

CSC 311: Introduction to Machine Learning

Lecture 6 - SVMs and Ensembles: Boosting

Richard Zemel & Murat A. Erdogdu

University of Toronto

Binary Classification with a Linear Model

- 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 ?
 - ▶ We know linear regression doesn't do well.

Zero-One Loss

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

$$\begin{aligned}\mathcal{L}_{0-1}(z, t) &= \begin{cases} 0 & \text{if } \text{sign}(z) = t \\ 1 & \text{if } \text{sign}(z) \neq t \end{cases} \\ &= \mathbb{I}\{y \neq t\}.\end{aligned}$$

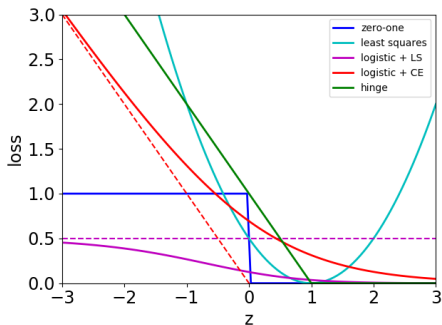
- But minimizing this loss is computationally difficult, and it can't distinguish different hypotheses that achieve the same accuracy.
- We investigated some other loss functions that are easier to minimize, e.g., square loss or logistic regression with the cross-entropy loss \mathcal{L}_{CE} .

A different relaxation to 0-1 loss: Hinge loss

Previously, we saw loss functions as relaxations to 0-1 loss. Now we consider a different relaxation: Hinge loss

$$\text{0-1 loss: } \mathcal{L}_{0-1}(z, t) = \mathbb{I}\{\text{sign}(z) \neq t\}$$

$$\text{Hinge loss: } \mathcal{L}_H(z, t) = \max\{0, 1 - zt\}$$



SVMs: Hinge Loss

If we use a linear classifier and write $z^{(i)}(\mathbf{w}, b) = \mathbf{w}^\top \mathbf{x} + b$, then minimizing the training loss leads to

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

- 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 formulation is called **Support Vector Machines (SVMs)**.
- Generally used with L_2 -regularization.

$$\min_{\mathbf{w}, b} \sum_{i=1}^N \max\{0, 1 - t^{(i)} z^{(i)}(\mathbf{w}, b)\} + \frac{\lambda}{2} \|\mathbf{w}\|_2^2$$

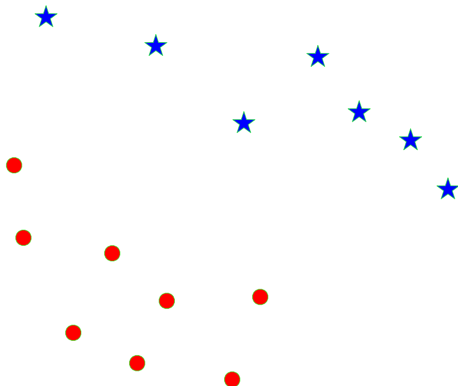
SVMs: How to train?

How to train?

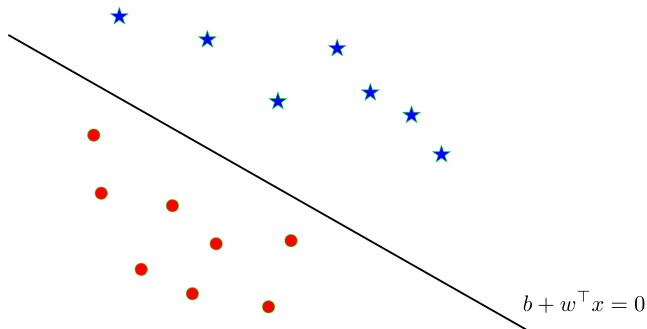
- How to fit \mathbf{w}, b :
 - ▶ One option: gradient descent

Separating Hyperplanes

- There is a much more elegant derivation of support vector machines that we won't 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.

