1 There is more than one way to average predictors

Classification will be the running example here, but most results hold for other prediction problems as well.

Denote \( B = \{ b \} \) a base classifier family

Averaging:

\[
f(x) = \sum_{k=1}^{M} \beta_k b_k(x)
\]

(1)

\( f \) is real-valued even if the \( b_k \)'s are \( \pm 1 \) valued

Why average predictors?

- to reduce bias
- to compensate for local optima (a form of bias)
- to reduce variance
- if \( b_1, b_2, \ldots, b_M \) make independent errors, averaging reduces expected error (loss). We say that \( b_1 \) and \( b_2 \) make independent errors \( \iff \)
  \[ P(b_1 \text{ wrong } | \ x) = P(b_1 \text{ wrong } | \ x, b_2 \text{ wrong}) \]
- because the \( b_k \) functions are given: real world domain experts (weather prediction), a set of black-box classifiers (from a software package), a set of [expert designed] features (speech recognition, car/human recognition) each of them weakly informative of \( y \)
• because \( \mathcal{B} \) is a set of (simple) “basis functions” and we need a more complex predictor in our task

**Averaging is not always the same thing.** Depending how we choose \( \mathcal{B}, b^1, b^2, \ldots b^M \) and \( \beta^1, \beta^2, \ldots \beta^M \), we can obtain very different effects.

Also note that the averaging denoted by \( \sum_{k=1}^{M} \beta^k b^k(x) \) can stand in different context for averaging in different probability spaces. For instance, we will see that in Bagging averaging is over samples of size \( N \), while in Bayesian averaging, the averaging is over the functions \( b \in \mathcal{B} \), and in Boosting the summation is not an averaging after all.

We will examine

- Bayesian averaging (briefly)
- Mixtures of experts (briefly)
- Bagging (briefly)
- Backfitting (briefly)
- Stacking (see Murphy)
- Boosting

### 1.1 Bayesian averaging

Assume any predictor \( b \in \mathcal{B} \) could be the “true” predictor with some prior probability \( P_0(b) \). Learning means changing the probability distribution of \( b \) after seeing the data.

<table>
<thead>
<tr>
<th>Before seeing data</th>
<th>( P_0(b) )</th>
<th>prior probability of ( b )</th>
</tr>
</thead>
<tbody>
<tr>
<td>After seeing ( \mathcal{D} )</td>
<td>( P(b</td>
<td>\mathcal{D}) )</td>
</tr>
</tbody>
</table>

Bayes formula

\[
P(b|\mathcal{D}) = \frac{P_0(b)P(\mathcal{D}|b)}{\sum_{b' \in \mathcal{B}} P_0(b')P(\mathcal{D}|b')}
\]

Classification of a new instance by Bayesian averaging:

\[
f(x) = \sum_{b \in \mathcal{B}} b(x)P(b|\mathcal{D})
\]

or

\[
P(y|x, \mathcal{D}) = \sum_{b \in \mathcal{B}} P(b|\mathcal{D})1_{b(x)=y}
\]
Hence classifiers (or more generally predictors) are weighted by their posterior probability.

**Intuition:** The likelihood becomes more concentrated when $N$ increases

**Bayesian averaging in practice.** Summation (or integration) over $b \in B$ usually intractable. Practically, one samples a few $b$’s: $b_1, b_2, \ldots b_M$

$$\hat{P}(b^k | D) = \frac{P_0(b^k)P(D|b^k)}{\sum_{k'=1}^{M} P_0(b_{k'})P(D|b_{k'})} \equiv \beta_k$$

**Priors in practice.** Priors should reflect our knowledge about $b$. If $B$ and $P_0$ represent exactly the function class that the true classifier belongs to and our prior knowledge, then Bayesian averaging is optimal, in the sense that it minimizes the cost of the average future misclassifications.

### 1.2 Reducing variance: Bagging

What if we had several (independently sampled) training sets $D_1, D_2, \ldots D_M$?

- we could train classifiers $\{b^1, b^2, \ldots b^M\}$ on the respective $D_1, D_2, \ldots D_M$
- we could estimate $E_{P(b)}[b] \sim f = \frac{1}{M}\sum_{k=1}^{M} b^k$
- $f$ has always lower variance than $b^k$

Idea of bagging: sample $D_1, D_2, \ldots D_M$ from the given $D$ and estimate $b^k$ on $D_k$

$$f(x) = \text{sign} \left( \frac{1}{M} \sum_{k=1}^{M} b^k(x) \right)$$
Resulting “bagged” classifier $F = \frac{1}{3}(b_1 + b_2 + b_3)$

Thus, bagging is a form of bootstrap. It was shown theoretically and empirically that bagging reduces variance.

Bagging is good for

- base classifiers with high variance (complex)
- unstable classifiers (decision trees, decision lists, neural networks)
- noisy data
Example 1  A variant of bagging for decision trees is called random forests. A large ensemble of decision trees is fitted to the same data set, introducing randomness in various ways, like (1) resampling the data set, (2) taking random splits, with probabilities that favor “good” splits, etc. The output predictor is the (unweighted) average of all the trees.

This method works reasonably well for prediction, and can be extended to more than classification (regression, feature selection, even clustering).

1.3 Mixtures of Experts

This is a method where $\beta_k = \beta_k(x)$, and $\sum_k \beta_k(x) = 1$ for all $x$ in the domain of the inputs. The idea is that each $b_k$ is an “expert” in one region of the input space. We want $f$ the Mixture of experts to predict something close to $b_k(x)$ in the region of expertise of expert $b_k$ and to ignore the other experts.

The vector function $\beta(x) = [\beta_1(x) \ldots \beta_M(x)]$ is sometimes called the gating function.

Example 2  Suppose the true function to learn is $f^*(x) = |x|, x \in \mathbb{R}$. This can be well approximated by two linear experts $f^1(x) = x, f^2(x) = -x$ with weights $\beta^1 = \phi(x), \beta^1 = \phi(-x)$, where $\phi$ is the sigmoid function (hence $\beta^1 + \beta^2 = 1$ everywhere).

The example highlights that, by using a mixture of experts we can construct a more complex classifier by from simple classifiers (linear). “Simple” can mean easy to fit, or low complexity (aka simple decision region), or both.

The effective sample size for each $b_k$ is smaller than $N$, and corresponds to the data for which $\beta_k(x)$ is away from 0.

For more than two experts, a natural gating function is the softmax function. E.g.,

$$
\beta^k(x) = \frac{e^{v^T_k x}}{\sum_{l=1}^M e^{v^T_l x}} \quad \text{with } v_l \in \mathbb{R}^n \text{ a vector of parameters}
$$

Training By descent methods. Often the experts $f^{1:M}$ and gating functions $\beta^{1:M}$ are trained simultaneously (to maximize log-likelihood) by steepest descent, or EM algorithm (which we’ll study later).
1.4 Backfitting

In this framework, usually we fit $b^k$ as well as $\beta^k$. Denote by
\[ f^{-k}(x) = \sum_{l \neq k} \beta^k b^k(x) \]  
the combined predictor $f$ “minus” the $k$-th base predictor $b^k$.

