1. Topic

Multithreading for decision tree induction process.

Nowadays, much of modern personal computers (PC) have multicore processors. The computer operates as if it had multiple processors. Software and data mining algorithms must be modified in order to benefit of this new feature.

Currently, few free tools exploit this opportunity because it is impossible to define a generic approach that would be valid regardless of the learning method used. We must modify each existing learning algorithm. For a given technique, decomposing an algorithm into elementary tasks that can execute in parallel is a research field in itself. In a second step, we must adopt a programming technology which is easy to implement.

In this tutorial, I propose a technology based on threads for the induction of decision trees. It is well suited in our context for various reasons. (1) It is easy to program with the modern programming languages. (2) Threads can share information; they can also modify common objects. Efficient synchronization tools enable to avoid data corruption. (3) We can launch multiple threads on a mono-core and mono-processor system. It is not really advantageous, but at least the system does not crash. (4) On a multiprocessor or multi-core system, the threads will actually run at the same time, with each processor or core running a particular thread (http://en.wikipedia.org/wiki/Thread_%28computer_science%29). But, because of the necessity of synchronization between threads, the computation time is not divided by the number of cores in this case.

In the next section, we briefly present the modification of the decision tree learning algorithm in order to benefit of the multithreading technology. Then, we show how to implement the approach with **SIPINA** (version 3.5 and later). We show also that the multithreaded decision tree learners are available in various tools such as **Knime 2.2.2** or **RapidMiner 5.0.011**. Last, we study the behavior of the multithreaded algorithms according to the dataset characteristics.

2. Multithreading for decision tree induction

Learning algorithm

We use a variant of the CHAID algorithm. Because it is based on a pre-pruning technique for the determination of the right size of the tree, it is especially fast on large datasets. Recently, authors show that the multithreading technique is only interesting in the growing phase for the learning algorithms such as C4.5. The post-pruning phase does not need to access the dataset, it is always very fast (Aldinucci et al., 2010[i]).

During the growing phase, when we want to split a node, we must search the most relevant attribute. The process maximizes a criterion which is computed for each candidate attribute. The algorithm can be summarized as follows.

```
Function Split (candidate attributes): selected attribute
Max = -∞
Selected attribute = NULL
For Each Candidate Attribute
      Relevance = Goodness of split (attribute)
      If (Relevance > Max) Then
            Max = Relevance
            Selected attribute = attribute
      End If
End For
Return (Selected attribute)
```

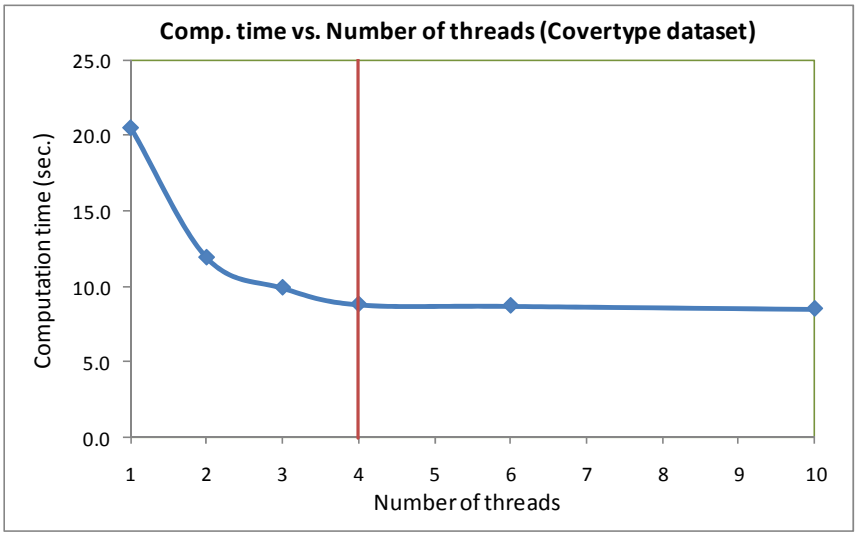
The computation of the goodness of split of the candidate attributes can be launched independently. But, we must not set a thread for each attribute. They can be very numerous. It is more judicious to use a queue. We set the maximum number of authorized threads. We launch initially this number of threads. When a thread is terminated, it verifies if there are still untreated candidate attributes. If the number of current threads is not overtaken, then it launches the analysis of the next attribute into the queue. The maximum number of threads can be the same as the number of cores of the processor. But we can set also another value, according to our specific constraints.

The reduction of the computation time is not linear in the number of processor cores. The first reason is that threads manipulate the same target variable. I do not really assess the delay caused by multiple threads accessing the same memory area. But I think it is not negligible. The second reason is more restrictive still. To detect the variable maximizing the relevance, threads must update common objects. We must not produce inappropriately inconsistencies during the updating of "max relevance" and "selected variable". To protect the process, I use critical sections into the source code. It is obvious that this kind of synchronization generates expectations cycles during execution of the algorithm. In fact, we find that adding one additional thread in the calculations leads to a reduction in execution time less important. Last, as we will see in the experiments below, it is useless to ask for more threads than cores on a system.

