Using the `SuperLearner` R Package

Eric Polley

Biometric Research Branch
National Cancer Institute
National Institute of Health

May 2011
Outline

1. SuperLearner
2. Boston Housing
3. ALL data
4. van’t Veer data
SuperLearner

The package is available at:
https://github.com/ecpolley/SuperLearner

These slides are available in the package and at:
https://github.com/ecpolley/

Need to install R packages nnls and quadprog before installing SuperLearner.
Table: Main functions in the **SuperLearner** package

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SuperLearner</td>
<td>fits super learner</td>
</tr>
<tr>
<td>CV.SuperLearner</td>
<td>cross-validate in super learner</td>
</tr>
<tr>
<td>listWrappers</td>
<td>returns list of wrappers in package</td>
</tr>
<tr>
<td>write.SL.template</td>
<td>prediction wrapper template</td>
</tr>
<tr>
<td>write.screen.template</td>
<td>screening wrapper template</td>
</tr>
<tr>
<td>write.method.template</td>
<td>method wrapper template</td>
</tr>
</tbody>
</table>
SuperLearner

```r
fitSL <- SuperLearner(Y = Y,
                        X = X,
                        SL.library = c('SL.glm'),
                        family = gaussian(),
                        method = 'method.NNLS',
                        verbose = TRUE,
                        cvControl = list(V = 10))
```
Table: Main arguments for SuperLearner

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Req.</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Y$</td>
<td>outcome</td>
<td>Y</td>
<td>–</td>
</tr>
<tr>
<td>$X$</td>
<td>data.frame for fit</td>
<td>Y</td>
<td>–</td>
</tr>
<tr>
<td>newX</td>
<td>data.frame for predict</td>
<td>N</td>
<td>x</td>
</tr>
<tr>
<td>SL.library</td>
<td>library of algorithms</td>
<td>Y</td>
<td>–</td>
</tr>
<tr>
<td>cvControl</td>
<td>list for CV control</td>
<td>N</td>
<td>–</td>
</tr>
<tr>
<td>control</td>
<td>optional controls</td>
<td>N</td>
<td>–</td>
</tr>
<tr>
<td>verbose</td>
<td>detailed report</td>
<td>N</td>
<td>FALSE</td>
</tr>
<tr>
<td>family</td>
<td>error distribution</td>
<td>N</td>
<td>gaussian</td>
</tr>
<tr>
<td>method</td>
<td>loss function &amp; model</td>
<td>N</td>
<td>NNLS</td>
</tr>
<tr>
<td>id</td>
<td>cluster id</td>
<td>N</td>
<td>–</td>
</tr>
<tr>
<td>obsWeights</td>
<td>observations weights</td>
<td>N</td>
<td>–</td>
</tr>
</tbody>
</table>
SuperLearner

- $Y$ and $X$ are the data used to fit each algorithm (the learning data)
- $newX$ is not required but can be a helpful shortcut. $newX$ will not be used to fit the models.

Example with $X$ and $newX$:

```r
fit <- glm(Y ~ ., data = X)
out <- predict(fit, newdata = newX)
```

The formula $Y \sim \cdot$ means an additive linear model using all columns of $X$.
newX might be a test set, the interesting values of X for prediction, a stacked data.frame with exposure levels set to be used with for G-computation, etc

Example setting exposure level for newX

```r
newData <- rbind(
  cbind(A = 0, subset(X, select = -A)),
  cbind(A = 1, subset(X, select = -A))
)
```
SuperLearner

- family: currently either `gaussian()` or `binomial()`.
- method: either `method.NNLS`, `method.NNloglik`, or your own method (see `create.method.template()`).
- verbose: helpful to set this to `TRUE` to see the progress of the estimation.
The ensemble model for “NNLS” is:

$$\Psi_{SL}(W) = \sum_{j=1}^{K} \alpha_j \Psi_j(W), \quad \alpha_j \geq 0, \quad \sum \alpha_j = 1$$

The ensemble model for “NNloglik” is:

$$\Psi_{SL}(W) = \frac{1}{1 + \exp \left\{ - \sum_{j=1}^{K} \alpha_j \logit_{\gamma} (\Psi_j(W)) \right\}}, \quad \alpha_j \geq 0, \quad \sum \alpha_j = 1$$

where logit$_{\gamma}$ is the trimmed logit function to control when $\Psi_j(W)$ is near 0 or 1.
There are two types of algorithms that can be used in SL.library:

1. **Prediction algorithms.** Algorithms that take as input $X$ and $Y$ and return a predicted $Y$ value.

2. **Screening algorithms.** Algorithms designed to reduce the dimension of $X$. They take as input $X$ and $Y$ and return a logical vector indicating the columns in $X$ passing the screening. Screening algorithms can be coupled with prediction algorithms to form new prediction algorithms.
listWrappers()
There are two ways to specify the algorithms in `SL.library`:

1. A character vector:
   
   ```
   c(‘SL.glm’, ‘SL.glmnet’, ‘SL.gam’) 
   ```

2. A list of character vectors:
   
   ```
   list(c(‘SL.glm’, ‘screen.corP’), ‘SL.gam’) 
   ```

If only using prediction algorithms, easier to use the first method.

If using screening algorithms, the list is required. The syntax for the elements in the list is the prediction algorithm is first, followed by the screening algorithms. Multiple screening algorithms can be used. If a singleton, the default is to apply to all variables.
see the help documents for SuperLearner for more examples of SL.library
Creating Wrappers

Many algorithms are included in the package (use listWrappers() for a list of included functions), but these are just enough to get you started.

