Intro to Practical Ensemble Learning

Erin LeDell

Group in Biostatistics
University of California, Berkeley

April 27, 2015

Overview

- Ensemble Learning
- Super Learning / Stacking
- Subsemble Algorithm
- Overview of Ensemble Software
- Code Demo

Ensemble Learning

In statistics and machine learning, **ensemble methods** use multiple learning algorithms to obtain better predictive performance than could be obtained from any of the constituent learning algorithms.

– Wikipedia, 2015

- Ensemble of weak learners (e.g. Random Forest)
- Generalized Model Stacking (combine the predictions from multiple models)

Ensemble Learning

Why ensembles?

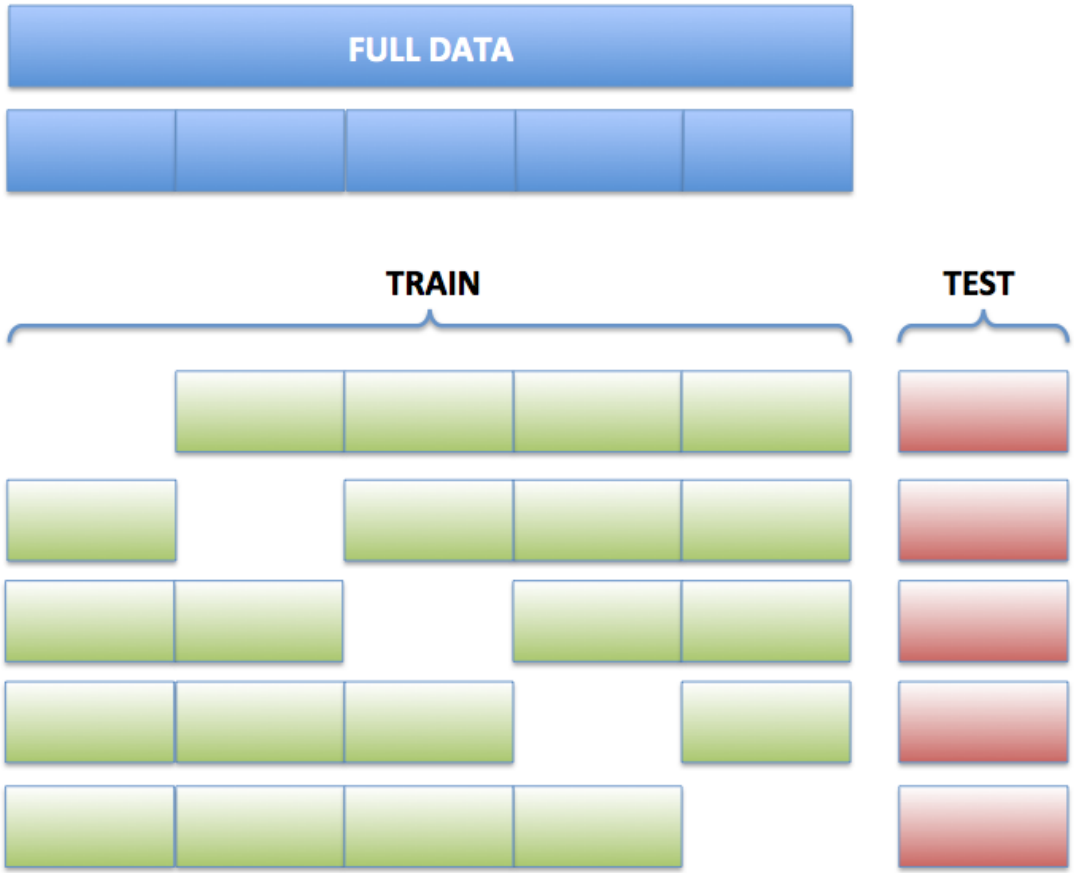
- If your set of base learners does not contain the true prediction function, ensembles can give a good approximation of that function.
- Ensembles perform better than the individual base algorithms.
- Ensembles $\geq$ Grid Search
- Win at Kaggle!

