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 subproblems 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.

Sub-problem	Reusable component	Input	Output	
Remove insignificant attributes	F TEST (numerical attributes) CHI SQUARE TEST (categorical attributes)	Dataset in current node	Dataset in current node (reduced)	
Create split (Numerical)	BINARY	Datasat in		
Croata calit	BINARY	Dataset III	A split candidate	
(Catagorical)	MULTIWAY	current noue		
(Categorical)	SIGNIFICANT			
	CHI SQUARE	CHI SQUARE		
	INFORMATION GAIN	A colit	The best split in current node	
Evaluate split	GAIN RATIO	A split		
	GINI	canuluate		
	DISTANCE MEASURE			
	MAXIMAL TREE DEPTH	Current tree	Signal for stopping	
Stop criteria	MINIMAL NODE SIZE	model	tree growth in current node	
	PESSIMISTIC ERROR	Current tree		
Prune tree	PRUNING (PEP)	model	Pruned tree model	
	MIN LEAF SIZE (MLS)	mouch		

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 subproblems, RCs and parameters).

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

🕌 Generic decision tree				×
			New Algorithm	Save Algorithm Open Algorithm
Remove insignificant atributes	Component Name Component Description	Remove insignificant atributes Remove insignificant atributes	Save Disable	Generic Decision Tree ⊢ ☐ Remove insignificant atributes ⊖ ☐ Create split ⊢ ☐ BinaryNumerical
Create split	Select Components ChiSquareTestCategorica FTestNumerical			MultiwayCategorical Galactic split Galactic split Galactic split Galactic split Galactic split Galactic split Galactic split
Evaluate split				Prune tree
Stop criteria				
Prune tree	Parameters ChiSquareTestCategorica			
	Alpha_Value	0.05 (Type: Double, Min:0.0, Max:0.5, Default:0.05)	,	
	Use_Percentage_Instead	0 (Type: int, Min:0, Max:1, Default:0)		
	Percentage_Remove	0.4 (Type: Double, Min:0.0, Max:1.0, Default:0.4)		

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

🖉 Parameters 💥 🚰 🗇 🔯						
🍒 🕫 🕫 🦻 🕵 👼	-					
WhiBo GDT Evolutionary Search						
Design Space						
evolutionary parameters 🛛 🖉 Edit List (10)						
wba file path macro name wbaFilePath						
log file path D:\logEA.csv						

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 subproblems, RCs and parameters).

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

Evolutionary generic decision tree		
		New Space Save Space Open Space
Remove insignificant atributes	Component Name Component Description	Save Disable Generic Decision Tree Remove insignificant atributes Create split
Create split	Select Components	Stop criteria Prune tree
Evaluate split		
Stop criteria		
Prune tree	Parameters	

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).

🚯 Edit Parameter List: parameters	
Edit Parameter List. parameters The parameters.	
parameter name	values
MAX_ALLOWED_EVOLUTIONS	50
POPULATION_SIZE	30
MUTATION_RATE	6
CROSSOVER_RATE	0.35
SWITCH_FROM_SURROGATE_PERCENTAGE_EVOLUTION:	0.4
SURROGATE_PERCENTAGE	0.3
mutateComponents	true
mutateParameters	false
componentMutationRate	1
parametersMutationRate	1
Add	Entry Remove Entry Apply Scancel

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.

🛛 🛃 Parameters 🕱 🚰 🖨 🔟					
🍇 😼 🕫 🦻 🕵 👼	•				
X-Validatio	on with log				
keep example set					
verage performances or	nly				
algorithm name	GDT				
dataset name	Iris				
number of folds	2				
number of repetitions	5				
sampling type	stratified sampling 🔹				
local random seed 1					
log file details esktop\LogDetails.csv					
log file averages ktop\LogAverages.csv					

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.

🖉 Parameters 🔀 🖓 🗄	
🏅 👒 🕫 🦻 🕵 👼	- -
5x2 X-Vali	dation F-test
alpha	0.05
local random seed	-1
sampling type	stratified 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.

Process 🕱 👔 XML 🕱	
+ + + Process +	að 🗕 🔝 🎲 🔟 🌛 🗕
inp Main Process Generic decis tra mod exa O	res (

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.

	🍃 Pa	rame	ters	×S	2 🗘	
8	5	5	>	•	Б,	•
			G	ener	ic de	cision tree
						Design algorithm

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 wellknown 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.

