BOOSTING

Overview

Boosting is an ensemble method, but unlike previous ensemble methods, in boosting, the ensemble classifier is a weighted majority vote, and the elements of the ensemble are determined sequentially.

Assume the labels are -1 and $+1$. The final classifier has the form

$$h_T(x) = \text{sign}\left\{ \sum_{t=1}^{T} \alpha_t f_t(x) \right\}$$

where $f_1, \dots, f_T$ are called base classifiers and $\alpha_1, \dots, \alpha_T > 0$ reflect the confidence in the various base classifiers.

Base Learners

Let $(x_1, y_1), \dots, (x_n, y_n)$ be the training data.
Let $\mathcal{F}$ be a fixed set of classifiers called the

base class.

A base learner for $\mathcal{F}$ is a rule that takes as input a set of weights $w = (w_1, \dots, w_n)$ such that $w_i \geq 0$, $\sum w_i = 1$, and outputs a classifier $f \in \mathcal{F}$ such that the weighted empirical risk

$$e_w(f) := \sum_{i=1}^{n} w_i \, \mathbb{1}_{\{f(x_i) \neq y_i\}}$$

is (approximately) minimized.

Examples of base classes

- Decision trees
- Decision stumps, i.e., decision trees with depth 1
- Radial basis functions, i.e.,

$$f(x) = \pm \operatorname{sign}\{ k_\sigma(x, x_i) + b \}$$

where $b \in \mathbb{R}$ and k_σ is a radial kernel.

The latter two cases highlight the fact that base classifiers can be very simple. In these cases, the weighted empirical risk can be minimized by exhaustive search over $\mathcal{F}$. For decision trees, the base learner may proceed by first resampling the training data (with replacement) according to $w_1, \dots, w_n$, and then apply a standard decision tree learning algorithm to the resampled data.

$\boxed{\text{The Boosting Principle}}$

The basic idea behind boosting is to learn $f_1, \dots, f_T$ sequentially, where f_t is produced by the base learner given a weight vector $w^t = (w_1^t, \dots, w_n^t)$ as input. The weights are updated to place more emphasis on training examples that are harder to classify. Thus, the weight update $w^t \to w^{t+1}$ is conceptually

- If $f_t(x_i) = y_i$, $w_i^{t+1} < w_i^t$ (downweight)
- If $f_t(x_i) \neq y_i$, $w_i^{t+1} > w_i^t$. (upweight)

Adaboost

Adaboost, short for adaptive boosting, was the first concrete algorithm to successfully realize the boosting principle.

Input: $\{(x_i, y_i)\}_{i=1}^{n}$, T, $\mathcal{F}$, base learner

Initialize: $w^1 = \left(\frac{1}{n}, \dots, \frac{1}{n}\right)$

For $t = 1, \dots, T$

- $w^t \longrightarrow \boxed{\text{base learner}} \longrightarrow f_t$
- $r_t := \sum_{i=1}^{n} w_i^t \mathbb{1}_{\{f_t(x_i) \neq y_i\}} = e_{w^t}(f_t)$
- $\alpha_t = \frac{1}{2} \ln\left(\frac{1-r_t}{r_t}\right)$
- $w_i^{t+1} = \dfrac{w_i^t \exp\left(-\alpha_t y_i f_t(x_i)\right)}{Z_t}$ ← normalization constant

End

Output: $h_T(x) = \operatorname{sign}\left\{\sum_{t=1}^{T} \alpha_t f_t(x)\right\}$

Weak Learning

Adaboost is justified by the following result.

Denote $\gamma_t = \frac{1}{2} - r_t$. Note that we may assume $\gamma_t \geq 0 \iff r_t \leq \frac{1}{2}$. If not, just replace f_t with $-f_t$ and note that for any f and w,

$$e_w(f) + e_w(-f) = 1.$$

Theorem | The training error of Adaboost satisfies

$$\frac{1}{n} \sum_{i=1}^{n} \mathbb{1}_{\{h_T(x_i) \neq y_i\}} \leq \exp\left(-2 \sum_{t=1}^{T} \gamma_t^2\right).$$

In particular, if $\gamma_t \geq \gamma > 0$ for all t, then

$$\frac{1}{n} \sum_{i=1}^{n} \mathbb{1}_{\{h_T(x_i) \neq y_i\}} \leq \exp\left(-2T\gamma^2\right).$$

We may interpret $r_t = \frac{1}{2}$ as corresponding to a base classifier f_t that randomly guesses. Thus $\gamma_t \geq \gamma > 0$ means f_t is at least slightly better than random guessing.