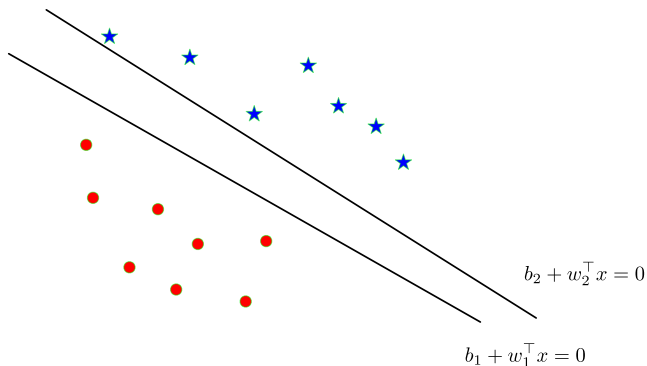


Separating Hyperplanes



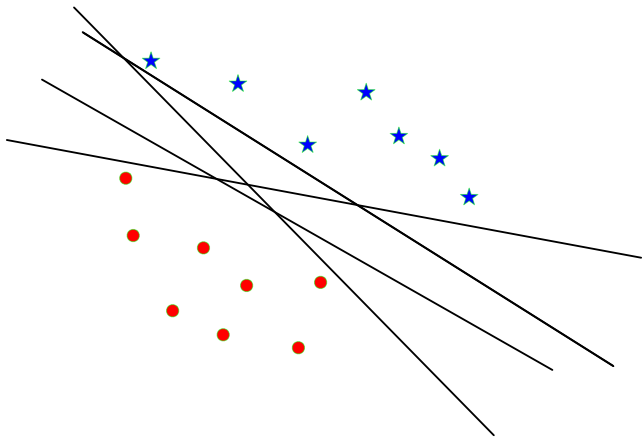
- The decision boundary looks like a line because $\mathbf{x} \in \mathbb{R}^2$, but think about it as a $D - 1$ dimensional hyperplane.
- Recall that a hyperplane is described by points $\mathbf{x} \in \mathbb{R}^D$ such that $f(\mathbf{x}) = \mathbf{w}^T x + b = 0$.

Separating Hyperplanes



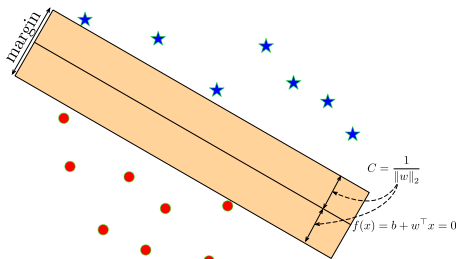
- There are multiple separating hyperplanes, described by different parameters (\mathbf{w}, b) .

Separating Hyperplanes



Optimal Separating Hyperplane

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.
- Support Vector Machines maximizes the above margin! (covered in Csc2515)

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).

Weighted Training set

- 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:

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

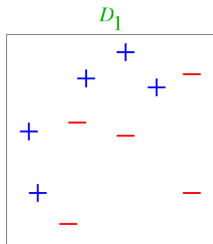
- Usually require each $w^{(n)} > 0$ and $\sum_{n=1}^N w^{(n)} = 1$

Weak Learner/Classifier

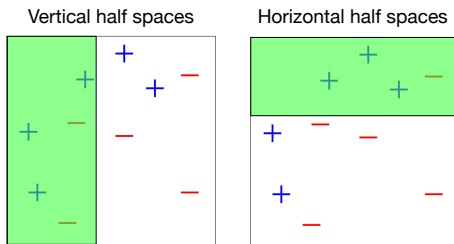
- (Informal) Weak learner is a learning algorithm that outputs a hypothesis (e.g., a classifier) that performs slightly better than chance, e.g., it predicts the correct label with probability 0.51 in binary label case.
 - ▶ It gets slightly less than 0.5 error rate (the worst case is 0.5)
- We are interested in weak learners that are *computationally* efficient.
 - ▶ Decision trees
 - ▶ Even simpler: **Decision Stump**: A decision tree with a single split

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

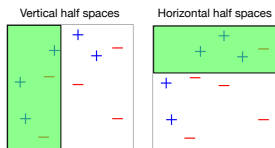
Weak Classifiers



These weak classifiers, which are decision stumps, consist of the set of horizontal and vertical half spaces.



