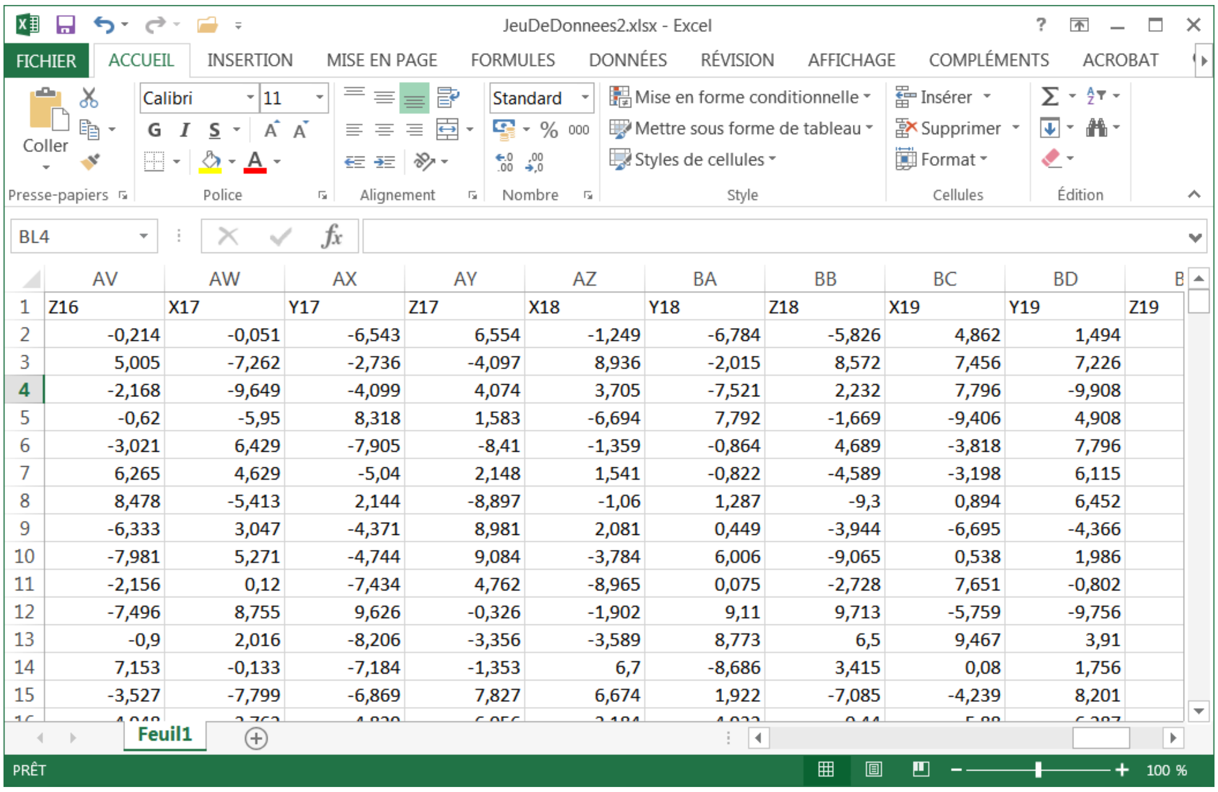
Case study for Random Forests in R

The objective of this tutorial is to show a complete example of the different steps necessary in a supervised learning process using Random Forests in R. More precisely, we will test the Random Forest for the recognition of positions from a dataset containing the joints of a skeleton. We'll see the whole process, starting from the creation of the dataset up to the creation of the forest and performance analysis of this forest. We will increase the dataset with new features and see how this changes the performance of the forest.

Kinect sends 30 frames per second. Each picture contains all the positions of the 20 joints of the person (or the 2 persons) tracked. For our study, we will start from an Excel file that will contain 10,000 records. Each record corresponds to the coordinates X, Y and Z of each of the 20 joints of the skeleton as well as a number corresponding to position. The coordinates are real random values between -10 and +10, and we will consider five different positions.

