MA8701 Advanced methods in statistical inference and learning

Part 3: Ensembles. L15: Stacked ensembles

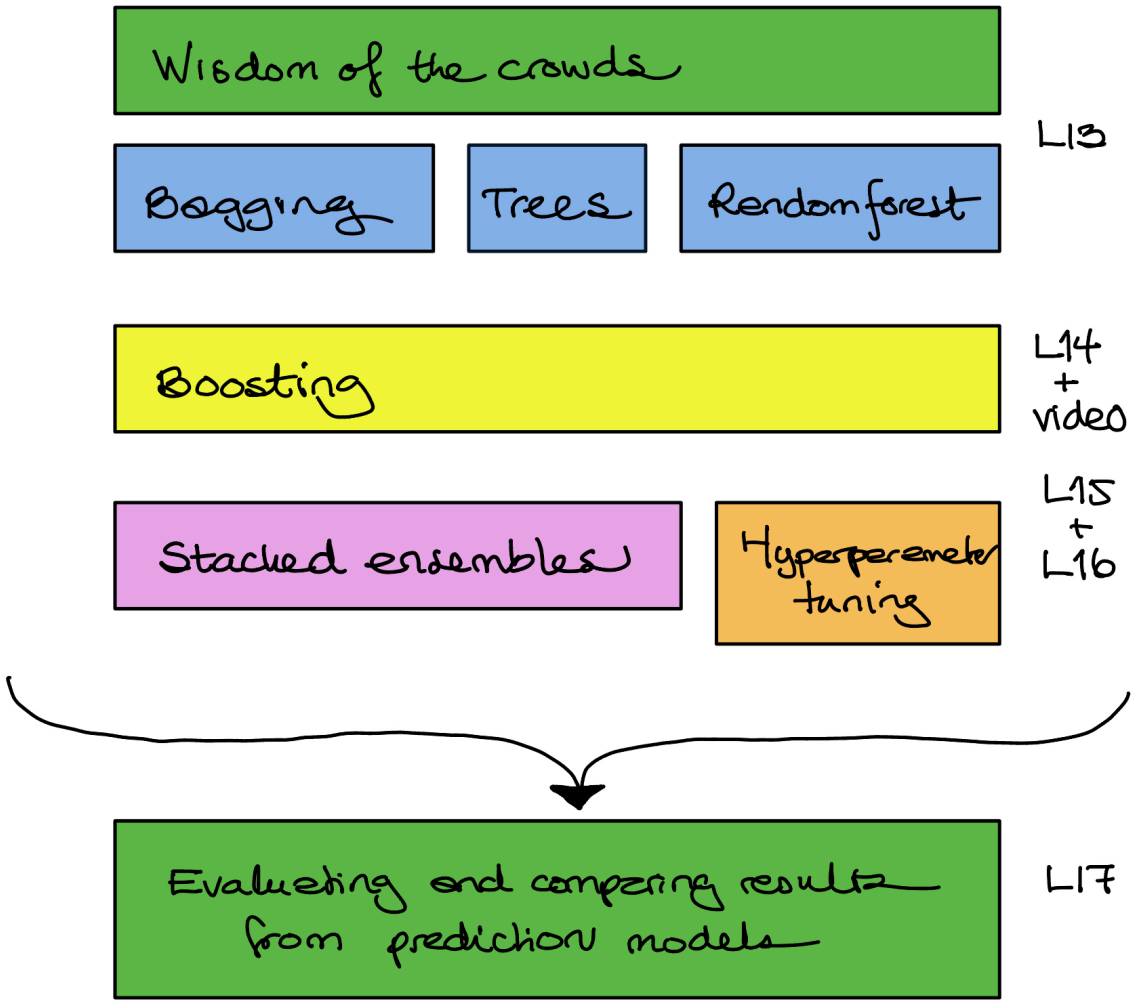
Mette Langaas

3/5/23

Table of contents

Course homepage: https://wiki.math.ntnu.no/ma8701/2023v/start

1 Before we start



1.1 Literature

- Erin Le Dell (2015): Scalable Ensemble Learning and Computationally Efficient Variance Estimation. PhD Thesis, University of California, Berkeley. or https://github.com/led

ell/phd-thesis. Section 2.

