STAT 535

11/30/23

Lecture 18

Wide NN

Boosting

Lecture VII 1,2

Boosting

+ alg – adaboost
gradient boost

HW 6 due

NTK during training – empirical evidence

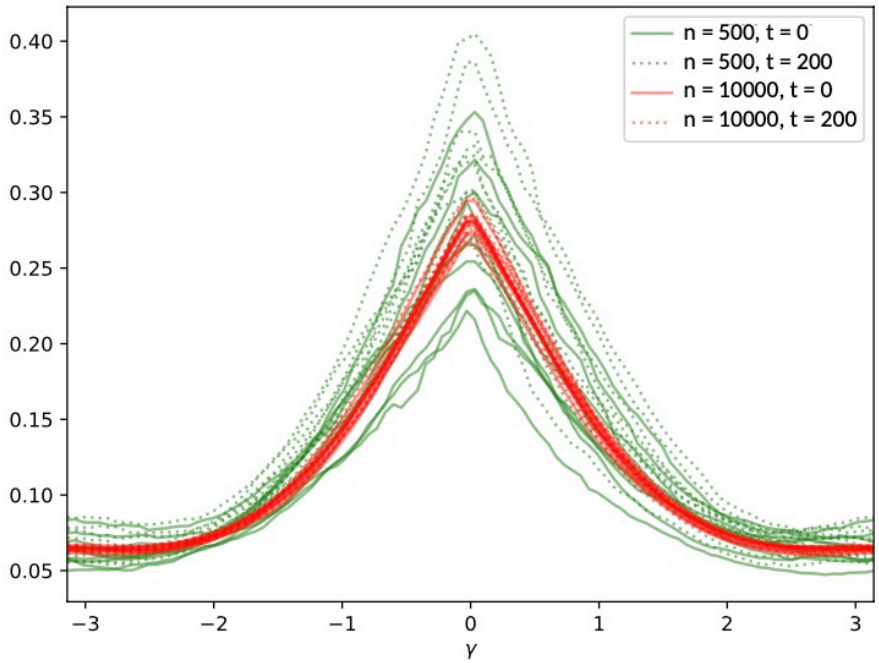


Figure 1: Convergence of the NTK to a fixed limit for two widths n and two times t.

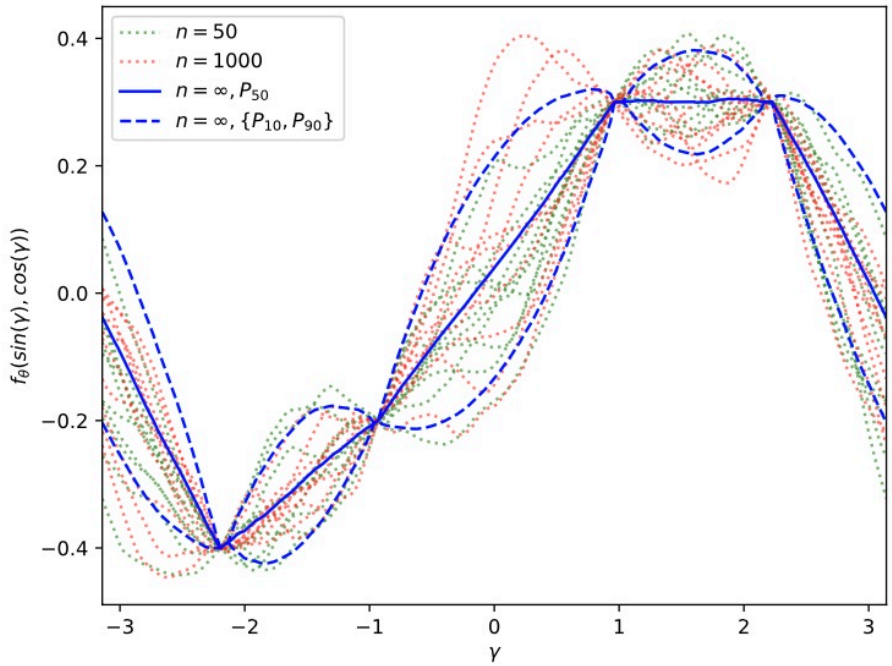


Figure 2: Networks function f_θ near convergence for two widths n and 10th, 50th and 90th percentiles of the asymptotic Gaussian distribution.

wide
1. NN ~ GP = init
≈ after train

Kernel = NTK

$K_\theta(x,x') = \nabla_\theta f_\theta(x)^T \nabla_\theta f_\theta(x')$

2. $K^{L,\varphi} \longrightarrow L \nearrow$ $K^{L,\varphi}$ smoother, $f \approx 0$
⇒ BAD for regression

Wide and deep neural networks for classification – Basic quantities and assumptions

[Radhakrishnan, Belkin, Ulher, 2022]

CLASSIFICATION

- This paper studies the limits of wide neural networks $m_l \to \infty$ for all $l = 1 : L$ when the depth $L \to \infty$
- It is already known that for regression $L \to \infty$ is NOT OPTIMAL
- Since the NTK depends only of the activation function ϕ, the limit shall only depend on ϕ as well.

of K

- In particular, the limit depends on ϕ only through the following

$$
\begin{array}{lll}
A = E[\phi(Z)] & \text{when } z \sim N(0,1) \\
A' = E[\phi'(Z)] & \text{when } z \sim N(0,1) \\
\\
B = E[(\phi'(Z))^2] & \text{when } z \sim N(0,1)
\end{array}
$$

- Classifier $f(x) = \lim_{L\to\infty} \operatorname{sgn} Y G^{-1} \kappa^L(X, x)$ with $G = [\kappa^L(x^i, x^j)]_{i,j=1:n}$.

