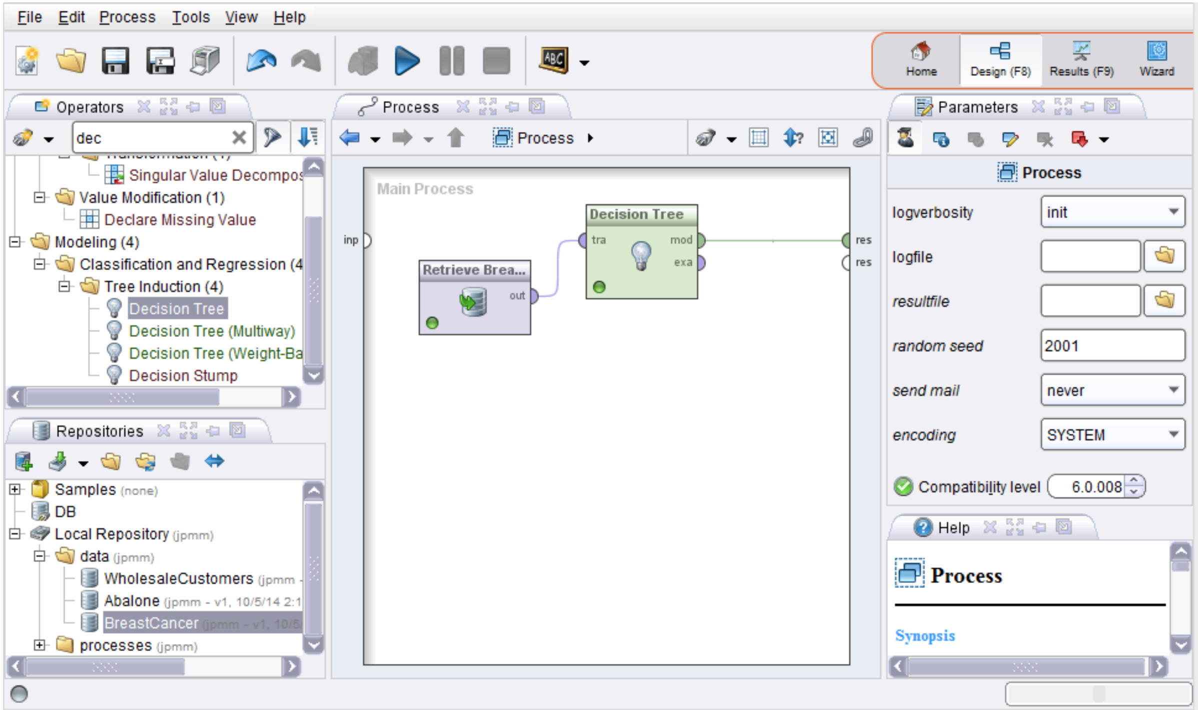
FEUP

Descoberta de conhecimento

Experimental Setup for Prediction

Previsão

O setup experimental apresentado na imagem é adequado para análise do conjunto de dados em apreço, mas não é adequado para previsão devido ao problema de sobre-ajustamento / *overfitting*



Experimental Setup

Prediction

Formalization:
Let y be the variable we intended to predict.
Let $\mathbf{x}$ be a vector with independent variables.
Let f be the unknown function that establishes the relationship between $\mathbf{x}$ and y:

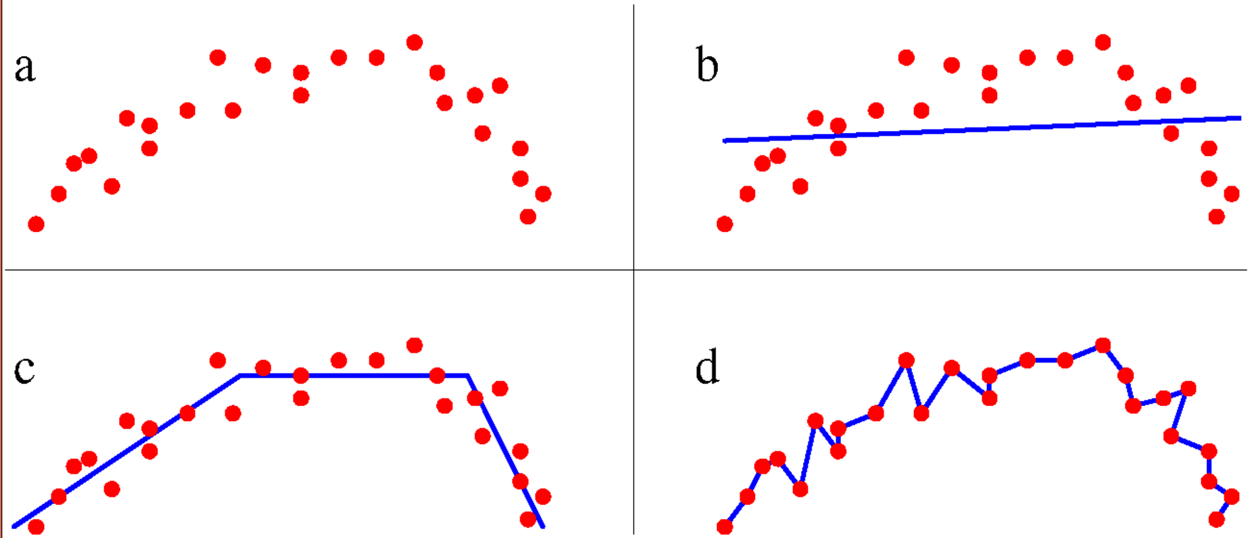
$$y = f(\mathbf{x}) + Er$$

'Er' is the component of y that cannot be explained by $\mathbf{x}$.

The prediction goal is to find a function to estimate y knowing the values of $\mathbf{x}$, i.e., to find a function $\hat{f}(\mathbf{x})$ that obtains y estimates (represented as $\hat{y}$), given the values of $\mathbf{x}$.

Overfitting

If the goal is to obtain a model that should be ble to predict new instances, which model would you choose: b? c? d?



There is no guarantee that a model that fits well the training data has similar ability to fit test data.

That is why a training set is used to build the model and a test set is used to evaluate the ability to predict new instances.

