What do they have in common?

- **A**verage over classifiers
- **B**agging, **B**oosting, and **B**ayesian **A**veraging


1 Classification

Input $x \in X$

Label $y = \pm 1$

Classifier $f : X \rightarrow \{+1, -1\}$

Data set (training set)

$$D = \{(x^1, y^1), (x^2, y^2), \ldots (x^N, y^N)\}$$

Learning algorithm $D$ estimates classifier $f \in \mathcal{F}$ based on data $D$

We will be averaging several $f \in \mathcal{F}$. Hence, we call $\mathcal{F}$ the **base classifier** family.
Examples of classifiers

Linear

Quadratic

Decision tree

\[ f(x) = \text{sign}(ax + b) \]

\[ f(x) = \text{sign}(ax^2 + bx + c) \]

Cost of errors \( C(y, f(x)) = 1 \)

Confidence and margin. Sometimes we construct real-valued “classifiers”

\[ f : X \rightarrow R \]

Then, the label of a new example is \( \hat{y} = \text{sign}f(x) \) and \( |f(x)| \) is the confidence of the classification. The margin is \( yf(x) \) which should be > 0
for correct classification.

1.1 Variance and Bias

**Variance.** For a fixed learning algorithm, if a new data set is sampled ⇒ the learned $f$ will be different. The variance measures the sensitivity of $f$ to changes in the training set.

- variance decreases with $N$
- variance increases with complexity (high for overfitted models)

**Bias.** When the classifier $f$ is too simple, it cannot fit the data well, no matter which $f \in \mathcal{F}$ we choose. Bias can be

- **deterministic (hard):** no $f \in \mathcal{F}$ near optimal generalization error (in the case the classes are separable, no $f \in \mathcal{F}$ fits the data)
- **stochastic (soft):** the prior of an $f \in \mathcal{F}$ with near optimal generalization error (i.e that fits the data if classes separable) is very small
1.2 Reasons to average over classifiers

Denote $\mathcal{F} = \{f\}$ a base classifier family

Averaging: $F(x) = \sum_{k=1}^{M} c_k f_k(x)$

$F$ is real-valued even if the $f_k$’s are $\pm 1$ valued

- can reduce variance
- can reduce bias
- can compensate for local optima (a form of bias)
- if $f_1, f_2, \ldots f_M$ make independent errors, averaging reduces error. We say that $f_1$ and $f_2$ make independent errors $\iff P(f_1 \text{ wrong} | x) = P(f_1 \text{ wrong} | x, f_2 \text{ wrong})$

Averaging is not always the same thing. Depending how we choose $\mathcal{F}, f_1, f_2, \ldots f_M$ and $c_1, c_2, \ldots c_M$, we can obtain very different effects.

2 Reducing variance: Bagging

What if we had several training sets $\mathcal{D}_1, \mathcal{D}_2, \ldots \mathcal{D}_M$?

- we could train classifiers $\{f_1, f_2, \ldots f_M\}$ (sample from $P(f)$)
- we could estimate $E_{P(f)}[f] \sim F = \frac{1}{M} \sum_{k=1}^{M} f_k$
- $F$ has always lower variance than $f$

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

$$F(x) = \text{sign} \left( \frac{1}{N} \sum_{k=1}^{M} f_k(x) \right)$$
sample $N' \leq N$ samples $\times M$ times

Resulting “bagged” classifier $F = \frac{1}{3}(f_1 + f_2 + f_3)$

It was shown theoretically and empirically that bagging reduces variance.

Good for

- base classifiers with high variance (complex)
- unstable classifiers (decision trees, decision lists, neural networks)
- noisy data
3 Reducing bias: Boosting

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

$$0 < \text{Err}(f) \leq \delta < \frac{1}{2}$$

Learning algorithm can take weighted data sets.

Idea of boosting: train a classifier $f_1$ on $\mathcal{D}$, then train a $f_2$ to correct the errors of $f_1$, then $f_3$ to correct the errors of $f_2$, etc.

