University of Belgrade
Faculty of Organizational Sciences
Center for Business Decision Making

WHIBO User guide

Belgrade, November 2013

Introduction

WhiBo is a RapidMiner (Mierswa et al. 2006) plug-in for "white-box" component based design of decision tree algorithms for classification and evaluation of these algorithms and their parts. It is intended to be used by typical end users, research scientists and algorithm developers. The main idea of WhiBo is to offer standardized components for algorithm design which will enable simple design and performance testing, easy extension of the component repository and creation of new generic algorithms. Currently, WhiBo provides one generic algorithm, a graphical interface and a component repository for design of decision trees for classification. A framework for performance testing is implemented in WhiBo as well. WhBo plug-in and source code, is available from www.whibo.fon.rs. Source code is documented thoroughly and accessible from the web site through the API documentation. The web site also provides installation guide and number of tutorials for end users, algorithm developers, and research scientists.

Black-box approach

Data mining algorithms are usually implemented in a "black-box" manner. This means that the user defines input data and parameters (if needed) for the algorithm, and the algorithm produces a model. The user has no other possibilities to modify the algorithm to better adjust to data. The "black-box" approach is satisfying for most users. On the other hand, implementation of algorithms as a "black-box" makes it more difficult for algorithm designers who want to use parts of the existing algorithm to create new algorithms. The structure of black box algorithms demands reimplementation of algorithms and their parts from the scratch. "Black-box" implemented algorithms are harder to evaluate and analyze, because it is not clear which part of the algorithm has influence on overall algorithm performance.

White-Box approach

The “white-box” approach allows the user to define parameters, and inputs (as in black-box algorithms) of an algorithm, but also the building blocks (i.e. components) of the algorithm. These components are solutions for typical sub-problems consistently encountered in the process of constructing the appropriate model for the data at hand. This way, algorithmic solution becomes more data and user driven, since it enables the users to intelligently select components of the algorithm which best address the problems of the specific data. Moreover, good ideas from algorithms are saved within components, so they can be used in other algorithms.

White-box approach offers several advances in comparison with black box algorithms (Sonnenburg et al, 2007).

- Combining advantages of various algorithms,
- Comparing algorithms in more details,
- Building on existing resources with less re-implementation,
- Easier “bug” detection on the level of components,
- Collaborative emergence of standards.

WhiBo component repository and Generic decision tree (GDT) algorithm

WhiBo includes a reusable component repository for design of decision tree algorithms. These components were extracted from “black-box” algorithms:

- ID3 (Quinlan JR, 1986),
- C4.5 (Quinlan JR, 1993),
- CART (Breiman et al, 1984),
- CHAID (Kass GV, 1980)

and improvements (distance measure identified in (Mantaras, 1991). Description of analyzed algorithms and partial improvements could be found in Appendix A.

Sub-problems and solutions (reusable components)

In WhiBo algorithms are built by choosing building blocks (i.e. reusable components - RCs) for each sub-problem. The problem of building decision tree model is divided into sub-problems that are generalized algorithm structures with the same input and output structure identified in all analyzed algorithms. Every sub-problem with defined inputs and outputs can be solved in many ways, i.e. with various a reusable components (RCs). That means that every RC solves a specific sub-problem which has the same I/O.

Table 1 shows identified sub-problems and components with their corresponding I/O that are currently implemented in WhiBo.

<table>
<tr><th>Sub-problem</th><th>Reusable component</th><th>Input</th><th>Output</th></tr>
<tr><td>Remove insignificant attributes</td><td>F TEST (numerical attributes)
CHI SQUARE TEST (categorical attributes)</td><td>Dataset in current node</td><td>Dataset in current node (reduced)</td></tr>
<tr><td>Create split (Numerical)</td><td>BINARY</td><td rowspan="4">Dataset in current node</td><td rowspan="4">A split candidate</td></tr>
<tr><td rowspan="3">Create split (Categorical)</td><td>BINARY</td></tr>
<tr><td>MULTIWAY</td></tr>
<tr><td>SIGNIFICANT</td></tr>
<tr><td rowspan="5">Evaluate split</td><td>CHI SQUARE</td><td rowspan="5">A split candidate</td><td rowspan="5">The best split in current node</td></tr>
<tr><td>INFORMATION GAIN</td></tr>
<tr><td>GAIN RATIO</td></tr>
<tr><td>GINI</td></tr>
<tr><td>DISTANCE MEASURE</td></tr>
<tr><td rowspan="2">Stop criteria</td><td>MAXIMAL TREE DEPTH</td><td rowspan="2">Current tree model</td><td rowspan="2">Signal for stopping tree growth in current node</td></tr>
<tr><td>MINIMAL NODE SIZE</td></tr>
<tr><td rowspan="2">Prune tree</td><td>PESSIMISTIC ERROR PRUNING (PEP)</td><td rowspan="2">Current tree model</td><td rowspan="2">Pruned tree model</td></tr>
<tr><td>MIN LEAF SIZE (MLS)</td></tr>
</table>

Table 1 - Sub-problems, reusable components with standardized I/O for Generic decision tree algorithm

Sub-problems and reusable components implemented in Whibo are described according to Tracz (1990) in Appendix B.

Generic decision tree (GDT) structure

The GDT structure proposed in WhiBo is shown on Figure 1. For sub-problems that are bolded it is necessary to define a sub-problem, while for other sub-problems RCs are optional to use. “Create split” (numerical, and categorical) and “Evaluate split” RCs are necessary for decision tree growth. Besides that, there are no restrictions for combinations of RCs.

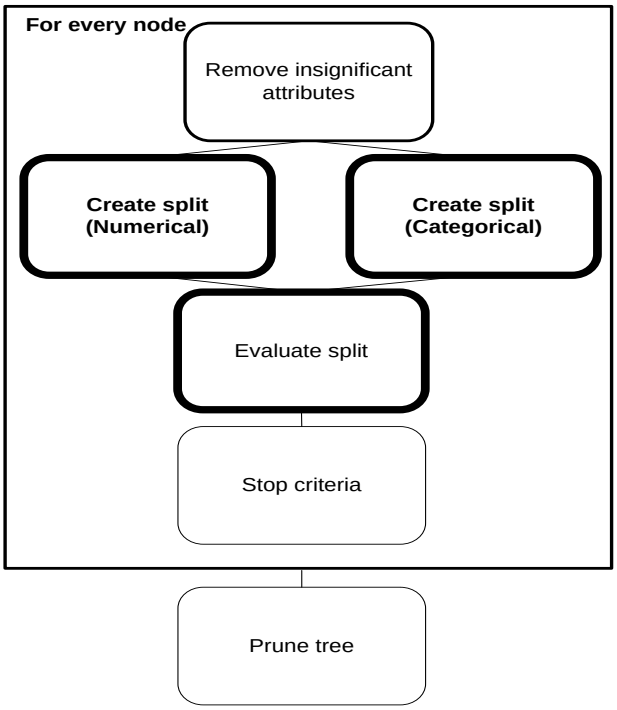


Figure 1 - Generic decision tree (GDT) algorithm

The proposed GDT structure and component repository enables:

- Reconstruction of the original algorithms in the parts that were analyzed.
- Creation of hybrid algorithms with components.
- Extension of the component repository by analyzing new algorithms or partial improvements which can be incorporated in sub-problems with the same input-output structure.
- Definition of new sub-problems which can be incorporated in GDT structure.

WHIBO environment

WhiBo environment currently implements two operator groups:

- Trees – contains *Generic decision tree operator* and *WhiBoGDT Evolutionary Search operator*.
- Validation – contains *Custom cross validation with log* and *Significance 5X2 cross validation F-test operators*.

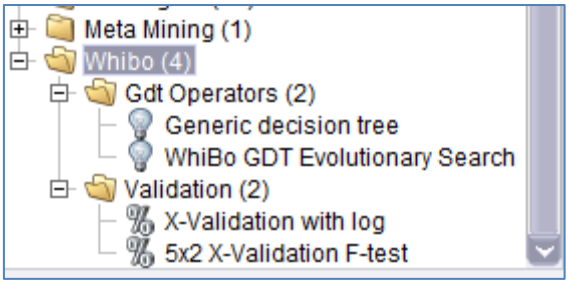


Figure 2 - WhiBo operator group

WHIBO generic decision tree (GDT) operator GUI manual

WhiBo generic decision tree user interface contains four panels:

Left panel contains an array of buttons. Every button represents a concrete sub-problem for a decision-tree algorithm design.

Central panel contains:

- Available RCs of selected sub-problem from the left panel.
- Available parameters (if available) for selected RCs.
- Buttons for including or disabling a RC from the current decision tree structure.

Right panel shows current state of user designed algorithm (saved sub-problems, RCs and parameters).

Top panel contains options for creating new, saving current or opening existing generic decision tree algorithm.

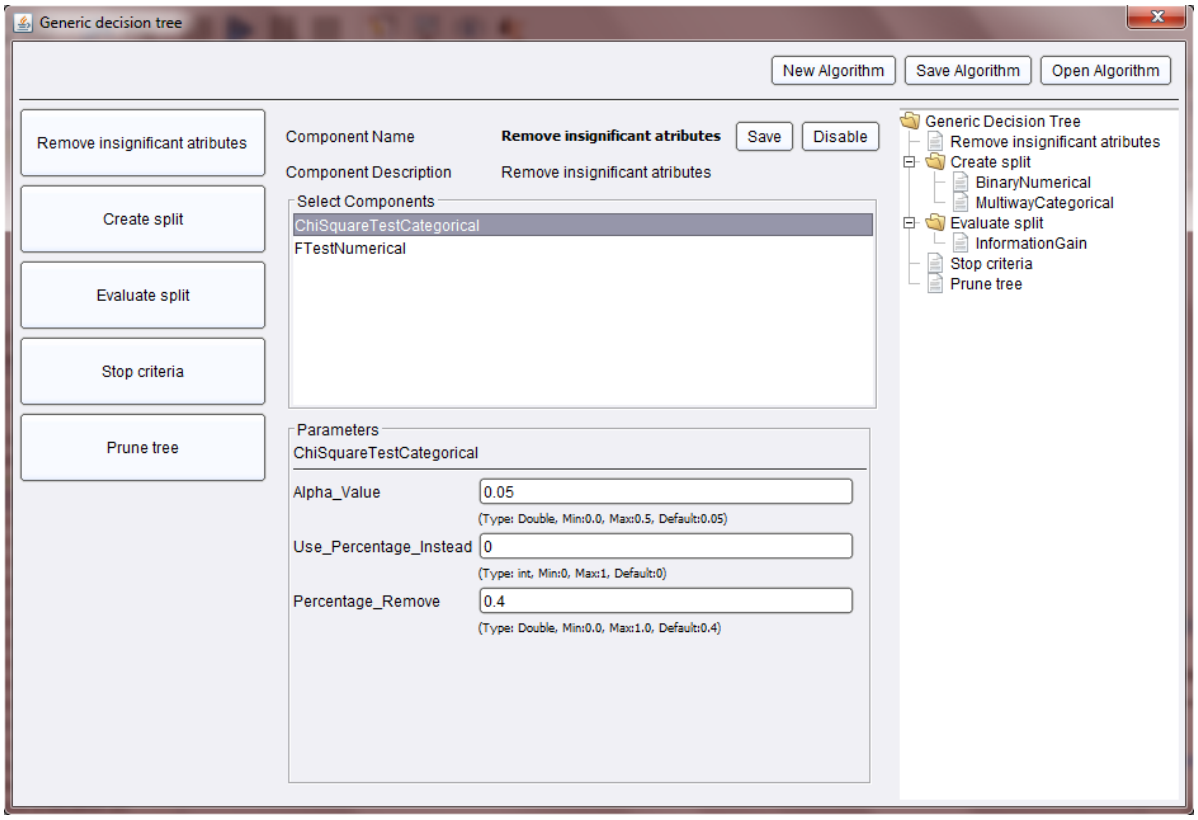


Figure 3 - WhiBo GDT user interface for design of decision tree algorithms

General procedure for designing new algorithms:

- Select sub-problem from left panel. When sub-problem is selected, possible solutions (RCs) are shown in central panel.

- Select RC (or components if multiple) for sub-problem from central panel. If parameters for component(s) are available, they will be shown in bottom part of central panel with their default values.

- Click on save component button. Components and defined parameters for selected sub-problem will be shown in the right panel as part of current GDT algorithm.

- This procedure should be repeated for every sub-problem (Create split and Evaluate split sub-problem are basic for decision tree growth and they must be defined. Definition of other sub-problems is optional). When all sub-problems, components and parameters are defined algorithm should be saved on file system (click on save button from upper panel). By default algorithms are saved with *.wba* (white box algorithm) extension.

WHIBO generic decision tree (GDT) evolutionary search operator GUI manual

WHIBO generic decision tree (GDT) evolutionary search operator implements genetic algorithm which selects reusable components defined in .ass (algorithm search space) file.

Parameters:

- Algorithm search space file location – location of .ass file
- Parameters – list of parameters of genetic algorithm
- Wba file path macro name – macro pointing to .wba file
- Log file path – path where log file will be saved

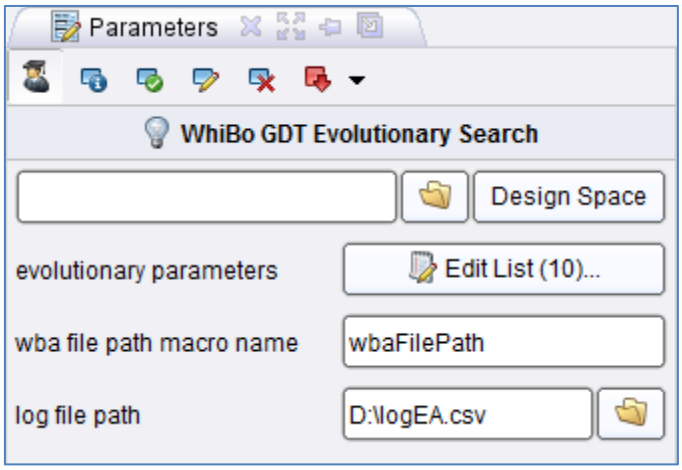


Figure 4 - Parameters panel for *WhiBo GDT Evolutionary Search*

Similarly like in WhiBo generic decision tree user interface contains four panels:

Left panel contains an array of buttons. Every button represents a concrete sub-problem for a decision-tree algorithm design.

Central panel contains:

- Available RCs of selected sub-problem from the left panel.
- Available parameters (if available) for selected RCs.
- Buttons for including or disabling a RC from the current decision tree structure.

Right panel shows current state of user designed algorithm (saved sub-problems, RCs and parameters).

Top panel contains options for creating new, saving current or opening existing generic decision tree algorithm.

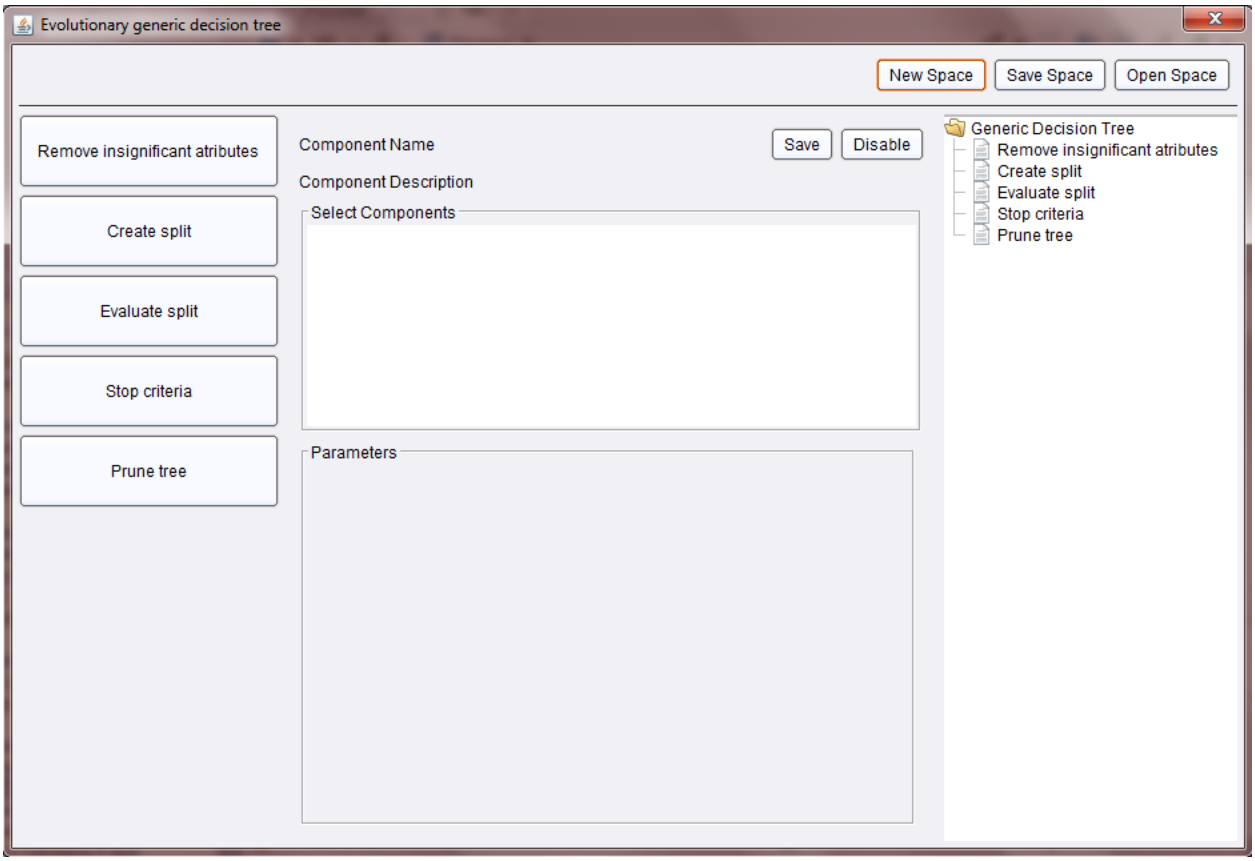


Figure 5 - WhiBo GDT evolutionary search user interface for design of algorithm search space

General procedure for designing algorithm search space:

- Select sub-problem from left panel. When sub-problem is selected, possible solutions (RCs) are shown in central panel.

- Select RC (or components if multiple) for sub-problem from central panel. If parameters for component(s) are available, they will be shown in bottom part of central panel with lower and upper values selected. User can modify these values.

- Click on save component button. Components and defined parameters for selected sub-problem will be shown in the right panel as part of current GDT algorithm.

- This procedure should be repeated for every sub-problem (Create split and Evaluate split sub-problem are basic for decision tree growth and they must be defined. Definition of other sub-problems is optional). When all sub-problems, components and parameters are defined

algorithm should be saved on file system (click on save button from upper panel). By default algorithms are saved with *.ass* (algorithm search space) extension.

After definition of algorithm search space parameters for genetic algorithm should be defined.

