Introduction to the SuperLearner R package

Eric Polley

polley.eric@mayo.edu

March 2, 2017

Super Learner

The super learner methodology is an example of ensemble learning where the individual base learners are a diverse set of algorithms and cross-validation is used to select the optimal ensemble of predictors. An example is a convex combination of a diverse collection of predictors:

$$f_{SL}(X) = \alpha_1 f_1(X) + \alpha_2 f_2(X) + \ldots + \alpha_p f_p(X) \tag{1}$$

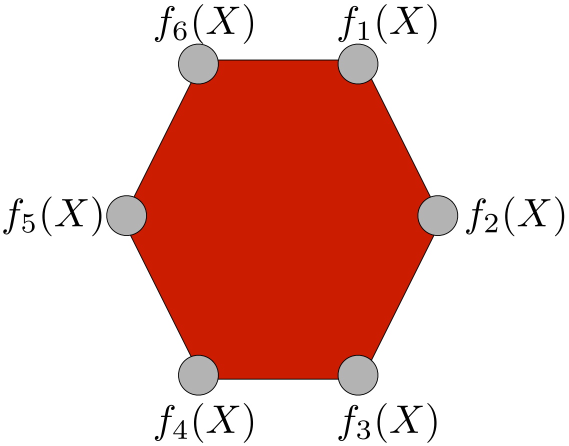
with $\alpha \geq 0$ and $\sum \alpha = 1$.

The method has a long history

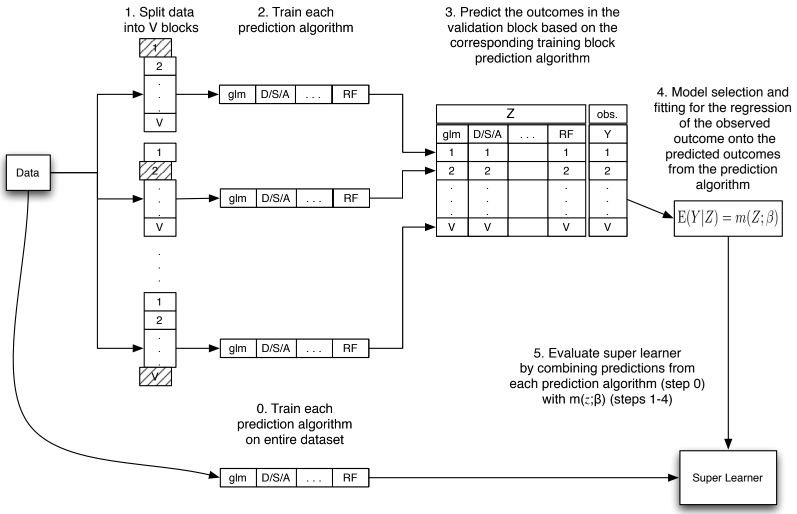
- "model-mix" (Stone, 1974)
- "predictive sample reuse" (Geisser, 1975)
- "stacked generalization" (Wolpert, 1992)
- "stacked regression" (Breiman, 1996)
- "super learner" (van der Laan and Polley, 2007)

Super Learner

Geometrically, this SL ensemble solution is within the simplex, and the vertices are the usual cross-validation selector



Super Learner



1. Split data into V blocks
2. Train each prediction algorithm
3. Predict the outcomes in the validation block based on the corresponding training block prediction algorithm
4. Model selection and fitting for the regression of the observed outcome onto the predicted outcomes from the prediction algorithm
5. Evaluate super learner by combining predictions from each prediction algorithm (step 0) with m(z;β) (steps 1-4)
0. Train each prediction algorithm on entire dataset
Data
1
2
V
glm
D/S/A
. . .
RF
Z
obs.
Y
$E(Y|Z) = m(Z;\beta)$
Super Learner

Figure 2: Super learner diagram

R package

- **SuperLearner** First publicly released in 2010
- Available on CRAN[1] and Github[2]
- Includes over 30 different prediction algorithms, plus framework to modify existing algorithms or add your own
- Maintain GitHub page **SuperLearnerExtra** as place to share additional add-ons
- Other implementations include **H2oEnsemble**[3] and **caretEnsemble**[4]

