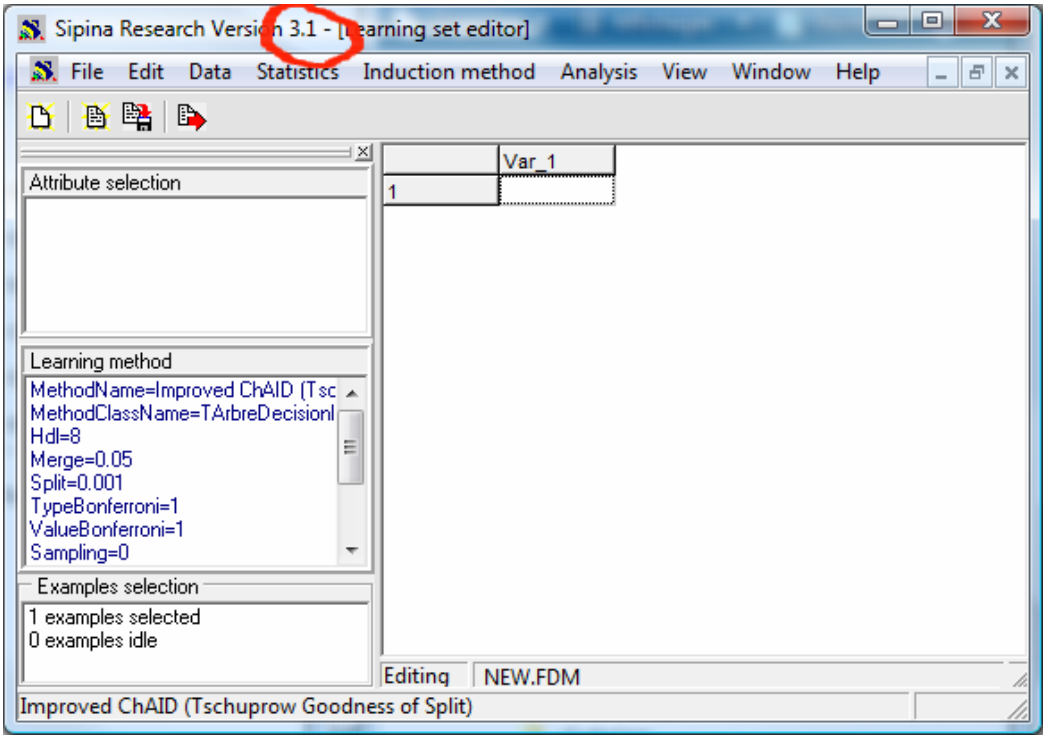
1 Introduction

Sipina: Speed up decision tree induction via local sampling.

Caution: in order to realize this tutorial, we must use the 3.1 version of Sipina Research. Please, check the version number in the title bar.



During the decision tree learning process, the algorithm detects the better variable according to a goodness of fit measure when it tries to split a node. The calculation can take a long time, particularly when it deals with a continuous descriptors for which it must detect the optimal cut point.

For all the decision tree algorithms, Sipina introduces a **local sampling option** when it searches the best splitting attribute on a node. The idea is the following: on a node, it draws a random sample of size **n**, and then all the computations are made on this sample. Of course, if **n** is lower than the number of the existing examples on the node, Sipina uses all the available examples. It occurs when we have a very large tree with a high number of nodes.

We have described this approach in a paper (Chauchat and Rakotomalala, IFCS-2000)[1]. We describe in this tutorial how to implement it with Sipina.

We note in this tutorial that using a sample on each node allows to reduce drastically the computation time without loss of accuracy. But, we can obtain a tree which can be different than dealing with the

[1] J.H. Chauchat and R. Rakotomalala, « A new sampling strategy for building decision trees from large databases », Proc. Of IFCS-2000, 199-204, 2000.

entire dataset. But is it really a problem? We know that the decision trees are very unstable. If we use a different dataset, we can obtain a very different tree (for instance, they do not use the same splitting attributes). But on the other hand, it does not mean that the models classify the examples of the test set differently. One way to check this idea is to compare the generalization error rate.

In the tutorial, we show the interest of the local sampling strategy for the induction of the decision tree:

- First, we build the tree on the whole dataset using the usual approach. We measure the computation time and the test error rate.
- Then, we use the local sampling strategy with n = 5000. We compare the performances (error rate and computation time).
- The sample size above is in fact very conservative. The enough sample size depends on various factors (complexity of the underlying concept, densities of the examples into the representation space, etc.). We see on our dataset that about a hundred of examples are enough when we search the best splitting attribute on a node, without loss of accuracy.
- Last, we compare the computation time with the implementation of the decision tree algorithm of Tanagra[2] where another strategy is used in order to speed up the learning process when we handle continuous attributes.

2 Dataset

We use the famous WAVEFORM dataset (Breiman and al., 1984)[3]. The class attribute has 3 values. There are 21 continuous descriptors. We have generated 2,000,000 examples; the half is used for the learning process, the other half for the evaluation of the classifier. The data file is in ARFF (Weka) file format[4].

We have used another version of this dataset previously[5]. It will be interesting to compare the behavior of Sipina and Tanagra on the same dataset (class attributes and descriptors) but with 4 times over examples.

3 Analysis with SIPINA

3.1 Preparing the learning process

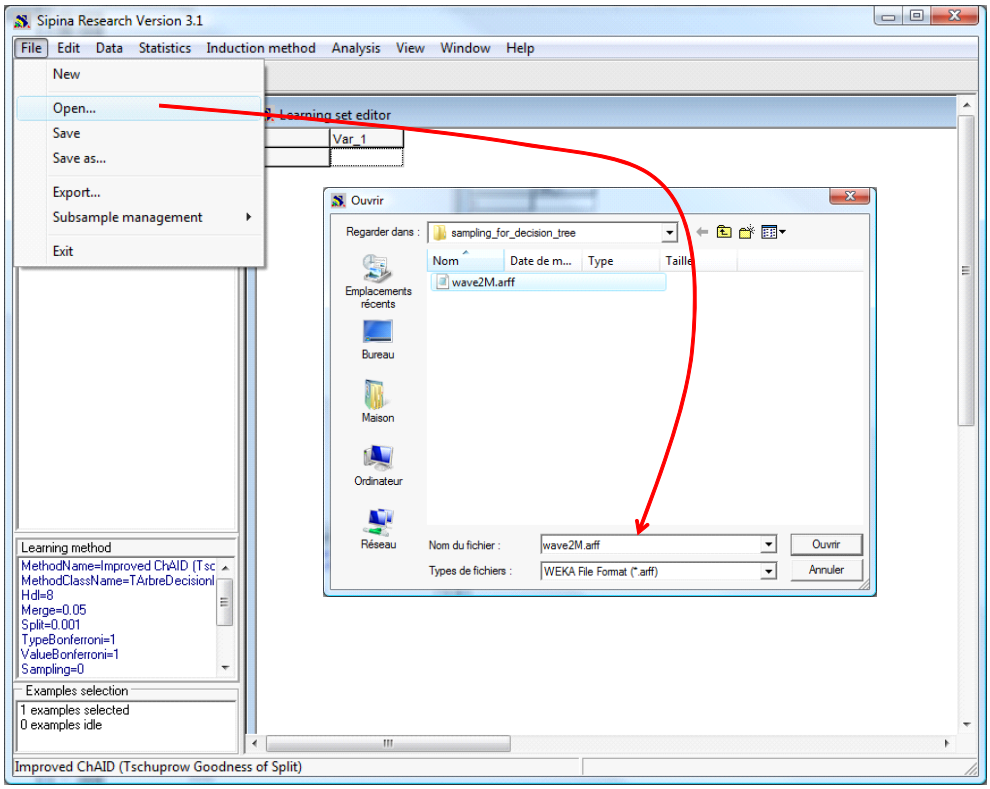
After we launch Sipina, we click on the FILE / OPEN menu. We choose the WEKA (ARFF) file format. We select the WAVE2M.ARFF data file. During the loading, the progress bar seems frozen. But it does not matter. The process is running.