**Backfitting Algorithm**

- **Input** $M$, labeled training set $\mathcal{D}$
- **Initialize** $b^{1:M} = 0, \beta^{1:M} = 0$
- **repeat**
  - **for** $k = 1, 2, \ldots M$
    - calculate $r^k(x_i), i = 1:N$
    - optimize $\hat{L}$ w.r.t $k$-th base predictor $\beta^k, b^k = \arg\min \hat{L}(f^{-k} + \beta b)$
  - **until** change in $\hat{L}$ small enough (or, change in $b^{1:M}$ small enough)
- **Output** $f(x) = \sum_{k=1}^{M} \beta^k b^k(x)$

Note that sometimes $\beta^k$ is redundant. A variant of this method is to fix $b^{1:M}$ in advance and to backfit only the weights $\beta^k$.

Backfitting is a method where $b^k$ are assumed to be simple, weak, and the predictor $f$ is built from the cooperation of several $b$’s. Thus, it is generally a bias reduction method.

**Example 3** Least squares regression. In this problem, it is useful to denote
\[ r^i(x^i) = y^i - f^{-k}(x^i) \]  
the residual of $f^{-k}$ at data point $x^i$. Then, $L_{LS}(y^i, f(x^i)) = y^i - f^{-k}(x^i) - \beta^k b^k(x^i)$. Hence, optimizing $L_{LS}$ w.r.t. $\beta^k, b^k$ can be expressed as
\[ \beta^k, b^k = \arg\min_{\beta, b} \sum_{i=1}^{N} (r^i - \beta b(x^i))^2. \]  

In other words, each step of backfitting is a least squares regression problem, where the output variable values $y^i$ are replaced with the current residuals $r^i$.

See also Additive Models in section 2.1
1.5 Reducing bias: Boosting

Base classifier family $\mathcal{B}$ has large bias (e.g. linear classifier, decision stumps) but learning always produces $b$ that is better (on the training set) than random guessing.

Preconditions for boosting

1. $\mathcal{B}$ is a weak classifier family. For any $\mathcal{D}$ there can be found $b \in \mathcal{B}$ such that the training error of $b$ on $\mathcal{D}$ is bounded below one half.

$$0 < \hat{L}_{01}(b) \leq \delta < \frac{1}{2}$$

2. Learning algorithm can take weighted data sets.

Idea of boosting: train a classifier $b^1$ on $\mathcal{D}$, then train a $b^2$ to correct the errors of $b^1$, then $b^3$ to correct the errors of $b^2$, etc.

Example 4 Boosting with stumps, i.e. decision trees with a single split ($c_1 \ldots c_4$ are the coefficients $\beta_{1,4}$).

AdaBoost Algorithm (Schapire-Singer variant)
Assume $B$ contains functions $b$ taking values in $[-1, 1]$ or $\{\pm 1\}$

**Input** $M$, labeled training set $D$

**Initialize** $f = 0$

$w_1^i = \frac{1}{N}$ weight of datapoint $x_i$

for $k = 1, 2, \ldots M$

1. "learn classifier for $D$ with weights $w^k$ ⇒ $b^k$"

2. compute error $\varepsilon^k = \sum_{i=1}^{N} w_i^k \frac{1-y_i b^k(x_i)}{2}$

3. set $\beta^k = \frac{1}{2} \ln \frac{1 - \varepsilon^k}{\varepsilon^k}$

4. compute new weights $w_i^{k+1} = \frac{1}{Z^k} w_i^k e^{-\beta^k y_i b^k(x_i)}$ where $Z^k$ is the normalization constant that makes $\sum_i w_i^{k+1} = 1$

**Output** $f(x) = \sum_{k=1}^{M} \beta^k b^k(x)$

**Remarks**

1. If $b(x) \in \{\pm 1\}$ then $y^i b(x) \in \{\pm 1\}$; if an error occurs $1-y^i b(x) = 1$; otherwise this expression is equal to 0. Thus, $\varepsilon^k$ in step 2 adds up the weights of the errors. If $b(x) \in [-1, 1]$ then the errors contribute different amounts to the loss depending on their margin. In both cases, $\varepsilon^k \in [0, \delta]$, $\delta < 0.5$.

2. The logarithm in step 3 is $> 0$ whenever $\varepsilon^k < 1/2$, which is guaranteed by the weak learner property of $B$.

3. If $b \in \{\pm 1\}$, then step 4 can be written equivalently (up to a multiplicative constant)

$$ w_i^{k+1} = \begin{cases} 
\frac{1}{Z^k} w_i^k & \text{if } b^k(x_i) = y_i \\
\frac{1}{Z^k} w_i^k e^{2\beta^k} & \text{if } b^k(x_i) \neq y_i
\end{cases} $$(6)

This form corresponds to the so-called DiscreteAdaBoost algorithm, the first AdaBoost algorithm published, which assumed $b(x) \in \{\pm 1\}$. As we shall see later, modern boosting algorithms dispense with the assumption $b \in [-1, 1]$ too.

1.6 **Boosting - properties on the training set**

1. An interpretation of the weights

$$ w_i^{k+1} = \frac{1}{N} \prod_{k' \leq k} \frac{e^{-\beta^{k'} y_i b^{k'}(x_i)}}{Z_{k'}} = \frac{e^{-y_i f^{k-1}(x_i)}}{N \prod_{k' \leq k} Z_{k'}} $$

Hence, the weight of example $i$ at step $k$ is proportional to $e^{-y_i f^{k-1}(x_i)}$ the exponential of its negative margin. Examples that have been hard
to classify get exponentially high weights. Examples that are classified with high margins get vanishingly small weights.

2. The normalization constant is an average “loss” If we sum both sides of (7) over $i$ we obtain

$$1 = \frac{\sum_i e^{-y_i f_k(x_i)}}{N \prod_{k' \leq k} Z_{k'}}$$

or

$$\prod_{k' \leq k} Z_k = \frac{\sum_i e^{-y_i f_k(x_i)}}{N} \equiv \hat{L}_\phi(f_k)$$

where

$$\phi(z) = e^{-z}.$$ (10)

and

$$L_\phi(y, f(x)) = \phi(yf(x))$$ (11)

Hence, the r.h.s of (9) is the average over the data set of the exponential loss $L_\phi$.

The function $\phi$ decreases with the margin, thus decreasing $\hat{L}_\phi$ will produce a better classifier (on the training set). In this sense, $L_\phi$ is an alternative loss function for classification.

3. $\hat{L}_\phi$ decreases exponentially with $M$.