[1] `https://cran.r-project.org/package=SuperLearner`

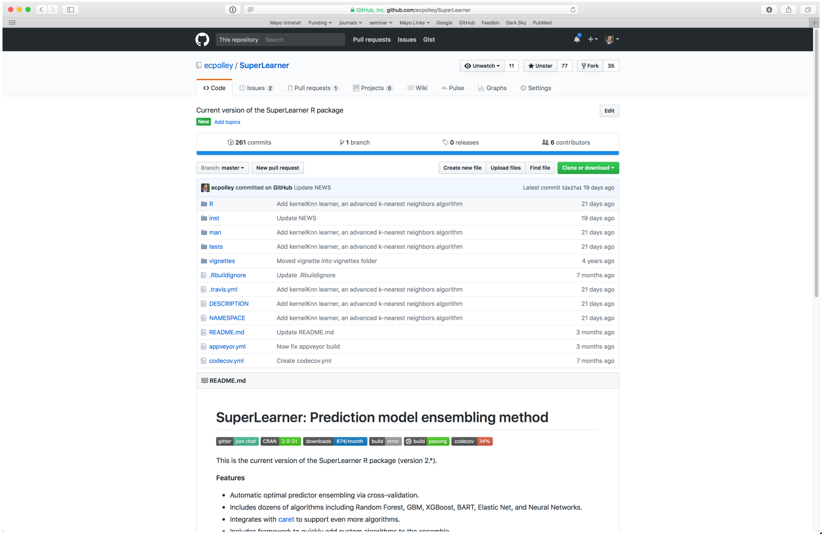
[2] `https://github.com/ecpolley/SuperLearner`

[3] `https://github.com/h2oai/h2o-3/tree/master/h2o-r/ensemble`

[4] `https://cran.r-project.org/web/packages/caretEnsemble/`

MAYO CLINIC

GitHub



Current version of the SuperLearner R package

ecpolley committed on GitHub Update NEWS — Latest commit 52e27a1 19 days ago

R	Add kernelKnn learner, an advanced k-nearest neighbors algorithm	21 days ago
inst	Update NEWS	19 days ago
man	Add kernelKnn learner, an advanced k-nearest neighbors algorithm	21 days ago
tests	Add kernelKnn learner, an advanced k-nearest neighbors algorithm	21 days ago
vignettes	Moved vignette into vignettes folder	4 years ago
.Rbuildignore	Update .Rbuildignore	7 months ago
.travis.yml	Add kernelKnn learner, an advanced k-nearest neighbors algorithm	21 days ago
DESCRIPTION	Add kernelKnn learner, an advanced k-nearest neighbors algorithm	21 days ago
NAMESPACE	Add kernelKnn learner, an advanced k-nearest neighbors algorithm	21 days ago
README.md	Update README.md	3 months ago
appveyor.yml	Now fix appveyor build	3 months ago
codecov.yml	Create codecov.yml	7 months ago

SuperLearner: Prediction model ensembling method

This is the current version of the SuperLearner R package (version 2.*).

Features

- Automatic optimal predictor ensembling via cross-validation.
- Includes dozens of algorithms including Random Forest, GBM, XGBoost, BART, Elastic Net, and Neural Networks.
- Integrates with caret to support even more algorithms.
- Includes framework to quickly add custom algorithms to the ensemble.

Figure 3: GitHub

Example

Using NCI60 cancer cell line drug sensitivity data and exome sequencing data[5]

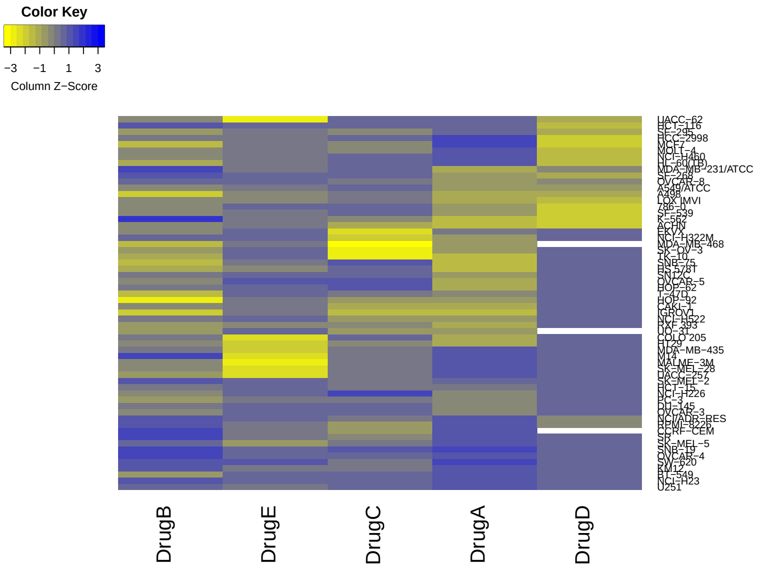
```r
# DrugGI50
load("Data/DrugGI50.RData")
# VariantTable: 10,746 genes
load("Data/VariantTableByGene.RData")
```

