1 Boosting

We have seen that in the case of random forests, combining many imperfect models can produce a single model that works very well. This is the idea of **ensemble methods**. However, random forests treat each member of the forest equally, taking a majority vote or an average over their outputs. The idea of **boosting** is to combine the models (typically called *weak learners* in this context) in a more principled manner. Roughly, we want to:

- Treat each of the weak classifiers (e.g., trees in a random forest) as "features"
- Hunt for better "features" based on the current performance of the overall classifier

The key idea is as follows: to improve our classifier, we should focus on finding learners that correctly classify the points which the overall boosted classifier is currently getting wrong. Boosting algorithms implement this idea by associating a weight with each training point and iteratively reweighting so that misclassified points have relatively high weights. Intuitively, some points are "harder" to classify than others, so the algorithm should focus its efforts on those.

2 AdaBoost

There are many flavors of boosting. We will discuss one of the most popular versions, known as **AdaBoost** (short for *adaptive boosting*). Its developers won the Gödel Prize for this work.

2.1 Algorithm

We present the algorithm first, then derive it later.

1. Initialize the weight of all n training points to $\frac{1}{n}$.
2. Repeat for $m = 1, \ldots, M$:
   (a) Build a classifier G_m with data weighted according to w_i.
   (b) Compute the weighted error $e_m = \frac{\sum_{i \text{ misclassified}} w_i}{\sum_i w_i}$.
   (c) Re-weight the training points as

$$w_i \leftarrow w_i \cdot \begin{cases} \sqrt{\frac{1-e_m}{e_m}} & \text{if misclassified} \\ \sqrt{\frac{e_m}{1-e_m}} & \text{otherwise} \end{cases}$$

(d) Optional: normalize the weights w_i to sum to 1.

We first address the issue of step (a): how do we train a classifier if we want to weight different samples differently? One common way to do this is to resample from the original training set every iteration to create a new training set that is fed to the next classifier. Specifically, we create a training set of size n by sampling n values from the original training data with replacement, according to the distribution w_i. (This is why we might renormalize the weights in step (d).) This way, data points with large values of w_i are more likely to be included in this training set, and the next classifier will place higher priority on such data points.

Suppose that our weak learners always produce an error $e_m < 1/2$. To make sense of the formulas we see in the algorithm, note that for step (c), if the i-th data point is misclassified, then the weight w_i gets increased by a factor of $\sqrt{\frac{1-e_m}{e_m}}$ (more priority placed on sample i), while if it is classified correctly, the priority gets decreased. AdaBoost does have a weakness in that this aggressive reweighting can cause the classifier to focus too much on certain training examples – if the data is noisy with outliers, then this will weaken the boosting algorithm's performance.

We have not yet discussed how to make a prediction given our classifiers (say, $G_1, \dots, G_M$). One conceivable method is to use logistic regression with $G_m(x)$ as features. However, a smarter choice that is based on the AdaBoost algorithm is to set

$$\alpha_m = \frac{1}{2} \ln \left(\frac{1-e_m}{e_m} \right)$$

and classify x by $\text{sign}(\sum_m \alpha_m G_m(x))$. Note that this choice of α_m (derived later) gives high weight to classifiers that have low error:

- As $e_m \to 0$, $\frac{1-e_m}{e_m} \to \infty$, so $\alpha_m \to \infty$.
- As $e_m \to 1$, $\frac{1-e_m}{e_m} \to 0$, so $\alpha_m \to -\infty$.

We now proceed to demystify the formulas in the algorithm by presenting a matching pursuit interpretation of AdaBoost.

2.2 Derivation of AdaBoost

Suppose we have computed classifiers $G_1, ..., G_{m-1}$ along with their corresponding weights α_k and we want to compute the next classifier G_m along with its weight α_m. The output of our model so far is $F_{m-1}(x) := \sum_{i=1}^{m-1} \alpha_i G_i(x)$, and we want to minimize the risk:

$$\arg\min_{\alpha_m, G_m} \sum_{i=1}^{n} L(y_i, F_{m-1}(x_i) + \alpha_m G_m(x_i))$$

for some suitable loss function L. Loss functions we have previously used include mean squared error for linear regression, cross-entropy loss for logistic regression, and hinge loss for SVM. For AdaBoost, we use the *exponential loss*:

$$L(y, h(x)) = e^{-yh(x)}$$

This loss function is illustrated in Figure 1. We have exponential decay as we increase the input – thus if $yh(x)$ is large and positive (so $h(x)$ has the correct sign and high magnitude), our loss is decreasing exponentially. Conversely, if $yh(x)$ is a large negative value, our loss is increasing exponentially, and thus we are heavily penalized for confidently making an incorrect prediction.

