1 Theme

Comparison of the implementation of the CART algorithm under Tanagra and R (rpart package).

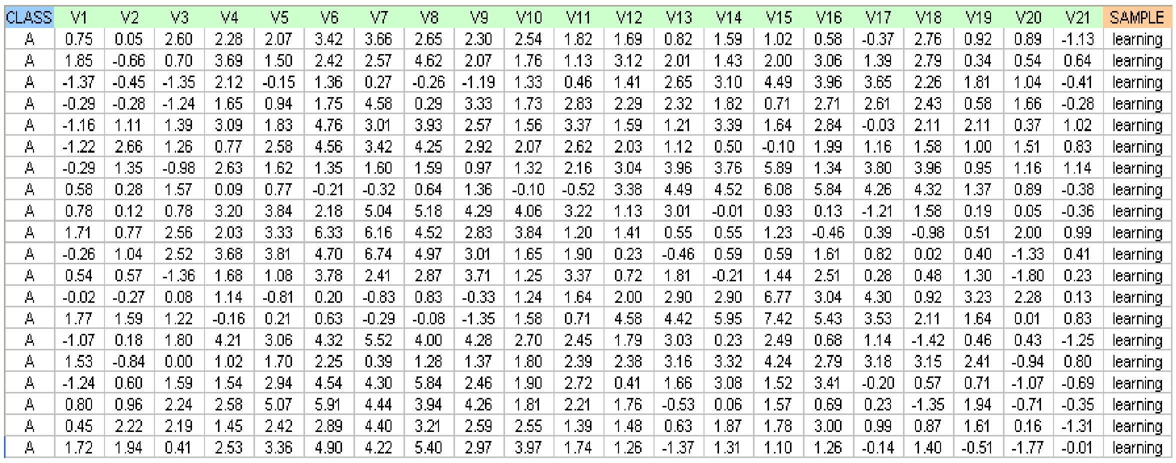
CART (Breiman and al., 1984) is a very popular classification tree (says also decision tree) learning algorithm. Rightly. CART incorporates all the ingredients of a good learning control: the post-pruning process enables to make the trade-off between the bias and the variance; the cost complexity mechanism enables to "smooth" the exploration of the space of solutions; we can control the preference for simplicity with the standard error rule (SE-rule); etc. Thus, the data miner can adjust the settings according to the goal of the study and the data characteristics.

The Breiman's algorithm is provided under different designations in the free data mining tools. Tanagra uses the "C-RT" name. R, through a specific package[1], provides the "rpart" function.

In this tutorial, we describe these implementations of the CART approach according to the original book (Breiman and al., 1984; chapters 3, 10 and 11). The main difference between them is the implementation of the post-pruning process. Tanagra uses a specific sample says "pruning set" (section 11.4); when rpart is based on the cross-validation principle (section 11.5)[2].

2 Dataset

We use the WAVEFORM dataset (section 2.6.2). The target attribute (CLASS) has 3 values. There are 21 continuous predictors (V1 to V21). We want to reproduce the experiment described into the Breiman's book (pages 49 and 50). We have 300 instances into the learning sample, and 5000 instances into the test sample. Thus the data file **wave5300.xls**[3] includes 5300 rows and 23 columns. The last column is a variable which specifies the membership of each instance to the train or the test samples (Figure 1).



CLASS	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	V15	V16	V17	V18	V19	V20	V21	SAMPLE
A	0.75	0.05	2.60	2.28	2.07	3.42	3.66	2.65	2.30	2.54	1.82	1.69	0.82	1.59	1.02	0.58	-0.37	2.76	0.92	0.89	-1.13	learning
A	1.85	-0.66	0.70	3.69	1.50	2.42	2.57	4.62	2.07	1.76	1.13	3.12	2.01	1.43	2.00	3.06	1.39	2.79	0.34	0.54	0.64	learning
A	-1.37	-0.45	-1.35	2.12	-0.15	1.36	0.27	-0.26	-1.19	1.33	0.46	1.41	2.65	3.10	4.49	3.96	3.65	2.26	1.81	1.04	-0.41	learning
A	-0.29	-0.28	-1.24	1.65	0.94	1.75	4.58	0.29	3.33	1.73	2.83	2.29	2.32	1.82	0.71	2.71	2.61	2.43	0.58	1.66	-0.28	learning
A	-1.16	1.11	1.39	3.09	1.83	4.76	3.01	3.93	2.57	1.56	3.37	1.59	1.21	3.39	1.64	2.84	-0.03	2.11	2.11	0.37	1.02	learning
A	-1.22	2.66	1.26	0.77	2.58	4.56	3.42	4.25	2.92	2.07	2.62	2.03	1.12	0.50	-0.10	1.99	1.16	1.58	1.00	1.51	0.83	learning
A	-0.29	1.35	-0.98	2.63	1.62	1.35	1.60	1.59	0.97	1.32	2.16	3.04	3.96	3.76	5.89	1.34	3.80	3.96	0.95	1.16	1.14	learning
A	0.58	0.28	1.57	0.09	0.77	-0.21	-0.32	0.64	1.36	-0.10	-0.52	3.38	4.49	4.52	6.08	5.84	4.26	4.32	1.37	0.89	-0.38	learning
A	0.78	0.12	0.78	3.20	3.84	2.18	5.04	5.18	4.29	4.06	3.22	1.13	3.01	-0.01	0.93	0.13	-1.21	1.58	0.19	0.05	-0.36	learning
A	1.71	0.77	2.56	2.03	3.33	6.33	6.16	4.52	2.83	3.84	1.20	1.41	0.55	0.55	1.23	-0.46	0.39	-0.98	0.51	2.00	0.99	learning
A	-0.26	1.04	2.52	3.68	3.81	4.70	6.74	4.97	3.01	1.65	1.90	0.23	-0.46	0.59	0.59	1.61	0.82	0.02	0.40	-1.33	0.41	learning
A	0.54	0.57	-1.36	1.68	1.08	3.78	2.41	2.87	3.71	1.25	3.37	0.72	1.81	-0.21	1.44	2.51	0.28	0.48	1.30	-1.80	0.23	learning
A	-0.02	-0.27	0.08	1.14	-0.81	0.20	-0.83	0.83	-0.33	1.24	1.64	2.00	2.90	2.90	6.77	3.04	4.30	0.92	3.23	2.28	0.13	learning
A	1.77	1.59	1.22	-0.16	0.21	0.63	-0.29	-0.08	-1.35	1.58	0.71	4.58	4.42	5.95	7.42	5.43	3.53	2.11	1.64	0.01	0.83	learning
A	-1.07	0.18	1.80	4.21	3.06	4.32	5.52	4.00	4.28	2.70	2.45	1.79	3.03	0.23	2.49	0.68	1.14	-1.42	0.46	0.43	-1.25	learning
A	1.53	-0.84	0.00	1.02	1.70	2.25	0.39	1.28	1.37	1.80	2.39	2.38	3.16	3.32	4.24	2.79	3.18	3.15	2.41	-0.94	0.80	learning
A	-1.24	0.60	1.59	1.54	2.94	4.54	4.30	5.84	2.46	1.90	2.72	0.41	1.66	3.08	1.52	3.41	-0.20	0.57	0.71	-1.07	-0.69	learning
A	0.80	0.96	2.24	2.58	5.07	5.91	4.44	3.94	4.26	1.81	2.21	1.76	-0.53	0.06	1.57	0.69	0.23	-1.35	1.94	-0.71	-0.35	learning
A	0.45	2.22	2.19	1.45	2.42	2.89	4.40	3.21	2.59	2.55	1.39	1.48	0.63	1.87	1.78	3.00	0.99	0.87	1.61	0.16	-1.31	learning
A	1.72	1.94	0.41	2.53	3.36	4.90	4.22	5.40	2.97	3.97	1.74	1.26	-1.37	1.31	1.10	1.26	-0.14	1.40	-0.51	-1.77	-0.01	learning

Figure 1 – The 20 first instances of the data file – wave5300.xls

[1] http://cran.r-project.org/web/packages/rpart/index.html

[2] We can use a pruning sample with "rpart" but it is not really easy to use. See http://www.math.univ-toulouse.fr/~besse/pub/TP/appr_se/tp7_cancer_tree_R.pdf; section 2.1.

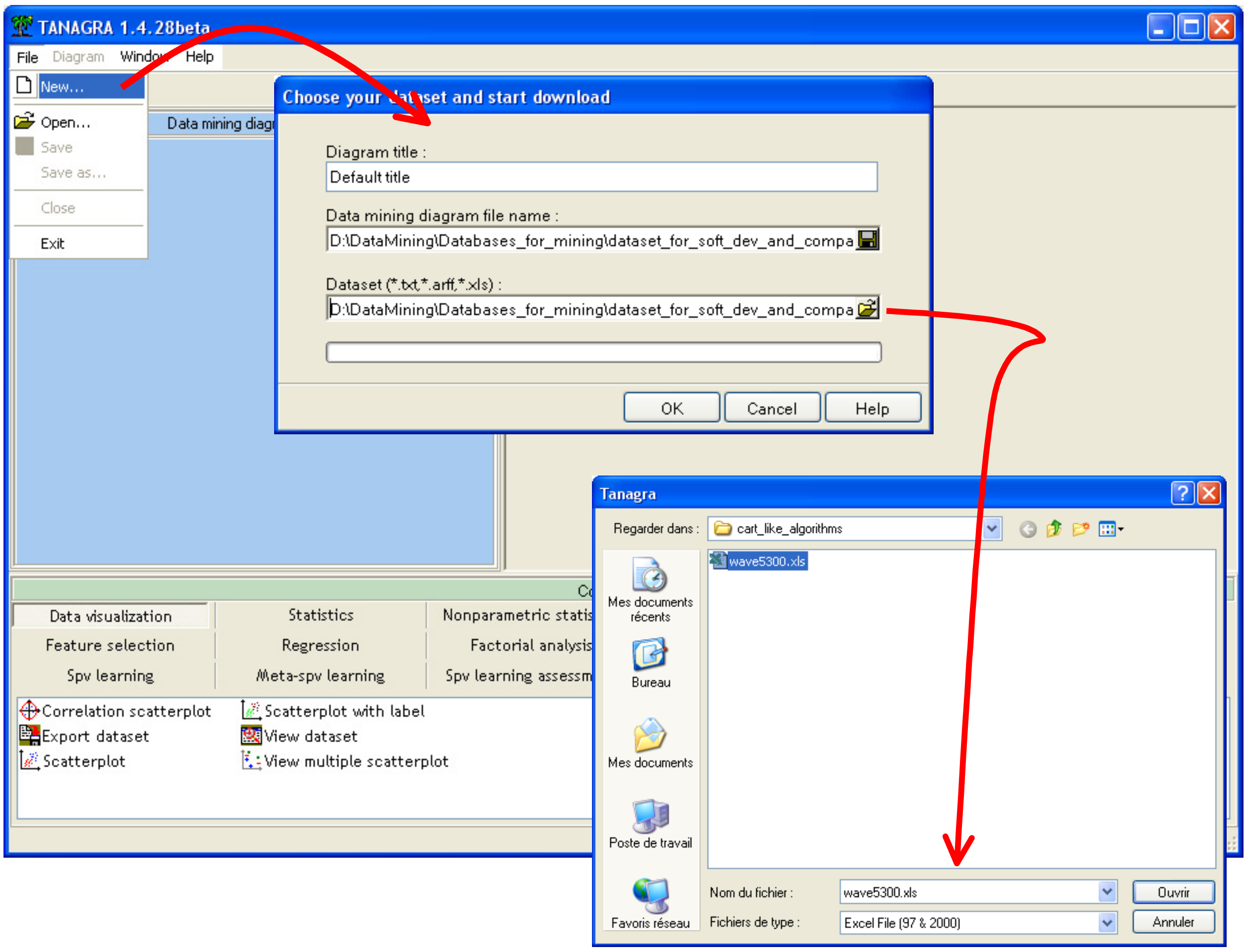
[3] http://eric.univ-lyon2.fr/~ricco/tanagra/fichiers/wave5300.xls

3 The C-RT component – TANAGRA

3.1 Creating a diagram and importing the data file

After we launch Tanagra, we want to create a diagram and import the data file.

