GINNet developper guide
Version 5.2.1
for DynNet and GINNet 2.2

Marie TONNELIER

15/02/2007

Contents

Chapter 1

Abstract

The purpose of this document is to provide all needed informations to DynNet/GINNet developpers. It includes first steps with GINNet project, a short description of some concepts used, a technical description of GINNet program.
Any developper can take part in this guide.
Last version of this document can be found on GINNet project Documents web page, at
`http://gforge.inria.fr/docman/?group_id=82`

Chapter 2

First steps

2.1 GINNet and DynNet

You can get a quick presentation of DynNet and GINNet project at this URL: `http://ginnet.gforge.inria.fr/`

2.2 Java

2.2.1 Why is GINNet developped in Java?

GINNet is 100% pure Java.
This language has been choosen because:

- required developpement time is relatively low
- oriented object conception allow reusability and extensions
- portability is assumed automatically
- application can be used easily (jar, applet, web start, etc.)
- JavaDoc tool provides quickly a good API documentation

However, Java is still slow, so you will have to take care of optimisation during developpement.
NB: GINNet is Java 5.0 compliant.

2.2.2 Where to find JRE and/or SDK

You can find Java JRE and SDK in CORTEX tools directory (*/users/cortex/tools*).

2.3 Charter

A charter has been written for all GINNet users to ensure code legibility, validity and quality. It specifies how to:

- name Java elements
- comment code
- properly use Java and optimize code
- improve ergonomics

You can find this charter under DynNet CVS repository (*docs/charter.pdf*).

2.4 Some descriptive informations

DynNet project started in 2002 and more than one dozen of programmers followed one another.
GINNet Java project contains near 360 classes and 50 000 source lines (and behong them, approximatively 40% of comments).
That's why this document tries to make this project more accessible for new developpers.

2.5 GForge

GINNet and DynNet project is on INRIA GForge since september 2005. This forge provides several usefull development tools, accessible via a web interface.
To be a GINNet and DynNet project member, you need to

- have an account on gforge site (to register, go to `http://gforge.inria.fr/account/register.php`)
- contact an administrator (Marie.Tonnelier@loria.fr or Laurent.Bougrain@loria.fr) and ask for being register as a project member (as developer, doc writer, tester, user, etc.)

You can access project web interface at
`https://gforge.inria.fr/projects/ginnet/`.
GForge utilities will now be described on this section.

2.5.1 Forums

Use forums to discuss about GINNet project (if you can't do this in person).

2.5.2 Tracker

Bug tracker

Bug tracker is used to manage bugs, so you can:

- See existing known bugs and their state. Most colored bugs are most important.
- Change a bug's state (for example, set it as resolved)
- Submit a new one

Support request

Support request section is a way to ask for help about any problem with GINNet or DynNet. Keeping trace of problems and theit solutions makes it possible to advance more easily thereafter.

Patches

If patches are available, they're here.

Feature requests

It's the place where you can ask for a new feature from any type.

2.5.3 Tasks

Tasks tab shows work to do and work in progress, and especially until next GINNet release.
Task manager is currently unused for this project and has been desactivated.

2.5.4 Docs

Docs tab regroups all public documentations available for GINNet project.
For the moment, you can only find development documentation, organized in two categories:

Development documentation: the category where you can find this guide and coding charter (Those documents are latex ones, exported to pdf, please prefer this format for new documentation.)

JavaDoc: redirections to DynNet and GINNet javadoc

(You can find more documentation on SubVersion repository, in *docs* directory.)

2.5.5 News

Hot informations about project advancement!

2.5.6 SCM

Software Configuration Management is the place where you can access SubVersion repository.

SubVersion

GINNet project is on GForge SubVersion server since the beginning of october 2005.
SubVersion is an open source version control system and a replacement of the Common Version Control System (CVS). (You can get more informations about SubVersion on `http://www.subversion.com/`)
You can find very short guide on how to use SubVersion at the end of this document, at chapter 6.1
N.B.: GINNet project was previously on INRIA CVS server, at Sophia-Antipolis, since the end of 2004.

GINNet repository

Repository name: ginnet.

Connection type: svn+ssh

Host: scm.gforge.inria.fr

Repository path: /svn/ginnet

User: gforge_login

Password: gforge_password

SSH authentification

Before accessing to GINNet repository, you need to be authorized to from SubVersion server. This means that your public SSH key must be known by GForge server. To do this, generate a pair SSH key and copy your public key via GForge interface. You can do it here: `http://gforge.inria.fr/account/editsshkeys.php`
After your key is registred (in following hour), you can access repository whenever you want to.

How to create a pair key?
% ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/users/cortex/user/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /users/cortex/user/.ssh/id_rsa.
Your public key has been saved in /users/cortex/user/.ssh/id_rsa.pub.
The key fingerprint is:
d0:16:79:4a:c4:40:f4:88:25:b2:04:41:46:0e:6b:ae user@villers.loria.fr
Your pair key is created. Your public key is here saved in */.ssh/id_rsa.pub* file.

2.5.7 Files

It is the place where you can quickly see and download all DynNet and GINNet versions.
But use SCM for development.

2.6 How to compile and run GINNet from sources?

An Ant build file has been created to easily compile, run and perform all other usefull tasks for GINNet and DynNet. See section 6.2.2 for more informations.

2.7 Architecture

2.7.1 Directories

SubVersion repository

SubVersion repository contains four directories :

- ***src***: contains GINNet source code, i.e.:
  - *fr.loria.cortex.ginnet* package that contains DynNet and GINNet code, and available in public packages.
  - *tests* package that contains classes only use for testing and especially test cases.
- ***examples***: contains example files:
  - *corpus*: all saved corpus examples, regrouped by type of corpus (categorization, classification, forecast)
  - *databases*: all data saved files (including some files from UCI Machine Learning Repository into *ucimlr* folder) (See `http://www.ics.uci.edu/~mlearn/MLRepository.html` for more data examples).
  - *networks*: saved networks
- ***lib***: contains external library in jar files
  - JUnit (*junit.jar*): for test cases
  - MySQL (*mysql.jar*): to manage MySQL databases
- ***docs***: this folder contains documentation ressources and especially:
  - *charter*: source files (LaTeX file and graphics) of GINNet charter
  - *guide*: source files (LaTeX file and graphics) of GINNet developper guide
  - *uml*: all UML ressources: diagram images and Poseidon source file (whose extension is *zuml*)
  - *profiling*: backups of java profiling files
- ***utils***: regroups some scripts and usefull programs:
  - *ant*: a directory containing build tools. For more informations about Ant see section 6.2.
    - *build.xml*: the Ant script allowing to compile separately GINNet and DynNet, make corresponding jar files, launch GINNet, generate the JavaDoc, copy useful files to the web site, launch test cases, build source zip packages.
    - *DynNet_build.xml* and *GINNet_build.xml*: the Ant scripts for DynNet and GINNet source releases. They are not usable as it, but are used by the *build.xml* script.
    - *jsch-0.1.28.jar*: a library used by the *build.xml* script to perform scp tasks.
  - *GINNetCorpusConverter.jar*: a version of GINNet that allow corpus convertion from old to new corpus format (changed in version 1.2.5). To convert an old corpus: run this jar, open corpus file and save it. It will be saved in new corpus format that can be opened with GINNet and DynNet 1.2.5 or higher.
  - *stats.csh*: gives some statistics on source code
  - *GINNetManifest.MF*: the manifest file to build GINNet.jar, used by the *build.xml* script.
  - *GINNetStore*: the keystore to sign GINNet.jar, used by the *build.xml* script.
  - *header*: a package containing a Java program to easily replace headers of all source files
- ***old***: this folder only exists temporally, to keep old code files and is not useful.

bin directory is not under SVN. It's the folder where to put compiled sources.
Root directory contains useful files:

- DynNet and GINNet packed in two jar files (not always up to date): *GINNet.jar* and *DynNet.jar*, and files used to build them: *GINNet.cer* certificate and *GINNetStore* keystore.
- *GINNet.jnlp*: JNLP (Java Network Launching Protocol) file to start GINNet with Java Web Start
- *applet.html*: GINNet in an applet (that uses *GINNet.jar* and *style.css*)

Releases

GINNet and DynNet are separately downloadable as zipped sources packages. Both of them contains two directories :

- ***src***: contains GINNet package (*fr.loria.cortex.ginnet*) or just DynNet package (*fr.loria.cortex.ginnet.dynnet*).
- ***lib***: contains external library in jar files
  - JUnit (*junit.jar*): for test cases
  - MySQL (*mysql.jar*): to manage MySQL databases

  Root directory contains useful files:

- GINNet license: in english (*CeCILL_license.txt*) and in french (*licence_CeCILL.txt*).
- *build.xml*: the Ant build script that can compile, make corresponding jar and generate JavaDoc. For GINNet, it can also launch the application. For more informations about Ant see section 6.2.

GINNet zipped file also contains *GINNetManifest.MF*, the manifest file to build GINNet.jar, used by the GINNet *build.xml* script.

2.7.2 Packages

GINNet package (called *fr.loria.cortex.ginnet*) contains three main sub-packages:

- ***data***: regroups all data pre-treatments
- ***dynnet***: DynNet library package that provides all Java elements needed to manage several types of data-mining methods (like neural networks and trees) and that can be used separatly
- ***gui***: all graphical interface classes and main class called *GINNet*

Chapter 3

Neural network concepts

This chapter refers to Laurent BOUGRAIN thesis "Étude de la construction par réseaux neuromimétiques de représentations interprétables : Application à la prédiction dans le domaine des télécommunications" (14 november 2000).
And "Réseau de neurones Méthodologie et applications", G. Dreyfus, J.-M. Martinez, M. Samuelides, M. B. Gordon, F. Badran, S. Thiria, L. Hérault, Eyrolles, 2002

3.1 Corpus

3.1.1 Neural network categories

Neural networks can be separated in two categories depending on which task they have to perform :

- **Supervised neural networks** are used for prediction.
  With this kind of corpus, you know the nature of your data (you know what classification you want). And you want to teach your neural network how to predict results.
  - Regression neural networks are used for forecasting.
  - Discrimination neural networks are used for classification.
- **Unsupervised neural networks** are used for data analyses
  - Categorization neural networks are used to find categories for data by attributing non-predefined classes to each kind of data found.

3.1.2 Variable categories

Data can be divided in three types :

- **Attributes** (entries) are input data, observated values. They are necessary variables specifying attribute valuation.

  Examples: If you analyse iris classification, attributes are leaf lengths. (You can find iris database at `ftp://ftp.ics.uci.edu/pub/machine-learning-databases/iris/`). If your data concern the weather, it can be a binary value for shiny or not, raining or not, etc.

- **Labels** are output data. They specify the names you assign to each category of data. It is usually a string.

  Examples: If you want to classify iris species, labels can be "Iris-setosa", "Iris-virginica" and "Iris-versicolor". If you want to study boolean comportment, like xor operator, labels will be possible result : "true" or "false". If your neural network is aimed to recognize letters, labels will be "a", "b", ..., "z".

- **Targets** are output data. They correspond to numerical values of the last layer of a neural network.

  Examples: If you classify two types of data, the last layer will contain two neurons N1 and N2. Targets values will be (0,1) or (1,0), corresponding to (N1, N2) values.
  With regression network, you already have targets but no name for labels.

3.1.3 Corpus members

Corpus	**Variables**		
	Input	Output	
	attributes	labels	targets
Forecast	x		x
Classification	x	x	x
Unsupervised	x	x	
Cluster	x	x	

In the case of cluster corpus, labels are known, but are only used to interpret results at posteriori.

3.2 Competitive networks

3.2.1 Description

Unsupervised networks doesn't work by teaching correspondance between desired output and computed output but by data organization.
Competitive networks imply a competition between output neurons. Only winner neuron is modified in order to better caracterize current input.
Kohonen, Neural Gas, Growing Neural Gas and KMeans networks are based on competitive algorithms.

3.2.2 Quantification and classification

Quantification identify a prototype for each class. A prototype is a "mean vector", a representative node.
Classification associates a class to each pattern. This class is the one of the nearest prototype.

3.2.3 Comparison of competitive networks

Type of competitive network	**Applications**	**Topology**	**Specification**	**Learning**
K-means	Intra-class inertia minimization	No links between nodes	Groups are linked after learning	
Kohonen self-organizing maps	Example categorization and neighborhood constrainst between classes	Defined with problem. Generally a grid.	Groups are linked before learning (Number of neighbors and resolution are fixed)	Winner and neighbors are modified.
Neural gas		Constructed with an hebbian learning. Number of connections depends on lifetime of neurons.		For each pattern, a connection is created between the winner and the second node. Connections have a finite lifetime.
Growing neural gas	Data partitionment, quantification and caracteristic extraction	Constructed and changed with a competitive hebbian learning.	Classes are dynamical. The less represented area is the one where new prototypes are created.	New prototypes are periodically added where error is highest. Winner and neighbors are modified (like Kohonen).

Network topology In competitive networks, the network is divided in two layers.

- The first layer is the input layer (so it contains as many neurons as number of attributes)
- The second layer depends on the topology of the network.

3.3 Variable selection

Variable selection methods can be divided in three groups:

- **Filter** methods study the dependance between the variables (Mutual information, correlation, etc.) to select a subset of variables before using a model.
- **Wrapper** methods use a model (neural network, tree, statistics, etc.) as a blackbox to evaluate a subset of variables. A subset is obtained using a backward selection, a forward selection or a mix of them.
- **Embedded** methods select a subset of variables during the training stage.
  These methods, in neural network domain, can be divided in two types:
  - Connection suppression:
    - Optimal Brain Damage (OBD): with this method, there is no need to recompute network weights before deleting connections
    - Optimal Brain Surgent (OBS): is based on OBD, but is more complicated because error is not considered as quadratic
    - Flexible OBS (F-OBS): is only applied to one layer
  - Neuron suppression:
    - Optimal Cell Damage (OCD)
    - Unit-OBS: deletes neurons having less relevant connections

Chapter 4

Technical description

4.1 Overview

Diagram 4.1 represents GINNet components and their interactions.