Gram matrix

- Additional assumptions
  - Data $X \subseteq \mathcal{S}_+^d$, vectors of norm 1 with all entries ≥ 0.
  - Simplifying assumptions on NTK parameters (e.g. $\sigma_w = \sigma_b = 1$)

Case $A \neq 0$: Networks implement majority vote

Theorem (Proposition 1 in [Radhakrishnan, Belkin, Ulher, 2022])

If there is a function $0 < c(L) < \infty$ *so that*

$$\lim_{L\to\infty} \frac{\kappa^L(x,x')}{c(L)} = c_1 > 0 \text{ for any } x \neq x', \text{ and } \lim_{L\to\infty} \frac{\kappa^L(x,x)}{c(L)} \neq c_1, \tag{27}$$

then

$$\lim_{L\to\infty} f(x) = \operatorname{sgn} \sum_{i=1}^{n} y^i \qquad \textit{MAJORITY CLASSIFIER} \tag{28}$$

- What ϕ's satisfy theorem? ReLU, all ϕ with $B \neq 1$.

Case $A = A' = 0$: Networks implement 1-nearest neighbor

Theorem (Theorem 3 in [Radhakrishnan, Belkin, Ulher, 2022])

Given x, assume w.l.o.g. that $x^T x^1 = \max_{i=1:n} x^T x^i$.

$$\lim_{L\to\infty} \frac{\kappa^L(x, x^i)}{\kappa^L(x, x^1)} = 0. \tag{29}$$

and

$$\lim_{L\to\infty} f(x) = \operatorname{sgn} y^1 \qquad \textit{1-nn} \tag{30}$$

Case $A = 0$, $A' \neq 0$: Networks implement singular kernel classifier

Theorem (Theorem 1 in [Radhakrishnan, Belkin, Ulher, 2022])

$$\lim_{L\to\infty} \frac{\kappa^L(x,x')}{(A')^{2L}(L+1)} = \frac{R(\|x-x'\|)}{\|x-x'\|^{\alpha}}, \quad \text{(31)}$$

← bounded

with $\alpha = -4\frac{\log A'}{\log B'}$ *and* $R() \geq 0$, *bounded, and* $R(u) > \delta$ *around 0.*

- if $\alpha > 0$, $\frac{R(\|x-x'\|)}{\|x-x'\|^{\alpha}}$ is **singular kernel**
- Computationally not a problem: if data $x^{1:n}$ distinct, G_0 is well defined
- If $x = x^i$, set $f(x) = y^i$.

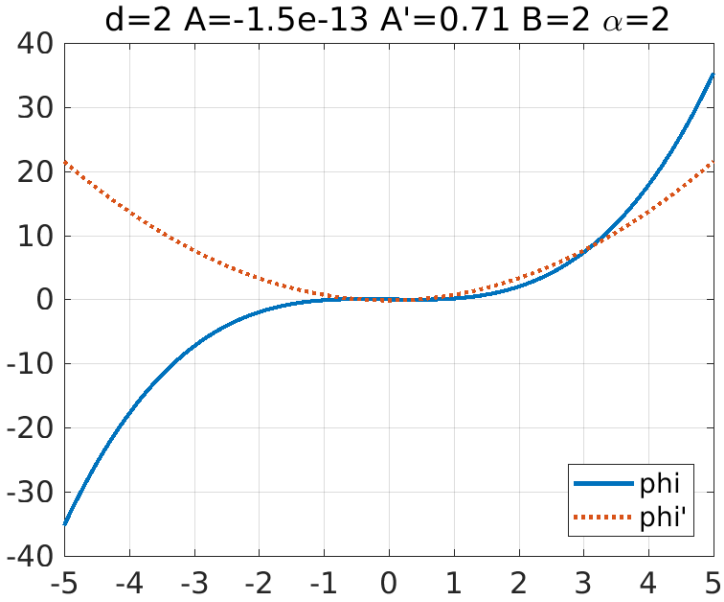
Optimality of singular kernel classifier