For that, we activate the FILE / NEW menu. We choose the XLS file format and we select wave5300.xls[4].



Choose your dataset and start download

Diagram title : Default title

Data mining diagram file name : D:\DataMining\Databases_for_mining\dataset_for_soft_dev_and_compa

Dataset (*.txt,*.arff,*.xls) : D:\DataMining\Databases_for_mining\dataset_for_soft_dev_and_compa

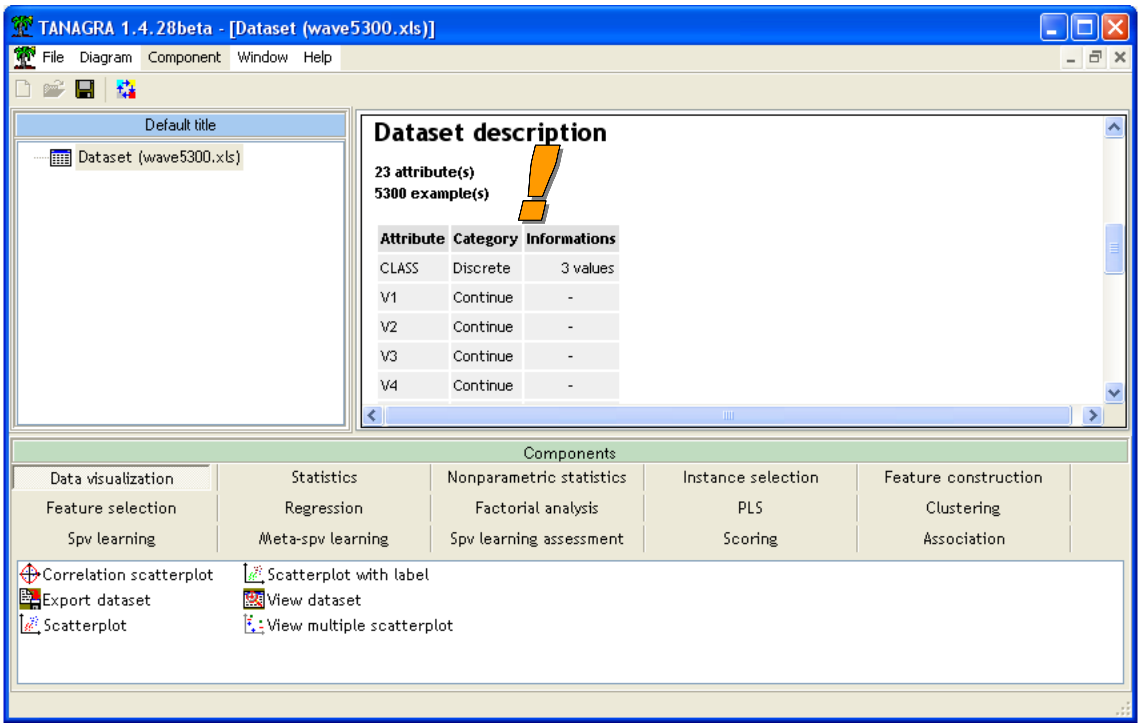
Nom du fichier : wave5300.xls

Fichiers de type : Excel File (97 & 2000)

The diagram is created. Into the first node, TANAGRA shows that the file contains 5300 instances and 23 variables.

[4] There are various ways to import a XLS data file. We can use the add-on for Excel (http://data-mining-tutorials.blogspot.com/2010/08/sipina-add-in-for-excel.html, http://data-mining-tutorials.blogspot.com/2010/08/tanagra-add-in-for-office-2007-and.html) or, as we do in this tutorial, directly import the dataset (http://data-mining-tutorials.blogspot.com/2008/10/excel-file-format-direct-importation.html). In this last case, the dataset must not be opened in the spreadsheet application. The values must be in the first sheet. The first row corresponds to the name of the variables.

The direct importation is faster than the use of the "tanagra.xla add-on. But, on the other hand, Tanagra can handle only the XLS format here (up to Excel 2003).



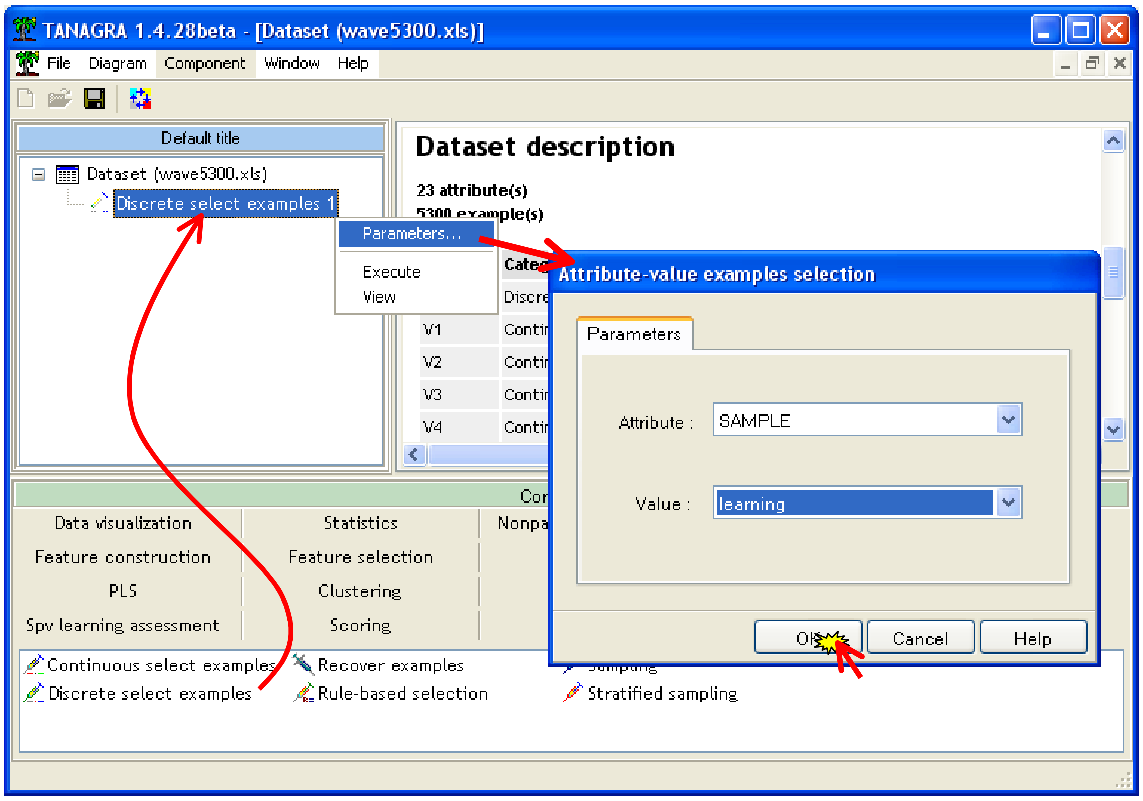
Dataset description

23 attribute(s)
5300 example(s)

Attribute	Category	Informations
CLASS	Discrete	3 values
V1	Continue	-
V2	Continue	-
V3	Continue	-
V4	Continue	-

3.2 Partitioning the dataset into train and test samples

We must define the training sample i.e. the instances used for the construction of the decision tree. We add the DISCRETE SELECT EXAMPLES (INSTANCE SELECTION tab) into the diagram. We click on the PARAMETERS contextual menu. We use the SAMPLE column for the splitting. The learning set corresponds to the instances labeled LEARNING.



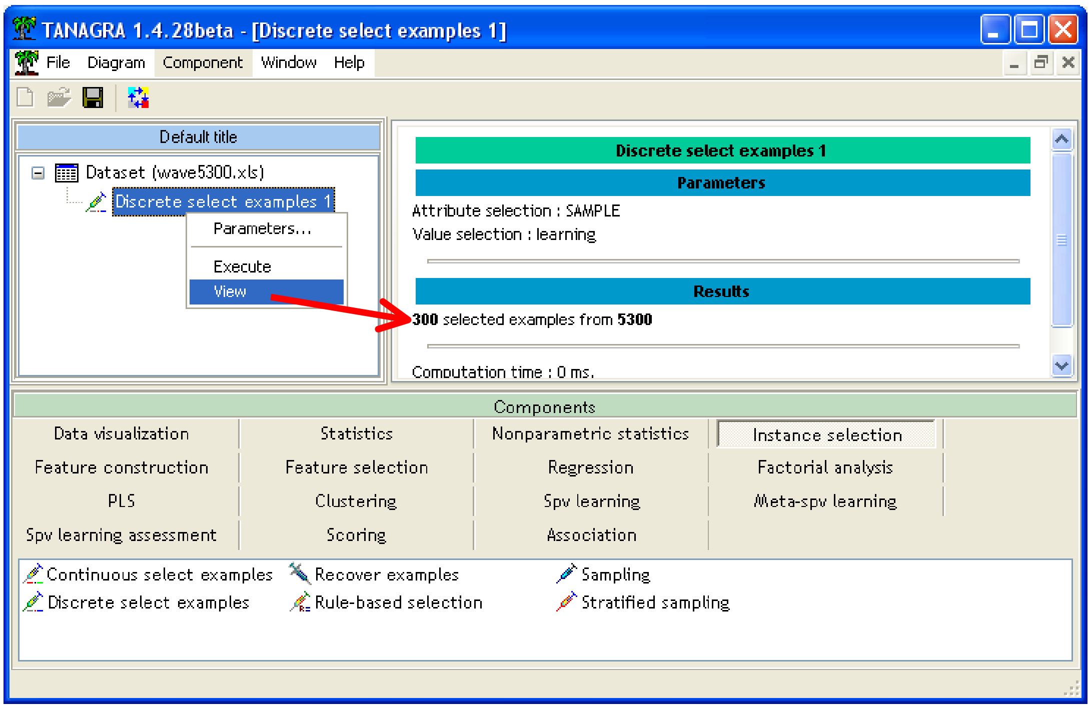
Dataset description

23 attribute(s)
5300 example(s)

Attribute : SAMPLE

Value : learning

We click on the VIEW menu: 300 instances (from 5300) are now used for the construction of the decision tree.



