

1 Topic

“Wrapper” for feature subset selection – Continuation.

This tutorial is the continuation of the preceding one about the wrapper feature selection in the supervised learning context (<http://data-mining-tutorials.blogspot.com/2010/03/wrapper-for-feature-selection.html>). We analyzed the behavior of [Sipina](#)¹, and we have described the source code for the wrapper process (forward search) under [R](http://www.r-project.org/) (<http://www.r-project.org/>). Now, we show the utilization of the same principle under [Knime 2.1.1](#), [Weka 3.6.0](#) and [RapidMiner 4.6](#).

The approach is as follows: (1) we use the training set for the selection of the most relevant variables for classification; (2) we learn the model on selected descriptors; (3) we assess the performance on a test set containing all the descriptors.

This third point is very important. We cannot know the variables that will be finally selected. We do not have to manually prepare the test file by including only those which have been selected by the wrapper procedure. This is essential for the automation of the process. Indeed, otherwise, each change of setting in the wrapper procedure leading to another subset of descriptors would require us to manually edit the test file. This is very tedious.

In the light of this specification, it appeared that only Knime was able to implement the complete process. With the other tools, it is possible to select the relevant variables on the training file. But, I could not (or I did not know) apply the model on a test file containing all the original variables.

The naive bayes classifier is the learning method used in this tutorial².

2 Dataset

We have partitioned the dataset into two distinct files³. We use the ARFF (Weka) file format. The first data file (**mushroom-train.arff**) is the training set, there are 2000 instances. We use this dataset for the wrapper process in order to select the most relevant descriptors, and for the construction of the final model with the selected descriptors.

The second one corresponds to the test set (6124 instances). We use this dataset for the assessment of the final model. As we say above, it must have the same organization than the training set. We do not know what will be the selected descriptors. We note also that this dataset is never used during the training phase, whether during the search of the relevant variables or during the construction of the final model.

¹ <http://sipina.over-blog.fr/ext/http://eric.univ-lyon2.fr/~ricco/sipina.html>

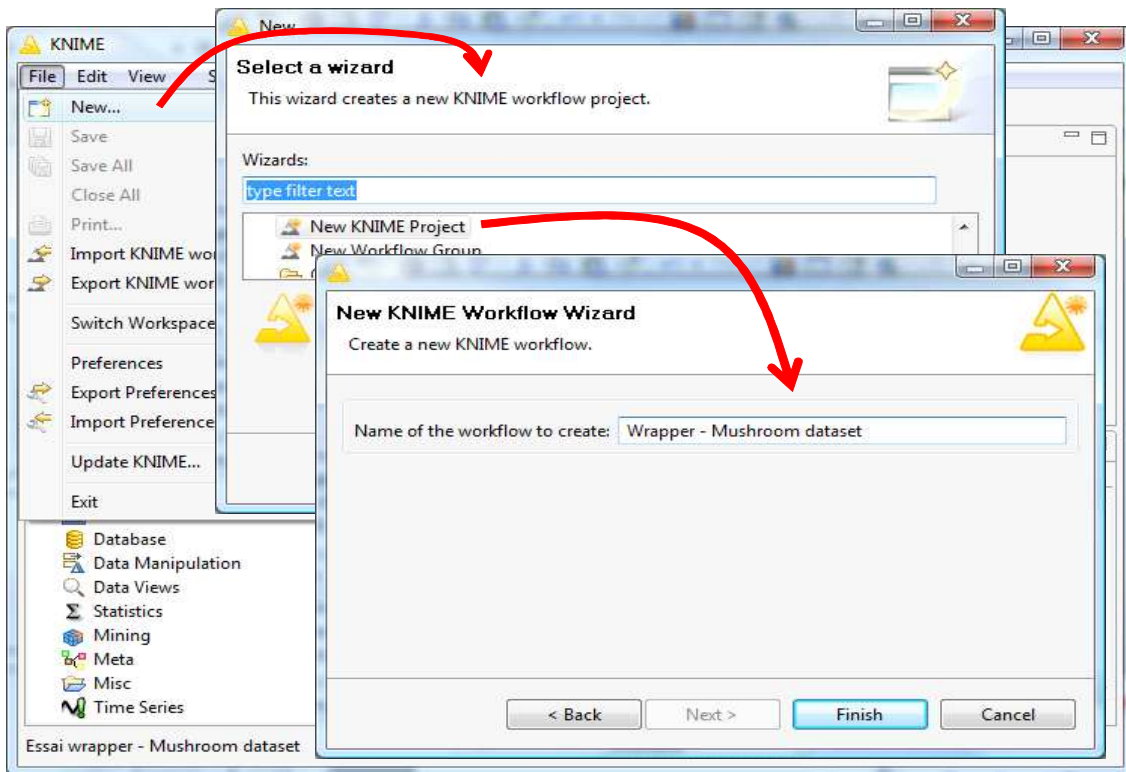
² http://en.wikipedia.org/wiki/Naive_Bayes_classifier

³ <http://eric.univ-lyon2.fr/~ricco/tanagra/fichiers/mushroom.wrapper.arff.zip>

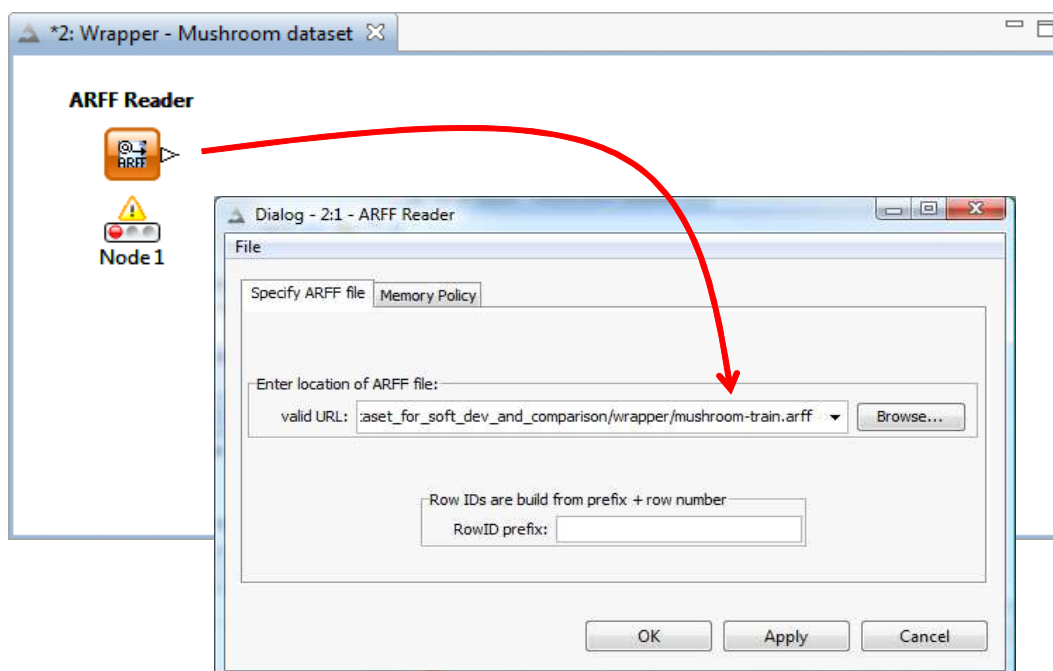
3 Wrapper process using Knime

3.1 Importing the data file

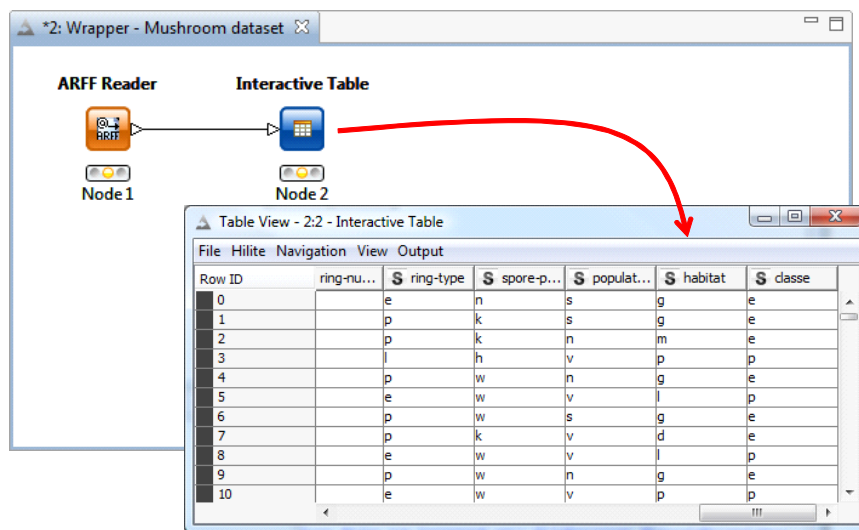
We launch Knime. We create a new project by clicking on the FILE / NEW menu. The name of the new project is « Wrapper – Mushroom dataset ».



Then, we add the ARFF READER (IO/READ branch into the "Node Repository") component into the workflow. We set the data filename **mushrom-train.arff**.



We check the dataset by using the INTERACTIVE TABLE component (DATAVIEWS). We can visualize the 2000 instances. "Classe" is the target attribute.

