Manual for Functor Bayes Net System

Yuke Zhu

July 19, 2012

Contents

1 Functor Bayes Net Package

1.1 Overview

Functor Bayes Nets[1] is a software package providing a series of learning algorithms for statistical relational learning and probabilistic inference. This package is written in Java and based on the open-source *The Tetrad Project*[2] and used *Weka*[3] for decision learning. Tetrad and Weka are open-source packages, of which the source codes are imported into our package.

Package	jbn
Operating System	Unix/Linux
Environments	Java version 1.6 or greater
	MySQL installation

[1]Learning BNs For Relational Data http://www2.cs.sfu.ca/∼oschulte/jbn/
[2]The Tetrad Project http://www.phil.cmu.edu/projects/tetrad/
[3]Weka 3: Data Mining Software in Java http://www.cs.waikato.ac.nz/ml/weka/

1.2 Functor Bayes Net

The source code for our Functor Bayes Net system can be found in the *jbn* folder. There are several parts in this package, structure learning, parameter learning of Functor Bayes Net, Markov Logic Network exporting and decision tree learning.
To make the package work, first we need to configure some global settings. The global parameters of the system is defined in this file:

ca.sfu.jbn.common.global.java

You can change the default name of database, the URL, username and password of the database server and some other useful settings. Although it may be impossible to go through each package one by one in the source code, there are some important packages (classes) you should pay attention to. These packages are mentioned below:

ca.sfu.jbn.ParameterLearning_main.java
Main entry for structure and parameter learning of Functor Bayes Net; when you want to use the structure and parameter learning code, export a jar file that takes this class as main entry (argument formats are mentioned as comments in the source file)

ca.sfu.jbn.MLN_ParameterLearning_main.java
Main entry for Markov Logic Network exporting from learned Bayes Net, further used for MLN evaluation. Typically, we use this class together with the Markov Logic Network Evaluation Package. The *evaluate/jbn.jar* file is an exported runnable Jar file of the whole project launched from this class.

ca.sfu.jbn.MLN_DecisionTreeLearner_main.java
Main entry for Markov Logic Network exporting with *Weka* decision tree learner. Similar with *MLN_ParameterLearning_main.java*, it is also used for MLN evaluation. Typically, we use this class together with the Markov Logic Network Evaluation Package. The *evaluate/jbndt.jar* file is an exported runnable Jar file of the whole project launched from this class.

ca.sfu.jbn.structureLearning
Package which contains all the codes related to structure learning of Functor Bayes Net.

ca.sfu.jbn.parameterLearning
Package which contains *not all* the codes related to parameter learning of Functor Bayes Net.

1. *ParamTet.java* provides a main entry for parameter learning (As you see, it is kind of messy of these main entries for some historical reasons, do not bother too much).

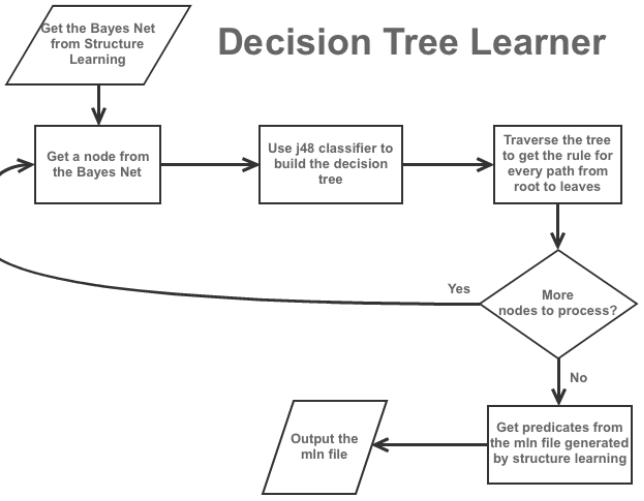

Decision Tree Learner
Get the Bayes Net from Structure Learning
Get a node from the Bayes Net
Use j48 classifier to build the decision tree
Traverse the tree to get the rule for every path from root to leaves
More nodes to process?
Yes
No
Get predicates from the mln file generated by structure learning
Output the mln file

