

1 Topic

Handling large dataset using the "filehash" package for R.

The processing of very large datasets is a crucial problem in data mining. To handle them, we must avoid to load the whole dataset into memory. The idea is quite simple: (1) we write all or a part of the dataset on the disk in a binary file format to allow a direct access; (2) the machine learning algorithms must be modified to efficiently access the values stored on the disk. Thus, the characteristics of the computer are no longer a bottleneck for the handling of a large dataset.

The drawback of this approach, especially when the data access is not very quick, is that the processing time can increase in a prohibitive way. The data access strategy should be particularly effective. A caching system, among others, can speed up the data transfer.

In this tutorial, we describe the great "filehash" package for R (<http://cran.r-project.org/web/packages/filehash/index.html>). It allows to copy (to dump) any kind of R objects into a file. We can handle these objects without loading them into main memory. This is especially useful for the data frame object. Indeed, we can perform a statistical analysis with the usual functions directly from a database on the disk. The processing capacities are vastly improved and, in the same time, we will note that the increase in computation time remains moderate.

However, we will observe that when we still increase the size of the dataset, the computation is no longer possible for some functions, whereas the computer resources are not all used. This is the main drawback of this kind of "generic" approach. The modification of the learning algorithm and an efficient implementation are often necessary to take advantage of the direct access to the data stored in databases.

To evaluate the "filehash" solution, we analyze the memory occupation and the computation time, with and without utilization of the package, during the performing of decision tree learning with **rpart** (rpart package) and a linear discriminant analysis with **lda** (MASS package). We perform the same experiments using SIPINA. Indeed, it provides also a [swapping system](#) (the data is dumped from the main memory to temporary files) for the handling of very large dataset. We can then compare the performances of the various solutions.

2 Dataset

We use the Breiman's and al. **wave** dataset (1984). The data file contains 2,000,000 instances. We already used this file elsewhere (<http://data-mining-tutorials.blogspot.com/2009/10/local-sampling-approach-for-decision.html>). There are 21 predictive variables in the data file. Depending on the configuration, we use a part or all of these variables.

The data file is in a tab delimited text file format. We include the R source code into the archive (<http://eric.univ-lyon2.fr/~ricco/tanagra/fichiers/wave2M.txt.zip>).

3 Standard analysis using R

3.1 Performing the analysis in memory (5 predictive variables)

In a first time, we load the dataset and we compute some descriptive statistics indicators. In a second time, we launch the **rpart** (decision tree) and **lda** (linear discriminant analysis) functions. We use only 5 predictive variables (V07, V11, V17, V06, and V10). Then, we compute the prediction of the model on the dataset. We get the confusion matrix and the resubstitution error rate.

```
#clear the memory
rm (list=ls())
#loading the data file
setwd("D:/DataMining/Databases_for_mining/dataset_for_soft_dev_and_comparison/swap_strategy")
wave.data <- read.table(file = "wave2M.txt", sep = "\t", dec = ".", header = T)
#checking the data
print(summary(wave.data))
#learning a decision tree - rpart
library(rpart)
settings <- rpart.control(xval=0,minsplit=100000,minbucket=100000,maxsurrogate=0,cp=0)
wave.tree <- rpart(Onde ~ V07+V11+V17+V06+V10, data = wave.data, control = settings)
print(wave.tree)
#linear discriminant analysis
library(MASS)
wave.lda <- lda(Onde ~ V07+V11+V17+V06+V10, data = wave.data)
print(wave.lda)
#prediction
pred.tree <- predict(wave.tree, newdata = wave.data, type = "class")
pred.lda <- predict(wave.lda, newdata = wave.data)$class
#confusion matrix
mc1 <- table(wave.data$Onde,pred.tree)
err1 <- 1.0 - (mc1[1,1]+mc1[2,2]+mc1[3,3])/sum(mc1)
mc2 <- table(wave.data$Onde,pred.lda)
err2 <- 1.0 - (mc2[1,1]+mc2[2,2]+mc2[3,3])/sum(mc2)
```

The calculation was possible on my computer. We obtain the following decision tree. The resubstitution error rate is 27.34%.

```

R Console
> print(wave.tree)
n= 2000000
node), split, n, loss
* denotes term

1) root 2000000 1332587 C (0.333328500 0.332965000 0.333706500)
2) V07>=2.375 1045218 481172 B (0.384956057 0.539644361 0.075399582)
4) V11< 3.025 527889 186894 A (0.645959662 0.341871113 0.012169225)
8) V17>=0.795 262791 44000 A (0.832566564 0.145446381 0.021987054) *
9) V17< 0.795 265098 122850 B (0.460976695 0.536586470 0.002436835)
18) V10< 3.065 133207 52040 A (0.609329840 0.388981060 0.001689100) *
19) V10>=3.065 131891 41458 B (0.311143293 0.685664678 0.003192030) *
5) V11>=3.025 517329 133753 B (0.118624705 0.741454664 0.139920631) *
3) V07< 2.375 954782 366178 C (0.276810832 0.106709175 0.616479992)
6) V11< 2.935 385893 167935 A (0.564814599 0.006380007 0.428805394)
12) V06>=1.265 126199 26549 A (0.789625908 0.017337697 0.193036395) *
13) V06< 1.265 259694 118582 C (0.455566936 0.001055088 0.543377976)
26) V11< 2.005 128416 56564 A (0.559525293 0.000202467 0.440272240) *
27) V11>=2.005 131278 46704 C (0.353874983 0.001889121 0.644235896) *
7) V11>=2.935 568889 145758 C (0.081449984 0.174765200 0.743784816) *

R Console
> print(mc1)
pred.tree
  A      B      C
A 471460 102405 92792
B 92251 474009 99670
C 86902 72806 507705
> err1 <- 1.0 - (mc1[1,1]+mc1[2,2]+mc1[3,3])/sum(mc1)
> print(err1)
[1] 0.273413

```

The computation is also possible for the linear discriminant analysis. The error rate is 20.75%.

```

R Console
> print(wave.lda)
Call:
lda(Onde ~ V07 + V11 + V17 + V06 + V10)

Prior probabilities of groups:
  A      B      C
0.3333285 0.3329650 0.3337065

Group means:
      V07      V11      V17      V06      V10
A 2.998920 2.001229 2.001245378 2.5021677 2.001537
B 4.000516 4.002317 -0.001723755 2.9952801 4.000647
C 1.001623 4.001476 1.998217116 0.4985761 3.001757

Coefficients of linear discriminants:
      LD1      LD2
V07 -0.3194377166 0.1096921
V11 -0.0003456258 -0.4206426
V17 0.2125826902 0.2090596
V06 -0.2655422770 0.1658616
V10 -0.1049753976 -0.3138073

Proportion of trace:
  LD1  LD2
0.5451 0.4549

R Console
> print(mc2)
pred.lda
  A      B      C
A 492349 86698 87610
B 61328 546133 58469
C 62000 58898 546515
> err1 <- 1.0 - (mc2[1,1]+mc2[2,2]+mc2[3,3])/sum(mc2)
> print(err1)
[1] 0.2075015

```

We summarize the results in a table (Figure 1). We provide the computation time and the memory occupation during the processing. About the memory occupation, we use the `gc(.)` function. Last, we retrieve also the memory occupation measured by the windows task manager.