🛃 Generic decision tree				
			New Algorithm	Save Algorithm Open Algorithm
Remove insignificant atributes	Component Name Component Description	Create split Create split	Save Disable	Generic Decision Tree
Create split	Select Components BinaryCategorical BinaryNumerical			BinaryCategorical BinaryNumerical Evaluate split Stop criteria
Evaluate split	MultiwayCategorical SignificantCategorical			🗆 📄 Prune tree
Stop criteria				
Prune tree	Parameters BinaryCategorical			
	BinaryNumerical			

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.

🛃 Generic decision tree	A 1 1 1 1			— ×-
			New Algorithm	Save Algorithm Open Algorithm
Remove insignificant atributes	Component Name Component Description	Evaluate split Evaluate split	Save Disable	Generic Decision Tree
Create split	Select Components ChiSquare DistanceMeasure			BinaryCategorical BinaryNumerical SinaryNumerical Given split Ginilndex
Evaluate split	GainRatio GiniIndex InformationGain			Prune tree
Stop criteria	Randomevai			
Prune tree	Parameters Ginilndex			

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.

🛃 Generic decision tree	A	THE R. P. LEWIS CO.		
			New Algorithm	Save Algorithm Open Algorithm
Remove insignificant atributes	Component Name Component Description	Stop criteria Stop criteria	Save Disable	Generic Decision Tree
Create split	Select Components LeafLabelConfidence MinNodeSize			EnaryCategorical BinaryNumerical E 🏐 Evaluate split
Evaluate split	Time TreeDepth			⊟- '' Stop criteria 白- '' TreeDepth 白- '' Tree_Depth └── 📄 5
Stop criteria				└─ 📄 Prune tree
Prune tree	Parameters TreeDepth			
	Tree_Depth (T	5 Type: Integer, Min:1, Max:100, Default:10)		

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.

🛓 Save File			— X —
📃 Desktop		•	= 🖷 🍁 🌰 🇐 🗸
Bookmarks	File Name	Size Type	Last Modified
🔆 Last Directory	🔋 📕 Big Data in Education	File Folder	Oct 26, 2013
		File Folder	Apr 19, 2013
	🌗 Data Mining with Weka	File Folder	Oct 23, 2013
	Hortonworks Sandbox	File Folder	Feb 13, 2013
	🌗 Ivica Svasta	File Folder	Aug 28, 2012
	I OSDEA	File Folder	Aug 29, 2012
	Projects	File Folder	Oct 22, 2013
	Red Hat Enterprice	File Folder	Oct 3, 2012
	🔋 Rezultati	File Folder	Sep 28, 2013
	Jan SIAM	File Folder	Oct 7, 2013
	Dol Davelance	File Feldes	E-5-40-0040
CART			
.wba file			•
			Save X Cancel
		at the second of the second second	

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.

🦳 🛃 Parameters 🛛 🚰 🖘 🔟 🔪
🍒 😼 🦻 🕸 😽 🖷 🗝
Generic decision tree
Jsers\lvica\Desktop\CART.wba
Choose a file.

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

🛃 Generic decision tree				×
			New Algorithm	Save Algorithm Open Algorithm
Remove insignificant atributes	Component Name Component Description	Create split Create split	Save Disable	Generic Decision Tree ☐ ☐ Remove insignificant atributes ☐ ☐ Create split ☐ ☐ BinaryNumerical
Create split	Select Components BinaryCategorical BinaryNumerical			MultiwayCategorical Evaluate split Stop criteria
Evaluate split	MultiwayCategorical SignificantCategorical			- 📄 Prune tree
Stop criteria				
Prune tree	Parameters BinaryNumerical			
	MultiwayCategorical			

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.

🕌 Generic decision tree				
			New Algorithm	Save Algorithm Open Algorithm
Remove insignificant atributes	Component Name Component Description	Evaluate split Evaluate split	Save Disable	Generic Decision Tree Remove insignificant atributes Create split Signature
Create split	Select Components ChiSquare DistanceMeasure			MultiwayCategorical GainRatio
Evaluate split	GainRatio GiniIndex InformationGain			Prune tree
Stop criteria	Randomeval			
Prune tree	Parameters GainRatio			

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.

