Boosting, Bagging, etc.

Based mostly on notes by Greg Ridgeway

David Madigan
Unstable predictors

We can always assume

\[ y = f(x) + \varepsilon, \] where \( E(\varepsilon | x) = 0 \)

Assume that we have a way of constructing a predictor, \( \hat{f}_D(x) \), from a dataset \( D \).

We want to choose the estimator of \( f \) that minimizes \( J \), squared loss for example.

\[ J(\hat{f}, D) = E_{y,x} (y - \hat{f}_D(x))^2 \]
Bias-variance decomposition

If we could average over all possible datasets, let the average prediction be

\[ \bar{f}(x) = \mathbb{E}_D \hat{f}_D(x) \]

The average prediction error over all datasets that we might see is decomposable

\[ E_D J(\hat{f}, D) = \mathbb{E} \mathcal{E}^2 + \mathbb{E}_x (f(x) - \bar{f}(x))^2 + \mathbb{E}_{x,D} (\hat{f}_D(x) - \bar{f}(x))^2 \]

= noise + bias + variance
Bias-variance decomposition (cont.)

\[ E_D J(\hat{f}, D) = E \mathcal{E}^2 + E_x (f(x) - \bar{f}(x))^2 + E_{x,D} (\hat{f}_D(x) - \bar{f}(x))^2 \]

= noise + bias + variance

• The noise cannot be reduced.
• The squared-bias term might be reducible
• The variance term is 0 if we use

\[ \hat{f}_D(x) = \bar{f}(x) \]

But this requires having an infinite number of datasets
Bagging (Bootstrap Aggregating)

Goal: Variance reduction

Method: Create bootstrap replicates of the dataset and fit a model to each. Average the predictions of each model.

Properties:
- Stabilizes “unstable” methods
- Easy to implement, parallelizable
- Theory is not fully explained
Bagging algorithm

1. Create $K$ bootstrap replicates of the dataset.
2. Fit a model to each of the replicates.
3. Average (or vote) the predictions of the $K$ models.

Bootstrapping simulates the stream of infinite datasets in the bias-variance decomposition.
Bagging Example
CART decision boundary
100 bagged trees
Regression results
Squared error loss
Classification results

Misclassification rates

[Bar chart showing misclassification rates for different datasets (Diabetes, Breast, Ionosphere, Heart, Soybean, Glass, Waveform) with two categories: CART and Bagged CART]
Random Forests

“ The key to accuracy is low correlation and bias. To keep bias low, trees are grown to maximum depth.

To keep correlation low, the current version uses this randomization.

1) Each tree is grown on a bootstrap sample of the training set.

2) A number m is specified much smaller than the total number of variables M. At each node, m variables are selected at random out of the M, and the split is the best split on these m variables. ”

(see Random Forests, Machine Learning(2001) 45 5-320)
An important feature is that it carries along an internal test set estimate of the prediction error.

For every tree grown, about one-third of the cases are out-of-bag (out of the bootstrap sample). Abbreviated oob.

Put these oob cases down the corresponding tree and get response estimates for them.

For each case $n$, average or pluralize the response estimates over all time that $n$ was oob to get a test set estimate $\hat{y}_n$ for $y_n$.

Averaging the loss over all $n$ give the test set estimate of prediction error.
<table>
<thead>
<tr>
<th>Data Set</th>
<th>Adaboost</th>
<th>Selection</th>
<th>Forest-RC</th>
<th>One Tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>glass</td>
<td>22.0</td>
<td>24.4</td>
<td>23.5</td>
<td>42.4</td>
</tr>
<tr>
<td>breast cancer</td>
<td>3.2</td>
<td>3.1</td>
<td>2.9</td>
<td>5.8</td>
</tr>
<tr>
<td>diabetes</td>
<td>26.6</td>
<td>23.0</td>
<td>23.1</td>
<td>32.1</td>
</tr>
<tr>
<td>sonar</td>
<td>15.6</td>
<td>13.6</td>
<td>13.8</td>
<td>31.7</td>
</tr>
<tr>
<td>vowel</td>
<td>4.1</td>
<td>3.3</td>
<td>3.3</td>
<td>30.4</td>
</tr>
<tr>
<td>ionosphere</td>
<td>6.4</td>
<td>5.5</td>
<td>5.7</td>
<td>14.2</td>
</tr>
<tr>
<td>vehicle</td>
<td>23.2</td>
<td>23.1</td>
<td>22.8</td>
<td>39.1</td>
</tr>
<tr>
<td>German credit</td>
<td>23.5</td>
<td>22.8</td>
<td>23.8</td>
<td>32.6</td>
</tr>
<tr>
<td>image</td>
<td>1.6</td>
<td>1.6</td>
<td>1.8</td>
<td>6.0</td>
</tr>
<tr>
<td>ecoli</td>
<td>14.8</td>
<td>12.9</td>
<td>12.4</td>
<td>25.3</td>
</tr>
<tr>
<td>votes</td>
<td>4.8</td>
<td>4.1</td>
<td>4.0</td>
<td>8.6</td>
</tr>
<tr>
<td>liver</td>
<td>30.7</td>
<td>27.3</td>
<td>27.2</td>
<td>40.3</td>
</tr>
<tr>
<td>letters</td>
<td>3.4</td>
<td>3.4</td>
<td>4.1</td>
<td>23.8</td>
</tr>
<tr>
<td>sat-images</td>
<td>8.8</td>
<td>9.1</td>
<td>10.2</td>
<td>17.3</td>
</tr>
<tr>
<td>zip-code</td>
<td>6.2</td>
<td>6.2</td>
<td>7.2</td>
<td>22.7</td>
</tr>
<tr>
<td>waveform</td>
<td>17.8</td>
<td>16.0</td>
<td>16.1</td>
<td>33.2</td>
</tr>
<tr>
<td>twonorm</td>
<td>4.9</td>
<td>3.8</td>
<td>3.9</td>
<td>20.9</td>
</tr>
<tr>
<td>threenorm</td>
<td>18.8</td>
<td>16.8</td>
<td>16.9</td>
<td>34.8</td>
</tr>
<tr>
<td>ringnorm</td>
<td>6.9</td>
<td>4.8</td>
<td>4.6</td>
<td>24.6</td>
</tr>
</tbody>
</table>
in R: library(randomForest)