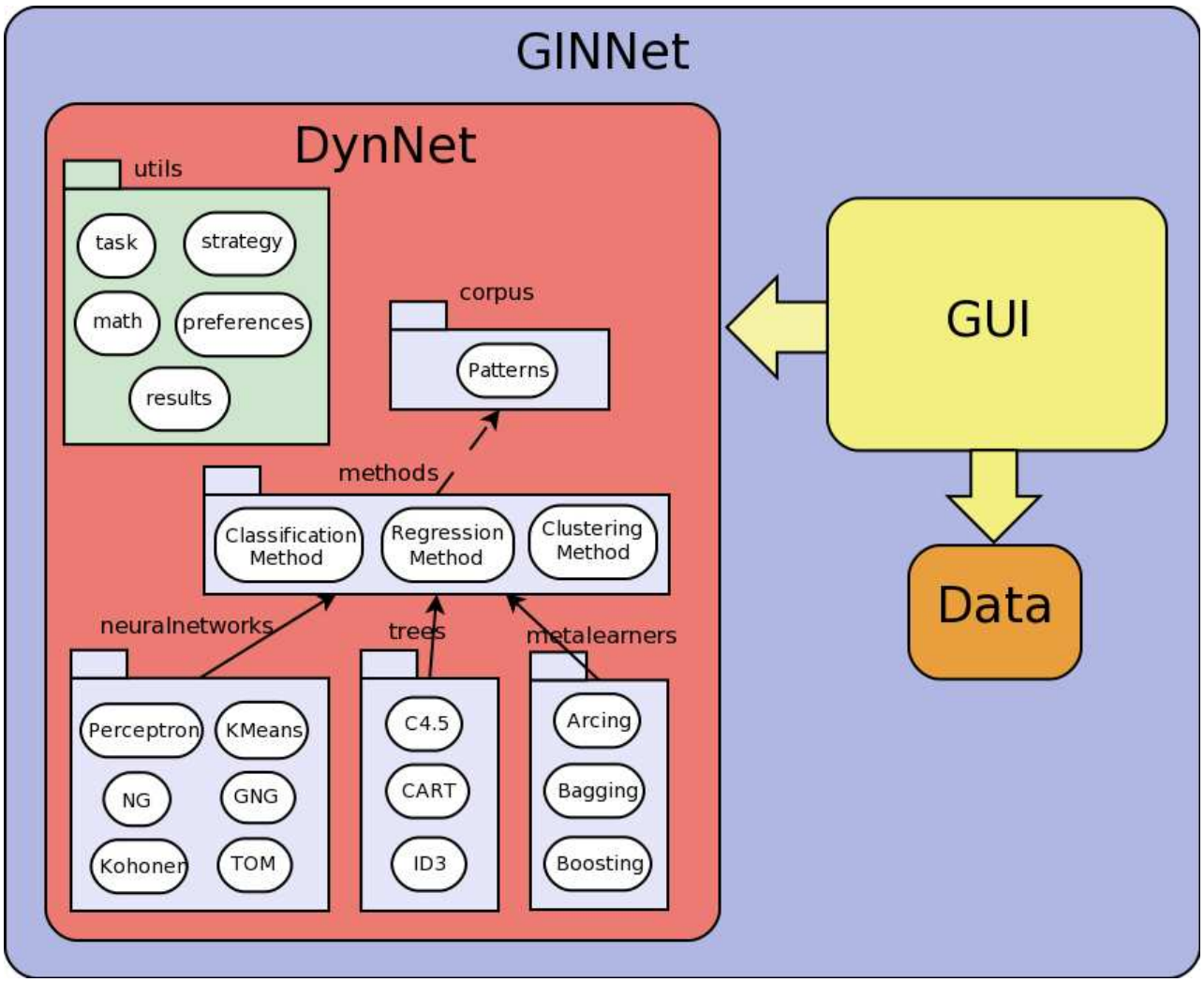


Figure 4.1: GINNet organization

As we can see, view classes (*gui* package) are separated from model classes (*data*, and *dynnet* packages).

4.2 DynNet

DynNet package is composed of five sub-packages:

- ***corpus***: contains classes to manipulate patterns
- ***metalearners***: meta-leraning implementation
- ***methods***: describes data-mining methods types and hierarchy. All concrete models are derived from those interfaces.
- ***neuralnetworks***: all basic classes relative to neural networks concepts and especially:
  - *competitivenetwork*: unsupervised network structure
  - *supervisednetwork*: supervised network structure
  - *models*: all implemented neural network models: Kohonen, Neural Gas, Growing Neural Gas, K-Means and Perceptron
- ***trees***: decision trees library that contains a sub-package:
  - *models*: tree implementations: C4.5, CART, ID3
  - *models*: information functions
- ***utils***: contains different utilities that can be used anywhere in DynNet or GINNet:
  - *criteriastrategy*: criteria used by variable selection
  - *math*: all mathematical methods usefull for DynNet
  - *preferences*: user preferences package
  - *results*: package used to display method'results in HTML
  - *strategy*: used to easily implement and use Strategy design pattern
  - *task*: used to display long task progression informations

4.2.1 Corpus

A corpus is a set of pre-treated data, directly usable by a neural network. So it's an obligatory step before network creation.
Corpus describes patterns, i.e. input and output variables of the network. To manipulate network's patterns, we choose to directly use the pattern set (instead of copying all data into a network variable).
Three types of corpus have been defined, depending on which task they are meant for:

- **Categorization corpus** are used to cluster data, i.e. gather examples into homogeneous groups (with Kohonen, TOM, Neural Gas, Growing Neural Gas or K-Means)
- **Classification corpus** are used to classify, i.e. assign examples to pre-defined classes (with Perceptron or trees)
- **Forecast corpus** are used to make regression, i.e. predict one or several continuous variables (with Perceptron or CART trees)

Class diagram at figure 4.3 describes corpus architecture.
Patterns can be viewed as a matrix of input data where a line is a pattern and a column is attribute's values. Those objects (patterns and attributes) are manipulated with indexes. There are no type for a pattern or a list of attribute (it's just a choice of implementation).
But a type has been declared to manipulate an attribute value: the class *Attribute*, described in diagram 4.2. DynNet manages:

- numerical attributes, whose type is *NumericalAttribute*
- symbolical attributes, whose type is *SymbolicAttribute*

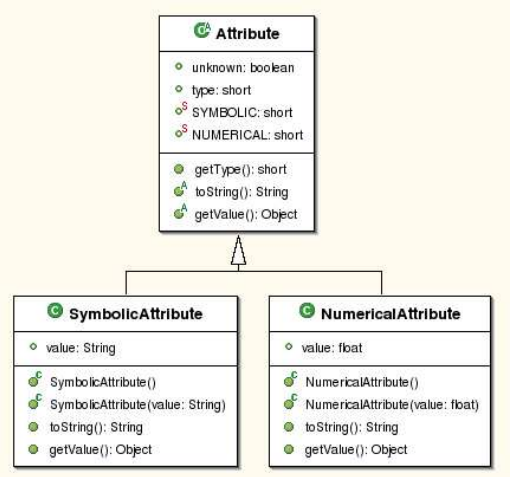


Figure 4.2: Attribute class diagram

NB: Corpus interfaces are used to separate the hierarchy of abstraction from concrete implementations of corpus, like described by Bridge design pattern. This implementation should be transparent for corpus user.

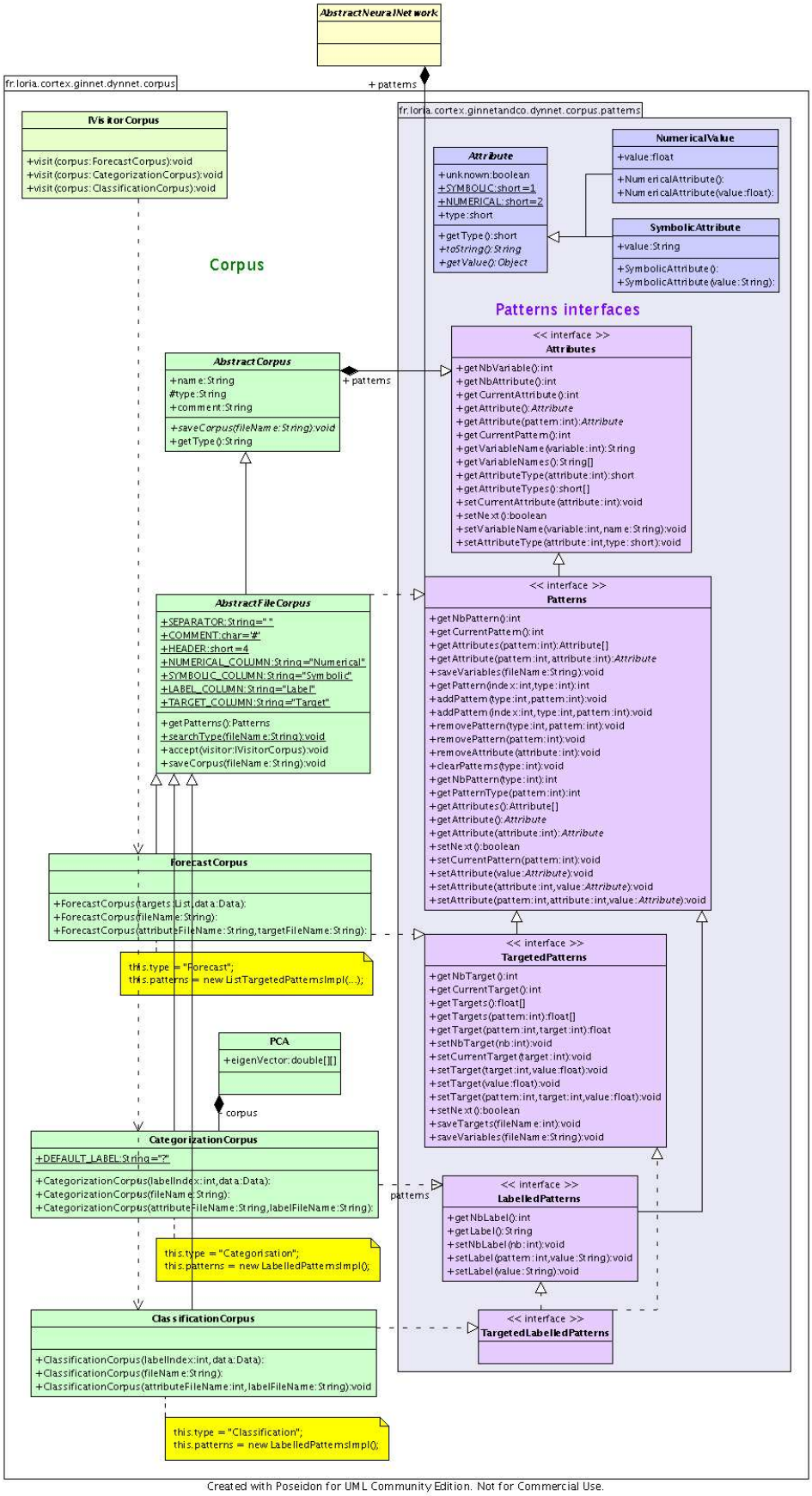


Figure 4.3: Corpuses class diagram

Corpus files

A file format has been defined for corpus:

- Fisrt lines are comments lines and begin with #.
- Next line is the name of the corpus.
- Next line is the type of corpus: *Categorization*, *Classification* or *Forecast*.
- Next line describes types of column. Columns are separated by a separator (a comma by default).
  - For each attribute, the label *Symbolic* or *Numerical*.
  - For each target, the label *Label* or *Target*.
- Next line specifies, for each column, its name.
- All following lines describe data. A line correspond to a pattern and is composed of:
  - For each variable, its value,
  - At the end of the line, the type of pattern: *(Learn)*, *(Test)* or *(Validation)*.

Figure 4.4 is a example of content of a discrimination corpus, named *golf*, that contains four attributes and a label that can be *Play* or *Don't Play*. It contains ten learn patterns, two test patterns and two validation patterns.

```
# Golf corpus
#
golf
Classification
Symbolic,Numerical,Numerical,Symbolic,Label,Target,Target
Outlook,Temperature,Humidity,Windy,Practice,Don't Play,Play
sunny,85.0,85.0,false,Don't Play,1.0,0.0,(Learn)
sunny,80.0,90.0,true,Don't Play,1.0,0.0,(Learn)
overcast,83.0,78.0,false,Play,0.0,1.0,(Learn)
rain,70.0,96.0,false,Play,0.0,1.0,(Learn)
rain,68.0,80.0,false,Play,0.0,1.0,(Learn)
rain,65.0,70.0,true,Don't Play,1.0,0.0,(Learn)
overcast,64.0,65.0,true,Play,0.0,1.0,(Learn)
sunny,72.0,95.0,false,Don't Play,1.0,0.0,(Learn)
sunny,69.0,70.0,false,Play,0.0,1.0,(Learn)
rain,75.0,80.0,false,Play,0.0,1.0,(Learn)
sunny,75.0,70.0,true,Play,0.0,1.0,(Test)
overcast,72.0,90.0,true,Play,0.0,1.0,(Test)
overcast,81.0,75.0,false,Play,0.0,1.0,(Validation)
rain,71.0,80.0,true,Don't Play,1.0,0.0,(Validation)
```

Figure 4.4: An example of classification corpus file content

4.2.2 Data-mining methods

methods package has been added to provide a hierarchy for data-mining models. There are all interfaces that describe common comportements. Class diagram 4.5 describes *methods* package and how it is used by datamining methods implemented in DynNet.