For simplicity, we show this in the special case $b(x) \in \{\pm 1\}$ for all $b \in B$.

$$Z_k = \sum_{i=1}^n w_i^k e^{-\beta((y_i b_k(x_i)))}$$ (12)

$$= e^{\beta_k} \sum_{i=err} w_i^k + e^{-\beta_k} \sum_{i=corr} w_i^k$$ (13)

$$= e^{\beta_k} \varepsilon_k + (1 - \varepsilon_k) e^{-\beta_k}$$ (14)

$$= \sqrt{\frac{1 - \varepsilon_k}{\varepsilon_k}} \varepsilon_k + \sqrt{\frac{\varepsilon_k}{1 - \varepsilon_k}} (1 - \varepsilon_k) = 2 \sqrt{(1 - \varepsilon_k) \varepsilon_k} \leq \gamma$$ (15)

where $\gamma < 1$ depends on $\delta$ the maximum error. It follows that

$$\hat{L}_\phi(f_k) = \prod_{k=1}^k Z_{k'} \leq \gamma_k$$ (16)
4. The training set error decreases exponentially with $M$.
Note that $\phi(z) \geq 1_{z<0}$ for all $z$ (see also figure 1). Therefore

$$
\hat{L}(f^k) = \frac{1}{N} \sum_{i=1}^{N} 1_{[y_i f_k(x_i) < 0]}
\leq \frac{1}{N} \sum_{i=1}^{N} e^{-y_i f_k(x_i)} = \hat{L}_\phi(f^k) \leq \gamma^k
$$

(17)

In other words, the training error $\hat{L}(f^k)$ is bounded by a decaying exponential. Moreover, since $\hat{L}(f^k) \in \{0, 1/N, 2/N, \ldots, 1\}$, it follows that after a finite number of steps, when $\gamma^{k_0} < 1/N$, the training error will become 0 and the training data will be perfectly classified!

5. The test set error and overfitting Experimental results have shown however that one should NOT take $M$ to be equal to the above $k^0$; the number of steps for good generalization error is often much larger than $k^0$ (and sometimes smaller). The above shows a typical plot of $\hat{L}$ and $L$ (which can be estimated from an independent sample) vs the number of boosting iterations.

Myth: “Boosting doesn’t overfit”
Reality: Any algorithm overfits, including boosting. But in practice, overfitting occurs much later than predicted initially by the existing theory. The next section will show the more recent view of boosting, that explains why overfitting occurs only after apparently a very large number of parameters have been fit to the data.
2 Boosting as descent in function space

In brief, we will show that AdaBoost is gradient descent for the cost $L_{\phi}(f)$ with: $b^k$ the direction at step $k$, $\beta^k$ the step size.

2.1 Boosted predictors are additive models

An additive model (for prediction) has the form

$$f(x) \equiv E[Y|x] = \alpha + b_1(x_1) + b_2(x_2) + \ldots + b_n(x_n)$$  \hspace{1cm} (19)

In other words, it is a linear model, where each coordinate has been non-linearly transformed. A generalization of the above definition, which is still called an additive model, is

$$f(x) = \alpha + \beta_1 b_1(x) + \beta_2 b_2(x) + \ldots + \beta_M b_M(x)$$  \hspace{1cm} (20)

This is a linear model over a set of new features $b_{1:M}$.

Example 5 Linear model and neural net If $b_j = x_j$, $j = 1 : n$, the model (20) is a linear model.

If $b_j \in \left\{ \frac{1}{1+e^{-\gamma_i x_j}}, \gamma \in \mathbb{R}^n \right\} = B$ (the family of logistic functions with parameter $\gamma \in \mathbb{R}^n$) then $f(x)$ is a [two layer] neural network.

Additive Logistic Regression While the predictors above are well suited for regression, for classification one may employ a logistic regression, i.e.

$$f(x) \equiv \frac{P(Y = 1|x)}{P(Y = -1|x)} = \alpha + \beta_1 b_1(x) + \beta_2 b_2(x) + \ldots + \beta_M b_M(x)$$  \hspace{1cm} (21)

[Generalized Additive Models. Link function. To write]

[Alg 9.2 HTF for Additive Logistic Regression]

Given the base family $B$, data, and a loss function $L$, an additive model for prediction can be fit to data in several different ways.

- Fix $M$ from the start and optimize over all the parameters and base functions at once.
• Fix $M$ from the start but optimize only one $b_j, \beta_j$ at a time, keeping the others fixed. This method of training is a form of what is called backfitting.

• Optimize $b_j, \beta_j$ sequentially, for $j = 1, 2, \ldots$ without refitting previously fit base models. In this case, $M$ need not be fixed in advance. This method is called forward fitting. It turns out that this is what boosting does.

We will show that boosting is a form of (stochastic) gradient descent on the surrogate cost $\hat{L}_\phi$. Then, we discuss statistical properties of boosting, and show various results that indicate that boosting is good in the expected surrogate cost $L_\phi$ or in the expected true cost $L_{01}$.

### 2.2 AdaBoost is steepest descent on training set

Assume that our goal is to minimize the surrogate cost $\hat{L}_\phi$ on the training set. For any finite $D$, $f$ and $b \in B$ affect $\hat{L}_\phi$ only via the $N$-dimensional vectors of their values on $D$ (which we will abusively denote by $f, b$)

$$f = \begin{bmatrix} f(x^1) \\ f(x^2) \\ \vdots \\ f(x^N) \end{bmatrix}, \quad b = \begin{bmatrix} b(x^1) \\ b(x^2) \\ \vdots \\ b(x^N) \end{bmatrix}$$

(22)

Thus, $\hat{L}_\phi(f)$ is a function of $N$ variables, with partial derivatives

$$\frac{\partial \hat{L}_\phi}{\partial f(x^i)} = \frac{\partial}{\partial f(x^i)} \left[ \frac{1}{N} \sum_{i=1}^{N} \phi(y^i f(x^i)) \right] = \frac{1}{N} y^i \phi'(y^i f(x^i)) = -\frac{1}{N} y^i e^{-y^i f(x^i)},$$

(23)

since $\phi'(z) = -e^{-z}$. Imagine a boosting step as trying to find a change $\beta b$ in $f$ which minimizes the cost $\hat{L}_\phi(f + \beta b)$. This minimization is equivalent to maximizing the decrease in cost $\hat{L}_\phi(f) - \hat{L}_\phi(f + \beta b)$.

**The direction of descent** The change in $L_\phi$ along “direction” $b$ with step size $\beta$ is approximately

$$\hat{L}_\phi(f) - \hat{L}_\phi(f + \beta b) \approx - \left( \nabla f \hat{L}_\phi(f) \right)^T (\beta b) = \sum_i \left[ \left( \frac{1}{N} y^i e^{-y^i f(x^i)} \right) (\beta b(x^i)) \right] = \frac{\beta}{N} \sum_i y^i b(x^i) w_i$$

(24)
(denoting/recalling \( w_i \propto e^{-y_i f(x^i)} \)).

