06-23856 Evaluation Methods and Statistics
Autumn Semester 2019
Chris Baber

Practical 8

Using R for Human Performance Modelling

We have been using R for its statistical and data analysis functionality. The package is far more powerful than this. In today's practical we will look at two ways in which R could be used to inform human performance modelling. This is only an introduction but gives you a sense of how you might think about using R for research projects.

1. Creating a Random Walk model

This example is taken from chapter 2 of Farrell and Lewandowsky (2018) *Computational Modeling of Cognition and Behavior* [which I would recommend if you are interested in human performance and decision modelling]

The 'random walk' model is a very approximate description of how a person might sample information that is available. Let's assume you are responding to a set of stimuli and you have to choose one rather than the others. What features of the stimuli do you decide to use? We might assume that you build a 'mental model' to predict the salient features, or you might pick information at random and gradually develop a response strategy (which will, in turn, inform your 'mental model'). Either way, we would expect your reaction / choice time to speed up with practice. In their example, Farrell and Lewandowsky simplify things even further and assume that the 'features' are whether a line is slanted left / or right \ (although it doesn't really matter what the task is because our model is abstract, it can be useful to make things concrete so that you can think about the likely outcomes and the way the model might operate). In this instance, as you gather evidence (about line slope) you can be more confident in your Left or Right decision.

We began by specifying parameters for model, that is how many decision we want it to run for and how many samples will be taken for each decision. We also want to define how the model will operate, that is, how much evidence will be available during sampling. This is termed drift – the higher the drift, the more likely the model will be to move to a boundary. Setting drift to 0 will mean there is no evidence, so the model will be entirely random (we will begin with drift of 0 and then play around with this). We will also include a definition of noise, defined as the standard deviation of the set of evidence that we sample. We will call this 'sdrw'. Next we set the criterion level at which a decision response will be made. Finally, we will record the model performance in terms of latencies, responses and evidence.

So, set up of the model will be:

```
> nreps<-10000
> nsamples<-2000
> drift<-0.0
> sdrw<-0.3
> criterion<-3
> latencies<-rep(0,nreps)
> responses<-rep(0,nreps)
> evidence<-matrix(0,nreps,nsamples+1)
```

The model will run as a loop that applies an rnorm function to our model parameters, which is returned as a cumulative sum of absolute values for the data.

```
> for (i in c(1:nreps)) {
+     evidence[i,] <- cumsum(c(0,rnorm(nsamples,drift,sdrw)))
+     p <- which(abs(evidence[i,])>criterion)[1]
+     latencies[i] <- p
+     responses[i] <- sign(evidence[i,p])
+ }
```

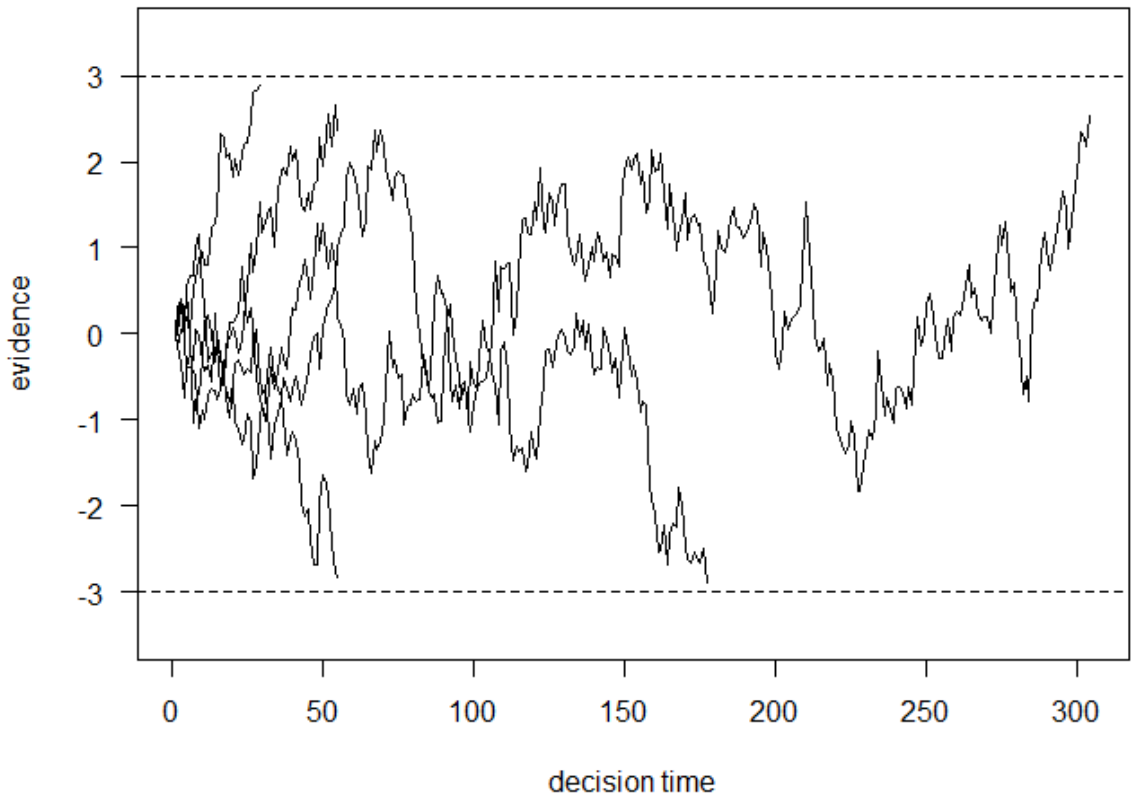
To look at the model, we define a number of paths to plot. Let's look at 5 lines.

```
> plot(1:max(latencies[1: tbpn])+10, type = "n", las=1, ylim=c(-criterion-.5,criterion+.5),
ylab="evidence", xlab="decision time")
> for (i in c (1:tbpn)){}
> for (i in c (1:tbpn)){}
> for (i in c (1:tbpn)){lines(evidence[i, 1: (latencies[i]-1)])}
```