Retirado de http://cg.postech.ac.kr/publications/images/YLee06b.png

Some issues concerning data splitting in classification

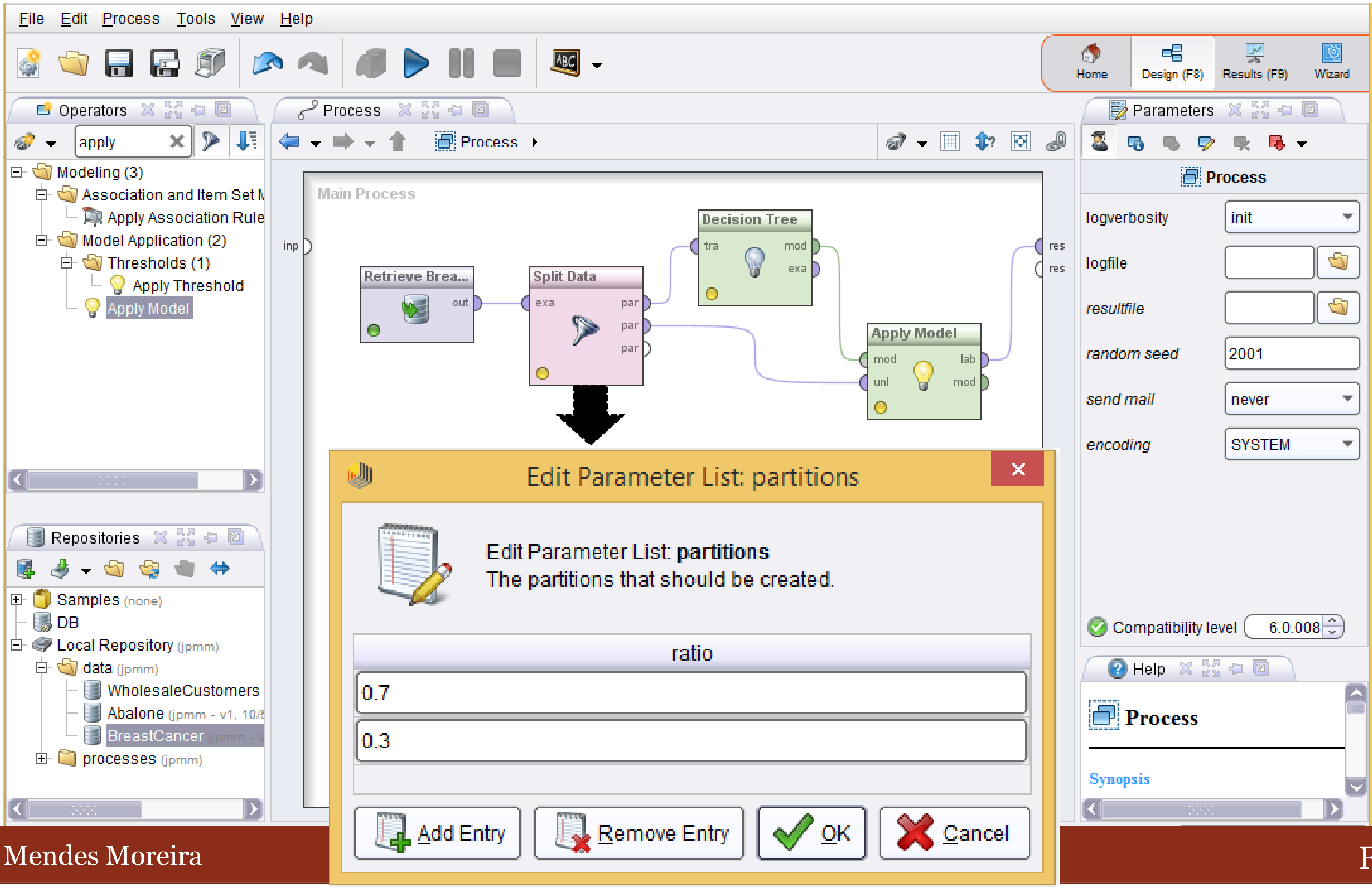
- When splitting data into train and test sets we should guaranttee that all classes are present in the same proportions in both sets.
- This is especially important when one class has a disproportionately small frequency compared to the others.
- When the test set has a class that is not present in the training set, an error will occur.
- In such cases, the levels of the target variable (typically a factor), should include all classes.

