Boosting

Note: Unless otherwise noted all references including images are from the required textbook, Machine Learning: A Probabilistic Perspective by Kevin P. Murphy.

Boosting

Boosting is a greedy algorithm for fitting adaptive basis-function models of the form

$$f(\mathbf{x}) = w_0 + \sum_{m=1}^{M} w_m \phi_m(\mathbf{x})$$

where the ϕ_m are generated by an algorithm called a **weak learner** or a **base learner**.

Boosting

The algorithm works by applying the weak learner sequentially to weighted versions of the data, where more weight is given to examples that were misclassified by earlier rounds.

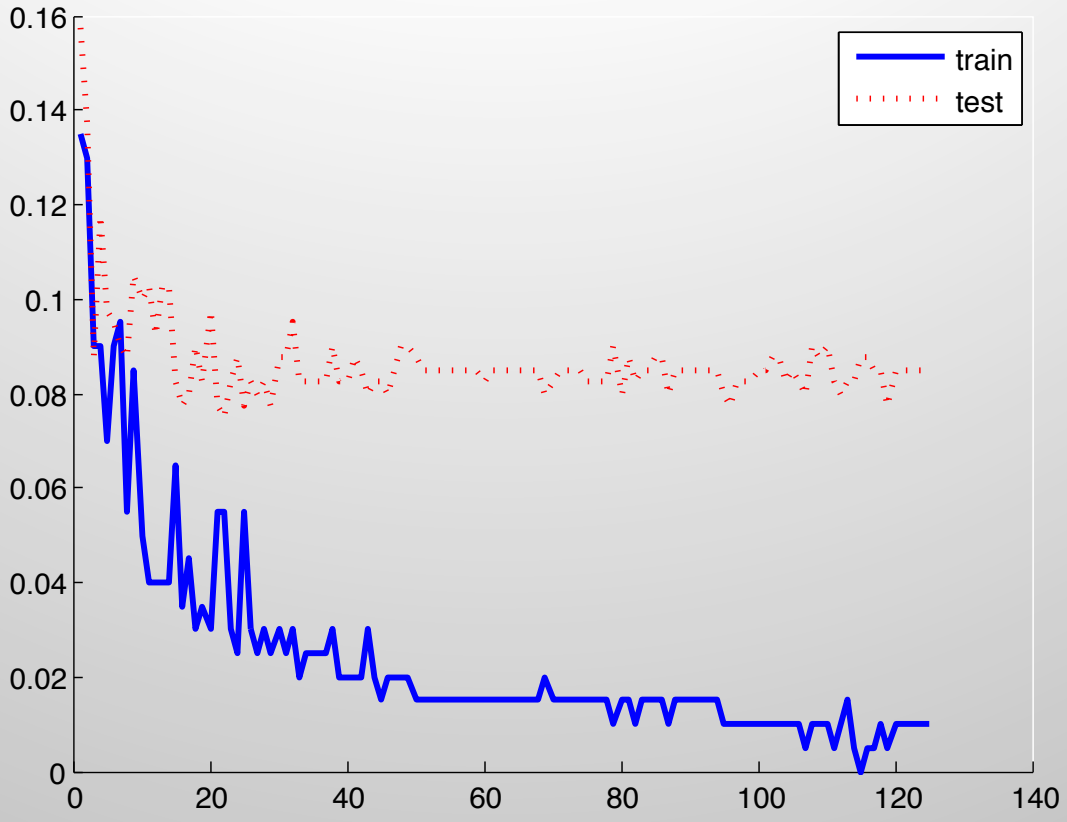
This weak learner can be any classification or regression algorithm, but it is common to use a CART model.

Boosting

Boosting was originally derived in the computational learning theory literature (Schapire 1990; Freund and Schapire 1996), where the focus is binary classification.

In these papers, it was proved that one could boost the performance (on the training set) of any weak learner arbitrarily high, provided the weak learner could always perform slightly better than chance.

Boosting



Performance of adaboost using a decision stump as a weak learner

Boosting

We plot the training and test error for boosted decision stumps on a 2d dataset. We see that the training set error rapidly goes to near zero. What is more surprising is that the test set error continues to decline even after the training set error hasreached zero (although the test set error will eventually go up). Thus boosting is very resistant to overfitting.

Forward Stagewise Additive Modeling

The goal of boosting is to solve the following optimization problem

$$\min_f \sum_{i=1}^{N} L(y_i, f(\mathbf{x}_i))$$

and $L(y, \hat{y})$ is some loss function, and f is assumed to be an Adaptive Basis Function (ABM) model.

Forward Stagewise Additive Modeling

If we use squared error loss, the optimal estimate is given by

$$f^*(\mathbf{x}) = \underset{f(\mathbf{x})}{\operatorname{argmin}} = \mathbb{E}_{y|\mathbf{x}}\left[(Y - f(\mathbf{x}))^2\right] = \mathbb{E}\left[Y|\mathbf{x}\right]$$

This cannot be computed in practice since it requires knowing the true conditional distribution $p(y|x)$. Hence this is sometimes called the population minimizer, where the expectation is interpreted in a frequentist sense.