**Discrete AdaBoost Algorithm**

**Assume** base classifiers $f_k$ outputs ±1 values

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

**Initialize** $F = 0$

1. $w_i^1 = \frac{1}{N}$ weight of datapoint $x_i$
   2. for $k = 1, 2, \ldots, M$
      3. learn classifier for $\mathcal{D}$ with weights $w^k \Rightarrow f_k$
      4. compute $e_k = \text{Err}(f_k) = \sum_{i=1}^{N} w_i^k \cdot 1_{f_k(x_i) \neq y_i} < \frac{1}{2}$
      5. compute coefficient of $f_k$: $c_k = \log \frac{1 - e_k}{e_k} > 0$
      6. compute new weights $w_i^{k+1} = \begin{cases} w_i^k & \text{if } f_k(x_i) = y_i \\ w_i^k e^{c_k} & \text{if } f_k(x_i) \neq y_i \end{cases}$ and normalize them to sum to 1

**Output** $F(x) = \sum_{k=1}^{M} c_k f_k(x)$

**Example:** Boosting with stumps

Below is a boosting algorithm for continuous-valued functions.

**Real AdaBoost Algorithm** (the Schapire-Singer variant)
Assume \( \mathcal{F} \) contains functions \( f \) taking values in \([-1, 1]\)

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

Initialize \( F = 0 \)

\[ w_i^1 = \frac{1}{N} \text{ weight of datapoint } x_i \]

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

1. “learn classifier for \( \mathcal{D} \) with weights \( w^k \Rightarrow f_k \)
2. compute “centered error” \( r_k = \sum_i w_i^k y_i f_k(x_i) \in [-1, 1] \)
   then \( e_k = (1 + r_k)/2, 1 - e_k = (1 - r_k)/2 \)
3. set \( c_k = \frac{1}{2} \ln \frac{1 + r_k}{1 - r_k} \)
4. compute new weights \( w_i^{k+1} = w_i^k e^{-c_k y_i f_k(x_i)} \) and normalize them to sum to 1

Output \( F(x) = \sum_{k=1}^{M} c_k f_k(x) \)

Note that

\[ w_i^k \propto \frac{1}{N} \prod_{k' < k} e^{-c_k y_i f_{k'}(x_i)} = \propto e^{-y_i F_{k-1}(x_i)} \quad (1) \]

3.1 Boosting - properties

1. The training set error decreases exponentially with \( M \)

\[ \text{Err}(F) \leq 2^M \prod_{k=1}^{M} \sqrt{e_k (1 - e_k)} \leq \left( 2 \sqrt{\delta (1 - \delta) } \right)^M \]

2. The test set error and overfitting
   Myth: “Boosting doesn’t overfit”
   Reality: Any algorithm overfits, including boosting. But in practice, overfitting occurs much later than predicted by the existing theory.
3. **Boosting increases the margin** – empirically observed, but not proved in all cases
   
   Margin \( (F) = \frac{1}{N} \sum_{i=1}^{N} y_i F(x^i) \)
   
   Why is a large margin good?

4. **Boosting as gradient descent** (See next section for the details).
   
   Let
   
   \[
   R(F) = \sum_{i=1}^{N} e^{-y_i F(x^i)}
   \]
   
   AdaBoost is gradient descent in \( R(F) \) with
   
   - \( f_k \) the direction at step \( k \)
   - \( c_k \) the step size

5. **Overfitting in noise.** Observe also the cost function(s) in the next section. When the classes overlap much (many examples impossible
to classify) boosting algorithms tend to focus too much on the hard
examples, at the expense of overall classification accuracy

4 Boosting as descent in function space

4.1 AdaBoost is steepest descent on training set

We will analyze this algorithm and show that it implements a form of steepest
descent in function space. It will be helpful to generalize the algorithm
slightly, by assuming temporarily that

- \( f(x) \in (-\infty, \infty) \)

- There is a cost of classification given by the decreasing function \( \phi(z) \),
where \( z = yF(x) \) represents the margin. In the Schapire-Singer algo-

Let the cost be

\[
R(F) = \sum_{i=1}^{N} \phi(y_i F(x_i)) \tag{2}
\]

At step \( k \), \( F_k \equiv F \) is fixed and we want to find an \( f \) which minimizes the
cost \( R(F + f) \). We have

\[
\frac{\partial R}{\partial f(x_i)} \bigg|_{f=0} = y_i \phi'(y_i F(x_i)) \tag{3}
\]

We imagine \( f \) as a vector of values \( [f(x_i)] \). Therefore the change in \( R \) along
“direction” \( f \) with step size \( \alpha \) is approximately

\[
R(F + \alpha f) - R(F) = \alpha \sum_i f(x_i) \cdot y_i \phi'(y_i F(x_i)) = \alpha \sum_i y_i f(x_i) \phi'(y_i F(x_i)) \tag{4}
\]

The best \( f \) is the one that maximizes the (positive) decrease in \( R \), i.e the
minimizer of

\[
\sum_i y_i f(x_i) [-\phi'(y_i F(x_i))] \tag{5}
\]
Recall that $\phi$ is a decreasing function, so $-\phi'$ is positive. If we set $\phi(z) = e^{-z}$ and denote $w_i = e^{-y_iF(x_i)}$ then (5) becomes

$$\min_{f \in F} \sum_i y_i f(x_i) e^{-y_i F(x_i)} = \min_{f \in F} \sum_i w_i y_i f(x_i)$$  \hfill (6)

Finding the direction $f$ is equivalent with step 1 of the algorithm, training a weak classifier on the weighted data. The resulting $f$ can be seen as the best approximate of the gradient of $R$ in $F$.

