1 Topic

Description of Revolution R Community. Comparison of the performance with base R.

The R software is a fascinating project. It becomes a reference tool for the data mining process[1]. With the R package system, we can extend its features potentially at the infinite. Almost all existing statistical / data mining techniques are available in R.

But if there are many packages, there are very few projects which intend to improve the R core itself. The source code is freely available. In theory anyone can modify a part or even the whole software. Revolution Analytics proposes an improved version of R. It provides Revolution R Enterprise, it seems (according to their website) that: it improves dramatically the fastness of some calculations; it can handle very large database; it provides a visual development environment with a debugger. Unfortunately, this is a commercial tool. I could not check these features[2]. Fortunately, a community version is available. Of course, I have downloaded the tool to study its behavior.

Revolution R Community is a slightly improved version of the Base R. The enhancements are essentially related to the calculations performances: it incorporates the Intel Math Kernal libary, which is especially efficient for the matrix calculations; it can take advantage also, in some circumstances, from the power of the multi-core processors. Performance benchmarks[3] are available on the editor's website. The results are impressive. But we note that they are based on datasets generated artificially.

In this tutorial, we extend the benchmark to other data mining methods. We analyze the behavior of the Revolution R Community 5.0 - 64 bit version in various contexts: binary logistic regression (**glm**); linear discriminant analysis (**lda** from the MASS package); induction of decision trees (**rpart** from the rpart package); principal component analysis based on two different principles, the first one is based on the calculations of the eigenvalues and eigenvectors from the correlation matrix (**princomp**), the second one is done by a singular value decomposition of the data matrix (**prcomp**).

We use a binary version of the WAVEFORM dataset (Breiman and al. 1984). This is not either a real dataset. But it is realistic. We know which kind of pattern we can obtain; we know what are the relevant variables; etc. It is valuable in our context because we can modify easily its characteristics (number of instance, number of variables - by adding irrelevant attributes).

2 Dataset

We treat the "**wavebin.txt**" data file. We have already used the data generator previously (e.g. "Introduction to SAS proc logistic"); the R source code is described in "Logistic regression on large dataset". Here, we generate a dataset with n = 500,000 instances and p = 121 independent variables.

The main program is the following:

[1] KDNUGGETS Polls – « What Analytics, Data Mining, Big Data software you used in the past 12 months for a real project? » - http://www.kdnuggets.com/polls/2012/analytics-data-mining-big-data-software.html

[2] In fact, it can be freely downloadable for the members of academic institutions. But, we must be registered.

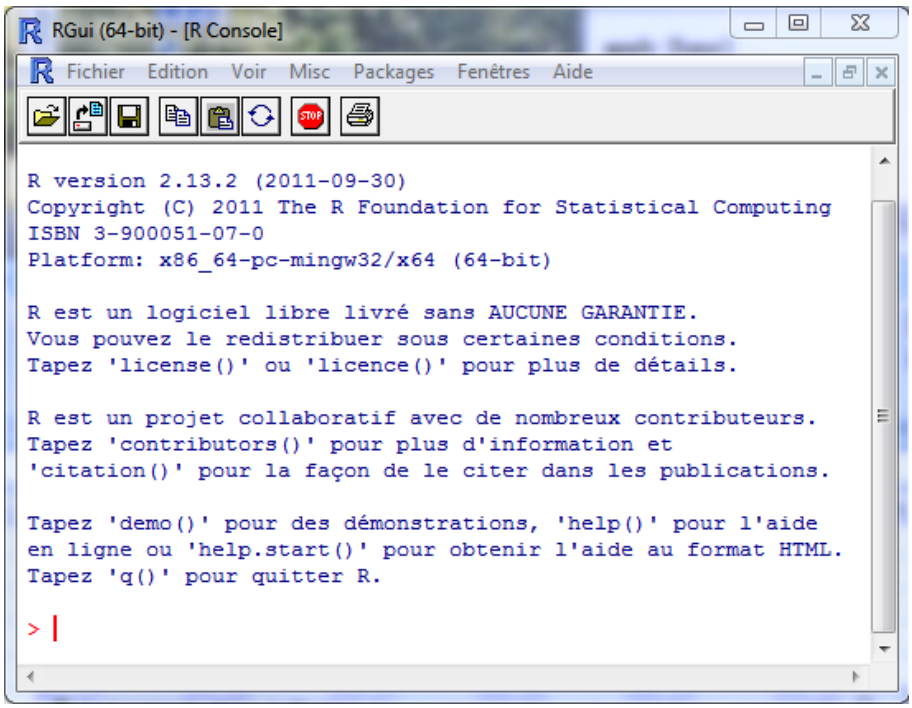
[3] http://www.revolutionanalytics.com/why-revolution-r/benchmarks.php

```
#generate and save a dataset
set.seed(1)
dataset.size <- 500000 #number of instances
nb.rnd <- 50 #number of random variables
nb.cor <- 50 #number of correlated variables
noise.level <- 1 #noise for correlated variables
data.wave <- generate.binary(dataset.size,nb.rnd,nb.cor,noise.level)
summary(data.wave)
#writting
write.table(data.wave,file="wavebin.txt",quote=F,sep="\t",row.names=F)
```

