Boosting

Charlotte Wickham

April 27, 2007

Boosting

Idea in the binary classification context ($Y \in \{-1, 1\}$).

- Take a weak classifier (one that does just a bit better than random guessing).
- Fit the classifier to the data to get $G_1(X)$.
- Give more weight to the observations in the training set it gets wrong.
- Re-fit the classifier to the reweighted data.
- Repeat M times to get a sequence of classifiers, $G_m(X)$.
- The predicted value of a new data point is a **weighted** sum of classifiers.

$$G(x) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m G_m(x)\right)$$

Choices:

- How are the observations reweighted?
- How are the classifiers weighted in the prediction?
- What weak classifier to use?

AdaBoost Algorithm

1. Initialize weights, $w_i = 1/N$ for $i = 1, \ldots, N$
2. For $m = 1$ to M
   1. Fit classifier to the training data using weight w_i to get $G_m(x)$.
   2. Calculate the the training set error

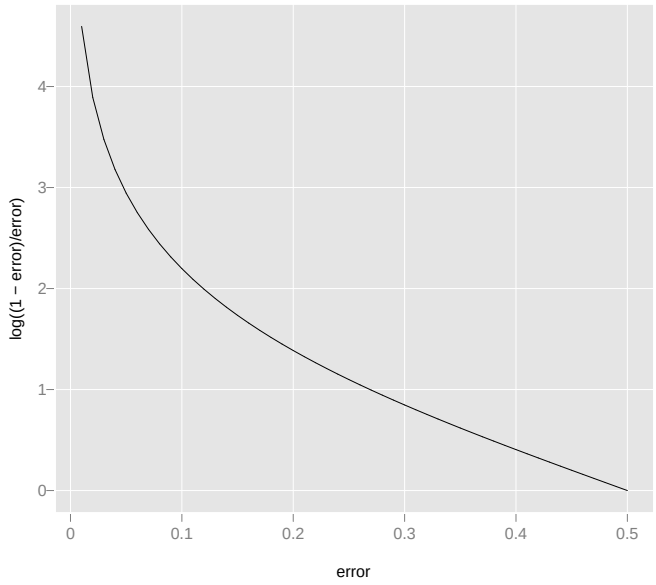
$$\text{error}_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}$$

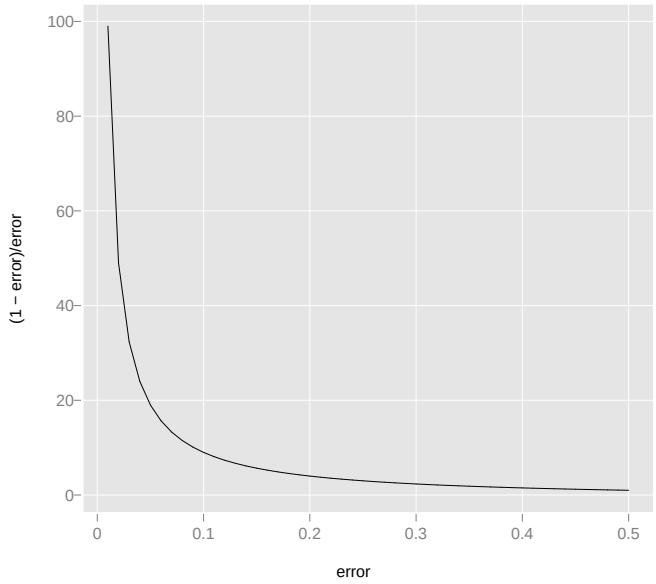
   3. Calculate

$$\alpha_m = \log \frac{1 - \text{error}_m}{\text{error}_m}$$

   4. Set $w_i \leftarrow w_i \exp(\alpha_m I(y_i \neq G_m(x_i)))$ for $i = 1, \ldots, N$
3. Output $G(x) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m G_m(x)\right)$

