A GUI for MLN

Estêvão F. Aguiar[1], Marcelo Ladeira[1], Rommel N. Carvalho[1], and Shou Matsumoto[1]

Department of Computer Science
University of Brasília
Campus Universitario Darcy Ribeiro
Brasília, Distrito Federal, Brazil
estevaofaguiar@gmail.com, mladeira@unb.br,
{rommel.carvalho,cardialfly}@gmail.com
http://www.cic.unb.br

Abstract. This paper focuses on the incorporation of the Markov Logic Network (MLN) formalism as a plug-in for UnBBayes, a Java framework for probabilistic reasoning based on graphical models. MLN is a formalism for probabilistic reasoning which combines the capacity of dealing with uncertainty tolerating imperfections and contradictory knowledge based a Markov Network (MN) with the expressiveness of First Order Logic. A MLN provides a compact language for specifying very large MNs and the ability to incorporate, in modular form, large domain of knowledge (expressed in First Order Logic sentences) inside itself. A Graphical User Interface for the software Tuffy was implemented into UnBBayes to facilitate the creation, and inference of MLN models. Tuffy is a Java open source MLN engine.

Keywords: Markov Logic Network, MLN, Tuffy, UnBBayes, Markov Network, probabilistic reasoning, probabilistic graphical models

1 Introduction

In the past decade, several languages have been proposed to deal with complex knowledge representation problems that also need to deal with uncertainty. A frequent approach is to combine both logic and probabilistic formalisms resulting in a powerful model for knowledge representation and treatment of uncertainty. Some examples of these approaches were build and have been improved every day as Markov Logic Networks (MLN) [1], Multi-Entity Bayesian Networks (MEBN) [3], Probabilistic Relational Models (PRM) [2], Relational Markov Networks (RMN) [10], and Structural Logistic Regression (SLR) [7].

Markov Logic Network (MLN) is a principled formalism which combines First-Order Logic (FOL) with Markov network (MN). An MLN, basically, is a first-order knowledge base where a weight is assigned to each formula. The weight of a formula indicates how strong the formula is as a constraint. Together with a finite set of constants, an MLN can be grounded as a Markov network. This way, a MLN can be seen as a template for building Markov networks [1].

There are a few implementations for MLN like Alchemy [1] in C++, Tuffy [6] in Java, ProbCog [11] in both Python and Java, and Markov TheBeast [9] in Java. In some of them there is no graphical user interface (GUI). The one that does, the interface is quite simple providing no real ease-of-use. In general, as these software are not very friendly, they become hard to use for users without previous experience with their programming tasks and command lines.

This paper presents an implementation of a Java tool that consists of a GUI to facilitate the task of making inferences, creating, and editing MLN models. This tool was developed at the University of Brasilia (UnB) and uses the software Tuffy as a library. Its current features include GUIs for modeling terms of a knowledge base into a tree structure and for searching them in order to help the user find terms easily in large models. Moreover, it is also possible to edit and to persist these structures as a standard MLN file (compatible with both Tuffy and Alchemy). Besides that, every parameter that can be set on Tuffy can be easily set in the GUI. It even supports the addition in the GUI of new parameters that might be present in future versions of Tuffy using only a configuration file. This tool was implemented as a plug-in for UnBBayes, a Java open source software developed at UnB that is capable of modeling, making inferences and learning probabilistic networks [4].

This paper is structured as follows. Section 2 presents the MLN. Section 3 overviews some implementations of MLN and presents the major reasons for choosing Tuffy as the application programming interface (API) behind this plug-in. Section 4 introduces the GUI developed as a plug-in for UnBBayes.