Weak Classifiers



- 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 $\{w^{(1)}, \dots, w^{(N)}\}$ (with $\sum_{n=1}^N w^{(n)} = 1$ and $w^{(n)} \geq 0$)

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

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 $\mathcal{D}_N = \{\mathbf{x}^{(n)}, t^{(n)}\}_{n=1}^N$ 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, otw equivalent.
- A classifier or hypothesis $h : \mathbf{x} \rightarrow \{-1, +1\}$
- 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 \mathcal{D}_N , weak classifier 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 $n = 1, \dots, N$
- For $t = 1, \dots, T$
 - ▶ Fit a classifier to data using weighted samples ($h_t \leftarrow \text{WeakLearn}(\mathcal{D}_N, \mathbf{w})$), e.g.,

$$h_t \leftarrow \operatorname{argmin}_{h \in \mathcal{H}} \sum_{n=1}^N w^{(n)} \mathbb{I}\{h(\mathbf{x}^{(n)}) \neq t^{(n)}\}$$

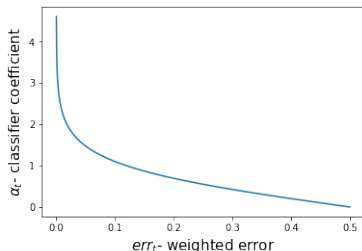
- ▶ Compute weighted error $\text{err}_t = \frac{\sum_{n=1}^N w^{(n)} \mathbb{I}\{h_t(\mathbf{x}^{(n)}) \neq t^{(n)}\}}{\sum_{n=1}^N w^{(n)}}$
- ▶ Compute classifier coefficient $\alpha_t = \frac{1}{2} \log \frac{1 - \text{err}_t}{\text{err}_t} \quad (\in (0, \infty))$
- ▶ Update data weights

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

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

Weighting Intuition

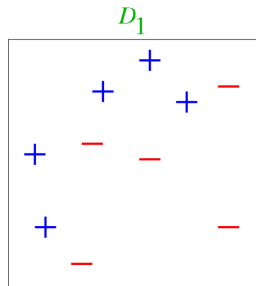
- Recall: $H(\mathbf{x}) = \text{sign} \left(\sum_{t=1}^T \alpha_t h_t(\mathbf{x}) \right)$ where $\alpha_t = \frac{1}{2} \log \frac{1 - \text{err}_t}{\text{err}_t}$



- Weak classifiers which get lower weighted error get more weight in the final classifier
- Also: $w^{(n)} \leftarrow w^{(n)} \exp \left(2\alpha_t \mathbb{I} \{ h_t(\mathbf{x}^{(n)}) \neq t^{(n)} \} \right)$
 - ▶ If $\text{err}_t \approx 0$, α_t high so misclassified examples get more attention
 - ▶ If $\text{err}_t \approx 0.5$, α_t low so misclassified examples are not emphasized

AdaBoost Example

- Training data

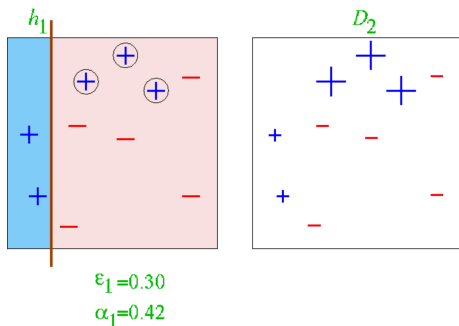


- \mathcal{H} : decision trees with a single split (decision stumps)

[Slide credit: Verma & Thrun]

