STAT 538 Lecture 4
Boosting and other Averaging Methods

©Marina Meilă
mmp@stat.washington.edu

First version May 2000. Revisions 2006, 2007, 2010, 2012

1 There's more than one way to average predictors

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

Denote $\mathcal{B} = \{b\}$ a **base classifier** family

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

f is real-valued even if the b^k's are ± 1 valued

- can reduce variance
- can reduce bias
- can compensate for local optima (a form of bias)
- if $b_1, b_2, \ldots b_M$ make independent errors, averaging reduces error. We say that b_1 and b_2 make **independent errors** $\Leftrightarrow P(b_1 \text{ wrong} \,|\, x) = P(b_1 \text{ wrong} \mid x,\, b_2 \text{ wrong})$

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)
- Bagging (briefly)
- Boosting

1.1 Bayesian averaging - a procedural summary

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.

Before seeing data	$P_0(b)$	prior distribution over f
After seeing $\mathcal{D}$	$P(b\|\mathcal{D})$	posterior distribution over f
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

$$P(y|x,\mathcal{D}) \ = \ \sum_{b\in\mathcal{B}} P(b|\mathcal{D})\mathbf{1}_{b(x)=y}$$

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 $b \in \mathcal{B}$ usually intractable. Practically, one samples a few b's: $b_1, b_2, \ldots b_M$

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

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

Even if we choose the $P_0(b)$ for computational convenience only, averaging can have benefic effects

- variance reduction

- bias reduction
- if there is enough data, the prior, even if “wrong”, is overriden by the likelihood

Usual priors

- uninformative prior - uniform in some parametrization
- “MDL” (Minimum Description Length) prior - penalizes complex models

  $$P_0(b) \;\propto\; 2^{-\#\text{bits to encode } b}$$

  # bits ∼

  - # splits in decision tree
  - degree of polynomial
  - “effective” number of parameters