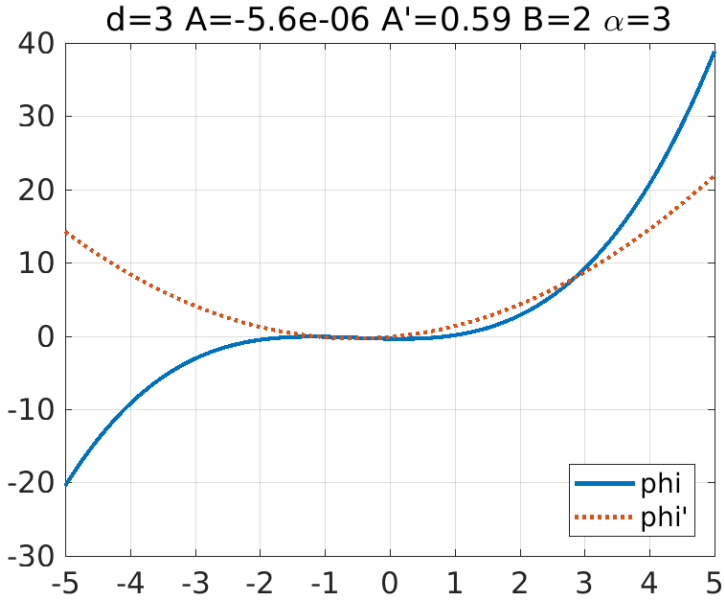
Theorem (Theorem 2 in [Radhakrishnan, Belkin, Ulher, 2022])

If $A = 0$, $A' \neq 0$ and $\alpha = -d$ then $\lim_{L\to\infty} f(x)$ is Bayes-optimal.

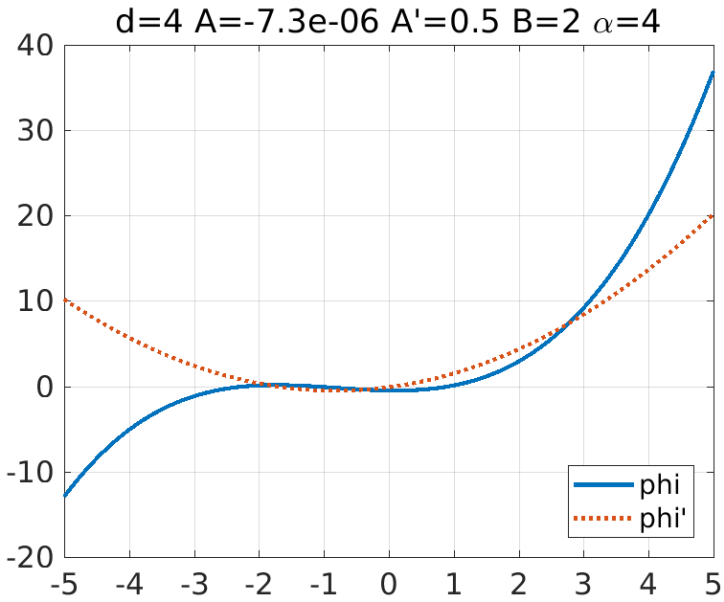
- What activations ϕ satisfy this theorem?

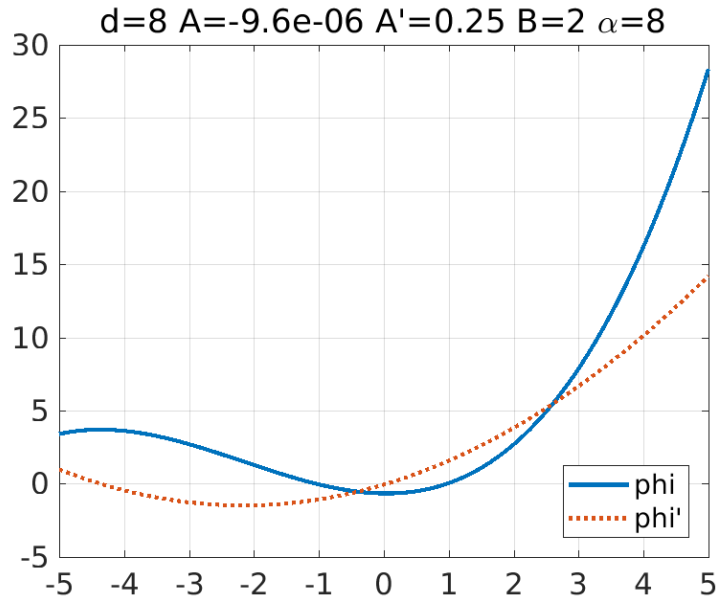
$$\phi^{\text{opt}}(z) = \frac{1}{2^{d/4}} \frac{z^3 - 3z}{\sqrt{6}} + \sqrt{1 - 2^{1-d/2}} \frac{z^2 - 1}{\sqrt{2}} + \frac{1}{2^{d/4}} z \quad \text{for } d \geq 2. \tag{32}$$

