Cplint_R Documentation

Franco Masotti

Table of Contents

1 About

1.1 About

The purpose of this library is to provide an interface between the Cplint[1] suite for SWI Prolog[2] and R[3] to handle graphs made with the ggplot2[4] package.

This means that the currently available C3js[5] graphing functions are also available for R.

1.2 Terminology

Any code between `< >` represents pseudocode.

The symbol `#` represents an unsigned integer.

Oprional parameters are encapsuled between `[ ]`.

[1] See item [**Cplint**] in Chapter 6 [References], page 9.
[2] See item [**SWI Prolog**] in Chapter 6 [References], page 9.
[3] See item [**R**] in Chapter 6 [References], page 9.
[4] See item [**ggplot2**] in Chapter 6 [References], page 9.
[5] See item [**C3js**] in Chapter 6 [References], page 9.

2 Installation

Cplint_R is provided as part of the[1] package.

You can also install it manually with `pack_install('cplint_r').` using the `swipl` command.

2.1 Dependencies

Cplint_R has both Prolog and R dependencies:

2.1.1 Prolog

- lists
- Cplint
- Rserve Client
- SWISH R library ???

2.1.2 R

- ggplot2

[1] See item **[Cplint on SWISH]** in Chapter 6 [References], page 9.

3 Examples

TODO

4 Protocol

4.1 Interface

The following is a list of exported predicates available to the library users'.

Each predicate corresponds to one of the following categories

- Helper: specific of this library.
- Pita: it's part of Cplint.
- Mcintyre: it's part of Cplint.
- Auc: it's part of Cplint.

Each argument of a predicate correponds to a data type. See the SWI Prolog data types manual[1] and the Learn Prolog Now manual[2]. Have a look at the Cplint help manual[3] to learn in details about the functionality of each predicate.

`build_xy_list` (*X:list*, *Y:list*, *Out:list*) [Helper]
: Given to lists X and Y build an output list *Out* in the form `[X1-Y1,X2-Y2,...,XN-YN]`.

`r_row` (*X:atom*, *Y:atom*, *r(X,Y):atom*) [Helper]
: Given two atoms X and Y, build a relationship *r(X,Y)*.

`get_set_from_xy_list` (*L:list*, *R:list*) [Helper]
: Given an input list L in the form `[X1-Y1,X2-Y2,...,XN-YN]`, transform it in an output list R in the form `[r(X1,Y1),r(X2,Y2),...,r(XN,YN)]`. This means that `R` will contain an X-Y relationship which can be then passed to an R data frame.

`prob_bar_r` (*:Query:atom*) [Pita]
: The predicate computes and plots the probability of *Query* as a bar chart with a bar for the probability of *Query* true and a bar for the probability of *Query* false. If *Query* is not ground, it returns in backtracking all ground instantiations of *Query* together with their probabilities.

`prob_bar_r` (*:Query:atom*, *:Evidence:atom*) [Pita]
: The predicate computes and plots the probability of *Query* given *Evidence* as a bar chart with a bar for the probability of *Query* true and a bar for the probability of *Query* false given *Evidence*. If *Query* / *Evidence* are not ground, it returns in backtracking all ground instantiations of *Query* / *Evidence* together with their probabilities.

`mc_prob_bar_r` (*:Query:atom*, *-Probability:dict*) [Mcintyre]
: See prob_bar_r/2.

---

[1] See item [**SWI Prolog data types**] in Chapter 6 [References], page 9.

[2] See item [**LPN**] in Chapter 6 [References], page 9.

[3] See item [**Cplint**] in Chapter 6 [References], page 9.

`mc_sample_bar_r` (*:Query:atom*, *+Samples:int*) [Mcintyre]
The predicate samples *Query* a number of *Samples* times and plots a bar chart with a bar for the number of successes and a bar for the number of failures. If *Query* is not ground, it considers it as an existential query.

