18-661 Introduction to Machine Learning

Ensemble Methods

Spring 2020

ECE – Carnegie Mellon University

Admin

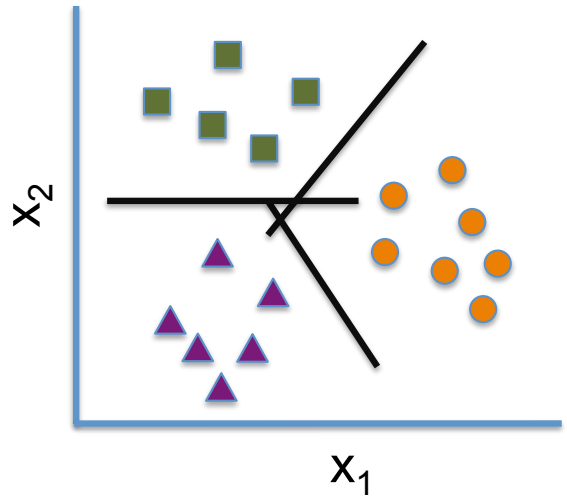
- **No recitation this Friday** (mid-semester break)
- Homework 4 will be released Friday, due on 3/18 (Wednesday after spring break)
- Midterm exam grades were released yesterday. Solutions have been posted on Canvas, and regrades will open for one week after this lecture.

Outline

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

Recap: Decision Trees

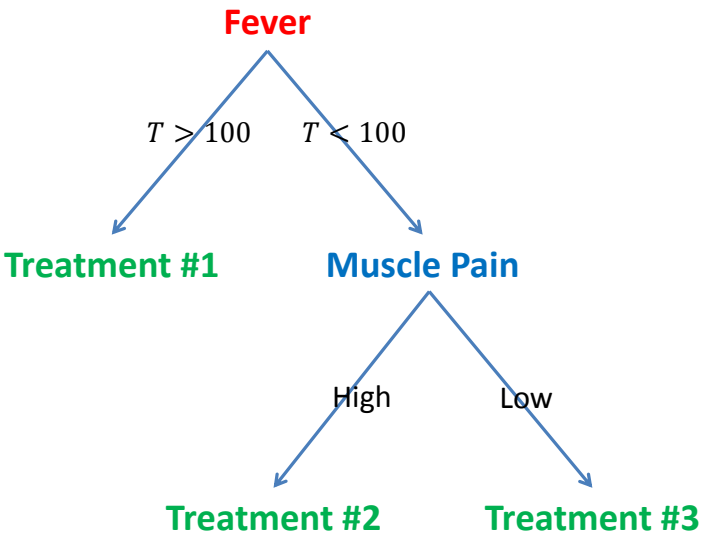
Need interpretable decision boundaries



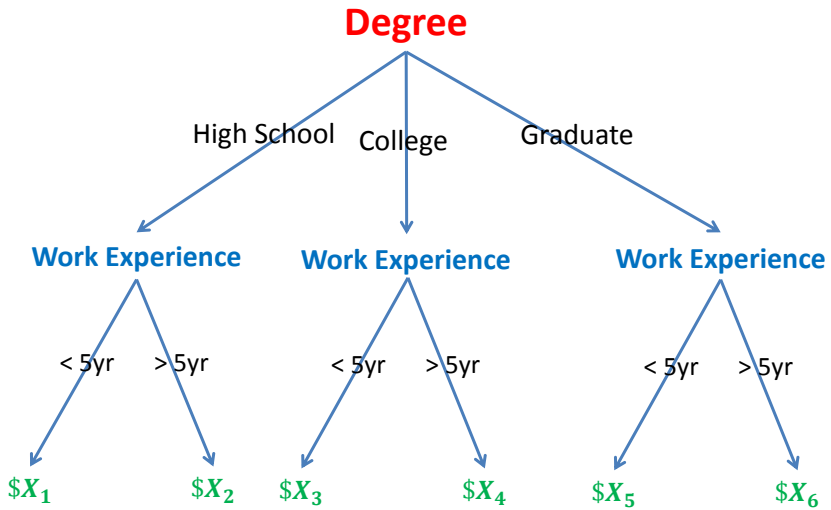
- Should be able to explain the reasoning in clear terms, e.g. "I always recommend treatment 1 when a patient has fever $\geq 100F$"
- The rules that you use to make decisions can be easily used by a lay-person without performing complex computations
- Decision trees can provide such simple decision rules