Experiments on the « covertype » dataset

We use the “Forest Cover” dataset in order to evaluate the relevance of the solution implemented into SIPINA. The data file contains 581,012 instances and 54 predictive attributes. The target attribute has 7 values. This is not really a very large dataset. But on a personal computer, handling this kind of data is already a challenge. We show in the

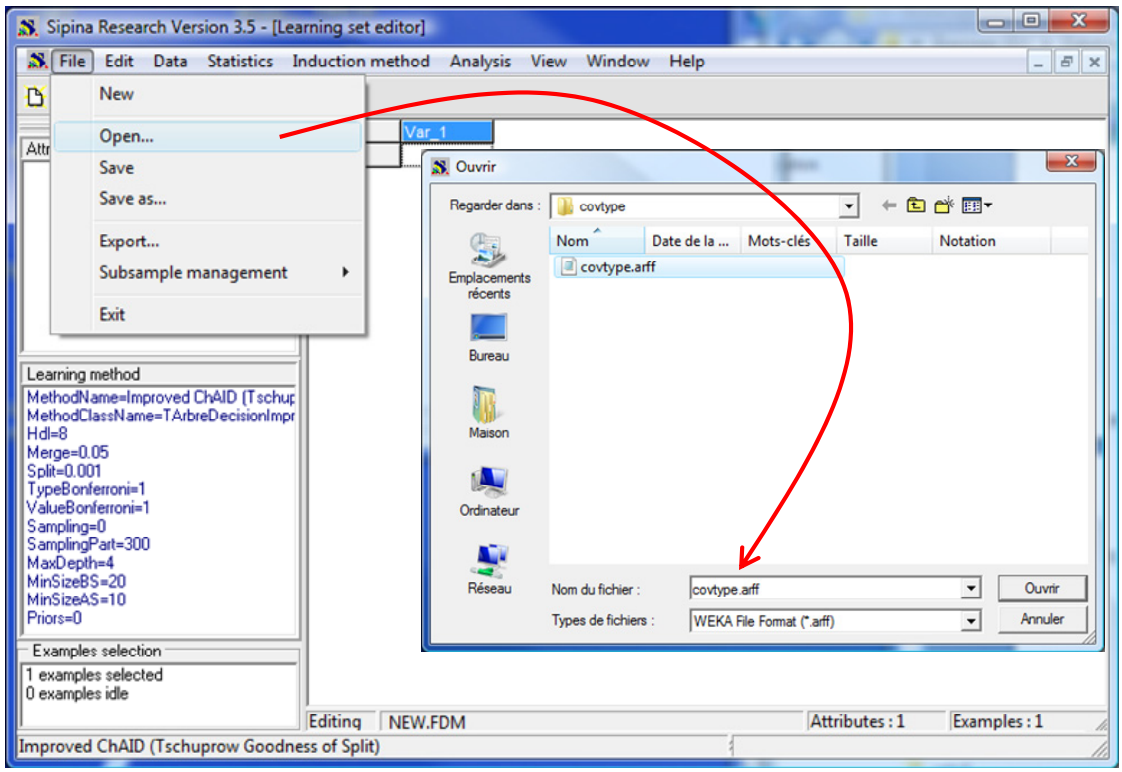
following figure the computation time according the number of thread used on a quad-core computer. As long as we stay within the limits of available cores, adding an extra thread reduces the computation time. Afterwards, indeed, as one moves to 6 threads or 10 threads, the computation time no longer decreases.



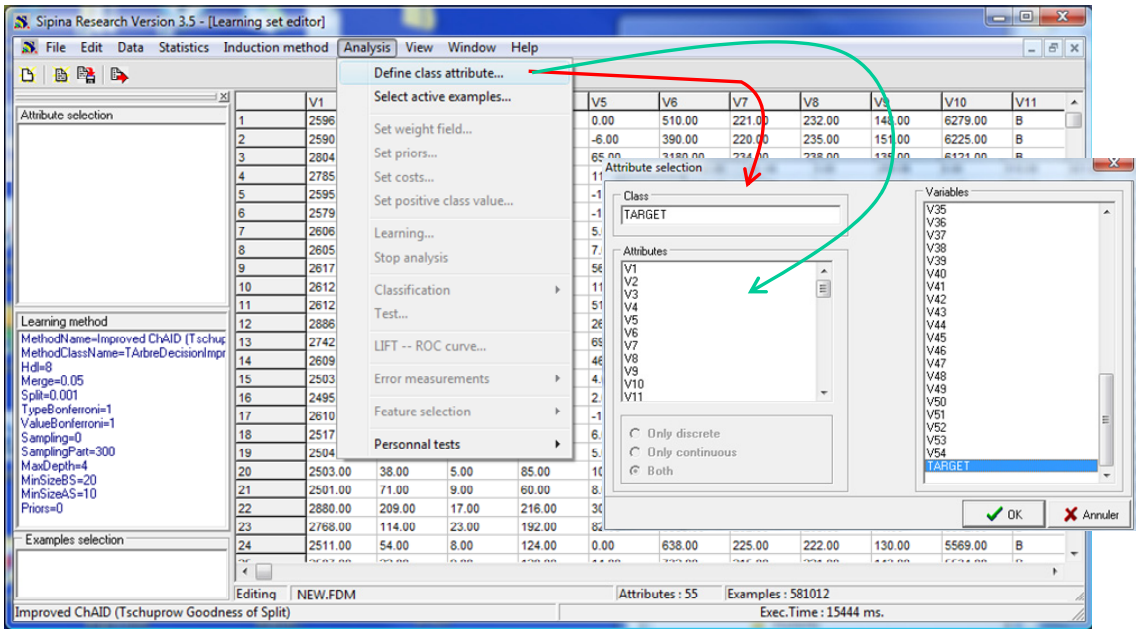
3. Multithreading with SIPINA 3.5

Importing the dataset

We click on the FILE / OPEN menu to load the data file. We select « covertype.arff » (WEKA file format).

