Classification: Predict a discrete-valued target

Binary classification: Targets $t \in \{-1, +1\}$ (This is different than previous lectures where we had $t \in \{0, +1\}$).

Linear model:

$$z = \mathbf{w}^\top \mathbf{x} + b$$

$$y = \text{sign}(z)$$

Question: How should we choose $\mathbf{w}$ and $b$?
Zero-One Loss

- We can use the 0 – 1 loss function, and find the weights that minimize it over data points

\[
\mathcal{L}_{0-1}(z, t) = \begin{cases} 
0 & \text{if } \text{sign}(z) = t \\
1 & \text{if } \text{sign}(z) \neq t
\end{cases}
\]

\[= \mathbb{I}\{y \neq t\}.\]

- But minimizing this loss is computationally difficult.

- We investigated some other loss functions that are easier to minimize, e.g., square loss or logistic regression with the cross-entropy loss \(\mathcal{L}_{\text{CE}}\).
Previously, we saw loss functions as relaxations to $0-1$ loss. We consider a new different relaxation here: Hinge loss

\[ L_{0-1}(z, t) = \mathbb{I}\{\text{sign}(z) \neq t\} \]

Hinge loss: \[ L_H(z, t) = \max\{0, 1 - zt\} \]

Note that the hinge loss is a convex upper bound of the $0-1$ loss.
Support Vector Machine (SVM)

If we use a linear classifier and write \( z^{(i)}(w, b) = w^\top x^{(i)} + b \), then minimizing the training loss with the hinge loss would be

\[
\min_{w,b} \sum_{i=1}^{N} \max\{0, 1 - t^{(i)} z^{(i)}(w, b)\}
\]

- The loss function \( \mathcal{L}_H(z,t) = \max\{0, 1 - tz\} \) is called the hinge loss (we wrote the loss in terms of \( z \) this time).
- This is often minimized along with the \( \ell_2 \) regularization:

\[
\min_{w,b} \sum_{i=1}^{N} \max\{0, 1 - t^{(i)} z^{(i)}(w, b)\} + \frac{\lambda}{2} \|w\|^2
\]

- This formulation is called Support Vector Machines (SVMs). SVM is a regularized empirical risk minimizer with the choice of hinge loss and the \( \ell_2 \) regularizer.
- SVMs are often formulated within a function spaced called reproducing kernel Hilbert space (RKHS).
How to train?

- How to fit $w, b$:
  - One option: gradient descent
Separating Hyperplanes

- There is a more elegant geometric derivation of support vector machines that we do not cover in detail in this class. Here is the idea.
- Suppose we are given these data points from two different classes and want to find a linear classifier that separates them.
The decision boundary looks like a line because $\mathbf{x} \in \mathbb{R}^2$, but think about it as a $D - 1$ dimensional hyperplane.

Recall that a hyperplane is described by points $\mathbf{x} \in \mathbb{R}^D$ such that $f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b = 0$. 
There are multiple separating hyperplanes, described by different parameters \((w, b)\).
There are more than one separating hyperplane, described by different $\mathbf{w}$. Which one should we choose?
Optimal Separating Hyperplane: A hyperplane that separates two classes and maximizes the distance to the closest point from either class, i.e., maximize the margin of the classifier.

- Intuitively, ensuring the decision boundary is not too close to any data points leads to better generalization on the test data.
- SVM maximizes the above margin. They are an example of large margin methods.
- If the data is not linearly separable, SVM allows some small number of margin violations.
Ensembles: Boosting

- Recall that an ensemble is a set of predictors whose individual decisions are combined in some way to classify new examples.

- (Previously) **Bagging**: Train classifiers independently on random subsamples of the training data.

- (This lecture) **Boosting**: Train classifiers sequentially, each time focusing on training data points that were previously misclassified.

- Let’s start with the concepts of weighted training sets and weak learner/classifier (or base classifiers).
The misclassification rate \( \frac{1}{N} \sum_{n=1}^{N} \mathbb{I}[h(x^{(n)}) \neq t^{(n)}] \) weights each training example equally.