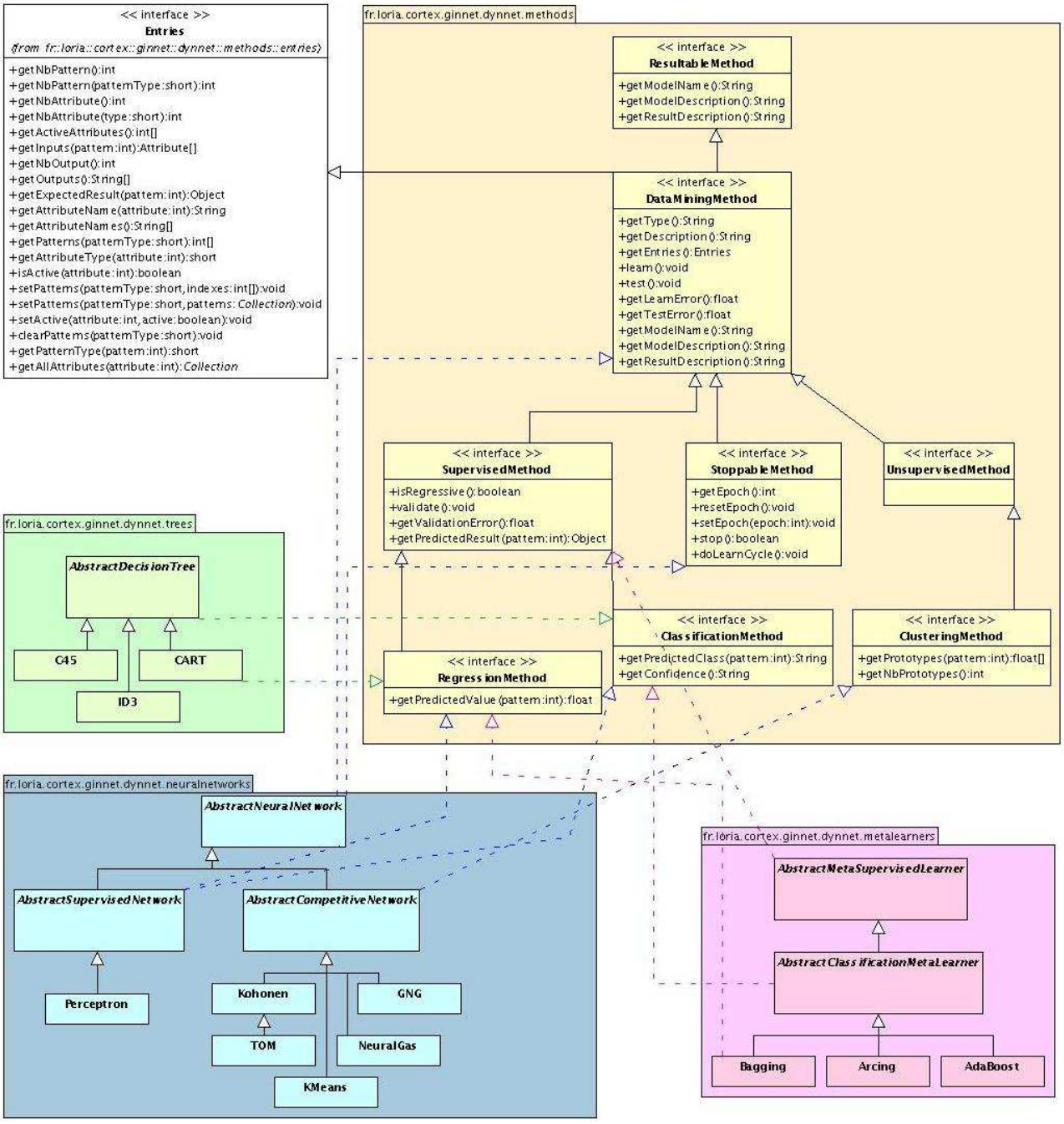


Figure 4.5: Data-mining methods hierarchy and models implementing them

All datamining methods displays HTML results.
DataMiningMethod extends *Entries* because a model can't be separated from its data in DynNet.
DynNet, in version 1.2.5, implements twelve concrete datamining methods:

- Decision trees (C4.5, CART and ID3) are classification methods. CART performs regression tasks too.
- Neural networks are stoppable datamining methods.
  - Supervised networks (Perceptron) are classification and regression methods.

  - Competitive networks (Kohonen, GNG, NeuralGas, KMeans and TOM) are clustering (and so unsupervised) methods.

- Supervised meta-learners are supervised methods. *ClassificationMetaLearner* concerns classfication methods (Bagging, Arcing and AdaBoost). Bagging meta-learners is also a regression method.

As we can see, implementing *methods* interfaces is the way to describe what methods can do and to easily use them is GINNet environment.

4.2.3 Neural networks

AbstractNeuralNetwork is the super class of all networks in GINNet. It is included into *neuralnetworks* package. It extends *DataMiningMethod* and *StoppableMethod* interfaces.
Learning rate update is made by a LearningRateFunction and learning end is decided by a StoppingFunction. There are both strategies.
Figure 4.6 shows neural networks package.

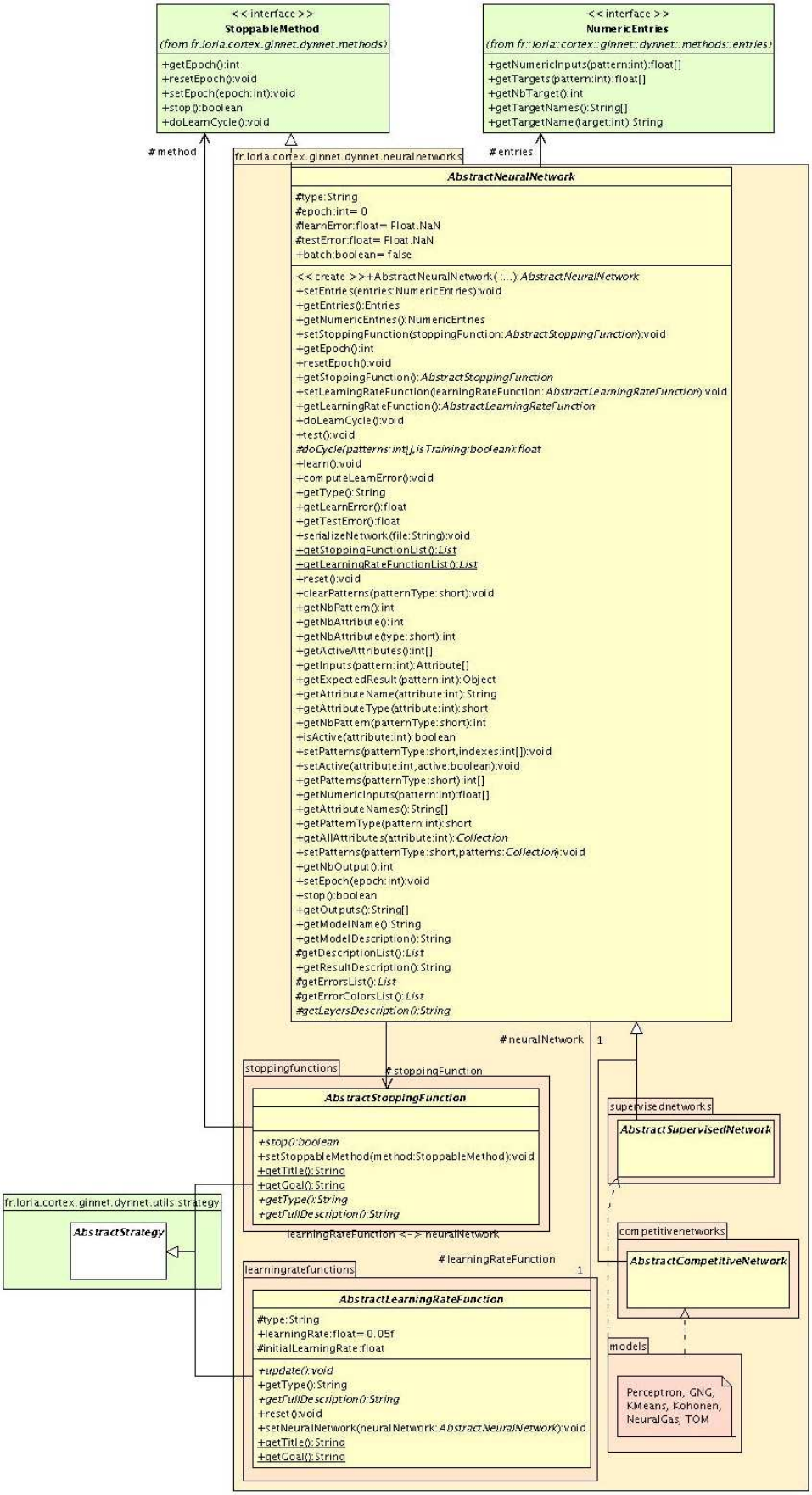


Figure 4.6: Neural networks class diagram

4.2.4 Supervised networks

Figure 4.7 shows the architecture of supervised network package.
A *NeuronUnit* is a neuron and a *SupervisedPopulation* is a layer. *InputSupervisedPopulation* is the class of input layer and provides methods to activate (add) and desactivate (remove) input neurons, useful for pruning. *Projection* describes connections between two populations and can contain bias.

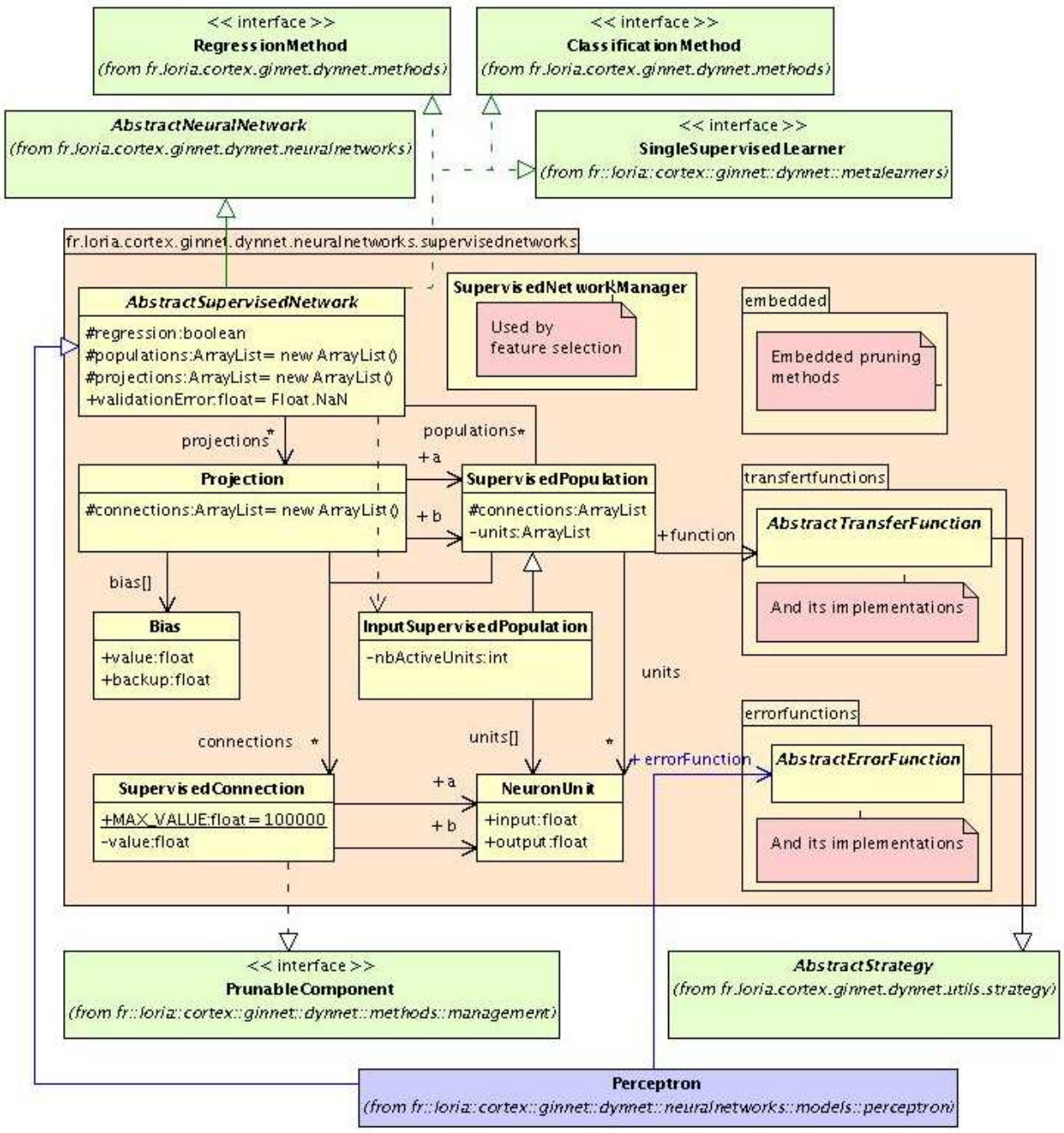


Figure 4.7: Supervised networks class diagram

Perceptron is currently the only concrete supervised network implemented in DynNet.

Perceptron

Figure 4.8 describes Perceptron classes.
A Perceptron is described by its layer layout (*sizes*), its error function and its classification error function (used to compute misclassification error), its momentum and its confusion matrix. It can have bias or not. Inputs (neurons of input layer) can be activate or desactivate with pruning methods.
PerceptronProjection and *PerceptronConnections* classes has be defined to extend classical *Projection* and *SupervisedConnection* (defined for any supervised network). *PerceptronConnection* keeps backup values of connection value and delta in memory, in order to restore best Perceptron configuration and *PerceptronProjection* uses it.
As a Perceptron can be saved (in binary format), all its components are serializable. Perceptron also uses *NumericEntries*.

