MA8701 Advanced methods in statistical inference and learning

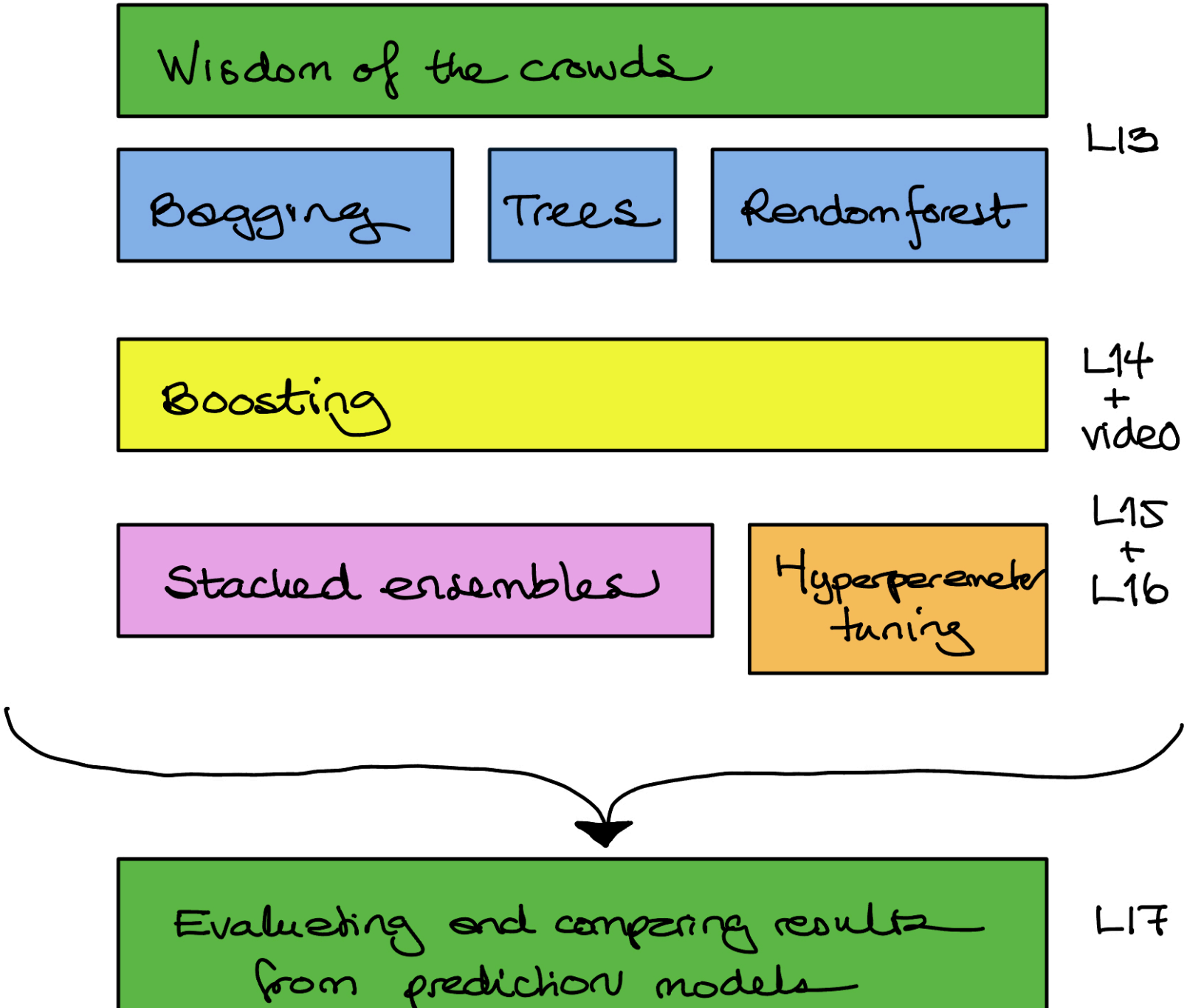
Part 3: Ensembles. L15: Stacked ensembles

Mette Langaas

3/5/23

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

Before we start



Literature

- Erin Le Dell (2015): Scalable Ensemble Learning and Computationally Efficient Variance Estimation. PhD Thesis, University of California, Berkeley. or https://github.com/ledell/phd-thesis. Section 2.