Now let us do line minimization: find the optimal step size $\alpha$ in direction $f$. For this we take the derivative of $R(F + \alpha f)$ w.r.t $\alpha$ and set it to 0.

$$\frac{dR(F + \alpha f)}{d\alpha} = \sum_i y_i f(x_i) \phi'(y_i F(x_i)) = -\sum_i y_i f(x_i) e^{-y_i F(x_i) - \alpha y_i f(x_i)}$$  \hfill (7)

Set the above to 0 and finding the (unique) root of

$$\sum_i w_i y_i f(x_i) e^{-\alpha y_i f(x_i)} = 0$$  \hfill (8)

For general functions $f$, this root can be found easily by numerical methods.

To get the Real AdaBoost updates we assume that $f(x) \in [-1, 1]$. Then $y_i f(x_i)$, $\sum_i w_i y_i f(x_i) \in [-1, 1]$. We also make the approximation

$$e^{-\alpha z} \leq \frac{1 + z}{2} e^{-\alpha} + \frac{1 - z}{2} e^\alpha$$  \hfill (9)

whenever $z \in [-1, 1]$. This follows from the convexity of $e^{-\alpha t}$ in the interval $[-1, 1]$ at point $t = (1 + z)/2$. By optimizing this upper bound w.r.t $\alpha$ we get

$$r = \sum_i w_i y_i f(x_i) \quad \alpha = \frac{1}{2} \ln \frac{1 + r}{1 - r}$$  \hfill (10)

We can also assume $f \in \{\pm 1\}$, the assumption of Discrete AdaBoost. In this case $y_i f(x_i) = \pm 1$ and we obtain

$$\frac{dR(F + \alpha f)}{d\alpha} = \sum_{i, \text{corr}} w_i e^{-\alpha} - \sum_{i, \text{err}} w_i e^\alpha = 0$$  \hfill (11)

$$0 = (1 - \sum_{i, \text{err}} w_i) - (\sum_{i, \text{err}} w_i) e^{2\alpha}$$  \hfill (12)

$$\alpha = \frac{1}{2} \ln \frac{1 - e_k}{e_k}$$  \hfill (13)
This gives us the $\alpha$ coefficient of the Discrete AdaBoost algorithm (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 $w_i$.

\begin{align*}
F & \leftarrow F + \alpha f \\
w_i & \leftarrow w_i e^{-\alpha y_i f(x_i)}
\end{align*} 

Hence, the boosting algorithm given at the beginning of this section and many other variants can be seen as minimizing a cost $R(F)$ by steepest descent in the function space $\mathcal{F}$.

4.2 A statistical view of boosting

It has been shown (0) (FHT) that boosting can also be seen as noisy gradient descent in function space when we replace the finite training set with the true data distribution. The cost function and gradient can be given a probabilistic interpretation. This point of view is useful in two ways:

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

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

**Assume**  $\mathcal{F}$ contains real-valued functions

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

**Initialize**  $F = 0$

\[ w_i^1 = \frac{1}{N} \text{ weight of datapoint } x_i \]

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

“learn classifier for $\mathcal{D}$ with weights $w^k \Rightarrow f_k$”

compute new weights

\[ w_i^{k+1} = w_i^k e^{-y_i f_k(x_i)} \]

and normalize them to sum to 1

**Output**  $F(x) = \sum_{k=1}^{M} f_k(x)$

In the above, we assume that “learning a classifier” means:
(theoretically, see Proposition 2) find the best possible approximation in $\mathcal{F}$ to $\frac{1}{2} \log \frac{p_w(y=1|x)}{p_w(y=-1|x)}$

(practically, also based on Proposition 2) find the best possible minimizer in $f_k \in \mathcal{F}$ to $\sum_{i=1}^{N} w_i^k e^{-y_i f(x_i)}$

Cost function for boosting: $R(F) = E[e^{-yF(x)}]$. The notation $E[]$ represents the expectation w.r.t the joint $x, y$ distribution. This is used in the proofs, while in practice it is replaced by the average over the data set.