> spam7.rf <- randomForest(yesno ~ ., data=spam7, importance=TRUE)
> print(spam7.rf)

Call:
randomForest(formula = yesno ~ ., data = spam7, importance = TRUE)
  Type of random forest: classification
  Number of trees: 500
No. of variables tried at each split: 2

OOB estimate of error rate: 11.89%
Confusion matrix:
           n   y class.error
         n 2642  146 0.05236729
         y  401 1412 0.22118036
> tuneRF(x=spam7[,,-7], y=spam7$yesno, trace=FALSE)
  -0.0582878 0.05
  -0.01821494 0.05
  mtry  OOBError
  1   1 0.1262769
  2   2 0.1193219
  4   4 0.1214953
>

> importance(spam7.rf)

<table>
<thead>
<tr>
<th></th>
<th>n</th>
<th>y MeanDecreaseAccuracy</th>
<th>MeanDecreaseGini</th>
</tr>
</thead>
<tbody>
<tr>
<td>crl.tot</td>
<td>0.6355890</td>
<td>0.8020131</td>
<td>0.5177274</td>
</tr>
<tr>
<td>dollar</td>
<td>0.6494899</td>
<td>0.7958824</td>
<td>0.5210265</td>
</tr>
<tr>
<td>bang</td>
<td>0.6810718</td>
<td>0.8493533</td>
<td>0.5349533</td>
</tr>
<tr>
<td>money</td>
<td>0.5864884</td>
<td>0.7986897</td>
<td>0.5048771</td>
</tr>
<tr>
<td>n000</td>
<td>0.6500864</td>
<td>0.4718939</td>
<td>0.5107114</td>
</tr>
<tr>
<td>make</td>
<td>0.3840864</td>
<td>0.5809521</td>
<td>0.4117850</td>
</tr>
</tbody>
</table>
Adaptive Bagging

Goal: Bias and variance reduction
Method: Sequentially fit bagged models, where each fits the current residuals
Properties:
• Bias and variance reduction
• No tuning parameters
Adaptive bagging algorithm

1. Fit a bagged regressor to the dataset $D$.
2. Predict “out-of-bag” observations.
3. Fit a new bagged regressor to the bias (error) and repeat.
For a new observation, sum the predictions from each stage.
Regression results
Squared error loss

Bagging
Adaptive bagging
Classification results

Misclassification rates

[Bar chart showing misclassification rates for different datasets such as Breast, Ionosphere, Sonar, Heart, German credit, Votes, Liver, and Diabetes, with bars in blue and purple indicating AdaBoost and Adaptive bagging, respectively.]
Bagging References

• Leo Breiman’s homepage
  www.stat.berkeley.edu/users/breiman/

  Machine Learning, 26:2, 123-140.

• Friedman, J. and P. Hall (1999) “On
  Bagging and Nonlinear Estimation”
  www.stat.stanford.edu/~jhf


Andreas Buja’s home page:

"The Effect of Bagging on Variance, Bias and Mean Squared Error"
A. Buja, W. Stuetzle.
Bootstrap aggregation ("bagging") is a device for reducing the variance of learning algorithms. We give a complete second-order analysis of the effect of bagging on finite sums of U-statistics.