Key idea: We can learn a classifier using different costs (aka weights) for examples.

- Classifier “tries harder” on examples with higher cost

Change cost function:

\[
\sum_{n=1}^{N} \frac{1}{N} \mathbb{I}[h(x^{(n)}) \neq t^{(n)}] \quad \text{becomes} \quad \sum_{n=1}^{N} w^{(n)} \mathbb{I}[h(x^{(n)}) \neq t^{(n)}]
\]

Usually require each \( w^{(n)} > 0 \) and \( \sum_{n=1}^{N} w^{(n)} = 1 \)
(Informal) Weak learner is a learning algorithm that outputs a hypothesis (i.e., a classifier) that performs slightly better than chance, e.g., it predicts the correct label with probability 0.51 in binary label case.

- It gets slightly less than 0.5 error rate (the worst case is 0.5)

- We are interested in weak learners that are \textit{computationally} efficient.
  - Decision trees
  - Even simpler: \textbf{Decision Stump}: A decision tree with a single split

[Formal definition of weak learnability has quantifiers such as “for any distribution over data” and the requirement that its guarantee holds only probabilistically.]
Weak Classifiers

These weak classifiers, which are decision stumps, consist of the set of horizontal and vertical half spaces.
A *single* weak classifier is not capable of making the training error very small. It only performs slightly better than chance, i.e., the error of classifier $h$ according to the given weights $\{w^{(1)}, \ldots, w^{(N)}\}$ (with $\sum_{n=1}^{N} w^{(n)} = 1$ and $w^{(n)} \geq 0$)

$$\text{err} = \sum_{n=1}^{N} w^{(n)} \mathbb{I}[h(x^{(n)}) \neq t^{(n)}]$$

is at most $\frac{1}{2} - \gamma$ for some small $\gamma > 0$.

Can we combine a set of weak classifiers in order to make a better ensemble of classifiers?
AdaBoost (Adaptive Boosting)

- **Boosting**: Train classifiers sequentially, each time assigning higher weight to training data points that were previously misclassified.

- **Key steps of AdaBoost**:
  1. At each iteration we re-weight the training samples by assigning larger weights to samples (i.e., data points) that were classified incorrectly.
  2. We train a new weak classifier based on the re-weighted samples.
  3. We add this weak classifier to the ensemble of weak classifiers. This ensemble is our new classifier.
  4. We repeat the process many times.

- The weak learner needs to minimize weighted error.

- AdaBoost reduces bias by making each classifier focus on previous mistakes.
Notation in This Lecture

- Input: Data $\mathcal{D}_N = \{x^{(n)}, t^{(n)}\}_{n=1}^N$ where $t^{(n)} \in \{-1, +1\}$
  - This is different from previous lectures where we had $t^{(n)} \in \{0, +1\}$
  - It is for notational convenience; otherwise, it is equivalent.

- A classifier or hypothesis $h : x \rightarrow \{-1, +1\}$

- 0-1 loss: $\mathbb{I}[h(x^{(n)}) \neq t^{(n)}] = \frac{1}{2} (1 - h(x^{(n)}) \cdot t^{(n)})$
AdaBoost Algorithm

- **Input:** Data $D_N$, weak classifier $\text{WeakLearn}$ (a classification procedure that returns a classifier $h$, e.g., best decision stump, from a set of classifiers $\mathcal{H}$, e.g., all possible decision stumps), number of iterations $T$
- **Output:** Classifier $H(x)$

Initialize sample weights: $w^{(n)} = \frac{1}{N}$ for $n = 1, \ldots, N$

For $t = 1, \ldots, T$

- Fit a classifier to data using weighted samples
  $(h_t \leftarrow \text{WeakLearn}(D_N, w))$, e.g.,
  
  $$h_t \leftarrow \arg\min_{h \in \mathcal{H}} \sum_{n=1}^{N} w^{(n)} I\{h(x^{(n)}) \neq t^{(n)}\}$$