Discrete select examples 1

Parameters

Attribute selection : SAMPLE
Value selection : learning

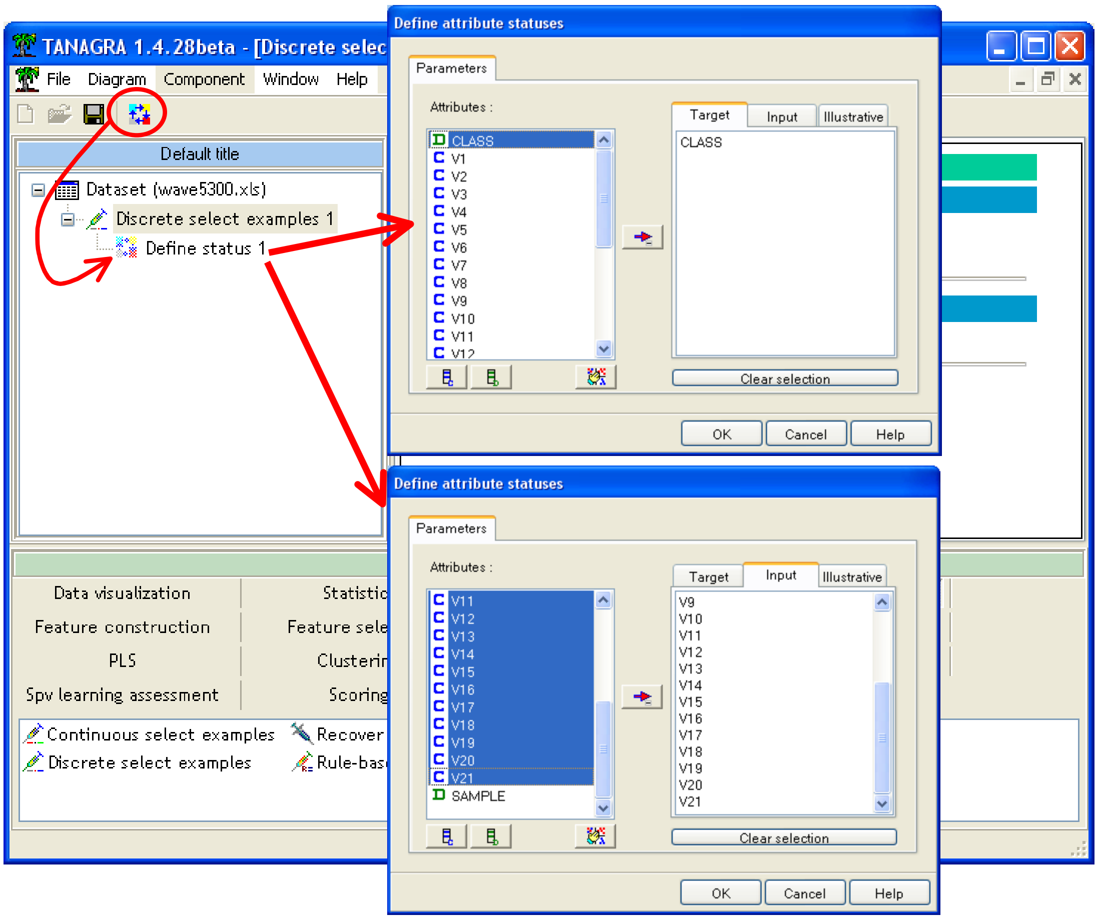
Results

300 selected examples from **5300**

Computation time : 0 ms.

3.3 Decision tree induction

Before the strictly speaking learning process, we must specify the variable types. We add the DEFINE STATUS component using the shortcut into the toolbar. We set CLASS as TARGET, the continuous variables (V1 to V21) as INPUT.



Define attribute statuses — Target: CLASS
Attributes: CLASS, V1, V2, V3, V4, V5, V6, V7, V8, V9, V10, V11, V12

Define attribute statuses — Input: V9, V10, V11, V12, V13, V14, V15, V16, V17, V18, V19, V20, V21
Attributes: V11, V12, V13, V14, V15, V16, V17, V18, V19, V20, V21, SAMPLE

We insert the C-RT component which implements the CART approach. Let us describe some of the method settings (SUPERVISED PARAMETERS menu).

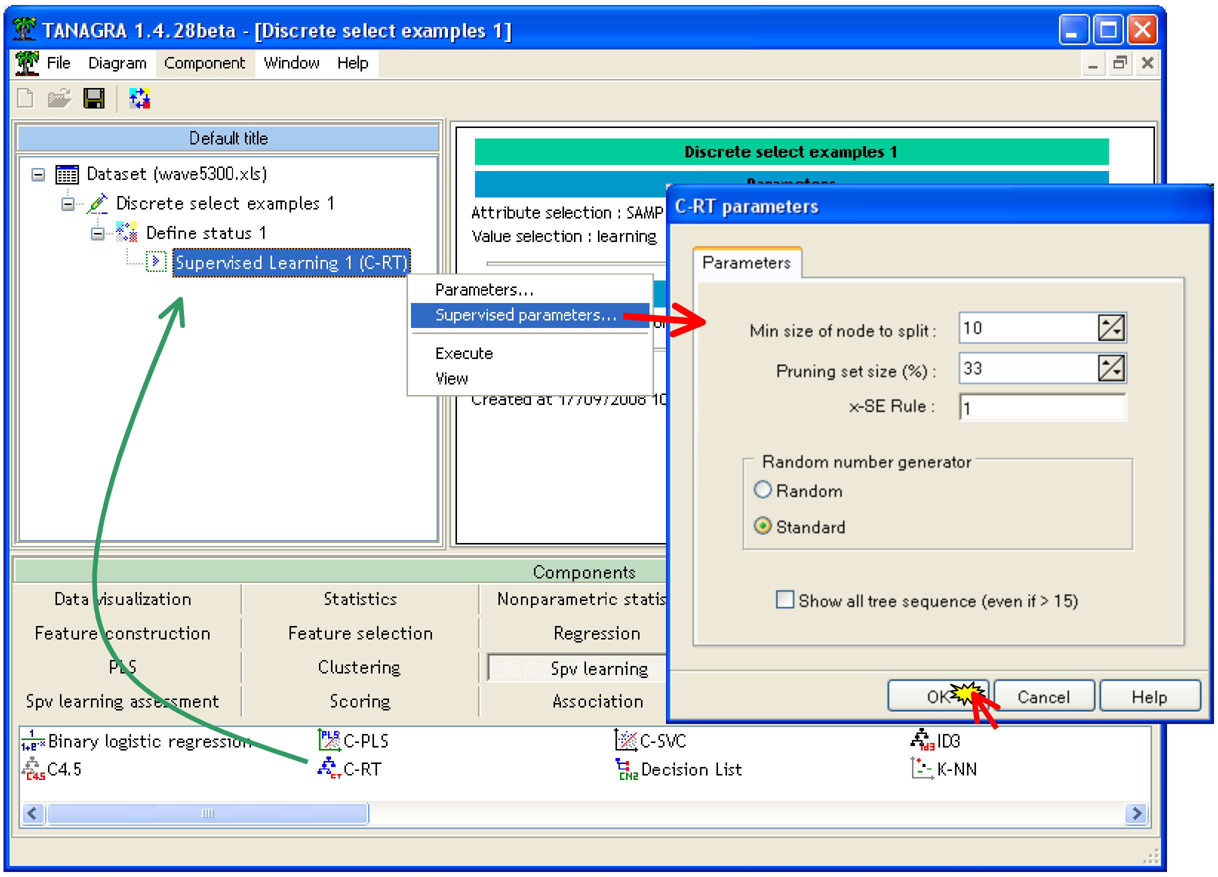
MIN SIZE OF NODE TO SPLIT means the minimum number of instances needed for performing a splitting of a node. For our dataset, we do not perform a split if there are less than **10** instances.

PRUNING SET SIZE means the part of the learning set used for the post pruning process. The default value is 33% i.e. 67% of the learning set is used for the growing phase (67% of 300 = 201 instances), 33% for the pruning phase (33% of 300 = 99 instances).

SE RULE allows to select the right tree in the post-pruning process. The default value is θ=**1** i.e. we select the smallest tree for which the error rate is lower than the "error rate + 1 standard error (of the error rate)" of the optimal tree. If θ=0, we select the optimal tree, the one which minimizes the pruning error rate (the error rate computed on the pruning sample).

RANDOM NUMBER GENERATOR allows to initialize the random number generator.

Last, **SHOW ALL TREE SEQUENCE**, when it is checked off, allows to show all the sequences of trees analyzed during the post-pruning process. It is not necessary to activate this setting here.



Attribute selection : SAMP
Value selection : learning

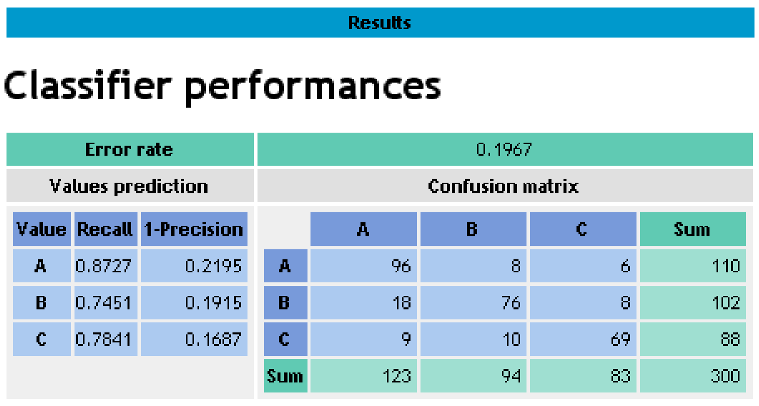
Min size of node to split : 10
Pruning set size (%) : 33
x-SE Rule : 1

Random number generator
Random [✓ Standard]

☐ Show all tree sequence (even if > 15)

We validate our settings by clicking on the OK button. Then, we click on the contextual VIEW menu.

We obtain first the resubstitution confusion matrix, computed on the learning set (300 instances, the growing and the pruning samples are merged). The error rate is 19.67%.



Results

Classifier performances

Error rate	0.1967
Values prediction	Confusion matrix

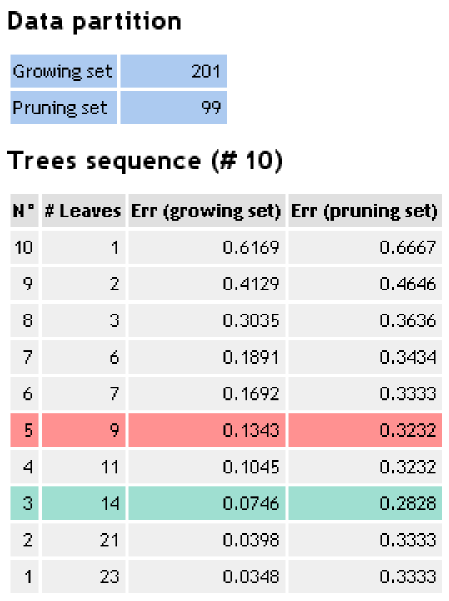
Value	Recall	1-Precision
A	0.8727	0.2195
B	0.7451	0.1915
C	0.7841	0.1687

	A	B	C	Sum
A	96	8	6	110
B	18	76	8	102
C	9	10	69	88
Sum	123	94	83	300

Below, we have the sequences of trees table, with the number of leaves, the error rate calculated on the growing set and pruning set.