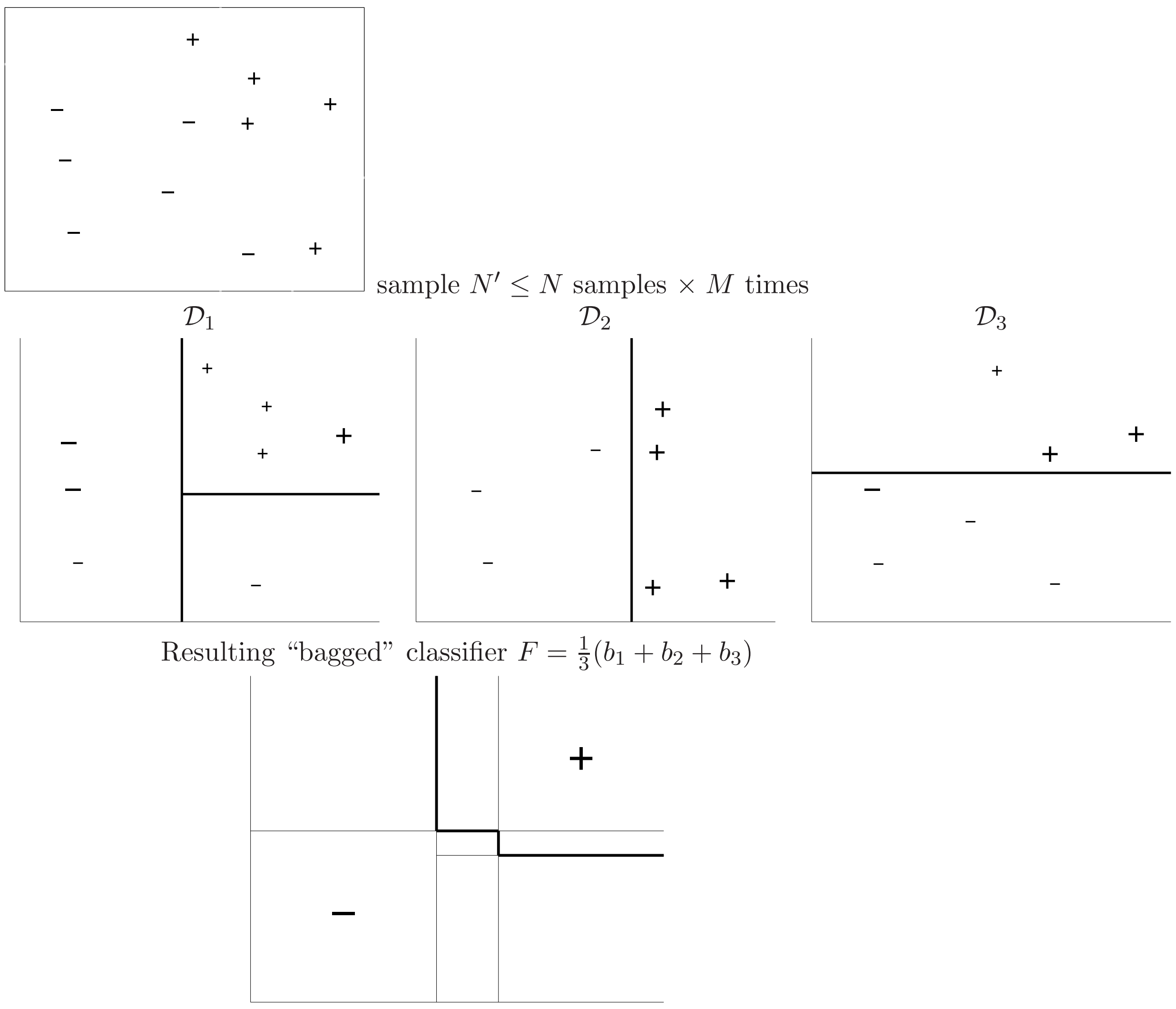
1.2 Reducing variance: Bagging

What if we had several (independently sampled) trainig sets $\mathcal{D}_1, \mathcal{D}_2, \ldots \mathcal{D}_M$?

- we could train classifiers $\{b^1, b^2, \ldots b^M\}$ on the respective $\mathcal{D}_1, \mathcal{D}_2, \ldots \mathcal{D}_M$
- we could estimate $E_{P(b)}[b] \;\sim\; f \;=\; \frac{1}{M}\Sigma_{k=1}^{M} b^k$
- f has always lower variance than b^k

Idea of **bagging**: sample $\mathcal{D}_1, \mathcal{D}_2, \ldots \mathcal{D}_M$ from the given $\mathcal{D}$ and estimate b^k on $\mathcal{D}_k$

$$f(x) \;=\; \operatorname{sign} \frac{1}{M} \sum_{k=1}^{M} b^k(x)$$


sample $N' \leq N$ samples $\times$ M times
$\mathcal{D}_1$
$\mathcal{D}_2$
$\mathcal{D}_3$
Resulting "bagged" classifier $F = \frac{1}{3}(b_1 + b_2 + b_3)$

Thus, bagging is a form of *boostrap.* 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 trees***. 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 Reducing bias: Boosting

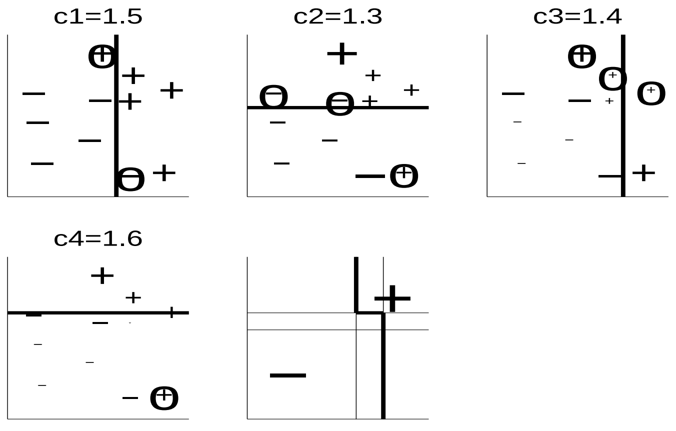
Base classifier family $\mathcal{B}$ has large bias (e.g. linear classifier, decision stumps) but learning always produces b that is better (on the training set) than random guessing.