1.2 Supporting literature

- Breiman (1996)
- Laan, Polley, and Hubbard (2007)
- Polley, Rose, and Laan (2011)

---

2 Ensembles - overview

(ELS Ch 16.1)

With ensembles we want to build *one prediction model* which combines the strength of *a collection of models*.

These models may be simple base models - or more elaborate models.

We have studied bagging - where we use the bootstrap to repeatedly fit a statistical model, and then take a simple average of the predictions (or majority vote). Here the base models can be trees - or other type of models.

Random forest is a version of bagging with trees, with trees made to be different (decorrelated).

We have studied boosting, where the models are trained on sequentially different data - from residuals or gradients of loss functions - and the ensemble members cast weighted votes (down-weighted by a learning rate). We have observed that there are many hyperparameters that need to be to tuned to optimize performance.

3 Stacked ensembles

aka super learner or generalized stacking

3.1 What is it?

The Stacked Esembles is an algorithm that combines

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

3.2 Development:

- 1992: stacking introduce for neural nets by Wolpert
- 1996: adapted to regression problems by Breiman - but only for one type of methods at once (CART with different number of terminal nodes, GLMs with subset selection, ridge regression with different ridge penalty parameters) Breiman (1996)
- 2006: proven to have asymptotic theoretical oracle property by Laan, Polley, and Hubbard (2007)
- 2015: extensions in phd thesis by Erin LeDell LeDell (2015)

3.3 Ingredients:

- *Training data* (level-zero data) $O_i = (X_i, Y_i)$ of N i.i.d observations.
- A total of L *base learning algorithms* Ψ^l for $l = 1, ..., L$, each from some algorithmic class and each with a specific set of model parameters.
- A *metalearner* Φ is used to find an *optimal combination* of the L base learners.

3.4 Algorithm

Step 1: Produce level-one data Z

a) Divide the training data X randomly into V roughly-equally sized validation folds $X_{(1)}, \ldots, X_{(V)}$. V is often 5 or 10. (The responses Y are also needed.)

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

This gives the level-one data set 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 Z together with the responses $(Y_1, \ldots, Y_N)$.

b) The metalearner is used to estimate the weights given to each base learner: $\widehat{\eta}_i = \alpha_1 z_{1i} + \cdots + \alpha_L z_{Li}$.

What could the metalearner be?

- the mean (bagging)
- constructed by minimizing the
  - squared loss (ordinary least squares) or

  - non-negative least squares (most popular)

- ridge or lasso regression
- logistic regression (for binary classification)
- constructed by minimizing 1-ROC-AUC

(Class notes: Study Figure 3.2 from Polley, Rose, and Laan (2011) and/or Figure 1 from Laan, Polley, and Hubbard (2007))

3.5 The metalearning

Some observations

- The term *discrete super learner* is used if the base learner with the lowest risk (i.e. CV-error) is selected.
- Since the predictions from multiple base learners may be highly correlated - the chosen method should perform well in that case (i.e. ridge and lasso).
- When minimizing the squared loss it has been found that adding a non-negativity constraint $\alpha_l \leq 0$ works well,
- and also the additivity constraint $\sum_{l=1}^{L} \alpha_l = 1$ - the ensemble is a *convex combination* of the base learners.
- Non-linear optimization methods may be employed for the metalearner if no existing algorithm is available
- Historically a regularized linear model has "mostly" been used
- For classification the logistic response function can be used on the linear combination of base learners (Figure 3.2 Polley, Rose, and Laan (2011)).

---

4 Examples

4.1 Simulation examples

(Class notes: Study Figure 3.3 and Table 3.2 from Polley, Rose, and Laan (2011))

4.2 Real data

(Class notes: Study Figure 3.4 and Table 3.3 from P@Polley2011. RE=MSE relative to the linear model OLS.)