We might also want to include the upper and lower criterion bands on our graph

```
> abline(h=c(criterion, -criterion), lty="dashed")
```

This should produce something like this:

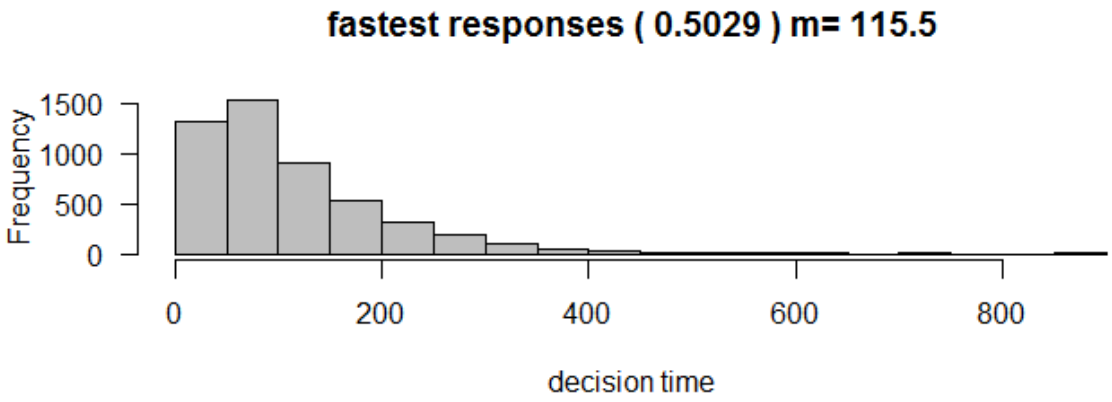


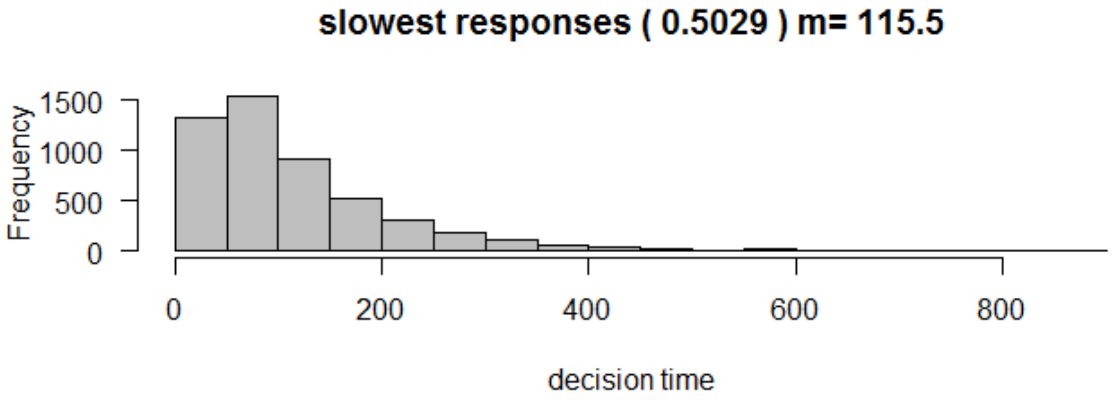
From our model, we want to know what the performance looks like (in terms of reaction time). So we can produce graphs from these data. Assume we want to have two graphs in our window, one for the fastest and one for the slowest times.

```
> par(mfrow=c(2,1))
> toprt<-latencies[responses>0]
> topprop<-length(toprt)/nreps
>botrt<-latencies[responses>0]
>botprop<-length(botrt)/nreps
```

```
> hist(toprt, col="gray", xlab="decision time", xlim=c(0,max(latencies)), main = paste("fastest responses (", as.numeric(topprop),") m=", as.character(signif(mean(toprt),4)),sep=" "),las=1)
```

```
> hist(botrt, col="gray", xlab="decision time", xlim=c(0,max(latencies)), main = paste("slowest responses (", as.numeric(botprop),") m=", as.character(signif(mean(botrt),4)),sep=" "),las=1)
```





