Ensemble Methods: Boosting

M. R. Hasan
Readings

• Bishop: 14.3
• Murphy: 16.4
• Geron: chapter 7
What We Will Cover

• Boosting
• Forward Stagewise Additive Modeling
• AdaBoost
• AdaBoost: Choice of Loss Function
• Effect of learning (shrinkage) hyperparameter
• Gradient Boosting
• Why does Boosting work so well?
• L1-AdaBoost
• Boosting: Bayesian View
• Boosting: Applications
WORK SMARTER NOT HARDER
Motivating Problem

• Consider the following classification problem.
• The **decision boundary is complex**.
• How do we **learn** this complex decision boundary?
Motivating Problem

- We may use **weak learners** to learn decision boundaries on **different parts of the training data**.
Motivating Problem

- Then, combine the decision boundaries of multiple weak learners to create a strong learner.

This is the main idea behind **Boosting**!
Motivating Problem

Boosting: *multiple weak learners* are combined to create a *strong learner*.
Boosting

• Boosting (originally called *hypothesis boosting*) refers to any Ensemble method that can combine several weak learners into a strong learner.
Boosting

- The general idea of most boosting methods is to train learners sequentially,

  each trying to **correct** its predecessor.
Boosting

• To correct previous learner’s errors, the **misclassified points are given larger weight**.
• And the correctly classified points are given **lower weight**.

Learner 1

Learner 2 & 3:
*Increase weight*: misclassified points of previous step
*Decrease weight*: correctly classified points
Boosting

- Once all the classifiers have been trained, their predictions are then combined through a weighted majority voting scheme.
Boosting

- Training cases
- Correctly classified
- Training case has large weight in this round
- This DT has a strong vote.
Boosting

• There are many boosting methods available.
• By far the most popular are:

  • **AdaBoost** (short for Adaptive Boosting).
  • **Gradient Boosting**.
Boosting

- The main idea in boosting is to make weak learners strong by reducing their biases.

Each learner is a decision tree stump (depth 1), hence weak.

They have high bias.
Boosting

• In boosting, the predictions of the weak learners’ are \textbf{weighted and then combined} to create a strong learner.

Strong learner has \textbf{lower bias}
Boosting: Historical Development

- Boosting was originally derived in the computational learning theory literature (Schapire 1990; Freund and Schapire 1996).
Boosting: Historical Development

• It was proved that one could boost the performance (on the training set) of any weak learner arbitrarily high!
Boosting vs Ensemble Methods (Bagging)

- The principal difference between boosting and the committee/ensemble methods is that the base classifiers are trained in sequence.
Boosting vs Ensemble Methods (Bagging)

• Another difference is that each base classifier is trained using a **weighted form of the data set**.

• The weighting coefficient associated with each data point depends on the **performance of the previous classifiers**.
Boosting

• In particular, **points that are misclassified** by one of the base classifiers are **given greater weight** when used to train the next classifier in the sequence.

• Once all the classifiers have been trained, their predictions are then combined through a **weighted majority voting scheme**.
Boosting

• The weak learners in boosting can be any classification or regression algorithm.
• It is common to use a CART model.

Boosting is the “best off-the-shelf classifier in the world” (Breiman, 1998).

The second best method is random forests.
Boosting

- Performance of AdaBoost using a decision stump as a weak learner.
- Training (solid blue) and test (dotted red) error vs. number of iterations.
Boosting

- The training set error **rapidly goes to near zero**.
- What is more surprising is that the **test set error continues to decline** even after the training set error has reached zero (although the test set error will eventually go up).

Thus, boosting is very resistant to overfitting.
The goal of boosting is to solve the following **optimization problem**.

Say that the **training data is** $x_i$ and the output label is $y_i$.

Then, we try to **minimize the loss/error function** $L$.

$$\min_f \sum_{i=1}^{N} L(y_i, f(x_i))$$
Boosting: General Framework

- Minimize the loss/error function $L$.

$$\min_f \sum_{i=1}^{N} L(y_i, f(\tilde{x}_i))$$

Here $f(x)$ is a **classifier** defined in terms of a **linear combination** of $M$ base classifiers.

$$f(\tilde{x}) = w_0 + \sum_{m=1}^{M} w_m \phi_m(\tilde{x})$$

$$f(\tilde{x}) = w_0 + \sum_{m=1}^{M} y_m(\tilde{x})$$
Boosting: General Framework

• Common choice for **loss function**:

<table>
<thead>
<tr>
<th>Name</th>
<th>Loss</th>
<th>Derivative</th>
<th>$f^*$</th>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Squared error</td>
<td>$\frac{1}{2}(y_i - f(x_i))^2$</td>
<td>$y_i - f(x_i)$</td>
<td>$\mathbb{E}[y</td>
<td>x_i]$</td>
</tr>
<tr>
<td>Absolute error</td>
<td>$</td>
<td>y_i - f(x_i)</td>
<td>$</td>
<td>$\text{sgn}(y_i - f(x_i))$</td>
</tr>
<tr>
<td>Exponential loss</td>
<td>$\exp(-\tilde{y}_i f(x_i))$</td>
<td>$-\tilde{y}_i \exp(-\tilde{y}_i f(x_i))$</td>
<td>$\frac{1}{2} \log \frac{\pi_i}{1-\pi_i}$</td>
<td>AdaBoost</td>
</tr>
<tr>
<td>Logloss</td>
<td>$\log(1 + e^{-\tilde{y}_i f_i})$</td>
<td>$y_i - \pi_i$</td>
<td>$\frac{1}{2} \log \frac{\pi_i}{1-\pi_i}$</td>
<td>LogitBoost</td>
</tr>
</tbody>
</table>

$$
\min_f \sum_{i=1}^{N} L(y_i, f(x_i))
$$
Boosting

\[ \min_f \sum_{i=1}^{N} L(y_i, f(\hat{x}_i)) \]

• Note that finding the optimal \( f \) is hard for m learners.

• Hence, we shall tackle it **sequentially**.

• We start with the **base learner** and **iterate through the remaining learners** by improving the performance.