- Compute weighted error
  $$err_t = \frac{\sum_{n=1}^{N} w^{(n)} I\{h_t(x^{(n)}) \neq t^{(n)}\}}{\sum_{n=1}^{N} w^{(n)}}$$

- Compute classifier coefficient
  $$\alpha_t = \frac{1}{2} \log \frac{1 - err_t}{err_t} \quad (\in (0, \infty))$$

- Update data weights
  $$w^{(n)} \leftarrow w^{(n)} \exp\left(-\alpha_t t^{(n)} h_t(x^{(n)})\right) \left[\equiv w^{(n)} \exp\left(2\alpha_t I\{h_t(x^{(n)}) \neq t^{(n)}\}\right)\right]$$

Return $H(x) = \text{sign} \left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$
Weighting Intuition

- Recall: \( H(x) = \text{sign} \left( \sum_{t=1}^{T} \alpha_t h_t(x) \right) \) where \( \alpha_t = \frac{1}{2} \log \frac{1-\text{err}_t}{\text{err}_t} \)

- Weak classifiers which get lower weighted error get more weight in the final classifier

- Also: \( w^{(n)} \leftarrow w^{(n)} \exp \left( 2\alpha_t \mathbb{I}\{h_t(x^{(n)}) \neq t^{(n)}\} \right) \)
  - If \( \text{err}_t \approx 0 \), \( \alpha_t \) high so misclassified examples get more attention
  - If \( \text{err}_t \approx 0.5 \), \( \alpha_t \) low so misclassified examples are not emphasized
AdaBoost Example

- Training data

\[ D_1 \]

- \( H \): decision trees with a single split (decision stumps)

[Slide credit: Verma & Thrun]
AdaBoost Example

- Round 1

\[ w = \left( \frac{1}{10}, \ldots, \frac{1}{10} \right) \Rightarrow \text{Train a classifier (using } w\text{)} \Rightarrow \text{err}_1 = \frac{\sum_{n=1}^{10} w^{(n)} [ h_1(x^{(n)}) \neq t^{(n)}]}{\sum_{n=1}^{10} w^{(n)}} = \frac{3}{10} \]

\[ \Rightarrow \alpha_1 = \frac{1}{2} \log \frac{1 - \text{err}_1}{\text{err}_1} = \frac{1}{2} \log \left( \frac{1}{0.3} - 1 \right) \approx 0.42 \Rightarrow H(x) = \text{sign} (\alpha_1 h_1(x)) \]

[Slide credit: Verma & Thrun]
AdaBoost Example

Round 2

\[ w \leftarrow \text{new weights} \Rightarrow \text{Train a classifier (using } w \text{)} \Rightarrow \text{err}_2 = \frac{\sum_{n=1}^{10} w^{(n)} I\{h_2(x^{(n)}) \neq t^{(n)}\}}{\sum_{n=1}^{10} w^{(n)}} = 0.21 \]

\[ \Rightarrow \alpha_2 = \frac{1}{2} \log \frac{1 - \text{err}_2}{\text{err}_2} = \frac{1}{2} \log \left( \frac{1}{0.21} - 1 \right) \approx 0.66 \Rightarrow H(x) = \text{sign} \left( \alpha_1 h_1(x) + \alpha_2 h_2(x) \right) \]
AdaBoost Example

Round 3

\[ w \leftarrow \text{new weights} \Rightarrow \text{Train a classifier (using } w) \Rightarrow \text{err}_3 = \frac{\sum_{n=1}^{10} w^{(n)}I\{h_3(x^{(n)}) \neq t^{(n)}\}}{\sum_{i=1}^{10} w^{(n)}} = 0.14 \]

\[ \Rightarrow \alpha_3 = \frac{1}{2} \log \frac{1 - \text{err}_3}{\text{err}_3} = \frac{1}{2} \log \left( \frac{1}{0.14} - 1 \right) \approx 0.91 \Rightarrow H(x) = \text{sign} (\alpha_1 h_1(x) + \alpha_2 h_2(x) + \alpha_3 h_3(x)) \]

\[ \varepsilon_3 = 0.14 \]

\[ \alpha_3 = 0.92 \]