Hold-out

1. Separate available instances in two sets:
   a) Training set: instances used for training the model;
   b) Test set: the remaining instances.
2. The model is evaluated in the test set.

Operator in RapidMiner: split
Functions in R: sample (base) and createDataPartition (caret)

Hold-out: RapidMiner



Hold-out: R

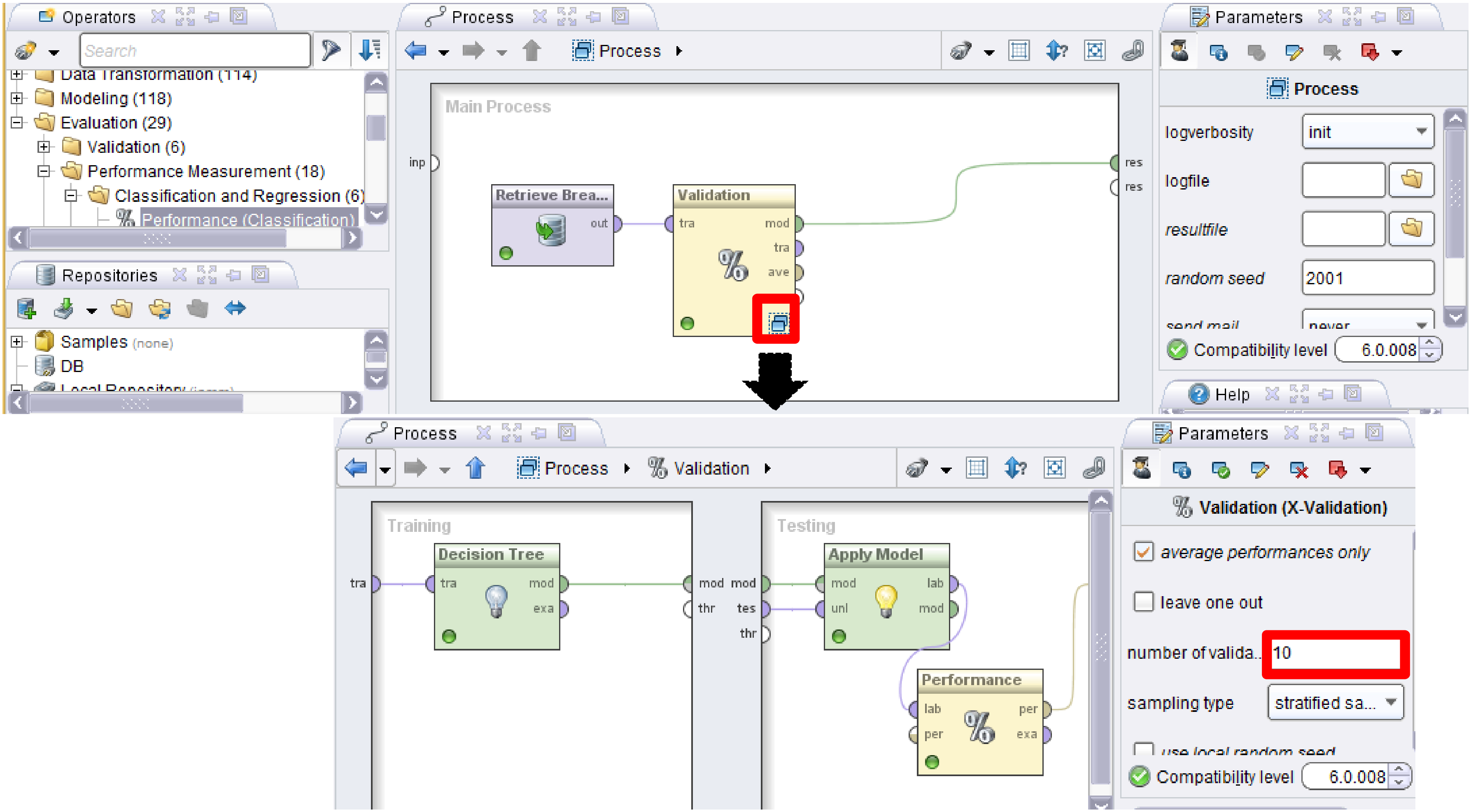

```
#Data Splitting
library(AppliedPredictiveModeling)
data(twoClassData)
str(predictors) #two independent variables
str(classes) # the target variable: a binary one

library(caret)
# Set the number seed so we can: (1) reproduce the results and; (2) test different algorithms with the same
partition.
set.seed(1)
# The parameter list is TRUE by default. If TRUE the numbers are returned as a list; If FALSE a matrix of row
# numbers is generated.
# createDataPartition does stratified sampling. See also the function sample (does not stratified sampling).
# The percent of data that will be allocated to the training set should be specified.
trainingRows <- createDataPartition(classes, p = .70, list=FALSE)
head(trainingRows)

trainPredictors <- predictors[trainingRows,]
trainClasses <- classes[trainingRows]
testPredictors <- predictors[-trainingRows,]
testClasses <- classes[-trainingRows]
str(trainPredictors)
str(testPredictors)
```

K-fold Cross Validation

1. Split randomly data into K disjoint sets.
2. Use one of the partitions as test set for evaluating the model generated using as training set the remaining k-1 partitions.
3. Repeat this process using always a different partition as test set. In the end use the predictions done for all partitions to evaluate the models thus obtained.

Operator in RapidMiner: X-Validation
Functions in R: createFolds (caret)

K-fold Cross Validation: RapidMiner



K-fold cross-validation: R


```
library(AppliedPredictiveModeling)
library(caret)
data(twoClassData)
set.seed(1)
cvSplits <- createFolds(classes, k=10, returnTrain = TRUE)
str(cvSplits)
fold1 <- cvSplits[[1]]
trainPredictors1 <- predictors[fold1,]
trainClasses1 <- classes[fold1]
testPredictors1 <- predictors[-fold1,]
testClasses1 <- classes[-fold1]
nrow(predictors)
nrow(trainPredictors1)
# This should be repeated k times. How?
# Use, for instance, the function knn3
```