Two data files:

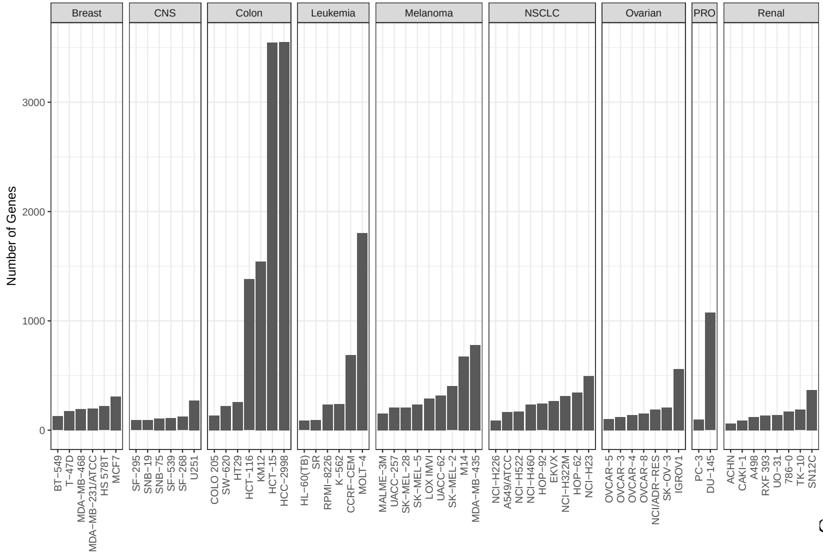
1. Sensitivity for 5 drugs
2. Genes with likely somatic mutations from exome sequencing

[5]Trivia question, how many cell lines are in the NCI-60 dataset?

Heatmap of Drug Sensitivity



Number of Variants per Cell Line



Check Gene Symbols

```r
VariantTable5 <- VariantTable[, which(colSums(VariantTable) > 4)]
GeneSymbols <- colnames(VariantTable5)
GeneSymbolsClean <- make.names(GeneSymbols, unique = TRUE)
setdiff(GeneSymbols, GeneSymbolsClean)
```

```
## [1] "HLA-DRB5"      "KRTAP5-1"      "STON1-GTF2A1L"
```

```r
VariantTable5 <- data.frame(VariantTable5, check.names = TRUE)
```

Setup SuperLearner

```r
listWrappers(what = "SL")
```

```
## All prediction algorithm wrappers in SuperLearner:
```

```
##  [1] "SL.bartMachine"      "SL.bayesglm"       "SL.caret"
##  [4] "SL.caret.rpart"      "SL.cforest"        "SL.earth"
##  [7] "SL.gam"              "SL.gbm"            "SL.glm"
## [10] "SL.glm.interaction"  "SL.glmnet"         "SL.ipredbagg"
## [13] "SL.knn"              "SL.leekasso"       "SL.loess"
## [16] "SL.logreg"           "SL.mean"           "SL.nnet"
## [19] "SL.nnls"             "SL.polymars"       "SL.randomForest"
## [22] "SL.ridge"            "SL.rpart"          "SL.rpartPrune"
## [25] "SL.step"             "SL.stepAIC"        "SL.step.forward"
## [28] "SL.step.interaction" "SL.svm"            "SL.template"
## [31] "SL.xgboost"
```