`mc_sample_arg_bar_r` (*:Query:atom*, *+Samples:int*, *?Arg:var*) [Mcintyre]
The predicate samples *Query Samples* times. *Arg* should be a variable in *Query*. The predicate plots a bar chart with a bar for each possible value of L, the list of values of *Arg* for which *Query* succeeds in a world sampled at random. The size of the bar is the number of samples returning that list of values.

`mc_sample_arg_first_bar_r` (*:Query:atom*, *+Samples:int*, *?Arg:var*) [Mcintyre]
The predicate samples *Query Samples* times. *Arg* should be a variable in *Query*. The predicate plots a bar chart with a bar for each value of *Arg* returned as a first answer by *Query* in a world sampled at random. The size of the bar is the number of samples that returned that value. The value is failure if the query fails.

`mc_rejection_sample_arg_bar_r` (*:Query:atom*, *:Evidence:atom*, *+Samples:int*, *?Arg:var*) [Mcintyre]
The predicate calls `mc_rejection_sample_arg/5` and builds an R graph of the results. It plots a bar chart with a bar for each possible value of L, the list of values of *Arg* for which *Query* succeeds given that *Evidence* is true The size of the bar is the number of samples returning that list of values.

`mc_mh_sample_arg_bar_r` (*:Query:atom*, *:Evidence:atom*, *+Samples:int*, *+Lag:int*, *?Arg:var*) [Mcintyre]
The predicate calls `mc_mh_sample_arg/6` and builds an R graph of the results. The predicate plots a bar chart with a bar for each possible value of L, the list of values of *Arg* for which *Query* succeeds in a world sampled at random. The size of the bar is the number of samples returning that list of values.

`mc_mh_sample_arg_bar_r` (*:Query:atom*, *:Evidence:atom*, *+Samples:int*, *+Mix:int*, *+Lag:int*, *?Arg:var*) [Mcintyre]
The predicate calls `mc_mh_sample_arg/7` and builds an R graph of the results. The predicate plots a bar chart with a bar for each possible value of L, the list of values of *Arg* for which *Query* succeeds in a world sampled at random. The size of the bar is the number of samples returning that list of values.

`histogram_r` (*+List:list*, *+NBins:int*) [Mcintyre]
Draws a histogram of the samples in List dividing the domain in *NBins* bins. *List* must be a list of couples of the form [V]-W or V-W where V is a sampled value and W is its weight.

`density_r` (*+List:list*, *+NBins:int*, *+Min:float*, *+Max:float*) [Mcintyre]
Draws a line chart of the density of a sets of samples. The samples are in *List* as couples [V]-W or V-W where V is a value and W its weigth. The lines are drawn dividing the domain in *NBins* bins.

`densities_r` (*+PriorList:list*, *+PostList:list*, *+NBins:int*) [Mcintyre]
: Draws a line chart of the density of two sets of samples, usually prior and post observations. The samples from the prior are in *PriorList* while the samples from the posterior are in *PostList* as couples [V]-W or V-W where V is a value and W its weigth. The lines are drawn dividing the domain in *NBins* bins.

`compute_areas_diagrams_r` (*+LG:list*, *-AUCROC:float*, *-AUCPR:float*) [Auc]
: The predicate takes as input a list *LG* of pairs probability-literal in asceding order on probability where the literal can be an Atom (incading a positive example) or `\+` Atom, indicating a negative example while the probability is the probability of Atom of being true. PR and ROC diagrams are plotted. The predicate returns:
  - *AUCROC*: the size of the area under the ROC curve
  - *AUCPR*: the size of the area under the PR curve See `http://cplint.lamping.unife.it/example/exauc.pl` for an example

4.2 Internals

Important predicates in this library follow a common structure to avoid confusion and promote standardization.

Interface predicates are involved in the interaction between input data from a program and the plot of that same data. These predicates are usable from the programs.

As the name suggests, plotting predicates are only involved in plotting the data.

Finally there are other functions which handle the lists and other types of data.