AdaBoost Algorithm





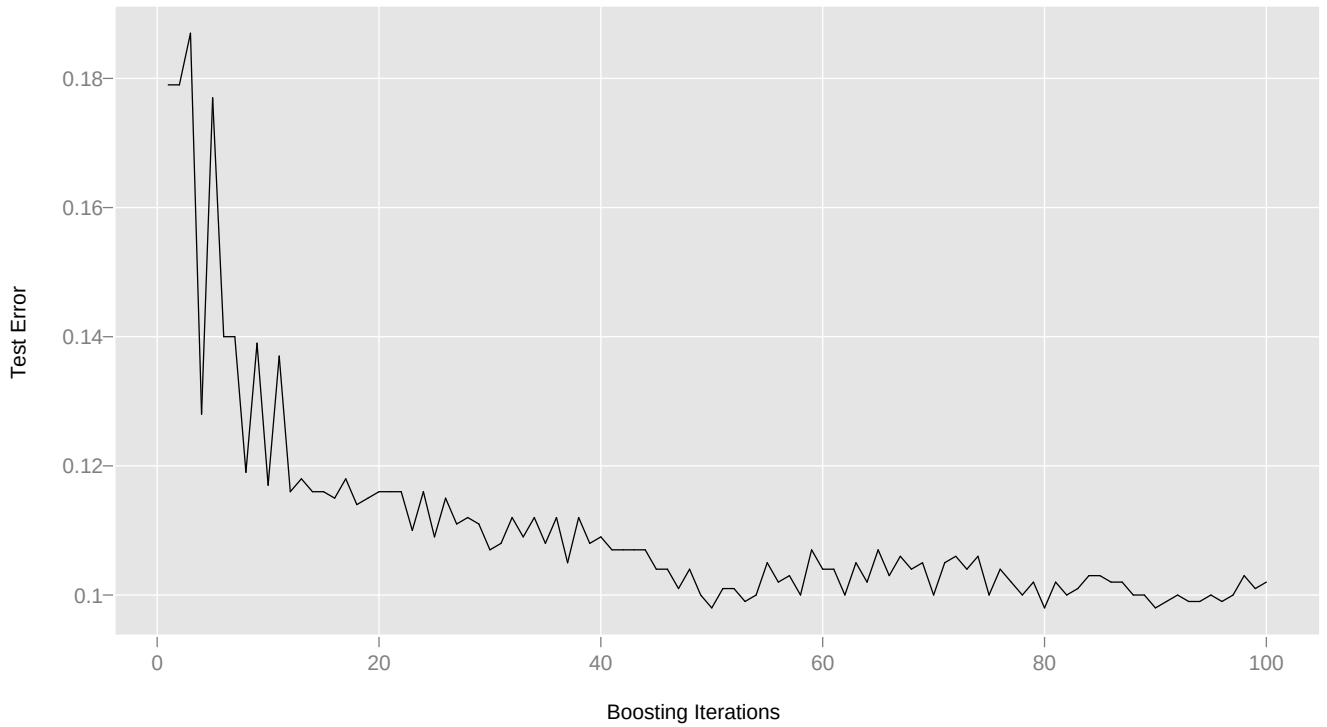
- Gives more weight to classifiers in the sequence with low training error.
- Gives more weight to observations that are misclassified. The better the classifier overall the bigger the weight on the misclassified observations.
- Choice of classifier is free. Often use trees with very few splits. Stump = tree with one split.

Performance on test set

- Trees with one split (stumps)

		Prediction	
		Other	Supernova
Actual	Other	451	49
	Supernova	47	453

Error: 9.6%



Got about 8% from CART and 6% from bagging and 5% from random forests.

- Trees with two splits

		Prediction	
		Other	Supernova
Actual	Other	455	45
	Supernova	45	455

Error: 9.0%

- Trees with three splits

		Prediction	
		Other	Supernova
Actual	Other	455	45
	Supernova	47	453

Error: 9.2%

More general idea

Our output is

$$G(x) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m G_m(x)\right)$$

Imagine we knew $G_m(x)$ and α_m for $m = 1, \ldots, M-1$ and let their contribution to the output be g_{M-1}. Then we could imagine searching for the optimal G_M and α_M that would minimize our error according to some loss function. I.e. find G_M and α_M

$$(G_m, \alpha_M) = \arg\min_{G_m, \alpha_M} \sum_{i=1}^{N} \mathcal{L}\left(y_i, g_{M-1}(x) + \alpha_m G_m(x)\right)$$

More general idea

- We could envisage a stagewise process where we begin with a base model and then iteratively add terms by the minimization presented.
- In fact, AdaBoost is equivalent to this process if you use the loss function,

$$\mathcal{L}(y, G(x)) = \exp(-yG(x)).$$

- Why use this loss?
  - Computationally easy
- There are other possibilities but they require more effort in the minimization step.

