Super Learner

(References: Le Dell (2015) Section 2.2 and van der Laan et al (2016), and Polley et al (2011))

The Super Learner or *generalized stacking* is an algorithm that combines

- multiple, (typically) diverse prediction methods (learning algorithms) called *base learners* into a
- a *metalearner* - which can be seen as a *single* method.

Algorithm

Step 1: Produce level-one data $\mathbf{Z}$

a) Divide the training data $\mathbf{X}$ randomly into V roughly-equally sized validation folds $\mathbf{X}_{(1)}, \ldots, \mathbf{X}_{(V)}$. V is often 5 or 10.

b) For each base learner Ψ^l perform V-fold cross-validation to produce prediction.

This gives the level-one data set $\mathbf{Z}$ consisting prediction of all the level-zero data - that is a matrix with N rows and L columns.

What could the base learners be?

"Any" method that produces a prediction - "all" types of problems.

- linear regression
- lasso
- cart
- random forest with mtry=value 1
- random forest with mtry=value 2
- xgboost with hyperparameter set 1
- xgboost with hyperparameter set 2
- neural net with hyperparameter set 1

Step 2: Fit the metalearner

a) The starting point is the level-one prediction data $\mathbf{Z}$ together with the responses $(Y_1, \ldots, Y_N)$.
b) The metalearner is used to estimate the weights given to each base learner: $\hat{Y}_i = \alpha_1 z_{1i} + \cdots + \alpha_L z_{Li}$. (Should probably also involve some link function, so that this may be the linear predictor.)

What could the metalearner be?

- the mean (bagging)
- ordinary least squares
- non-negative least squares
- ridge or lasso regression
- 1-ROC-AUC