The best \( b \) is therefore the maximizer of

\[
\text{argmax}_{b \in B} \sum_i w_i y_i b(x^i)
\]  

(25)

where in the sum on the r.h.s we recognize the “centered error” \( r \) of AdaBoost. More generally (we will use this later), the direction \( b \) maximizes

\[
\sum_i y_i b(x^i) [-\phi'(y_i f(x^i))]
\]  

(26)

Finding the direction \( b \) is equivalent with step 1 of the AdaBoost algorithm, training a weak classifier on the weighted data (note that here the weights are not normalized by \( \prod Z^k \)). The resulting \( b \) can be seen as the best approximate of the gradient of \( L_\phi \) in \( B \).

**The line minimization.** Now let us do line minimization: find the optimal step size \( \beta \) in direction \( b \). For this we take the derivative of \( L_\phi(f + \beta b) \) w.r.t \( \beta \) and set it to 0.

\[
\frac{dL_\phi(f + \beta b)}{d\beta} = \sum_i y_i b(x^i) \phi'(y_i f(x^i)) = -\sum_i y_i b(x^i) e^{-y_i f(x^i) - \beta y_i b(x^i)}
\]

(27)

\( \beta \) is the (unique) root of

\[
\sum_i w_i y_i b(x^i) e^{-\beta y_i b(x^i)} = 0 \quad (28)
\]

If

- \( b(x) \in (-\infty, \infty) \) then \( \beta \) amounts to a rescaling of \( b \) and is redundant.
- \( b(x) \in [-1, 1] \) then the line optimization gives us \( \beta^k \) from AdaBoost.
- \( b(x) \in \{-1, 1\} \) then the line optimization gives us \( \beta^k \) from DiscreteAdaBoost.

- The first case corresponds to the RealAdaBoost in the FHT variant, described here for completeness.

**Real AdaBoost Algorithm** (in the FHT variant)

- **Assume** \( B \) contains real-valued functions
- **Input** \( M \), labeled training set \( \mathcal{D} \)
- **Initialize** \( f = 0 \)
- \( w_1^i = \frac{1}{N} \) weight of datapoint \( x^i \)
- **for** \( k = 1, 2, \ldots, M \)
  - “learn classifier for \( \mathcal{D} \) with weights \( w^k \rightarrow b^k \)”
  - compute new weights \( w_{k+1}^i = w_k^i e^{-y_i b^k(x^i)} \) and normalize them to sum to 1
- **Output** \( f(x) = \sum_{k=1}^M b^k(x) \)
• To get the AdaBoost updates we assume that $b(x) \in [-1, 1]$. Then $y^i b(x^i), \sum_i w_i y^i b(x^i) \in [-1, 1]$. We also make the approximation
\[ e^{-\beta z} \leq \frac{1 + z}{2} e^{-\beta} + \frac{1 - z}{2} e^\beta \] (29)
which is true whenever $z \in [-1, 1]$. This follows from the convexity of $e^{-\beta t}$ in the interval [-1,1] at point $t = (1 + z)/2$. By optimizing this upper bound w.r.t $\beta$ we get
\[ z = \sum_i w_i y^i f(x^i) \quad \beta = \frac{1}{2} \ln \frac{1 + z}{1 - z} \] (30)

• We can also assume $b(x) \in \{\pm 1\}$, the assumption of Discrete AdaBoost. In this case $y^i b(x^i) = \pm 1$ and we obtain
\[
\frac{dL_\phi(f + \beta b)}{d\beta} = \sum_{i \text{ corr}} w_i e^{-\beta} - \sum_{i \text{ err}} w_i e^\beta = 0
\] (31)
\[
0 = (1 - \sum_{i \text{ err}} w_i) - (\sum_{i \text{ corr}} w_i) e^{2\beta}
\] (32)
\[
\beta = \frac{1}{2} \ln \frac{1 - \varepsilon^k}{\varepsilon^k}
\] (33)

This gives us the $\beta^k$ coefficient of the Discrete AdaBoost algorithm (for alternative formulations of Discrete AdaBoost, this $\beta^k$ is up to a factor of 2 which does not affect the algorithm).

To finish the analysis, let us look at the updated $f$ and weights $w_i$.
\[
f \leftarrow f + \beta b
\] (34)
\[
w_i \leftarrow w_i e^{-\beta y^i f(x^i)}
\] (35)

Hence, the boosting algorithm given at the beginning of this section and many other variants can be seen as minimizing the cost $L_\phi(f)$ by steepest descent in the function space $B$. 

14
3 A statistical view of boosting (and more algorithms)

It has been shown [Friedman et al., 1999] (FHT) that boosting can also be seen as noisy gradient descent in function space when we replace the finite training set with the true data distribution. The cost function and gradient can be given a probabilistic interpretation. This point of view is useful in two ways:

1. It shows that boosting is asymptotically minimizing a reasonable cost function, so that we can expect the performance/and algorithm behavior on finite samples to be a good predictor on its behaviour with much larger samples.
2. It is an interpretation that allows on to create a very large variety of boosting algorithms, like the Logit and Gentle ADABoost presented hereafter.

We will consider the first point above, by now imagining that we do boosting “at the distribution level”, i.e using $P_{XY}$ instead of the empirical distribution given by the sample.

Hence, in the previous sections, we assumed that “learning a classifier” means (practically, but see also Proposition 2 below) “find the best possible minimizer in $b^k \in B$ to $\sum_{i=1}^{N} w^k_i e^{-y^i b(x^i)}$”. Here we will assume that it means (theoretically, see Proposition 2) “find the best possible approximation in $B$ to $\frac{1}{2} \log \frac{P_{y=1|x}}{P_{y=-1|x}}$.

The cost function for boosting is now $L_{\phi}(f) = E[e^{-yf(x)}]$. The notation $E[]$ represents the expectation w.r.t the joint $P_{XY}$ distribution. This is used in the proofs, while in practice it is replaced by the average over the data set.

**Proposition 1** Denote $p_x = P(y = 1|x)$. The cost $L_{\phi}(f)$ is minimized by

$$f^*(x) = \frac{1}{2} \ln \frac{P(y = 1|x)}{P(y = -1|x)} = \frac{1}{2} \ln \frac{p_x}{1-p_x}$$

And

$$p_x = \frac{e^{f(x)}}{e^{f(x)} + e^{-f(x)}} \quad P(y = -1|x) = \frac{e^{-f(x)}}{e^{f(x)} + e^{-f(x)}} = 1 - p_x \quad (36)$$
Proof Since we are minimizing over all possible $f$’s with no restrictions, we can minimize separately for every $f(x)$. Hence, let $x$ be fixed

$$E_{P[y|X=x]}[e^{-yf(x)}] = P(y = 1|x)e^{-f(x)} + P(y = -1|x)e^{f(x)}$$

and the gradient is

$$\frac{\partial E[e^{-yf(x)}|x]}{\partial f(x)} = -P(y = 1|x)e^{-f(x)} + P(y = -1|x)e^{f(x)}$$

By setting this to 0 the result follows.