🛃 Generic decision tree			Record of the second se	— X—
			New Algorithm	Save Algorithm Open Algorithm
Remove insignificant atributes	Component Name Component Description	Stop criteria Stop criteria	Save Disable	Generic Decision Tree
Create split	Select Components LeafLabelConfidence MinNodeSize			MultiwayCategorical GainRatio GainRatio
Evaluate split	Time TreeDepth			Er G Stop Cirella Er G TreeDepth Er G Tree_Depth □ I 10
Stop criteria				└─ 🚊 Prune tree
Prune tree	Parameters TreeDepth			
	Tree_Depth	10 (Type: Integer, Min:1, Max:100, Default:10)		

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.

🛃 Generic decision tree				×
			New Algorithm	Save Algorithm Open Algorithm
Remove insignificant atributes	Component Name Component Description	Prune tree Prune tree	Save Disable	Generic Decision Tree Remove insignificant atributes Create split RinaryNumerical
Create split	Select Components CostComplexity MinimalError			MultiwayCategorical SainRatio
Evaluate split	MinLeafSize PessimisticError ReducedError		_	다 🥥 Stop criteria 다 🟐 TreeDepth 다 🏐 Tree_Depth 니 📄 10
Stop criteria				E · · · · · · · · · · · · · · · · · · ·
Prune tree	Parameters PessimisticError			
	Confidence_Level	0.2 (Type: Double, Min:0.0, Max:0.5, Default:0.25)		

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.



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 fr	om central	panel.	

🛃 Generic decision tree		— X —
	New Algorit	m Save Algorithm Open Algorithm
Remove insignificant atributes	Component Name Create split Save Component Description Create split	Generic Decision Tree
Create split	Select Components BinaryCategorical BinaryNumerical	SignificantCategorical
Evaluate split	MultiwayCategorical SignificantCategorical	
Stop criteria		Prune tree
Prune tree	Parameters BinaryNumerical	
	SignificantCategorical Merge_Alpha_Value 0.050 (Type: Double, Min:0, Max:1, Default:0.050) Split_Alpha_Value 0.049 (Type: Double, Min:0, Max:1, Default:0.049)	

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.

🛃 Generic decision tree			And and a second second	
			New Algorithm	Save Algorithm Open Algorithm
Remove insignificant atributes	Component Name Component Description	Evaluate split Evaluate split	Save Disable	Generic Decision Tree Remove insignificant atributes Create split Rinapolyumerical
Create split	Select Components ChiSquare DistanceMeasure			SignificantCategorical SignificantCategorica SignificantCategorica SignificantCategor
Evaluate split	GainRatio GiniIndex InformationGain			E- 🤤 Spirt_Alpha_Value L 📄 0.049 E- 🏐 Evaluate split L 📄 ChiSquare
Stop criteria	Randomevai			Prune tree
Prune tree	Parameters ChiSquare			

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.

🛃 Generic decision tree				×
			New Algorithm	Save Algorithm Open Algorithm
Remove insignificant atributes	Component Name Component Description	Stop criteria Stop criteria	Save Disable	Generic Decision Tree
Create split	Select Components LeafLabelConfidence MinNodeSize			SignificantCategorical SignificantCategorical SignificantCategorical SignificantCategorical SignificantCategorical SignificantCategorical SignificantCategorical
Evaluate split	Time TreeDepth		_	□ □ □ 0.49 □ □ □ 0.49 □ □ □ ChiSquare
Stop criteria				E - ∰ Stop criteria E - ∰ TreeDepth E - ∰ Tree_Depth □ □ 5
Prune tree	Parameters TreeDepth			Prune tree
	Tree_Depth (T	5 ype: Integer, Min:1, Max:100, Default:10)		

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.

🛃 Generic decision tree			Research Control of Co	X
			New Algorithm	Save Algorithm Open Algorithm
Remove insignificant atributes	Component Name Component Descriptior	Prune tree	Save Disable	Generic Decision Tree ☐ ☐ Remove insignificant atributes ☐ ☐ Create split ☐ ☐ BinarvNumerical
Create split	Select Components – CostComplexity MinimalError			 ⇒ SignificantCategorical ⇒ SignificantCateg
Evaluate split	MinLeafSize PessimisticError ReducedError			E- Split_Alpha_value L D 0.049 E- Split L ChiSquare
Stop criteria				⊟ '∰ Stop criteria ⊟ '∰ TreeDepth ⊟ ∰ Tree_Depth └─ 📄 5
Prune tree	Parameters PessimisticError			Gonfidence_Level Gonf
	Confidence_Level	U.2 (Type: Double, Min:0.0, Max:0.5, Default:0.25	i)	

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.