A few reasons to build your own wrappers:

• Want to use an algorithm not currently included
• Problem suggests different values for the tuning parameters
• Want to include a range of tuning parameters, not just the default
• Want to select tuning parameters in a different way (e.g. SL.glmnet selecting \( \lambda \))
• Force variables to be used in step-wise methods
The **SuperLearner** vignette contains a table of tuning parameters for the algorithms in the package.

```r
vignette("SuperLearner")
```
Example: Creating new prediction algorithm wrapper
Creating wrappers

Consider the `polymars` algorithm in the `polspline` package.

- continuous outcome Y
- data.frame of covariates X
- data.frame of covariates newX

```r
fit.mars <- polymars(Y, X)
out <- predict.polymars(fit.mars,
    x = as.matrix(newX))
```

Now we know how to fit the model and return predicted values, next we check out `write.SL.template` for integrating the code above into the correct syntax for SuperLearner
Creating wrappers

```r
SL.template <- function(Y, X, newX, family, obsWeights, id, ...) {
  # require ('pkg ')
  if(family$family == 'gaussian') { }
  if(family$family == 'binomial') { }
  # pred is the predicted responses for newX
  pred <- numeric()
  # fit returns all objects needed for predict.SL.*
  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)
}
write.SL.template()
```
## Creating wrappers

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Y$</td>
<td>the outcome variable</td>
</tr>
<tr>
<td>$X$</td>
<td>the training data set (the observations used to fit the model)</td>
</tr>
<tr>
<td>newX</td>
<td>the validation data set (the observations to return predictions for)</td>
</tr>
<tr>
<td>family</td>
<td>a description of the error distribution</td>
</tr>
<tr>
<td>id</td>
<td>a cluster identification</td>
</tr>
<tr>
<td>obsWeights</td>
<td>observation weights</td>
</tr>
</tbody>
</table>

You do not need to use all these arguments, but if you use any of them, the name must match exactly.
Creating wrappers

My.SL.polymars <- function(Y, X, newX, family, ...) {
  if(family$family == "gaussian") {
    fit.mars <- polymars(Y, X)
    out <- predict.polymars(fit.mars, x = as.matrix(newX))
  }
  if(family$family == "binomial") {
# insert estimation function
  }
  ...
# next slide
}

SL.polymars
What about family = binomial()?

Can leave this blank (or add stop('only gaussian')) if only for a specific example with continuous outcome.

To be complete, we could look up the code for a binary outcome and add this case:

```r
fit.mars <- polyclass(Y, X, cv = 5)
out <- ppolyclass(cov = newX,
    fit = fit.mars)[, 2]
```
Creating wrappers

My.SL.polymars <- function (Y, X, newX, family, ...) {
  if(family$family == "gaussian") {
    fit.mars <- polymars(Y, X)
    out <- predict.polymars(fit.mars, x = as.matrix(newX))
  }
  if(family$family == "binomial") {
    fit.mars <- polycalss(Y, X, cv = 5)
    out <- ppolycalss(cov = newX, fit = fit.mars)[, 2]
  }
  ...
} # next slide
Creating wrappers

Wrappers need to return 2 values:

1. **pred**: predicted $Y$ values for rows in $\text{newX}$
2. **fit**: a list with everything needed to use `predict` method

In the `polymars` example: For the gaussian case, `predict()` needs:
```
object = fit.mars
```
For the binomial case, `predict()` needs:
```
fit = fit.mars
```

**Note**: `SuperLearner` does not use the `fit` list. If you do not plan to use the function `predict.SuperLearner` you can leave the `fit` object as:
```
fit <- vector("list", length = 0)
```
Creating wrappers

My.SL.polymars <- function(Y, X, newX, family, ...) {
  if(family$family == "gaussian") {
    fit.mars <- polymars(Y, X)
    out <- predict.polymars(fit.mars, x = as.matrix(newX))
    fit <- list(object = fit.mars)
  }
  if(family$family == "binomial") {
    fit.mars <- polyclass(Y, X, cv = 5)
    out <- ppolyclass(cov = newX, fit = fit.mars)[, 2]
    fit <- list(fit = fit.mars)
  }
  ...
  # next slide
}
Creating wrappers

Final step is putting everything together into a list object. The list must have 2 elements and the names must be `pred` and `fit`.

Can also assign a class to the `fit` list. This will be used to look up the correct `predict` method. I’m using S3 methods here. This is only important if using `predict.SuperLearner` afterwards.

Creating wrappers

```
My.SL.polymars <- function(Y, X, newX, family, ...) {
    ... # previous slides
    out <- list(pred = pred, fit = fit)
    class(out$fit) <- c("SL.polymars")
    return(out)
}
```

Note: `out` is just a temporary variable name here.