You obtain this kind of file:


JeuDeDonnees2.xlsx - Excel
FICHIER ACCUEIL INSERTION MISE EN PAGE FORMULES DONNÉES RÉVISION AFFICHAGE COMPLÉMENTS ACROBAT
BL4
Feuil1
PRÊT

	AV	AW	AX	AY	AZ	BA	BB	BC	BD
1	Z16	X17	Y17	Z17	X18	Y18	Z18	X19	Y19
2	-0,214	-0,051	-6,543	6,554	-1,249	-6,784	-5,826	4,862	1,494
3	5,005	-7,262	-2,736	-4,097	8,936	-2,015	8,572	7,456	7,226
4	-2,168	-9,649	-4,099	4,074	3,705	-7,521	2,232	7,796	-9,908
5	-0,62	-5,95	8,318	1,583	-6,694	7,792	-1,669	-9,406	4,908
6	-3,021	6,429	-7,905	-8,41	-1,359	-0,864	4,689	-3,818	7,796
7	6,265	4,629	-5,04	2,148	1,541	-0,822	-4,589	-3,198	6,115
8	8,478	-5,413	2,144	-8,897	-1,06	1,287	-9,3	0,894	6,452
9	-6,333	3,047	-4,371	8,981	2,081	0,449	-3,944	-6,695	-4,366
10	-7,981	5,271	-4,744	9,084	-3,784	6,006	-9,065	0,538	1,986
11	-2,156	0,12	-7,434	4,762	-8,965	0,075	-2,728	7,651	-0,802
12	-7,496	8,755	9,626	-0,326	-1,902	9,11	9,713	-5,759	-9,756
13	-0,9	2,016	-8,206	-3,356	-3,589	8,773	6,5	9,467	3,91
14	7,153	-0,133	-7,184	-1,353	6,7	-8,686	3,415	0,08	1,756
15	-3,527	-7,799	-6,869	7,827	6,674	1,922	-7,085	-4,239	8,201

In R, extract the data of your Excel file with the read.xlsx function.

```
mat<-read.xlsx(file="JeuDeDonnees2.xlsx",1)
```

We must now prepare the data set to create a train set and a test set. Choose randomly 80% of the records for the train and the remaining 20% for the test.

Christel Dartigues-Pallez
dartigue@unice.fr

To do this, create a vector using the **sample** function that will contain a random permutation of numbers 10000 records, then use the first 8000 values of this vector to create the matrix and being the last in 2000 to create the test matrix.

```
nbLignes <- nrow(mat)*0.8

rowIdx <- sample(nrow(mat),size = nrow(mat))

train = mat[rowIdx[1:(nbLignes)],]

test = mat[rowIdx[(nbLignes+1):length(rowIdx)],]
```

Create 2 factors responseTrain and responseTest that will contain the values of the last column of the train and test matrices.

```
responseTrain <- factor (train [,ncol(train)])

responseTest  <- factor (test  [,ncol(test)])
```

Then delete the last column in the test and train matrices.

```
train <- train [,-ncol(train)]

test <- test [,-ncol(test)]
```