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.

🛃 Open File					x
📃 Desktop				🐅 🌒 🎕	a -
Bookmarks	File Name	Size	Туре	Last Modifie	d
📌 Last Directory	🐌 SQL Developer		File Folder	Feb 10, 2013	
	Umlet		File Folder	Aug 28, 2012	
	🐌 WhiBoStari		File Folder	Oct 30, 2013	
	1 99733.pdf	1 MB	Adobe Acrobat Doc	Oct 12, 2013	
	Adobe Dreamweaver CS6.Ink	1 KB	Prečica	Oct 4, 2013	_
	Balcor 2013 Proceedings.pdf	21 MB	Adobe Acrobat Doc	Oct 9, 2013	
	C4.5.wba	5 KB	WBA Datoteka	Nov 2, 2013	20
	CART.wba	4 KB	WBA Datoteka	Nov 2, 2013	12
	CCleaner.Ink	1 KB	Prečica	Aug 28, 2012	
	CHAID.wba	6 KB	WBA Datoteka	Nov 2, 2013	
	Customize Fences.Ink	1 KB	Prečica	Aug 28, 2012	
	DAEMON Tools Lite.Ink	1 KB	Prečica	Jun 7, 2013	
	Eclipse.Ink	1 KB	Prečica	Sep 28, 2013	
	🛐 Get Started With Oracle Database 11g Express	2 KB	Prečica	Feb 9, 2013	
	🔂 Git Shell.Ink	2 KB	Prečica	Oct 25, 2013	
	GitHub.appref-ms	1 KB	ClickOnce Applicati	Oct 25, 2013	
	📴 Google Web Designer.Ink	2 KB	Prečica	Oct 1, 2013	
CHAID.wba					
All Files					-
				Open 🔀 Ca	ancel

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.

🛃 Generic decision tree		
	New Algorithm	Save Algorithm Open Algorithm
Remove insignificant atributes	Component Name Save Disable	Generic Decision Tree ☐ Remove insignificant atributes ☐ Greate split
Create split	Select Components	SignificantCategorical SignificantCategorica SignificantCategorica SignificantCategor
Evaluate split		E- Spit_Alpha_Value C @ 0.049 E- @ Evaluate split C @ ChiSquare
Stop criteria		⊡ - ∰ Stop criteria ⊡ - ∰ TreeDepth ⊡ - ∰ Tree_Depth □ - ☐ 5
Prune tree	Parameters	B- '쉨 Prune tree È- '쉨 PessimisticError È- '쉨 Confidence_Level └

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.

💡 Tree (Generic decision tree) 🚿 🛒 Result Overview 🗶 Graph View O Text View Annotations Zoom **a**4 0 0 > 0.800 < 0.800Mode Iris-setosa a4 > 1.750 1.750 ≤ Tree Ŧ Iris-virginica a3 Node Labels > 5.050 ≤ 5.050 Edge Labels Iris-virginica Iris-versicolor Save Image... Help

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.

🛃 Generic decision tree			
			New Algorithm Save Algorithm Open Algorithm
Remove insignificant atributes	Component Name	Evaluate split Evaluate split	Save Disable Generic Decision Tree Bave Disable Remove insignificant atributes Create split Binare/lumerical
Create split	Select Components ChiSquare DistanceMeasure		SignificantCategorical
Evaluate split	GainRatio GiniIndex InformationGain		□ → · · · · · · · · · · · · · · · · · ·
Stop criteria			E · · · · · · · · · · · · · · · · · · ·
Prune tree	Parameters DistanceMeasure		Prune tree

Figure 34 - Replacing components in CHAID algorithm in GDT GUI

Result from executed algorithm is shown on figure below.

🔀 Result Overview 🗶 🦳 💡 Tree (Generic decision tree) 🔀				
Oraph View	Text View O Annotations			
Zoom	a3			
P	> 2 450 < 2 450			
Mode	Iris-setosa			
	84			
Tree 💌	> 1.750 ≤ 1.750 a3 a3			
Vode Labels				
🖌 Edge Labels				
Save Image				
Help				

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.