Many decisions are tree structured

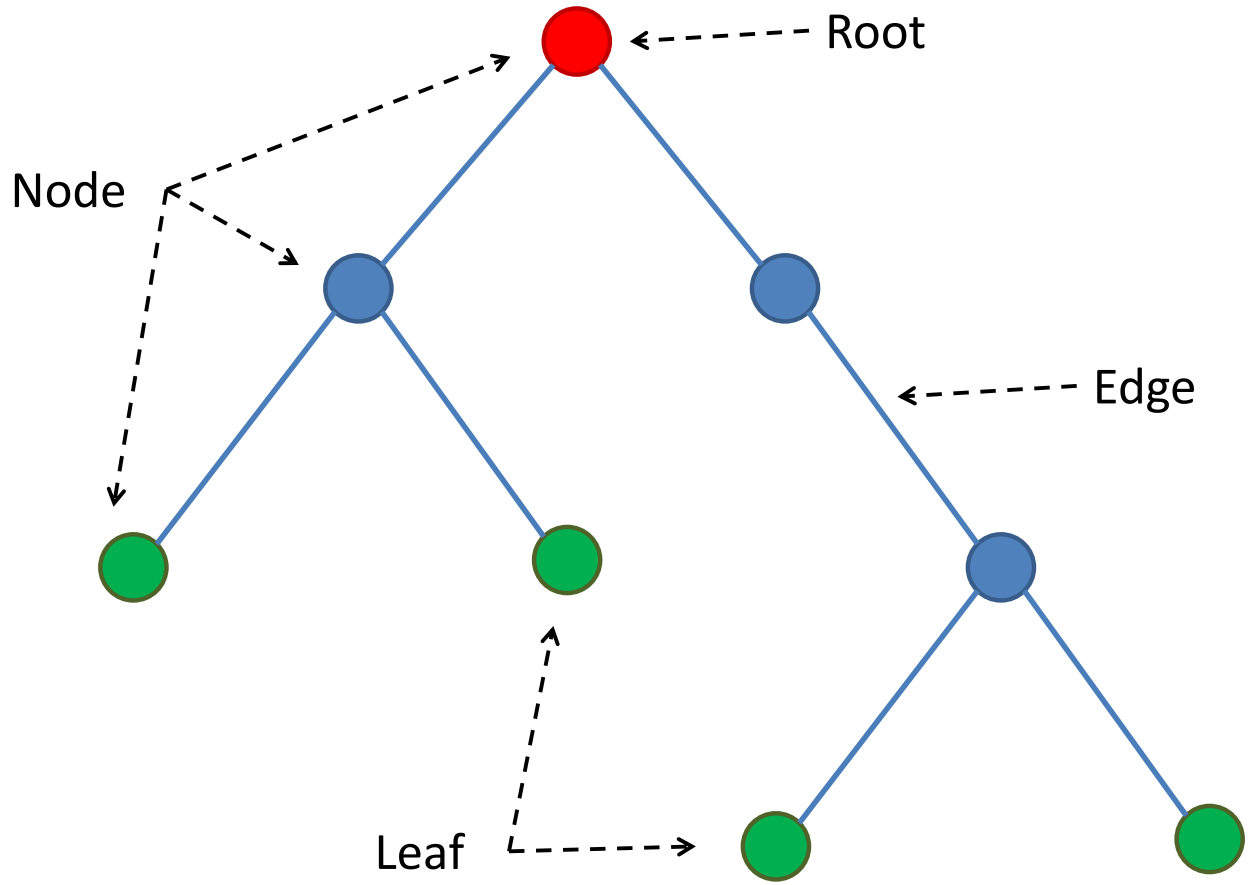
Medical treatment



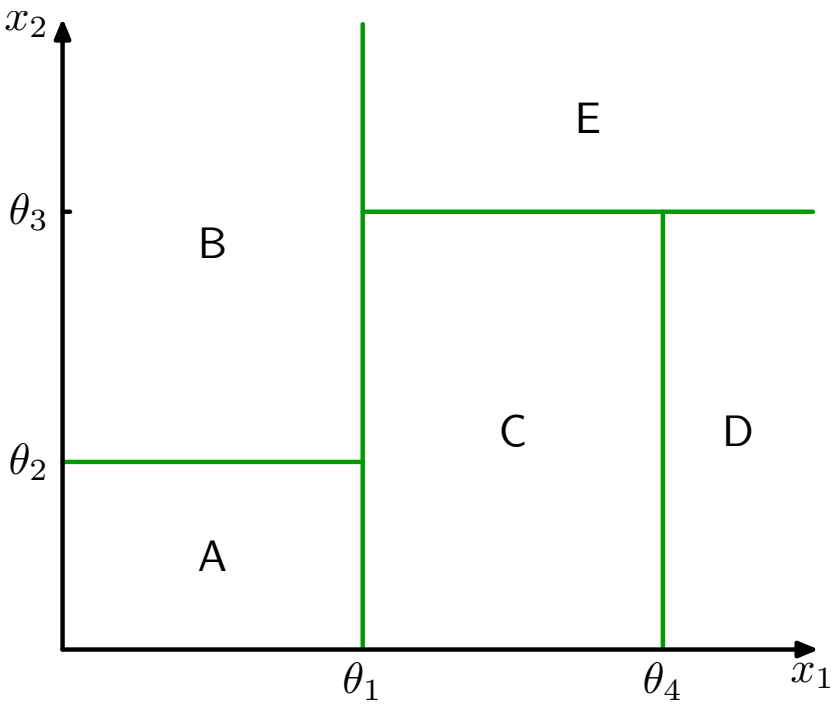
Salary in a company

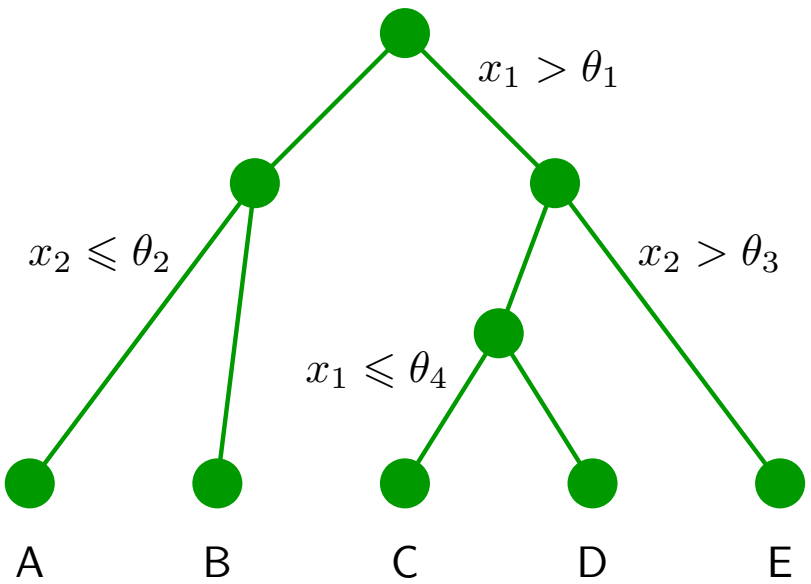


Special Names for Nodes in a Tree

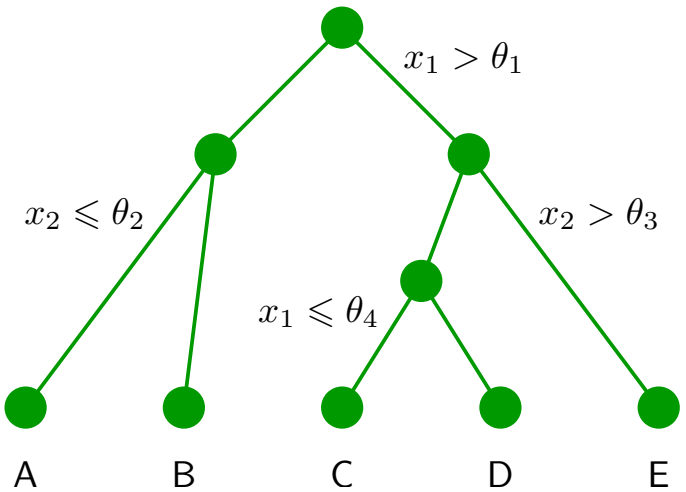