Super Learner algorithm

The **Super Learner algorithm** is a loss-based supervised learning method that finds the optimal combination of a collection of prediction algorithms.

Super Learner performs asymptotically as well as best possible weighted combination of the base learners.

Performance Evaluation: Cross validation



Example: 5-fold cross validation

Super Learner algorithm

Super Learner / Stacking Terminology

- **Level-zero data**: The original training set, $X_{n \times p}$
- **Level-one data**: The cross-validated predicted values, $Z_{n \times L}$, generated from cross-validating base learners on X
- **Learner Library**: Set of L base learning algorithms
- **Metalearner**: Algorithm trained using Z as design matrix and original outcome Y

Super Learning: Algorithm History

- (1992) David H. Wolpert: "Stacked Generalization" in the context of neural net ensembles. Used leave-one-out CV to generate level-one data.
- (1996) Leo Breiman: "Stacked Regressions" extended Wolpert's stacking framework to regression problems. Proposed k-fold CV to generate level-one data. Suggested non-negativity constraints for the metalearner.
- (1996) Michael Leblanc, Rob Tibshirani: "Combining Estimates in Regression and Classification" provided a general framework for stacking and compared CV-generated level-one data to bootstrapped level-one data.
- (2007) Mark van der Laan, Eric Polley, Alan Hubbard: "Super Learner" provided the theory for stacking. Guarantees asymptotic equivalence to the oracle.

Super Learner algorithm

Super Learner: The setup

1. Define a base learner library of L learners, $\Psi^1, ..., \Psi^L$.
2. Specify a metalearning method, Φ.
3. Partition the training observations into V folds.

Super Learner algorithm

Super Learner: The algorithm

1. Generate a matrix Z, of dimension $n \times L$, of cross-validated predictions as follows: During cross-validation, we obtain fits, $\hat{\Psi}^l_{-v}$, defined as fitting Ψ^l on the observations that are not in fold v. Predictions are then generated for the observations in the v^{th} fold.
2. Find the optimal combination of subset-specific fits according to a user-specified metalearner algorithm, $\hat{\Phi}$, with a new design matrix, Z.
3. Fit L models (one for each base learner) on the original training set, X, and save the L individual model fit objects along with $\hat{\Phi}$. This ensemble model can be used to generate predictions on new data.

Subsemble algorithm

"Subsemble: An ensemble method for combining subset-specific algorithm fits"

Stephanie Sapp, Mark J. van der Laan & John Canny
Journal of Applied Statistics (2013)

Subsemble algorithm

Subsemble: The Setup

1. Define a candidate learner library of L learners, $\Psi^{(1)}, ..., \Psi^{(L)}$, and specify a metalearning method, Φ.
2. Partition the data set into J subsets (currently, at random).
3. Partition each subset into V folds. The v^{th} validation fold spans across J subsets.

Note: A Subsemble with $J = 1$ is equivalent to Super Learner algorithm.

Subsemble algorithm

Subsemble: The Algorithm

1. Generate a matrix Z, of dimension $n \times (J \times L)$, of cross-validated predictions as follows. During cross-validation, we obtain fits, $\hat{\Psi}^{(l)}_{j,-v}$, defined as fitting $\Psi^{(l)}$ on the observations that are in subset j, but not in fold v. Predictions are then generated for the observations in the v^{th} fold, which spans across subsets. The CV step can be done in parallel.
2. Find the optimal combination of subset-specific fits according to a user-specified metalearner algorithm, $\hat{\Phi}$, with a new design matrix, Z.
3. Fit $J \times L$ models on the subsets (in parallel) and save the individual fits along with $\hat{\Phi}$. This model can be used to generate predictions on new data.

