Contributing to `catlearn`

Catlearn Research Group*

August 20, 2016

This guide assumes you are able to checkout and commit changes to the `catlearn` *repository on R-Forge. If this is not true, read Theußl & Zeileis (2009), or contact me for help.*

Formal models are useful in psychology when they allow unambiguous comparison of the relative adequacy of theories across a broad range of phenomena (Wills & Pothos, 2012). However, the goal of broad comparison is not achievable by any one individual within a reasonable amount of time. `Catlearn` is designed to solve this problem by enabling efficient distributed collaboration.

`Catlearn`[1] is a framework and archive for formal modeling in psychology. It makes use of R's in-built documentation system to build an archival record of Canonical Independently Replicated Phenomena (CIRP), an archive of formal models implemented as stateful list processors, and an archive of simulations. It is intended to supplement, rather than supplant, traditional publication routes. In particular, the documentation within `catlearn` is concise to the point of terseness, and does not obviate the need for more fulsome descriptions provided through traditional publication routes.

Here's how to use `catlearn` to supplement different types of research publication:

Publication of independent replication If this is the first time this phenomenon has been replicated, write a new CIRP for it (Section 1). Cite the CIRP function name in your article. Write the corresponding OAT function (Section 2) and cite that, too. If you have conducted an additional replication of a CIRP already in `catlearn`, edit the CIRP documentation to include a citation of your manuscript, and cite the existing CIRP function name in your manuscript.

Review publication If you have identified one or more CIRP not already registered within `catlearn`, register them and cite those registrations in your review article. Do the same for the corresponding OAT functions.

Modeling publication Make the models available within `catlearn` (Section 3), and cite the model function name in your manuscript. Archive your simulations (Section 5) and input representations (Section 4). Register each simulation

*Email: andy@willslab.co.uk

[1] `Catlearn` was once an acronym (CATegory LEARNing), but should now be treated as an arbitrary proper noun

in the Grid (Section 6). Cite the unique simulation IDs from the Grid in your article.

1 Writing a CIRP entry

1.1 What's a CIRP?

A CIRP is a canonical independently replicated phenomenon. An independent replication is defined as one with no shared authors that is as close to being a direct replication as the published literature will allow. A replication is successful if the key results are statistically significant, and in the same direction as the original study. Of course, many phenomena have boundary conditions. In the CIRP approach, it's a replication if, when you run the experiment the way the original authors did, you get the results they did.

A CIRP is canonical because it involves the selection of a replication that is representative of what is known empirically, and is sufficiently well specified in terms of psychological stimulus representation, and empirical results, to allow meaningful modeling. In the registration of a CIRP, an explanation of the choice of CIRP is provided, so that the choices are transparent and open to rational challenge.

1.2 How do I write a CIRP entry?

A CIRP is added to `catlearn` as a data object. For a concrete example, type `?shin92`. To register a new CIRP, create the appropriate data frame within R, name it by the first author and two-digit year (e.g. `shin92`), and save it as RData with the that name.

Next, write the help file. This is in the `Rd` format. The `Rd` format is straightforward and well-documented (R Foundation, 2016), although normally it is easiest to modify a `Rd` file created by others, which are available in the `catlearn` R-Forge archive, in the `man` directory.

A CIRP Rd file has the following principal sections:

Description A one-paragraph summary of the CIRP.

Format A description of the content of each of the columns of the dataset.

Details A brief summary of how the CIRP was derived, including a reference for a more detailed derivation where appropriate. Also, a brief summary of the structure of the experiment (for fuller details the reader is expected to read the original journal article).

Source States the reference from which the data were drawn.

References Contains any other references used in the help file.

Author Your name and email address.

Finally, add the data file and Rd document to the `catlearn` package, and commit to R-Forge.

2 Writing an OAT function

Every CIRP should have a corresponding OAT (Ordinal Adequacy Test) function. An OAT function determines whether the output of a simulation-archive function reproduces the ordinal results defined in the CIRP documentation. Only one OAT function per CIRP is required, as simulation-archive functions output their predictions in the same format and order as the CIRP function. OAT functions should have the same name as the CIRP function, suffixed by "oat". For an example of an OAT function, see `shin92oat`.

An OAT function takes as input a set of model predictions. It's output is Boolean, indicating whether the OAT has been passed (TRUE) or failed (FALSE).