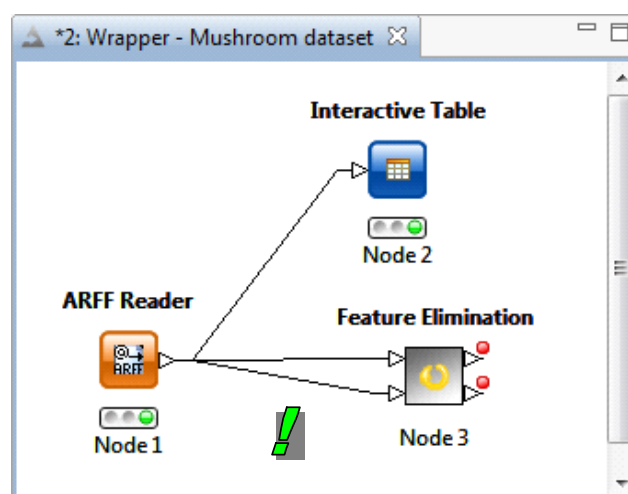


3.2 The « Feature Elimination » meta-node

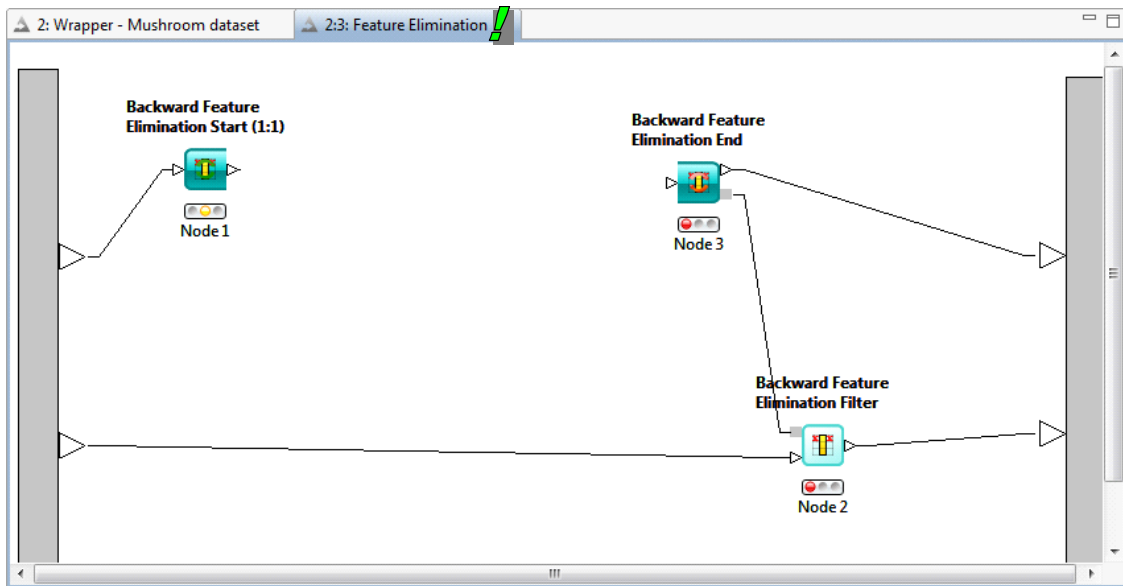
We can define meta-nodes with Knime. This very enthusiastic functionality allows to define a typical sequence of operations. In our context, we use the predefined FEATURE ELIMINATION meta-node (META). It is based on the backward elimination of the irrelevant variables. It uses explicitly the error rate during the selection process.

Roughly speaking, the approach can be described as follows: we start with all the features; we remove the worst variable i.e. which induces the most favorable evolution of the error rate; we repeat the process until we removed all the variables. The selected subset is the one which is the best according the error rate.

We insert the FEATURE ELIMINATION component into the workflow. We connect twice the ARFF READER to this one. Twice, because we use the training set in order to select the best subset of variables and in order to learn the final model on the selected variables.



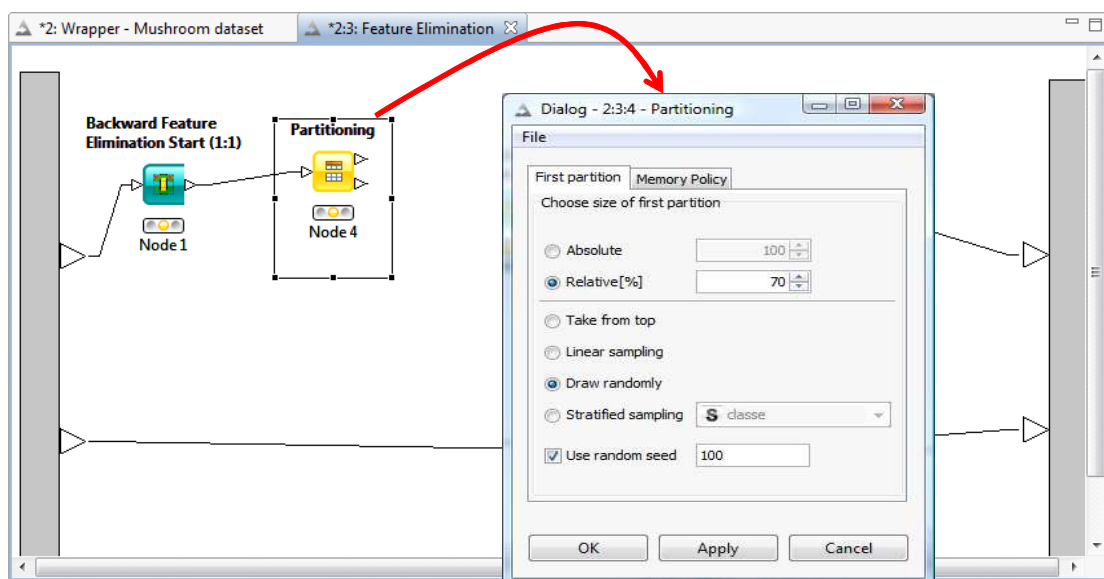
We double-click on the FEATURE ELIMINATION component to edit it.



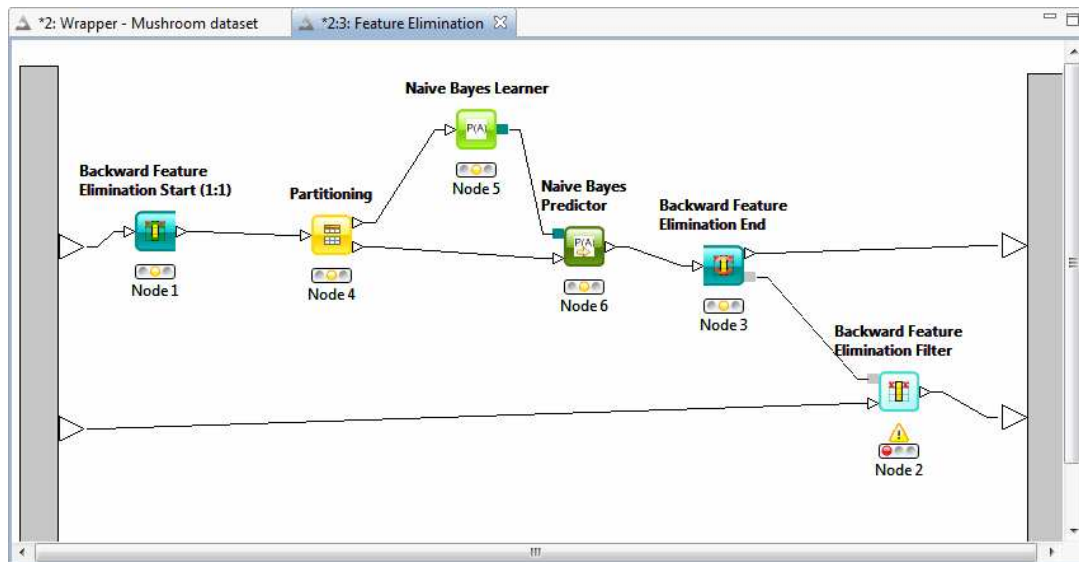
The idea is to program a search loop. In our context, the BACKWARD FEATURE ELIMINATION START component is at the beginning of the loop, BACKWARD FEATURE ELIMINATION END is at the end. Inside of the loop, we partition the dataset (2000 instances) into a learning set (70%), it is used for the construction of the model on the current selection of the variables; and a validation set (30%), it allows to compute an unbiased estimation of the error rate. This subdivision is very important. Indeed, if we use the same dataset during these two phases, we have a biased estimation of the error, and above all, we favor the models with many variables.

Note: In fact, we have to use a resampling approach in order to measure the error rate during the wrapper process (e.g. cross validation). But, the implementation of this approach under Knime seems very complicated, even if it should be possible using another meta-node.

We add the PARTITIONING component (DATA MANIPULATION / ROW / TRANSFORM) into the workflow. We connect the first component. We set the parameters (CONFIGURE menu).



We add the NAIVE BAYES LEARNER and the NAIVE BAYES PREDICTOR as follows.



We can launch the wrapper process now. We click on the EXECUTE menu of the BACKWARD FEATURE ELIMINATION END search component.

We can follow the progress of operations in the "Console" window. When the process is finished, all components are green. Then, we configure the BACKWARD ELIMINATION FILTER FEATURE component which allows to choose the "optimal" subset of variables.

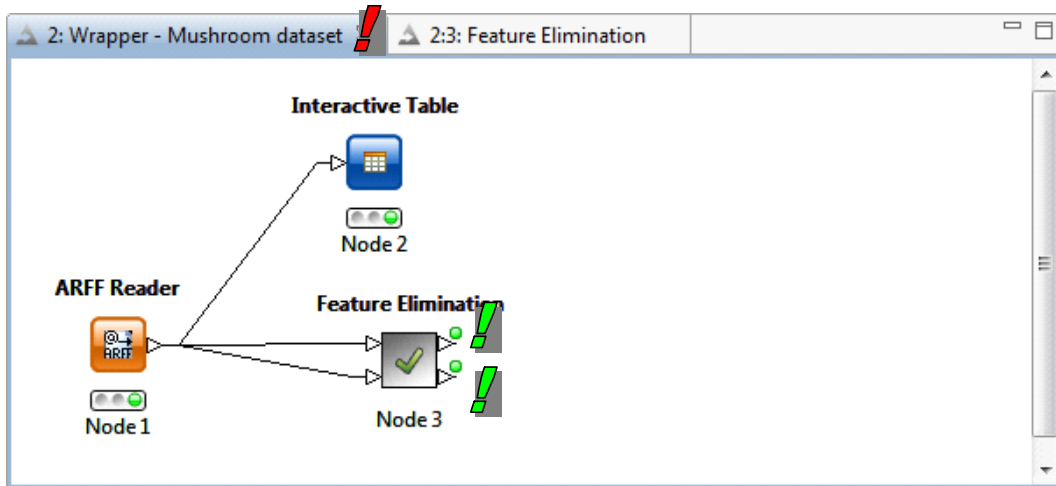
In the dialog box, we note that the minimal error rate is 0.002. It corresponds to a subset of 4 variables: *odor*, *stalk-surface-above*, *ring-type* and *spore-print-color*.

