Getting Started with PSL

July 15, 2018 • Eriq Augustine

Tags: tutorial • v2.1.0

Probabilistic Soft Logic (PSL) is a Statistical Relational Learning (SRL) framework. It is intended to be usable by people at all levels of Computer Science and Machine Learning.

PSL currently has two primary interfaces: a command line interface (CLI) and a Groovy interface. In this tutorial, we will cover the command line interface. However, all the examples we reference also have implemented Groovy versions on the same repository.

Setup

For the CLI, the only hard requirement is that you have Java 7 or 8 installed. However we will be using some scripts meant for UNIX-style systems (Linux and Mac), so Windows users will have some additional steps. We will also be using git to fetch some examples, but users without git can just use a browser to go to the repository and download it directly.

Getting the code

As with all the PSL examples, you can find all the code in our psl-examples repository. For this tutorial, we will be using the `simple-acquaintances` example.

```
git clone https://github.com/linqs/psl-examples.git
cd psl-examples/simple-acquaintances/cli
```

Run your first PSL program

The `simple-acquaintances` example is a toy model with the goal of inferring whether or not a pair of people already know each other. To do this, we will use information about the locations of some people, who people know already, and what people enjoy to infer who knows each other. Before we dive into the specifics of the model, lets just get it running.

All examples come with a `run.sh` script that should handle everything for us. So on the command line, invoke the run script:

```
./run.sh
```

The PSL jar will be fetched automatically. You can also select what version of PSL is fetched/used at the top of this script.

You should now see output that looks like this:

```
Running PSL Inference
0    [main] INFO  org.linqs.psl.cli.Launcher  - Loading data
82   [main] INFO  org.linqs.psl.cli.Launcher  - Data loading complete
82   [main] INFO  org.linqs.psl.cli.Launcher  - Loading model
178  [main] INFO  org.linqs.psl.cli.Launcher  - Model loading complete
178  [main] INFO  org.linqs.psl.cli.Launcher  - Starting inference with class: or
243  [main] INFO  org.linqs.psl.application.inference.MPEInference  - Grounding o
334  [main] INFO  org.linqs.psl.application.inference.MPEInference  - Beginning i
448  [main] INFO  org.linqs.psl.reasoner.admm.ADMMReasoner  - Optimization comple
448  [main] INFO  org.linqs.psl.application.inference.MPEInference  - Inference c
472  [main] INFO  org.linqs.psl.application.inference.MPEInference  - Results com
472  [main] INFO  org.linqs.psl.cli.Launcher  - Inference Complete
477  [main] INFO  org.linqs.psl.cli.Launcher  - Starting evaluation with class: o
493  [main] INFO  org.linqs.psl.cli.Launcher  - Evaluation results for KNOWS -- A
493  [main] INFO  org.linqs.psl.cli.Launcher  - Evaluation complete.
```

Where are the predictions?

By default, the PSL examples output the results into the `inferred-predicates` directory. The results for this program will looks something like:

```
$ cat inferred-predicates/KNOWS.txt | sort
'Alex'  'Arti'    0.9966434240341187
'Alex'  'Ben'     0.5923646092414856
'Alex'  'Dhanya'  0.48785600066185
'Alex'  'Elena'   0.7109028697013855
'Alex'  'Jay'     0.6563224792480469
'Alex'  'Sabina'  0.5790407657623291
< --- 40 lines omitted for brevity --- >
'Steve'  'Arti'   0.5648082494735718
'Steve'  'Ben'    0.4413864314556122
'Steve'  'Dhanya' 0.4964808523654938
'Steve'  'Elena'  0.4955149292945862
'Steve'  'Jay'    0.6143320798873901
'Steve'  'Sabina' 0.5134021043777466
```

What did it do?

Now that we've run our first program that performs link prediction to infer who knows who, let's understand the steps that we went through to infer the unknown values: defining the underlying model, providing data to the model, and running inference to classify the unknown values.

Defining a Model

A model in PSL is a set of logic-like rules.

The model is defined inside a text file with the format `.psl`. We describe this model in the file `simple-acquaintances.psl`.

Let's have a look at the rules that make up our model:

```
20: Lived(P1, L) & Lived(P2, L) & (P1 != P2) -> Knows(P1, P2) ^2
5: Lived(P1, L1) & Lived(P2, L2) & (P1 != P2) & (L1 != L2) -> !Knows(P1, P2) ^2
10: Likes(P1, L) & Likes(P2, L) & (P1 != P2) -> Knows(P1, P2) ^2
5: Knows(P1, P2) & Knows(P2, P3) & (P1 != P3) -> Knows(P1, P3) ^2
Knows(P1, P2) = Knows(P2, P1) .
5: !Knows(P1, P2) ^2
```

The model is expressing the intuition that people who have lived in the same location or like the same thing may know each other. The values at the beginning of rules indicate the weight of the rule. Intuitively, this tells us the relative importance of satisfying this rule compared to the other rules. (Weight only matters relative to other rules, not in an absolute sense.) The `^2` at the end of the rules indicates that the hinge-loss function for this rule should be squared. This makes for a smoother tradeoff between conflicting values.