[Slide credit: Verma & Thrun]
AdaBoost Example

- Final classifier

\[ H_{\text{final}} = \text{sign} (0.42 + 0.65 + 0.92) \]

[Slide credit: Verma & Thrun]
AdaBoost Algorithm

\[ H(x) = \text{sign} \left( \sum_{t=1}^{T} \alpha_t h_t(x) \right) \]

\[ w_i \leftarrow w_i \exp \left( 2\alpha_t \mathbb{I}\{h_t(x^{(i)}) \neq t^{(i)}\} \right) \]

\[ \alpha_t = \frac{1}{2} \log \left( \frac{1 - \text{err}_t}{\text{err}_t} \right) \]

\[ \text{err}_t = \frac{\sum_{i=1}^{N} w_i \mathbb{I}\{h_t(x^{(i)}) \neq t^{(i)}\}}{\sum_{i=1}^{N} w_i} \]
AdaBoost Minimizes the Training Error

**Theorem**

Assume that at each iteration of AdaBoost the WeakLearn returns a hypothesis with error $\text{err}_t \leq \frac{1}{2} - \gamma$ for all $t = 1, \ldots, T$ with $\gamma > 0$. The training error of the output hypothesis $H(x) = \text{sign} \left( \sum_{t=1}^{T} \alpha_t h_t(x) \right)$ is at most

$$L_N(H) = \frac{1}{N} \sum_{i=1}^{N} \mathbb{I}\{H(x^{(i)}) \neq t^{(i)}\} \leq \exp \left(-2\gamma^2 T \right).$$

- This is under the simplifying assumption that each weak learner is $\gamma$-better than a random predictor.
- This is called geometric convergence. It is fast!
Generalization Error of AdaBoost

- AdaBoost’s training error (loss) converges to zero. What about the test error of $H$?
- As we add more weak classifiers, the overall classifier $H$ becomes more “complex”.
- We expect more complex classifiers overfit.
- If one runs AdaBoost long enough, it can in fact overfit.

![Graph showing test and train error over rounds](image-url)
Generalization Error of AdaBoost

- But often it does not!
- Sometimes the test error decreases even after the training error is zero!

How does that happen?
- Occam's razor wrongly predicts "simpler" rule is better

Next, we provide an alternative viewpoint on AdaBoost.

Additive Models

Next, we interpret AdaBoost as a way of fitting an additive model.

- Consider a hypothesis class $\mathcal{H}$ with each $h_i : \mathbf{x} \mapsto \{-1, +1\}$ within $\mathcal{H}$, i.e., $h_i \in \mathcal{H}$. These are the “weak learners”, and in this context they’re also called bases.

- An additive model with $m$ terms is given by

$$H_m(x) = \sum_{i=1}^{m} \alpha_i h_i(x),$$

where $(\alpha_1, \cdots, \alpha_m) \in \mathbb{R}^m$ (generally $\alpha_i \geq 0$ and $\sum_i \alpha_i = 1$).

- Observe that we’re taking a linear combination of base classifiers $h_i(x)$, just like in boosting.

- How can we learn it? Two ways to learn additive models:
  1. Learn all $m$ hypotheses $h_i$ and $\alpha_i$ at the same time:

$$\min_{\{h_i \in \mathcal{H}, \alpha_i\}} \sum_{n=1}^{N} \mathcal{L} \left( H_m(x^{(n)}, t^{(n)}) \right) = \sum_{n=1}^{N} \mathcal{L} \left( \sum_{i=1}^{m} \alpha_i h_i(x^{(n)}), t^{(n)} \right).$$

  2. Learn them one by one, i.e., learn $h_{m+1}$ while fixing $h_1, \ldots, h_m$. 
Stagewise Training of Additive Models

A greedy approach to fitting additive models, known as \textbf{stagewise training}:

1. Initialize $H_0(x) = 0$

2. For $m = 1$ to $T$:
   \begin{itemize}
   \item Compute the $m$-th hypothesis $H_m = H_{m-1} + \alpha_m h_m$, i.e. $h_m$ and $\alpha_m$, assuming previous additive model $H_{m-1}$ is fixed:
   \begin{equation}
   (h_m, \alpha_m) \leftarrow \arg \min_{h \in \mathcal{H}, \alpha} \sum_{i=1}^{N} \mathcal{L} \left( H_{m-1}(x^{(i)}) + \alpha h(x^{(i)}), t^{(i)} \right)
   \end{equation}
   \item Add it to the additive model
   \begin{equation}
   H_m = H_{m-1} + \alpha_m h_m
   \end{equation}
   \end{itemize}
Additive Models with Exponential Loss

Consider the exponential loss

$$\mathcal{L}_E(z, t) = \exp(-tz).$$

We want to see how the stagewise training of additive models can be done.
Consider the exponential loss

\[ \mathcal{L}_E(z, t) = \exp(-tz). \]

We want to see how the stagewise training of additive models can be done.

\[
(h_m, \alpha_m) \leftarrow \arg\min_{h \in \mathcal{H}, \alpha} \sum_{i=1}^{N} \exp \left( - \left[ H_{m-1}(x^{(i)}) + \alpha h(x^{(i)}) \right] t^{(i)} \right)
\]

\[
= \sum_{i=1}^{N} \exp \left( -H_{m-1}(x^{(i)}) t^{(i)} - \alpha h(x^{(i)}) t^{(i)} \right)
\]

\[
= \sum_{i=1}^{N} \exp \left( -H_{m-1}(x^{(i)}) t^{(i)} \right) \exp \left( -\alpha h(x^{(i)}) t^{(i)} \right)
\]

\[
= \sum_{i=1}^{N} w_i^{(m)} \exp \left( -\alpha h(x^{(i)}) t^{(i)} \right).
\]

Here we defined \( w_i^{(m)} \triangleq \exp \left( -H_{m-1}(x^{(i)}) t^{(i)} \right) \) (doesn’t depend on \( h, \alpha \)).
We want to solve the following minimization problem:

\[(h_m, \alpha_m) \leftarrow \arg\min_{h \in H, \alpha} \sum_{i=1}^{N} w_i^{(m)} \exp \left( -\alpha h(x^{(i)}) t^{(i)} \right) .\]

- If \( h(x^{(i)}) = t^{(i)} \), we have \( \exp \left( -\alpha h(x^{(i)}) t^{(i)} \right) = \exp(-\alpha) \).
- If \( h(x^{(i)}) \neq t^{(i)} \), we have \( \exp \left( -\alpha h(x^{(i)}) t^{(i)} \right) = \exp(+\alpha) \).

(recall that we are in the binary classification case with \{-1, +1\} output values). We can decompose the summation above into two parts:

\[\sum_{i=1}^{N} w_i^{(m)} \exp \left( -\alpha h(x^{(i)}) t^{(i)} \right) = e^{-\alpha} \sum_{i=1}^{N} w_i^{(m)} \mathbb{I}\{h(x^{(i)}) = t^{(i)}\} + e^{\alpha} \sum_{i=1}^{N} w_i^{(m)} \mathbb{I}\{h(x^{(i)}) \neq t^{(i)}\}\]

- correct predictions
- incorrect predictions
We now add and subtract $e^{-\alpha} \sum_{i=1}^{N} w^{(m)}_i \mathbb{I}\{h(x^{(i)}) \neq t^{(i)}\}$:

$$\sum_{i=1}^{N} w^{(m)}_i \exp \left(-\alpha h(x^{(i)}) t^{(i)}\right) = e^{-\alpha} \sum_{i=1}^{N} w^{(m)}_i \mathbb{I}\{h(x^{(i)}) = t^{(i)}\} + e^{\alpha} \sum_{i=1}^{N} w^{(m)}_i \mathbb{I}\{h(x^{(i)}) \neq t^{(i)}\}$$