[2] http://eric.univ-lyon2.fr/~ricco/tanagra/en/tanagra.html

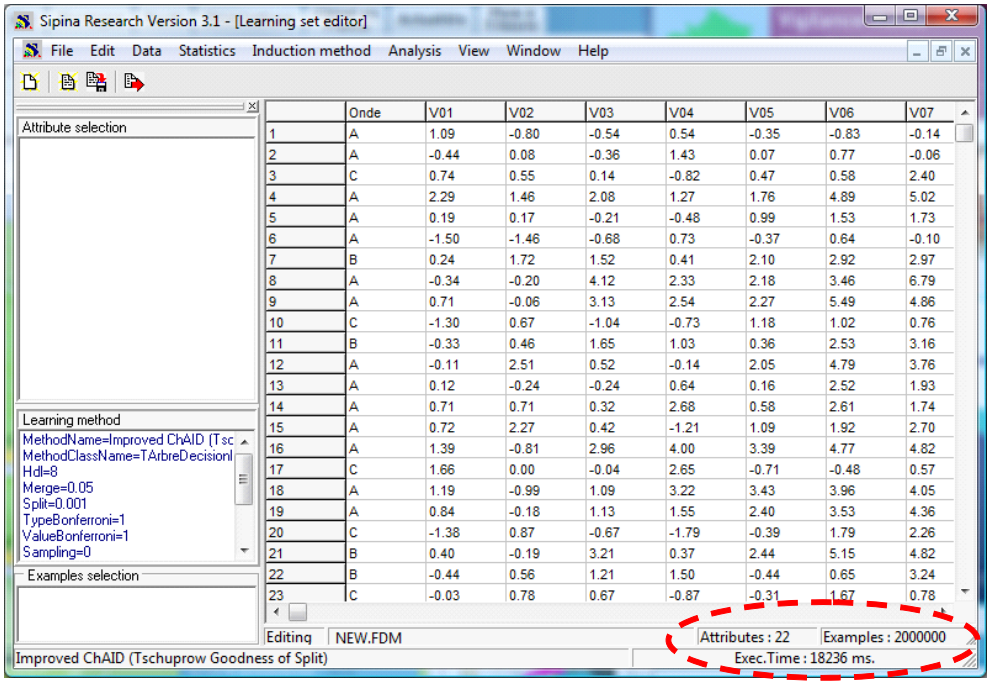
[3] http://archive.ics.uci.edu/ml/datasets/Waveform+Database+Generator+(Version+1)

[4] http://eric.univ-lyon2.fr/~ricco/tanagra/fichiers/wave2M.zip

[5] http://data-mining-tutorials.blogspot.com/2008/11/decision-tree-and-large-dataset.html

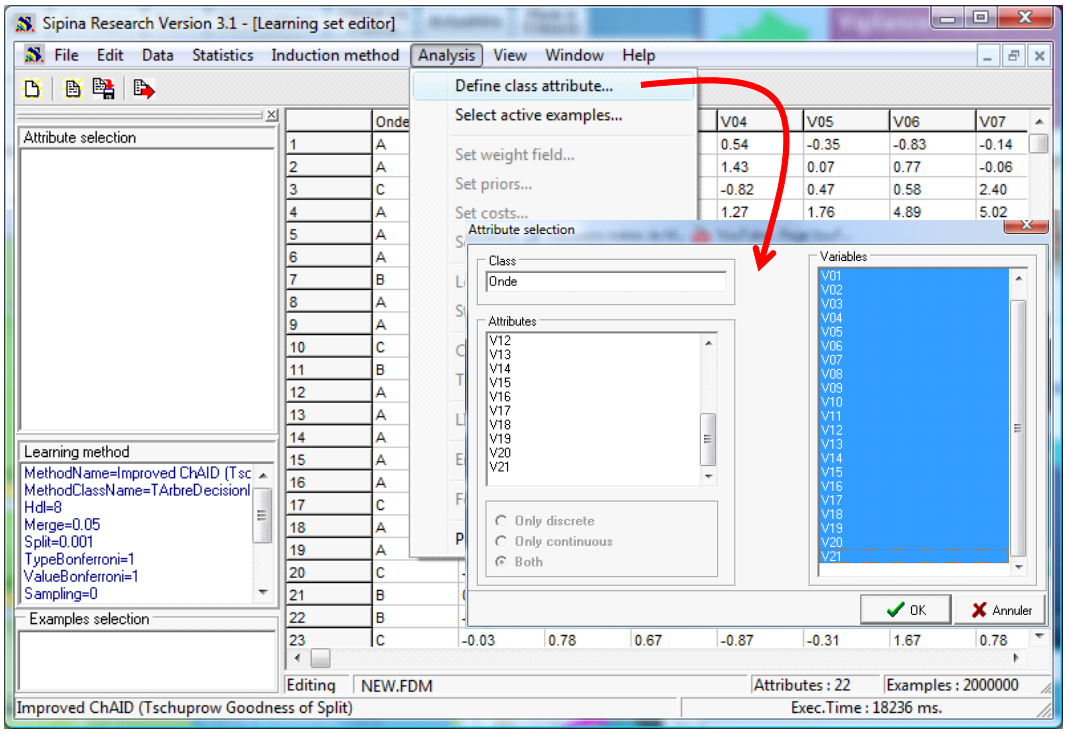


After 18 seconds, the values are displayed in the visualization grid. We have 22 columns and 2,000,000 rows.



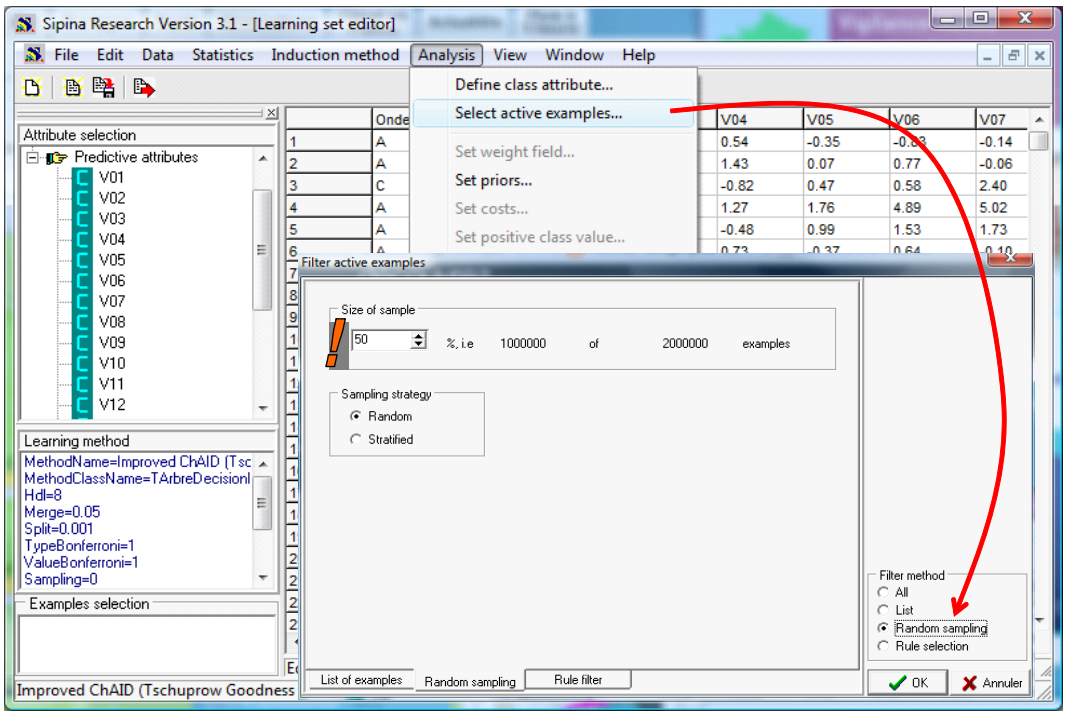
Then, we want to define the types of the variables and to specify the train and the test set.

In order to define the types of the variables, we click on the ANALYSIS / DEFINE CLASS ATTRIBUTES menu. Using drag and drop, we set ONDE as CLASS, the others (V1 to V21) as ATTRIBUTES.



The resulting selection appears in the left part of the main window.

