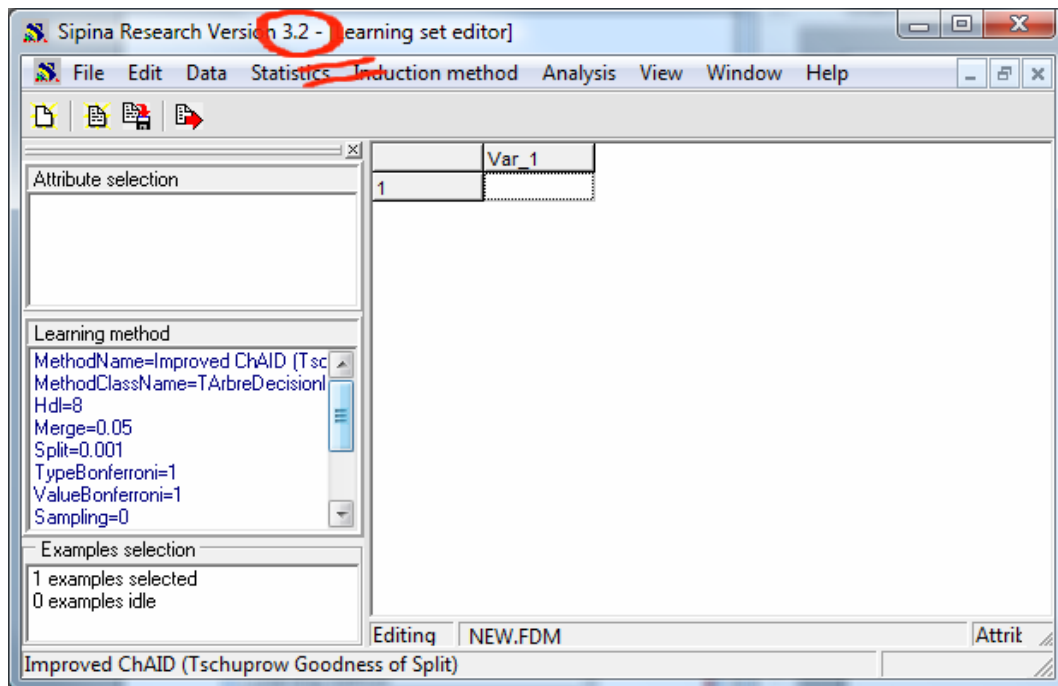


1 Introduction

Dealing with large dataset in SIPINA (9,634,198 observations and 41 variables).

To fully reproduce this tutorial, make sure you have the version 3.2 (or later) of Sipina. You can verify this by looking at the version number in the title bar when the software is launched.



The ability to handle large databases is a crucial problem in the data mining context. We want to handle a large dataset in order to detect the hidden information. Most of the free data mining tools have problems with large dataset because they load all the instances and variables into memory. Thus, the limitation of these tools is the available memory (see <http://data-mining-tutorials.blogspot.com/2008/11/decision-tree-and-large-dataset.html>; or <http://data-mining-tutorials.blogspot.com/2009/01/performance-comparison-under-linux.html>).

Sometimes, a sampling strategy enables to perform a fruitful analysis; but most often, because the "pattern" that we want to highlight concerns a very small part of the population, it is necessary to deal with the full dataset. Indeed, a standard sampling strategy can lead to a too low presence of the instances that we want to analyze e.g. when we deal with a very imbalanced dataset in the supervised learning context (see for a specific local sampling strategy for the decision tree induction algorithm which is much more better than global sampling - <http://data-mining-tutorials.blogspot.com/2009/10/local-sampling-approach-for-decision.html>).

To overcome this limitation, we should design solutions that allow to copy all or part of the data on disk, and perform treatments by loading into memory only what is necessary at each step of the algorithm (the instances and/or the variables). If the solution is theoretically simple, it is difficult in practice. Indeed, the processing time should remain reasonable even if we increase the disk access. It is very difficult to implement a strategy that is effective regardless of the learning algorithm used (supervised

learning, clustering, factorial analysis, etc.). They handle the data in very different way: some of them use intensively matrix operations; the others search mainly the co-occurrence between attribute-value pairs, etc.

In this tutorial, we present a specific solution in the induction tree context. The solution is integrated into SIPINA (as optional) because its internal data structure is especially intended to the decision tree induction. Developing an approach which takes advantages of the specificities of the learning algorithm was easy in this context. We show that it is then possible to handle a very large dataset (**41 variables** and **9,634,198 observations!**) and to use all the functionalities of the tool (interactive construction of the tree, local descriptive statistics on nodes, etc.).

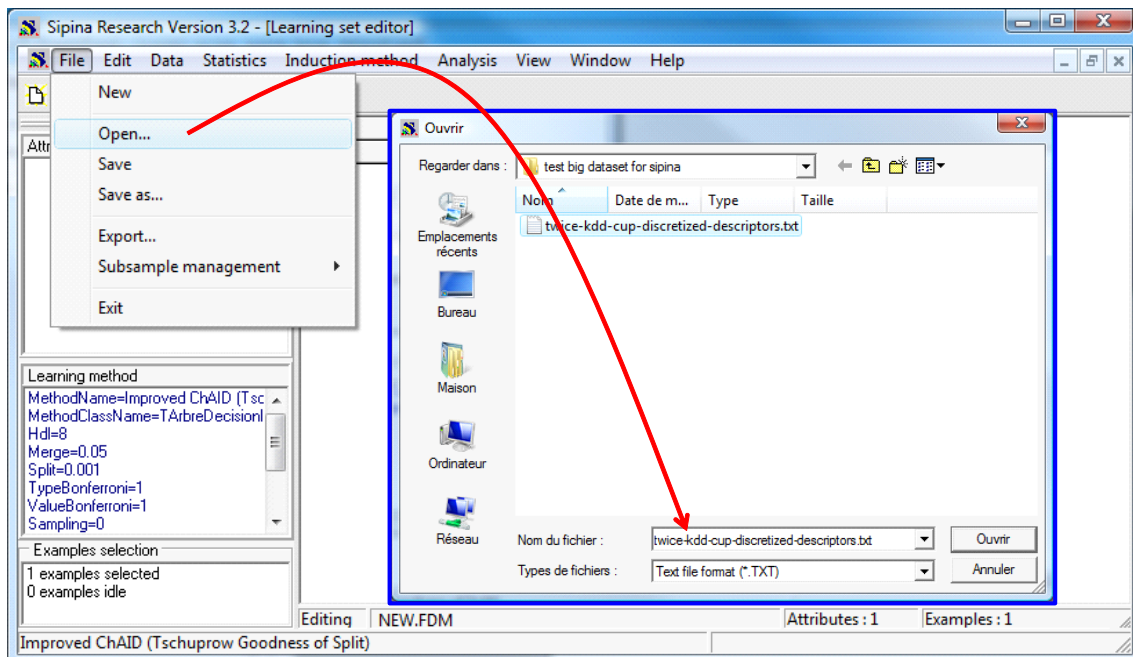
To fully appreciate the solution proposed by Sipina, we compare its behavior in comparison to generalist data mining tools such as **Tanagra 1.4.33** or **Knime 2.03**.