2. Using R to create a Naïve Bayes Decision Model
This exercise uses the library e1071 from Technische Universitat Wien. You can find more information about it from the R package description here:

https://CRAN.R-project.org/package=e1071

e1071 provides a nice library of pre-built functions that could be very useful for classification. To use this, go to 'Tools' / 'Install packages'. Then, in the 'packages' bar type in e1071 and then Install. In your R code you can then call this by

>library(e1071)

We'll build a simple project that applies a naiveBayes classifier to cold and 'flu symptoms. This comes from an example developed by Ricardo Hertel (and I have put this into the Lab Class folder). First we need to create a set of data. Each line in this table represents an individual patient and their symptoms. Create this as a spreadsheet and save as .csv file. [note: for some reason, .csv files created in OpenOffice don't seem to open properly in R. Option 1 is to create the file in Excel and then save as ms-dos format. Option 2 is to create and save the file as .txt]. Let's call the file 'symptoms'

fatigue	s_nose	sneeze	s_throat	diag
F	F	T	T	COLD
T	T	T	F	COLD

T	T	F	T	COLD
T	F	F	T	COLD
T	T	F	T	COLD
F	T	T	T	COLD
T	F	F	T	COLD
T	T	F	F	COLD
F	T	F	F	COLD
F	F	T	T	COLD
T	F	F	T	FLU
T	F	F	T	FLU
F	T	T	F	FLU
T	T	F	F	FLU
T	F	T	F	FLU
T	F	F	F	FLU
T	F	F	T	ALLERGIES
F	T	T	F	ALLERGIES
F	F	T	F	ALLERGIES
T	F	T	F	ALLERGIES
F	T	T	T	UNKNOWN

We can then use the 'File' / 'Import dataset' function. A .csv file should be able to be imported as 'Text(base)'. [note: I found that working from an Excel file meant that the commands listed below do not work, unless I reformat the file and it is just easier to work directly from the .csv]

In our file, we have listed the symptoms as T or F. It is easier to work with these as numbers. So, we convert the logical values into numbers. It is not so important here what numbers are used but we want to make sure they differ between columns to help train the classifier.

```
fatigue<- 1 * symptoms [, 1, drop = F]
s_nose<- 2 * symptoms [, 2, drop = F]
sneeze<- 3 * symptoms [, 3, drop = F]
s_throat<- 4 * symptoms [, 4, drop = F]
```

We now want to create a matrix that we can use to train our classifier

```
num_patients<-cbind(fatigue, s_nose, sneeze, s_throat, symptoms [,5,drop=F])
```

This creates a fifth column in our matrix to indicate individual patients

```
patients_sympt_train<-num_patients[-21,-5]
```

This defines the 21 x 5 training set

```
illness<-factor(symptoms[-21,5])
```

This defines what we intend to model, i.e., something called 'illness'

Next, we build our classification model

```
patient_classifier<-naiveBayes(patients_sympt_train, illness)
```

We'll take a specific patient and see what their illness is – so you should be able to create queries by reusing this line and changing the patient number

patient_21_sympts<-num_patients[21,-5];

patient_prediction<-predict(patient_classifier, patient_21_sympts, type = "raw")

print(patient_prediction)

This should produce the output:

```
     ALLERGIES      COLD       FLU
[1,] 0.1245792 0.8370676 0.03835324
```

This tells us that, based on the classifiers, the probability of this patient having allergies is around 12%, having flu is around 4% and just having a cold is 84%.

