18 AdaBoost

Boosting is a general strategy for learning classifiers by combining simpler ones. The idea of boosting is to take a "weak classifier" — that is, any classifier that will do at least slightly better than chance — and use it to build a much better classifier, thereby boosting the performance of the weak classification algorithm. This boosting is done by averaging the outputs of a collection of weak classifiers. The most popular boosting algorithm is **AdaBoost**, so-called because it is "adaptive."[1] AdaBoost is extremely simple to use and implement (far simpler than SVMs), and often gives very effective results. There is tremendous flexibility in the choice of weak classifier as well. Boosting is a specific example of a general class of learning algorithms called **ensemble methods,** which attempt to build better learning algorithms by combining multiple simpler algorithms.

Suppose we are given training data $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$, where $\mathbf{x}_i \in \mathbb{R}^K$ and $y_i \in \{-1, 1\}$. And suppose we are given a (potentially large) number of weak classifiers, denoted $f_m(\mathbf{x}) \in \{-1, 1\}$, and a **0-1** loss function I, defined as

$$I(f_m(\mathbf{x}), y) = \begin{cases} 0 & \text{if} f_m(\mathbf{x}_i) = y_i \\ 1 & \text{if} f_m(\mathbf{x}_i) \neq y_i \end{cases} \tag{1}$$

Then, the pseudocode of the AdaBoost algorithm is as follows:

for i from 1 to N, $w_i^{(1)} = 1$
for $m = 1$ to M **do**
  Fit weak classifier m to minimize the objective function:
    $\epsilon_m = \frac{\sum_{i=1}^N w_i^{(m)} I(f_m(\mathbf{x}_i) \neq y_i)}{\sum_i w_i^{(m)}}$
  where $I(f_m(\mathbf{x}_i) \neq y_i) = 1$ if $f_m(\mathbf{x}_i) \neq y_i$ and 0 otherwise
  $\alpha_m = \ln \frac{1-\epsilon_m}{\epsilon_m}$
  for all i **do**
    $w_i^{(m+1)} = w_i^{(m)} e^{\alpha_m I(f_m(\mathbf{x}_i) \neq y_i)}$
  end for
end for

After learning, the final classifier is based on a linear combination of the weak classifiers:

$$g(\mathbf{x}) = \text{sign}\left(\sum_{m=1}^M \alpha_m f_m(\mathbf{x})\right) \tag{2}$$

Essentially, AdaBoost is a greedy algorithm that builds up a "strong classifier", i.e., $g(\mathbf{x})$, incrementally, by optimizing the weights for, and adding, one weak classifier at a time.

[1] AdaBoost was called adaptive because, unlike previous boosting algorithms, it does not need to know error bounds on the weak classifiers, nor does it need to know the number of classifiers in advance.