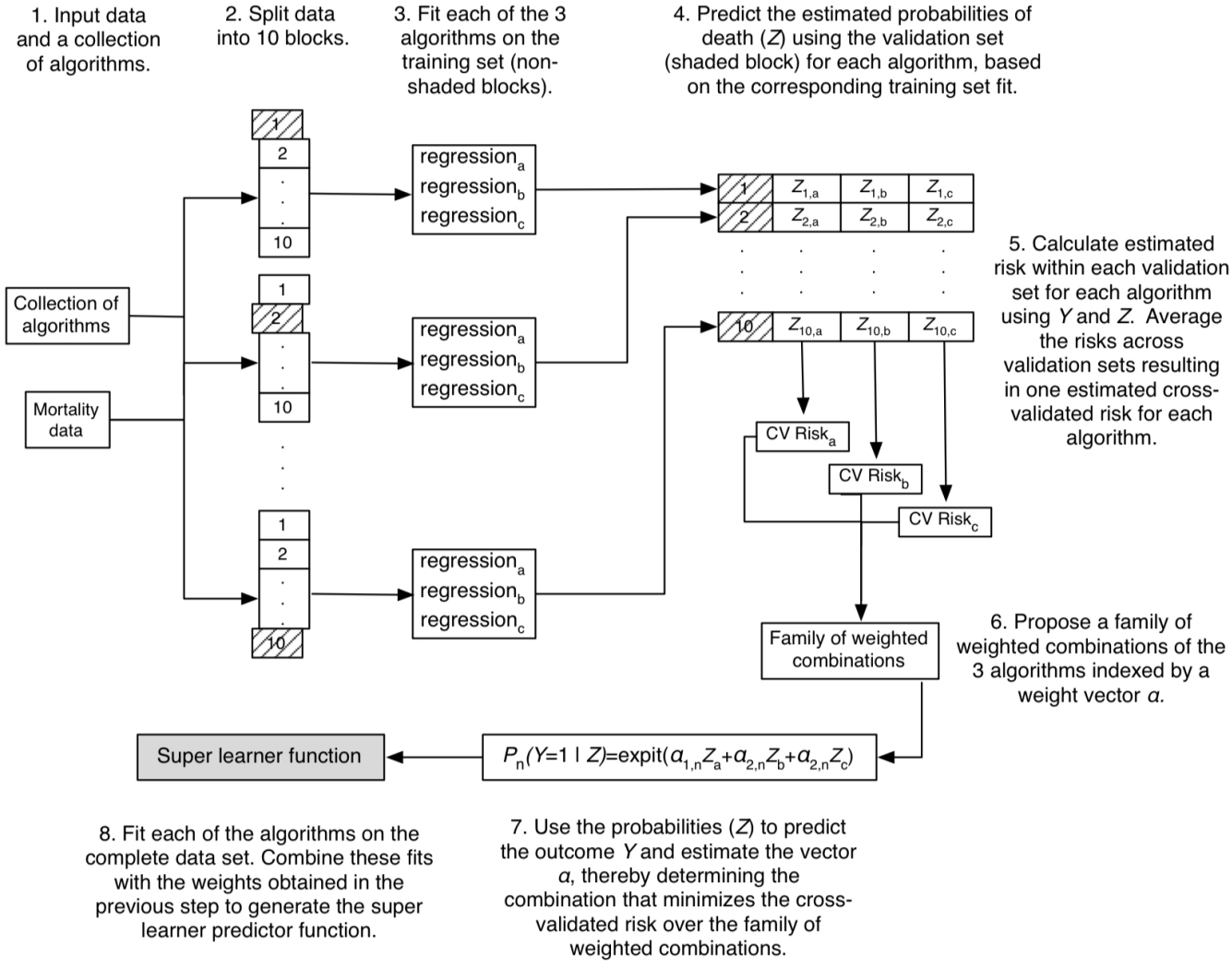


Fig. 3.2 Super learner algorithm for the mortality study example

Step 3: Re-estimate base learners and combine into superlearner on full training data

a) Fit each of the L base learners to the full training set.
b) The *ensemble fit* consists the L base learner fits together with the metalearner fit.

Step 4: Using the ensemble for prediction

For a new observaton $\mathbf{x}^*$

a) Use each of the L base learners to produce a prediction $\mathbf{z}^*$, and
b) feed this to the metalearnerfit to produce the final prediction y^*.