| Error | Nr. of features | Feature Name |
|-------|-----------------|----------------------|
| 0.002 | 21 | cap-shape |
| 0.002 | 20 | cap-surface |
| 0.002 | 20 | cap-color |
| 0.002 | 19 | bruites? |
| 0.002 | 18 | odor |
| 0.002 | 17 | gill-attachment |
| 0.002 | 16 | gill-spacing |
| 0.002 | 15 | gill-size |
| 0.002 | 14 | gill-color |
| 0.002 | 13 | stalk-shape |
| 0.002 | 12 | stalk-root |
| 0.002 | 11 | stalk-surface-above |
| 0.002 | 10 | stalk-surface-below |
| 0.002 | 9 | stalk-color-above-ri |
| 0.002 | 8 | stalk-color-below-ri |
| 0.002 | 7 | veil-type |
| 0.002 | 6 | veil-color |
| 0.002 | 4 | ring-number |
| 0.002 | 3 | ring-type |
| 0.005 | 2 | spore-print-color |
| 0.008 | 2 | population |
| 0.012 | 23 | habitat |
| 0.012 | 1 | classe |

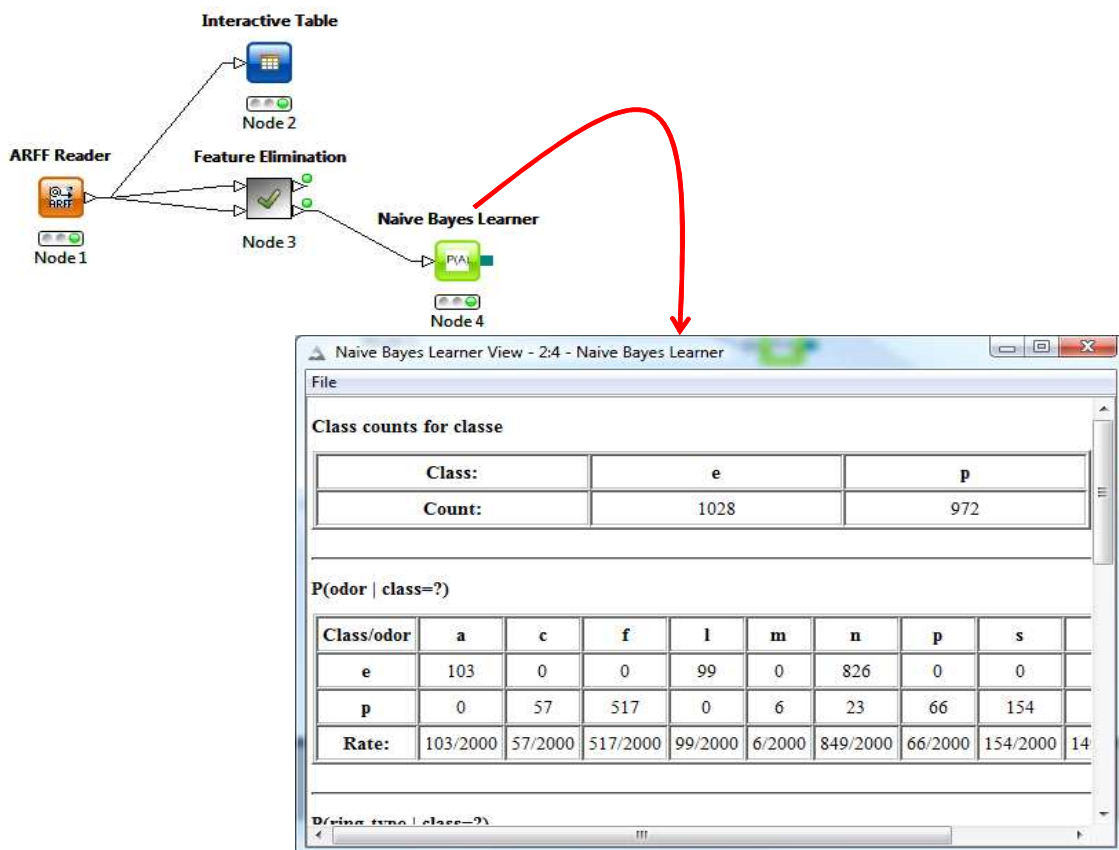
We confirm this subset. They are supplied at the output of the component. We select also the « Include Target Column » option. Then we click on the EXECUTE contextual menu.

3.3 Construction of the final model

We return in the original workspace tab. We observe that the two outputs of the FEATURE ELIMINATION meta-node are green.

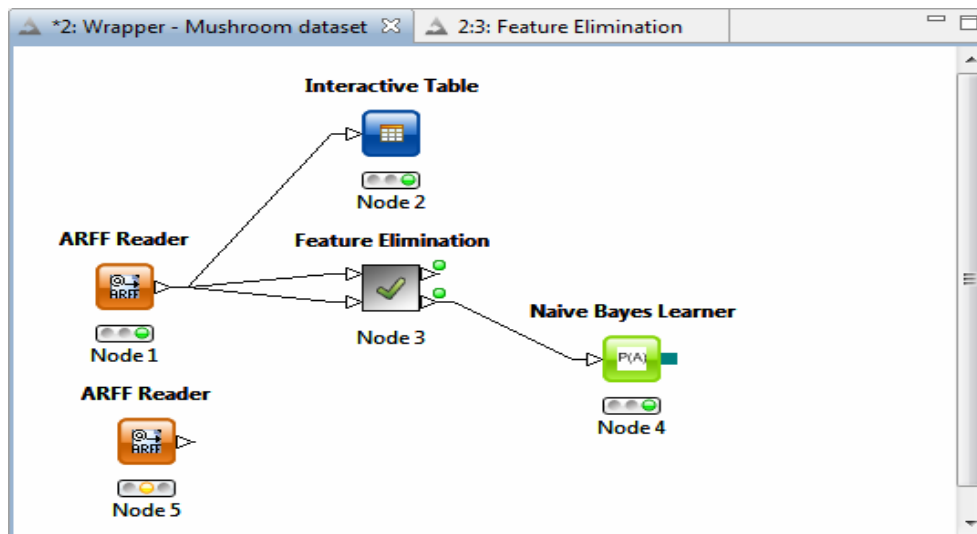


The second output is very interesting for us. Because, it supplies all the instances of the training set with only the selected variables (and the target column). We add the NAÏVE BAYES LEARNER (CLASSE is the target attribute, CONFIGURE menu). We click on the EXECUTE AND OPEN VIEW menu. We obtain the following results. These are the cross-tabulation of each predictor with the target attribute.

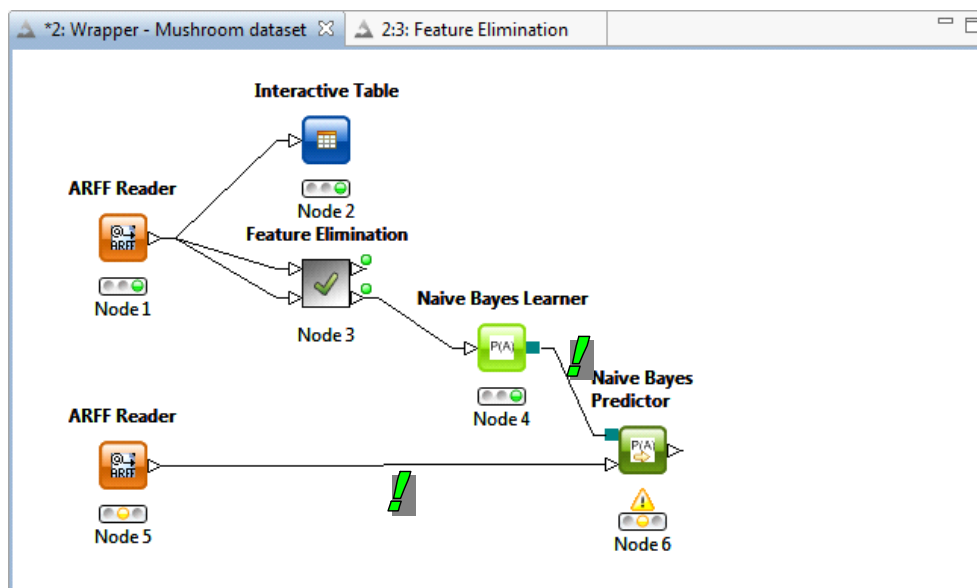


3.4 Error rate evaluation on the test set

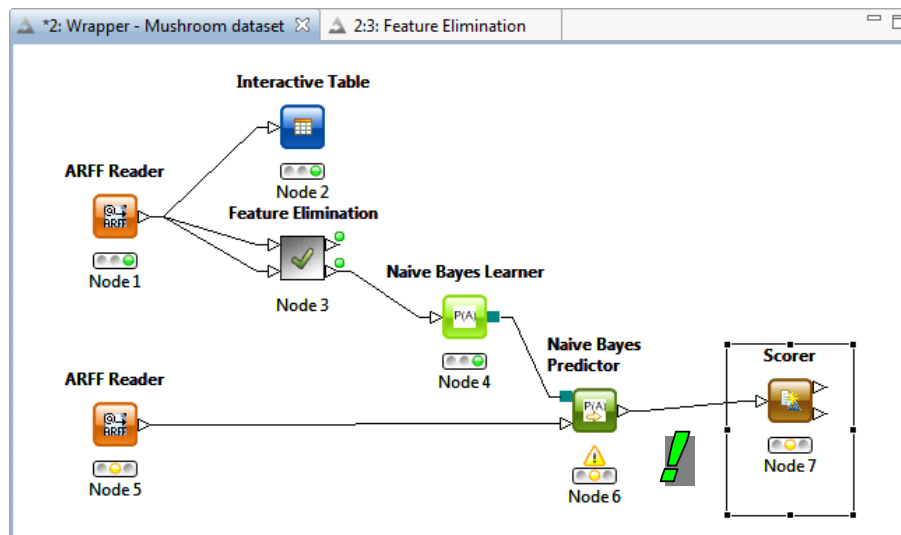
We intend to use the test set (6124 instances) in order to evaluate the accuracy of the classifier. We load the data file with the ARFF READER component.



Then we add the NAIVE BAYES PREDICTOR (*MINING / BAYES*). It takes as input the test sample and the classifier from the "learner".



To compute the confusion matrix and the test error rate, we add the SCORER component (*MINING / SCORING*). With the CONFIGURE menu, we indicate that CLASS corresponds to the target attribute and WINNER contains the predicted values of the model.