Forward Stagewise Additive Modeling

Name	Loss	Derivative	f^*	Algorithm
Squared error	$\frac{1}{2}(y_i - f(\mathbf{x}_i))^2$	$y_i - f(\mathbf{x}_i)$	$\mathbb{E}[y\|\mathbf{x}_i]$	L2Boosting
Absolute error	$\|y_i - f(\mathbf{x}_i)\|$	$\text{sgn}(y_i - f(\mathbf{x}_i))$	$\text{median}(y\|\mathbf{x}_i)$	Gradient boosting
Exponential loss	$\exp(-\tilde{y}_i f(\mathbf{x}_i))$	$-\tilde{y}_i \exp(-\tilde{y}_i f(\mathbf{x}_i))$	$\frac{1}{2}\log\frac{\pi_i}{1-\pi_i}$	AdaBoost
Logloss	$\log(1 + e^{-\tilde{y}_i f_i})$	$y_i - \pi_i$	$\frac{1}{2}\log\frac{\pi_i}{1-\pi_i}$	LogitBoost

Some commonly used loss functions, their gradients, their population minimizers f^*, and some algorithms to minimize the loss

Forward Stagewise Additive Modeling

For binary classification, it is common to use logloss, defined by

$$\log(1 + e^{-\tilde{y}_i f_i})$$

In this case, the optimal estimate is given by

$$f^*(\mathbf{x}) = \frac{1}{2} \log \frac{p(\tilde{y} = 1|\mathbf{x})}{p(\tilde{y} = -1|\mathbf{x})}$$

where $y \in \{-1, +1\}$

Forward Stagewise Additive Modeling

An alternative convex upper bound is **exponential loss**, defined by

$$L(\tilde{y}, f) = \exp(-\tilde{y}f)$$

In this case also, the optimal estimate is given by

$$f^*(\mathbf{x}) = \frac{1}{2}\log\frac{p(\tilde{y}=1|\mathbf{x})}{p(\tilde{y}=-1|\mathbf{x})}$$

where $y \in \{-1, +1\}$

Forward Stagewise Additive Modeling

Since finding the optimal f is hard, we shall tackle it sequentially. We initialize by defining

$$f_0(\mathbf{x}) = \arg\min_{\gamma} \sum_{i=1}^{N} L(y_i, f(\mathbf{x}_i; \gamma))$$

For example, if we use squared error, we can set $f_0(\mathbf{x}) = \bar{y}$, and if we use log-loss or exponential loss , we can set $f_0(\mathbf{x}) = \frac{1}{2}\log\frac{\hat{\pi}}{1-\hat{\pi}}$ where $\hat{\pi} = \frac{1}{N}\sum_{i=1}^{N} \mathbb{I}(y_i = 1)$

Forward Stagewise Additive Modeling

Then at iteration m, we compute

$$(\beta_m, \gamma_m) = \underset{\beta,\gamma}{\operatorname{argmin}} \sum_{i=1}^{N} L(y_i, f_{m-1}(\mathbf{x}_i) + \beta\phi(\mathbf{x}_i;\gamma))$$

and then we set

$$f_m(\mathbf{x}) = f_{m-1}(\mathbf{x}) + \beta_m\phi(\mathbf{x};\gamma_m)$$

The key point is that we do not go back and adjust earlier parameters. This is why the method is called **forward stagewise additive modeling**.

Forward Stagewise Additive Modeling

We continue this for a fixed number of iterations M. In fact M is the main tuning parameter of the method. Often we pick it by monitoring the performance on a separate validation set, and then stopping once performance starts to decrease; this is called **early stopping**.

L2boosting

Suppose we used squared error loss. Then at step m the loss has the form

$$L(y_i, f_{m-1}(\mathbf{x}_i) + \beta\phi(\mathbf{x}_i;\gamma)) = (r_{im} - \phi(\mathbf{x}_i;\gamma))^2$$

where $r_{im} \triangleq y_i - f_{m-1}(\mathbf{x}_i)$ is the current residual and $\beta = 1$. Hence we can find the new basis function by using the weak learner to predict r_m. This is called **L2boosting**, or **least squares boosting**.

AdaBoost

Consider a binary classification problem with exponential loss. At step m we have to minimize

$$L_m(\phi) = \sum_{i=1}^{N} \exp[-\tilde{y}_i(f_{m-1}(\mathbf{x}_i) + \beta\phi(\mathbf{x}_i))] = \sum_{i=1}^{N} w_{i,m} \exp(-\beta\tilde{y}_i\phi(\mathbf{x}_i))$$

where $w_{i,m} \triangleq \exp\left(-\tilde{y}_i f_{m-1}(\mathbf{x}_i)\right)$ is a weight applied to data instance i, and $\tilde{y}_i \in \{-1, +1\}$.