The error rate computed on the growing set decreases as the number of leaves increases. We cannot use this information to select the right model.

We use the pruning sample to select the "best" model. The tree which minimizes the error rate on the pruning set is highlighted in green. It has 14 leaves. Its error rate is 28.28%.



Data partition

Growing set	201
Pruning set	99

Trees sequence (# 10)

N°	# Leaves	Err (growing set)	Err (pruning set)
10	1	0.6169	0.6667
9	2	0.4129	0.4646
8	3	0.3035	0.3636
7	6	0.1891	0.3434
6	7	0.1692	0.3333
5	9	0.1343	0.3232
4	11	0.1045	0.3232
3	14	0.0746	0.2828
2	21	0.0398	0.3333
1	23	0.0348	0.3333

Figure 2 - Sequences of trees for the model selection

With the θ = 1 SE-rule, we select the smallest tree (with the smallest number of leaves) for which the pruning error rate is lower than

$$\varepsilon_{seuil} = 0.2828 + 1 \times \sqrt{\frac{0.2828 \times (1 - 0.2828)}{99}} = 0.3281$$

This is the tree n° 5, with 9 leaves. The pruning error rate is 32.32% (highlighted in red). The growing error rate is 13.43%.

The chart depicting the change of the error rate computed on the growing and the pruning samples is available in the CHART tab of the visualization window. We can visually detect the interesting trees.

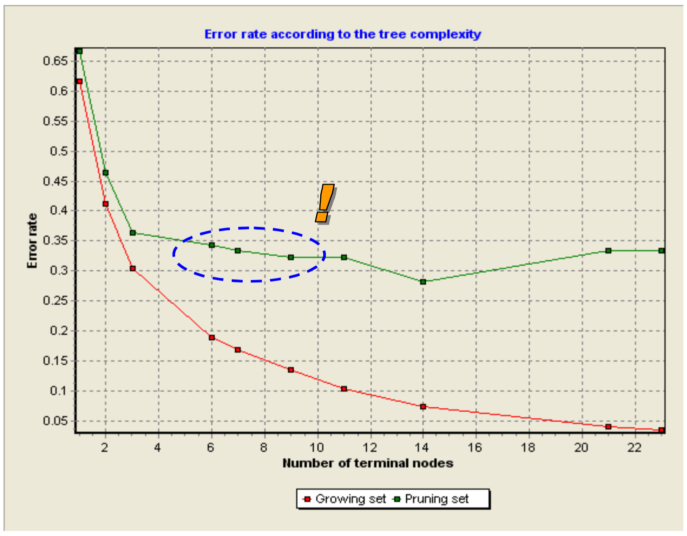
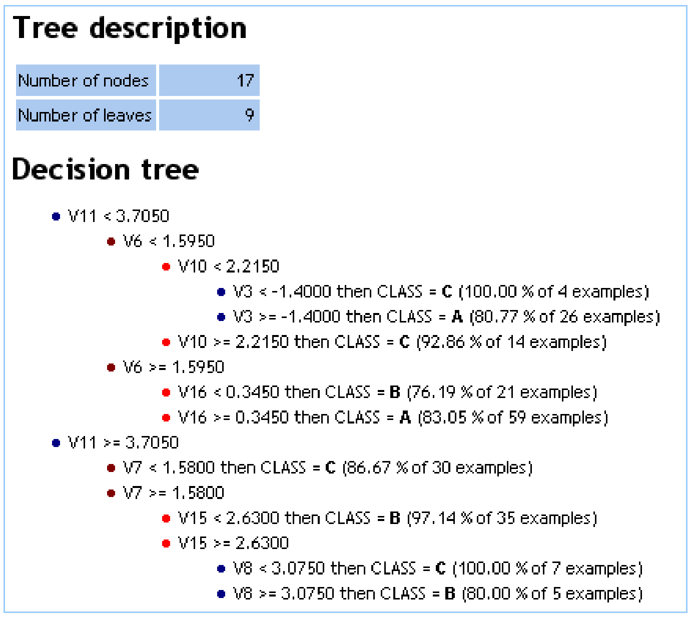


Figure 3 - The error rate according the number of leaves

We note that the trees with 7 or 6 leaves are equivalent in terms of prediction performance. The way to modify the setting θ to obtain one of these trees is described in another tutorial[5].

In the lower part of the report (HTML tab), we have a description of the selected tree.



Tree description

Number of nodes	17
Number of leaves	9

Decision tree

- V11 < 3.7050
  - V6 < 1.5950
    - V10 < 2.2150
      - V3 < -1.4000 then CLASS = **C** (100.00 % of 4 examples)
      - V3 >= -1.4000 then CLASS = **A** (80.77 % of 26 examples)
    - V10 >= 2.2150 then CLASS = **C** (92.86 % of 14 examples)
  - V6 >= 1.5950
    - V16 < 0.3450 then CLASS = **B** (76.19 % of 21 examples)
    - V16 >= 0.3450 then CLASS = **A** (83.05 % of 59 examples)
- V11 >= 3.7050
  - V7 < 1.5800 then CLASS = **C** (86.67 % of 30 examples)
  - V7 >= 1.5800
    - V15 < 2.6300 then CLASS = **B** (97.14 % of 35 examples)
    - V15 >= 2.6300
      - V8 < 3.0750 then CLASS = **C** (100.00 % of 7 examples)
      - V8 >= 3.0750 then CLASS = **B** (80.00 % of 5 examples)

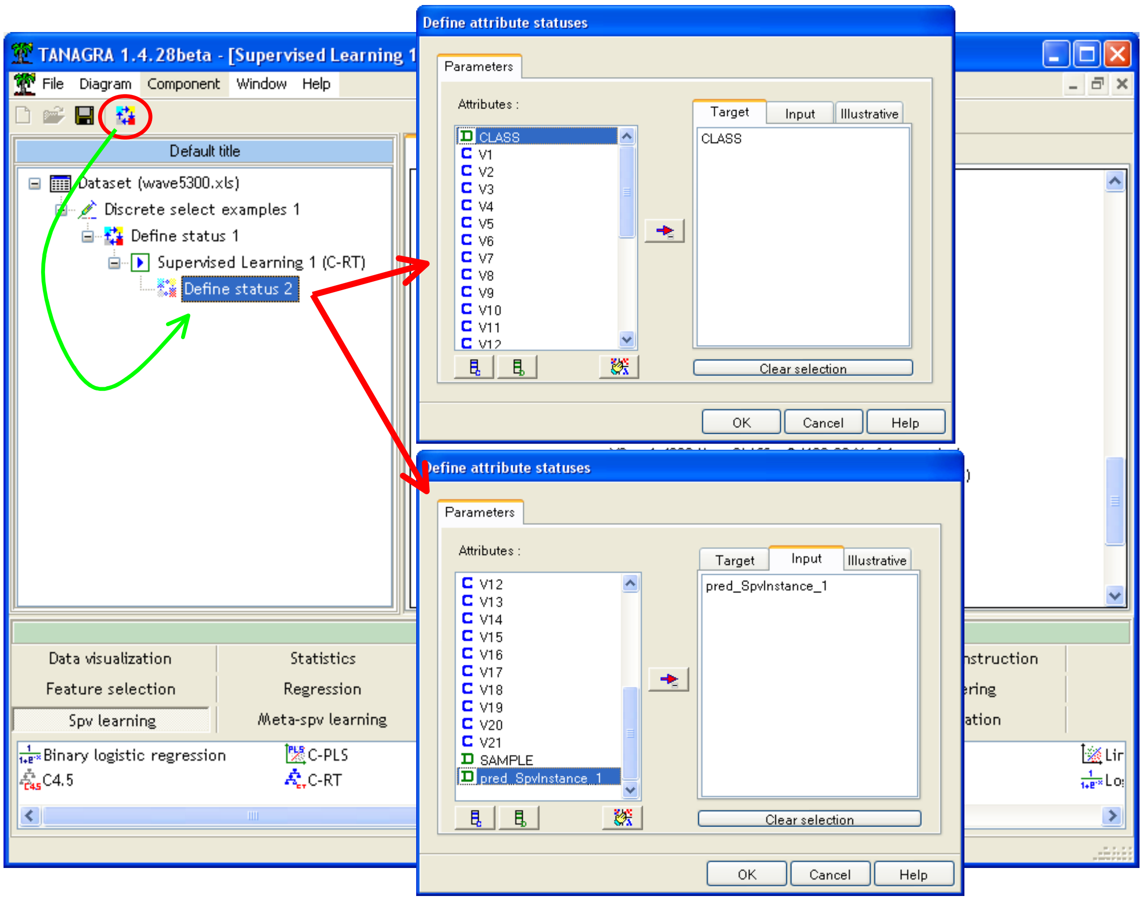
3.4 Assessment on the test set

To obtain a good estimation of the generalization error rate, we use a separated sample, the test set, unused during the learning phase. On our dataset, the test set is the rest of our data file i.e. 5000 instances set aside in the previous part of the diagram (DISCRETE SELECT EXAMPLES node).

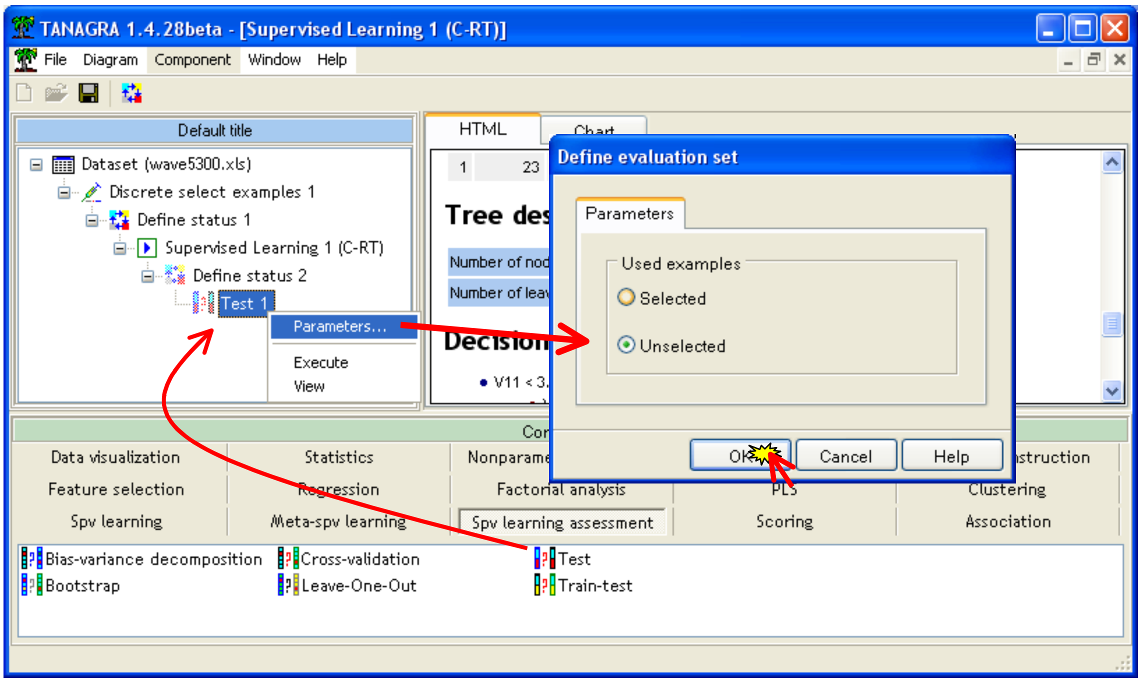
Tanagra generates automatically a new column which contains the prediction of the model. The trick is that the prediction is computed both on the selected instances (the learning sample) and the unselected ones (the test sample).