We click on the EXECUTE AND OPEN VIEW menu. We obtain the following confusion matrix. The test error rate is 0.212% (13 misclassified instances among 6124).

| classe \ Wi... | e | p |
|----------------|------|------|
| e | 3180 | 0 |
| p | 13 | 2931 |

Correct classified: 6,111 Wrong classified: 13
 Accuracy: 99.788 % Error: 0.212 %

4 Wrapper process using Weka

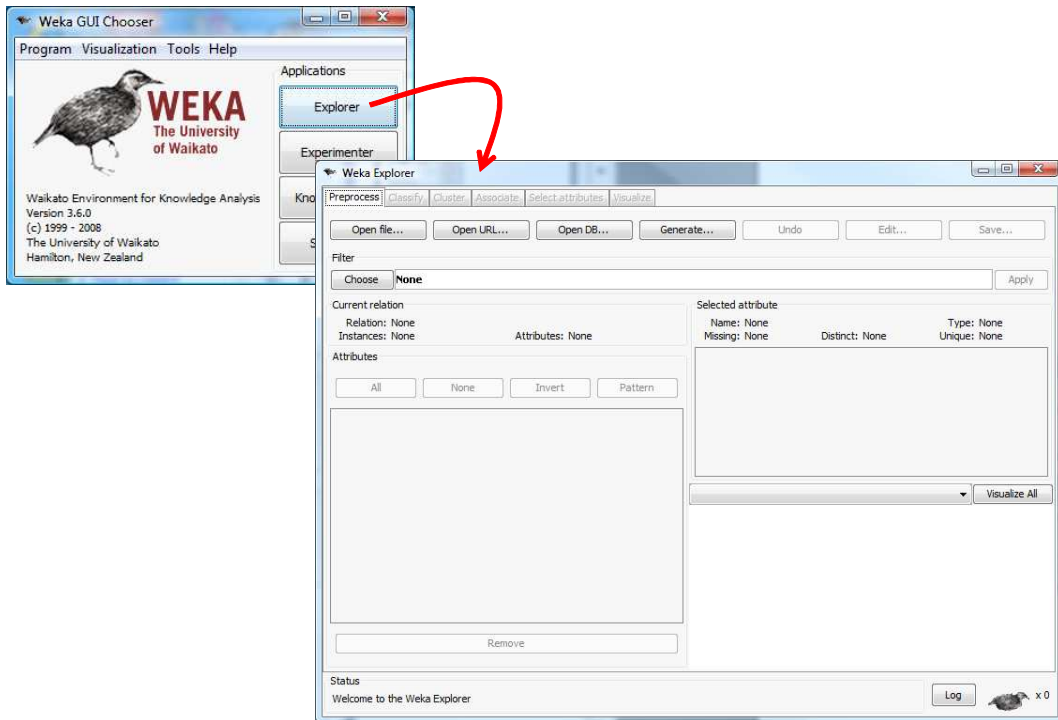
Into the EXPLORER mode, Weka allows to implement efficiently the wrapper selection approach.

Unlike Knime, it is very easy to perform the cross validation error rate evaluation during the process. But, on the other hand, it seems not possible (*I did not found*) to apply the classifier computed on the selected variables on a test set including all the descriptors.

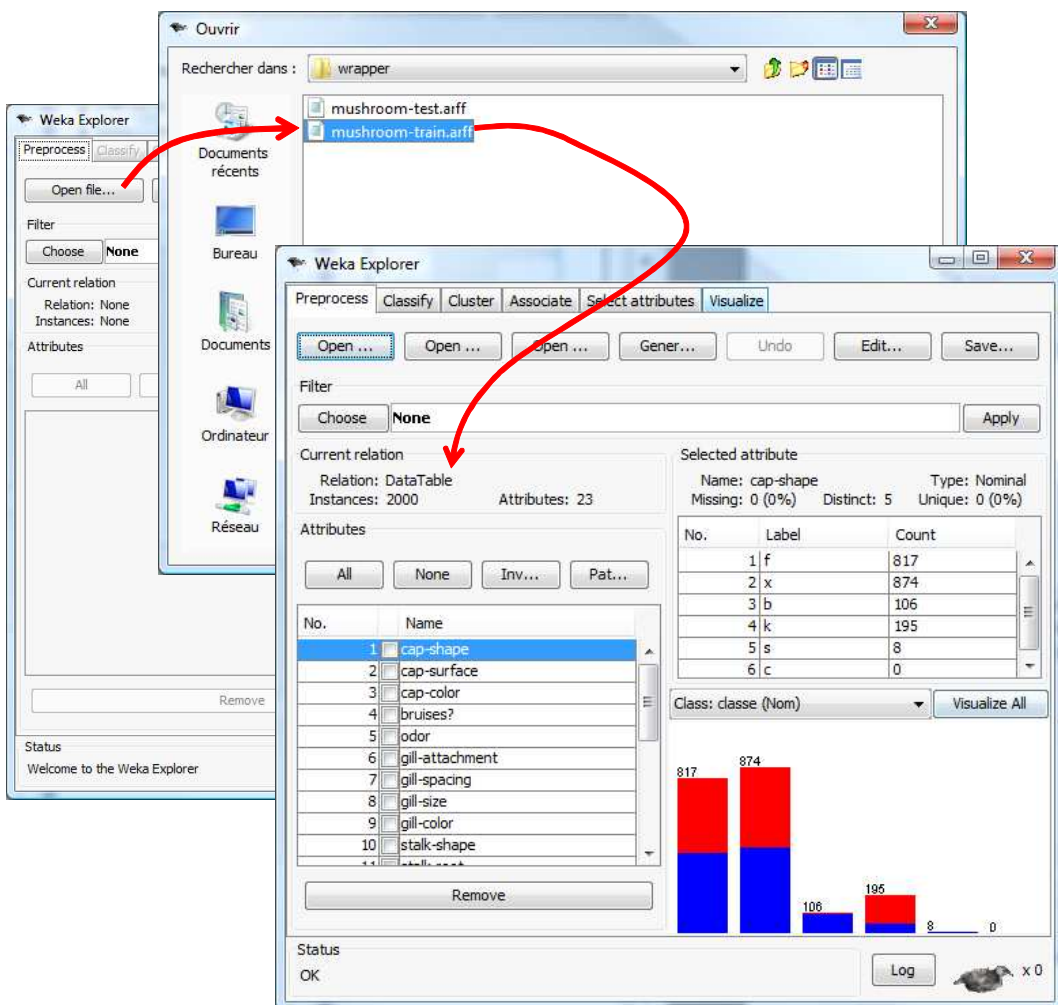
Thus, in this section, we show only how to perform the wrapper feature selection process with Weka. The error rate is computed with the cross validation resampling scheme.

4.1 Importing the learning set

We launch Weka. We select the EXPLORER mode.

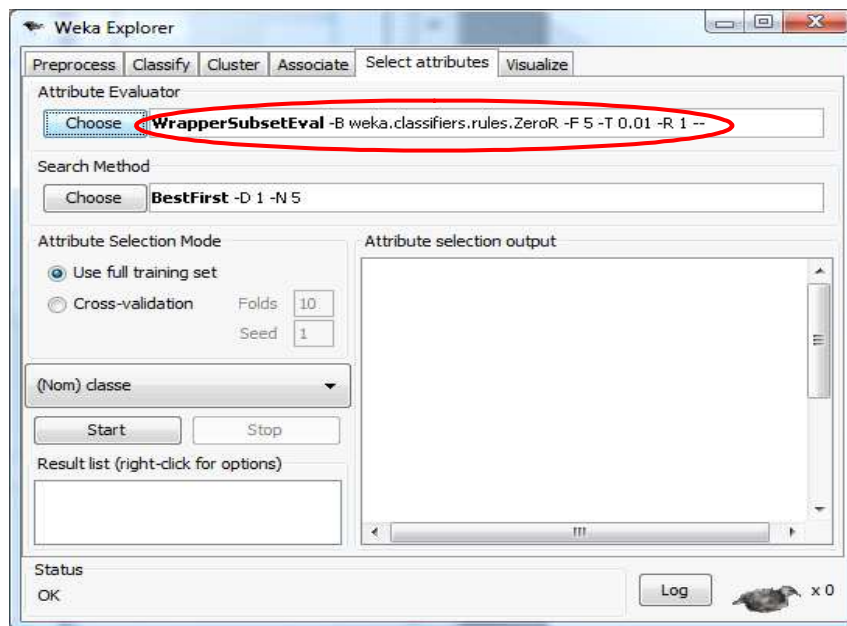


Into the PREPROCESS tab, we click on the OPEN FILE button. We select **mushroom-train.arff**.