2 Dataset

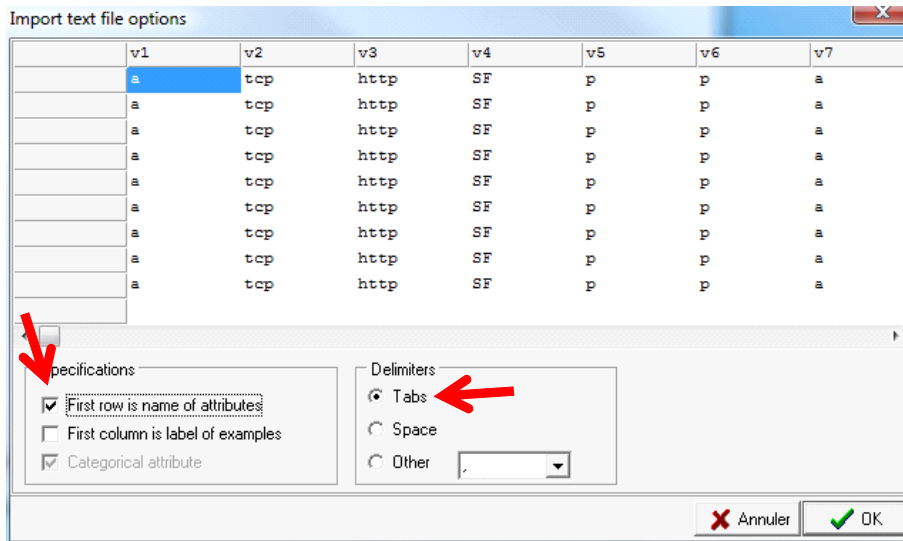
We use a version of KDD-CUP 99 dataset (<http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html>). We have duplicated twice the original data file (TWICE-KDD-CUP-DISCRETIZED-DESCRIPTORS.TXT - <http://eric.univ-lyon2.fr/~ricco/dataset/twice-kdd-cup-discretized-descriptors.zip>). The target attribute (CLASSE) is at the last column. All the descriptors are discrete or were discretized. Loading this data file into a Data Mining tool is already a challenge.

3 Loading the whole dataset into memory

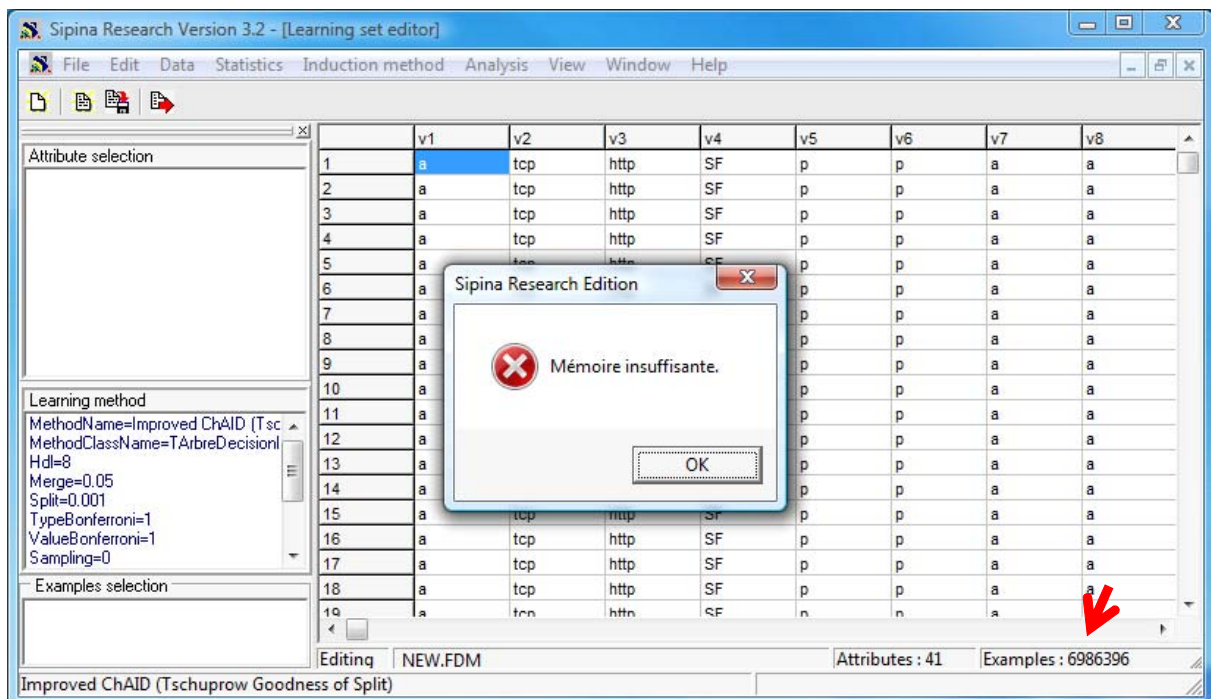
The standard working of Sipina is to load all data in memory. We show in this section that it is unfeasible on our dataset. We launch SIPINA. We click on the FILE / OPEN menu. We select our data file.



In the following dialog box, we set the appropriate settings: tab as column separator, the first row corresponds to the attributes' name.

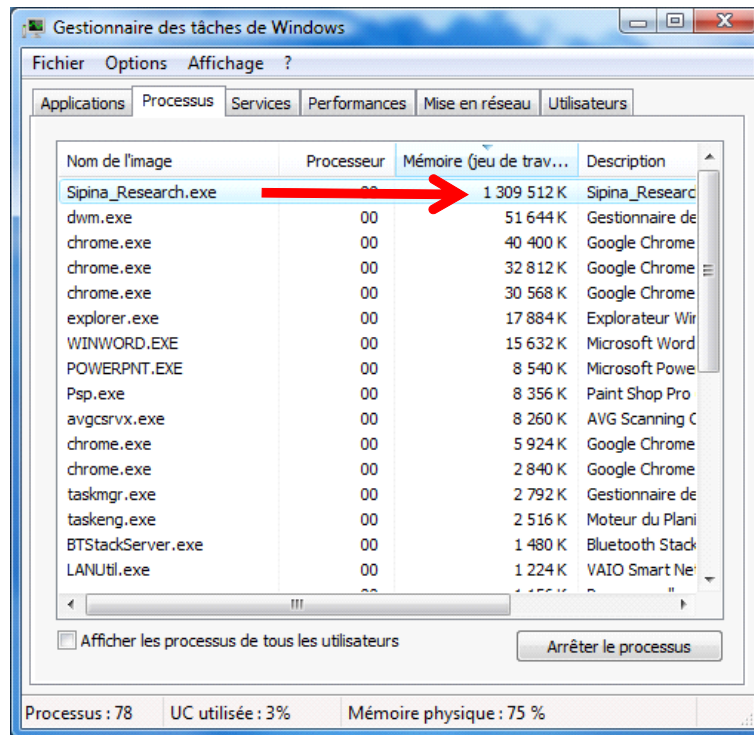


We click on the OK button. A progress bar allows watching the operations. Unfortunately, after a while, an error message appears (Insufficient Memory). We observe that only a part of the instances was loaded (6.986.396 observations).



The windows task manager shows that the memory occupation of SIPINA is 1,309,512 KB.

The ability to load almost 7 million of observations is already important. But we note that we cannot make any operation, even descriptive statistics. Every asked task causes an error.



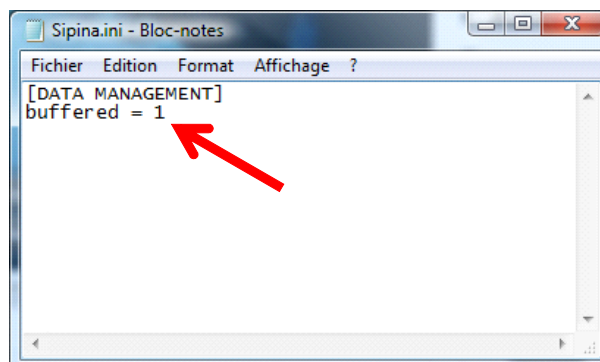
4 Swapping system for decision tree induction

The feature that I present in this tutorial has never been outlined before today. I rediscovered it recently. I realized that it is totally operational. We must simply activate the right option.

4.1 Modifying the launching parameters

We must close SIPINA before making changes.

