

Chapter 11

Boosting

Xiaogang Su

Department of Statistics
University of Central Florida

Perturb and Combine (P&C)

- Methods have been devised to take advantage of the instability of trees to create models that are more powerful.
- Perturb and combine (P&C) methods generate multiple models by manipulating the distribution of the data or altering the construction method and then averaging the results (Breiman, 1998).
- Any unstable modeling method can be used, but trees are most often chosen because of their speed and flexibility.
- Commonly-Used Methods:
 - Boosting
 - Bagging
 - Random Forest

Variance Reduction by P&C

- The attractiveness of P&C methods is their improved performance over single models.
- Bauer and Kohavi (1999) demonstrated the superiority of P&C methods with extensive experimentation.
- One reason why simple P&C methods give improved performance is variance reduction. If the base models have low bias and high variance, then averaging decreases the variance. In contrast, combining stable models can negatively affect performance

Perturb Methods

- Resample
- Subsample
- Add noise
- Adaptively reweight
- Randomly choose from among the competitor splits

Combine

- An ensemble model is the combination of multiple models.
- The combinations can be formed by
 - Voting on the classifications
 - Weighted voting where some models have more weight
 - Averaging (weighted or unweighted) of the predicted values.
- Ensemble methods are a very active area of research in the field of machine learning and statistics. Many other P&C methods have been devised.