• We initialize by defining:

\[ f_0(\hat{x}) = \arg\min_{\gamma} \sum_{i=1}^{N} L(y_i, f(\hat{x}_i, \gamma)) \]

Here \( \gamma \) is the **model parameter** that minimizes \( L \).
Boosting

- At iteration $m$, we compute:

$$ (\alpha_m, \gamma_m) = \arg\min_{\alpha, \gamma} \sum_{i=1}^{N} L(y_i, f_{m-1}(\hat{x}_i) + \alpha \phi(\hat{x}_i, \gamma)) $$

- $\alpha_m$ is the learner coefficient that represents the weight (confidence) of each learner

- $\gamma_m$ is the learner parameter
Boosting

• We keep the model parameter and coefficient of the previous learners (m – 1 stage) fixed.

• And only learn parameters ($\alpha_m$ & $\gamma_m$) for the m’th learner.

• Then, we set:

$$f_m(\tilde{x}_i) = f_{m-1}(\tilde{x}_i) + \alpha_m \phi(\tilde{x}_i, \gamma_m)$$

$$(\alpha_m, \gamma_m) = \arg\min_{\alpha, \gamma} \sum_{i=1}^{N} L(y_i, f_{m-1}(\tilde{x}_i) + \alpha \phi(\tilde{x}_i, \gamma))$$
Boosting

- The key point is that we **do not go back and adjust earlier parameters**.
- This is why the method is called **forward stagewise additive modeling**.

\[
(\alpha_m, \gamma_m) = \arg\min_{\alpha, \gamma} \sum_{i=1}^{N} L(y_i, f_{m-1}(\tilde{x}_i) + \alpha \phi(\tilde{x}_i, \gamma))
\]
Boosting

• We continue this for a **fixed number of iterations** $M$.
• In fact, $M$ is the **main tuning parameter** of the method.
• Often we pick it by monitoring the performance on a **separate validation set**, and then stopping once performance starts to decrease.

• This is called **early stopping**.

\[
(\alpha_m, \gamma_m) = \arg\min_{\alpha, \gamma} \sum_{i=1}^{N} L(y_i, f_{m-1}(\tilde{x}_i) + \alpha \phi(\tilde{x}_i, \gamma))
\]
In practice, better (test set) performance can be obtained by performing “partial updates” of the following form.

\[ f_m(\tilde{x}_i) = f_{m-1}(\tilde{x}_i) + \alpha_m \phi(\tilde{x}_i, \gamma_m) \]

Here \( 0 < \eta \leq 1 \) is a step-size parameter, known as the shrinkage parameter.

It is common to use a small value such as \( \eta = 0.1 \).
Boosting

• The shrinkage hyperparameter $\eta$ provides a *regularization technique*.

• It *scales the contribution of each tree*.

• If we set it to a low value (such as 0.1), we will need **more trees in the ensemble** to fit the training set, but the predictions will usually generalize better.

\[
f_m(\vec{x}_i) = f_{m-1}(\vec{x}_i) + \eta \alpha_m \phi(\vec{x}_i, \gamma_m)
\]
Boosting: Solving the Optimization Problem

- Now we will **solve the minimization problem**.
- This will **depend on the form of loss function**.
- However, it is **independent of the form of weak learner**.

\[
(\alpha_m, \gamma_m) = \arg\min_{\alpha, \gamma} \sum_{i=1}^{N} L(y_i, f_{m-1}(\tilde{x}_i) + \alpha \phi(\tilde{x}_i, \gamma))
\]

<table>
<thead>
<tr>
<th>Name</th>
<th>Loss</th>
<th>Derivative</th>
<th>(f^*)</th>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Squared error</td>
<td>(\frac{1}{2}(y_i - f(x_i))^2)</td>
<td>(y_i - f(x_i))</td>
<td>(\mathbb{E}[y</td>
<td>x_i])</td>
</tr>
<tr>
<td>Absolute error</td>
<td>(</td>
<td>y_i - f(x_i)</td>
<td>)</td>
<td>(\text{sgn}(y_i - f(x_i)))</td>
</tr>
<tr>
<td>Exponential loss</td>
<td>(\exp(-\tilde{y}_i f(x_i)))</td>
<td>(-\tilde{y}_i \exp(-\tilde{y}_i f(x_i)))</td>
<td>(\frac{1}{2} \log \frac{\pi_i}{1-\pi_i})</td>
<td>AdaBoost</td>
</tr>
<tr>
<td>Logloss</td>
<td>(\log(1 + e^{-\tilde{y}_i f_i}))</td>
<td>(y_i - \pi_i)</td>
<td>(\frac{1}{2} \log \frac{\pi_i}{1-\pi_i})</td>
<td>LogitBoost</td>
</tr>
</tbody>
</table>
Boosting: Solving the Optimization Problem

- We will discuss the AdaBoost algorithm to solve the minimization problem.

\[
(\alpha_m, \gamma_m) = \arg\min_{\alpha, \gamma} \sum_{i=1}^{N} L(y_i, f_{m-1}(\hat{x}_i) + \alpha \phi(\hat{x}_i, \gamma))
\]

<table>
<thead>
<tr>
<th>Name</th>
<th>Loss</th>
<th>Derivative</th>
<th>(f^*)</th>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Squared error</td>
<td>(\frac{1}{2}(y_i - f(x_i))^2)</td>
<td>(y_i - f(x_i))</td>
<td>(\mathbb{E}[y</td>
<td>x_i])</td>
</tr>
<tr>
<td>Absolute error</td>
<td>(</td>
<td>y_i - f(x_i)</td>
<td>)</td>
<td>(\text{sgn}(y_i - f(x_i)))</td>
</tr>
<tr>
<td>Exponential loss</td>
<td>(\exp(-\tilde{y}_i f(x_i)))</td>
<td>(-\tilde{y}_i \exp(-\tilde{y}_i f(x_i)))</td>
<td>(\frac{1}{2} \log \frac{\pi_i}{1-\pi_i})</td>
<td>AdaBoost</td>
</tr>
<tr>
<td>Logloss</td>
<td>(\log(1 + e^{-\tilde{y}_i f(x_i)}))</td>
<td>(y_i - \pi_i)</td>
<td>(\frac{1}{2} \log \frac{\pi_i}{1-\pi_i})</td>
<td>LogitBoost</td>
</tr>
</tbody>
</table>
AdaBoost

- One way for a new predictor to correct its predecessor is to pay a **bit more attention to the training instances** that the predecessor **underfitted**.
- This results in **new predictors focusing** more and more on the **hard cases**.
- This is the technique used by AdaBoost.
AdaBoost

• For example, to build an AdaBoost classifier, a first base classifier (such as a Decision Tree) is trained and used to make predictions on the training set.
AdaBoost

• The relative **weight of misclassified training instances** is then increased.