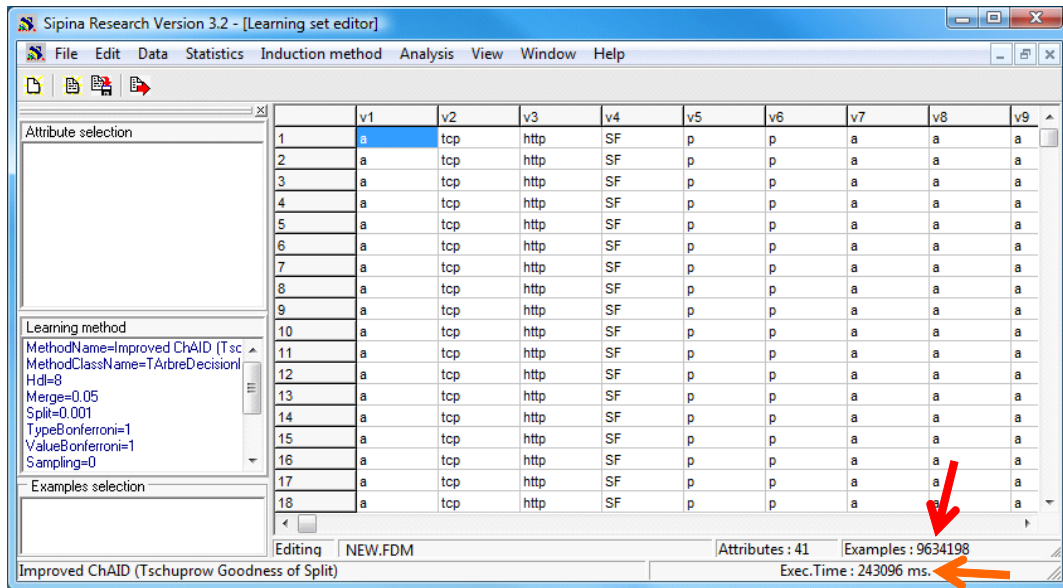
We open the installation directory of SIPINA (under Windows Vista - C:\Programmes\StatPackages). We open the **SIPINA.INI** configuration file. We set the BUFFERED parameter to 1 as in the screenshot below. We save this modification.



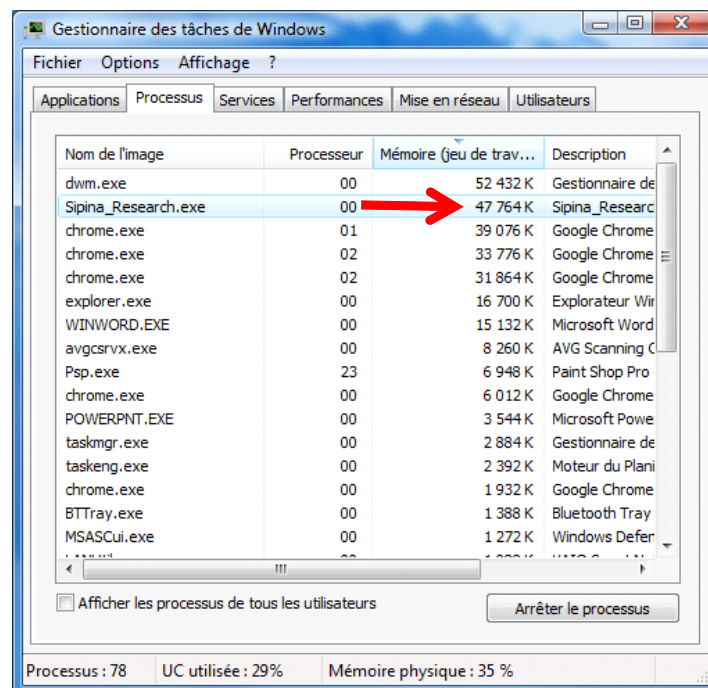
4.2 Loading the data file

Then, we launch SIPINA again. We proceed as above. We select the dataset. This time, the whole

dataset is loaded. The 9,634,198 instances were processed in 243 seconds (about 4 minutes).



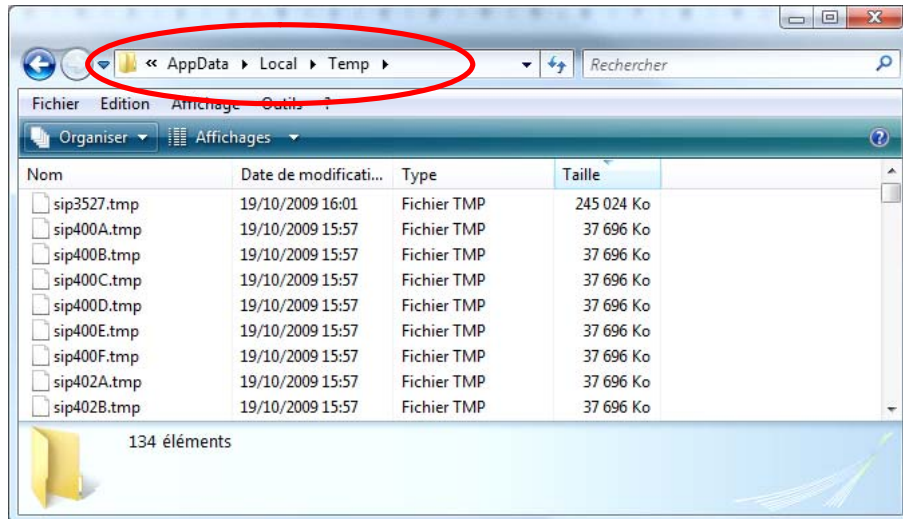
Now, the Windows task manager shows that the memory occupation of SIPINA is 47,764 KB. Clearly, the data are not stored in memory.



We can launch the decision tree induction now.

4.3 Storage of temporary files

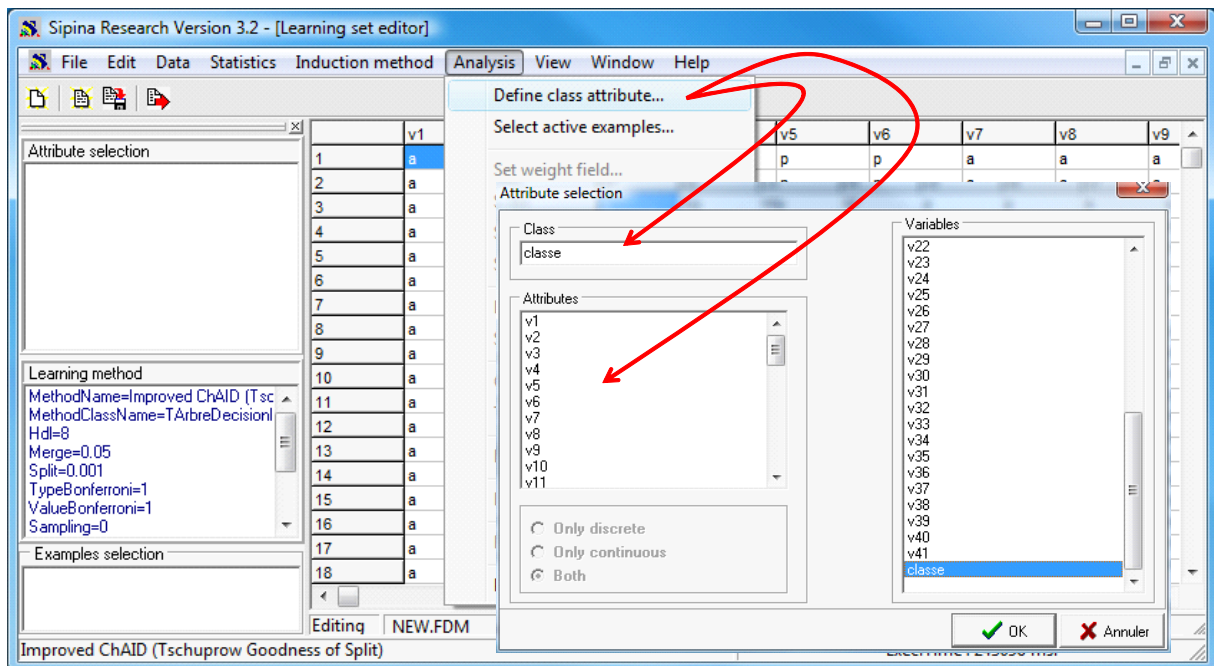
But first, it is interesting to specify the storage directory used by temporary files. They use the temporary directory of the current user. We have one file for each variable and one file for the column labels. As we can see, the file size can be large.