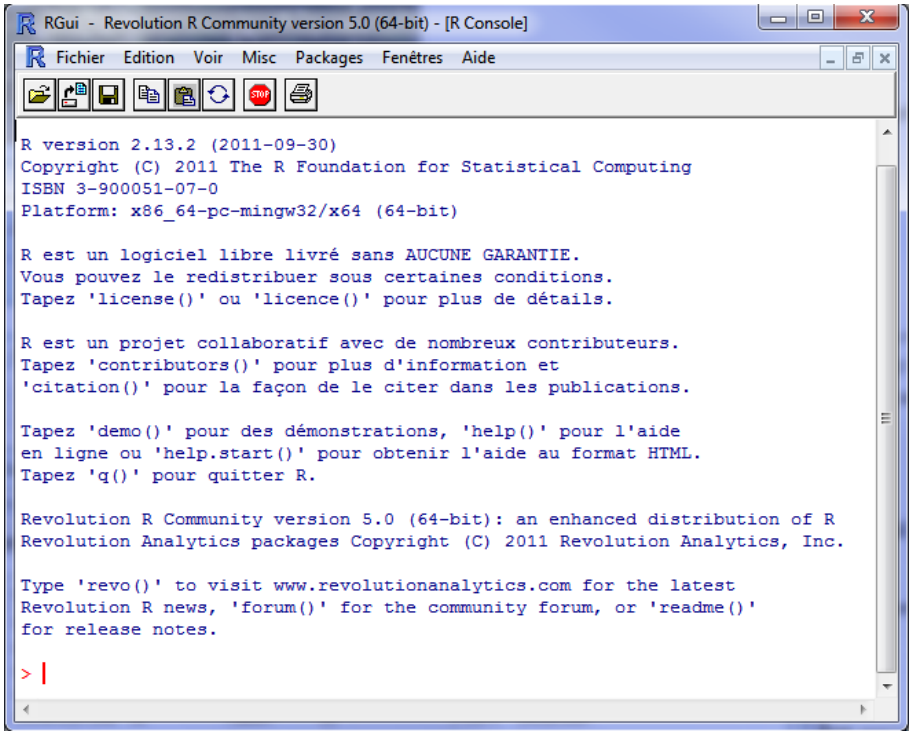
3 Studying the performances of Revolution Community

3.1 Revolution R Community 5.0

The performance measurements are only interesting if we have references values to compare them. In our case, we take as reference the results of the Base R 2.13.2 64-bit version. It corresponds to the Revolution R Community 5.0 that we used in this tutorial. Indeed, if we start the Base R, we can see the following in the main window.



And we have the following when we launch Revolution R:



The calculations should be faster for some kind of tasks according to the editor's website.

