AdaBoost

Karl Stratos

1 Empirical Loss

Let **pop** denote a population distribution over input-label pairs in $\mathcal{X} \times \{\pm 1\}$. Let $(x_1, y_1) \dots (x_N, y_N) \sim \textbf{pop}$ denote iid samples. Given $f : \mathcal{X} \to \mathbb{R}$, define

$$\hat{\epsilon}(f) := \frac{1}{N} \sum_{i=1}^{N} [[y_i f(x_i) \leq 0]] \qquad \text{(empirical zero-one loss)}$$

$$\hat{l}(f) := \frac{1}{N} \sum_{i=1}^{N} \exp(-y_i f(x_i)) \qquad \text{(empirical exponential loss)}$$

$\hat{l}(f)$ is a convex upper bound on $\hat{\epsilon}(f)$. More generally, given a distribution D over $\{1 \dots N\}$, define

$$\hat{\epsilon}_D(f) := \sum_{i=1}^{N} D(i) \, [[y_i f(x_i) \leq 0]] \qquad \text{(expected empirical zero-one loss)}$$

$$\hat{l}_D(f) := \sum_{i=1}^{N} D(i) \exp(-y_i f(x_i)) \qquad \text{(expected empirical exponential loss)}$$

$$\hat{\gamma}_D(f) := \sum_{i=1}^{N} D(i) y_i f(x_i) \qquad \text{(expected empirical margin, aka. edge)}$$

1.1 For Classifiers

Let $h : \mathcal{X} \to \{\pm 1\}$.

Lemma 1.1. $\hat{\gamma}_D(h) = 1 - 2\hat{\epsilon}_D(h)$

Lemma 1.2. $\hat{l}_D(\alpha h) \geq \hat{\epsilon}_D(h)$ for all $\alpha \geq 0$

Lemma 1.3. Assume $\hat{\epsilon}_D(h) \in (0, 1/2]$. Then

$$\min_{\alpha \geq 0} \hat{l}_D(\alpha h) = 2\sqrt{\hat{\epsilon}_D(h)(1 - \hat{\epsilon}_D(h))} = \sqrt{1 - \hat{\gamma}_D(h)^2}$$

where the unique minimizer is

$$\alpha = \log \sqrt{\frac{1 - \hat{\epsilon}_D(h)}{\hat{\epsilon}_D(h)}} = \log \sqrt{\frac{1 + \hat{\gamma}_D(h)}{1 - \hat{\gamma}_D(h)}} \geq 0$$

In words: we can approximate $\hat{\epsilon}_D(h) > 0$ by tightening the upper bound $\hat{l}_D(\alpha h)$ where $\alpha \geq 0$ controls the margin. The better h is, the larger α. This is useful because $\hat{l}_D$ is easier to optimize than $\hat{\epsilon}_D$.

2 Ensemble

Let $h_1 \dots h_T : \mathcal{X} \to \{\pm 1\}$. An ensemble with weights $\alpha_1 \dots \alpha_T \in \mathbb{R}$ is the function $g_T : \mathcal{X} \to \mathbb{R}$ defined by

$$g_T(x) := \sum_{t=1}^{T} \alpha_t h_t(x)$$

For $t = 1 \dots T$, define a distribution D_t over $\{1 \dots N\}$ by

$$D_t(i) := \frac{\exp(-y_i g_t(x_i))}{\sum_{j=1}^N \exp(-y_j g_t(x_j))} = \frac{\exp(-y_i g_t(x_i))}{N\hat{l}(g_t)}$$

A softmax over negative margins is simply normalized exponential losses. We can pull $(\alpha_1, h_1) \dots (\alpha_{t-1}, h_{t-1})$ from D_t to express it as a function of D_{t-1} and (α_t, h_t) thanks to the linearity of $g_t = \sum_{s=1}^t \alpha_s h_s$. For convenience we define $g_0(x) = 0$ so that $D_0(i) = 1/N$.

Lemma 2.1. For $t \geq 1$,

$$D_t(i) = \frac{D_{t-1}(i)\exp(-y_i\alpha_t h_t(x_i))}{\sum_{j=1}^N D_{t-1}(j)\exp(-y_j\alpha_t h_t(x_j))} = \frac{D_{t-1}(i)\exp(-y_i\alpha_t h_t(x_i))}{\hat{l}_{D_{t-1}}(\alpha_t h_t)} = \frac{\exp(-y_i g_t(x_i))}{N\prod_{s=1}^t \hat{l}_{D_{s-1}}(\alpha_s h_s)}$$

By equating the two expressions of the normalizer, we obtain

$$\hat{l}(g_t) = \prod_{s=1}^t \hat{l}_{D_{s-1}}(\alpha_s h_s) = \left(\prod_{s=1}^{t-1} \hat{l}_{D_{s-1}}(\alpha_s h_s)\right) \hat{l}_{D_{t-1}}(\alpha_t h_t) \tag{1}$$

This suggests a *greedy* strategy: minimize $\hat{l}(g_t)$ over $\alpha_t \in \mathbb{R}$ while holding $\alpha_1 \dots \alpha_{t-1}$ fixed. This is reduced to minimizing $\hat{l}_{D_{t-1}}(\alpha_t h_t)$ where D_{t-1} is constant. From Lemma 1.3 we know the optimal solution

$$\alpha_t = \log \sqrt{\frac{1 - \hat{\epsilon}_{D_{t-1}}(h_t)}{\hat{\epsilon}_{D_{t-1}}(h_t)}} = \log \sqrt{\frac{1 + \hat{\gamma}_{D_{t-1}}(h_t)}{1 - \hat{\gamma}_{D_{t-1}}(h_t)}} \geq 0 \tag{2}$$

assuming $\hat{\epsilon}_{D_{t-1}}(h_t) \in (0, 1/2]$. The greedy selection of $\alpha_1 \dots \alpha_T$ gives

$$\hat{l}(g_T) = \prod_{t=1}^T 2\sqrt{\hat{\epsilon}_{D_{t-1}}(h_t)(1 - \hat{\epsilon}_{D_{t-1}}(h_t))} = \prod_{t=1}^T \sqrt{1 - \hat{\gamma}_{D_{t-1}}(h_t)^2} \tag{3}$$

Plugging Eq. (2) in the recursive expression of D_t in Lemma 2.1, we can verify the following update:

$$D_t(i) = \begin{cases} \frac{1}{2(1-\hat{\epsilon}_{D_{t-1}}(h_t))} D_{t-1}(i) = \frac{1}{1+\hat{\gamma}_{D_{t-1}}(h_t)} D_{t-1}(i) & \text{if } h_t(x_i) = y_i \\ \frac{1}{2\hat{\epsilon}_{D_{t-1}}(h_t)} D_{t-1}(i) = \frac{1}{1-\hat{\gamma}_{D_{t-1}}(h_t)} D_{t-1}(i) & \text{otherwise} \end{cases}$$

The edge formulation, where $\hat{\gamma}_{D_{t-1}}(h_t) \in [0, 1)$, makes it clear that the probility of an example is downweighted if correctly classified but upweighted if misclassified. This is the AdaBoost algorithm. We give it in the edge form: the zero-one loss form is found in Appendix B.

