18-661 Introduction to Machine Learning

Ensemble Methods

Spring 2025

ECE – Carnegie Mellon University

Announcements

- Midsemester grades will be released later this week. These are a rough estimate of your final grades, based on your scores until now.
- This Friday's recitation will go over review materials for HW3

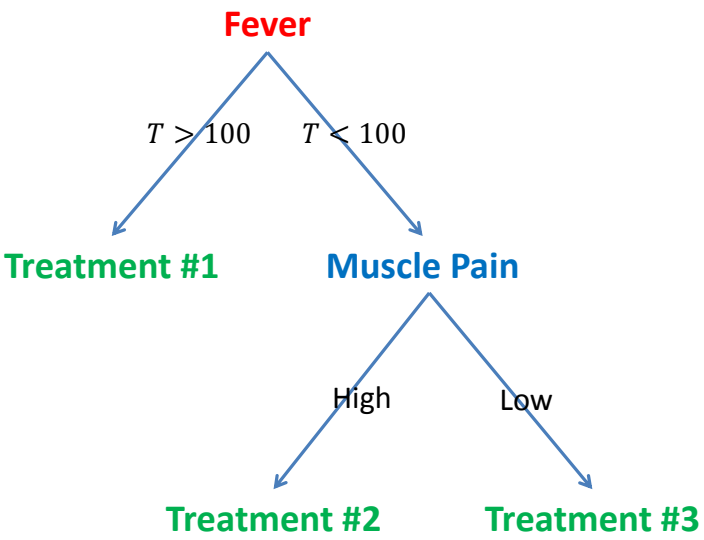
Outline

1. Recap: Decision Trees
2. Ensemble Methods: Motivation
3. Bagging and Random Forests
4. Boosting and AdaBoost

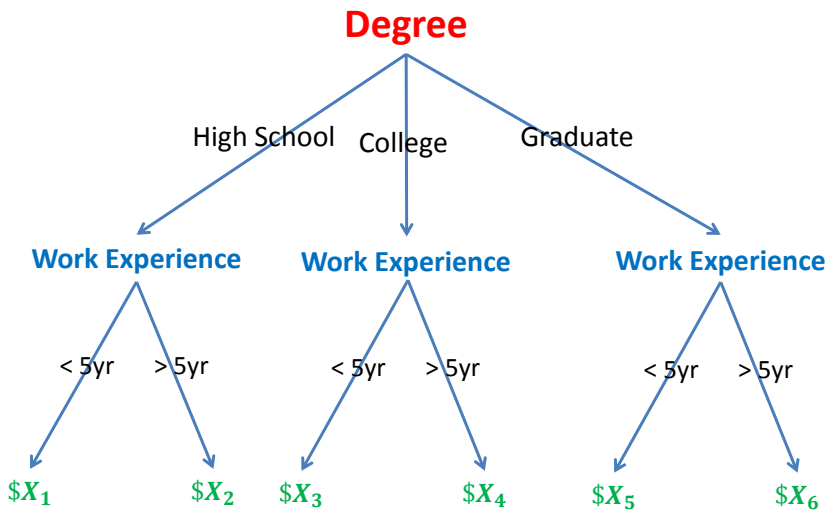
Recap: Decision Trees

Interpretable, Tree-Structured Decision Rules

Medical treatment



Salary in a company



Learning a Tree Model

1. The structure of the tree.
   - Keep splitting until no more nodes can be split.
   - Here, a node cannot be split if the samples are (almost) the same label, OR maximum depth reached.
2. The split rule at each node for *binary tree* setting.
   - Pick the feature with highest information gain.
   - *Numerical feature:* maximize information gain over $n - 1$ possible thresholds. This process will determine both the information gain for this feature, and the associated threshold.
   - *Categorical feature:* Arrange the categories in an "order", and similar procedures apply.
3. The prediction for the leaves.
   - Obvious if all training samples in the leaf has the same label.
   - Otherwise? Majority vote based on the samples in the leaf.

Formalizing the Information Gain

Definition (Entropy)
If a random variable Y takes K values, a_1, $a_2 ... a_K$, its entropy is

$$H[Y] = -\sum_{i=1}^{K} \Pr(Y = a_i) \log \Pr(Y = a_i)$$

Definition (Conditional Entropy)
Given two random variables X and Y

$$H[Y|X] = \sum_{k} P(X = a_k) H[Y|X = a_k]$$

Definition (Information Gain)

$$I(X; Y) = H[Y] - H[Y|X]$$

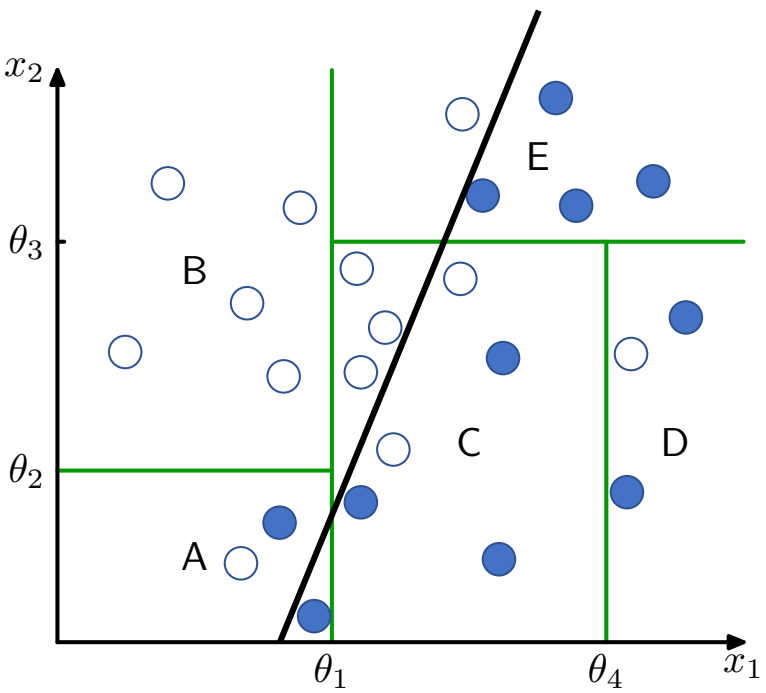
Measures the reduction in entropy (i.e., the reduction of uncertainty in Y) when we also consider X.