Parameters:

- MAX_ALLOWED_EVOLUTIONS – maximal numbers of generations of genetic algorithms (default value - 50).
- POPULATION_SIZE – number of units (decision trees) in one generation (default value - 30).
- MUTATION_RATE – percentage of genes (components) will be changed (default value - 6).
- CROSSOVER_RATE – rate of crossover of chromosomes in genetic algorithm (default value – 0.35)
- SWITCH_FROM_SURROGATE_PERCENTAGE_EVOLUTIONS – defines how many units should be removed from previous generation (default value – 0.4)
- SURROGATE_PERCENTAGE – defines how many units should be selected from previous generation (default value – 0.4)
- mutateComponents – boolean value indicating weather reusable components should be mutated (default value - true).
- mutateParameters – boolean value indicating weather parameters should be mutated (default value - true).
- componentsMutationRate – mutation rate of components (default value - 1).
- parametersMutationRate – mutation rate of parameters (default value - 1).

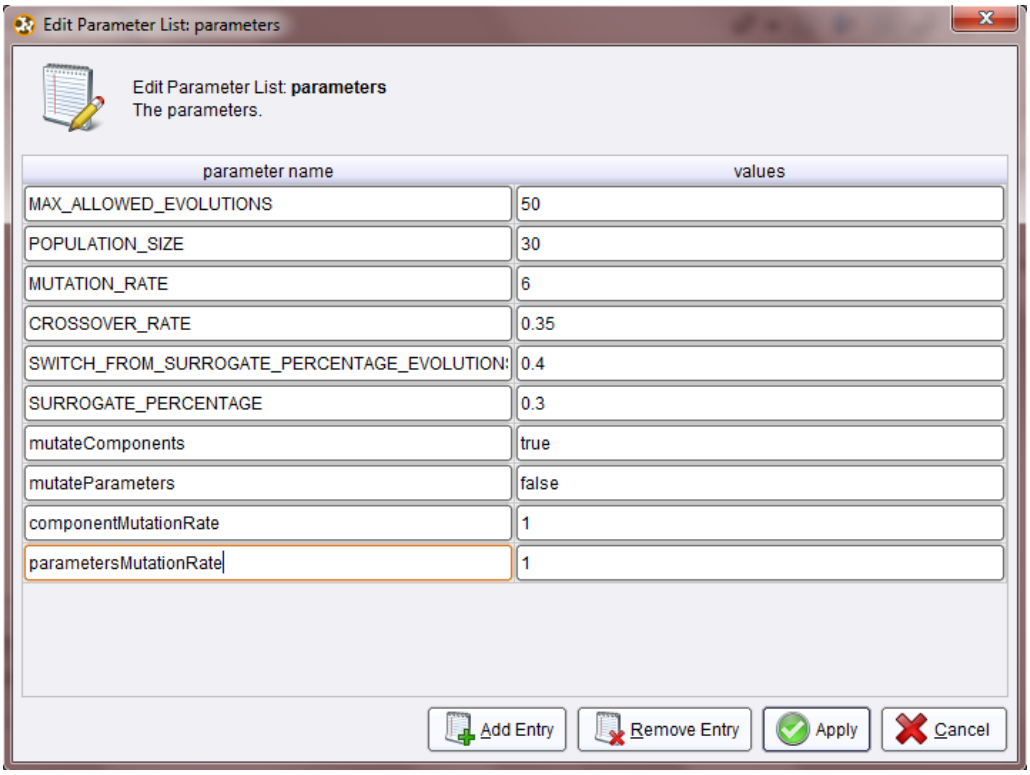


Figure 6 - Parameters of genetic algorithm

WHIBO testing environment manual

WhiBo provides operators for testing performance and significance of differences in algorithm performance.

Custom cross validation with log - implements cross validation with custom defined number of folds and number of iterations and also enables writing results in log in CSV format. The results are written in average, but also for every fold and iteration. This operator writes accuracy of classifier, but also: Maximum tree depth, weighted average tree depth, Total nodes, Total leaves, and Execution time.

Parameters:

- *Average_performances_only* – check if there is no need for logging the results for every fold and iteration.
- *Algorithm_name* – name of the algorithm.
- *Dataset_name* – name of the dataset.
- *Number_of_folds* – number of folds for cross-validation.
- *Number_of_repetitions* – number of repetitions for cross-validation.
- *Sampling_type* – stratified sampling, linear sampling or shuffled sampling.
- *Log_file_details* – file path for logging detailed results.
- *Log_file_averages* - file path for logging average results.

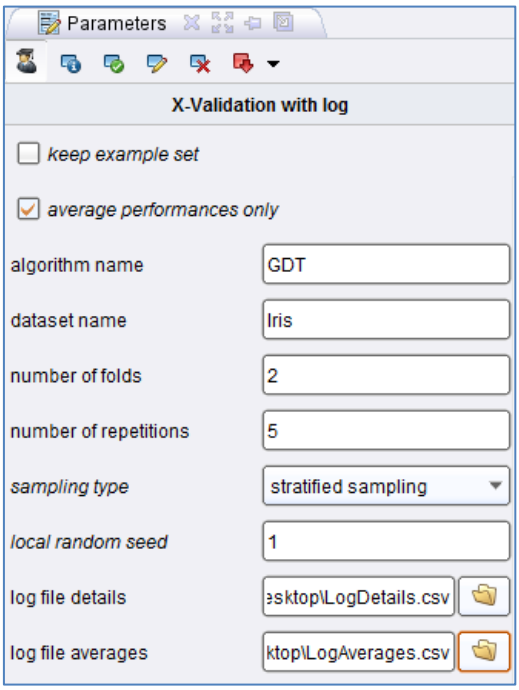


Figure 7 - *Custom cross validation with log* operator with parameters

Significance 5X2 cross validation F-test – This is the best significance tester for classifiers according to (Salzberg, 1999). The 5x2 cross validation F-test (Alpaydin E. (1999)) is testing significance of differences in algorithm performance.

Parameters:

- Alpha – significance parameter (Default value – 0.05).
- Local random seed – number used for initialization of pseudorandom number generator.
- *Sampling_type* – stratified sampling, linear sampling or shuffled sampling.

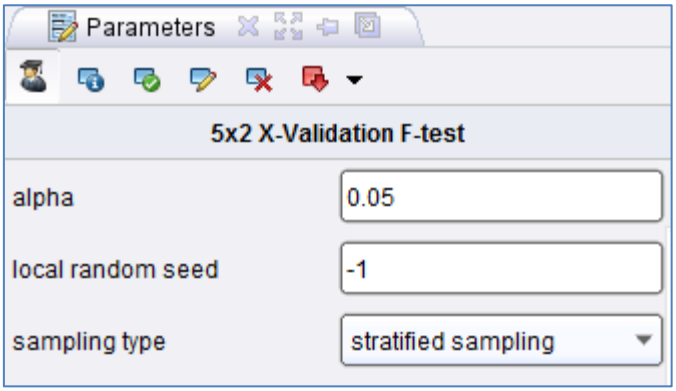


Figure 8 - *Significance 5X2cv F-test* operator with parameters

Application examples

WhiBo GDT is implemented as RapidMiner operator. WhiBo decision tree operators require *ExampleSet* as input and produce *TreeModel* and *ExampleSet* on output, so they are compatible with all Rapid miner's evaluation and visualization operators.

For these examples we use "Iris" dataset from UCI repository as a data source (definition of data source can be done through RapidMiner's sample data repository).

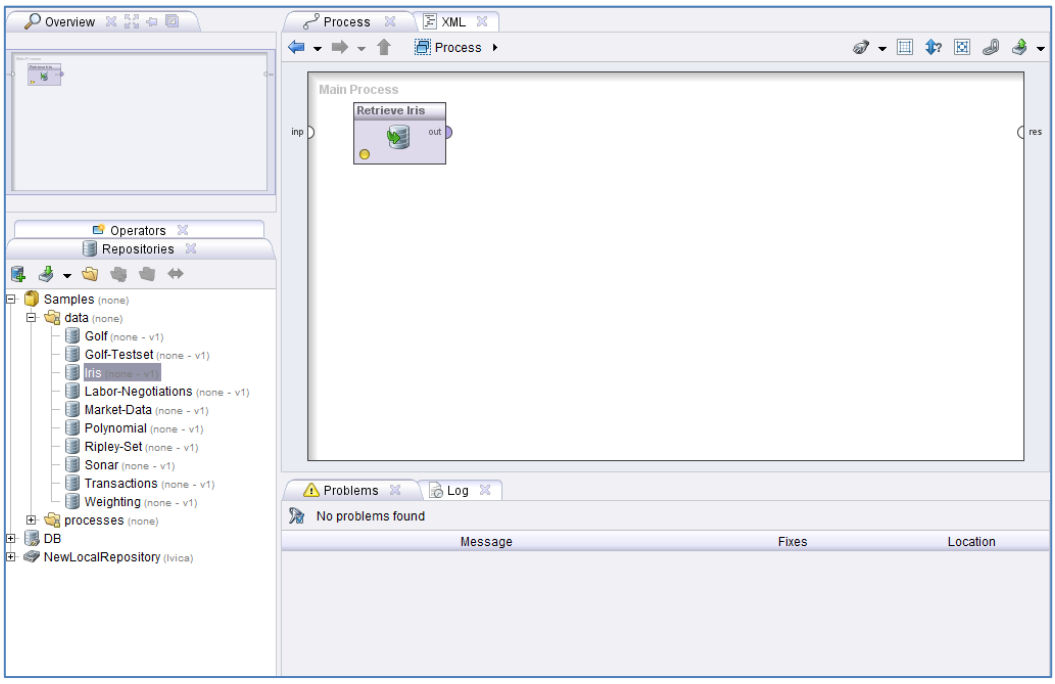


Figure 9 - Basic definition of RapidMiner process

On the lower left side of the screen local repository can be seen. From there, "Iris" dataset was dragged to Main Process panel. With that step input *ExampleSet* is defined.

White-box component based design and application

Using WhiBo GDT with RapidMiner will be explained on examples of creating well-known algorithms, modifying these algorithms and designing new algorithms.

When *ExampleSet* is defined, add GDT operator to root process. GDT can be found in *WhiBo/GDT Operators* operator group.

Figure 10 - Adding GDT operator into stream

When the example source is defined and Generic Tree operator is added in process, new generic decision tree can be designed, by clicking on Design new algorithm button.

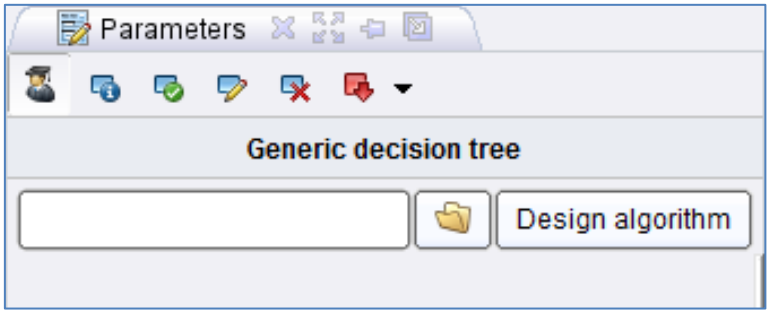


Figure 11 - Parameters panel for GDT operator

Recreation of well-known algorithms with component based approach

Application of white-box approach will be first explained on recreation of well-known algorithms.

CART algorithm

First, define Create split sub-problem:

- Click on Create split sub-problem on the left panel.
- Select *BinaryNumerical* and *BinaryCategorical* components from central panel (multiple components for one sub-problem are selected by holding CTRL key and clicking on components).
- Click on save component button from central panel.

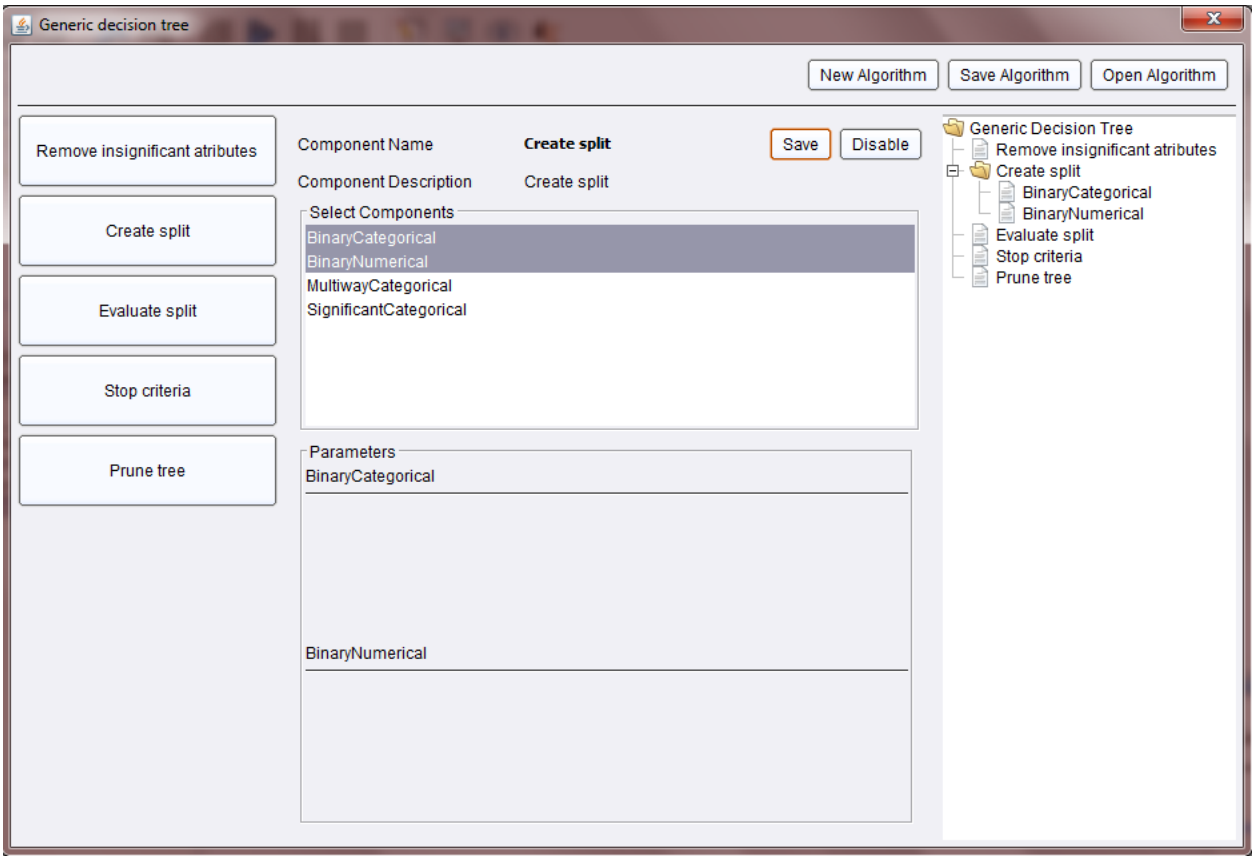


Figure 12 - Definition of *Create split* sub-problem for CART algorithm

On the right panel defined components for a sub-problem is visualized through a Tree view (Figure above).

Next step is definition of evaluate split sub-problem.

- Click on *Evaluate split* sub-problem.
- Select *Gini index* component.
- Click on save component button.

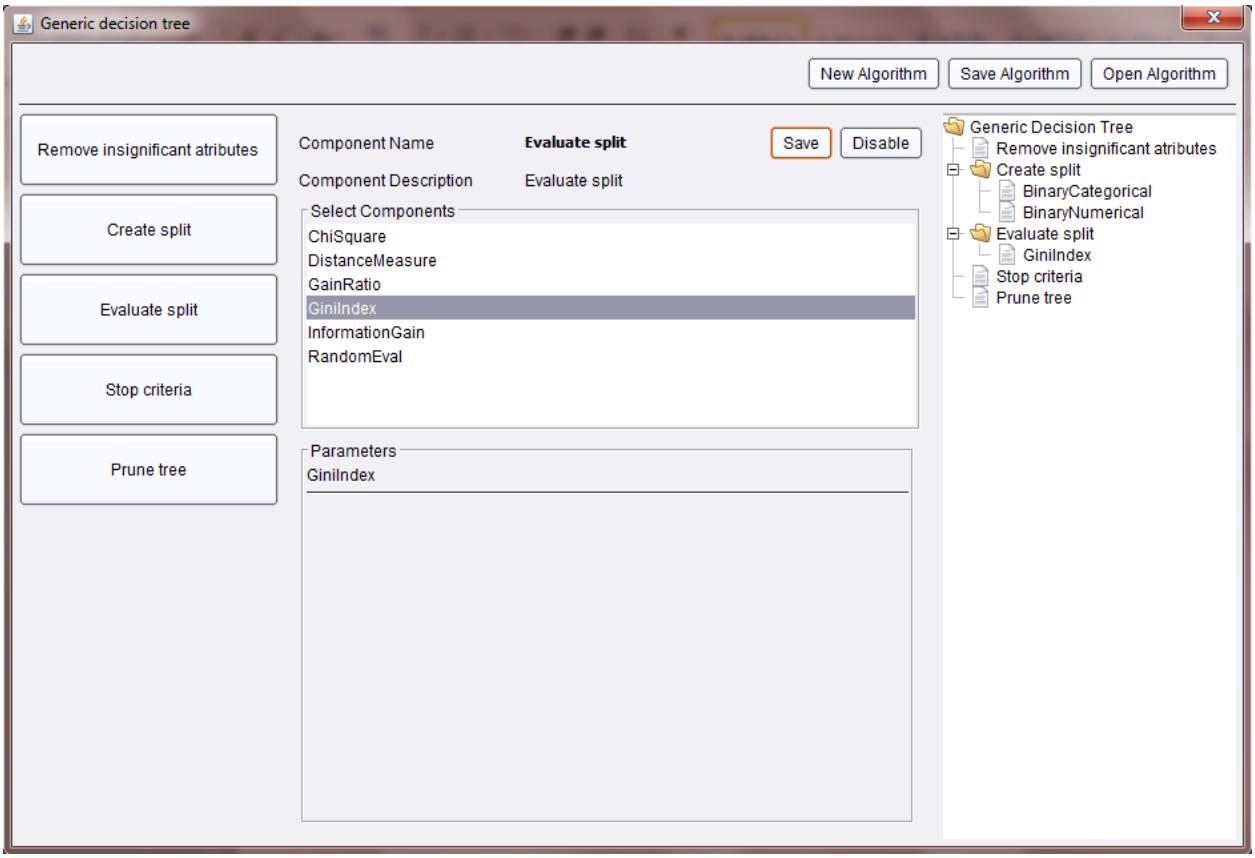


Figure 13 - Definition of *Evaluate split* sub-problem for CART algorithm

Now, the basic components for CART algorithm are defined. Before saving the algorithm we will define Stop criteria sub-problem:

- Click on *Stop criteria* sub-problem.
- Select *Tree depth* component
- Set *Tree_Depth* parameter on 5 (default value is 10).
- Click on save component button.