Note: This information about their location is essential. While Sipina crashes for some reason, the temporary files will not be removed. We must do it manually.

4.4 Specifying the target and the input attributes

We click on the ANALYSIS / DEFINE CLASS ATTRIBUTE. Using drag and drop, we set the appropriate selection.



4.5 Launching the learning process

By default, SIPINA uses the IMPROVED CHAID method. It limits the depth of the tree to 4 levels. Of course, we can modify this parameter. We perform the analysis by clicking on the ANALYSIS / LEARNING menu. The calculation time is 267 seconds (# 4 minutes and 30 seconds).

As we said above, the high memory occupation of SIPINA during the decision tree induction is due to the storage of much temporary information on each node: goodness of split for each descriptor, the associated split description, the list of covered examples, some descriptive statistics, etc. In our context, although the dataset is paged on the disk, all these functionalities remain available. It is possible, among others, to launch a new analysis from a selected node. It is very useful when to perform a deeper analysis on a specific group of instances related to a node.

For instance, we select a node of the tree with a right click. We activate the contextual menu **NODE INFORMATION**. In the dialog box, we select the goodness of split value for the first variable, and then we double click on this value. A splitting is performed on the selected node.

The screenshot shows the Sipina Research Version 3.2 interface. The main window displays a decision tree with nodes containing counts and percentages. A right-click context menu is open over a node, with 'Node information...' selected. The 'Node information' dialog box is open, showing a table of variables and their goodness of split and correlation values. A red arrow points from the 'Node information' dialog to the decision tree, indicating the location of the selected node.

Characterization	Descriptor	Importance	
Select an attribute to view the suggested split Double-click to split with the selected attribute			
	Goodness of split	Correlation	Accept or Rejec
v40	0.34698424	0.3470	█
v11	0.25404169	0.2540	█
v10	0.22439654	0.2244	█
v3	0.21718857	0.2172	█
v35	0.18485279	0.1849	█
v29	0.15913448	0.1591	█
v12	0.11324158	0.1132	█
v5	0.10322406	0.1032	█
v4	0.09243306	0.0924	█
v1	0.07983717	0.0798	█
v37	0.07844353	0.0784	█
v34	0.07758900	0.0776	█
v36	0.04102273	0.0410	█
v26	0.01804028	0.0180	█
v30	0.00970836	0.0097	█
v28	0.00774291	0.0077	█
v25	0.00452920	0.0045	█
v17	0.00422705	0.0042	█
v33	0.00000000	0.0000	█
v38	0.00000000	0.0000	█

Split suggestion

	in [a]	in [p]
normal	748	116
other	76	224

1164 examples (0.01% of the learning set)

Two new leaves are created.

During this process, Sipina retrieves the temporary files on the disk, but we note that the computation time remains moderate.

Similarly, we can inspect the positioning of the variables in the sub-groups associated with nodes; calculate local descriptive statistics, etc.

Informations on : Level 5, Node 2

IF v6 in [p] and v13 in [a] and v27 in [p] and v40 in [p]

Characterization | Descriptors' importance

Continuous attributes | Discrete attributes

v4 (0.0002)				
Values	Strength	Local Dist.	Global Dist.	Recall
SF	-24.22	68 (20%)	7326658 (76%)	0%
S2	-0.10	0 (0%)	308 (0%)	0%
S1	10.38	2 (1%)	1014 (0%)	0%
S3	34.30	2 (1%)	96 (0%)	2%
OTH	-0.06	0 (0%)	114 (0%)	0%
REJ	-4.48	0 (0%)	537748 (6%)	0%
RSTO	303.50	186 (55%)	10612 (0%)	2%
S0	-8.65	0 (0%)	1739170 (18%)	0%
RSTR	107.94	82 (24%)	16154 (0%)	1%
RSTOS0	-0.09	0 (0%)	244 (0%)	0%
SH	-0.27	0 (0%)	2080 (0%)	0%

v27 (0.0001)				
Values	Strength	Local Dist.	Global Dist.	Recall
a	-73.66	0 (0%)	9066120 (94%)	0%
p	73.66	340 (100%)	568078 (6%)	0%

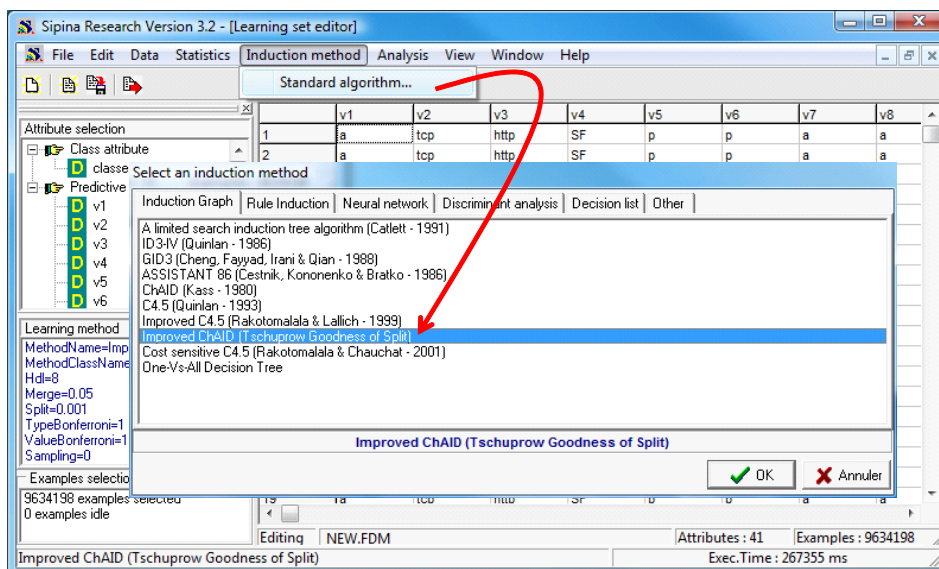
v40 (0.0001)				
Values	Strength	Local Dist.	Global Dist.	Recall
a	-73.66	0 (0%)	9066120 (94%)	0%
p	73.66	340 (100%)	568078 (6%)	0%

340 examples (0.00% of the learning set)

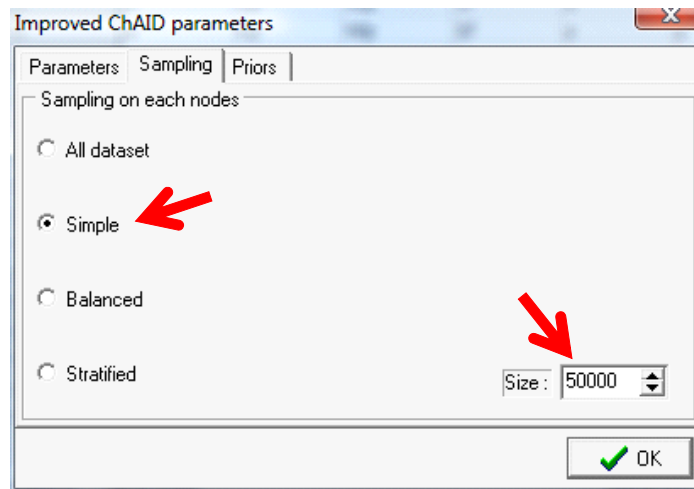
5 Local sampling strategy in SIPINA

It is possible to decrease the computation time, without loss of accuracy, by applying a local sampling strategy in the decision tree induction context. We have showed that this approach is efficient in a previous tutorial (<http://data-mining-tutorials.blogspot.com/2009/10/local-sampling-approach-for-decision.html>). We show here that we can combine it with our swapping system. The result is very impressive: we can handle very large databases, and the computation time is significantly reduced. For instance, on our dataset, we can create a decision tree from a dataset with **41 variables** and **9,634,198 observations** in **83 seconds** (about 1 minute and 20 seconds).