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.
$$0 \;<\; \hat{L}(b) \;\leq\; \delta \;<\; \frac{1}{2}$$
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 2 *Boosting with* **stumps***, i.e. decision trees with a single ssplit ($c_1 \dots c_4$ are the coefficients $\beta_{1:4}$).*



```
ADABOOST ALGORITHM (Schapire-Singer variant)
    Assume      $\mathcal{B}$ contains functions $b$ taking values in $[-1,1]$ or $\{\pm 1\}$
    Input       $M$, labeled training set $\mathcal{D}$
    Initialize  $f = 0$
                $w_i^1 = \frac{1}{N}$ weight of datapoint $x_i$
    for         $k = 1, 2, \ldots M$
                1. "learn classifier for $\mathcal{D}$ with weights $w^k$" $\Rightarrow$ $b^k$
                2. compute "centered error" $r^k = \sum_i w_i^k y_i b^k(x_i) \in [-1,1]$
                then $\varepsilon^k = (1-r^k)/2,\ 1-\varepsilon^k = (1+r^k)/2$
                3. set $\beta^k = \frac{1}{2}\ln\frac{1+r^k}{1-r^k} = \frac{1}{2}\ln\frac{1-\varepsilon^k}{\varepsilon^k}$
                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 $y^i b(x^i) \in \{\pm 1\}$; if an error occurs $\frac{1-y^i b(x^i)}{2} = 1$; otherwise this expression is equal to 0. Thus, $\varepsilon^k = (1-r^k)/2$ "counts" the errors.
   If $b(x) \in [-1,1]$ then $r^k \in [-1,1]$ and the logarithm in step 3. is always defined.
2. If $b \in \{\pm 1\}$, then step 4 can be written equivalently (up to a multiplicative constant)

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

   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.
3. The logarithm in step 3 is > 0 whenever $\varepsilon^k < 1/2$.

1.4 Boosting - properties on the training set

1. An interpretation of the weights

$$w_i^k = \frac{1}{N} \prod_{k' \leq k} \frac{e^{-\beta^{k'} y_i b^{k'}(x_i)}}{Z_{k'}} = \frac{e^{-y_i f^{k-1}(x_i)}}{N \prod_{k' \leq k} Z^k} \tag{2}$$

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 (2) over i we obtain

$$1 \;=\; \frac{\sum_i e^{-y_i f^{k-1}(x_i)}}{N \prod_{k' \le k} Z^k}, \tag{3}$$

or

$$\prod_{k' \le k} Z^k \;=\; \frac{\sum_i e^{-y_i f^{k-1}(x_i)}}{N} \;\equiv\; \hat{L}_\phi(f^{k-1}) \tag{4}$$

where

$$\phi(z) \;=\; e^{-z}. \tag{5}$$

and

$$L_\phi(y, f(x)) \;=\; \phi(yf(x)) \tag{6}$$

Hence, the r.h.s of (4) is the average over the data set of the **exponential loss** L_ϕ.

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

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

For simplicity, we show this in the case of DiscreteAdaBoost

$$\begin{aligned}
Z^k &= \sum_{i=1}^{n} w_i^k e^{\beta^k \operatorname{sgn}(y_i b^k(x_i))} && (7)\\
&= e^{\beta^k} \underbrace{\sum_{i=err} w_i^k}_{\varepsilon^k} + e^{-\beta^k} \underbrace{\sum_{i=corr} w_i^k}_{1-\varepsilon^k} && (8)\\
&= e^{\beta^k}\varepsilon^k + (1-\varepsilon^k)e^{-\beta^k} && (9)\\
&= \sqrt{\frac{1-\varepsilon^k}{\varepsilon^k}}\varepsilon^k + \sqrt{\frac{\varepsilon^k}{1-\varepsilon^k}}(1-\varepsilon^k) \;=\; 2\sqrt{(1-\varepsilon^k)\varepsilon^k} \;\le \gamma && (10)
\end{aligned}$$

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

$$\hat{L}_\phi(f^M) \;=\; \prod_{k=1}^{M} Z^k \;\le\; \gamma^M \tag{11}$$

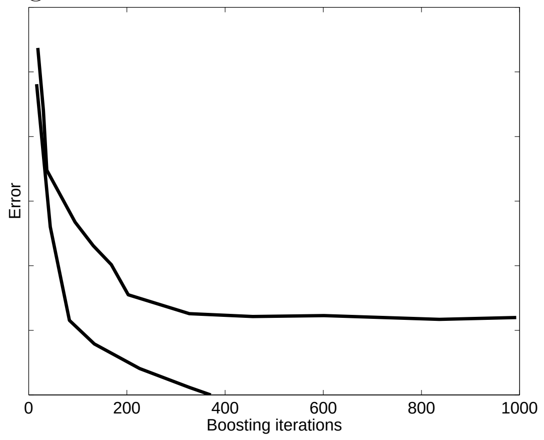
4. The training set error decreases exponentially with M.
Note that $\phi(z) \geq 1_{z<0}$ for all z (see also figure 1). Therefore

$$\hat{L}(f^k) = \frac{1}{N}\sum_{i=1}^{N} 1_{[y_i f^k(x_i)<0]} \tag{12}$$