Ensemble Software

- **SuperLearner** R package
- **subsemble** R package
- **h2oEnsemble** R package

Ensemble Software

Install required R packages:

Example

```
install.packages("devtools")
library("devtools")

install_github("ecpolley/SuperLearner")
install_github("ledell/subsemble")

install.packages("h2o")
install_github("h2oai/h2o/R/ensemble/h2oEnsemble-package")
```

SuperLearner R package

SuperLearner: Set up the ensemble

Example

```
SL.library <- c("SL.knn",
                "SL.glm",
                "SL.randomForest")

method <- "method.NNLS"

family <- "binomial"
```

SuperLearner R package

SuperLearner: How to train & test

Example

```
fit <- SuperLearner(Y = Y, X = X,
                    family = family,
                    SL.library = SL.library,
                    method = method)

pred <- predict(fit, newdata = newX)
```

subsemble R package

subsemble: Set up the ensemble

Example

```
learner <- c("SL.knn",
             "SL.glm",
             "SL.randomForest")

metalearner <- "SL.nnls"

family <- "binomial"
subsets <- 4
```

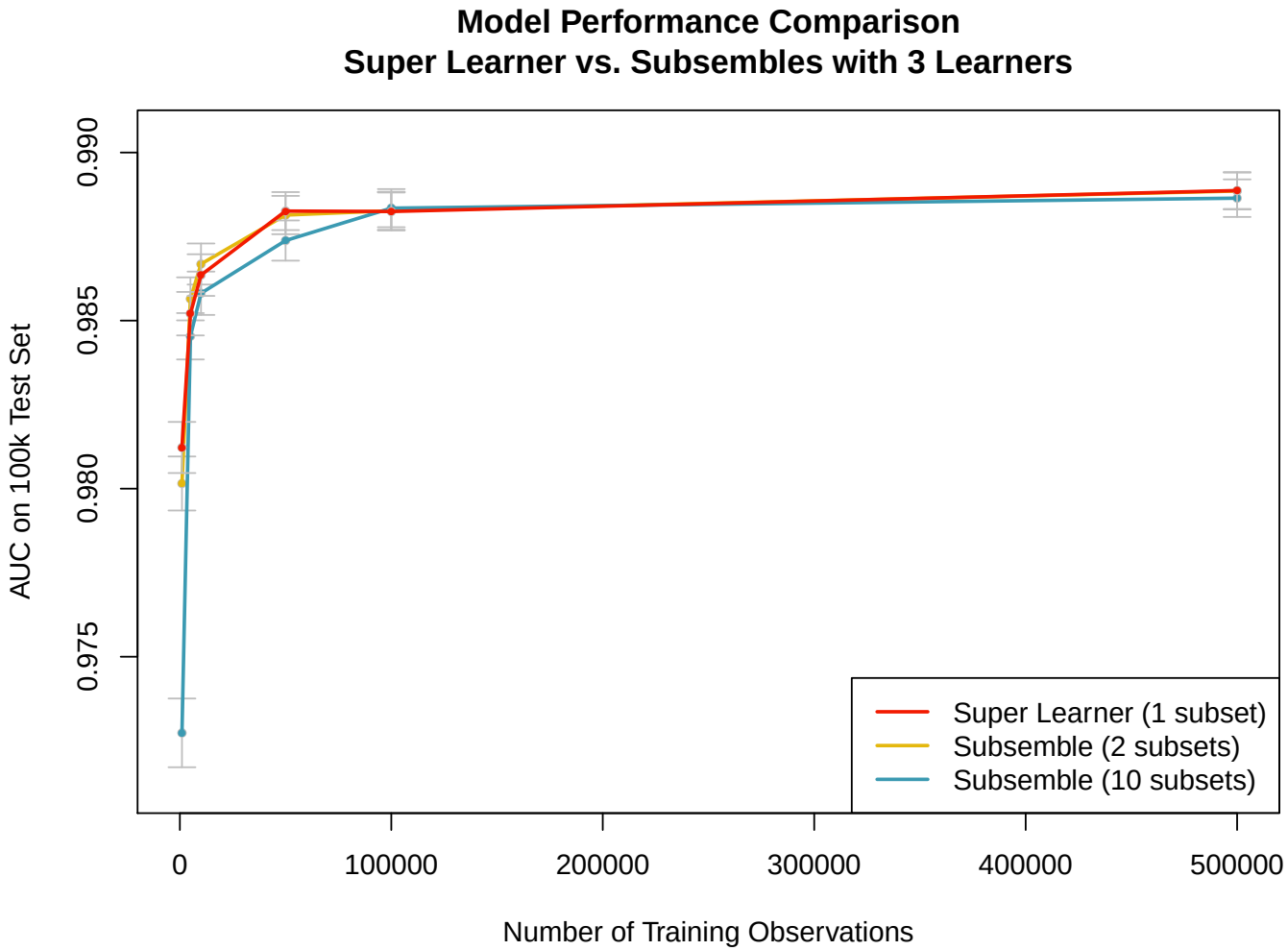
subsemble R package

subsemble: How to train & test

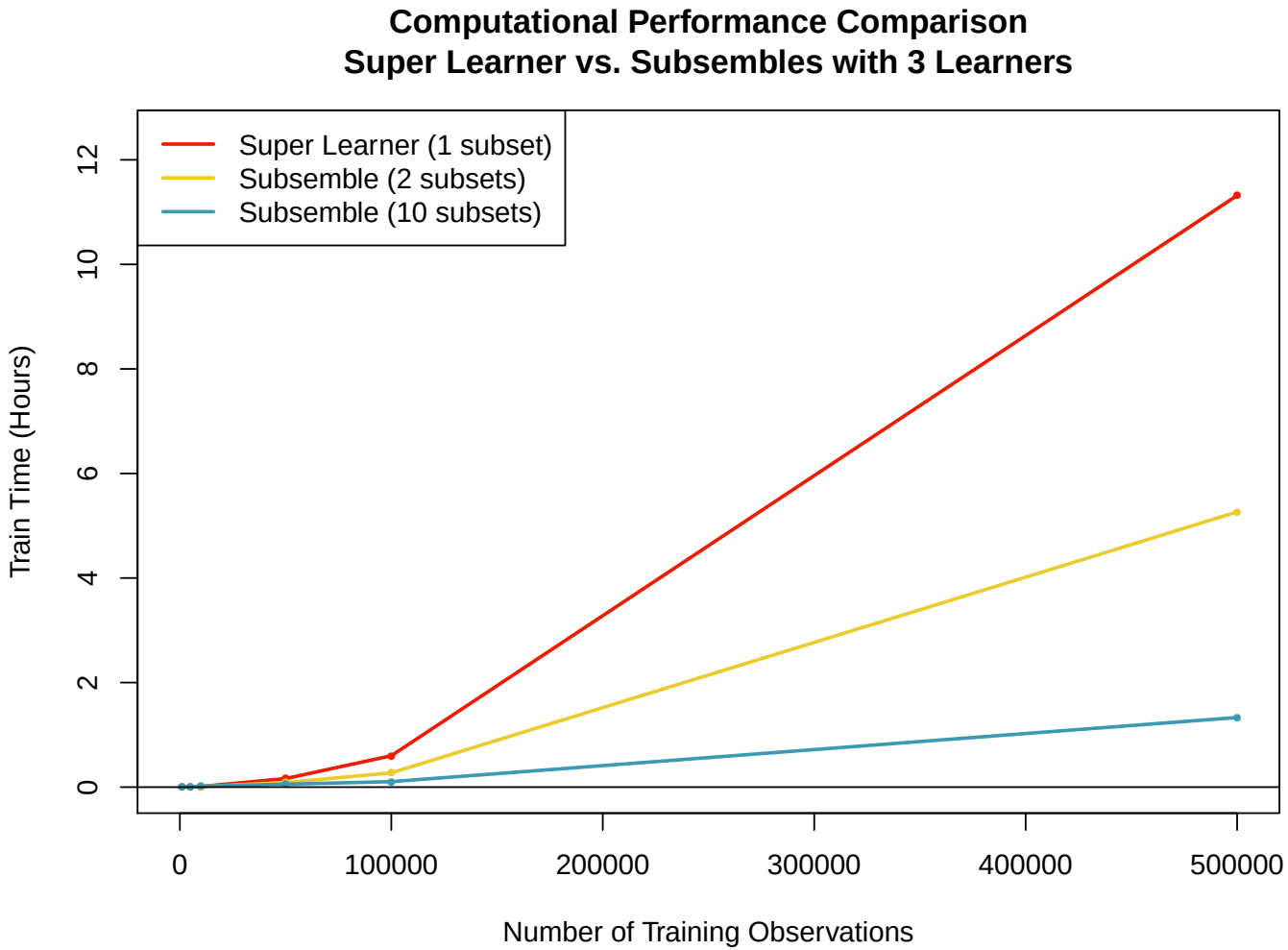
Example

```
fit <- subsemble(x = x, y = y,
                 family = family,
                 learner = learner,
                 metalearner = metalearner,
                 subsets = subsets)

pred <- predict(fit, newx)
```