Processing into memory		
Learning phase	Time (sec.)	Mem. Occup. MB (Ncells+Vcells)
Connecting / loading	51.57	331.8
Descriptive statistics	5.54	332
rpart	38.52	427.7
lda	11.79	428.1
Saving decision tree	--	--
Saving lda	--	--
Prediction phase		
Connection / loading	--	--
Prediction with a tree	3.9	443.4
Prediction with a lda	38.41	451.1
Cinfusion matrix	1.15	451.1
Windows mem. Occup (MB)	--	591

Figure 1 - Performances – Processing into main memory

```

> gc()
      used (Mb) gc trigger      (Mb) max used   (Mb)
Ncells 2150671 57.5   6910418 184.6 6792403 181.4
Vcells 51578175 393.6 144293183 1100.9 143580345 1095.5

```

The `gc()` function shows that we can handle a larger dataset. The current processing does not use all the available memory.

3.2 Processing into memory with all the predictive variables

We want to repeat the same experiment but using all the 21 predictive variables now. The source code is the following.

```

#clear the memory
rm(list=ls())
#loading the data file
setwd("D:/DataMining/Databases_for_mining/dataset_for_soft_dev_and_comparison/swap_strategy")
wave.data <- read.table(file = "wave2M.txt", sep = "\t", dec = ".", header = T)
#checking the data
print(summary(wave.data))
#learning a decision tree - rpart
library(rpart)
settings <- rpart.control(xval=0, minsplit=100000, minbucket=100000, maxsurrogate=0, cp=0)
wave.tree <- rpart(Onde ~ ., data = wave.data, control = settings)
print(wave.tree)
#linear discriminant analysis
library(MASS)
wave.lda <- lda(Onde ~ ., data = wave.data)
print(wave.lda)

```

It seems that, in the standard configuration of our computer, the processing is not possible. An exception is raised because the statistical functions want to allocate a large vector.

```

R Console
> system.time(wave.cree <- rpart(Onde ~ ., data = wave.data, control = $
Erreur : impossible d'allouer un vecteur de taille 15.3 Mo
De plus : Warning messages:
1: In structure(list(message = as.character(message), call = call), :
  Reached total allocation of 1535Mb: see help(memory.size)
2: In structure(list(message = as.character(message), call = call), :
  Reached total allocation of 1535Mb: see help(memory.size)
3: In attributes(.Data) <- c(attributes(.Data), attrib) :
  Reached total allocation of 1535Mb: see help(memory.size)
4: In attributes(.Data) <- c(attributes(.Data), attrib) :
  Reached total allocation of 1535Mb: see help(memory.size)
5: In attributes(.Data) <- c(attributes(.Data), attrib) :
  Reached total allocation of 1535Mb: see help(memory.size)
6: In attributes(.Data) <- c(attributes(.Data), attrib) :
  Reached total allocation of 1535Mb: see help(memory.size)
Timing stopped at: 96.53 1.14 98.01

R Console
> system.time(wave.lda <- lda(Onde ~ ., data = wave.data))
Erreur : impossible d'allouer un vecteur de taille 15.3 Mo
De plus : Warning messages:
1: In oldClass(x) <- oldClass(xx) :
  Reached total allocation of 1535Mb: see help(memory.size)
2: In oldClass(x) <- oldClass(xx) :
  Reached total allocation of 1535Mb: see help(memory.size)
3: In oldClass(x) <- oldClass(xx) :
  Reached total allocation of 1535Mb: see help(memory.size)
4: In oldClass(x) <- oldClass(xx) :
  Reached total allocation of 1535Mb: see help(memory.size)
Timing stopped at: 91.31 0.57 92.66

```

It would require extensive knowledge about R to understand the error message and to provide solutions to overcome this limitation. In our case, we prefer another strategy. We hope that the "filehash" library will enable us to make the analysis directly from the data file on disk without having to load the dataset into memory. Thus, the memory limitation will not be a barrier.

4 Using the “filehash” package in R

The « filehash » package (<http://cran.r-project.org/web/packages/filehash/index.html>) allows to dump any kind of R object into a file. In the first time, we perform the analysis with the 5 predictive variables. We evaluate mainly the computation time and the memory occupation. In the second time, we check if it will be possible to perform the analysis with all the predictive variables (21).

4.1 Creating the file «.db » for the storage of a data frame object

First, we must install the “filehash” package. Then, the following commands allow to load the dataset into a data frame (R object), and to store this one in the file “**wave.dataset.db**” on the disk.

The processing time can be very long. Be patient and do not interrupt operations.

```

#clear the memory
rm (list=ls())

```

```
#loading the data file into a data.frame "wave.data" in memory
setwd("D:/DataMining/Databases_for_mining/dataset_for_soft_dev_and_comparison/swap_strategy")
wave.data <- read.table(file = "wave2M.txt", sep = "\t", dec = ".", header = T)
#creating a database and insert the data.frame into the database
#loading the "filehash" package
library(filehash)
#creating a database with the name "wave.dataset.db"
dbCreate("wave.dataset.db")
#initialize the database
db <- dbInit("wave.dataset.db")
#insert the data.frame into the database - the key is "wave.data"
dbInsert(db,"wave.data",wave.data)
```

Here are some indications about the size of the files. The original tab delimited text file size is 213,988 KB. The size of the "filehash" binary file is 335,939 KB. This increasing is not surprising. Indeed, the file size depends on the coding of the values in memory. The real value is coded in 8 bytes, the discrete value (factor type in R) in 4 bytes. Thus, for 2,000,000 instances with 21 continuous predictive attributes and one discrete target attribute, we obtain

$$(2,000,000 \times 21 \times 8 + 2,000,000 \times 1 \times 4) / 1024 \approx 335,937.5 \text{ KB}$$

We add the space for the list structure of the data frame object.