Gradient Boosting with trees

We want to solve,

$$\arg\min_{\Theta_m} \sum_{i=1}^{N} \mathcal{L}(y_i, g_{m-1}(x_i) + T(x_i; \Theta_m))$$

where $\Theta_m = \{R_{jm}, \phi_{jm}\}_1^{J_m}$ is the region sets and constants for the tree.

- For complicated loss functions this can be really hard.
- We want a way to approximate this minimisation.
- In general how do we minimise things: gradient descent

Steepest descent

Say we want to find $\hat{G} = \arg\min_G \mathcal{L}(G)$ where $\mathcal{L}(G) = \sum_{i=1}^{N} \mathcal{L}(y_i, G(x_i))$.
At each iteration we take a step in the direction that most minimizes our error. The components of this gradient, h_m are:

$$h_{im} = \left[\frac{\delta \mathcal{L}(y_i, G(x_i))}{\delta G(x_i)} \right]_{G(x_i)=g_{m-1}(x_i)}$$

and the step length, ρ_m, is the solution to

$$\rho_m = \arg\min_{\rho} \mathcal{L}(g_{m-1} - \rho h_m).$$

Then the updated solution is

$$g_m = g_{m-1} - \rho_m h_m$$

Gradient Boosting

So the stepwise procedure outlined earlier is a lot like steepest descent where the new tree is analogous to the components of the negative gradient.

- Problems

$$\arg\min_{\Theta_m} \sum_{i=1}^{N} \mathcal{L}(y_i, g_{m-1}(x_i) + T(x_i; \Theta_m))$$

is hard to calculate for robust losses. We would like to use the gradient descent method instead since,

$$h_{im} = \left[\frac{\delta \mathcal{L}(y_i, G(x_i))}{\delta G(x_i)} \right]_{G(x_i)=g_{m-1}(x_i)}$$

is easy to calculate for most loss functions. However, it is only defined at x_i and eventually we want to generalize to new cases.

Gradient Boosting

Possible resolution

- Induce a tree at the m^{th} iteration that approximates the negative gradient

$$\tilde{\Theta}_m = \arg\min_{\Theta} \sum_{i=1}^{N} (-h_{im} - T(x_i; \Theta))^2$$

- This is just a normal regression tree and there are good algorithms for finding it.
- the constants for each region (γ_{jm}) are then found from

$$\gamma_{jm} = \arg\min_{\gamma_{jm}} \sum_{x_i \in R_{jm}} \mathcal{L}(y_i, g_{m-1}(x_i) + \gamma_{jm})$$

Leads to the MART algorithm

MART for regression

1. Initialize $g_0(x) = \arg\min_\gamma \sum_{i=1}^{N} \mathcal{L}(y_i, \gamma)$
2. for $m = 1$ to M
   1. For $i = 1, 2, \ldots, N$ compute

$$r_{im} = -\left[\frac{\delta \mathcal{L}(y_i, G(x_i))}{\delta G(x_i)}\right]_{G(x_i)=g_{m-1}(x_i)}$$

   2. Fit a regression tree to the target r_{jm} giving terminal regions R_{jm}, $j = 1, 2, \ldots, J_m$.
   3. For $j = 1, 2, \ldots, J_m$ compute

$$\gamma_{jm} = \arg\min_\gamma \sum_{x_i \in R_{jm}} \mathcal{L}(y_i, g_{m-1}(x_i) + \gamma)$$

   4. Update $g_m = g_{m-1} + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$
3. Output $\hat{f}(x) = f_M(x)$

MART

- Extends to classification by repeating steps for each class K.
- Tuning parameters are the size of the trees J_m and the number of iterations M.
- In practice choose $J_m = J$ between 4 and 8.
- Choose M based on cross validation or a separate training set.