subsemble benchmarks



subsemble benchmarks



H2O Machine Learning platform

H2O is an open source Java machine learning library.

Algorithms have distributed implementations that work over clusters.

APIs available in:
R, Python, Java, Scala and JSON

H2O Machine Learning platform

Distributed Supervised ML Algorithms available in H2O

- Generalized Linear Model with Elastic Net regularization
- Gradient Boosting Machines (w/ trees)
- Random Forest
- Deep Learning: Multi-Layer Feed-Forward Neural Networks

H2O Machine Learning platform

Other Algorithms available in H2O

- K-means
- Naive-Bayes
- Principal Components Analysis (PCA)
- Cox Proportional Hazards Models

h2o R package

h2o: How to start H2O & load data

Example

```
library(h2o)  # First install from CRAN
localH2O <- h2o.init()

# Data directly into H2O cluster (avoids R)
train <- h2o.importFile(localH2O, path = "train.csv")

# Data into H2O from R data.frame
train <- as.h2o(localH2O, object = df)
```

h2o R package

h2o: How to train & test

Example

```
y <- "Class"
x <- setdiff(names(train), y)

fit <- h2o.deeplearning(x = x, y = y, data = train)

pred <- h2o.predict(fit, test)
```

h2oEnsemble R package

h2oEnsemble R package

- Uses the **h2o** REST API to interact with the H2O cluster.
- Supports binary classification and regression using the H2O supervised learning algorithms as base learners.
- Supports `SL.*` based metalearners and H2O metalearners.
- Parallelized over any size cluster via the H2O base algorithms.

`https://github.com/h2oai/h2o/tree/master/R/ensemble`

h2oEnsemble R package

h2oEnsemble: Set up the super learner ensemble

Example

```
learner <- c("h2o.glm.1",
             "h2o.glm.2",
             "h2o.randomForest.1",
             "h2o.deeplearning.1")

metalearner <- "SL.glm"

family <- "binomial"
```

h2oEnsemble R package

h2oEnsemble: Create base learners

Example

```
h2o.glm.1 <- function(..., alpha = 1.0) {
    h2o.glm.wrapper(..., alpha = alpha))
}

h2o.glm.2 <- function(..., alpha = 0.5) {
    h2o.glm.wrapper(..., alpha = alpha))
}
```