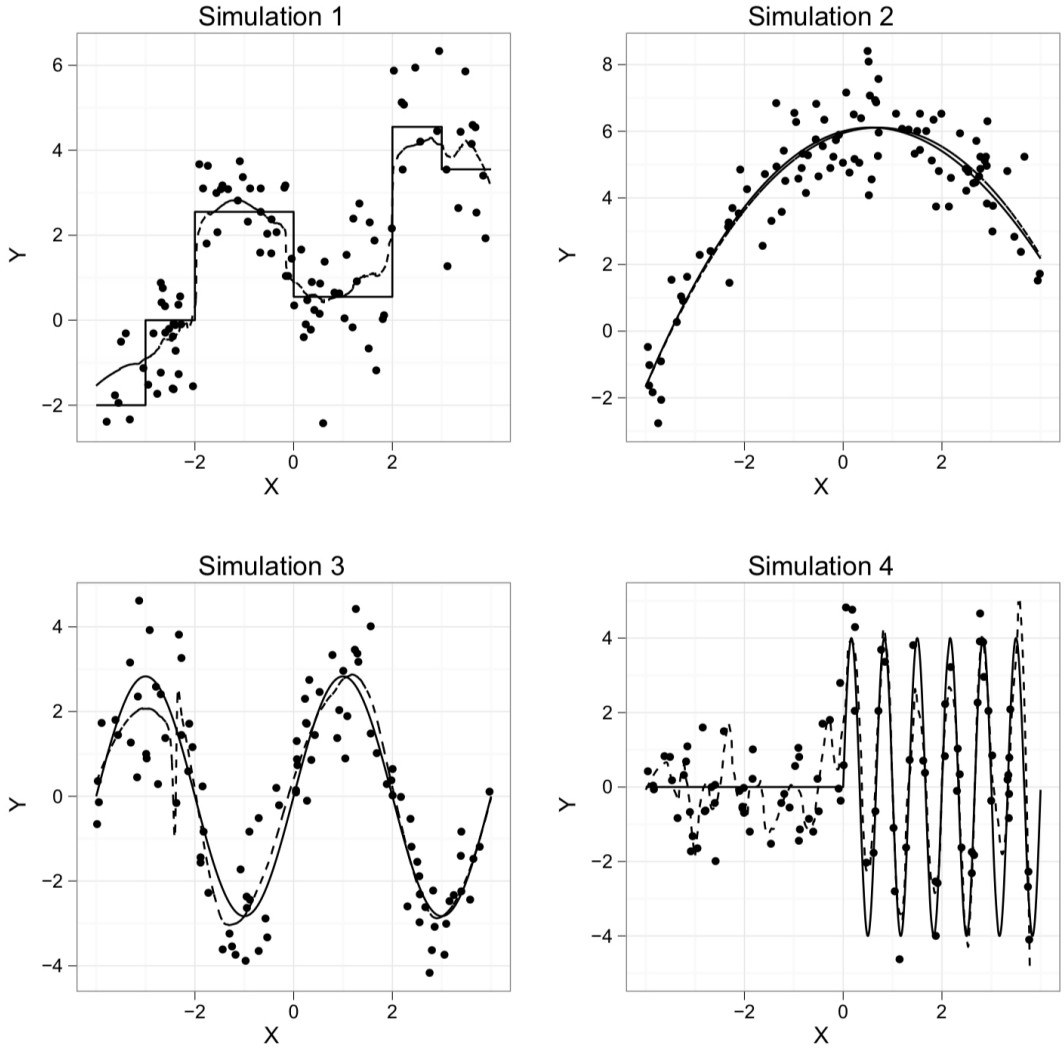
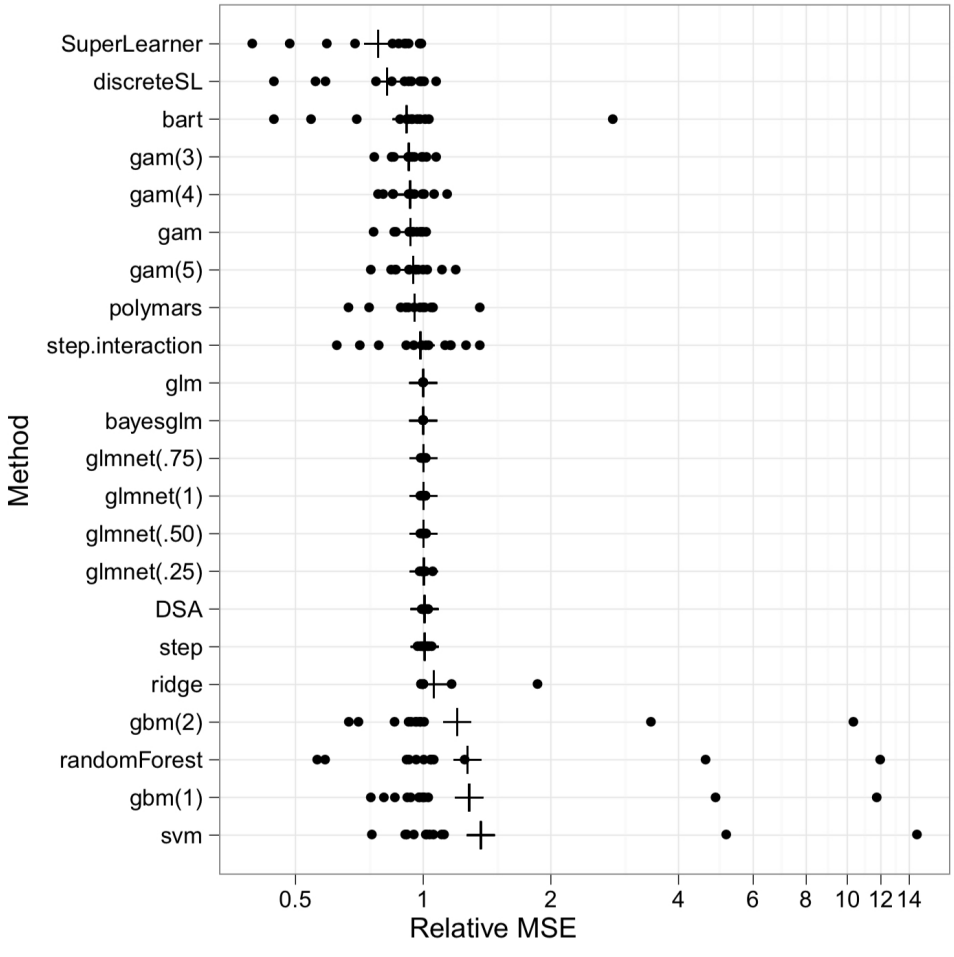