	Base R	Revolution R Community
Target Use	Open Source	Product Evaluation & Simple Prototyping
Software		
100% Compatible with R language	X	X
Certified for Stability		
Command-Line Programming	X	X
Getting Started Guide		X
Performance & Scalability		
Analyze larger data sets with 64-bit RAM	X	X
Optimized for Multi-processor workstations		**X**
Multi-threaded Math libraries		**X**
Parallel Programming (Single Workstation)		**X**
Out-of-the-Box Cluster-Ready		
"Big Data" Analysis with RevoScaleR		
Terabyte-Class File Structure		
Specialized "Big Data" Algorithms		
Integrated Web Services		
Scalable Web Services Platform		
User Interface		
IDE with Visual Debugger		
Technical Support		
Discussion Forums	X	X
Online Support	**Mailing List**	**Forum**
Email Support		
Phone Support		
Support for Base & Recommended R Packages	X	X
Authorized Training & Consulting		
Platforms		
Single User	X	X
Multi-User Server	X	
32-bit Windows	X	X
64-bit Windows	X	X
64-bit Red Hat Enterprise Linux	X	X

We try to determine the data mining methods which benefit to these optimizations in this tutorial.

3.2 The comparison protocol

The program used in R to compare the calculations capabilities is the following:

```
rm(list=ls())

#data importation
system.time(wave <- read.table(file="wavebin.txt",sep="\t",dec=".",header=T))

#select the 21 first columns for the pca analyses
wave.subset <- subset(wave, select = 1:21)

#package for lda
library(MASS)

#package for rpart
library(rpart)

#the benchmark function
benchmark <- function(){

  tps <- numeric(8)

  #logistic regression
  a <- system.time(model.glm <- glm(y ~ ., data = wave, family=binomial()))
  tps[1] <- a["elapsed"]
  print(model.glm)
  a <- system.time(predict(model.glm, newdata = wave, type = "response"))
  tps[2] <- a["elapsed"]

  #linear discriminant analysis
  a <- system.time(model.lda <- lda(y ~ ., data = wave))
  tps[3] <- a["elapsed"]
  print(model.lda)
  a <- system.time(predict(model.lda, newdata = wave))
  tps[4] <- a["elapsed"]

  #decision tree induction
  a <- system.time(model.rpart <- rpart(y ~ ., data = wave))
  tps[5] <- a["elapsed"]
  print(model.rpart)
  a <- system.time(predict(model.rpart, newdata = wave, type = "class"))
  tps[6] <- a["elapsed"]

  #principal component analysis (using eigen)
  a <- system.time(pca.princomp <- princomp(wave.subset, cor = T))
  tps[7] <- a["elapsed"]
  print(pca.princomp)

  #principal component analysis (using singular value decomposition)
```

```
  a <- system.time(pca.prcomp <- prcomp(wave.subset, center = T, scale = T))
  tps[8] <- a["elapsed"]
  print(pca.prcomp)

  #return the vector
  return(tps)
}

#repeat 10 times the computation time measurement
m <- replicate(10,benchmark())
print(m)

#mean and median
print(apply(m,1,mean))
print(apply(m,1,median))
```

Each data mining method is launched 10 times in order to obtain a reliable estimation of the calculation time. The median and the mean are very similar for our experiment, that means that there is not problem (outlier) during the experiments. The mean of the calculations time are the following:

n = 500.000, p = 121

Computation time (sec.)	R 2.13.2	Rev-R CE 5.0	Speedup
Data importation	75.0	75.0	
glm	288.6	279.4	0.03
predict.glm	6.6	6.5	0.02
lda	256.3	176.0	0.46
predict.lda	16.0	15.9	0.01
rpart	1395.4	1383.1	0.01
predict.rpart	6.9	6.8	0.01
princomp (21 variables)	4.9	3.8	0.29
prcomp (21 variables)	7.5	5.2	0.42