4.2 Processing the ".db" data file and storing the models

We want to perform the same analysis as before with the 5 predictive variables (section 3.1). But now, we do not load the whole dataset in memory. We process directly the dataset on the disk.

```
#clear the memory
rm (list=ls())
#connecting the database
#loading the "filehash" package
library(filehash)
#initialize the database
setwd("D:/DataMining/Databases_for_mining/dataset_for_soft_dev_and_comparison/swap_strategy")
db <- dbInit("wave.dataset.db")
#attach the database to an environnement
wave.env <- db2env(db)
#checking the dataset
print(summary(wave.env$wave.data))
#learning a decision tree - rpart
library(rpart)
settings <- rpart.control(xval=0,minsplit=100000,minbucket=100000,maxsurrogate=0,cp=0)
wave.tree <- rpart(Onde ~ V07+V11+V17+V06+V10, data = wave.env$wave.data, control = settings)
print(wave.tree)
#linear discriminant analysis
library(MASS)
wave.lda <- lda(Onde ~ V07+V11+V17+V06+V10, data = wave.env$wave.data)
print(wave.lda)
#saving the models into a database
```

```
#saving the tree
dumpObjects(wave.tree, dbName = "wave.models.db")
#saving the lda
dumpObjects(wave.lda, dbName = "wave.models.db")
```

Here are some details about the used commands:

- The dataset is never loaded in memory. We connect to the database with the **dbInit()** command.
- We associate the database to an environment with the **dbzenv()** instruction.
- At the end of the process, we can store the models in another database on the disk [**dumpObjects()**]. It is really useful. Indeed, we can thus distribute the models and deploy them on other databases. Beyond the possibility of processing the database from the disk, this feature fully justifies the use of this package.

The rest of the process is the same as the processing in memory. The utilization of the database does not modify the instructions for the data analysis (rpart and lda). Last, we obtain exactly the same result (model).

About the computation time and the memory occupation, we obtain the following values.

	Processing on the disk (filehash)	
Learning	Time (sec.)	Memory occup. (MB)
Connecting the database	0.02	6
Descriptive statistics	13.44	6.3
rpart	48.67	101.9
lda	23.61	102.2
Storing the tree	250.82	102.2
Storing for lda object	0.02	102.2

The increasing of computation time is undeniable. But, in the same time, we reduce dramatically the memory occupation. In the Windows task manager, the memory occupation for R is also reduced (111.9 MB against 591 MB).

It is clear that the “filehash” package is an elegant solution for the dealing with large dataset. Only the initial creation of the binary file is a bit long and may be problematic. But, we perform this operation only one times. It will not be repeated thereafter.

4.3 Model deployment

As we saw earlier, one of the exciting functionality of “filehash” package is the ability to store the models in a file. Then, we can re-utilize them in another context, especially when we want to apply the model on another database.

The models are stored in a binary file. We apply them on the learning database, thus we can check the correctness of the prediction by comparing the confusion matrix with those obtained earlier when we perform all the analysis into main memory. But the process is easily generalized on any database. The only restriction is that the original variables must be available.

```
#clear the memory
```

```

rm (list=ls())
#connecting the dataset into a database
#loading the "filehash" package
library(filehash)
#initialize the database
setwd("D:/DataMining/Databases_for_mining/dataset_for_soft_dev_and_comparison/swap_strategy")
db.data <- dbInit("wave.dataset.db")
#attach the database to an environnement
wave.env.data <- db2env(db.data)
#loading the packages associated to the models
library(rpart)
library(MASS)
#connecting the models into the database
db.models <- dbInit("wave.models.db")
#attach the db to an environnement
wave.env.models <- db2env(db = db.models)
#printing the models
print(wave.env.models$wave.tree)
print(wave.env.models$wave.lda)
#prediction on the dataset
pred.tree <- predict(wave.env.models$wave.tree, newdata = wave.env.data$wave.data, type = "class")
pred.lda <- predict(wave.env.models$wave.lda, newdata = wave.env.data$wave.data)$class
#confusion matrix
mc1 <- table(wave.env.data$wave.data$Onde,pred.tree)
err1 <- 1.0 - (mc1[1,1]+mc1[2,2]+mc1[3,3])/sum(mc1)
print(err1)
mc2 <- table(wave.env.data$wave.data$Onde,pred.lda)
err2 <- 1.0 - (mc2[1,1]+mc2[2,2]+mc2[3,3])/sum(mc2)
print(err2)

```

About the computation time and the memory occupation, we have.

Prediction	Processing on disk (filehash)	
	Time (sec.)	Memory occup. (MB)
Database connection	0.02	6
Predict - Tree	22.45	94.3
Predict - Lda	46.49	102
Confusion matrix	15.36	102

Clearly, we do not use all the available resource of the computer here. We can easily handle larger database. We note however that, in comparison with the processing in memory (Figure 1), increasing in computation time is more significant for the tree.