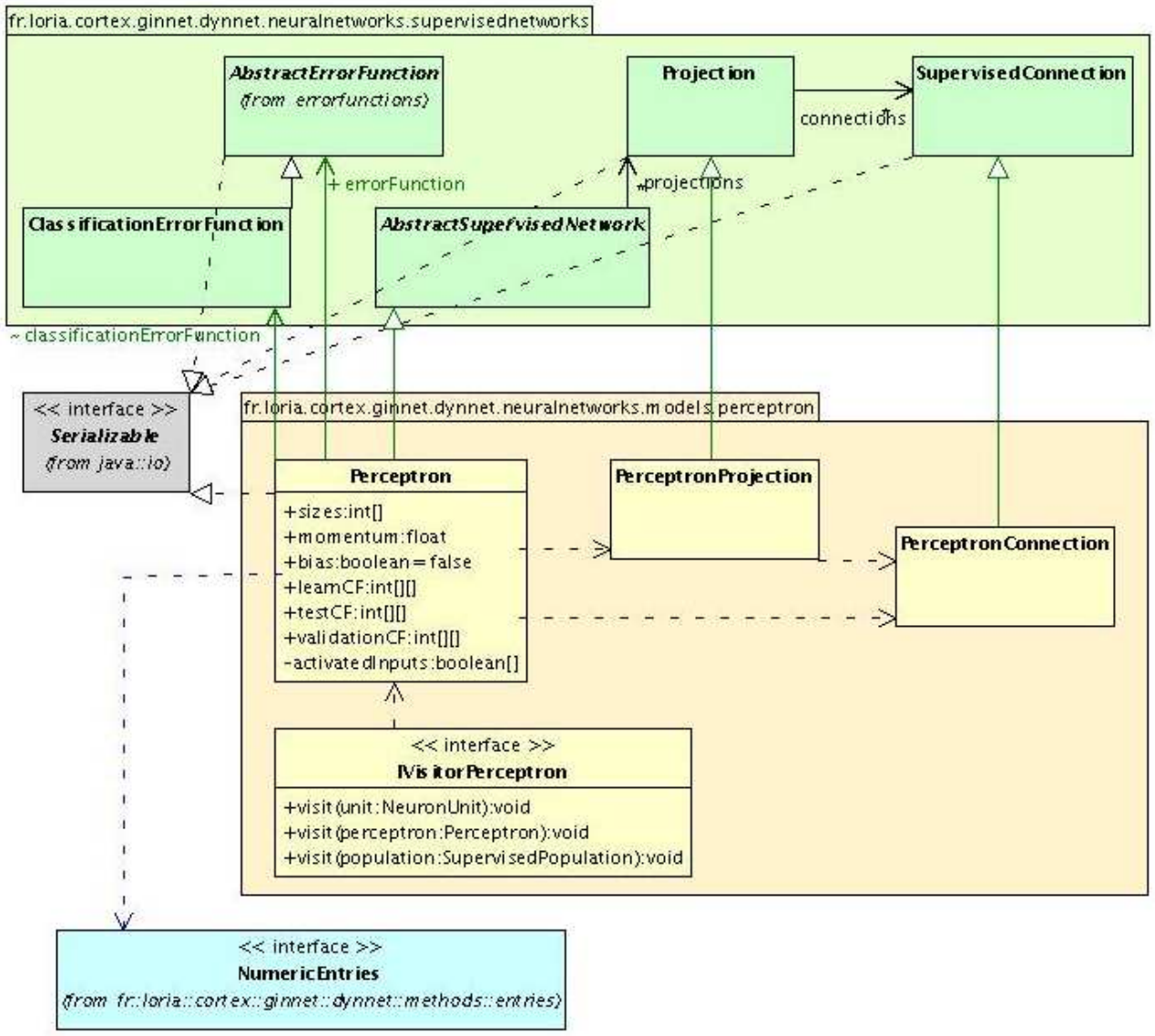


Figure 4.8: Perceptron class diagram

4.2.5 Competitive networks

Competitive network architecture is simplier as shown in figure 4.9.
CompetitivePopulation corresponds to output layer of any competitive network.
A *Node* is a neuron and a *Bond* is a connection between two neurons.
Topology is used by some competitive network implementations like Neural Gas and Growing Neural Gas.
To parametrize output adaptation, two specific strategies are linked to a competitive network:

- A distance function that describe how to compute the difference between desired outputs and real outputs, in order to correct weights.
- An aggregating function that determine how to update prototypes.

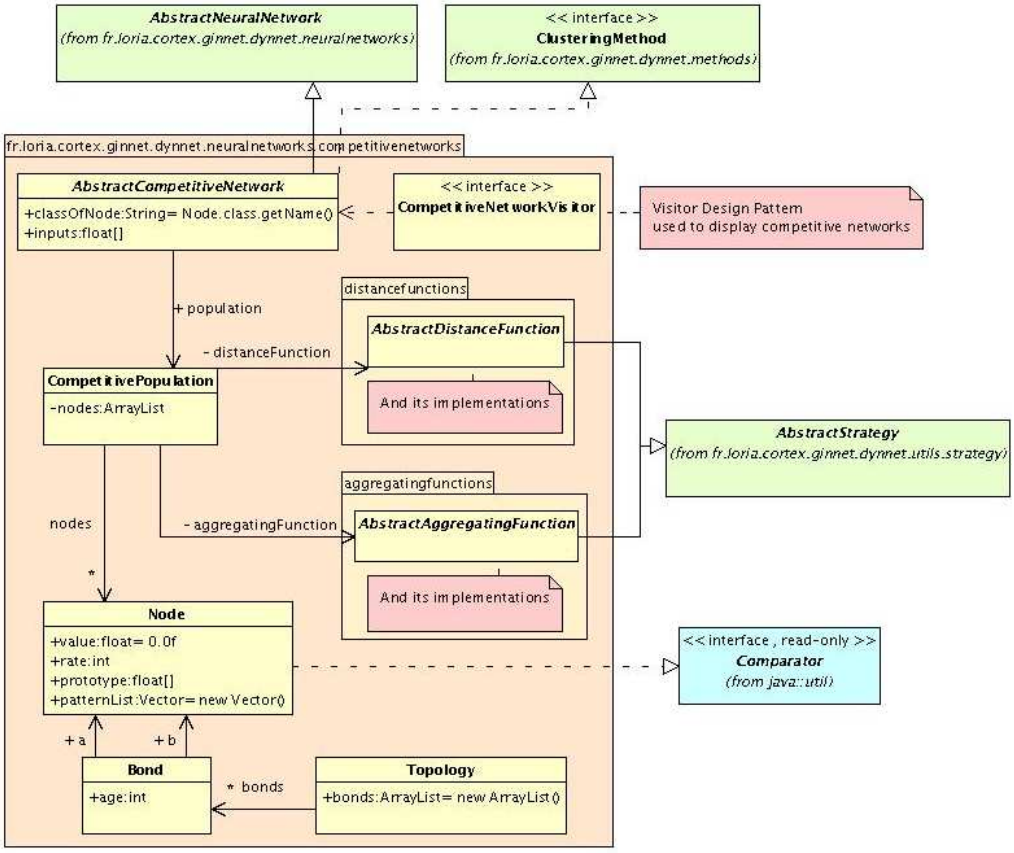


Figure 4.9: Competitive networks class diagram

Currently, five models implements competitive networks: GNG, KMeans, Kohonen, NeuralGas and TOM.

4.2.6 Decision trees

What are decision trees ?

Decision trees are data-mining methods used to split populations into homogeneous groups, according to discriminant variables, and in function of known target.
A decision tree is a predictive model; that is, a mapping of observations about an item to conclusions about the item's target value. Each interior node corresponds to a variable; an arc to a child represents a possible value of that variable. A leaf represents the predicted value of target variable given the values of the variables represented by the path from the root.
Figure 4.10 shows an example of decision tree that predict if you should play golf or not, according to outlook (a symbolical variable), humidity (a numeric variable) and windy (a boolean variable). Data of this example also contains another numeric variable (temperature) that is needed by this tree to predict target without any error.

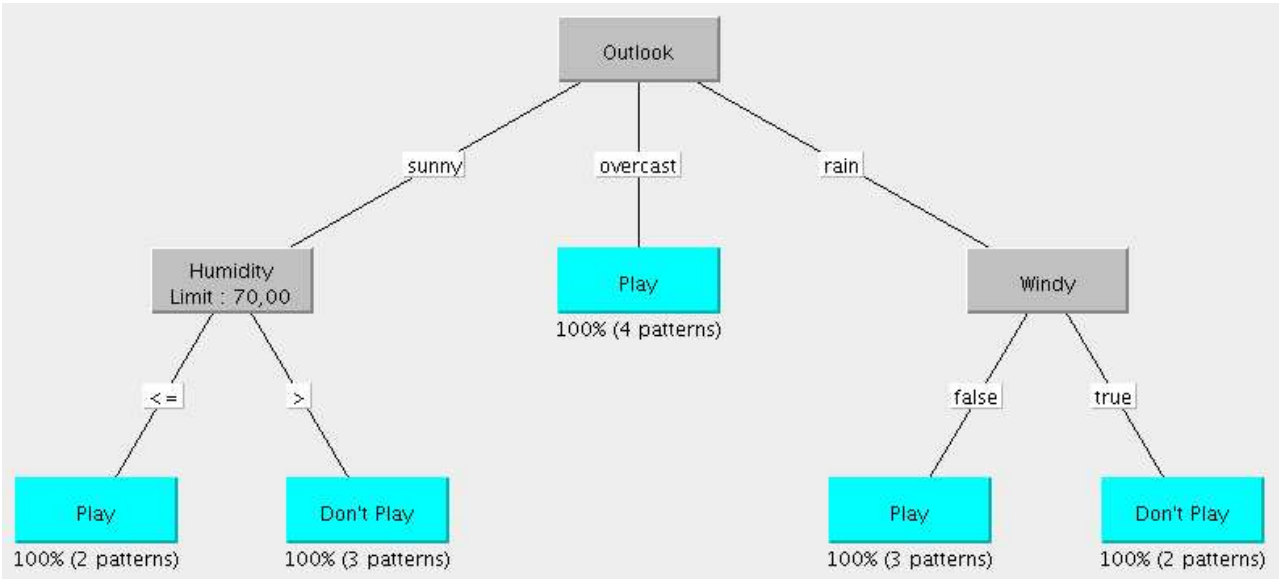


Figure 4.10: An example of a decision tree

How are they implemented in DynNet ?

Figure 4.11 shows decision trees package organization in DynNet.

AbstractDecisionTree is the super-class of all decision trees. At the moment, three algorithms have been implemented in DynNet : C4.5, CART and ID3. They are all regrouped in *models* package. All decision trees are classification methods and some can also perform regression tasks, like CART.
As decision trees are trees, they are implemented with a recursive structure: all nodes are described by the *TreeNode* class. Each node keeps a reference to its parent node and a decision tree has only one reference to the root node.
Information function describe the way information quantity is evaluate. All decision trees have their own information function. They are included in *informationfunction* package. At the moment, three algorithm have been implemented:

- Entropy (used by C4.5 and ID3),
- Gini (used by CART),
- Variance (used by CART).

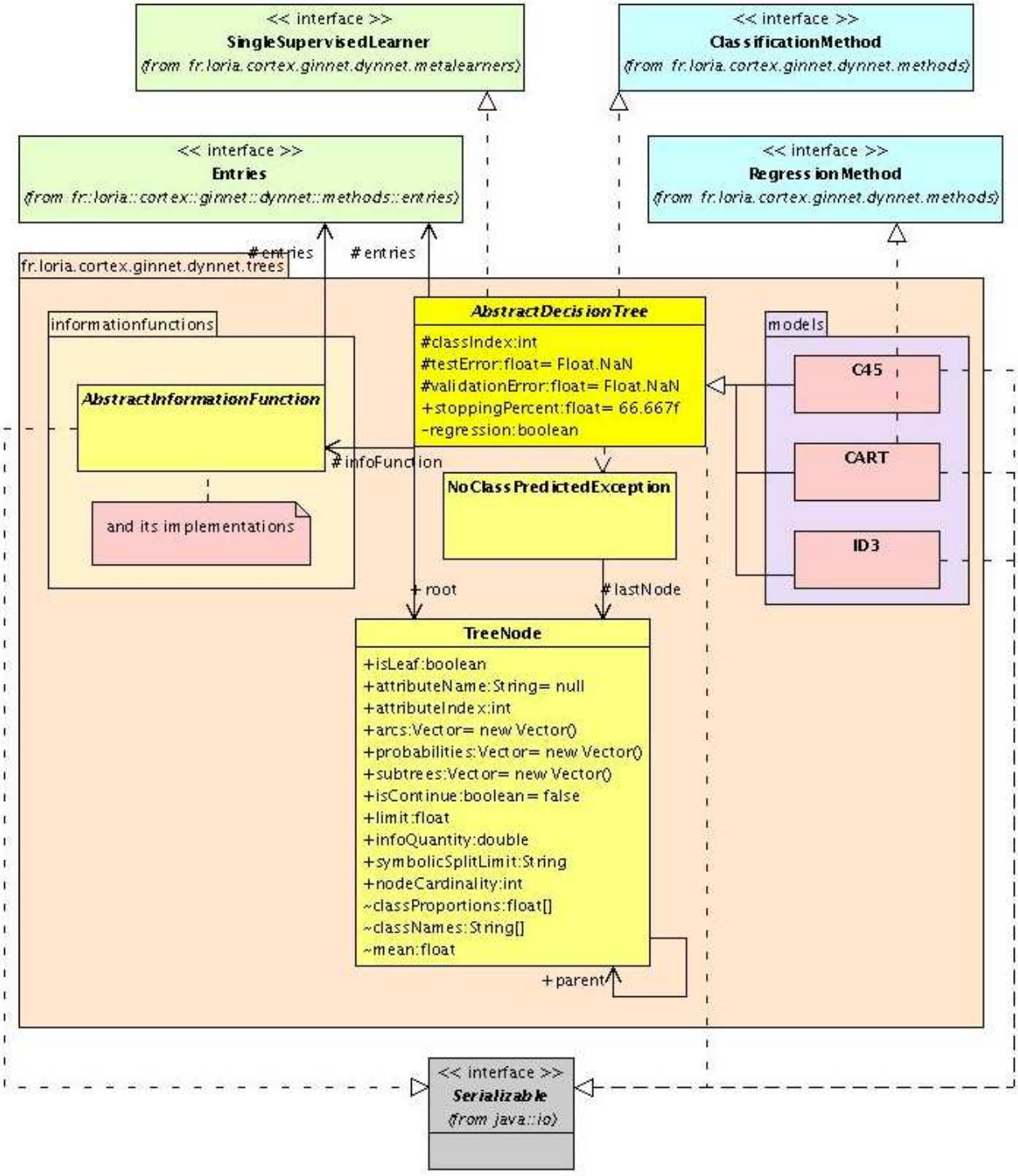


Figure 4.11: Decision trees class diagram

4.2.7 Meta-learners

Meta-learners are data-mining methods based on meta-learning. They use sub-learners (or weak learners), that are simple learning models, and learn how to distribute samples among them (sampling) and how to combine their results (combining) in order to optimize performances.