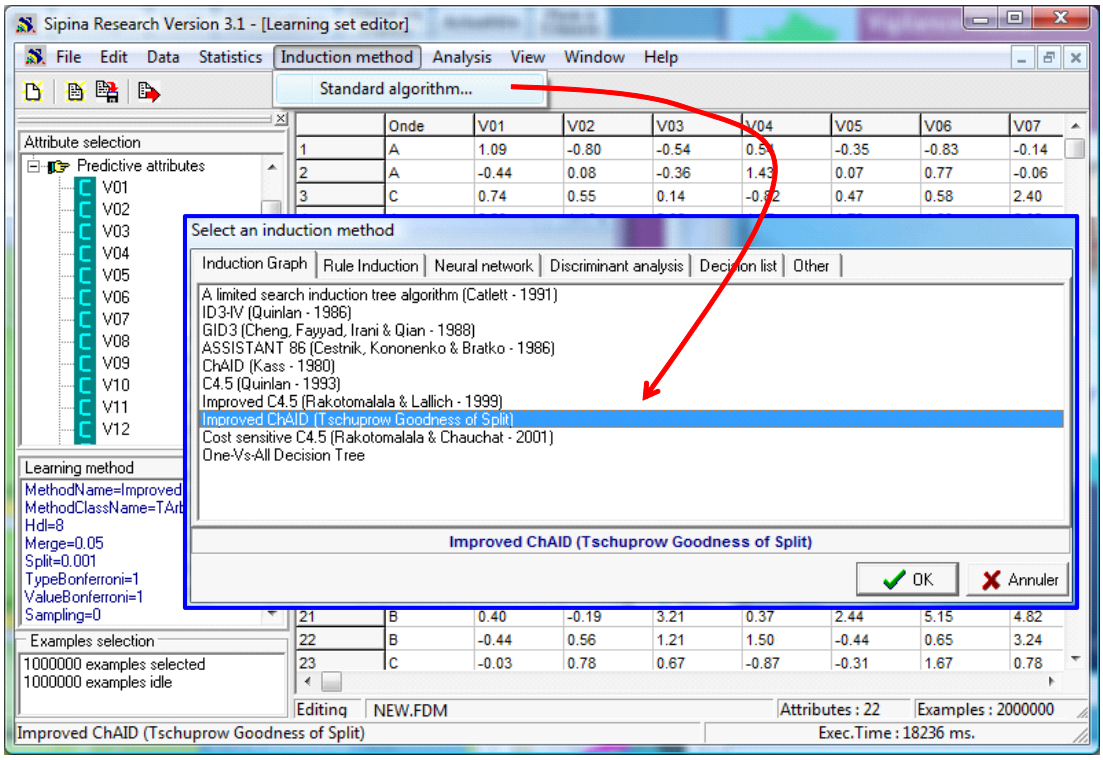
In order to subdivide the examples, we click on the ANALYSIS / SELECT ACTIVE EXAMPLES menu. We select the RANDOM SAMPLING 50/50 option.



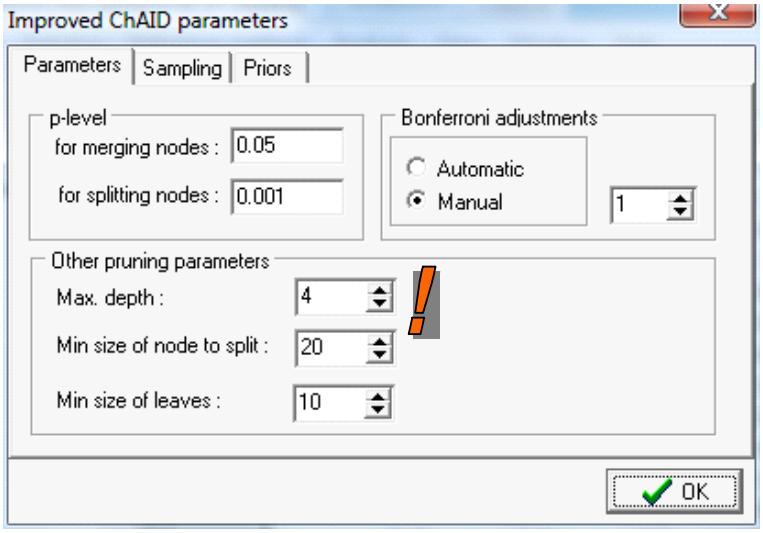
3.2 The usual decision tree learning algorithm

Selecting the method and the associated settings. For specifying the learning method, we click on the

INDUCTION METHOD / STANDARD ALGORITHM menu.

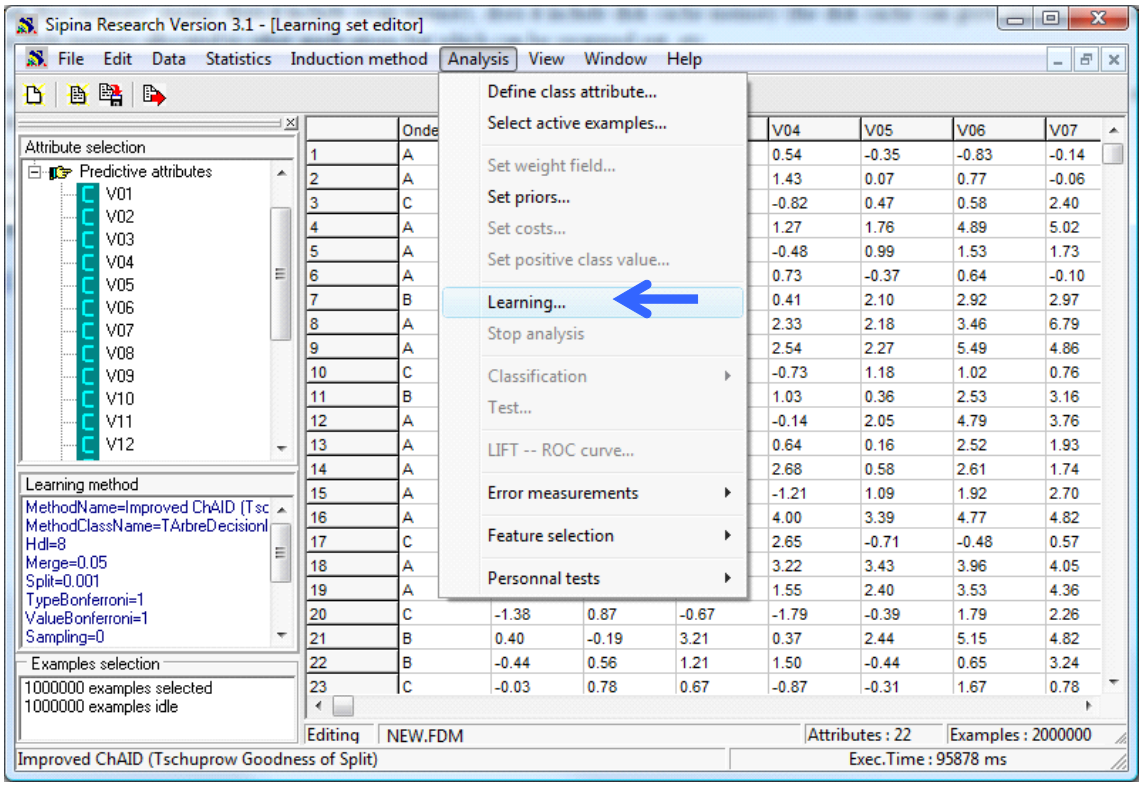


We select the IMPROVED CHAID approach in the dialog box. When we click on OK, the settings of the method appear.