Supporting literature

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

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 (downweighted by a learning rate). We have observed that there are many hyperparameters that need to be to tuned to optimize performance.

Stacked ensembles

aka super learner or generalized stacking

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.

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)

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 $l = 1, \ldots, L$, 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.

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: $\hat{\eta}_i = \alpha_1 z_{1i} + \cdots + \alpha_L z_{Li}$.

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

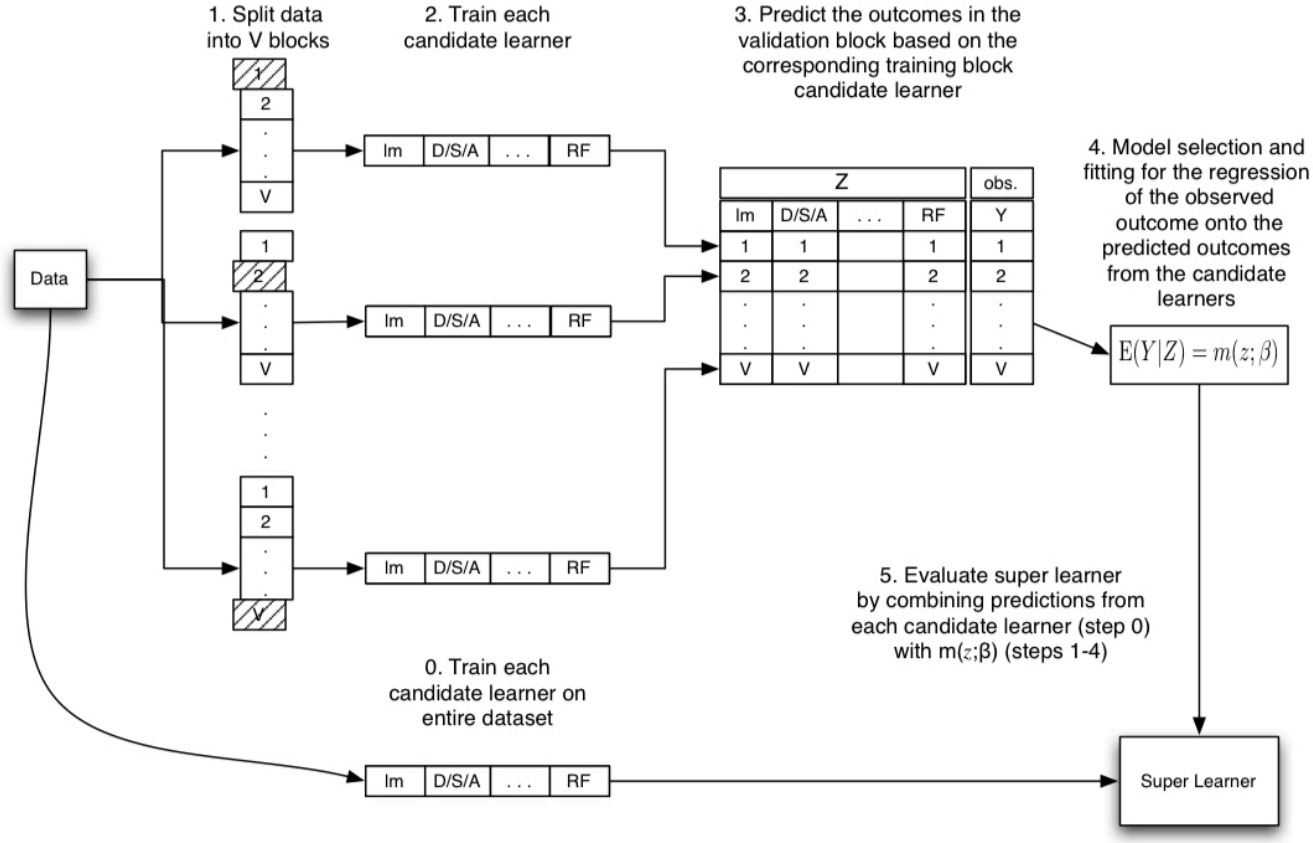


Figure 1: Flow Diagram for Super Learner

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

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)).