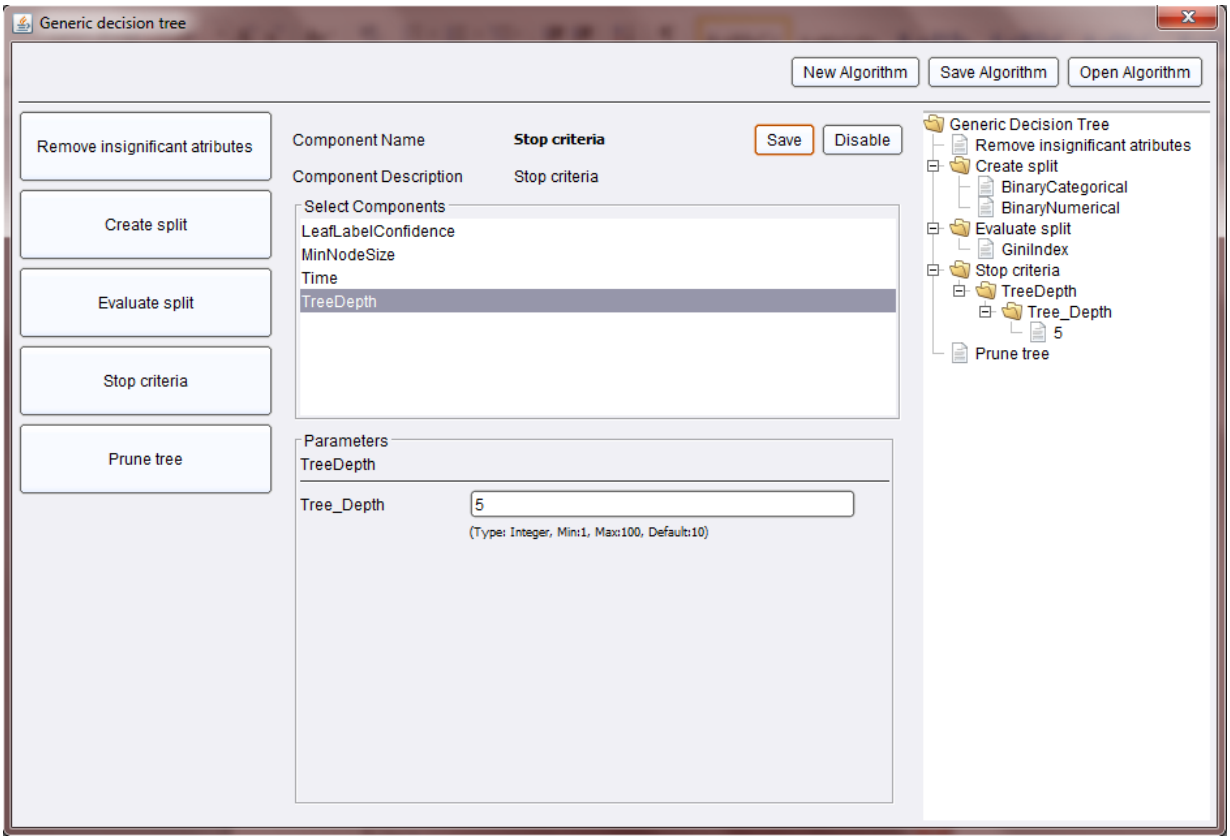


Figure 14 - Definition of *Stop criteria* sub-problem for CART algorithm

Finally Cart algorithm with tree depth stopping criteria is defined and can be saved on file system.
Click on Save algorithm button from upper panel.

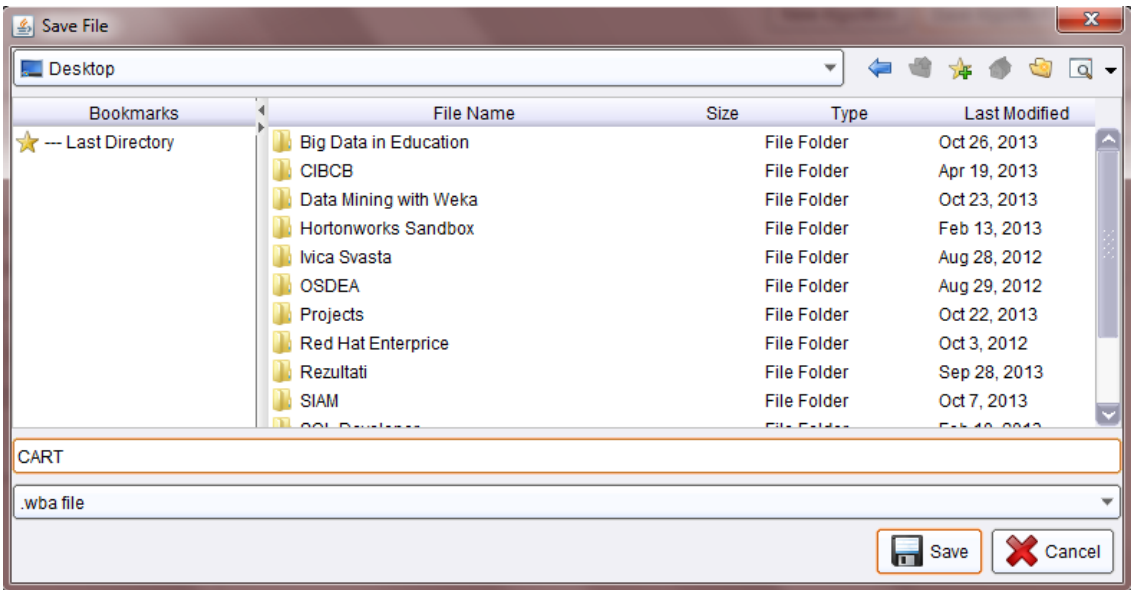


Figure 15 - Saving CART algorithm on file system

After saving algorithm it must be loaded in GDT GUI clicking on folder button in parameters panel.

Figure 16 - Loading CART algorithm in GDT operator

When stream is executed, graphic a text tree model will be shown.

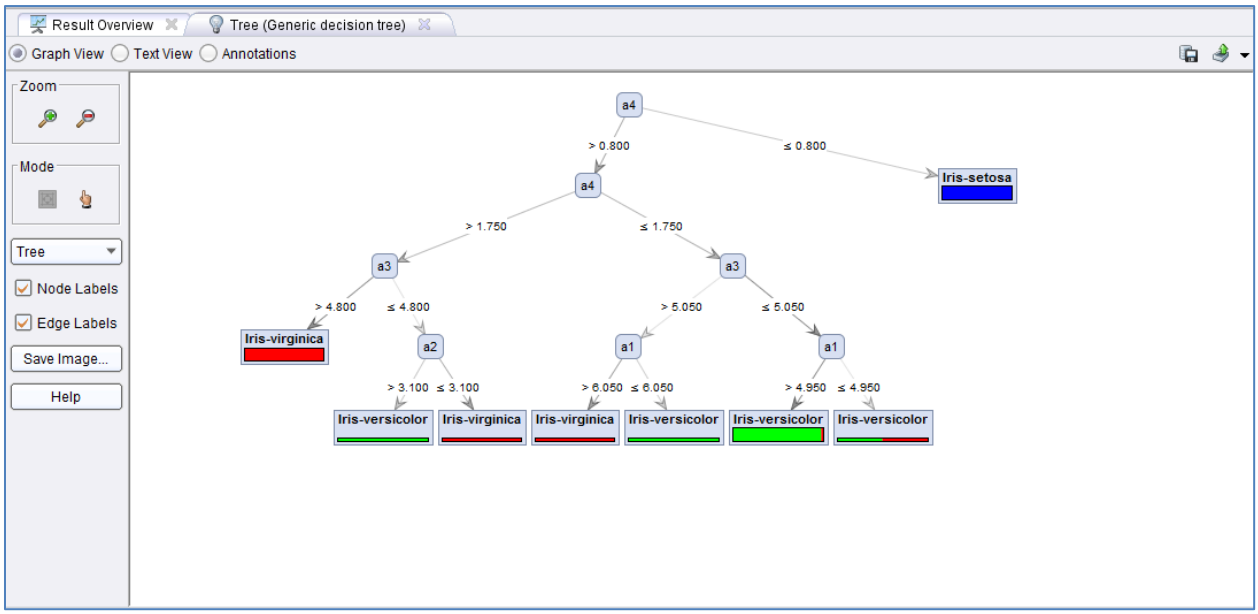


Figure 17 - Result of executed CART algorithm on Iris dataset

C4.5 algorithm

First, define Create split sub-problem:

- Click on Create split sub-problem on the left panel.
- Select *BinaryNumerical* and *MultiwayCategorical* components from central panel (multiple components for one sub-problem are selected by holding CTRL key and clicking on components)
- Click on save component button from central panel

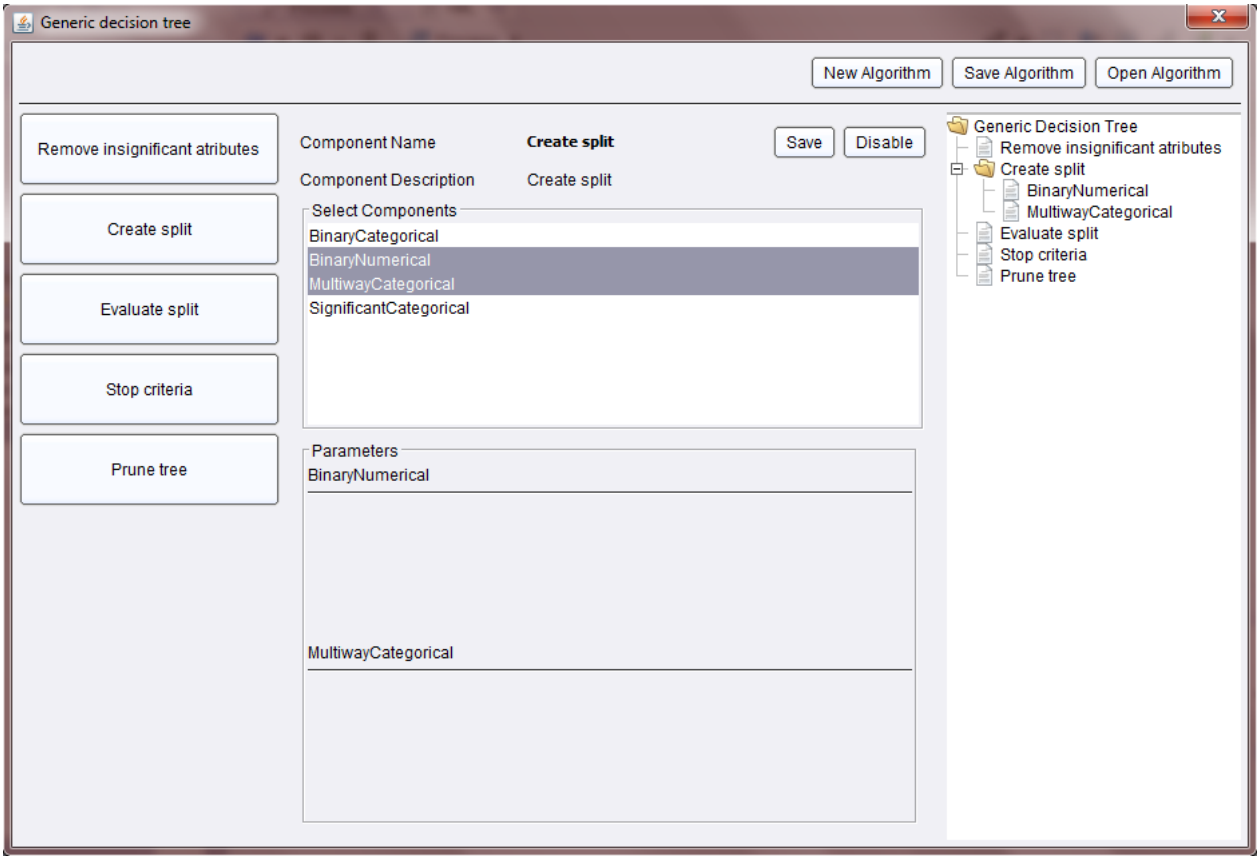


Figure 18 - Definition of *Create split* sub-problem for C4.5 algorithm

Second, define Create split sub-problem:

- Click on *Evaluate split* sub-problem on the left panel.
- Select *GainRatio* component from central panel.
- Click on save component button from central panel.

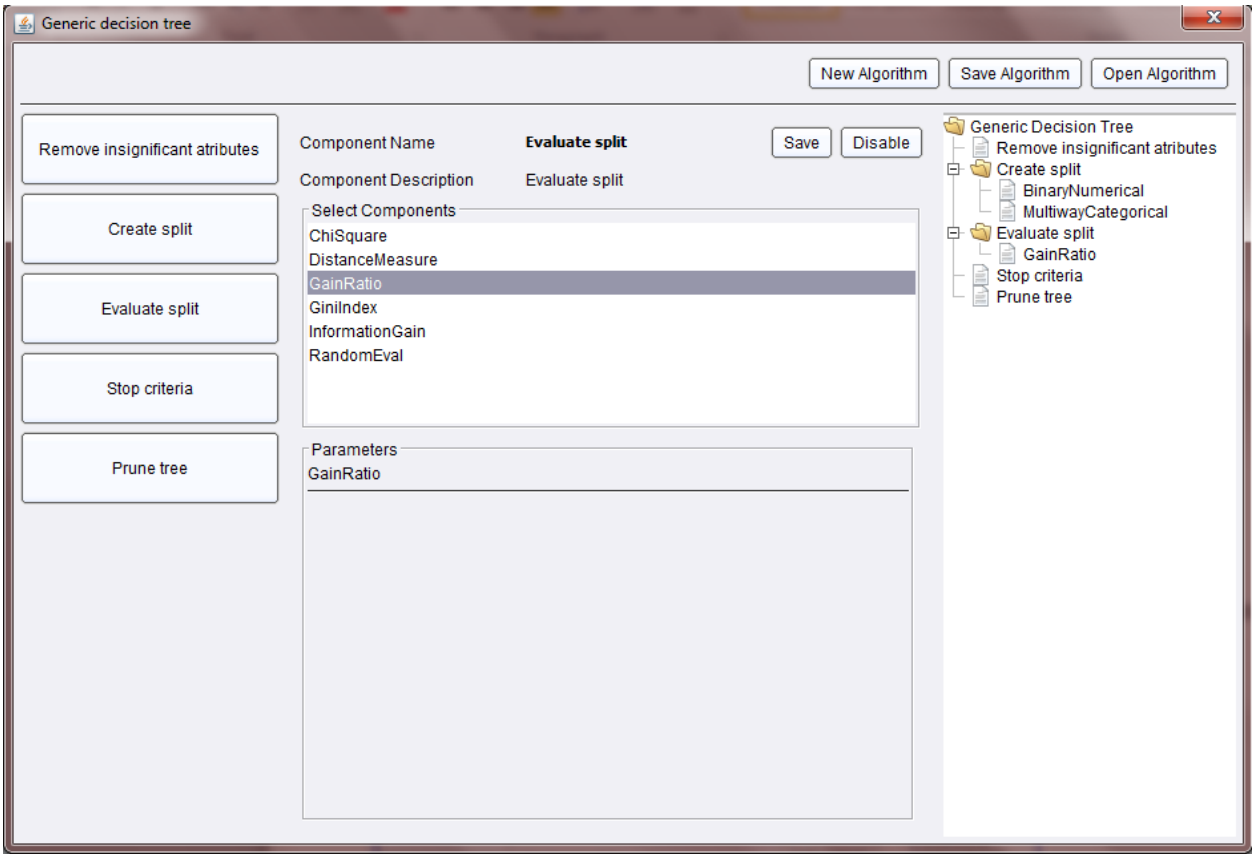


Figure 19 - Definition of *Evaluate split* sub-problem for C4.5 algorithm

Third, define *Stop criteria* sub-problem:

- Click on *Stop criteria* sub-problem on the left panel.
- Select *TreeDepth* component from central panel.
- Set *Tree_Depth* parameter for example on 10.
- Click on save component button from central panel.

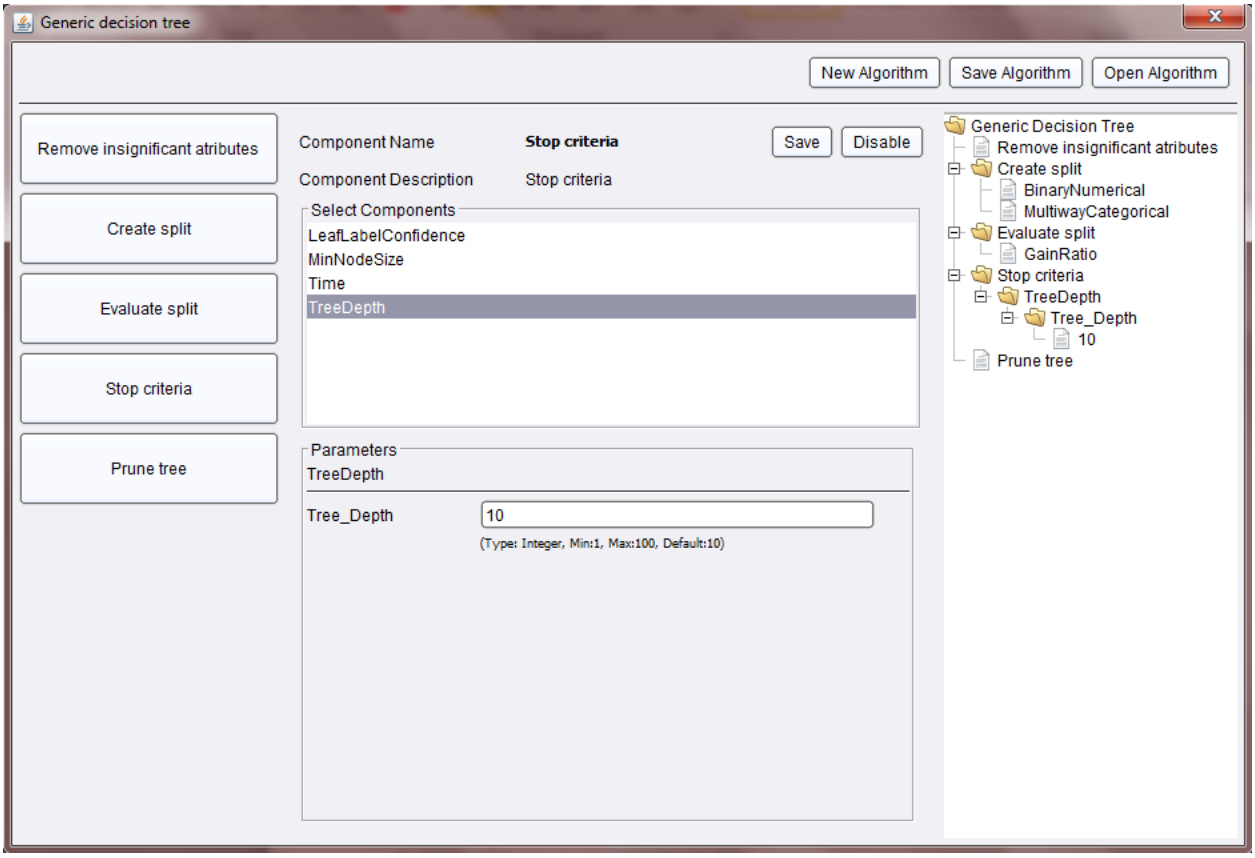


Figure 20 - Definition of *Stop criteria* sub-problem for C4.5 algorithm

Fourth, define *Prune tree* sub-problem:

- Click on *Prune tree* sub-problem on the left panel.
- Select *PessimisticError* component from central panel.
- Set *Confidence_Level* parameter on 0.2.
- Click on save component button from central panel.