Proposition 2 The Real AdaBoost (with “learn a classifier” defined at the distribution level) algorithm fits an additive logistic regression model $f$ by iterative optimization of $L_\phi(f)$.

Proof Suppose we have a current estimate $f(x)$ and seek to improve it by minimizing $L_\phi(f + b)$ over $b$. In the proof we assume that $b$ is an arbitrary function, while in practice $b$ will be chosen to best approximate the ideal $f$ within the class $\mathcal{B}$.

Denote by $p_x = P[y = 1|x]$ (the true value) and by $\hat{p}_x$ the “estimate”

$$\hat{p}_x = \frac{e^{f(x)}}{e^{f(x)} + e^{-f(x)}}$$

Assume again $x$ is fixed. Then, by a similar reasoning as above we have

$$L_\phi(f + b) = E[e^{-yf(x)} - yb(x)]$$

$$= E[e^{-f(x)}e^{-b(x)}p_x + (1 - p_x)e^{f(x)}e^{b(x)}]$$

By taking the derivative and setting it to 0 in a similar way to the previous proof, we obtain that the new step is

$$b(x) = \frac{1}{2} \ln \frac{p_x e^{-f(x)}}{(1 - p_x)e^{f(x)}} = \frac{1}{2} \left[ \ln \frac{p_x}{1 - p_x} - \ln \frac{\hat{p}_x}{1 - \hat{p}_x} \right]$$

Note that if one could exactly obtain the $b$ prescribed by (38) the iteration would not be necessary.

More interesting than the exact form of $b$ above is the optimization problem that leads to it.
Denote \( w(x, y) = e^{-yf(x)} \). Then, \( b \) is the solution of

\[
  b = \arg\min_{b \in \mathcal{B}} \mathbb{E}_{P_{XY}w(X,Y)}[e^{-Yb}] 
\]  

(39)

where \( P_{XY}w(X,Y) \) denotes the (unnormalized) twisted distribution obtained by multiplying the original data distribution with \( w(x, y) \). (Of course, one may have to put some restrictions on \( P_{XY} \) and \( \mathcal{B} \) in order to obtain a proper distribution.) Finally, note that the new \( f \) is \( f + b \) and the new weights are \( w(x, y)e^{-yb(x)} \) which finishes the proof.

Hence, the Real AdaBoost algorithm can be seen as a form of “noisy gradient” algorithm at the distribution level. Note the absence of the coefficient \( \alpha \) in the FHT formulation of Real AdaBoost. That’s because in the Schapire and Singer version the minimization was first done by finding a direction, then optimizing for step size, while in the FHT version the minimization in equation (39) is over both direction and scale of \( f \).

3.1 Why the \( e^{-yf} \) cost? and other \( L_\phi \) costs

- the “true” classification cost is \( 1_{yf > 0} \) (called the 0–1 cost) is nonconvex, nonsmooth (has 0/no derivatives). We would like a surrogate cost to have the following properties (satisfied by \( e^{-z} \))
- \( \phi(z) \) is an upper bound of the 0–1 cost (this helps prove bounds of the form \( \hat{L} \leq \hat{L}_\phi \))
- \( \phi(z) \) is smooth (has continuous derivatives of any order if \( f \) has them); (this lets us use continuous optimization techniques to fit the classifier)
- \( \phi(z) \) is convex (this leads to global optimization, which has been recognized as beneficial in practice; it also allows to prove bounds, rates of convergence and so on)
- \( \phi(z) \) is monothone (decreasing) (thus, when \( z > 0 \), driving the margins to increase even if the classification is correct). It is sometimes an advantage to have \( \phi(z) \) decreasing for all \( z < 0 \), and sometimes a disadvantage (e.g., causes overfitting)

The following algorithm are derived by using different \( \phi \) functions, and different descent strategies.
3.1.1 LogitBoost

Log likelihood cost of classification

\[ L_{\text{logit}}(f) = E_{P_X} [P(y = 1|x) \ln \hat{P}(y = 1|x) + P(y = -1|x) \ln \hat{P}(y = -1|x)] \]

(40)

where \( \hat{P} \) is the class probability predicted by the model \( f \), as given in equation (36). We shall find the following notation useful:

\[ y^* = \frac{y + 1}{2} \in \{0, 1\} \quad \text{for} \quad y \in \{\pm 1\} \]

(41)

LogitBoost Algorithm

\textbf{Assume} \( \mathcal{B} \) contains real-valued functions

\textbf{Input} \( M \), labeled training set \( \mathcal{D} \)

\textbf{Initialize} weights \( w_i = \frac{1}{N}, f(x) = 0 \), class probabilities \( \hat{P}_i = \frac{1}{2} \)

for \( k = 1, 2, \ldots M \)

- compute \( w_i = \hat{P}_i(1 - \hat{P}_i) \quad z_i = \frac{y^* - \hat{P}_i}{w_i} \)
- fit \( b^k(x) \) to \( z_i \) by weighted least squares, i.e. minimize \( \sum_i w_i (b(x^i) - z_i)^2 \)
- update \( f(x) \leftarrow f(x) + \frac{1}{2} b^k(x) \) and \( \hat{P}_i \leftarrow \frac{e^{f(x^i)}}{e^{f(x^i)} + e^{-f(x^i)}} \)

\textbf{Output} sign \( f(x) \)

Proposition 3 The LogitBoost algorithm uses (approximate) Newton-
Raphson steps to maximize the log-likelihood of the data $L_{\text{logit}}(f)$.

**Proof** We can rewrite the log-likelihood as follows

\[
L_{\text{logit}}(f) = E[y^* \ln \frac{e^f}{e^f + e^{-f}} + (1 - y^*) \frac{e^{-f}}{e^f + e^{-f}}] \tag{42}
\]

Conditioning on $x$, we compute the first and second derivative of $L_{\text{logit}}(f + b)$ w.r.t $b(x)$ in $b(x) = 0$.

\[
s(x) = \frac{\partial E[2y^*(f(x) + b(x)) - \ln(1 + e^{2f(x) + 2b(x)})]}{\partial b(x)} \bigg|_{b(x) = 0} \tag{46}
\]

\[
= E[2y^* - \frac{2e^{2f(x) + 2b(x)}}{1 + e^{2f(x) + 2b(x)}}] \bigg|_{b(x) = 0} \tag{47}
\]

\[
= 2y^* - E \left[ \frac{2e^{2f(x)}}{1 + e^{2f(x)}} \right] \tag{48}
\]

\[
H(x) = \frac{\partial^2 E[2y^*(f(x) + b(x)) - \ln(1 + e^{2f(x) + 2b(x)})]}{\partial b(x)^2} \bigg|_{b(x) = 0} \tag{50}
\]

\[
= \frac{\partial \left\{ 2y^* - E \left[ \frac{2e^{2b(x)}}{1 + e^{2b(x)}} \right] \right\}}{\partial b(x)} \bigg|_{b(x) = 0} \tag{51}
\]

\[
= 2E \left[ \frac{-2e^{2f(x) + 2b(x)}}{(1 + e^{2f(x) + 2b(x)})^2} \bigg|_{b(x) = 0} \right] \tag{52}
\]

\[
= -4E \left[ \frac{-2e^{2f(x)}}{(1 + e^{2f(x)})^2} \bigg|_{x} \right] \tag{53}
\]

\[
= -4E \left[ \hat{P}(x)(1 - \hat{P}(x)) \bigg|_{x} \right] \tag{54}
\]