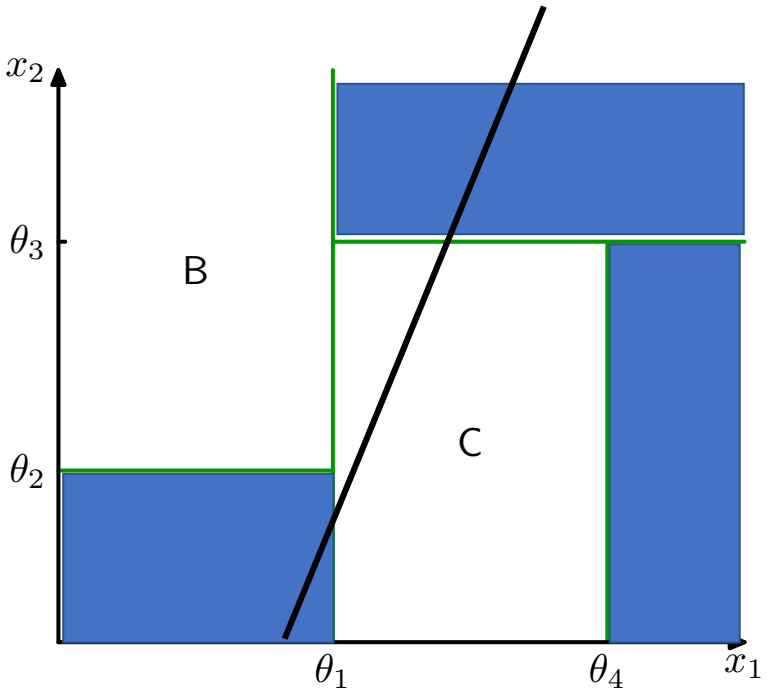
Summary: Advantages of Decision Trees

- Can be interpreted by humans (as long as the tree is not too big)
- Computationally efficient (for shallow trees)
- Handles both numerical and categorical features.
- Can be used for both classification and regression
- Like Nearest Neighbors, decision trees are nonparametric because we do not try to learn a fixed parameter vector. The number of parameters depends on the training data.
- Unlike Nearest Neighbors we don't need training data when making predictions.

Summary: Disadvantages of Decision Trees

- Binary decision trees find it hard to learn linear boundaries.
- Decision trees are prone to overfitting!





Outline

1. Recap: Decision Trees
2. Ensemble Methods: Motivation
3. Bagging and Random Forests
4. Boosting and AdaBoost

Ensemble Methods: Motivation

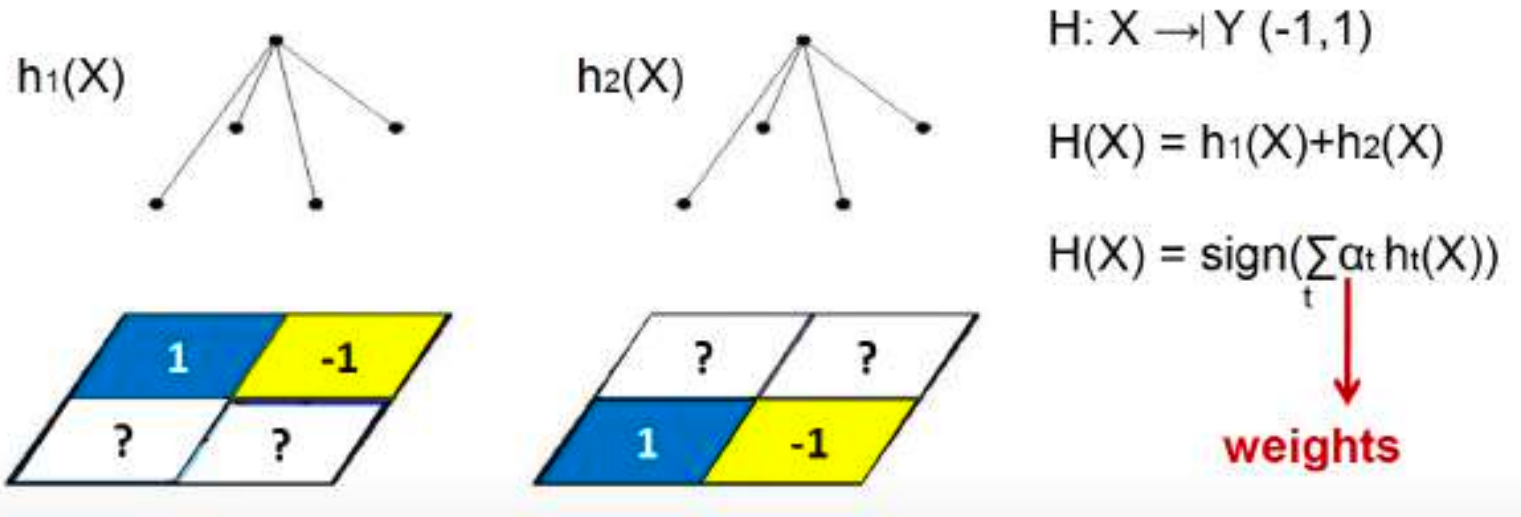
Motivation: Fighting the Bias-Variance Tradeoff

- Shallow trees have low variance (don't overfit) but high bias (underfit).
- Can we combine several shallow trees to reduce bias (without sacrificing variance)?
- Deep trees have low bias, but high variance.
- Can we combine several deep trees to reduce variance (without sacrificing bias)?

Ensemble Methods

Can we combine several shallow trees to reduce bias (without sacrificing variance?)

- A shallow tree is a "weak" classifier. We can learn many weak classifiers, preferably those that are good at different parts of the input spaces
- Predicted Class: (Weighted) Average or Majority of output of the weak classifiers. Strength in Diversity!



Ensemble Methods

In this lecture, we will cover the following ensemble methods:

Ensembles that combine fully grown trees to reduce variance (without sacrificing much bias):

- Bagging or Bootstrap Aggregation
- Random Forests

Ensembles that combine shallow trees to reduce bias (without sacrificing much variance):

- AdaBoost

Bagging and Random Forests

Bagging or Bootstrap Aggregating

To avoid overfitting a decision tree, we can train an ensemble of trees each with some "random variation" in the training process.

Bagging: train each tree on a random subset of the training data.