The function should match `SL.polymars` in the SuperLearner package.
Creating wrappers

Important notes for creating wrappers

- Input must following naming syntax: Y, X, ...
- Name of new function must be different than one already in the package
- Must return a list with 2 elements named `pred` and `fit`
- `pred` must be a vector with the predicted Y values for the rows in `newX`
- `fit` can be anything if not using predict method, otherwise is a list with elements needed for `predict`
Creating wrappers

```r
predict.SL.template <- function (object, newdata, family, X = NULL, Y = NULL, ...)
{
  pred <- numeric()
  return(pred)
}
```
Creating wrappers

predict.SL.polymars

predict.SL.polymars <-
function (object, newdata, family, ...) {
  if (family$family == "gaussian") {
    pred <- predict.polymars(object = object$object,
                           x = as.matrix(newdata))
  }
  if (family$family == "binomial") {
    pred <- ppolyclass(cov = newdata,
                       fit = object$fit)[, 2]
  }
  return(pred)
}


Creating wrappers

Example: creating screening algorithm
CREATING WRAPPERS

screening template

```r
screen.template <- function (Y, X, family, obsWeights, id, ...) {
  # 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)
}
```
## Creating wrappers

Table: The arguments passed to a screening algorithm in SuperLearner

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Y$</td>
<td>the outcome variable</td>
</tr>
<tr>
<td>$X$</td>
<td>the training data set (the observations used to fit the model)</td>
</tr>
<tr>
<td>family</td>
<td>a description of the error distribution</td>
</tr>
<tr>
<td>id</td>
<td>a cluster identification</td>
</tr>
<tr>
<td>obsWeights</td>
<td>observation weights</td>
</tr>
</tbody>
</table>

You do not need to use all these arguments, but if you use any of them, the name must match exactly.
screen.randomForest <- function (Y, X, 
  family, nVar = 10, ntree = 1000, ...) {
  if (family$family == "gaussian") {
    rank.rf.fit <- randomForest(Y ~ ., 
      data = X, ntree = ntree)
  }
  if (family$family == "binomial") {
    rank.rf.fit <- randomForest(
      y = as.factor(Y), x = X, 
      ntree = ntree)
  }
  whichVariable <- as.logical(
    rank(-rank.rf.fit$importance) <= nVar)
  return(whichVariable)
}
The outcome variable is the median home value (cmedv) for the 506 census tracts of Boston from the 1970 census.

The covariates are a mix of geographical and socioeconomic variables, like per capita crime rate (crim), average number of rooms per house (rm), distance to Boston employment centres (dis), indicator of tract being on the Charles river (chas), etc.
The Boston Housing data can be found in the `mlbench` package.

```r
library(mlbench)
data(BostonHousing2)
# convert factors to numeric
BostonHousing2$chas <- as.numeric(BostonHousing2$chas=="1")
# select subset of variables
DATA <- BostonHousing2[, c("cmedv", "crim", "zn", "indus", "chas", "nox", "rm", "age", "dis", "rad", "tax", "ptratio", "b", "lstat")]
```
Boston Housing example

First need to decide which prediction algorithms to include in the library
## Boston Housing

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Description</th>
<th>Package</th>
</tr>
</thead>
<tbody>
<tr>
<td>glm</td>
<td>linear model</td>
<td>stats</td>
</tr>
<tr>
<td>randomForest</td>
<td>random Forest</td>
<td>randomForest</td>
</tr>
<tr>
<td>bagging</td>
<td>bootstrap aggregation of trees</td>
<td>ipred</td>
</tr>
<tr>
<td>gam</td>
<td>generalized additive models</td>
<td>gam</td>
</tr>
<tr>
<td>gbm</td>
<td>gradient boosting</td>
<td>gbm</td>
</tr>
<tr>
<td>nnet</td>
<td>neural network</td>
<td>nnet</td>
</tr>
<tr>
<td>polymars</td>
<td>polynomial spline regr.</td>
<td>polspline</td>
</tr>
<tr>
<td>bart</td>
<td>Bayesian additive regr. trees</td>
<td>BayesTree</td>
</tr>
<tr>
<td>glmnet</td>
<td>elastic net</td>
<td>glmnet</td>
</tr>
<tr>
<td>svm</td>
<td>support vector machine</td>
<td>e1071</td>
</tr>
<tr>
<td>bayesglm</td>
<td>Bayesian glm</td>
<td>arm</td>
</tr>
<tr>
<td>step</td>
<td>stepwise glm</td>
<td>stats</td>
</tr>
</tbody>
</table>
One algorithm to consider is the generalized additive model algorithm. This algorithm has a tuning parameter for the degrees of freedom in the smoother. I have set this to be 2 in \texttt{SL.gam} but we might want to consider larger values.

We could create an entirely new wrapper for \texttt{gam} and \texttt{df} = 3, or we can write a wrapper for the wrapper and only change the degrees of freedom value.
Boston Housing

look at SL.gam to see how the degrees of freedom parameter is specified:
SL.gam <- function(Y, X, newX, family, obsWeights, deg.gam = 2, ...)
{
  ... # model: Y ~ s(X, deg.gam)
  # see full functions for details
  fit.gam <- gam::gam(gam.model, data = X,
                     family = family,
                     control = gam.control(maxit = 50, bf.maxit = 50),
                     weights = obsWeights)
  pred <- predict(fit.gam, newdata = newX,
                  type = "response")
  ... # returns list here
}
The $\text{SL.gam}$ function contains the argument $\text{deg.gam} = 2$.

Wrappers can have additional arguments, but they must have default values.
For the new wrapper, only need to change the value of `deg.gam`. Use ... to pass everything else between `SL.gam.3` and `SL.gam`.