Note that $H(x)$ represents only the diagonal of the (infinite-dimensional) Hessian for this problem. The objective needs to be maximized and the
$H(x)$ terms are negative. Therefore the (approximate) Newton-Raphson update is

$$f(x) \leftarrow f(x) - H(x)^{-1}s(x)$$  \quad \text{(55)}

$$= f(x) + \frac{1}{2} E[y^* - \hat{P}(x)]$$

$$= f(x) + \frac{1}{2} E \left[ \frac{y^* - \hat{P}(x)}{\hat{P}(x)(1 - \hat{P}(x))} \right]$$

$$= f(x) + \frac{1}{2} E_w \left[ \frac{y^* - \hat{P}(x)}{\hat{P}(x)(1 - \hat{P}(x))} \right]$$  \quad \text{(58)}

with $w(x) = \frac{\hat{P}(x)(1 - \hat{P}(x))}{E[\hat{P}(x)(1 - \hat{P}(x))]}$. The update above is equivalent to

$$\min_f E_w \left[ \left( f(x) + \frac{1}{2} \frac{y^* - \hat{P}(x)}{\hat{P}(x)(1 - \hat{P}(x))} - (f(x) + b(x)) \right)^2 \right]$$  \quad \text{(59)}

In practice we have the condition $b \in \mathcal{B}$ and the minimization is done over $\mathcal{B}$ and with the sample average replacing the expectation $E[\cdot]$.

Numerically

- it is better to enforce a lower threshold on the weights (e.g. $2 \times \epsilon_{\text{machine}}$) and to set the small weights either to the threshold value or to 0. The latter option can substantially speed up computation, especially for large data sets.
- compute $z_i$ as

$$z_i = \begin{cases} 
\min(1/\hat{P}_i, \hat{z}_{\text{max}}) & y = 1 \\
\min(1/(1 - \hat{P}_i), \hat{z}_{\text{max}}) & y = -1 
\end{cases}$$  \quad \text{(60)}

<table>
<thead>
<tr>
<th>Cost $\phi$</th>
<th>$e^{-yf}$</th>
<th>$y \ln p(f) + (1 - y) \ln(1 - p(f))$</th>
</tr>
</thead>
<tbody>
<tr>
<td>steepest + line search</td>
<td>DISCRETE</td>
<td>$p(f) = e^f/(e^f + e^{-f})$</td>
</tr>
<tr>
<td>Newton</td>
<td>GENTLE</td>
<td>LOGIT</td>
</tr>
</tbody>
</table>

### 3.1.2 AnyBoost and any cost $\phi$

Arbitrary cost: $\hat{L}_\phi(f) = \frac{1}{N} \sum_{i=1}^{N} \phi(y^i f(x^i))$
Learning algorithm: finds $b \sim -\frac{1}{N} \nabla \sum_{i=1}^{N} \phi(y^i f(x'))$

**ANYBoost Algorithm**

Initialize $f^0 = b^0$

for $k = 1, \ldots, M$

learn $b^k$

find $\beta_k$ by line search

$f^k = f^{k-1} + \beta_k b^k$

Properties

- can use better cost
- local optima

### 3.2 How to choose a cost $\phi$?

Can we analyze which cost functions $\phi$ are “better”? Can we offer some guarantees in terms of generalization bounds? The answers are in [Bartlett et al., 2006] Bartlett, Jordan & McAuliffe,”Convexity, classification and risk bounds”, 2005 (BJM).

We will restrict ourselves to convex, almost everywhere differentiable costs $\phi$ that are upper bounds of the 0-1 cost.

Let $p = P[Y = 1|X]$, $z = f(X)$. Then the expected cost of classification at $X$ is

$$C_p(z) = p\phi(z) + (1 - p)\phi(-z) \quad (61)$$

and the optimal cost is

$$H(p) = \min b^* C_p(z) \quad \text{attained for } f^* = \frac{1}{2} \ln \frac{p}{1-p} \quad (62)$$

Let $H^-$ denote the smallest cost for a misclassification

$$H^-(p) = \inf_{\text{sgn}z = -\text{sgn}(2p - 1)} C_p(z) \quad (63)$$

Intuitively, we are minimizing $\phi$ instead of the “true” misclassification cost, and we want to measure how much we can be off when doing this. The following results say that we can bound the “true” loss $L_{01}(f)$ in terms of the $\phi$-loss $L_\phi$. 

21
We say $\phi$ is **classification calibrated** if $H^{-}(p) > H(p)$ for all $p \neq 1/2$. For $\phi$ convex, we have that $\phi$ is classification calibrated iff $\phi$ differentiable at 0 and $\phi'(0) < 0$.

**Proposition 4** *(Theorem 4 in BJM)* If $\phi$ is classification calibrated and convex, then for any classifier $F$

$$\psi(L_{01}(f) - L_{01}^*) \leq L_{\phi}(f) - L_{\phi}^*$$

(64)

where $L_{\phi}^*, L_{01}^*$ represent respectively the optimal $\phi$-loss and optimal classification loss on the given data distribution and $\psi$ is

$$\psi(\theta) = \phi(0) - H\left(\frac{1 + \theta}{2}\right)$$

(65)

<table>
<thead>
<tr>
<th>Loss function $\phi(z)$</th>
<th>Transform function $\psi(\theta)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>exponential: $e^{-z}$</td>
<td>$1 - \sqrt{1 - \theta^2}$</td>
</tr>
<tr>
<td>truncated quadratic: $(\max(1 - z, 0))^2$</td>
<td>$\theta^2$</td>
</tr>
<tr>
<td>hinge: $\max(1 - z, 0)$</td>
<td>$</td>
</tr>
</tbody>
</table>

In BJM there are also more general theorems that do not assume $\phi$ is convex.

Furthermore, a convergence rate bound is given, which depends on: the noise in the labels, a complexity parameter of the function class $B$, the curvature of $\phi$. By optimizing this expression w.r.t to $B$ and $\phi$ one can theoretically choose the loss function and/or the base classifier.

### 3.3 Consistency of AdaBoost

[Bartlett and Traskin, 2007] proved that a large family of boosting algorithms is consistent. Below is an informal version of their main theorem.