• A second classifier is **trained using the updated weights** and again it makes predictions on the training set, weights are updated, and so on.
AdaBoost: Algorithm

- Consider a two-class classification problem.
- The training data comprises input vectors $x_1, \ldots, x_N$.
- The corresponding binary target variables: $t_1, \ldots, t_N$.
- Here $t_n \in \{-1, 1\}$.
- Given a vector of predictor variables $x$, a classifier $f(x)$ produces a prediction taking one of the two values $\{-1, 1\}$.
AdaBoost: Algorithm

• The purpose of boosting is to **sequentially apply the weak classification algorithm** to repeatedly modified versions of the data.
• Hence, it produces a **sequence of weak classifiers** $f_m(x)$, $m = 1, 2, \ldots, M$.
• The predictions from all of them are then **combined through a weighted majority vote** to produce the final prediction:

$$Y_M(x) = \text{sign} \left( \sum_{m=1}^{M} \alpha_m y_m(x) \right)$$

Here $\alpha_1, \alpha_2, \ldots, \alpha_M$ are **computed by the boosting algorithm**, and weight the contribution of each respective $f_m(x)$

Their effect is to **give higher influence to the more accurate classifiers** in the sequence.
AdaBoost: Algorithm

Here $\alpha_1, \alpha_2, \ldots, \alpha_M$ are computed by the boosting algorithm, and weight the contribution of each respective $f_m(x)$.

Their effect is to give higher influence to the more accurate classifiers in the sequence.
AdaBoost: Algorithm

• The data modifications at each boosting step consist of applying weights $w_1, w_2, \ldots, w_N$ to each of the training observations $(x_i, y_i)$, $i = 1, 2, \ldots, N$.

• The weight is initially set $1/N$ for all data points.

All data points have the same weight as represented by the size of the circle.
AdaBoost: Algorithm

- So the **first step** simply trains the classifier on the data **in the usual manner**.
- Assume that we have a procedure available for **training a base classifier** using weighted data to give a function $y(x) \in \{-1, 1\}$.

**Decision boundary** for this base learner is shown by the **dashed black line**.
AdaBoost: Algorithm

- Clearly some data points are **misclassified by the base learner**.
- The next learner ($m = 2$) will need to **focus on these misclassified points**.
- So, the weights of the data points **need to be adjusted**.
AdaBoost: Algorithm

- At step $m = 2$, those observations that were *misclassified by the classifier* $f_{m-1}(x)$ *induced at the previous step* have their weights increased (boosted), represented by larger circles.
- Whereas the weights are decreased (smaller circles) for those that were classified correctly.
AdaBoost: Algorithm

- Thus as iterations proceed, observations that are **difficult to classify** correctly **receive ever-increasing influence**.
- Each successive classifier is thereby **forced to concentrate on those training observations** that are missed by previous ones in the sequence.
AdaBoost: Algorithm

- Finally, when the desired number of base classifiers have been trained, they are combined to form a committee using coefficients that give different weights to different base classifiers.
AdaBoost: Algorithm

- Each data point is depicted by a circle whose **radius indicates the weight assigned to that data point** when training the most recently added base learner.
- Thus, for instance, we see that points that are misclassified by the $m = 1$ base learner are **given greater weight** when training the $m = 2$ base learner.
AdaBoost: Algorithm

- We will present the AdaBoost algorithm using an intuitive justification.
- Then, we will provide the mathematical justification for the formulas used in the algorithm.
AdaBoost: Algorithm

- Note that our goal is to find the learner’s weighted coefficient $\alpha_m$. 

$$Y_M(x) = \text{sign} \left( \sum_{m=1}^{M} \alpha_m y_m(x) \right)$$
AdaBoost: Algorithm

- The $\alpha_m$ depends on the **weighted error rate** of the learners $\varepsilon_m$.
- The $\varepsilon_m$ is computed by using the **weighting coefficient** $w_n$.

$$
\alpha_m = \eta \ln \left\{ \frac{1 - \varepsilon_m}{\varepsilon_m} \right\}
$$

Therefore, our goal is to compute these **three parameters**: $\alpha_m$, $\varepsilon_m$, $w_n$. 
AdaBoost: Algorithm

1. **Initialize data weighting coefficient**: \( w_i = 1/N \)

2. For \( m = 1: M \) do
   a. Fit a classifier \( y_m(x) \) to the training data by minimizing the weighted error function
   b. Compute the weighted measure of the error rate \( \varepsilon_m \) of the predictor
   c. Evaluate predictor’s weighted coefficient \( \alpha_m \)
   d. Update the data weighting coefficients:
      \[
      w_n^{(m+1)} = w_n^{(m)} \exp \left\{ \alpha_m I(y_m(x_n) \neq t_n) \right\}
      \]

3. Make predictions using the final model, which is given by:
   \[
   Y_M(x) = \text{sign} \left( \sum_{m=1}^{M} \alpha_m y_m(x) \right)
   \]
AdaBoost: Algorithm

1. Initialize the data weighting coefficients \{w_n\} by setting \(w_n^{(1)} = 1/N\) for \(n = 1, \ldots, N\).

2. For \(m = 1, \ldots, M\):

   2(a). Fit a classifier \(y_m(x)\) to the training data by minimizing the weighted error function

   \[
   J_m = \sum_{n=1}^{N} w_n^{(m)} I(y_m(x_n) \neq t_n)
   \]

   Note: here \(w_n^{(m)}\) is the weight of the misclassified data points.

At each iteration this weight \(w_n^{(m)}\) for the misclassified data points will be increased (boosted).

Next classifier will try to improve the error by classifying these points.
2(b). Compute the weighted measure of the error rate:

$$
\epsilon_m = \frac{\sum_{n=1}^{N} w_n^{(m)} I(y_m(x_n) \neq t_n)}{\sum_{n=1}^{N} w_n^{(m)}}
$$

2(c). Evaluate learner’s weighted coefficient:

$$
\alpha_m = \eta \ln \left( \frac{1 - \epsilon_m}{\epsilon_m} \right)
$$

- The more accurate the predictor is, the higher its weight $\alpha$ will be.
- If it is just guessing randomly, then its weight will be close to zero.
- However, if it is most often wrong (i.e., less accurate than random guessing), then its weight will be negative.