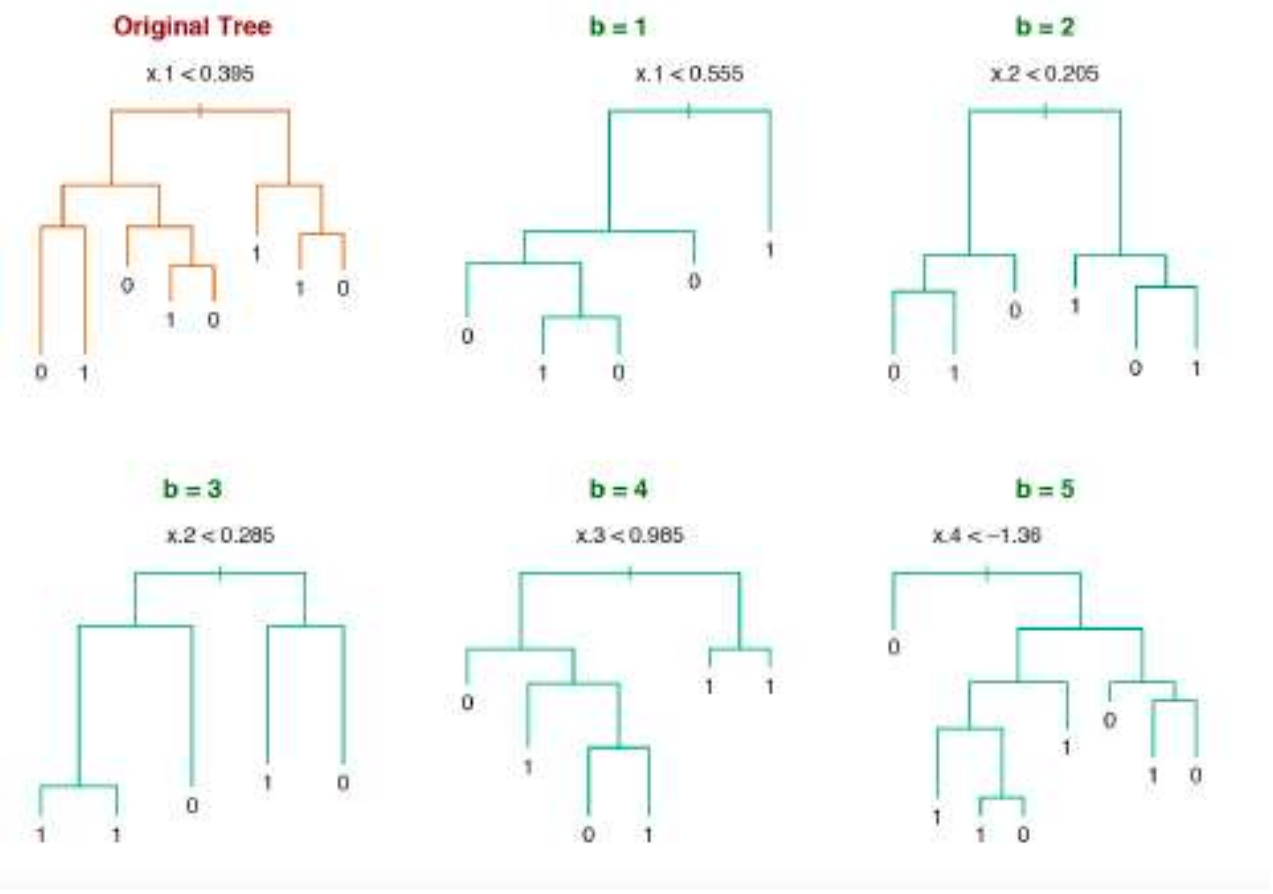
Bagging Trees (Training Phase)

- For $b = 1, 2, \cdots, B$
  - Choose n training samples $(\mathbf{x}_i, y_i)$ from $\mathcal{D}$ uniformly at random with replacement
  - Learn a decision tree h_b on these n samples
- Store the B decision trees $h_1, h_2, \ldots h_B$
- Optimal B (typically in 1000s) chosen using cross-validation

Bagging Trees (Test Phase)

- For a test unlabeled example $\mathbf{x}$
- Find the decision from each of the B trees
- Assign the majority (or most popular) label as the label for $\mathbf{x}$

Bagging: Example

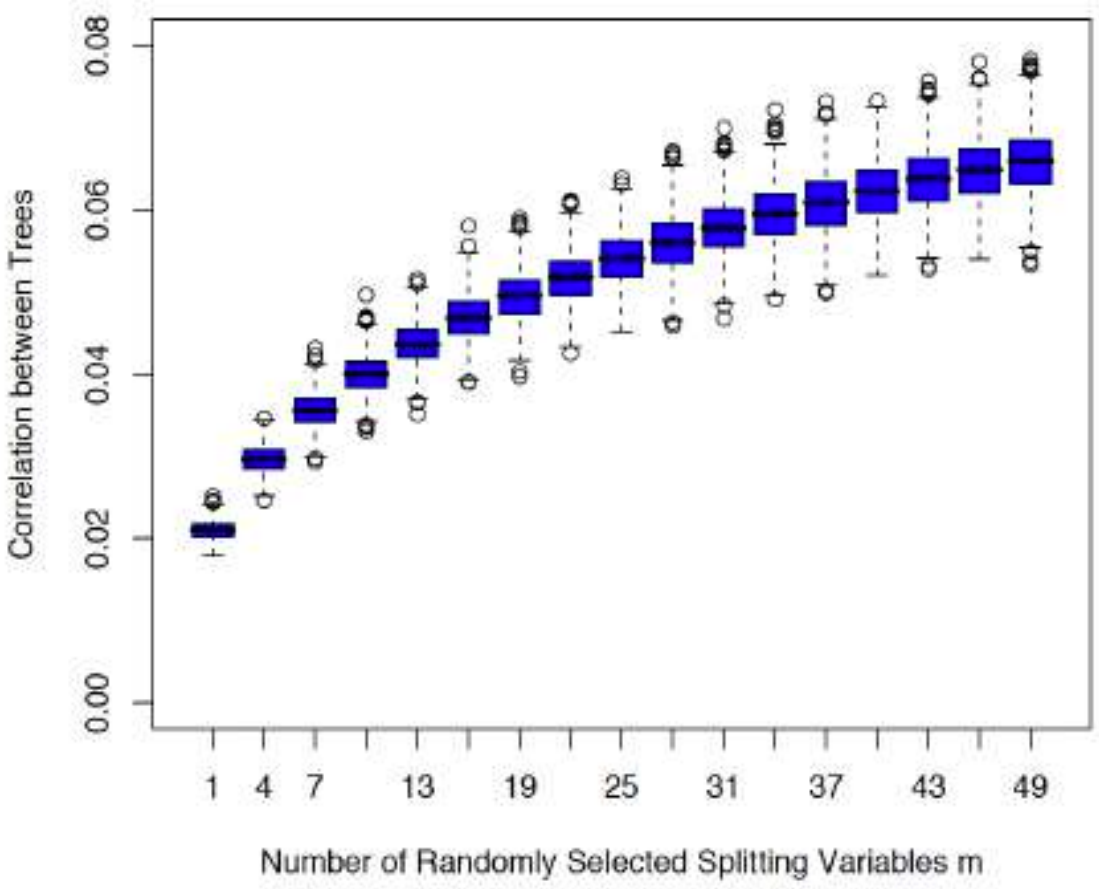


- We get different splits and thresholds for different trees b
- Predict the label assigned by majority of the B trees
- Reduces variance (thus avoiding overfitting) without increasing bias.