A tree partitions the feature space





Learning a tree model



Three things to learn:

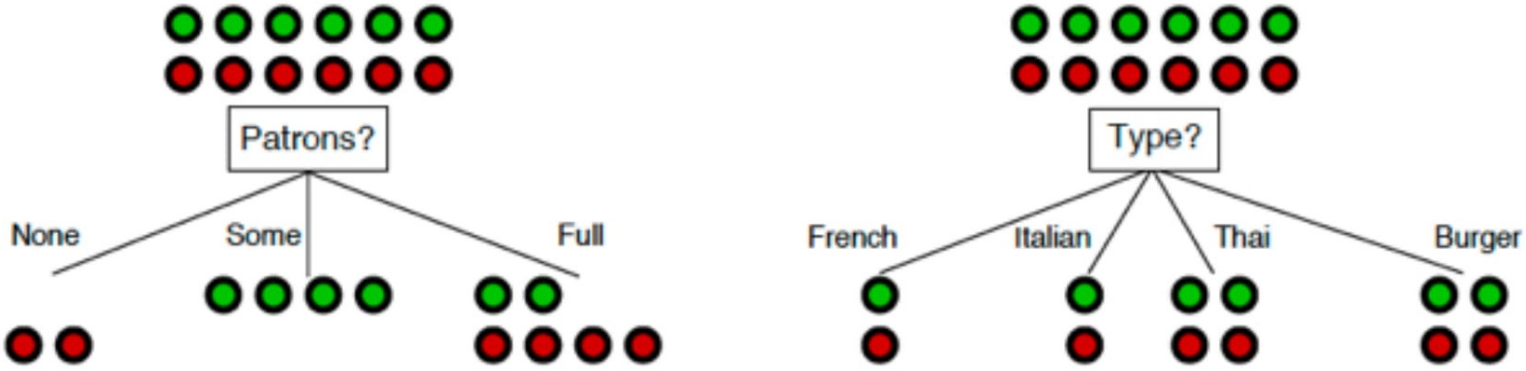
1. The structure of the tree.
2. The threshold values (θ_i).
3. The values for the leafs ($A, B, \ldots$).