Figure 4.12 shows meta-learners package organization.

AbstractMetaSupervisedLearner is the super-class of all meta-learners. A meta-learner has:

- several sub-learners, whose class is *SingleSupervisedLearner*, and regrouped in an array whose size is given by the attribute *nbLearners*
- a factory to create those sub-learners
- a sampler: a class extending *AbstractSampler* and defining an algorithm to distribue samples among sub-learners
- a combiner: a class implementing *Combiner* interface and defining an algorithm to combine sub-learner results to get final result
- patterns (*entries*), whose type are *Entries*

All sub-learners of a meta-learner have the same concrete type. Actually, two classes implement *SingleSupervisedLearner* interface: *AbstractSupervisedNetwork* and *AbstractDecisionTree*. This means that sub-learners used by any meta-learner can be a Perceptron, a C4.5, a CART or an ID3. If you want another learning method to be used as a sub-learner, add an implementation of *SingleSupervisedLearner* in its declaration and check its instantiation in *SingleLearnerFactory*.
samplers package regroups classes used for sampling and *combiners* package regroups classes used for combining.

At the moment, three meta-learning models are implemented in DynNet. They are all classification methods:

- **Arcing** (Adaptatively Resample and Combine), that uses *ArcingSampler* for sampling and *MajorityVoteCombiner* for combining
- **Bagging** (Bootstrap AGGregatING), that uses *BaggingSampler* for sampling and *MajorityVoteCombiner* for combining
- **AdaBoost** (ADAptative BOOSTing), that uses *AdaBoostSamplerAndCombiner* for sampling and combining

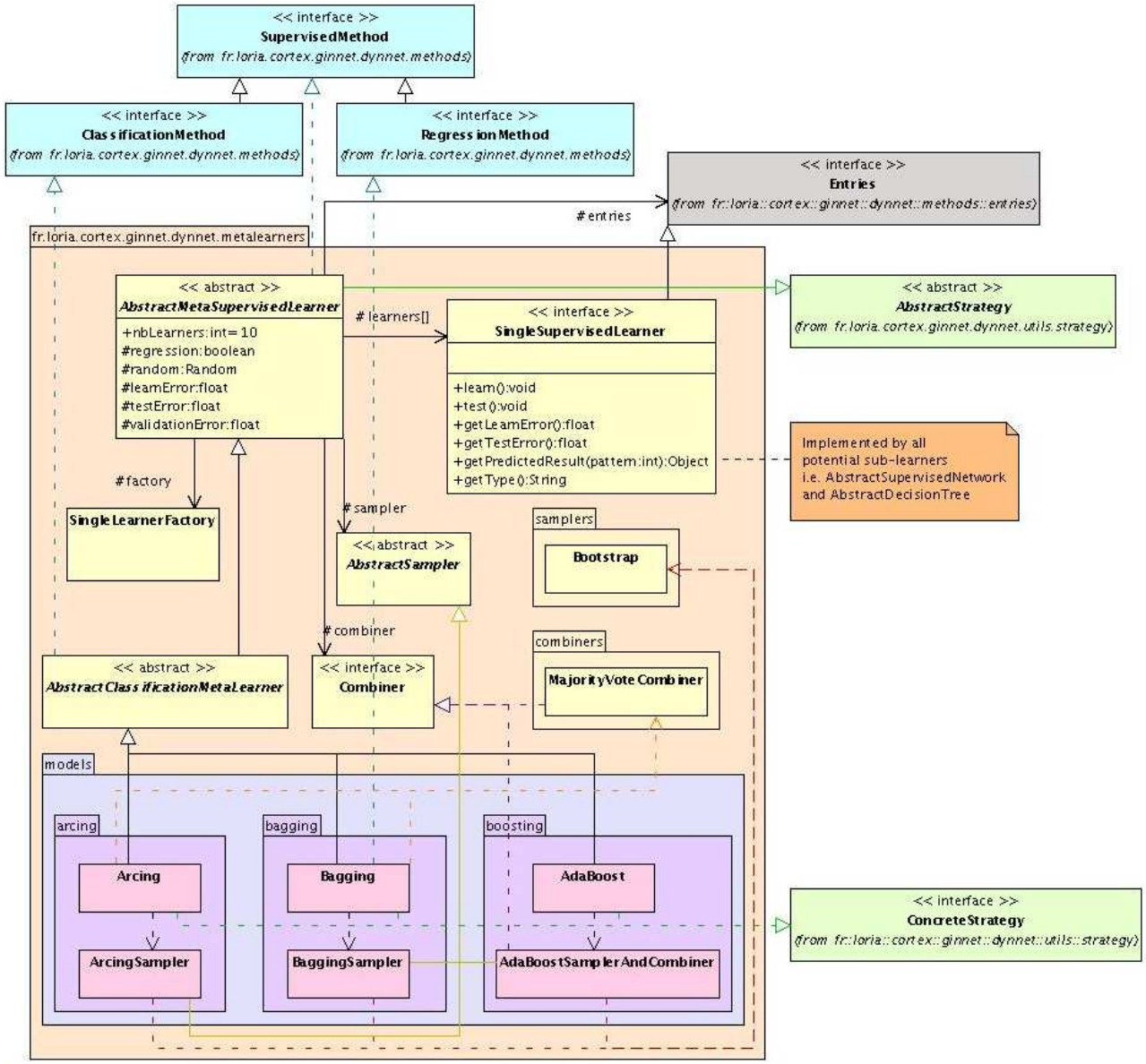


Figure 4.12: Meta-learners class diagram

4.3 GUI

GINNet is the main class and is GINNet frame. It's a Singleton.
Figure 4.13 is a screenshot of GINNet. We can see that GINNet frame is composed of a menu bar and zero or more tabs. Here, current tab (active tab) is a network tab (an extension of *AbstractMethodTab* class), splitted in two parts: main panel above and command panel below.

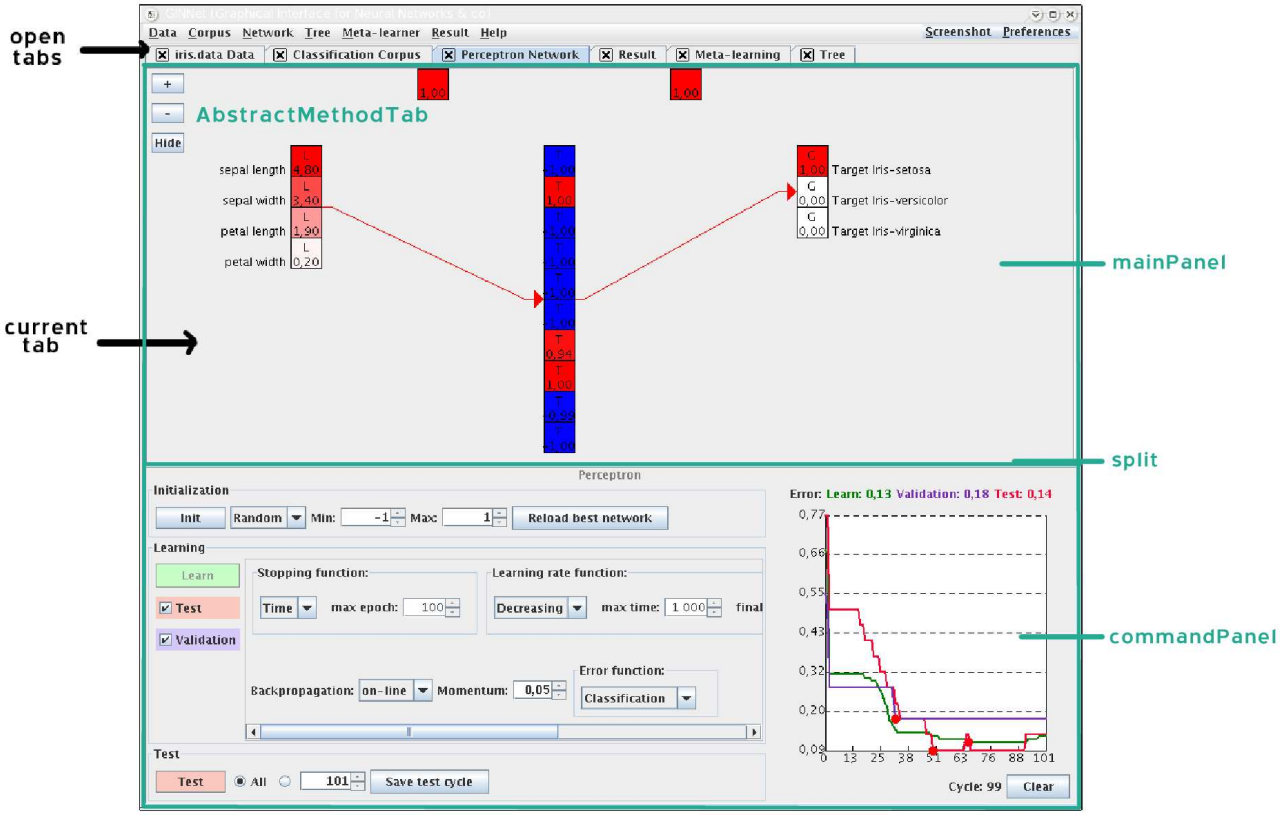


Figure 4.13: A view of GINNet frame

Figure 4.14 is an incomplete diagram of GUI architecture.
All graphical components are linked to *GINNet* main class. Tabs presented are attributes of *GINNet* and are all commandable tabs available. (Commandable tabs are tabs with a command panel).

4.3.1 GINNet thread

GINNet treatments, and especially network display, require most process time. That's why computations are performed in a separated thread, the main Thread that runs in GINNet (called *fr.loria.cortex.ginnet.gui.GINNetThread*). This allow the user to click on "Stop" button while learning for example.
N.B.: GINNetThread is a Singleton.

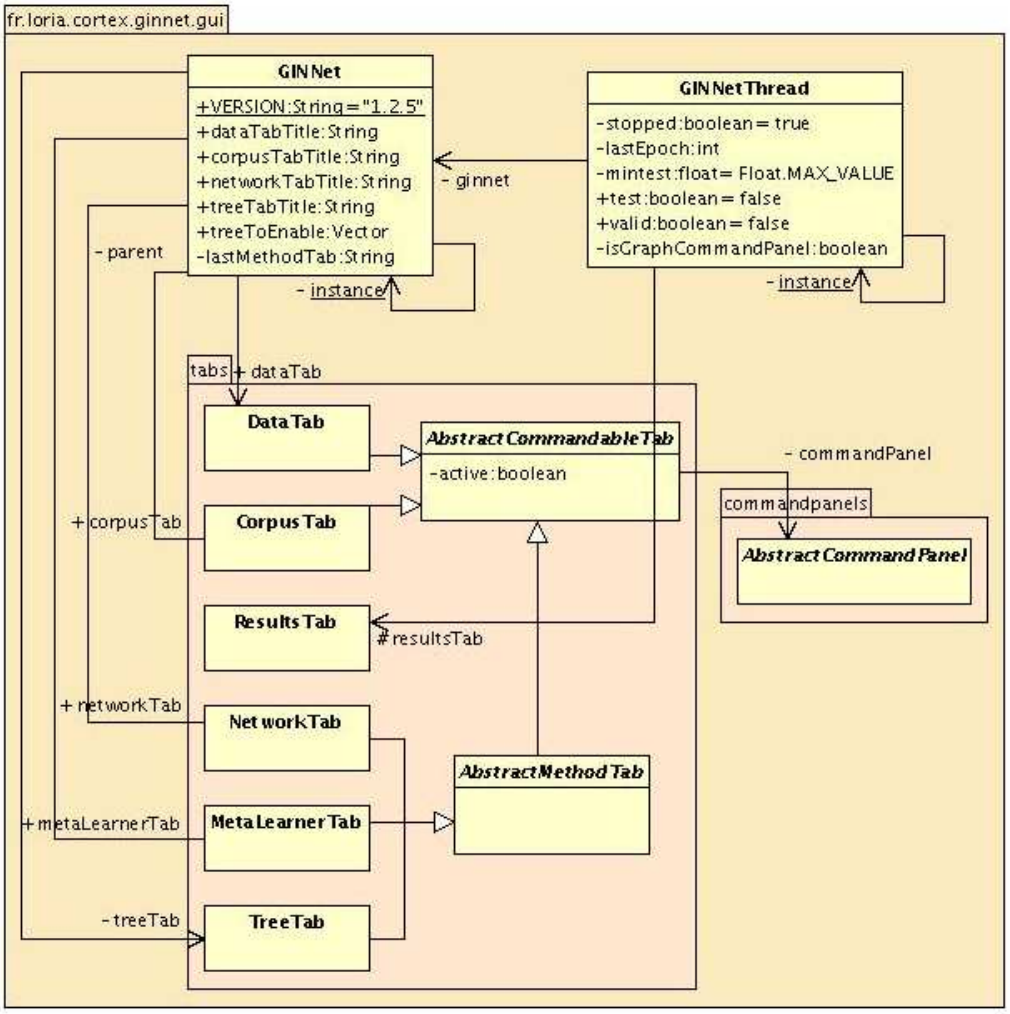


Figure 4.14: Partial GUI class diagram

4.3.2 Network command panels

Package *fr.loria.cortex.ginnet.gui.commandpanels.networkcommandpanels* manages all network command panels. Command panels are displayed at the bottom of the frame and allow the user to interact with models. Figure 4.15 presents an UML overview of this package.

To implement a control panel for a network (or just to change one), you need to know what contains default panels and where to add new elements. Figure 4.16 shows AbstractNetworkCommandPanel parts.

Add new learning graphical elements only relative to a specific neural network type into *personalLearnPanel*. Initializations and testing elements should be included into *initPanel* and *testPanel*.