4.4 Using all the predictive variables

Since the package "filehash" gives us a breathing space on computer memory resources, we can legitimately think that it is now possible to process all predictors of the dataset. So we launched the learning with 21 predictors.

```

#clear the memory
rm (list=ls())
#loading the "filehash" package

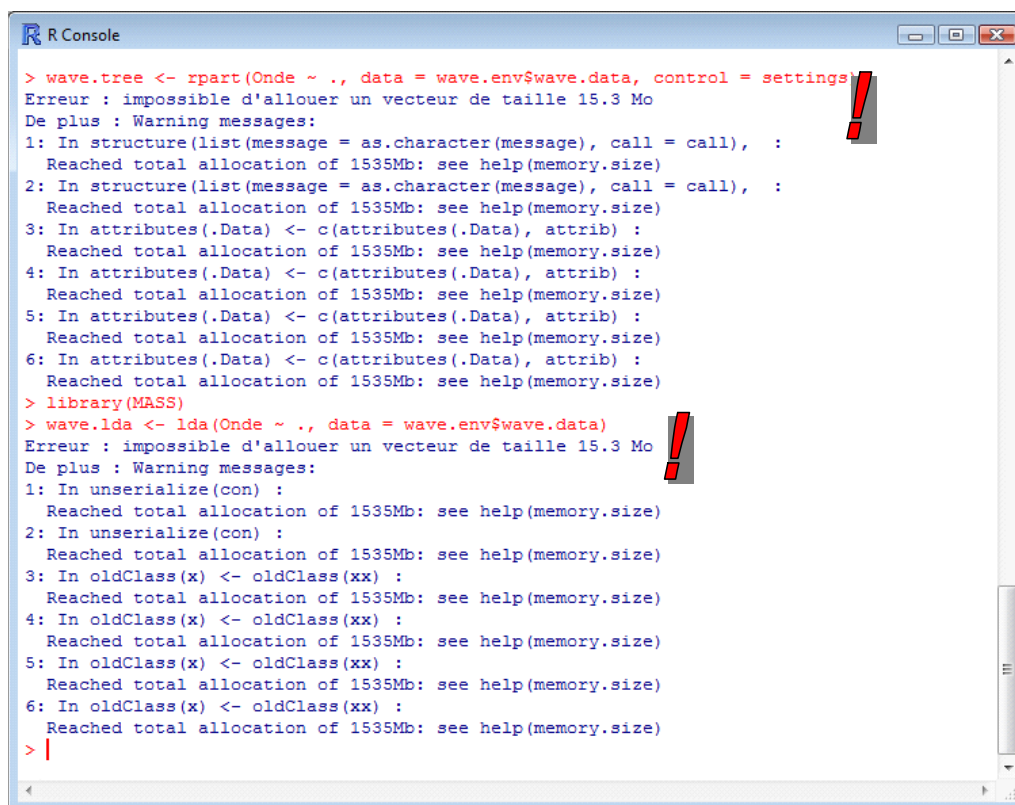
```

```

library(filehash)
#initialize the database
setwd("D:/DataMining/Databases_for_mining/dataset_for_soft_dev_and_comparison/swap_strategy")
db <- dbInit("wave.dataset.db")
#attach the database to an environnement
wave.env <- db2env(db)
#checking the dataset
print(summary(wave.env$wave.data))
#learning a decision tree - rpart
library(rpart)
settings <- rpart.control(xval=0,minsplit=100000,minbucket=100000,maxsurrogate=0,cp=0)
wave.tree <- rpart(Onde ~ ., data = wave.env$wave.data, control = settings)
print(wave.tree)
#linear discriminant analysis
library(MASS)
wave.lda <- lda(Onde ~ ., data = wave.env$wave.data)
print(wave.lda)

```

The result is disappointing. The calculation had failed, R sends us the same error messages when processing in memory.



```

R Console
> wave.tree <- rpart(Onde ~ ., data = wave.env$wave.data, control = settings)
Erreur : impossible d'allouer un vecteur de taille 15.3 Mo
De plus : Warning messages:
1: In structure(list(message = as.character(message), call = call), :
  Reached total allocation of 1535Mb: see help(memory.size)
2: In structure(list(message = as.character(message), call = call), :
  Reached total allocation of 1535Mb: see help(memory.size)
3: In attributes(.Data) <- c(attributes(.Data), attrib) :
  Reached total allocation of 1535Mb: see help(memory.size)
4: In attributes(.Data) <- c(attributes(.Data), attrib) :
  Reached total allocation of 1535Mb: see help(memory.size)
5: In attributes(.Data) <- c(attributes(.Data), attrib) :
  Reached total allocation of 1535Mb: see help(memory.size)
6: In attributes(.Data) <- c(attributes(.Data), attrib) :
  Reached total allocation of 1535Mb: see help(memory.size)
> library(MASS)
> wave.lda <- lda(Onde ~ ., data = wave.env$wave.data)
Erreur : impossible d'allouer un vecteur de taille 15.3 Mo
De plus : Warning messages:
1: In unserialize(con) :
  Reached total allocation of 1535Mb: see help(memory.size)
2: In unserialize(con) :
  Reached total allocation of 1535Mb: see help(memory.size)
3: In oldClass(x) <- oldClass(xx) :
  Reached total allocation of 1535Mb: see help(memory.size)
4: In oldClass(x) <- oldClass(xx) :
  Reached total allocation of 1535Mb: see help(memory.size)
5: In oldClass(x) <- oldClass(xx) :
  Reached total allocation of 1535Mb: see help(memory.size)
6: In oldClass(x) <- oldClass(xx) :
  Reached total allocation of 1535Mb: see help(memory.size)
> |

```

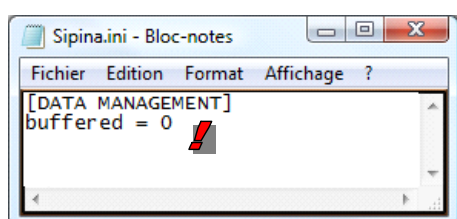
This is really disappointing because we do not use all the available resources. The probably reason of this failure is that the **rpart** and **lda** functions have not been designed to process data on disk. They mechanically allocate a memory space proportionally to the size of the database. This confirms the comments that we make above. To handle large databases to disk, it is essential to establish an effective swapping system, but we must also adapt the implementation of learning algorithms.

5 Swapping system for Sipina

We have already described the [swapping system of Sipina for the processing of very large dataset](#). In this section, we repeat the experiment carried out in R, with and without the swapping system. We bring to mind that the system implemented in Sipina is especially intended to the induction of decision tree. It is very interesting to evaluate the behavior of the system when we use the linear discriminant analysis algorithm. Here also, we lead the analysis with the 5 predictive variables first; then we perform the analysis with all the predictors, if this is possible.

Sipina does not provide indications about the memory occupation. We use the Windows task manager to measure it.