---

5 Theoretical result

LeDell (2015) (page 6)

- 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 Laan, Polley, and Hubbard (2007)
- 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 functions

---

5.1 Uncertainty in the ensemble

(Class notes: Study "Road map" 2 from Polley, Rose, and Laan (2011))

- 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 LeDell (2015))

5.2 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 package another.

---

6 R example from Superlearner package

Comment - this package is still in use, but the h2o-superlearner might be more "easy" to use.

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]
```

```r
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".

```r
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"
```

```r
# how does SL.glm differ from glm? obsWeight added to easy use the traning fold in the CV
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: 0x14c2979e0>
<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: 0x14c2f1350>
<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?).

```r
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
```

```r
#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.

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

```r
sl$times$everything
```

```
   user  system elapsed
  1.489   0.029   1.521
```

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=10,family=binomial(),SL.library=
```

```
   user  system elapsed
 15.003   0.202  15.207
```

```
summary(cv_sl)
```

```
Call:
CV.SuperLearner(Y = y_train, X = x_train, V = 10, family = binomial(), SL.library = c("SL.mea
    "SL.glmnet", "SL.randomForest"))

Risk is based on: Mean Squared Error

All risk estimates are based on V =  10

           Algorithm      Ave        se       Min     Max
       Super Learner 0.077041 0.0123412 0.0223396 0.12156
         Discrete SL 0.079783 0.0132389 0.0245396 0.12151
         SL.mean_All 0.242535 0.0093204 0.1947874 0.29619
       SL.glmnet_All 0.088109 0.0152056 0.0098891 0.14402
 SL.randomForest_All 0.073960 0.0115466 0.0245396 0.12151
```

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

7 R example from H2o-package

https://docs.h2o.ai/h2o/latest-stable/h2o-docs/data-science/stacked-ensembles.html

Python examples available from the same page

The Higgs boson data is used - but which version is not specified, maybe this https://archive.ics.uci.edu/ml/datasets/HIGGS or a specifically made data set. The problem is binary, so maybe to detect signal vs noise.

Default metalearner: Options include 'AUTO' (GLM with non negative weights; if validation_frame is present, a lambda search is performed)

```
h2o.init()
```

```
 Connection successful!

R is connected to the H2O cluster:
    H2O cluster uptime:         1 days 3 hours
    H2O cluster timezone:       Europe/Oslo
    H2O data parsing timezone:  UTC
    H2O cluster version:        3.40.0.1
    H2O cluster version age:    25 days
    H2O cluster name:           H2O_started_from_R_mettela_bze126
    H2O cluster total nodes:    1
    H2O cluster total memory:   2.96 GB
    H2O cluster total cores:    10
    H2O cluster allowed cores:  10
    H2O cluster healthy:        TRUE
    H2O Connection ip:          localhost
    H2O Connection port:        54321
    H2O Connection proxy:       NA
    H2O Internal Security:      FALSE
    R Version:                  R version 4.2.2 (2022-10-31)
```

```
# Import a sample binary outcome train/test set into H2O
train <- h2o.importFile("https://s3.amazonaws.com/erin-data/higgs/higgs_train_10k.csv")
```

```
  |
  |                                                                      |   0%
  |
  |====                                                                  |   5%
  |
  |==============                                                        |  20%
  |
  |=======================                                               |  32%
  |
  |================================                                      |  50%
  |
  |==============================================                        |  67%
```

```
  |
  |=============================================================          |  87%
  |
  |======================================================================| 100%
```

```
test <- h2o.importFile("https://s3.amazonaws.com/erin-data/higgs/higgs_test_5k.csv")
```

```
  |
  |                                                                      |   0%
  |
  |=================                                                     |  25%
  |
  |=================================================                     |  72%
  |
  |======================================================================| 100%
```

```
# Identify predictors and response
y <- "response"
x <- setdiff(names(train), y)