**correct predictions**

$$= e^{-\alpha} \sum_{i=1}^{N} w^{(m)}_i \mathbb{I}\{h(x^{(i)}) = t^{(i)}\} + e^{\alpha} \sum_{i=1}^{N} w^{(m)}_i \mathbb{I}\{h(x^{(i)}) \neq t^{(i)}\}$$

**incorrect predictions**

$$- e^{-\alpha} \sum_{i=1}^{N} w^{(m)}_i \mathbb{I}\{h(x^{(i)}) \neq t^{(i)}\} + e^{-\alpha} \sum_{i=1}^{N} w^{(m)}_i \mathbb{I}\{h(x^{(i)}) \neq t^{(i)}\}$$

$$= (e^{\alpha} - e^{-\alpha}) \sum_{i=1}^{N} w^{(m)}_i \mathbb{I}\{h(x^{(i)}) \neq t^{(i)}\} + e^{-\alpha} \sum_{i=1}^{N} w^{(m)}_i \left[ \mathbb{I}\{h(x^{(i)}) \neq t^{(i)}\} + \mathbb{I}\{h(x^{(i)}) = t^{(i)}\} \right] .$$
\[
\sum_{i=1}^{N} w^{(m)}_i \exp \left( -\alpha h(x^{(i)}) t^{(i)} \right) = (e^\alpha - e^{-\alpha}) \sum_{i=1}^{N} w^{(m)}_i \mathbb{I}\{h(x^{(i)} \neq t^{(i)})\} + \\
e^{-\alpha} \sum_{i=1}^{N} w^{(m)}_i \left[ \mathbb{I}\{h(x^{(i)} \neq t^{(i)})\} + \mathbb{I}\{h(x^{(i)}) = t^{(i)}\} \right] \\
= (e^\alpha - e^{-\alpha}) \sum_{i=1}^{N} w^{(m)}_i \mathbb{I}\{h(x^{(i)}) \neq t^{(i)}\} + e^{-\alpha} \sum_{i=1}^{N} w^{(m)}_i.
\]

Let us first optimize \( h \): The second term on the RHS does not depend on \( h \). So we get

\[
h_m \leftarrow \text{argmin}_{h \in \mathcal{H}} \sum_{i=1}^{N} w^{(m)}_i \exp \left( -\alpha h(x^{(i)}) t^{(i)} \right) \equiv \text{argmin}_{h \in \mathcal{H}} \sum_{i=1}^{N} w^{(m)}_i \mathbb{I}\{h(x^{(i)}) \neq t^{(i)}\}.
\]

This means that \( h_m \) is the minimizer of the weighted 0–1-loss.
Now that we obtained $h_m$, we want to find $\alpha$: Define the weighted classification error:

$$\text{err}_m = \frac{\sum_{i=1}^{N} w_i^{(m)} \mathbb{I}\{h_m(x^{(i)}) \neq t^{(i)}\}}{\sum_{i=1}^{N} w_i^{(m)}}$$

With this definition, and $h_m = \text{argmin}_{h \in \mathcal{H}} \sum_{i=1}^{N} w_i^{(m)} \exp(-\alpha h(x^{(i)}) t^{(i)})$

$$\min_{\alpha} \min_{h \in \mathcal{H}} \sum_{i=1}^{N} w_i^{(m)} \exp \left( -\alpha h(x^{(i)}) t^{(i)} \right) =$$

$$\min_{\alpha} \left\{ (e^\alpha - e^{-\alpha}) \sum_{i=1}^{N} w_i^{(m)} \mathbb{I}\{h_m(x^{(i)}) \neq t^{(i)}\} + e^{-\alpha} \sum_{i=1}^{N} w_i^{(m)} \right\}$$

$$= \min_{\alpha} \left\{ (e^\alpha - e^{-\alpha}) \text{err}_m \sum_{i=1}^{N} w_i^{(m)} + e^{-\alpha} \sum_{i=1}^{N} w_i^{(m)} \right\}$$