Example: Choosing whether you want to wait at a restaurant

Attributes										Target
Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
T	*F*	*F*	*T*	*Some*	*$$$*	*F*	*T*	*French*	*0–10*	*T*
T	*F*	*F*	*T*	*Full*	*$*	*F*	*F*	*Thai*	*30–60*	*F*
F	*T*	*F*	*F*	*Some*	*$*	*F*	*F*	*Burger*	*0–10*	*T*
T	*F*	*T*	*T*	*Full*	*$*	*F*	*F*	*Thai*	*10–30*	*T*
T	*F*	*T*	*F*	*Full*	*$$$*	*F*	*T*	*French*	*>60*	*F*
F	*T*	*F*	*T*	*Some*	*$$*	*T*	*T*	*Italian*	*0–10*	*T*
F	*T*	*F*	*F*	*None*	*$*	*T*	*F*	*Burger*	*0–10*	*F*
F	*F*	*F*	*T*	*Some*	*$$*	*T*	*T*	*Thai*	*0–10*	*T*
F	*T*	*T*	*F*	*Full*	*$*	*T*	*F*	*Burger*	*>60*	*F*
T	*T*	*T*	*T*	*Full*	*$$$*	*F*	*T*	*Italian*	*10–30*	*F*
F	*F*	*F*	*F*	*None*	*$*	*F*	*F*	*Thai*	*0–10*	*F*
T	*T*	*T*	*T*	*Full*	*$*	*F*	*F*	*Burger*	*30–60*	*T*

Use the attributes to decide whether to wait (T) or not wait (F)

Which attribute to split first?



- Patron is a better choice – gives more information to help distinguish between the labels
- Intuition: Like playing 20 questions and choosing carefully which question to ask first
- Idea: use information gain to choose which attribute to split

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

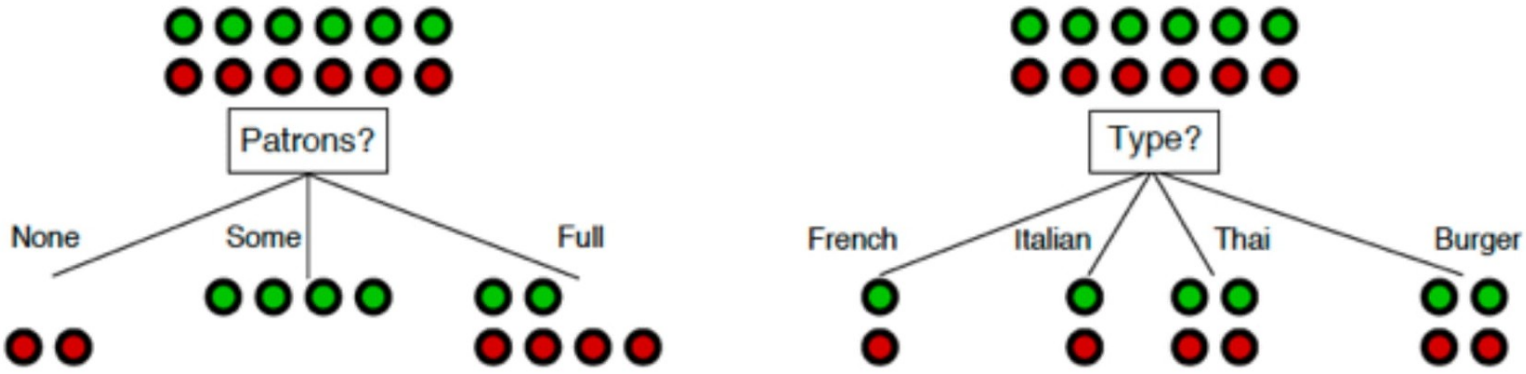
$$H[Y|X] = \sum_{k} P(X = a_k) H[Y|X = a_k]$$

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.

Splitting on "Patron" or "Type"?



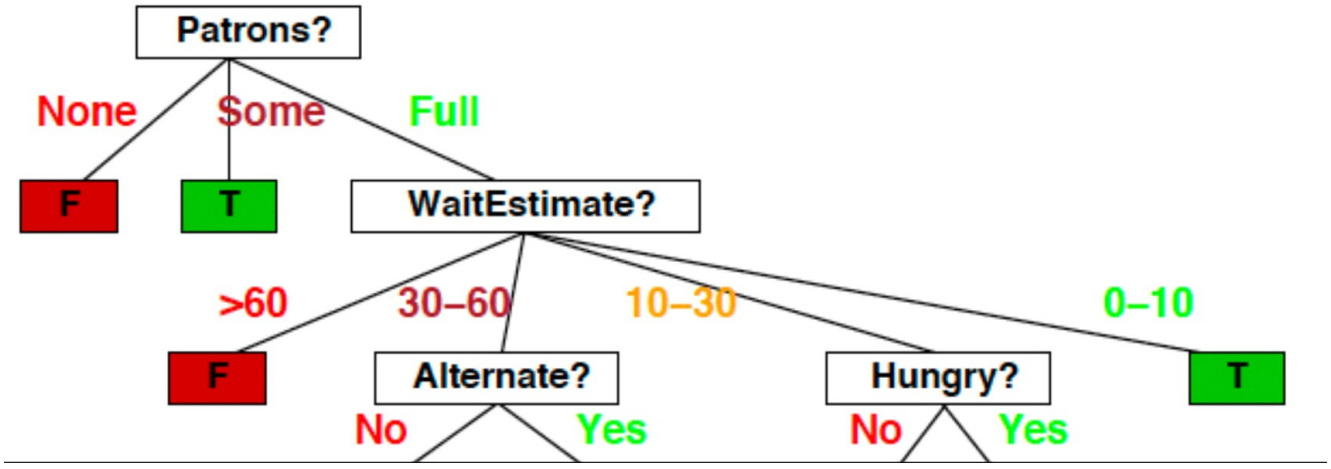
- Information gain from "Patron" is 0.55 bits.
- Information gain from "Type" is 0 bits.

