Stacked Ensembles

Introduction

Ensemble machine learning methods use multiple learning algorithms to obtain better predictive performance than could be obtained from any of the constituent learning algorithms. Many of the popular modern machine learning algorithms are actually ensembles. For example, Random Forest and Gradient Boosting Machine (GBM) are both ensemble learners. Both bagging (e.g. Random Forest) and boosting (e.g. GBM) are methods for ensembling that take a collection of weak learners (e.g. decision tree) and form a single, strong learner.

H2O's Stacked Ensemble method is a supervised ensemble machine learning algorithm that finds the optimal combination of a collection of prediction algorithms using a process called stacking. Like all supervised models in H2O, Stacked Ensemeble supports regression, binary classification, and multiclass classification.

Native support for ensembles of H2O algorithms was added into core H2O in version 3.10.3.1. A separate implementation, the **h2oEnsemble** R package, is also still available, however for new projects we recommend using the native H2O version, documented below.

Stacking / Super Learning

Stacking, also called Super Learning [3] or Stacked Regression [2], is a class of algorithms that involves training a second-level "metalearner" to find the optimal combination of the base learners. Unlike bagging and boosting, the goal in stacking is to ensemble strong, diverse sets of learners together.

Although the concept of stacking was originally developed in 1992 [1], the theoretical guarantees for stacking were not proven until the publication of a paper titled, "Super Learner", in 2007 [3]. In this paper, it was shown that the Super Learner ensemble represents an asymptotically optimal system for learning.

There are some ensemble methods that are broadly labeled as stacking, however, the Super Learner ensemble is distinguished by the use of cross-validation to form what is called the "level-one" data, or the data that the metalearning or "combiner" algorithm is trained on. More detail about the Super Learner algorithm is provided below.

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.

Training Base Models for the Ensemble

Before training a stacked ensemble, you will need to train and cross-validate a set of "base models" which will make up the ensemble. In order to stack these models together, a few things are required:

- The models must be cross-validated using the same number of folds (e.g. `nfolds = 5` or use the same `fold_column` across base learners).
- The cross-validated predictions from all of the models must be preserved by setting `keep_cross_validation_predictions` to True. This is the data which is used to train the metalearner, or "combiner", algorithm in the ensemble.
- You can train these models manually, or you can use a group of models from a grid search.
- The models must be trained on the same `training_frame`. The rows must be identical, but you can use different sets of predictor columns, `x`, across models if you choose. Using base models trained on different subsets of the feature space will add more randomness/diversity to the set of base models, which in theory can improve ensemble performance. However, using all available predictor columns for each base model will often still yield the best results (the more data, the better the models).

Note: You can train a Stacked Ensemble model using only monotonic models by specifying `monotone_constraints` in AutoML and creating at least 2 monotonic models.

Defining a Stacked Ensemble Model

- y: (Required) Specify the index or column name of the column to use as the dependent variable (response column). The response column can be numeric (regression) or categorical (classification).
- x: (Optional) Specify a vector containing the names or indices of the predictor variables to use when building the model. If `x` is missing, then all columns except `y` are used. The only use for `x` is to get the correct training set so that we can compute ensemble training metrics.
- training_frame (Required) Specify the dataset used to build the model. In a Stacked Ensemble model, the training frame is used only to retreive the response column (needed for training the metalearner) and also to compute training metrics for the ensemble model.
- validation_frame: (Optional) Specify the dataset to use for tuning the model. The validation frame will be passed through to the metalearner for tuning.
- blending_frame: (Optional) Specify a frame to be used for computing the predictions that serve as the training frame for the metalearner. This triggers blending mode if provided.
- model_id: (Optional) Specify a custom name for the model to use as a reference. By default, H2O automatically generates a destination key.
- base_models: (Required) Specify a list of models (or model IDs) that can be stacked together. Models must have been cross-validated (i.e. `nfolds`>1 or `fold_column` was specified), they all must use the same cross-validation folds, and `keep_cross_validation_predictions` must have been set to True. One way to guarantee identical folds across base models is to set `fold_assignment = "Modulo"` in all the base models. It is also possible to get identical folds by setting `fold_assignment = "Random"` when the same seed is used in all base models.
- 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)

- offset_column: (Optional; Availability depends on the `metalearner_algorithm`) Specify a column to use as the offset.
- weights_column: (Optional) Specifies a column with observation weights. Giving some observation a weight of zero is equivalent to excluding it from the dataset; giving an observation a relative weight of 2 is equivalent to repeating that row twice. Negative weights are not allowed.
- **keep_levelone_frame**: (Optional) Keep the level one data frame that's constructed for the metalearning step. Defaults to False.
- max_runtime_secs: (Optional) Maximum allowed runtime in seconds for the metalearner model training. Use 0 to disable the time limit. Defaults to 0.
- seed: (Optional) Seed for random numbers; passed through to the metalearner algorithm. Defaults to -1 (time-based random number).
- export_checkpoints_dir: Specify a directory to which generated models will automatically be exported.

You can follow the progress of H2O's Stacked Ensemble development here.

Examples

Below is a simple example showing how to build a Stacked Ensembles model.