🛃 Generic decision tree				
			New Algorithm	Save Algorithm Open Algorithm
Remove insignificant atributes	Component Name Component Description	Prune tree Prune tree	Save Disable	Generic Decision Tree
Create split	Select Components CostComplexity MinimalError			Ginarjivumencal GinificantCategorical Ginificategorical GinificantCategorica Ginificategorica Gin
Evaluate split	MinLeafSize PessimisticError ReducedError			E- 🦦 Split_Alpha_Value L 📄 0.049 E- 😋 Evaluate split L 📄 ChiSquare
Stop criteria				i ← 崎 Stop criteria i ← 崎 TreeDepth i ← 🏐 Tree_Depth └ 📄 5
Prune tree	Parameters MinLeafSize			E ∰ Prune tree E ∰ MinLeafSize E ∰ Size_Of_Leaf
	Size_Of_Leaf	15 Type: Double, Min:1, Max:1000, Default:30)		L 🗐 15

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.

🖉 Parameters 🛛 🖓 🖨					
🏅 🕫 🕫 🦻 🕵 👼	-				
💡 WhiBo GDT Ev	olutionary Search				
utorijal\ass file\SearchSpace	Cutorijal\ass file\SearchSpace.ass				
evolutionary parameters 🛛 🖉 Edit List (10)					
wba file path macro name					
log file path	D:\logEA.csv				

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.

🛃 Evolutionary generic decision tree				x
			New Space Save Space Open Spa	се
Remove insignificant atributes	Component Name Component Description	Evaluate split Evaluate split	Save Disable Generic Decision Tree	es
Create split	Select Components		BinaryNumerical BinaryNumerical BinaryNumerical BinaryNumerical BinaryNumerical	
Evaluate split			P → Merge_Angna_value	
Stop criteria			Evaluate split Stop criteria Prune tree	
Prune tree	Parameters			

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.

🛃 Evolutionary generic decision tree				X
			New	Space Space Open Space
Remove insignificant atributes	Component Name Component Description	Evaluate split Evaluate split	Save Disable	Generic Decision Tree
Create split	Select Components ChiSquare DistanceMeasure			BinaryOtategorical BinaryNumerical MultiwayCategorical SignificantCategorical
Evaluate split	GainRatio GiniIndex InformationGain RandomEval			i inege_npria_value i i inege_npria_value i i i i i i i i i i i i i i i i i i i
Stop criteria				ChiSquare DistanceMeasure GainRatio
Prune tree	Parameters ChiSquare			Prune tree
	DistanceMeasure			
	GainRatio			

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.

🛓 Evolutionary generic decision tree		
		New Space Save Space Open Space
Remove insignificant atributes	Component Name Stop criteria Component Description Stop criteria	Save Disable Generic Decision Tree
Create split	Select Components LeafLabelConfidence MinNodeSize	Binaryoategorical BinaryNumerical MultiwayCategorical Comparison Alpha Value
Evaluate split	Time TreeDepth	Grand Merge_Appra_value Grand Cont Grand Cont Gran
Stop criteria		ChiSquare ChiSquare Cainestio
Prune tree	Parameters TreeDepth Tree_Depth 1	10

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.

🛃 Evolutionary generic decision tree		
		New Space Save Space Open Space
Remove insignificant atributes	Component Name Prune tree Component Description Prune tree	Save Disable Generic Decision Tree Remove insignificant atributes Generic Decision Tree Remove insignificant atributes Remove split Remove ategorical
Create split	Select Components CostComplexity MinLeafSize	BinaryNumerical - BinaryNumerical - MultiwayCategorical - SignificantCategorical
Evaluate split	MinimalError PessimisticError ReducedError	E ∰ Merge_Alpha_Value ↓ L ≧ 0 - 1 E ∰ 0 - 1 □ L ≧ 0 - 1
Stop criteria		Evaluate split
Prune tree	Parameters PessimisticError	E Stop criteria E Stop criteria E Stop criteria E Stop criteria Tree_Depth
	Confidence_Level 0 (Type: Double, Min:0.0, Max:0.5, Default:0.25)	0.5 Prune tree

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.

3 Edit Parameter List: parameters	
Edit Parameter List: parameters The parameters.	
parameter name	values
MAX_ALLOWED_EVOLUTIONS	10
POPULATION_SIZE	5
MUTATION_RATE	6
CROSSOVER_RATE	0.35
SWITCH_FROM_SURROGATE_PERCENTAGE_EVOLUTION:	0.4
SURROGATE_PERCENTAGE	0.3
mutateComponents	true
mutateParameters	true
componentMutationRate	1
parametersMutationRate	1
Add	Entry Remove Entry Apply X Cancel

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

😤 Result Overview 🗶 🦷 🖏 Performance/Vector (Performance) 🕺 📲 ExampleSet (Retrieve Weighting) 🕺 🜍 Tree (Generic decision tree) 🕺						
Table / Plot View 1	Text View 🔘 Annotations			E 4		
Criterion Selector	Multiclass Classification Performance	ce O Annotations		E 🤌 •		
	Table View OPlot View					
	accuracy: 89.80% +/- 4.85% (mikro: 89.80%)					
		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.