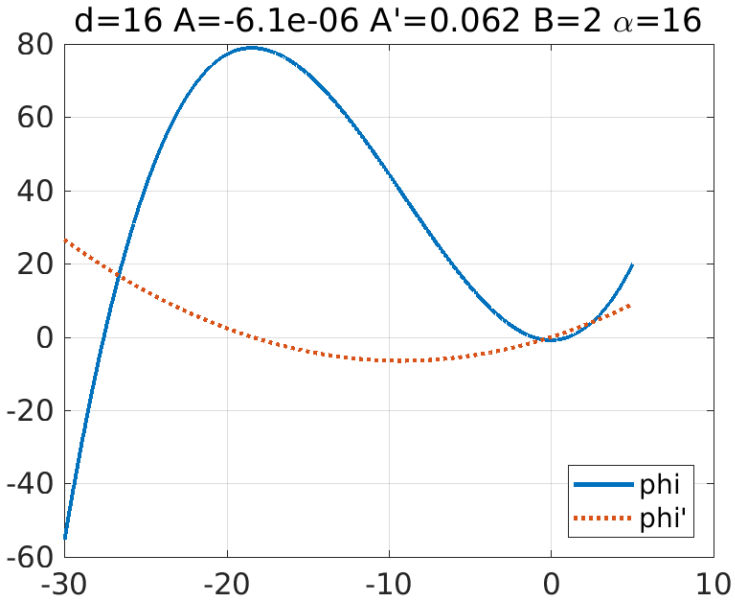


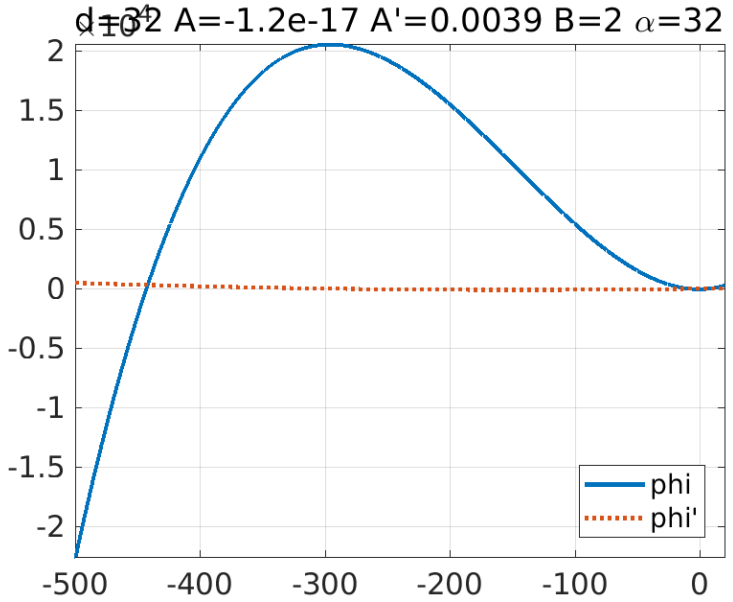


Optimal singular kernels for $d = 4, 8, 16, 32$

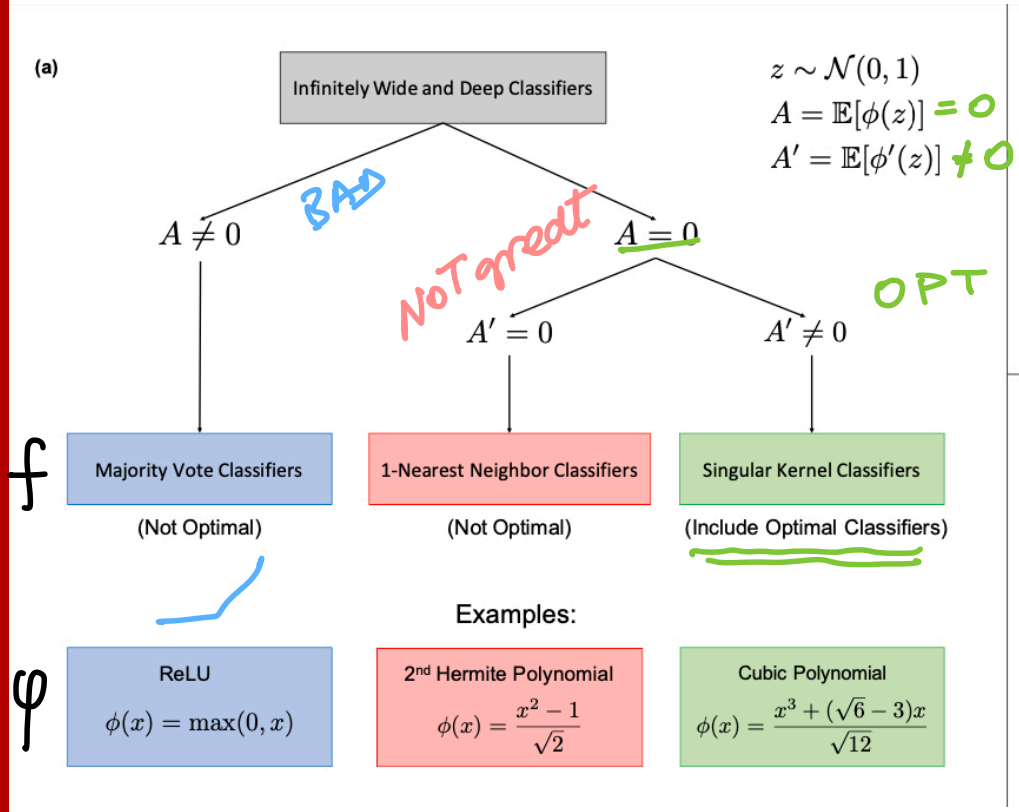


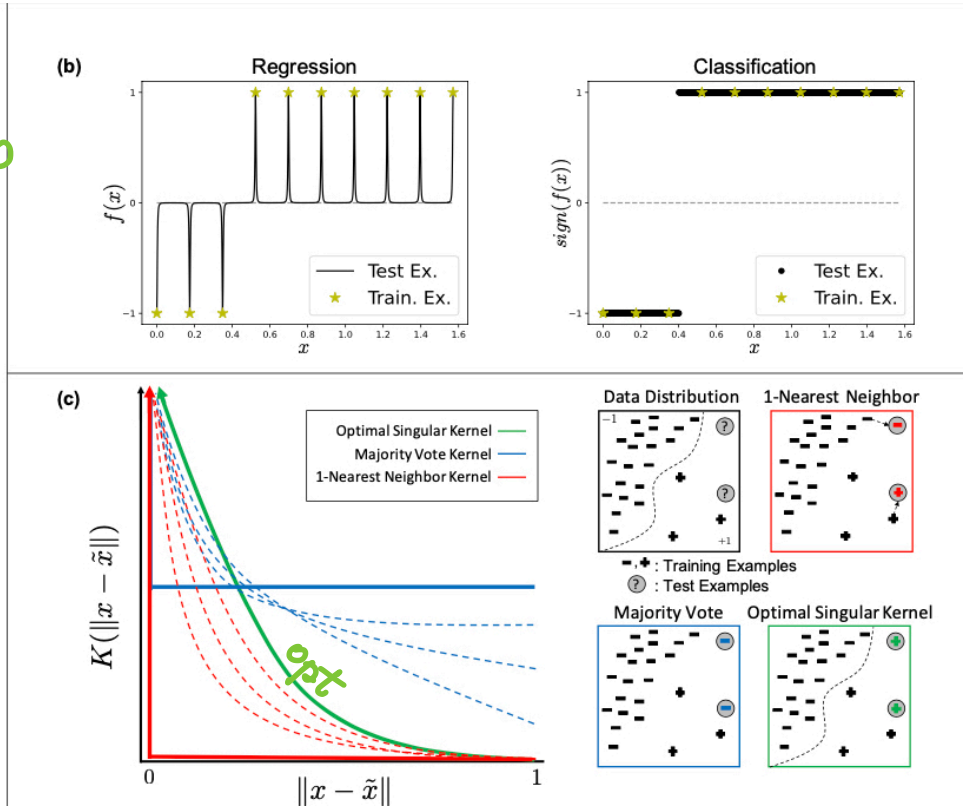






Summary





b when $A \neq 0$, $\lim_{L\to\infty} \kappa^L(x, x') = 0$ for $x \neq x'$, and $f(x) = 0$ is vanishingly small (useless for regression), but $\mathrm{sgn} f(x)$ can be optimal for classification

c Singular kernel $\alpha > d$,$\alpha < d$, majority vote kernel, and 1-nn kernel

Limits of some activation functions

ϕ^{opt}	Bayes classifier
ReLU	majority vote
sigmoid	$\frac{1}{1+e^{-z}} - \frac{1}{2}$ 1-nearest neighbor

Lecture VII: Combining predictors – Part I

Marina Meilă
mmp@stat.washington.edu

Department of Statistics
University of Washington

December, 2023

Boosting

— wrapper alg.

There is more than one way to average predictors

- Bayesian averaging
- Bagging → Random Forest (Reduce Var)
- Stacking
- Mixtures of Experts
- Forward Fitting and Backfitting

Reducing bias: Boosting ← NOT averaging

- AdaBoost
- Properties on the training set

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

There is more than one way to average predictors

Classification will be the running exaple here, but most results hold for other prediction problems as well.
Denote $\mathcal{B} = \{b\}$ a **base classifier** family
Averaging:

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

f is real-valued even if the b^k's are ± 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** $\Leftrightarrow$
  $P(b_1 \text{ wrong} \,|\, x) = P(b_1 \text{ wrong} \,|\, x,\, b_2 \text{ wrong})$
- 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.

We will examine

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

Reducing bias: Boosting

Task = classification

$\mathcal{B} = \{b()\}$ weak classifier family

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. Learning algorithm accepts **weighted** data sets. Training minimizes

$$\hat{L}_{01}^{w}(b) = \sum_{i=1}^{n} w_i L_{01}(y^i, b(x^i)) \quad \text{with} \quad \sum_{i=1}^{n} w_i = 1.$$

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

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

$\mathcal{B}$ classifies $> \frac{1}{2}$ data correctly

First

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.

TRAIN:

Test $f(x) = \sum_{k=1}^{m} \beta^k b^k(x)$

b^3

β^k ← coeff.