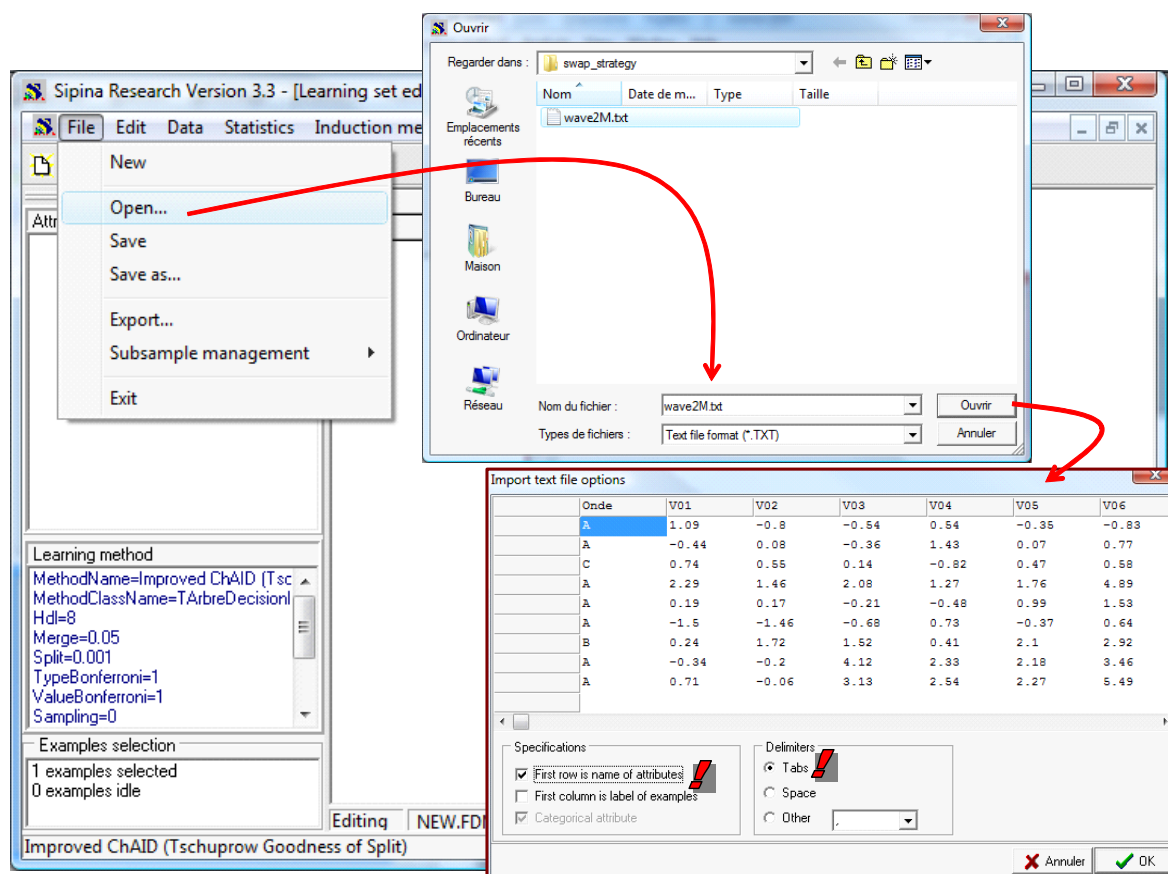
5.1 Processing in memory – 5 predictors



To know the current settings of Sipina, we open the SIPINA.INI configuration file: "buffered = 0" means that all the dataset are loaded in memory.

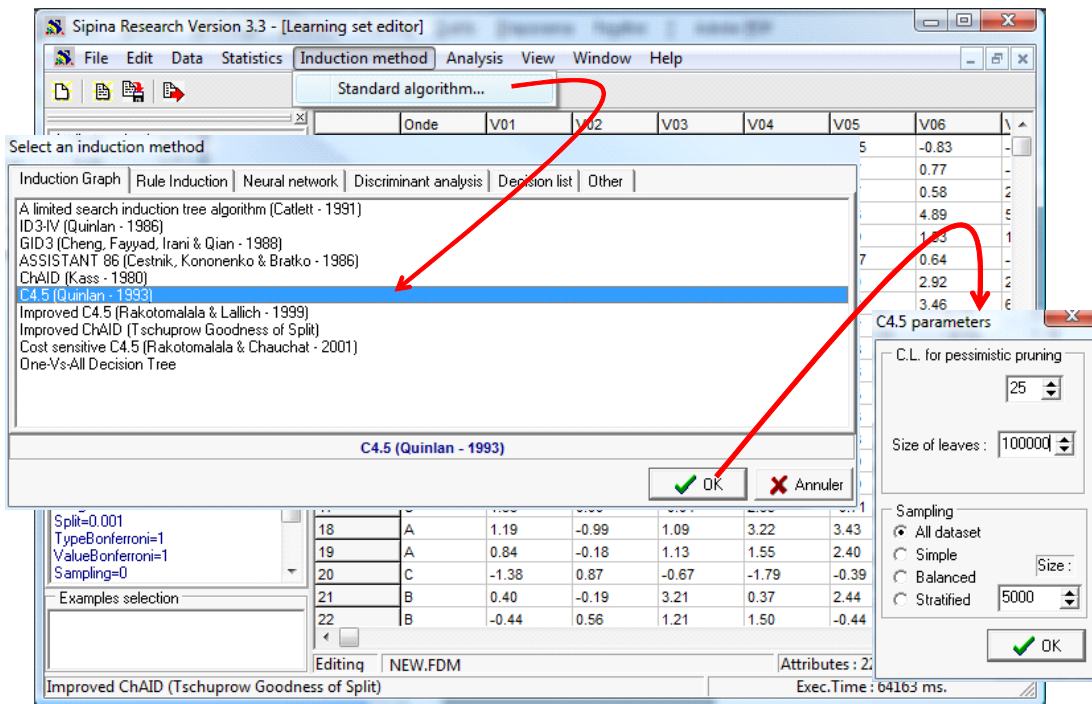
5.1.1 Importing the data file

We click on the FILE / OPEN menu to import the « wave2M.txt » data file. We set the following parameters.