**Proposition 1** The cost $R(F)$ is minimized by

$$F(x) = \frac{1}{2} \ln \frac{P(y=1|x)}{P(y=-1|x)}$$

Hence,

$$P(y=1|x) = \frac{e^{F(x)}}{e^{F(x)} + e^{-F(x)}} \quad P(y=-1|x) = \frac{e^{-F(x)}}{e^{F(x)} + e^{-F(x)}} = 1 - P(y=1|x)$$

(16)

**Proof** Let $x$ be fixed. Hence

$$E[e^{-yF(x)}|x] = P(y=1|x)e^{-F(x)} + P(y=-1|x)e^{F(x)}$$

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 algorithm fits an additive logistic regression model $F$ by iterative optimization of $R(F)$.

**Proof** Suppose we have a current estimate $F(x)$ and seek to improve it by minimizing $R(F + f)$ over $f$. In the proof we assume that $f$ is an arbitrary function, while in practice $f$ will be chosen to best approximate the ideal $f$ within the class $\mathcal{F}$. 
Denote by $p_x = P[y = 1|x]$ (the true value) and by $\hat{p}_x$ the “estimate”

$$\hat{p}_x = \frac{e^{F(x)}}{e^{F(x)} + e^{-F(x)}} \quad (17)$$

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

$$R(F + f) = E[e^{-yF(x) - yf(x)}]$$
$$= E[e^{-f(x)}e^{-F(x)}p_x + (1 - p_x)e^{f(x)}e^{F(x)}]$$

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

$$f(x) = \frac{1}{2} \ln \frac{p_x e^{-F(x)}}{(1 - p_x) e^{F(x)}} = \frac{1}{2} \left[ \ln \frac{p_x}{1 - p_x} - \ln \frac{\hat{p}_x}{1 - \hat{p}_x} \right] \quad (18)$$

Note that if one could exactly obtain the $f$ prescribed by (18) the iteration would not be necessary.

More interesting than the above, exact form of $f$ is rewriting the optimization problem that leads to it.

Denote $w(x, y) = e^{-yF(x)}$. Then, $f$ is the solution of

$$f = \arg\min_f E_{P_{XY}w(X,Y)}[e^{-Yf}] \quad (19)$$

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 $\mathcal{F}$ in order to obtain a proper distribution.) Finally, note that the new $F$ is $F + f$ and the new weights are $w(x,y)e^{-yF(x)}$ which finishes the proof.

Hence, the Real AdaBoost algorithm can be seen as a form of “noisy gradient” algorithm at the distribution level. Note the absence of the coefficient $\alpha$ in the FHT formulation of Real AdaBoost. That’s because in the Schapire and Singer version the minimization was first done by finding a direction, then optimizing for step size, while in the FHT version the minimization in equation (19) is over both direction and scale of $f$.

Why the $e^{-yF}$ cost?

- the “true” classification cost is $1_{yF > 0}$ (called the 0–1 cost) – nonconvex, nonsmooth
The following algorithm uses the log-likelihood cost of classification (and is very competitive in practice)

Log likelihood cost of classification

\[ R_L(F) = E_X[P(y = 1|x) \ln \hat{P}(y = 1|x) + P(y = -1|x) \ln \hat{P}(y = -1|x)] \] (20)

where \( \hat{P} \) is the class probability predicted by the model \( F \), as given in equation (16). We shall find the following notation useful:

\[ y^* = (y + 1)/2 \in \{0, 1\} \text{ for } y \in \{\pm 1\} \] (21)
LogitBoost Algorithm

**Assume** \( \mathcal{F} \) 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 \( f_k(x) \) to \( z_i \) by weighted least squares, i.e. minimize \( \sum_i w_i (f(x_i) - z_i)^2 \); this is the “weak learning” step

update \( F(x) \leftarrow F(x) + \frac{1}{2} f_k(x) \) and \( \hat{P}_i \leftarrow \frac{e^{F(x_i)}}{e^{F(x_i)} + e^{-F(x_i)}} \)

**Output** sign \( F(x) \)

**Proposition 3** The LogitBoost algorithm uses (approximate) Newton-Raphson steps to maximize the log-likelihood of the data \( R_L(F) \).