R Python Scala

```r
library(h2o)
h2o.init()

# Import a sample binary outcome train/test set into H2O
train <- h2o.importFile("https://s3.amazonaws.com/erin-data/higgs/higgs_train_10k.csv")
test <- h2o.importFile("https://s3.amazonaws.com/erin-data/higgs/higgs_test_5k.csv")

# 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])

# 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
# 2. Train a grid of models
# 3. Train several grids of models
# Note: All base models must have the same cross-validation folds and
# the cross-validated predicted values must be kept.


# 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)

# 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)

# 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))

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


# 2. Generate a random grid of models and stack them together

# GBM Hyperparamters
learn_rate_opt <- c(0.01, 0.03)
max_depth_opt <- c(3, 4, 5, 6, 9)
sample_rate_opt <- c(0.7, 0.8, 0.9, 1.0)
col_sample_rate_opt <- c(0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8)
hyper_params <- list(learn_rate = learn_rate_opt,
                     max_depth = max_depth_opt,
                     sample_rate = sample_rate_opt,
                     col_sample_rate = col_sample_rate_opt)

search_criteria <- list(strategy = "RandomDiscrete",
                        max_models = 3,
                        seed = 1)

gbm_grid <- h2o.grid(algorithm = "gbm",
                     grid_id = "gbm_grid_binomial",
                     x = x,
                     y = y,
                     training_frame = train,
                     ntrees = 10,
                     seed = 1,
                     nfolds = nfolds,
                     keep_cross_validation_predictions = TRUE,
                     hyper_params = hyper_params,
                     search_criteria = search_criteria)

# Train a stacked ensemble using the GBM grid
ensemble <- h2o.stackedEnsemble(x = x,
                                y = y,
                                training_frame = train,
                                base_models = gbm_grid@model_ids)

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

# Compare to base learner performance on the test set
.getauc <- function(mm) h2o.auc(h2o.performance(h2o.getModel(mm), newdata = test))
baselearner_aucs <- sapply(gbm_grid@model_ids, .getauc)
baselearner_best_auc_test <- max(baselearner_aucs)
ensemble_auc_test <- h2o.auc(perf)
print(sprintf("Best Base-learner Test AUC:  %s", baselearner_best_auc_test))
print(sprintf("Ensemble Test AUC:  %s", ensemble_auc_test))
# [1] "Best Base-learner Test AUC:  0.748146530400473"
# [1] "Ensemble Test AUC:  0.773501212640419"
# Generate predictions on a test set (if neccessary)
pred <- h2o.predict(ensemble, newdata = test)
```

FAQ

- **How do I save ensemble models?**

  H2O now supports saving and loading ensemble models. The steps are the same as those described in the Saving and Loading a Model section. For productionizing Stacked Ensemble models, we recommend using MOJOs.

- **Will an stacked ensemble always perform better than a single model?**

  Hopefully, but it's not always the case (especially if you have very small data). That's why it always a good idea to check the performance of your stacked ensemble and compare it against the performance of the individual base learners.

- **How do I improve the performance of an ensemble?**

  If you find that your ensemble is not performing better than the best base learner, then you can try a few different things. First make sure to try the default metalearner ("AUTO") and then try the other options for `metalearner_algorithm`. Additionally, the custom parameters could be passed to `metalearner_params` (e.g., a GBM with `ntrees=1000`, `max_depth=10`, etc.)

  Second, look to see if there are base learners that are performing much worse than the other base learners (for example, a GLM). If so, remove them from the ensemble and try again.

  You can also try adding more models to the ensemble, especially models that add diversity to your set of base models. Training a random grid of models (or multiple random grids, one for each algorithm type) is a good way to generate a diverse set of base learners.

- **How does the algorithm handle missing values during training?**

  This is handled by the base algorithms of the ensemble. See the documentation for those algorithms to find out more information.

- **How does the algorithm handle missing values during testing?**

This is handled by the base algorithms of the ensemble. See the documentation for those algorithms to find out more information.

- **What happens if the response has missing values?**

  No errors will occur, but nothing will be learned from rows containing missing values in the response column.

- **What happens when you try to predict on a categorical level not seen during training?**

  This is handled by the base algorithms of the ensemble. See the documentation for those algorithms to find out more information.

- **How does the algorithm handle highly imbalanced data in a response column?**

  In the base learners, specify `balance_classes`, `class_sampling_factors` and `max_after_balance_size` to control over/under-sampling.

Additional Information

- An Ensemble slidedeck from January 2017 provides a summary of the new Stacked Ensemble method in H2O, along with a comparison to the pre-existing h2oEnsemble R package.
- Python Stacked Ensemble tests are available in the H2O-3 GitHub repository.
- R Stacked Enemble tests are available in the H2O-3 GitHub repository.

References

[1] David H. Wolpert. “Stacked Generalization.” Neural Networks. Volume 5. (1992)

[2] Leo Breiman. “Stacked Regressions.” Machine Learning, 24, 49-64 (1996)

[3] Mark J van der Laan, Eric C Polley, and Alan E Hubbard. “Super Learner.” Journal of the American Statistical Applications in Genetics and Molecular Biology. Volume 6, Issue 1. (September 2007).

[4] LeDell, E. “Scalable Ensemble Learning and Computationally Efficient Variance Estimation” (Doctoral Dissertation). University of California, Berkeley, USA. (2015)