Error = sum of the weights of the misclassified points / Sum of the weights of all points

Note: initially all data points have the same weight = 1/N

Here, $\eta$ is the learning rate
AdaBoost: Algorithm

2(d). **Update** the data weighting coefficients.

$$w_n^{(m+1)} = \begin{cases} w_n^{(m)} & \text{if } y_m(x_n) = t_n \\ w_n^{(m)} \exp\{\alpha_m\} & \text{if } y_m(x_n) \neq t_n \end{cases}$$

It **boosts** the misclassified instances.

3. Make predictions using the final model, which is given by:

$$Y_M(x) = \text{sign} \left( \sum_{m=1}^{M} \alpha_m y_m(x) \right)$$

To make predictions, compute the predictions of all the predictors and **weigh** them using the predictor weights $\alpha_m$.

The predicted class is the one that receives the **majority of weighted votes**.
AdaBoost: Algorithm

• Now we provide the mathematical justification for the formulas used in the AdaBoost algorithm.
AdaBoost: Algorithm

• For AdaBoost, we use an **exponential error/loss function** as follows.

• Here, \( t_n \in \{-1, 1\} \) are the **training set target values**.

\[
E = \sum_{n=1}^{N} \exp \{ -t_n f_m(x_n) \}
\]

Also \( f_m(x) \) is a classifier defined in terms of a linear combination of base classifiers \( y_l(x) \) of the form:

\[
f_m(x) = \frac{1}{2} \sum_{l=1}^{m} \alpha_l y_l(x)
\]
AdaBoost: Algorithm

- Our goal is to minimize $E$ with respect to both
  - the *weighting coefficients* $\alpha_i$ and
  - the parameters of the base classifiers $y_i(x)$.

\[
E = \sum_{n=1}^{N} \exp \{-t_n f_m(x_n)\}
\]

\[
f_m(x) = \frac{1}{2} \sum_{l=1}^{m} \alpha_l y_l(x)
\]
AdaBoost: Algorithm

• Note that performing a global error function minimization is infeasible.
• Hence, we assume that the 1 to (m - 1) base classifiers $y_1(x), \ldots, y_{m-1}(x)$ are fixed.
• Also their coefficients 1 to (m - 1) are fixed: $\alpha_1, \ldots, \alpha_{m-1}$.
• So we are minimizing $E$ only with respect to $\alpha_m$ and $y_m(x)$.

$$E = \sum_{n=1}^{N} \exp \{-t_n f_m(x_n)\}$$

$$f_m(x) = \frac{1}{2} \sum_{l=1}^{m} \alpha_l y_l(x)$$
AdaBoost: Algorithm

- Separating off the contribution from base classifier $y_m(x)$, we can then re-write the error function in the form:

$$f_m(x) = \frac{1}{2} \sum_{l=1}^{m} \alpha_l y_l(x)$$

Here $w_n^{(m)}$ is viewed as a constant as we are optimizing only $\alpha_m$ and $y_m(x)$.
AdaBoost:
Algorithm

• Let’s define expressions for correctly & incorrectly classified data points.
• Denoted by $T_m$: the set of data points that are correctly classified by $y_m(x)$.
• Denoted by $M_m$: the remaining misclassified points.
• Rewrite the error function in the form:

$$E = \sum_{n=1}^{N} w_n^{(m)} \exp\left\{-\frac{1}{2} t_n \alpha_m y_m(x_n)\right\}$$

$$E = e^{-\alpha_m/2} \sum_{n \in T_m} w_n^{(m)} + e^{\alpha_m/2} \sum_{n \in M_m} w_n^{(m)}$$

Let’s try to understand how did we rewrite the error function
AdaBoost: Algorithm

Consider the 1st term of the following equation (correct classification).

We will derive it now from the above equation.

\[
E = \sum_{n=1}^{N} w_n^{(m)} \exp \left\{-\frac{1}{2} t_n \alpha_m y_m(x_n) \right\}
\]

Correct classification:
\[t_n = y_m(x_n), \text{ which is } -1 = -1 \text{ or } +1 = +1\]
Then, \[t_n \ast y_m(x_n) = 1\]

Hence, \[\exp(-\frac{1}{2} \alpha_m \ast t_n \ast y_m(x_n)) = \exp(-\frac{1}{2} \alpha_m)\]
AdaBoost: Algorithm

- Consider the 2\textsuperscript{nd} term the following equation (incorrect classification).
- We will derive it now from the above equation.

\[
E = \sum_{n=1}^{N} w_n^{(m)} \exp\left\{-\frac{1}{2} t_n \alpha_m y_m(x_n)\right\}
\]

**Incorrect classification:**
\[t_n \neq y_m(x_n), \text{ which is } -1 = +1 \text{ or } +1 = -1\]
Then, \[t_n \ast y_m(x_n) = -1\]

Hence, \[
\exp(-(1/2)\alpha_m t_n \ast y_m(x_n)) = \exp((1/2)\alpha_m)
\]
AdaBoost: Algorithm

- Denoted by $T_m$: the set of data points that are **correctly classified** by $y_m(x)$.
- Denoted by $M_m$: the **remaining misclassified points**.
- **Rewrite the error function** in the form:

\[
E = e^{-\alpha_m/2} \sum_{n \in T_m} w_n^{(m)} + e^{\alpha_m/2} \sum_{n \in M_m} w_n^{(m)}
\]

\[
= (e^{\alpha_m/2} - e^{-\alpha_m/2}) \sum_{n=1}^{N} w_n^{(m)} I(y_m(x_n) \neq t_n) + e^{-\alpha_m/2} \sum_{n=1}^{N} w_n^{(m)}.
\]

Here $I(y_m(x_n) \neq t_n)$ is the **indicator function**

- $I(y_m(x_n) \neq t_n) = 1$ when $y_m(x_n) \neq t_n$
- $I(y_m(x_n) \neq t_n) = 0$ when $y_m(x_n) = t_n$
AdaBoost: Algorithm

Let’s try to understand the following equation.

Here we have re-written the equation for all N data points.

1\textsuperscript{st} part: subtracting correct classification from the incorrect classification.