Consistency is defined as $L_{01}(f^{t_N}) \rightarrow L_{01}^*$ for $N \rightarrow \infty$ and a $t_N$ a certain sequence of stopping times that tends to infinity with $N$.

The conditions for consistency are as follows:

1. The function $\phi$ is convex, lower bounded (by 0) and calibrated as in section 3.2
2. The boosting step satisfies a weak leaning condition

$$\hat{L}_{\phi}(f^k) \leq \gamma \inf_b \hat{L}_{\phi}(f^{k-1} + ab) + (1 - \gamma)\hat{L}_{\phi}(f^{k-1})$$

(66)
3. The set $B$ is rich enough that the Bayes cost $L^*_\phi$ can be attained by convex combinations in $B$. There is a sequence $\bar{f}^k$ of $k$ terms from $B$ whose $L^*_\phi$ tends to $L^*_\phi$

4. The empirical cost $\hat{L}_\phi$ converges to $L_\phi$ when $N \to \infty$ uniformly over all $f$ which are $t_N$ combinations

5. The empirical risks of $\bar{f}^k$ converge, i.e $\max\{0, |\hat{L}_\phi(\bar{f}^N) - L_\phi(\bar{f}^N)|\} \to 0$, a.s. when $N \to \infty$

6. Algorithmic convergence $\max\{0, |\hat{L}_\phi(f^N) - \hat{L}_\phi(\bar{f}^N)|\} \to 0$, a.s. when $N \to \infty$. In other words, the boosting algorithm produces an approximate minimizer of the $L_\phi$ risk if run for $t_N$ iterations

7. $t_N = n^{1-\varepsilon}$ for some $\varepsilon \in (0, 1)$ (e.g. increases slowly with $N$).

...and a loss bound

The following result applies to any classifier, but it was developed in response to the idea that “boosting increases the margin” (which we now know is often, but not always true). It proves essentially that large margins counter overfitting. As with all worst-case bounds, the bounds are usually not realistic or practically applicable.

**Proposition 5 (to find citation)** Let $F$ be a model class of VC-dimension $h$, with $f(x) \in [-1, 1]$ for all $x$ and for all $f \in F$. Let $\delta > 0$ and $\theta \in (0, 1)$. Then, with probability w.p. $> 1 - \delta$ over training sets

$$L_{01}(f) \leq \frac{1}{N}|\{i\, | \, |y^i f(x^i)| \leq \theta\}| + \tilde{O}\left(\frac{\sqrt{h}}{N\theta^2}\right)$$

(67)

for any $f \in F$.

This theorem upper bounds the true loss $L_{01}(f)$ using the number of margin errors for an arbitrary margin $\theta$. Note that for $\theta = 0$ a margin error is also a classification error, and for $\theta > 0$ the number of margin errors is greater or equal to that of classification errors. Hence, the first term of the bound increases with $\theta$, while the second term decreases. So, if most examples can be classified with a large margin (not necessarily 1), then the bound of Theorem 5 can be tighter.
3.4 When to stop boosting?

The idea of Cross-Validation (CV) is to use an independent sample from $P_{XY}$, denoted $D'$ and called the validation set to estimate the expected loss $L_{01}(f)$. When overfitting starts, $L_{01}(f^k)$ will start increasing with $k$. Boosting is stopped at the value $M$ that minimizes $\hat{L}_{01}(b^k; D')$ (denoted $L_{cv}$ below to simplify notation).

**AdaBoost with Cross-Validation**

Given Training set $\mathcal{D}$ of size $N$, validation set $\mathcal{D}'$ of size $N'$, base classifier $\mathcal{B}$

Initialize

1. while $L_{CV}$ decreases (but for at least 1 step)
   - do a round of boosting on $\mathcal{D}$
   - for $i' = 1 : N'$ compute $f(x^{i'}) \leftarrow f(x^{i'}) + \beta_k b_k(x^{i'})$
   - compute $R_{CV} = \frac{1}{N'} \sum_{i'} 1[y^{i'} f(x^{i'}) < 0]$

3.5 Other practical aspects

**Overfitting in noise** When the classes overlap much (many examples in $\mathcal{D}$ hard/impossible to classify correctly) boosting algorithms tend to focus too much on the hard examples, at the expense of overall classification accuracy. Observe also the cost function(s) in figure 1.

**Choice of features** Often times, the base class $\mathcal{B}$ consists of function of the form $f(x) = x_j - a$, which perform a split on coordinate $x_j$ at point $x_j = a$. They have the advantage that they can be learned and evaluated extremely fast. One can also augment the coordinate vector $x$ with functions of the coordinates (e.g. $x \rightarrow [x_1 \ldots x_d x_1x_2 x_1x_3 \ldots]$) essentially creating a large set of features, which corresponds to finite but very large $\mathcal{B}$. In such a situation, the number of features $d$ can easily be larger than $M$ the number of $b$’s in the final $f$. Thus, boosting will be implicitly performing a feature selection task.
4 Extensions of boosting

4.1 Multiplicative updates algorithms

Boosting also can be seen as part of a larger class of multiplicative updates algorithms. This area is under intense development, especially for algorithms at the frontier between game theory and computer science. For a general and clear discussion of these algorithms see [Arora et al., ] “The Multiplicative updates method” by Arora, Hazan and Kale.

Here is a very simple example: We have “experts” \( b^{1:M} \) who can predict the stock market (with some error). The predictions in this problem are binary, i.e \{up, down\} We will also try to predict the stock market, by combining their predictions in the function \( f = \sum_{k} w^{k}b^{k} \). The following algorithm learns \( f \) by optimizing the weights \( w^{k} \).

**Weighted Majority Algorithm**

1. Initialize \( w^{0} \leftarrow 1 \)
2. for \( t = 1, 2, \ldots \)
   1. \( w^{t} \leftarrow w^{t-1}(1 - \epsilon) \) if expert \( i \) makes a mistake at time \( t \)
   2. predict the outcome that agrees with the weighted majority of the experts

It can be shown that the number of mistakes \( m^{t} \) of \( f \) up to time \( t \) is bounded by

\[
 m^{t} \leq \frac{2 \ln M}{\epsilon} + 2(1 + \epsilon)m^{t}_{j} \tag{68}
\]

where \( m^{t}_{j} \) is the number of mistakes of any expert \( j \). Thus, asymptotically, the number of mistakes of the algorithm is about twice those of the best expert.

For a more general algorithm, that includes the above case, Arora & al prove that to achieve a tolerance \( \delta \) w.r.t to the optimal average cost, one needs to make \( O(\ln M/\delta^{2}) \) updates.