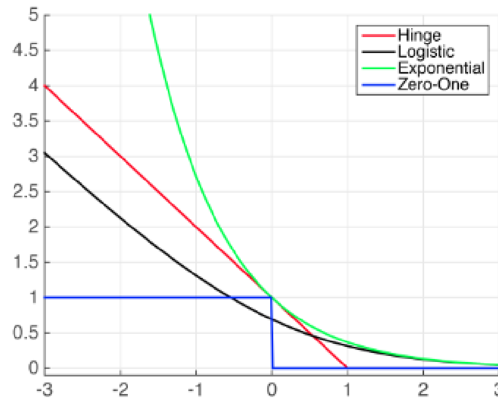

Figure 1: The exponential loss provides exponentially increasing penalty for confident incorrect predictions. This figure is from Cornell CS4780 notes.

We can write the AdaBoost optimization problem with exponential loss as follows:

$$\begin{aligned}\alpha_m, G_m &= \arg\min_{\alpha, G} \sum_{i=1}^{n} e^{-y_i(F_{m-1}(x_i)+\alpha G(x_i))} \\ &= \arg\min_{\alpha, G} \sum_{i=1}^{n} e^{-y_i F_{m-1}(x_i)} e^{-y_i \alpha G(x_i)}\end{aligned}$$

The term $w_i^{(m)} := e^{-y_i F_{m-1}(x)}$ is a constant with respect to our optimization variables. We can split out this sum into the components with correctly classified points and incorrectly classified points:

$$\begin{aligned}\alpha_m, G_m &= \arg\min_{\alpha, G} \sum_{y_i = G(x)} w_i^{(m)} e^{-\alpha} + \sum_{y_i \neq G(x)} w_i^{(m)} e^{+\alpha} \qquad (*) \\ &= \arg\min_{\alpha, G} \, (e^{\alpha} - e^{-\alpha}) \sum_{y_i \neq G(x_i)} w_i^{(m)} + e^{-\alpha} \sum_{i=1}^{n} w_i^{(m)}\end{aligned}$$

For a fixed value of α, the second term does not depend on G. Thus we can see that the best choice of $G_m(x)$ is the classifier that minimizes the error given the weights $w_i^{(m)}$. Let

$$e_m = \frac{\sum_{y_i \neq G_m(x_i)} w_i^{(m)}}{\sum_i w_i^{(m)}}$$

Once we have obtained G_m, we can solve for α_m: by dividing $(*)$ by the constant $\sum_{i=1}^{n} w_i^{(m)}$, we obtain

$$\alpha_m = \arg\min_{\alpha} \, (1 - e_m) e^{-\alpha} + e_m e^{\alpha}$$

which has the solution (left as exercise)

$$\alpha_m = \frac{1}{2}\ln\left(\frac{1-e_m}{e_m}\right)$$

as claimed earlier.

From the optimal α_m, we can derive the weights:

$$\begin{aligned}
w_i^{(m+1)} &= \exp\left(-y_i F_m(x_i)\right) \\
&= \exp\left(-y_i[F_{m-1}(x_i) + \alpha_m G_m(x_i)]\right) \\
&= w_i^{(m)} \exp\left(-y_i G_m(x_i)\alpha_m\right) \\
&= w_i^{(m)} \exp\left(-y_i G_m(x_i)\frac{1}{2}\ln\left(\frac{1-e_m}{e_m}\right)\right) \\
&= w_i^{(m)} \exp\left(\ln\left[\left(\frac{1-e_m}{e_m}\right)^{-\frac{1}{2}y_i G_m(x_i)}\right]\right) \\
&= w_i^{(m)} \left(\frac{1-e_m}{e_m}\right)^{-\frac{1}{2}y_i G_m(x_i)}
\end{aligned}$$

Here we see that the multiplicative factor is $\sqrt{\frac{e_m}{1-e_m}}$ when $y_i = G_m(x_i)$ and $\sqrt{\frac{1-e_m}{e_m}}$ otherwise. This completes the derivation of the algorithm.

As a final note about the intuition, we can view these α updates as pushing towards a solution in some direction until we can no longer improve our performance. More precisely, whenever we compute α_m (and thus $w^{(m+1)}$), for the incorrectly classified entries, we have

$$\sum_{y_i \neq G_m(x_i)} w_i^{(m+1)} = \sum_{y_i \neq G_m(x_i)} w_i^{(m)} \sqrt{\frac{1-e_m}{e_m}}$$

Dividing the right-hand side by $\sum_{i=1}^n w_i^{(m)}$, we obtain $e_m\sqrt{\frac{1-e_m}{e_m}} = \sqrt{e_m(1-e_m)}$. Similarly, for the correctly classified entries, we have

$$\frac{\sum_{y_i = G_m(x_i)} w_i^{(m+1)}}{\sum_{i=1}^n w_i^{(m)}} = (1-e_m)\sqrt{\frac{e_m}{1-e_m}} = \sqrt{e_m(1-e_m)}$$

Thus these two quantities are the same once we have adjusted our α, so the misclassified and correctly classified sets both get equal total weight.

This observation has an interesting practical implication. Even after the training error goes to zero, the AdaBoost test error may continue to decrease. This may be counter-intuitive, as one would expect the classifier to be overfitting to the training data at this point. One interpretation for this phenomenon is that even though the boosted classifier has achieved perfect training error, it is still refining its fit in a max-margin fashion, which increases its generalization capabilities.