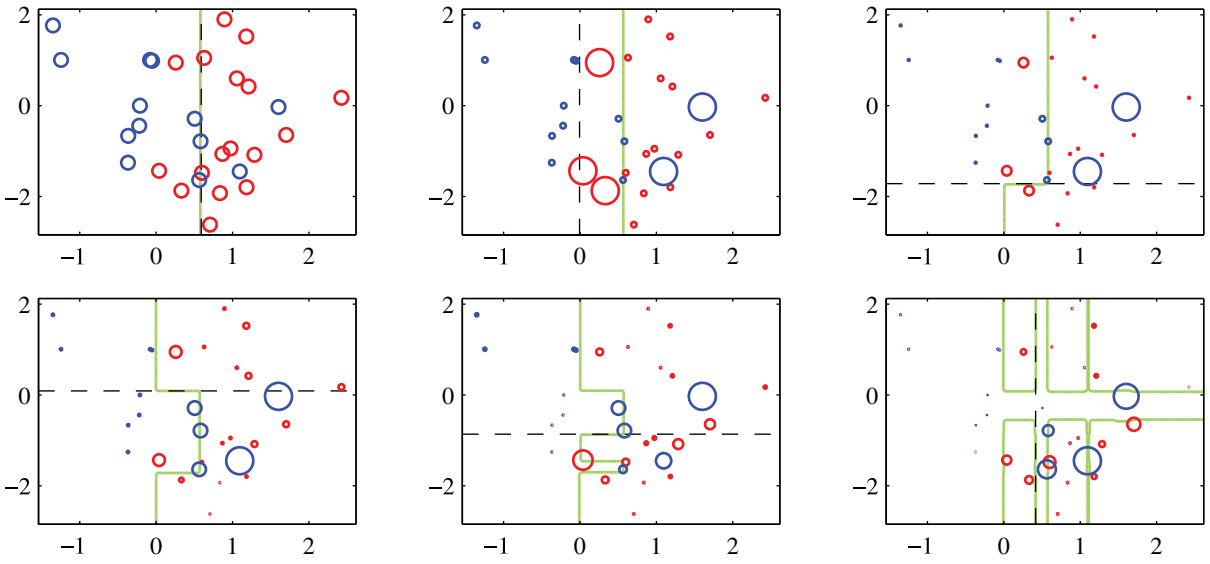

Figure 1: Illustration of the steps of AdaBoost. The decision boundary is shown in green for each step, and the decision stump for each step shown as a dashed line. The results are shown after 1, 2, 3, 6, 10, and 150 steps of AdaBoost. (Figure from *Pattern Recognition and Machine Learning* by Chris Bishop.)