Fig. 3.3 Scatterplots of the four simulations. The *solid line* is the true relationship. The *points* represent one of the simulated data sets of size $n = 100$. The *dashed line* is the super learner fit for the shown data set

Table 3.2 Results for four simulations. Average R^2 based on 100 simulations and the corresponding standard errors

Algorithm	**Sim 1**		**Sim 2**		**Sim 3**		**Sim 4**	
	R^2	SE(R^2)	R^2	SE(R^2)	R^2	SE(R^2)	R^2	SE(R^2)
Super learner	0.741	0.032	0.754	0.025	0.760	0.025	0.496	0.122
Discrete SL	0.729	0.079	0.758	0.029	0.757	0.055	0.509	0.132
SL.glm	0.422	0.012	0.189	0.016	0.107	0.016	−0.018	0.021
SL.interaction	0.428	0.016	0.769	0.011	0.100	0.020	−0.018	0.029
SL.randomForest	0.715	0.021	0.702	0.027	0.724	0.018	0.460	0.109
SL.bagging(0.01)	0.751	0.022	0.722	0.036	0.723	0.018	0.091	0.054
SL.bagging(0.1)	0.635	0.120	0.455	0.195	0.661	0.029	0.020	0.025
SL.bagging(0.0)	0.752	0.021	0.722	0.034	0.727	0.017	0.102	0.060
SL.bagging(ms5)	0.747	0.020	0.727	0.030	0.741	0.016	0.369	0.104
SL.gam(2)	0.489	0.013	0.649	0.026	0.213	0.029	−0.014	0.023
SL.gam(3)	0.535	0.033	0.748	0.024	0.412	0.037	−0.017	0.029
SL.gam(4)	0.586	0.027	0.759	0.020	0.555	0.022	−0.020	0.034
SL.gbm	0.717	0.035	0.694	0.038	0.679	0.022	0.063	0.040
SL.nnet(2)	0.476	0.235	0.591	0.245	0.283	0.285	−0.008	0.030
SL.nnet(3)	0.700	0.096	0.700	0.136	0.652	0.218	0.009	0.035
SL.nnet(4)	0.719	0.077	0.730	0.062	0.738	0.102	0.032	0.052
SL.nnet(5)	0.705	0.079	0.716	0.070	0.731	0.077	0.042	0.060
SL.polymars	0.704	0.033	0.733	0.032	0.745	0.034	0.003	0.040
SL.bart	0.740	0.015	0.737	0.027	0.764	0.014	0.077	0.034
SL.loess(0.75)	0.599	0.023	0.761	0.019	0.487	0.028	−0.023	0.033
SL.loess(0.50)	0.695	0.018	0.754	0.022	0.744	0.029	−0.033	0.038
SL.loess(0.25)	0.729	0.016	0.738	0.025	0.772	0.015	−0.076	0.068
SL.loess(0.1)	0.690	0.044	0.680	0.064	0.699	0.039	0.544	0.118

Fig. 3.4 Tenfold cross-validated relative mean squared error compared to glm across 13 real data sets. Sorted by geometric mean, denoted by the plus (+) sign



Name	n	p	Source
ais	202	10	Cook and Weisberg (1994)
diamond	308	17	Chu (2001)
cps78	550	18	Berndt (1991)
cps85	534	17	Berndt (1991)
cpu	209	6	Kibler et al. (1989)
FEV	654	4	Rosner (1999)
Pima	392	7	Newman et al. (1998)
laheart	200	10	Afifi and Azen (1979)
mussels	201	3	Cook (1998)
enroll	258	6	Liu and Stengos (1999)
fat	252	14	Penrose et al. (1985)
diabetes	366	15	Harrell (2001)
house	506	13	Newman et al. (1998)

Theoretical result

- Oracle selector: the estimator among all possible weighted combinations of the base prediction function that minimizes the risk under the *true data generating distribution*.
- The *oracle result* was established for the Super Learner by van der Laan et al (2006).
- If the *true prediction function* cannot be represented by a combination of the base learners (available), then "optimal" will be the closest linear combination that would be optimal if the true data-generating function was known.
- The oracle result require an *uniformly bounded loss function*. Using the convex restriction (sum alphas $=1$) implies that if each based learner is bounded so is the convex combination. In practice: truncation of the predicted values to the range of the outcome in the training set is sufficient to allow for unbounded loss fuctions (Le Dell page 6).

