The Journal of Open Source Software

origami: A Generalized Framework for Cross-Validation in R

Jeremy R. Coyle[1] **and Nima S. Hejazi**[1]

1 Division of Biostatistics, University of California, Berkeley

DOI: 10.21105/joss.00512

Software

- Review
- Repository
- Archive

Submitted: 19 June 2017
Published: 30 January 2018

Summary

Cross-validation is an essential tool for evaluating how any given data analytic procedure extends from a sample to the target population from which the sample is derived. It has seen widespread application in all facets of statistics, perhaps most notably statistical machine learning. When used for model selection, cross-validation has powerful optimality properties (van der Vaart, Dudoit, and van der Laan 2006; van der Laan, Polley, and Hubbard 2007).

Cross-validation works by partitioning a sample into complementary subsets, applying a particular data analytic (statistical) routine on a subset (the "training" set), and evaluating the routine of choice on the complementary subset (the "testing" set). This procedure is repeated across multiple partitions of the data, and a variety of different partitioning schemes exist, such as V-fold cross-validation and bootstrap cross-validation. `origami`, a package for the R language for statistical computing (R Core Team 2017), supports many of the existing cross-validation schemes, providing a suite of tools that generalize the application of cross-validation to arbitrary data analytic procedures.

General workflow

The main function in the `origami` R package is `cross_validate`. To start off, the user must define folds and a function that operates on each fold. Once these are passed to `cross_validate`, this function will map the fold-specific function across the folds, combining the results in a reasonable way. Specific details on each each step of this process are given below.

(1) Define folds

The `folds` object passed to `cross_validate` is a list of folds. Such lists can be generated using the `make_folds` function. Each fold consists of a list with a `training` index vector, a `validation` index vector, and a `fold_index` (its order in the list of folds). This function supports a variety of cross-validation schemes including V-fold and bootstrap cross-validation, as well as time series methods like *"rolling window"*. See (van der Laan, Polley, and Hubbard 2007) for formal definitions of these schemes. `make_folds` can balance across levels of a variable (`stratify_ids`), and it can also keep all observations from the same independent unit together (`cluster`). We invite interested users to consult the documentation of the `make_folds` function for further details.

(2) Define fold function

The `cv_fun` argument to `cross_validate` is a function that will perform some operation on each fold. The first argument to this function must be `fold`, which will receive an individual fold object to operate on. Additional arguments can be passed to `cv_fun` using the `...` argument to `cross_validate`. Within this function, the convenience functions `training`, `validation` and `fold_index` can return the various components of a fold object. If `training` or `validation` is passed an object, it will index into it in a sensible way. For instance, if it is a vector, it will index the vector directly. If it is a `data.frame` or `matrix`, it will index rows. This allows the user to easily partition data into training and validation sets. This fold function must return a named list of results containing whatever fold-specific outputs are generated.

(3) Apply `cross-validate`

After defining folds, `cross_validate` can be used to map the `cv_fun` across the `folds` using `future_lapply`. This means that it can be easily parallelized by specifying a parallelization scheme (i.e., a `plan`). See the `future` package for further details.

The application of `cross_validate` generates a list of results. As described above, each call to `cv_fun` itself returns a list of results, with different elements for each type of result we care about. The main loop generates a list of these individual lists of results (a sort of "meta-list"). This "meta-list" is then inverted such that there is one element per result type (this too is a list of the results for each fold). By default, `combine_results` is used to combine these results type lists in a sensible manner. How results are combined is determined automatically by examining the data types of the results from the first fold. This can be modified by specifying a list of arguments to `.combine_control`. See the help for `combine_results` for more details. In most cases, the defaults should suffice.

References

R Core Team. 2017. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. https://www.R-project.org/.

van der Laan, M.J., E.C. Polley, and A.E. Hubbard. 2007. "Super learner." *Statistical Applications in Genetics and Molecular Biology* 6:Art. 25–23 pp. (electronic).

van der Vaart, A.W., S. Dudoit, and M.J. van der Laan. 2006. "Oracle inequalities for multi-fold cross validation." *Statistics & Decisions* 24 (3):351–71.