"Smoothing Effects of Bagging"
A. Buja, W. Stuetzle.
An short note on bagging. It relates the von Mises expansion of a bagged statistical functional to the Efron-Stein ANOVA expansion of the unbagged functional to show that the bagged functional is always smooth.
Boosting

Goal: Improve misclassification rates

Method: Sequentially fit models, each more heavily weighting those observations poorly predicted by the previous model

Properties:

• Bias and variance reduction
• Easy to implement
• Theory is not fully (but almost) explained
Generic boosting algorithm

Equally weight the observations \((y, x)_i\)

For \(t\) in 1,\ldots,T

Using the weights, fit a classifier \(f_t(x) \to y\)

Upweight the poorly predicted observations

Downweight the well-predicted observations

Merge \(f_1, \ldots, f_T\) to form the boosted classifier
Real AdaBoost
Schapire & Singer 1998

\[ y_i \in \{-1, 1\}, \; w_i = 1/N \]

For \( t \) in 1, \ldots, \( T \) do

1. Estimate \( P_w(y = 1|x) \).

2. Set \( f_t(x) = \frac{1}{2} \log \frac{\hat{P}_w(y = 1|x)}{\hat{P}_w(y = -1|x)} \)

3. \( w_i \leftarrow w_i \exp(-y_i f_t(x_i)) \) and renormalize

Output the classifier \( F(x) = \text{sign} \left( \sum f_t(x) \right) \)
Boosting Example
After one iteration

CART splits, larger points have great weight
After 3 iterations
After 20 iterations
Decision boundary after 100 iterations
Boosting as optimization


- Let $y \in \{-1, 1\}$, $F(x) \in (-\infty, \infty)$

\[
J(F) = E\left(e^{-yF(x)} \mid x\right)
\]
Criterion

- \( E(e^{-yF(x)}) \) bounds the misclassification rate.
  \[
  I(yF(x) < 0) < e^{-yF(x)}
  \]

- The minimizer of \( E(e^{-yF(x)}) \) coincides with the maximizer of the expected Bernoulli likelihood.

\[
J(F) = E\ell(F) = E \left[ y^* F(x) - \log \left( 1 + e^{F(x)} \right) \right] | x
\]

\[
y^* = \frac{1}{2} (1 + y) \in \{0, 1\}
\]
Optimization step

\[ J(F + f) = E\left(e^{-y(F(x) + f(x))} \mid x\right) \]

- Select \( f \) to minimize \( J \)…

\[ F^{(t+1)} \leftarrow F^{(t)} + \frac{1}{2} \log \frac{E_w[I(y = 1) \mid x]}{1 - E_w[I(y = 1) \mid x]} \]

\[ w(x, y) = e^{-yF^{(t)}(x)} \]
Let $J(F) = E[e^{-yF(x)}]$. Suppose we have a current estimate $F(x)$ and seek an improved estimate $F(x) + cf(x)$. For fixed $c$ (and $x$), we expand $J(F(x) + cf(x))$ to second order about $f(x) = 0$

$$J(F + cf) = E[e^{-y(F(x) + cf(x))}]$$

$$\approx E[e^{-yF(x)}(1 - ycf(x) + c^2 y^2 f(x)^2 / 2)]$$

$$= E[e^{-yF(x)}(1 - ycf(x) + c^2 / 2)]$$

since $y^2 = 1$ and $f(x)^2 = 1$. Minimizing pointwise with respect to $f(x) \in \{-1, 1\}$, we write

$$f(x) = \text{arg min}_f E_w(1 - ycf(x) + c^2 / 2|x) \quad (16)$$

Here the notation $E_w(\cdot|x)$ refers to a weighted conditional expectation, where $w = w(x, y) = e^{-yF(x)}$, and

$$E_w[g(x, y)|x] \overset{\text{def}}{=} \frac{E[w(x, y)g(x, y)|x]}{E[w(x, y)|x]}.$$  

For $c > 0$, minimizing (16) is equivalent to maximizing

$$E_w[yf(x)] \quad (17)$$

The solution is

$$f(x) = \begin{cases} 1 & \text{if } E_w(y|x) = P_w(y = 1|x) - P_w(y = -1|x) > 0 \\ -1 & \text{otherwise} \end{cases} \quad (18)$$
LogitBoost
Friedman, Hastie, Tibshirani [1998]

- Logistic regression

\[ y = \begin{cases} 
1 & \text{with probability } p(x) \\
0 & \text{with probability } 1 - p(x) 
\end{cases} \]

\[ p(x) = \frac{1}{1 + e^{-F(x)}} \]

- Expected log-likelihood of a regressor, \( F(x) \)

\[ \mathbb{E} \ell(F) = \mathbb{E} \left( yF(x) - \log(1 + e^{F(x)}) \mid x \right) \]
Newton steps

\[ J(F + f) = E\left( y(F(x) + f(x)) - \log(1 + e^{F(x) + f(x)}) \mid x \right) \]

- Iterate to optimize expected log-likelihood.

\[ F^{(t+1)}(x) \leftarrow F^{(t)}(x) - \frac{\partial}{\partial f} J(F^{(t)} + f) \bigg|_{f=0} - \frac{\partial^2}{\partial f^2} J(F^{(t)} + f) \bigg|_{f=0} \]
LogitBoost, continued

- Newton steps for Bernoulli likelihood

\[
F(x) \leftarrow F(x) + E_w \left( \frac{y - p(x)}{p(x)(1 - p(x))} \right|_x
\]

\[w(x) = p(x)(1 - p(x))\]

- In practice the \(E_w(\bullet|x)\) can be any regressor - trees, smoothers, etc.

- Trees are adaptive and work well for high dimensional data.
Boosting References

• Rob Schapire’s homepage
  http://www.cs.princeton.edu/~schapire/boost.html


• Jerry Friedman’s homepage
  www.stat.stanford.edu/~jhf