Random Forests

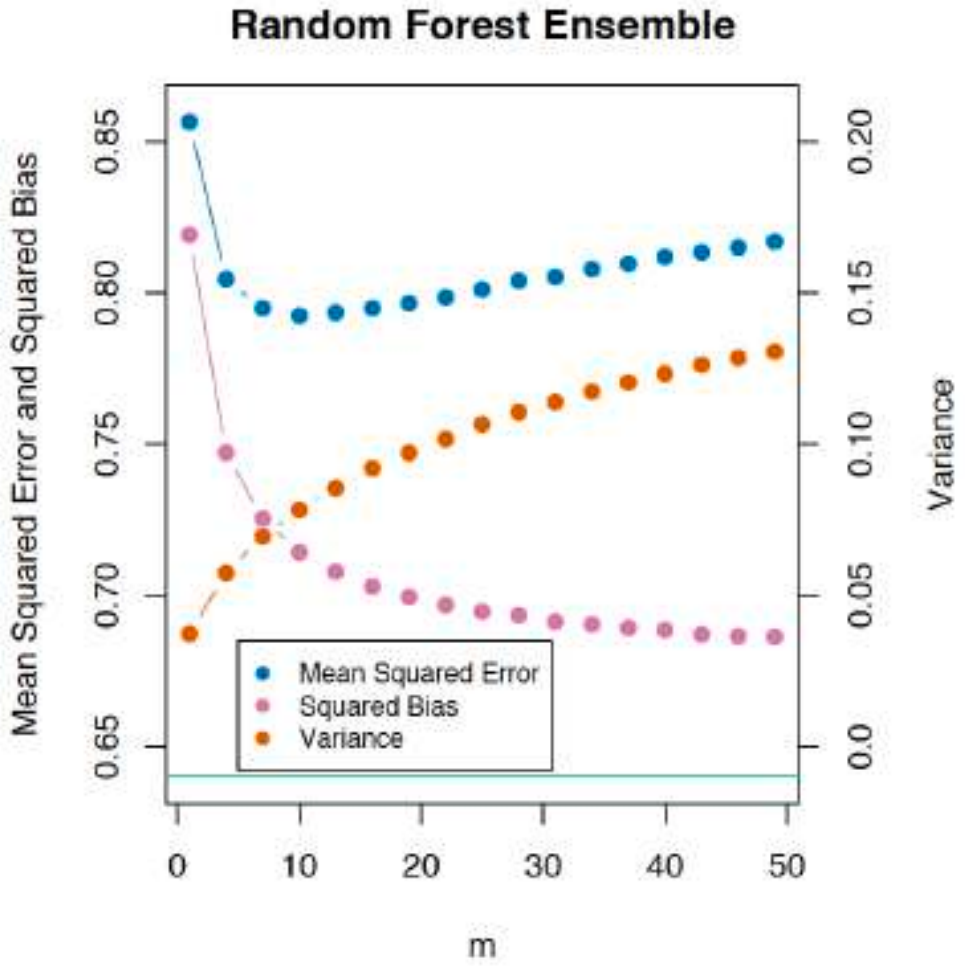
- Limitation of Bagging: If one or more features are very informative, they will be selected first by every tree in the bag, which increases the correlation between the trees (thus harms our goal of creating "random variation" in the training process to reduce variance).
- Key Idea behind Random Forests: Reduces correlation between trees.
  - Same as bagging in terms of sampling training data
  - When constructing each tree, before each split, select $m \leq d$ features at random. Then, select one feature from the m features to split. Note: for the original decision tree, one selects the best feature from all features; now, we only select the best from the m features.
  - Typically, $m \sim \sqrt{d}$.
  - Same as bagging in prediction: Take majority vote of B such trees

Random Forests



Reducing m, the number of splitting candidates, decreases the correlation among the trees in the bag, which helps with reducing the variance.

Random Forests



While reducing m decreases the variance, it will increase the bias!

Outline

1. Recap: Decision Trees
2. Ensemble Methods: Motivation
3. Bagging and Random Forests
4. Boosting and AdaBoost

Boosting and AdaBoost

Random Forests vs Boosting

Bagging and Random Forests: Reduce variance of fully grown trees.

- Key idea: train multiple trees (with some "random variations") and classify by majority vote.
- Bagging: train multiple trees on random subsets of the training data.
- Random forests: Add more "random variations" by restricting the candidates features at each split, thus reducing the correlation between the trees.

Boosting: Reduce bias of shallow trees (weak classifiers).

Boosting

Boosting: Reduce bias of shallow trees.

- Sequentially construct these *weak* classifiers, $h_t(\cdot)$, one at a time
- Use *weak* classifiers to arrive at a complex decision boundary (*strong* classifier), where β_t is the contribution of each weak classifier

$$h[\boldsymbol{x}] = \text{sign}\left[\sum_{t=1}^{T} \beta_t h_t(\boldsymbol{x})\right]$$

Our plan:

- Describe AdaBoost algorithm
- Understand why it works

Adaboost Algorithm

- Given: N samples $\{\boldsymbol{x}_n, y_n\}$, where $y_n \in \{+1, -1\}$, and some way of constructing weak (or base) classifiers
- Initialize weights $w_1(n) = \frac{1}{N}$ for every training sample n
- For $t = 1$ to T
  1. Train a weak classifier $h_t(\boldsymbol{x})$ using weights $w_t(n)$, by minimizing

$$\epsilon_t = \sum_n w_t(n) \mathbb{I}[y_n \neq h_t(\boldsymbol{x}_n)] \quad \text{(the weighted classification error)}$$

  2. Compute contribution for this classifier: $\beta_t = \frac{1}{2} \log \frac{1-\epsilon_t}{\epsilon_t}$
     (smaller $\epsilon_t \Rightarrow$ larger β_t)
  3. Update weights on each training sample n

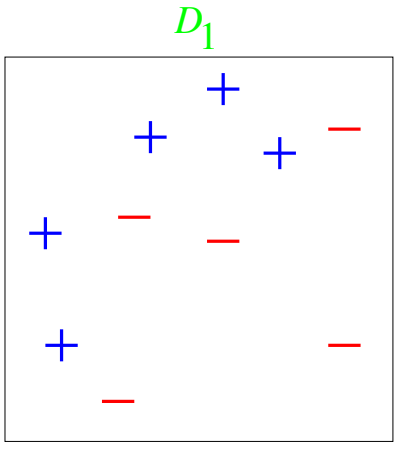
$$w_{t+1}(n) \propto w_t(n) e^{-\beta_t y_n h_t(\boldsymbol{x}_n)}$$

     ($w_t(n)$ decreased if $y_n = h_t(\boldsymbol{x}_n)$; increased if $y_n \neq h_t(\boldsymbol{x}_n)$)
     and normalize them such that $\sum_n w_{t+1}(n) = 1$.
- Output the final classifier

$$h[\boldsymbol{x}] = \text{sign}\left[\sum_{t=1}^{T} \beta_t h_t(\boldsymbol{x})\right]$$