Examine SL Wrapper

```r
SL.glm
```

```
## function (Y, X, newX, family, obsWeights, ...)
## {
##     fit.glm <- glm(Y ~ ., data = X, family = family, weights = obsWe
##     pred <- predict(fit.glm, newdata = newX, type = "response")
##     fit <- list(object = fit.glm)
##     class(fit) <- "SL.glm"
##     out <- list(pred = pred, fit = fit)
##     return(out)
## }
## <environment: namespace:SuperLearner>
```

Specify SuperLearner library

The first method to specific a library of algorithms is as a character vector with the algorithm names

```r
SL_lib <- c("SL.glmnet",
            "SL.randomForest",
            "SL.leekasso",
            "SL.rpartPrune",
            "SL.mean")
```

## Call SuperLearner

```r
fit <- SuperLearner(Y = DrugGI50$DrugE,
                    X = VariantTable5,
                    SL.library = SL_lib,
                    family = gaussian(),
                    method = "method.NNLS",
                    cvControl = list(V = 10))
fit
```

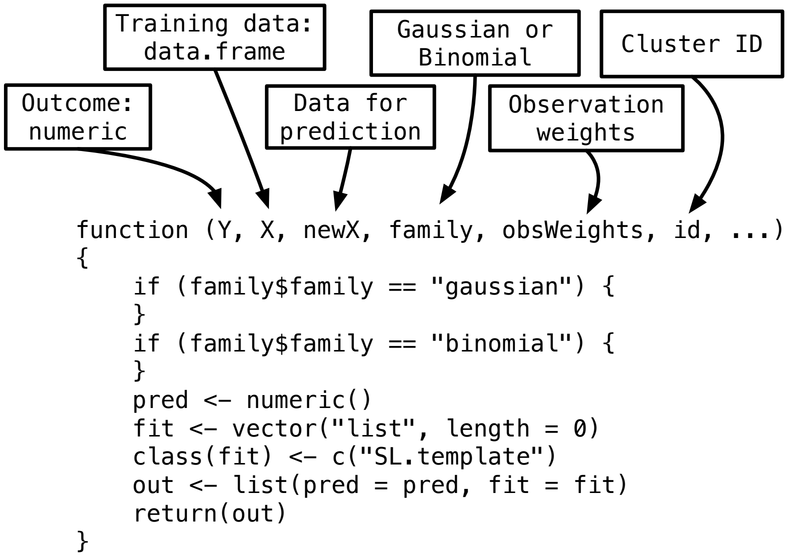
```
## Call:
## SuperLearner(Y = DrugGI50$DrugE, X = VariantTable5, family = gaussia
##     SL.library = SL_lib, method = "method.NNLS", cvControl = list(V
##
##                          Risk       Coef
## SL.glmnet_All       0.1703059 0.14107618
## SL.randomForest_All 0.2879813 0.00000000
## SL.leekasso_All     0.3005714 0.03777997
## SL.rpartPrune_All   0.1497285 0.82114385
## SL.mean_All         0.5008175 0.00000000
```

Templates for New SL Wrappers

```r
write.SL.template()
```

```
## 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)
## }
```

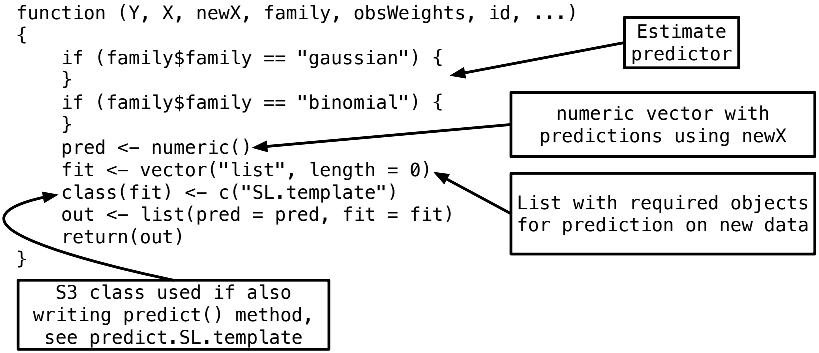
New SL Wrappers



```
function (Y, X, newX, family, obsWeights, id, ...)
{
    if (family$family == "gaussian") {
    }
    if (family$family == "binomial") {
    }
    pred <- numeric()
    fit <- vector("list", length = 0)
    class(fit) <- c("SL.template")
    out <- list(pred = pred, fit = fit)
    return(out)
}
```