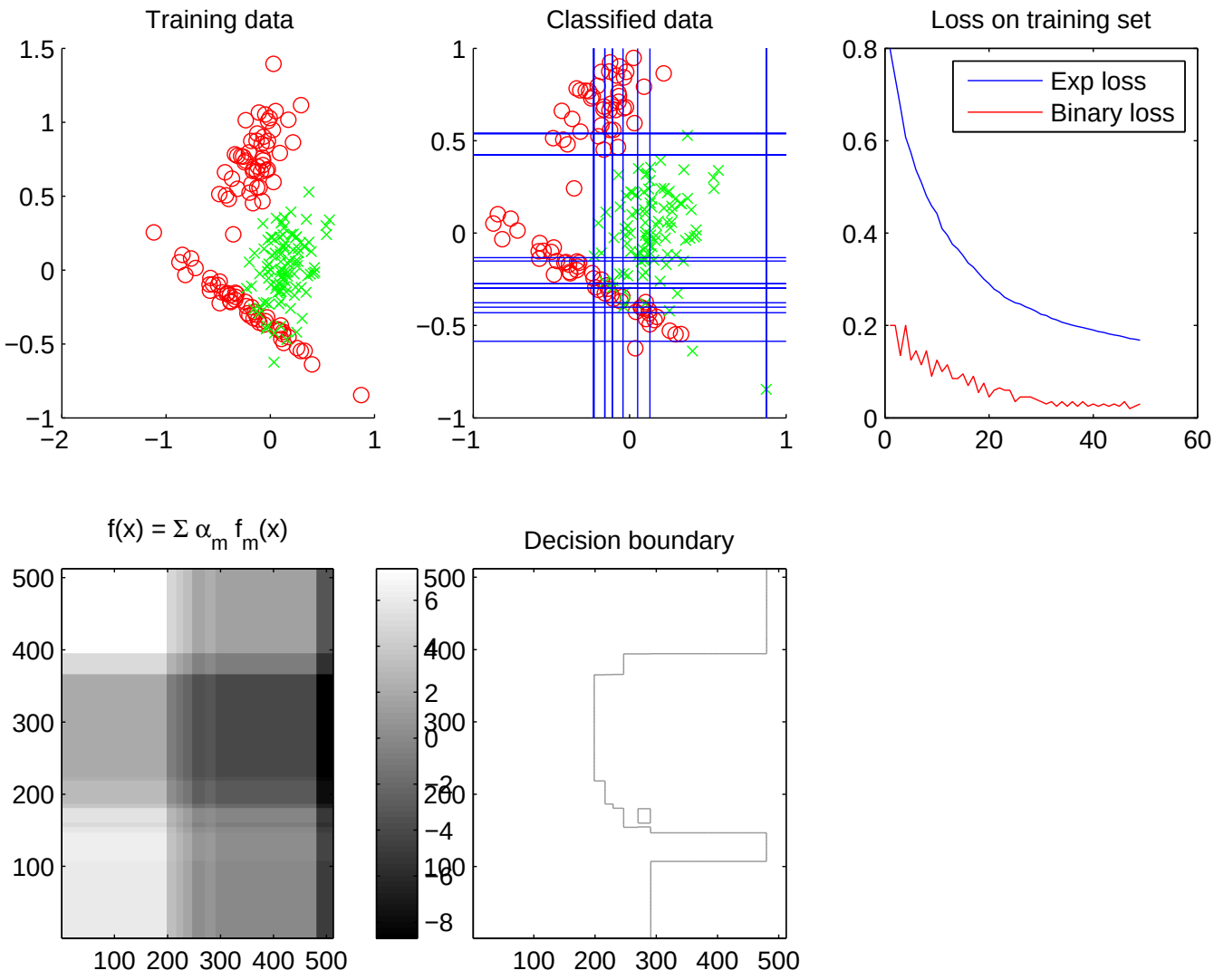

Figure 2: 50 steps of AdaBoost used to learn a classifier with decision stumps.

18.1 Decision stumps

As an example of a weak classifier, we consider "decision stumps," which are a trivial special case of decision trees. A decision stump has the following form:

$$f(\mathbf{x}) = s(x_k > c) \tag{3}$$

where the value in the parentheses is 1 if the k-th element of the vector $\mathbf{x}$ is greater than c, and -1 otherwise. The scalar s is either -1 or 1 which allows one the classifier to respond with class 1 when $x_k \leq c$. Accordingly, there are three parameters to a decision stump:

- $c \in \mathbb{R}$
- $k \in \{1, ...K\}$, where K is the dimension of $\mathbf{x}$, and
- $s \in \{-1, 1\}$

Because the number of possible parameter settings is relatively small, a decision stump is often trained by brute force: discretize the real numbers from the smallest to the largest value in the training set, enumerate all possible classifiers, and pick the one with the lowest training error. One can be more clever in the discretization: between each pair of data points, only one classifier must be tested (since any stump in this range will give the same value). More sophisticated methods, for example, based on binning the data, or building CDFs of the data, may also be possible.

18.2 Why does it work?

There are many different ways to analyze AdaBoost; none of them alone gives a full picture of why AdaBoost works so well. AdaBoost was first invented based on optimization of certain bounds on training, and, since then, a number of new theoretical properties have been discovered.

Loss function view. Here we discuss the loss function interpretation of AdaBoost. As was shown (decades after AdaBoost was first invented), AdaBoost can be viewed as greedy optimization of a particular loss function. We define $f(\mathbf{x}) = \frac{1}{2}\sum_m \alpha_m f_m(\mathbf{x})$, and rewrite the classifier as $g(\mathbf{x}) = \text{sign}(f(\mathbf{x}))$ (the factor of 1/2 has no effect on the classifier output). AdaBoost can then be viewed as optimizing the **exponential loss:**

$$L_{exp}(\mathbf{x}, y) = e^{-yf(\mathbf{x})} \tag{4}$$

so that the full learning objective function, given training data $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$, is

$$E = \sum_i e^{-\frac{1}{2}y_i \sum_{m=1}^M \alpha_m f_m(\mathbf{x}_i)} \tag{5}$$

which must be optimized with respect to the weights α and the parameters of the weak classifiers. The optimization process is greedy and sequential: we add one weak classifier at a time, choosing it

and its α to be optimal with respect to E, and then never change it again. Note that the exponential loss is an upper-bound on the 0-1 loss:

$$L_{exp}(\mathbf{x}, y) \geq L_{0-1}(\mathbf{x}, y) \tag{6}$$

Hence, if exponential loss of zero is achieved, then the 0-1 loss is zero as well, and all training points are correctly classified.