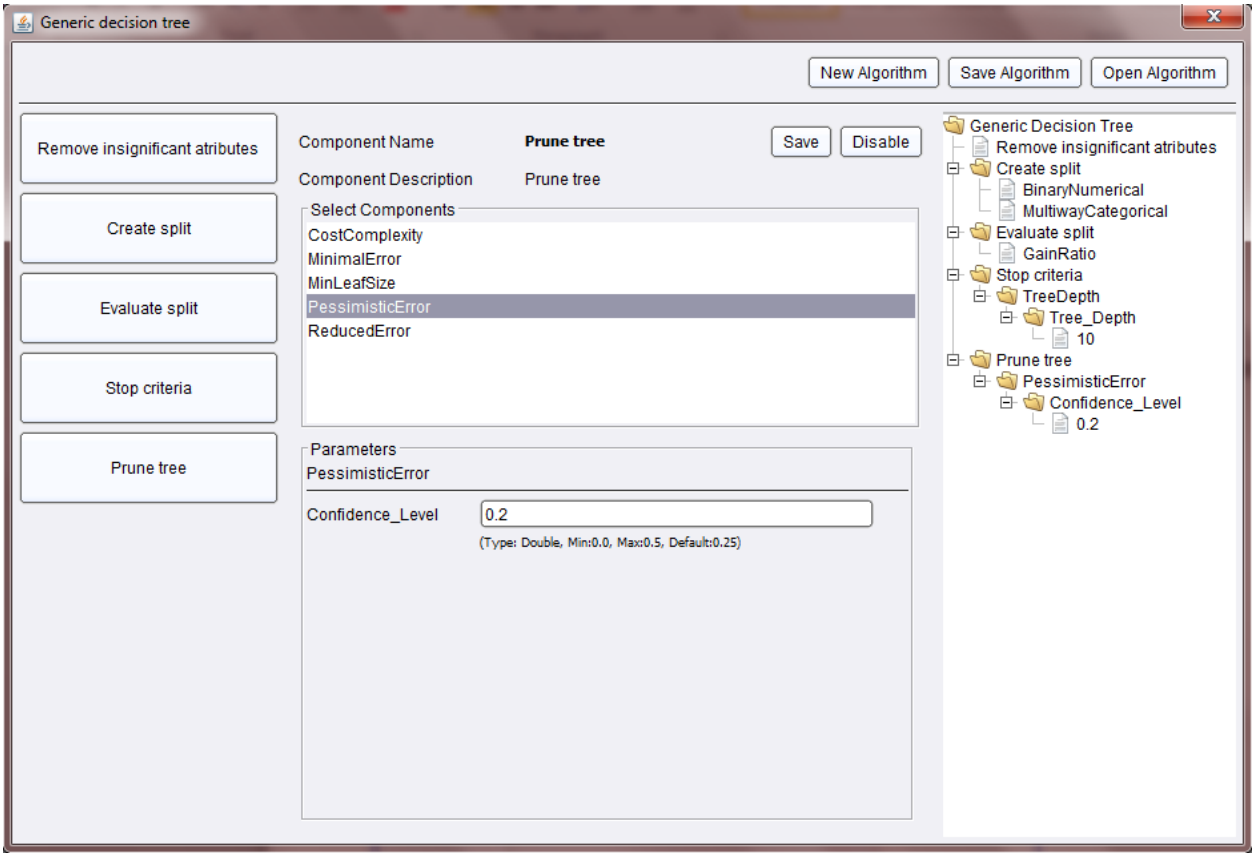


Figure 21 - Definition of *Prune tree* sub-problem for C4.5 algorithm

Finally, C4.5 algorithm is defined and can be saved on file system. Click on Save algorithm button from upper panel. When algorithm is defined, load it from file system and execute stream.

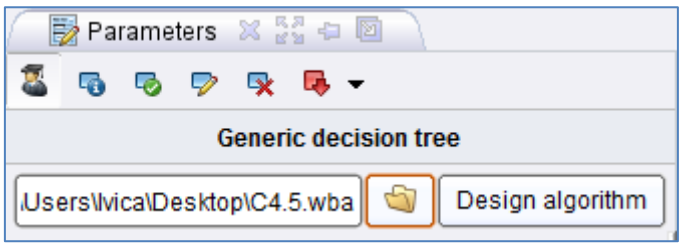


Figure 22 - Loading C4.5 algorithm in GDT operator

When stream is executed, graphic a text tree model will be shown.

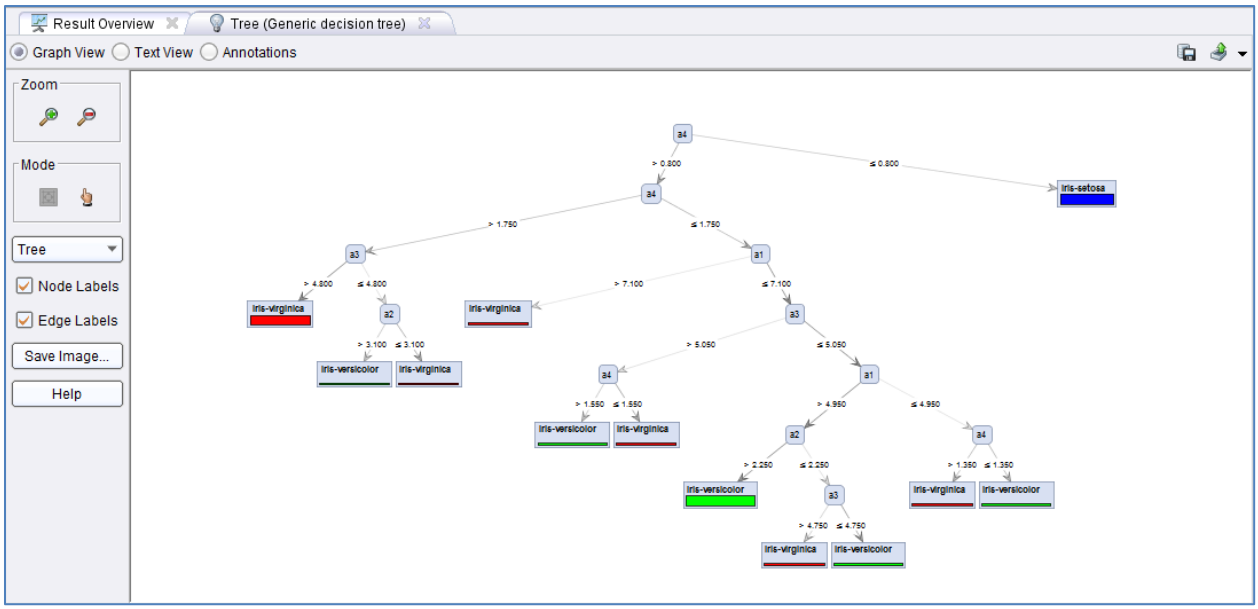


Figure 23 - Result of executed C4.5 algorithm on Iris dataset

CHAID algorithm

First, define Create split sub-problem:

- Click on Create split sub-problem on the left panel.
- Select *BinaryNumerical* and *SignificantCategorical* components from central panel (multiple components for one sub-problem are selected by holding CTRL key and clicking on components)
- Set default parameters *Merge_Alpha_Value* and *Split_Alpha_Value* for *SignificantCategoricalComponent*.
- Click on Save component button from central panel.

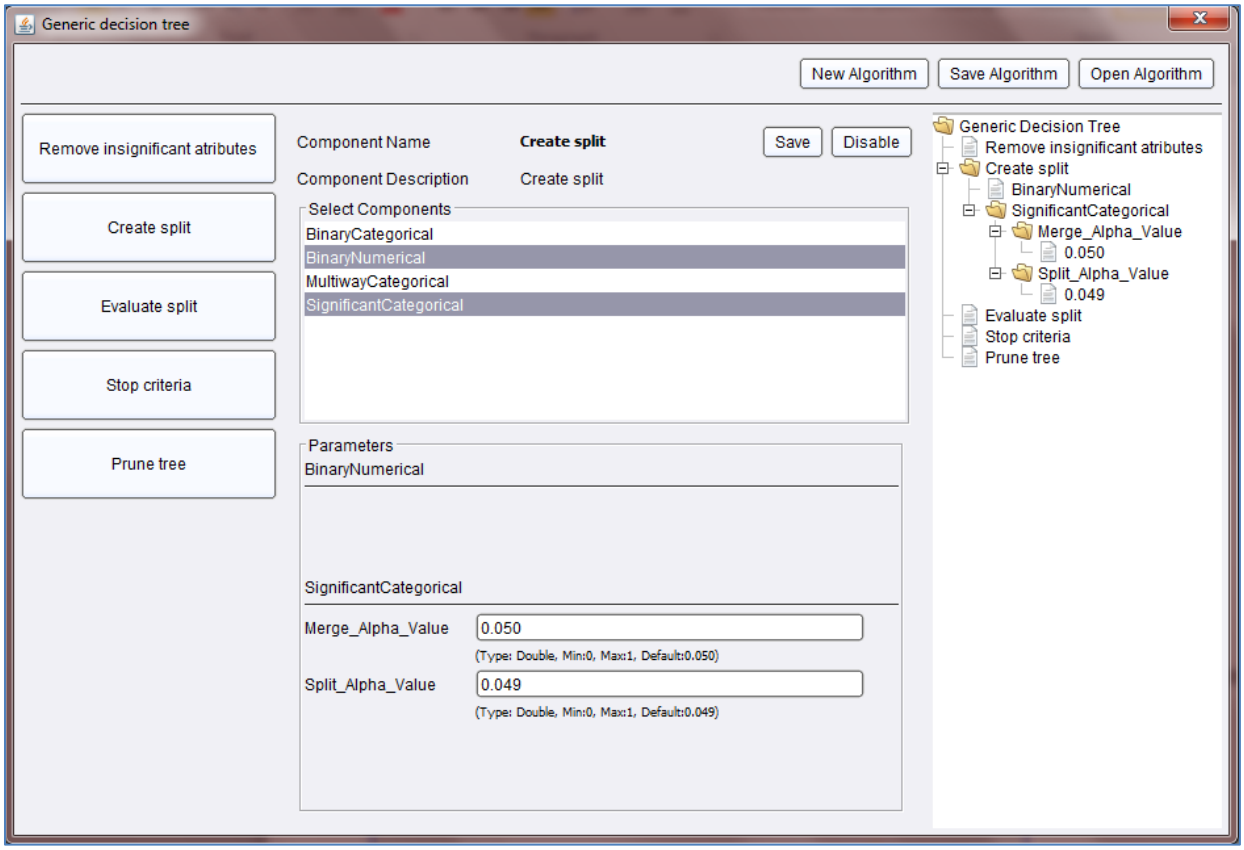


Figure 24 - Definition of *Create split* sub-problem for CHAID algorithm

Second, define Evaluate split sub-problem:

- Click on *Evaluate split* sub-problem on the left panel.
- Select *ChiSquare* component from central panel.
- Click on save component button from central panel.

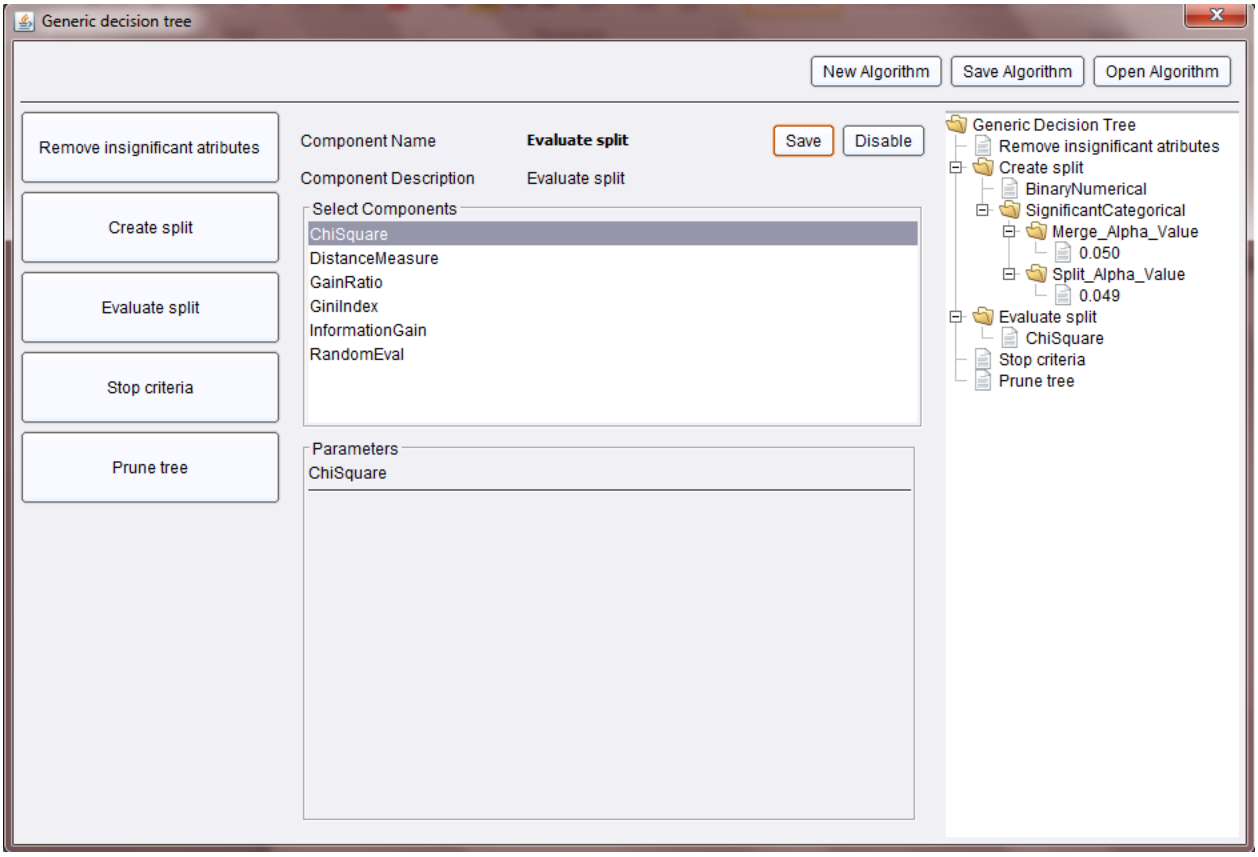


Figure 25 - Definition of *Evaluate split* sub-problem for CHAID algorithm

Third, define *Stop criteria* sub-problem:

- Click on *Stop criteria* sub-problem on the left panel.
- Select *TreeDepth* component from central panel.
- Set *Tree_Depth* parameter for example on 5.
- Click on save component button from central panel.

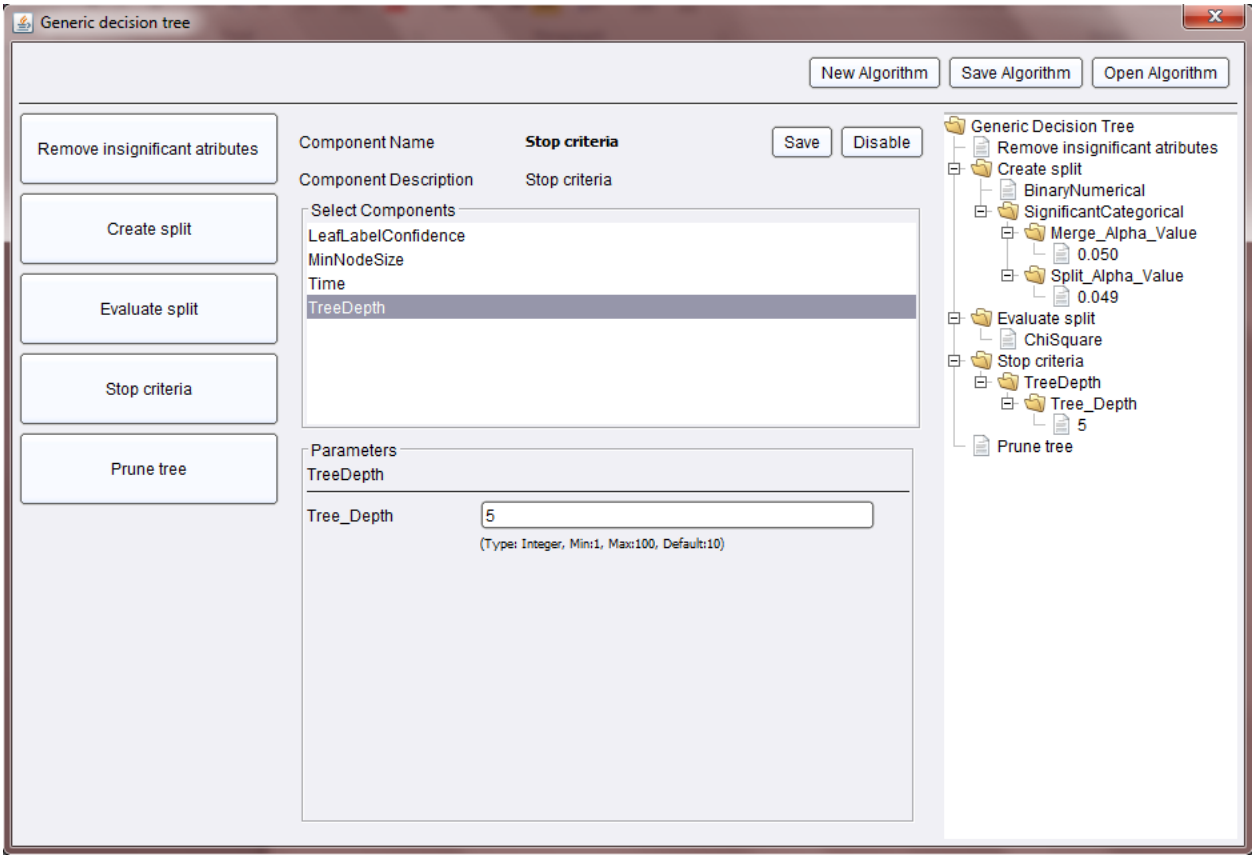


Figure 26 - Definition of *Stop criteria* sub-problem for CHAID algorithm

Fourth, define *Prune tree* sub-problem:

- Click on *Prune tree* sub-problem on the left panel.
- Select *PessimisticError* component from central panel.
- Set *Confidence_Level* parameter for example on 0.2.
- Click on save component button from central panel.