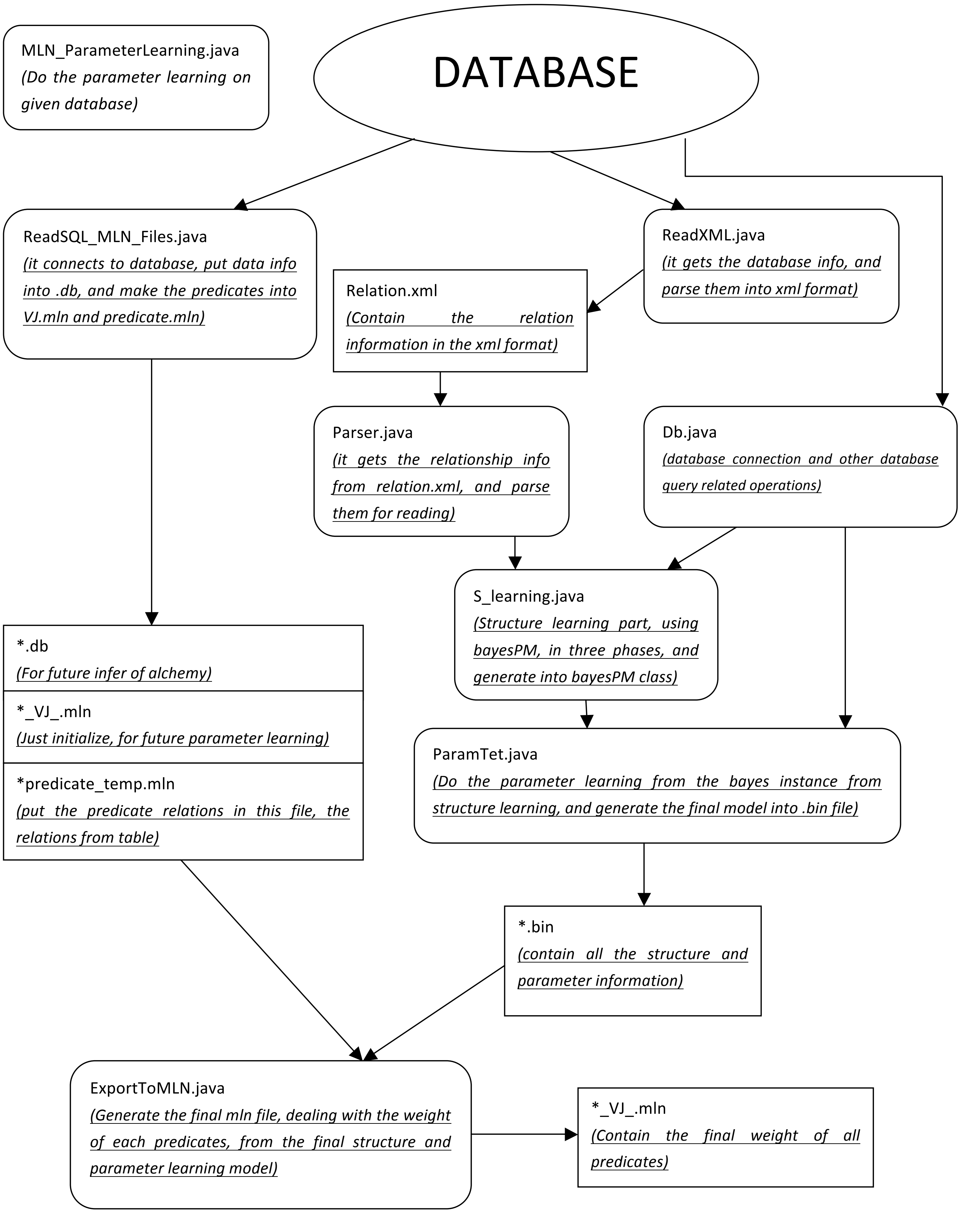
Figure 1: Flow chart to illustrate the process of decision tree learner

2. *Decisiontree.java* is the most important code for decision tree learner, in which we use *Weka* J48 decision learner to learn the tree structure and use the frequencies from the database to calculate the weights for each clause in MLN. This class is directly called from main entry of the learner *ca.sfu.jbn.MLN_DecisionTreeLearner_main.java*.

You may want to take a look at the package *ca.sfu.jbn.frequency* to know better about the process of parameter learning.

ca.sfu.jbn.frequency
Very important package for parameter learning. *BayesStat.java* is the key code to obtain frequency and probability statistics directly from the database by doing SQL queries. One thing to notice: there are some constraints for this code. It did work well for our research now, but you may have to modify it to satisfy new requirements (it may become buggy when there are multiple negative relations in your query).

edu.cmu.tetradapp.Tetrad.java
Tetrad has a GUI application for you to play with. We have done a lot of work to get our JBN system integrated with this user interface. Try this if you are interested. An exported *Tetrad* Jar file can be found in *tetrad/SFUTetrad.jar*.