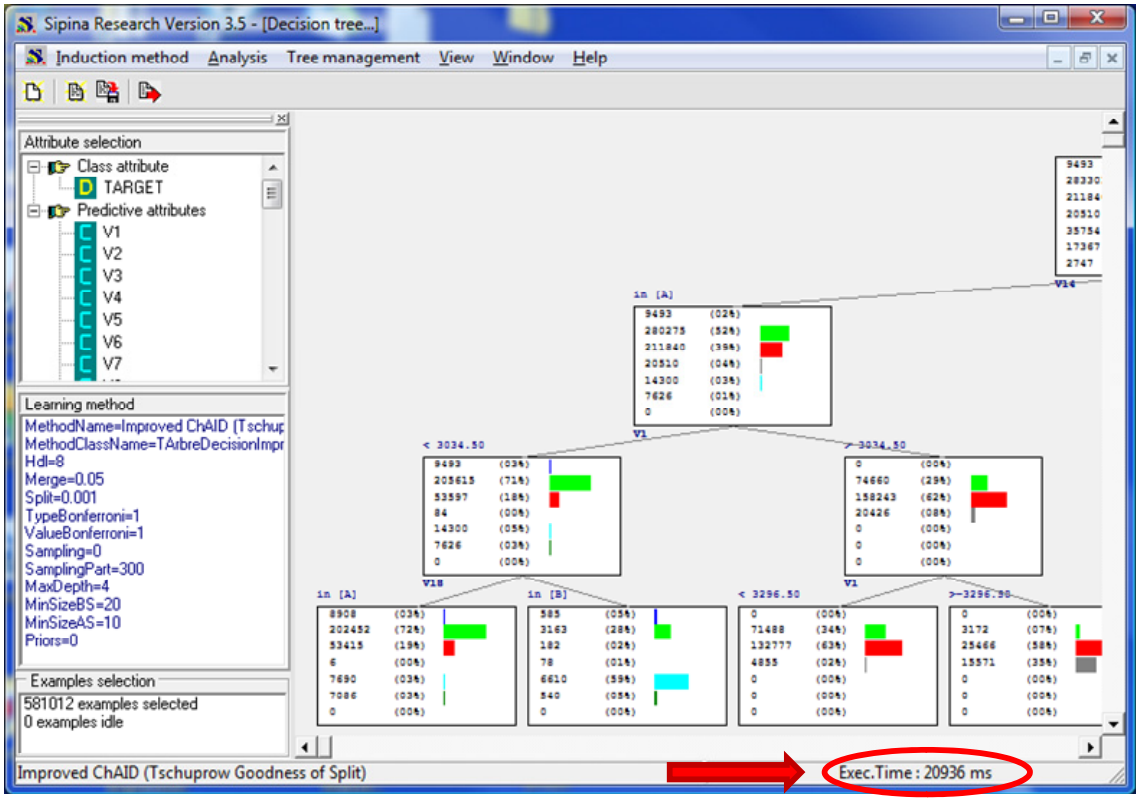


When the dataset is loaded, we select the ANALYSIS / DEFINE CLASS ATTRIBUTE menu to specify the target and the input attributes.

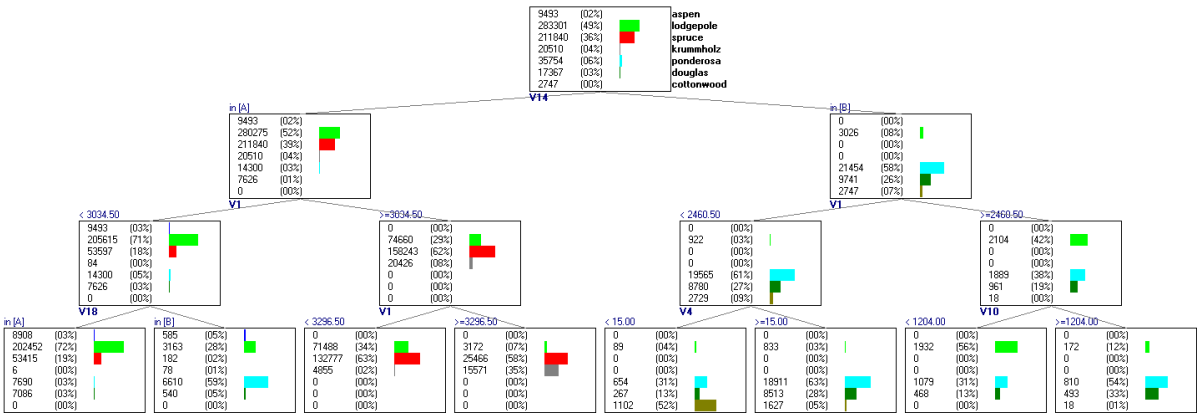


Monothreaded decision tree induction (CHAID)