AdaBoost Example

- Round 1

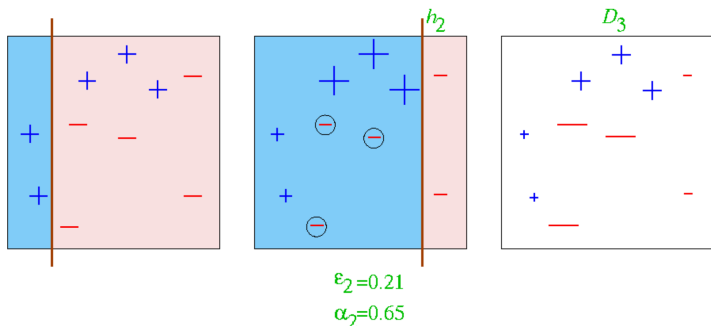


$$\mathbf{w} = \left(\frac{1}{10}, \dots, \frac{1}{10} \right) \Rightarrow \text{Train a classifier (using } \mathbf{w} \text{)} \Rightarrow \text{err}_1 = \frac{\sum_{n=1}^{10} w^{(n)} \mathbb{I}[h_1(\mathbf{x}^{(n)}) \neq t^{(n)}]}{\sum_{n=1}^{10} w^{(n)}} = \frac{3}{10}$$
$$\Rightarrow \alpha_1 = \frac{1}{2} \log \frac{1 - \text{err}_1}{\text{err}_1} = \frac{1}{2} \log \left(\frac{1}{0.3} - 1 \right) \approx 0.42 \Rightarrow H(\mathbf{x}) = \text{sign}(\alpha_1 h_1(\mathbf{x}))$$

[Slide credit: Verma & Thrun]

AdaBoost Example

- Round 2



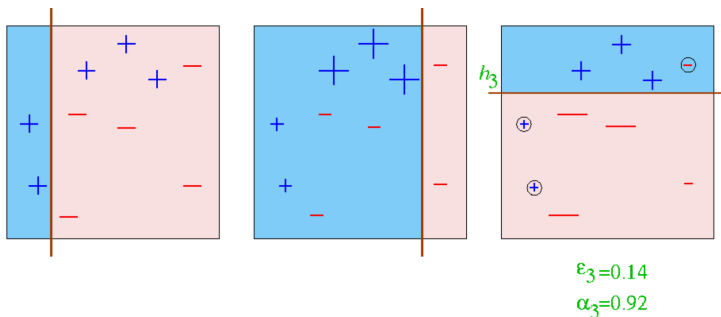
$\mathbf{w} \leftarrow$ new weights \Rightarrow Train a classifier (using \mathbf{w}) $\Rightarrow \text{err}_2 = \frac{\sum_{n=1}^{10} w^{(n)} \mathbb{I}\{h_2(\mathbf{x}^{(n)}) \neq t^{(n)}\}}{\sum_{n=1}^{10} w^{(n)}} = 0.21$

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

[Slide credit: Verma & Thrun]

AdaBoost Example

- Round 3

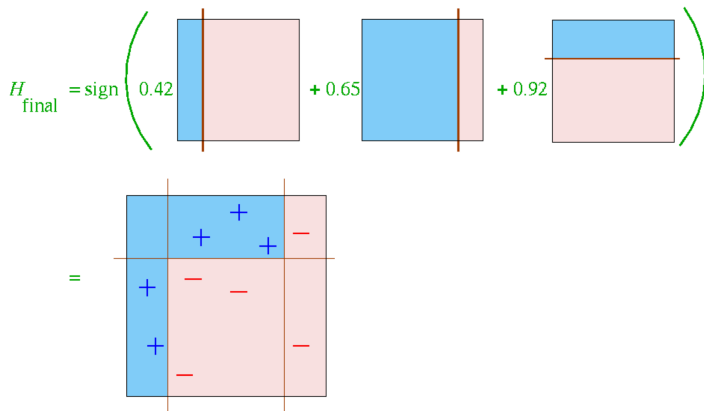


$$\mathbf{w} \leftarrow \text{new weights} \Rightarrow \text{Train a classifier (using } \mathbf{w} \text{)} \Rightarrow \text{err}_3 = \frac{\sum_{n=1}^{10} w^{(n)} \mathbb{I}\{h_3(\mathbf{x}^{(n)}) \neq t^{(n)}\}}{\sum_{i=1}^{10} w^{(i)}} = 0.14$$
$$\Rightarrow \alpha_3 = \frac{1}{2} \log \frac{1 - \text{err}_3}{\text{err}_3} = \frac{1}{2} \log \left(\frac{1}{0.14} - 1 \right) \approx 0.91 \Rightarrow H(\mathbf{x}) = \text{sign}(\alpha_1 h_1(\mathbf{x}) + \alpha_2 h_2(\mathbf{x}) + \alpha_3 h_3(\mathbf{x}))$$