Another example that falls under the same framework is the Covering (Admissibility) Linear Program problem with an oracle. The task is to find a point \( x \in \mathbb{R}^{n} \) satisfying \( M \) linear constraints given by

\[
 Ax \geq b, \quad A \in \mathbb{R}^{M \times n}, \quad b \in \mathbb{R}^{M} \tag{69}
\]
We have an oracle which, given a single constraint $c^T x \geq d$ returns a point $x$ satisfying it whenever the constraint is feasible. It is assumed that the oracle’s responses $x$ satisfy $A_i x - b_i \in [-\rho, \rho]$ for all rows $i$ of $A$ and that $\rho$ is known.

We run the multiplicative updates algorithm for $T$ steps, where $T \propto \rho^2$ as follows

\begin{enumerate}
\item Call ORACLE with $c = \sum_i w_i A_i$, $d = \sum_i w_i b_i$ and obtain $x^t$
\item Penalty for equation $i$ is $r_i^t = A_i x^t - b_i$
\item Update weights by
\end{enumerate}

\[
 w_i^{t+1} \leftarrow w_i^t (1 - \epsilon \cdot \text{sign} r_i^t |r_i^t|)
\]  \hspace{1cm} (70)

with $\epsilon = \frac{\delta}{4\rho}$ and renormalize the weights.

\textbf{Output} $x = \sum_i x^t / T$

In [Arora et al.,] it is shown (Exercise: prove it based on the initial assumptions!) that (1) if ORACLE returns a feasible $x^t$ at all steps, then $x$ satisfies $A_i x - b + \delta \geq 0$ i.e the system is satisfied with tolerance $\delta$; (2) if ORACLE declares infeasibility in some step, then the program is infeasible.

4.2 Boosting for multiclass and ranking

\textbf{Multilabel classification} is a setting where an example $x$ can have multiple labels, represented by the set $Y(x)$, from a given finite set $\mathcal{Y}$, with $|\mathcal{Y}| = L$. One remaps the set of labels to the binary vector $y \in \{\pm 1\}^L$ by

\[
 y_l = \begin{cases} 
 1 & \text{if } l \in Y(x) \\
 -1 & \text{if } l \notin Y(x)
\end{cases} \text{ for } l \in \mathcal{Y}
\]  \hspace{1cm} (71)

There is a weight $w_{il}$ for each example $i$ and each label $l \in \mathcal{Y}$

\textbf{AdaBoost.MH} []
Input $M$, labeled training set $D$

Initialize $f = 0$

$$w^{1}_d = \frac{1}{N}$$

for $k = 1, 2, \ldots M$

learn classifier $b^k_l$ on $D$ with weights $w^k_l$

predict label $y_i$, $l = 1 : L$

evaluate error $\varepsilon^k = \sum_{i=1}^{N} \sum_{l=1}^{L} w^k_i y_i b^k_l(x_i)$

calculate $\beta^k = \frac{1}{2} \ln \frac{1-\varepsilon^k}{\varepsilon^k}$

compute new weights $w^{k+1}_d = \frac{1}{Z^k} w^k_d e^{-y_i b^k_l(x)}$ and normalize them to sum to 1

Output $f(x) = [f_l(x)]_{l \in \mathcal{Y}} = \left[ \sum_{k=1}^{M} b^k_l(x) \right]_{l \in \mathcal{Y}}$

The error $\varepsilon^k$ is the sum of the errors on each label, in other words the Hamming distance between the true vector $y^i$ and the vector $[b^k_l(x^i)]_l$. This is symbolized by the “H” in the name of the algorithm.

A variant of AdaBoost.MH that uses Error Correcting Output Codes (ECOC) for multiclass single label classification is called AdaBoost.MO[].

**Ranking.** Often ranking is considered in a broader sense, that includes

- **ranking proper**: finding a total ordering of the items $x \in \mathcal{X}$ in a given set. This is done for example by search engines when they present web pages ranked by their relevance to the query.
- **rating**: giving each $x$ a label in a finite set $\mathcal{Y}$. This is the “Netflix” task, where each movie is rated from 1 to 5 stars.
- **retrieval**: each item is labeled either 1 (relevant) or 0 (irrelevant). This is the first task some search engines perform when they are given a query. Note that this task looks like a classification, but it differs in the fact that the $x$ items are not sampled iid.

Underlying assumptions (hidden in the algorithm below)

- The items $x$ are as usual represented by feature vectors in $\mathbb{R}^n$. Thus $x$ will represent in the same time a web page and the set of features describing this web page, which will be used to infer the page label.
- Typically, one gets a training set of many “queries”, each with an associated list of “documents” $x$. For the purpose of ranking, only pairs $x, x'$ which correspond to the same query and have different ranks must be considered. These pairs are called crucial pairs. Thus, an algorithm that learns how to rank data will look at pairs of $x$’s and the
differences in their labels in the same way as a classification algorithm looks at single \( x \) vectors and their ±1 labels.

Moreover, the algorithm will ignore the grouping by query. This will be OK because the algorithm will only see pairs that belong to the same query.

- The ranking is given by a real valued function \( f(x) \). In the ranking case, the items \( x \) are ordered by their values; in the second case \( f \) is constrained to take discrete values; in the third case \( \text{sign} \ f \) provides the label.

Below is an example of a boosting algorithm for ranking under these paradigms.

**RankBoost Algorithm**

**Initialize** \( w_{x,x'}^1 \propto 1 \) for all \((x, x')\) crucial pairs in the data. It is assumed that \( x \succ x' \), i.e. that \( x \) is ranked after \( x' \).

for \( k = 1, 2, \ldots K \)

1. train weak classifier \( b_k \) on the crucial pairs weighted by \( w_{x,x'}^k \).
2. calculate \( r_k = \sum_{(x,x')} w_{x,x'}^k (b_k(x) - b_k(x')) \); this is the average margin
3. calculate \( \beta_k = \frac{1}{2} \ln \frac{1+r_k}{1-r_k} \) (see also notes below)
4. update weights \( w_{x,x'}^{k+1} = \frac{1}{Z_k} w_{x,x'}^k e^{-\beta_k (b_k(x) - b_k(x'))} \) with \( Z_k \) a normalization constant

**Output** \( f(x) = \sum_{k=1}^K \beta_k b_k(x) \)

Remark: there is more than one way to choose \( \beta_k \), and they are analogous to the methods described on pages 11–12. The one in the algorithm above corresponds to the case \( b(x) \in [0, 1] \). For \( b(x)^k \in \{0, 1\} \) a formula similar to that for DiscreteAdaBoost is obtained. For \( b(x)^k \in \mathbb{R} \) it is recommended to minimize \( Z^k \) w.r.t \( \beta \) in order to obtain the best value. [Exercise: why?]

### 5 Conclusions

- There are many different methods for averaging classifiers. They have different effects and should be applied in different situations.
- Rule of thumb: Averaging is particularly recommended when \( \mathcal{B} \) is not the “true” classifier family for the given problem (we don’t expect a single classifier to perform well enough).
- Interpretability of the result of learning is generally lost by averaging.
References