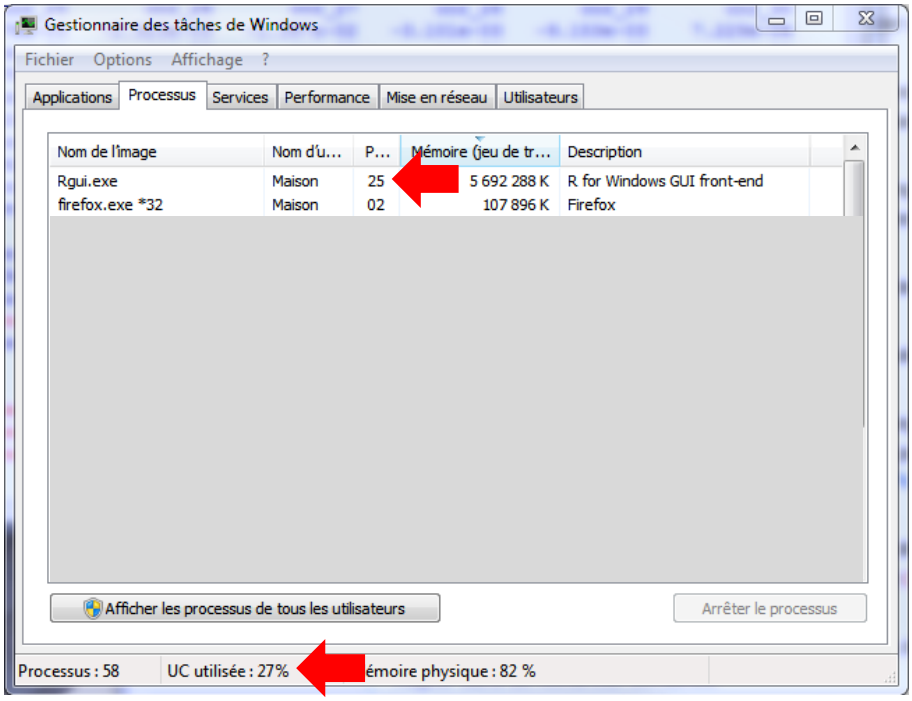
Speedup = Lower Time / Faster Time - 1

Linear discriminant analysis (lda) and principal component analysis. The enhancement is significant. But they are less spectacular to those shown on the editor's website. Obviously, the matrix calculations are improved. The singular value decomposition used by LDA and PRCOMP is more optimized than the matrix diagonalization (PRINCOMP).

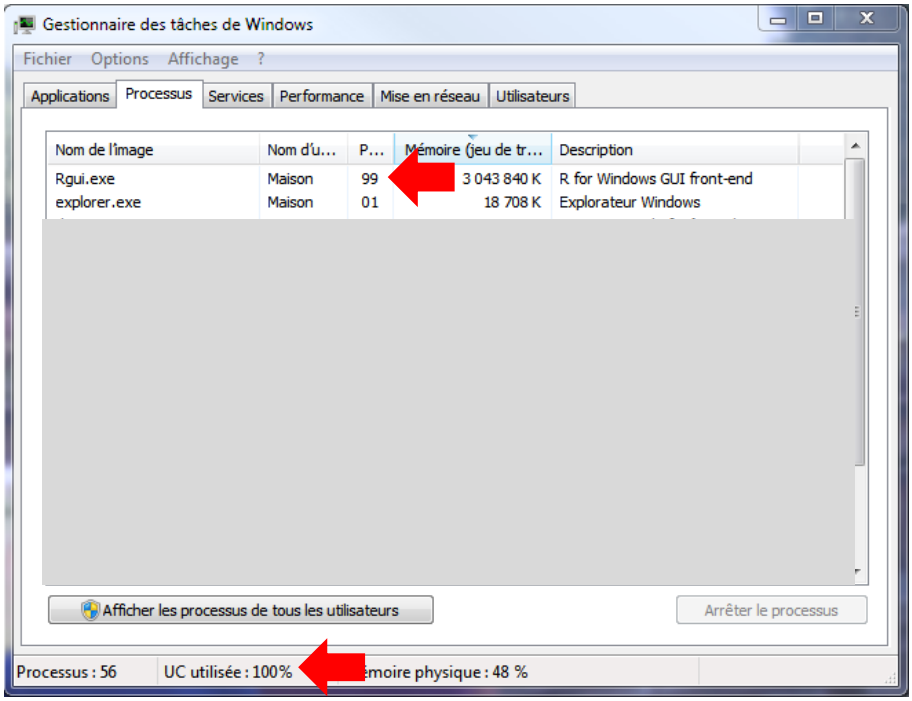
Logistic regression (glm). The improvement is not really spectacular. This is due to the characteristics of our dataset (large number of instances, and a moderate number of descriptors). The filling of the information matrix takes a lot of time in relation to its inversion (solve of a system of equations) during the log-likelihood optimization[4].