Easy to create new wrappers by changing tuning parameter values. Check the code for the wrappers by typing the name of the function without parentheses to see what tuning parameter values are in the arguments.
Similar to the SL.gam example above, the function create.SL.glmnet in the **SuperLearnerExtra** package can be used to create new SL.glmnet wrappers:

```r
create.SL.glmnet(alpha = c(0.25, 0.50, 0.75))
# and set gbm to no interactions:
SL.gbm.1 <- function(...) {
  SL.gbm(..., interaction.depth = 1)
}
```

Additional wrappers available at: https://github.com/ecpolley/SuperLearnerExtra
Boston Housing example

SL.library <- c("SL.gam",
    "SL.gam.3", "SL.gam.4",
    "SL.gam.5", "SL.gbm.1",
    "SL.gbm", "SL.glm",
    "SL.glmnet", "SL.glmnet.0.25",
    "SL.glmnet.alpha.0.5", "SL.glmnet.0.75",
    "SL.polymars", "SL.randomForest",
    "SL.ridge", "SL.svm",
    "SL.bayesglm", "SL.step",
    "SL.step.interaction",
    "SL.bart")
Boston Housing example

```r
fitSL <- SuperLearner(Y = log(DATA$cmedv),
                      X = subset(DATA, select = -c(cmedv)),
                      SL.library = SL.library,
                      family = gaussian()
)
```

<table>
<thead>
<tr>
<th>Model</th>
<th>Risk</th>
<th>Coef</th>
</tr>
</thead>
<tbody>
<tr>
<td>SL.gam_All</td>
<td>0.03834031</td>
<td>0.00000000</td>
</tr>
<tr>
<td>SL.gam.3_All</td>
<td>0.03666449</td>
<td>0.00000000</td>
</tr>
<tr>
<td>SL.gam.4_All</td>
<td>0.03589859</td>
<td>0.00000000</td>
</tr>
<tr>
<td>SL.gam.5_All</td>
<td>0.03529692</td>
<td>0.00000000</td>
</tr>
<tr>
<td>SL.gbm.1_All</td>
<td>0.03040543</td>
<td>0.00000000</td>
</tr>
<tr>
<td>SL.gbm.2_All</td>
<td>0.02501729</td>
<td>0.00000000</td>
</tr>
<tr>
<td>SL.glm_All</td>
<td>0.03754472</td>
<td>0.00000000</td>
</tr>
<tr>
<td>SL.glmnet_All</td>
<td>0.03765112</td>
<td>0.00000000</td>
</tr>
<tr>
<td>SL.glmnet.alpha25_All</td>
<td>0.03754278</td>
<td>0.00000000</td>
</tr>
<tr>
<td>SL.glmnet.alpha50_All</td>
<td>0.03758802</td>
<td>0.00000000</td>
</tr>
<tr>
<td>SL.glmnet.alpha75_All</td>
<td>0.03763085</td>
<td>0.00000000</td>
</tr>
<tr>
<td>SL.polymars_All</td>
<td>0.04587432</td>
<td>0.00000000</td>
</tr>
<tr>
<td>SL.randomForest_All</td>
<td>0.02105987</td>
<td>0.2956277</td>
</tr>
<tr>
<td>SL.ridge_All</td>
<td>0.03753661</td>
<td>0.00000000</td>
</tr>
<tr>
<td>SL.svm_All</td>
<td>0.02678290</td>
<td>0.00000000</td>
</tr>
<tr>
<td>SL.bayesglm_All</td>
<td>0.03754318</td>
<td>0.00000000</td>
</tr>
<tr>
<td>SL.step_All</td>
<td>0.03753337</td>
<td>0.00000000</td>
</tr>
</tbody>
</table>
Table: Elements of the output from SuperLearner

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SL.predict</td>
<td>super learner predicted values for newX</td>
</tr>
<tr>
<td>coef</td>
<td>coefficient for each algorithm</td>
</tr>
<tr>
<td>libraryNames</td>
<td>names of algorithms in library</td>
</tr>
<tr>
<td>library.predict</td>
<td>matrix of predicted values for newX from each algorithm in the library</td>
</tr>
<tr>
<td>cvRisk</td>
<td>V-fold cross-validated risk for each algorithm in the library</td>
</tr>
</tbody>
</table>
The final super learner prediction model is the weighted combination of the library algorithms where the estimates of the weights can be found with `coef(fitSL)`.

To attain predictions on new observations (not in `newX`), the `predict` function will usually work. If you created new wrappers, you also need to create `predict` S3 methods for those new wrappers.
**Boston Housing**

SuperLearner is a model selection algorithm. It does not contain a good estimate for model assessment (you could use the re-substitution method to estimate the risk but this is optimistic).

Our suggestion to assess the performance of the super learner is to run CV.SuperLearner (example in the next case study).
The outcome variable is an indicator of the molecular biology of the cancer tissue, either Negative or BCR/ABL.
The sample consists of 79 individuals (42 Neg, 37 BCR/ABL).
The data contain 2200 features (X) to be used after the filtering steps.
Need to select algorithms appropriate for a binary outcome and a large number of covariates.
load ALL data

# source("http://bioconductor.org/biocLite.R")
# biocLite()
# biocLite("ALL")
library(ALL)
library(genefilter)
data(ALL)

The next 2 slides are the processing steps following in Gentleman, Huber and Carey (2008) “Supervised Machine Learning” in Bioconductor Case Studies.
# restrict to only the NEG and BCR/ABL outcomes
bcell <- grep("^B", as.character(ALL$BT))
moltyp <- which(as.character(ALL$mol.biol) %in% c("NEG", "BCR/ABL"))
ALL_bcrneg <- ALL[, intersect(bcell, moltyp)]
# drops unused levels
ALL_bcrneg$mol.biol <- factor(ALL_bcrneg$mol.biol)