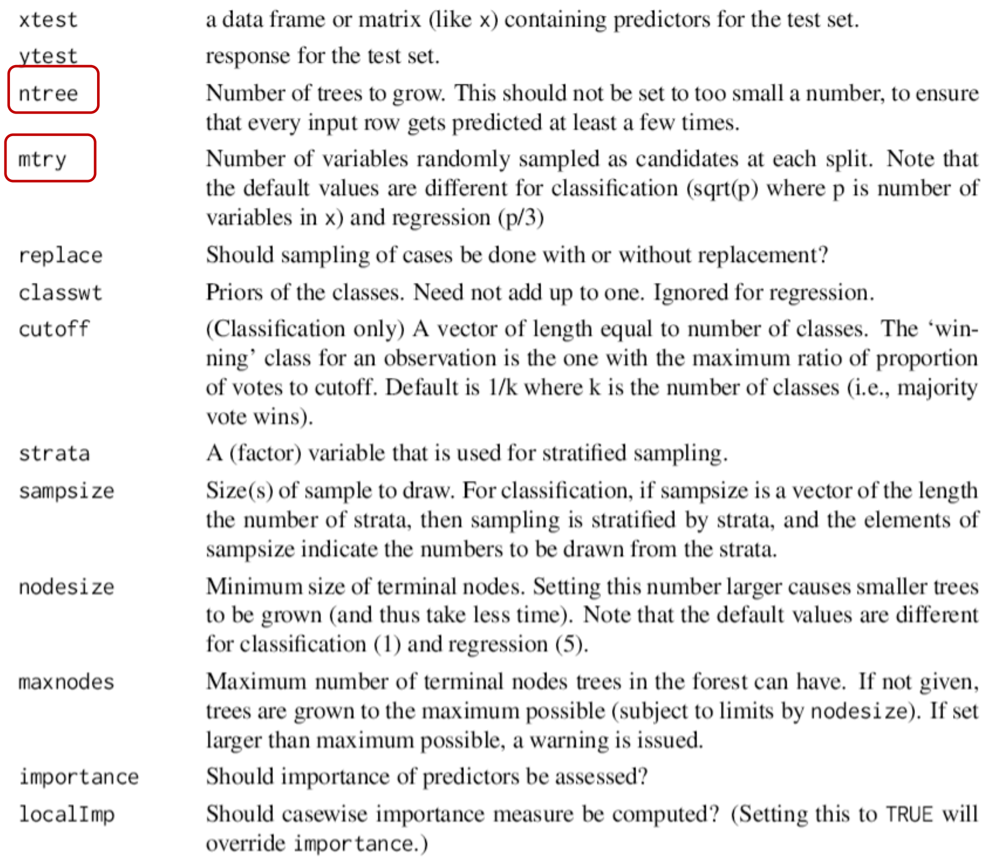
Create the forest using the tool randomForest

```
randomForest(x, y=NULL,  xtest=NULL, ytest=NULL, ntree=500,
             mtry=if (!is.null(y) && !is.factor(y))
             max(floor(ncol(x)/3), 1) else floor(sqrt(ncol(x))),
             replace=TRUE, classwt=NULL, cutoff, strata,
             sampsize = if (replace) nrow(x) else ceiling(.632*nrow(x)),
             nodesize = if (!is.null(y) && !is.factor(y)) 5 else 1,
             maxnodes = NULL,
             importance=FALSE, localImp=FALSE, nPerm=1,
             proximity, oob.prox=proximity,
             norm.votes=TRUE, do.trace=FALSE,
             keep.forest=!is.null(y) && is.null(xtest), corr.bias=FALSE,
             keep.inbag=FALSE, ...)
## S3 method for class 'randomForest'
print(x, ...)
```

You will use the arguments surrounded.

Arguments

[✓ data]	an optional data frame containing the variables in the model. By default the variables are taken from the environment which randomForest is called from.
subset	an index vector indicating which rows should be used. (NOTE: If given, this argument must be named.)
na.action	A function to specify the action to be taken if NAs are found. (NOTE: If given, this argument must be named.)
x, formula	a data frame or a matrix of predictors, or a formula describing the model to be fitted (for the print method, an randomForest object).
[✓ y]	A response vector. If a factor, classification is assumed, otherwise regression is assumed. If omitted, randomForest will run in unsupervised mode.