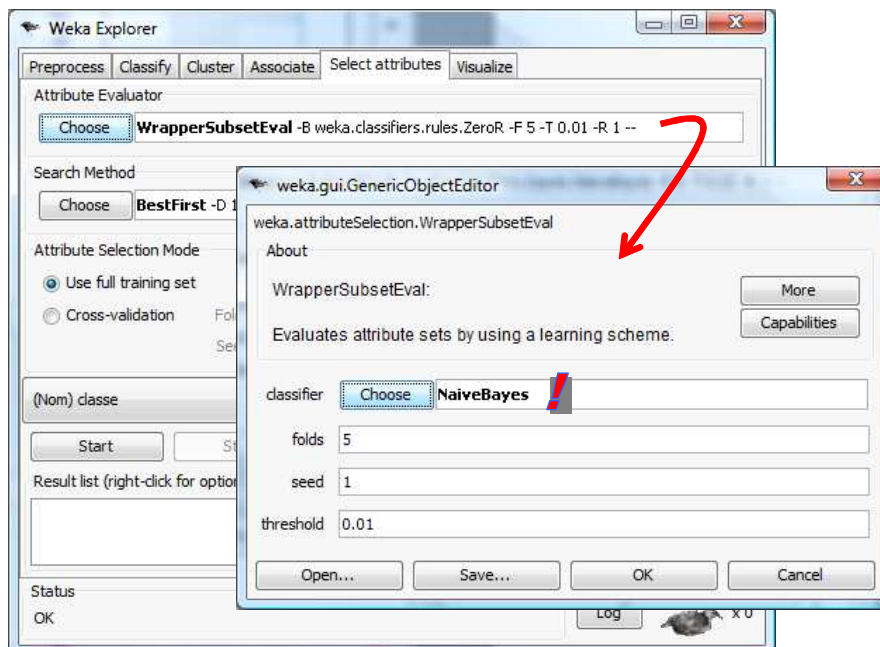


4.2 Implementing the “wrapper” process using Weka

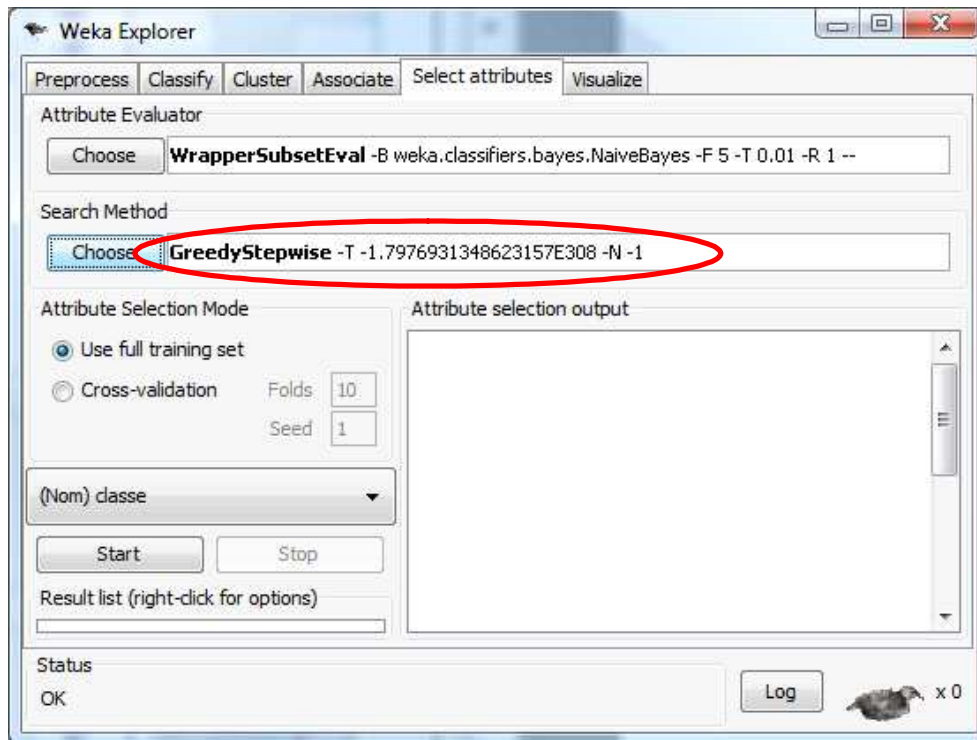
We activate the SELECT ATTRIBUTES tab. We choose (CHOOSE button) the WRAPPERSUBSETEVAL approach as ATTRIBUTE EVALUATOR.



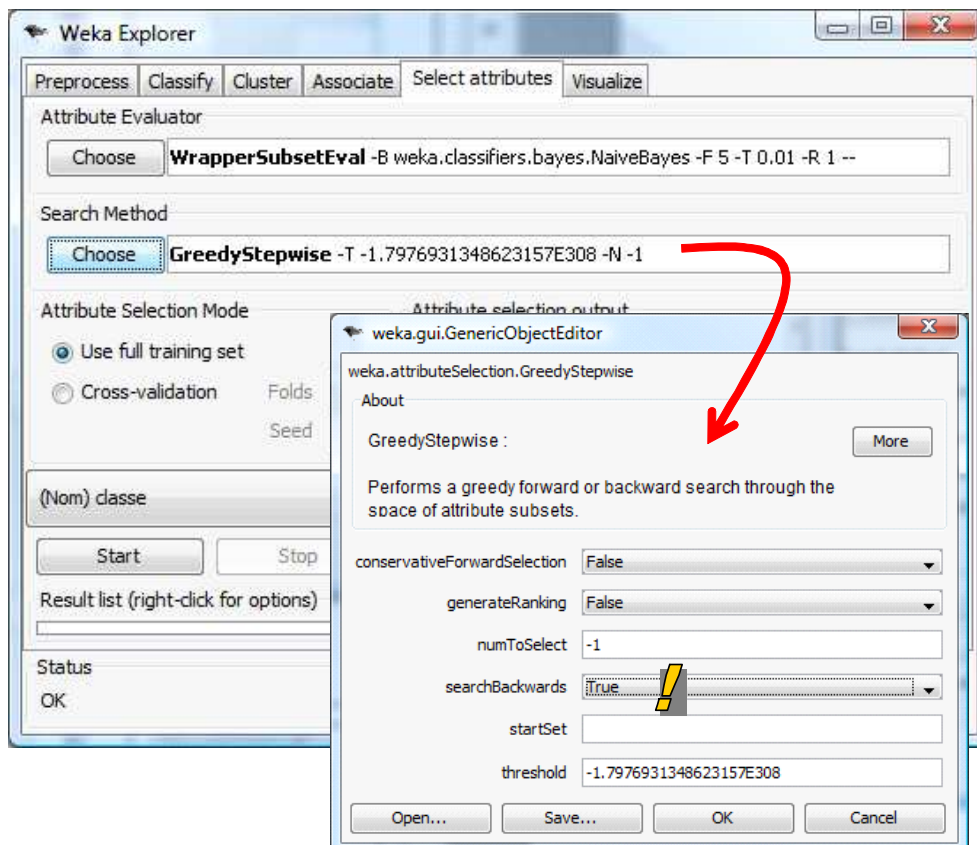
We click on the description of the approach to define the settings. We set the learning method and the number of folds for the cross validation.



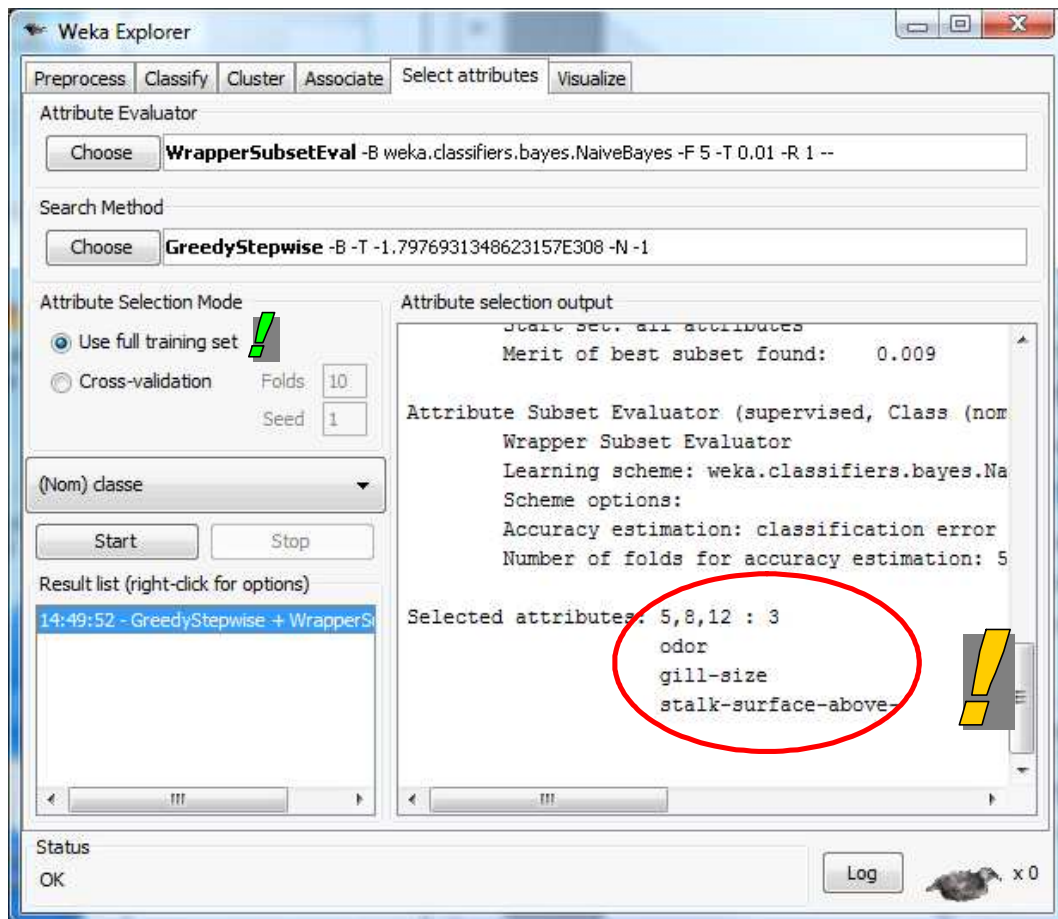
About the SEARCH METHOD, we want to use the BACKWARD strategy. We select the GREEDY STEPWISE approach.



By clicking on the description of the approach, we set SEARCHBACKWARDS = TRUE.



We launch the search process by clicking on the START button. We note that the procedure uses all the available instances (ATTRIBUTE SELECTION MODE = USE FULL TRAINING SET).



Odor, gill-size and stalk-surface-above are the selected variables.

