Probabilistic Soft Logic

Subjects: Others
Contributor: HandWiki Xu

Probabilistic Soft Logic (PSL) is a statistical relational learning (SRL) framework for modeling probabilistic and relational domains. It is applicable to a variety of machine learning problems, such as collective classification, entity resolution, link prediction, and ontology alignment. PSL combines two tools: first-order logic, with its ability to succinctly represent complex phenomena, and probabilistic graphical models, which capture the uncertainty and incompleteness inherent in real-world knowledge. More specifically, PSL uses "soft" logic as its logical component and Markov random fields as its statistical model. PSL provides sophisticated inference techniques for finding the most likely answer (i.e. the maximum a posteriori (MAP) state). The "softening" of the logical formulas makes inference a polynomial time operation rather than an NP-hard operation.

Keywords: machine learning ; ontology ; modeling

1. Description

The SRL community has introduced multiple approaches that combine graphical models and first-order logic to allow the development of complex probabilistic models with relational structures. A notable example of such approaches is Markov logic networks (MLNs). [1] Like MLNs, PSL is a modelling language (with an accompanying implementation[2]) for learning and predicting in relational domains. Unlike MLNs, PSL uses soft truth values for predicates in an interval between [0,1]. This allows for the underlying inference to be solved quickly as a convex optimization problem. This is useful in problems such as collective classification, link prediction, social network modelling, and object identification/entity resolution/record linkage.

Probabilistic Soft Logic was first released in 2009 by Lise Getoor and Matthias Broecheler. [3] This first version focused heavily on reasoning about similarities between entities. Later versions of PSL would still keep the ability to reason about similarities, but generalize the language to be more expressive.

In 2017, a Journal of Machine Learning Research article detailing PSL and the underlying graphical model was published along with the release of a new major version of PSL (2.0.0). [4] The major new features in PSL 2.0.0 was a new type of rule mainly used in specifying constraints and a command-line interface.

2. Syntax and Semantics

2.1. Terminology

- PSL Program — A collection of rules, each of which is a template for a potential in a graphical model.
- Rule — An expression relating atoms. Rules will typically take the form of either a first-order logical implication or a linear combination.
- Constant — A string or number that represents a real element in the universe over which a PSL program represents. Constants can represent attributes or entire entities.
- Variable — An identifier for which constants can be substituted.
- Term — Either a constant or a variable.
- Predicate — A relation defined by a unique name and a number of arguments it accepts.
- Atom — A predicate along with its term arguments.
- Ground Atom — An atom where all arguments are constants.

2.2. Syntax

A PSL model is composed of a series of weighted rules and constraints. PSL supports two types of rules: Logical and Arithmetic. [5]

Logical rules are composed of an implication with only a single atom or a conjunction of atoms in the body and a single atom or a disjunction of atoms in the head. Since PSL uses soft logic, hard logic operators are replaced with Łukasiewicz soft logical operators. An example of a logical rule expression is:

```
Similar(A, B) & HasLabel(A, X) -> HasLabel(B, X)
```

This rule can be interpreted to mean: *If A and B are similar and A has the label X, then there is evidence that B also has the label X.*

Arithmetic rules are relations of two linear combinations of atoms. Restricting each side to a linear combination ensures that the resulting potential is convex. The following relational operators are supported: `=`, `<=`, and `>=`.

```
Similar(A, B) = Similar(B, A)
```

This rule encodes the notion that similarity is symmetric in this model.

A commonly used feature of arithmetic rules is the summation operation. The summation operation can be used to aggregate multiple atoms. When used, the atom is replaced with the sum of all possible atoms where the non-summation variables are fixed. Summation variables are made by prefixing a variable with a `+`. Fox example:

```
HasLabel(A, +X) = 1.0
```

If the possible values for X are *label1*, *label2*, and *label3*, then the above rule is equivalent to:

```
HasLabel(A, 'label1') + HasLabel(A, 'label2') + HasLabel(A, 'label3') = 1.0
```

Both of these rules force the sum of all possible labels for an entity to sum to 1.0. This type of rule is especially useful for collective classification problems, where only one class can be selected.