[5] http://data-mining-tutorials.blogspot.com/2010/01/cart-determining-right-size-of-tree.html

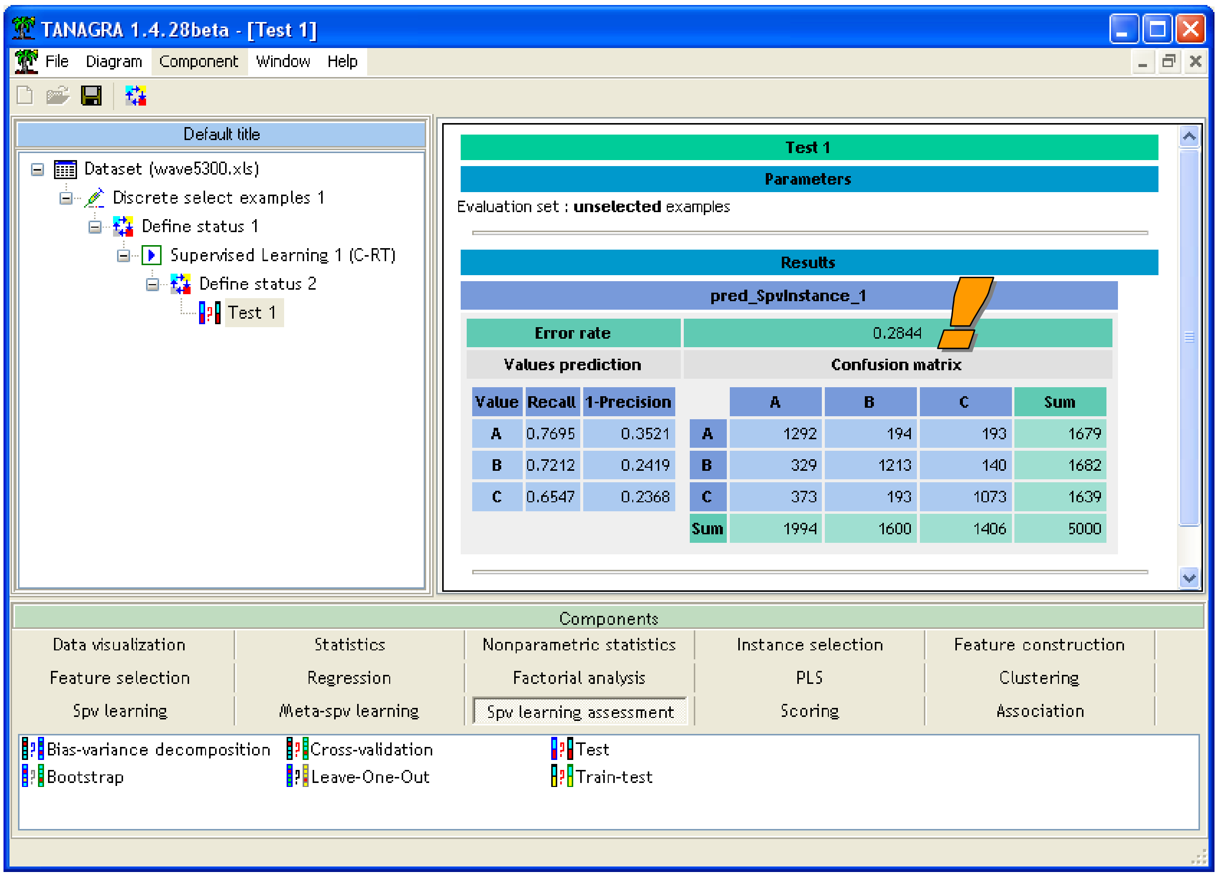
We add the DEFINE STATUS into the diagram. We set CLASS as TARGET and PRED_SPVINSTANCE_1, the new column generated by the supervised learning algorithm, as INPUT.


TANAGRA 1.4.28beta - [Supervised Learning 1
File Diagram Component Window Help
Default title
Dataset (wave5300.xls)
Discrete select examples 1
Define status 1
Supervised Learning 1 (C-RT)
Define status 2
Define attribute statuses
Parameters
Attributes :
Target Input Illustrative
CLASS
Clear selection
OK Cancel Help
pred_SpvInstance_1
SAMPLE
Data visualization Statistics
Feature selection Regression
Spv learning Meta-spv learning
Binary logistic regression C-PLS
C4.5 C-RT

We add the TEST component (SPV LEARNING ASSESSMENT tab). We click on the VIEW menu. We observe that, by default, the confusion matrix is computed on the unselected instances i.e. the test sample (Note: The confusion matrix can be computed on the selected instances i.e. the learning sample. In this case, we must obtain the same confusion matrix as the one showed into the model visualization window.).


TANAGRA 1.4.28beta - [Supervised Learning 1 (C-RT)]
File Diagram Component Window Help
Default title
Dataset (wave5300.xls)
Discrete select examples 1
Define status 1
Supervised Learning 1 (C-RT)
Define status 2
Test 1
Parameters...
Execute
View
Define evaluation set
Parameters
Used examples
Selected
Unselected
OK Cancel Help
Data visualization Statistics Nonparametric
Feature selection Regression Factorial analysis PLS Clustering
Spv learning Meta-spv learning Spv learning assessment Scoring Association
Bias-variance decomposition Cross-validation Test
Bootstrap Leave-One-Out Train-test

We click on the VIEW menu. The test error rate is 28.44%. When we use the tree for the prediction on an unseen instance, the probability of misclassification is 28.44%.



Dataset (wave5300.xls)
- Discrete select examples 1
  - Define status 1
    - Supervised Learning 1 (C-RT)
      - Define status 2
        - Test 1

Test 1

Parameters

Evaluation set : **unselected** examples

Results

pred_SpvInstance_1

Error rate	0.2844

Values prediction

Value	Recall	1-Precision
A	0.7695	0.3521
B	0.7212	0.2419
C	0.6547	0.2368

Confusion matrix

	A	B	C	Sum
A	1292	194	193	1679
B	329	1213	140	1682
C	373	193	1073	1639
Sum	1994	1600	1406	5000

4 The CART approach using the RPART package of R

We suppose that the reader is familiar with the R software. If don't, we can found many tutorials on line, on the official R website for instance: http://cran.r-project.org/manuals.html and http://cran.r-project.org/other-docs.html; see also http://www.r-tutor.com/, etc… The documents are very numerous on the internet.

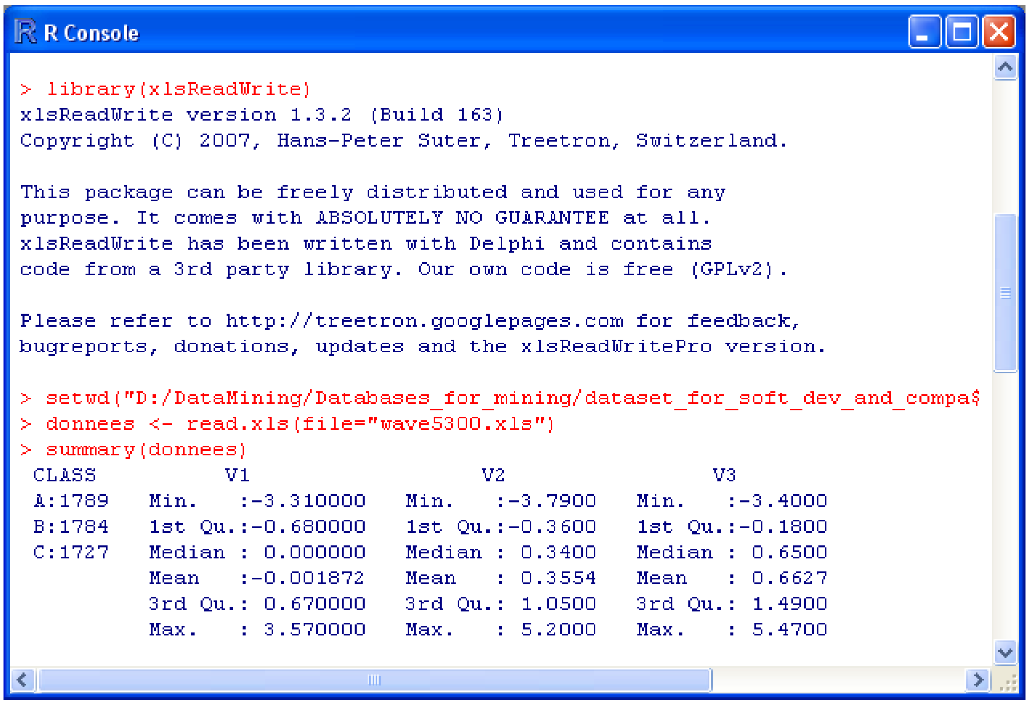
4.1 Importing the data file using the xlsReadWrite package

To import the XLS file format, the easiest way is to use the xlsReadWrite package.

We load it using the **library(.)** instruction.

Then, we can import the dataset using the **read.xls(.)** command. We obtain a short description of the data frame (the structure which manages the dataset into memory) with **summary(.)**[6].

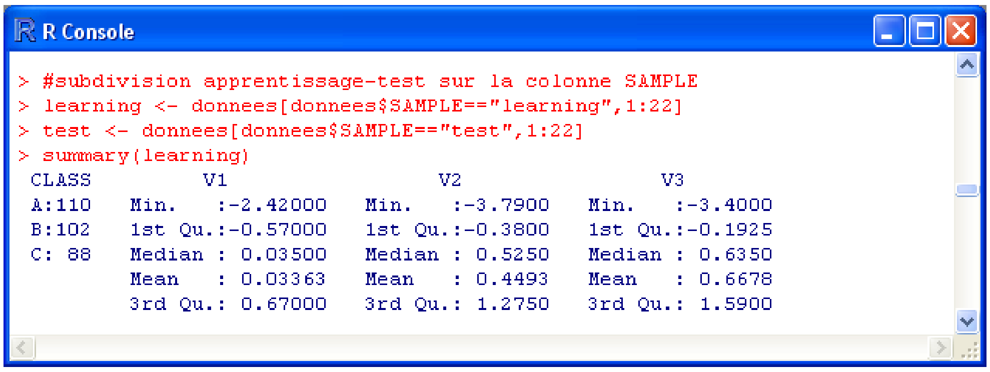
[6] The full results of the command summary (.) are not shown in our screenshots. It would take too much space. It is nevertheless essential at every stage to check the data integrity.



```
> library(xlsReadWrite)
xlsReadWrite version 1.3.2 (Build 163)
Copyright (C) 2007, Hans-Peter Suter, Treetron, Switzerland.

This package can be freely distributed and used for any
purpose. It comes with ABSOLUTELY NO GUARANTEE at all.
xlsReadWrite has been written with Delphi and contains
code from a 3rd party library. Our own code is free (GPLv2).

Please refer to http://treetron.googlepages.com for feedback,
bugreports, donations, updates and the xlsReadWritePro version.

> setwd("D:/DataMining/Databases_for_mining/dataset_for_soft_dev_and_compa$
> donnees <- read.xls(file="wave5300.xls")
> summary(donnees)
 CLASS          V1                   V2                  V3
 A:1789   Min.   :-3.310000   Min.   :-3.7900   Min.   :-3.4000
 B:1784   1st Qu.:-0.680000   1st Qu.:-0.3600   1st Qu.:-0.1800
 C:1727   Median : 0.000000   Median : 0.3400   Median : 0.6500
          Mean   :-0.001872   Mean   : 0.3554   Mean   : 0.6627
          3rd Qu.: 0.670000   3rd Qu.: 1.0500   3rd Qu.: 1.4900
          Max.   : 3.570000   Max.   : 5.2000   Max.   : 5.4700
```

4.2 Partitioning the data file into train and test samples

We use the SAMPLE column to partition the dataset ("learning" and "test"). The SAMPLE column is then removed.