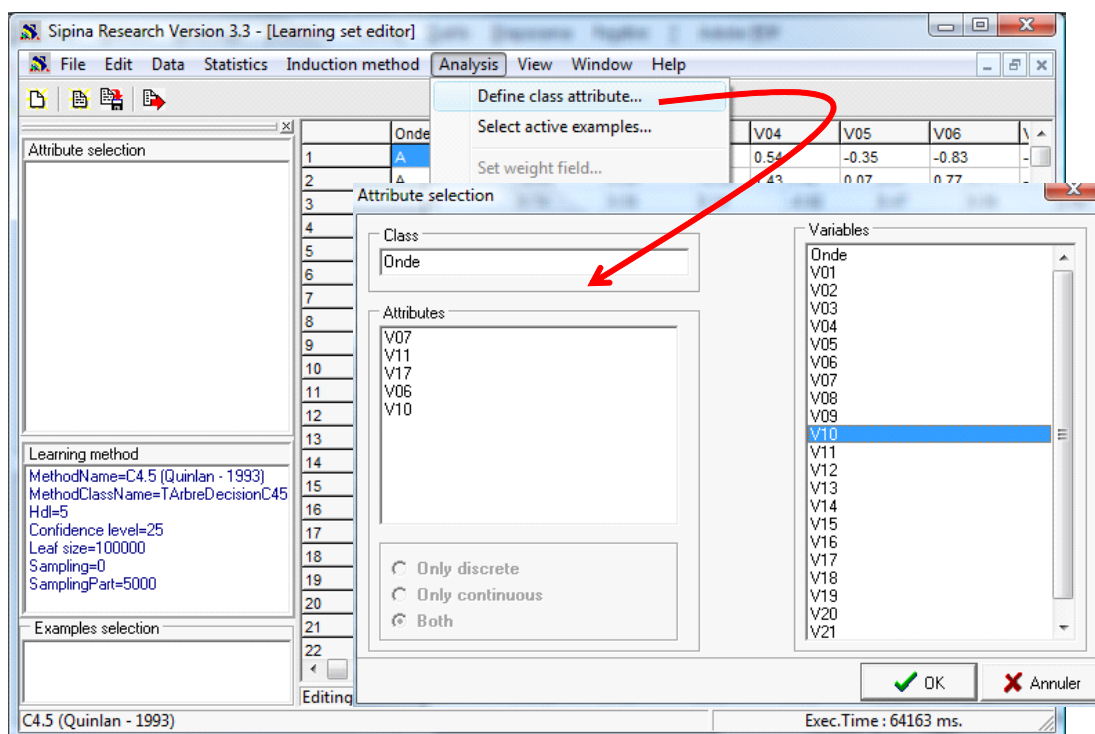


5.1.2 Induction of decision tree

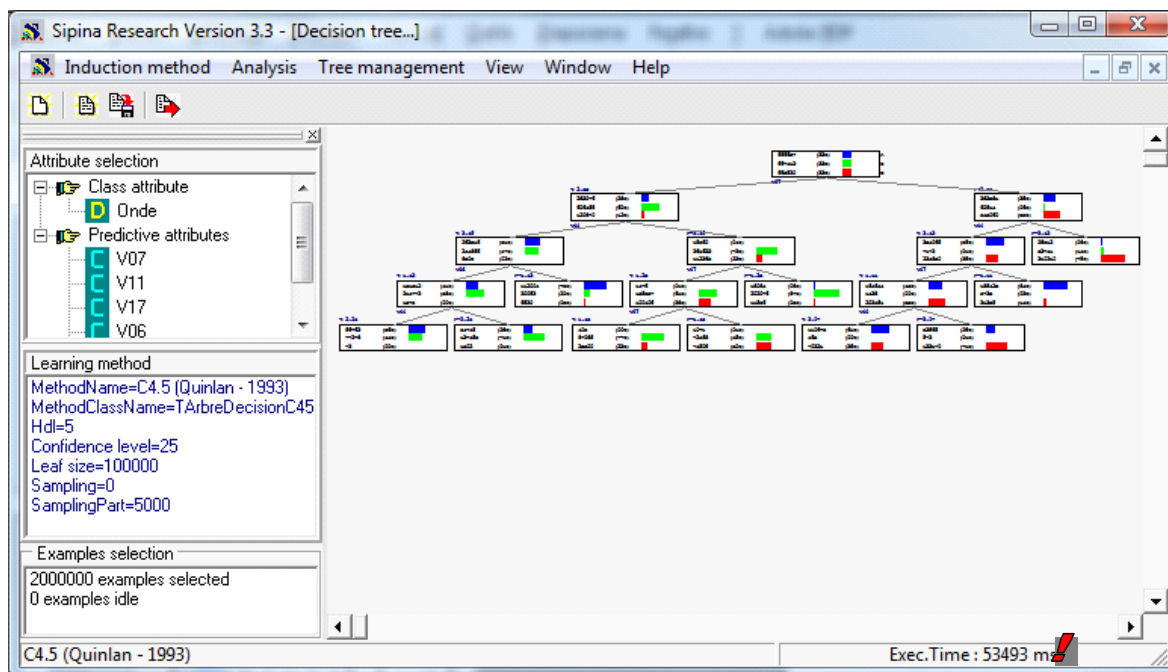
We use the C4.5 algorithm. We set the parameters which make comparable the computation time to the *rpart* function of R. We click on the INDUCTION METHOD / STANDARD ALGORITHM menu. In the dialog settings, we select the C4.5 approach. We click on the OK button. Then, we set the parameters of the learning process. The leaves of the tree contain up to 100,000 instances.



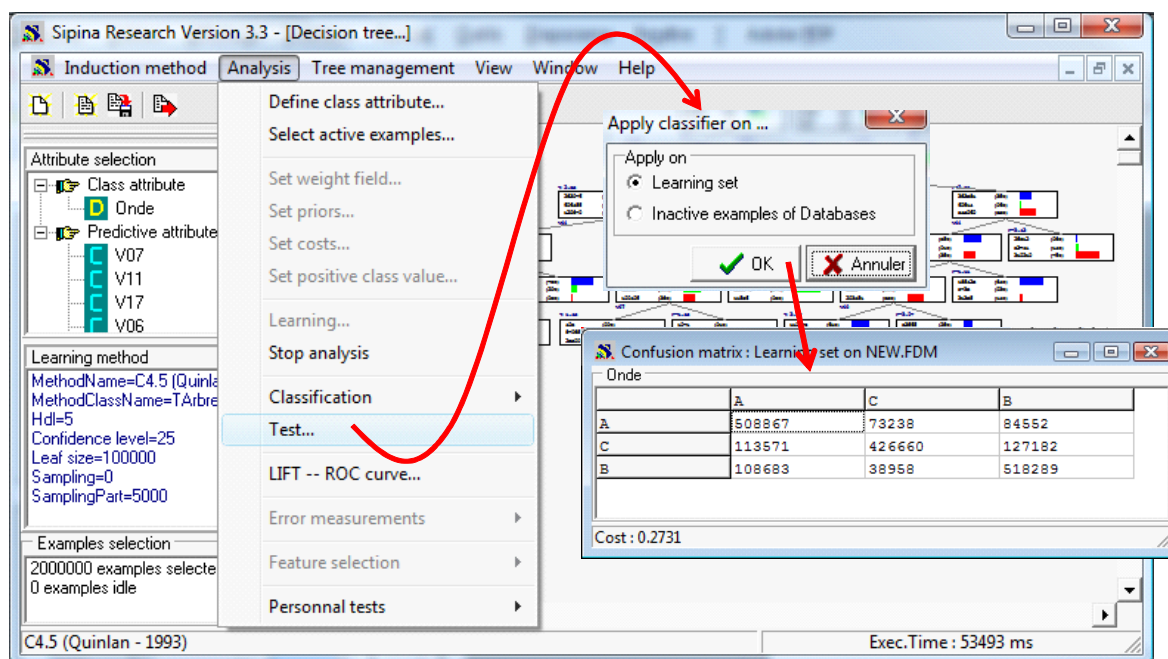
We click on the ANALYSIS / DEFINE CLASS ATTRIBUTE menu. We specify the target attribute (ONDE) and the input ones (V07, V11, V17, V06 and V10).