# For binary classification, response should be a factor
train[, y] <- as.factor(train[, y])
test[, y] <- as.factor(test[, y])

print(dim(train))
```

```
[1] 10000    29
```

```
print(colnames(train))
```

```
 [1] "response" "x1"       "x2"       "x3"       "x4"       "x5"
 [7] "x6"       "x7"       "x8"       "x9"       "x10"      "x11"
[13] "x12"      "x13"      "x14"      "x15"      "x16"      "x17"
[19] "x18"      "x19"      "x20"      "x21"      "x22"      "x23"
[25] "x24"      "x25"      "x26"      "x27"      "x28"
```

```
print(dim(test))
```

```
[1] 5000   29
```

```
# Number of CV folds (to generate level-one data for stacking)
nfolds <- 5

# There are a few ways to assemble a list of models to stack toegether:
# 1. Train individual models and put them in a list


# 1. Generate a 2-model ensemble (GBM + RF)

# Train & Cross-validate a GBM
my_gbm <- h2o.gbm(x = x,
                  y = y,
                  training_frame = train,
                  distribution = "bernoulli",
                  ntrees = 10,
                  max_depth = 3,
                  min_rows = 2,
                  learn_rate = 0.2,
                  nfolds = nfolds,
                  keep_cross_validation_predictions = TRUE,
                  seed = 1)
```

```
|
|                                                                       |   0%
|
|=============================================                          |  67%
|
|======================================================================| 100%
```

```
# Train & Cross-validate a RF
my_rf <- h2o.randomForest(x = x,
                          y = y,
                          training_frame = train,
                          ntrees = 50,
                          nfolds = nfolds,
```

```
                          keep_cross_validation_predictions = TRUE,
                          seed = 1)
```

```
|
|                                                                      |   0%
|
|=======                                                               |   9%
|
|==================                                                    |  26%
|
|============================                                          |  42%
|
|=======================================                               |  58%
|
|===================================================                   |  74%
|
|===============================================================       |  90%
|
|======================================================================| 100%
```

7.1 Now the default metalearner

AUTO: glm with non-negative weights

```
# Train a stacked ensemble using the GBM and RF above
ensemble <- h2o.stackedEnsemble(x = x,
                                y = y,
                                training_frame = train,
                                base_models = list(my_gbm, my_rf))
```

```
|
|                                                                      |   0%
|
|======================================================================| 100%
```

```
# default metalearner_transform should be NONE
#print(summary(ensemble))
#ensemble@model
# Eval ensemble performance on a test set
perf <- h2o.performance(ensemble, newdata = test)

# Compare to base learner performance on the test set
perf_gbm_test <- h2o.performance(my_gbm, newdata = test)
perf_rf_test <- h2o.performance(my_rf, newdata = test)
baselearner_best_auc_test <- max(h2o.auc(perf_gbm_test), h2o.auc(perf_rf_test))
ensemble_auc_test <- h2o.auc(perf)
print(sprintf("Best Base-learner Test AUC:  %s", baselearner_best_auc_test))
```

```
[1] "Best Base-learner Test AUC:  0.769204725074508"
```

```
print(sprintf("Ensemble Test AUC:  %s", ensemble_auc_test))
```

```
[1] "Ensemble Test AUC:  0.773144298176816"
```

```
# [1] "Best Base-learner Test AUC:  0.76979821502548"
# [1] "Ensemble Test AUC:  0.773501212640419"

# Generate predictions on a test set (if neccessary)
pred <- h2o.predict(ensemble, newdata = test)
```

```
  |
  |                                                                      |   0%
  |
  |======================================================================| 100%
```

```
print(head(pred))
```

```
  predict        p0        p1
1       0 0.6839178 0.3160822
2       1 0.5825428 0.4174572
3       1 0.5869431 0.4130569
```

```
4       1 0.1927212 0.8072788
5       1 0.4509384 0.5490616
6       1 0.3275080 0.6724920
```