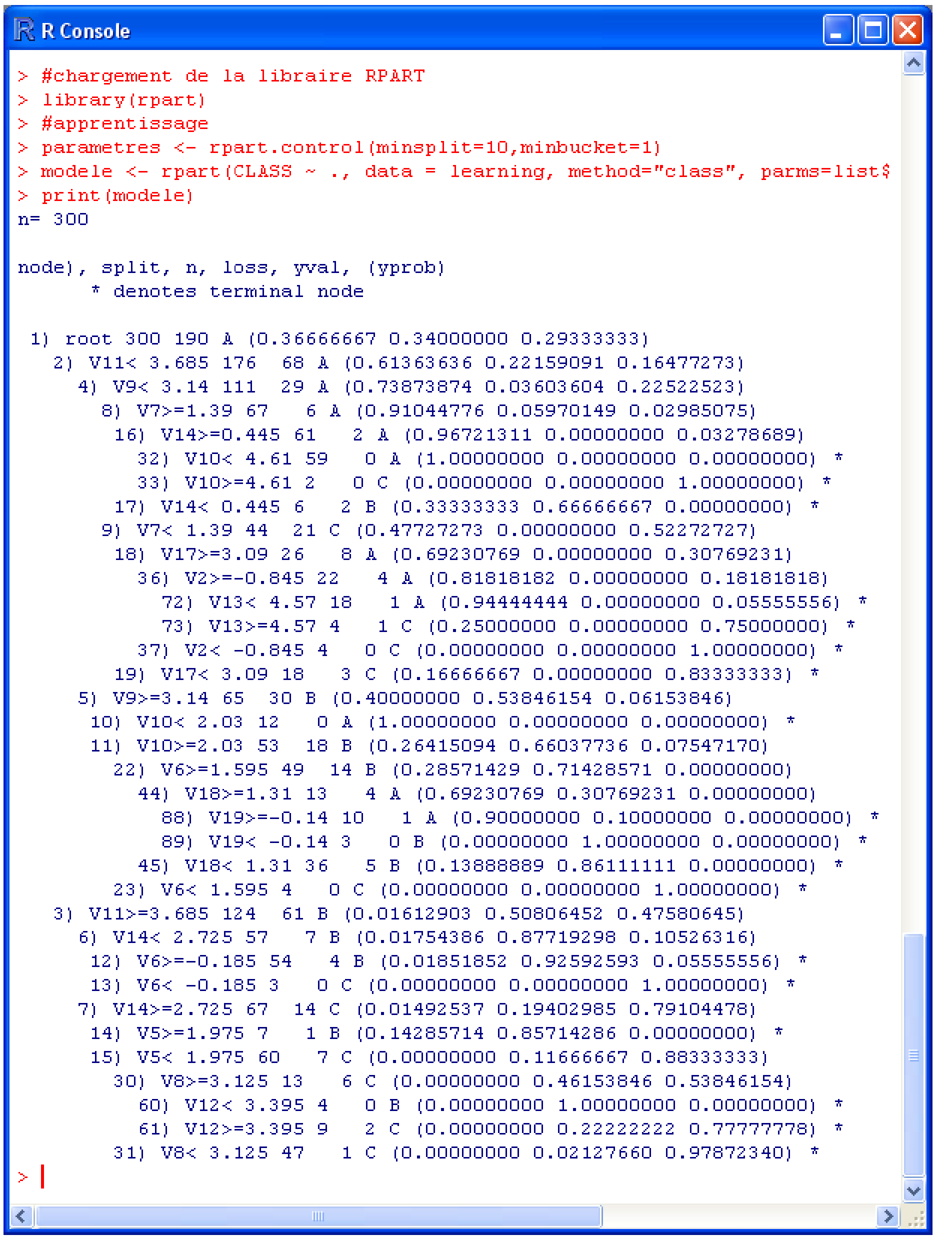
```
> #subdivision apprentissage-test sur la colonne SAMPLE
> learning <- donnees[donnees$SAMPLE=="learning",1:22]
> test <- donnees[donnees$SAMPLE=="test",1:22]
> summary(learning)
 CLASS         V1                  V2                 V3
 A:110   Min.   :-2.42000   Min.   :-3.7900   Min.   :-3.4000
 B:102   1st Qu.:-0.57000   1st Qu.:-0.3800   1st Qu.:-0.1925
 C: 88   Median : 0.03500   Median : 0.5250   Median : 0.6350
         Mean   : 0.03363   Mean   : 0.4493   Mean   : 0.6678
         3rd Qu.: 0.67000   3rd Qu.: 1.2750   3rd Qu.: 1.5900
```

4.3 Creating the maximal tree

First, we must develop the maximal tree on the learning set i.e. the tree with the maximum number of leaves. It will be pruned in a second phase.

We use the following settings[7]: MINSPLIT = 10, it means that the process does not split a node with less than 10 instances; MINBUCKET = 1, the process does not validate a split where one the leaves contains less than 1 instance. The maximal tree obtained should be similar to the one of Tanagra. One of the main differences is that RPART uses all the instances of the learning set (Tanagra uses only the growing sample, a part of the learning set).

[7] About the other settings: METHOD = "CLASS" indicates that we are dealing with a classification problem (instead of a regression); SPLIT = GINI indicates the measure used for the selection of splitting variables.



```
> #chargement de la libraire RPART
> library(rpart)
> #apprentissage
> parametres <- rpart.control(minsplit=10,minbucket=1)
> modele <- rpart(CLASS ~ ., data = learning, method="class", parms=list$
> print(modele)
n= 300

node), split, n, loss, yval, (yprob)
      * denotes terminal node

 1) root 300 190 A (0.36666667 0.34000000 0.29333333)
   2) V11< 3.685 176  68 A (0.61363636 0.22159091 0.16477273)
     4) V9< 3.14 111  29 A (0.73873874 0.03603604 0.22522523)
       8) V7>=1.39 67   6 A (0.91044776 0.05970149 0.02985075)
        16) V14>=0.445 61   2 A (0.96721311 0.00000000 0.03278689)
          32) V10< 4.61 59   0 A (1.00000000 0.00000000 0.00000000) *
          33) V10>=4.61 2   0 C (0.00000000 0.00000000 1.00000000) *
        17) V14< 0.445 6   2 B (0.33333333 0.66666667 0.00000000) *
       9) V7< 1.39 44  21 C (0.47727273 0.00000000 0.52272727)
        18) V17>=3.09 26   8 A (0.69230769 0.00000000 0.30769231)
          36) V2>=-0.845 22   4 A (0.81818182 0.00000000 0.18181818)
            72) V13< 4.57 18   1 A (0.94444444 0.00000000 0.05555556) *
            73) V13>=4.57 4   1 C (0.25000000 0.00000000 0.75000000) *
          37) V2< -0.845 4   0 C (0.00000000 0.00000000 1.00000000) *
        19) V17< 3.09 18   3 C (0.16666667 0.00000000 0.83333333) *
     5) V9>=3.14 65  30 B (0.40000000 0.53846154 0.06153846)
      10) V10< 2.03 12   0 A (1.00000000 0.00000000 0.00000000) *
      11) V10>=2.03 53  18 B (0.26415094 0.66037736 0.07547170)
        22) V6>=1.595 49  14 B (0.28571429 0.71428571 0.00000000)
          44) V18>=1.31 13   4 A (0.69230769 0.30769231 0.00000000)
            88) V19>=-0.14 10   1 A (0.90000000 0.10000000 0.00000000) *
            89) V19< -0.14 3   0 B (0.00000000 1.00000000 0.00000000) *
          45) V18< 1.31 36   5 B (0.13888889 0.86111111 0.00000000) *
        23) V6< 1.595 4   0 C (0.00000000 0.00000000 1.00000000) *
   3) V11>=3.685 124  61 B (0.01612903 0.50806452 0.47580645)
     6) V14< 2.725 57   7 B (0.01754386 0.87719298 0.10526316)
      12) V6>=-0.185 54   4 B (0.01851852 0.92592593 0.05555556) *
      13) V6< -0.185 3   0 C (0.00000000 0.00000000 1.00000000) *
     7) V14>=2.725 67  14 C (0.01492537 0.19402985 0.79104478)
      14) V5>=1.975 7   1 B (0.14285714 0.85714286 0.00000000) *
      15) V5< 1.975 60   7 C (0.00000000 0.11666667 0.88333333)
        30) V8>=3.125 13   6 C (0.00000000 0.46153846 0.53846154)
          60) V12< 3.395 4   0 B (0.00000000 1.00000000 0.00000000) *
          61) V12>=3.395 9   2 C (0.00000000 0.22222222 0.77777778) *
        31) V8< 3.125 47   1 C (0.00000000 0.02127660 0.97872340) *
> 
```

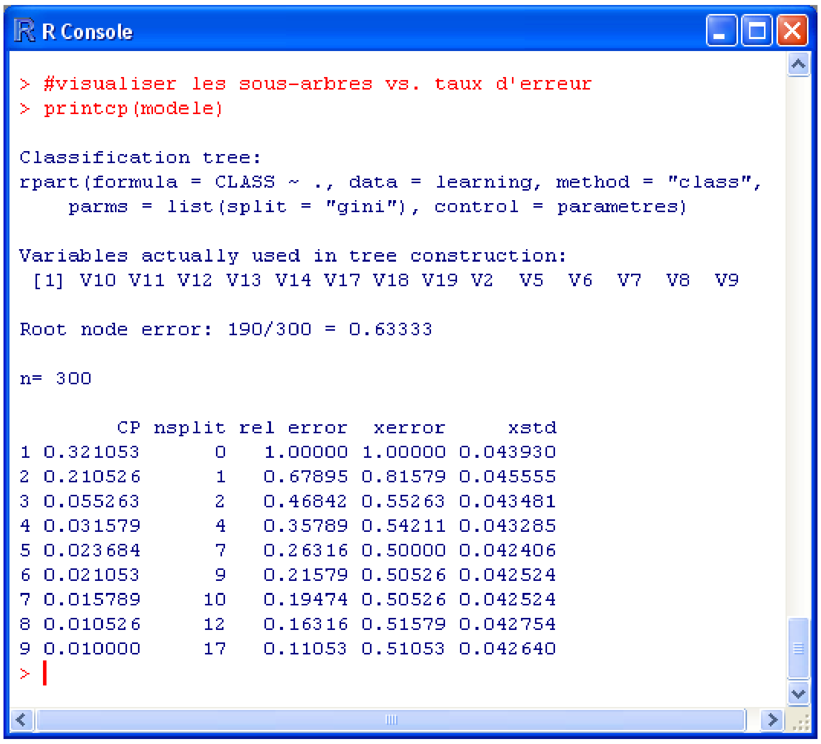
We obtain a tree with 18 leaves. This is less than those of Tanagra. One of reason is the CP parameter of which the default value (cp=0.01) introduces a pre-pruning during the growing phase. We see below that the same CP parameter is used for the post-pruning procedure.