2 Markov Logic Networks

A knowledge base of First-Order Logic (FOL) can be viewed as a set of constraints on possible worlds. Each formula has an associated weight that reflects how strong this formula is as a constraint [1]. An MLN is a set with formulas of FOL assigned to a real-valued weight for each formula. Together with a finite set of constants, it defines a Markov Network (MN). Each state of the MN generated represents a possible world of the generic MLN representation. A possible world determines the truth value of each ground (*i.e.* instantiated) predicate. Thus, it is said that an MLN is like a template for constructing MNs. Given different set of constants, it will produce different MNs with different values and sizes. However they have the same parameters and regularities in structure. Different instantiated formulas still have the same weights. So, in MLN it is possible to use inference methods generally used for MNs, since the used network is a grounded one. However, due to the fact that most of time the grounded network is large and complex, to use this method could be infeasible. Therefore, approximate and lifted inference algorithms have been proposed [1].

Maximum a Posteriori (MAP) inference (*i.e.* finding the most likely state of the world consistent with some evidence) and marginal inference (*i.e.* computing arbitrary conditional probabilities) are common approaches to making inferences in MLN. Learning algorithms are used to build, from historic data, models that represent a problem to be treated. For this formalism, learning methods are used to construct or refine a MLN. Two types of learning are specified: weight learning (*i.e.* which tries to find the weights of the specified formulas that maximize the likelihood or conditional likelihood of a relational database) and the harder technique of structure learning (*i.e.* which tries to learn the formulas themselves).

More details on MLN can be found in [1] and will not be covered in this paper.

3 The choice of an implementation

With the intent of building a GUI for MLN, the first step is to implement or find an existing implementation of the formalism. So, pros and cons of some implementations have to be analyzed. If no implementation had compatibility with UnBBayes, it would be necessary to create a new one. Fortunately it was not the case. The pros and cons of the more common implementations are presented below. As our goal was to build a plug-in for UnBBayes, the programming language had a larger weight than the features available on the tool. UnBBayes [4] is an open source application developed in Java that provides a framework for building and reasoning with probabilistic graphical models. Since version 1.5.1, it works with Java Plugin Framework (JPF). JPF allows the construction of scalable projects, loading plug-ins at runtime. The MLN GUI has been built as a plug-in for UnBBayes.The software analyzed were Alchemy [1], ProbCog [11], TheBeast [9] and Tuffy [6].

3.1 Alchemy

Alchemy is the reference for other implementations of MLN and is the most complete of them. It covers MAP Inference, marginal inference, weight learning, structure learning and other features from each of the mentioned topics. Alchemy is an open source software developed in C/C++. It does not have a GUI and it works only in Linux or Linux shell emulator. Alchemy was the first option to extend, but its programing language is not easily integrated with Java.

3.2 TheBeast

TheBeast [8] is an open source and is a Statistical Relational Learning software based on MLN. Although it is developed in Java, it does not have much documentation and it does not work similarly to Alchemy. This fact impacts on that it would be harder to work with it. TheBeast has no GUI implemented either.

3.3 ProbCog

ProbCog is an open source software for Statistical Relational Learning that supports learning and inference for relational domains. Merged to ProbCog, is PyMLN, a toolbox and a software library for learning and reasoning in MLN.

It has a GUI for MLN but it, seemingly, shows the necessary files for inference and the main parameters to be more easily selected, but nothing beyond the basic. Most of the code of ProgCog is developed in Java, although its MLN tool is developed in Python.

3.4 Tuffy

Tuffy is an open source Markov Logic Network engine. It is developed in Java and makes use of a PostgreSQL database. Tuffy is in version 0.3 and is capable of MRF partitioning, MAP inference, marginal inference and weight learning. Since Tuffy has many similar features to Alchemy, as weel as the same structure for input files, it has no GUI, and it is implemented in the same programming language as UnBBayes, it ended up being the most suitable MLN implementation to be used in the MLN GUI plug-in.

4 The GUI for MLN

There are several helpful easy to use GUI tools for Bayesian networks. However, this is not true to MLN yet. For most of them, the only way to make it work is to set command line parameters and then enter commands through a console. Sometimes you must memorize a bunch of commands if you want to realize a task fast, while you could just press buttons and choose options with some clicks in a more easy to use GUI interface. Creating a GUI to simplify this process of designing and using MLNs was the main motivation of this research. The following paragraphs describe the main features of a proposed GUI for MLN.