Evolutionary generic decision tree	and a second sec	×
		New Space Save Space Open Space
Remove insignificant atributes	Component Name Create split Save	Disable Generic Decision Tree Remove insignificant atributes Create split
Create split	Select Components BinaryCategorical BinaryNumerical	BinaryNumerical MultiwayCategorical
Evaluate split	MultiwayCategorical SignificantCategorical	Herge_Appha_Value □ □ 0.1 - 0.4 □ □ □ Split_Alpha_Value □ □ □ 0 - 1
Stop criteria		ChiSquare DistanceMeasure GainRatio
Prune tree	Parameters BinaryCategorical	
	BinaryNumerical	
	MultiwayCategorical	in in restitute for in in in the second se
	SignificantCategorical	
	Merge_Alpha_Value 0.1	0.4
	Split_Alpha_Value 0 [] [] (Type: Double, Min:0, Max:1, Default:0.049)	1

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.

🛃 Evolutionary generic decision tree						
			News	Space Save Space Open Space		
Remove insignificant atributes	Component Name Component Description	Evaluate split Evaluate split	Save Disable	Generic Decision Tree		
Create split	Select Components ChiSquare DistanceMeasure			Binaryoategorical BinaryNumerical MultiwayCategorical SignificantCategorical		
Evaluate split	GainRatio GiniIndex InformationGain			Herminian Merge_Alpha_Value ↓ ↓ ⊕ 0.1 - 0.4 Herminian Split_Alpha_Value ↓ ↓ ⊕ 0 - 1		
Stop criteria				ChiSquare DistanceMeasure InformationGain		
Prune tree	Parameters ChiSquare			E Stop criteria E Stop criteria E Stop CreeDepth E Stop Tree_Depth E Stop Prune tree E Stop Prune tree		
	DistanceMeasure			i in testinisticchoi i in testinistic i in testini i in testinistic i i		
	InformationGain					

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.

🛃 Evolutionary generic decision tree	Contraction of the local division of the loc		-	
			New	Space Save Space Open Space
Remove insignificant atributes	Component Name Component Description	Remove insignificant atributes Remove insignificant atributes	Save Disable	Generic Decision Tree 다 쉨 Remove insignificant atributes 다 쉨 FTestNumerical
Create split	Select Components ChiSquareTestCategorical FTestNumerical			L ⊇ 0 - 1 L ⊇ 0 - 1 L ⊇ 0 - 1 L ⊇ 0 - 1
Evaluate split				└ @ 0 - 1 Create split BinaryCategorical
Stop criteria				MultiwayCategorical GyrinicantCategorical GyrinicantCategorical GyrinicantCategorical GyrinicantCategorical
Prune tree	Parameters FTestNumerical			□ 📄 0.1 - 0.4 □ 🈋 Split_Alpha_Value □ 📄 0 - 1
	Alpha_Value (Use_Percentage_Instead ((Percentage_Remove ()) Type: Double, Min:0.0, Max:0.5, Default:0.05))) Type: int, Min:0, Max:1, Default:0)	() 0.5 () 1	Valuate split ChiSquare DistanceMeasure InformationGain Stop criteria TreeDepth TreeDepth TreeDepth
	(- U Type: Double, Min:0.0, Max:1.0, Default:0.4)	υ	
				< <u> </u>

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.

🛛 🐺 Result Overview 🕱 🧏 PerformanceVector (Performance) 🕺 🗋 ExampleSet (Retrieve Weighting) 🕺 🖓 Tree (Generic decision tree) 🕺					
Table / Plot View Text View Annotations					
Criterion Selector	Multiclass Classification Performance Annotations			🔓 🤞 -	
accuracy	Table View Plot View				
accuracy: 91.00% +/- 2.72% (mikro: 91.00%)					
		true negative	true positive	class precision	
	pred. negative	213	20	91.42%	
	pred. positive	25	242	90.64%	
	class recall	89.50%	92.37%		

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