# filter features
ALLfilt_bcrneg <- nsFilter(ALL_bcrneg, var.cutoff = 0.75)$eset
# standardize the features
rowIQRs <- function(eSet) {
  numSamp <- ncol(eSet)
  lowQ <- rowQ(eSet, floor(0.25 * numSamp))
  upQ <- rowQ(eSet, ceiling(0.75 * numSamp))
  upQ - lowQ
}
standardize <- function(x) {
  (x - rowMedians(x)) / rowIQRs(x)
}
exprs(ALLfilt_bcrneg) <- standardize(exprs(ALLfilt_bcrneg))

# convert to numeric matrix for the SuperLearner
Y <- as.numeric(ALLfilt_bcrneg$mol.biol == "BCR/ABL")
X <- t(exprs(ALLfilt_bcrneg))
Possible prediction algorithms include:
  • k-nearest neighbors
  • elastic net (penalized regression)
  • random forest
These algorithms have tuning parameters:
  • knn: k
  • glmnet: α
  • randomForest: mtry and nodesize
tuneGrid <- expand.grid(mtry = c(500, 1000, 2200),
                        nodesize = c(1, 5, 10))

for (mm in seq(nrow(tuneGrid))) {
  eval(parse(file = "", text =
    paste("SL.randomForest.", mm,
    " <- function(..., mtry = ", tuneGrid[mm, 1],
    ", nodesize = ", tuneGrid[mm, 2], ") {
    SL.randomForest(..., mtry = mtry,
    nodesize = nodesize) "}", sep = "")))
}
The code above is hard to follow, but I’m doing the same thing we did with `SL.gam.3` just in a for loop.

```r
> SL.randomForest.1
function(..., mtry = 500, nodesize = 1) {
  SL.randomForest(..., mtry = mtry,
                 nodesize = nodesize)
}

> SL.randomForest.2
function(..., mtry = 1000, nodesize = 1) {
  SL.randomForest(..., mtry = mtry,
                 nodesize = nodesize)
}

> SL.randomForest.9
function(..., mtry = 2200, nodesize = 10) {
  SL.randomForest(..., mtry = mtry,
                 nodesize = nodesize)
}
```
Add additional knn wrappers using functions in SuperLearnerExtra

create.SL.knn

create.SL.knn(k = c(k = 20, 30, 40, 50))