4.4 Selecting the right tree – Post-pruning process with RPART

One of the main drawbacks of the rpart is that we must perform ourselves the post-pruning procedure[8]. For that, we use the results showed into the CPTABLE obtained with the **printcp(.)** command[9] (Figure 4).

[8] We see below that this can be an advantage in another point of view.

[9] Because rpart partitions randomly the dataset during the cross-validation process, it may be that we do not have exactly the same results. To obtain the same results at each session, we can define the seed of the random number generator using the **set.seed(.)** command.



```
> #visualiser les sous-arbres vs. taux d'erreur
> printcp(modele)

Classification tree:
rpart(formula = CLASS ~ ., data = learning, method = "class",
    parms = list(split = "gini"), control = parametres)

Variables actually used in tree construction:
 [1] V10 V11 V12 V13 V14 V17 V18 V19 V2  V5  V6  V7  V8  V9

Root node error: 190/300 = 0.63333

n= 300

        CP nsplit rel error  xerror     xstd
1 0.321053      0   1.00000 1.00000 0.043930
2 0.210526      1   0.67895 0.81579 0.045555
3 0.055263      2   0.46842 0.55263 0.043481
4 0.031579      4   0.35789 0.54211 0.043285
5 0.023684      7   0.26316 0.50000 0.042406
6 0.021053      9   0.21579 0.50526 0.042524
7 0.015789     10   0.19474 0.50526 0.042524
8 0.010526     12   0.16316 0.51579 0.042754
9 0.010000     17   0.11053 0.51053 0.042640
> 
```

Figure 4 – The CPTABLE generated by RPART

It describes the sequence of trees by associating the complexity parameter (CP) with: the number of splitting (equal to the number of leaves - 1 since we have a binary tree), the error calculated on the training sample (normalized so that the error on the root is equal to 1), the error calculated by cross-validation (also normalized), the standard error of the error rate.

This table is very similar to the one generated by Tanagra (Figure 2). But instead of using a separated part of the learning set (the pruning sample), rpart is based on the cross-validation principle.

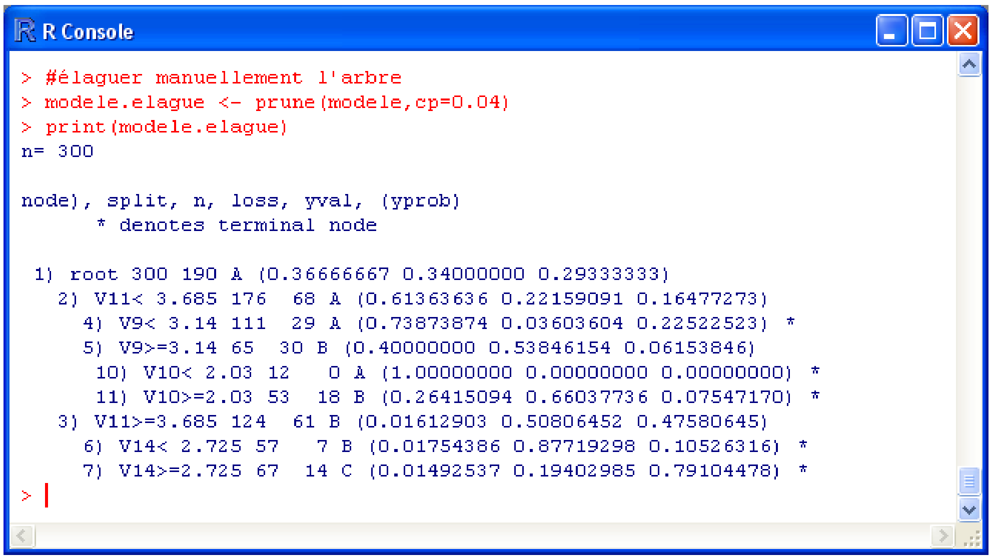
On the one hand, it is advantageous when we handle a small learning sample, it is not necessary to partition this last one. But, on the other hand, the computation time is dramatically increased when we handle a large dataset.

Selection of the final tree. We want to use the values provided by the CPTABLE to select the "optimal" tree. The tree n°5 with 7 splitting (8 leaves) is the one which minimizes the cross-validation error rate (ε = 0.50000). To obtain this tree, we must specify a CP value included between (>) 0.023684 and (≤) 0.031579.

But this tree is not the best one. Breiman claims that it is more interesting to introduce a preference to simplicity by selecting the tree according to the 1-SE rule. (Breiman and al., 1984; section 3.4.3, pages 78 to 81). The threshold error rate is computed as follows from the values provided by the CPTABLE

$$\varepsilon_{seuil} = 0.5 + 1 \times 0.042406 = 0.542406$$

The smallest tree of which the cross-validation error rate is lower than this threshold is the n°4 with 4 splitting (5 leaves). To obtain this tree, we set CP between the range (0.031579 < cp ≤ 0.055263). Let's say cp = 0.04 for instance. We use the **prunecp(.)** command

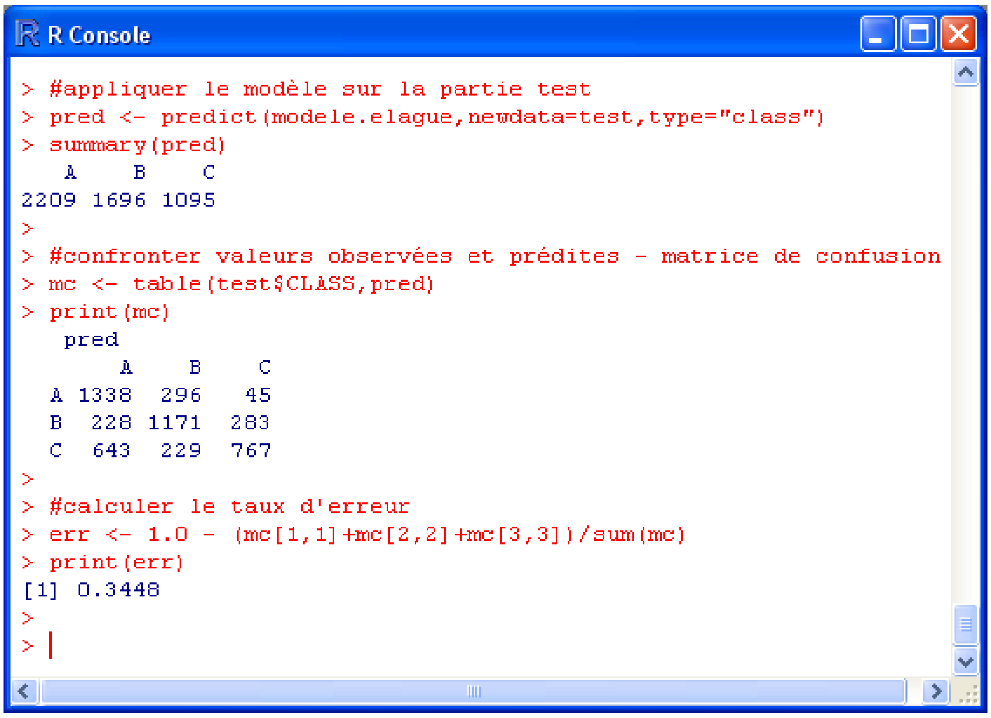


```
> #élaguer manuellement l'arbre
> modele.elague <- prune(modele,cp=0.04)
> print(modele.elague)
n= 300

node), split, n, loss, yval, (yprob)
      * denotes terminal node

 1) root 300 190 A (0.36666667 0.34000000 0.29333333)
   2) V11< 3.685 176  68 A (0.61363636 0.22159091 0.16477273)
     4) V9< 3.14 111  29 A (0.73873874 0.03603604 0.22522523) *
     5) V9>=3.14 65  30 B (0.40000000 0.53846154 0.06153846)
      10) V10< 2.03 12   0 A (1.00000000 0.00000000 0.00000000) *
      11) V10>=2.03 53  18 B (0.26415094 0.66037736 0.07547170) *
   3) V11>=3.685 124  61 B (0.01612903 0.50806452 0.47580645)
     6) V14< 2.725 57   7 B (0.01754386 0.87719298 0.10526316) *
     7) V14>=2.725 67  14 C (0.01492537 0.19402985 0.79104478) *
> 
```

Now, we must evaluate this tree (5 leaves) on the test sample.

4.5 Assessment on the test sample

We create the prediction column using the **predict(.)** command. Of course, the prediction is computed on the "test" sample. Then, we compute the confusion matrix and the error rate.

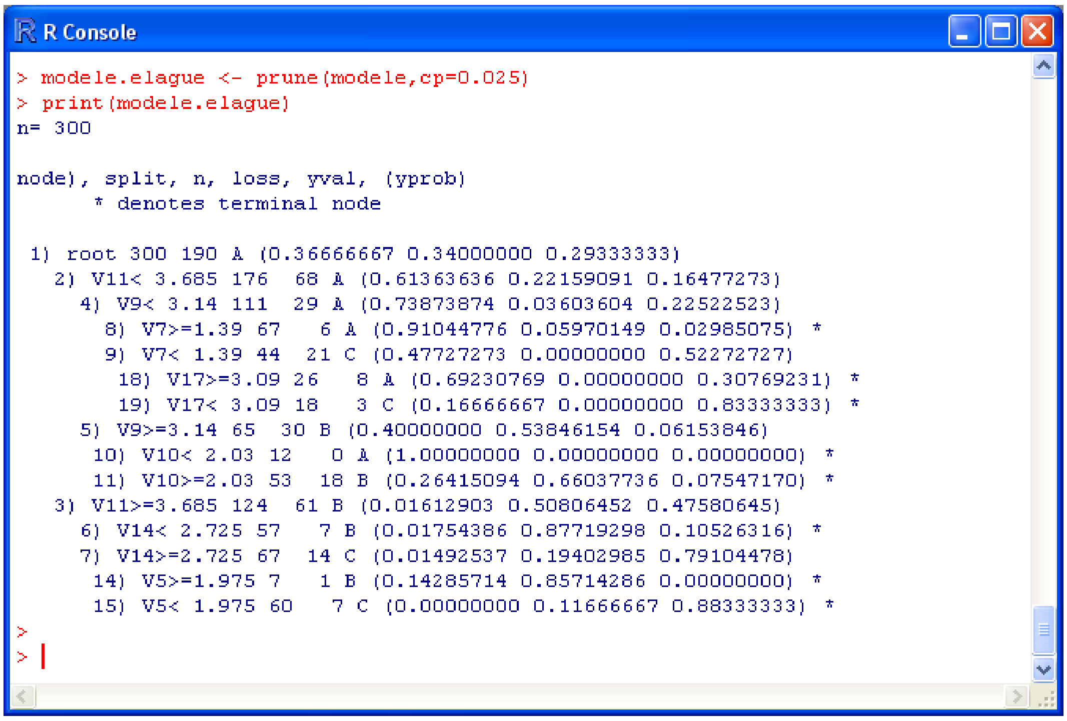


```
> #appliquer le modèle sur la partie test
> pred <- predict(modele.elague,newdata=test,type="class")
> summary(pred)
   A    B    C 
2209 1696 1095 
> 
> #confronter valeurs observées et prédites - matrice de confusion
> mc <- table(test$CLASS,pred)
> print(mc)
   pred
       A    B    C
  A 1338  296   45
  B  228 1171  283
  C  643  229  767
> 
> #calculer le taux d'erreur
> err <- 1.0 - (mc[1,1]+mc[2,2]+mc[3,3])/sum(mc)
> print(err)
[1] 0.3448
> 
> 
```

In this case, the error rate test is 34.48%. This is significantly worse than that created by Tanagra. Perhaps, we have too pruned the tree.