$$\leq \frac{1}{N}\sum_{i=1}^{N} e^{-y_i f^k(x_i)} = \hat{L}_\phi(f^k) \leq \gamma^k \tag{13}$$

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.



6. 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.

7. **Boosting as gradient descent** is the topic of the next section. In brief, AdaBoost is gradient descent in $L_\phi(f)$ with

- b^k the direction at step k
- β^k the step size

2 Boosting as descent in function space

2.1 Boosted predictors are additive models

An *additive model* (for prediction) has the form

$$f(x) \;\equiv\; E[Y|x] \;=\; \alpha + b_1(x_1) + b_2(x_2) + \ldots + b_n(x_n) \tag{14}$$

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

$$f(x) \;=\; \alpha + \beta_1 b_1(x) + \beta_2 b_2(x) + \ldots + \beta_M b_M(x) \tag{15}$$

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

Example 3 Linear model and neural net *If $b_j = x_j,\ j = 1 : n$, the model (15) is a linear model.*

If $b_j \in \{\frac{1}{1+e^{-\gamma^T x}},\ \gamma \in \mathbb{R}^n\} = \mathcal{B}$ (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

$$f(x) \;\equiv\; \frac{P(Y=1|x)}{P(Y=-1|x)} \;=\; \alpha + \beta_1 b_1(x) + \beta_2 b_2(x) + \ldots + \beta_M b_M(x) \tag{16}$$

[Generalized Additive Models. Link function. To write]

[Alg 9.2 HTF for Additive Logistic Regression]

Given the base family $\mathcal{B}$, 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, β_j at a time, keeping the others fixed. This method of training is called *backfitting*.
- Optimize b_j, β_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_ϕ 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.

At step k, $f^k \equiv f$ is fixed and we want to find a b which minimizes the cost $\hat{L}_\phi(f+b) = \frac{1}{N}\sum_i \phi(y^i[f(x^i)+b(x^i)])$. We have

$$\left.\frac{\partial \hat{L}_\phi}{\partial b(x^i)}(f+b)\right|_{b=0} = y^i \phi'(y^i f(x^i)) \qquad (17)$$

The direction of descent We imagine b as a vector of values $[b(x^i)]$. Therefore the change in L_ϕ along "direction" b with step size β is approximately

$$\hat{L}_\phi(f+\beta b) - L(f) = \beta \sum_i b(x^i) y^i \phi'(y^i f(x^i)) \qquad (18)$$

The best b is the one that maximizes the (positive) decrease in $\hat{L}_\phi$, i.e the minimizer of

$$\sum_i y_i b(x^i)[-\phi'(y_i f(x^i))] \qquad (19)$$

If we replace now $\phi'(z) = -e^{-z}$ and denote $w_i = e^{-y_i f(x^i)}$ then (19) becomes

$$\operatorname*{argmax}_{b\in\mathcal{B}} \sum_i y_i b(x^i) e^{-y_i f(x^i)} = \operatorname*{argmax}_{b\in\mathcal{B}} \sum_i w_i y_i b(x^i) \qquad (20)$$

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_ϕ in $\mathcal{B}$.

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

$$\frac{dL_\phi(f+\beta b)}{d\beta} \;=\; \sum_i y_i b(x^i)\phi'(y_i f(x^i)) \;=\; -\sum_i y_i b(x^i) e^{-y_i f(x^i) - \beta y_i b(x^i)} \tag{21}$$

Set the above to 0 is equivalent to finding the (unique) root of

$$\sum_i w_i y_i b(x^i) e^{-\beta y_i b(x^i)} \;=\; 0 \tag{22}$$

If
- $b(x) \in (-\infty, \infty)$ then β amounts to a rescaling of b and is redundant.
- $b(x) \in [-1, 1]$ then the line optimization gives us β^k from AdaBoost
- $b(x) \in \{-1, 1\}$ then the line optimization gives us β^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   $\mathcal{B}$ contains real-valued functions
        Input   $M$, labeled training set $\mathcal{D}$
   Initialize   $f = 0$
                $w_i^1 \;=\; \frac{1}{N}$ weight of datapoint $x^i$