Try running this to look at other patients in the sample. [note: you'll need to change the range in the 'patient_x_sympts' command to the patient id number [x, -5]

3. Decision Modelling using 'fast and frugal' heuristics.
This example is from https://cran.r-project.org/web/packages/FFTrees/vignettes/FFTrees_heart.html

It requires installation of the 'FFTrees' package (which you might not be able to do on machines in the lab, but you can do when you run R on your own computer).

```
> install.packages("FFTrees")
```

You will be asked (on another pop-up window) if you want to want to install dependent packages (agree to this and the installation will take a couple of minutes).

```
> library(FFTrees)

  O
 / \
F   O
   / \
  F   Trees 1.4.0

Nathaniel.D.Phillips.is@gmail.com
FFTrees.guide() opens the guide.
Warning message:
package 'FFTrees' was built under R version 3.5.3
```

In the FFTrees library, there is a 'heart disease' dataset, which we can use to train a decision model (' heart.train') and evaluate using 'heart.test'. In this example, 'diagnosis' is simply a 'yes / no' decisio n based on these data.

```
> heart.fft <- FFTrees(formula = diagnosis ~ .,          # Criterion and (all) predictors
+                      data = heart.train,               # Training data
+                      data.test = heart.test,           # Testing data
+                      main = "Heart Disease",           # General label
```

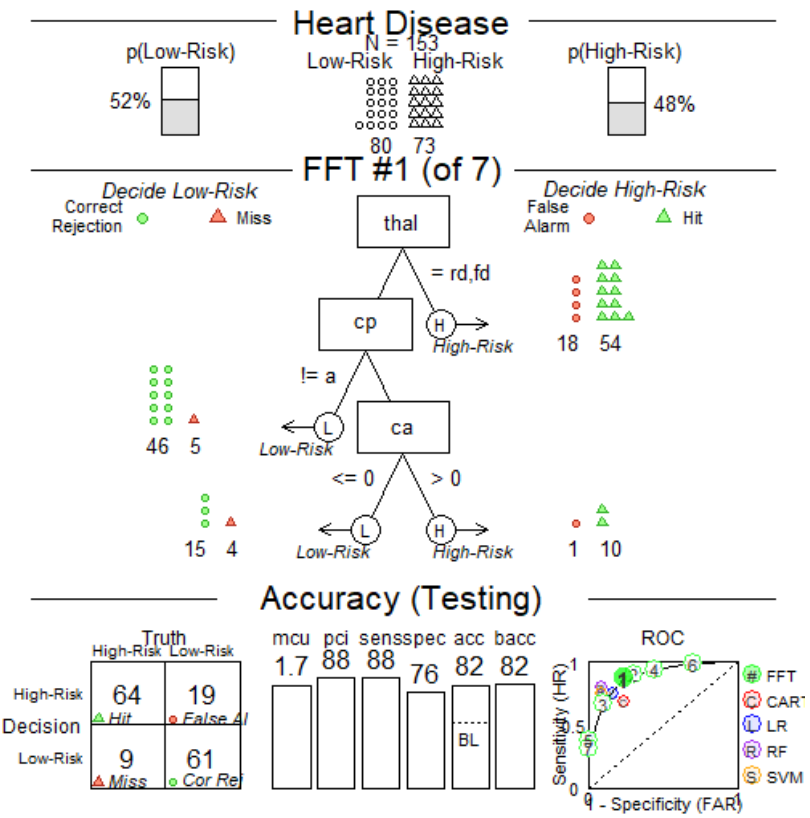
```
+                 decision.labels = c("Low-Risk", "High-Risk"))  # Labels for decisions
```

We can inspect the resulting model and its output…

```
> heart.fft
Heart Disease
FFT 1 (of 7) predicts diagnosis using 3 cues: {thal, cp, ca}

[1] If thal = {rd,fd}, decide High-Risk.
[2] If cp != {a}, decide Low-Risk.
[3] If ca <= 0, decide Low-Risk, otherwise, decide High-Risk.

                 train   test
cases       .n    150.00 153.00
hits        .hi    54.00  64.00
misses      .mi    12.00   9.00
false al    .fa    18.00  19.00
corr rej    .cr    66.00  61.00
speed       .mcu    1.74   1.73
frugality   .pci    0.88   0.88
cost        .cost   0.20   0.18
accuracy    .acc    0.80   0.82
balanced    .bacc   0.80   0.82
sensitivity .sens   0.82   0.88
specificity .spec   0.79   0.76

pars: algorithm = 'ifan', goal = 'wacc', goal.chase = 'wacc', sens.w = 0.5, max.levels = 4
```

