Lost API documentation

Lost Members

March 12, 2010

Contents

Chapter 1

Setting up the lost framework

1.1 Obtaining a copy of the lost framework

The lost framework can be obtained using *git* if you have an account on the *mox* server. Assuming that you have *git* installed on your local machine, to get a copy you need to clone the central repository:

```
$ git clone ssh://your-username@mox.ruc.dk/var/git/lost.git
```

1.2 Configuring your copy of the lost framework

After obtaining the lost framework, for instance by checking it out from git, a little configuration is needed to get started.

The file `lost.pl` in the top-most directory of the copy of the framework. In the beginning of the file there are two important facts you may need to change,

```
lost_config(prism_command,'prism').
lost_config(lost_base_directory, '/change/to/local/lost/dir/').
lost_config(platform, windows_or_unix).
```

The option `prism_command` should point to a the main PRISM executable binary. If it is in your *$PATH* then you can usually leave it unchanged. `lost_base_directory` should be the full path of the directory (including trailing /) containing the `lost.pl` file. Note, that even on windows platforms you should use forward slash rather than backslash in the path specification. The value of `platform` should be either `windows` or `unix`.

To get started you can examine and run `example.pl` which is located in the in the `$LOST/scripts/` directory.

Chapter 2

Creating lost models

2.1 Lost model conventions

Each model is located in its own subdirectory of the of the {lost}/models/ directory, henceforth called $MODELS. So for instance, the sample model called sample_model1 is located in {lost}/models/sample_model1/. We will refer to directory as $MODEL.

To integrate into the framework each model must provide a file called interface.pl, which must be located in the same directory as the model. interface.pl can then implement various predefined predicates which serves as an entry point of using the model.

The supported interface predicates which a model may provide are:

- lost_best_annotation/3.
- lost_learn/3

By convention models are expected to store switch probabilities the directory$MODEL/parameters/. Switch parameter files should be given the extension .prb.

Models are allowed to consult files with paths relative to the $MODEL directory, but should under normal circumstances only directly consult file which are located in the $MODEL directory or a subdirectory of it. The exception to this is the file $LOST/lost.pl. Consulting this file gives access to all the shared APIs.

2.1.1 Model interface predicates

This section describes predicates, that when implemented by the interface.pl provided by a model, allows the model provide functionalities to the general framework.

lost_best_annotation(+InputFileNames,+Options,+OutputFilename) The framework calls this predicate to obtain a "best annotation" from the model. The model is free to provide this annotation in any way it sees fit. It is the models responsibility to save the annotation to `OutputFilename`, before the completion of `lost_best_annotation`.

`InputFileNames`: Is a list of filenames (with absolute paths), each containing an input to the model. There is no restriction on the format of the files and the model is expected to be able to parse then. Predicates to parse a wide range of fileformats are supplied in the *io* API (see section 3.2).

`Options`: Is a list of facts on the form, `option(Key,Value)`. This list is use to paramterize the model in various ways. For convinience, option values can be checked an extracted using the the predicates `lost_option` and `lost_required_option`, (see section **??**). Some options may be quite common and it is suggested to use the same `Keys` for such option. An incomplete list of these common option keys are,

- `parameter_file`: Indicates that the model should use the switch probability associated with the `Value`.

`OutputFilename`: Full filename which the resulting "best annotation" should be saved to. The model is expected to save the resulting annotation to this file before the completion of `lost_best_annot`. The *io* API contains some common predicates for saving annotations (see section 3.2).

lost_learn(+InputFileNames,+Options,+OutputFilename) This predicates is used for training models. The model is expected to save the result of the training session (e.g. a switch parameter file or similar) to `OutputFilename`.

`InputFileNames`: Is a list of filenames (with absolute paths), each containing an input to the model. These are used for providing the traning data. There is no restriction on the format of the files and the model is expected to be able to parse then. Predicates to parse a wide range of fileformats are supplied in the *io* API (see section 3.2).

`Options`: Is a list of facts on the form, `option(Key,Value)`. This list is use to paramterize the model in various ways. For convinience, option values can be checked an extracted using the the predicates `lost_option` and `lost_required_option`, (see section **??**).

`OutputFilename`: Full filename which the resulting switch parameters or similar should be saved to. The model is expected to save the result to this file before the completion of `lost_learn`. The *io* API contains some common predicates for saving annotations (see section 3.2).

Chapter 3

Lost shared APIs

To use the lost APIs, the file `$LOST/lost.pl` located in the top-most {`lost`} directory must be consulted. Then, APIs, which are located in the `$LOST/shared` directory can be consulted using the goal,

```
lost_include_api(+APIName)
```

where `APIName` is the name of a Prolog file located in the `$LOST/shared/` directory except the `.pl` extension.