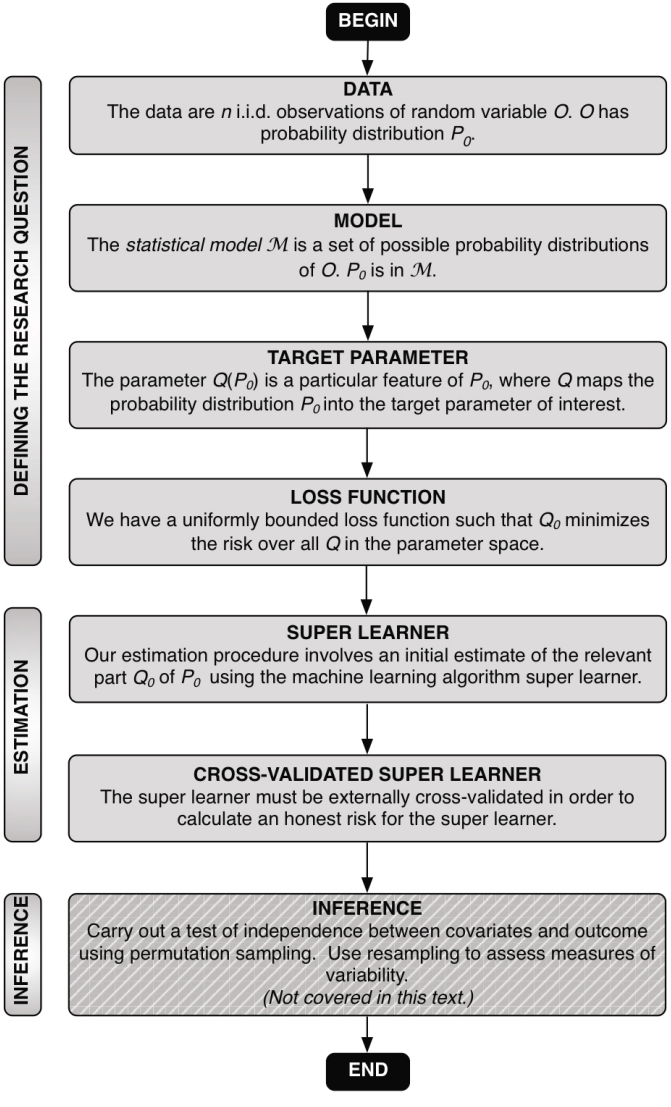


Fig. 3.6 Road map for prediction

Uncertainty in the ensemble

(Class notes: Study "Road map" 2 from Polley et al)

- Add an outer (external) cross validation loop (where the super learner loop is inside). Suggestion: use 20-fold, especially when small sample size.
- Overfitting? Check if the super learner does as well or better than any of the base learners in the ensemble.
- Results using *influence functions* for estimation of the variance for the Super Learner are based on asymptotic variances in the use of V-fold cross-validation (see Ch 5.3 of Le Dell, 2015)

Other issues

- Many different implementations available, and much work on parallell processing and speed and memory efficient execution.
- Super learner implicitly can handle hyperparameter tuning by including the same base learner with different model parameter sets in the ensemble.
- Speed and memory improvements for large data sets involves subsampling, and the R `subsemble` package is one solution, the H2O Ensemble project another.

R example

Code is copied from Guide to SuperLearner and the presentation follows this guide. The data used is the Boston housing dataset from `MASS`, but with the median value of a house dichotomized into a classification problem.

