Application Note

An R package for an ensemble learning stacking

Taichi Nukui, Akio Onogi

Department of Life Sciences, Faculty of Agriculture, Ryukoku University, 1-5, Yokotani, Seta, Oe-cho, Otsu, Shiga, 520-2194, Japan

Corresponding author:

Akio Onogi

Postal address: 1-5, Yokotani, Seta, Oe-cho, Otsu, Shiga, 520-2194, Japan

Phone & Fax: +81-(0)77-599-5719

E-mail: onogiakio@gmail.com

Abstract

Summary

An R package for stacking, an ensemble approach for supervised learning, has been developed. With the package, training and prediction of stacking can be conducted in one-row scripts, respectively.

Availability and implementation

The R package stacking is available at the Comprehensive R Archive Network (CRAN) (https://cran.r-project.org/) and at github (https://github.com/Onogi/stacking).

Contact

onogiakio@gmail.com

Supplementary information

This manuscript has no supplementary information.

Text

Ensemble learning is a promising approach in prediction tasks in biology (e.g., Sammut *et al.*, 2022). Stacking is one of the ensemble learning methods and is potentially applicable to various biological issues. An application will be for genomic prediction, a statistical technique predicting phenotypes/genetic merits using genome-wide DNA markers. In fact, Liang *et al.* (2021) showed that stacking using support vector regression, kernel ridge regression, and elastic nets as base learners had on average 7.70% higher prediction accuracy in three datasets than single models. However, stacking requires cumbersome scripting. It also requires longer computation time to train the models because multiple models should be trained. In this study, we developed the R package "stacking" to overcome these problems. This package is based on the R

package caret (Kuhn, 2008) which enable users to use various supervised learnings with the same manner via wrapper functions provided by the package. Users of stacking can choose any models supported by caret and can conduct stacking without cumbersome scripting. The package implements parallel computations using the R package snow to be able to parallel training of multiple learners.

The strategy of stacking implemented in our package is illustrated below using an example of regression tasks in which the number of cross-validation (CV) folds is 5 and the number of learners (members of ensemble) is 3. First, CV is conducted with each learner using the training data. Then, using the predicted values of each learner as the explanatory variables (i.e., using three explanatory variables), a meta learner is trained. In this training process, because each learner is trained 5 times, total 15 (3×5) base models are built. In prediction, first, the testing data is given to the base models resulting 15 predicted values. Then the predicted values are averaged for each learner resulting three predicted values. Giving these values as the explanatory variables to the meta model, the final predicted values are obtained. In prediction of classification tasks, predicted values of base models are not averaged. Instead, the most frequent categories are used as explanatory variables for the meta model.

The package stacking mainly consists of four functions: *stacking_train*, *train_basemodel*, *train_metamode*l, and *stacking_predict*. *stacking_train* and *stacking_predict* are functions for training and prediction, respectively. The former internally calls *train_basemodel* and *train_metamodel*, but users can also call these functions themselves to optimize base learners and meta learners. Because the training process of base learners can take long computational time, *train_basemodel* implements a parallel computation using the package snow.

stacking_train mainly takes six arguments, *Y* , *X* , *Nfold* , *Method* , *Metamodel*, and *core*. *Y* and *X* are the vector and matrix of objective and explanatory variables, respectively. *Nfold* is the scalar indicating the number of CV folds. *Method* is a list containing data frames as elements. Names of the elements specify the methods of base learners and are passed to caret functions. Each element (i.e., data frame) includes the hyperparameters of each learner. When the number of rows of the data frame is > 1, i.e., multiple hyperparameter values are given, all combinations of hyperparameter values are automatically created and treated as different base learners. *Metamodel* is a character indicating the meta learner. *core* is a scalar indicating the number of cores for parallel computing. *stacking_train* pass the arguments *Y*, *X*, *Nfold*, *Method*, and *core* to *train_basemodel*, then the list output by *train_basemodel* and the argument *Metamodel* are given to *train_metamodel*. *train_metamodel* also takes an additional argument *which_to_use* which indicating which base models are used for training the meta model. *train_metamodel* then outputs the training results of the meta model as a list which is then output by *stacking_train*. *stacking_predict* mainly takes two arguments, *newX* and *stacking_train_result*. *newX* is a matrix containing explanatory variables of new data, and *stacking_train_result* is a list output by *stacking_train*. *stacking_predict* outputs a vector of predicted values.