2.3. Semantics

HL-MRF

A PSL program defines a family of probabilistic graphical models that are parameterized by data. More specifically, the family of graphical models it defines belongs to a special class of Markov random field known as a Hinge-Loss Markov Field (HL-MRF). An HL-MRF determines a density function over a set of continuous variables $\mathbf{y} = (y_1, \cdots, y_n)$ with joint domain $[0,1]^n$ using set of evidence $\mathbf{x} = (x_1, \cdots, x_m)$, weights $\mathbf{w} = (w_1, \cdots, w_k)$, and potential functions $\phi = (\phi_1, \cdots, \phi_k)$ of the form $\phi_\mathbf{i}(\mathbf{x}, \mathbf{y}) = \max(\ell_i(\mathbf{x}, \mathbf{y}), 0)^{d_i}$ where ℓ_i is a linear function and $d_i \in \{1, 2\}$. The conditional distribution of $\mathbf{y}$ given the observed data $\mathbf{x}$ is defined as

$$P(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{y})}\exp(\sum_{i=1}^{k} w_i \phi_i(\mathbf{x}, \mathbf{y}))$$

Where $Z(\mathbf{y}) = \int_\mathbf{y} \exp(\sum_{i=1}^{k} w_i \phi_i(\mathbf{x}, \mathbf{y}))$ is the partition function. This density is a logarithmically convex function, and thus the common inference task in PSL of finding a maximum a posteriori estimation of the joint state of $\mathbf{y}$ is a convex problem. This allows inference in PSL to be achievable in polynomial-time.

Open/Closed Predicates -- Closed World Assumption

Predicates in PSL can be labeled as open or closed.

When a predicate is labeled closed, PSL makes the closed-world assumption: any predicates that are not explicitly provided to PSL are assumed to be false. In other words, the closed world assumption presumes that a predicate that is partially true is also known to be partially true. For example, if we had the following constants in the data for representing people: $\{Alice, Bob\}$ and the following constant for movies: $\{Avatar\}$, and we provided PSL with the predicate data $\{rating(Alice, Avatar) = 0.8\}$ and $rating(\cdot)$ was labeled closed, then PSL would assume that $\{rating(Bob, Avatar) = 0\}$ even though this data was never explicitly provided to the system.

If a predicate is labeled as open, then PSL does not make the closed-world assumption. Instead, PSL will attempt to collectively infer the unobserved instances.

Grounding

Data is used to instantiate several potential functions in a process called grounding. The resulting potential functions are then used to define the HL-MRF.

Grounding predicates in PSL is the process of making all possible substitutions of the variables in each predicate with the existing constants in the data, resulting in a collection of ground atoms, $\mathbf{y} = \{y_1, \cdots, y_n\}$. Then, all possible substitutions of the ground atoms for the predicates in the rules are made to create ground rules.

Each of the ground rules are interpreted as either potentials or hard constraints in the induced HL-MRF. A logical rule is translated as a continuous relaxation of Boolean connectives using Łukasiewicz logic. A ground logical rule is transformed into its disjunctive normal form. Let I^+ be the set of indices of the variables that correspond to atoms that are not negated, and, likewise I^- the set of indices corresponding to atoms that are negated, in the disjunctive clause. Then the logical rule maps to the inequality:

$$1 - \sum_{i \in I^+} y_i - \sum_{i \in I^-} (1 - y_i) \leq 0$$

If the logical rule is weighted with a weight w and exponentiated with $d \in \{1, 2\}$, then the potential

$$\phi(\mathbf{y}) = \left(\max \left\{ 1 - \sum_{i \in I^+} y_i - \sum_{i \in I^-} (1 - y_i), 0 \right\} \right)^d$$

is added to the HL-MRF with a weight parameter of w.

An arithmetic rule is manipulated to $\ell(\mathbf{y}) \leq 0$ and the resulting potential takes the form $\phi(\mathbf{y}) = (\max\{\ell(\mathbf{y}), 0\})^d$.