xtest	a data frame or matrix (like x) containing predictors for the test set.
ytest	response for the test set.
ntree	Number of trees to grow. This should not be set to too small a number, to ensure that every input row gets predicted at least a few times.
mtry	Number of variables randomly sampled as candidates at each split. Note that the default values are different for classification (sqrt(p) where p is number of variables in x) and regression (p/3)
replace	Should sampling of cases be done with or without replacement?
classwt	Priors of the classes. Need not add up to one. Ignored for regression.
cutoff	(Classification only) A vector of length equal to number of classes. The 'winning' class for an observation is the one with the maximum ratio of proportion of votes to cutoff. Default is 1/k where k is the number of classes (i.e., majority vote wins).
strata	A (factor) variable that is used for stratified sampling.
sampsize	Size(s) of sample to draw. For classification, if sampsize is a vector of the length the number of strata, then sampling is stratified by strata, and the elements of sampsize indicate the numbers to be drawn from the strata.
nodesize	Minimum size of terminal nodes. Setting this number larger causes smaller trees to be grown (and thus take less time). Note that the default values are different for classification (1) and regression (5).
maxnodes	Maximum number of terminal nodes trees in the forest can have. If not given, trees are grown to the maximum possible (subject to limits by nodesize). If set larger than maximum possible, a warning is issued.
importance	Should importance of predictors be assessed?
localImp	Should casewise importance measure be computed? (Setting this to TRUE will override importance.)

Now that the forest is created, we can visualize the information contained therein. Some ways to view information about the forest created:

The rfVersion1 statement (or print (rfVersion1) gives us the following information:

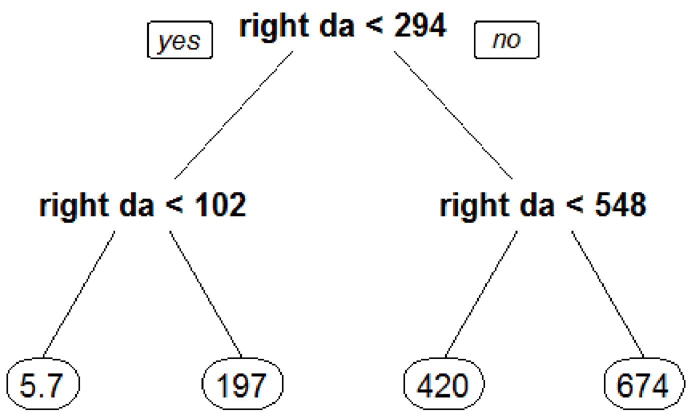
```
> rfVersion1

Call:
 randomForest(formula = responseTrain ~ ., data = train, prox = TRUE,      ntree = 100, mtry = 7, n$
               Type of random forest: classification
                     Number of trees: 100
No. of variables tried at each split: 7

        OOB estimate of  error rate: 80.5%
Confusion matrix:
    1   2   3   4   5 class.error
1 304 314 234 364 382   0.8097622
2 314 313 215 390 392   0.8072660
3 266 287 205 366 379   0.8636061
4 313 326 240 352 408   0.7852349
5 339 350 193 368 386   0.7640587
```

To extract a tree of the forest, we use the getTree function

getTree (rfVersion1, k, labelVar=TRUE)

We can transform the tree in an rpart object to be able to use the prp function : prp(rpart(myTree)). You obtain this kind of result:

Christel Dartigues-Pallez
dartigue@unice.fr



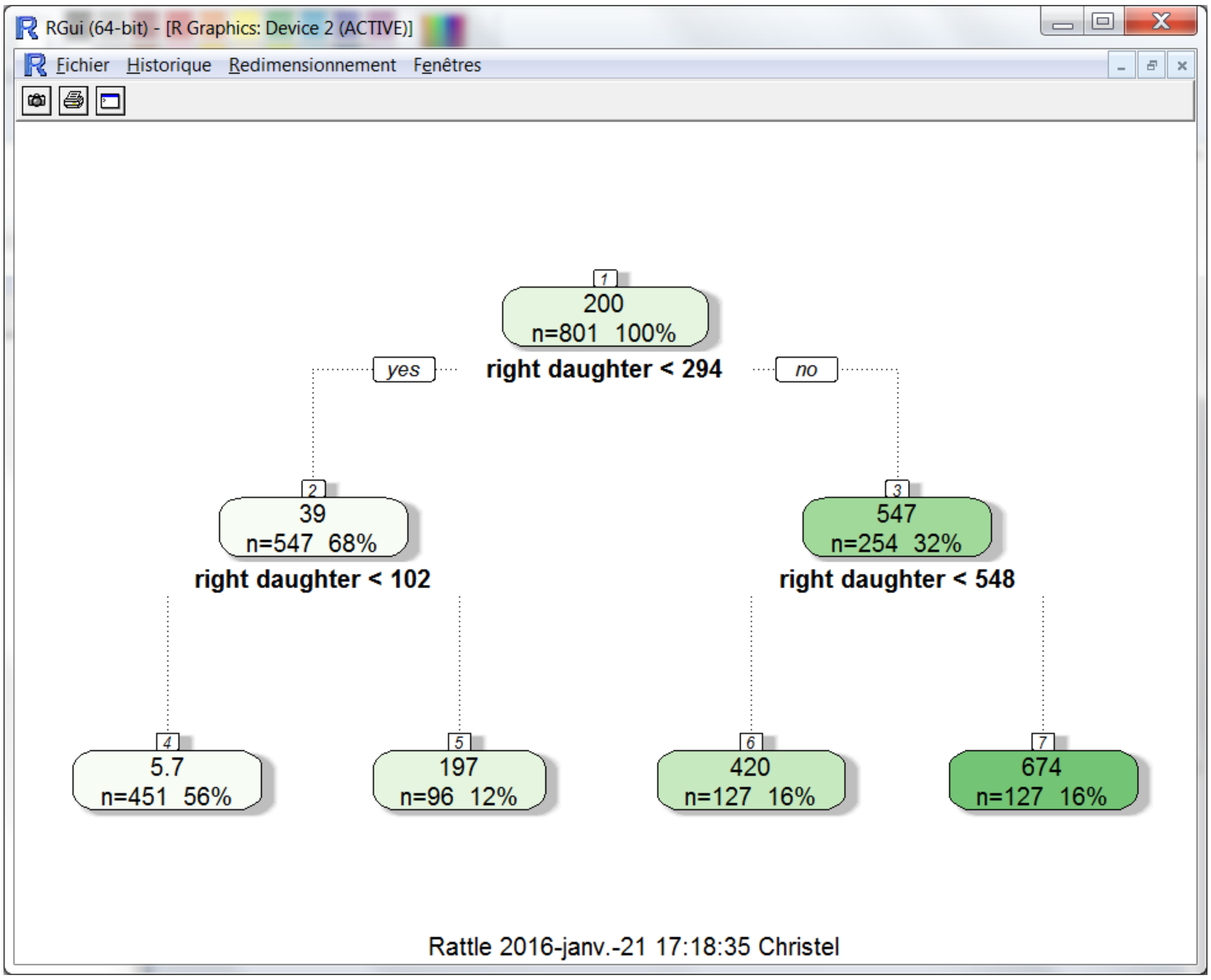
Specific arguments of prp:

type: determine the basic plotting style. 5 possible values (from 0 to 4)

extra : gives extra information at the nodes (in our case 0 or 1)

…

Another function allows you to visualized a tree: fancyRpartPlot (rpart(myTree))

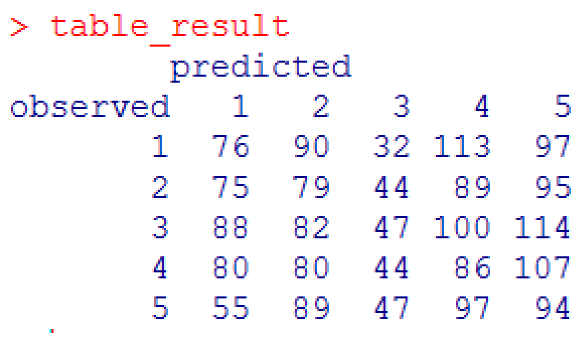


After creating the forest with the randomForest function, we can test it with the function predict, by using the test dataset:

```
predVersion1 <- predict(rfVersion1, test)
```