2\textsuperscript{nd} part: adding the correct classification.

Here the 2\textsuperscript{nd} part is a constant.

The goal is to isolate the part that depends on $y_m(x_n)$.

$$E = \left( e^{\alpha_m/2} - e^{-\alpha_m/2} \right) \sum_{n=1}^{N} w_n^{(m)} I(y_m(x_n) \neq t_n) + e^{-\alpha_m/2} \sum_{n=1}^{N} w_n^{(m)}$$
AdaBoost: Algorithm

• In summary: the 1\textsuperscript{st} term isolates the **misclassified data points** from all N points.

• Our goal is to **minimize the misclassification error** of the 1\textsuperscript{st} term.

\[
E = (e^{\alpha_m/2} - e^{-\alpha_m/2}) \sum_{n=1}^{N} w_n^{(m)} I(y_m(x_n) \neq t_n) + e^{-\alpha_m/2} \sum_{n=1}^{N} w_n^{(m)}
\]
AdaBoost: Algorithm

- We **minimize** this (1st term) with respect to $y_m(x)$.
- The overall multiplicative factor in front of the summation does not affect the location of the minimum.

\[
E = \left( e^{\alpha_m/2} - e^{-\alpha_m/2} \right) \sum_{n=1}^{N} w_n^{(m)} I(y_m(x_n) \neq t_n) + e^{-\alpha_m/2} \sum_{n=1}^{N} w_n^{(m)}
\]

Hence we minimize, the cost function denoted by $J$:

\[
J_m = \sum_{n=1}^{N} w_n^{(m)} I(y_m(x_n) \neq t_n)
\]

$w_n^{(m)}$ is a constant:

\[
w_n^{(m)} = \exp\{-t_n f_{m-1}(x_n)\}\]
Adam Boost: Algorithm

Similarly, minimizing with respect to $\alpha_m$, we obtain:

$$\frac{\partial E}{\partial \alpha_m} = \frac{1}{2} \left( (e^{\alpha_m/2} + e^{-\alpha_m/2}) \sum_{n=1}^{N} w_n^{(m)} I(y_m(x_n) \neq t_n) - e^{-\alpha_m/2} \sum_{n=1}^{N} w_n^{(m)} \right).$$

Using the following observations:

$$\frac{d}{d\alpha_m} e^{\alpha_m/2} = \frac{1}{2} e^{\alpha_m/2}$$

$$\frac{d}{d\alpha_m} e^{-\alpha_m/2} = -\frac{1}{2} e^{-\alpha_m/2}$$

$$E = (e^{\alpha_m/2} - e^{-\alpha_m/2}) \sum_{n=1}^{N} w_n^{(m)} I(y_m(x_n) \neq t_n) + e^{-\alpha_m/2} \sum_{n=1}^{N} w_n^{(m)}$$
AdaBoost: Algorithm

Correct Classification: $e^{-\alpha_m/2}$

Incorrect Classification: $e^{+\alpha_m/2}$

- Setting this equal to zero and rearranging we get:

$$\frac{\partial E}{\partial \alpha_m} = \frac{1}{2} \left( (e^{\alpha_m/2} + e^{-\alpha_m/2}) \sum_{n=1}^{N} w_n^{(m)} I(y_m(x_n) \neq t_n) - e^{-\alpha_m/2} \sum_{n=1}^{N} w_n^{(m)} \right).$$

$$\frac{\sum_n w_n^{(m)} I(y_m(x_n) \neq t_n)}{\sum_n w_n^{(m)}} = \frac{e^{-\alpha_m/2}}{e^{\alpha_m/2} + e^{-\alpha_m/2}} = \frac{1}{e^{\alpha_m} + 1}.$$
AdaBoost: Algorithm

Based on this we define the weighted measure of the error rate $\varepsilon_m$ of the $m$’th predictor:

$$
\frac{\sum_n w_n^{(m)} I(y_m(x_n) \neq t_n)}{\sum_n w_n^{(m)}} = \frac{e^{-\alpha_m/2}}{e^{\alpha_m/2} + e^{-\alpha_m/2}} = \frac{1}{e^{\alpha_m} + 1}.
$$

$w_n^{(m)}$ is a constant:

$$
w_n^{(m)} = \exp\{-t_n f_{m-1}(x_n)\}
$$
AdaBoost: Algorithm

- Using $\epsilon_m$ we can evaluate the weighting coefficient $\alpha_m$:

$$
e^{\alpha_m} = \frac{1 - \epsilon_m}{\epsilon_m}$$

$$
\alpha_m = \ln \left\{ \frac{1 - \epsilon_m}{\epsilon_m} \right\}
$$

The more accurate the predictor is, the higher its weight will be.

However, if it is most often wrong (i.e., less accurate than random guessing), then its weight will be negative.

If it is just guessing randomly, then its weight will be close to zero.
AdaBoost: Algorithm

• We see that, having found $\alpha_m$ and $y_m(x)$, the weights on the data points are updated using:

$$w^{(m+1)}_n = w^{(m)}_n \exp \left\{ -\frac{1}{2} t_n \alpha_m y_m(x_n) \right\}$$

$$E = \sum_{n=1}^{N} \exp \left\{ -t_n f_{m-1}(x_n) - \frac{1}{2} t_n \alpha_m y_m(x_n) \right\}$$

$$= \sum_{n=1}^{N} w^{(m)}_n \exp \left\{ -\frac{1}{2} t_n \alpha_m y_m(x_n) \right\}$$

$$f_m(x) = \frac{1}{2} \sum_{l=1}^{m} \alpha_l y_l(x)$$

$$\epsilon_m = \frac{\sum_{n=1}^{N} w^{(m)}_n I(y_m(x_n) \neq t_n)}{\sum_{n=1}^{N} w^{(m)}_n}$$

$$\alpha_m = \ln \left\{ \frac{1 - \epsilon_m}{\epsilon_m} \right\}$$
AdaBoost: Algorithm

• We see that the weights $w_n^{(m)}$ are **updated at the next iteration using**:

$$w_n^{(m+1)} = w_n^{(m)} \exp(-\alpha_m/2) \exp \{\alpha_m I(y_m(x_n) \neq t_n)\}$$

Because the term $\exp(-\alpha_m/2)$ is independent of $n$, we see that it weighs all data points by the same factor and so can be discarded.