OAT functions also provide a convenient place to locate the calculation of summary results, and the `xtdo` parameter provides a standard way of doing this. For example, `shin92exalcove()` produces the full CIRP output, but `shin92oat(shin92,xtdo=TRUE)` produces a user-friendly summary of the main results.

3 Writing a model implementation

Models are implemented in `catlearn` as stateful list processors. The concept of a stateful list processor is explained below, using the `slpALCOVE` model implementation as a concrete example.

Coding of the model function itself is at the discretion of the author, but for cross-compatibility with other parts of the `catlearn` package, the input, output, and basic operating procedures of the model implementation must follow the general schema provided in this section. If your model is computationally intensive, consider writing it in a compiled language (e.g. `C++` with the `Rcpp` package)[2]. For information on `Rcpp` see Eddelbuettel & Francois (2011) or Eddelbuettel (2013).

Figure 1 illustrates the basic operating principles of a stateful list processor. It takes two primary inputs from the user: `st` and `tr`. Input `st` ("state") is a list containing the model parameters and the model's initial state. Object `tr` ("training") is a matrix, where each row is one event that is presented to the model. The nature of a list-processor architecture is that the model processes all of these events in the order they are presented.

The first column of `tr`, *ctrl* ("control"), is normally zero, but can be set to other values to change the mode of operation of the model mid-list. Standard options are: 1 = reset the model to its initial state, 2 = freeze learning on the current trial. After *ctrl*, there are a variable number of optional columns that can contain any numerical information the user wishes. These columns are ignored by the list processor; to achieve this the user must set *colskip* equal

[2]Also consider providing an option to run across multiple CPU cores, and/or to use a GPU.

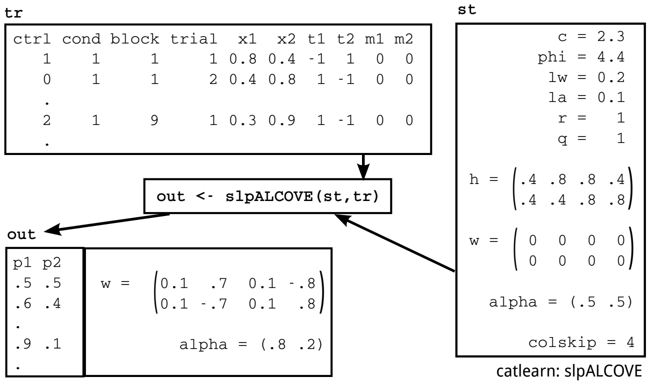

Figure 1: ALCOVE implemented as a stateful list processor in catlearn. Image author: Andy J. Wills. CC BY 4.0.

the number of optional columns, plus one. After these optional columns, the remaining columns contain the input representation.

The simulation is run with the command `slpModelName(st,tr)`, which can have additional non-compulsory arguments. When the simulation completes it returns a list, as illustrated in Figure 1. The returned list has two mandatory components: the model's output on each trial (`p`), and the model's final state. The returning of the final model state is the property that makes the implementation a *stateful* list processor.

Simulation functions must provide concise self-contained documentation for their use with their Rd help file. For an example of the expected format, type `?slpALCOVE`.

When naming your simulation function, please use, `slpModelName`, replacing `ModelName` with a short comprehensible name for the implemented model.

4 Writing an input-representation function

An input-representation function generates, for one or models, an input representation for a specific CIRP. It is a mandatory part of the `catlearn` specification that model input representations are kept separate from both CIRP entries and model implementations. An input-representation function should have a name that begins with the name of the CIRP to be modeled. For example, the `shin92` CIRP has the input-representation function `shin92train`.

Input-representation functions must be documented within their Rd help function. The documentation format is the same as the CIRP entries, with the following minor differences:

Description Single-paragraph summary of the command's purpose.

Usage List of the arguments taken by the function, and their default values.

Arguments Concise description of every argument specified in *Usage*.

Details Documents the meaning of the columns of the matrix produced by the command.

For an example of an input-representation function, see `?shin92train`.

5 Writing a simulation-archive function

A simulation-archive function runs a simulation of *one* CIRP with *one* model. The function's name should include both the CIRP name and the model name, in that order. For example, the simulation of the `shin92` CIRP with the `slpALCOVE` model implementation is named `shin92exalcove`, the "ex" indicating the particular variant of the ALCOVE model used (exemplar variant).