We stop the current analysis session by clicking on the WINDOW / CLOSE ALL menu. Then, we select the learning algorithm by clicking on the INDUCTION METHOD / STANDARD ALGORITHM menu. We select the IMPROVED CHAID method.

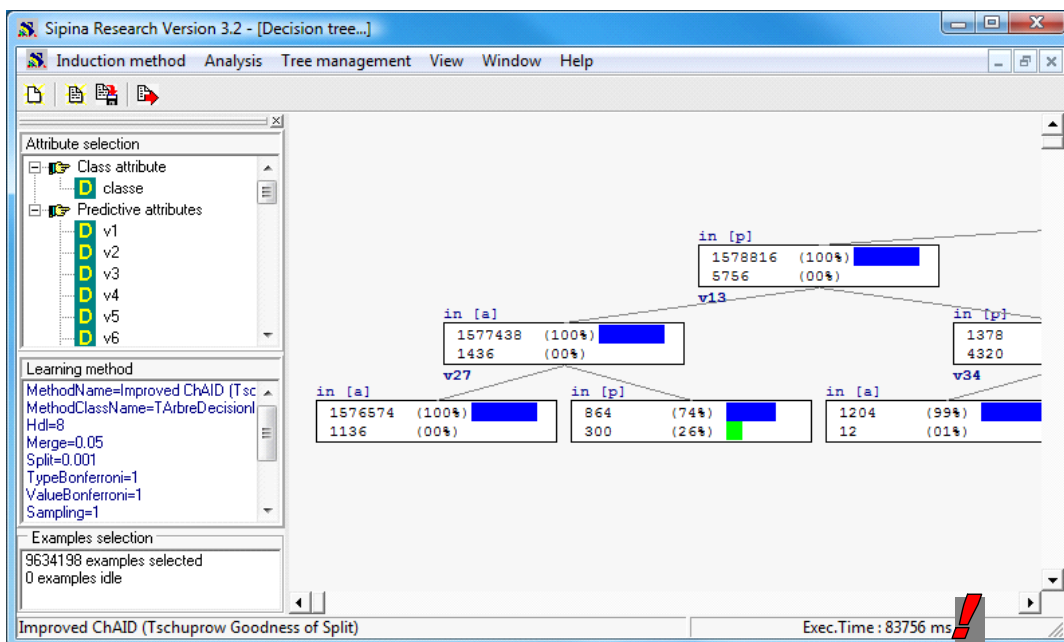


In the dialog settings, we select the SAMPLING tab. We choose the SAMPLING approach, the sample size is 50000. Experiments show that lesser sample size is already efficient in a large broad of context. We choose this size here in order to stabilize of the induced decision tree.

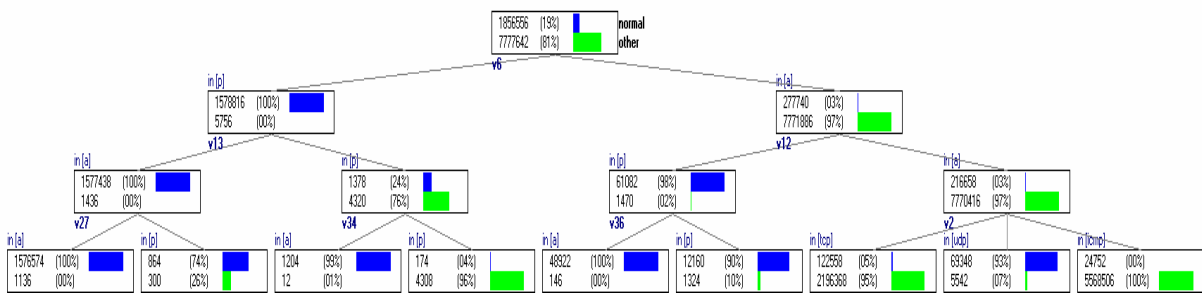


We validate the settings. Then we click again on the ANALYSIS / LEARNING. We obtain the tree after 83 seconds ((#1 minute and 20 seconds)).

Note: The reduction of computing time will be even much more spectacular when many descriptors are continuous. The discretization process is dramatically accelerated when we work with a sample on the nodes.



Here are the details of the induced tree. This is the same as the one computed without the sampling strategy.



Of course, here also, all the interactive functionalities are available.

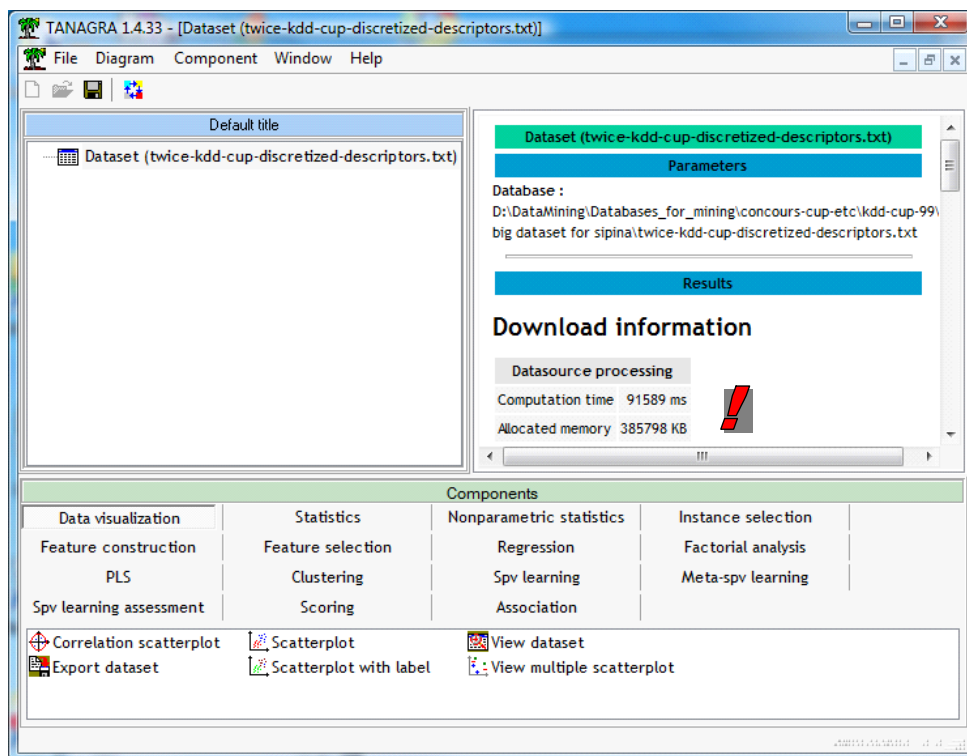
6 Working with the other data mining tools

When they deal with a very large dataset like this one we handle in this tutorial, what is the behavior of the other free tools?

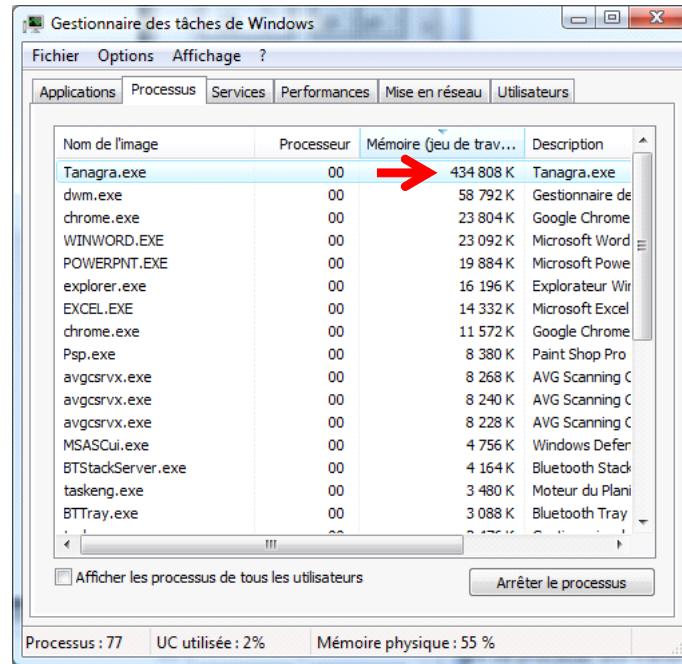
6.1 Tanagra

TANAGRA loads the whole dataset into main memory. But it uses another encoding approach, especially for the discrete variables (it uses less memory). This is the reason for which it can handle our dataset in this tutorial.