Thus we obtain:

$$w_n^{(m+1)} = w_n^{(m)} \exp \{\alpha_m I(y_m(x_n) \neq t_n)\}$$
AdaBoost: Algorithm

- Notice that in the weight update rule the misclassified instances are boosted.

Weight update Rule:

\[
w_n^{(m+1)} = w_n^{(m)} \exp \{ \alpha_m I(y_m(x_n) \neq t_n) \}
\]
AdaBoost: Algorithm

- To make predictions, AdaBoost simply computes the predictions of all the predictors and weighs them using the predictor weights $\alpha_m$.
- The predicted class is the one that receives the majority of weighted votes.

$$f_m(x) = \frac{1}{2} \sum_{l=1}^{m} \alpha_l y_l(x)$$

$$\alpha_m = \ln \left\{ \frac{1 - \epsilon_m}{\epsilon_m} \right\}$$

$$\epsilon_m = \frac{\sum_{n=1}^{N} w_n^{(m)} I(y_m(x_n) \neq t_n)}{\sum_{n=1}^{N} w_n^{(m)}}$$
AdaBoost: Algorithm

• Recall that, \( t_n \in \{-1, 1\} \) are the training set target values.
• So the target is either negative or positive.
• The predicted class \( y_l \) would be either negative or positive.
• Hence, once all the base classifiers are trained, new data points are classified by evaluating the sign of the combined function defined according to:

\[
    f_m(x) = \frac{1}{2} \sum_{l=1}^{m} \alpha_l y_l(x)
\]

Because the factor of 1/2 does not affect the sign it can be omitted, giving:

\[
    Y_M(x) = \text{sign} \left( \sum_{m=1}^{M} \alpha_m y_m(x) \right)
\]
AdaBoost: Algorithm

1. **Initialize data weighting coefficient**: \( w_i = 1/N \)

2. For \( m = 1: M \) do
   a. Fit a classifier \( y_m(x) \) to the training data by minimizing the weighted error function
   b. Compute the weighted measure of the error rate \( \varepsilon_m \) of the predictor
   c. Evaluate predictor’s weighted coefficient \( \alpha_m \)
   d. Update the data weighting coefficients: \( w_n^{(m+1)} = w_n^{(m)} \exp \{\alpha_m I(y_m(x_n) \neq t_n)\} \)

3. Make predictions using the final model, which is given by:

\[
Y_M(x) = \text{sign} \left( \sum_{m=1}^{M} \alpha_m y_m(x) \right)
\]
AdaBoost: Algorithm

1. Initialize the data weighting coefficients \( \{w_n\} \) by setting \( w_n^{(1)} = 1/N \) for \( n = 1, \ldots, N \).

2. For \( m = 1, \ldots, M \):
   
   2(a). Fit a classifier \( y_m(x) \) to the training data by minimizing the weighted error function

   \[
   J_m = \sum_{n=1}^{N} w_n^{(m)} I(y_m(x_n) \neq t_n)
   \]

   Note: here \( w_n^{(m)} \) is the weight of the misclassified data points.

At each iteration this weight \( w_n^{(m)} \) for the misclassified data points will be increased (boosted).

Next classifier will try to improve the error by classifying these points.
2(b). Compute the **weighted measure of the error rate**:

\[
\epsilon_m = \frac{\sum_{n=1}^{N} w_n^{(m)} I(y_m(x_n) \neq t_n)}{\sum_{n=1}^{N} w_n^{(m)}}
\]

2(c). Evaluate **predictor’s weighted coefficient**:

\[
\alpha_m = \eta \ln \left( \frac{1 - \epsilon_m}{\epsilon_m} \right)
\]

The **more accurate** the predictor is, the **higher its weight** will be.

If it is just guessing randomly, then its **weight will be close to zero**.

However, if it is most often wrong (i.e., less accurate than random guessing), then its weight will be **negative**.

Error = sum of the weights of the **misclassified points** / Sum of the weights of all points

Note: initially all data points have the same weight = 1/N

Here, \( \eta \) is the **learning rate**
AdaBoost: Algorithm

2(d). **Update** the data weighting coefficients.

\[ w_n^{(m+1)} = w_n^{(m)} \exp \{ \alpha_m I(y_m(x_n) \neq t_n) \} \]

It **boosts** the misclassified instances.

3. Make predictions using the final model, which is given by:

\[ Y_M(x) = \text{sign} \left( \sum_{m=1}^{M} \alpha_m y_m(x) \right) \]

To make predictions, compute the predictions of all the predictors and **weighs** them using the predictor weights \( \alpha_m \).

The predicted class is the one that receives the **majority of weighted votes**.
AdaBoost

\[ \alpha_m = \eta \ln \left( \frac{1 - \epsilon_m}{\epsilon_m} \right) \]

- The following figure shows the decision boundaries of five consecutive predictors on the moons dataset.
- In this example, each predictor is a highly regularized SVM classifier with an RBF kernel.
AdaBoost

$\alpha_m = \eta \ln \left( \frac{1 - \epsilon_m}{\epsilon_m} \right)$

- The first classifier gets many instances wrong, so their weights get boosted.
- The second classifier therefore does a better job on these instances, and so on.
AdaBoost

\[ \alpha_m = \eta \ln \left( \frac{1 - \epsilon_m}{\epsilon_m} \right) \]

- The **plot on the right** represents the same sequence of predictors except that the learning rate is halved.
- I.e., the misclassified instance weights are **boosted half as much** at every iteration.
AdaBoost

\[ \alpha_m = \eta \ln \left( \frac{1 - \epsilon_m}{\epsilon_m} \right) \]

- We see this sequential learning technique has some similarities with Gradient Descent.
- Here the difference is that instead of tweaking a single predictor’s parameters to minimize a cost function, AdaBoost adds predictors to the ensemble, gradually making it better.
AdaBoost

\[ \alpha_m = \eta \ln \left( \frac{1 - \epsilon_m}{\epsilon_m} \right) \]

- Once all predictors are trained, the ensemble makes predictions very much like bagging or pasting.
- The difference is that predictors have different weights depending on their overall accuracy on the weighted training set.
AdaBoost

$$\alpha_m = \eta \ln \left( \frac{1 - \epsilon_m}{\epsilon_m} \right)$$

- Effect of the **learning rate or shrinkage parameter**:
- Two Gradient Boosted Regression Tree (GBRT) ensembles trained with a low learning rate.
- For low shrinkage, we will **need more trees in the ensemble** to fit the training set, but the predictions will usually generalize better.
AdaBoost

\[ \alpha_m = \eta \ln \left( \frac{1 - \epsilon_m}{\epsilon_m} \right) \]

- Left figure: does not have enough trees to fit the training set.
- Right figure: has too many trees and overfits the training set.
Boosting: Choice of Loss Function

- Why did we choose the **exponential loss function** in the AdaBoost algorithm?