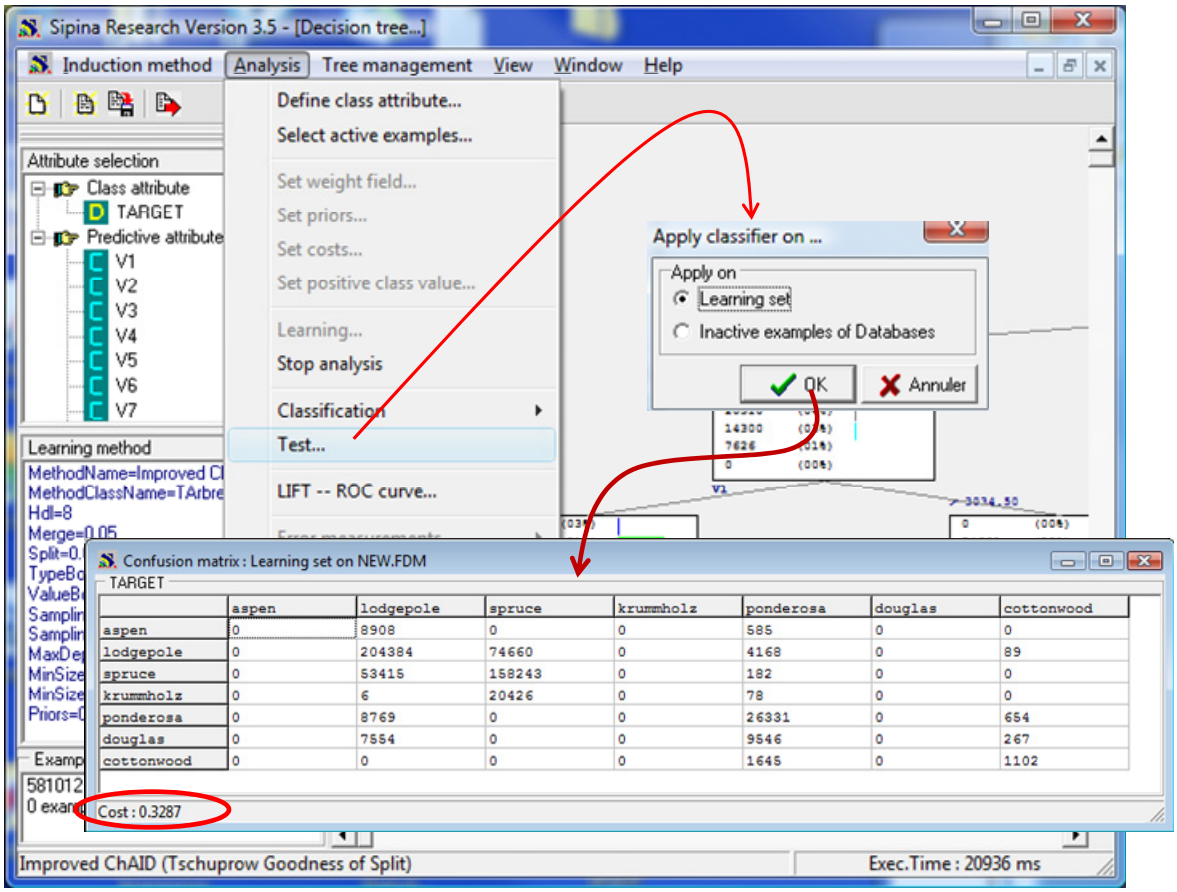
By default, SIPINA uses a monothreaded variant of the CHAID algorithm. The number of levels of the tree is limited to 4. It is often adequate in an exploration phase of the dataset. We click on ANALYSIS / LEARNING the menu.



After **20.9 seconds**, the learning phase is completed. Here are the details of the tree.



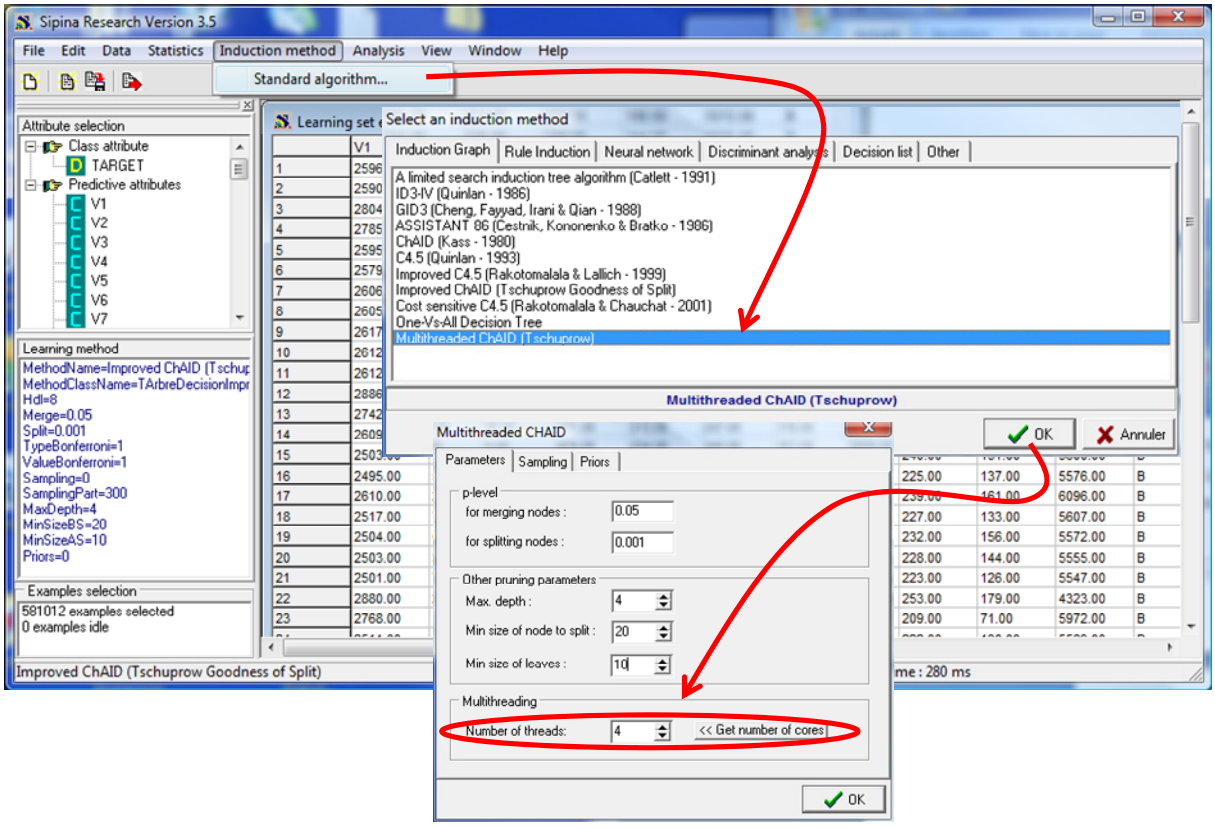
We compute the confusion matrix and the error rate on the learning set. The aim is not really to evaluate the generalization performance of the tree, but rather to obtain some indicators that we can compare to the multithreaded algorithm. We click on the ANALYSIS / TEST menu; then we select the LEARNING SET option into the dialog settings.