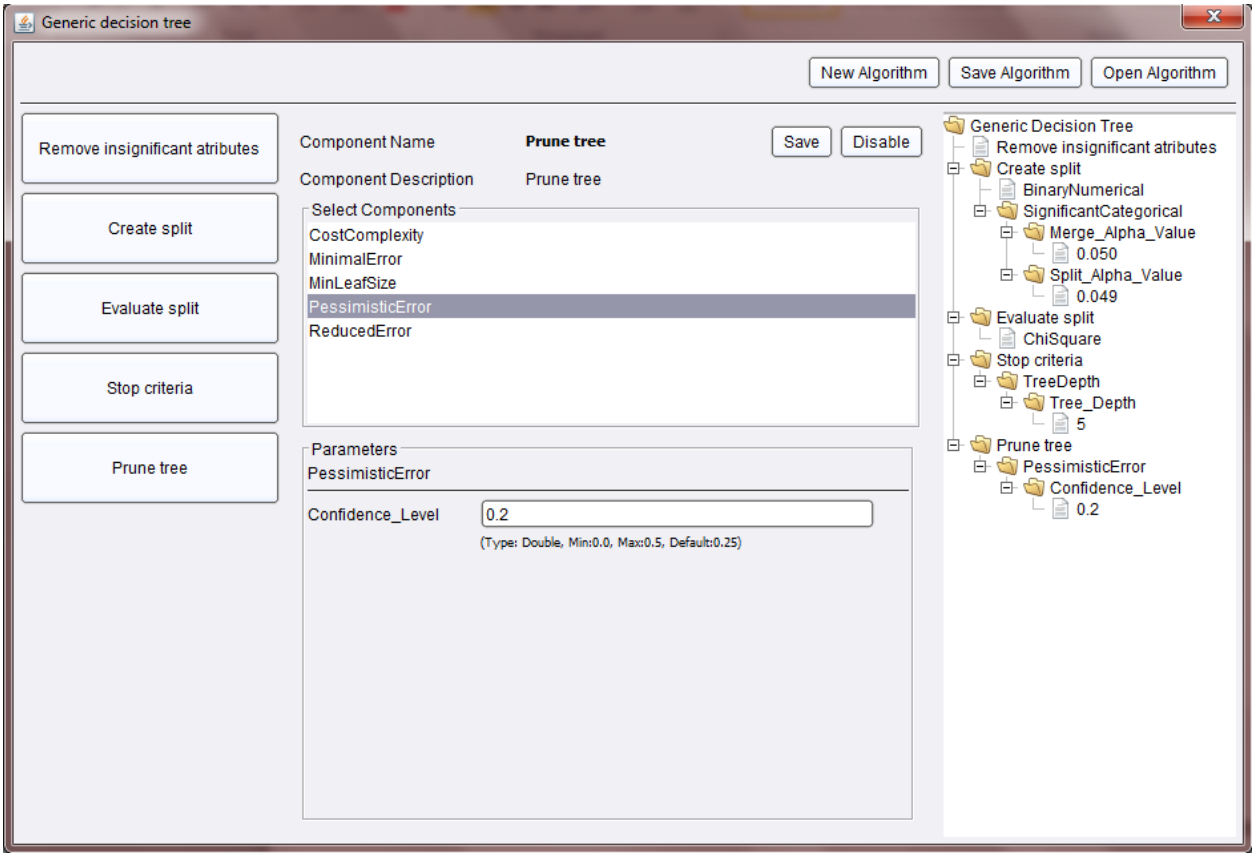


Figure 27 - Definition of *Prune tree* sub-problem for CHAID algorithm

Finally, CHAID algorithm is defined and can be saved on file system. Click on Save algorithm button from upper panel. When algorithm is defined, load it from file system and execute stream.

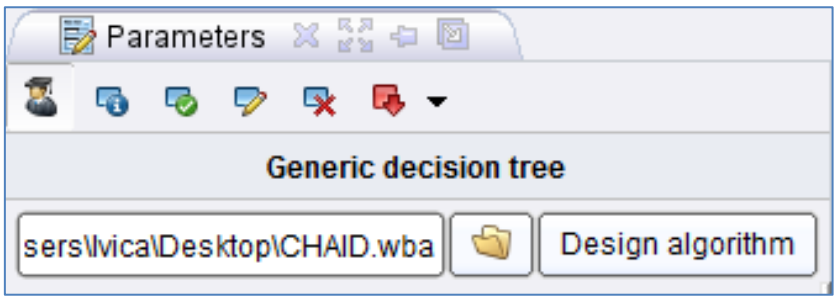


Figure 28 - Loading CHAID algorithm in GDT operator

When stream is executed, graphic a text tree model will be shown.

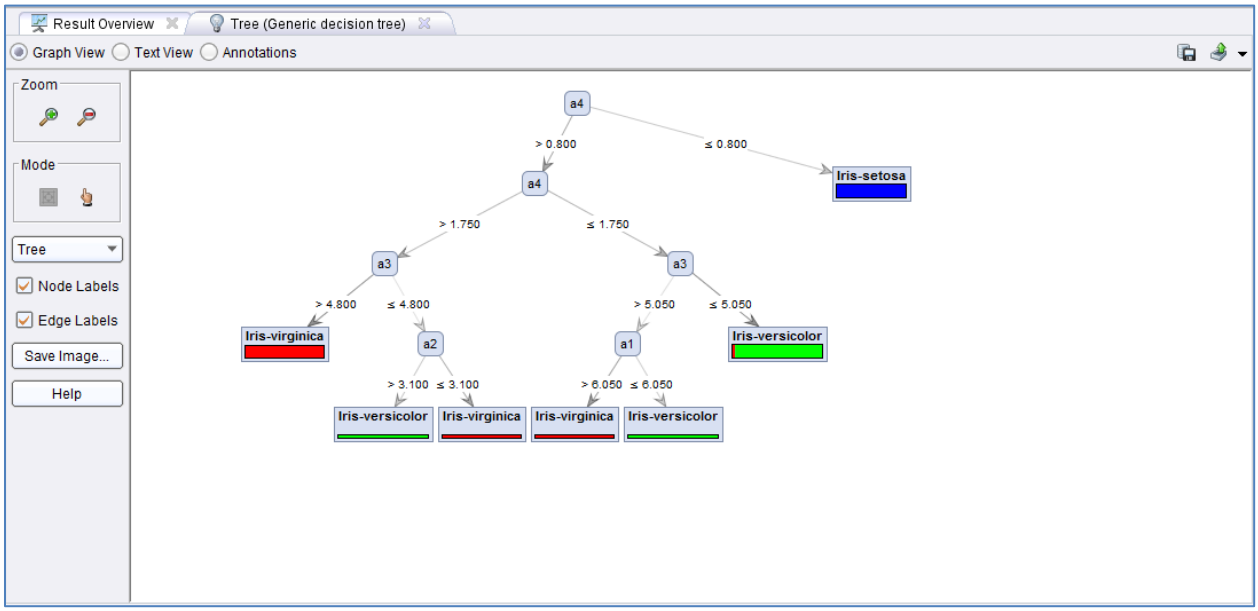


Figure 29 - Result of executed CHAID algorithm on Iris dataset

Modifying generic decision tree algorithms

Algorithms created through WhiBo interface could be easily modified by parameters, sub-problems or components. WhiBo algorithms are saved on a file system by .wba (WhiBo algorithm) extension. Existing algorithms can be loaded for editing by clicking on *Open algorithm* button from the top panel of WhiBo interface.

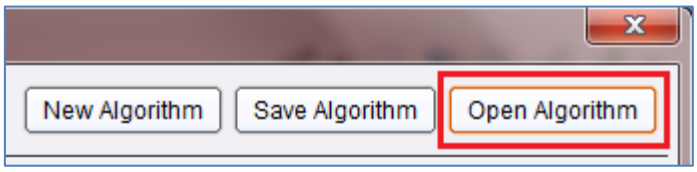


Figure 30 - Opening existing algorithm for modification

In this section it will be explained modifying of CHAID algorithm that was created and saved in previous example.

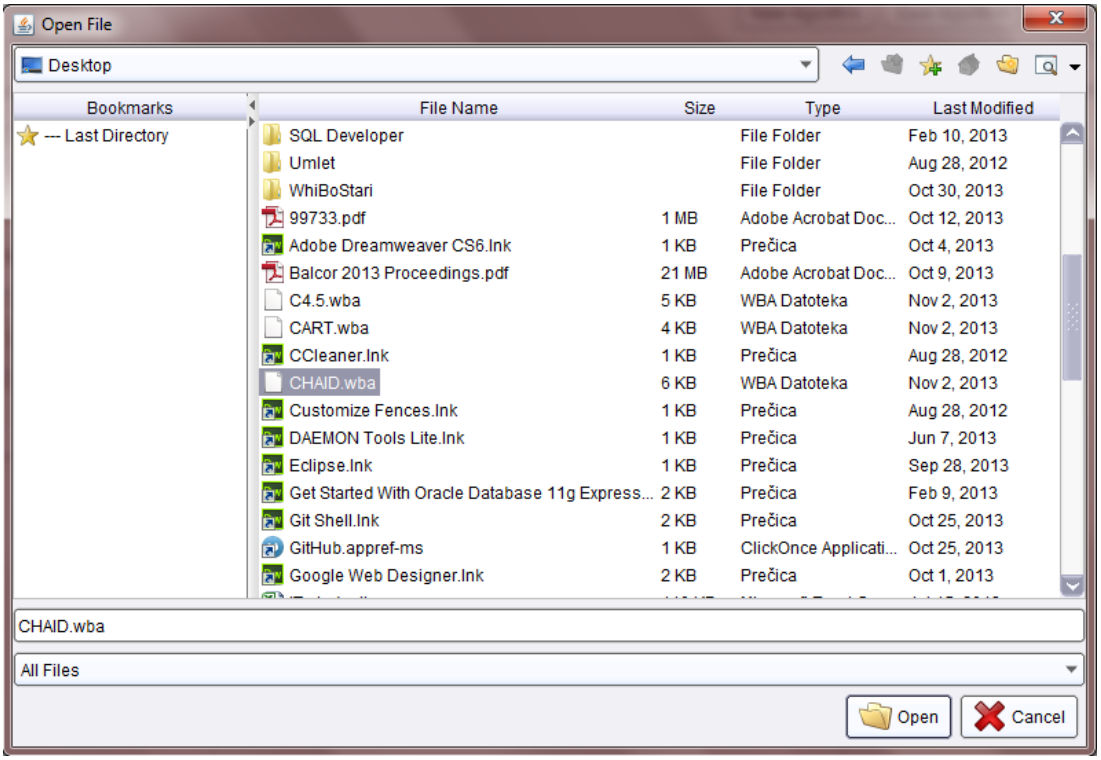


Figure 31 - Selecting existing algorithm from file system

When existing algorithm is opened, its sub-problems and RCs are shown in a tree view on the right panel of WhiBo interface.

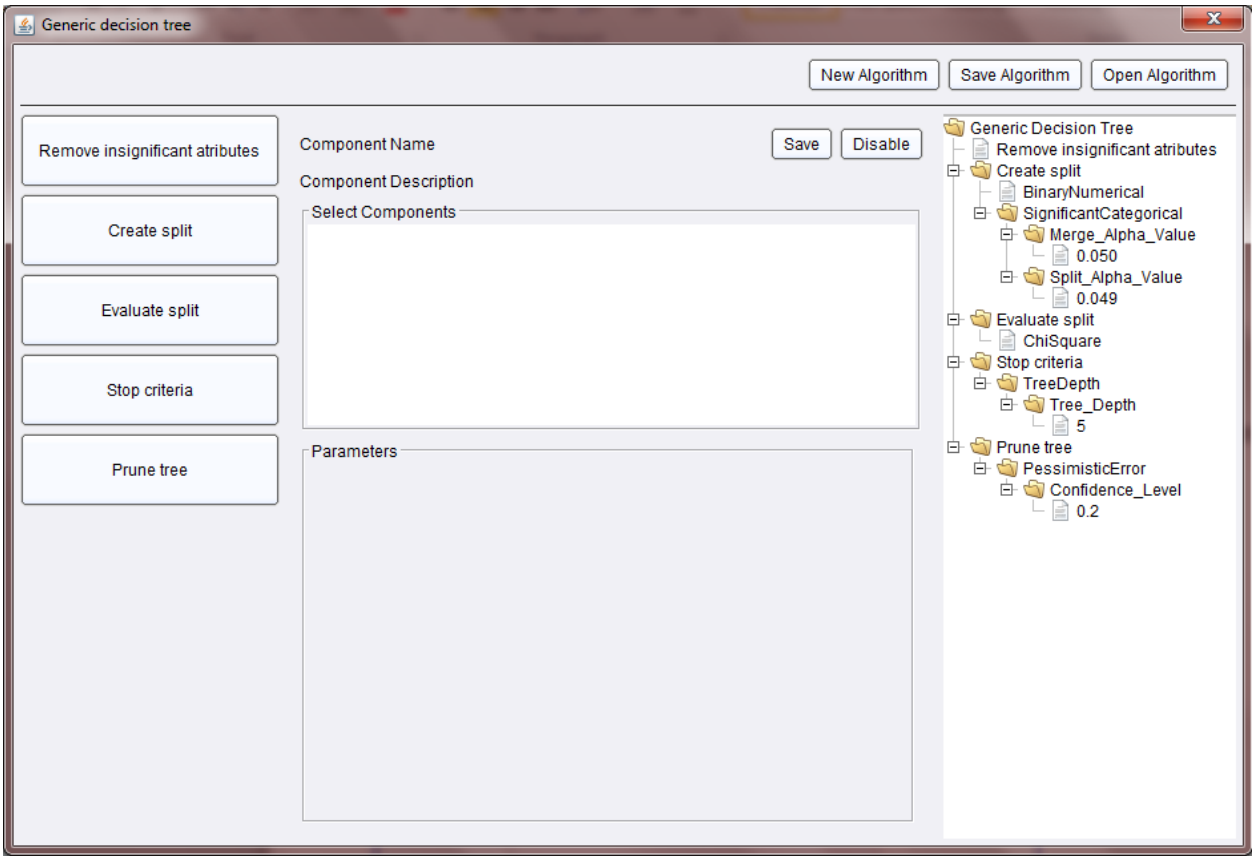


Figure 32 - Loaded algorithm in GDT GUI

Modifying of existing algorithms is done the same way as creating of new algorithms. In this example we will show how to

- modify parameter of already defined component,
- change component of defined sub-problem,
- adding new sub-problem and component for its solution.

Modifying parameters

In this example we will modify Merge_Alpha_Value and Split_Alpha_Value parameters of *Create split – SignificantCategorical* component and *Tree_Depth* parameter of *Stop criteria - TreeDepth*:

- Click on *Create split* sub-problem on the left panel.
- Set *Merge_Alpha_Value* on 0.03.
- Set *Split_Alpha_Value* on 0.02.
- Click on save component button from central panel.
- Click on *StopCriteria* sub-problem on the left panel.
- Set *Tree_Depth* parameter on 4.
- Save algorithm as *CHAIDModifiedParameters* by clicking on *Save algorithm* button from top panel.

Result from executed algorithm is shown on figure below.

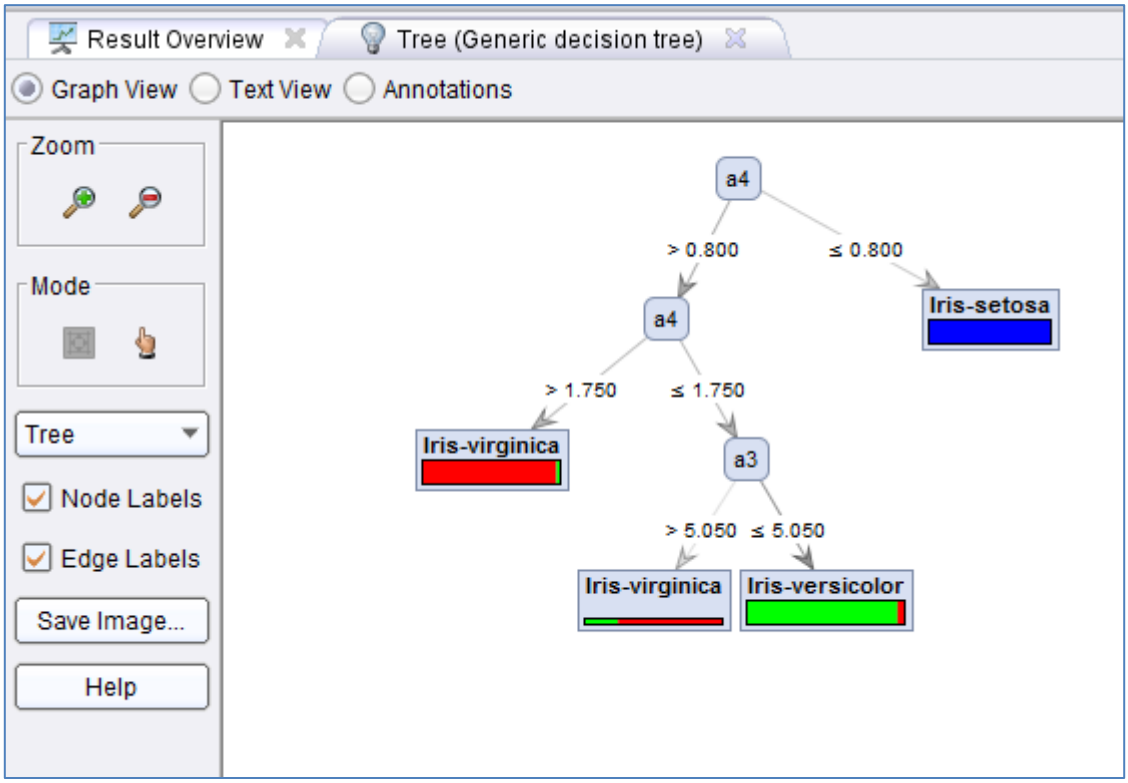


Figure 33 - Loaded algorithm in GDT GUI

Replacing components for sub-problem

In this example we will replace Evaluation measure component in CHAIDModifiedParameters algorithm that is created in previous subsection.

- Click on *Evaluate split* sub-problem on the left panel.
- Select DistanceMeasure component from central panel.
- Click on save component button from central panel.
- Save algorithm as *CHAIDModifiedParametersDistance* by clicking on *Save algorithm* button from top panel.