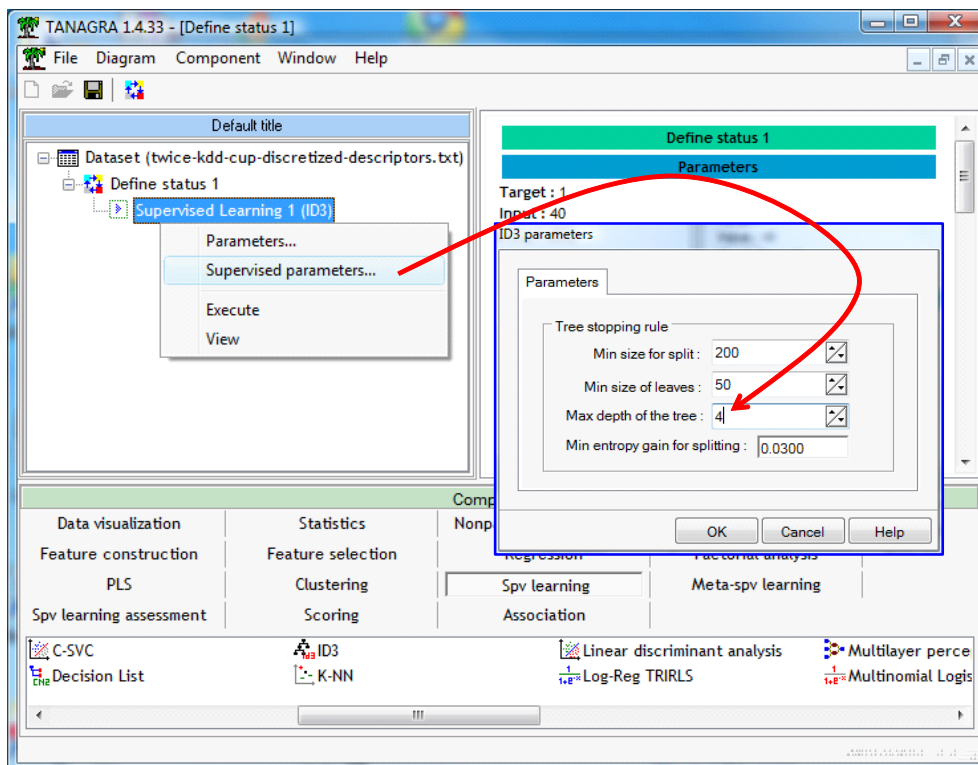
After we launch TANAGRA, we create a new diagram by clicking on the FILE / NEW menu. We select our data file. The loading time is 91 seconds.



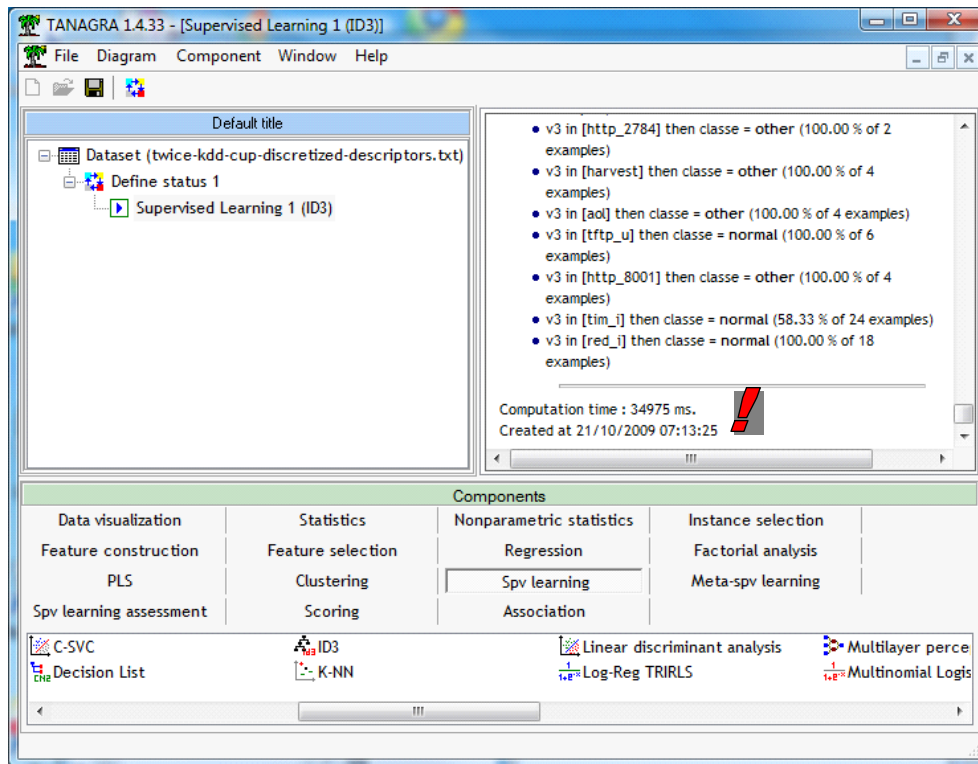
The Tanagra memory occupation is 434,808 KB according the Windows task manager.



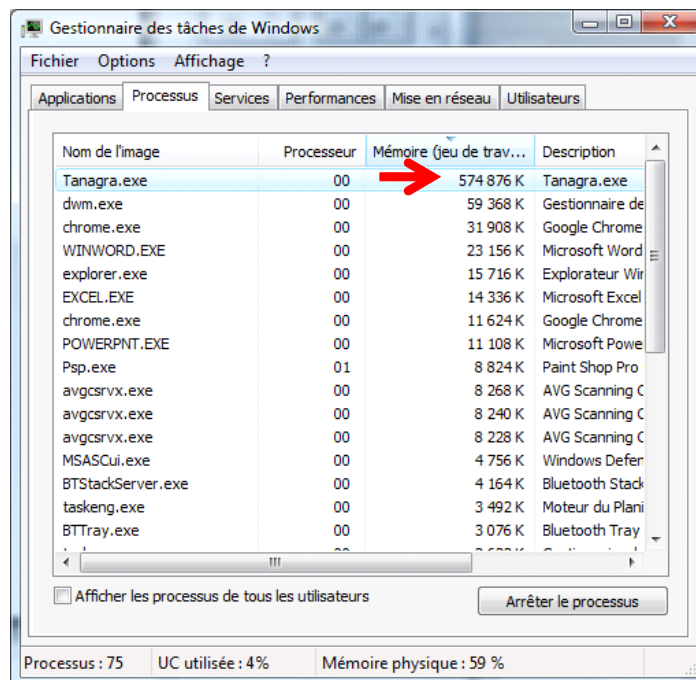
We use the DEFINE STATUS component in order to specify the TARGET (CLASSE) attribute and the INPUT ones (V1...V41). Then we add the ID3 parameters. We restrict the depth of the tree to 4 levels (contextual menu SUPERVISED PARAMETERS).



Then we launch the computation by clicking on the VIEW menu.



The computation time is 35 seconds. The Tanagra memory occupation is 574,876 KB.



Tanagra uses less memory than Sipina in the decision tree induction context. The reason is that the induced tree is not interactive into Tanagra. Thus, it does not keep temporary information on each node (we do not even have information on the other variables on each node e.g. their goodness of split, etc.).