$M \sim n^{\gamma}$ $\gamma < 1$

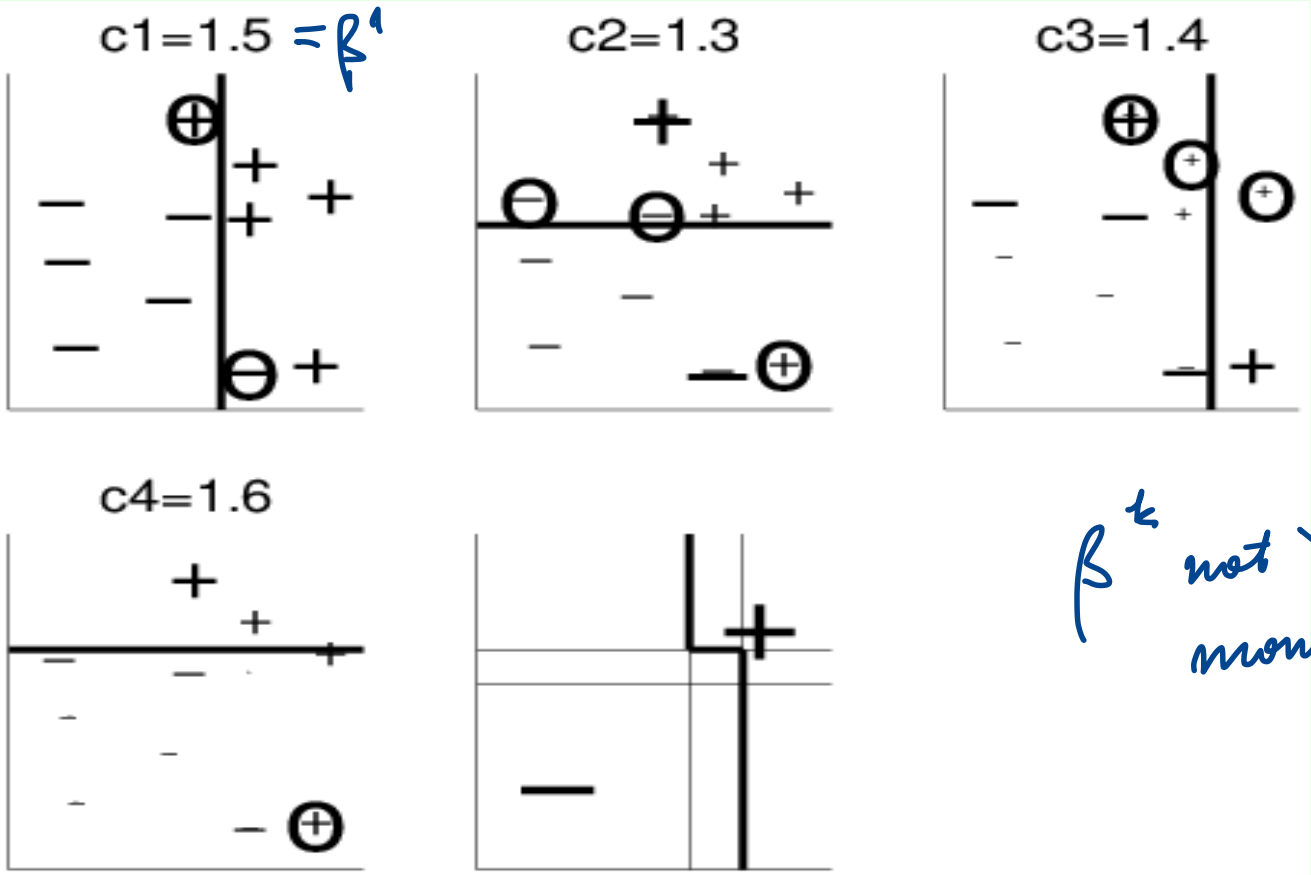
Example

Boosting with stumps

Stumps are decision trees with a single split.

(below, $c_1 \ldots c_4$ denote the coefficients $\beta_{1:4}$).

β^2 β^3



β^1

β^k not ↘ monotonically

AdaBoost Algorithm

weak classifier family $\{b_\ell\}$

ADABOOST ALGORITHM

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" $\Rightarrow$ predictor b^k — ε^k
2. compute error $\varepsilon^k = \sum_{i=1}^n w_i^k \frac{1 - y_i b^k(x_i)}{2}$ — weighted error
3. set $\beta^k = \frac{1}{2} \ln \frac{1-\varepsilon^k}{\varepsilon^k} > 0$
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 $\sum_i w_i^{k+1} = 1$

$\varepsilon^k \geq \frac{1}{2} - \delta$

$e^{-\beta^k}$ if correct, e^{β^k} if error

Normalized

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

Remarks

1. If $b(x) \in \{\pm 1\}$ then $y^i b(x^i) \in \{\pm 1\}$, and $\frac{1-y^i b(x^i)}{2} = 1$ if an error occurs and 0 otherwise. Thus, ε^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.
2. In both cases, $\varepsilon^k \in [0, \delta]$, $\delta < 0.5$ by the weak learner property of $\mathcal{B}$
3. $\beta^k > 0$ whenever $\varepsilon^k < 1/2$.
4. If $b \in \{\pm 1\}$, then step 4 can be written equivalently (up to a multiplicative constant)

$$w_i^{k+1} = \begin{cases} \frac{1}{Z^k} w_i^k & \text{if } b^k(x_i) = y_i \\ \frac{1}{Z^k} w_i^k e^{2\beta^k} & \text{if } b^k(x_i) \neq y_i \end{cases} \tag{7}$$

   This form corresponds to the 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.