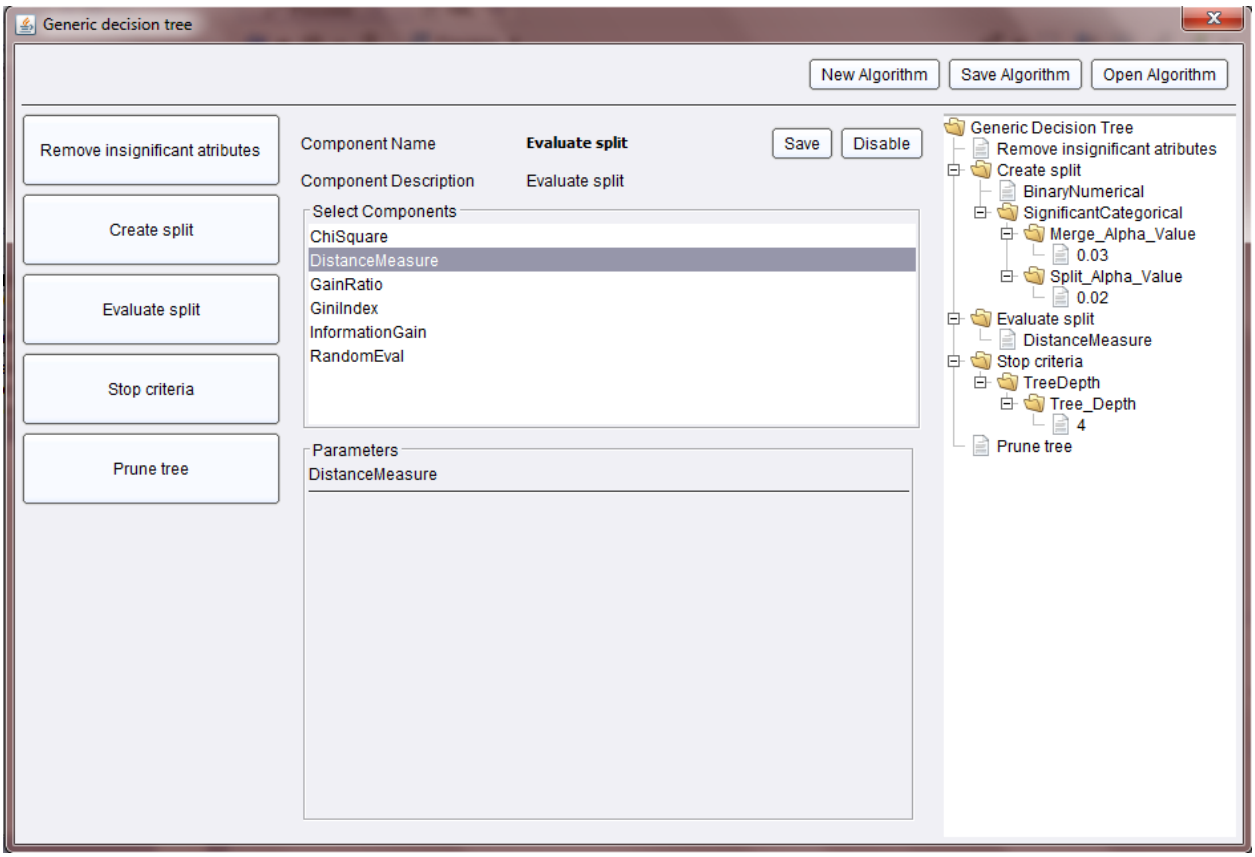


Figure 34 - Replacing components in CHAID algorithm in GDT GUI

Result from executed algorithm is shown on figure below.

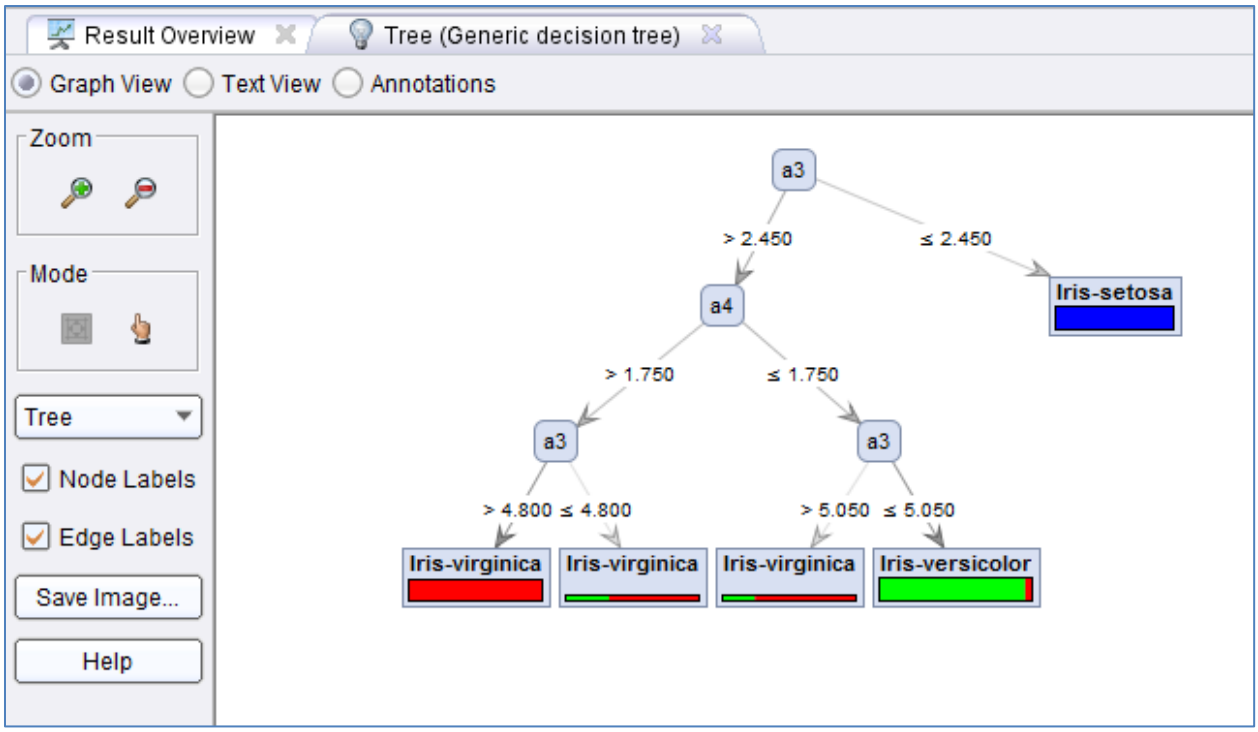


Figure 35 - Result of modified CHAID algorithm

Adding new sub-problem and component to an existing algorithm

Besides changing of parameters and components existing algorithms could be extended with new sub-problems and components. We will explain this extension by adding Prune tree sub-problem to CHAID algorithm that is defined in previous subsection.

- Load CHAID algorithm from file system by clicking on *Open algorithm* button from top panel.
- Select *Prune tree* sub-problem from left panel.
- Click disable.
- Select *MinLeafSize* component from central panel.
- Set *Size_Of_Leaf* parameter to 15.
- Click on save component button from central panel.
- Save algorithm as *CHAIDPrune* by clicking on *Save algorithm* button from top panel.

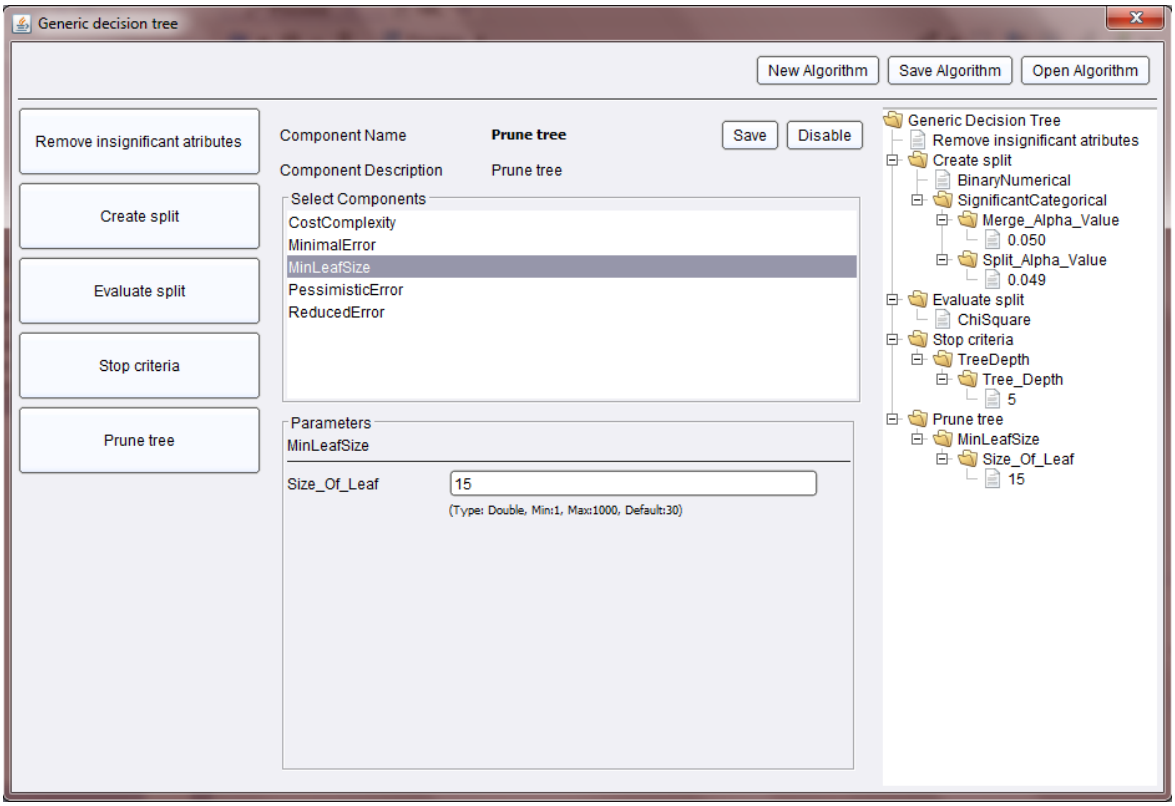


Figure 36 - Adding new sub-problem in existing algorithm

Result from executed algorithm is shown on figure below.

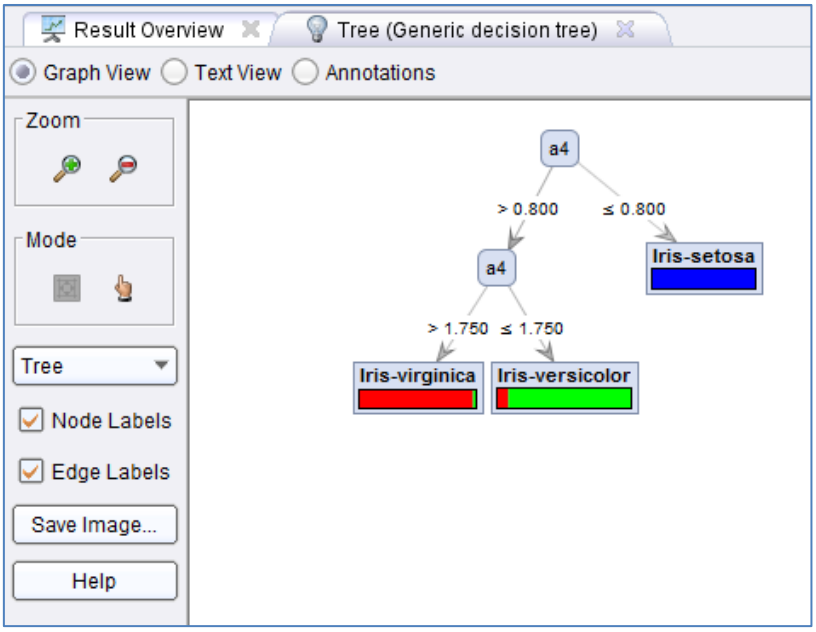


Figure 37 - Result of executed CHAID with prune algorithm on Iris dataset

Besides recreation and modification of existing algorithms WhiBo also enables:

- Design of new algorithms, by combination of components that are derived from well-known algorithms (C4.5, CART, CHAID) or partial algorithm improvements (e.g. distance measure).
- Incorporating partial improvements of algorithms that can be found in literature, but are not incorporated in any specific algorithm (e.g. Distance evaluation measure).
- Incorporating a new sub-problem in an algorithm (e.g. Remove insignificant attributes)
- Multiple component selection for sub-problem - it is possible to define more Splitting components stopping criteria.

Generic decision tree evolutionary search design and application

WhiBo evolutionary search GDT is implemented as RapidMiner operator chain. WhiBo evolutionary search decision tree operators require *ExampleSet* as input and produce *TreeModel, ExampleSet* and *Performance* on output.

For these examples we use "Weighting" dataset from RapidMiner's sample data repository.

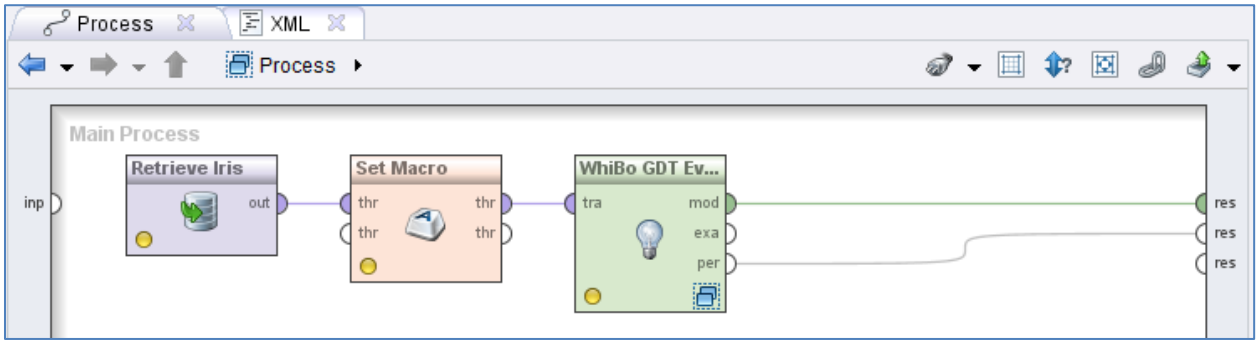


Figure 38 - Main process for WhiBo evolutionary search decision tree

Main process should contain at least three operators. Those are dataset (in this case Weighting dataset), Set Macro and WhiBo GDT Evolutionary Search. Since WhiBo GDT Evolutionary Search is operator chain it contains subprocess. Inside of this subprocess Generic decision tree should be placed.

After loading dataset Macro must be provided. Macro points to WhiBo algorithm file which is needed to *GDT Evolutionary Search* operator.

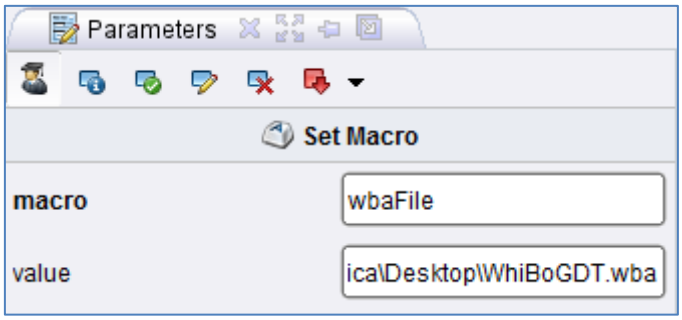


Figure 39 - Parameters panel of *Set Macro* operator

GDT Evolutionary Search operator is set after this. Previously defined Macro is set in *wba file path macro name* parameter text box. Also, log file path is defined. In that file results will be stored. Setup for algorithm search space and parameters of genetic algorithms are below.

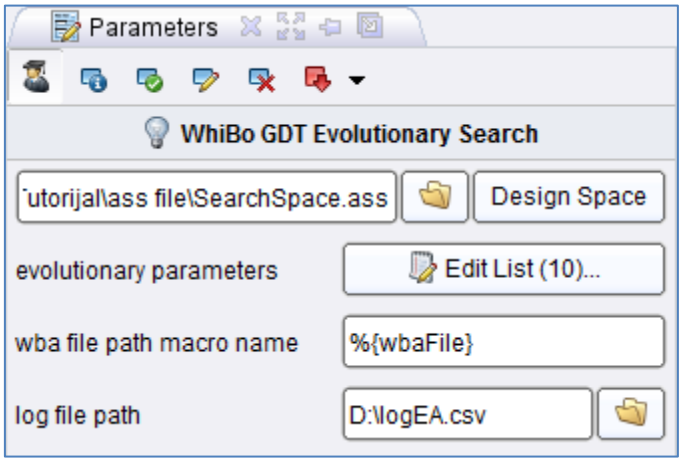


Figure 40 - Parameters panel of *GDT Evolutionary Search* operator

Algorithm search space must be specified.

First, define Create split sub-problem space:

- Click on Create split sub-problem on the left panel.
- Select all components from central panel (multiple components for one sub-problem are selected by holding CTRL key and clicking on components)
- Set default lower and upper values for *Merge_Alpha_Value* and *Split_Alpha_Value* for *SignificantCategoricalComponent*.
- Click on Save component button from central panel.

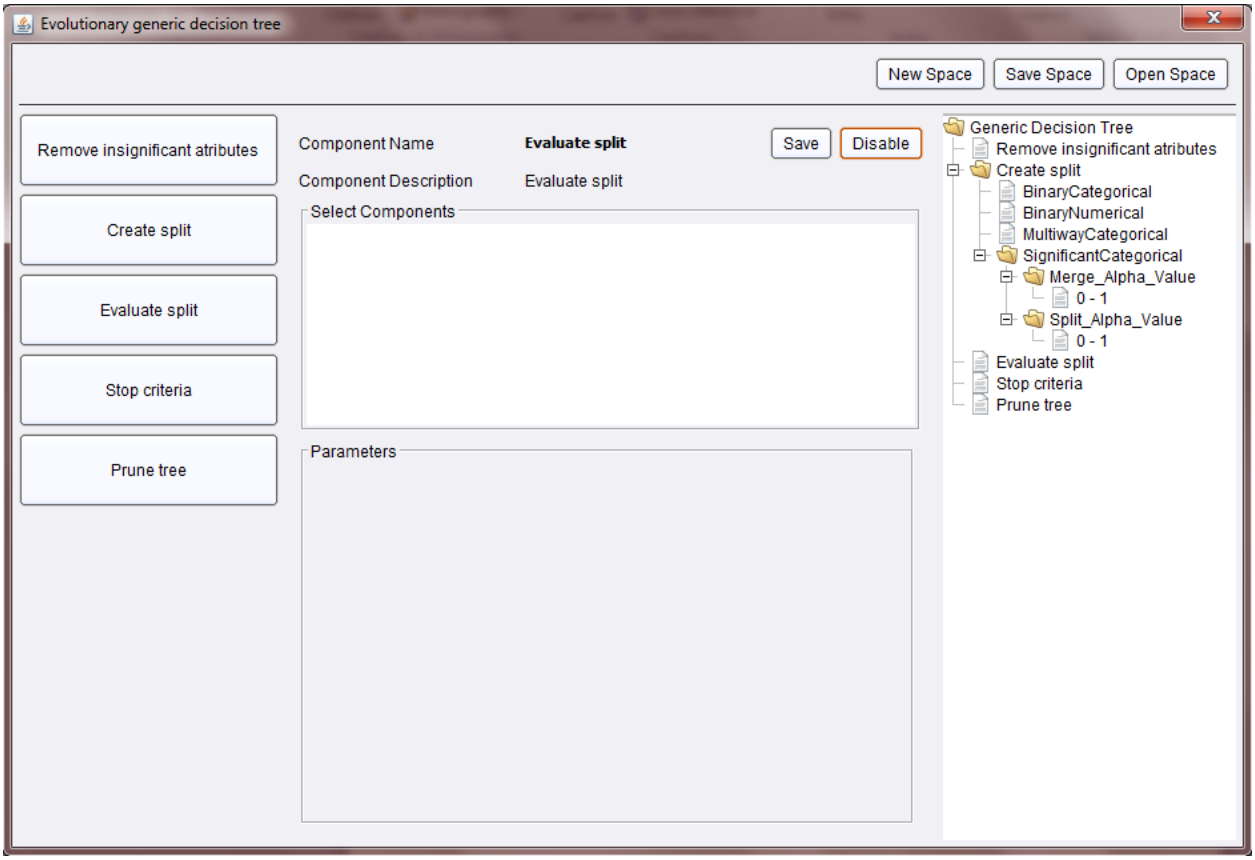


Figure 41 - Definition of *Create split* components for algorithm search space

Second, define Evaluate split sub-problem:

- Click on *Evaluate split* sub-problem on the left panel.
- Select *ChiSquare, DistanceMeasure* and *GainRatio* component from central panel.
- Click on save component button from central panel.

