Lecture 4: Combining predictors – Part II

Marina Meilă
mmp@stat.washington.edu

Department of Statistics
University of Washington

November, 2015

Boosting as descent in function space

- Boosted predictors are additive models
- AdaBoost is steepest descent on training set
- A statistical view of boosting

More surrogate losses, more boosting algorithms

- Why the e^{-yf} loss? other surrogate losses

Practical remarks and theoretical results

- Practicalities
- Theoretical results

Extensions of boosting

- Boosting for multiclass and ranking
- [Multiplicative updates algorithms]

Reading HTF: 15. Random Forests, 16. Ensembles of predictors, 8. Model inference and averaging, 10. Boosting, Murphy: 16.3, 16.3.1 Generalized Additive models (ignore the regularization, for "smoother" read "minimize loss function"), 16.4.1-5 [and optionally 8] Bosting, 3.2.4 Bayesian model averaging, 16.6.3, 16.6.1 Ensembles of predictors

Boosted predictors are additive models

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

$$f(x) \equiv E[Y|x] = \alpha + b_1(x_1) + b_2(x_2) + \ldots + b_n(x_n) \quad (1)$$

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

$$f(x) = \alpha + \beta_1 b_1(x) + \beta_2 b_2(x) + \ldots + \beta_M b_M(x) \quad (2)$$

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

Example (**Linear model and neural net**)

If $b_j = x_j$, $j = 1 : n$, the model (2) 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

$$f(x) \equiv \frac{P(Y=1|x)}{P(Y=-1|x)} = \alpha + \beta_1 b_1(x) + \beta_2 b_2(x) + \ldots + \beta_M b_M(x) \quad (3)$$

[Generalized Additive Models. Link function. See supplementary notes]

How to train an Additive Model?

[Alg 9.2 HTF for Additive Logistic Regression]
An additive model for prediction can be trained in several different ways.

Given base family $\mathcal{B}$, data $\mathcal{D}$, loss function L

- Fix M from the start and optimize over all the parameters and base functions at once.
- Backfitting Fix M from the start but optimize only one b_j, β_j at a time, keeping the others fixed
- Forward fitting Optimize b_k, β_k sequentially, for $k = 1, 2, \ldots$ without refitting previously fit base models. In this case, M need not be fixed in advance.
  It turns out that this is what boosting does.

AdaBoost is steepest descent on training set

We will show that boosting is a form of (stochastic) gradient descent on the surrogate loss $\hat{L}_\phi$ (we already know from Part I that AdaBoost pushes $\hat{L}_\phi$ asymptotically towards 0).

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

$$f = \begin{bmatrix} f(x^1) \\ f(x^2) \\ \dots \\ f(x^N) \end{bmatrix} \quad b = \begin{bmatrix} b(x^1) \\ b(x^2) \\ \dots \\ b(x^N) \end{bmatrix} \tag{4}$$

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

$$\frac{\partial \hat{L}_\phi}{\partial f(x^i)} = \frac{\partial}{\partial f(x^i)} \left[\frac{1}{N} \sum_{i=1}^{N} \phi(y^i f(x^i)) \right] = \frac{1}{N} y^i \phi'(y^i f(x^i)) = -\frac{1}{N} y^i e^{-y^i f(x^i)}, \tag{5}$$

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

The direction of descent

The change in L_ϕ along "direction" b with step size β is approximately

$$\hat{L}_\phi(f) - \hat{L}_\phi(f+\beta b) \approx -\left(\nabla_f \hat{L}_\phi(f)\right)^T(\beta b) = \sum_i \left[\left(\frac{1}{N} y^i e^{-y^i f(x^i)}\right)(\beta b(x^i))\right] \propto \sum_i y^i b(x^i) w_i \tag{6}$$

(denoting/recalling $w_i \propto e^{-y_i f(x^i)}$).
The direction of steepest descent b is therefore the maximizer of

$$\operatorname*{argmax}_{b \in \mathcal{B}} \sum_i w_i y_i b(x^i) \tag{7}$$

where in the sum on the r.h.s we recognize the r of AdaBoost.

- If $b(x^i) = \pm 1$ values, then $1 - y_i b(x^i) = \mathbf{1}_{[i \text{ error}]}$, and maximizing (7) is the same as minimizing the weighted training error $\hat{L}_{01}^w$.
- If b takes real values, then $y_i b(x^i)$ is the margin of example i, and maximizing (7) is a natural objectiv for many training algorithm. Exercise Can you find examples of algorithms/predictors which do/don't maximize the loss in (7)?