```
metalearner_model
Model Details:
==============

H2OBinomialModel: glm
Model ID:  metalearner_AUTO_StackedEnsemble_model_R_1677945156774_1824
GLM Model: summary
    family  link                                regularization number_of_predictors_total
1 binomial logit Elastic Net (alpha = 0.5, lambda = 8.399E-5 )                          2
                                          training_frame
1 levelone_training_StackedEnsemble_model_R_1677945156774_1824

Coefficients: glm coefficients
                         names coefficients standardized_coefficients
1                    Intercept    -3.603549                 0.149102
2 GBM_model_R_1677945156774_1086     3.298011                 0.493334
3 DRF_model_R_1677945156774_1214     3.809905                 0.701246
```

```
# Train a stacked ensemble using the GBM and RF above
ensemble <- h2o.stackedEnsemble(x = x,
                                y = y,
                                training_frame = train,
                                base_models = list(my_gbm, my_rf),
                                metalearner_transform = "Logit")
```

```
|
|                                                                      |   0%
|
|======================================================================| 100%
```

```
#print(summary(ensemble))
#print(ensemble@model)

# Eval ensemble performance on a test set
perf <- h2o.performance(ensemble, newdata = test)
```

```r
# Compare to base learner performance on the test set
perf_gbm_test <- h2o.performance(my_gbm, newdata = test)
perf_rf_test <- h2o.performance(my_rf, newdata = test)
baselearner_best_auc_test <- max(h2o.auc(perf_gbm_test), h2o.auc(perf_rf_test))
ensemble_auc_test <- h2o.auc(perf)
print(sprintf("Best Base-learner Test AUC:  %s", baselearner_best_auc_test))
```

```
[1] "Best Base-learner Test AUC:  0.769204725074508"
```

```r
print(sprintf("Ensemble Test AUC:  %s", ensemble_auc_test))
```

```
[1] "Ensemble Test AUC:  0.773096033881535"
```

```r
# Generate predictions on a test set (if neccessary)
pred <- h2o.predict(ensemble, newdata = test)
```

```
  |
  |                                                                      |   0%
  |
  |======================================================================| 100%
```

```r
print(head(pred))
```

```
  predict        p0        p1
1       0 0.6739209 0.3260791
2       1 0.5814741 0.4185259
3       1 0.5826643 0.4173357
4       1 0.1971804 0.8028196
5       1 0.4561659 0.5438341
6       1 0.3365841 0.6634159
```

```
$metalearner_model
Model Details:
==============
```

```
H2OBinomialModel: glm
Model ID:  metalearner_AUTO_StackedEnsemble_model_R_1677945156774_1830
GLM Model: summary
    family  link                                regularization number_of_predictors_total
1 binomial logit Elastic Net (alpha = 0.5, lambda = 3.885E-4 )                         2
                                            training_frame
1 levelone_training_StackedEnsemble_model_R_1677945156774_1830

Coefficients: glm coefficients
                         names coefficients standardized_coefficients
1                    Intercept    -0.053725                  0.154528
2 GBM_model_R_1677945156774_1086     0.791767                  0.515081
3 DRF_model_R_1677945156774_1214     0.845217                  0.731991
```

8 References

Breiman, Leo. 1996. "Stacked Regressions." *Machine Learning* 24 (1): 49–64. https://doi.org/10.1007/BF00117832.

Hastie, Trevor, Robert Tibshirani, and Jerome Friedman. 2009. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Vol. 2. Springer series in statistics New York. hastie.su.domains/ElemStatLearn.

Laan, Mark J. van der, Eric C Polley, and Alan E. Hubbard. 2007. *Statistical Applications in Genetics and Molecular Biology* 6 (1). https://doi.org/doi:10.2202/1544-6115.1309.

LeDell, Erin. 2015. "Scalable Ensemble Learning and Computationally Efficient Variance Estimation."

Polley, Eric C., Sherri Rose, and Mark J. van der Laan. 2011. "Super Learning." In *Targeted Learning: Causal Inference for Observational and Experimental Data*, 43–66. New York, NY: Springer New York. https://doi.org/10.1007/978-1-4419-9782-1_3.