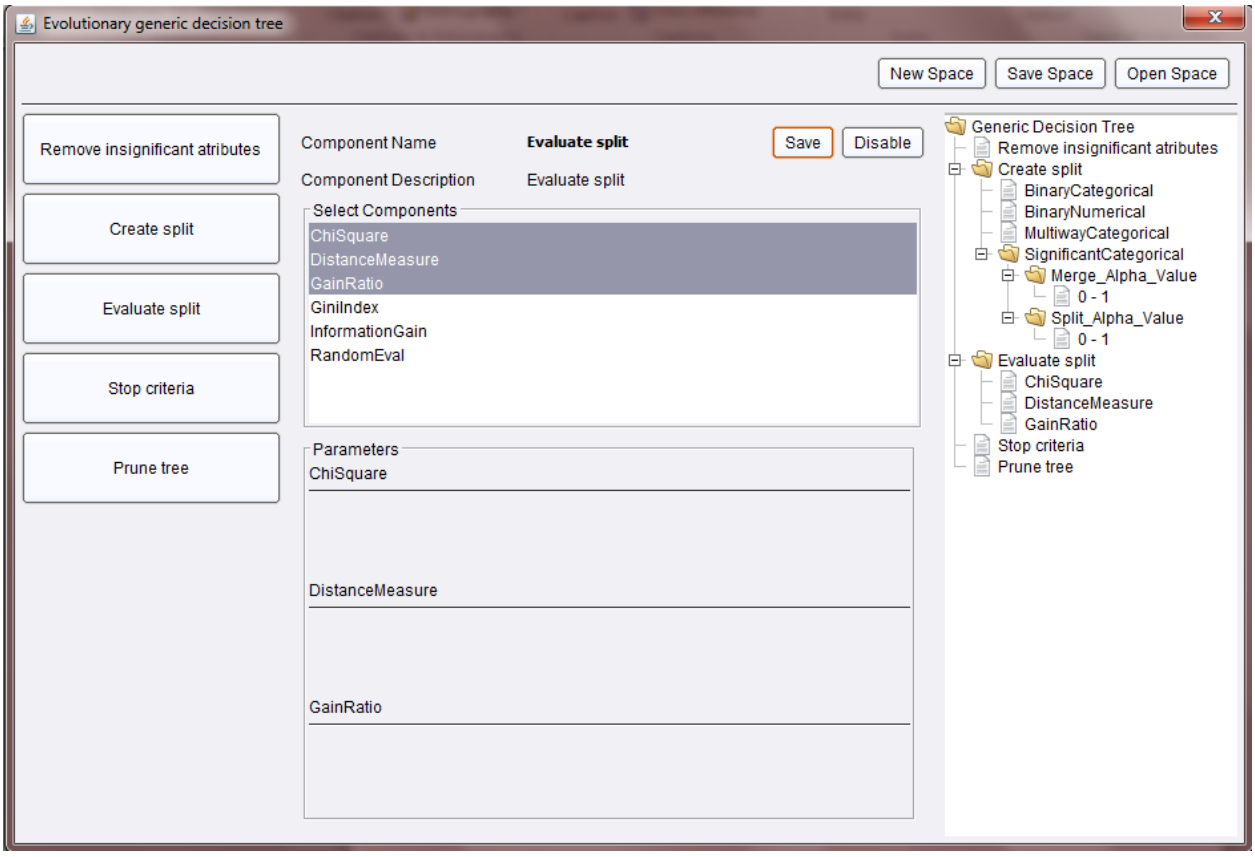


Figure 42 - Definition of *Evaluate split* components for algorithm search space

Third, define *Stop criteria* sub-problem:

- Click on *Stop criteria* sub-problem on the left panel.
- Select *TreeDepth* component from central panel.
- Set *Tree_Depth* parameter from 1 to 10.
- Click on save component button from central panel.

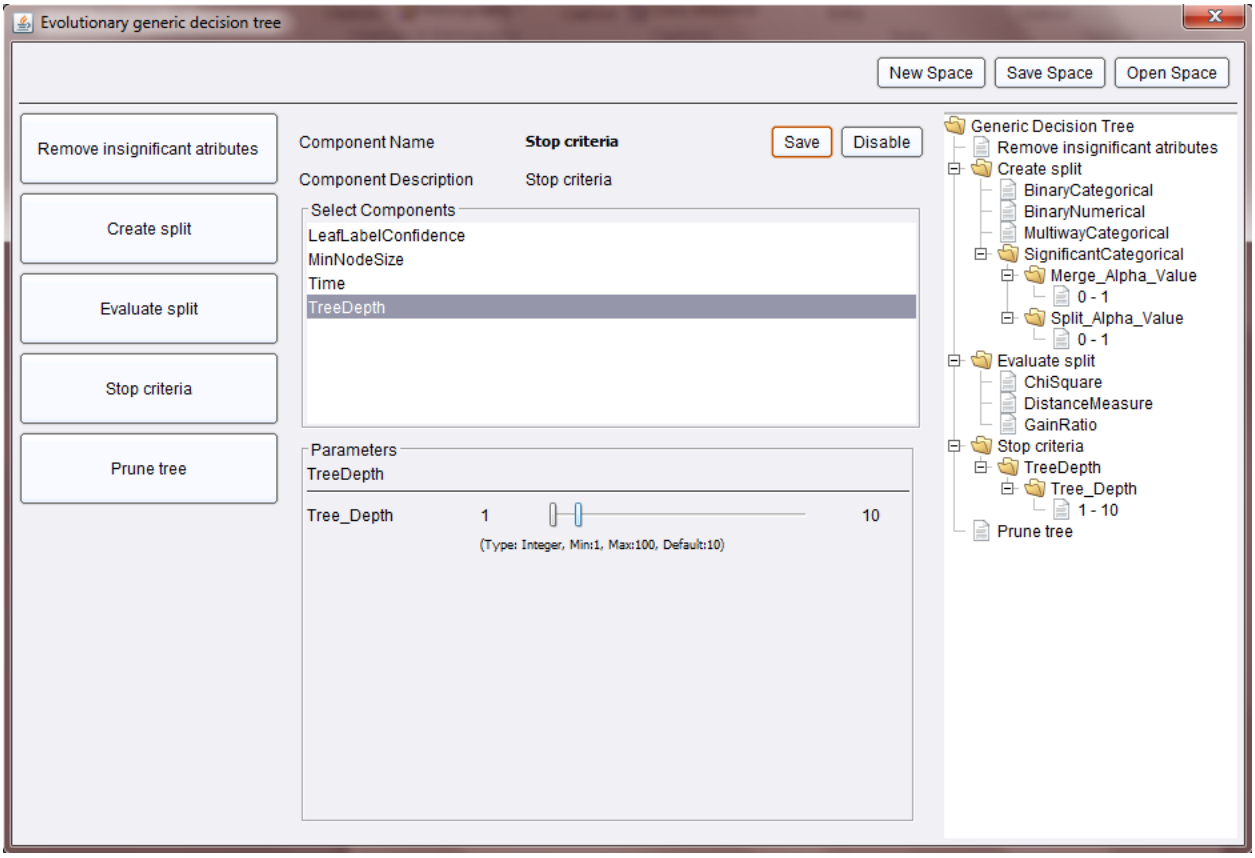


Figure 43 - Definition of *Stop criteria* components for algorithm search space

Fourth, define *Prune tree* sub-problem:

- Click on *Prune tree* sub-problem on the left panel.
- Select *PessimisticError* component from central panel.
- Set *Confidence_Level* parameter from 0 to 0.5.
- Click on save component button from central panel.

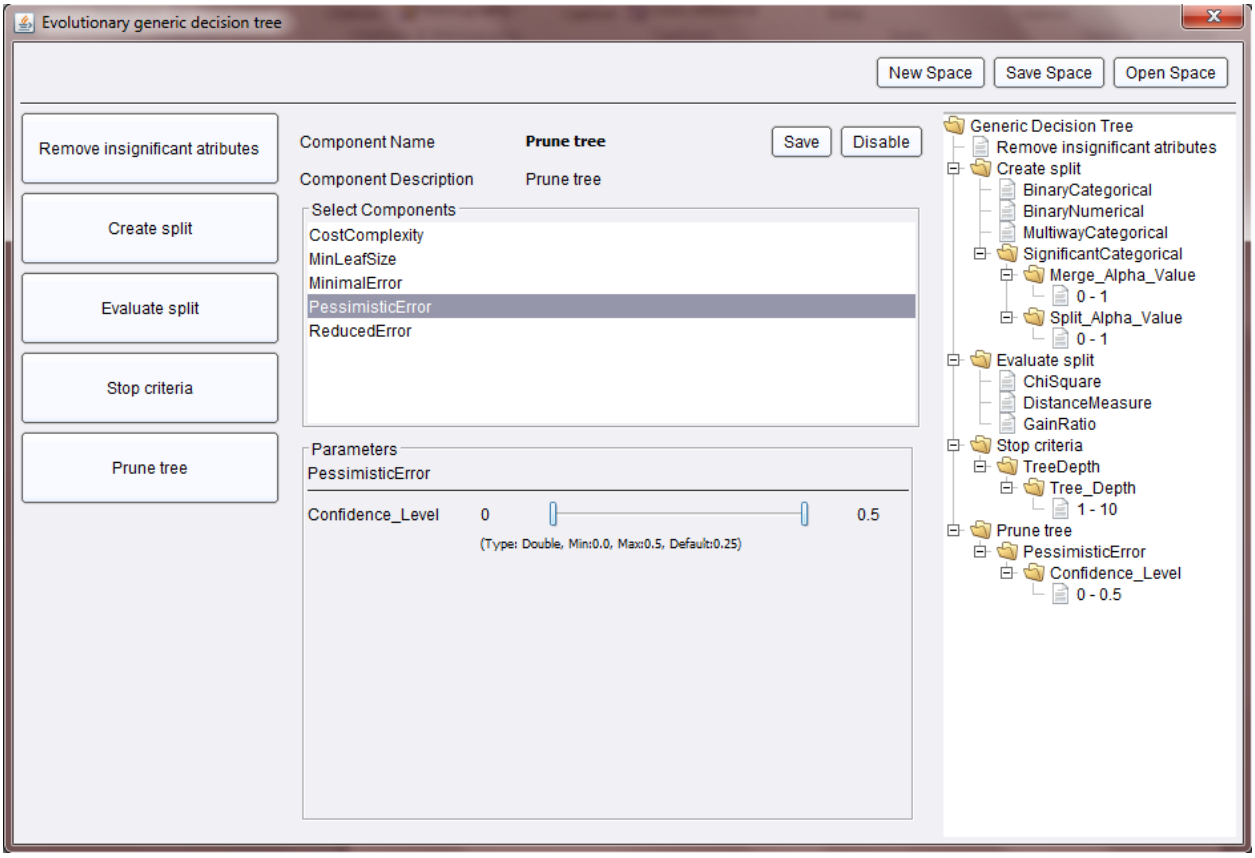


Figure 44 - Definition of *Prune tree* components for algorithm search space

Next step in configuring *GDT Evolutionary Search* operator is parameter settings of genetic algorithm by clicking Edit List button next to parameters in Parameters panel:

- Set MAX_ALLOWED_POPULATION to 10.
- Set POPULATION_SIZE to 5
- Set mutateParameters to true.
- Click on Apply button.

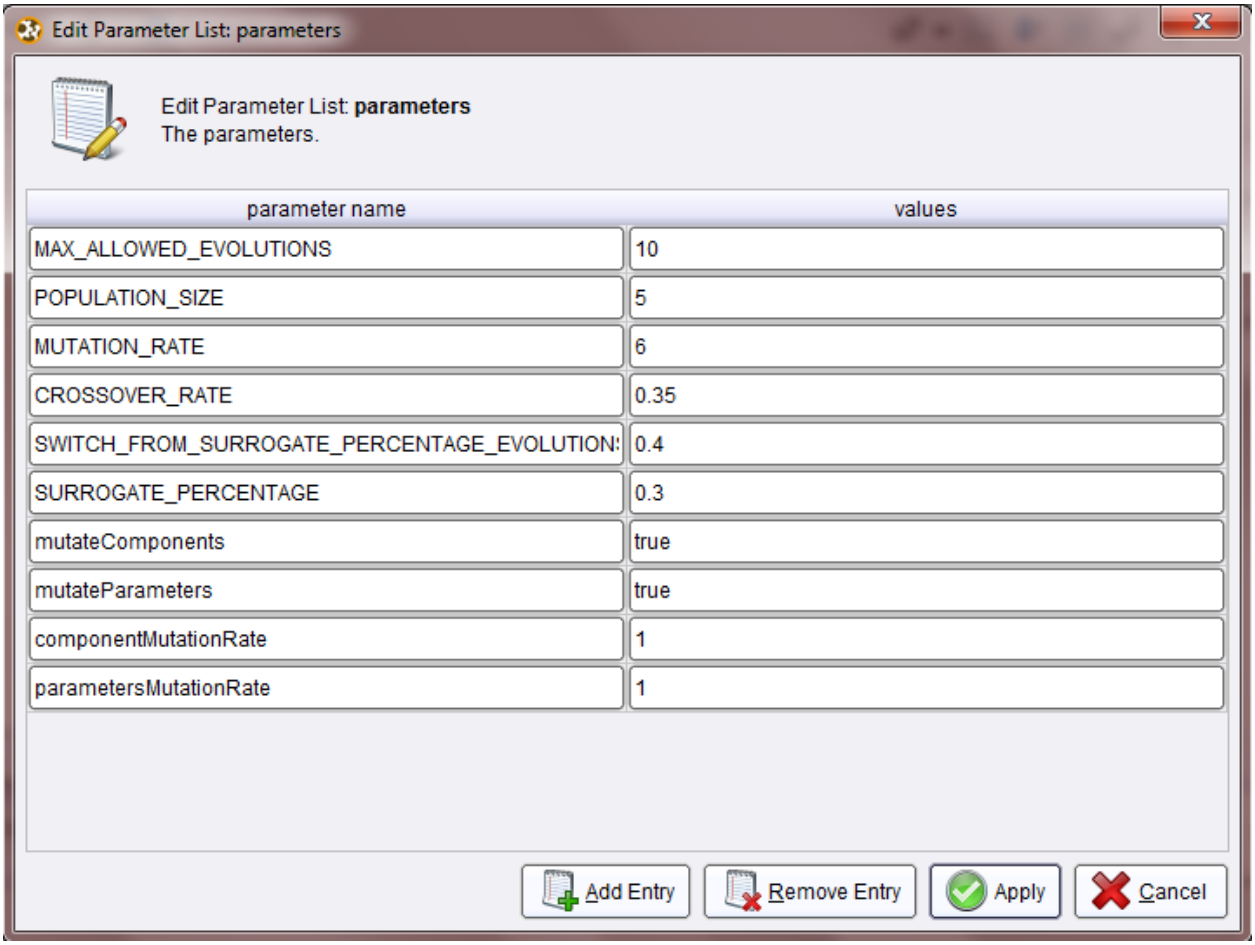


Figure 45 - Definition of parameters for genetic algorithm

Inside *GDT Evolutionary Search* operator is cross validation operator, and inside cross validation there is *GDT* operator in Training section, while *Apply Model* and *Performance* operators are in Testing section. GDT should have valid .wba file, with defined *Create split* and *Evaluate split* components.

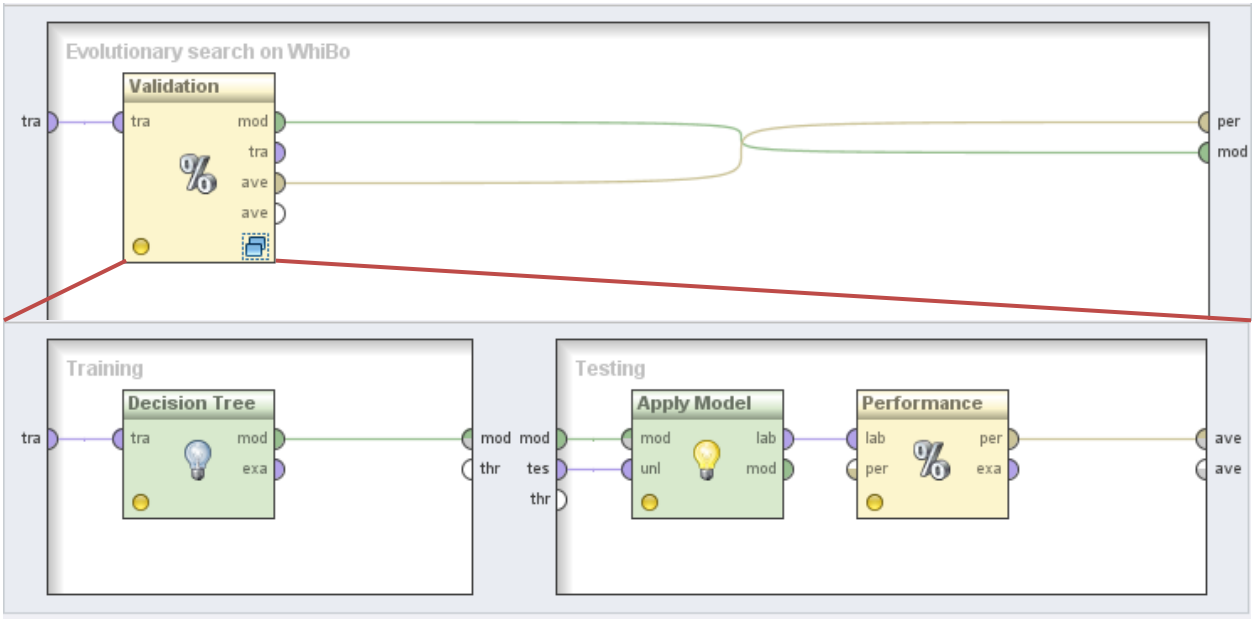


Figure 46 - *GDT Evolutionary Search* subprocess

Result from executed algorithm is shown on figures below.