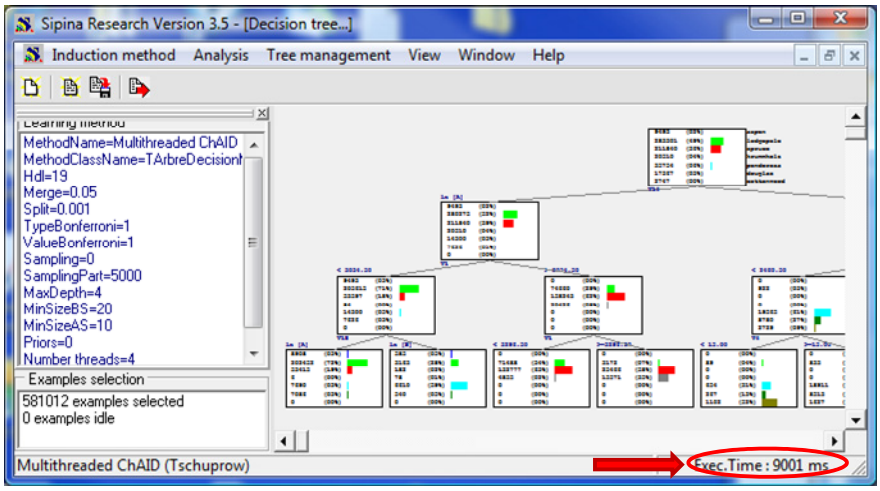
The resubstitution error rate is 32.87%.

Multithreaded decision tree induction

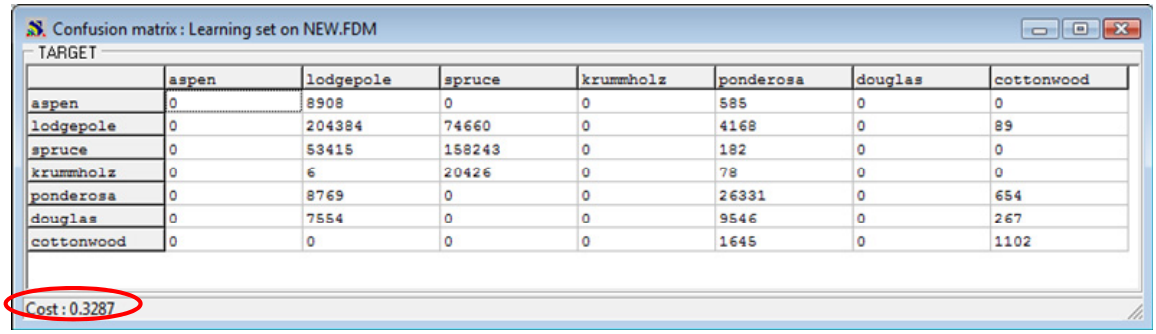
We stop the current analysis by clicking on the ANALYSIS / STOP ANALYSIS menu. In order to select the new induction algorithm, we activate the INDUCTION METHOD / STANDARD ALGORITHM menu. Into the dialog settings, we choose MULTITHREAD CHAID (TSCHUPROW). We click on the OK button. A new dialog box appears. We click on the GET NUMBER OF CORES button. SIPINA detects automatically the number of cores of the processor.



Again, we click on the ANALYSIS / LEARNING menu. We obtain the tree after **9 seconds**!



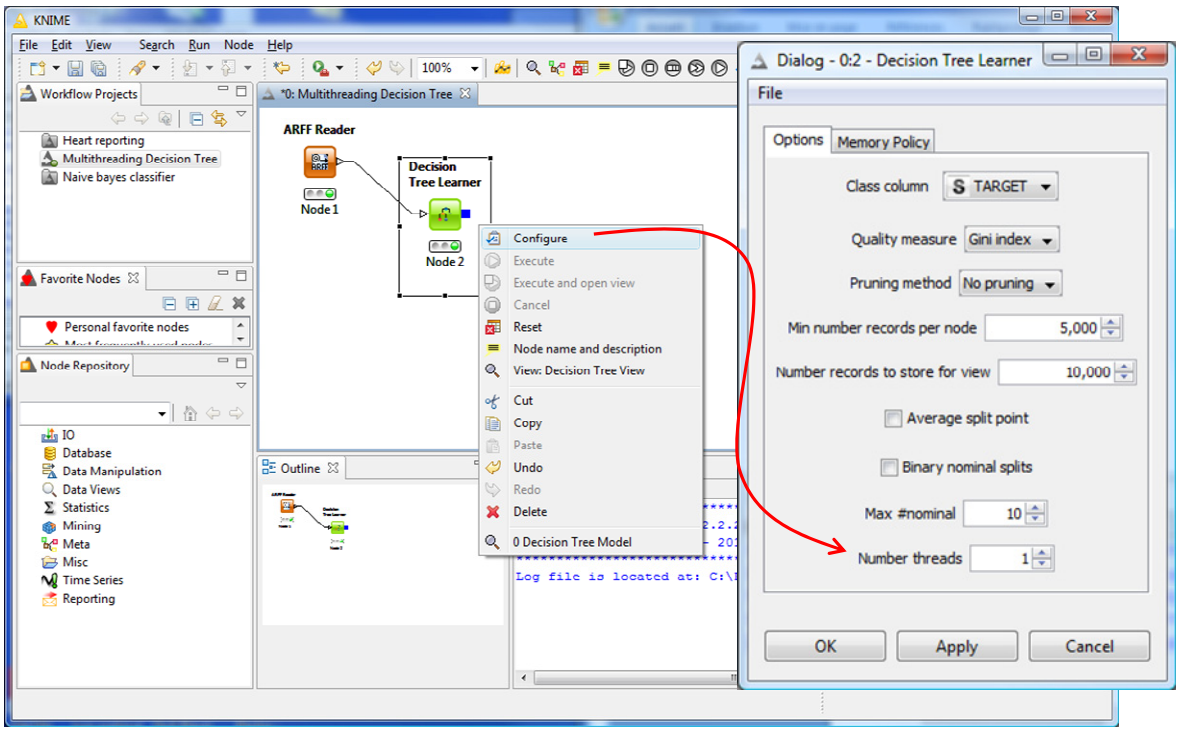
By analyzing the tree, we note that this is the same as before. This result is confirmed by the confusion matrix computed on the learning set.