![Exponential loss function formula](image)

Also \( f_m(x) \) is a classifier defined in terms of a linear combination of base classifiers \( y_l(x) \) of the form:

![Classifier formula](image)

<table>
<thead>
<tr>
<th>Name</th>
<th>Loss</th>
<th>Derivative</th>
<th>( f^* )</th>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Squared error</td>
<td>( \frac{1}{2}(y_i - f(x_i))^2 )</td>
<td>( y_i - f(x_i) )</td>
<td>( \mathbb{E}[y</td>
<td>x_i] )</td>
</tr>
<tr>
<td>Absolute error</td>
<td>(</td>
<td>y_i - f(x_i)</td>
<td>)</td>
<td>( \text{sgn}(y_i - f(x_i)) )</td>
</tr>
<tr>
<td>Exponential loss</td>
<td>( \exp(-\tilde{y}_i f(x_i)) )</td>
<td>( -\tilde{y}_i \exp(-\tilde{y}_i f(x_i)) )</td>
<td>( \frac{1}{2} \log \frac{\pi_i}{1-\pi_i} )</td>
<td>AdaBoost</td>
</tr>
<tr>
<td>Logloss</td>
<td>( \log(1 + e^{-\tilde{y}_i f_i}) )</td>
<td>( y_i - \pi_i )</td>
<td>( \frac{1}{2} \log \frac{\pi_i}{1-\pi_i} )</td>
<td>LogitBoost</td>
</tr>
</tbody>
</table>
Boosting: Choice of Loss Function

• The AdaBoost algorithm is based on the forward stagewise additive modeling.

• The principal attraction of exponential loss in the context of additive modeling is computational.

• It leads to the simple modular reweighting AdaBoost algorithm.
Boosting: Choice of Loss Function

• The exponential loss function makes the additive modeling convenient.

• Interestingly the AdaBoost algorithm was originally motivated from a very different perspective than we presented.

• Its equivalence to forward stagewise additive modeling based on exponential loss was only discovered five years after its inception.
Gradient Boosting

• Another very popular Boosting algorithm is Gradient Boosting.

• Just like AdaBoost, Gradient Boosting works by sequentially adding predictors to an ensemble, each one correcting its predecessor.
Gradient Boosting

• However, instead of *tweaking the instance weights* at every iteration like AdaBoost does,

  this method tries to **fit the new predictor to the residual errors** made by the previous predictor.
Gradient Boosting: Example

- Consider a **simple regression example**.
- It uses **Decision Trees** as the base predictors.
- This is called Gradient Tree Boosting, or **Gradient Boosted Regression Trees (GBRT)**.
Gradient Boosting: Example

- We have noisy quadratic training data set.
- We will use three Decision Trees to train the GBRT model.
- Assume the “X” is the training data.
- Also assume that “y” is the label.
- Train the 1\textsuperscript{st} Decision tree, and predict “\(y_{1\text{\_predict}}\)”.
- Then, compute the \textbf{residual error} made by the first predictor:
  \[
  y_2 = y - y_{1\text{\_predict}}
  \]
- Train the 2\textsuperscript{nd} Decision Tree using this residual error (\(y_2\)) and predict “\(y_{2\text{\_predict}}\)”.
Gradient Boosting: Example

• Then, compute the residual error made by the 2\textsuperscript{nd} predictor:
  \[ y_3 = y_2 - y_2\text{\_predict} \]
• Train the 3\textsuperscript{rd} Decision Tree using this residual error (y3).
• Now we have an ensemble containing three trees.
• It can make predictions on a new instance simply by adding up the predictions of all the trees.
Gradient Boosting: Example

Predictions of the three trees in the left column, and the ensemble’s predictions in the right column.
Gradient Boosting: Example

Ensemble’s predictions gradually get better as trees are added to the ensemble.
Gradient Boosting: Formalization

- Step 1: Initialization
- Here, $x_i$ is the training data, $y_i$ is the label and $\Phi(x_i; \gamma)$ is the predictor/learner.
- Goal is to learn the \textbf{optimal parameter $\gamma$} for the learner by \textbf{minimizing loss}.

$$f_0(x) = \arg\min_{\gamma} \sum_{i=1}^{N} L(y_i, \phi(x_i; \gamma))$$
Gradient Boosting: Formalization

- Step 2: for $m = 1: M$ do
- 2(a): Compute the **gradient residual error**

$$ r_{i,m} = - \left[ \frac{\partial L(y_i, f(x_i))}{\partial f(x_i)} \right] f(x_i) = f_{m-1}(x_i) $$

**Gradients** for some common loss functions:

<table>
<thead>
<tr>
<th>Name</th>
<th>Loss</th>
<th>Derivative</th>
<th>$f^*$</th>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Squared error</td>
<td>$\frac{1}{2}(y_i - f(x_i))^2$</td>
<td>$y_i - f(x_i)$</td>
<td>$\mathbb{E}[y</td>
<td>x_i]$</td>
</tr>
<tr>
<td>Absolute error</td>
<td>$</td>
<td>y_i - f(x_i)</td>
<td>$</td>
<td>$\text{sgn}(y_i - f(x_i))$</td>
</tr>
<tr>
<td>Exponential loss</td>
<td>$\exp(-\tilde{y}_i f(x_i))$</td>
<td>$-\tilde{y}_i \exp(-\tilde{y}_i f(x_i))$</td>
<td>$\frac{1}{2} \log \frac{\pi_i}{1-\pi_i}$</td>
<td>AdaBoost</td>
</tr>
<tr>
<td>Logloss</td>
<td>$\log(1 + e^{-\tilde{y}_i f(x_i)})$</td>
<td>$y_i - \pi_i$</td>
<td>$\frac{1}{2} \log \frac{\pi_i}{1-\pi_i}$</td>
<td>LogitBoost</td>
</tr>
</tbody>
</table>
Gradient Boosting: Formalization

• Step 2: for \( m = 1: M \) do
  2(b): Use the weak learner to compute \( \gamma_m \) that minimizes:

\[
\sum_{i=1}^{N} (r_{im} - \phi(x_i; \gamma_m))^2
\]

2(c): Update

\[
f_m(\hat{x}) = f_{m-1}(\hat{x}) + \eta \phi(\hat{x}, \gamma_m)
\]

Step 3: Return

\[
f(x) = f_M(x)
\]
Boosting: Pitfalls

• There is **one important drawback** to this sequential learning technique.