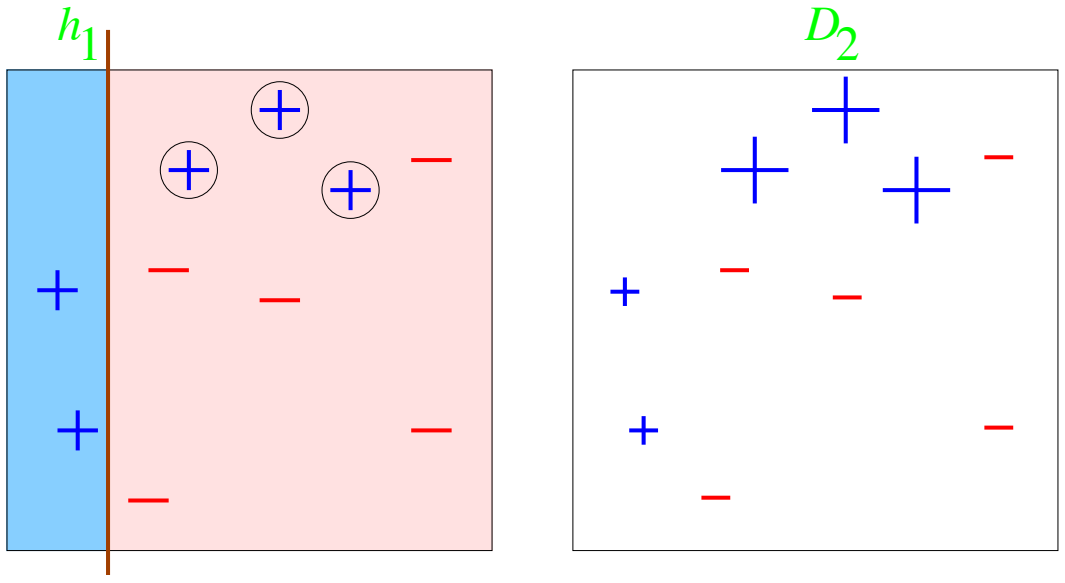
Example

10 data points and 2 features



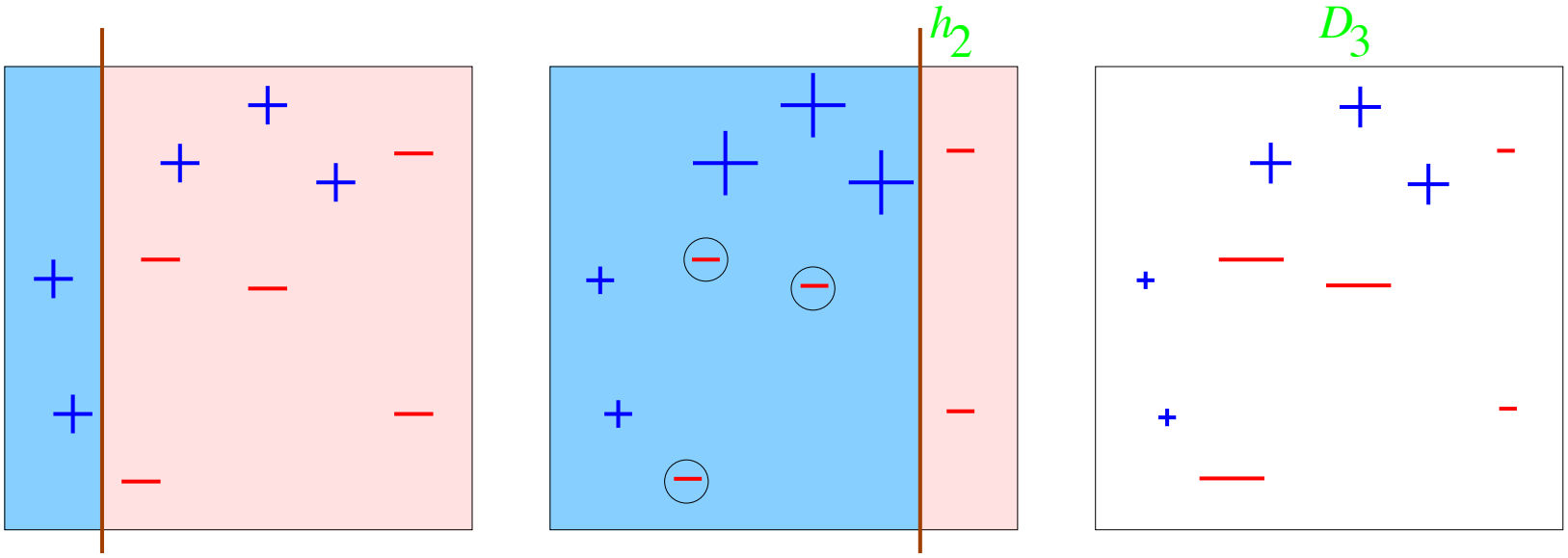
- The data points are clearly not linearly separable
- In the beginning, all data points have equal weights (the size of the data markers "+" or "-")
- Base classifier $h(\cdot)$: horizontal or vertical lines ('decision stumps')
  - Depth-1 decision trees, i.e., classify data based on a single attribute.

Round 1: $t = 1$



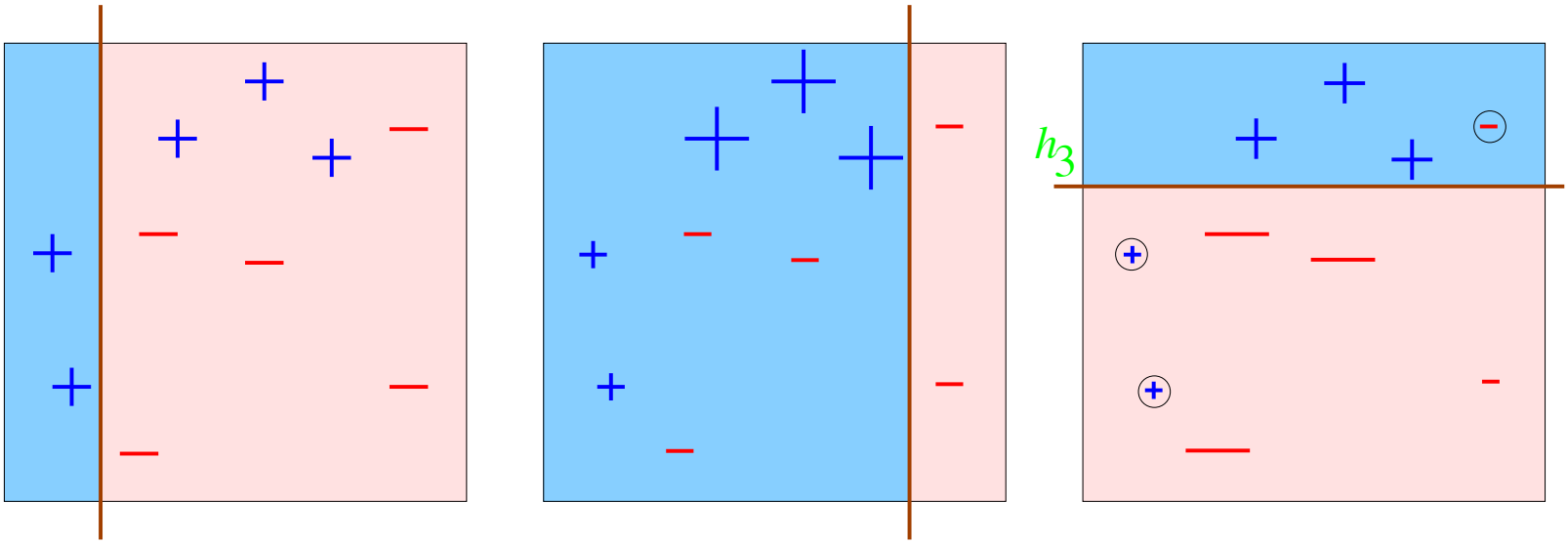
- Equal weights $w_1(n) = 1/10 = 0.1$
- 3 misclassified (with circles): $\epsilon_1 = 0.3 \rightarrow \beta_1 = \frac{1}{2} \log \frac{1-\epsilon_1}{\epsilon_1} = 0.42$.
- Recompute the weights; the 3 misclassified data points receive larger weights