available at: https://github.com/ecpolly/SuperLearnerExtra
SL.library <- c("SL.knn",
    "SL.knn.20",
    "SL.knn.30",
    "SL.knn.40",
    "SL.knn.50",
    "SL.randomForest",
    "SL.glmnet",
    "SL.glmnet.0.25",
    "SL.glmnet.0.5",
    "SL.glmnet.0.75",
    "SL.mean",
    paste("SL.randomForest.",
          seq(nrow(tuneGrid)), sep = ""))
```r
fitSL <- SuperLearner(Y = Y, X = X, 
SL.library = SL.library, family = binomial(), 
method = "NNLS",
cvControl = list(stratifyCV = TRUE))
fitSL
```
<table>
<thead>
<tr>
<th>Model</th>
<th>Risk</th>
<th>Coef</th>
</tr>
</thead>
<tbody>
<tr>
<td>SL.knn_All</td>
<td>0.20658228</td>
<td>0.00000000</td>
</tr>
<tr>
<td>SL.knn20_All</td>
<td>0.22423347</td>
<td>0.00000000</td>
</tr>
<tr>
<td>SL.knn30_All</td>
<td>0.22299664</td>
<td>0.00000000</td>
</tr>
<tr>
<td>SL.knn40_All</td>
<td>0.23445986</td>
<td>0.00000000</td>
</tr>
<tr>
<td>SL.knn50_All</td>
<td>0.23920321</td>
<td>0.00000000</td>
</tr>
<tr>
<td>SL.randomForest_All</td>
<td>0.12418009</td>
<td>0.00000000</td>
</tr>
<tr>
<td>SL.glmnet_All</td>
<td>0.08430633</td>
<td>0.98039534</td>
</tr>
<tr>
<td>SL.glmnet.alpha25_All</td>
<td>0.10487930</td>
<td>0.00000000</td>
</tr>
<tr>
<td>SL.glmnet.alpha50_All</td>
<td>0.09331539</td>
<td>0.00000000</td>
</tr>
<tr>
<td>SL.glmnet.alpha75_All</td>
<td>0.08681511</td>
<td>0.00000000</td>
</tr>
<tr>
<td>SL.randomForest.1_All</td>
<td>0.13103528</td>
<td>0.00000000</td>
</tr>
<tr>
<td>SL.randomForest.2_All</td>
<td>0.12269094</td>
<td>0.00000000</td>
</tr>
<tr>
<td>SL.randomForest.3_All</td>
<td>0.11918439</td>
<td>0.00000000</td>
</tr>
<tr>
<td>SL.randomForest.4_All</td>
<td>0.13024104</td>
<td>0.00000000</td>
</tr>
<tr>
<td>SL.randomForest.5_All</td>
<td>0.12351049</td>
<td>0.00000000</td>
</tr>
<tr>
<td>SL.randomForest.6_All</td>
<td>0.11752733</td>
<td>0.01960466</td>
</tr>
<tr>
<td>SL.randomForest.7_All</td>
<td>0.12871385</td>
<td>0.00000000</td>
</tr>
</tbody>
</table>
```r
fitSL.CV <- CV.SuperLearner(Y = Y, X = X,
    SL.library = SL.library,
    V = 20, family = binomial(),
    method = "method.NNLS",
    cvControl = list(stratifyCV = TRUE))

summary(fitSL.CV)

# can also print the LaTeX table
# requires Hmisc package
# latex(summary(fitSL.CV))
```
<table>
<thead>
<tr>
<th>Algorithm</th>
<th>subset</th>
<th>Risk</th>
<th>SE</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>SuperLearner</td>
<td>–</td>
<td>0.101</td>
<td>0.020</td>
<td>0.003</td>
<td>0.382</td>
</tr>
<tr>
<td>Discrete SL</td>
<td>–</td>
<td>0.095</td>
<td>0.021</td>
<td>0.004</td>
<td>0.347</td>
</tr>
<tr>
<td>SL.knn(10)</td>
<td>All</td>
<td>0.212</td>
<td>0.018</td>
<td>0.070</td>
<td>0.460</td>
</tr>
<tr>
<td>SL.knn(20)</td>
<td>All</td>
<td>0.220</td>
<td>0.011</td>
<td>0.152</td>
<td>0.322</td>
</tr>
<tr>
<td>SL.knn(30)</td>
<td>All</td>
<td>0.223</td>
<td>0.008</td>
<td>0.192</td>
<td>0.274</td>
</tr>
<tr>
<td>SL.knn(40)</td>
<td>All</td>
<td>0.232</td>
<td>0.006</td>
<td>0.187</td>
<td>0.268</td>
</tr>
<tr>
<td>SL.knn(50)</td>
<td>All</td>
<td>0.238</td>
<td>0.004</td>
<td>0.218</td>
<td>0.260</td>
</tr>
<tr>
<td>SL.randomForest</td>
<td>All</td>
<td>0.120</td>
<td>0.014</td>
<td>0.026</td>
<td>0.256</td>
</tr>
<tr>
<td>SL.glmnet(α = 1.0)</td>
<td>All</td>
<td>0.088</td>
<td>0.022</td>
<td>0.002</td>
<td>0.395</td>
</tr>
<tr>
<td>SL.glmnet(α = 0.25)</td>
<td>All</td>
<td>0.113</td>
<td>0.022</td>
<td>0.007</td>
<td>0.451</td>
</tr>
<tr>
<td>SL.glmnet(α = 0.50)</td>
<td>All</td>
<td>0.106</td>
<td>0.023</td>
<td>0.004</td>
<td>0.447</td>
</tr>
<tr>
<td>SL.glmnet(α = 0.75)</td>
<td>All</td>
<td>0.093</td>
<td>0.021</td>
<td>0.004</td>
<td>0.347</td>
</tr>
<tr>
<td>SL.mean</td>
<td>All</td>
<td>0.249</td>
<td>0.004</td>
<td>0.242</td>
<td>0.251</td>
</tr>
<tr>
<td>SL.randomForest.1</td>
<td>All</td>
<td>0.125</td>
<td>0.014</td>
<td>0.034</td>
<td>0.269</td>
</tr>
<tr>
<td>SL.randomForest.2</td>
<td>All</td>
<td>0.114</td>
<td>0.014</td>
<td>0.023</td>
<td>0.250</td>
</tr>
<tr>
<td>SL.randomForest.3</td>
<td>All</td>
<td>0.111</td>
<td>0.015</td>
<td>0.016</td>
<td>0.238</td>
</tr>
<tr>
<td>SL.randomForest.4</td>
<td>All</td>
<td>0.123</td>
<td>0.014</td>
<td>0.036</td>
<td>0.264</td>
</tr>
<tr>
<td>SL.randomForest.5</td>
<td>All</td>
<td>0.117</td>
<td>0.014</td>
<td>0.023</td>
<td>0.262</td>
</tr>
<tr>
<td>SL.randomForest.6</td>
<td>All</td>
<td>0.110</td>
<td>0.015</td>
<td>0.015</td>
<td>0.252</td>
</tr>
<tr>
<td>SL.randomForest.7</td>
<td>All</td>
<td>0.126</td>
<td>0.014</td>
<td>0.034</td>
<td>0.266</td>
</tr>
<tr>
<td>SL.randomForest.8</td>
<td>All</td>
<td>0.117</td>
<td>0.014</td>
<td>0.023</td>
<td>0.259</td>
</tr>
<tr>
<td>SL.randomForest.9</td>
<td>All</td>
<td>0.110</td>
<td>0.015</td>
<td>0.013</td>
<td>0.245</td>
</tr>
</tbody>
</table>
van’t Veer example

- 97 breast cancer patients followed for 5 years.
- Outcome is binary yes/no recur in 5 years (we do not have the date of recurrence)
- 7 clinical variables are available (age, tumor grade, etc.)
- 4348 gene expression values post-filtering
van't Veer data

The original data is available at:
One interesting “screening” is to consider the prediction algorithms on only the clinical variables or on only the gene expression variables.

```r
screen.clinical <- function(...){
  return(c(rep(TRUE, 7), rep(FALSE, 4348)))
}
screen.array <- function(...){
  return(c(rep(FALSE, 7), rep(TRUE, 4348)))
}
```
SL.library <- list(
  c("SL.knn", "All", "screen.clinical",
    "screen.corP", "screen.corP.01", "screen.glmnet"),
  c("SL.knn.20", "All", "screen.clinical",
    "screen.corP", "screen.corP.01", "screen.glmnet"),
  c("SL(glmnet", "screen.corRank.50",
    "screen.corRank.20"),
  c("SL(glmnet.0.75", "screen.corRank.50",
    "screen.corRank.20"),
  c("SL(glmnet.0.5", "screen.corRank.50",
    "screen.corRank.20"),
  c("SL(glmnet.0.25", "screen.corRank.50",
    "screen.corRank.20"),
  c("SL.randomForest", "screen.clinical",
    "screen.corP.01", "screen.glmnet"),
  c("SL.bagging", "screen.clinical",
    "screen.corP.01", "screen.glmnet"),
  c("SL.bart", "screen.clinical"),
  c("SL.mean", "All"))
fitSL <- SuperLearner(Y = surv.resp, X = X,
SL.library = SL.library,
family = binomial(),
method = "method.NNLS",
control = list(saveFitLibrary = FALSE))

fitSL
Only presenting results for non-zero coefficients. Table does not fit on a slide.