We note however that the tree induction has successfully completed on our dataset. But the solution is not "scalable." If we have 4 times more observations, the loading of all data in memory is not possible

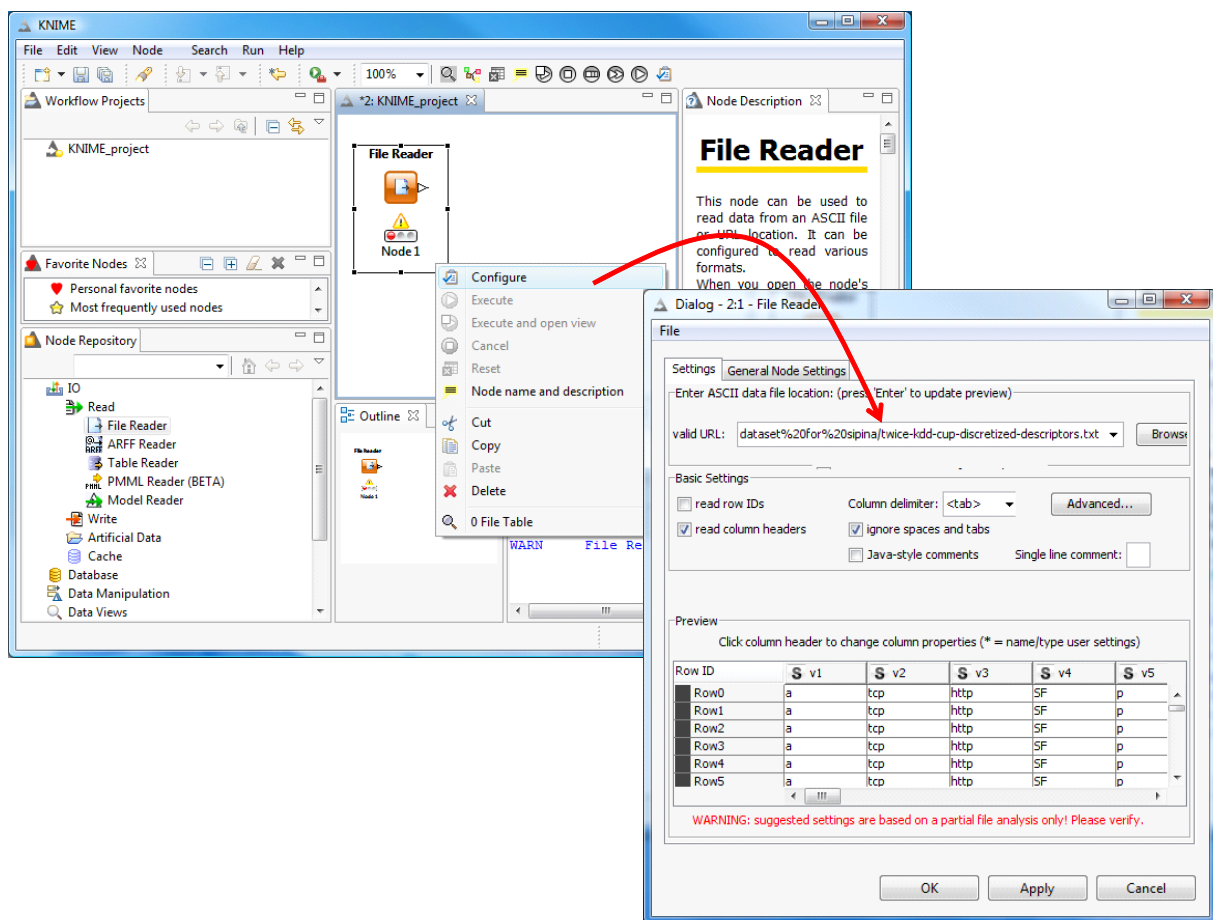
with Tanagra. On the other hand, Sipina with the configuration described in this tutorial will not suffer from such problems.

6.2 Knime

Among the free tools, Knime (<http://www.knime.org/>) seems to be able to keep a part or a whole of the dataset on the disk.

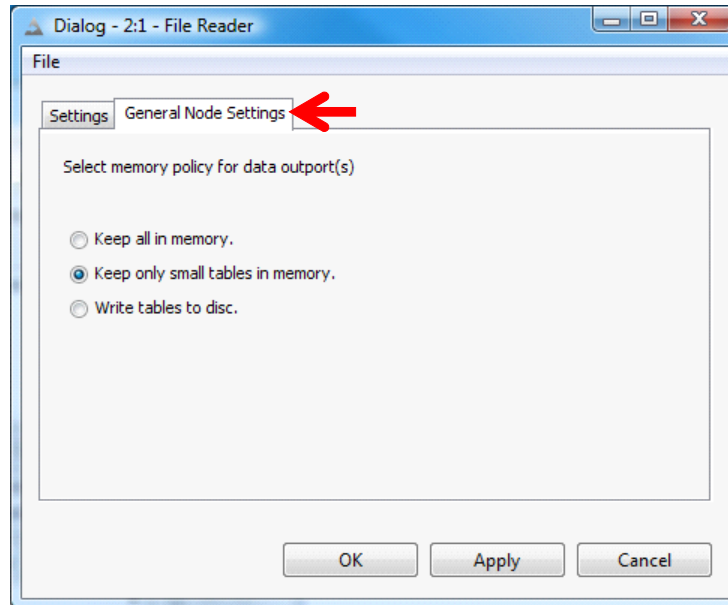
We use the 2.0.3 version in this tutorial.

After we launch the tool, we insert the FILE READER component into the workflow. We select our data file. The first rows are visualized in a data grid.



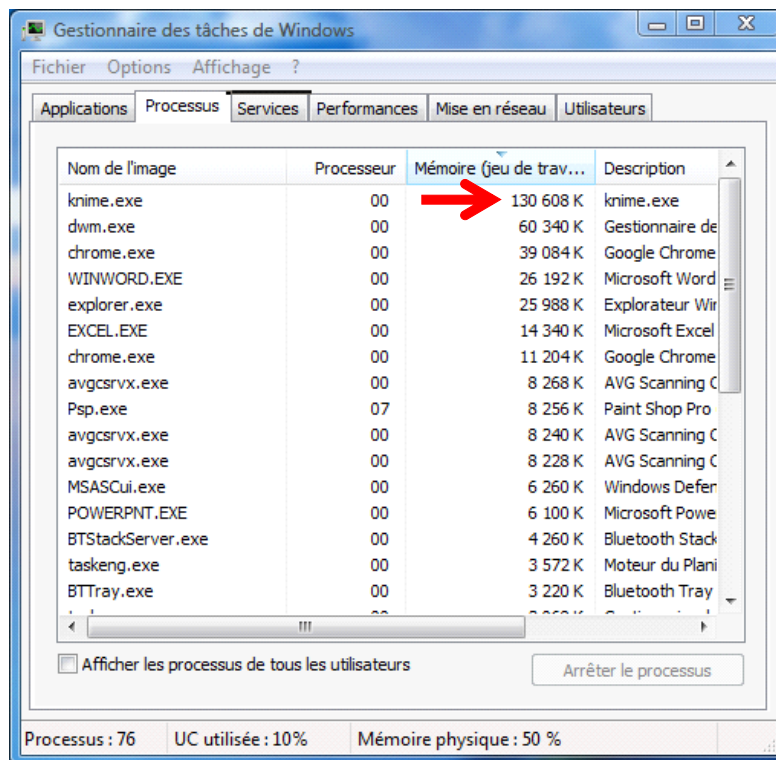
Into the GENERAL NODE SETTINGS, Knime reports that it keeps in memory only the small tables. We can even ask to write systematically the tables to disc.

The main advantage of this approach is that the dataset is loaded into memory only when we perform an analysis with the machine learning algorithms. The data are not always kept in memory in the workflow.



We validate our choice and we click on the EXECUTE contextual menu.

When the dataset is loaded, the Knime memory occupation is 130,608 KB. The value is small in relation to the dataset size. This is proof that the whole dataset is not stored in memory.