We can obtained the confusion matrix by creating a table containing the observed responses and the predictied values:

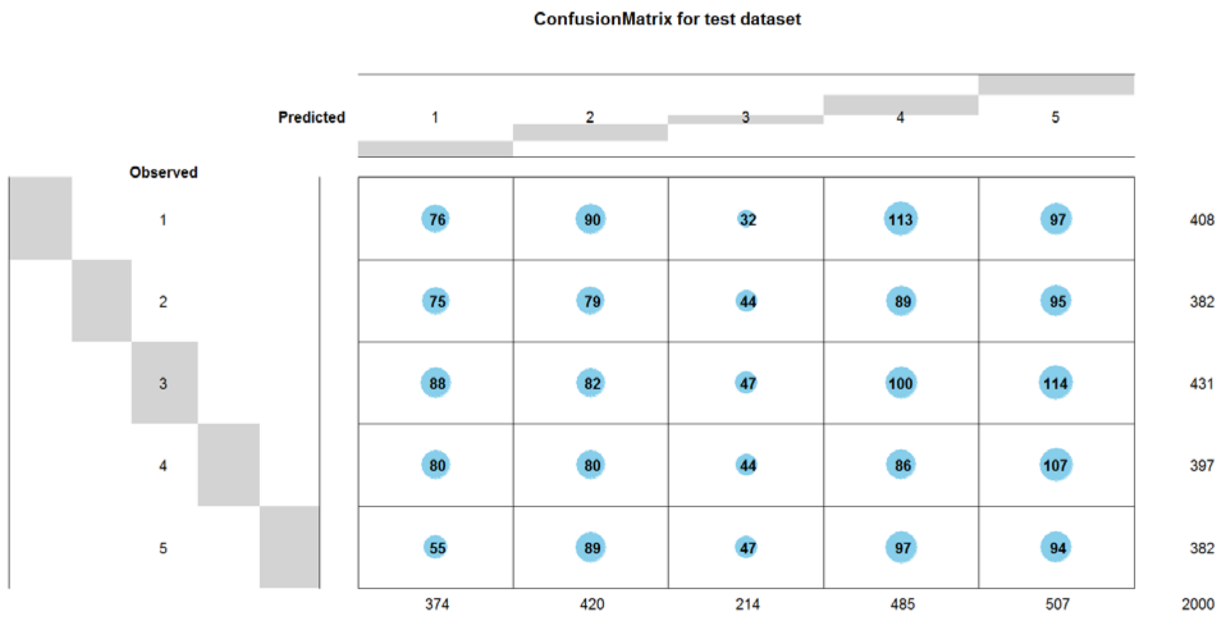
```
table_result <- table(observed = responseTest, predicted = predVersion1)
```



```
> table_result
        predicted
observed   1   2   3   4   5
       1  76  90  32 113  97
       2  75  79  44  89  95
       3  88  82  47 100 114
       4  80  80  44  86 107
       5  55  89  47  97  94
```

We can visualized a confusion matrix with the ballonplot function. You have to transform table_result in a data frame before using ballonplot:

```
table_result2 <- as.data.frame (table_result)

balloonplot (table_result2$predicted, table_result2$observed, table_result2$Freq,
xlab="Predicted",ylab="Observed", main="ConfusionMatrix for test dataset")
```



Build another forest with this dataset by changing the number of trees in the forest.

You will now build another dataset based on the first one by augmenting the number of features. You will add all the possible distances between the 20 joints.

Christel Dartigues-Pallez
dartigue@unice.fr

One you add those new features in your dataset, you can again separate this dataset in 2 parts, the train set and the test set and you can create your forest. Then use the predict function with the new test set. Compare the results obtained with this augmented dataset to the results of the first one and use the different function to visualize the forest.

Do the same steps with a third dataset containing the angles between the different joints.

Install the package : rattle. This package adds a GUI to R. It allows you to visualize your data and your models.

Christel Dartigues-Pallez
dartigue@unice.fr