

# SuperLearner

**Eric C. Polley**  
Biometric Research Branch  
National Cancer Institute

**Mark J. van der Laan**  
Division of Biostatistics  
University of California, Berkeley

---

## Abstract

An R package for the Super Learner [van der Laan, Polley, and Hubbard \(2007\)](#) is presented.

*Keywords:* prediction models, cross-validation, R.

---

## 1. Introduction

This vignette explains how to use the **SuperLearner** R package. The **SuperLearner** package provides a syntax and structure to implement the super learner algorithm [van der Laan \*et al.\* \(2007\)](#). A characteristic of the super learner algorithm is the ability to combine many different prediction algorithms together and let the data decide on the optimal ensemble. R is the perfect language for such an algorithm because of the wealth of available prediction algorithms already available in the community. One problem is that prediction algorithms do not have a common syntax, so one of the goals of the **SuperLearner** package is to translate these prediction algorithms into a common syntax to allow for easy programming of the super learner.

## 2. The Super Learner Algorithm

Table 1: Details of prediction algorithm wrappers

Function	Package	Tuning Parameters	Description
bagging	ipred	nbagg minsplit cp maxdepth	Bagging CART trees

(Continued on next page)

Table 1: Details of prediction algorithm wrappers

Function	Package	Tuning Parameters	Description
bart	BayesTree	ntree sigdf sigquant k power base ndpost nskip	Bayesian Regression Trees
bayesglm	arm	prior.mean prior.scale prior.df	Bayesian glm
cforest	party	ntree mtry mincriterion teststat testtype replace fraction	Conditional Tree Forest
cv.spls	spls	K eta	Sparse partial least squares
DSA	DSA	maxsize maxorderint maxsumofpow Dmove Smove vfold	Deletion\Substitution\Addition
earth	earth	degree penalty nk thresh minspan newvar.penalty fast.k fast.beta nfold pmethod	Adaptive Regression Splines
gam	gam	deg.gam	Generalized additive model

(Continued on next page)

Table 1: Details of prediction algorithm wrappers

Function	Package	Tuning Parameters	Description
gbm	gbm	gbm.trees interaction.depth cv.folds shrinkage n.minobsinnode bag.fraction train.fraction	Gradient boosting
glm	stats	–	Generalized linear model
glmnet	glmnet	alpha lambda nlambda lambda.min dfmax type	Elastic Net
knn	class	k use.all	k-Nearest neighbors
loess	stats	span family degree	Local polynomial regression
logreg	LogicReg	ntrees nleaves select penalty kfold control	Logic regression
mars	mda	degree nk penalty thresh prune forward.step	Adaptive Regression Splines
nnet	nnet	size decay rang	Neural network
polymars	polspline	maxsize gcv additive knots	Adaptive polynomial splines
polyclass	polspline	know.space maxdim cv additive	Polychotomous regression

(Continued on next page)

Table 1: Details of prediction algorithm wrappers

Function	Package	Tuning Parameters	Description
randomForest	randomForest	ntree mtry nodesizes sampsize replace maxnodes	Random Forest
Ridge	MASS	lambda	Ridge regression
rpart	rpart	cp minsplit xval maxdepth minbucket	Regression tree
step	stats	scope scale direction steps k	Stepwise regression
step.plr	stepPlr	type lambda cp max.terms	Stepwise penalized logistic
svm	e1071	type kernel nu degree gamma coef0 cost cachesize tolerance epsilon cross	Support vector machine