[Slide credit: Verma & Thrun]

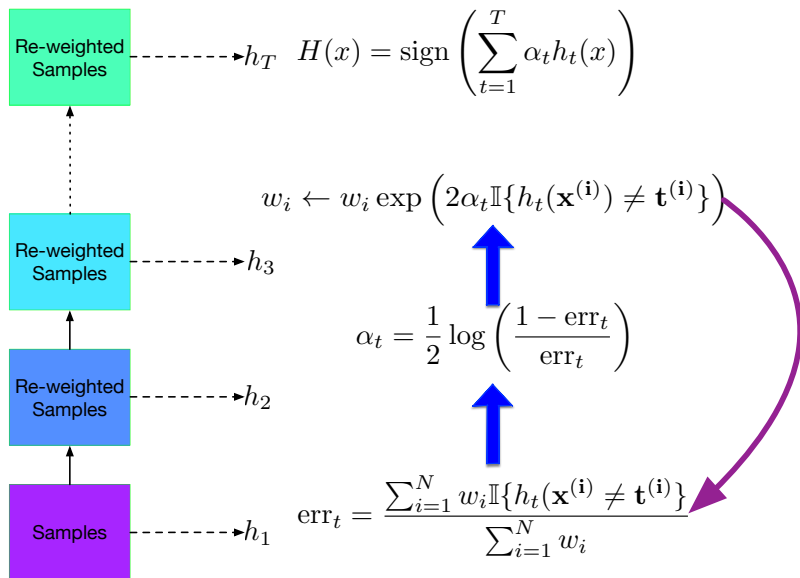
AdaBoost Example

- Final classifier



[Slide credit: Verma & Thrun]

AdaBoost Algorithm



AdaBoost Minimizes the Training Error

Theorem

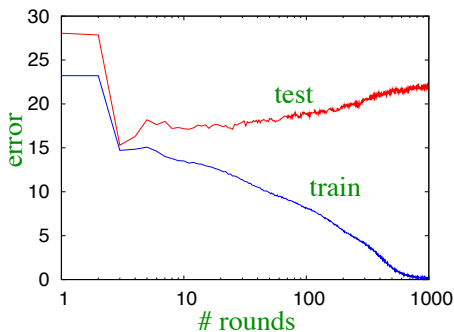
Assume that at each iteration of AdaBoost the WeakLearn returns a hypothesis with error $\text{err}_t \leq \frac{1}{2} - \gamma$ for all $t = 1, \dots, T$ with $\gamma > 0$. The training error of the output hypothesis $H(\mathbf{x}) = \text{sign} \left(\sum_{t=1}^T \alpha_t h_t(\mathbf{x}) \right)$ is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

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

- This is under the simplifying assumption that each weak learner is γ -better than a random predictor.
- This is called geometric convergence. It is fast!

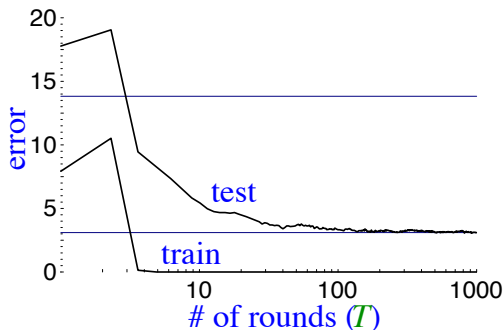
Generalization Error of AdaBoost

- 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.