If the base learner is guaranteed to satisfy $\gamma_t \geq \gamma > 0$

$\forall t$, it is said to satisfy the weak learning hypothesis.

The theorem says that under the weak learning hypothesis, the Adaboost training error converges to zero exponentially fast. To avoid overfitting, the parameter T should be chosen carefully, e.g., by cross-validation.

For a proof of the theorem, see Mohri et al, Foundations of Machine Learning, 2012.

Remark: If $r_t = 0$, then $\alpha_t = \infty$. In other words, if $\exists$ a classifier in $\mathcal{F}$ that perfectly separates the data, Adaboost says to just use that classifier.

Boosting as Functional Gradient Descent

It turns out that Adaboost can be viewed as an iterative algorithm for minimizing the empirical risk corresponding to the exponential loss. By generalizing the loss, we get different boosting algorithms with different properties.

For a fixed base class $\mathcal{F}$, define

$$\text{span}(\mathcal{F}) = \left\{ \sum_{t=1}^{T} \alpha_t f_t \;\middle|\; T \geq 1,\ \alpha_t \in \mathbb{R},\ f_t \in \mathcal{F} \right\}.$$

Now consider the problem

$$\min_{F \in \text{span}(\mathcal{F})} \frac{1}{n} \sum \mathbb{1}_{\{\text{sign}(F(x_i)) \neq y_i\}}$$

As we have discussed previously, for computational reasons it is desirable to replace the 0/1 loss with a surrogate loss ϕ, leading to

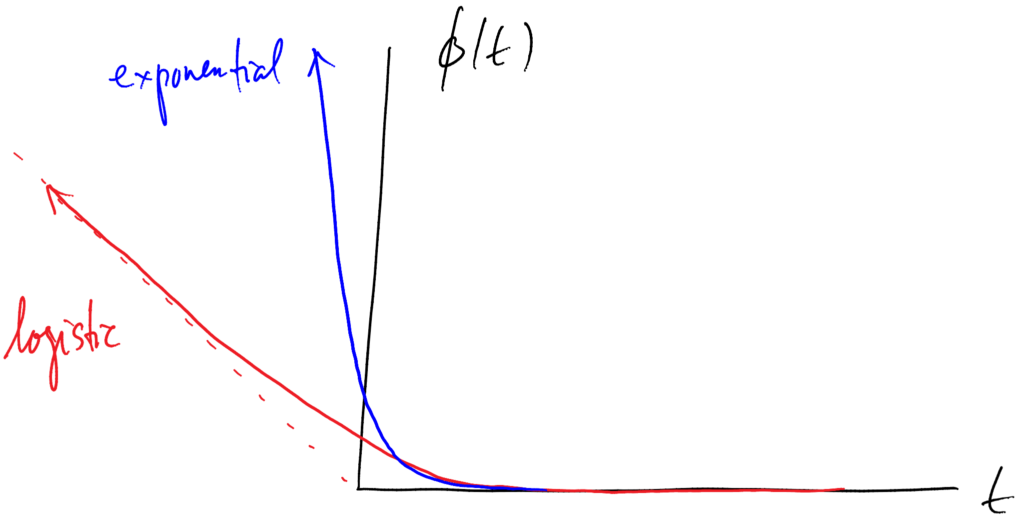
$$\min_{F \in \text{span}(\mathcal{F})} \frac{1}{n} \sum_{i=1}^{n} \phi(y_i F(x_i))$$

Examples of surrogate losses are

- $\phi(t) = \exp(-t)$ exponential loss
- $\phi(t) = \log(1 + \exp(-t))$ logistic loss
- $\phi(t) = \max(0, 1-t)$ hinge loss

- $\phi(t) = \max(0, 1-t)$ hinge loss

We will assume ϕ is differentiable and $\phi' < 0$ everywhere.



To solve this optimization problem we may apply functional gradient descent: This is just gradient descent on a space consisting of functions.