          for   $k = 1, 2, \dots M$
                "learn classifier for $\mathcal{D}$ with weights $w^k \;\Rightarrow\; b^k$"
                compute new weights $w_i^{k+1} \;=\; w_i^k e^{-y^i b^k(x^i)}$ 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

$$e^{-\beta z} \;\leq\; \frac{1+z}{2} e^{-\beta} + \frac{1-z}{2} e^{\beta} \tag{23}$$

  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 β we get

$$z \;=\; \sum_i w_i y^i f(x^i) \qquad \beta = \frac{1}{2} \ln \frac{1+z}{1-z} \tag{24}$$

- 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

$$\frac{d\hat{L}_\phi(f+\beta b)}{d\beta} = \sum_{i\ corr} w_i e^{-\beta} - \sum_{i\ err} w_i e^{\beta} = 0 \tag{25}$$

$$0 = (1 - \sum_{i\ err} w_i) - \underbrace{(\sum_{i\ err} w_i)}_{\varepsilon^k} e^{2\beta} \tag{26}$$

$$\beta = \frac{1}{2} \ln \frac{1-\varepsilon^k}{\varepsilon^k} \tag{27}$$

  This gives us the β^k coefficient of the DISCRETE ADABOOST algorithm (for alternative formulations of DISCRETE ADABOOST, this β^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.

$$f \leftarrow f + \beta b \tag{28}$$

$$w_i \leftarrow w_i e^{-\beta y^i f(x^i)} \tag{29}$$

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 $\mathcal{B}$.

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 performace/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 $b^k \in \mathcal{B}$ to $\sum_{i=1}^N w_i^k e^{-y^i b(x^i)}$". Here we will assume that it means (theoretically, see Proposition 2) "find the best possible approximation in $\mathcal{B}$ to $\frac{1}{2} \log \frac{P_w(y=1|x)}{P_w(y=-1|x)}$".

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

$$p_x = \frac{e^{f(x)}}{e^{f(x)} + e^{-f(x)}} \qquad P(y=-1|x) = \frac{e^{-f(x)}}{e^{f(x)} + e^{-f(x)}} = 1 - p_x \tag{30}$$

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"

$$\hat{p}_x \;=\; \frac{e^{b(x)}}{e^{b(x)}+e^{-b(x)}} \tag{31}$$

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

$$\begin{aligned} L_\phi(f+b) &= E[e^{-yf(x)-yb(x)}] \\ &= E[e^{-f(x)}e^{-b(x)}p_x + (1-p_x)e^{f(x)}e^{b(x)}] \end{aligned}$$

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

$$f(x) \;=\; \frac{1}{2}\ln\frac{p_x e^{-f(x)}}{(1-p_x)e^{f(x)}} \;=\; \frac{1}{2}\left[\ln\frac{p_x}{1-p_x} - \ln\frac{\hat{p}_x}{1-\hat{p}_x}\right] \tag{32}$$

Note that if one could exactly obtain the b prescribed by (32) 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

$$b \;=\; \underset{b\in\mathcal{B}}{\operatorname{argmin}} E_{P_{XY}w(X,Y)}[e^{-Yb}] \tag{33}$$

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 α 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 (33) is over both direction and scale of f.

3.1 Why the e^{-yf} cost? and other L_ϕ costs

- the "true" classification cost is $\mathbf{1}_{yf>0}$ (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 ϕ functions, and different descent strategies.

3.1.1 LOGITBOOST

Log likelihood cost of classification

$$L_{logit}(f) = E_{P_X}[P(y=1|x)\ln \hat{P}(y=1|x) + P(y=-1|x)\ln \hat{P}(y=-1|x)] \tag{34}$$

where $\hat{P}$ is the class probability predicted by the model f, as given in equation (30). We shall find the following notation useful:

$$y^* \;=\; (y+1)/2 \;\in\; \{0,1\} \text{ for } y \in \{\pm 1\} \tag{35}$$

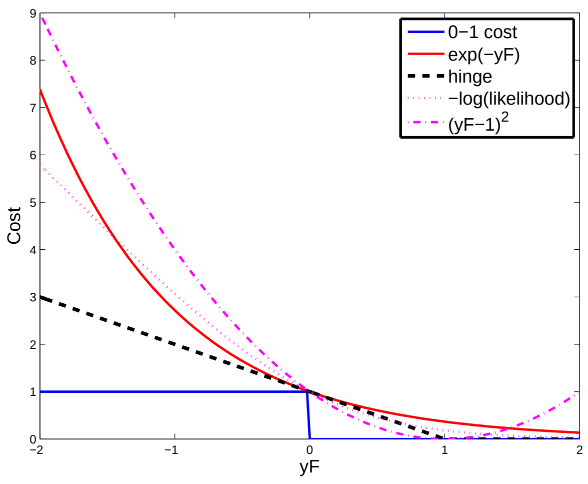


Figure 1: Examples of cost functions $\phi(z)$ for classification.

LogitBoost Algorithm