More generally (we will use this later), the direction b maximizes

$$\sum_i y_i b(x^i)[-\phi'(y_i f(x^i))] \tag{8}$$

Finding the direction b is equivalent with step 1 of the AdaBoost algorithm, training a weak classifier on the weighted data. 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 $\hat{L}_\phi(f + \beta b)$ w.r.t β and set it to 0.

$$\frac{d\hat{L}_\phi(f + \beta b)}{d\beta} = \sum_i y_i b(x^i)\phi'(y_i f(x^i)) = -\sum_i y_i b(x^i) e^{-y_i f(x^i) - \beta y_i b(x^i)} \quad (9)$$

β is the (unique) root of

$$\sum_i w_i y_i b(x^i) e^{-\beta y_i b(x^i)} = 0 \quad (10)$$

If
- $b(x) \in \{-1, 1\}$ then line optimization gives β^k from AdaBoost
- $b(x) \in [-1, 1]$ then line optimization gives β^k from AdaBoost approximately
- $b(x) \in (-\infty, \infty)$ then β amounts to a rescaling of b and is redundant.

Calculating β^k for binary b's

Assume $b(x) \in \{\pm 1\}$.
In this case $y^i b(x^i) = \pm 1$ and we obtain

$$\frac{d\hat{L}_\phi(f + \beta b)}{d\beta} = \sum_{i \ corr} w_i e^{-\beta} - \sum_{i \ err} w_i e^{\beta} = 0 \tag{11}$$

$$0 = (1 - \sum_{i \ err} w_i) - \underbrace{(\sum_{i \ err} w_i)}_{\varepsilon^k} e^{2\beta} \tag{12}$$

$$\beta = \frac{1}{2} \ln \frac{1 - \varepsilon^k}{\varepsilon^k} \tag{13}$$

This is the β^k coefficient of step 4 of AdaBoost

Hence, the AdaBoost algorithm can be seen as minimizing the loss $L_\phi(f)$ by steepest descent in the function space span $\mathcal{B}$.

RealAdaBoost

The third 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 $k = 1, 2, \ldots M$

"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 $f(x) = \sum_{k=1}^{M} b^k(x)$

A statistical view of boosting

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 loss 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 loss 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 LogitBost, Gentle AdaBoost and GradientBoost, presented hereafter.

Assume

- we do boosting "at the distribution level", i.e using P_{XY} instead of the empirical distribution given by $\mathcal{D}$.
- The loss function is $L_\phi(f) = E[e^{-yf(x)}]$.
  The notation $E[]$ denotes expectation w.r.t the joint P_{XY} distribution.
- learning a classifier means "find the best possible minimizer to $L_\phi(f)$"

Is L_ϕ a good loss?

Proposition

Denote $p_x = P_{XY}(y = 1|x)$. *The loss* $L_\phi(f)$ *is minimized by*

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

And $p_x \;=\; \frac{e^{f(x)}}{e^{f(x)}+e^{-f(x)}}$ *the logistic function.*

Exercise Does the expresion of p_x look familiar? What is the connection?

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. □

In summary f^* is the Bayes optimal predictor for L_ϕ. But by the Proposition, f^* is also Bayes optimal for L_{01}. **(Good!)**

Steepest descent on $L_\phi(f)$ is (like) RealAdaboost

Proposition

The Real AdaBoost (with "learn a classifier" defined at the distribution level) fits an additive logistic regression model f by iterative descent on $L_\phi(f)$.

Proof The proof is similar to that for the training set case.
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"

$$\hat{p}_x = \frac{e^{f(x)}}{e^{f(x)} + e^{-f(x)}} \tag{14}$$

Assume again x is fixed. Then,

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

Taking the derivative and setting it to 0 we obtain the new step:

$$b(x) = \frac{1}{2}\ln\frac{p_x e^{-f(x)}}{(1-p_x)e^{f(x)}} = \frac{1}{2}\left[\ln\frac{p_x}{1-p_x} - \ln\frac{\hat{p}_x}{1-\hat{p}_x}\right] \tag{15}$$

Note that if one could exactly obtain the b prescribed by (15) the iteration would not be necessary.

(Proof, continued)
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

$$b = \operatorname*{argmin}_{b \in \mathcal{B}} E_{P_{XY} w(X,Y)}[e^{-Yb}] \tag{16}$$

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 that the minimization in equation (16) is over both direction and scale of f.)