4.4 Remaining problems

GINNet and DynNet architecture and code details still contain heaviness and programming awkwardness. We try to list them below:

- All tabs are created in *GINNet* class during its initialization. This makes the code heavy.

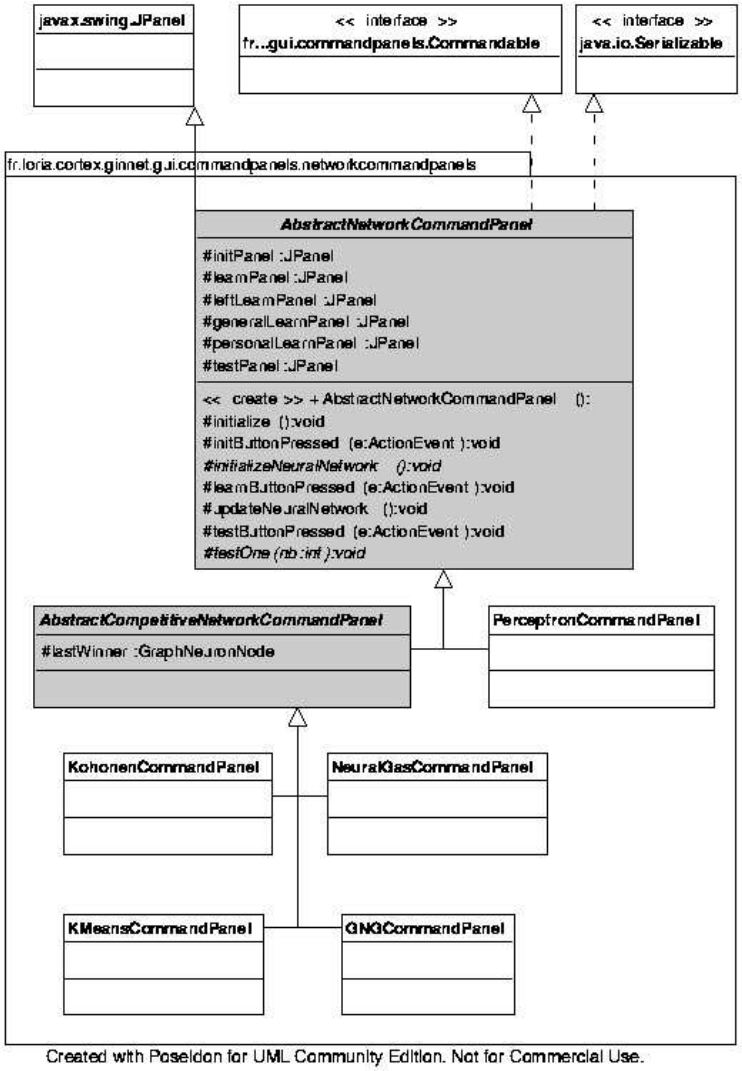


Figure 4.15: Network command panels class diagram

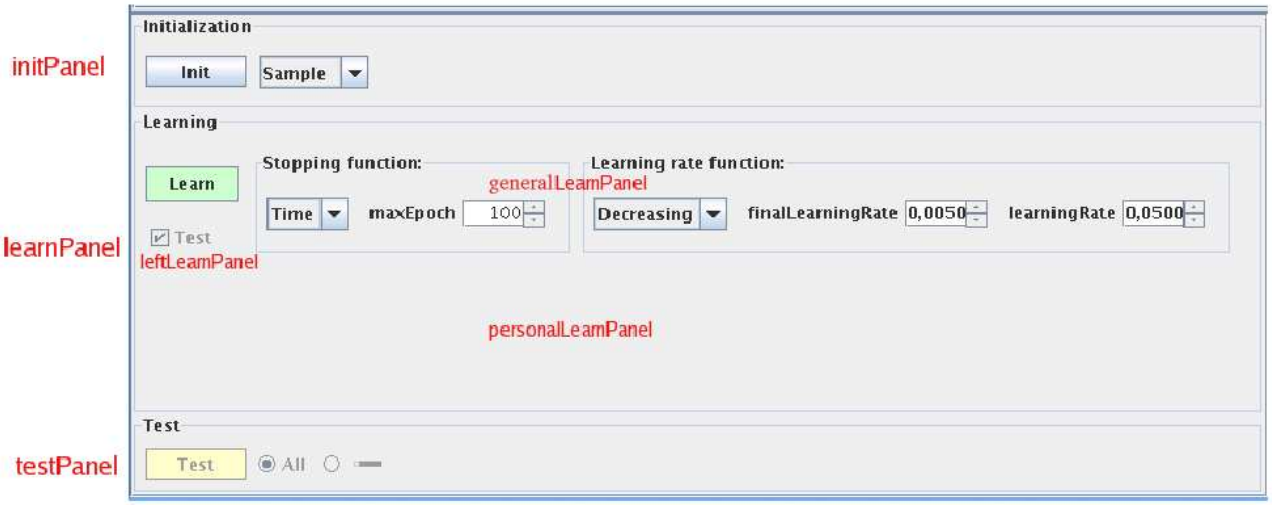


Figure 4.16: Screenshot of an empty network command panel

Chapter 5

GINNet facilities

5.1 Changing DynNet or GINNet version

To edit DynNet version:

- Change the constant named *VERSION* in *fr.loria.cortex.ginnet.dynnet.utils.DynNetConstants* class
- Change the constant named *DynNet.version* in *build.xml* and *DynNet.build* files (in utils/ant directory)

To edit GINNet version:

- Change the constant named *VERSION* in *fr.loria.cortex.ginnet.gui.GINNet* class
- Change the constant named *GINNet.version* in *build.xml* and *GINNet.build* files (in utils/ant directory)

5.2 Implementing strategies

5.2.1 Strategy design pattern

Strategy design pattern defines a family of algorithms, encapsulates each one, and makes them interchangeable. Strategy lets the algorithm vary independently from clients that use it.
You can find a complete description at
`http://www.dofactory.com/Patterns/PatternStrategy.aspx`.

5.2.2 Strategies in GINNet

GINNet uses several strategies in different configurations.
For example, each neural network needs a stopping function to decide if learning is finished or not. Three strategies are implemented for this stopping function :

- *ErrorStoppingFunction* stops the learning when given error is reached
- *TimeStoppingFunction* stops the learning after a given number of iterations
- *PermutationStoppingFunction* stops the learning when there are no more permutation (only for KMeans)

Other strategies concern learning rate functions, error functions, transfert functions, kernel functions, topology functions, and so on.

5.2.3 How to implement strategies in GINNet

An easy way to implement those strategy and incorporate them dynamically in any Swing interface has been set.

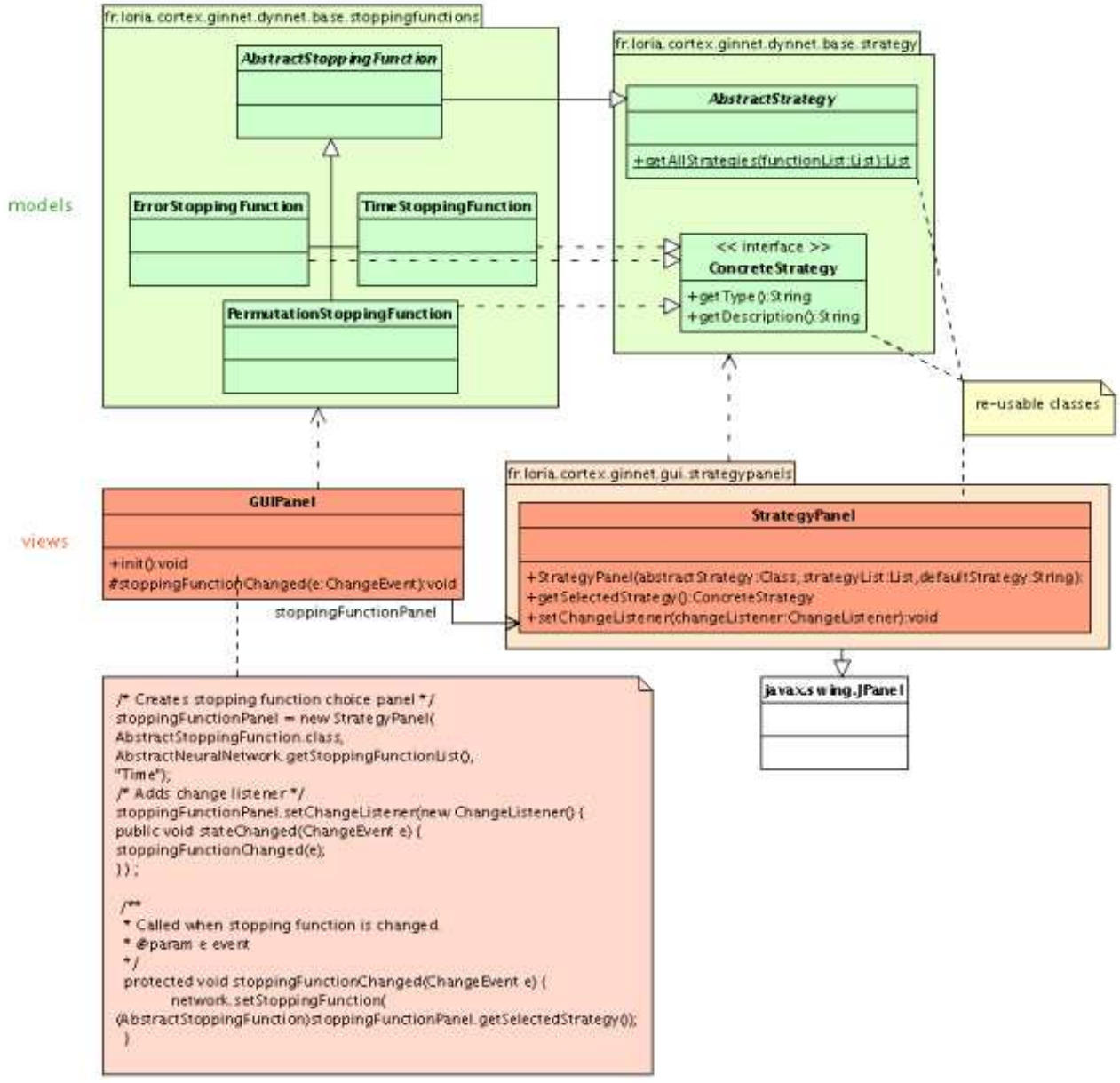


Figure 5.1: UML Diagram of stopping function strategy implementation

Example

UML diagram of stopping function implementation example is given in figure 5.1.

Strategies are defined by extending strategy classes and are totally independent from view.
A view can be created for any strategy thanks to generic StrategyPanel class. Java reflexivity is used to instanciate concrete classes with default constructor (without parameters) and have access to strategies methods.

First step : Strategy implementations First, *fr.loria.cortex.ginnet.dynnet.utils.strategy* package gives a structure for every strategy.
AbstractStrategy abstract class is a super-type for strategies and ConcreteStrategy interface is implemented by each concrete kind of strategy.
In example, AbstractStoppingFunction is an AbstractStrategy and ErrorStoppingFunction, TimeStoppingFunction and PermutationStoppingFunction are ConcreteStrategy.

Abstract strategy defines common fields and methods for all strategies and especially at lest one abstract method whose comportement depends on concrete implementations. For example, AbstractStoppingFunction declares abstract method *public abstract boolean stop()*.
To stop the network, strategies must have access to specific informations about associated neural network. So AbstractStoppingFunction declares a field of type *AbstractNeuramNetwork*. (If this field was public, it will be parametrizable in GUI).

Concrete strategies implement abstract method. The parameters of this method can not depend on concrete strategies, so they have access to required informations thanks to common object: the neural network. For exemple, TimeStoppingFunction stop() result will depend on the *epoch* field of the neural network, and ErrorStoppingFunction on its errors.
Remind that concrete strategies must implement a default constructor.

Second step : View implementation Now, if you want to add a panel to select a network's stopping function, you will create a StrategyPanel object. To be instanciated, this object must specify

- its abstract strategy : the class AbstractStoppingFunction
- the list of available strategies: a List object
  This list depends on network type (KMeans or not). So a method called *public static List getStoppingFunctionList()* has been defined in AbstractNeuralNetwork to get the list of available concrete strategies. This method is overridden in KMeans class.
- the name/type of default strategy selected : for example "Time"

And you need to listen to strategy change events by setting a change listener.
The resulting panel is in figure 5.2 and 5.3.

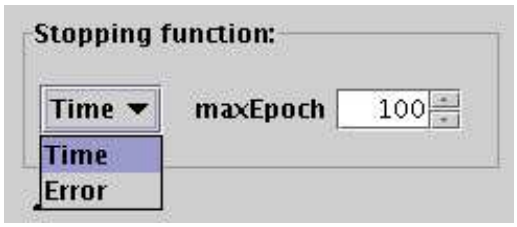


Figure 5.2: A screenshot of stopping function panel

Fields displayed are all public fields of ConcreteStrategy class, retrieved thanks to Java reflexivity. So don't add a public attribute to a strategy class (abstract or concrete) if you don't want it to be displayed on strategy panel.
Current supported field types are *int* and *float*, but FieldPanel class can be completed to edit String or other objects.
When adding a new strategy, you should be aware of compilation requirement, and compile all concrete strategies subsclasses (for example by adding a new line to *make_ginnet.csh*).

Figure 5.3: A screenshot of stopping function panel with another strategy selected

5.3 Progress bar

fr.loria.cortex.ginnet.dynnet.utils.task package provides two classes useful if you need to perform a long task and know that GUI may want to display a progress bar during computation.
Figure 5.4 gives an example of utilisation when computing a corpus' matrix correlation, that could take from less than one second to several hours.