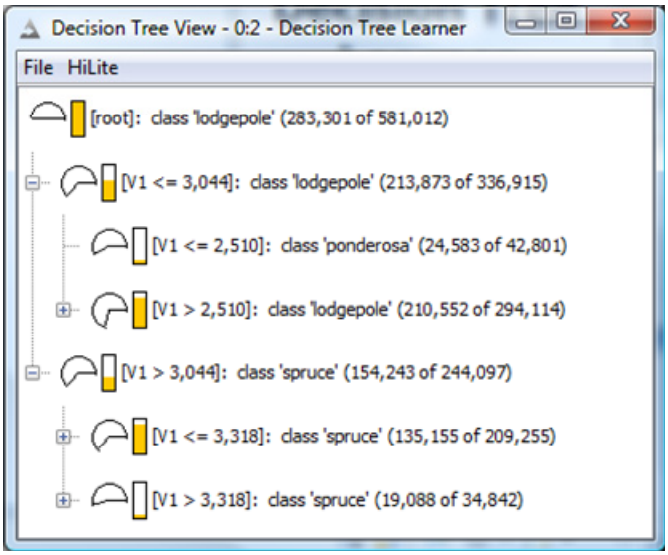


	aspen	lodgepole	spruce	krummholz	ponderosa	douglas	cottonwood
aspen	0	8908	0	0	585	0	0
lodgepole	0	204384	74660	0	4168	0	89
spruce	0	53415	158243	0	182	0	0
krummholz	0	6	20426	0	78	0	0
ponderosa	0	8769	0	0	26331	0	654
douglas	0	7554	0	0	9546	0	267
cottonwood	0	0	0	0	1645	0	1102

4. Multithreaded decision tree learner with Knime 2.2.2

Knime incorporates a multithreaded decision tree learner. I define the following workflow project. Let us see the settings of the decision tree algorithm.

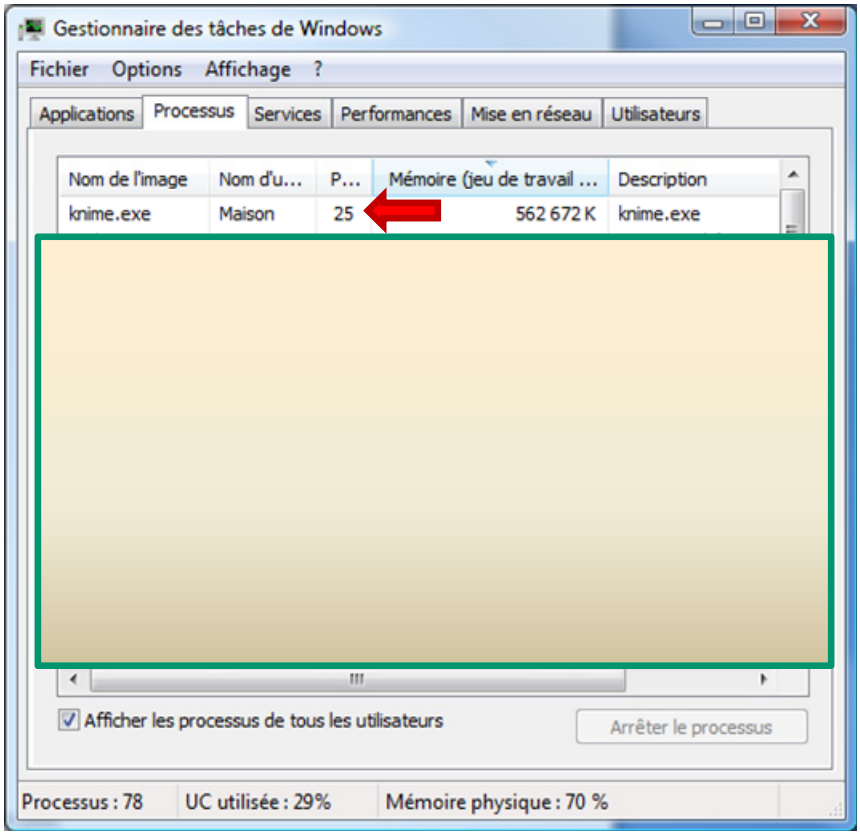




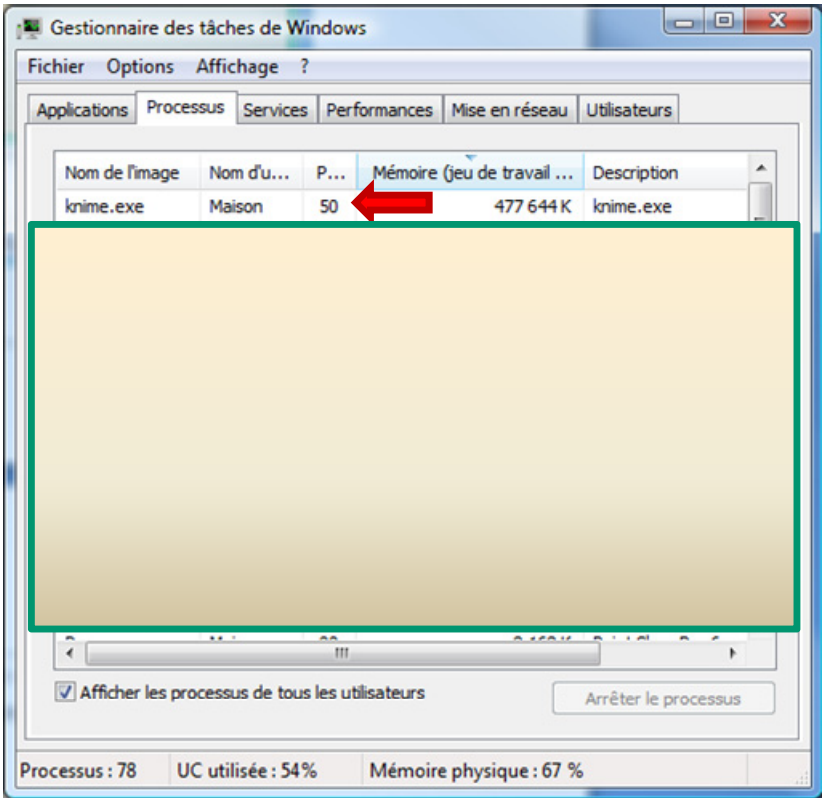
In a first time, we ask only one thread for the learning phase. We want to obtain a limited sized tree by increasing the minimum number of instances of a node (5000 instances). To measure the execution time, we use a watch because Knime gives no indication about this.