Round 2: $t = 2$



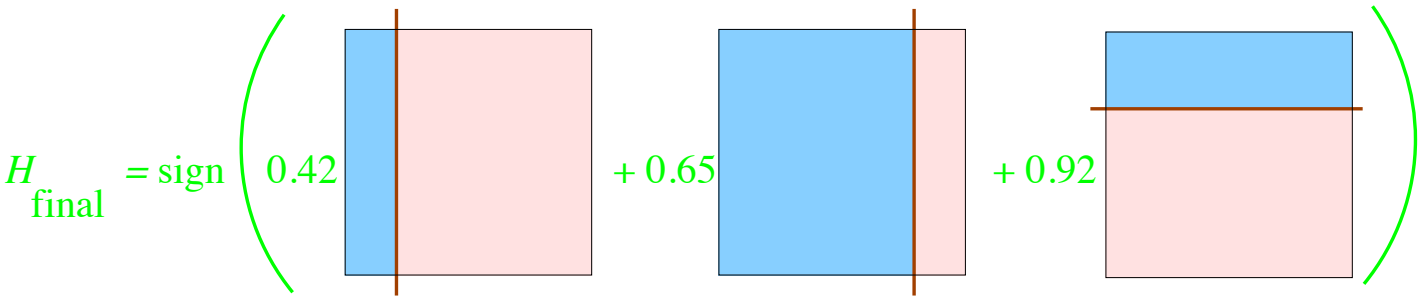
- 3 misclassified (with circles): $\epsilon_2 = 0.21 \rightarrow \beta_2 = 0.65$.
  $\epsilon_2 < 0.3$ as those 3 misclassified data points have weights < 0.1
- 3 newly misclassified data points get larger weights
- Data points classified correctly in both rounds have small weights

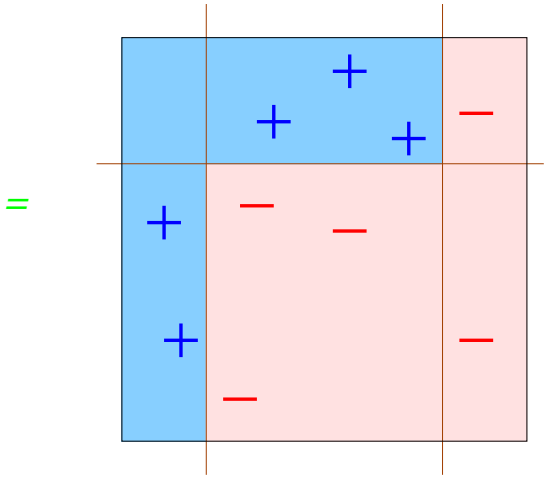
Round 3: $t = 3$



- 3 misclassified (with circles): $\epsilon_3 = 0.14 \rightarrow \beta_3 = 0.92$.
- Previously correctly classified data points are now misclassified, but overall our error is low. Why?
  - Since they have been consistently classified correctly, this round's mistake will not have a huge impact on the overall prediction

Final Classifier: Combining 3 Classifiers





- All data points are now classified correctly!

Adaboost Algorithm

- Given: N samples $\{\boldsymbol{x}_n, y_n\}$, where $y_n \in \{+1, -1\}$, and some way of constructing weak (or base) classifiers
- Initialize weights $w_1(n) = \frac{1}{N}$ for every training sample n
- For $t = 1$ to T
  1. Train a weak classifier $h_t(\boldsymbol{x})$ using weights $w_t(n)$, by minimizing

     $$\epsilon_t = \sum_n w_t(n) \mathbb{I}[y_n \neq h_t(\boldsymbol{x}_n)]$$ (the weighted classification error; *Why?*)

  2. Compute contribution for this classifier: $\beta_t = \frac{1}{2} \log \frac{1-\epsilon_t}{\epsilon_t}$; *Why?* (smaller $\epsilon_t \Rightarrow$ larger β_t)
  3. Update weights on each training sample n; *Why?*

     $$w_{t+1}(n) \propto w_t(n) e^{-\beta_t y_n h_t(\boldsymbol{x}_n)}$$

     ($w_t(n)$ decreased if $y_n = h_t(\boldsymbol{x}_n)$; increased if $y_n \neq h_t(\boldsymbol{x}_n)$) and normalize them such that $\sum_n w_{t+1}(n) = 1$.
- Output the final classifier

$$h[\boldsymbol{x}] = \text{sign}\left[\sum_{t=1}^{T} \beta_t h_t(\boldsymbol{x})\right]$$

Why Does AdaBoost Work?

It iteratively minimizes a loss function related to classification error.

Classification loss

- Suppose we want to have a classifier

$$h(\boldsymbol{x}) = \text{sign}[f(\boldsymbol{x})] = \begin{cases} 1 & \text{if } f(\boldsymbol{x}) > 0 \\ -1 & \text{if } f(\boldsymbol{x}) < 0 \end{cases}$$

- One seemingly natural loss function is 0-1 loss:

$$\ell(h(\boldsymbol{x}), y) = \begin{cases} 0 & \text{if } yf(\boldsymbol{x}) > 0 \\ 1 & \text{if } yf(\boldsymbol{x}) < 0 \end{cases}$$

  Namely, the function $f(\boldsymbol{x})$ and the target label y should have the same sign to avoid a loss of 1.

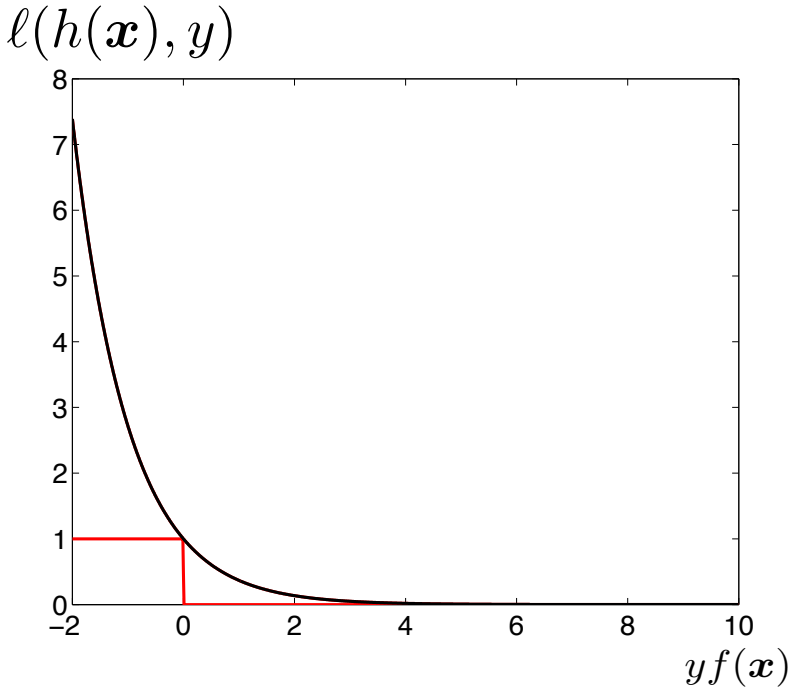
Surrogate Loss

$0-1$ loss function $\ell(h(\boldsymbol{x}), y)$ is non-convex and difficult to optimize.