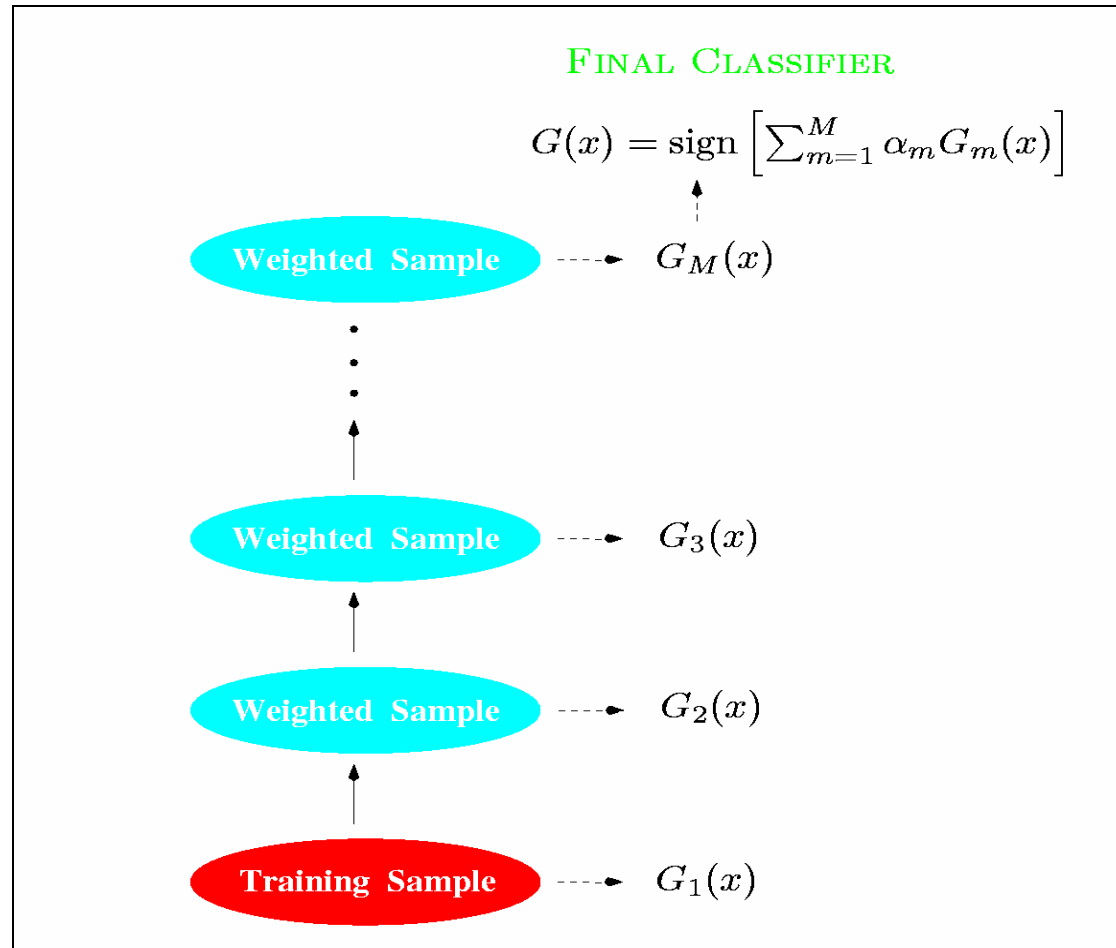
Boosting – A Short History

- In 1996, Freund and Schapire proposed the well-known AdaBoost.M1 algorithm
 - Discrete AdaBoost
 - Real AdaBoost
 - Gentle AdaBoost
- Arcing (adaptive resampling and combining) by Breiman (1998)
 - Arc-x4
- Stochastic Gradient Boosting (SGB) by Friedman (2001, *Annals of Statistics*)

AdaBoost – Basic Idea

- AdaBoost generates a sequentially weighted set of weak base classifiers that are combined to form an overall strong classifier.
- In each step of the sequence, AdaBoost attempts to find an optimal classifier according to the current distribution of weights on the observations.
- If an observation is incorrectly classified using the current distribution of weights, then the observation will receive more weight in the next iteration. On the other hand, correctly classified observations under the current distribution of weights will receive less weight in the next iteration.
- In the final overall model, classifiers that are accurate predictors of the training data receive more weight, whereas, classifiers that are poor predictors receive less weight.

AdaBoost – The Schematic



AdaBoost – The Algorithm

1. Initialize the observation weights $w_i = 1/N$, $i = 1, 2, \dots, N$.
 2. For $m = 1$ to M :
 - (a) Fit a classifier $G_m(x)$ to the training data using weights w_i .
 - (b) Compute
$$\text{err}_m = \frac{\sum_{i=1}^N w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^N w_i}.$$
 - (c) Compute $\alpha_m = \log((1 - \text{err}_m)/\text{err}_m)$.
 - (d) Set $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x_i))]$, $i = 1, 2, \dots, N$.
 3. Output $G(x) = \text{sign} \left[\sum_{m=1}^M \alpha_m G_m(x) \right]$.
-

Arcing

- The Arcing (adaptive resampling and combining) (Arc-x4; Breiman, 1998) is a simplified version of the AdaBoost (adaptive boosting) algorithm of Freund and Schapire (1996).
- It gives similar performance to AdaBoost (Breiman, 1998 and Bauer and Kohavi, 1999).
- Unlike bagging, pruning the individual trees and selecting the optimally-sized tree improves performance (Bauer and Kohavi, 1999) in boosting.

Arcing – The Weights

- At the k -th step a model (decision tree) is fitted using weights for each case. For the i -th case the arc-x4 weights are

$$p(i) = \frac{1 + m(i)^4}{\sum \{1 + m(i)^4\}},$$

where $0 \leq m(i) \leq k$ is the number of times that the i th case is misclassified in the preceding steps.

Arcing – Two Ways of Using the Weights

- The weights are incorporated either by using a weighted analysis or by resampling the data such that the probability that the i th case is selected is $p(i)$. For convenience, the weights can be normalized to frequencies by multiplying by the sample size n (as shown in the following table).
- Bauer and Kohavi (1999) found that resampling performed better than reweighting for arc-x4 but did not change the performance of AdaBoost. AdaBoost uses a different (more complicated) formula for $p(i)$. Both formulas put greater weight on cases that are frequently misclassified.