Simulation-archive functions must take the argument `params`, a set of model parameters, and `params` must have default values. These defaults define the archived simulation. Defining them thus allows others to straightforwardly rerun the simulation with different parameters, and examine the results. The fact that these parameters are available to the user also makes it possible to use the simulation-archive function as the subject of a model-fitting procedure. `params` should not include fixed parameters, whether these are fixed because they are universal constants of the model, or because they are determined (formally or otherwise) by the combination of CIRP and model.[3] However, any parameter whose value is at the discretion of the modeler should be included. This includes parameters that have not been formally optimized, but which can nevertheless differ between CIRPs at the discretion of the modeler.

The output of a simulation-archive function is a set of model predictions in the same format and order as the CIRP that is the subject of the simulation. In their default form, simulation-archive functions must complete their operation without any interaction from the user.

The help file has the normal sections: Description, Usage, Arguments, Details, Author, References. There are two types of things that are important to include in the *Details* section of the simulation archive: (1) information that could be derived from the source code but is more convenient if also described in the help file, (2) information that is important to replicating the simulation but is not in the source code. Under the first heading, it is helpful to specify the value of any model parameter not made available through the *params* argument.

Under the second heading, the most likely entry for the help file would be details of how the values in argument *param* were decided (e.g. non-linear optimization). Please consider archiving your process of model fitting within `catlearn`. Do this using a separate function. For example a simulation-archive function `jones2017covis` would have the model-fitting function:

`jones2017covis.opt`.

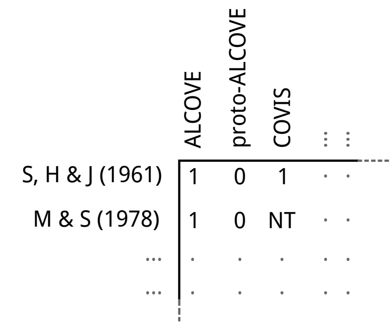

Figure 2: Illustration of the concept of the "Grid". Columns represent formal models, rows represent empirical phenomena. The cell contents indicate: ordinal success of model (1), ordinal failure of model (0), model not tested (NT). Image author: Andy J. Wills. CC BY 4.0.

6 Writing a Grid entry

6.1 What's the "Grid"?

The Grid is a data object that provides a central database of all completed simulations included within `catlearn`. The term "the Grid" derives from an illustration of the concept of broad model comparisons (Figure 2).

The Grid is stored within `catlearn` as a long-format data frame. Load it with the command `data(thegrid)`. Each row of `thegrid` is one completed simulation. The columns record, in turn, a unique ID number for the simulation, the CIRP simulated (e.g. `shin92`), a descriptive name for the model tested (e.g. "protoALCOVE"), the result of the ordinal adequacy test (1 or 0), the command for the simulation archive (e.g. `shin92exalcove`) and the command for the OAT (e.g. `shin92oat`).

In order that The Grid can be automatically checked for accuracy, it is essential that the combination of the simulation-archive command and the OAT command produce the result of the OAT. For example, the following command must always produce the output "1" or "0":

```
shin92oat(shin92exalcove())
```

6.2 How do I write a Grid entry?

Just add a row to the data frame in the normal way, and commit the updated version to R-Forge.

References

Eddelbuettel, D. (2013). *Seamless R and C++ integration with Rcpp*. New York: Springer.

[3]For example, one of the parameters of the ALCOVE model depends on whether the stimuli are integral or separable. Hence, this parameter is determined by the known properties of the CIRP, and should not be included in `params`.

Eddelbuettel, D., & Francois, R. (2011). Rcpp: Seamless R and C++ integration. *Journal of Statistical Software*, *40*, 1–18.

R Foundation. (2016). *Writing R documentation files.* `https://cran.r-project.org/doc/manuals/r-release/R-exts.html#Writing-R-documentation-files`.

Theußl, S., & Zeileis, A. (2009). Collaborative software development using R-Forge. *The R Journal*, *1*, 9–14.

Wills, A. J., & Pothos, E. M. (2012). On the adequacy of current empirical evaluations of formal models of categorization. *Psychological Bulletin*, *138*, 102–125.

ATTRIBUTION Please attribute the text of this document to the *Catlearn Research Group.* Current and previous contributors to the CRG: Andy J. Wills, Garret O'Connell, C.E.R. Edmunds, and Angus B. Inkster.