h2oEnsemble R package

h2oEnsemble: How to train & test

Example

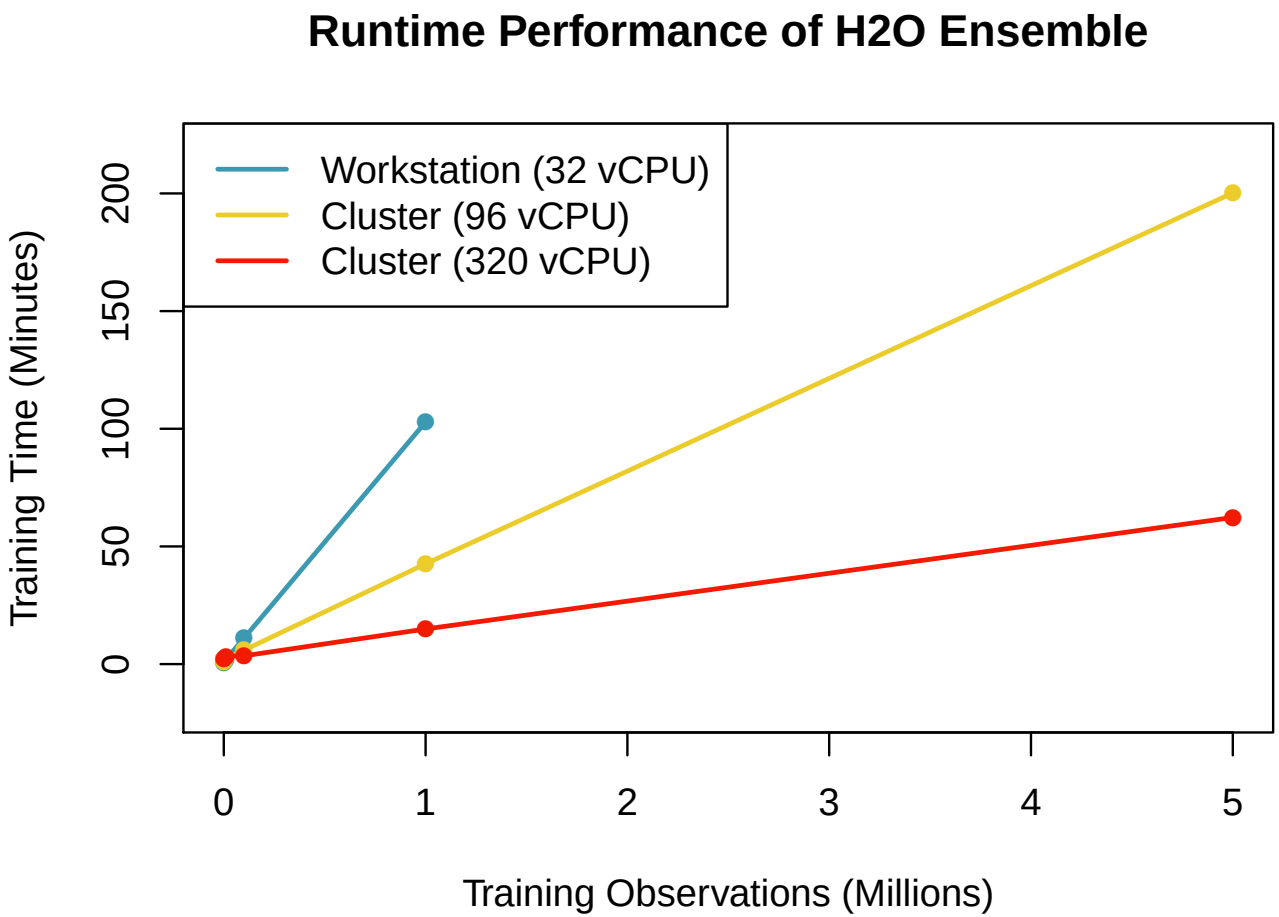
```
fit <- h2o.ensemble(x = x, y = y, data = train,
                    family = family,
                    learner = learner,
                    metalearner = metalearner)

pred <- predict(object = fit, newdata = test)
```

h2oEnsemble R package benchmarks

	RF	DNN-Dropout	DNN-L_2	SL: GLM, NNLS
$n = 1,000$	0.730	0.683	0.660	0.729, 0.730
$n = 10,000$	0.785	0.722	0.707	0.786, 0.788
$n = 100,000$	0.825	0.812	0.809	0.818, 0.819
$n = 1,000,000$	0.823	0.812	0.838	0.841, 0.841
$n = 5,000,000$	0.839	0.817	0.845	0.852, 0.851

Table : Base learner model performance (test set AUC) compared to **h2oEnsemble** model performance performance using 2-fold CV (ensemble results for both GLM and NNLS metalearners).