AdaBoost (Freund and Schapire, 1997)
Input:

- $S = \{(x_1, y_1) \dots (x_N, y_N)\}$ where $x_i \in \mathcal{X}$ and $y_i \in \{\pm 1\}$
- Hypothesis class $\mathcal{H}$ of base classifiers
- Number of boosting rounds T

Output: $g_T : \mathcal{X} \to \mathbb{R}$ with $\hat{\epsilon}(g_T) \leq \exp(-\frac{1}{2}\gamma^2 T)$ for some $\gamma \in [0, 1)$ (Theorem 2.2).

1. Set the initial distribution $D_0(i) \leftarrow 1/N$ for $i = 1 \dots N$.
2. For $t = 1 \dots T$,
   (a) Let h_t be a maximizer of $\hat{\gamma}_{D_{t-1}}(h) \in [-1, 1]$ over $\left\{h \in \mathcal{H} : \hat{\gamma}_{D_{t-1}}(h) < 1\right\}$.
   (b) Compute the weight $\alpha_t = \log \sqrt{(1 + \hat{\gamma}_{D_{t-1}}(h_t))/(1 - \hat{\gamma}_{D_{t-1}}(h_t))} \geq 0$.
   (c) Set the next distribution: for $i = 1 \dots N$,

$$D_t(i) = \begin{cases} \frac{1}{1+\hat{\gamma}_{D_{t-1}}(h_t))} D_{t-1}(i) & \text{if } h_t(x_i) = y_i \\ \frac{1}{1-\hat{\gamma}_{D_{t-1}}(h_t))} D_{t-1}(i) & \text{otherwise} \end{cases}$$

3. Return $g_T(x) \leftarrow \sum_{t=1}^T \alpha_t h_t(x)$.

Theorem 2.2. $g_T \leftarrow \textbf{AdaBoost}(S, \mathcal{H}, T)$ satisfies

$$\hat{\epsilon}(g_T) \leq \exp\left(-\frac{1}{2}\hat{\gamma}_{\min}^2 T\right)$$

where $\hat{\gamma}_{\min} := \min_{t=1}^{T} \hat{\gamma}_{D_{t-1}}(h_t)$ is the minimum edge among the T base classifiers trained in AdaBoost.

Proof.

$$\hat{\epsilon}(g_T) \leq \hat{l}(g_T) = \prod_{t=1}^{T} \sqrt{1 - \hat{\gamma}_{D_{t-1}}(h_t)^2} \leq \prod_{t=1}^{T} \exp\left(\frac{1}{2}\hat{\gamma}_{D_{t-1}}(h_t)^2\right) \leq \exp\left(\frac{1}{2}\hat{\gamma}_{\min}^2 T\right)$$

where the first equality is from Eq. (3). □

AdaBoost is agnostic to the choice of h_t (Step 2a): any base classifier satisfying $\hat{\gamma}_{D_{t-1}}(h_t) \in [0, 1)$ can be used, and Theorem 2.2 holds. But it is natural to explicitly optimize the edge in each iteration, hence the algorithm is given in this form. Below, we give an interpretation of this optimization as taking a gradient step in coordinate descent.

2.1 Degenerate cases

AdaBoost typically assumes that h_t is a "weak" classifier: better than random $\hat{\gamma}_{D_{t-1}}(h_t) > 0$, but imperfect $\hat{\gamma}_{D_{t-1}}(h_t) < 1$. However, if

- $\hat{\gamma}_{D_{t-1}}(h_t) = 0$: This implies $\alpha_t = 0$, and furthermore $D_t = D_{t-1}$, thus more iterations will not change the final ensemble and the algorithm should terminate. Intuitively, we have exhausted $\mathcal{H}$.
- $\hat{\gamma}_{D_{t-1}}(h_t) = 1$: If this happens, the algorithm is undefined. But this implies that h_t is perfect and there is no need for more ensembling. We can either stop and return the current ensemble, or continue ensembling by introducing some noise (e.g., subsample data).

2.2 Interpretations

We motivated AdaBoost as a greedy sequential minimization of $\hat{l}(g_T) = \prod_{t=1}^{T} \hat{l}_{D_{t-1}}(\alpha_t h_t)$ (Eq. (1)), which upper bounds $\hat{\epsilon}(g_T)$. AdaBoost can also be derived as an adversarial step-wise calibration of D_t: select α_t so that

$$\begin{aligned}\hat{\epsilon}_{D_t}(h_t) = \sum_{i=1}^{N} [[h_t(x_i) \neq y_i]]\, D_t(i) &= \frac{\sum_{i=1}^{N} [[h_t(x_i) \neq y_i]]\, D_{t-1}(i) \exp(-y_i \alpha_t h_t(x_i))}{\sum_{j=1}^{N} D_{t-1}(j) \exp(-y_j \alpha_t h_t(x_j))} \\ &= \frac{\hat{\epsilon}_{D_{t-1}}(h_t) \exp(\alpha_t)}{\hat{\epsilon}_{D_{t-1}}(h_t) \exp(\alpha_t) + (1 - \hat{\epsilon}_{D_{t-1}}(h_t)) \exp(-\alpha_t)}\end{aligned}$$

is equal to $1/2$. Solving for α_t yields the same solution as Eq. (2). More generally, AdaBoost can be seen a coordinate descent on $\hat{\epsilon}(g)$ over all ensembles g. For simplicity assume a finite hypothesis class $\mathcal{H} = \{h_1 \dots h_H\}$ and write $\mathcal{H}(x) := (h_1(x) \dots h_H(x)) \in \{\pm 1\}^H$. Then any ensemble can be written as

$$\langle \alpha, \mathcal{H}(x) \rangle = \sum_{k=1}^{H} \alpha_k h_k(x)$$

where $\alpha \in \mathbb{R}^H$. Thus the goal is simplified to finding α, which implicitly finds base classifiers. Let U be a convex upper bound on the zero-one loss: $U(z) \geq [[z \leq 0]]$ where z is the margin. This gives a convex loss of $\langle \alpha, \mathcal{H} \rangle$

$$\hat{l}_U(\alpha) := \frac{1}{N} \sum_{i=1}^{N} U(y_i \langle \alpha, \mathcal{H}(x_i) \rangle)$$

which upper bounds $\hat{\epsilon}(\langle \alpha, \mathcal{H} \rangle)$. By minimizing $\hat{l}_U(\alpha)$ over $\alpha \in \mathbb{R}^H$, we implicitly minimize $\hat{\epsilon}(g)$ over all ensembles g. We do coordinate descent: at each step, find the coordinate $k \in \{1 \dots H\}$ with a largest decrease in the loss (equivalently, steepest descent with an l_1-norm constraint) and take a step in that coordinate by $\alpha + \eta e_k$ where $\eta \in \mathbb{R}$ is an optimal step size.