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?
- Next, we provide an alternative viewpoint on AdaBoost.

[Slide credit: Robert Shapire's Slides,
<http://www.cs.princeton.edu/courses/archive/spring12/cos598A/schedule.html>]

Additive Models

Next, we'll now 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

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

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

- Observe that we're taking a linear combination of base classifiers $h_i(\mathbf{x})$, just like in boosting.

Stagewise Training of Additive Models

A greedy approach to fitting additive models, known as **stagewise training**:

1. Initialize $H_0(x) = 0$
2. For $m = 1$ to T :
 - ▶ Compute the m -th hypothesis $H_m = H_{m-1} + \alpha_m h_m$, i.e. h_m and α_m , assuming previous additive model H_{m-1} is fixed:

$$(h_m, \alpha_m) \leftarrow \operatorname{argmin}_{h \in \mathcal{H}, \alpha} \sum_{i=1}^N \mathcal{L} \left(H_{m-1}(\mathbf{x}^{(i)}) + \alpha h(\mathbf{x}^{(i)}), t^{(i)} \right)$$

- ▶ Add it to the additive model

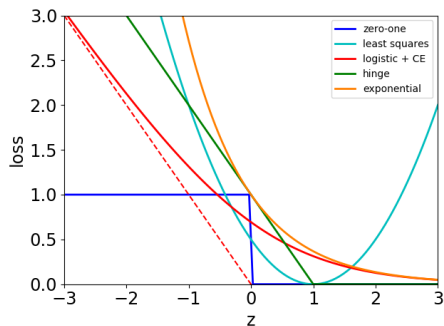
$$H_m = H_{m-1} + \alpha_m h_m$$

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.



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.

$$\begin{aligned}(h_m, \alpha_m) &\leftarrow \operatorname{argmin}_{h \in \mathcal{H}, \alpha} \sum_{i=1}^N \exp\left(-\left[H_{m-1}(\mathbf{x}^{(i)}) + \alpha h(\mathbf{x}^{(i)})\right] t^{(i)}\right) \\ &= \sum_{i=1}^N \exp\left(-H_{m-1}(\mathbf{x}^{(i)}) t^{(i)} - \alpha h(\mathbf{x}^{(i)}) t^{(i)}\right) \\ &= \sum_{i=1}^N \exp\left(-H_{m-1}(\mathbf{x}^{(i)}) t^{(i)}\right) \exp\left(-\alpha h(\mathbf{x}^{(i)}) t^{(i)}\right) \\ &= \sum_{i=1}^N w_i^{(m)} \exp\left(-\alpha h(\mathbf{x}^{(i)}) t^{(i)}\right).\end{aligned}$$

Here we defined $w_i^{(m)} \triangleq \exp\left(-H_{m-1}(\mathbf{x}^{(i)}) t^{(i)}\right)$ (doesn't depend on h, α).

Additive Models with Exponential Loss

We want to solve the following minimization problem:

$$(h_m, \alpha_m) \leftarrow \operatorname{argmin}_{h \in \mathcal{H}, \alpha} \sum_{i=1}^N w_i^{(m)} \exp\left(-\alpha h(\mathbf{x}^{(i)})t^{(i)}\right).$$

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

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

$$\sum_{i=1}^N w_i^{(m)} \exp\left(-\alpha h(\mathbf{x}^{(i)})t^{(i)}\right) = e^{-\alpha} \underbrace{\sum_{i=1}^N w_i^{(m)} \mathbb{I}\{h(\mathbf{x}^{(i)}) = t^{(i)}\}}_{\text{correct predictions}} + e^{\alpha} \underbrace{\sum_{i=1}^N w_i^{(m)} \mathbb{I}\{h(\mathbf{x}^{(i)}) \neq t^{(i)}\}}_{\text{incorrect predictions}}$$