3.1 interface.pl

The API provides the interface to lost models following the conventions described in section 2.1.

get_annotation_file(Model, Inputs, Options, Filename)

This API provides `get_annotation_file/4` which is used to retrieve the best annotation generated by a specified model with specified parameters and input sequences. If no such file currently exists, then the model will be run (e.g. the `lost_best_annotation/3` provided by the model will be called).

The generated annotation files are named according to a convention. All annotation files will be placed in the {`lost`}`/sequences/` directory. The `Filename` is construed according to the following convention:

```
{Modelname}_annot_{Id}.seq
```

The first time an annotation is generated the file `annotation.idx` will be created in this directory. This file serves as a database to map filenames of the generated annotation files to the (models ,inputs,probability parameters) that generated the particular annotations. This database file contains Prolog facts on the form,

```
fileid(Id,Filename,Model,SwitchParameters,InputFiles).
```

The annotation index is automatically maintained by `get_annotation/4` and should normally not be edited by hand.

If annotation for a particular run of a model is not present then `get_annotation_file/4` will start a new PRISM process that invokes the `lost_best_annotation` predicate provided be the model `interface.pl` file. By the contract of model conventions, the model will generate the annotation and save it to the file indicated by the provided filename.

3.2 Input-Output API

In this module (`io.pl`), severals predicates are defined to manipulate `*.seq` files :

- loading information from files that extracts from a file data information used as input of models (sequence annotation for example);
- saving information into a file;
- and maybe more.

3.2.1 Loading information from files

- `load_annotation_from_file(++Type_Info,++Options,++File,--Annotation)`: Generate from `File` a sequence of `Annotation`. It is assumed that `File` is composed of terms. `Type_Info` is used to specify what format of information into file
  - `sequence` means that information is stored into a list. For example,

    ```
    data(Key_Index,1,10,[a,t,c,c,c..]).
    ```

  - `db` means that information is represented by a set of range that specified specific zone (coding region for example)

    ```
    gb(Key_Index,1,10).
    ```

  For each `Type_Info`, several options are available represented by the list `Options`. Options available for `sequence`:

  - [] (default): data list is the 2^{th} argument of the terms and these lists of data are appended;
  - data_position(Num) specified that data list is `Num`th argument of term;
  - `range(Min,Max)` extracts from the list of the complete annotation the sublist from position `Min` to position `Max`;
  - `all_lists`: generate a list of each data list by term. Warning: `range(Min,Max)` is not support by this option.

File Example `toto.seq`:

```
data(Key_Index,1,5,[1,2,3,4,5]).
data(Key_Index,6,10,[6,7,8,9,10]).
data(Key_Index,11,15,[11,12,13,14,15]).
```

Results of request are:

```
| ?- load_annotation_from_file(sequence,[data_position(4)],'toto.seq',R).
R = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15] ?
| ?- load_annotation_from_file(sequence,[data_position(4),range(4,10)],'toto.seq',R).
R = [4,5,6,7,8,9,10] ?
load_annotation_from_file(sequence,[data_position(4),all_lists],'toto.seq',R).
R = [[1,2,3,4,5],[6,7,8,9,10],[11,12,13,14,15]] ?
```

Options available for `db`:

- [] (default): first and the second element of the term defined a range. A list of 0-1 values is generated, 0 when you are outside ranges and 1 you are inside at least one;
- `in_db(Letter)` replaces the default value 1 by `Letter`;
- `out_db(Letter)` replaces the default value 0 by `Letter`;
- `range_position(Min,Max)` allows to specify the position argument number of a term of the minimal and maximal value of term;
- `range(Min,Max)` extracts from the list of the complete annotation the sublist from position `Min` to position `Max`;

File Example `toto.seq`:

```
gb(3,5).
gb(7,9).
gb(8,11). % Overlap ;)
```

Results of request are:

```
| ?- load_annotation_from_file(db,[],'toto.seq',R).
R = [0,0,1,1,1,0,1,1,1,1,1] ?
| ?- load_annotation_from_file(db,[in_db(c),out_db(nc)],'toto.seq',R).
R =  [nc,nc,c,c,c,nc,c,c,c,c,c]?
| ?- load_annotation_from_file(db,[range(3,7)],R).
R = [1,1,1,0,1] ?
| ?- load_annotation_from_file(db,[in_db(c),out_db(nc),range(8,16)],'toto.seq',R).
R =  [c,c,c,nc,nc,nc,nc,nc]?
```

3.3 The accuracy API

Not implemented/included yet.

Chapter 4

Models

Pre-defined models are introduced. These models are used to build more complex models.

4.1 Parsers of Biological Data

To extract information from different Biological database, several parsers have been designed to parse report of analyses available on different web-servers (Easygene, Genemark) and database (Genbank). These parsers generated a series of Prolog terms that can be used after that input of different probabilistic models. `script_parser.pl` collects different scripts to generate different data files.

4.1.1 Parser_fna

4.1.2 Parser_ptt

4.1.3 Parser_Easygene

4.1.4 Parser_Genemark

Index