This project of a GUI for MLN into UnBBayes was built as a JPF plug-in. The plug-in structure provides a way to run a new project inside the running environment of UnBBayes. The bind between the new plug-in and the core of UnBBayes happens in a way that no changes are needed in the core structure.

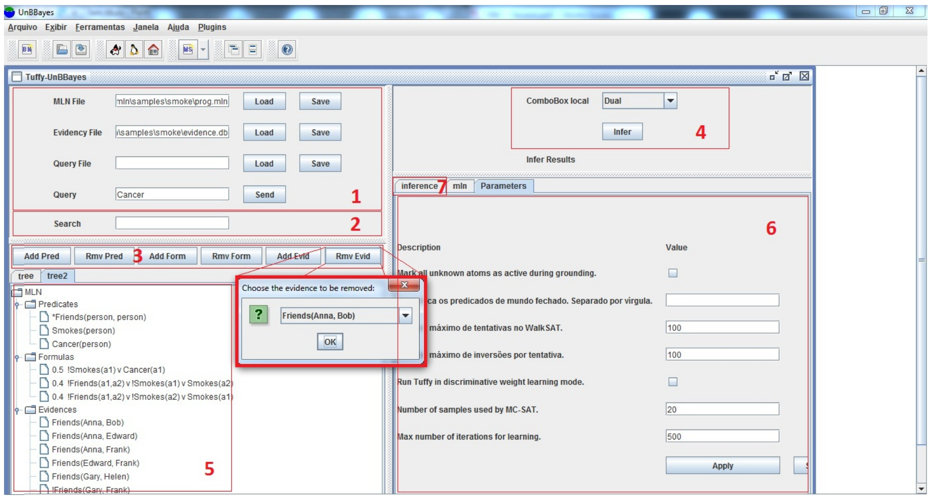
Basically, building new plug-in implementations for UnBBayes is really simple, since a stub implementation is available in [5].

Figure 1 presents the GUI divided in numbered parts. Each part is described bellow.

The Tuffy input files are: a MLN file (.mln), an evidence file (.db) and a query file (.db). The last one can be replaced passing its content through command line. Figure 1 Part 1 shows the possibility to load this three files and the possibility to send the query predicates through a text field.

When the MLN file and the evidence file are loaded, their terms (*i.e.* predicates, weighted formulas and evidences) are separated and organized in a tree structure as shown in Figure 1 Part 5. This tree structure gives a great gain of visualization and differences between structures into the MLN.

Figure 1 Part 2 presents a very useful search tool. It searches dynamically predicates, formulas, and evidence that match the inputted string. This feature is useful when working with very large MLNs.



MLN File: mln\samples\smoke\prog.mln
Evidency File: \samples\smoke\evidence.db
Query File:
Query: Cancer
Search:

- MLN
  - Predicates
    - *Friends(person, person)
    - Smokes(person)
    - Cancer(person)
  - Formulas
    - 0.5 !Smokes(a1) v Cancer(a1)
    - 0.4 !Friends(a1,a2) v !Smokes(a1) v Smokes(a2)
    - 0.4 !Friends(a1,a2) v !Smokes(a2) v Smokes(a1)
  - Evidences
    - Friends(Anna, Bob)
    - Friends(Anna, Edward)
    - Friends(Anna, Frank)
    - Friends(Edward, Frank)
    - Friends(Gary, Helen)
    - !Friends(Gary, Frank)

Choose the evidence to be removed: Friends(Anna, Bob)

ComboBox local: Dual

Description	Value
Mark all unknown atoms as active during grounding.	☐
ca os predicados de mundo fechado. Separado por vírgula.	
máximo de tentativas no WalkSAT.	100
máximo de inversões por tentativa.	100
Run Tuffy in discriminative weight learning mode.	☐
Number of samples used by MC-SAT.	20
Max number of iterations for learning.	500

Fig. 1. GUI for MLN implemented as plugin into UnBBayes

The GUI also presents a way to add and remove predicates, formulas and evidence. This feature is shown in Figure 1 Part 3. Lots of terms can be directly inputted into the correct classification. The deletion is made from a drop down list which brings to the user all the existing terms. Every change made through this feature is persisted in the original file. This feature makes it easier for the user to include or remove terms in a MLN model.