AdaBoost is a special case with the exponential loss $U(z) = \exp(-z)$. The main idea is that by initializing $\alpha = 0_H$, each step corresponds to adding a single classifier in $\mathcal{H}$. The coordinate and step size coincide with the classifier and its weight selected in AdaBoost. This is due to the recursive property of the exponential derivative,

$$\frac{\partial}{\partial \alpha_k} \hat{l}_U(\alpha) = \frac{1}{N} \sum_{i=1}^{N} (-y_i h_k(x_i)) \exp(-y_i \langle \alpha, \mathcal{H}(x_i) \rangle)$$

The partial derivative "selects" h_k while keeping the exponential loss, which can be normalized to give an expression in expected zero-one loss. The partial derivative of $\hat{l}_U(\alpha + \eta e_k)$ (which is convex in η) with respect to η is similar.

CoordinateDescent
Input:

- $S = \{(x_1, y_1) \dots (x_N, y_N)\}$ where $x_i \in \mathcal{X}$ and $y_i \in \{\pm 1\}$
- Finite hypothesis class $\mathcal{H} = \{h_1 \dots h_H\}$, where we write $\mathcal{H}(x) := (h_1(x) \dots h_H(x)) \in \{\pm 1\}^H$
- Convex upper bound $U(z) \geq [[z \leq 0]]$, which approximates $\hat{\epsilon}(g)$ over all possible ensembles g by

$$\hat{l}_U(\alpha) := \frac{1}{N} \sum_{i=1}^{N} U(y_i \langle \alpha, \mathcal{H}(x_i) \rangle) \geq \hat{\epsilon}\left(\langle \alpha, \mathcal{H} \rangle\right)$$

- Number of gradient steps T

Output: Estimation of $\arg\min_{\alpha \in \mathbb{R}^H} \hat{l}_U(\alpha)$

1. $\alpha^{(0)} \leftarrow 0_H$
2. For $t = 1 \dots T$, compute

$$k_t \in \underset{k \in \{1 \dots H\}}{\arg\min} \; [\nabla \hat{l}_U(\alpha^{(t-1)})]_k \qquad \eta_t \in \underset{\eta \in \mathbb{R}}{\arg\min} \; \hat{l}_U(\alpha^{(t-1)} + \eta e_{k_t})$$

   and set $\alpha^{(t)} \leftarrow \alpha^{(t-1)} + \eta_t e_{k_t}$ where e_{k_t} is the k_t-th basis vector.
3. Return $\alpha^{(T)}$.

Theorem 2.3. Let $\mathcal{H}$ be finite and $g_T \leftarrow \textbf{AdaBoost}(S, \mathcal{H}, T)$. There exists an output

$$\alpha^{(T)} \leftarrow \textbf{CoordinateDescent}(S, \mathcal{H}, \exp(-z), T)$$

(accounting for ties in optimization) such that $g_T = \left\langle \alpha^{(T)}, \mathcal{H} \right\rangle$.

This interpretation generalizes AdaBoost to other convex surrogates of the zero-one loss (e.g., hinge or logistic). It also hints at a deeper connection between gradient descent and ensemble learning. Namely, taking gradient steps in a *function space* is equivalent to taking an ensemble of functions. This motivates gradient boosting (Appendix E); **CoordinateDescent** can be seen as a special case of gradient boosting.

3 Decision Stumps

A popular version of AdaBoost assumes the input space $\mathcal{X} = \mathbb{R}^d$ and the hypothesis class of **decision stumps**:

$$\mathcal{H}_{\text{stumps}} := \{x \mapsto b \times \textbf{Ind}(x_r > \tau) : \; r \in \{1 \dots d\}, b \in \{\pm 1\}, \tau \in \mathbb{R}\}$$

where $\textbf{Ind}(A)$ is 1 if A is true and -1 otherwise. Without loss of generality we assume that each dimension is sorted, so that $[x_1]_r \leq \dots \leq [x_N]_r$ for $r = 1 \dots d$ (we can implement this assumption by iterating through examples in a presorted list for each r). Under this assumption, we define for each $r = 1 \dots d$ and $i = 2 \dots N$,

$$\tau_i^{(r)} := \frac{[x_{i-1}]_r + [x_i]_r}{2}$$

If $[x_{i-1}]_r < [x_i]_r$ then $\tau_i^{(r)}$ is a threshold in dimension r that partitions examples into $x_1 \dots x_r$ and $x_{r+1} \dots x_N$.

Expressiveness. A decision stump is a (restricted) linear classifier and cannot realize nonlinear labelings. This works in our favor since we are unlikely to run into the degenerate case of perfect edge $\hat{\gamma}_D(h) = 1$. Specifically, given N samples with 2^N possible labelings, $\mathcal{H}_{\text{stumps}}$ can realize at most $2dN$ labelings. For instance, if $d = 1$ and $N = 3$ so that the inputs are scalars $x_1 \le x_2 \le x_3$, there exists no stump that can realize $(1, -1, 1)$ or $(-1, 1, -1)$, so only 6 out of 8 possible labelings are realized. Note that even fewer labelings would be realized if there are duplicate input values (e.g., $x_1 = x_2$) since we cannot find a threshold. If we have another dimension in which the inputs are ordered differently (e.g., $x_3 \le x_1 \le x_2$), the missing labelings may be realized. But since each dimension realizes at most $2N$ labelings and many labelings are duplicates across dimensions, $\mathcal{H}_{\text{stumps}}$ can realize at most $2dN$ labelings.

Learning. There are at most dN dimension-wise linear separations of N examples. Since $\mathcal{H}_{\text{stumps}}$ can only induce labelings based on these separations, we can consider any stumps corresponding to these separations to do an exhaustive search over $\mathcal{H}_{\text{stumps}}$. In particular, we use stumps with $\tau_i^{(r)}$ as thresholds. If we calculate the edge value for each threshold it would take $O(N^2 d)$ time. Below we give a single-sweep approach with a linear runtime $O(Nd)$ as described by Kégl (2009).