```
Assume      $\mathcal{B}$ contains real-valued functions
Input       $M$, labeled training set $\mathcal{D}$
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 $w_i = \hat{P}_i(1-\hat{P}_i)$   $z_i = \frac{y^{i*}-\hat{P}_i}{w_i}$
            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 $\hat{P}_i \leftarrow \frac{e^{f(x^i)}}{e^{f(x^i)}+e^{-f(x^i)}}$
Ouput       sign $f(x)$
```

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

Proof We can rewrite the log-likelihood as follows

$$L_{logit}(f) = E[y^* \ln \frac{e^f}{e^f + e^{-f}} + (1-y^*)\frac{e^{-f}}{e^f + e^{-f}}] \quad (36)$$

$$= E[(2y^* - 1)f - \ln(e^f + e^{-f})] \quad (37)$$

$$= E[(2y^* - 1)f + f - \ln(e^{2f} + e^0)] \quad (38)$$

$$= E[2y^* f - \ln(1 + e^{2f})] \quad (39)$$

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

$$\begin{aligned}
s(x) &= \left.\frac{\partial E[2y^*(f(x)+b(x)) - \ln(1+e^{2f(x)+2b(x)})]}{\partial b(x)}\right|_{b(x)=0} && (40)\\
&= E[2y^* - \left.\frac{2e^{2f(x)+2b(x)}}{1+e^{2f(x)+2b(x)}}]\right|_{b(x)=0} && (41)\\
&= 2y^* - E\left[\frac{2e^{2f(x)}}{1+e^{2f(x)}}\right] && (42)\\
&= 2(y^* - E[\hat{P}(x)]) && (43)\\
H(x) &= \left.\frac{\partial^2 E[2y^*(f(x)+b(x)) - \ln(1+e^{2f(x)+2b(x)})]}{\partial b(x)^2}\right|_{b(x)=0} && (44)\\
&= \left.\frac{\partial\left\{2y^* - E\left[\frac{2e^{2b(x)}}{1+e^{2b(x)}}\right]\right\}}{\partial b(x)}\right|_{b(x)=0} && (45)\\
&= 2E\left.\left[\frac{-2e^{2f(x)+2b(x)}}{(1+e^{2f(x)+2b(x)})^2}|x\right]\right|_{b(x)=0} && (46)\\
&= -4E\left[\frac{-2e^{2f(x)}}{(1+e^{2f(x)})^2}\middle| x\right] && (47)\\
&= -4E\left[\hat{P}(x)(1-\hat{P}(x))|x\right] && (48)
\end{aligned}$$

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

$$\begin{aligned}
f(x) &\leftarrow f(x) - H(x)^{-1}s(x) && (49)\\
&= f(x) + \frac{1}{2}\frac{E[y^* - \hat{P}(x)|x]}{E[\hat{P}(x)(1-\hat{P}(x))]} && (50)\\
&= f(x) + \frac{1}{2}E[\frac{y^* - \hat{P}(x)}{\hat{P}(x)(1-\hat{P}(x))}\frac{\hat{P}(x)(1-\hat{P}(x))}{E[\hat{P}(x)(1-\hat{P}(x))|x]}|x] && (51)\\
&= f(x) + \frac{1}{2}E_w[\frac{y^* - \hat{P}(x)}{\hat{P}(x)(1-\hat{P}(x))}|x] && (52)
\end{aligned}$$

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

$$\min_f E_w \left[\left(f(x) + \frac{1}{2} \frac{y^* - \hat{P}(x)}{\hat{P}(x)(1-\hat{P}(x))} - (f(x) + b(x)) \right)^2 \right] \tag{53}$$

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 $E[]$.

Numerically

- it is better to enforce a lower threshold on the weights (e.g. $2 \times \epsilon_{machine}$) 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

$$z_i = \begin{cases} \min(1/\hat{P}_i, z_{max}) & y = 1 \\ \min(1/(1\hat{P}_i), z_{max}) & y = -1 \end{cases} \tag{54}$$

Cost ϕ	e^{-yf}	$y \ln p(f) + (1-y)\ln(1-p(f))$ $p(f) = e^f/(e^f + e^{-f})$
steepest + line search	Discrete Real	–
Newton	Gentle	Logit

3.1.2 AnyBoost and any cost ϕ

Arbitrary cost: $L_\phi(F) = \sum_{i=1}^N \phi(y^i F(x^i))$

Learning algorithm: finds $f \sim -\nabla \sum_{i=1}^N \phi(y^i F(x^i))$

AnyBoost Algorithm