Thus, consider the t^{th} iteration of functional gradient descent. The current iterate is

$$F_{t-1} = \sum_{s=1}^{t-1} \alpha_s f_s .$$

The next iterate will have the form

$$F_{t-1} + \alpha_t f_t .$$

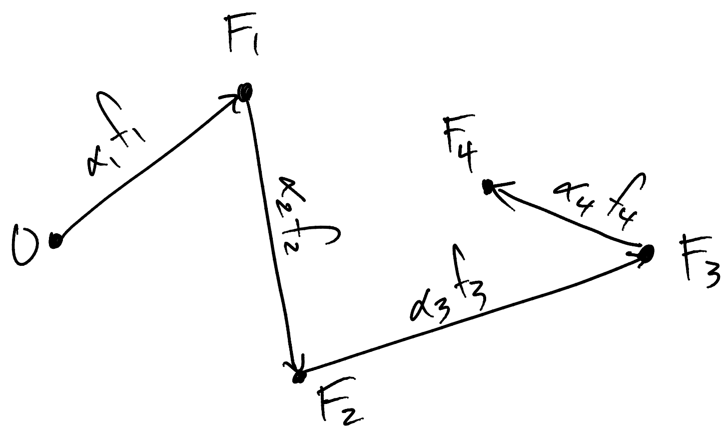
View $\alpha_1, f_1, \ldots, \alpha_{t-1}, f_{t-1}$ as fixed, and define

$$B_t(\alpha, f) = \frac{1}{n}\sum_{i=1}^{n} \phi\left(y_i F_{t-1}(x) + y_i \alpha f(x_i)\right)$$

Then α_t, f_t are chosen by

1) f_t = function $f \in \mathcal{F}$ for which the directional derivative of B_t in the direction f is minimized

2) α_t = stepsize $\alpha > 0$ in the direction f_t for which $B_t(\alpha, f_t)$ is minimized



In detail:

1) $$\left.\frac{\partial B(\alpha, f)}{\partial \alpha}\right|_{\alpha=0} = \frac{1}{n}\sum_{i=1}^{n} y_i f(x_i)\, \phi'(y_i F_{t-1}(x_i))$$

Minimizing this (wrt f) is equivalent to minimizing

$$-\sum^{n} y_i f(x_i) \cdot \frac{\phi'(y_i F_{t-1}(x_i))}{\sum^{n} \cdots}$$

$$-\sum_{i=1}^{n} y_i f(x_i) \cdot \underbrace{\frac{\phi'(y_i F_{t-1}(x_i))}{\sum_{j=1}^{n} \phi'(y_j F_{t-1}(x_j))}}_{\text{call this } w_i^t}$$

since $\phi' < 0$

$$= \sum_{i=1}^{n} w_i^t \mathbb{1}_{\{f(x_i) \neq y_i\}} - \sum_{i=1}^{n} w_i^t \mathbb{1}_{\{f(x_i) = y_i\}}$$

$$= 2\left(\sum_{i=1}^{n} w_i^t \mathbb{1}_{\{f(x_i) \neq y_i\}}\right) - 1$$

Thus, to solve the first step we just apply the base learner.

2) $$\alpha_t = \arg\min_{\alpha} B_t(\alpha, f_t)$$

$$= \arg\min_{\alpha} \frac{1}{n} \sum_{i=1}^{n} \phi\left(y_i F_{t-1}(x_i) + y_i \alpha f_t(x_i)\right)$$

This is just a scalar minimization problem that can be solved numerically, e.g., via Newton's method, if no closed form solution is available.

In summary, here is the generalized boosting algorithm

Input: $\{(x_i, y_i)\}_{i=1}^{n}$, T, $\mathcal{F}$, base learner,

loss function ϕ (differentiable, $\phi' < 0$ everywhere)

Input: $\{(x_i, y_i)\}_{i=1}^{n}$, n, T, base learner,

surrogate loss ϕ (differentiable, $\phi' < 0$ everywhere)

Initialize: $w^1 = \left(\frac{1}{n}, \dots, \frac{1}{n}\right)$, $F_0 = 0$

For $t = 1, \dots, T$

- $w_t \longrightarrow \boxed{\text{base learner}} \longrightarrow f_t$
- $\alpha_t = \arg\min_{\alpha} \frac{1}{n} \sum_{i=1}^{n} \phi\left(y_i F_{t-1}(x_i) + y_i \alpha f_t(x_i)\right)$
- $F_t = F_{t-1} + \alpha_t f_t$
- $$w_i^{t+1} = \frac{\phi'(y_i F_t(x_i))}{\sum_{j=1}^{n} \phi'(y_j F_t(x_j))}$$

End

Output

$$h_T(x) = \operatorname{sign}\left(F_T(x)\right)$$

When $\phi(t) = e^{-t}$, it can easily be verified that the above algorithm specializes to Adaboost. The main advantage of the exponential loss is that

- w_i^t has a nice multiplicative update, which stems from the property $\phi'(a+b) = -\phi'(a)\,\phi(b)$

- α_t has a closed form expression.

For other losses, the updates are less simple but still not difficult to implement. Other losses may have better statical properties. For example, the logistic loss is considerably less sensitive to outliers than the exponential loss.