AdaBoost

We can rewrite this objective as follows:

$$
\begin{aligned}
L_m &= e^{-\beta} \sum_{\tilde{y}_i = \phi(\mathbf{x}_i)} w_{i,m} + e^{\beta} \sum_{\tilde{y}_i \neq \phi(\mathbf{x}_i)} w_{i,m} \\
&= (e^{\beta} - e^{-\beta}) \sum_{i=1}^{N} w_{i,m} \mathbb{I}(\tilde{y}_i \neq \phi(\mathbf{x}_i)) + e^{-\beta} \sum_{i=1}^{N} w_{i,m}
\end{aligned}
$$

Consequently the optimal function to add is

$$
\phi_m = \operatorname*{argmin}_{\phi} w_{i,m} \mathbb{I}(\tilde{y}_i \neq \phi(\mathbf{x}_i))
$$

AdaBoost

This can be found by applying the weak learner to a weighted version of the dataset, with weights $w_{i,m}$. We solve for β by substituting ϕ_m into L_m

$$\beta_m = \frac{1}{2}\log\frac{1-\mathrm{err}_m}{\mathrm{err}_m}$$

where

$$\mathrm{err}_m = \frac{\sum_{i=1}^{N} w_i \mathbb{I}(\tilde{y}_i \neq \phi_m(\mathbf{x}_i))}{\sum_{i=1}^{N} w_{i,m}}$$

AdaBoost

The overall update becomes

$$f_m(\mathbf{x}) = f_{m-1}(\mathbf{x}) + \beta_m \phi_m(\mathbf{x})$$

With this, the weights at the next iteration become

$$\begin{aligned} w_{i,m+1} &= w_{i,m} e^{-\beta_m \tilde{y}_i \phi_m(\mathbf{x}_i)} \\ &= w_{i,m} e^{\beta_m (2\mathbb{I}(\tilde{y}_i \neq \phi_m(\mathbf{x}_i)) - 1)} \\ &= w_{i,m} e^{2\beta_m \mathbb{I}(\tilde{y}_i \neq \phi_m(\mathbf{x}_i))} e^{-\beta_m} \end{aligned}$$

AdaBoost

Algorithm 16.2: Adaboost.M1, for binary classification with exponential loss

1 $w_i = 1/N$;
2 **for** $m = 1 : M$ **do**
3 Fit a classifier $\phi_m(\mathbf{x})$ to the training set using weights $\mathbf{w}$;
4 Compute $\text{err}_m = \frac{\sum_{i=1}^{N} w_{i,m} \mathbb{I}(\tilde{y}_i \neq \phi_m(\mathbf{x}_i))}{\sum_{i=1}^{N} w_{i,m})}$;
5 Compute $\alpha_m = \log[(1 - \text{err}_m)/\text{err}_m]$;
6 Set $w_i \leftarrow w_i \exp[\alpha_m \mathbb{I}(\tilde{y}_i \neq \phi_m(\mathbf{x}_i))]$;
7 Return $f(\mathbf{x}) = \text{sgn}\left[\sum_{m=1}^{M} \alpha_m \phi_m(\mathbf{x})\right]$;

LogitBoost

The trouble with exponential loss is that it puts a lot of weight on misclassified examples. This makes the method very sensitive to outliers (mislabeled examples).
A natural alternative is to use logloss instead.
This only punishes mistakes linearly.

LogitBoost

It means that we will be able to extract probabilities from the final learned function, using

$$p(y=1|\mathbf{x}) = \frac{e^{f(\mathbf{x})}}{e^{-f(\mathbf{x})} + e^{f(\mathbf{x})}} = \frac{1}{1+e^{-2f(\mathbf{x})}}$$

The goal is to minimize the expected log-loss

$$L_m(\phi) \quad = \quad \sum_{i=1}^{N} \log\left[1 + \exp\left(-2\tilde{y}_i(f_{m-1}(\mathbf{x}) + \phi(\mathbf{x}_i))\right)\right]$$

LogitBoost

Algorithm 16.3: LogitBoost, for binary classification with log-loss

$w_i = 1/N$, $\pi_i = 1/2$;

for $m = 1 : M$ **do**

  Compute the working response $z_i = \frac{y_i^* - \pi_i}{\pi_i(1-\pi_i)}$;

  Compute the weights $w_i = \pi_i(1 - \pi_i)$;

  $\phi_m = \text{argmin}_\phi \sum_{i=1}^N w_i(z_i - \phi(\mathbf{x}_i))^2$;

  Update $f(\mathbf{x}) \leftarrow f(\mathbf{x}) + \frac{1}{2}\phi_m(\mathbf{x})$;

  Compute $\pi_i = 1/(1 + \exp(-2f(\mathbf{x}_i)))$;

Return $f(\mathbf{x}) = \text{sgn}\left[\sum_{m=1}^M \phi_m(\mathbf{x})\right]$;