```
Initialize  $f^0 = b^0$
       for  $k = 1, \dots M$
            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 ϕ?

Can we analyze which cost functions ϕ are "better"? Can we offer some guarantees in terms of generalization bounds? The answers are in [Bartlett et al., 2006] Bartlett, Jodan & McAuliffe,"Convextity, classification and risk bounds", 2005 (BJM).

We will restrict ourselves to convex, almost everywhere differentiable costs ϕ 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

$$C_p(z) \ = \ p\phi(z) + (1-p)\phi(-z) \tag{55}$$

and the optimal cost is

$$H(p) \ = \ \min b^z C_p(z) \quad \text{attained for } f^* = \frac{1}{2}\ln\frac{p}{1-p} \tag{56}$$

Let H^- denote the smallest cost for a misclassification

$$H^-(p) \ = \ \inf_{\text{sgn}z=-\text{sgn}(2p-1)} C_p(z) \tag{57}$$

Intuitively, we are minimizing ϕ 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 ϕ-loss L_ϕ.

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

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

$$\psi(L_{01}(f) - L^*_{01}) \leq L_\phi(f) - L^*_\phi \tag{58}$$

where L^*_ϕ, L^*_{01} represent respectively the optimal ϕ-loss and optimal classification loss on the given data distribution and ψ is

$$\psi(\theta) \ = \ \phi(0) - H(\frac{1+\theta}{2}) \tag{59}$$

Loss function $\phi(z)$	**Transform function** $\psi(\theta)$
exponential: e^{-z}	$1 - \sqrt{1-\theta^2}$
truncated quadratic: $(\max(1-z,\ 0))^2$	θ^2
hinge: $\max(1-z, 0)$	$\|\theta\|$

In BJM there are also more general theorems that do not assume ϕ is convex.

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

3.3 A generic loss bound

TBW

3.4 When to stop boosting?

The idea of Cross-Validation (CV) is to use an idependent sample from P_{XY}, denoted $\mathcal{D}'$ 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 the stopped at the value M that minimizes $\hat{L}_{01}(b^k; \mathcal{D}')$ (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. $x \to [\, x_1 \ldots x_d \; x_1x_2 \; x_1x_3 \ldots])$ 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

**Initialize** $w_i^0 \leftarrow 1$
for $t = 1, 2, \ldots$
1. $w_i^t \leftarrow w_i^{t-1}(1-\epsilon)$ 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

$$m^t \;\le\; \frac{2\ln M}{\epsilon} + 2(1+\epsilon)m^t_j \tag{60}$$

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 δ w.r.t to the optimal average cost, one needs to make $\mathcal{O}(\ln M/\delta^2)$ 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

$$Ax \ge b, \qquad A \in \mathbb{R}^{M\times n},\ b \in \mathbb{R}^M \tag{61}$$

We have an oracle which, given a single constraint $c^T x \ge d$ 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 ρ is known.

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

```
   LINEAR PROGRAM WITH ORACLE parameters $\rho, \delta$
   Initialize $w_i = 1/M$ the weight of each constraint
   for $t = 1, 2, \ldots T$
1. Call ORACLE with $c = \sum_i w_i A_i$, $d = \sum_i w_i b^i$ and obtain $x^t$
2. Penalty for equation $i$ is $r^t_i = A_i x^t - b^i$
3. Update weights by
        $$w^{t+1}_i \;\leftarrow\; w^t_i (1 - \epsilon \cdot \operatorname{sign} r^t_i)^{|r^t_i|} \tag{62}$$
   with $\epsilon = \frac{\delta}{4\rho}$ and renormalize the weights.
   **Output** $x = \sum_t x^t / T$
```

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 $A_i x - b + \delta \ge 0$ i.e the system is satisfied with tolerance δ; (2) if ORACLE declares infeasibility in some step, then the program is infeasible.