Thus, we should split on "Patron" and not "Type" (higher information gain). This is consistent with our intuition.

What is the optimal Tree Depth?

- What happens if we pick the wrong depth?
  - If the tree is too deep, we can overfit
  - If the tree is too shallow, we underfit
- Max depth is a hyperparameter that should be tuned by the data
- Alternative strategy is to create a very deep tree, and then to prune it (see Section 9.2.2 in ESL for details)

How to classify with a pruned decision tree?



- If we stop here, not all training samples would be classified correctly
- More importantly, how do we classify a new instance?
- We label the leaves of this smaller tree with the majority of training sample's labels.

Computational Considerations

Numerical Features

- How should we decide the threshold to use in splitting the feature?
- Can we do this efficiently?
  - Yes – for a given feature we only need to consider the n values in the training data!
  - If we sort each feature by these n values, we can quickly compute and maximize the information gain
  - This takes $O(dn \log n)$ time, where d is the no. of features

Categorical Features

- Assuming q distinct categories, there are $2^{q-1} - 1$ possible partitions
- Things simplify in the case of binary classification or regression
  - Can sort the features by the frac. of labels falling in class 1
  - Suffices to consider only $q - 1$ possible splits (see Section 9.2.4 in ESL)

Avoiding Overfitting in Decision Trees

Overfitting in Decision Trees

- Irrelevant attributes can result in overfitting the training example data.
- If we have too little training data, even a reasonable hypothesis space will overfit.

How can we avoid overfitting?

- Acquire more training data.
- Remove irrelevant attributes (manual process — not always possible).
- Stop growing when data split is not statistically significant.
- Grow full tree, then post-prune

Summary of Decision Trees

Advantages of decision trees

- Can be interpreted by humans (as long as the tree is not too big)
- Handles both numerical and categorical data
- Compact representation: unlike Nearest Neighbors we don't need training data at test time
- But, like NN, decision trees are *nonparametric* because the number of parameters depends on the data
- Building block for various ensemble methods (more on this later)

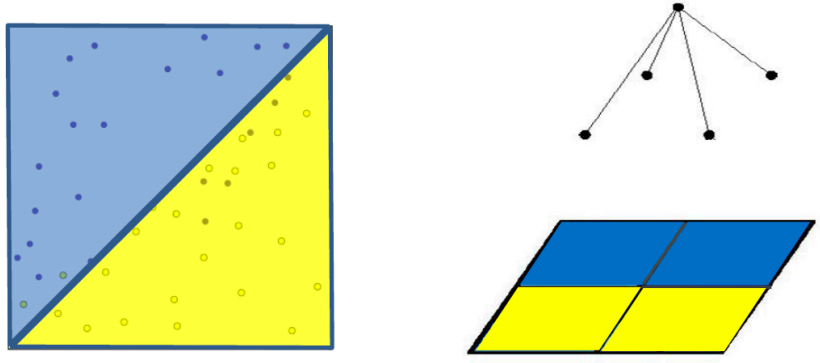
Disadvantages of decision trees

- Binary decision trees find it hard to learn linear boundaries.
- Decision trees can have high variance.
- We use heuristic training techniques: finding the optimal tree is NP-hard.

Ensemble Methods: Motivation

Motivation: Fighting the bias-variance tradeoff

- Simple (a.k.a weak) learners such as Naive Bayes, logistic regression, decision stumps (shallow decision trees)



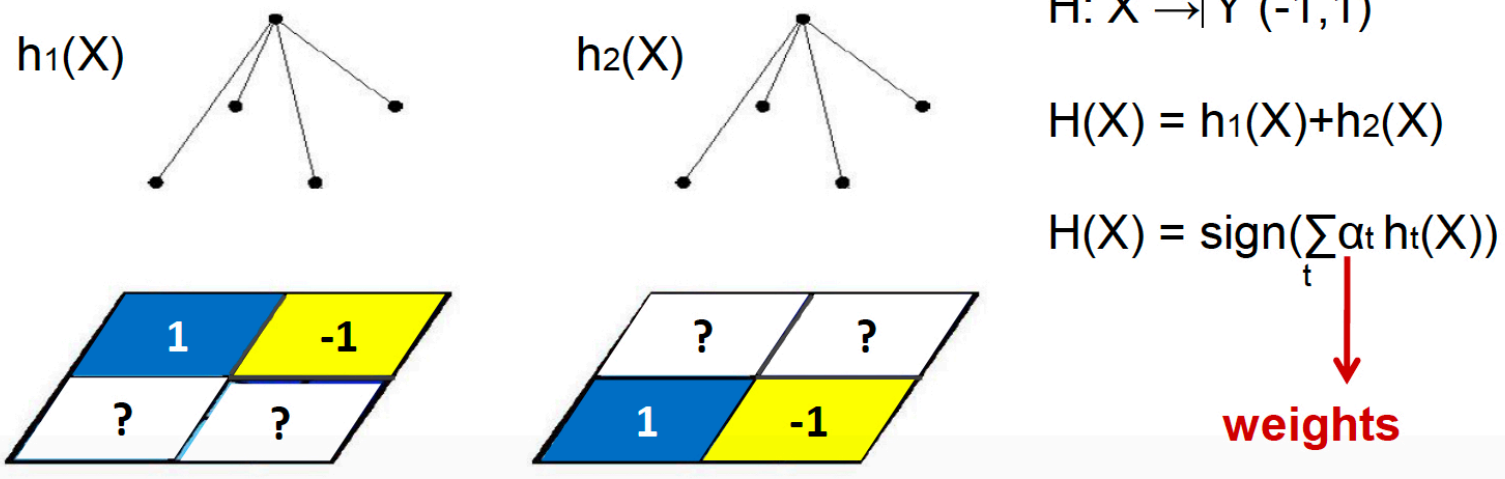
- Pros: Low variance, don't overfit
- Cons: High Bias, can't fit complex boundaries