<table>
<thead>
<tr>
<th>Model</th>
<th>Risk</th>
<th>Coef</th>
</tr>
</thead>
<tbody>
<tr>
<td>SL.knn_screen.corP.01</td>
<td>0.2129897</td>
<td>0.27517459</td>
</tr>
<tr>
<td>SL.glmnet_screen.corRank.20</td>
<td>0.2210815</td>
<td>0.22256164</td>
</tr>
<tr>
<td>SL.randomForest_clinical</td>
<td>0.2151708</td>
<td>0.08636558</td>
</tr>
<tr>
<td>SL.bart_clinical</td>
<td>0.2084039</td>
<td>0.41589818</td>
</tr>
</tbody>
</table>
```
fitSL.CV <- CV.SuperLearner(Y=surv.resp, X=X, 
V = 20, 
SL.library = SL.library, 
family = binomial(), 
method = "method.NNLS", 
cvControl = list(stratifyCV = TRUE))

summary(fitSL.CV)
```
<table>
<thead>
<tr>
<th>Algorithm</th>
<th>subset</th>
<th>Risk</th>
<th>SE</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>SuperLearner</td>
<td>–</td>
<td>0.194</td>
<td>0.017</td>
<td>0.103</td>
<td>0.309</td>
</tr>
<tr>
<td>Discrete SL</td>
<td>–</td>
<td>0.238</td>
<td>0.024</td>
<td>0.127</td>
<td>0.415</td>
</tr>
<tr>
<td>SL.knn(10)</td>
<td>All</td>
<td>0.249</td>
<td>0.020</td>
<td>0.144</td>
<td>0.532</td>
</tr>
<tr>
<td>SL.knn(10)</td>
<td>clinical</td>
<td>0.239</td>
<td>0.019</td>
<td>0.138</td>
<td>0.496</td>
</tr>
<tr>
<td>SL.knn(10)</td>
<td>cor (p &lt; 0.1)</td>
<td>0.262</td>
<td>0.023</td>
<td>0.095</td>
<td>0.443</td>
</tr>
<tr>
<td>SL.knn(10)</td>
<td>cor (p &lt; 0.01)</td>
<td>0.224</td>
<td>0.020</td>
<td>0.088</td>
<td>0.365</td>
</tr>
<tr>
<td>SL.knn(10)</td>
<td>glmnet</td>
<td>0.219</td>
<td>0.028</td>
<td>0.007</td>
<td>0.465</td>
</tr>
<tr>
<td>SL.knn(20)</td>
<td>All</td>
<td>0.242</td>
<td>0.013</td>
<td>0.171</td>
<td>0.397</td>
</tr>
<tr>
<td>SL.knn(20)</td>
<td>clinical</td>
<td>0.236</td>
<td>0.012</td>
<td>0.154</td>
<td>0.382</td>
</tr>
<tr>
<td>SL.knn(20)</td>
<td>cor (p &lt; 0.1)</td>
<td>0.233</td>
<td>0.017</td>
<td>0.108</td>
<td>0.342</td>
</tr>
<tr>
<td>SL.knn(20)</td>
<td>cor (p &lt; 0.01)</td>
<td>0.206</td>
<td>0.018</td>
<td>0.121</td>
<td>0.321</td>
</tr>
<tr>
<td>SL.knn(20)</td>
<td>glmnet</td>
<td>0.217</td>
<td>0.026</td>
<td>0.018</td>
<td>0.405</td>
</tr>
<tr>
<td>SL.knn(30)</td>
<td>All</td>
<td>0.239</td>
<td>0.013</td>
<td>0.171</td>
<td>0.396</td>
</tr>
<tr>
<td>SL.knn(30)</td>
<td>clinical</td>
<td>0.236</td>
<td>0.012</td>
<td>0.169</td>
<td>0.386</td>
</tr>
<tr>
<td>SL.knn(30)</td>
<td>cor (p &lt; 0.1)</td>
<td>0.232</td>
<td>0.014</td>
<td>0.143</td>
<td>0.319</td>
</tr>
<tr>
<td>SL.knn(30)</td>
<td>cor (p &lt; 0.01)</td>
<td>0.215</td>
<td>0.017</td>
<td>0.136</td>
<td>0.346</td>
</tr>
<tr>
<td>SL.knn(30)</td>
<td>glmnet</td>
<td>0.210</td>
<td>0.023</td>
<td>0.039</td>
<td>0.402</td>
</tr>
<tr>
<td>SL.knn(40)</td>
<td>All</td>
<td>0.240</td>
<td>0.011</td>
<td>0.182</td>
<td>0.331</td>
</tr>
<tr>
<td>SL.knn(40)</td>
<td>clinical</td>
<td>0.238</td>
<td>0.010</td>
<td>0.179</td>
<td>0.319</td>
</tr>
<tr>
<td>SL.knn(40)</td>
<td>cor (p &lt; 0.1)</td>
<td>0.236</td>
<td>0.012</td>
<td>0.166</td>
<td>0.316</td>
</tr>
<tr>
<td>SL.knn(40)</td>
<td>cor (p &lt; 0.01)</td>
<td>0.219</td>
<td>0.015</td>
<td>0.154</td>
<td>0.309</td>
</tr>
<tr>
<td>SL.knn(40)</td>
<td>glmnet</td>
<td>0.211</td>
<td>0.021</td>
<td>0.060</td>
<td>0.346</td>