After **110 seconds**, we obtain the following tree. We show only the upper part of the tree.

More importantly, using the Windows Task Manager, we note that Knime uses only one core of the processor. This is consistent with the parameters that we have specified.



Let us see what happens when we ask 4 threads for the processing (NUMBER THREADS = 4).

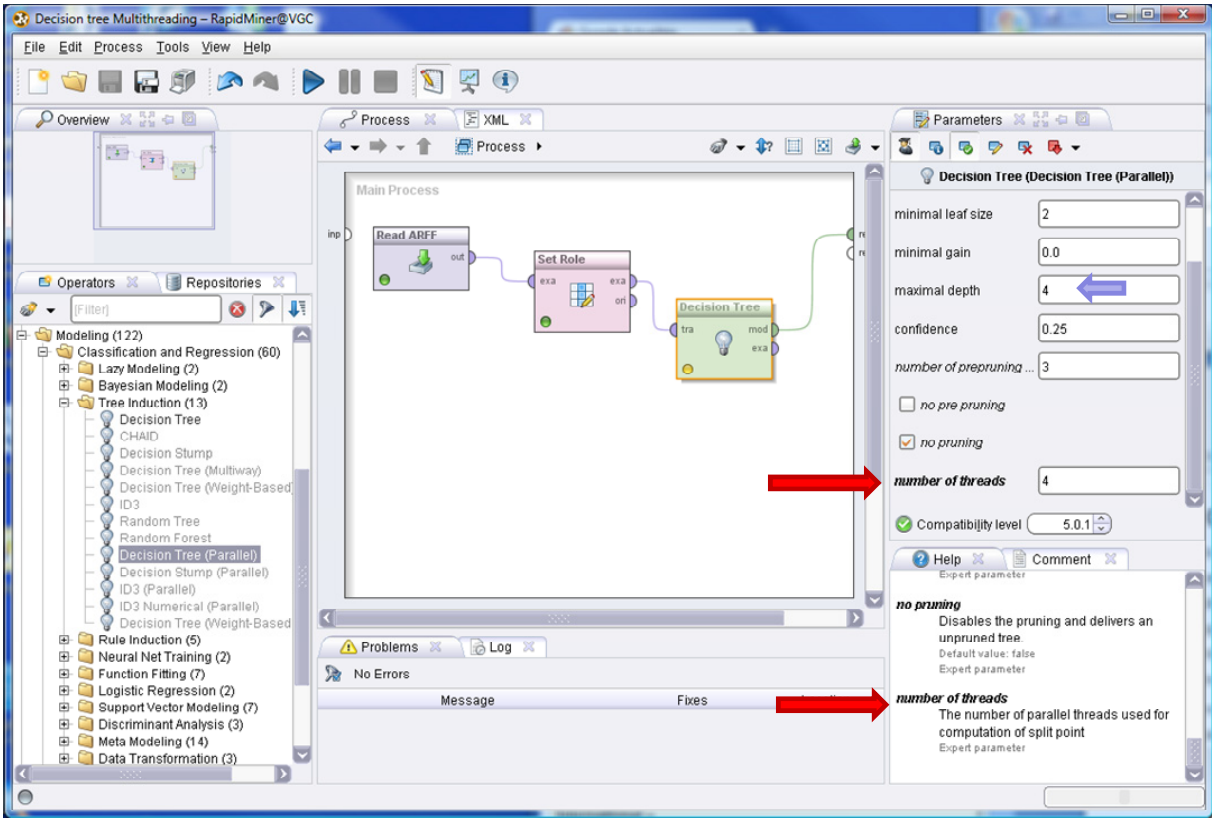


The computation time becomes **85 seconds**. But, Knime seems to use only two cores according to the Windows task manager. It seems that Knime implements multithreading differently during the induction of the tree. To have experienced it myself, I guess it allocates each thread to the node processing as a whole (and not the treatment of variables within a node). Because the tree is binary, only two threads are created at each step during the learning process. But that remains a guess. A look at the source code should confirm or not this idea.