For a full description of rule syntax, see the Rule Specification in the wiki.

Loading the Data

PSL rules consist of predicates joined by logical operations. The names of the predicates and the data to load into them are defined inside the file `simple-acquaintances.data`.

Let's have a look:

```
predicates:
  Knows/2: open
  Likes/2: closed
  Lived/2: closed

observations:
  Knows : ../data/knows_obs.txt
  Lived : ../data/lived_obs.txt
  Likes : ../data/likes_obs.txt

targets:
  Knows : ../data/knows_targets.txt

truth:
  Knows : ../data/knows_truth.txt
```

In the `predicates` section, we list all the predicates that will be used in rules for this model. The keyword `open` indicates that we want to infer some values of this predicate while `closed` indicates that this predicate is fully observed. I.e. all substitutions of this predicate have known values and will behave as evidence for inference.

For our simple example, we fully observe where people have lived (`Lived`) and what things they like or dislike (`Likes`). Thus, `Likes` and `Lived` are both closed predicates. We are aware of some

instances of people knowing each other, but wish to infer the other instances. Therefore, `Knows` an open predicate.

In the `observations` section, for each predicate that we have observations on, we specify the name of the tab-separated file containing the observations. For example, `knows_obs.txt` and `lived_obs.txt` specifies which people know each other and where some of these people live, respectively.

The `targets` section specifies a file that, for each open predicate, lists all substitutions of that predicate that we wish to infer. In `knows_targets.txt`, we specify the pairs of people for whom we wish to infer.

The `truth` section specifies a file that provides a set of ground truth observations for each open predicate. Here, we give the actual values for the `Knows` predicate for all the people in the network as training labels. We describe the general data loading scheme in more detail in the sections below. Truth data is only necessary if you are running weight learning or evaluation.

Writing PSL Rules

To create a PSL model, you should define a set of rules in a `.psl` file. Let's go over the basic syntax to write rules. Consider this very general rule form:

```
w: P(A,B) & Q(B,C) -> R(A,C) ^2
```

The first part of the rule, `w`, is a float value that specifies the weight of the rule. In this example, `P`, `Q` and `R` are predicates. Logical rules consist of the rule "body" and the rule "head." The body of the rule appears before the `->` which denotes logical implication. The body can have one or more predicates conjuncted together with the logical conjunction operator: `&`. The head can have one or more predicates disjuncted together with the logical disjunction operator: `|`. The predicates that appear in the body and head can be any combination of open and closed predicate types.

The Rule Specification wiki page contains the full syntax for PSL rules.

Organizing your Data

In a `.data` file, you should first define your `predicates:` as shown in the above example. Use the `open` and `closed` keywords to characterize each predicate.

A `closed` predicate is a predicate whose values are always observed. For example, the `knows` predicate from the simple example is closed because we fully observe the entire network of people that know one another. On the other hand, an `open` predicate is a predicate where some values may be observed, but some values are missing and thus, need to be inferred.

As shown above, then create your `observations:`, `targets:` and `truth:` sections that list the names of files that specify the observed values for predicates, values you want to infer for open predicates, and observed ground truth values for open predicates.

For all predicates, all possible substitutions should be specified either in the target files or in the observation files. The observations files should contain the known values for all closed predicates and

can contain some of the known values for the open predicates. The target files tell PSL which substitutions of the open predicates it needs to infer. Target files cannot be specified for closed predicates as they are fully observed.

The truth files provide training labels in order learn the weights of the rules directly from data. This is similar to learning the weights of coefficients in a logistic regression model from training data.

Running Inference

Run inference with the general command:

```
java -jar psl-cli.jar --infer --model [name of model file].psl --data [name of da
```

When we run inference, the inferred values are outputted to the screen. If you want to write the outputs to a file and use the inferred values in various ways downstream, you can use:

```
java -jar psl-cli.jar --infer --model [name of model file].psl --data [name of da
```

Values for all predicates will be output as tab-separated files in the specified output directory.

With the inferred values, some downstream tasks that you can perform are:

- if you have a gold standard set of labels, you can evaluate your model by computing standard metrics like accuracy, AUC, F1, etc.
- you may want to use the predicted outputs of PSL as inputs for another model.
- you may want to visualize the predicted values and use the outputs of PSL as inputs to a data visualization program.

More PSL Resources

For more information on PSL, you can check out:

- Key Papers
- Support Forum
- Source Code
- PSL Examples