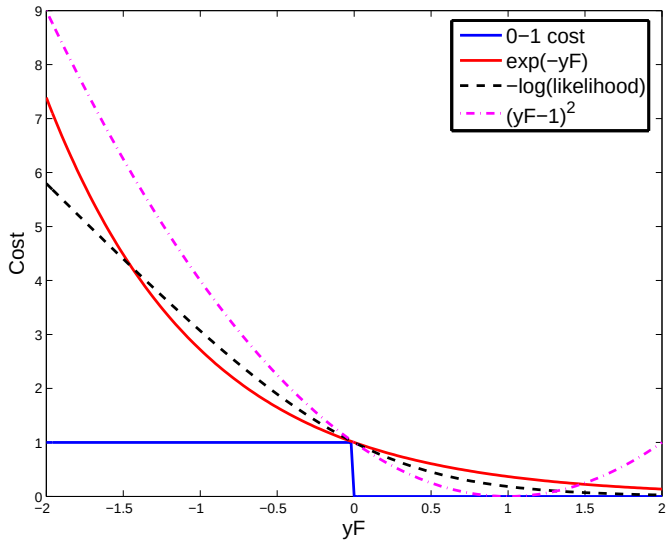
Why the e^{-yf} loss? and other L_ϕ losss

- We saw that L_ϕ is statistically motivated. Now we will see that it is computationally motivated as well.
- Recall: The "true" classification loss L_{01} is nonsmooth (has 0/no derivatives), non-convex. For training, one uses **surrogate losses**.
- Want surrogate L to have the following properties
  - $\phi(z)$ is an upper bound of the 0–1 loss
  - $\phi(z)$ 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).

  These properties are satisfied by $L_{exp}(z) = e^{-z}$

Surrogate losses and boosting algorithms

A cornucopia of loss functions



(sometimes good to have $L(z)$ decrease for all $z < 0$, and sometimes bad – causes overfitting)

. . . and of boosting algorithms

- *GentleAdaBoost*: approx Newton on L_{exp}
- LEAST-SQUARESBOOST: many operations in closed form
- LOGITBOOST $L_{logit}(z) = \ln(1 + e^{-z})$ slower (almost linear) decrease for $z \ll 0$
- ANYBOOST, GRADIENTBOOST work with any L

GradientBoost

GradientBoost Algorithm

Given $\mathcal{B}$ contains real-valued functions, loss L differentiable

Input M, labeled training set $\mathcal{D}$

Initialize $f^0(x) = \beta_0 = \text{argmin}_{\beta \in \mathbb{R}} \hat{L}(\beta)$

for $k = 0, 1, 2, \ldots M-1$

1. compute $r_i = -y^i \phi'(y^i f(x^i))$
2. fit $b^k(x)$ to outputs r_i
3. find $\beta_k = \text{argmin}_{\beta \in \mathbb{R}} \hat{L}(f^k + \beta b_k)$ (univariate optimization)

update $f^{k+1}(x) = f^k(x) + \beta^k b^k(x)$

Ouput $f^M(x)$

- Can be used for either classification or regression
- Works with any L
- If L convex, step 3 is convex optimization (efficient)
- Proposed first as AnyBoost, later specialized for $\mathcal{B}$ =decision/regression trees, with other tweaks and new name GradientBoost
- When $\mathcal{B}$ =CART
  - step 3 optimizes over every leaf separately
  - depth of trees J represents maximum number of interactions in f; should not be too large ($\mathcal{B}$ must be weak)

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. The same happens for outliers. Observe also that the loss function(s) in the previous figure, which penalize more as the margin becomes more negative.

Choice of features Often times, the base class $\mathcal{B}$ consists of function of the form $b(x) = x_j - a$, 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.

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 $L_{01}^{CV} = \frac{1}{N'} \sum_{i'} 1_{[y^{i'} f(x^{i'}) < 0]}$

How to choose a loss ϕ?

Can we analyze which loss 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 losss ϕ that are upper bounds of the 0-1 loss.
Let $p = P[Y = 1|X]$, $z = f(X)$. Then the expected loss of classification at X is

$$C_p(z) \;=\; p\phi(z) + (1-p)\phi(-z) \tag{17}$$

and the optimal loss is

$$H(p) \;=\; \min b^z C_p(z) \quad \text{attained for } f^* = \frac{1}{2} \ln \frac{p}{1-p} \tag{18}$$

Let H^- denote the smallest loss for a misclassification

$$H^-(p) \;=\; \inf_{\text{sgn}z = -\text{sgn}(2p-1)} C_p(z) \tag{19}$$

Intuitively, we are minimizing ϕ instead of the "true" misclassification loss, 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

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

$$\psi(L_{01}(f) - L_{01}^*) \leq L_\phi(f) - L_\phi^* \tag{20}$$

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

$$\psi(\theta) \;=\; \phi(0) - H\left(\frac{1+\theta}{2}\right) \tag{21}$$

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: $\max(1-z, 0)$	$\lvert\theta\rvert$

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.