We can instead use a surrogate loss – what are examples?

Exponential Loss

$$\ell^{\text{EXP}}(h(\boldsymbol{x}), y) = e^{-yf(\boldsymbol{x})}$$



Choosing the t-th Classifier

Boosting iteratively minimizes the loss. What do we mean by iteratively?

Suppose we have classifier $f_{t-1}(\boldsymbol{x}) = \beta_1 h_1(\boldsymbol{x}) + \cdots + \beta_{t-1} h_{t-1}(\boldsymbol{x})$.

Now, we want to add a weak learner $h_t(\boldsymbol{x})$

$$f(\boldsymbol{x}) = f_{t-1}(\boldsymbol{x}) + \beta_t h_t(\boldsymbol{x})$$

How can we 'optimally' choose $h_t(\boldsymbol{x})$ and combination coefficient β_t?

Adaboost greedily *minimizes the exponential loss function*!

$$\begin{aligned}
(h_t^*(\boldsymbol{x}), \beta_t^*) &= \text{argmin}_{(h_t(\boldsymbol{x}), \beta_t)} \sum_n e^{-y_n f(\boldsymbol{x}_n)} \\
&= \text{argmin}_{(h_t(\boldsymbol{x}), \beta_t)} \sum_n e^{-y_n [f_{t-1}(\boldsymbol{x}_n) + \beta_t h_t(\boldsymbol{x}_n)]} \\
&= \text{argmin}_{(h_t(\boldsymbol{x}), \beta_t)} \sum_n w_t(n) e^{-y_n \beta_t h_t(\boldsymbol{x}_n)}
\end{aligned}$$

where we have used $w_t(n)$ as a shorthand for $e^{-y_n f_{t-1}(\boldsymbol{x}_n)}$

The New Classifier

We can decompose the *weighted* loss function into two parts

$$\begin{aligned}
&\sum_n w_t(n) e^{-y_n \beta_t h_t(\boldsymbol{x}_n)} \\
&= \sum_n w_t(n) e^{\beta_t} \mathbb{I}[y_n \neq h_t(\boldsymbol{x}_n)] + \sum_n w_t(n) e^{-\beta_t} \mathbb{I}[y_n = h_t(\boldsymbol{x}_n)] \\
&= \sum_n w_t(n) e^{\beta_t} \mathbb{I}[y_n \neq h_t(\boldsymbol{x}_n)] + \sum_n w_t(n) e^{-\beta_t} (1 - \mathbb{I}[y_n \neq h_t(\boldsymbol{x}_n)]) \\
&= (e^{\beta_t} - e^{-\beta_t}) \sum_n w_t(n) \mathbb{I}[y_n \neq h_t(\boldsymbol{x}_n)] + e^{-\beta_t} \sum_n w_t(n)
\end{aligned}$$

We have used the following properties to derive the above

- $y_n h_t(\boldsymbol{x}_n)$ is either 1 or -1 as $h_t(\boldsymbol{x}_n)$ is the output of a binary classifier
- The indicator function $\mathbb{I}[y_n = h_t(\boldsymbol{x}_n)]$ is either 0 or 1, so it equals $1 - \mathbb{I}[y_n \neq h_t(\boldsymbol{x}_n)]$

Finding the Optimal Weak Learner

The loss minimization problem now becomes:

$$
\begin{aligned}
(h_t^*(\boldsymbol{x}), \beta_t^*) &= \text{argmin}_{(h_t(\boldsymbol{x}),\beta_t)} \sum_n w_t(n) e^{-y_n \beta_t h_t(\boldsymbol{x}_n)} \\
&= \text{argmin}_{(h_t(\boldsymbol{x}),\beta_t)} (e^{\beta_t} - e^{-\beta_t}) \sum_n w_t(n) \mathbb{I}[y_n \neq h_t(\boldsymbol{x}_n)] \\
&\qquad + e^{-\beta_t} \sum_n w_t(n)
\end{aligned}
$$

What term(s) must we optimize to choose $h_t(\boldsymbol{x}_n)$?

$$
h_t^*(\boldsymbol{x}) = \text{argmin}_{h_t(\boldsymbol{x})} \underbrace{\sum_n w_t(n) \mathbb{I}[y_n \neq h_t(\boldsymbol{x}_n)]}_{=\epsilon_t}
$$

Minimize weighted classification error as noted in step 1 of Adaboost!

How to Choose β_t?

The loss minimization problem now becomes:

$$
\begin{aligned}
(h_t^*(\boldsymbol{x}), \beta_t^*) &= \text{argmin}_{(h_t(\boldsymbol{x}),\beta_t)} \sum_n w_t(n) e^{-y_n \beta_t h_t(\boldsymbol{x}_n)} \\
&= \text{argmin}_{(h_t(\boldsymbol{x}),\beta_t)} (e^{\beta_t} - e^{-\beta_t}) \sum_n w_t(n) \mathbb{I}[y_n \neq h_t(\boldsymbol{x}_n)] \\
&\qquad + e^{-\beta_t} \sum_n w_t(n) \\
&= \text{argmin}_{\beta_t} (e^{\beta_t} - e^{-\beta_t}) \epsilon_t + e^{-\beta_t} \sum_n w_t(n)
\end{aligned}
$$

Plugging in ϵ_t (since we optimized over $h_t(\boldsymbol{x})$), now how do we minimize over β_t?