Figure 4: SL Wrapper Inputs

New SL Wrappers



```
function (Y, X, newX, family, obsWeights, id, ...)
{
    if (family$family == "gaussian") {
    }
    if (family$family == "binomial") {
    }
    pred <- numeric()
    fit <- vector("list", length = 0)
    class(fit) <- c("SL.template")
    out <- list(pred = pred, fit = fit)
    return(out)
}
```

Estimate predictor
numeric vector with predictions using newX
List with required objects for prediction on new data
S3 class used if also writing predict() method, see predict.SL.template

Figure 5: SL Wrapper Code and Output

variable screening with SuperLearner

```r
listWrappers(what = "screen")
```

```
## All screening algorithm wrappers in SuperLearner:
```

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

MAYO CLINIC

Specify SuperLearner library with screening

The second method to specific a library of algorithms is as a list with character vectors. The first string within a list element is the prediction algorithm, all other elements are screening functions to define data subsets for the corresponding algorithm

```r
SL_lib2 <- list(c("SL.glmnet", "All", "screen.corRank"),
                c("SL.randomForest", "screen.corRank"),
                "SL.leekasso",
                "SL.mean")
```

Cross Validation of Super Learner

```r
cv_fit <- CV.SuperLearner(Y = DrugGI50$DrugE,
                          X = VariantTable5,
                          SL.library = SL_lib,
                          family = gaussian(),
                          method = "method.NNLS",
                          cvControl = list(V = 20))
```

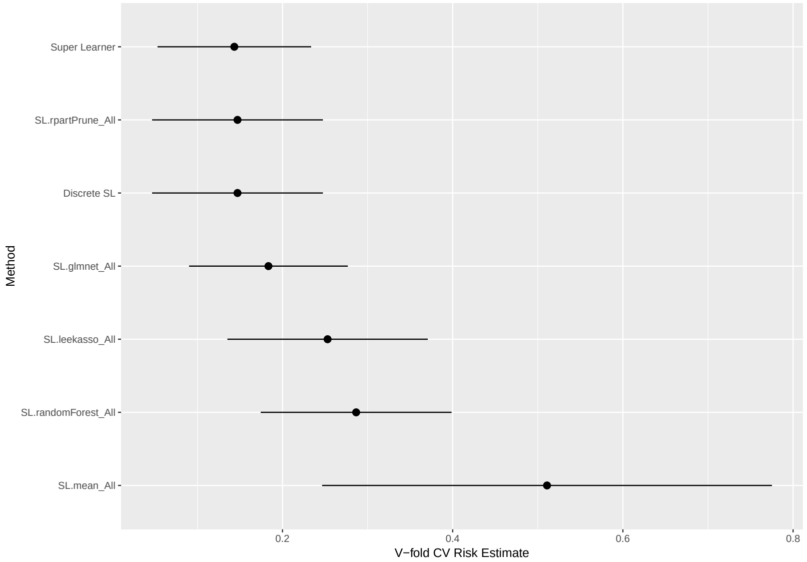
```r
summary(cv_fit)
```

```
## All risk estimates are based on V =  20
##              Algorithm     Ave       se       Min     Max
##          Super Learner 0.14338 0.046061 0.0030223 0.80044
##            Discrete SL 0.14712 0.051217 0.0038117 0.90264
##          SL.glmnet_All 0.18347 0.047599 0.0117963 0.67884
##    SL.randomForest_All 0.28655 0.057216 0.0326594 0.81517
##        SL.leekasso_All 0.25298 0.060073 0.0063578 0.74982
##      SL.rpartPrune_All 0.14712 0.051217 0.0038117 0.90264
##            SL.mean_All 0.51085 0.134847 0.0337425 2.04343
```

MAYO CLINIC

Cross validation of Super Learner

```r
plot(cv_fit)
```



Other topics

- Add your own loss functions and metalearners, see `method.template`
- Parallel computation with `mcSuperLearner` or `snowSuperLearner`. Parallelizes the V-fold cross validation step, so well balanced, but limited by number of folds
- Write your own screening functions
- Control parameters for the cross validation splits, `cvControl`
- Built in functions for creating new wrappers, `create.Learner`