4.2 Boosting for multiclass and ranking

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

$$y_l \;=\; \begin{cases} 1 & \text{if } l \in Y(x) \\ -1 & \text{if } l \notin Y(x) \end{cases} \qquad \text{for } l \in \mathcal{Y} \tag{63}$$

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

```
AdaBoost.MH []
     Input   $M$, labeled training set $\mathcal{D}$
Initialize   $f = 0$
             $w^1_{il} \;=\; \frac{1}{NL}$
       for   $k = 1, 2, \dots M$
             learn classifier $b^k_l$ on $\mathcal{D}$ with weights $w^k_{il}$ predict label $y_l$, $l = 1 : L$
             evaluate error $\varepsilon^k = \sum_{i=1}^{N} \sum_{l=1}^{L} w^k_{il} y^i_l b^k_l(x_i)$
             calculate $\beta^k = \frac{1}{2} \ln \frac{1-\varepsilon^k}{1+\varepsilon^k}$
             compute new weights $w^{k+1}_{il} \;=\; \frac{1}{Z^k} w^k_{il} e^{-y^i_l b^k_l(x^i)}$ and normalize them to sum to 1
    Output   $f(x) \;=\; [f_l(x)]_{l\in\mathcal{Y}} \;=\; [\sum_{k=1}^{M} b^k_l(x)]_{l\in\mathcal{Y}}$
```

The error ε^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 symbolyzed 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 sign f provides the label.

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

```
RankBoost Algorithm
Initialize $w^1_{x,x'} \propto 1$ 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 $w^k_{xx'}$
    2. calculate $r^k = \sum_{(x,x')} w^k_{xx'}(b^k(x) - b^k(x'))$; 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 $w^{k+1} = \frac{1}{Z^k} \sum_{(x,x')} w^k_{xx'} e^{\beta^k(b^k(x) - b^k(x'))}$ with $Z^k$ a nor-
       malization constant
Output $f(x) = \sum_{k=1}^{K} \beta^k b^k(x)$
```

Remark: there is more than one way to choose β^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 $b^k(x) \in \mathbb{R}$ it is recommended to minimize Z^k w.r.t β 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

[Arora et al.,] Arora, S., Hazan, E., and Kale, S. The multiplicative weights update method: a meta algorithm and applications.

[Bartlett et al., 2006] Bartlett, P. L., Jordan, M. I., and McAuliffe, J. D. (2006). Convexity, classification, and risk bounds. *Journal of the American Statistical Association*, 101(473):138–156.

[Bartlett and Traskin, 2007] Bartlett, P. L. and Traskin, M. (2007). Adaboost is consistent. *Journal of Machine Learning Research*, 8:2347–2368.

[Bauer and Kohavi, 1999] Bauer, E. and Kohavi, R. (1999). An empirical comparison of voting classification algorithms: bagging, boosting and variants. *MAchine learning*, 36:105–142.

[Breiman, 1994] Breiman, L. (1994). Bagging predictors. Technical report 421, Department of Statistics, University of California, Berkeley.

[Dietterich, 1999] Dietterich, T. G. (1999). An experimental comaprison of three methods for constructing ensembles of decision trees: bagging, boosting and randomization. *Machine learning*, pages 1–22.

[Dietterich, 2000] Dietterich, T. G. (2000). Ensemble methods in machine learning. In Roli, F., editor, *First International Workshop on Multiple Classifier Systems, Lecture Notes in Computer Science*, New York. Springer Verlag.

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

[Friedman et al., 1999] Friedman, J., Hastie, T., and Tibshani, R. (1999). Additive logistic regression: a statistical view of boosting. In *Advances in Neural Information Processing systems (NIPS)*, number 11. MIT Press. (see also technical report at http://www-stat.stanford.edu/ jhf/.

[Mason et al., 1999] Mason, L., Baxter, J., Bartlett, P., and Frean, M. (1999). Boosting algorithms as gradient descent in function space. Technical report RSISE, Australian national university.

[Mason et al., 2000] Mason, L., Baxter, J., Bartlett, P., and Frean, M. (2000). Boosting algorithms as gradient descent. In *Advances in Neural Information Processing systems (NIPS)*, number 12. MIT Press. (to appear).

[Schapire et al., 1998] Schapire, R. E., Freund, Y., Bartlett, P. L., and Lee, W. S. (1998). Boosting the margin: a new explanation for the effectiveness of voting methods. *Annals of statistics*, 26(5):1651–1686.

[Schapire and Singer, 1998] Schapire, R. E. and Singer, Y. (1998). Improved boosting algorithms using confidence-rated predictions. In *Proceedings of the Eleventh Annual conference on computational learning theory (COLT)*, pages 80–91.