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

\[
R_L(F) = E[y^* \ln \frac{e^F}{e^F + e^{-F}} + (1 - y^*) \ln \frac{e^{-F}}{e^F + e^{-F}}] \tag{22}
\]

\[
= E[(2y^* - 1)F - \ln(e^F + e^{-F})] \tag{23}
\]

\[
= E[(2y^* - 1)F + F - \ln(e^{2F} + e^0)] \tag{24}
\]

\[
= E[2y^*F - \ln(1 + e^{2F})] \tag{25}
\]
Conditioning on $x$, we compute the first and second derivative of $R_L(F + f)$ w.r.t $f(x)$ in $f(x) = 0$.

$$s(x) = \frac{\partial E[2y^*(F(x) + f(x)) - \ln(1 + e^{2F(x)} + 2f(x))]}{\partial f(x)} \bigg|_{f(x)=0}$$ (26)

$$= E[2y^* - \frac{2e^{2F(x)} + 2f(x)}{1 + e^{2F(x)} + 2f(x)}] \bigg|_{f(x)=0}$$ (27)

$$= 2y^* - E \left[ \frac{2e^{2F(x)}}{1 + e^{2F(x)}} \right]$$ (28)

$$= 2(y^* - E[\hat{P}(x)])$$ (29)

$$H(x) = \frac{\partial^2 E[2y^*(F(x) + f(x)) - \ln(1 + e^{2F(x)} + 2f(x))]}{\partial f(x)^2} \bigg|_{f(x)=0}$$ (30)

$$= \frac{\partial \left\{ 2y^* - E \left[ \frac{2e^{2F(x)}}{1 + e^{2F(x)}} \right] \right\}}{\partial f(x)} \bigg|_{f(x)=0}$$ (31)

$$= 2E \left[ \frac{-2e^{2F(x)} + 2f(x)}{(1 + e^{2F(x)} + 2f(x))^2} \right] \bigg|_{f(x)=0}$$ (32)

$$= -4E \left[ \frac{-2e^{2F(x)}}{(1 + e^{2F(x)})^2} \right]$$ (33)

$$= -4E \left[ \hat{P}(x)(1 - \hat{P}(x)) \right]$$ (34)

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

$$F(x) \leftarrow F(x) - H(x)^{-1}s(x)$$ (35)

$$= F(x) + \frac{1}{2} E[y^* - \hat{P}(x)|x]$$ (36)

$$= F(x) + \frac{1}{2} E[y^* - \hat{P}(x)|x] \frac{\hat{P}(x)(1 - \hat{P}(x))}{E[\hat{P}(x)(1 - \hat{P}(x))]|x]$$ (37)

$$= F(x) + \frac{1}{2} E_w[y^* - \hat{P}(x)|x]$$ (38)
with \( w(x) = \frac{\hat{p}(x)(1 - \hat{p}(x))}{E[p(x)(1 - p(x))]} \). The update above is equivalent to

\[
\min_f E_w \left[ \left( F(x) + \frac{1}{2} \frac{y^* - \hat{p}(x)}{P(x)(1 - P(x))} - (F(x) + f(x)) \right)^2 \right]
\]

(39)

In practice we have the condition \( f \in \mathcal{F} \) and the minimization is done over \( \mathcal{F} \) and with the sample average replacing the expectation \( E[\cdot] \).

Numerically

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

\[
z_i = \begin{cases} 
\min(1/\hat{P}_i, z_{\text{max}}) & y = 1 \\
\min(1/(1 - \hat{P}_i), z_{\text{max}}) & y = -1
\end{cases}
\]

(40)

<table>
<thead>
<tr>
<th>Cost ( \phi )</th>
<th>( e^{-yF} )</th>
<th>( y \ln p(F) + (1 - y) \ln(1 - p(F)) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{steepest} + line search</td>
<td>\text{Discrete}</td>
<td>\text{Real}</td>
</tr>
<tr>
<td>\text{Newton}</td>
<td>\text{Gentle}</td>
<td>\text{Logit}</td>
</tr>
</tbody>
</table>

5 Generalization: AnyBoost and any cost \( \phi \)

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

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

\text{AnyBoost Algorithm}

\begin{align*}
\text{Initialize} & \quad F_0 = f_0 \\
\text{for} & \quad k = 1, \ldots, M \\
& \quad \text{learn } f_k \\
& \quad \text{find } c_k \text{ by line search} \\
& \quad F_k = F_{k-1} + c_k f_k \\
\end{align*}

Properties
Can we analyze which cost functions \( \phi \) are “better”? Can we offer some guarantees in terms of generalization bounds? The answers are in (0) Bartlett, Jordan & McAuliffe, ”Convexity, classification and risk bounds”, 2005 (BJM).