We can take the derivative with respect to β_t, set it to zero, and solve for β_t. After some calculation and using $\sum_n w_t(n) = 1$...

$$\beta_t^* = \frac{1}{2} \log \frac{1 - \epsilon_t}{\epsilon_t}$$

which is precisely step 2 of Adaboost! (*Exercise – verify the solution*)

Updating the Weights

Once we find the optimal weak learner we can update our classifier:

$$f_t(\boldsymbol{x}) = f_{t-1}(\boldsymbol{x}) + \beta_t^* h_t^*(\boldsymbol{x})$$

Earlier, we set $w_t(n) = e^{-y_n f_{t-1}(\boldsymbol{x}_n)}$

Therefore, to get $w_{t+1}(n)$, we have

$$w_{t+1}(n) = e^{-y_n f_t(\boldsymbol{x}_n)} = e^{-y_n[f_{t-1}(\boldsymbol{x}_n)+\beta_t^* h_t^*(\boldsymbol{x}_n)]}$$

$$= w_t(n) e^{-y_n \beta_t^* h_t^*(\boldsymbol{x}_n)} = \begin{cases} w_t(n) e^{\beta_t^*} & \text{if } y_n \neq h_t^*(\boldsymbol{x}_n) \\ w_t(n) e^{-\beta_t^*} & \text{if } y_n = h_t^*(\boldsymbol{x}_n) \end{cases}$$

Intuition Misclassified data points will get their weights increased, while correctly classified data points will get their weight decreased

This is basically step 3 of Adaboost! The difference is that Adaboost also normalizes the weight (the derivation for that is similar).

Adaboost Algorithm

- For $t = 1$ to T
  1. Train a weak classifier $h_t(\boldsymbol{x})$ using weights $w_t(n)$, by minimizing

     $$\epsilon_t = \sum_n w_t(n) \mathbb{I}[y_n \neq h_t(\boldsymbol{x}_n)]$$ (the weighted classification error; *Why?*)

  2. Compute contribution for this classifier: $\beta_t = \frac{1}{2} \log \frac{1-\epsilon_t}{\epsilon_t}$; *Why?* (smaller $\epsilon_t \Rightarrow$ larger β_t)
  3. Update weights on each training sample n; *Why?*

     $$w_{t+1}(n) \propto w_t(n) e^{-\beta_t y_n h_t(\boldsymbol{x}_n)}$$

     ($w_t(n)$ decreased if $y_n = h_t(\boldsymbol{x}_n)$; increased if $y_n \neq h_t(\boldsymbol{x}_n)$)
     and normalize them such that $\sum_n w_{t+1}(n) = 1$.
- Output the final classifier

$$h[\boldsymbol{x}] = \text{sign}\left[\sum_{t=1}^{T} \beta_t h_t(\boldsymbol{x})\right]$$

The three steps are derived based on iteratively minimizing the exponential loss function!

Meta-Algorithm

Note that the AdaBoost algorithm itself never specifies how we would get $h_t^*(\boldsymbol{x})$ as long as it minimizes the weighted classification error

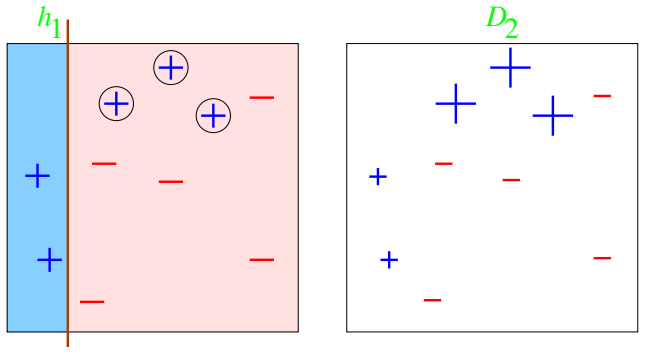
$$\epsilon_t = \sum_n w_t(n) \mathbb{I}[y_n \neq h_t^*(\boldsymbol{x}_n)]$$

In this sense, the AdaBoost algorithm is a meta-algorithm and can be used with any type of classifier

Often, We Use Decision Stumps

Often, we use decision stumps (binary trees with depth 1).

How do we choose the decision stump classifier given the weights at the second round of the following distribution?



We can simply enumerate all possible ways of putting vertical and horizontal lines to separate the data points into two classes and find the one with the smallest weighted classification error! Runtime?

- Presort data by each feature in $O(dN \log N)$ time
- Evaluate $N + 1$ thresholds for each feature at each round in $O(dN)$ time
- In total $O(dN \log N + dNT)$ time – this efficiency is an attractive quality of boosting!

History of AdaBoost

A decision-theoretic generalization of on-line learning and an application to boosting

Authors	Yoav Freund, Robert E Schapire
Publication date	1997/8/1
Journal	Journal of computer and system sciences
Volume	55
Issue	1
Pages	119-139
Publisher	Academic Press
Description	In the first part of the paper we consider the problem of dynamically apportioning resources among a set of options in a worst-case on-line framework. The model we study can be interpreted as a broad, abstract extension of the well-studied on-line prediction model to a general decision-theoretic setting. We show that the multiplicative weight-update Littlestone–Warmuth rule can be adapted to this model, yielding bounds that are slightly weaker in some cases, but applicable to a considerably more general class of learning problems. We show how the resulting learning algorithm can be applied to a variety of problems, including gambling, multiple-outcome prediction, repeated games, and prediction of points in R^n. In the second part of the paper we apply the multiplicative weight-update technique to derive a new boosting algorithm. This boosting algorithm does not require any prior knowledge about the …
Total citations	Cited by 27757

- Still a very popular algorithm!
- First boosting algorithm to adaptively choose weak learners.

Other Boosting Algorithms

- AdaBoost is by far the most popular boosting algorithm.
- Gradient boosting generalizes AdaBoost by substituting another (smooth) loss function for exponential loss.
  - Squared loss, logistic loss, ...
  - Choose the candidate learner that greedily minimizes the error.
  - Reduce overfitting by constraining the candidate learner (e.g., computing the residuals only over a sample of points).
- LogitBoost minimizes the logistic loss instead of exponential loss.
- Gentle AdaBoost bounds the contribution (β_t) of each learner update.

XGBoost (eXtreme Gradient Boosting)

Very popular boosting library that has been used in several machine learning competitions. Compared to AdaBoost:

- AdaBoost can be viewed as a form of gradient boosting, where we choose the weak learners in the direction of steepest descent of the exponential loss function.

$$(h_t^*(\boldsymbol{x}), \beta_t^*) = \text{argmin}_{(h_t(\boldsymbol{x}), \beta_t)} \sum_n e^{-y_n f(\boldsymbol{x}_n)}$$

  - XGBoost uses Newton-Raphson to choose the weak learners so as to minimize a loss function.
  - Newton-Raphson uses second derivatives to minimize a function.
- XGBoost includes efficient implementations of decision tree training and runs on distributed systems.

Ensemble Methods: You Should Know

Fight the bias-variance tradeoff by combining (by a weighted sum or majority vote) the outputs of many classifiers.

Bagging and Random Forests: Reduce variance of fully grown trees.

- Key idea: train multiple trees (with some "random variations") and classify by majority vote.
- Bagging: training multiple trees on random subsets of the train set.
- Random forests: Add more "random variations" by constraining splits to random subsets of features, which decreases the correlation between the trees.

Boosting: Reduce bias of weak classifiers. It sequentially adds weak classifiers and optimizes their contributions, increasing the weight of "hard" data points at each step.

- Greedily minimizes a surrogate classification loss
- Commonly uses shallow decision trees as weak classifiers