DecisionStump
Input:

- $S = \{(x_1, y_1) \dots (x_N, y_N)\}$ where $x_i \in \mathbb{R}^d$ and $y_i \in \{\pm 1\}$; $[x_1]_r \le \dots \le [x_N]_r$ for $r = 1 \dots d$; $\tau_i^{(r)} := ([x_{i-1}]_r + [x_i]_r)/2$
- Distribution D over $\{1 \dots N\}$

Output: $h^* \in \arg\max_{h \in \mathcal{H}_{\text{stumps}}} \hat{\gamma}_D(h)$

1. $\gamma^* \leftarrow \sum_{i=1}^N D(i) y_i$
2. For $r = 1 \dots d$:
   (a) $\gamma \leftarrow \sum_{i=1}^N D(i) y_i$
   (b) For $j = 2 \dots N$ such that $[x_{j-1}]_r < [x_j]_r$: # Can we use r for thresholding?
      i. $\gamma \leftarrow \gamma - 2D(j-1) y_{j-1}$
      ii. If $|\gamma| > |\gamma^*|$, set $\gamma^* \leftarrow \gamma$, $r^* \leftarrow r$, and $\tau^* \leftarrow \tau_j^{(r)}$.
3. If $\gamma^* = \sum_{i=1}^N D(i) y_i$, return a constant classifier $x \mapsto \text{sign}(\gamma^*)$. Otherwise, return

$$h(x) \leftarrow \text{sign}(\gamma^*) \times \mathbf{Ind}(x_{r^*} > \tau^*)$$

We maintain the best edge value γ^* over all dN linear separations. In each dimension, we start from the edge $\sum_{i=1}^N D(i) y_i$ of the constant classifier $x \mapsto 1$ and subtract $2D(j-1) y_{j-1}$ for $j = 1 \dots N$. Note that the j-th value is $\sum_{i=j+1}^N D(i) y_i - \sum_{i=1}^j D(i) y_i$ (i.e., the edge of a stump that labels $(-1, -1, \dots, 1, 1)$ with j negative ones). Multiplying the value by its sign makes it nonnegative, implying that the sign is the parameter b of the underlying decision stump. We examine the absolute value of the edge to pick whichever side that gives the highest value.

Decision trees. A decision stump is a special case of decision tree with 2 leaves: $\mathcal{H}_{\text{stumps}} = \mathcal{H}_{\text{trees}(2)}$ (Appendix D). Unlike a decision stump, a decision tree (with more leaves) is nonlinear and can induce $O(2^N)$ labelings on S. However, exact learning is intractable and requires heuristics.

4 Generalization

The VC dimension of $\mathcal{H}_{\text{stumps}}$ is 2 (i.e., the maximum number of points that can be shattered by a decision stump is two). Let $\mathcal{H}_{\text{stumps}^T} = \{\sum_{t=1}^T \alpha_t h_t : \alpha_t \ge 0, h_t \in \mathcal{H}_{\text{stumps}}\}$. It is intuitively clear that the VC dimension of $\mathcal{H}_{\text{stumps}^T}$ is 2^T. A standard application of Hoeffding's inequality gives the following.

Theorem 4.1. Draw $S = \{(x_1, y_1) \dots (x_N, y_N)\} \sim \mathbf{pop}^N$ and $g_T \leftarrow \mathbf{AdaBoost}(S, \mathcal{H}_{\text{stumps}}, T)$. Then with high probability,

$$\Pr_{(x,y) \sim \mathbf{pop}} (y g_T(x) \le 0) \le \hat{\epsilon}(g_T) + O\left(\sqrt{\frac{T}{N}}\right)$$

$\hat{\epsilon}(g_T)$ can be further bounded using Theorem 2.2. The bound becomes looser as T increases (due to the increased complexity of $\mathcal{H}_{\text{stumps}^T}$ and the danger of overfitting). In contrast, it is observed empirically that g_T generalizes better as T goes up—even after $\hat{\epsilon}(g_T) = 0$. This motivated researchers to find a better generalization statement based on the margin (Theorem 1, Schapire *et al.* (1998)).

Theorem 4.2. Draw $S = \{(x_1, y_1) \dots (x_N, y_N)\} \sim \mathbf{pop}^N$ and $g_T \leftarrow \mathbf{AdaBoost}(S, \mathcal{H}_{\text{stumps}}, T)$. Then with high probability,

$$\Pr_{(x,y)\sim\mathbf{pop}}(yg_T(x) \leq 0) \leq \frac{1}{N}\sum_{i=1}^{N}[[y_i g_T(x_i) \leq 0.1]] + O\left(\sqrt{\frac{1}{N}}\right)$$

The first term on the RHS is the empirical probility that the ensemble has a small margin on S (0.1 arbitrarily picked). Intuitively, this becomes smaller as T goes up because AdaBoost focuses on hard examples ($\approx$ support vectors) to increase the margin, even after the training error becomes zero. Since the second term is free of T, the bound becomes tighter with more rounds of boosting.

References

Freund, Y. and Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, **55**(1), 119–139.

Kégl, B. (2009). Introduction to adaboost.

Schapire, R. E., Freund, Y., Bartlett, P., Lee, W. S., *et al.* (1998). Boosting the margin: A new explanation for the effectiveness of voting methods. *The annals of statistics*, **26**(5), 1651–1686.

A Proofs

Proof of Lemma 1.1. We have $y_i h(x_i) = 1 - 2[[y_i h(x_i) \leq 0]]$. Thus

$$\hat{\gamma}_D(h) = \sum_{i=1}^{N} D(i)(1 - 2[[y_i h(x_i) \leq 0]]) = 1 - 2\hat{\epsilon}_D(h)$$

■

Proof of Lemma 1.2. If $\alpha = 0$ the statement holds trivially since $\hat{l}_D(\alpha h) = 1 \geq \hat{\epsilon}_D(h)$. If $\alpha > 0$,

$$\hat{l}_D(\alpha h) = \sum_{i=1}^{N} D(i) \exp(-y_i \alpha h(x_i)) \geq \sum_{i=1}^{N} D(i) \left[[y_i \alpha h(x_i) \leq 0]\right] = \sum_{i=1}^{N} D(i) \left[[y_i h(x_i) \leq 0]\right] = \hat{\epsilon}_D(h)$$

■

Proof of Lemma 1.3. For any $\epsilon \in (0, 1/2]$, consider the objective $J_\epsilon : \mathbb{R} \to \mathbb{R}$ defined by

$$J_\epsilon(\alpha) = \epsilon \exp(\alpha) + (1 - \epsilon) \exp(-\alpha)$$

The first and second derivatives are

$$J'_\epsilon(\alpha) = \epsilon \exp(\alpha) - (1 - \epsilon) \exp(-\alpha)$$
$$J''_\epsilon(\alpha) = \epsilon \exp(\alpha) + (1 - \epsilon) \exp(-\alpha)$$

$J''_\epsilon(\alpha) > 0$ for all $\alpha \in \mathbb{R}$, thus it is sufficient to find a stationary point to find the unique optimal solution. Setting $J'_\epsilon(\alpha) = 0$ gives

$$\begin{aligned} \epsilon \exp(\alpha) = (1 - \epsilon) \exp(-\alpha) \quad &\Leftrightarrow \quad \log \epsilon + \alpha = \log(1 - \epsilon) - \alpha \\ &\Leftrightarrow \quad \alpha = \log \sqrt{\frac{1 - \epsilon}{\epsilon}} \end{aligned}$$