K-fold cross-validation: R


```
#Example using 10-fold CV
library(AppliedPredictiveModeling)
library(caret)
data(twoClassData)
set.seed(1)
cvSplits <- createFolds(classes, k=10, returnTrain = TRUE)
fullPredictions <- c()
for (i in 1:10) {
  trainFold <- cvSplits[[i]]
  trainPredictors <- predictors[trainFold,]
  trainClasses <- classes[trainFold]
  testPredictors <- predictors[-trainFold,]
  testClasses <- classes[-trainFold]
  #Function to train the model: knn3 (caret). k is a parameter of this function.
  knnFit <- knn3(x=trainPredictors, y=trainClasses, k=5)
  testPredictions <- predict(knnFit, newdata = testPredictors, type="class")
  fullPredictions <- c(fullPredictions, testPredictions)
}
head(fullPredictions)
str(fullPredictions)
```

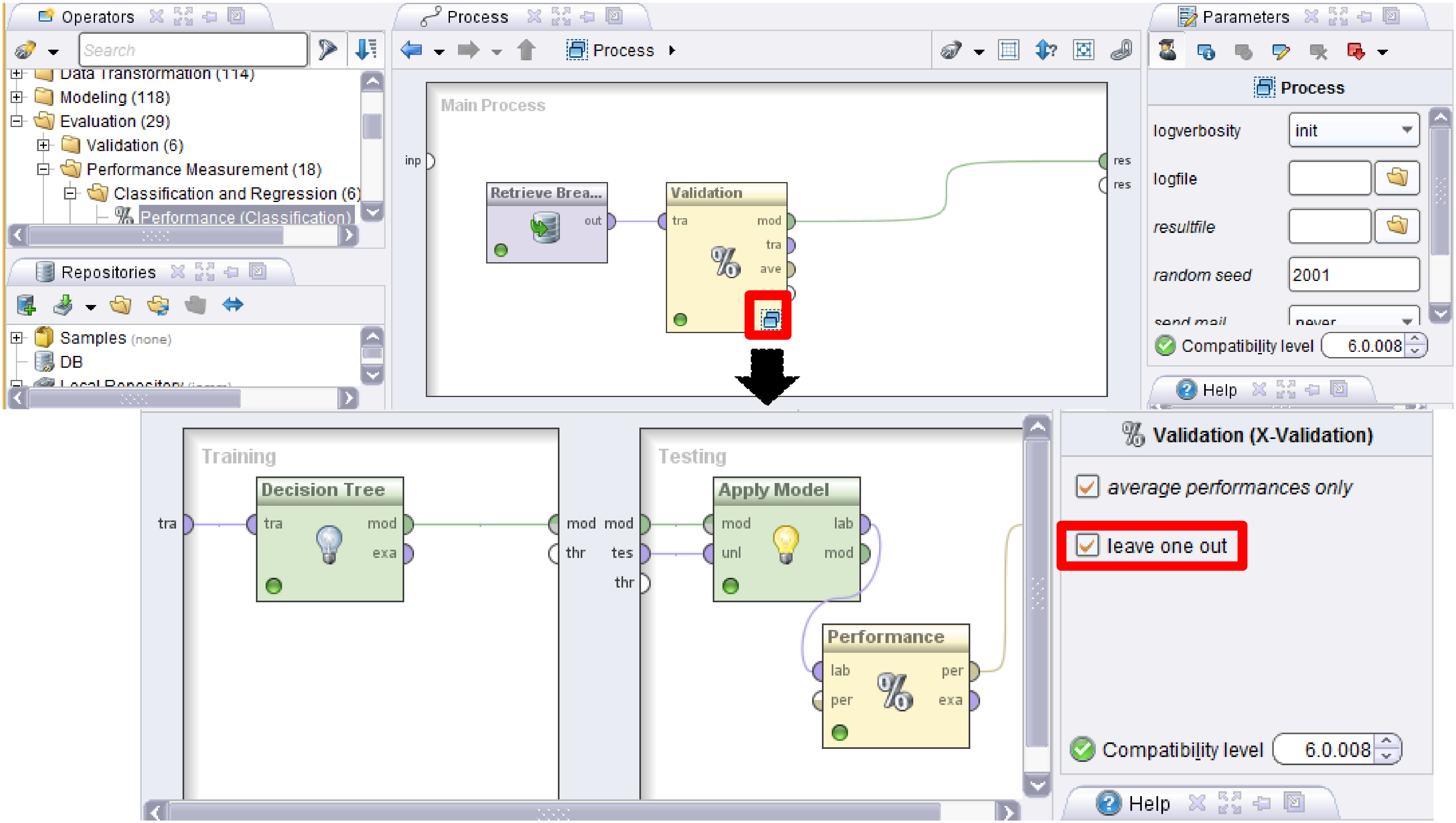
Leave one out

1. Use one of the instances to test the model that was generated with the remaining instances;
2. Store the prediction done for that instance;
3. Repeat these two steps for all instances and computes the evaluations carried out.