3. Interfaces

PSL is available via three different language interfaces: CLI, Java, and Python. PSL's command line interface (CLI) is the recommended way to use PSL. [6] It supports all the features commonly used in a reproducible form that does not require compilation. Since PSL is written in Java, the PSL Java interface is the most expansive and users can call directly into the core of PSL. [7] The Java interface is available through the Maven central repository. [8] The PSL Python interface is available through PyPi [9] and uses pandas DataFrames to pass data between PSL and the user. [10]

PSL previously provided a Groovy interface. [11] It has been deprecated in 2.2.1 release of PSL, and is scheduled to be removed in the 2.3.0 release. [12]

4. Examples

The LINQS lab, developers of the official PSL implementation, maintain a collection of PSL examples. [13] These examples cover both synthetic and real-world datasets and include examples from academic publications using PSL. Below is a toy example from this repository that can be used to infer relations in a social network. Along with each rule is a comment describing the motivating intuition behind the statements.

```
/* People living in the same location are more likely to know one another. */ 20: Lived(P1, L) & Lived(P2, L) & (P1 != P2) -> Knows(P1, P2) ^2 /* People who have not lived in the same location are not likely to know one another. */ 5: Lived(P1, L1) & Lived(P2, L2) & (P1 != P2) & (L1 != L2) -> !Knows(P1, P2) ^2 /* Two people with similar interests are more likely to know one another. */ 10: Likes(P1, X) & Likes(P2, X) & (P1 != P2) -> Knows(P1, P2) ^2 /* People in the same circles tend to know one another (transitivity). */ 5: Knows(P1, P2) & Knows(P2, P3) & (P1 != P3) -> Knows(P1, P3) ^2 /* Knowing one another is symmetric. */ Knows(P1, P2) = Knows(P2, P1) . /* By default, assume that two arbitrary people do not know one another (negative prior). */ 5: !Knows(P1, P2) ^2
```

References

1. Getoor, Lise; Taskar, Ben (2007). Introduction to Statistical Relational Learning. MIT Press. ISBN 978-0262072885. https://linqs.github.io/linqs-website/publications/#id:getoor-book07.
2. "GitHub repository". https://github.com/linqs/psl. Retrieved 26 March 2018.

3. Broecheler, Matthias; Getoor, Lise (2009). "Probabilistic Similarity Logic". International Workshop on Statistical Relational Learning (SRL). https://linqs.github.io/linqs-website/publications/#id:broecheler-srl09.
4. Bach, Stephen; Broecheler, Matthias; Huang, Bert; Getoor, Lise (2017). "Hinge-Loss Markov Random Fields and Probabilistic Soft Logic". Journal of Machine Learning Research 18: 1–67.
5. "Rule Specification". LINQS Lab. December 6, 2019. https://psl.linqs.org/wiki/master/Rule-Specification.html.
6. Augustine, Eriq (15 July 2018). "Getting Started with PSL" (in en). https://psl.linqs.org/blog/2018/07/15/getting-started-with-psl.html. Retrieved 15 July 2020.
7. "PSL API Reference" (in en). https://psl.linqs.org/api/. Retrieved 15 July 2020.
8. "Maven Repository: org.linqs » psl-java". https://mvnrepository.com/artifact/org.linqs/psl-java. Retrieved 15 July 2020.
9. "pslpython: A python inferface to the PSL SRL/ML software.". https://pypi.org/project/pslpython/. Retrieved 15 July 2020.
10. Augustine, Eriq (6 December 2019). "PSL 2.2.1 Release" (in en). https://psl.linqs.org/blog/2019/12/06/psl-2.2.1-release.html#new-python-interface. Retrieved 15 July 2020.
11. "Maven Repository: org.linqs » psl-groovy". https://mvnrepository.com/artifact/org.linqs/psl-groovy.
12. Augustine, Eriq (6 December 2019). "PSL 2.2.1 Release" (in en). https://psl.linqs.org/blog/2019/12/06/psl-2.2.1-release.html#groovy-interface-deprecated. Retrieved 15 July 2020.
13. "linqs/psl-examples". linqs. 19 June 2020. https://github.com/linqs/psl-examples.