This is nonnegative because $(1 - \epsilon)/\epsilon \geq 1$. The optimal value is

$$J_\epsilon\left(\log \sqrt{\frac{1 - \epsilon}{\epsilon}}\right) = \epsilon \exp\left(\log \sqrt{\frac{1 - \epsilon}{\epsilon}}\right) + (1 - \epsilon) \exp\left(\log \sqrt{\frac{\epsilon}{1 - \epsilon}}\right) = 2\sqrt{\epsilon(1 - \epsilon)}$$

Now we view $\hat{l}_D(\alpha h)$ as a function of $\alpha \in \mathbb{R}$, where

$$\begin{aligned} \hat{l}_D(\alpha h) &= \sum_{i=1}^{N} D(i) \exp(-y_i \alpha h(x_i)) \\ &= \sum_{i=1}^{N} D(i) \left[[h(x_i) \neq y_i]\right] \exp(\alpha) + \sum_{i=1}^{N} D(i) \left[[h(x_i) = y_i]\right] \exp(-\alpha) \\ &= \hat{\epsilon}_D(h) \exp(\alpha) + (1 - \hat{\epsilon}_D(h)) \exp(-\alpha) \end{aligned}$$

Thus $\alpha = \log \sqrt{(1 - \hat{\epsilon}_D(h))/\hat{\epsilon}_D(h)} \geq 0$ is the unique minimizer and $2\sqrt{\hat{\epsilon}_D(h)(1 - \hat{\epsilon}_D(h))}$ is the minimum. Plugging $\hat{\epsilon}_D(h) = 1/2(1 - \hat{\gamma}_D(h))$ (Lemma 1.1) in the minimizer we also have $\alpha = \log \sqrt{(1 + \hat{\gamma}_D(h))/(1 - \hat{\gamma}_D(h))}$. To get the expression of the minimum in the edge, we use the algebraic fact $4z(1 - z) = 1 - (1 - 4z + 4z^2) = 1 - (1 - 2z)^2$ for any $z \in \mathbb{R}$. Then

$$2\sqrt{\hat{\epsilon}_D(h)(1 - \hat{\epsilon}_D(h))} = \sqrt{4\hat{\epsilon}_D(h)(1 - \hat{\epsilon}_D(h))} = \sqrt{1 - (1 - 2\hat{\epsilon}_D(h))^2} = \sqrt{1 - \hat{\gamma}_D(h)^2}$$

■

Proof of Lemma 2.1.

$$\begin{aligned}
D_t(i) &= \frac{\exp(-y_i g_t(x_i))}{\sum_{j=1}^N \exp(-y_j g_t(x_j))} \\
&= \frac{\exp(-y_i g_{t-1}(x_i))\exp(-y_i \alpha_t h_t(x_i))}{\sum_{j=1}^N \exp(-y_j g_{t-1}(x_j))\exp(-y_j \alpha_t h_t(x_j))} \\
&= \frac{(\exp(-y_i g_{t-1}(x_i))/C)\exp(-y_i \alpha_t h_t(x_i))}{\sum_{j=1}^N (\exp(-y_j g_{t-1}(x_j))/C)\exp(-y_j \alpha_t h_t(x_j))} \\
&= \frac{D_{t-1}(i)\exp(-y_i \alpha_t h_t(x_i))}{\sum_{j=1}^N D_{t-1}(j)\exp(-y_j \alpha_t h_t(x_j))}
\end{aligned}$$

where we define the constant $C = \sum_{k=1}^N \exp(-y_k g_{t-1}(x_k))$. This proves the first equality. The second equality holds by the definition of expected weighted exponential loss. The third equality then holds inductively. ■

Proof of Theorem 2.3. Define $\bar{D}_t(i) := \exp\left(-y_i \left\langle \alpha^{(t)}, \mathcal{H}(x_i)\right\rangle\right)/M_t$ where $M_t := \sum_{j=1}^N \exp\left(-y_j \left\langle \alpha^{(t)}, \mathcal{H}(x_j)\right\rangle\right)$. We can assume by the technical lemmas A.1 and A.2 that

$$k_t \in \arg\min_{k=1}^{H} \hat{\epsilon}_{\bar{D}_{t-1}}(h_k) \qquad\qquad \eta_t = \log\sqrt{\frac{1-\hat{\epsilon}_{\bar{D}_{t-1}}(h_k)}{\hat{\epsilon}_{\bar{D}_{t-1}}(h_k)}}$$

$\bar{D}_0$ is uniform since $\alpha^{(0)} = 0_H$, hence $\bar{D}_0 = D_0$. Then $h_{k_1} \in \arg\min_{h\in\mathcal{H}} \hat{\epsilon}_{D_0}(h)$ and $\left\langle \alpha^{(1)}, \mathcal{H}(x)\right\rangle = \langle \eta_1 e_{k_1}, \mathcal{H}(x)\rangle = \eta_1 h_{k_1}$ is the same as the step-1 ensemble in **AdaBoost**. At step $t > 1$, assume $\left\langle \alpha^{(t-1)}, \mathcal{H}(x)\right\rangle$ is the same as the step-$(t-1)$ ensemble in **AdaBoost**. Then $\bar{D}_{t-1} = D_{t-1}$, so $h_{k_t} \in \arg\min_{h\in\mathcal{H}} \hat{\epsilon}_{D_{t-1}}(h)$ and $\left\langle \alpha^{(t)}, \mathcal{H}(x)\right\rangle = \left\langle \alpha^{(t-1)}, \mathcal{H}(x)\right\rangle + \eta_t h_{k_t}$ is the same as the step-t ensemble in **AdaBoost**, ■

Lemma A.1.

$$[\nabla \hat{l}_U(\alpha^{(t-1)})]_k = \frac{M_t}{N}\left(2\hat{\epsilon}_{\bar{D}_{t-1}}(h_k) - 1\right)$$

Proof.

$$\begin{aligned}
[\nabla \hat{l}_U(\alpha^{(t-1)})]_k &= \frac{\partial}{\partial \alpha_k}\frac{1}{N}\sum_{i=1}^N \exp\left(-y_i\left\langle \alpha^{(t-1)}, \mathcal{H}(x_i)\right\rangle\right) \\
&= -\frac{1}{N}\sum_{i=1}^N y_i h_k(x_i)\exp\left(-y_i\left\langle \alpha^{(t-1)}, \mathcal{H}(x_i)\right\rangle\right) \\
&= -\frac{M_{t-1}}{N}\sum_{i=1}^N y_i h_k(x_i)\bar{D}_{t-1}(i) \\
&= \frac{M_{t-1}}{N}\left(\hat{\epsilon}_{\bar{D}_{t-1}}(h_k) - (1-\hat{\epsilon}_{\bar{D}_{t-1}}(h_k))\right) \\
&= \frac{M_{t-1}}{N}\left(2\hat{\epsilon}_{\bar{D}_{t-1}}(h_k) - 1\right)
\end{aligned}$$

□

Lemma A.2.

$$\log\sqrt{\frac{1-\hat{\epsilon}_{\bar{D}_{t-1}}(h_k)}{\hat{\epsilon}_{\bar{D}_{t-1}}(h_k)}} \in \arg\min_{\eta\in\mathbb{R}} \hat{l}_U(\alpha^{(t-1)} + \eta e_k)$$