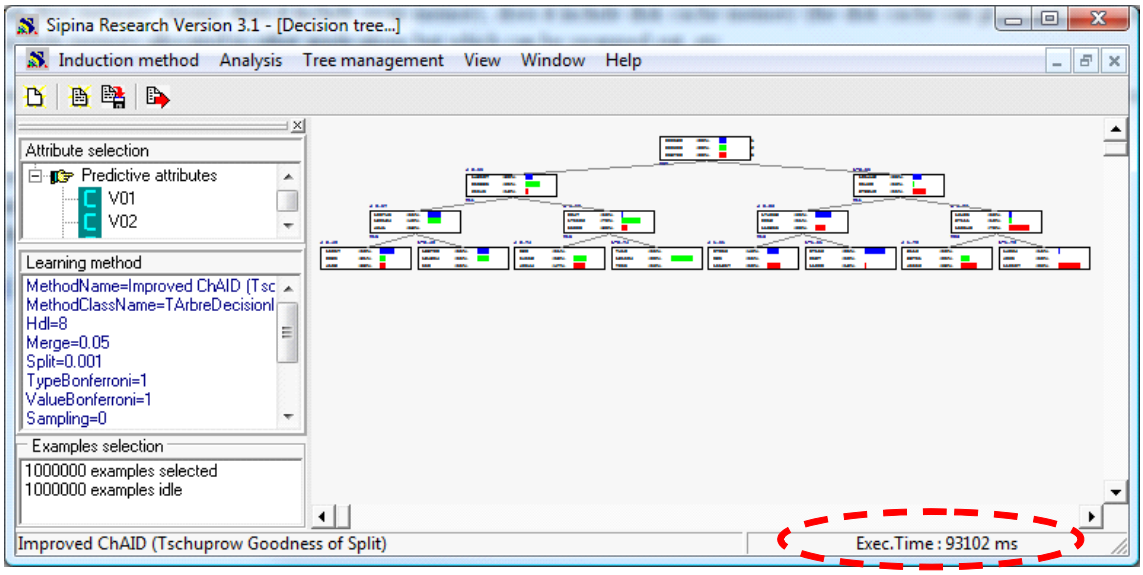


We do not modify anything for the moment. We note here that the number of levels of the tree is limited to 4 (MAX DEPTH = 4). This setting has no theoretical justification. It allows to control the size of the tree in order to obtain an easy to read model.

Training process. We launch the analysis by clicking on the ANALYSIS / LEARNING menu.



We obtain a tree with 8 leaves and, indeed, 4 levels (the first level is the root of the tree). The computation time is 93 seconds.



If we see the details the tree (Figure 1), we note the splitting attribute for the root node is V07; then, the nodes on the second level are split with the same attribute (V11), but not with the same cut points (3.57 at the left, 3.36 at the right); etc.

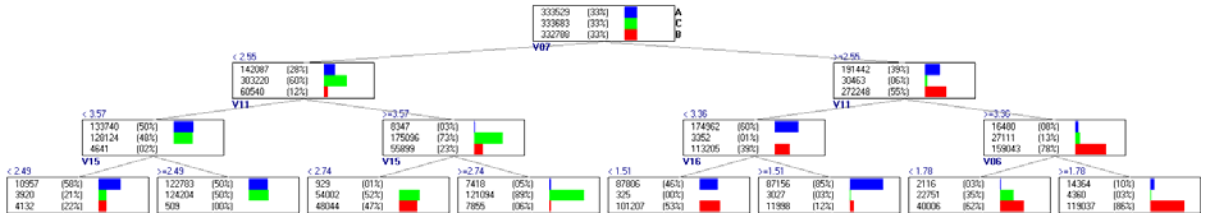


Figure 1 – Decision tree, without local sampling strategy

Evaluation on the test set. In order to compute the error rate on the test set, we click on the ANALYSIS / TEST menu. We can compute the confusion matrix on the training sample or the test sample. We choose this second option (Figure 2).

We obtain the confusion matrix and the error rate (34% - Figure 3). We keep in mind this value. It is the reference which we will use for evaluating the local sampling approach for the decision tree learning process.