Figure 1 Part 4 allows the user to choose what inference method to use and the button to trigger the inference process, which will be executed by Tuffy in the background. Tuffy is embedded into UnBBayes and used as a library through its API.

Figure 1 Part 7 is displayed when the "inference" tab is chosen. It presents the output in a text area, the same way that it is presented in the output file in Tuffy.

Figure 1 Part 6 presents the parameters of Tuffy in an easy way to set and save. The parameters of Tuffy were parameterized by type that they represent (*e.g.* integer, float, boolean and string). This allows the parameters to be loaded to the interface from a configuration file and new parameters added in new versions of Tuffy can be easily incorporated to UnBBayes without the need to change any programming code. The dynamic values of the parameters are defined in another configuration file.

5 Conclusion

This paper presents a GUI for Tuffy, a Java Markov Logic Network inference engine. As shown, this GUI facilitates the task of creating MLNs models and reasoning with them. This GUI was implemented as a JFP plug-in for the

UnBBayes software. UnBBayes and this plug-in[1] is available from `http://unbbayes.sourceforge.net/` under GPL license.

References

1. P. Domingos and D. Lowd. *Markov Logic: An Interface Layer for AI.* Morgan and Claypool, 2009.
2. Nir Friedman, Lise Getoor, Daphne Koller, and Avi Pfeffer. Learning probabilistic relational models. In *International Joint Conference on Artificial Intelligence*, volume 16, pages 1300–1309. LAWRENCE ERLBAUM ASSOCIATES LTD, 1999.
3. K.B. Laskey. MEBN: A Language for First-Order Bayesian Knowledge Bases. *Artificial Intelligence, 172(2-3): 140-178*, 2008.
4. M. Vieira M. Onishi R.N. Carvalho M. Ladeira, D. da Silva and W. da Silva. Platform independent and open tool for probabilistic networks. *Proceedings of the IV Artificial Intelligence National Meeting (ENIA 2003) on the XXIII Congress of the Brazilian Computer Society (SBC 2003), Unicamp, Campinas, Brazil*, 2003.
5. Shou Matsumoto, Rommel Novaes Carvalho, Marcelo Ladeira, Paulo Cesar G. da Costa, Laecio Lima Santos, Danilo Silva, Michael Onishi, Emerson Machado, and Ke Cai. UnBBayes: a Java Framework for Probabilistic Models in AI. In *Java in Academia and Research.* iConcept Press, 2011.
6. R C. Doan A. Shavlik J. Niu, F. Tuffy: Scaling up statistical inference in markov logic networks using an RDBMS. *Proceedings of the VLDB Endowment, 4(6), 373-384*, 2011.
7. Alexandrin Popescul and Lyle H Ungar. Structural logistic regression for link analysis. *Departmental Papers (CIS)*, page 133, 2003.
8. Sebastian Riedel. Improving the accuracy and efficiency of MAP inference for Markov Logic. In *Proceedings of the 24th Annual Conference on Uncertainty in AI (UAI '08)*, pages 468–475, 2008.
9. Sebastian Riedel. Markov thebeast user manual. 2008.
10. Ben Taskar, Pieter Abbeel, and Daphne Koller. Discriminative probabilistic models for relational data. In *Proceedings of the Eighteenth conference on Uncertainty in artificial intelligence*, pages 485–492. Morgan Kaufmann Publishers Inc., 2002.
11. Jain D. Beetz M. Tenorth, M. Knowledge representation for cognitive robots. *Knstliche Intelligenz, Springer, volume 24*, 2010.

[1] This plug-in can only be downloaded from the SVN repository. Soon a distribution will be released for simple download and installation.