</tr>
<tr>
<td>SL.glmnet(α = 1.0)</td>
<td>corRank.50</td>
<td>0.229</td>
<td>0.029</td>
<td>0.078</td>
<td>0.445</td>
</tr>
<tr>
<td>SL.glmnet(α = 1.0)</td>
<td>corRank.20</td>
<td>0.208</td>
<td>0.026</td>
<td>0.048</td>
<td>0.424</td>
</tr>
<tr>
<td>SL.glmnet(α = 0.75)</td>
<td>corRank.50</td>
<td>0.221</td>
<td>0.027</td>
<td>0.077</td>
<td>0.420</td>
</tr>
<tr>
<td>SL.glmnet(α = 0.75)</td>
<td>corRank.20</td>
<td>0.209</td>
<td>0.026</td>
<td>0.046</td>
<td>0.421</td>
</tr>
<tr>
<td>SL.glmnet(α = 0.50)</td>
<td>corRank.50</td>
<td>0.226</td>
<td>0.027</td>
<td>0.077</td>
<td>0.426</td>
</tr>
<tr>
<td>SL.glmnet(α = 0.50)</td>
<td>corRank.20</td>
<td>0.211</td>
<td>0.026</td>
<td>0.059</td>
<td>0.419</td>
</tr>
<tr>
<td>SL.glmnet(α = 0.25)</td>
<td>corRank.50</td>
<td>0.229</td>
<td>0.027</td>
<td>0.084</td>
<td>0.424</td>
</tr>
<tr>
<td>SL.glmnet(α = 0.25)</td>
<td>corRank.20</td>
<td>0.216</td>
<td>0.025</td>
<td>0.072</td>
<td>0.406</td>
</tr>
<tr>
<td>SL.randomForest</td>
<td>clinical</td>
<td>0.198</td>
<td>0.019</td>
<td>0.098</td>
<td>0.391</td>
</tr>
<tr>
<td>SL.randomForest</td>
<td>cor (p &lt; 0.01)</td>
<td>0.204</td>
<td>0.018</td>
<td>0.101</td>
<td>0.341</td>
</tr>
<tr>
<td>SL.randomForest</td>
<td>glmnet</td>
<td>0.220</td>
<td>0.025</td>
<td>0.072</td>
<td>0.378</td>
</tr>
<tr>
<td>SL.bagging</td>
<td>clinical</td>
<td>0.207</td>
<td>0.016</td>
<td>0.108</td>
<td>0.408</td>
</tr>
<tr>
<td>SL.bagging</td>
<td>cor (p &lt; 0.01)</td>
<td>0.205</td>
<td>0.018</td>
<td>0.107</td>
<td>0.353</td>
</tr>
<tr>
<td>SL.bagging</td>
<td>glmnet</td>
<td>0.206</td>
<td>0.022</td>
<td>0.077</td>
<td>0.388</td>
</tr>
<tr>
<td>SL.bart</td>
<td>clinical</td>
<td>0.202</td>
<td>0.018</td>
<td>0.109</td>
<td>0.365</td>
</tr>
<tr>
<td>SL.bart</td>
<td>cor (p &lt; 0.01)</td>
<td>0.210</td>
<td>0.021</td>
<td>0.092</td>
<td>0.376</td>
</tr>
<tr>
<td>SL.bart</td>
<td>glmnet</td>
<td>0.220</td>
<td>0.028</td>
<td>0.043</td>
<td>0.423</td>
</tr>
<tr>
<td>SL.mean</td>
<td>All</td>
<td>0.250</td>
<td>0.003</td>
<td>0.246</td>
<td>0.251</td>
</tr>
</tbody>
</table>
Installing suggested packages

```r
install.packages(c("glmnet","randomForest","class","gam","gbm","nnet","polspline","MASS","e1071","stepPlr","arm","party","spls","LogicReg","nnls","multicore","SIS","BayesTree","quadprog","ipred","mlbench","rpart","caret","mda","earth"),
type="source",
repos="http://cran.cnr.Berkeley.edu",
dependencies=c("Depends","Imports"))
```

# missing DSA, not available on CRAN

Can remove type = ‘source’ if system not setup to install packages from source.
Colophon

- Slides created with \LaTeX{} package Beamer
- Code blocks adapted from the tikzDevice R package
- \LaTeX{} package tikz and sweave for code styles
- R version 2.13.0 and SuperLearner version 2.0-1
- Other packages: arm 1.4-10, BayesTree 0.3-1, caret 4.88, class 7.3-3, DSA 3.1.4, e1071 1.5-25, earth 2.6-2, gam 1.04, gbm 1.6-3, glmnet 1.6, Hmisc 3.8-3, ipred 0.8-11, leaps 2.9, lme4 0.999375-39, LogicReg 1.4.10, MASS 7.3-13, mda 0.4-2, mlbench 2.1-0, modelUtils 3.1.4, nnet 7.3-1, nnls 1.3, party 0.9-9994, polsplines 1.1.5, quadprog 1.5-4, randomForest 4.6-2, rpart 3.1-50, SIS 0.6,