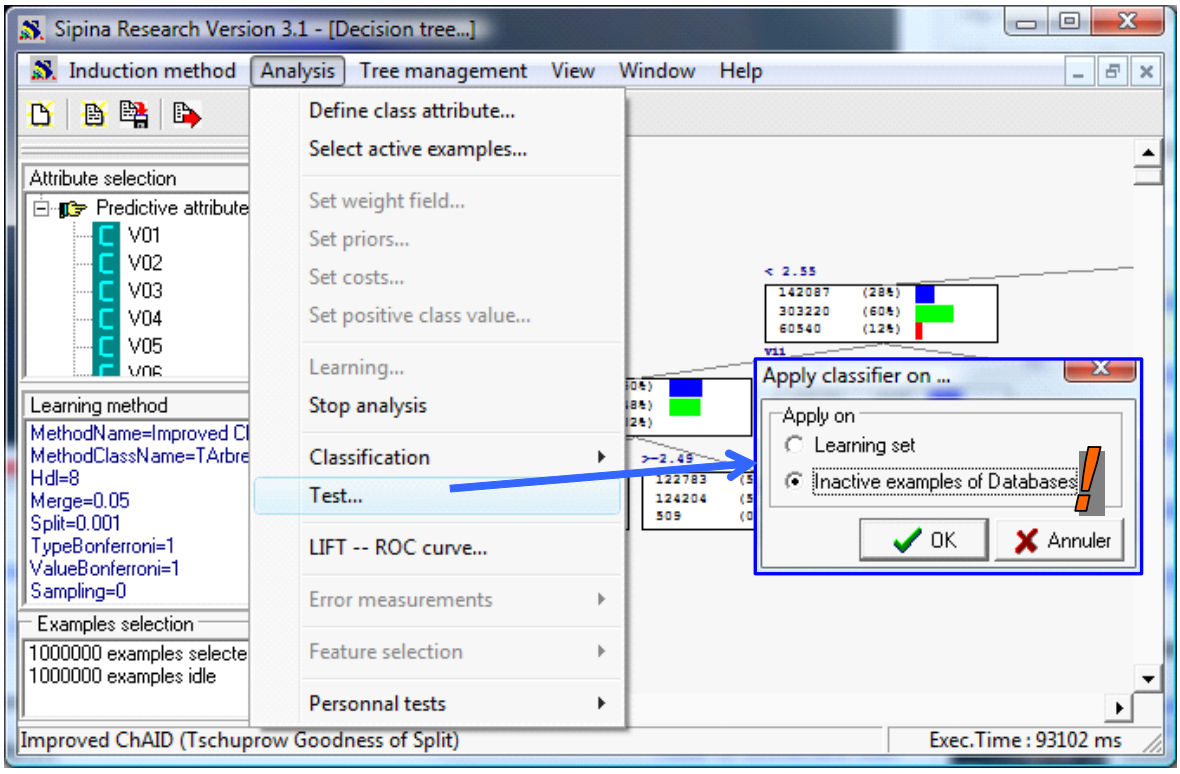


Figure 2 - Assessment on the test set

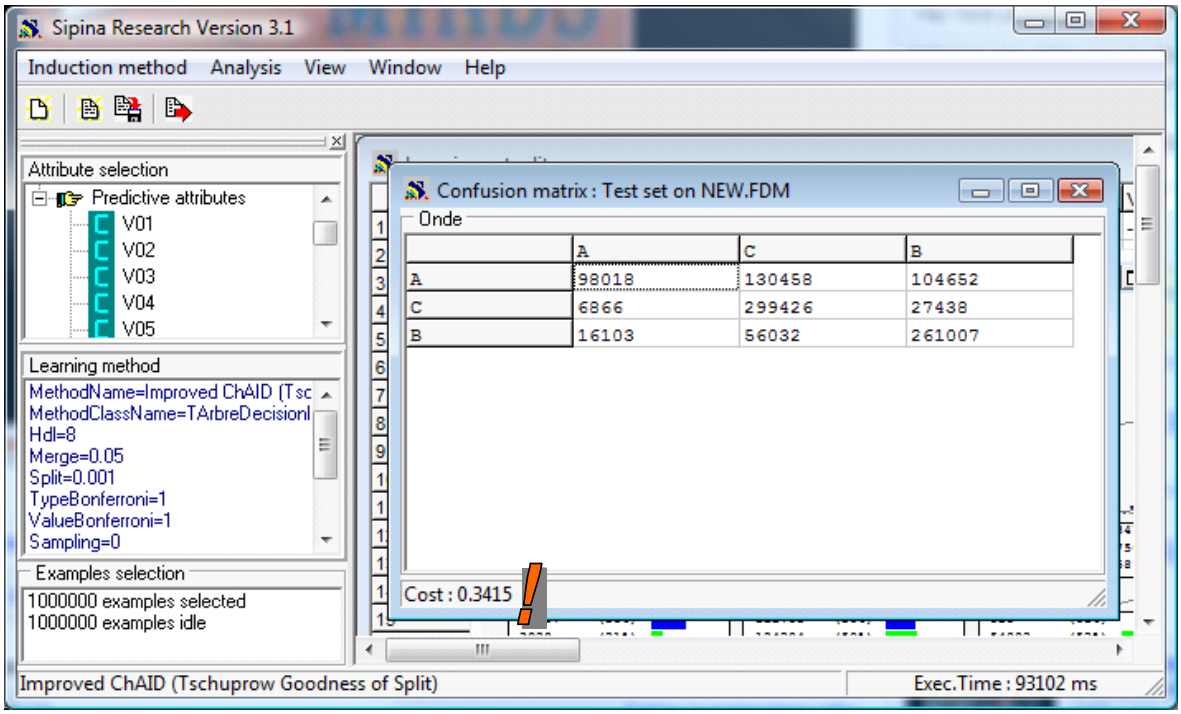


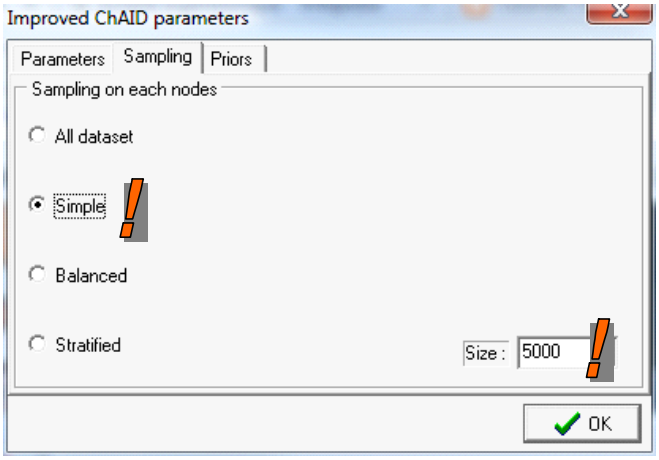
Figure 3 – Confusion matrix and test error rate

3.3 Local sampling strategy for the decision tree induction

We quit the current analysis by clicking on the WINDOW / CLOSE ALL menu.

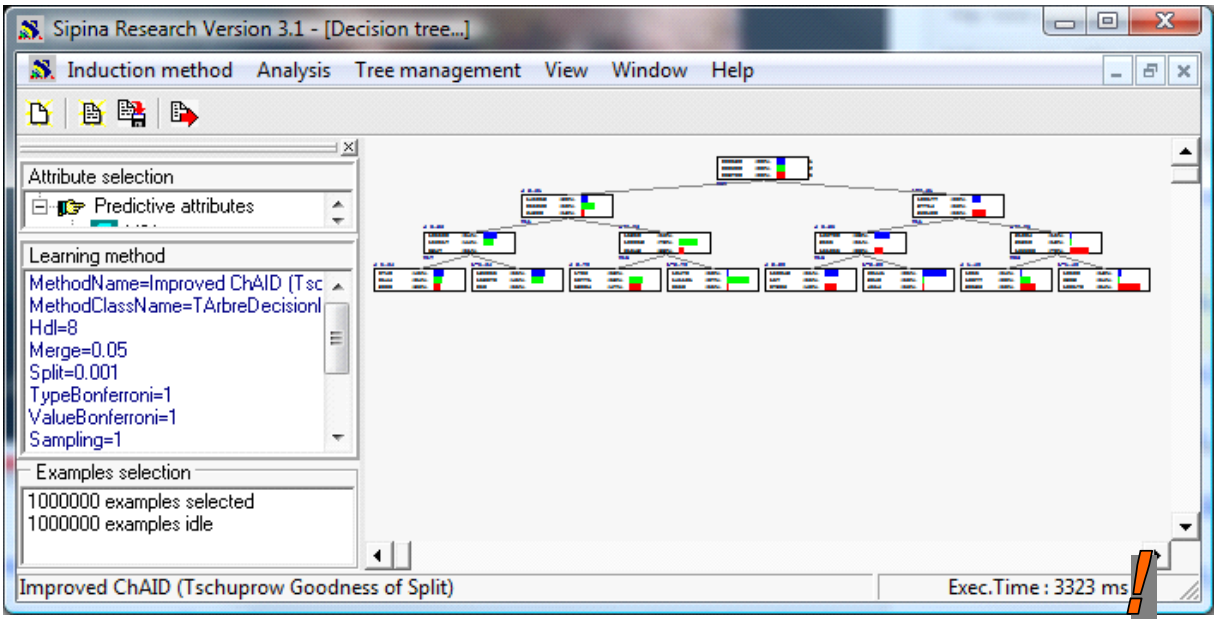
We want now to implement the local sampling strategy during the learning process. Again, we select the INDUCTION METHOD / STANDARD ALGORITHM menu then, as above, we select the IMPROVED CHAID method.

In the dialog settings, the SAMPLING tab is very important here. Rather than using all the examples on a node, we perform a sampling with n = 5000. For each node, all the calculations (detecting the right cut point, select the most important splitting attribute) are performed on the local sample.



We validate this choice. We can launch a new analysis by clicking on ANALYSIS / LEARNING menu. We

obtain a tree after 3 seconds! The computation time is drastically reduced.



SIPINA gives the class distribution on the whole dataset for each node, but the computations related to the decision tree induction are performed on the samples.

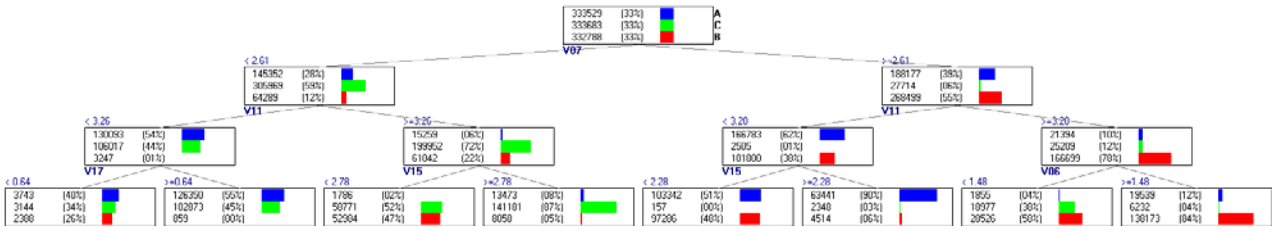


Figure 4 – Decision tree with the local sampling strategy (n = 5000)

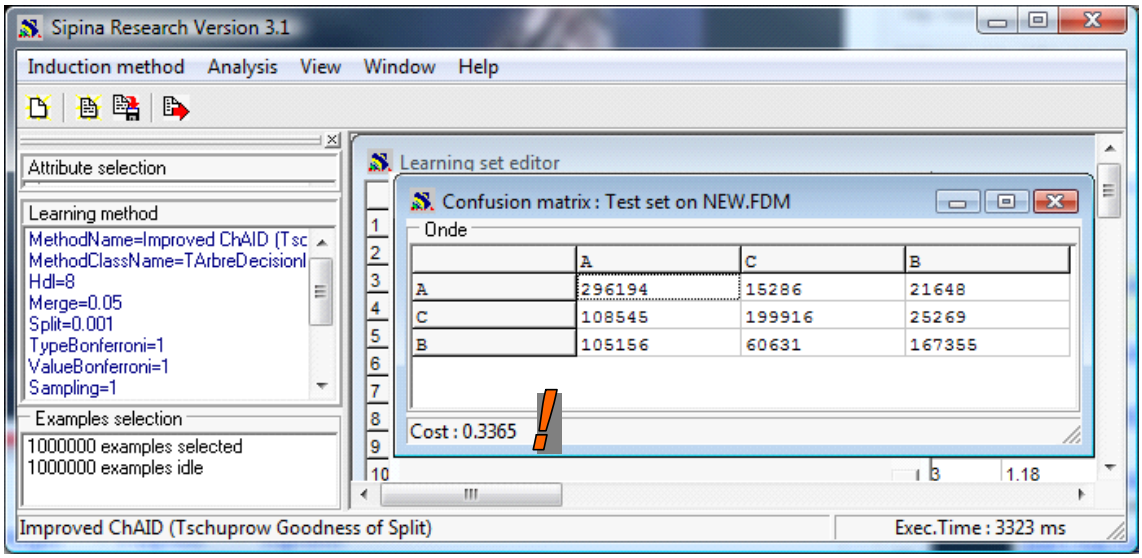
The obtained tree (Figure 4) is a little different than the previous one (Figure 1). The three first levels of the trees are identical. From the 4th level, there are some differences.

But what about their behaviors?

In order to check this, we apply the new tree on the test set[6]. We click on the ANALYSIS / TEST menu. We select the test set (INACTIVE EXAMPLES OF THE DATABASE).