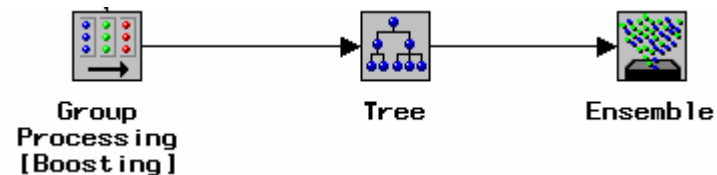
Arcing – A Simple Example by Hand

Table 1: An Illustration of arc- $\times 4$

Case	k=1		k=2		k=3		k=4
	Freq	m	Freq	m	Freq	m	Freq
1	1	1	1.5	1	.5	2	.97
2	1	0	.75	0	.25	0	.06
3	1	1	1.5	2	4.25	3	4.69
4	1	0	.75	1	.5	1	.11
5	1	0	.75	0	.25	0	.06
6	1	0	.75	0	.25	1	.11
Total	$n = 6$		$n = 6$		$n = 6$		$n = 6$

Arcing

- The process is repeated K times and the K models are combined by voting or averaging the posterior probabilities. AdaBoost used weighted voting where models with fewer misclassifications, particularly of the hard-to-classify cases, are given more weight. Breiman (1998) used $K=50$. Bauer and Kohavi (1999) used $K=25$.
- Arcing improves performance to a greater degree than bagging, but the improvement is less consistent (Breiman, 1998 and Bauer and Kohavi, 1999).
- SAS EM Implemented the Arc-x4 idea.



Stochastic Gradient Boosting

- Boosting inherently relies on a gradient descent search for optimizing the underlying loss function to determine both the weights and the learner at each iteration.
- In Stochastic Gradient Boosting (SGB) a random permutation sampling strategy is employed at each iteration to obtain a refined training set.
- The full SGB algorithm with the gradient boosting modification relies on the regularization parameter ν in $[0, 1]$, the so-called learning rate.

SGB – the Algorithm

Algorithm 2 Stochastic Gradient Boosting Algorithm

- 1: Initialize $F(x) := 0$
 - 2: **for** $m = 1$ to M **do**
 - 3: Set $w_i = -\frac{\delta L(y, g)}{\delta g} \Big|_{g=F(x)}$
 - 4: Fit $y = \eta(h_m(x))$ as the base weighted classifier using $|w_i|$, with training sample π_m
 - 5: Compute line search step $\alpha_m = \arg \min_{\alpha} \sum_{i \in \pi_m} L(y_i, F(x) + \alpha \eta(h_m(x)))$
 (in some cases this step may be omitted or $\alpha_m = 1$)
 - 6: Update $F(x) = F(x) + \nu \alpha_m \eta(h_m(x))$
 - 7: **end for**
-

Source: Taken from *ada: an R Package for Stochastic Boosting* by Mark Culp, Kjell Johnson, and George Michailidis.

SGB

- The algorithm in its general form can operate under an arbitrary loss function: the exponential $L(y, f) = e^{-yf}$ and logistic $L(y, f) = \log(1 + e^{-yf})$ loss functions.
- The $\eta(\cdot)$ function specifies the type of boosting: discrete $\eta(x) = \text{sign}(x)$, real $\eta(x) = 0.5 \log(x/(1-x))$, and gentle $\eta(x) = x$.