Additive Models with Exponential Loss

We can divide the summation to two parts:

$$\begin{aligned}\sum_{i=1}^N w_i^{(m)} \exp\left(-\alpha h(\mathbf{x}^{(i)})t^{(i)}\right) &= \underbrace{e^{-\alpha} \sum_{i=1}^N w_i^{(m)} \mathbb{I}\{h(\mathbf{x}^{(i)}) = t^{(i)}\}}_{\text{correct predictions}} + \underbrace{e^{\alpha} \sum_{i=1}^N w_i^{(m)} \mathbb{I}\{h(\mathbf{x}^{(i)}) \neq t^{(i)}\}}_{\text{incorrect predictions}} \\ &= e^{-\alpha} \sum_{i=1}^N w_i^{(m)} \mathbb{I}\{h(\mathbf{x}^{(i)}) = t^{(i)}\} + e^{\alpha} \sum_{i=1}^N w_i^{(m)} \mathbb{I}\{h(\mathbf{x}^{(i)}) \neq t^{(i)}\} \\ &\quad - e^{-\alpha} \sum_{i=1}^N w_i^{(m)} \mathbb{I}\{h(\mathbf{x}^{(i)}) \neq t^{(i)}\} + e^{-\alpha} \sum_{i=1}^N w_i^{(m)} \mathbb{I}\{h(\mathbf{x}^{(i)}) \neq t^{(i)}\} \\ &= (e^{\alpha} - e^{-\alpha}) \sum_{i=1}^N w_i^{(m)} \mathbb{I}\{h(\mathbf{x}^{(i)}) \neq t^{(i)}\} + \\ &\quad e^{-\alpha} \sum_{i=1}^N w_i^{(m)} \underbrace{\left[\mathbb{I}\{h(\mathbf{x}^{(i)}) \neq t^{(i)}\} + \mathbb{I}\{h(\mathbf{x}^{(i)}) = t^{(i)}\} \right]}_{=1}\end{aligned}$$

Additive Models with Exponential Loss

$$\begin{aligned}\sum_{i=1}^N w_i^{(m)} \exp\left(-\alpha h(\mathbf{x}^{(i)})t^{(i)}\right) &= (e^\alpha - e^{-\alpha}) \sum_{i=1}^N w_i^{(m)} \mathbb{I}\{h(\mathbf{x}^{(i)}) \neq t^{(i)}\} + \\ &\quad e^{-\alpha} \sum_{i=1}^N w_i^{(m)} \left[\mathbb{I}\{h(\mathbf{x}^{(i)}) \neq t^{(i)}\} + \mathbb{I}\{h(\mathbf{x}^{(i)}) = t^{(i)}\} \right] \\ &= (e^\alpha - e^{-\alpha}) \sum_{i=1}^N w_i^{(m)} \mathbb{I}\{h(\mathbf{x}^{(i)}) \neq t^{(i)}\} + e^{-\alpha} \sum_{i=1}^N w_i^{(m)}.\end{aligned}$$

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

$$h_m \leftarrow \operatorname{argmin}_{h \in \mathcal{H}} \sum_{i=1}^N w_i^{(m)} \exp\left(-\alpha h(\mathbf{x}^{(i)})t^{(i)}\right) \equiv \operatorname{argmin}_{h \in \mathcal{H}} \sum_{i=1}^N w_i^{(m)} \mathbb{I}\{h(\mathbf{x}^{(i)}) \neq t^{(i)}\}.$$

This means that h_m is the minimizer of the weighted 0/1-loss.

Now that we obtained h_m , we want to find α : Define the weighted classification error:

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

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

$$\begin{aligned} & \min_{\alpha} \min_{h \in \mathcal{H}} \sum_{i=1}^N w_i^{(m)} \exp(-\alpha h(\mathbf{x}^{(i)})t^{(i)}) = \\ & \min_{\alpha} \left\{ (e^{\alpha} - e^{-\alpha}) \sum_{i=1}^N w_i^{(m)} \mathbb{I}\{h_m(\mathbf{x}^{(i)}) \neq t^{(i)}\} + e^{-\alpha} \sum_{i=1}^N w_i^{(m)} \right\} \\ & = \min_{\alpha} \left\{ (e^{\alpha} - e^{-\alpha}) \text{err}_m \sum_{i=1}^N w_i^{(m)} + e^{-\alpha} \sum_{i=1}^N w_i^{(m)} \right\} \end{aligned}$$