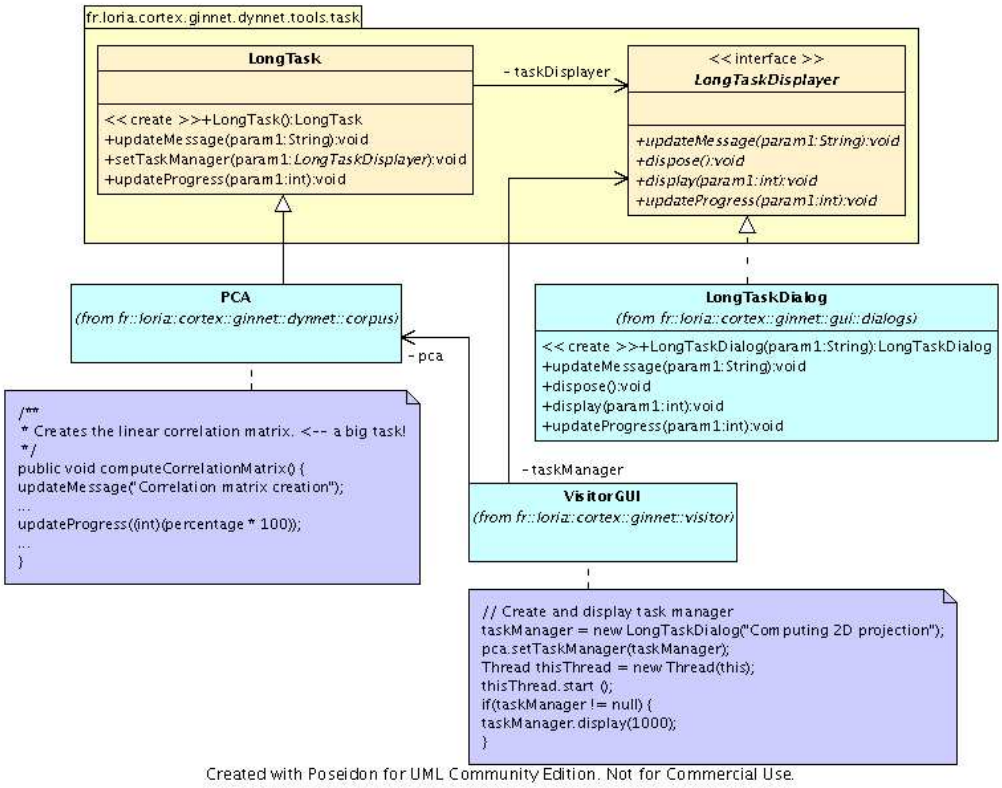


Figure 5.4: An example of long task computation and display

5.4 Tests

5.4.1 Test cases

Test cases are classes used to automatically verify class comportments. They are all regrouped in package *tests*. This package is not available on GForge releases, but you can ask for it (to *Laurent.Bougrain@loria.fr* or *Marie.Tonnelier@loria.fr*).
Those tests use the JUnit framework, and are easy to read and to test. JUnit main page is `http://www.junit.org`, you could also find a getting started with JUnit guide at `http://junit.sourceforge.net/`.
Ideally, every developper should add test cases for new features implemented. Read "JUnit Test Infected: Programmers Love Writing Tests" (at `http://junit.sourceforge.net/doc/testinfected/testing.htm`).

Eclipse provides utils to easily use JUnit test case.

How to launch a test case in command line?

JUnit provides test runners to run test cases. You can use graphical test runner (*junit.swingui.TestRunner*) or text test runner (*junit.textui.TestRunner*). Just pass the name of the test case to test as a parameter like this: *java -cp bin:ginnet/lib/junit.jar junit.swingui.TestRunner tests.tom.TOMTest*

5.4.2 Extreme programming

See `http://extremeprogramming.org/` (or in french at `http://extremeprogramming.free.fr/`).

Chapter 6

External tools

6.1 SubVersion

This chapter quickly explains how to work with SubVersion on command-line.
To perform *svn* commands, you need a subversion client installed on your computer, and you must work in subversion working directory (which contains *.svn* folder).

6.1.1 SubVersion notation

In SubVersion messages, a letter is displayed before name of files or folders described.
Actions refers to repository modification (in case of *commit* for example) or to your working copy (in case of *checkout* for example).

- **A** means that the file has been **Added**.
- **U** means that the file has been **Updated**.
- **D** means that the file has been **Deleted**.
- **M** indicates that the file has been **Modified**.
- **R** means that the file has been **Replaced** (i.e. old version has been deleted and a new one with the same name has been added)
- **G** indicates that the file has received new changes, while original file contained other modifications. But those two types of changes don't overlap each other and SubVersion has **merGe** the files.
- **C** means that a **Conflict** happened. Changes have been made on the two (or more) files and SubVersion can't merge them. So this conflict must be resolved by you.

6.1.2 How to get a local copy of last GINNet?

If you have no local copy

You want to get complete tree of the SubVersion referentiel into your local workspace.
Go to the folder in which you want to get *ginnet* folder and execute ***checkout*** command: *svn checkout svn+ssh://developername@scm.gforge.inria.fr/svn/ginnet*

If you already have a local copy

You want to get last changes made on GINNet.
Use ***update*** command to synchronize your working copy with most recent version of the repository : type *svn update*

6.1.3 How to modify arborescence?

Following actions won't be effective on repository until you commit changes and only file deletion is immediatly made on your working copy.

- If a file has been modified, SubVersion will detect it automatically and you have nothing to do.
- If you want to add a file, use ***svn add foo***. If *foo* is a folder, all its content will be added recursively.
- To delete a file or a folder and all its content, use ***svn delete foo***.
  Notice that deletion is not definitive, as SubVersion keeps history of the repository and allow you to come back to an older revision.
- ***svn copy foo foobar*** will create a new element named "*foobar*" as a copy of "*foo*" file or folder and its history will be kept by SubVersion.
- ***svn move foo foobar*** is equivalent to *svn copy foo foobar; svn delete foo* and so history is kept too.
- Use ***svn revert foo*** to cancel operation planed for *foo*.

6.1.4 How to see changes between 2 versions?

How to get information about revision?

- The command ***svn log*** gives log messages associated with revisions and modified path for all revisions.
- ***svn cat*** *--revision REVISION_NUMBER* will display a file as it was with given revision number.
- The command ***svn list*** *--revision REVISION_NUMBER foo* lists files in folder *foo* at given revision number.

How to see changes between your local version and repository?

- ***svn status*** is probably the most used SubVersion command.
  Used without argument, this command will detect all modifications made on your working copy.
  Here are the meanings of *svn status* letter code:
  - Common notation is described here : 6.1.1.
  - ***L foo***: SubVersion has a lock on file *foo*.
  - ***? foo***: *foo* is not managed by SubVersion.
  - ***! foo***: SubVersion manages *foo*, but it is missing or incomplete.
    This can happend if you interrupt a *checkout* or an *update*. In this case, use *svn update* or *svn revert*.
  - ***A + foo***: *foo* has been added with its original log.
  - ***M + foo***: *foo* has been added with its log and modified locally.
  - ...
- ***svn diff*** command details changes made on each file, using *diff* format.
  So added lines are preceeded by + and deleted lines by -. This command displays also name of modified files and shifting information.

How to see changes between your local version and last SubVersion version?

Use command: *svn diff - -revision 0*

6.1.5 How to put local version on the repository

It is wiser to verify changes made between your working copy and the repository before commiting them. To do this, see chapter 6.1.4 below.
And don't forget to use *svn add*, *svn delete*, *svn copy* or *svn move* to change folders or files status.
To put registred changes into the repository, use this command: *svn commit --message "message_describing_changes_commited"*
If you don't type any commit message, SubVersion will try to load your default text editor.

6.1.6 How to resolve conflicts?

You can detect conflicts via *svn status*. Conflicted files are labelled as ***C***. This means that server changes and your changes intersect and that you will have to decide which changes to keep.
Each time that there is a conflict, three events happen to help you to study and resolve it:

- SubVersion displays a ***C*** during update and remember that this file is in a conflicted state.
- SubVersion places conflict marks - specific characters delimiting conflicted areas - in the concerned file, in order to emphasize them.
- For each conflicted file *foo.ext*, SubVersion places three additional files in your working copy:

  ***foo.mine*:** This is your file before update.

  ***foo.rOLD_REVISION*:** This is your file before you change it.

  ***foo.rNEW_REVISION*:** This is the last version on the server of the conflicted file (without your changes).

A conflicted file can't be commited as long as those three temporary files are not removed.
So if you have a conflict, you can:

- Merge the files manually,
- or copy one of the temporary files into your working file,
- or execute *svn revert foo.ext* command, to undo changes.

Once a conflict is resolved, you have to inform SubVersion by the command: *svn resolved foo.ext* and this will remove the three temporary files.

6.2 Ant

Apache Ant (`http://ant.apache.org/`) is a software tool for automating software build processes. It is similar to make but is written in the Java language and is primarily intended for use with Java. Contrary to make, Ant is portable.
The most immediately noticeable difference between Ant and make is that Ant uses a file in XML format to describe the build process and its dependencies, whereas make has its own Makefile format. By default the XML file is named build.xml.

6.2.1 How to launch Ant build

Where to find Ant First, Ant program must be installed on your computer. You can find it on Ant web page or at LORIA in *tools* directory of CORTEX team (the executable is */users/cortex/tools/apache-ant-1.6.5/bin/ant*.

How to run Ant To run Ant script on command line, just type *ant*. By default, this will run *build.xml* file and run the target specified in the *default* attribute. If you want to run another targets, specify them as arguments.
You can also set properties on the command line. This can be done with the *-Dproperty=value* option, where property is the name of the property, and value is the value for that property.

6.2.2 How to launch GINNet and DynNet Ant build from anywhere

To run GINNet and DynNet Ant build file from any console, executes this command from *ginnet* directory :
ant -lib utils/ant/jsch-0.1.28.jar -lib lib/junit.jar -f utils/ant/build.xml -DGINNet.jar.password=*thefamouspassw*
-Dusername=*yourSSHusername* -Duserhome=*yourhomedirectory* -Dpassphrase=*yourpassphrase*
"*first target*"["*second target*" ...]

NB: On Linux, you can alias Ant command by executing this command: **alias ant2='ant -lib utils/ant/jsch-0.1.28.jar -lib lib/junit.jar -f utils/ant/build.xml -DGINNet.jar.password=*thefamouspassword* -Dusername=*yourSSHusername* -Duserhome=*yourhomedirectory* -Dpassphrase=*yourpassphrase*'**
Then you only have to type **ant2 "*first target*"["*second target*" ...]**

Some explanations :

- the first option **-lib utils/ant/jsch-0.1.28.jar** specifies that Ant must use JSch (Java Secure Channel) library. This library is needed to perform scp tasks.
- the second option **-lib lib/junit.jar** specifies that Ant must use JUnit library. This library is needed to perform tests.
- the third option **-f utils/ant/build.xml** specifies the XML build file to use.
- the option **-DGINNet.jar.password=*thefamouspassword*** specifies the password of GINNet.jar and is used when creating this jar.
- the option **-Dusername=*yourSSHusername*** specifies your username on InriaGForge and is used when performing scp tasks.
- the option **-Duserhome=*yourhomedirectory*** specifies your current home directory and is used when performing scp tasks in order to access to your SSH public key.
- the option **-Dpassphrase=*yourpassphrase*** specifies your passphrase on Inria GForge and is used when performing scp tasks. If you have no passphrase, just forgot this option.
- last arguments are targets. This Ant script provides many targets. To know which targets are available, see target names specified between <*target*> tags in the build file, they are all explained in *description* attribute. The most used targets are :
  - **"make all"** : Compiles DynNet and GINNet, create jars and launches GINNet.jar
  - **"launch GINNet"** : Compiles and launches GINNet
  - **"jar DynNet"** : Makes DynNet.jar
  - **"jar GINNet"** : Makes GINNet.jar
  - **"javadoc all"** : Makes DynNet and GINNet JavaDoc
  - **"scp GINNet.jar"** : Copies GINNet.jar to GINNet web site
  - **"make guide"** : Compiles guide and creates guide.pdf file
  - **"test"** : Compiles and runs all GINNet JUnit tests ; Results are saved in text files in testresults directory
  - **"zip GINNet"** : Creates a zip file containing GINNet source code
  - etc.

6.2.3 How to launch GINNet and DynNet Ant build from Eclipse IDE

To run Ant build file from Eclipse IDE, see section 6.3.2.

6.2.4 For more informations about Ant

You can find the official Ant user manuel at this address: `http://ant.apache.org/manual/index.html`.

6.3 Eclipse

Eclipse is a complete Java development tool (see `http://eclipse.org` for more informations).
Version 3.0.1 is installed in CORTEX tools.
You need to create a workspace directory into your home (so that all developpers can work separately on the same Eclipse application).
Launch Eclipse with this command (from LORIA): */users/cortex/tools/eclipse/eclipse -data workspace_directory*

6.3.1 First steps

Project creation

1. First step: Working copy checkouting
   As last GINNet version is under SubVersion repository, you'll have to check out a local copy of it, using command line.
   - Open a console.
   - Create a new folder, called "GINNet", into your workspace and then go into it.
   - Execute command *svn checkout svn+ssh://developername@scm.gforge.inria.fr/svn/ginnet* (Where *developername* is replaced by your GForge login)
   - A new folder has been added to project folder. It is called "ginnet" and will be your SubVersion local copy (the working folder where to perform *svn* commands)
2. Second step: Java project creation
   Now you need to create a new Java project in Eclipse IDE.
   - Launch Eclipse.
   - Menu: "File" → "New" → "Project..."
   - "New Project" window appears.
     Choose "Java Project" in wizard list and then click "Next >"
   - "New Java Project" window appears, as shown in figure 6.1.
     Enter project name: "GINNet" (exactly the same name than folder previously created) and then click on "Finish >"

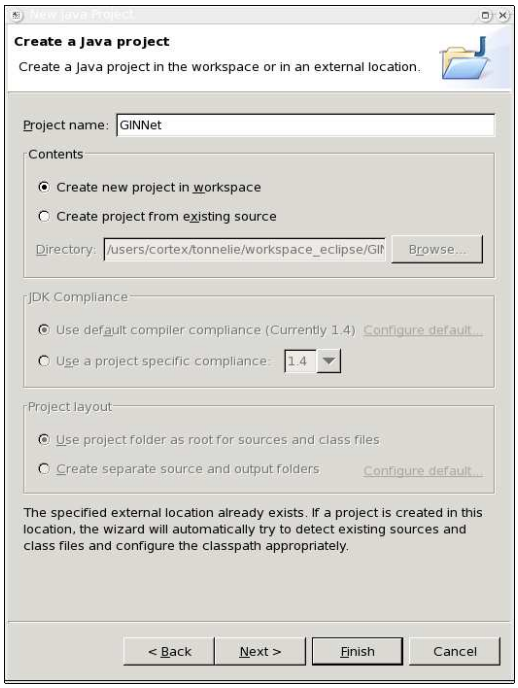


Figure 6.1: A screenshot of GINNet project creation window

Now project should compile without error and you can start working...