Note: leave-one-out = k-fold cross validation with *k*=#instances

Operator in RapidMiner: X-Validation with option leave one out
Functions in R: createFolds (caret)

Leave one out: Rapid Miner



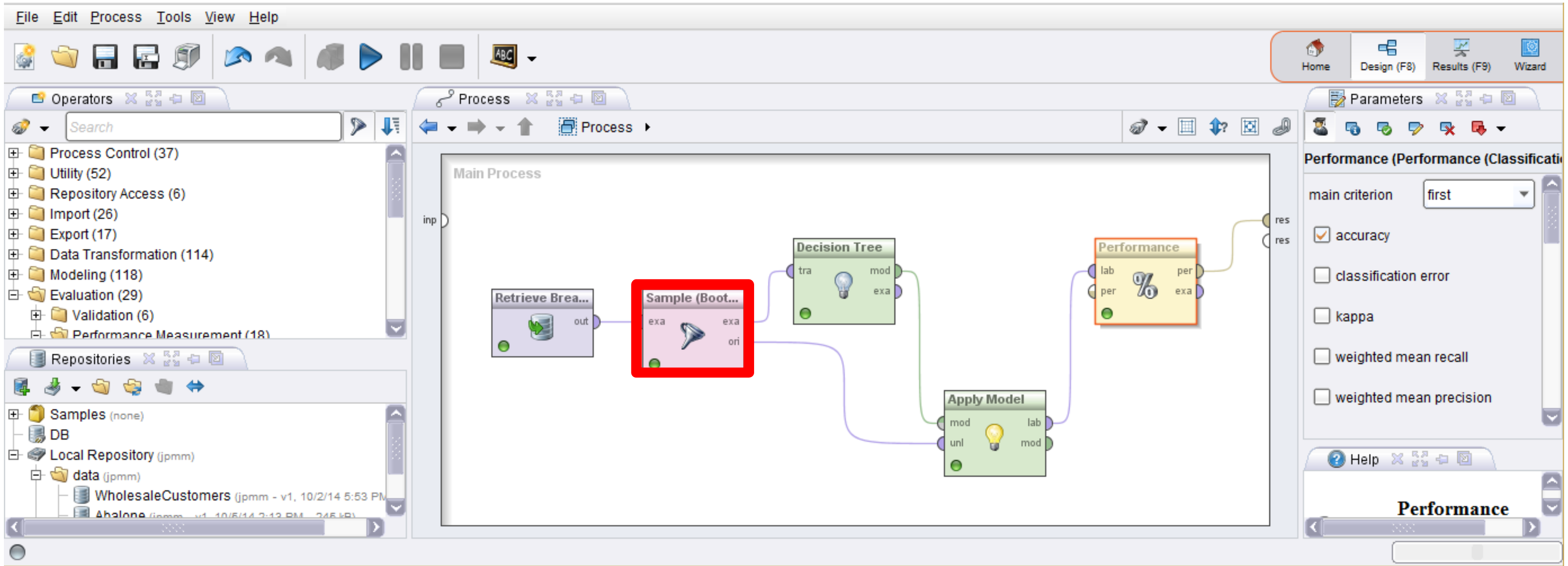
Leave one out: R


```
#Example using leave one out
library(AppliedPredictiveModeling)
library(caret)
data(twoClassData)
set.seed(1)
cvSplits <- createFolds(classes, k=length(classes), returnTrain = TRUE)
fullPredictions <- c()
for (i in 1:length(classes)) {
  trainFold <- cvSplits[[i]]
  trainPredictors <- predictors[trainFold,]
  trainClasses <- classes[trainFold]
  testPredictors <- predictors[-trainFold,]
  testClasses <- classes[-trainFold]
  #Function to train the model: knn3 (caret). k is a parameter of this function.
  knnFit <- knn3(x=trainPredictors, y=trainClasses, k=5)
  testPredictions <- predict(knnFit, newdata = testPredictors, type="class")
  fullPredictions <- c(fullPredictions, testPredictions)
}
head(fullPredictions)
str(fullPredictions)
```