MLN_ParameterLearning.java
(Do the parameter learning on given database)
DATABASE
ReadSQL_MLN_Files.java
(it connects to database, put data info into .db, and make the predicates into VJ.mln and predicate.mln)
ReadXML.java
(it gets the database info, and parse them into xml format)
Relation.xml
(Contain the relation information in the xml format)
Parser.java
(it gets the relationship info from relation.xml, and parse them for reading)
Db.java
(database connection and other database query related operations)
S_learning.java
(Structure learning part, using bayesPM, in three phases, and generate into bayesPM class)
*.db
(For future infer of alchemy)
*_VJ_.mln
(Just initialize, for future parameter learning)
*predicate_temp.mln
(put the predicate relations in this file, the relations from table)
ParamTet.java
(Do the parameter learning from the bayes instance from structure learning, and generate the final model into .bin file)
*.bin
(contain all the structure and parameter information)
ExportToMLN.java
(Generate the final mln file, dealing with the weight of each predicates, from the final structure and parameter learning model)
*_VJ_.mln
(Contain the final weight of all predicates)

2 Markov Logic Network Evaluation Package

2.1 Overview

This package is to help evaluate the structure and parameters of a learned Markov Logic Network file (.mln in the Alchemy[4] format). The MLN evaluation package outputs statistics, such as the average accuracy, conditional log-likelihood, over all predicates in the dataset.

Package	evaluate
Operating System	Unix/Linux
Environments	Java version 1.6 or greater
	Korn shell (ksh)
	Complied binaries of Alchemy (included in the package)

2.2 Inference

There are several parts of this package, of which each part is relatively independent with others. The MLN inference code is written in Java, you can find the code in *infer* folder. Here are some important packages (classes) in this code:

MLN.MLN.inference
Main entry for the MLN inference code, used for MLN evaluation. Typically, we use this class together with *evaluate* package. The *evaluate/evaluation/infer.jar* file is an exported runnable Jar file of the whole project launched from this class.

Usually, we do not use the *infer* project alone, it is just one part of the whole evaluation system.

Now, what is the most important to know about this package is how to get the codes work. First, we need to know the functionality of these files in the *evaluate* folder:

1. **test.sh**
   Korn shell script for evaluating the performance of different methods by training and testing on a single database. This is one of the main entry of the whole system, of which the argument format will be introduced later.

2. **testcross.sh**
   Korn shell script for evaluating the performance of different methods by training and testing on 5-fold cross-evaluation. This is the other one of the main entry of the whole system, of which the argument format will be introduced later.

3. **learnwts**
   Compiled code for *Alchemy* MLN weight learner. We can get the code on

[4]Alchemy - Open Source AI http://alchemy.cs.washington.edu/

the website, but if you do not want to waste several weeks on changing and compiling it, just use this one.

4. **jbn.jar**
   As we have seen, this is an exported Jar file of the Functor Bayes Net package launched from *ca.sfu.jbn.MLN_ParameterLearning_main.java*.

5. **jbndt.jar**
   As we have seen, this is an exported Jar file of the Functor Bayes Net package launched from *ca.sfu.jbn.MLN_DecisionTreeLearner_main.java*.

6. **evaluation/infer.jar**
   As we have seen, this is an exported Jar file of the Inference package launched from *MLN.MLN.inference*.

7. **evaluation/lib**
   This is a folder that contains some ancient codes used for analyzing the evaluation results and generating measurement statistics such as accuracy and log-likelihood. Some of the codes come from the *ca.sfu.jbn.Analyzer* of the *Inference* package. However, I have never read through them, why not just put there and use them?

8. **config.xml**
   Global configuration setting file that contains the database server, username and password.

Thus, we can use *test.sh* and *testcross.sh* to run the evaluation system. The argument format of these two scripts are the same; therefore, I will just explain the cross-validation script *testcross.sh*.

2.3 5-fold Cross-validation

Before we begin to use the script, make sure you have set up the cross-validation databases. Since 5-fold cross-validation is used, we create 5 training databases and corresponding 5 testing databases, which are a subset of the whole databases. The scripts in the *cross* folder are used to generate these databases, which will be explained later.

The names of these training and testing databases are derived from the whole database. For example, for a database named *MovieLens*, the generated training databases are named with suffix *_Training#* and *_Test#*:

MovieLens_Training1, *MovieLens_Training2*, . . . , *MovieLens_Training5*,

and the testing databases are named in a similar way:

MovieLens_Test1, *MovieLens_Test2*, . . . , *MovieLens_Test5*.

After we set up these databases, we can finally start our experiments. We use the following command to run a cross-validation experiment:

$ *testcross.sh dbName method geo-parameter*

Here is a detailed description of these three arguments:

1. **dbName**
   The name of the whole database you want to do cross-validation on. For instance, the cross-validation databases are *MovieLens_Training#* and *MovieLens_Test#*, you should use the name *MovieLens*.