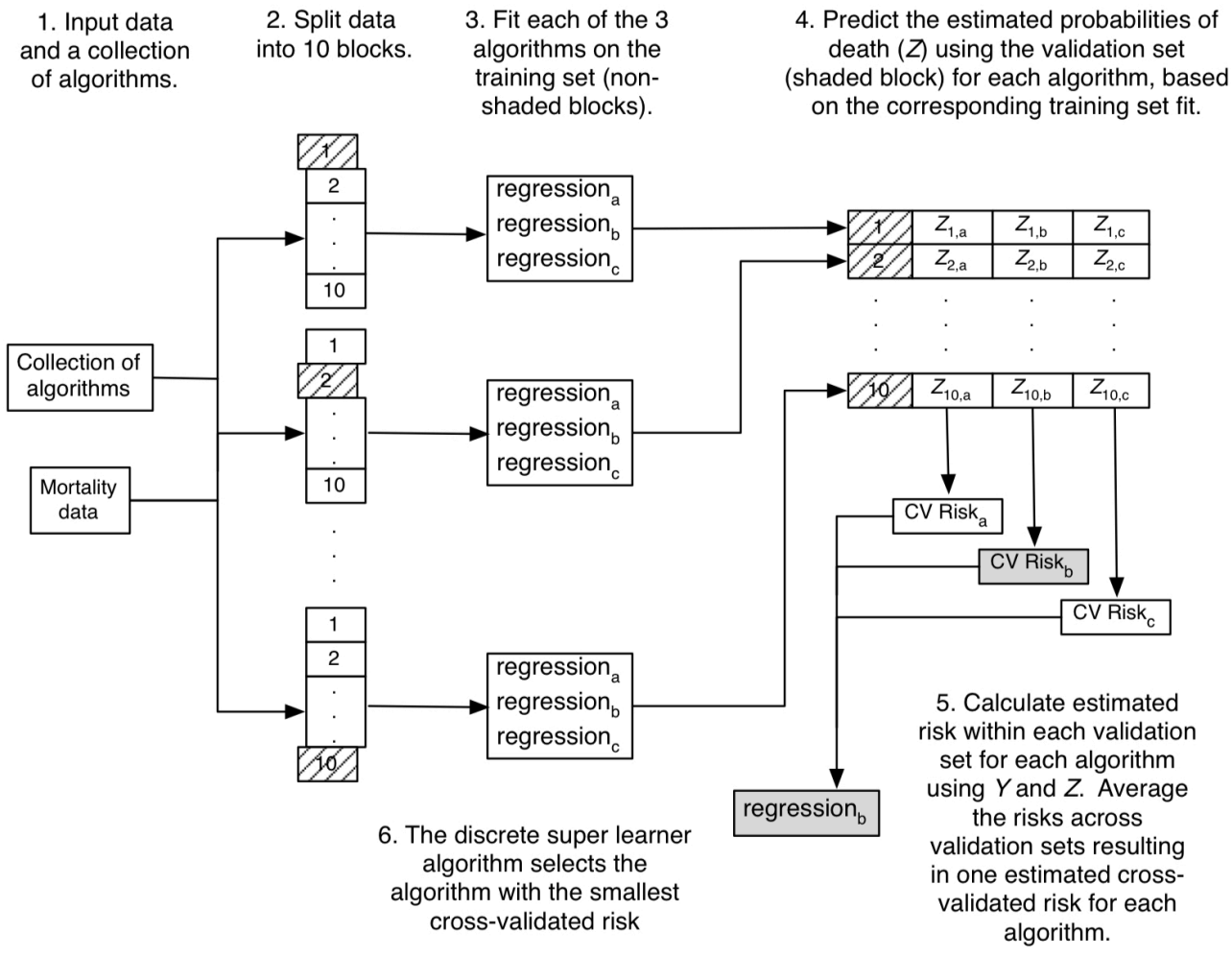


Fig. 3.1 Discrete super learner algorithm for the mortality study example where $\bar{Q}_n^b(A, W)$ is the algorithm with the smallest cross-validated risk