Can we improve weak learners?

Idea: Train several weak learners and take an average or majority vote of their outputs

Ensemble methods

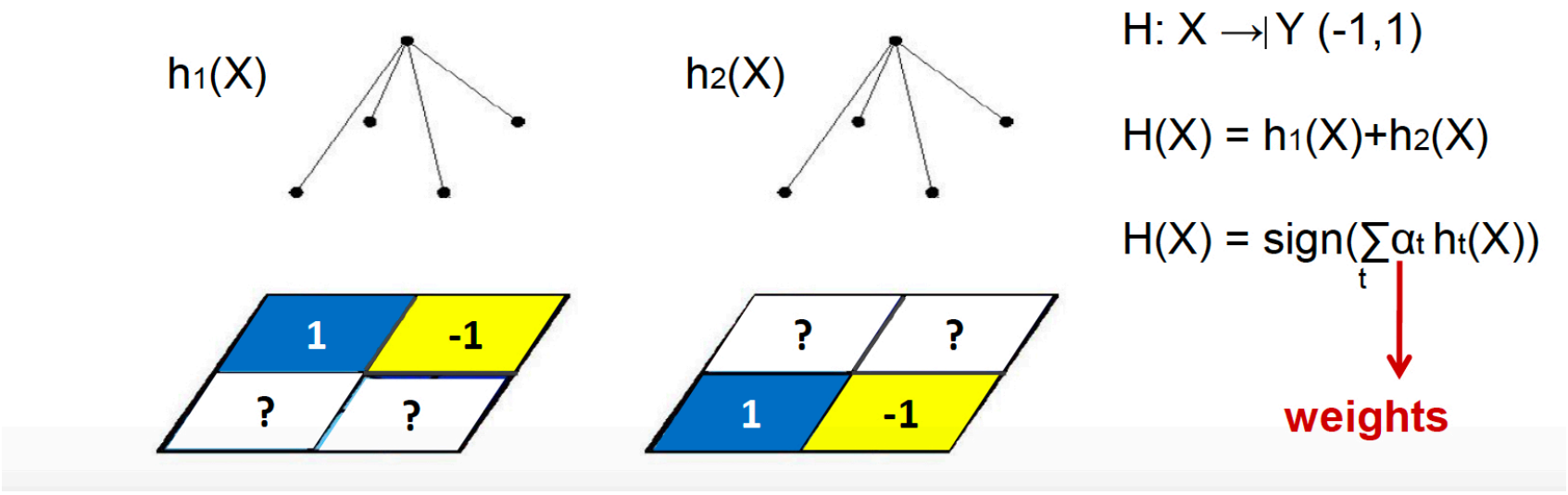
- Instead of learning a single (weak) classifier, learn many weak classifiers, preferably those that are good at different parts of the input spaces
- Predicted Class: (Weighted) Average or Majority of output of the weak classifiers
- Strength in Diversity!



Ensemble methods

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

- Bagging or Bootstrap Aggregation
- Random Forests
- AdaBoost



Bagging and Random Forests

Bagging or Bootstrap Aggregating

To avoid overfitting a decision tree to a given dataset we can average an ensemble of trees learnt on random subsets of the training data.