4.2.1 Interface predicates

All interface predicates have a similar structure. Their names end with `_r` (except the Helpers) in order to distinguish them from the original Cplint predicates.

First and last operations are always `load_r_libraries` and `finalize_r_graph` respectively.

Plotting is done right before the last operation with one of the `geom_` predicates.

A skeleton of the structure follows.

```
<cplint_graphing_predicate>_r(<input>):-
    load_r_libraries,
    <operations on the input>,
    geom_<smt>(<new input, possibly lists>),
    finalize_r_graph.
```

4.2.2 Plotting predicates

Predicates directly involving plotting all start with `geom_` as prefix.

These predicates work with lists wich are then transformed into R data frames, and, as a final instruction, a corresponding plot is generated.

You can visualize the structure with the following pseudocode:

```
geom_<smt>(<Lists or other input>) :-
    <handle lists>,
    <create one or more R data frame with the lists data>,
```

```
    <rename data frame colnames to avoid using default ones>,
    <- ggplot <smt>
```

4.2.3 List handling

List handling is useful to pass information between Prolog and R. This is done thanks to `build_xy_list/3`, `r_row/3` and `get_set_from_xy_list/2` predicates, described in the interface section.

In case there are multiple distributions we simply have to call `get_set_from_xy_list/2` the appropriate number of times, like: `get_set_from_xy_list(<smt>,R#)`.

4.2.4 Main data frame creation

As descibed before, a data frame is useful to pass structured information between Prolog and R.

In Cplint_R in particular, we use `r_data_frame_from_rows/2` provided by the Rserve Client[4] library, in the following manner:

`r_data_frame_from_rows(df[#], R[#])`

For each distribution the optional number is incremented by one.

In case it is the last (or only) data frame then its name will simply be `df`.

4.2.5 Helpers

What follows are some trivial predicates indicated as internal helpers.

```
bin_width(Min,Max,NBins,Width) :-
  D is Max-Min,
  Width is D/NBins.

load_r_libraries :-
    <- library("ggplot2").

finalize_r_graph :-
    r_download.
```

[4] See item **[Rserve Client]** in Chapter 6 [References], page 9.

5 Thanks

TODO

6 References

Some quotations reported here are taken directly from the respective web sites.

- [**Cplint**] "A suite of programs for reasoning with probabilistic logic programs". See `https://github.com/friguzzi/cplint` and `https://github.com/friguzzi/cplint/blob/master/doc/help-cplint.pdf` for the Cplint help manual.

- [**SWI Prolog**] "SWI-Prolog offers a comprehensive free Prolog environment. Since its start in 1987, SWI-Prolog development has been driven by the needs of real world applications. SWI-Prolog is widely used in research and education as well as commercial applications". See `http://www.swi-prolog.org/`

- [**R**] "R is an integrated suite of software facilities for data manipulation, calculation and graphical display". See `https://cran.r-project.org/doc/manuals/r-release/R-intro.html`

- [**ggplot2**] "ggplot2 is a plotting system for R, based on the grammar of graphics, which tries to take the good parts of base and lattice graphics and none of the bad parts. It takes care of many of the fiddly details that make plotting a hassle (like drawing legends) as well as providing a powerful model of graphics that makes it easy to produce complex multi-layered graphics". See `http://ggplot2.org/`

- [**C3js**] "C3 enables deeper integration of charts into your application". See `http://c3js.org/`

- [**SWI Prolog data types**] See `http://www.swi-prolog.org/datatypes.html`.

- [**LPN**] "Learn Prolog Now". See `http://www.learnprolognow.org/lpnpage.php?pagetype=html&pageid=lpn-htmlch1`.

- [**Cplint on SWISH**] "A set of packages that are able to build and install SWISH, Cplint on SWISH and an R environment". See `https://frnmst.github.io/swish-installer/`

- [**Rserve client**] "Rserve client for SWI-Prolog/SWISH". See `https://github.com/JanWielemaker/rserve_client`