Refactoring

To refactor means to improve a computer program by reorganising its internal structure without altering its external behaviour.
Eclipse provides powerfull refactoring tools.

6.3.2 Ant build

An Ant script, called *build.xml* is provided with GINNet. To find it see section 2.7.1.

Run *build.xml*

To run *build.xml* :

- Select it in the package explorer.
- Left click and select "Run As" menu and then "Ant Build...".
- Select in the list targets that you want to run, as shown in figure 6.2.
- Click on "Run" button. Execution informations will appear in the Eclipse console.

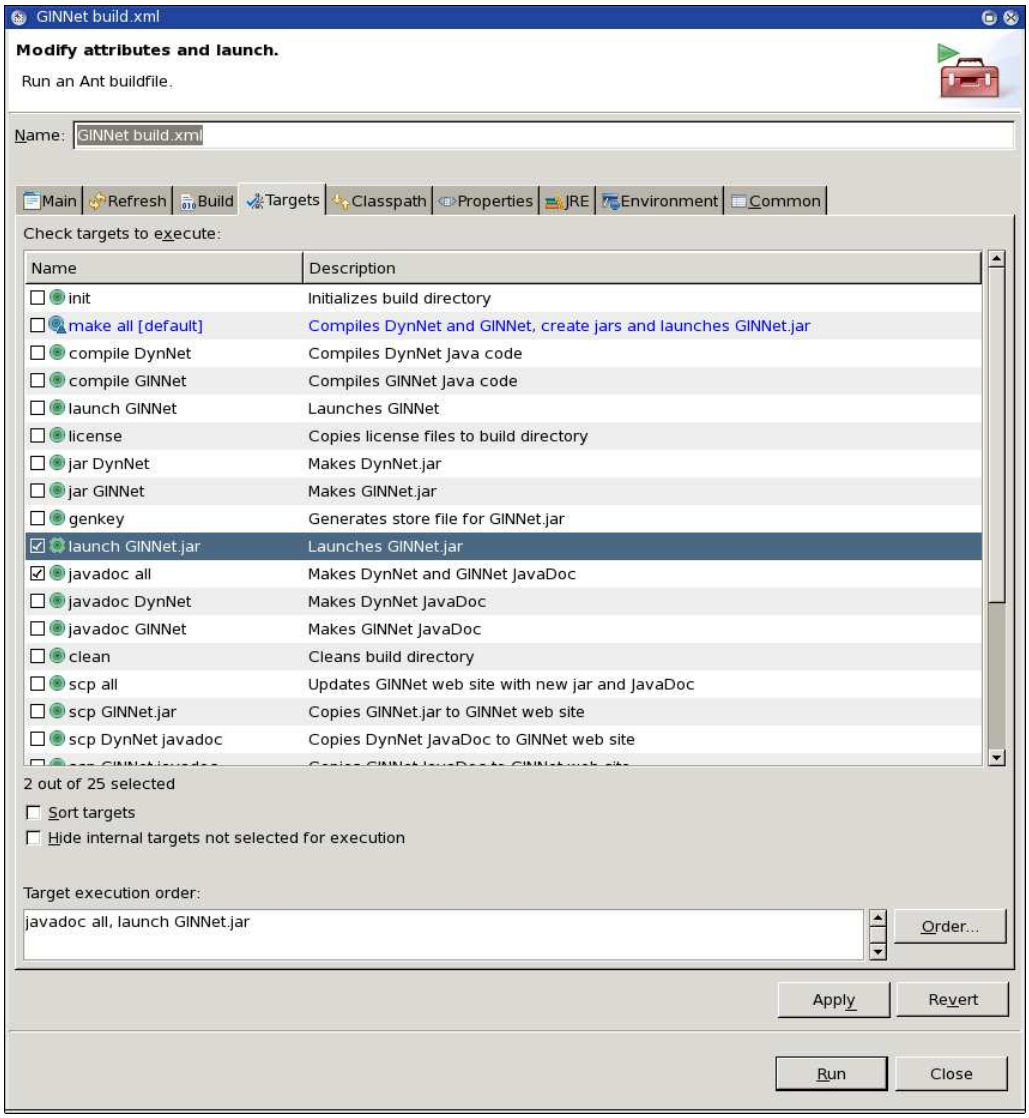


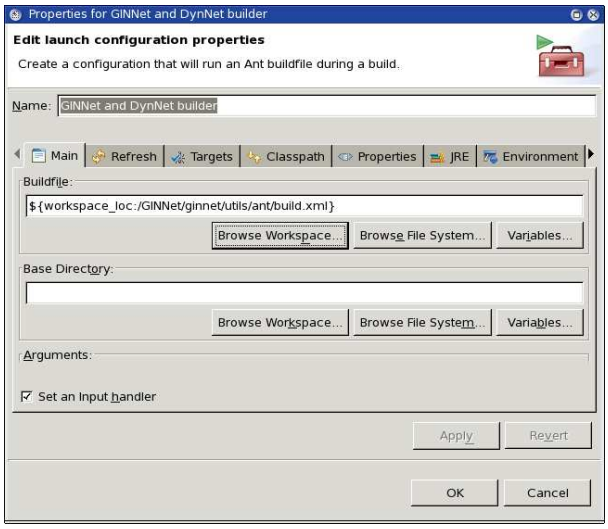
Figure 6.2: A screenshot of Ant file window where two targets are selected

As we have seen in section 6.2.2, GINNet and DynNet Ant build file needs JUnit and JSch libraries to run all targets. You can configure Eclipse so that it uses those two libraries for Ant build. To do this, go to "Window" menu, "Preferences...", "Ant", "Runtime", "Classpath", "Ant Home Entries" and click on "Add jars" button, select *lib/junit.jar* and *utils/ant/jsch-0.1.28.jar* and validate.

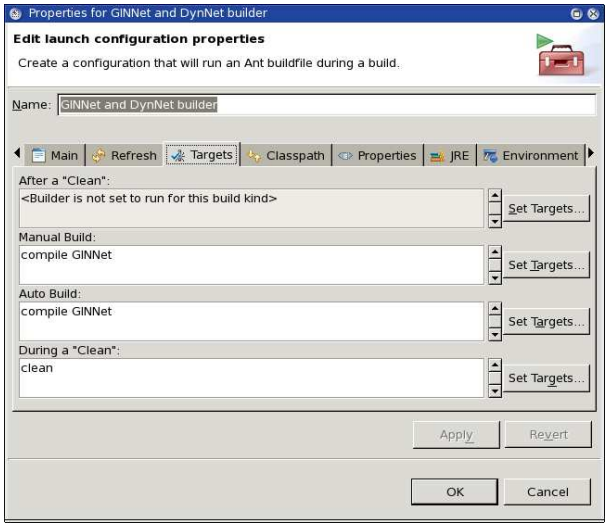
Configure project builder

You can also configure your project so that *build.xml* file will be used as the project builder, when you use build and clean feature of the IDE. To do this:

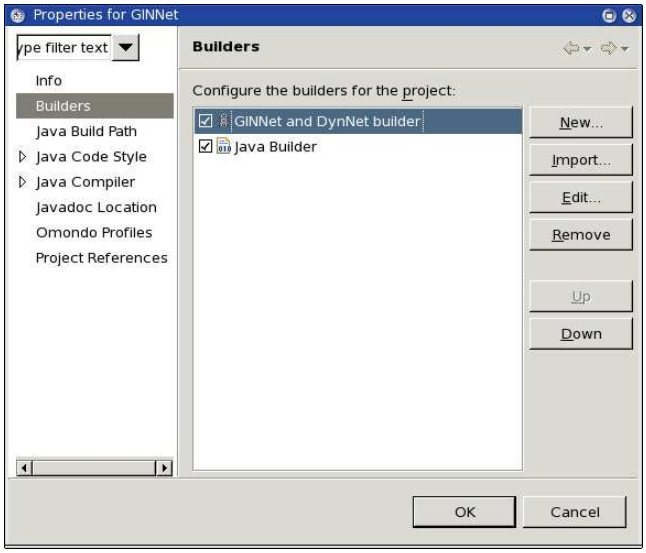
- Select your project in the package explorer.
- Left click and select "Properties" menu. Project property window appears.
- Select "Builders". By default, project only contains "Java builder".
- Click on "New..." button. Then select "Ant Build".
- Enter a name for this builder.
- In "Main" tab, set the *build.xml* file as the Buildfile.



- In "Targets" tab, select corresponding tasks for all actions, by clicking on "Set Targets..." buttons:
  - **After a "Clean":** unselect all targets.
  - **Manual Build:** select "compile GINNet" (or "compile DynNet") target.
  - **Auto Build:** select "compile GINNet" (or "compile DynNet") target.
  - **During a "Clean"** select "clean" target.



- Click on "OK" button.
- Finally move up this new builder and click on "OK" button.



6.3.3 Profiler

Eclipse Profiler Plugin is already installed for Eclipse. It works with version 3.0 of Eclipse (available in */users/cortex/tools/eclipse3.0/eclipse* directory).

- Click on main class to select it.
- Run → "Run as" → "Profiler"
- If profiler view doesn't open:
  - "Window" → "Open Perspective" → "Other..."
  - Select "Profiler"
- Don't forget to start profiling.

This profiler is incorporated into IDE and easy to use. You can quickly get informations on packages, classes and methods calls or ressources taken. Use it to optimize your code! :)
More informations on Eclipse profiler plugin installation and use at
`http://eclipsecolorer.sourceforge.net/index_profiler.html`.

Linux installation To install eclipsecolorer plugin into Eclipse:

- Download *profiler_linux.tgz* on `http://sourceforge.net/project/showfiles.php?group_id=48823&package_id=71547`,
- Untar it (*tar xvzf profiler_linux.tgz*)
- Compile and add library:
  - Edit *m* file and change JDK path to your JDK path
  - Execute it
  - Copy *libProfilerDLL.so* generated file into *lib/i386/* directory of your JRE
  - Copy it also into eclipse plugin directory
- Download *ru.nlmk.eclipse.plugins.profiler* zip file (at the same address)
- Unzip it
- Copy it into Eclipse plugin directory
- Relaunch Eclipse and enjoy!

6.4 UML tools

6.4.1 Poseidon

UML modeling software used for GINNet project is *Poseidon for UML* (`http://gentleware.com/`), an extension of ArgoUML, complete but slow, developped in Java.
A free version, called *Community Edition* is available for non commercial use. Version 4.1-0 is installed in CORTEX tools disk space. If needed, update evaluation key.
Launch it with this command (from LORIA):
/users/cortex/tools/Poseidon_For_UML_CE_4.1/Poseidon_For_UML

6.4.2 EclipseUML

EclipseUML (`http://www.omondo.com/`) is an UML plugin for Eclipse, developped by Omondo.
Free version is installed in CORTEX Eclipse version, but it is not considered has suitable for GINNet project.

Chapter 7

Frequently Asked Questions

If you have questions or suggestions about GINNet or this document, ask Marie.Tonnelier@loria.fr

- **Is neural network library independent from view?**
  DynNet regroups all neural network models. As shown in figure 4.1, they are totally independant from GUI. So you can use only DynNet and, for example, display results on console, without lauching GINNet interface. Packages *gui* depends on *dynnet*, but *dynnet* doesn't depend on *gui*.

- **How can I see weight values in GUI?**
  You can see Perceptron weight values in Network view.
  When links between layers are simple (only one red arrow), left-click on a neuron to display only connections relative to this neuron. Move the mouse upon a connection to see an help bubble (tool tip) with available informations (weight, delta and saliency), as shown in figure 7.1.

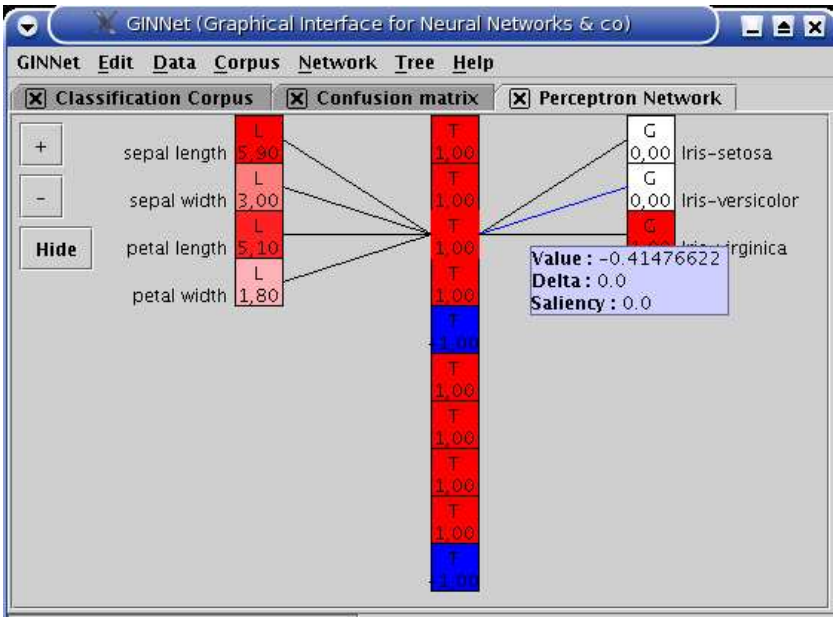


Figure 7.1: A screenshot of a connection view

Then, if you right-click, fully connected view is displayed and connections relative to previously selected neuron are still displayed with tool tip.