Examples

Simulation examples

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

Real data

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

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

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))

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.

Super Learner Algorithm

The steps below describe the individual tasks involved in training and testing a Super Learner ensemble. H2O automates most of the steps below so that you can quickly and easily build ensembles of H2O models.

1. Set up the ensemble.

    a. Specify a list of L base algorithms (with a specific set of model parameters).
    b. Specify a metalearning algorithm.

2. Train the ensemble.

    a. Train each of the L base algorithms on the training set.
    b. Perform k-fold cross-validation on each of these learners and collect the cross-validated predicted values from each of the L algorithms.
    c. The N cross-validated predicted values from each of the L algorithms can be combined to form a new N x L matrix. This matrix, along with the original response vector, is called the "level-one" data. (N = number of rows in the training set.)
    d. Train the metalearning algorithm on the level-one data. The "ensemble model" consists of the L base learning models and the metalearning model, which can then be used to generate predictions on a test set.

3. Predict on new data.

    a. To generate ensemble predictions, first generate predictions from the base learners.
    b. Feed those predictions into the metalearner to generate the ensemble prediction.

- metalearner_algorithm (Optional) Specify the metalearner algorithm type. Options include:

  - `"AUTO"` (GLM with non negative weights & standardization turned off, and if `validation_frame` is present, then `lambda_search` is set to True; may change over time). This is the default.
  - `"glm"` (GLM with default parameters)
  - `"gbm"` (GBM with default parameters)
  - `"drf"` (Random Forest with default parameters)
  - `"deeplearning"` (Deep Learning with default parameters)
  - `"naivebayes"` (NaiveBayes with default parameters)
  - `"xgboost"` (if available, XGBoost with default parameters)

- metalearner_params: (Optional) If a `metalearner_algorithm` is specified, then you can also specify a list of customized parameters for that algorithm (for example, a GBM with `ntrees=100`, `max_depth=10`, etc.)
- metalearner_nfolds: (Optional) Specify the number of folds for cross-validation of the metalearning algorithm. Defaults to 0 (no cross-validation). If you want to compare the cross-validated performance of the ensemble model to the cross-validated performance of the base learners or other algorithms, you should make use of this option.
- metalearner_fold_assignment: (Optional; Applicable only if a value for `metalearner_nfolds` is specified) Specify the cross-validation fold assignment scheme for the metalearner. The available options are AUTO (which is Random), Random, Modulo, or Stratified (which will stratify the folds based on the response variable for classification problems). This value defaults to AUTO.
- metalearner_fold_column: (Optional; Cannot be used at the same time as `nfolds`) Specify the name of the column that contains the cross-validation fold assignment per observation for cross-validation of the metalearner. The column can be numeric (e.g. fold index or other integer value) or it can be categorical. The number of folds is equal to the number of unique values in this column.
- metalearner_transform: (Optional) Specify the transformation used on predictions from the base models in order to make a level one frame. Options include:

  - `"NONE"` (no transform applied)
  - `"Logit"` (applicable only to classification tasks, use logit transformation on the predicted probabilities)

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.

```
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
```

Now the default metalearner

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

```
# 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))
```

```
  |
  |
  |
  |=======================================================================
```

```
# 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_r
```

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

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

Adding transform “logit”

```
# 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")
```

```
  |
  |
  |
  |=============================================================
```

```
#print(summary(ensemble))
#print(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_r
ensemble_auc_test <- h2o.auc(perf)
```

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

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

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.