Bootstrap

1. Collect a *m*-size (m is the #instances in the set) random sample with repetitions
2. It uses the selected instances for training
3. The model is tested in the out-of-bag instances, i.e., the instances that were not selected
   - in average, 36.8% of the instances are not selected using this procedure

Bootstrap: Rapid Miner



Bootstrap: R


```
library(AppliedPredictiveModeling)
library(caret)
data(twoClassData)
set.seed(1)
trainFold <- createResample(classes, times=length(classes), list=FALSE)
aux <- unique(trainFold)
trainPredictors <- predictors[trainFold,]
trainClasses <- classes[trainFold]
testPredictors <- predictors[-aux,]
testClasses <- classes[aux]
knnFit <- knn3(x=trainPredictors, y=trainClasses, k=5)
testPredictions <- predict(knnFit, newdata = testPredictors, type="class")
```

Data splitting recommendations

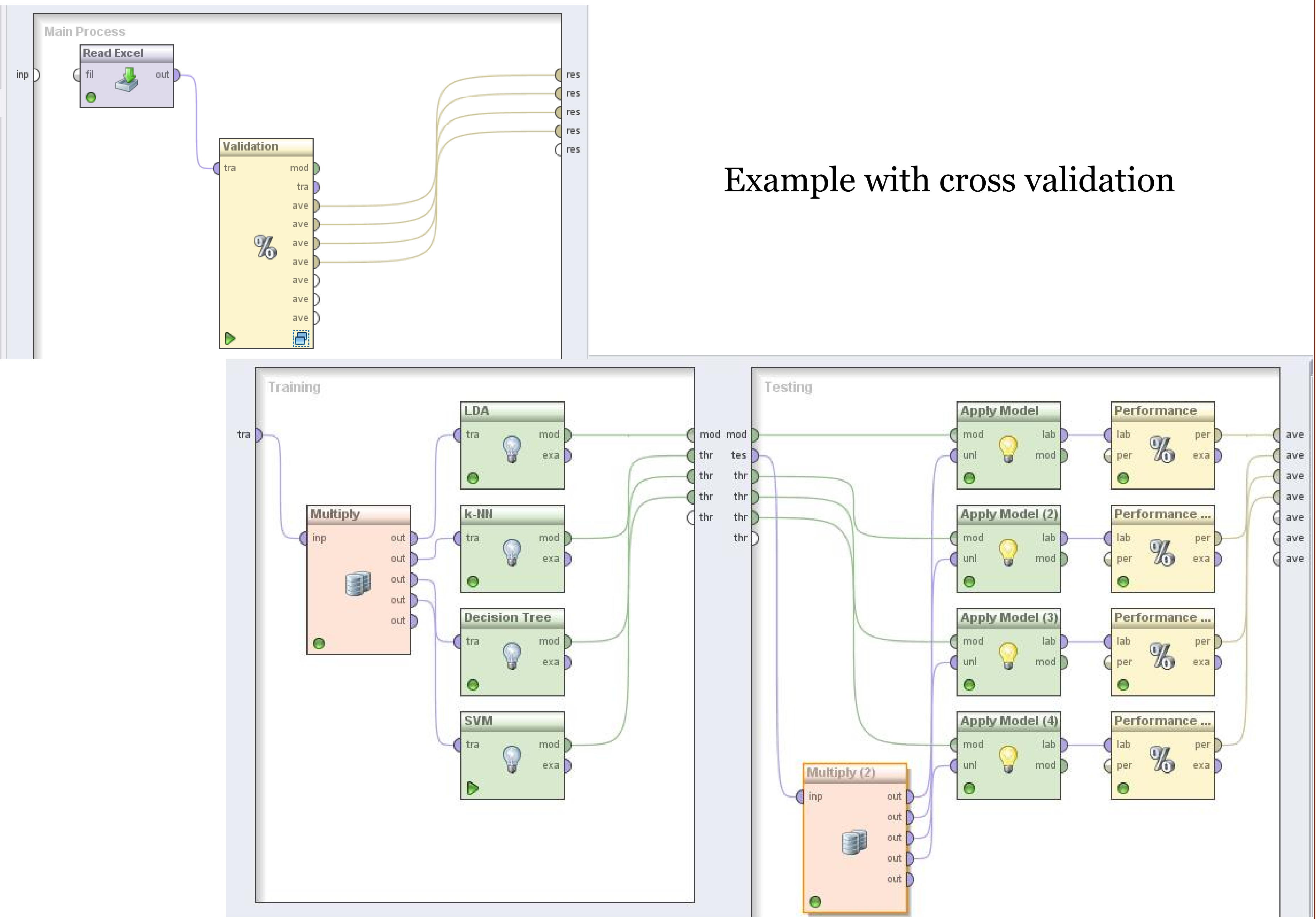
- No resampling method is uniformly better than another
- For small datasets use 10-fold cross validation with repetitions
  - see parameter *times* of *createDataPartition*
- Still for small datasets: if the goal is to choose between models instead of getting the best indicator of performance, bootstrap can be a good option
- For large datasets 10-fold cross validation is a good option