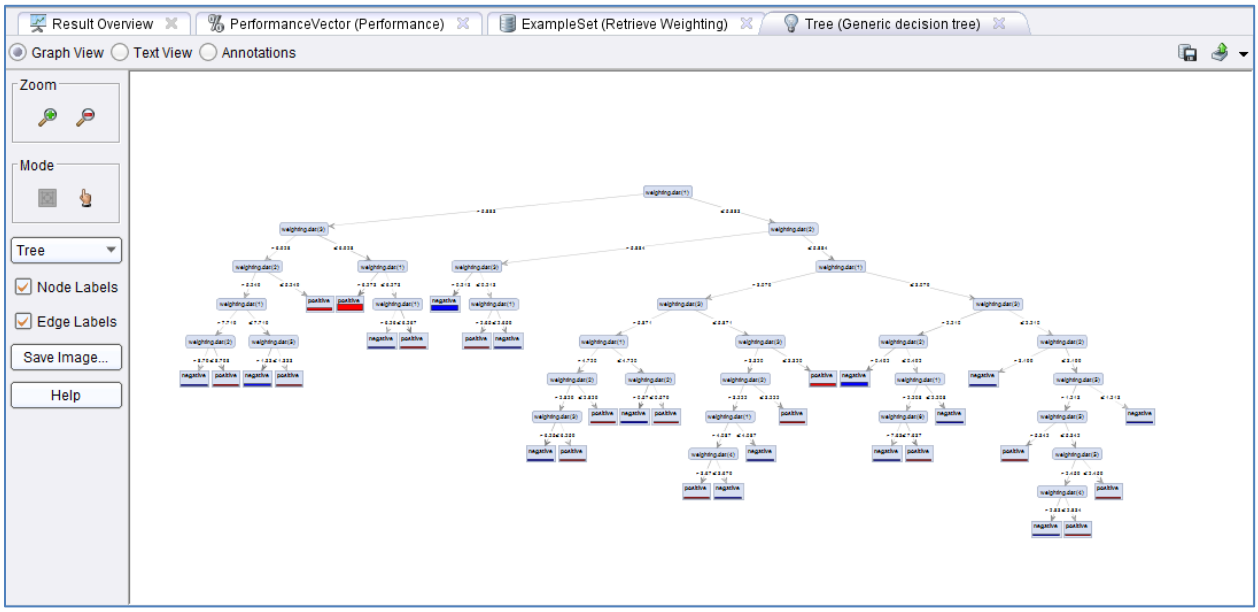
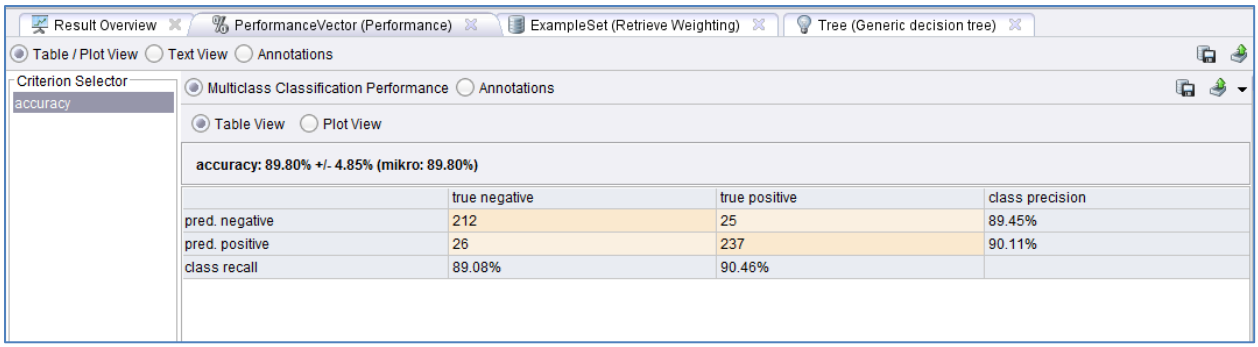


Figure 47 - Result of executed *GDT Evolutionary Search* example



	true negative	true positive	class precision
pred. negative	212	25	89.45%
pred. positive	26	237	90.11%
class recall	89.08%	90.46%	

Figure 48 - Performance of executed *GDT Evolutionary Search* example

21	Number of values returned from cache: 6					
22	Number of evaluations of fitness function: 16					
23	Execution time: 00:07					
24	Cache cleared----------					
25	null	BinaryCategorical	ChiSquare	TreeDepth(46)	PessimisticError(0.08941713426889725)	0.92
26	null	BinaryCategorical	ChiSquare	TreeDepth(77)	PessimisticError(0.08941713426889725)	0.91
27	null	BinaryCategorical	DistanceMeasure	TreeDepth(77)	PessimisticError(0.08941713426889725)	0.892
28	null	BinaryCategorical	GainRatio	TreeDepth(77)	PessimisticError(0.08941713426889725)	0.898
29	null	BinaryCategorical	ChiSquare	TreeDepth(77)	PessimisticError(0.3948348474061291)	0.912
30	null	BinaryCategorical	ChiSquare	TreeDepth(15)	PessimisticError(0.3948348474061291)	0.894
31	null	BinaryCategorical	ChiSquare	TreeDepth(78)	PessimisticError(0.3948348474061291)	0.896
32	null	BinaryCategorical	ChiSquare	TreeDepth(78)	null	0.908
33	null	BinaryCategorical	ChiSquare	TreeDepth(17)	PessimisticError(0.3948348474061291)	0.902
34	null	BinaryCategorical	ChiSquare	null	PessimisticError(0.3948348474061291)	0.902
35	The best solution fitness value:	0.92				
36	Best Solution:					
37	null	BinaryCategorical	ChiSquare	TreeDepth(17)	PessimisticError(0.3948348474061291)	
38	Number of values returned from cache: 25					
39	Number of evaluations of fitness function: 10					

Figure 49 - Log file of executed *GDT Evolutionary Search* example

Modifying algorithm search space

Modifying of existing algorithm search space is done the same way as creating of new algorithms, by clicking Design algorithm button. In this example we will show how to

- modify parameter of already defined component,
- change component of defined sub-problem,
- adding new sub-problem and component for its solution.

Modifying parameters

In this example we will modify Merge_Alpha_Value parameters of *Create split – SignificantCategorical* component:

- Click on *Create split* sub-problem on the left panel.
- Set *Merge_Alpha_Value* from 0.1 to 0.4.
- Click on save component button from central panel.
- Save algorithm search space.

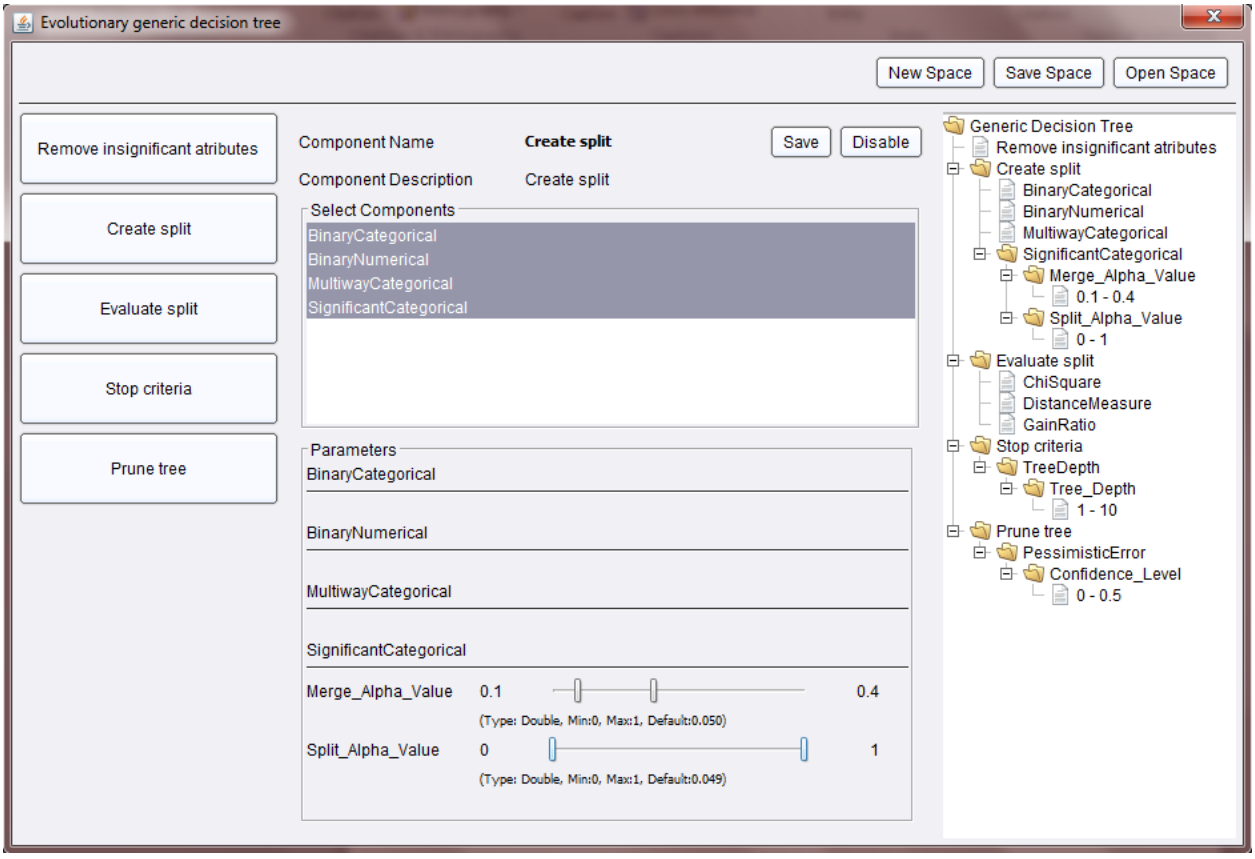


Figure 50 - Modifying parameters of *Merge_Alpha_Value*

Replacing components for sub-problem

Replacing components for sub-problem is done in following way:

- Click on *Evaluate split* sub-problem on the left panel.
- Add InformationGain component (using CTRL button) from central panel.
- Remove GainRatio component (using CTRL button).
- Click on save component button from central panel.
- Save algorithm search space.

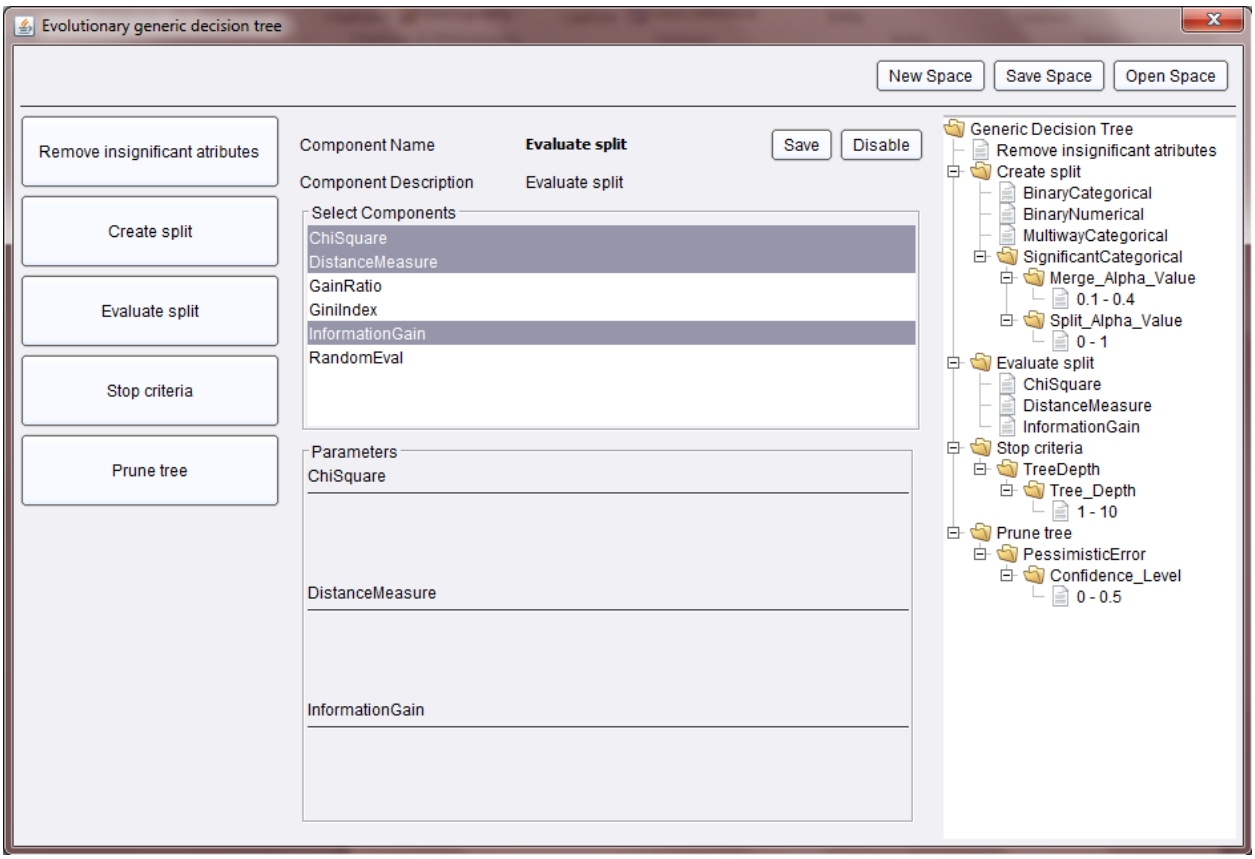


Figure 51 - Replacing components in *GDT Evolutionary Search*

Adding new sub-problem and component to an existing algorithm

Besides changing of parameters and components existing algorithm search space could be extended with new sub-problems and components. We will explain this extension by adding Remove insignificant attributes sub-problem.

- Select *Remove insignificant attributes* sub-problem from left panel.
- Select *FTestNumerical* component from central panel.
- Leave default paramaters setting.
- Click on save component button from central panel.
- Save algorithm search space.

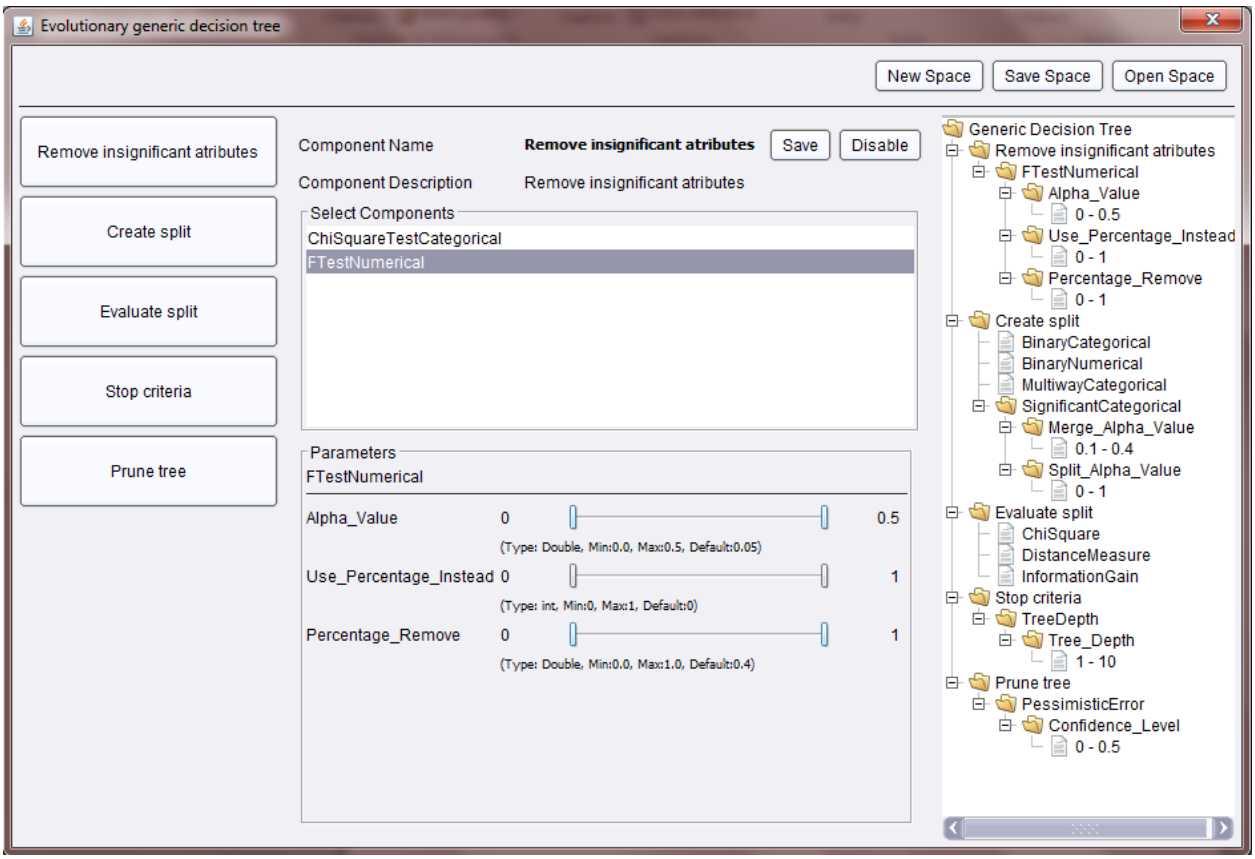


Figure 52 - Adding new subproblem in *GDT Evolutionary Search*

After conducting experiment with this algorithm search space setup results shown on figures below were gathered.

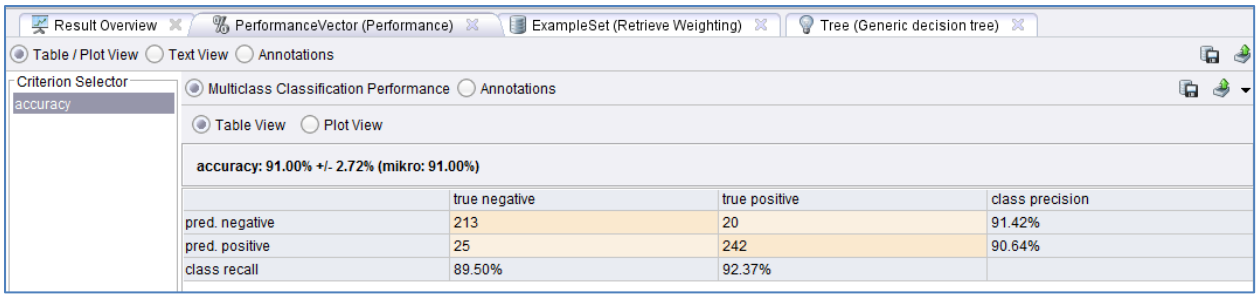


Figure 53 - Performance of executed *GDT Evolutionary Search* example

54	Cache cleared----------					
55	Number of values returned from cache: 13					
56	Number of evaluations of fitness function: 12					
57	Execution time: 00:04					
58	Cache cleared----------					
59	FTestNumerical(0.9729494240097137)	MultiwayCategorical	DistanceMeasure	null	PessimisticError(0.43504839077645285)	0.898
60	FTestNumerical(0.5242083226527987)	MultiwayCategorical	DistanceMeasure	null	PessimisticError(0.291387730739167)	0.912
61	FTestNumerical(0.27080173073754676)	MultiwayCategorical	DistanceMeasure	null	PessimisticError(0.291387730739167)	0.902
62	FTestNumerical(0.9053335432718563)	MultiwayCategorical	DistanceMeasure	null	PessimisticError(0.2137296357883106)	0.904
63	FTestNumerical(0.34203456281548106)	MultiwayCategorical	DistanceMeasure	null	PessimisticError(0.2137296357883106)	0.912
64	FTestNumerical(0.662099165297556)	MultiwayCategorical	DistanceMeasure	null	PessimisticError(0.2137296357883106)	0.898
65	The best solution fitness value:	0.912				
66	Best Solution:					
67	FTestNumerical(0.34203456281548106)	MultiwayCategorical	DistanceMeasure	null	PessimisticError(0.2137296357883106)	
68	Number of values returned from cache: 26					
69	Number of evaluations of fitness function: 6					

Figure 54 - Log file of executed *GDT Evolutionary Search* example