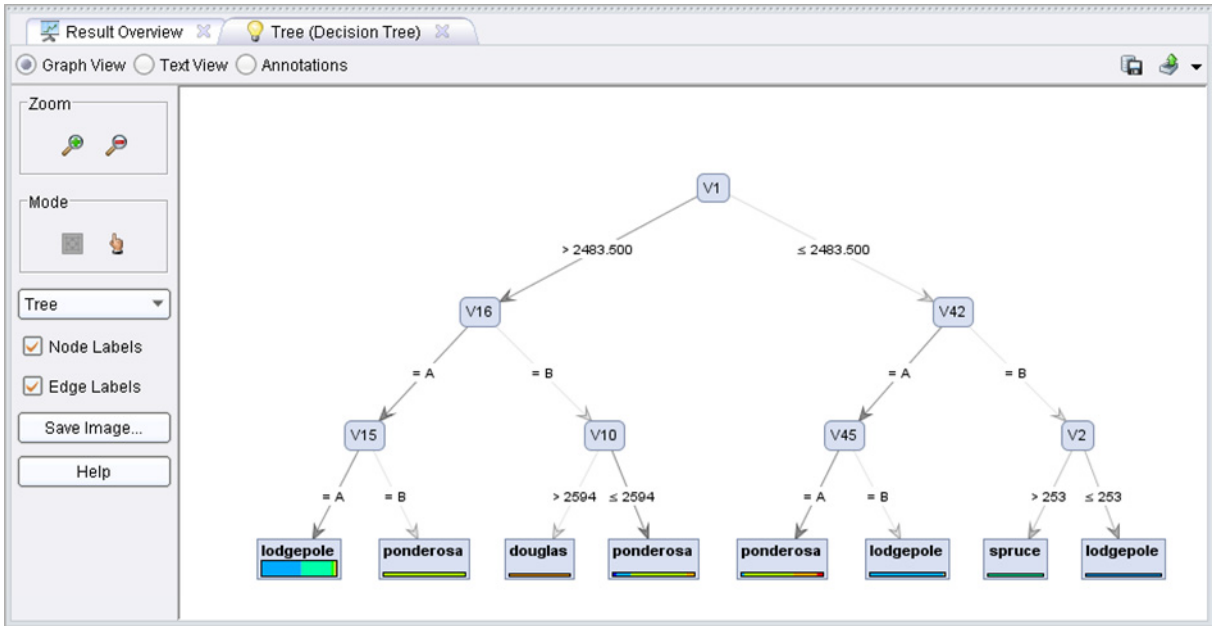
5. Multithreaded tree learner with RapidMiner 5.0.011

RapidMiner 5.0.011 incorporates the multithreading strategy for the decision tree learning algorithm. In fact, it provides both the simple (mono-threaded) and the multithreaded versions of each decision tree learner. RapidMiner computes only the duration of the whole process (including the loading of the data file).

We create the following diagram. We set the parameters in order to obtain a tree with a maximum of 4 levels.



We obtain the following tree.



With the mono-threaded strategy, the computation time is 87 seconds, including the loading of the dataset. With 4 threads, it becomes 57 seconds. We gain 30 seconds during the learning of the tree. According to the available papers, it seems that the implementations of the multithreading strategy are very similar between RapidMiner and Sipina.

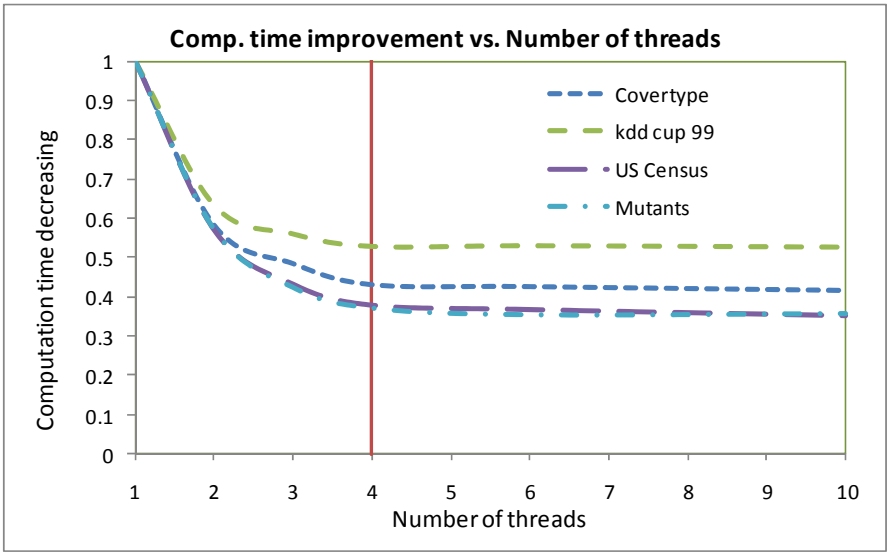
6. Conclusion

In this tutorial, we showed that multithreading appropriately incorporated into the process of decision tree induction can reduce the computing time on machines with multicore processors. It remains, however, two important aspects to explore: the gain certainly depends on the characteristics of the induced tree (depth, number of nodes); it also depends on the characteristics of the dataset (number of instances, number and type of predictors).

About this last subject, we have assessed the behavior of SIPINA on various dataset.

Dataset	# Instances	# all predictors	# continuous predictors
Covertype	581012	54	10
Kdd-cup 99	4817099	41	0
US Census 1990	2458285	67	67
Mutants	16592	5408	5408

Based on the computation time of the monothreaded algorithm, we observe the relative decrease in execution time depending on the number of threads in the following figure (CHAID algorithm, the tree is limited to four levels).



These experiments confirm that asking more threads than cores (CPU) is useless.

Seemingly, the increase in the number of threads (when we do not overtake the number of available cores) is especially beneficial for databases with a high proportion of continuous descriptors (e.g. U.S. Census 1990, Mutants). It seems logical. Indeed, the handling of these attributes during the node splitting process is very CPU intensive if one refers to the strategy implemented in SIPINA,. It requires the sorting of instances and the research of the optimal discretization cut point.

Perhaps, if we adopt another approach – the one which is implemented in Tanagra for instance, where the variables are sorted once and for all before the construction of the tree – the behavior of multithreading will be a little different according to the data characteristics.

[i] Aldinucci, Ruggieri, Torquati, « Porting Decision Tree Algorithms to Multicore using FastFlow », Pkdd-2010.
http://www.di.unipi.it/~ruggieri/Papers/pkdd2010.pdf