Proof.

$$\begin{aligned}\frac{\partial \hat{l}_U(\alpha^{(t-1)}+\eta e_k)}{\partial \eta} &= \frac{\partial}{\partial \eta}\frac{1}{N}\sum_{i=1}^{N}\exp\left(-y_i\left\langle \alpha^{(t-1)}, \mathcal{H}(x_i)\right\rangle - \eta y_i h_k(x_i)\right)\\ &= -\frac{1}{N}\sum_{i=1}^{N} y_i h_k(x_i)\exp\left(-y_i\left\langle \alpha^{(t-1)}, \mathcal{H}(x_i)\right\rangle\right)\exp\left(-\eta y_i h_k(x_i)\right)\\ &= -\frac{M_{t-1}}{N}\sum_{i=1}^{N} y_i h_k(x_i)\bar{D}_{t-1}(i)\exp\left(-\eta y_i h_k(x_i)\right)\end{aligned}$$

$\hat{l}_U(\alpha^{(t-1)}+\eta e_k)$ is a composition of a convex (by premise) and a linear function in η, thus convex. We set the derivative to zero to find a minimizer:

$$\begin{aligned}-\frac{M_{t-1}}{N}\sum_{i=1}^{N} y_i h_k(x_i)\bar{D}_{t-1}(i)\exp\left(-\eta y_i h_k(x_i)\right) = 0 \quad &\Rightarrow \quad \sum_{i=1}^{N} y_i h_k(x_i)\bar{D}_{t-1}(i)\exp\left(-\eta y_i h_k(x_i)\right) = 0\\ &\Leftrightarrow \quad -\hat{\epsilon}_{\bar{D}_{t-1}}(h_k)\exp(\eta) + (1-\hat{\epsilon}_{\bar{D}_{t-1}}(h_k))\exp(\eta) = 0\end{aligned}$$

Solving for η yields $\eta = \log\sqrt{(1-\hat{\epsilon}_{\bar{D}_{t-1}}(h_k))/\hat{\epsilon}_{\bar{D}_{t-1}}(h_k)}$ assuming $\hat{\epsilon}_{\bar{D}_{t-1}}(h_k) \neq 0$. □

B Zero-One Loss Form of AdaBoost

AdaBoost (zero-one loss)
Input:

- $S = \{(x_1, y_1)\ldots(x_N, y_N)\}$ where $x_i \in \mathcal{X}$ and $y_i \in \{\pm 1\}$
- Hypothesis class $\mathcal{H}$ of base classifiers
- Number of boosting rounds T

Output: $g_T : \mathcal{X} \to \mathbb{R}$ with $\hat{\epsilon}(g_T) \leq \exp(-\frac{1}{2}\gamma^2 T)$ for some $\gamma \in [0, 1)$ (Theorem 2.2).

1. Set the initial distribution $D_0(i) \leftarrow 1/N$ for $i = 1\ldots N$.
2. For $t = 1\ldots T$,
   (a) Let h_t be a minimizer of $\hat{\epsilon}_{D_{t-1}}(h) \in [0, 1]$ over $\left\{h \in \mathcal{H} :\ \hat{\epsilon}_{D_{t-1}}(h) > 0\right\}$.
   (b) Compute the weight $\alpha_t = \log\sqrt{(1-\hat{\epsilon}_{D_{t-1}}(h_t))/\hat{\epsilon}_{D_{t-1}}(h_t)} \geq 0$.
   (c) Set the next distribution: for $i = 1\ldots N$,

$$D_t(i) = \begin{cases} \frac{1}{2(1-\hat{\epsilon}_{D_{t-1}}(h_t))}D_{t-1}(i) & \text{if } h_t(x_i) = y_i \\ \frac{1}{2\hat{\epsilon}_{D_{t-1}}(h_t)}D_{t-1}(i) & \text{otherwise}\end{cases}$$

3. Return $g_T(x) \leftarrow \sum_{t=1}^{T}\alpha_t h_t(x)$.

C Gini Impurity

Given any set of N labeled examples S and a full-support distribution D over their indices, the **Gini impurity** is defined as $2p(1-p)$ where $p = \sum_{i:y_i=1}^{N} D(i)$ is the probability of drawing label 1 from S under D. This is minimized at 0 if all examples are labeled as either 1 or -1, and maximized at $1/2$ if $p = 1/2$. A split is optimal if the expected Giny impurity of the resulting partition is the smallest. We can derive a $O(Nd)$-time algorithm to find an optimal split, again assuming that each dimension is sorted:

MinimizeGiniImpurity
Input:

- $S = \{(x_1, y_1) \dots (x_N, y_N)\}$ where $N \geq 2$, $x_i \in \mathbb{R}^d$, and $y_i \in \{\pm 1\}$; $[x_1]_r \leq \dots \leq [x_N]_r$ for $r = 1 \dots d$; $\tau_i^{(r)} := ([x_{i-1}]_r + [x_i]_r)/2$
- Distribution D over $\{1 \dots N\}$

Output: split (r, τ) with minimum Gini impurity, or fail if there is no split

1. $\gamma^* \leftarrow \infty$
2. $D_{+1} \leftarrow \sum_{j=1}^N [[y_j = 1]] \, D(j)$
3. For $r = 1 \dots d$:
   (a) $p \leftarrow 0$ # Left label-1 probability
   (b) $p' \leftarrow D_{+1}$ # Right label-1 probability
   (c) $\beta \leftarrow 0$ # Left probability
   (d) For $j = 2 \dots N$ such that $[x_{j-1}]_r < [x_j]_r$: # Can we use r for thresholding?
      i. $p \leftarrow p + [[y_{j-1} = 1]] \, D(j-1)$
      ii. $p' \leftarrow p' - [[y_{j-1} = 1]] \, D(j-1)$
      iii. $\beta \leftarrow \beta + D(j-1)$
      iv. $\gamma \leftarrow \beta 2p(1-p) + (1-\beta)2p'(1-p')$
      v. If $|\gamma| < |\gamma^*|$, set $\gamma^* \leftarrow \gamma$, $r^* \leftarrow r$, and $\tau^* \leftarrow \tau_j^{(r)}$.
4. If $\gamma^* = \infty$, fail. Otherwise, return (r^*, τ^*).

