

Boosting

Boosting is a technique for turning a bad classifier into a good classifier by “boosting” its accuracy. The idea is: if we have a PAC learning algorithm that always returns $\epsilon < 0.5$ (the “strictly” in “strictly less than” is key), then we can “boost” it into a PAC learning algorithm that gets arbitrarily small error.

The key is to constantly change the distribution of data on which we train our “weak learner” so that it learns to correct its own mistakes. The most popular boosting algorithm is AdaBoost. It runs in T rounds and in each round learns a weak hypothesis h_t (for convenience, assume h_t returns $+1$ or -1). It also learns a coefficient α_t for each weak hypothesis. The final prediction for an input \hat{x} is:

$$\hat{y} = \text{sign} \left[\sum_{t=1}^T \alpha_t h_t(\hat{x}) \right]$$

The way that AdaBoost works is by maintaining a *distribution* over the input points. Initially this distribution is uniform (each example is just as important as any other). But as the algorithm proceeds, it will adjust the distribution to concentrate on examples that are being incorrectly classified by the weak learner.

More formally, suppose we have training point $(x_1, y_1), \dots, (x_N, y_N)$, where $y_n \in \{-1, +1\}$. We also have a classification algorithm (the weak learner) \mathcal{A} that takes as input a set of input points, a set of labels, and an importance weighting for each example. It returns a classifier that gets error less than 0.5. In other words, if we let $D = \langle \frac{1}{N}, \frac{1}{N}, \dots, \frac{1}{N} \rangle$ be a distribution (this is uniform), then $\mathcal{A}(\langle x_n, y_n, D_n \rangle_{n=1}^N)$ returns a classifier h . Given this set up, AdaBoost works as follows:

1. Initialize $D = \langle \frac{1}{N}, \frac{1}{N}, \dots, \frac{1}{N} \rangle$
2. For $t = 1 \dots T$:
 - (a) Let $h_t = \mathcal{A}(\langle x_n, y_n, D_n \rangle_{n=1}^N)$
 - (b) Compute the error ϵ_t of h_t on the training data *with respect to* D
 - (c) Compute coefficient $\alpha_t = \frac{1}{2} \log((1 - \epsilon_t)/\epsilon_t)$
 - (d) Update distribution:

$$\begin{aligned} D_n &\leftarrow \frac{1}{Z} D_n \times \begin{cases} e^{-\alpha_t} & \text{if } h_t(x_n) = y_n \\ e^{\alpha_t} & \text{otherwise} \end{cases} \\ &= \frac{1}{Z} D_n \exp \left[-\alpha_t y_n h_t(x_n) \right] \end{aligned}$$

where Z is chosen so that $\sum_{n=1}^N D_n = 1$

3. Return classifiers $h_{1:T}$ and coefficient $\alpha_{1:T}$

The point of (2d) is that we want to increase the weight of examples we’re getting wrong and decrease the weight of examples we’re getting right. The intuition behind the choice of α_t is as follows. We know $\epsilon_t < 0.5$, so $(1 - \epsilon_t)/\epsilon_t$ is in the range $(1, \infty)$, so (half) the log is in the range $(0, \infty)$. Let’s suppose ϵ_t is really close to zero, which means that we’re doing quite a good job. In this case, α_t will approach ∞ , which means that in (2d), we will really increase the weights of the (few!) examples we’re getting wrong. On the other hand, if $\epsilon_t \approx 0.5$, then $\alpha_t \approx 0$, which means we’re not changing the distribution by much. The way I think about this is like taking an exam in a class. If you only get one question wrong, then you really want to concentrate and learn that one question. On the other hand, if you get everything wrong, you should probably re-study everything.