We obtain a new confusion matrix. The test error rate is 34%. We observe that the local sampling approach preserves the classifier performance and reduces drastically the computation time. The approach is very attractive.

[6] Other approaches are possible. For instance, by comparing the prediction on the two trees on the test set; by checking if the examples in the same leaf on the first tree are in the same leaf on the second tree.



3.4 Decreasing the sample size

The main difficult is to determine the appropriate sample size. We use a very conservative default value (n = 5000) with Sipina. But, in the most of the databases, we can use lower values.

On the WAVEFORM dataset, we observe that about a hundred of instances are sufficient to perform the calculations. We varied the sample size (n = 100, 200, 400, 800, 1600, 3200, 6400). We measure both the test error rate (Figure 6) and the calculation time (Figure 5).

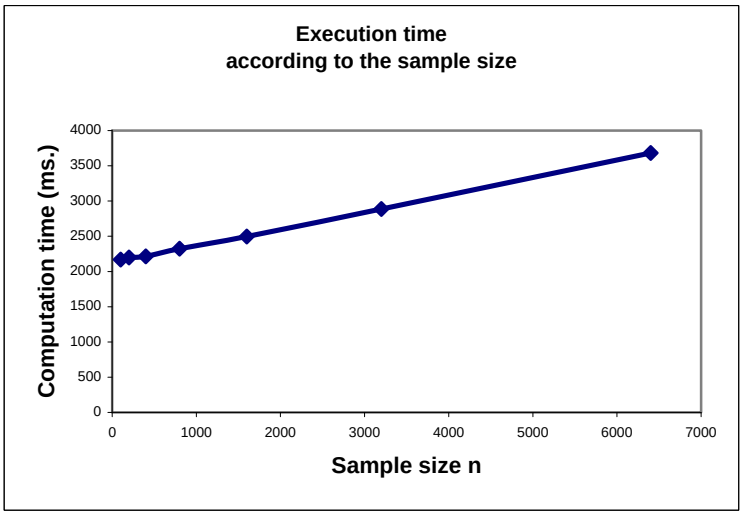


Figure 5 – Calculation time according to the sample size

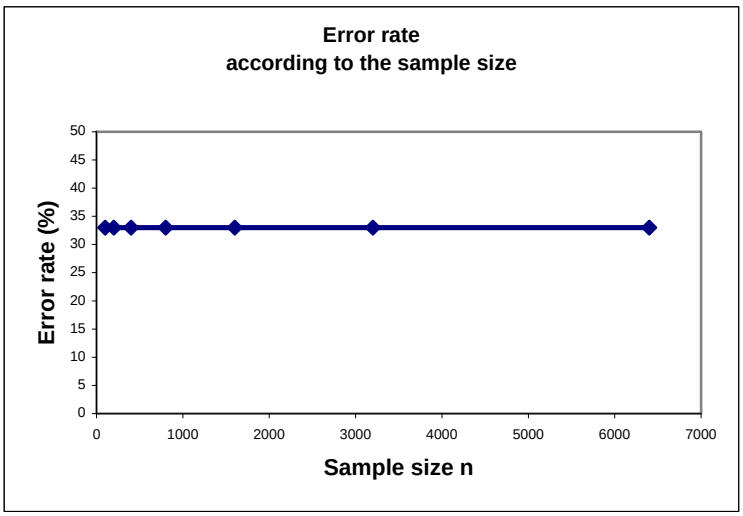


Figure 6 – Test error rate according to the sample size

As we can see, we obtain satisfactory results from n = 100 on this dataset.

We note here that the learning algorithm and its settings can influence also the appropriate sample size. In our paper (Chauchat and Rakotomalala, IFCS-2000), we use the C4.5 algorithm. We obtain deeper trees. In this context, the error rate becomes stable from n = 300.

Concerning the computation time, it increases linearly with the sample size.

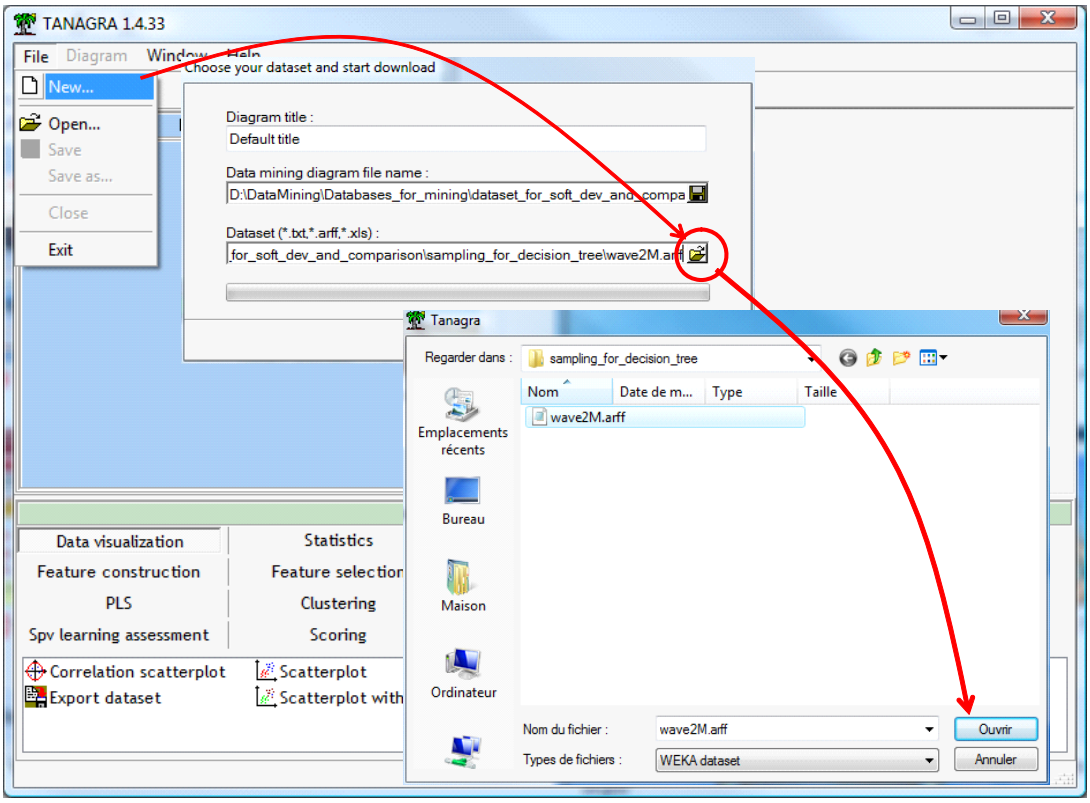
4 Analysis with TANAGRA

We choose another approach with Tanagra. The continuous descriptors are sorted before the decision tree learning process. We maintain in memory an index of the sorted examples for each continuous descriptor. For our dataset, we have 1,000,000 examples and 21 descriptors, wee need (21 x 4 x 1.000.000 # 80 MB) in memory.

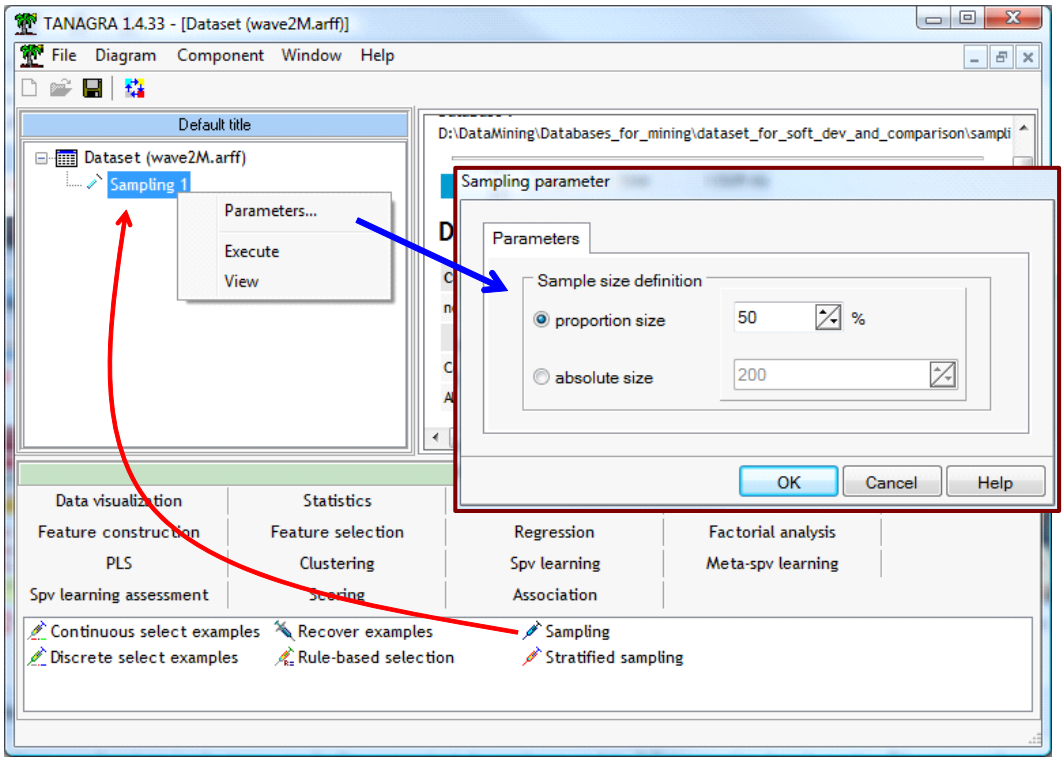
We choose this approach because the memory is abundant when I developed Tanagra (2004) in comparison with the time I had programmed Sipina (1998). The aim is to speed up the learning process by using all the available examples. It is suitable for the most of the dataset. But it can become a problem if we handle a very large database with a high number of continuous descriptors.