• It **cannot be parallelized** (or only partially), since each predictor can only be trained after the previous predictor has been trained and evaluated.

• As a result, it **does not scale as well** as bagging or pasting.
Why Does Boosting Work So Well?

• There are **two main reasons** for this.

• First, it can be seen as a form of $l_1$ regularization, which is known to help prevent overfitting by eliminating “irrelevant” features.
Why Does Boosting Work So Well?

• To see this, imagine **pre-computing all possible weak-learners**, and defining a feature vector of the form
  \[ \phi(x) = [\phi_1(x), \ldots, \phi_K(x)] \].

• We could use **l_1 regularization to select a subset of these**.

• Alternatively we can use boosting, where at each step, the **weak learner creates a new \( \phi_k \) on the fly**.
Why Does Boosting Work So Well?

• It is possible to combine boosting and $l_1$ regularization.

• Then, we get an algorithm known as **L1-Adaboost**.

• Essentially this method **greedily adds the best features** (weak learners) using boosting, and then **prunes off irrelevant ones** using $l_1$ regularization.
Why Does Boosting Work So Well?

• Another explanation has to do with the concept of margin.

• AdaBoost maximizes the margin on the training data.
So far, our presentation of boosting has been very frequentist, since it has focused on greedily minimizing loss functions.

We can also give a likelihood interpretation of the algorithm. The idea is to consider a mixture of experts model of the following form.

Here each expert $p(y \mid x, \gamma_m)$ is like a weak learner.

$$p(y \mid x, \theta) = \sum_{m=1}^{M} \pi_m p(y \mid x, \gamma_m)$$
Boosting: A Bayesian View

• We usually fit all M experts at once using the **Expectation Maximization (EM) algorithm**.

• But we can **imagine a sequential scheme**, whereby we only update the parameters for **one expert at a time**.

\[
p(y|x, \theta) = \sum_{m=1}^{M} \pi_m p(y|x, \gamma_m)
\]
Boosting: A Bayesian View

• In the “E” step, the posterior responsibilities will reflect how well the existing experts explain a given data point.

• If this is a poor fit, these data points will have more influence on the next expert that is fitted.

\[ p(y|x, \theta) = \sum_{m=1}^{M} \pi_m p(y|x, \gamma_m) \]
Boosting: A Bayesian View

• This view naturally suggest a way to use a boosting-like algorithm for unsupervised learning: we simply sequentially fit mixture models, instead of mixtures of experts.

\[ p(y|x, \theta) = \sum_{m=1}^{M} \pi_m p(y|x, \gamma_m) \]
Boosting: Application

• Visually detecting all instances of some object type, such as human faces, in photographs, movies, and other digital images is a fundamental problem in computer vision.
• Viola and Jones (2001). Rapid object detection using a boosted cascade of simple features.
Boosting: Application

- Transform what is really a search task (looking for faces) into a classification problem.
- To do so, we can regard our instances as small subimages of size, say, $24 \times 24$ pixels.
- Each of which would be considered positive if and only if it captures a full frontal shot of a face at a standard scale.
Boosting: Application

• Consider the following picture.
• Clearly, an **accurate classifier** for such subimages can be used to **detect all faces in an image** simply by scanning the entire image and reporting the presence of a face anywhere that it registers a positive instance.
Boosting: Application

• Faces of varying sizes can be found by repeating this process at various scales.

• Needless to say, such an exhaustive process demands that a very fast classifier be used in the innermost loop.
Boosting: Application

- We can use **weak classifiers** which **merely detect rectangular patterns** of relative light and darkness in the image.
- The one on the left figure is sensitive to a dark region over a light region **at the specified location of the image**.
- The one on the right figure is similarly sensitive to dark regions surrounding a light region.
Boosting: Application

• In more precise terms, such a pattern defines a real-valued feature that is equal to the sum of the intensities of all the pixels in the black rectangle(s) minus the sum of the intensities of all the pixels in the white rectangle(s).

• Such a feature can be used to define a decision stump that makes its predictions based on whether the feature value for a particular image is above or below some threshold.
Boosting: Application

• The first feature (top-left) apparently exploits the tendency of the eyes to appear darker than the upper cheeks.
• The second feature (top-right) exploits a similar tendency for the eyes to appear darker than the bridge of the nose.
Boosting: Application

- Based on these two features a **weak learner will individually do a very poor job** of identifying faces.
Boosting: Application

• During training, we can consider features defined by all possible patterns of a small number of types, such as the four shown below.
• Each one of these types defines a large number of patterns, each of which is identified with a feature.
• For instance, the one on the left defines all possible patterns consisting of a white rectangle directly above a black rectangle of equal size.
• In 24 × 24 pixel images, the four types define some 45,396 features.
Boosting: Application

- Having defined this large set of real-valued features, we can apply AdaBoost to find the best decision stump.
- Following figure shows the two features found on the first two rounds of boosting.
Boosting: Application

• The performance of AdaBoost’s final classifier is extremely good.
• For instance, after 200 rounds of boosting, on one test dataset, the final classifier was able to detect 95% of the faces while reporting false positives at a rate of only 1 in 14,084.
Boosting: Application

• The classifier was trained on 1000s of manually labeled images of faces and non-faces, and then was applied to a dense set of overlapping patches in the test image.
• Only the patches whose probability of containing a face was sufficiently high were returned.
Boosting: Application

• In addition to its high accuracy, this approach to face detection can be made extremely fast.

• Evaluation can be made even faster by using a cascading technique in which relatively small and rough classifiers are trained which are good enough to quickly eliminate the vast majority of background images as non-faces.

• The entire system is so fast that it can be used, for instance, to find all faces in video in real time at 15 frames per second.