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
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.


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


(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.

Super Learner: The setup

1. Define a base learner library of \( L \) learners, \( \Psi^1, ..., \Psi^L \).
2. Specify a metalearning method, \( \Phi \).
3. Partition the training observations into \( V \) folds.
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, $\hat{\Psi}_v^l$, defined as fitting $\Psi^l$ 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: An ensemble method for combining subset-specific algorithm fits”

Stephanie Sapp, Mark J. van der Laan & John Canny

Subsemble: The Setup

1. Define a candidate learner library of $L$ learners, $\psi^{(1)}, \ldots, \psi^{(L)}$, and specify a metalearning method, $\phi$.

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: 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, $\hat{\Psi}_{j,-v}^{(l)}$, 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
Install required R packages:

Example

```r
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**

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

method <- "method.NNLS"

family <- "binomial"
```

@ledell
Intro to Practical Ensemble Learning
April 27, 2015
SuperLearner: How to train & test

Example

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

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

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

metalearner <- "SL.nnls"

family <- "binomial"

subsets <- 4
```
**subsemble** R package

**Example**

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

pred <- predict(fit, newx)
```
Model Performance Comparison
Super Learner vs. Subsembles with 3 Learners

AUC on 100k Test Set

Number of Training Observations

Super Learner (1 subset)
Subsemble (2 subsets)
Subsemble (10 subsets)
Computational Performance Comparison
Super Learner vs. Subsembles with 3 Learners

Super Learner (1 subset)
Subsemble (2 subsets)
Subsemble (10 subsets)
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
Other Algorithms available in H2O

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

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

fit <- h2o.deeplearning(x = x, y = y, data = train)
pred <- h2o.predict(fit, test)
```
**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

**Example**

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

metalearner <- "SL.glm"

family <- "binomial"
```
**h2oEnsemble**: Create base learners

Example

```r
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**

**Example**

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

pred <- predict(object = fit, newdata = test)
```
Table: Base learner model performance (test set AUC) compared to \texttt{h2oEnsemble} model performance using 2-fold CV (ensemble results for both GLM and NNLS metalearners).

<table>
<thead>
<tr>
<th></th>
<th>RF</th>
<th>DNN-Dropout</th>
<th>DNN-$L_2$</th>
<th>SL: GLM, NNLS</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n = 1,000$</td>
<td>0.730</td>
<td>0.683</td>
<td>0.660</td>
<td>0.729, 0.730</td>
</tr>
<tr>
<td>$n = 10,000$</td>
<td>0.785</td>
<td>0.722</td>
<td>0.707</td>
<td>0.786, 0.788</td>
</tr>
<tr>
<td>$n = 100,000$</td>
<td>0.825</td>
<td>0.812</td>
<td>0.809</td>
<td>0.818, 0.819</td>
</tr>
<tr>
<td>$n = 1,000,000$</td>
<td>0.823</td>
<td>0.812</td>
<td>0.838</td>
<td>0.841, 0.841</td>
</tr>
<tr>
<td>$n = 5,000,000$</td>
<td>0.839</td>
<td>0.817</td>
<td>0.845</td>
<td>0.852, 0.851</td>
</tr>
</tbody>
</table>
h2oEnsemble R package benchmarks

Runtime Performance of H2O Ensemble

- Workstation (32 vCPU)
- Cluster (96 vCPU)
- Cluster (320 vCPU)

Training Time (Minutes)

Training Observations (Millions)
### h2oEnsemble R package benchmarks

<table>
<thead>
<tr>
<th>n</th>
<th>Cluster (320)</th>
<th>Cluster (96)</th>
<th>Workstation (32)</th>
</tr>
</thead>
<tbody>
<tr>
<td>n = 1,000</td>
<td>2.1 min</td>
<td>1.1 min</td>
<td>0.5 min</td>
</tr>
<tr>
<td>n = 10,000</td>
<td>3.3 min</td>
<td>2.5 min</td>
<td>2.0 min</td>
</tr>
<tr>
<td>n = 100,000</td>
<td>3.5 min</td>
<td>5.9 min</td>
<td>11.0 min</td>
</tr>
<tr>
<td>n = 1,000,000</td>
<td>14.9 min</td>
<td>42.6 min</td>
<td>102.9 min</td>
</tr>
<tr>
<td>n = 5,000,000</td>
<td>62.3 min</td>
<td>200.2 min</td>
<td>-</td>
</tr>
</tbody>
</table>

**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.
H2O Ubuntu AMI on Amazon EC2

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