4.1 Diagram creation and data importation

After we launch Tanagra (**version 1.4.33 or later**), we create a diagram by clicking on the FILE / NEW menu. We select the WAVE2M.ARFF data file.

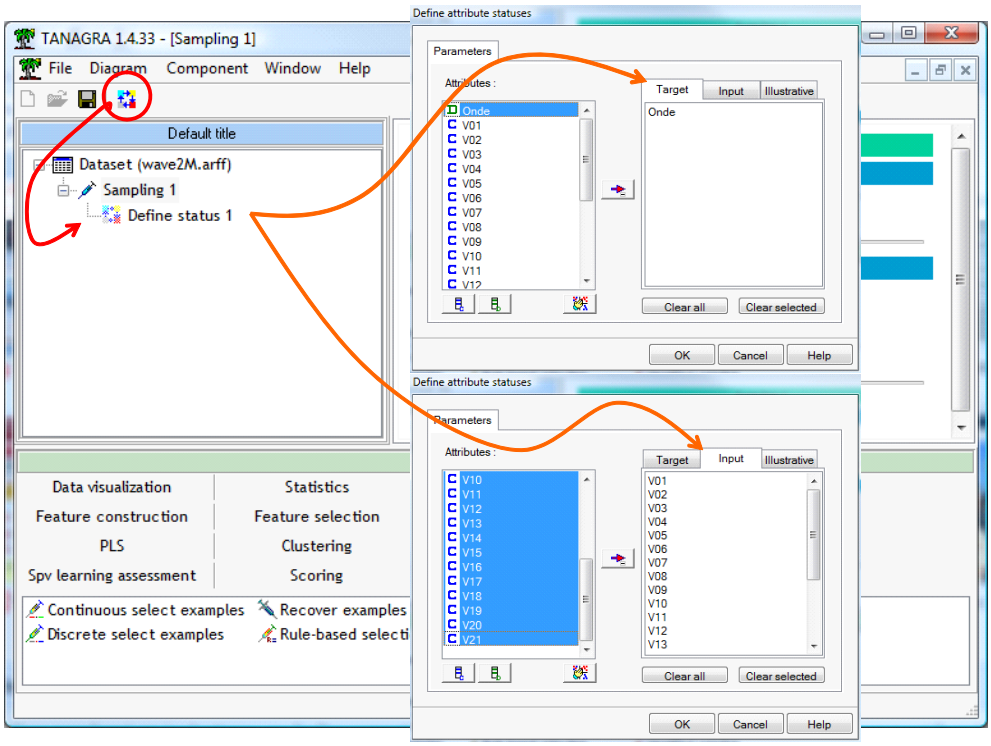


In the next step, we want to subdivide the database in a train and test samples. We use the SAMPLING (INSTANCE SELECTION tab) component. We open the dialog settings by clicking on the contextual PARAMETERS menu. We use the half of the database for the training phase.



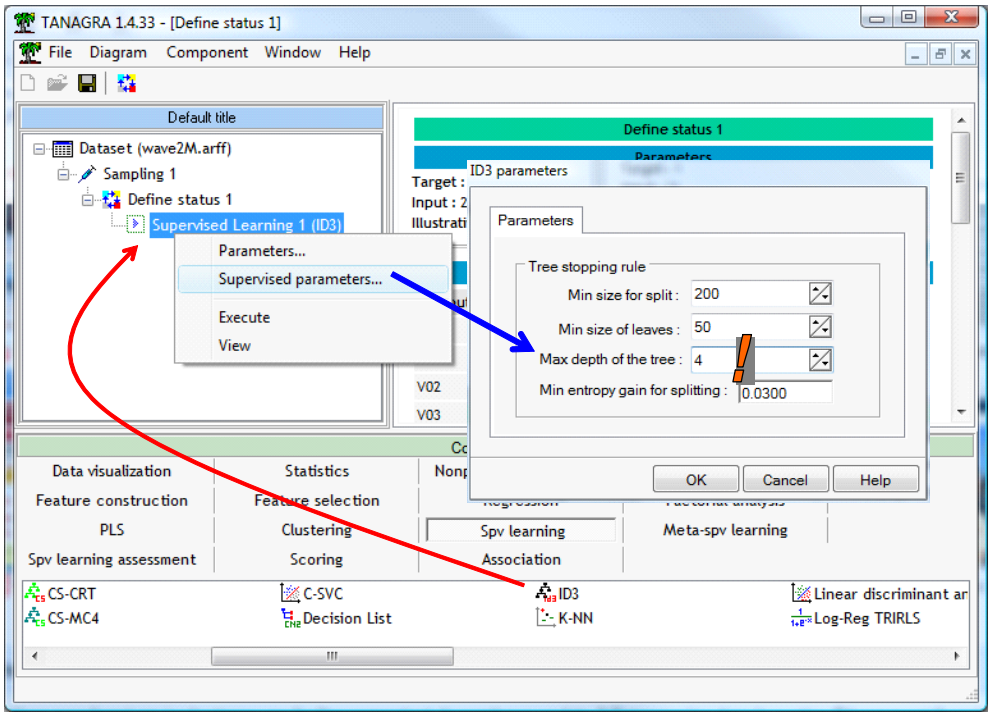
In order to define the status of the variables, we use the DEFINE STATUS component by using the

shortcut in the toolbar. We set ONDE as TARGET and the others as INPUT.

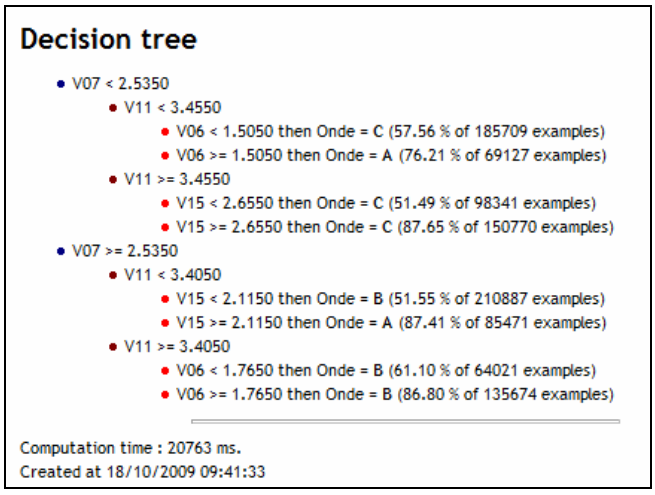


4.2 ID3 learning algorithm

We plan to use the ID3 (SPV LEARNING tab) algorithm because it is similar to the IMPROVED CHAID of SIPINA: we can limit the depth of the tree. We insert the component into the diagram. We click on the SUPERVISED PARAMETERS menu. We set MAX DEPTH OF THE TREE = 4.