D Decision Trees

Let $\mathcal{X} = \mathbb{R}^d$. We say $R \subseteq \mathbb{R}^d$ is a **hyperrectangle** if there exist $a_R, b_R \in (\mathbb{R} \cup \{\pm\infty\})^d$ such that $R = \left\{x \in \mathbb{R}^d : \, a_R < x \leq b_R\right\}$ where the inequalities are element-wise. We say $\mathcal{R} = \{R_1 \dots R_M\}$ is a **hyperrectangle partition** if $R_1 \dots R_M$ are hyperrectangles such that $\cup_{R \in \mathcal{R}} R = \mathbb{R}^d$ and $R_i \cap R_j = \varnothing$ for all $i \neq j$. For $x \in \mathbb{R}^d$, we write $\mathcal{R}(x) \in \{1 \dots M\}$ to denote the unique hyperrectangle in $\mathcal{R}$ that x belongs to. A **decision tree** is a mapping $x \mapsto \pi(\mathcal{R}(x))$ where $\mathcal{R}$ is a hyperrectangle partition and $\pi : \{1 \dots M\} \to \{\pm 1\}$ is a region-labeling. It is called a decision tree because it can be expressed as a binary tree with M leaves. Let ν denote a node object. If internal, it is equipped with a dimension $r \in \{1 \dots d\}$ and a threshold $\tau \in \mathbb{R}$ as well as left and right child nodes; if leaf, it is equipped with $y \in \{\pm\}$. Given any $(\mathcal{R}, \pi)$, by definition there is binary tree with a root node ν_{root} such that $\pi(\mathcal{R}(x)) = \textbf{Traverse}(x, \nu_{\text{root}})$ where

Traverse(x, ν)

1. If ν is a leaf node, return $\nu.y$.
2. Otherwise,
   (a) If $[x]_{\nu.r} > \nu.\tau$, return **Traverse**(x, ν.left)
   (b) If $[x]_{\nu.r} \leq \nu.\tau$, return **Traverse**($x$, ν.right)

Let $\mathcal{H}_{\text{trees}(M)}$ denote the hypothesis class of decision trees with M leaves. We would like to minimize a loss function over $\mathcal{H}_{\text{trees}(M)}$ but an exhaustive search is intractable unless M is small (e.g., $M = 2$ in **DecisionStump**): the number of labelings is exponential in N since a decision tree is highly nonlinear (e.g., it can fit the XOR mapping with $M = 4$ regions). Thus we adopt a top-down greedy heuristic. We assume a single-dimension splitting algorithm $\mathcal{A}$ that maps any data-distribution pair (S', D') to a best dimension $r \in \{1 \dots d\}$ and threshold $\tau \in \mathbb{R}$ according to some metric.

BuildTree
Input: $S = \{(x_1, y_1) \dots (x_N, y_N)\}$, distribution D over $\{1 \dots N\}$, number of splits $P \leq \lfloor \log N \rfloor$, single-dimension splitting algoirthm $\mathcal{A}$
Output: root node ν_{root} of a binary tree with 2^P leaves

1. Initialize ν_{root} and $q \leftarrow \text{queue}([(\nu_{\text{root}}, S, D)])$
2. For P times or until q is empty,
   (a) $(\nu, S', D') \leftarrow q.\text{pop}()$; if the labels in S' are pure, go to Step 2.
   (b) $(\nu.r, \nu.\tau) \leftarrow \mathcal{A}(S', D')$
   (c) Partition S' into S'_1, S'_2 by thresholding dimension $\nu.r$ using $\nu.\tau$.
   (d) Compute distributions D'_1, D'_2 over S'_1, S'_2 by renormalizing D'.
   (e) Initialize $\nu.\text{left}$ and push $(\nu.\text{left}, S'_1, D'_1)$ onto q.
   (f) Initialize $\nu.\text{right}$ and push $(\nu.\text{right}, S'_2, D'_2)$ onto q.
3. Return ν_{root}.

This is a heuristic with no optimality guarantee. For example, if we use **DecisionStump** (which does find an optimal stump) as our choice of $\mathcal{A}$, the resulting *tree* is generally suboptimal for $P \geq 2$: that is, there may be a different tree $h \in \mathcal{H}_{\text{trees}(2^P)}$ that achieves a larger edge. One popular approach is to grow a full tree (i.e., $P = \lfloor \log N \rfloor$) using a "label purity" splitting metric (e.g., Gini impurity, Appendix C) and then prune the full tree to minimize the misclassification rate of the leaf nodes. Decision trees can be used as base classifiers of AdaBoost, regardless of the specifics of how they are learned.

E Gradient Boosting

Let $\mathcal{F}$ denote the set of all functions $f : \mathcal{X} \to \mathbb{R}$. This is a vector space because functions are closed under (element-wise) addition and scalar multiplication. We may assume an inner product $\langle \cdot, \cdot \rangle : \mathcal{F} \times \mathcal{F} \to \mathbb{R}$, with the norm $||f|| := \sqrt{\langle f, f \rangle}$, for certain subspaces like square-integrable or RKHS.[1] We will assume that $\mathcal{F}$ is an RKSH.

A **functional** is a mapping $E : \mathcal{F} \to \mathbb{R}$. We say E is differentiable at $f \in \mathcal{F}$ if there is a *function* $\nabla E(f) \in \mathcal{F}$ such that

$$\lim_{||g|| \to 0} \frac{||E(f+g) - (E(f) + \langle \nabla E(f), g \rangle)||}{||g||} = 0$$

Thus $E(f) + \langle \nabla E(f), g \rangle$ is a linear approximation of E around f. We call $\nabla E(f) \in \mathcal{F}$ the **(functional) gradient** of E at f. We have the following results from functional analysis:

$$\nabla(aE_1 + bE_2)(f) = a\nabla E_1(f) + b\nabla E_2(f) \qquad \text{(linearity of functional differentiation)}$$
$$\nabla(g \circ E)(f) = g'(E(f))\nabla E(f) \qquad \text{(chain rule for any differentiable } g : \mathbb{R} \to \mathbb{R})$$

Some examples:

- $f(x)$, the evaluation functional for $x \in \mathcal{X}$. The gradient is $K(\cdot, x)$ since
$$(f+g)(x) = f(x) + g(x) = f(x) + \langle K(\cdot, x), g \rangle$$
- $||f||^2$, the squared-norm functional. The gradient is $2f$ since $||f+g||^2 = ||f||^2 + \langle 2f, g \rangle + ||g||^2$.
- $\widehat{L}_{\text{sq}}(f) := \frac{1}{2}\sum_{i=1}^N (y_i - f(x_i))^2$, the squared-error functional for $(x_1, y_1) \dots (x_N, y_N) \in \mathcal{X} \times \mathbb{R}$. By the linearity of differentiation and the chain rule,
$$\nabla \widehat{L}_{\text{sq}}(f) = \frac{1}{2}\sum_{i=1}^N \nabla (y_i - f(x_i))^2 = -\sum_{i=1}^N (y_i - f(x_i)) K(\cdot, x_i)$$

[1] If $\mathcal{F}$ is a reproducing kernel Hilbert space (RKHS), it means there is some associated kernel $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ such that every $f \in \mathcal{F}$ can be written as $f = \sum_{x \in C_f} \alpha_x^f K(\cdot, x)$ where $C_f \subset \mathcal{X}$ is a finite set of "center points" and α_x^f is a coefficient for each center point. Note that $K(\cdot, x) \in \mathcal{F}$ for every $x \in \mathcal{X}$. The inner product is defined as $\langle f, g \rangle := (\alpha^f)^\top K^{f,g} \alpha^g$, in particular

$$\langle f, K(\cdot, x) \rangle = \sum_{x' \in C_f} \alpha_{x'}^f K(x', x) = f(x)$$

hence the name "reproducing". By the Moore-Aronszajn theorem, any kernel induces a unique associated RKSH.