First, we demonstrate how to use stacking using simulations. An explanatory variable matrix (*X*) with 1000 samples and 200 explanatory variables was randomly generated from a standard normal distribution as training data. A vector of objective variables (*Y*) was then created by setting the regression coefficient of the first 20 explanatory variables to one and the remaining to zero. Interactions between neighboring variables (i.e., between 1st and 2nd, 2nd and 3rd, and so on) were also

considered by multiplying corresponding variables and giving regression coefficients of one. Then random noises were added such that the signal-noise ratio was four. Testing data of the same size was also generated with the same manner. As base learners, major supervised learning methods including random forests implemented by ranger (Wright and Ziegler, 2017), boosting by xgboost (Chen and Guestrin, 2016) and gbm (Greg, 2007), support vector machine with radial basis function (RBF) kernel by kernlab (Karatzoglou *et al.*, 2004), elastic net by glmnet (Friedman *et al.*, 2010), and partial least squares regression by pls (Mevik and Wehrens, 2015) were used. The numbers of hyperparameter sets were nine for ranger, eight for xgboost, nine for gbm, nine for support vector machine, six for glmnet, and seven for pls, resulting in total 48 base learners. The hyperparameters to be specified can be confirmed by using the *modelLookup* function of caret, and plausible values can be confirmed by applying methods to data with caret. Training and prediction of stacking can be exceeded by one-row scripts, for example,

```
stacking_train_result <- stacking_train(X.train, Y.train, 5, Method, Metamodel, 4)
stacking_predict(X.test, stacking_train_result)
```

where *X.train* and *Y.train* are the explanatory and objective variables of training data, 5 is the number of CV, *Method* is the list specifying the base learners and their hyperparameters, *Metamodel* specifies the meta learner, 4 is the number of cores used for parallel computation, and *X.test* is the explanatory variables of testing data. Prediction accuracy evaluated as the Pearson correlation coefficient between predicted and true values is presented in Figure 1(A). Here stacking was compared with the methods used as base learners. Stacking showed the best accuracy on average.

Next, we used a real data of loblolly pine (Resende *et al.*, 2012) as an example

of regression tasks. The explanatory variables were the genotypes of 4853 SNPs, and the objective variables are the phenotypes of 17 traits. The number of samples was 926: 20% of samples were randomly assigned to testing data, and the remaining 80 % was used for training. The base learners and hyperparameters of stacking were same as the simulation study except for *mtry* of ranger and *ncomp* of pls which were modified according to the data size. The prediction accuracy evaluated as the Pearson correlation coefficient is presented in Figure 1(B). stacking showed best accuracy on average. stacking showed best or nearly best accuracy for most traits whereas rankings of compared methods fluctuated across traits.

Lastly, we used a real data of breast cancer therapy as an example of classification tasks (Sammut *et al.*, 2022). The objective variable is the response to neoadjuvant treatment consisting of four categories, pCR, RCB-, RCB-II, and RCB-III. pCR denotes no tumor, and the remaining indicate tumor residuals in which the magnitude is worse as the number increases. To predict the response, multiple types of information such as clinical features (e.g., tumor grade), RNA features (amounts of transcripts), and DNA features (e.g., mutation burden) were collected from patients. Here, clinical and RNA features were used as explanatory variables because of their high contributions on the response. Prior to evaluation, from total 57905 transcripts, top 2000 transcripts were selected based on p values obtained with Wilcoxon rank sum tests between pCR and the other categories to reduce computational time. The base learners used for regression example above were also used except for RBF kernel of support vector machine replaced with polynomial kernel because RBF kernel often failed to predict minor categories. Random forests implemented by ranger were chosen as the meta learner. Prediction accuracy was evaluated with a five-fold CV. The kappa

coefficients between the observed and predicted categories are presented in Figure 1(C). stacking also showed the best accuracy on average and showed stable performance in terms of ranking among methods irrespective of explanatory variables used.

In conclusion, the package stacking enables users to conduct stacking without cumbersome scripting. Presented experiments show that stacking is able to make prediction accurate and robust irrespective types of tasks and explanatory variables.