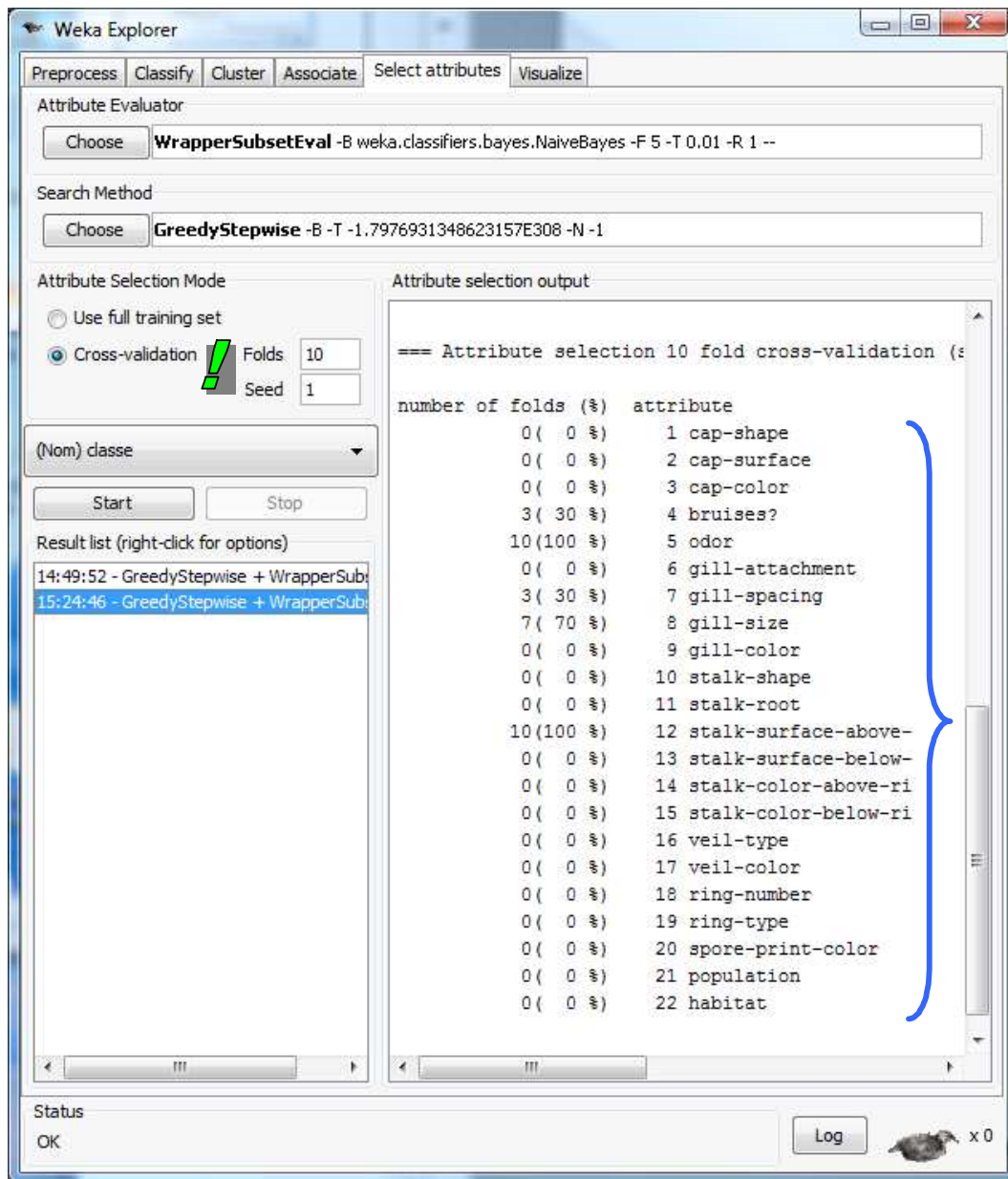
4.3 Weighting of attributes using the cross validation

The approach described above is the same as the one implemented using Sipina, R (<http://data-mining-tutorials.blogspot.com/2010/03/wrapper-for-feature-selection.html>) and Knime (above).

We obtain an "optimal" subset of variables. But we know that this solution relies heavily on the learning set. If we use another dataset, we could obtain another subset. Sometimes, it can be very different. This is rather embarrassing, especially when we want to interpret the results in relation with the domain knowledge.

Weka puts forward a very interesting approach to overcome this drawback. Instead of performing a unique search session, it repeats the search using a resampling scheme. We thus obtain various subsets of variables. We profit from these solutions to count the number of occurrence of each variable in these subsets. The goal is to obtain a weighting of the variables according to their apparition during the whole process.

We select the ATTRIBUTE SELECTION MODE = CROSS-VALIDATION (FOLDS = 10) option. We click again on the START button.



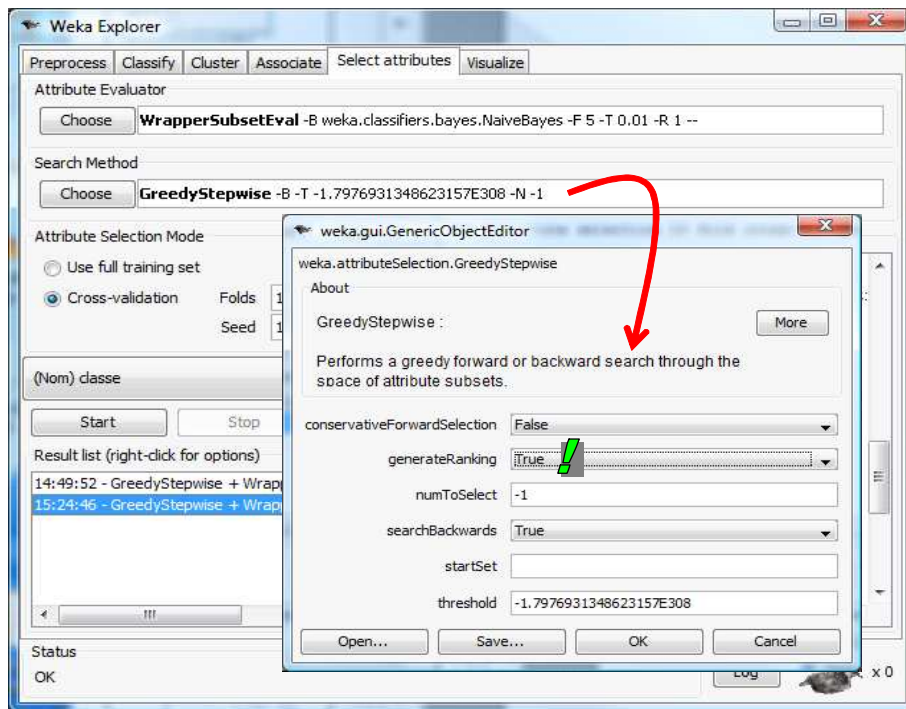
Some attributes are highlighted. For the 10 sessions of the cross validation, 5 of them are included at least one times in an "optimal" subset: bruises (3 times), odor (10), gill-spacing (3), gill-size (7) and stalk-surface-above (10). That relativizes the result with only 3 variables highlighted in the previous section.

4.4 « Ranking » of attributes

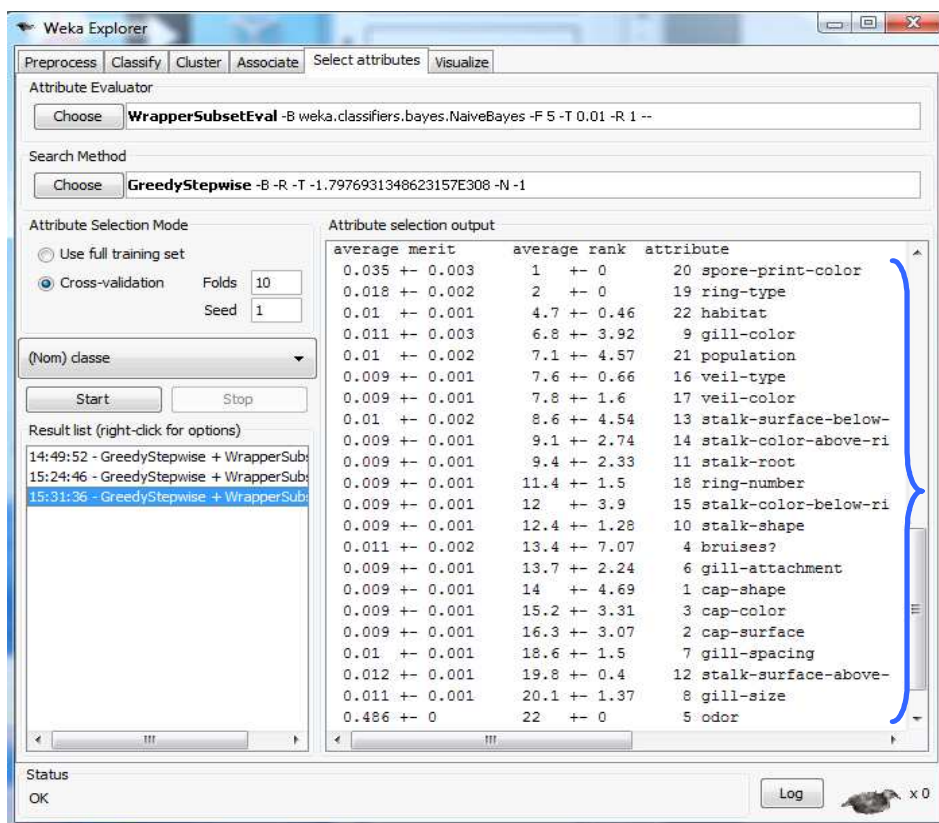
Weka puts forward another approach to overcome again the drawback of the "crisp" solution supplied by the standard wrapper selection process: this is the "ranking" approach.

About the backward search, the least important variable is the first removed. The most important one is the last removed. Weka assigns a weight to each variable according to this point of view. The result (the ranking) is smoothed by the encapsulation of this process inside a resampling scheme.

We modify the SEARCH METHOD, we set GENERATE RANKING = TRUE.



Then we click again on the START button.



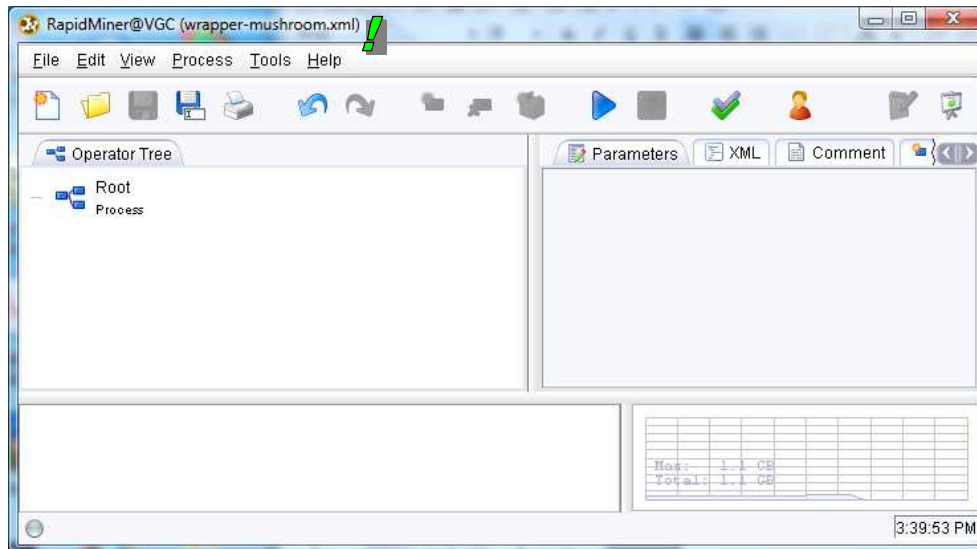
Average rank. Because we use a backward search, we must read the result from the bottom to the top. The most important variables are in the bottom part of the table. We have the average of the ranks of the variable. We have also an indication about the reliability of the assigned rank. It seems it is the standard error. We note for instance that "odor" has obtained always the rank 22.