To launch the calculations, we activate the ANALYSIS / LEARNING menu. The learning process is achieved in 53 seconds.

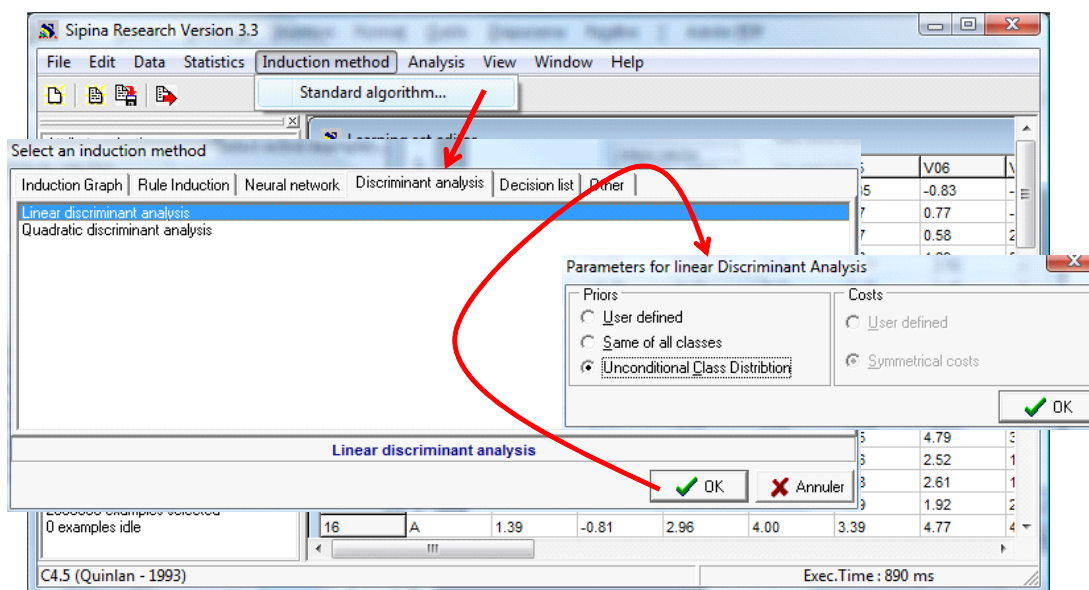


In order to compute the confusion matrix, we click on the ANALYSIS / TEST menu. We select the "Learning set" option to obtain the resubstitution error rate.

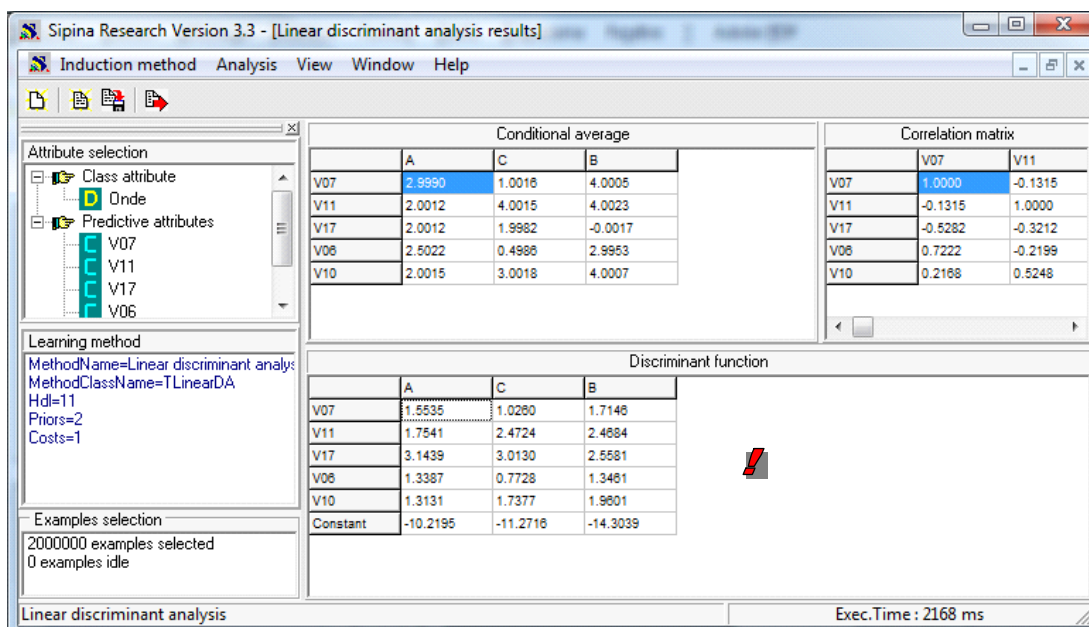


5.1.3 Linear discriminant analysis

We stop the current analysis by clicking on the ANALYSIS / STOP ANALYSIS menu. Then, we select the linear discriminant analysis method (INDUCTION METHOD / STANDARD ALGORITHM menu, DISCRIMINANT tab). We validate the default settings.