h2oEnsemble R package benchmarks



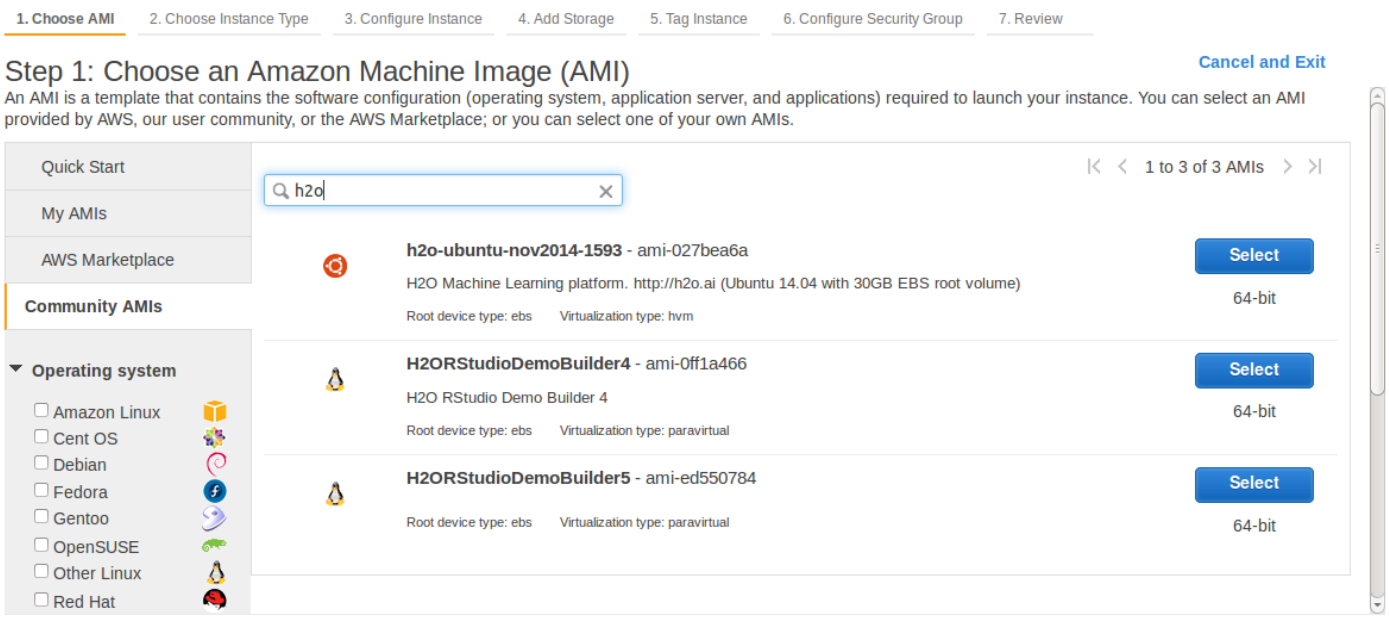
h2oEnsemble R package benchmarks

	Cluster (320)	Cluster (96)	Workstation (32)
$n = 1,000$	2.1 min	1.1 min	0.5 min
$n = 10,000$	3.3 min	2.5 min	2.0 min
$n = 100,000$	3.5 min	5.9 min	11.0 min
$n = 1,000,000$	14.9 min	42.6 min	102.9 min
$n = 5,000,000$	62.3 min	200.2 min	-

Table : Training times (minutes) for **H2O Ensemble** with a 3-learner library using various cluster configurations, including a single workstation with 32 vCPUs. The number of vCPUs for each cluster is noted in parenthesis. Results for $n = 5$ million are not available for the single workstation setting.

Amazon EC2 demo

H2O Ubuntu AMI on Amazon EC2



AMI ID: "ami-027bea6a" in us-east-1 (N. Virginia)

Amazon EC2 demo

h2oEnsemble Demo on Amazon EC2

- Instance Type: c3.8xlarge
  32 vCPUs, 60GB RAM, 2 x 320 GB SSD, 10 Gigabit
- EC2 API: Python boto library scripts
- Data: "Higgs" dataset from UCI ML Data Repository.

`https://github.com/ledell/h2oEnsemble-benchmarks`