5 Wrapper process using RapidMiner

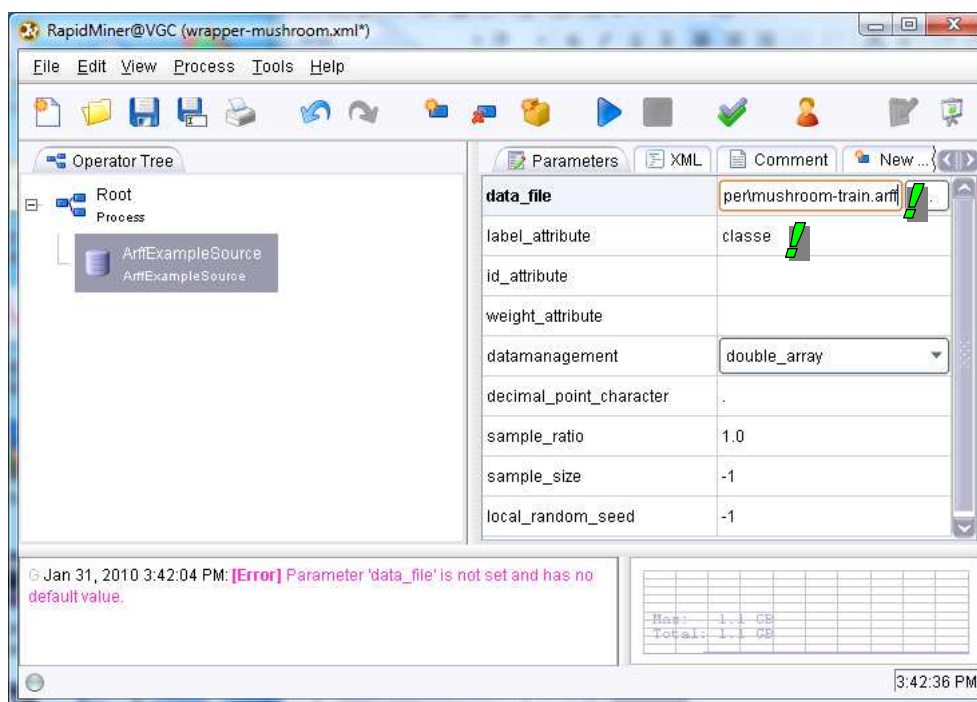
RapidMiner is very similar to Weka about the wrapper process. It can supply a ranking or a weighting of attributes based on a resampling scheme. As Weka also, we cannot deploy the final model with the selected variables on a dataset containing all the original attributes.

5.1 Standard wrapper using RapidMiner

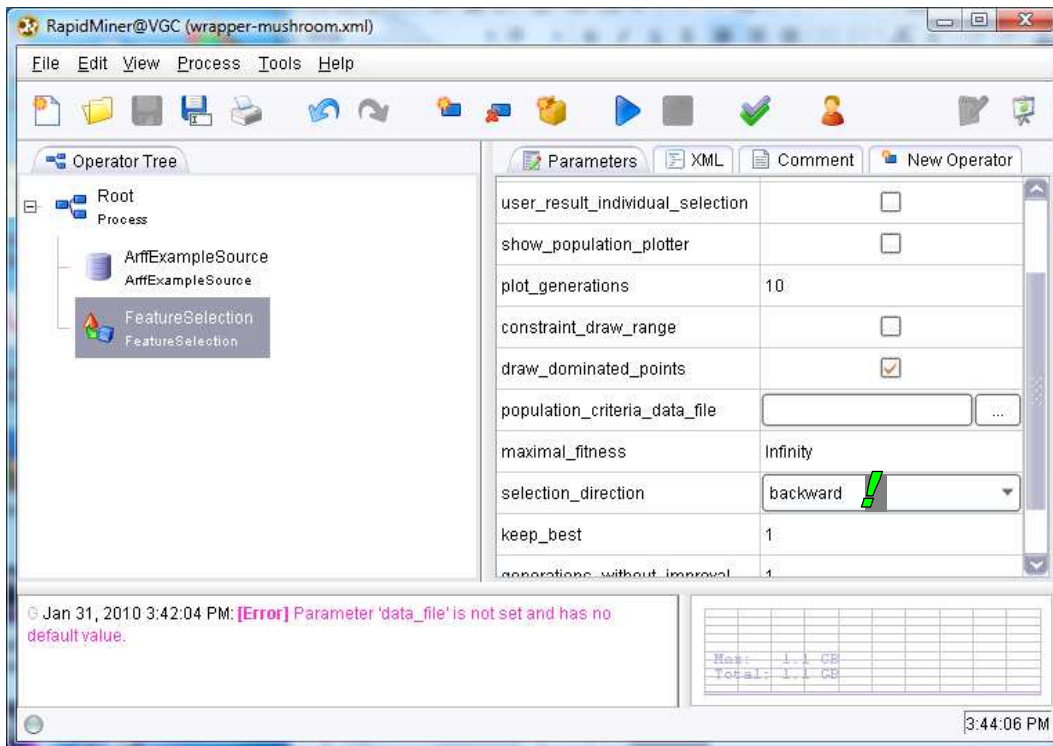
We launch RapidMiner. We create a new diagram by clicking on the FILE / NEW menu. We save the diagram " wrapper-mushroom.xml ".



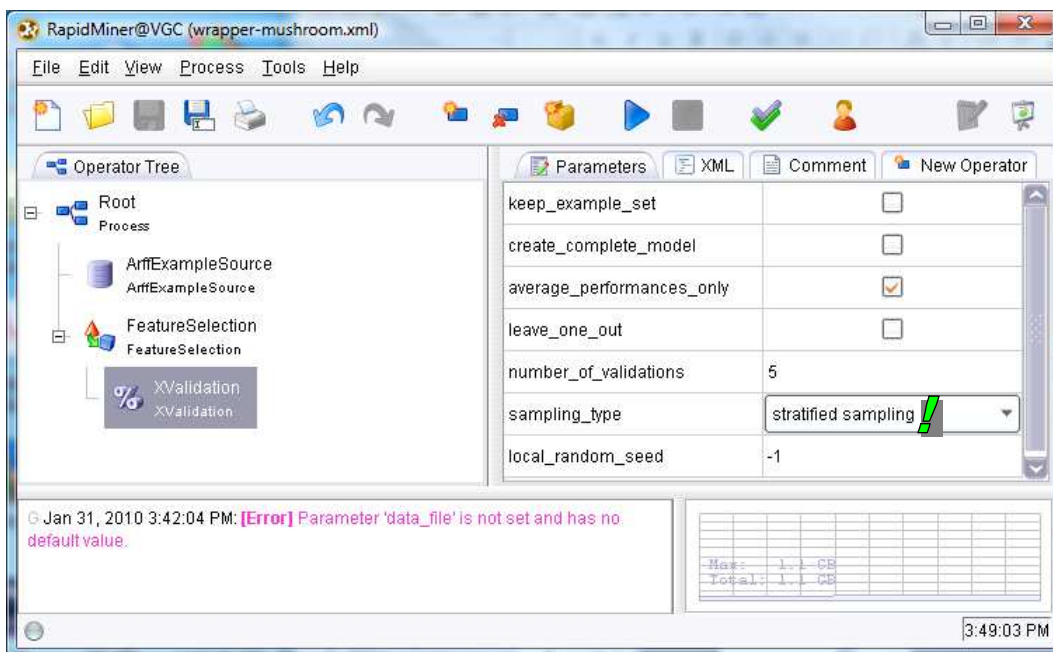
We must first load the dataset. We add the ARFFEXAMPLESOURCE component (*NEW OPERATOR / IO / EXAMPLES*) into the diagram. We set CLASSE as target (label) variable.



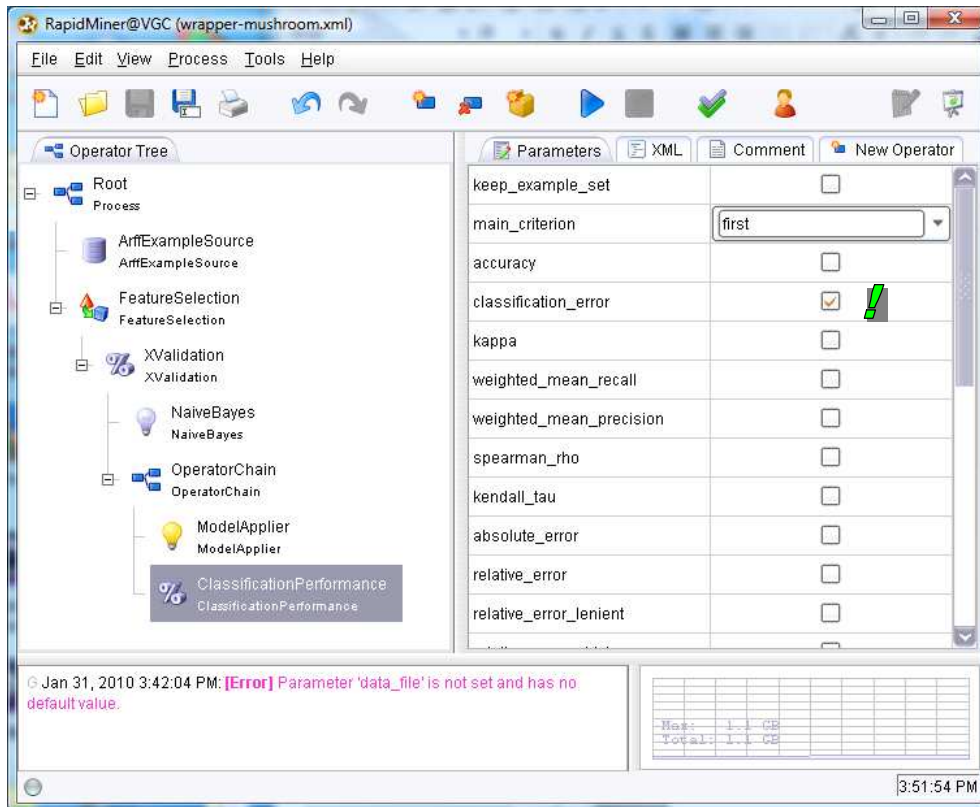
We add the FEATURE SELECTION component (*PREPROCESSING / ATTRIBUTES / SELECTION*). We specify the search strategy (*SELECTION_DIRECTION = BACKWARD*).