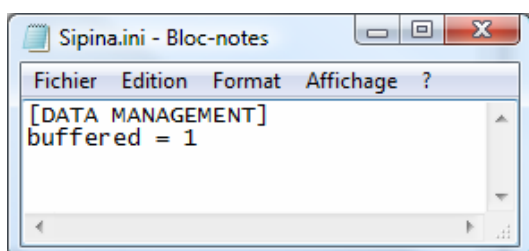


We click again on the ANALYSIS / LEARNING menu. We obtain the following visualization window.



We repeat the same processing as above to obtain the confusion matrix.

5.2 Using the swap files – 5 predictors



We close the SIPINA application. Again, we open the SIPINA.INI configuration file. We set "buffered = 1". In this case, each column of the dataset is dumped in a temporary file. Only a part of the observations are loaded in a caching system. This system was optimized for the learning of decision trees; it is interesting to observe its behavior when we perform a discriminant analysis.

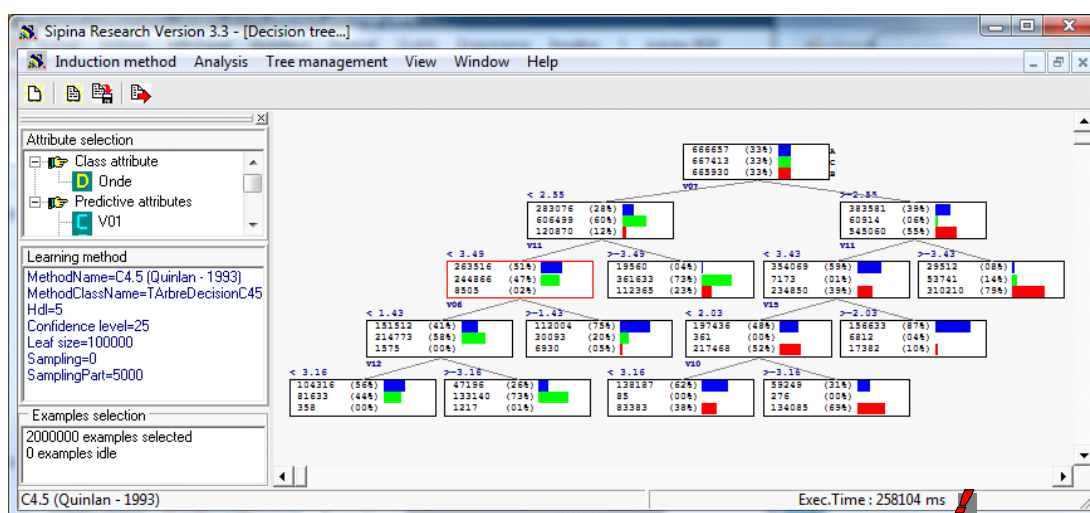
We repeat the processing above. We report here the computation time and the memory occupation.

Learning process	Processing in memory	Using a swapping system
	Time (sec.)	Time (sec.)
Data importation	64	67
Decision tree learning	53	64
Linear discriminant analysis	2	3
Confusion matrix	3	3
Memory occupation after the importation of the data file	229	19

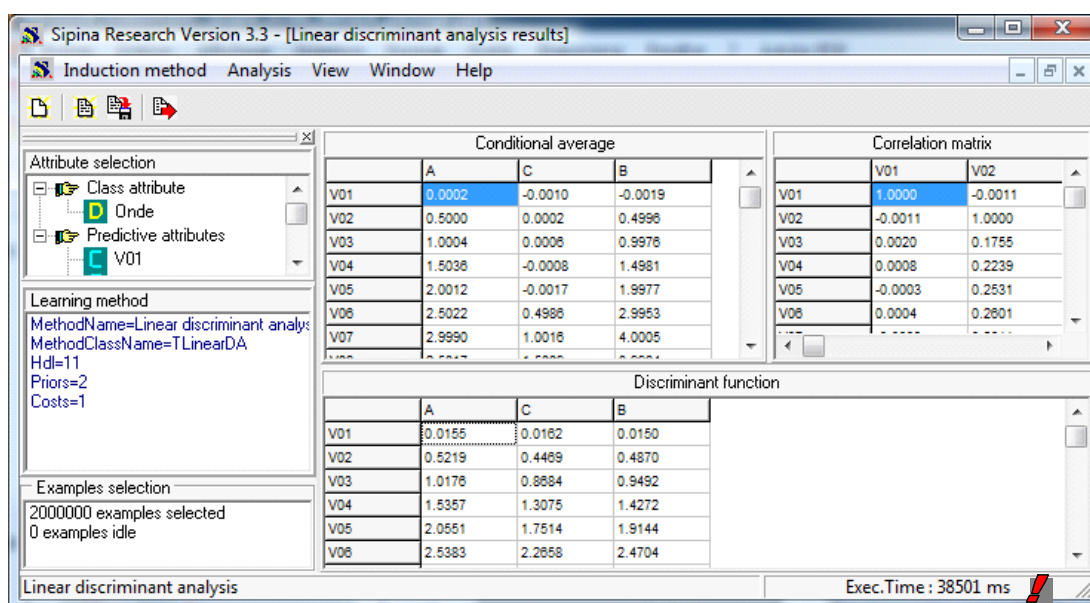
Using the swapping system is definitely advantageous about the memory occupation. It is not really surprising. But, we observe here that both decision tree and discriminant analysis are not much penalized in terms of computation time. It is rather good news.

5.3 Using the swap files – All the predictive variables

Contrary to R, we can perform the analysis with all the predictive variables in Sipina. We repeat the same process as above, but when we specify the input variables, we include all the continuous attributes (V01 to V21 -- ANALYSIS / DEFINE CLASS ATTRIBUTES menu). We obtain the following decision tree in 28 seconds.



The processing time is 38 seconds for the linear discriminant analysis.



6 Conclusion

In this tutorial, we describe the filehash package for R. It allows to handle directly the dataset from the disk. The memory occupation is thus dramatically reduced. The processing capabilities are increased. The library is rather efficient. Even if the values are repeatedly accessed on the disk during the data mining process, the increase in calculation time is not prohibitive.

However, we note that when we still increase the dataset size, some learning functions such as rpart or lda fail. It means that, even if we have an efficient solution to accessing the dataset directly on the disk, we must adapt the implementation of data mining algorithms. The filehash package is not the panacea.

Note: I try the `dumpDF` function to separately store each column of the data frame into the database. The storing is much faster. When I wanted to perform the analysis with rpart and lda, they still failed, but the error messages are different. My investigations have stopped at this point.