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 $L_{01}(f^{t_N}) \to L_{01}^*$ for $N \to \infty$ 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 ϕ is convex, lower bounded (by 0) and calibrated as in section 2
2. The boosting step satisfies a weak leaning condition

$$\hat{L}_\phi(f^k) \leq \gamma \inf_b \hat{L}_\phi(f^{k-1} + \alpha b) + (1-\gamma)\hat{L}_\phi(f^{k-1}) \tag{22}$$

3. The set $\mathcal{B}$ is rich enough that the Bayes loss L_ϕ^* can be attained by convex combinations in $\mathcal{B}$. There is a sequence $\bar{f}^k$ of k terms from $\mathcal{B}$ whose L_ϕ tends to L_ϕ^*
4. The empirical loss $\hat{L}_\phi$ converges to L_ϕ 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 $\max\{0, |\hat{L}_\phi(f^{t_N}) - \hat{L}_\phi(\bar{f}^N)|\} \to 0$, a.s. when $N \to \infty$. In other words, the boosting algorithm produces an approximate minimizer of the L_ϕ 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 (to find citation)

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

$$L_{01}(f) \leq \frac{1}{N} |\{i \mid y^i f(x^i) \leq \theta\}| + \tilde{\mathcal{O}}\left(\sqrt{\frac{h}{N\theta^2}}\right) \tag{23}$$

for any $f \in \mathcal{F}$.

This theorem upper bounds the the true loss $L_{01}(f)$ using the number of margin errors for an arbitrary margin θ. 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 θ, while the second term decreases. So, if most examples can be classified with a large margin (not necessarily 1), then the bound of Theorem 4 can be tighter.

Multilabel classification

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

$$y_l = \begin{cases} 1 & \text{if } l \in Y(x) \\ -1 & \text{if } l \notin Y(x) \end{cases} \qquad \text{for } l \in \mathcal{Y} \tag{24}$$

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 $f = 0$

$w_{il}^1 = \frac{1}{NL}$

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

learn classifier b_l^k on $\mathcal{D}$ with weights w_{il}^k predict label y_l, $l = 1 : L$

evaluate error $r^k = \sum_{i=1}^N \sum_{l=1}^L w_{il}^k y_l^i b_l^k(x_i)$

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

compute new weights $w_{il}^{k+1} = \frac{1}{Z^k} w_{il}^k e^{-y_l^i b_l^k(x^i)}$ and normalize them to sum to 1

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

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 $[b_l^k(x^i)]_l$. 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 [].

Boosting for ranking

Ranking is considered in a broader sense, that includes

- ranking proper find a total ordering of the items $x \in \mathcal{X}$ (a given set). E.g when search engines present web pages ranked by their relevance to the query.
- **rating** assign each x a label in a finite set $\mathcal{Y}$. E.g. the "Netflix" problem, where each movie is rated from 1 to 5 stars.
- **retrieval** label each item either 1 (relevant) or -1 (irrelevant). E.g 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 represented by feature vectors in $\mathbb{R}^n$; e.g x denotes in same time a web page and the set of features describing this web page, which will be used to infer the page label.
- The training set $\mathcal{D}$ contains many "queries", each with an associated list of labeled "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.
- The algorithm ignores the grouping by query. OK because the algorithm only sees 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 $\operatorname{sgn} f$ provides the label.

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 $w^{k+1}_{xx'} = \frac{1}{Z^k} w^k_{xx'} e^{-\beta^k (b^k(x) - b^k(x'))}$ 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 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?]

Multiplicative updates algorithms

Boosting can be seen as part of a larger class of multiplicative updates algorithms. This area developed at the frontier between game theory and computer science, and in machine learning. For a general and clear discussion of these algorithms see [Arora et al.,] "The Multiplicative updates method".

Weighted majority 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 want 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

$$m^t \leq \frac{2 \ln M}{\epsilon} + 2(1+\epsilon) m_j^t \tag{25}$$

where m_j^t 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 et al.,] prove that to achieve a tolerance δ w.r.t to the optimal average loss, one needs to make $\mathcal{O}(\ln M/\delta^2)$ updates.

Feasibility problem for LP, with oracle

The problem is to find a point $x \in \mathbb{R}^n$ satisfying M linear constraints given by

$$Ax \geq b, \qquad A \in \mathbb{R}^{M \times n},\ b \in \mathbb{R}^M \tag{26}$$

The **oracle** is a blackbox 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 ρ is known.

LINEAR PROGRAM WITH ORACLE parameters ρ, δ
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 $r_i^t = A_i x^t - b^i$
3. Update weights by

$$w_i^{t+1} \leftarrow w_i^t (1 - \epsilon \cdot \mathrm{sgn} r_i^t)^{|r_i^t|} \tag{27}$$

with $\epsilon = \frac{\delta}{4\rho}$ and renormalize the weights.
Output $x = \sum_t x^t / T$

The number of steps $T \propto \rho^2$.
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 δ; (2) if ORACLE declares infeasibility in some step, then the program is infeasible.


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


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


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


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


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, 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, Y. and Schapire, R. E. (1997).