Functional gradient descent. Let $\widehat{L} : \mathcal{F} \to \mathbb{R}$ be a differentiable loss over $S = \{(x_1, y_1) \dots (x_N, y_N)\} \subset \mathcal{X} \times \mathbb{R}$ (e.g., $\widehat{L}_{\text{sq}}$). We may do steepest descent on $\widehat{L}$ over $\mathcal{F}$. This means we pick an initial $f_0 \in \mathcal{F}$ and for $t = 1 \dots T$ find $g_t \in \mathcal{F}$ with a bounded norm such that $\widehat{L}(f_{t-1} + g_t)$ is small. Using the linear approximation $\widehat{L}(f_{t-1} + g_t) \approx \widehat{L}(f_{t-1}) + \langle \nabla \widehat{L}(f_{t-1}), g_t \rangle$, the steepest descent direction is given by $g_t = \eta_t(-\nabla \widehat{L}(f_{t-1}))$ where $\eta_t \in \mathbb{R}$. Then we set

$$f_t = f_{t-1} + g_t$$

Note that $f_T = f_0 + \sum_{t=1}^{T} g_t$ can be viewed as an *ensemble*; at test time, given $x \in \mathcal{X}$ the model computes $f_0(x) \in \mathbb{R}$ and $g_1(x) \dots g_T(x) \in \mathbb{R}$, then returns $f_T(x) = f_0(x) + g_1(x) + \cdots + g_T(x)$.

While conceptually nice, functional gradient descent is not implementable due to its nonparametric nature. For instance, the descent direction $-\nabla \widehat{L}_{\text{sq}}(f_{t-1}) = \sum_{i=1}^{N} (y_i - f_{t-1}(x_i)) K(\cdot, x_i)$ is an abstract real-valued mapping over $\mathcal{X}$. However, we can easily *evaluate* it on S. Define

$$S_t = \left\{ (x_i, y_i') : \; y_i' = -\nabla \widehat{L}(f_{t-1})(x_i), i \in \{1 \dots N\} \right\}$$

We can treat S_t as labeled data and fit some parametric model to approximate $h_t \approx -\nabla \widehat{L}(f_{t-1})$. We decide η_t by some learning rate schedule or line search (i.e., $\min_{\eta \in \mathbb{R}} \widehat{L}(f_{t-1} + \eta h_t)$), and set $g_t = \eta_t h_t$.

E.1 Application to Regression

Let $K(x, x') = [[x = x']]$ be the identity kernel and $\mathcal{F}$ be the associated RKHS. Given $S = \{(x_1, y_1) \dots (x_N, y_N)\} \subset \mathcal{X} \times \mathbb{R}$, we optimize the squared-error functional $\widehat{L}_{\text{sq}}(f) = \frac{1}{2} \sum_{i=1}^{N} (y_i - f(x_i))^2$. For any $i \in \{1 \dots N\}$,

$$-\nabla \widehat{L}_{\text{sq}}(f)(x_i) = y_i - f(x_i)$$

namely the i-th residual of f. We will use the squared loss to fit the residuals. We assume a parametric hypothesis class that is easy to optimize over (e.g., decision trees, linear regressors). With the constant step size $\eta_t = 1$, the gradient boosting algorithm is

1. Initialize $h_0 \in \arg\min_{h : \mathcal{X} \to \mathbb{R}} \sum_{i=1}^{N} (y_i - h(x_i))^2$.
2. For $t = 1 \dots T$, find

$$h_t \in \operatorname*{arg\,min}_{h : \mathcal{X} \to \mathbb{R}} \sum_{i=1}^{N} \left(\left(y_i - \sum_{s=0}^{t-1} h_s(x_i) \right) - h(x_i) \right)^2$$

3. Given $x \in \mathcal{X}$, predict $\sum_{t=0}^{T} h_t(x)$.

E.2 Application to Classification

Consider K-way classification. A classifier can be viewed as an array of K regressors $f(x) = (f^{(1)}(x) \dots f^{(K)}(x)) \in \mathbb{R}^K$ where $f^{(k)} \in \mathcal{F}$ calculates the k-th logit for input $x \in \mathcal{X}$. Given labeled data $S = \{(x_1, y_1) \dots (x_N, y_N)\} \subset \mathcal{X} \times \{1 \dots K\}$, the cross-entropy functional $\widehat{L}_{\text{CE}} : \mathcal{F}^K \to \mathbb{R}$ is defined as

$$\widehat{L}_{\text{CE}}(f) := \sum_{i=1}^{N} \log \left(\sum_{k=1}^{K} \exp \left(f^{(k)}(x_i) \right) \right) - f^{(y_i)}(x_i)$$

For $k \in \{1 \dots K\}$, the negative functional gradient of $\widehat{L}_{\text{CE}}(f)$ with respect to f_k is

$$-\nabla_k \widehat{L}_{\text{CE}}(f) = \sum_{i=1}^{N} \left([[y_i = k]] - p_f(k|x_i) \right) K(\cdot, x_i)$$

where $p_f(y|x) := \exp(f^{(y)}(x)) / \sum_{k=1}^{K} \exp(f^{(k)}(x))$. Assuming the identity kernel, upon receiving x_i it evaluates to

$$-\nabla_k \widehat{L}_{\text{CE}}(f)(x_i) = [[y_i = k]] - p_f(k|x_i)$$

namely the i-th residual of f for the k-th class. We will again use the squared loss to fit the residuals, an easy-to-optimize-over hypothesis class, and $\eta_t = 1$. Then the gradient boosting algorithm is

1. Initialize $h_0 \in \arg\min_{h:\mathcal{X}\to\mathbb{R}^K} \sum_{i=1}^N \log\left(\sum_{k=1}^K \exp\left(h^{(k)}(x_i)\right)\right) - h^{(y_i)}(x_i)$.

2. For $t = 1 \dots T$, for $k = 1 \dots K$ find

$$h_t^{(k)} \in \arg\min_{h:\mathcal{X}\to\mathbb{R}} \sum_{i=1}^N \left(\left([[y_i = k]] - \frac{\exp\left(\sum_{s=0}^{t-1} h_s^{(k)}(x)\right)}{\sum_{k'=1}^K \exp\left(\sum_{s=0}^{t-1} h_s^{(k')}(x)\right)} \right) - h(x_i) \right)^2$$

3. Given $x \in \mathcal{X}$, predict $y^* \in \arg\max_{y=1}^K \sum_{t=0}^T h_t^{(y)}(x)$.