To get a better sense of what is happening, we can visualise the decision tree using the plot function…

```
> plot(heart.fft,data="test")
```

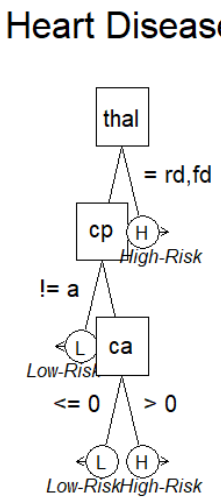


This contains a lot of information (and you need to consult the guide for more detail

>FFTrees.guide()

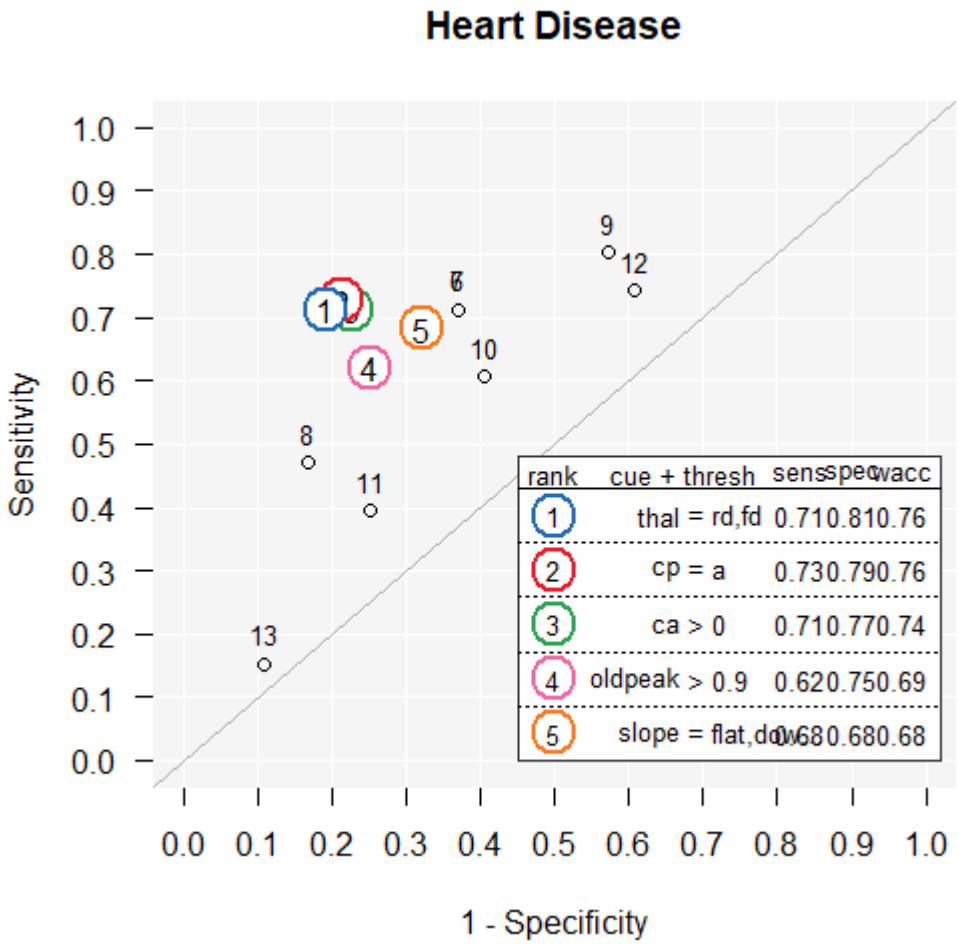
Or we can produce only the tree without the accompanying statistics…

```
> plot(heart.fft,stats=FALSE)
```



Finally, we can evaluate the models performance…

```
> plot(heart.fft,what ="cues")
```


Heart Disease
Sensitivity
1 - Specificity
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
9
12
10
8
11
13
rank cue + thresh sens spec wacc
1 thal = rd,fd 0.71 0.81 0.76
2 cp = a 0.73 0.79 0.76
3 ca > 0 0.71 0.77 0.74
4 oldpeak > 0.9 0.62 0.75 0.69
5 slope = flat,do 0.68 0.68 0.68