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

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

\[
C_p(z) = p\phi(z) + (1-p)\phi(-z)
\]

and the optimal cost is

\[
H(p) = \inf_z C_p(z) \text{ attained for } F^* = \frac{1}{2} \ln \frac{p}{1-p}
\]

Let \( H^- \) denote the smallest cost for a misclassification

\[
H^-(p) = \inf_{\sgn z = -\sgn(2p-1)} C_p(z)
\]

Intuitively, we are minimizing \( \phi \) instead of the “true” misclassification cost, and we want to measure how much we can be off when doing this. The following results say that we can bound the “true” risk \( R(F) \) in terms of the \( \phi \)-risk \( R_\phi \).

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

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

\[
\psi(R(F) - R^*) \leq R_\phi(F) - R^*_\phi
\]

where \( R^*_\phi, R^* \) represent respectively the optimal \( \phi \)-risk and optimal risk on the given data distribution and \( \psi \) is

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

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

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

### 5.1 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 (0) “The Multiplicative updates method” by Arora, Hazan and Kale.

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

**Weighted Majority Algorithm**

**Initialize** $w_k^0 \leftarrow 1$

for $t = 1, 2, \ldots$

1. $w_k^t \leftarrow w_k^{t-1} (1 - \epsilon)$ if expert $k$ makes a mistake at time $t$
2. predict the outcome that agrees with the weighted majority of the experts

It can be shown that the number of mistakes $m^t$ of $F$ up to time $t$ is bounded by

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

(46)

where $m^t_j$ is the number of mistakes of any expert $j$. Thus, asymptotically, the number of mistakes of the algorithm is about twice those of the best
expert.

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

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

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

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

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

**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 $r_i^t = A_i x^t - b_i$
3. Update weights by
   $$w_i^{t+1} \leftarrow w_i^t (1 - \epsilon \cdot \text{sign} r_i^t |r_i^t|)$$
   with $\epsilon = \frac{\delta}{4\rho}$ and renormalize the weights.

**Output** $x = \sum_t x^t / T$

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

### 5.2 Boosting for multiclass and ranking

TBW
6 Non-commitment: Bayesian averaging

This section was not updated since 2000 when it was first written.

Assumed \( f \in \mathcal{F} \); learning means changing the probability distribution of \( f \) after seeing the data.

Before seeing data \( P_0(f) \) prior distribution over \( f \)

After seeing \( \mathcal{D} \) \( P(f|\mathcal{D}) \) posterior distribution over \( f \)

Bayes formula \( P(f|\mathcal{D}) = \frac{P_0(f)P(\mathcal{D}|f)}{\sum_{f'\in\mathcal{F}} P_0(f')P(\mathcal{D}|f')} \)

Classification of a new instance by Bayesian averaging:

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

or

\[
P(y|x, \mathcal{D}) = \sum_{f \in \mathcal{F}} P(f|\mathcal{D})1_{f(x)=y}
\]

Note that classifiers are weighted by their posterior probability.

**Intuition:** The likelihood becomes more concentrated when \( N \) increases.

**Bayesian averaging and model complexity** Bayesian averaging mitigates overfitting.

**Bayesian averaging in practice.** Summation over \( f \in \mathcal{F} \) usually intractable. Practically, one samples a few \( f \)'s: \( f_1, f_2, \ldots f_M \)

\[
\hat{P}(f_k|\mathcal{D}) = \frac{P_0(f_k)P(\mathcal{D}|f_k)}{\sum_{k'=1}^{M} P_0(f_{k'})P(\mathcal{D}|f_{k'})}
\]

What is a good choice for \( f_k \)? (local) maxima of \( P(f|\mathcal{D}) \) or \( P(\mathcal{D}|f) = \prod_{i=1}^{N} P(y_i|f_k) \)

Unlike Bagging or Boosting, (approximate) Bayesian averaging is not a completely specified recipe. It can behave like either one of them, depending on the choices for \( \mathcal{F} \) and the specific \( f_1, f_2, \ldots f_M \).

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

Even if we choose the \( P_0(f) \) for computational convenience only, averaging can have beneficial effects

- variance reduction
- bias reduction
- if the averaged \( f \)'s are diverse enough, it reduces the overall classification error
- 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
  \[
  P_0(f) \propto 2^{-\#\text{bits to encode } f}
  \]
  
  \# bits ~
  - \# splits in decision tree
  - degree of polynomial
  - “effective” number of parameters

### 7 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 \( \mathcal{F} \) 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