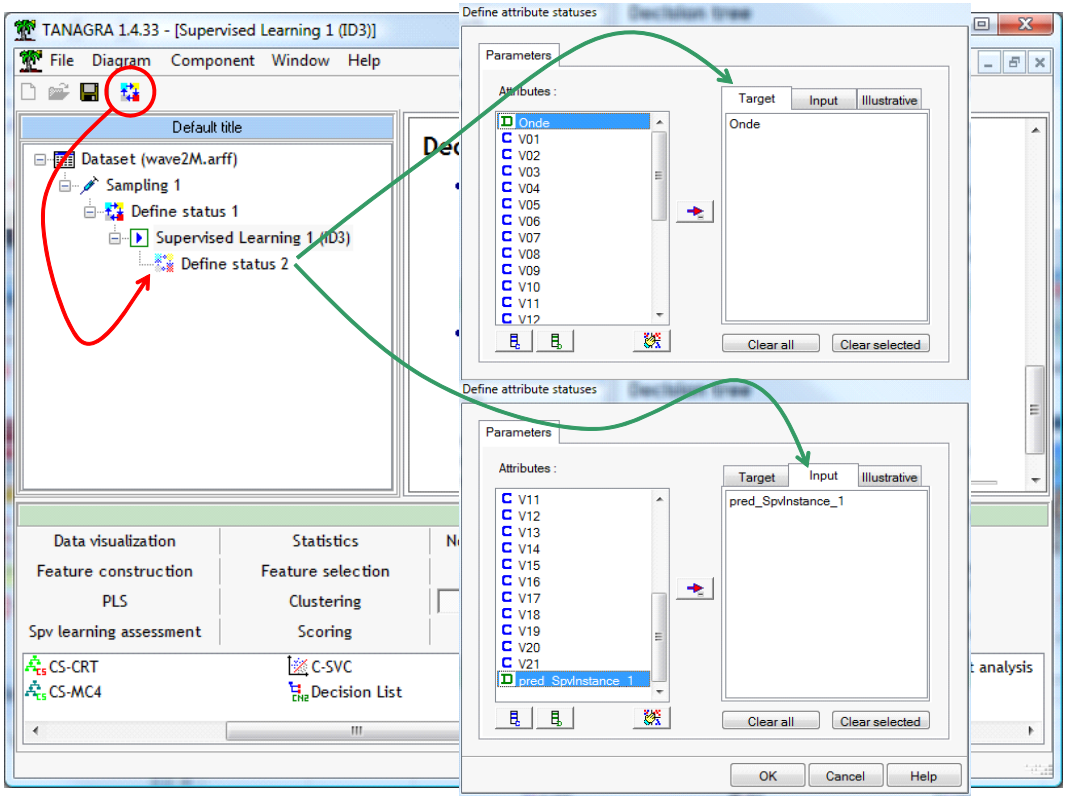
We validate the choice. We launch the learning process by clicking on the VIEW menu.



The execution time is 21 seconds. With the same settings, we obtain a tree with 4 levels (8 leaves); the preliminary computation of the sorted index enables to dramatically reduce the execution time when we use all the available examples on each node (93 sec. for Sipina, 21 sec. for Tanagra).

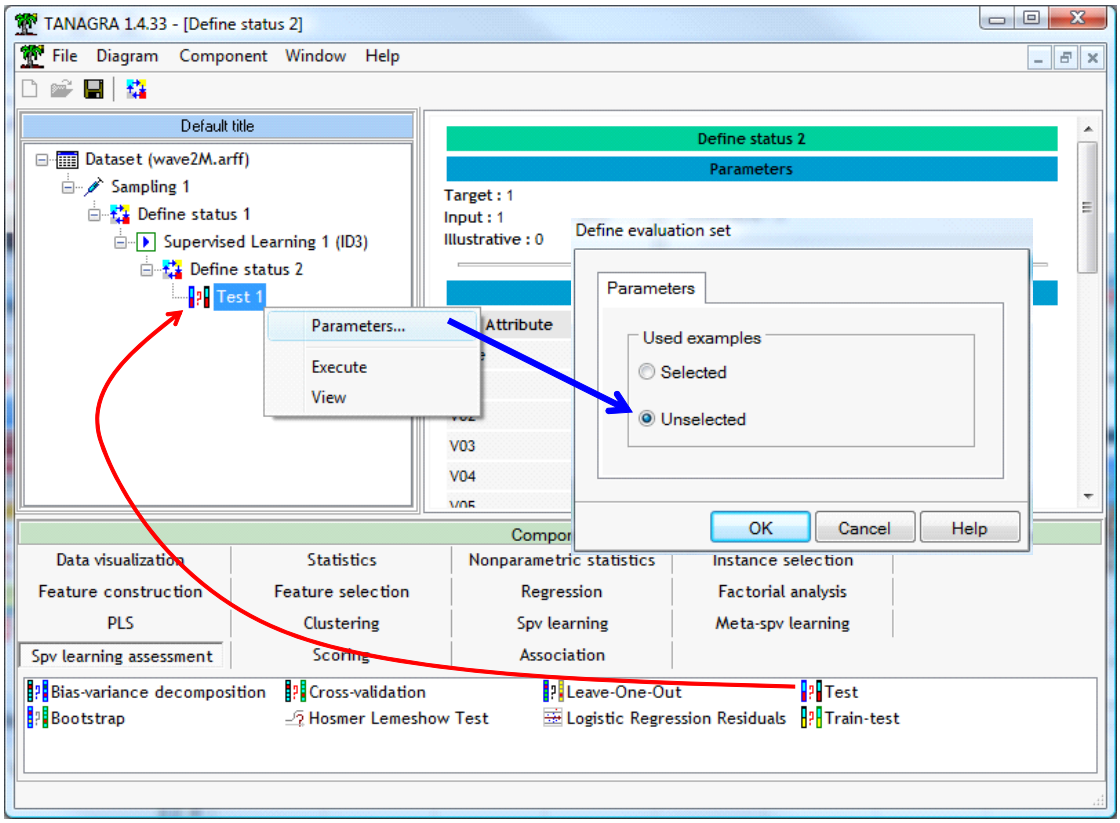
4.3 Test error rate

We insert again the DEFINE STATUS component for the evaluation of the tree on the test set. We set ONDE as TARGET, the predicted values supplied by the tree PRED_SPVINSTANCE_1 as INPUT.



Then, we insert the TEST component (SPV LEARNING ASSESSMENT tab) into the diagram. We click on

the PARAMETERS menu. We check that the confusion matrix is computed on the test sample (UNSELECTED instances).



We click on the VIEW menu. The test error rate is 32%. The values is similar to the one supplied by Sipina. We have not identical value because, even if SIPINA and TANAGRA use the same proportion, we have not the same instances in the train and test samples.

Test 1

Parameters

Evaluation set : unselected examples

Results

pred_SpvInstance_1

Error rate	0.3163

Values prediction

Value	Recall	1-Precision
A	0.3846	0.1744
C	0.8682	0.3332
B	0.7987	0.3522

Confusion matrix

	A	C	B	Sum
A	128316	88274	117061	333651
C	16489	289544	27467	333500
B	10609	56396	265844	332849
Sum	155414	434214	410372	1000000

Computation time : 280 ms.
Created at 18/10/2009 10:13:18

5 Conclusion

As we see in this tutorial, working on a sample on each node enables to decrease dramatically the execution time without a loss of accuracy. The generalization error rate of the resulting classifier is similar to those computed on the whole examples. The local sampling strategy can be very useful when we handle a very large database.

We summarize the main results in the following table:

	Execution time (sec.)	**Test error rate (%)**
Sipina without local sampling	93	34
Tanagra	21	32
Sipina with sampling (n = 5000)	3	34

The improvements of TANAGRA speed up the calculations compared to the traditional implementation into SIPINA (without sampling). When we use the local sampling strategy, we reduce even more strongly the execution time (30 times!). The generalization error rate is not modified.

Let's be honest. We are in a very favorable configuration in this tutorial. We create a small tree and all the descriptors are continuous. In the contrary, when we create a very large tree (with the C4.5 method for instance) or if most of the descriptors are discrete, the improvement is less spectacular. Of course, the user can evaluate the efficiency of this approach on its dataset.

Last, we are concerned about local sampling in this tutorial. Another approach is the global sampling. We draw a sample before the usual learning process. The determination of the right size of the sample remains the main problem in this context (see for instance the Quinlan's "Windowing" approach, 1993; or the John's "Dynamic sampling" approach, 1996).