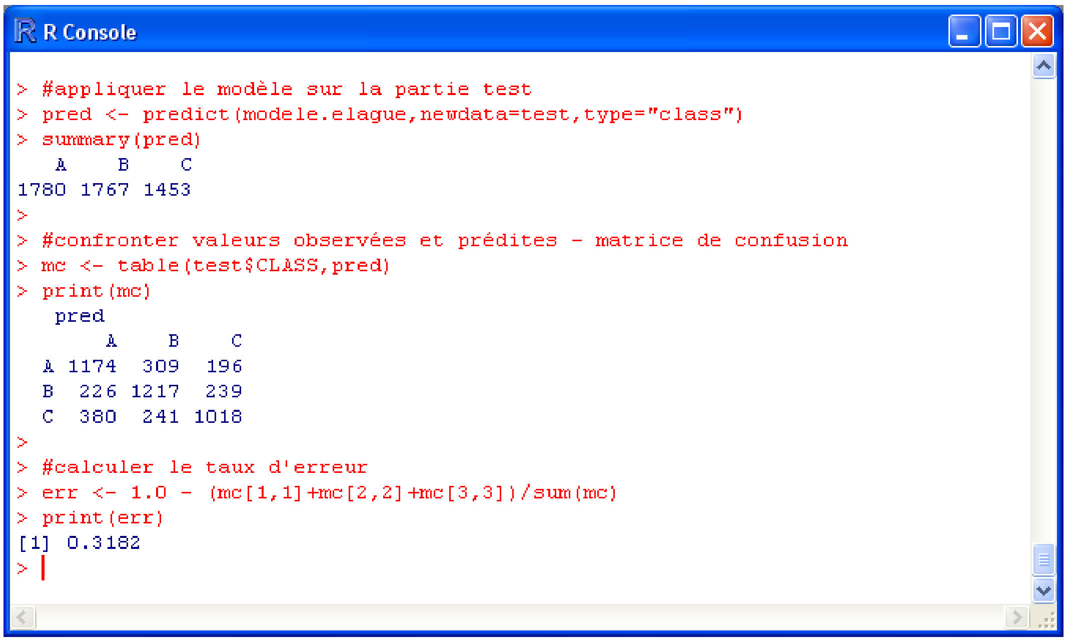
We can try another solution by specifying another value of CP. **This is a very attractive functionality of rpart**. For instance, if we set cp = 0.025, we obtain a tree with 8 leaves[10].

[10] Not showed in this tutorial, **the plotcp(.)** command allows to generate a chart showing the cross-validation error rate according to the number of splitting.



```
> modele.elague <- prune(modele,cp=0.025)
> print(modele.elague)
n= 300

node), split, n, loss, yval, (yprob)
      * denotes terminal node

 1) root 300 190 A (0.36666667 0.34000000 0.29333333)
   2) V11< 3.685 176  68 A (0.61363636 0.22159091 0.16477273)
     4) V9< 3.14 111  29 A (0.73873874 0.03603604 0.22522523)
       8) V7>=1.39 67   6 A (0.91044776 0.05970149 0.02985075) *
       9) V7< 1.39 44  21 C (0.47727273 0.00000000 0.52272727)
        18) V17>=3.09 26   8 A (0.69230769 0.00000000 0.30769231) *
        19) V17< 3.09 18   3 C (0.16666667 0.00000000 0.83333333) *
     5) V9>=3.14 65  30 B (0.40000000 0.53846154 0.06153846)
      10) V10< 2.03 12   0 A (1.00000000 0.00000000 0.00000000) *
      11) V10>=2.03 53  18 B (0.26415094 0.66037736 0.07547170) *
   3) V11>=3.685 124  61 B (0.01612903 0.50806452 0.47580645)
     6) V14< 2.725 57   7 B (0.01754386 0.87719298 0.10526316) *
     7) V14>=2.725 67  14 C (0.01492537 0.19402985 0.79104478)
      14) V5>=1.975 7   1 B (0.14285714 0.85714286 0.00000000) *
      15) V5< 1.975 60   7 C (0.00000000 0.11666667 0.88333333) *
>
> 
```

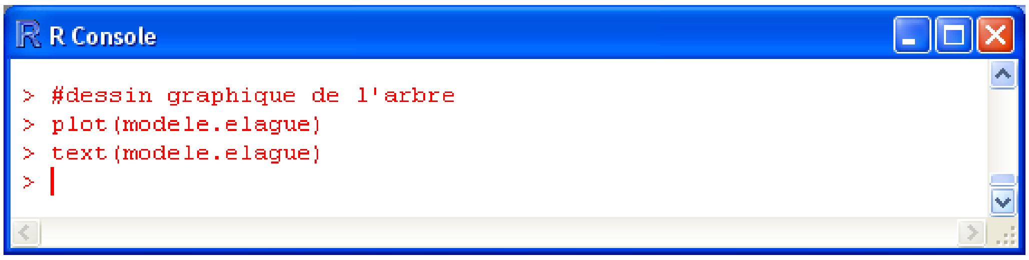
And the test error rate becomes 31.82%.



```
> #appliquer le modèle sur la partie test
> pred <- predict(modele.elague,newdata=test,type="class")
> summary(pred)
   A    B    C
1780 1767 1453
>
> #confronter valeurs observées et prédites - matrice de confusion
> mc <- table(test$CLASS,pred)
> print(mc)
   pred
       A    B    C
  A 1174  309  196
  B  226 1217  239
  C  380  241 1018
>
> #calculer le taux d'erreur
> err <- 1.0 - (mc[1,1]+mc[2,2]+mc[3,3])/sum(mc)
> print(err)
[1] 0.3182
> 
```

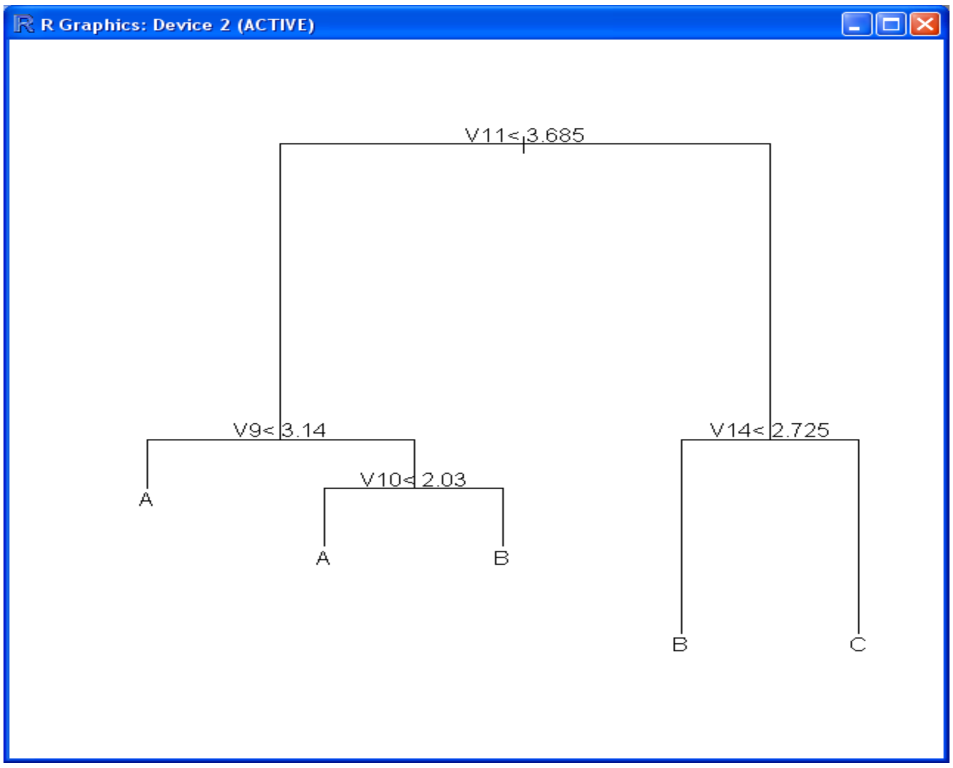
4.6 Graphical representation of the tree

RPART provides some command to generate a graphical representation of the decision tree.



```
> #dessin graphique de l'arbre
> plot(modele.elague)
> text(modele.elague)
> 
```

We obtain:



5 Conclusion

CART has undeniable qualities compared to other techniques for inducing classification trees. It is able to produce models "turnkey", with performances at least as good as the others.

But in addition, it can provide various scenarios of solutions. The tools such as the (Figure 3) or (Figure 4) allow to the user to select the best model according to their problem and dataset characteristics. This is an essential advantage in the practice of Data Mining.

6 References

- L. Breiman, J. Friedman, R. Olsen, C. Stone, *Classification and Regression Trees*, Chapman & Hall, 1984.
- D. Zighed, R. Rakotomalala, *Graphes d'Induction : Apprentissage et Data mining*, Hermès, 2000; chapitre 5, "CART".
- B. Ripley, *Pattern Recognition and Neural Networks*, Cambridge University Press, 1996; chapter 7, "Tree-Structured Classifiers".
- W. Venables, B. Ripley, *Modern Applied Statistics with S*, Springer, 2002; chapter 9, "Tree-based methods".
- About the RPART package
    - http://cran.cict.fr/web/packages/rpart/rpart.pdf for the technical description of the package;
    - « An introduction to Recursive Partitioning – Using the RPART Routines » (short version) - http://eric.univ-lyon2.fr/~ricco/cours/didacticiels/r/short_doc_rpart.pdf
    - « An introduction to Recursive Partitioning – Using the RPART Routines » (long version) http://eric.univ-lyon2.fr/~ricco/cours/didacticiels/r/long_doc_rpart.pdf