Consider the weak classifier f_m to be added at step m. The entire objective function can be written to separate out the contribution of this classifier:

$$E = \sum_i e^{-\frac{1}{2} y_i \sum_{j=1}^{m-1} \alpha_j f_j(\mathbf{x}_i) - \frac{1}{2} y_i \alpha_m f_m(\mathbf{x}_i)} \tag{7}$$

$$= \sum_i e^{-\frac{1}{2} y_i \sum_{j=1}^{m-1} \alpha_j f_j(\mathbf{x}_i)} \; e^{-\frac{1}{2} y_i \alpha_m f_m(\mathbf{x}_i)} \tag{8}$$

Since we are holding constant the first $m-1$ terms, we can replace them with a single constant $w_i^{(m)} = e^{-\frac{1}{2} y_i \sum_{j=1}^{m-1} \alpha_j f_j(\mathbf{x}_i)}$. Note that these are the same weights computed by the recursion used by AdaBoost, i.e., $w_i^{(m)} \propto w_i^{(m-1)} e^{-\frac{1}{2} y_i \alpha_j f_{m-1}(\mathbf{x}_i)}$. (There is a proportionality constant that can be ignored). Hence, we have

$$E = \sum_i w_i^{(m)} e^{-\frac{1}{2} y_i \alpha_m f_m(\mathbf{x}_i)} \tag{9}$$

We can split this into two summations, one for data correctly classified by f_m, and one for those misclassified:

$$E = \sum_{i: f_m(\mathbf{x}_i) = y_i} w_i^{(m)} e^{-\frac{\alpha_m}{2}} + \sum_{i: f_m(\mathbf{x}_i) \neq y_i} w_i^{(m)} e^{\frac{\alpha_m}{2}} \tag{10}$$

Rearranging terms, we have

$$E = (e^{\frac{\alpha_m}{2}} - e^{-\frac{\alpha_m}{2}}) \sum_i w_i^{(m)} I(f_m(\mathbf{x}_i) \neq y_i) + e^{-\frac{\alpha_m}{2}} \sum_i w_i^{(m)} \tag{11}$$

Optimizing this with respect to f_m is equivalent to optimizing $\sum_i w_i^{(m)} I(f_m(\mathbf{x}_i) \neq y_i)$, which is what AdaBoost does. The optimal value for α_m can be derived by solving $\frac{dE}{d\alpha_m} = 0$:

$$\frac{dE}{d\alpha_m} = \frac{\alpha_m}{2} \left(e^{\frac{\alpha_m}{2}} + e^{-\frac{\alpha_m}{2}} \right) \sum_i w_i^{(m)} I(f_m(\mathbf{x}_i) \neq y_i) - \frac{\alpha_m}{2} e^{-\frac{\alpha_m}{2}} \sum_i w_i^{(m)} = 0 \tag{12}$$

Dividing both sides by $\frac{\alpha_m}{2 \sum_i w_i^{(m)}}$, we have

$$0 = e^{\frac{\alpha_m}{2}} \epsilon_m + e^{-\frac{\alpha_m}{2}} \epsilon_m - e^{-\frac{\alpha_m}{2}} \tag{13}$$

$$e^{\frac{\alpha_m}{2}} \epsilon_m = e^{-\frac{\alpha_m}{2}} (1 - \epsilon_m) \tag{14}$$

$$\frac{\alpha_m}{2} + \ln \epsilon_m = -\frac{\alpha_m}{2} + \ln(1 - \epsilon_m) \tag{15}$$