Comparing different algorithms

How to do it

- Given a dataset, it is important to guaranttee that:
  - All predictions are done using models generated with the same data
- Why:
  - Guaranteeing that the predictions are pairwised, the hypothesis test used to validate statisticaly the results

Example with cross validation



K-fold cross-validation: R


```
library(AppliedPredictiveModeling); library(caret); library(MASS); library(rpart); library(e1071)
fulllda <- c(); fullknn <- c(); fullrpart <- c(); fullsvm <- c()
data(twoClassData); set.seed(1)
cvSplits <- createFolds(classes, k=10, returnTrain = TRUE)
for (i in 1:10) {
  trainFold <- cvSplits[[i]]
  trainPredictors <- predictors[trainFold,]
  trainClasses <- classes[trainFold]
  testPredictors <- predictors[-trainFold,]
  testClasses <- classes[-trainFold]
  ldaFit <- lda(x=trainPredictors, grouping=trainClasses)          #lda (MASS)
  ldaPredictions <- predict(ldaFit, newdata = testPredictors, type="class")
  fulllda <- c(fulllda, ldaPredictions)
  knnFit <- knn3(x=trainPredictors, y=trainClasses, k=5)           # knn3 (caret)
  knnPredictions <- predict(knnFit, newdata = testPredictors, type="class")
  fullknn <- c(fullknn, knnPredictions)
  aux <- cbind(trainPredictors, trainClasses)
  rpartFit <- rpart(trainClasses ~ ., data=aux)                    #rpart (rpart)
  rpartPredictions <- predict(rpartFit, newdata = testPredictors, type="class")
  fullrpart <- c(fullrpart, rpartPredictions)
  svmFit <- svm(trainClasses ~ ., data=aux)                        #svm (e1071)
  svmPredictions <- predict(svmFit, newdata = testPredictors, type="class")
  fullsvm <- c(fullsvm, svmPredictions)}
```

Tuning parameters

Validation set

- When it is necessary to tune parameters, a portion of the dataset is taken in order to validate results. It is the **validation set**.
- Typically around 20% to 30% of the original dataset is used for validation.
- Then, the resamplig method is used on the remaining data, i.e., 80% to 70% of the original data.

Functions for parameter tuning


```
library(caret)
data(iris)
set.seed(1056)
svmFit <- train(x=iris[,-ncol(iris)], y=iris[,ncol(iris)], method = "svmRadial")
#If we want to normalize data we can do it using preProc
set.seed(1056)
svmFit <- train(x=iris[,-ncol(iris)], y=iris[,ncol(iris)], method = "svmRadial", preProc = c("center",
"scale"))
#We can test predefined parameter values from 2^-2 to 2^7, doing
set.seed(1056)
svmFit <- train(x=iris[,-ncol(iris)], y=iris[,ncol(iris)], method = "svmRadial", preProc = c("center",
"scale"), tuneLength=10)
#By default the train function uses bootstrap. Repeated 10-fold cross validation can be used
# through trainControl function
set.seed(1056)
svmFit <- train(x=iris[,-ncol(iris)], y=iris[,ncol(iris)], method = "svmRadial", preProc = c("center",
"scale"), tuneLength=10, trControl=trainControl(method="repeatedcv", repeats=5))
svmFit
```