R Implementation

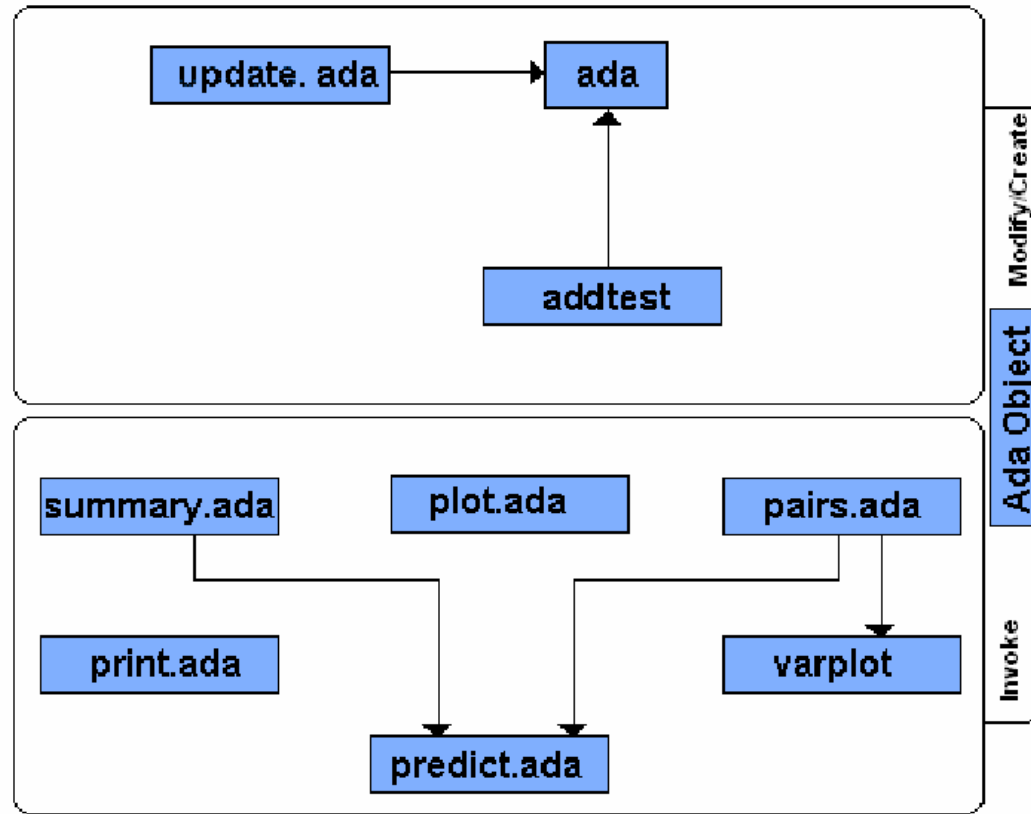
Several Packages Available

1. `gbm` - The `gbm` package offers two versions of boosting for classification (gentle boost under logistic and exponential loss). In addition, it includes squared error, absolute error, Poisson and Cox type loss functions.
2. `mboost` - The `mboost` package has to a large extent similar functionality to the `gbm` package and in addition implements the general gradient boosting framework using regression based learners.
3. `ada` – the one we are going to use.
4. `boost` – Implementation of boosting for gene data.

R `ada` Package

- The `ada` package implements the original AdaBoost algorithm, along with the Gentle and Real AdaBoost variants, using both exponential and logistic loss functions for classification problems.
- In addition, it allows the user to implement regularized versions of these methods by using the learning rate as a tuning parameter, which lead to improved computational performance.
- The base classifiers employed are classification/regression trees and therefore the underlying engine is the `rpart` package.

ada - The Function Flow



- The top section consists of the functions used to create the ada object, while the bottom section are the functions invoked using an initialized ada object.

Appendix: Kappa Statistic

Kappa Statistics: an index which compares the agreement against that which might be expected by chance. Kappa can be thought of as the chance-corrected proportional agreement, and possible values range from +1 (perfect agreement) via 0 (no agreement above that expected by chance) to -1 (complete disagreement).

Hypothetical Example: 29 patients are examined by two independent doctors (see Table). 'Yes' denotes the patient is diagnosed with disease X by a doctor. 'No' denotes the patient is classified as no disease X by a doctor.

		Doctor A		Total
		No	Yes	
Doctor B	No	10 (34.5%)	7 (24.1%)	17 (58.6%)
	Yes	0 (0.0%)	12 (41.4%)	12 (41.4%)
Total		10 (34.5%)	19 (65.5%)	29

$\text{Kappa} = (\text{Observed agreement} - \text{Chance agreement}) / (1 - \text{Chance agreement})$

Observed agreement = $(10 + 12) / 29 = 0.76$

Chance agreement = $0.586 * 0.345 + 0.655 * 0.414 = 0.474$

$\text{Kappa} = (0.76 - 0.474) / (1 - 0.474) = 0.54$