Take derivative w.r.t. α and set it to zero. We get that

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

Additive Models with Exponential Loss

The updated weights for the next iteration is

$$\begin{aligned}w_i^{(m+1)} &= \exp\left(-H_m(\mathbf{x}^{(i)})t^{(i)}\right) \\&= \exp\left(-\left[H_{m-1}(\mathbf{x}^{(i)}) + \alpha_m h_m(\mathbf{x}^{(i)})\right]t^{(i)}\right) \\&= \exp\left(-H_{m-1}(\mathbf{x}^{(i)})t^{(i)}\right) \exp\left(-\alpha_m h_m(\mathbf{x}^{(i)})t^{(i)}\right) \\&= w_i^{(m)} \exp\left(-\alpha_m h_m(\mathbf{x}^{(i)})t^{(i)}\right)\end{aligned}$$

Additive Models with Exponential Loss

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

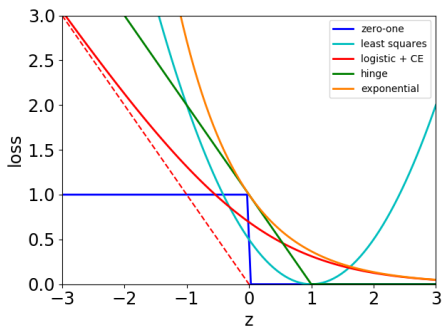
$$h_m \leftarrow \operatorname{argmin}_{h \in \mathcal{H}} \sum_{i=1}^N w_i^{(m)} \mathbb{I}\{h(\mathbf{x}^{(i)}) \neq t^{(i)}\},$$

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

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

We derived the AdaBoost algorithm!

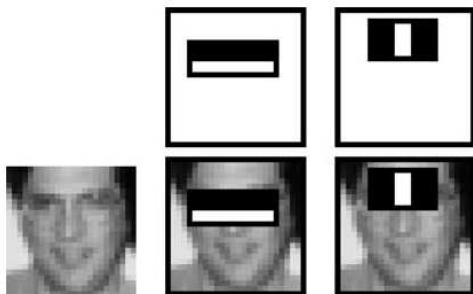
Revisiting Loss Functions for Classification



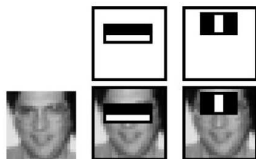
- If AdaBoost is minimizing exponential loss, what does that say about its behavior (compared to, say, logistic regression)?
- This interpretation allows boosting to be generalized to lots of other loss functions.

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)

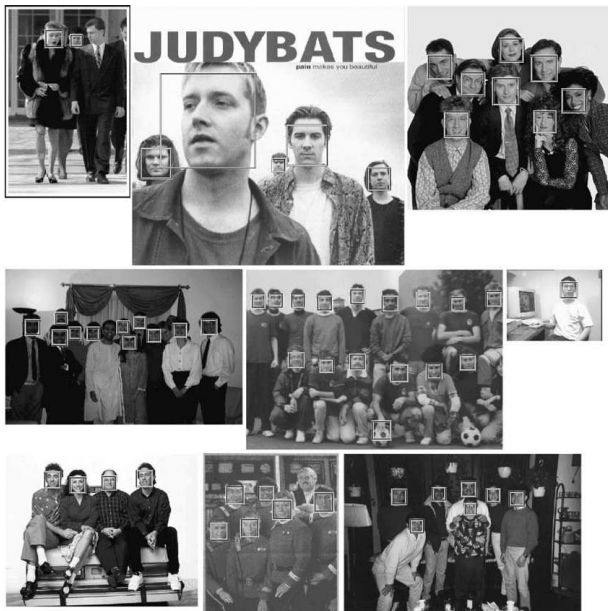


AdaBoost for Face Recognition



- 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.

Summary: Ensembles

- 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.