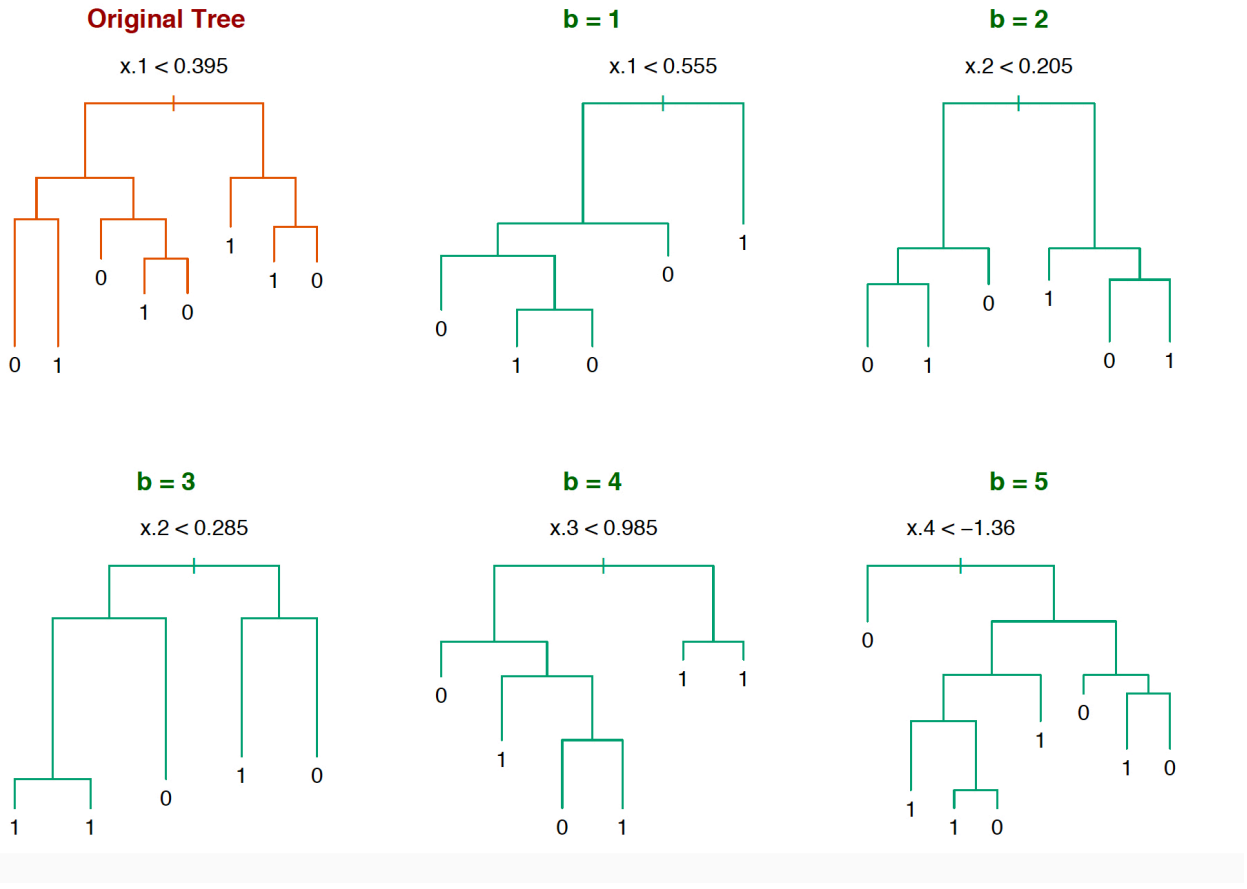
Bagging Trees (Training Phase)

- For $b = 1, 2, \cdots, B$
  - Choose n training samples $(\mathbf{x}_i, y_i)$ from $\mathcal{D}$ uniformly at random
  - 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 label as the label for $\mathbf{x}$