By taking derivative w.r.t. $\alpha$ and set it to zero, we get

$$e^{2\alpha} = \frac{1 - \text{err}_m}{\text{err}_m} \Rightarrow \alpha = \frac{1}{2} \log \left( \frac{1 - \text{err}_m}{\text{err}_m} \right).$$
Additive Models with Exponential Loss

The updated weights for the next iteration is

\[ w_i^{(m+1)} = \exp \left( -H_m(x^{(i)})t^{(i)} \right) \]
\[ = \exp \left( - \left[ H_{m-1}(x^{(i)}) + \alpha_m h_m(x^{(i)}) \right] t^{(i)} \right) \]
\[ = \exp \left( -H_{m-1}(x^{(i)})t^{(i)} \right) \exp \left( -\alpha_m h_m(x^{(i)})t^{(i)} \right) \]
\[ = w_i^{(m)} \exp \left( -\alpha_m h_m(x^{(i)})t^{(i)} \right) \]
Additive Models with Exponential Loss

To summarize, we obtain the additive model $H_m(x) = \sum_{i=1}^{m} \alpha_i h_i(x)$ with

$$h_m \leftarrow \arg\min_{h \in \mathcal{H}} \sum_{i=1}^{N} w_i^{(m)} \mathbb{I}\{h(x^{(i)}) \neq t^{(i)}\},$$

$$\alpha = \frac{1}{2} \log \left( \frac{1 - \text{err}_m}{\text{err}_m} \right), \quad \text{where } \text{err}_m = \frac{\sum_{i=1}^{N} w_i^{(m)} \mathbb{I}\{h_m(x^{(i)}) \neq t^{(i)}\}}{\sum_{i=1}^{N} w_i^{(m)}},$$

$$w_i^{(m+1)} = w_i^{(m)} \exp\left( -\alpha_m h_m(x^{(i)})t^{(i)} \right).$$

We derived the AdaBoost algorithm!
If AdaBoost is minimizing exponential loss, what does that say about its behavior (compared to, say, logistic regression)?

This interpretation allows boosting to be generalized to lots of other loss functions.
AdaBoost for Face Detection

- Famous application of boosting: detecting faces in images
- Viola and Jones created a very fast face detector that can be scanned across a large image to find the faces.
- A few twists on standard algorithm
  - Change loss function for weak learners: false positives less costly than misses
  - Smart way to do inference in real-time (in 2001 hardware)
AdaBoost for Face Recognition

- The base classifier/weak learner just compares the total intensity in two rectangular pieces of the image and classifies based on comparison of this difference to some threshold.
  - There is a neat trick for computing the total intensity in a rectangle in a few operations.
    - So it is easy to evaluate a huge number of base classifiers and they are very fast at runtime.
  - The algorithm adds classifiers greedily based on their quality on the weighted training cases
    - Each classifier uses just one feature
AdaBoost Face Detection Results
Summary: Boosting

- Boosting reduces bias by generating an ensemble of weak classifiers.
- Each classifier is trained to reduce errors of previous ensemble.
- It is quite resilient to overfitting, though it can overfit.
- Loss minimization viewpoint of AdaBoost allows us to derive other boosting algorithms for regression, ranking, etc.
Ensembles combine classifiers to improve performance

Boosting

- Reduces bias
- Increases variance (large ensemble can cause overfitting)
- Sequential
- High dependency between ensemble elements

Bagging

- Reduces variance (large ensemble can’t cause overfitting)
- Bias is not changed (much)
- Parallel
- Want to minimize correlation between ensemble elements.
Summary: SVM and Boosting

- Support Vector Machine (or Classifier) is formulated as the regularized empirical risk minimizer problem with the choice of hinge loss and the $\ell_2$ regularizer.

- SVM finds the optimal separating hyperplane (separable data) and has a margin maximization property.

- AdaBoost is the solution of stagewise training of an additive model using the exponential loss.