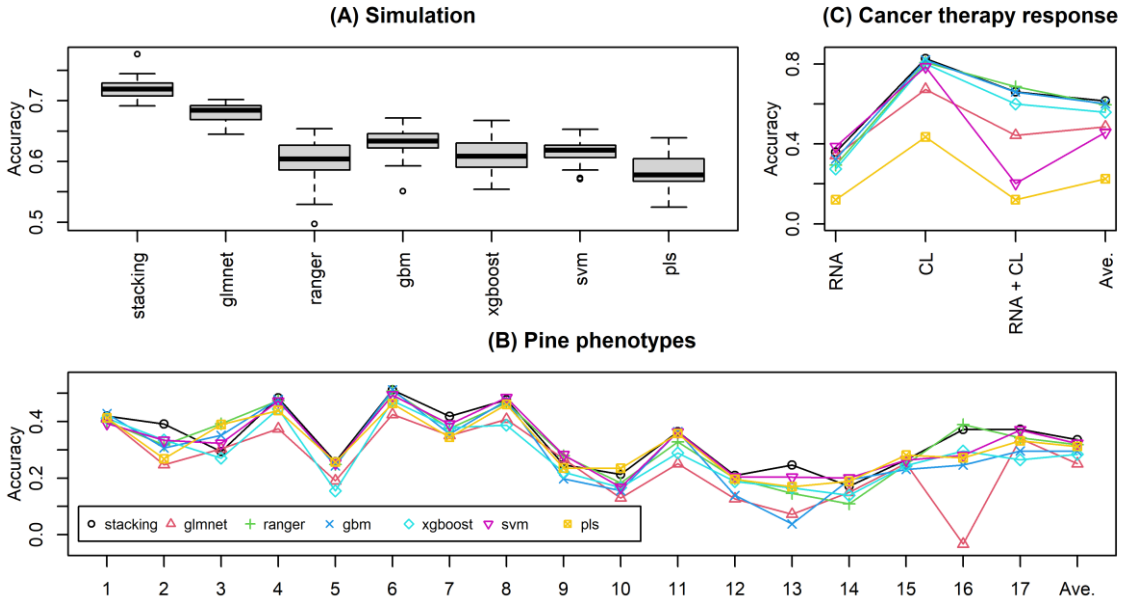

Figure 1 (A) Prediction accuracy evaluated with the Pearson correlation coefficients in simulation analyses (the number of replicates was 20). (B) Prediction accuracy evaluated with the Pearson correlation coefficients in the loblolly pine data analyses. The numbers of x axis indicate traits. The trait abbreviations are 1, HTLC; 2, BA; 3, BD; 4, BLC; 5, C5C6; 6, CWAS; 7, CWAL; 8, DBH; 9, Density; 10, Gall; 11, HT; 12, LateWood.4; 13, Lignin; 14, Rootnum; 15, Rootnumbin; 16, Rustbin; and 17, StiffnessTree. See Resende *et al.* (2012) for the descriptions of these traits. (C) Prediction accuracy evaluated with the kappa coefficients in the cancer therapy data analyses. CL and RNA denote clinical and RNA features, respectively.

Funding information

This study was financially supported by Ryukoku University.

References

Mevik, B.H. and Wehrens, R. (2015) Introduction to the pls Package. Help section of the "Pls" package of R studio software, 1–23.

Chen, T. and Guestrin, C. (2016) Xgboost: A scalable tree boosting system. Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining 785–794.

Friedman, J. *et al.* (2010) Regularization paths for generalized linear models via coordinate descent. *J. Stat. Soft*. **33**, 1–22.

.

Greg, R. (2007) Generalized Boosted Models: A guide to the gbm package. Available at pbil.univ-lyon1.fr.

Karatzoglou, A. *et al.* (2004) kernlab-an S4 package for kernel methods in R. *J. Stat. Soft*. **11**, 1–20.

Kuhn, M. (2008) Building predictive models in R using the caret package. *J. Stat. Soft*. **28**, 1–26.

Liang, M. *et al.* (2021) A stacking ensemble learning framework for genomic prediction.

Front. Genet. **12**, 600040.

Resende, M.F.J. *et al.* (2012) Accuracy of genomic selection methods in a standard data set of loblolly pine (*Pinus taeda* L.). *Genetics*, **190**, 1503–1510.

Sammut, S.J. *et al.* (2022) Multi-omic machine learning predictor of breast cancer therapy response. *Nature* **601**, 623–629.

Wright, M.N. and Ziegler, A. (2017) ranger: A fast implementation of random forests for high dimensional data in C++ and R. *J. Stat. Soft*. **77**, 1–17.