2. **method**
   You have many options on *method* parameter. This parameter determines how to generate clauses and how to get the weights of clauses in MLN file. Here is a list:

   (a) **mbn**
   To export MLN structure file from Bayes Net and use Alchemy weight learner *mbn* to learn the weights of clauses in MLN file

   (b) **dtmbn**
   To export MLN structure file using decision tree learner from Bayes Net and use Alchemy weight learner *learnwts* to learn the weights of clauses in MLN file

   (c) **log**
   Deprecated To export weighted MLN file from Bayes Net, here we obtain the weight by taking log of the conditional probability

   (d) **dtlog**
   Deprecated To export weighted MLN file using decision tree learner from Bayes Net, here we obtain the weight by taking log of the conditional probability

   (e) **lsn**
   To export weighted MLN file from Bayes Net. Similar with *log*, here we obtain the weight by taking log of the conditional probability and then subtracting the log of unit prior

   (f) **dtlsn**
   To export weighted MLN file using decision tree learner from Bayes Net. Similar with *log*, here we obtain the weight by taking log of the conditional probability and then subtracting the log of unit prior

3. **geo-parameter**
   This parameter determines whether to use *geometric mean* when we do MLN inference. There are two options:

   (a) **geo**
   To using *geometric mean* when we do inference; the performance test usually favors this option

   (b) **nom**
   To disable *geometric mean* option

The content of the output contains statistics for each run in the 5-fold cross-validation, including the accuracy, log-likelihood for each attribute and overall accuracy and log-likelihood in average with regard to attributes.

Here is an example to illustrate how to run a 5-fold cross-validation on *MovieLens* database (using *mbn* method and geometric mean):

$$\$ \quad testcross.sh \quad MovieLens \quad mbn \quad geo$$

3 Datasets

3.1 Databases

Our datasets are some real-world and synthesized databases. There are five real-world databases[5] that we most commonly use in our experiments. You can find more details about these datasets from the references in our published papers and on-line sources. *Schema* is the name of database in our database server.

1. **MovieLens Database**
   Schema: *MovieLens*
   This is a standard dataset from the UC Irvine machine learning repository.

2. **Mutagenesis Database**
   Schema: *Muta*
   This dataset is widely used in ILP research. It contains information on Atoms, Molecules, and Bonds between them.

3. **Hepatitis**
   Schema: *hepelwin*
   This data is a modified version of the PKDD02 Discovery Challenge database. The database contains information on the laboratory examinations of hepatitis B and C infected patients.

4. **Mondial**
   Schema: *mondelwin*
   This dataset contains data from multiple geographical web data sources. We used a subset of the tables and features for fast inference.

5. **Financial**
   Schema: *fin*
   A dataset from the PKDD 1999 cup.

[5] Some of these databases are slightly modified from the original ones to meet specific requirements of our system

6. **UW**
   Schema: *uw*
   A dataset from University of Washington, containing information of faculty, students and courses.

The backup of these databases are stored in the *databases* folder. For convenience, each table is stored in one *.sql* file. Please always make sure you have the backups before you modify these databases.

4 Useful Scripts

4.1 Generator

In some of our experiments, we have to generate sub-databases from the original databases; or, make synthesized databases for some special purpose. The most frequently-used generator is to generate the 5-fold cross-validation sub-databases; other generators include the scripts to generate synthesized testbed used in our *parameter learning* evaluation.

Script Name	cross
Operating System	Unix/Linux
Environments	Java version 1.6 or greater Korn shell (ksh)
Description	Generate the 5-fold cross-validation databases. The name of the cross-validation databases are named by appending a suffix to the name of the original database. For example, the original database is *MovieLens*, it will create ten databases (five training databases and five test databases) with names: *MovieLens_Training[1-5]* and *MovieLens_Test[1-5]*
Example	See the file *example.txt* in the same folder to get the idea

Script Name	TestbedGenerator.py
Environments	Python version 2.4.3 or greater
Description	*deprecated* This script is used to generate a synthesized university database containing information about students and courses. We can control the number of students and courses in the databases and the ratio of enrollment.
Example	$ python TestbedGenerator.py <Students> <Courses>

4.2 Statistics

We have some scripts to help us handle with the outputs of our systems. You can find them in the *scripts* folder. These scripts are very simple and most of them are written in *Python*. Be free to use them to simplify your work.

Script Name	clause-count.py
Environments	Python version 2.4.3 or greater
Description	Count the number of clauses in a MLN file
Output	The number of clause in this MLN file
Example	$ python clause-count.py < MovieLens.mln

Script Name	clause-display.py
Environments	Python version 2.4.3 or greater
Description	Print out all the clauses in a MLN file
Example	$ python clause-display.py < MovieLens.mln

Script Name	cross-stat.py
Environments	Python version 2.4.3 or greater
Description	*Very useful.* Print out statistics of the cross-validation result, including accuracy, log-likelihood for each attributes, also with average accuracy, average log-likelihood and standard deviation
Example	First, we need to save all the output generated in one cross-validation experiment into a temporary file, say *cross.output*. Then run with this command: $ python cross-stat.py < cross.output