An interpretation of the weights

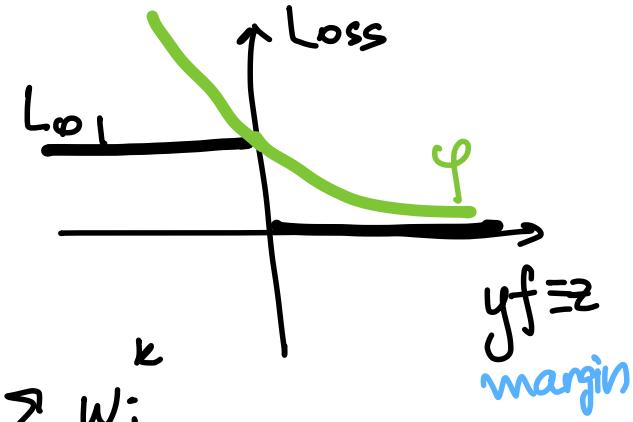
$$w_i^{k+1} = \frac{1}{n} \prod_{k' \leq k} \left(\frac{e^{-\beta^{k'} y_i b^{k'}(x_i)}}{Z_{k'}} \right) = \frac{e^{-y_i f^k(x_i)}}{n \prod_{k' \leq k} Z^k} \tag{8}$$

round $k+1$, point i; w^0; round k'; $e^{-y^i \sum_{k'=1}^{k} \beta^{k'} b^{k'}(x^i)}$

- 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.

$y^i f(x^i) =$ margin

The normalization constant is an average loss



If we sum both sides of (8) over i we obtain

$$1 = \frac{\sum_i e^{-y_i f^k(x_i)}}{n \prod_{k' \le k} Z^k}, \qquad (9)$$

or

$$\hat{L}_\varphi^k = \prod_{k' \le k} Z^k = \frac{\sum_i e^{-y_i f^k(x_i)}}{n} \equiv \hat{L}_\phi(f^k) = \frac{1}{n} \sum_i e^{-y^i f^k(x^i)} \qquad (10)$$

where

$$\phi(z) = e^{-z}. \Rightarrow L_{01}(z) = \frac{1}{n} \sum_i \varphi(y^i f(x^i)) \qquad (11)$$

and

New Loss → $$L_\phi(y, f(x)) = \phi(yf(x)) \qquad (12)$$

Hence, the r.h.s of (10) 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.

$$\hat{L}_\varphi^k \le \gamma^k \qquad \gamma < 1$$

f^k = f after step k

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

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

$$Z^k = \sum_{i=1}^{n} w_i^k e^{-\beta^k (y_i b^k(x_i))} \tag{13}$$

$$= e^{\beta^k} \underbrace{\sum_{i=err} w_i^k}_{\varepsilon^k} + e^{-\beta^k} \underbrace{\sum_{i=corr} w_i^k}_{1-\varepsilon^k} \tag{14}$$

$$= e^{\beta^k} \varepsilon^k + (1-\varepsilon^k) e^{-\beta^k} \tag{15}$$

$$= \sqrt{\frac{1-\varepsilon^k}{\varepsilon^k}} \varepsilon^k + \sqrt{\frac{\varepsilon^k}{1-\varepsilon^k}} (1-\varepsilon^k) = 2\sqrt{(1-\varepsilon^k)\varepsilon^k} \leq \gamma \tag{16}$$