Observe that only 150 of the 560 observations is used (to speed up things, but of cause that gives less accurate results).

```
data(Boston, package = "MASS")
#colSums(is.na(Boston)) # no missing values
outcome = Boston$medv
# Create a dataframe to contain our explanatory variables.
data = subset(Boston, select = -medv)
#Set a seed for reproducibility in this random sampling.
set.seed(1)
# Reduce to a dataset of 150 observations to speed up model fitting.
train_obs = sample(nrow(data), 150)
# X is our training sample.
x_train = data[train_obs, ]
# Create a holdout set for evaluating model performance.
# Note: cross-validation is even better than a single holdout sample.
x_holdout = data[-train_obs, ]
```

```
# Create a binary outcome variable: towns in which median home value is > 22,000.
outcome_bin = as.numeric(outcome > 22)
y_train = outcome_bin[train_obs]
y_holdout = outcome_bin[-train_obs]
table(y_train, useNA = "ifany")
```

```
## y_train
##  0  1
## 92 58
```

Then checking out the possible functions and how they differ from their "original versions".

```
listWrappers()
```

```
##  [1] "SL.bartMachine"      "SL.bayesglm"         "SL.biglasso"
##  [4] "SL.caret"            "SL.caret.rpart"      "SL.cforest"
##  [7] "SL.earth"            "SL.extraTrees"       "SL.gam"
## [10] "SL.gbm"              "SL.glm"              "SL.glm.interaction"
## [13] "SL.glmnet"           "SL.ipredbagg"        "SL.kernelKnn"
## [16] "SL.knn"              "SL.ksvm"             "SL.lda"
## [19] "SL.leekasso"         "SL.lm"               "SL.loess"
## [22] "SL.logreg"           "SL.mean"             "SL.nnet"
## [25] "SL.nnls"             "SL.polymars"         "SL.qda"
## [28] "SL.randomForest"     "SL.ranger"           "SL.ridge"
## [31] "SL.rpart"            "SL.rpartPrune"       "SL.speedglm"
## [34] "SL.speedlm"          "SL.step"             "SL.step.forward"
## [37] "SL.step.interaction" "SL.stepAIC"          "SL.svm"
## [40] "SL.template"         "SL.xgboost"
## [1] "All"
## [1] "screen.corP"           "screen.corRank"        "screen.glmnet"
## [4] "screen.randomForest"   "screen.SIS"            "screen.template"
## [7] "screen.ttest"          "write.screen.template"
```

```
# how does SL.glm differ from glm? obsWeight added to easy use the traning fold in the CV and returns a
SL.glm
```

```
## function (Y, X, newX, family, obsWeights, model = TRUE, ...)
## {
##     if (is.matrix(X)) {
##         X = as.data.frame(X)
##     }
##     fit.glm <- glm(Y ~ ., data = X, family = family, weights = obsWeights,
##         model = model)
##     if (is.matrix(newX)) {
##         newX = as.data.frame(newX)
##     }
##     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)
## }
## <bytecode: 0x7fd5a90f6ac0>
## <environment: namespace:SuperLearner>
```

```
# min and not 1sd used, again obsWeights, make sure model matrix correctly specified
SL.glmnet
```

```
## function (Y, X, newX, family, obsWeights, id, alpha = 1, nfolds = 10,
##     nlambda = 100, useMin = TRUE, loss = "deviance", ...)
## {
##     .SL.require("glmnet")
##     if (!is.matrix(X)) {
##         X <- model.matrix(~-1 + ., X)
##         newX <- model.matrix(~-1 + ., newX)
##     }
##     fitCV <- glmnet::cv.glmnet(x = X, y = Y, weights = obsWeights,
##         lambda = NULL, type.measure = loss, nfolds = nfolds,
##         family = family$family, alpha = alpha, nlambda = nlambda,
##         ...)
##     pred <- predict(fitCV, newx = newX, type = "response", s = ifelse(useMin,
##         "lambda.min", "lambda.1se"))
##     fit <- list(object = fitCV, useMin = useMin)
##     class(fit) <- "SL.glmnet"
##     out <- list(pred = pred, fit = fit)
##     return(out)
## }
## <bytecode: 0x7fd5a91588c8>
## <environment: namespace:SuperLearner>
```