$$\alpha_m = \ln \frac{1 - \epsilon_m}{\epsilon_m} \tag{16}$$

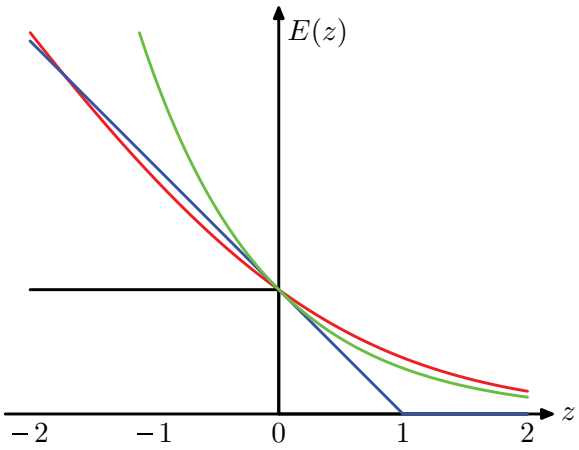

Figure 3: Loss functions for learning: Black: 0-1 loss. Blue: Hinge Loss. Red: Logistic regression. Green: Exponential loss. (Figure from *Pattern Recognition and Machine Learning* by Chris Bishop.)

Problems with the loss function view. The exponential loss is not a very good loss function to use in general. For example, if we directly optimize the exponential loss over all variables in the classifier (e.g., with gradient descent), we will often get terrible performance. So the loss-function interpretation of AdaBoost does not tell the whole story.

Margin view. One might expect that, when AdaBoost reaches zero training set error, adding any new weak classifiers would cause overfitting. In practice, the opposite often occurs: continuing to add weak classifiers actually improves test set performance in many situations. One explanation comes from looking at the margins: adding classifiers tends to increase the margin size. The formal details of this will not be discussed here.

18.3 Early stopping

It is nonetheless possible to overfit with AdaBoost, by adding too many classifiers. The solution that is normally used practice is a procedure called **early stopping**. The idea is as follows. We partition our data set into two pieces, a training set and a test set. The training set is used to train the algorithm normally. However, at each step of the algorithm, we also compute the 0-1 binary loss on the test set. During the course of the algorithm, the exponential loss on the training set is guaranteed to decrease, and the 0-1 binary loss will generally decrease as well. The errors on the testing set will also generally decrease in the first steps of the algorithm, however, at some point, the testing error will begin to get noticeably worse. When this happens, we revert the classifier to the form that gave the best test error, and discard any subsequent changes (i.e., additional weak classifiers).

The intuition for the algorithm is as follows. When we begin learning, our initial classifier is extremely simple and smooth. During the learning process, we add more and more complexity to

the model to improve the fit to the data. At some point, adding additional complexity to the model overfits: we are no longer modeling the decision boundary we wish to fit, but are fitting the noise in the data instead. We use the test set to determine when overfitting begins, and stop learning at that point.

Early stopping can be used for most iterative learning algorithms. For example, suppose we use gradient descent to learn a regression algorithm. If we begin with weights $\mathbf{w} = 0$, we are beginning with a very smooth curve. Each step of gradient descent will make the curve less smooth, as the entries of $\mathbf{w}$ get larger and larger; stopping early can prevent $\mathbf{w}$ from getting too large (and thus too non-smooth).

Early stopping is very simple and very general; however, it is heuristic, as the final result one gets will depend on the particulars in the optimization algorithm being used, and not just on the objective function. However, AdaBoost's procedure is suboptimal anyway (once a weak classifier is added, it is never updated).

An even more aggressive form of early stopping is to simply stop learning at a fixed number of iterations, or by some other criteria unrelated to test set error (e.g., when the result "looks good.") In fact, pracitioners often using early stopping to regularize **unintentionally,** simply because they halt the optimizer before it has converged, e.g., because the convergence threshold is set too high, or because they are too impatient to wait.