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

$$\hat{L}_\phi(f^k) = \prod_{k=1}^{k} Z^{k'} \leq \gamma^k \tag{17}$$

The training set error $\hat{L}_{01}$ decreases exponentially with M

Note that $\phi(z) \geq 1_{z<0}$ for all z (see also figure **??**). Therefore

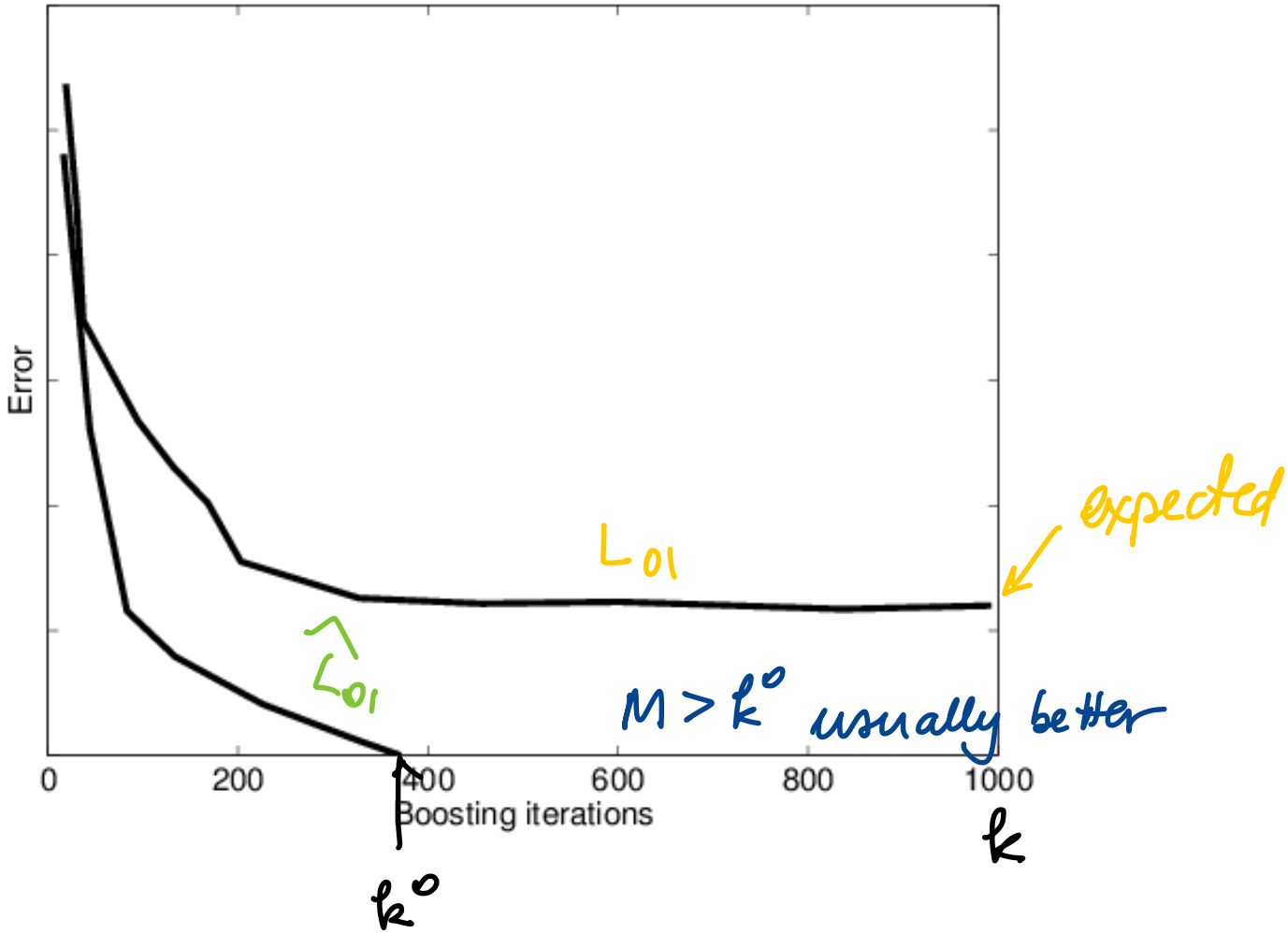
$$\hat{L}(f^k) = \frac{1}{n}\sum_{i=1}^{n} 1_{[y_i f^k(x_i)<0]} \tag{18}$$

$$\leq \frac{1}{n}\sum_{i=1}^{n} e^{-y_i f^k(x_i)} = \hat{L}_\phi(f^k) \leq \gamma^k \tag{19}$$

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!

The test set error and overfitting

- Do NOT take $M = k^0$. The number of steps M for good generalization error is often much larger than k^0 (and sometimes smaller).
- Below is a typical plot of $\hat{L}$ and L (which can be estimated from an independent sample) vs the number of boosting iterations.



Lecture VII: Combining predictors – Part II

Marina Meilă
mmp@stat.washington.edu

Department of Statistics
University of Washington

November, 2023

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
- GradientBoost

φ = surrogate Loss

Practical remarks and theoretical results

- Practicalities
- [Theoretical results]

Extensions of boosting

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

Reading HTF Ch.: 15. Random Forests, 16. Ensembles of predictors, 8. Model inference and averaging, 10. Boosting, Murphy Ch.: 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, Bach Ch.:

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 $\hat{L}_\phi(f) - \hat{L}_\phi(f + \beta b)$.

Gradient of $\hat{L}_\varphi$

$$\underline{f}^k = \begin{bmatrix} f^k(x^i) \\ \vdots \\ \end{bmatrix}_{i=1:w} \quad \text{at step } k$$

$$y = \begin{bmatrix} y^i \\ \vdots \\ \end{bmatrix}_{i=1:w}$$

$$b^k = \begin{bmatrix} b^k(x^i) \\ \\ \end{bmatrix}_{i=1:w}$$

$$\varphi(z) = e^{-z}$$

$$\varphi'(z) = -\varphi(z)$$

$$\frac{\partial}{\partial f}\varphi(yf) = -y\varphi(yf)$$

$$\hat{L}_\varphi(\underline{f}) = \frac{1}{n}\sum_{i=1}^{n}\varphi(y^i \underbrace{f(x^i)}_{f_i,\, i=1:n})$$

want $f^* \in \mathbb{R}^n$ to minimize $\hat{L}_\varphi$

$f(x) = \sum_k \beta_k b^k(x)$ restricted !!

decrease $\hat{L}_\varphi$ iteratively

$$f^{k+1} = f^k + \beta_k b^k \in \mathbb{R}^n$$

β_k: step size; b^k: direction

The direction of descent

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

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

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

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

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

- If $b(x^i) = \pm 1$ values, then $1 - y_i b(x^i) = 1_{[i \text{ error}]}$, 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

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

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.

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

β is the (unique) root of

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

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

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

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

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

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