The fitting lasso to check what is being done. The default metalearner is "method.NNLS" (both for regression and two-class classification - probably then for linear predictor NNLS?).

```
set.seed(1)
sl_lasso=SuperLearner(Y=y_train, X=x_train,family=binomial(),SL.library="SL.glmnet")
sl_lasso
```

```
##
## Call:
## SuperLearner(Y = y_train, X = x_train, family = binomial(), SL.library = "SL.glmnet")
##
##
##
##                      Risk Coef
## SL.glmnet_All 0.08484849    1
```

```
#str(sl_lasso)
sl_lasso$cvRisk
```

```
## SL.glmnet_All
##    0.08484849
```

Now use lasso and randomforest, and also add the average of ys just as the benchmark.

```
set.seed(1)
sl=SuperLearner(Y=y_train, X=x_train,family=binomial(),SL.library=c("SL.mean","SL.glmnet","SL.randomFor
sl
```

```
##
## Call:
## SuperLearner(Y = y_train, X = x_train, family = binomial(), SL.library = c("SL.mean",
##     "SL.glmnet", "SL.randomForest"))
```

```
##
##
##                            Risk     Coef
## SL.mean_All          0.23773937 0.000000
## SL.glmnet_All        0.08725786 0.134252
## SL.randomForest_All 0.07213058 0.865748
```

```
sl$times$everything
```

```
##    user  system elapsed
##   3.160   0.070   3.246
```

Our ensemble give weight 0.13 to lasso and 0.86 to the random forest. (The guide used a different implementation of the random forest called ranger, and got 0.02 and 0.98.)

Predict on the part of the dataset not used for the training.

```
pred=predict(sl,x_holdout=x_holdout,onlySL=TRUE)
str(pred)
```

```
## List of 2
##  $ pred           : num [1:150, 1] 0.3029 0.07 0.97847 0.00726 0.00523 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:150] "505" "324" "167" "129" ...
##   .. ..$ : NULL
##  $ library.predict: num [1:150, 1:3] 0.387 0.387 0.387 0.387 0.387 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:150] "505" "324" "167" "129" ...
##   .. ..$ : chr [1:3] "SL.mean_All" "SL.glmnet_All" "SL.randomForest_All"
```

```
summary(pred$pred)
```

```
##        V1
##  Min.   :0.0003034
##  1st Qu.:0.0183955
##  Median :0.1135270
##  Mean   :0.3855066
##  3rd Qu.:0.9036164
##  Max.   :0.9955802
```

```
summary(pred$library.predict)
```

```
##   SL.mean_All     SL.glmnet_All       SL.randomForest_All
##  Min.   :0.3867   Min.   :0.0000014   Min.   :0.0000
##  1st Qu.:0.3867   1st Qu.:0.0244935   1st Qu.:0.0160
##  Median :0.3867   Median :0.2063204   Median :0.1020
##  Mean   :0.3867   Mean   :0.3866667   Mean   :0.3853
##  3rd Qu.:0.3867   3rd Qu.:0.8169726   3rd Qu.:0.9123
##  Max.   :0.3867   Max.   :0.9997871   Max.   :0.9980
```

Add now an external cross-validation loop - only using the training data. Here the default $V = 10$ is used for the inner loop, and we set the value for the outer loop (here $V = 3$ for speed).

```
system.time({cv_sl=CV.SuperLearner(Y=y_train, X=x_train,V=3,family=binomial(),SL.library=c("SL.mean","SL
```

```
##    user  system elapsed
##   8.540   0.178   8.746
```

```
summary(cv_sl)
```

```
##
## Call:
## CV.SuperLearner(Y = y_train, X = x_train, V = 3, family = binomial(), SL.library = c("SL.mean",
##     "SL.glmnet", "SL.randomForest"))
##
## Risk is based on: Mean Squared Error
##
## All risk estimates are based on V =  3
##
##            Algorithm      Ave        se      Min     Max
##        Super Learner 0.091052 0.0154191 0.056368 0.13943
##          Discrete SL 0.095636 0.0168328 0.056368 0.15197
##          SL.mean_All 0.242933 0.0096227 0.227600 0.26920
##        SL.glmnet_All 0.100032 0.0166562 0.062690 0.15197
##  SL.randomForest_All 0.078871 0.0119605 0.056368 0.10168
```

See the guide for more information on running multiple versions of one base learner, and parallellisation.