### 3. Using the SuperLearner Package

```
> library(SuperLearner)
```

#### 3.1. Creating prediction wrappers

A full list of the built-in prediction wrappers can be found with the function:

```
> listWrappers(what = 'SL')
```

```
[1] "SL.bart"           "SL.bayesglm"
[3] "SL.caret"          "SL.caret.rpart"
[5] "SL.cforest"        "SL.earth"
[7] "SL.gam"            "SL.gbm"
[9] "SL.glm"            "SL.glm.interaction"
[11] "SL.glmnet"         "SL.ipredbagg"
[13] "SL.knn"            "SL.leekasso"
[15] "SL.loess"          "SL.logreg"
[17] "SL.mean"           "SL.nnet"
[19] "SL.polymars"       "SL.randomForest"
[21] "SL.ridge"          "SL.rpart"
[23] "SL.rpartPrune"     "SL.step"
[25] "SL.step.forward"   "SL.step.interaction"
[27] "SL.stepAIC"        "SL.svm"
[29] "SL.template"
```

And the included template creator:

```
> write.SL.template(file = '')
```

```
SL.template <- function(Y, X, newX, family, obsWeights, id, ...) {
  # load required packages
  # require('pkg')
  if(family$family == 'gaussian') {

  }
  if(family$family == 'binomial') {

  }
  # pred is the predicted responses for newX (on the scale of the outcome)
  pred <- numeric()
  # fit returns all objects needed for predict.SL.template
  fit <- list(object = )
  # declare class of fit for predict.SL.template
  class(fit) <- 'SL.template'
  # return a list with pred and fit
  out <- list(pred = pred, fit = fit)
  return(out)
}
```

### 3.2. Creating screening wrappers

A full list of the built-in screening wrappers can be found with the function:

```
> listWrappers(what = 'screen')
```

```
[1] "All"
[1] "screen.SIS"          "screen.corP"
[3] "screen.corRank"      "screen.glmnet"
[5] "screen.randomForest" "screen.template"
[7] "screen.ttest"        "write.screen.template"
```

And the included template creator:

```
> write.screen.template(file = '')

screen.template <- function(Y, X, family, obsWeights, id, ...) {
  # load required packages
  # require('pkg')
  if (family$family == 'gaussian') {

  }
  if (family$family == 'binomial') {

  }
  # whichVariable is a logical vector,
  # TRUE indicates variable will be used
  whichVariable <- rep(TRUE, ncol(X))
  return(whichVariable)
}
```

### 3.3. Creating methods wrappers

The included template creator:

```
> write.method.template(file = '')

method.template <- function() {
  out <- list(
    # require allows you to pass a character vector with required packages
    # use NULL if no required packages
    require = NULL,

    # computeCoef is a function that returns a list with two elements:
    # 1) coef: the weights (coefficients) for each algorithm
    # 2) cvRisk: the V-fold CV risk for each algorithm
    computeCoef = function(Z, Y, libraryNames, obsWeights, control, verbose, ...) {
      cvRisk <- numeric()
      coef <- numeric()
      out <- list(cvRisk = cvRisk, coef = coef)
      return(out)
    },
```

```
# computePred is a function that takes the weights and the predicted values
# from each algorithm in the library and combines them based on the model to
# output the super learner predicted values
computePred = function(predY, coef, control, ...) {
  out <- crossprod(t(predY), coef)
  return(out)
}
)
invisible(out)
}
```

## References

van der Laan MJ, Polley EC, Hubbard AE (2007). “Super Learner.” *Statistical Applications in Genetics and Molecular Biology*, **6**(25).

## 4. Computing Environment

- R version 2.14.2 (2012-02-29), x86\_64-apple-darwin9.8.0
- Base packages: base, datasets, grDevices, graphics, methods, stats, utils
- Other packages: SuperLearner 2.0-6, nnls 1.3, quadprog 1.5-4
- Loaded via a namespace (and not attached): tools 2.14.2

### Affiliation:

Eric Polley  
Biometric Research Branch  
National Cancer Institute  
E-mail: [eric.polley@nih.gov](mailto:eric.polley@nih.gov)  
URL: <http://linus.nci.nih.gov/>

Mark van der Laan  
Division of Biostatistics  
University of California, Berkeley  
E-mail: [laan@berkeley.edu](mailto:laan@berkeley.edu)  
URL: <http://www.stat.berkeley.edu/~laan/>