Another very interesting thing is the multithreading capabilities of the Revolution R. My computer is a quad-core (Quad Core Q9400 at 2.66 Ghz - Windows 7, 64-bit). Base R uses only one core as we see in the Windows task manager.

[4] **glm( )** uses the « Fisher scoring » approach - http://data.princeton.edu/wws509/notes/a1s1.html



But when we launch Revolution R Community, according to the internal operations handled, it takes advantage **intermittently** of all the available cores.



Decision tree induction. The improvement is negligible for the decision tree induction because: on the one hand, the process is not based on the matrix calculations; on the other hand, the rpart procedure, which is included in an external package, was probably not optimized. We can think that this behavior may be generalized to the other rule extraction processes.

4 Conclusion

Revolution R is uncommon because it tries to improve the core R itself. It is very different in this sense from all the other projects which try only to extend the R features with packages. We analyze the behavior of the Community version in this tutorial. We observe that we obtain significant improvements, especially for the methods based on the matrix calculations. But the results are not as spectacular as those described on the editor's website as we see in the screenshot below (Figure 1).

	Base R 2.13.2 32	Revolution R (1-core)	Revolution R (4-core)	Speedup (4 core)
Matrix Multiply	174.6 sec	31.0 sec	10.4 sec	15.8x
Cholesky Factorization	25.7 sec	5.0 sec	1.4 sec	17.6x
Singular Value Decomposition	67.6 sec	18.3 sec	7.8 sec	7.6x
Principal Components Analysis	266.2 sec	53.7 sec	20.1 sec	12.2x
Linear Discriminant Analysis	224.4 sec	84.4 sec	61.4 sec	2.7x

Speedup = Slower time / Faster Time - 1

Figure 1 - Source: http://www.revolutionanalytics.com/why-revolution-r/benchmarks.php

One of the possible reasons is the characteristics of my computer. To check this, we repeated the experiment using the code source available on the website. We obtained similar results.

Calc. time (sec)	Base R 2.13.2 - 64	Revolution R Community 5.0 - 64	Speedup
Matrix Multiply	206.1	7.5	26.5
Cholesky Factorization	25.7	1.6	15.4
Singular Value Decomposition	68.3	11.6	4.9
Principal Components Analysis	283.8	26.7	9.6
Lnear Discriminant Analysis	225.6	56.0	3.0

So, the main conclusion is that these are the dataset characteristics which mainly influence the performance enhancements. Revolution R is more efficient when the number of variables is increased compared to the number of instances. This is finally not surprising if we consider the characteristics of the matrix calculations utilized in these data mining algorithms.

To verify this idea, I launch again the experiments with a new version of the WAVE dataset: n = 5,000 instances; p = 221 variables. We obtain the following results:

n = 5000, p = 221			
Computation time (sec.)	**R 2.13.2**	**Rev-R CE 5.0**	**Speedup**
Data importation	1.6	1.6	
glm	6.07	4.27	0.42
predict.glm	0.13	0.13	-0.01
lda	4.04	2.59	0.56
predict.lda	0.19	0.19	0.03
rpart	5.45	5.44	0.00
predict.rpart	0.15	0.15	-0.03
princomp (21 variables)	0.03	0.03	0.00
prcomp (21 variables)	0.03	0.03	0.36

Compared to our first version of the wave dataset, the speedup is higher, especially for the logistic regression. The inversion of the information matrix takes more importance than the filling up during the learning process.

About the Enterprise version, its ability to handle very large database seems to be its main characteristic. That is moreover the reading of an article that caught my attention on the Revolution Analytics tools (http://blog.revolutionanalytics.com/2011/07/fast-logistic-regression-big-data.html and http://www.youtube.com/watch?v=KZHioV-DOD8). A logistic regression on 1.2 billion of rows in 75 seconds is really impressive!!!