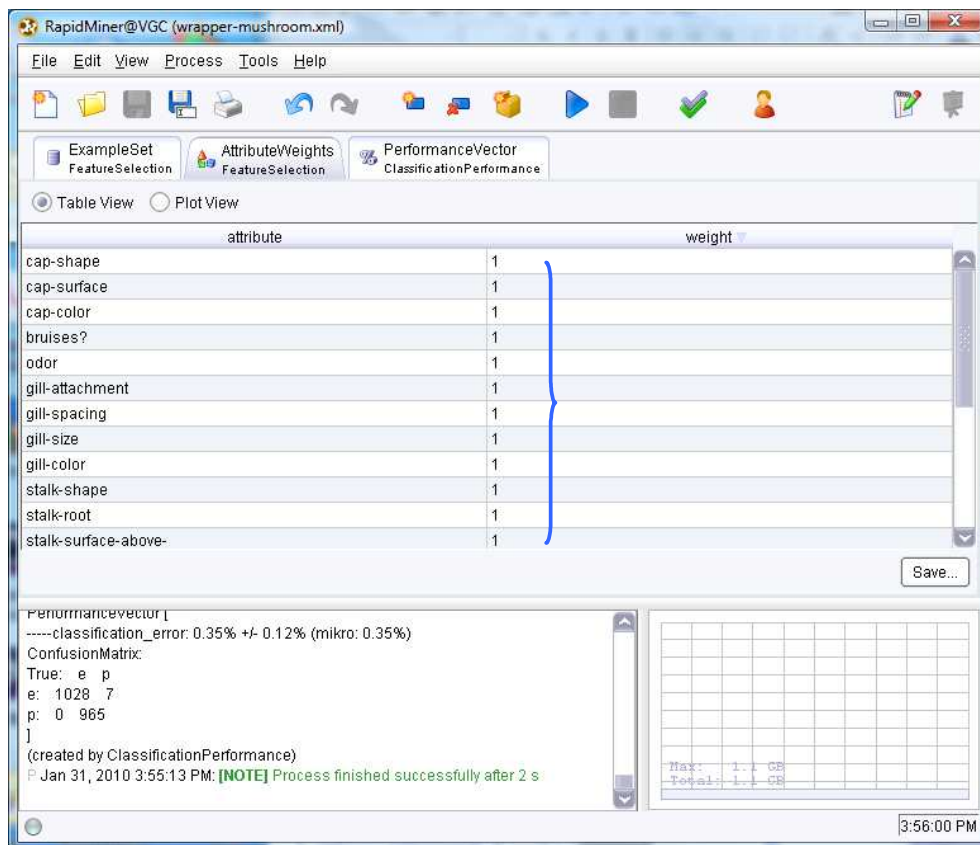
We use the cross validation to evaluate the error rate (5 folds and stratified sampling).



Then we add the train-test sequence inside of the cross-validation (*NAIVE BAYES + MODEL APPLIER + CLASSIFICATION PERFORMANCE*). The last component allows to compute the error rate on the various folds.

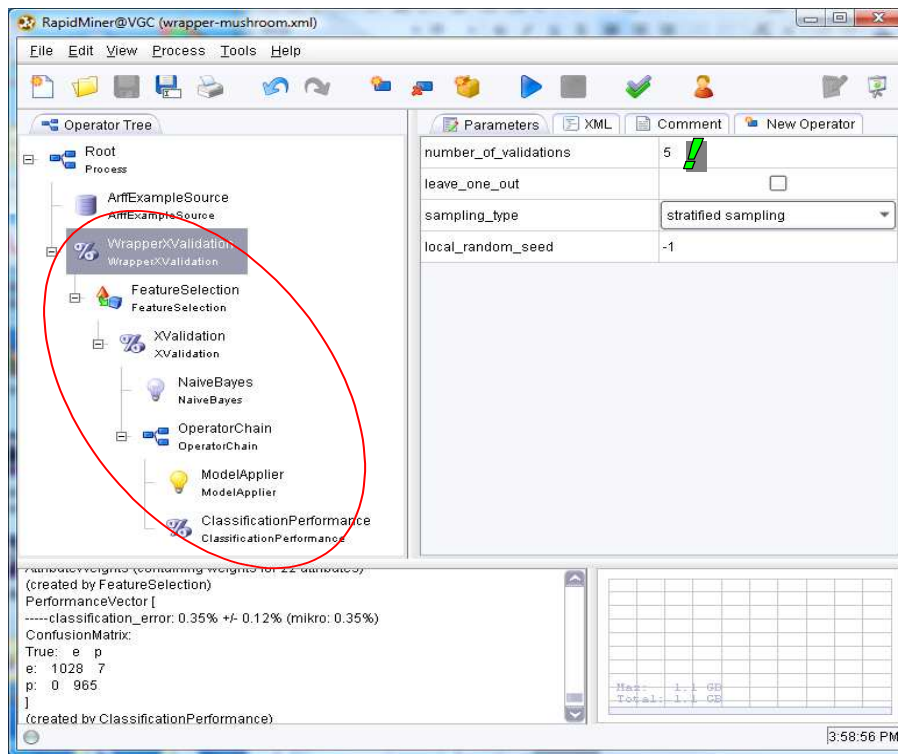


We launch the process by clicking on the RUN button. We obtain the selected variables (ATTRIBUTE WEIGHTS) and the confusion matrix of the final model (PERFORMANCE VECTOR).



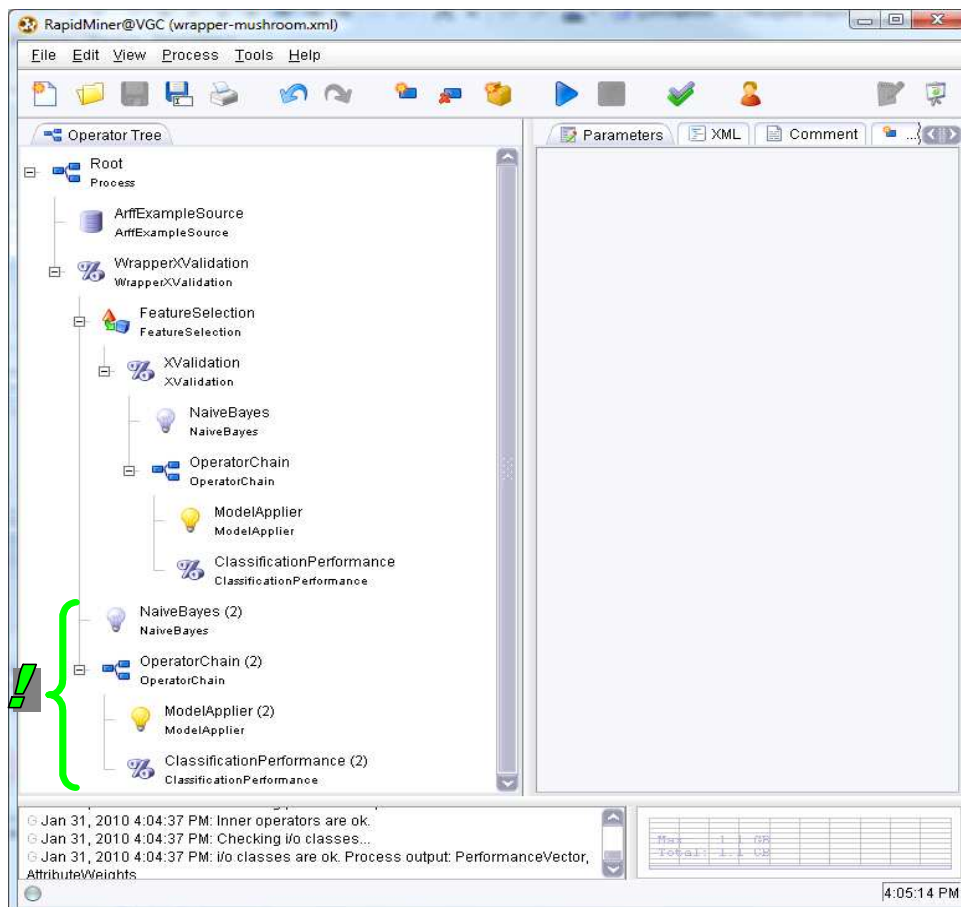
RapidMiner surprisingly selects many variables compared with the other tools.

5.2 Weighting of the attributes according to their relevance



Like Weka, RapidMiner can compute a weighting of attributes using a resampling scheme. We come back to the diagram. We add the WRAPPER XVALIDATION (VALIDATION) component. It embeds the selection process into a cross validation. We copy the previous branch of the diagram devoted to the wrapper process inside to this component.

Then we define again the train test sequence using the naive bayes classifier.



We click on the RUN button. Of course, the computation time is higher. We select the ATTRIBUTE WEIGHTS tab.

| attribute | weight |
|----------------------|--------|
| cap-surface | 1 |
| cap-color | 1 |
| odor | 1 |
| gill-attachment | 1 |
| gill-spacing | 1 |
| gill-size | 1 |
| gill-color | 1 |
| stalk-shape | 1 |
| stalk-root | 1 |
| stalk-surface-above- | 1 |
| veil-type | 1 |
| veil-color | 1 |
| spore-print-color | 1 |
| population | 1 |
| stalk-surface-below- | 0.800 |
| stalk-color-above-ri | 0.800 |
| ring-type | 0.800 |
| ... | 0.800 |

AttributeWeights (containing weights for 22 attributes)
 (created by WrapperXValidation)
 Jan 31, 2010 4:07:52 PM: [NOTE] Process finished successfully after 16 s

Here also, the previous result is relativized. But, the number of relevant variables is clearly overestimated. Perhaps it is more appropriate to use the forward strategy with RapidMiner. It seems that it selects the first subset found according to the search direction: the largest subset in the backward search; the smallest subset in the forward search.

6 Conclusion

Wrapper strategy explicitly uses an estimation of the error rate to determine the "optimal" subset of descriptors. It does not rely on the specificities of the learning algorithm when it explores the solutions. These are two characteristics that make it a preferred tool in a variable selection process.

However, as we noted in this tutorial, it is demanding on computational resources, especially when we want to highlight the rank or the weight of the attributes using a resampling scheme. Its implementation when dealing with very large databases, both in numbers of variables and observations, becomes very expensive. This characteristic is even more pronounced than we use a supervised learning technique which is also costly in computing time, such as some neural network or support vector machine algorithms.