We add the DECISION TREE LEARNER component into the workflow. The contextual documentation indicates that the decision tree learning algorithm works only into the main memory. We will see if it is problematic in our context.

Into the dialog settings, we cannot specify the maximum depth of the tree. We modify the other

parameters. Among others, we set MIN NUMBER OF RECORDS PER NODE to 50,000 in order to limit the tree size (Figure 1).

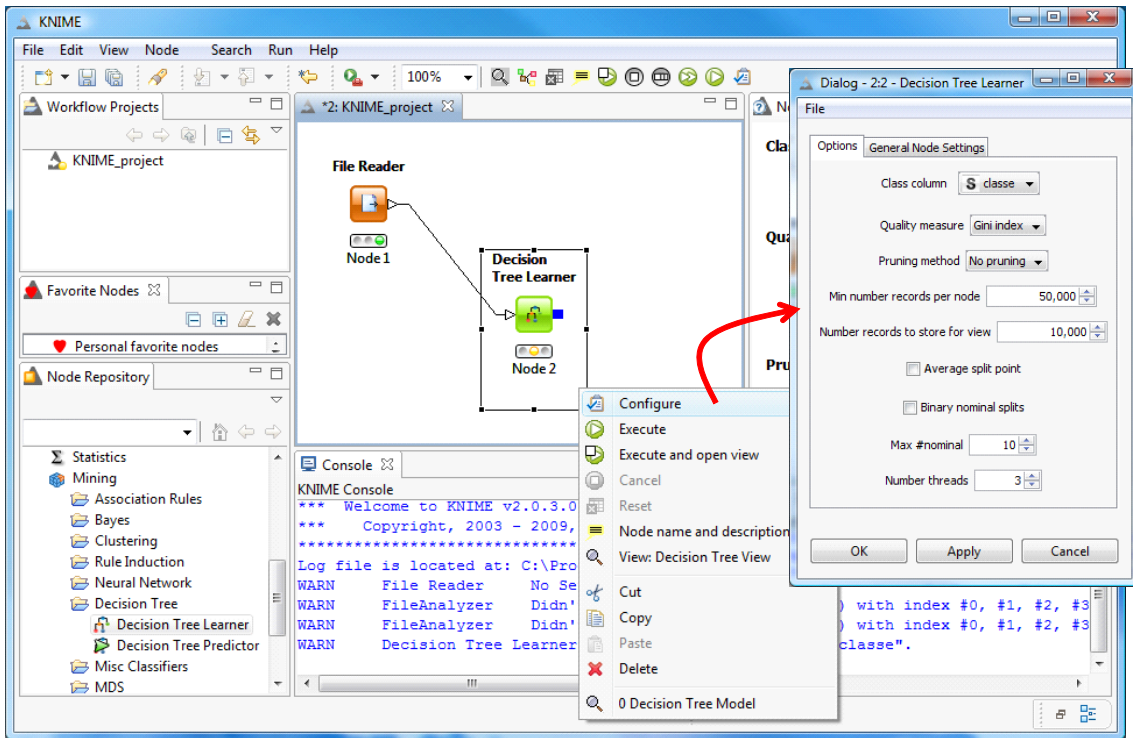


Figure 1 - DECISION TREE LEARNER – Dialog settings

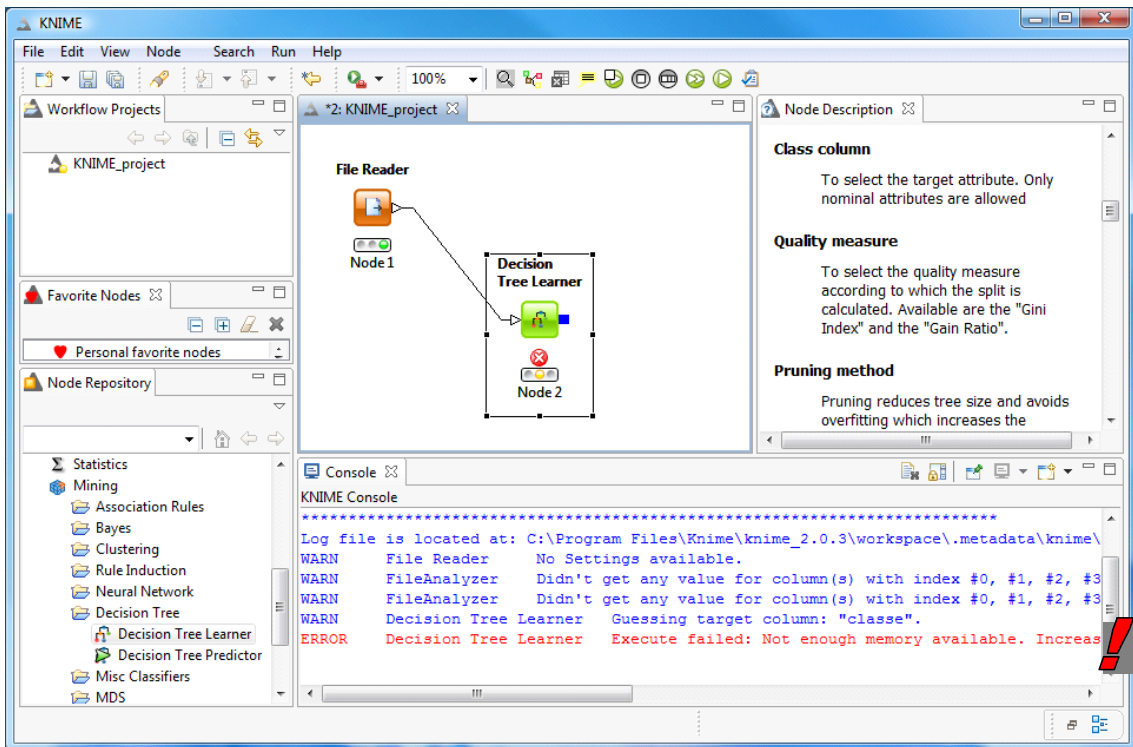


Figure 2 - Knime – Not enough memory available

We click on the EXECUTE and OPEN VIEW menu to launch the calculations. Unfortunately, after some seconds, the computation is suspended. It seems that there is not enough memory for the decision tree induction (Figure 2). Knime says also that we can modify the memory configuration of the JRE to improve the capacity of the tool (see the KNIME.INI configuration file - Figure 3).

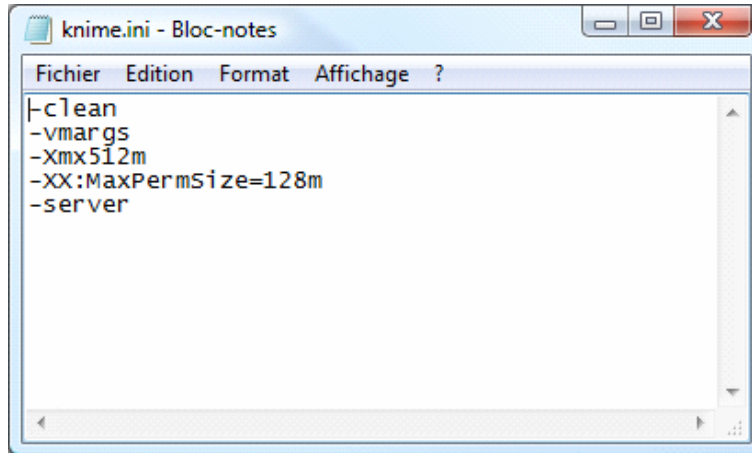


Figure 3 – “Knime.ini” configuration file

7 Conclusion

In this tutorial, we show that using a swapping system can improve dramatically the ability to a decision tree learner to handle a very large dataset.

More generally, such system can improve the ability of the data mining tools to handle a large dataset. However, I think that it is really difficult to implement a system which will be efficient for all the machine learning algorithms. The solution must be specific to a learning algorithm. This is the reason for which this approach is only currently implemented into Sipina, which is mainly intended to the decision tree learning.