Bagging: Example

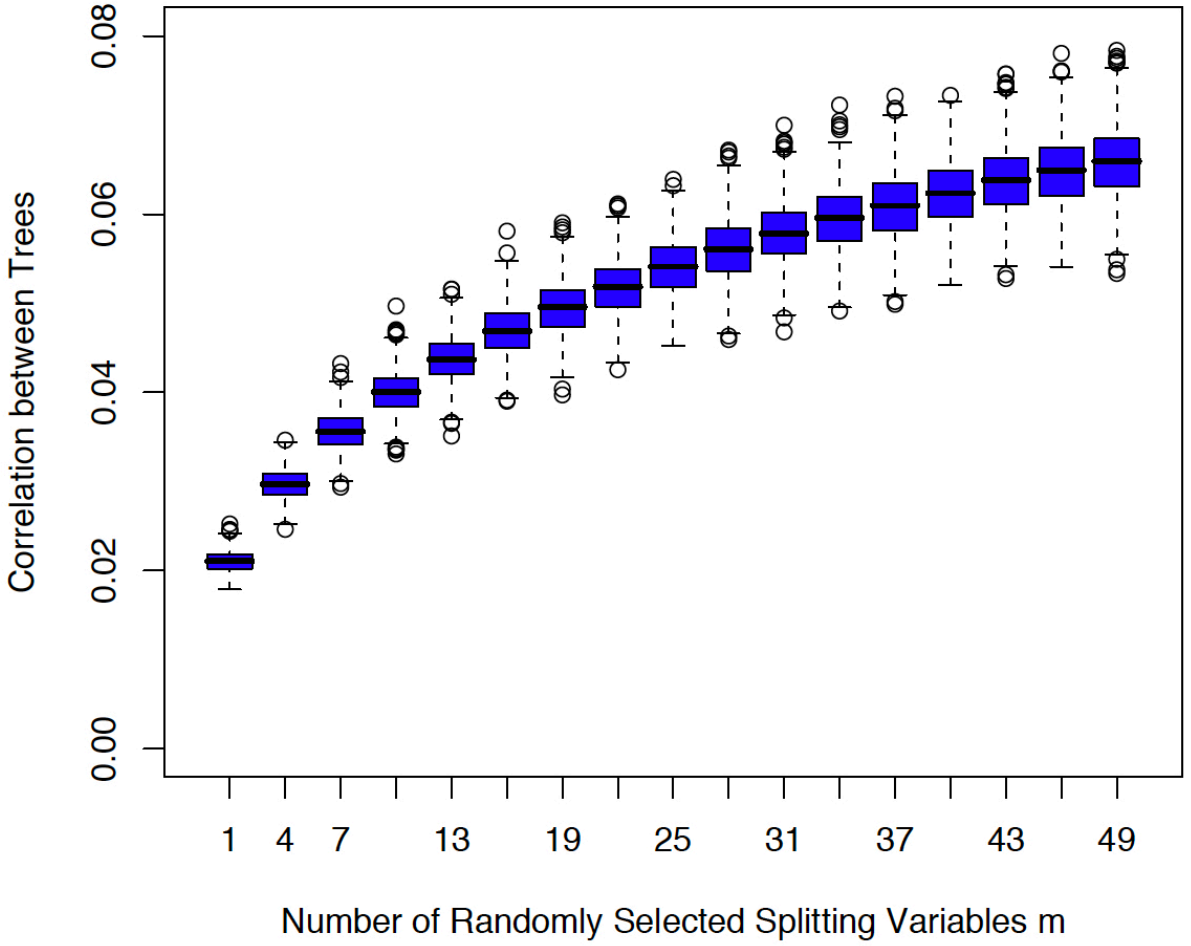


- We get different splits and thresholds for different trees b
- Predict the label assigned by majority of the B trees
- Reduces variance without increasing bias, thus avoiding overfitting

Random Forests

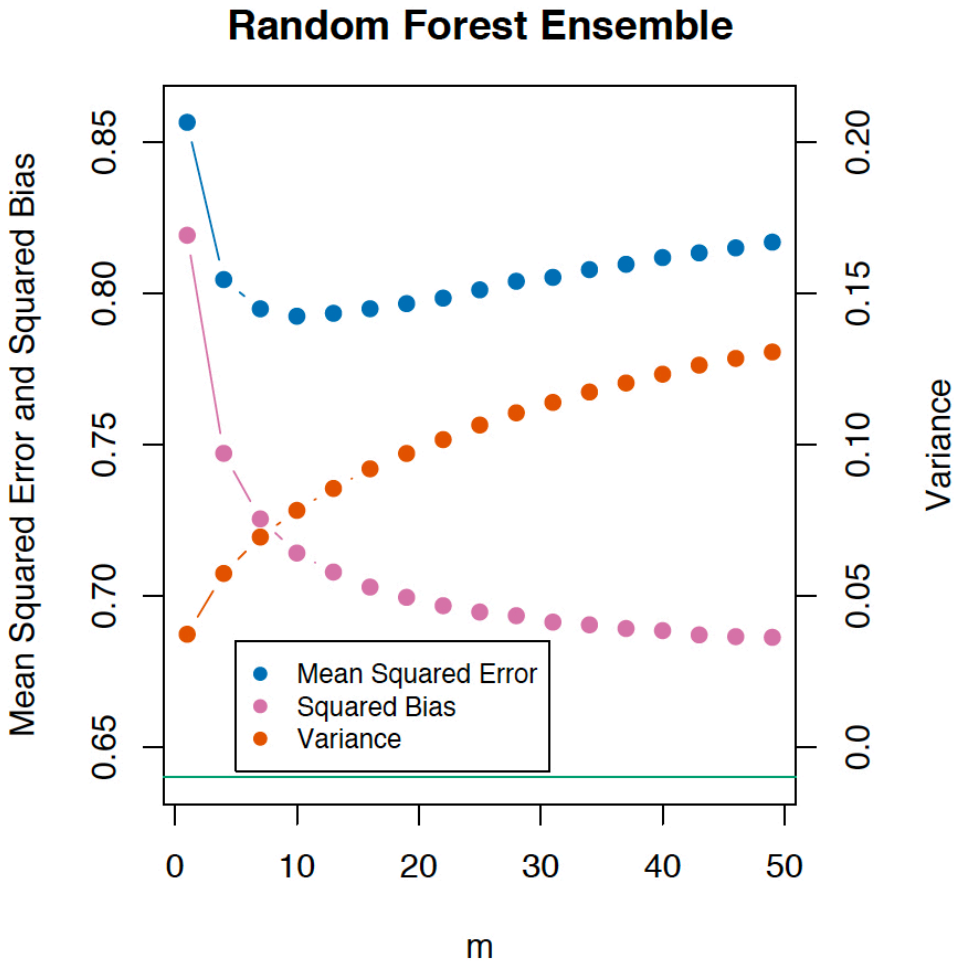
- Limitation of Bagging: If one or more features are very informative, they will be selected by almost every tree in the bag, reducing the diversity (and potentially increasing the bias).
- Key Idea behind Random Forests: Reduces correlation between trees in the bag without increasing variance too much
  - Same as bagging in terms of sampling training data
  - Before each split, select $m \leq d$ features at random as candidates for splitting $m \sim \sqrt{d}$
  - Take majority vote of B such trees

Random Forests



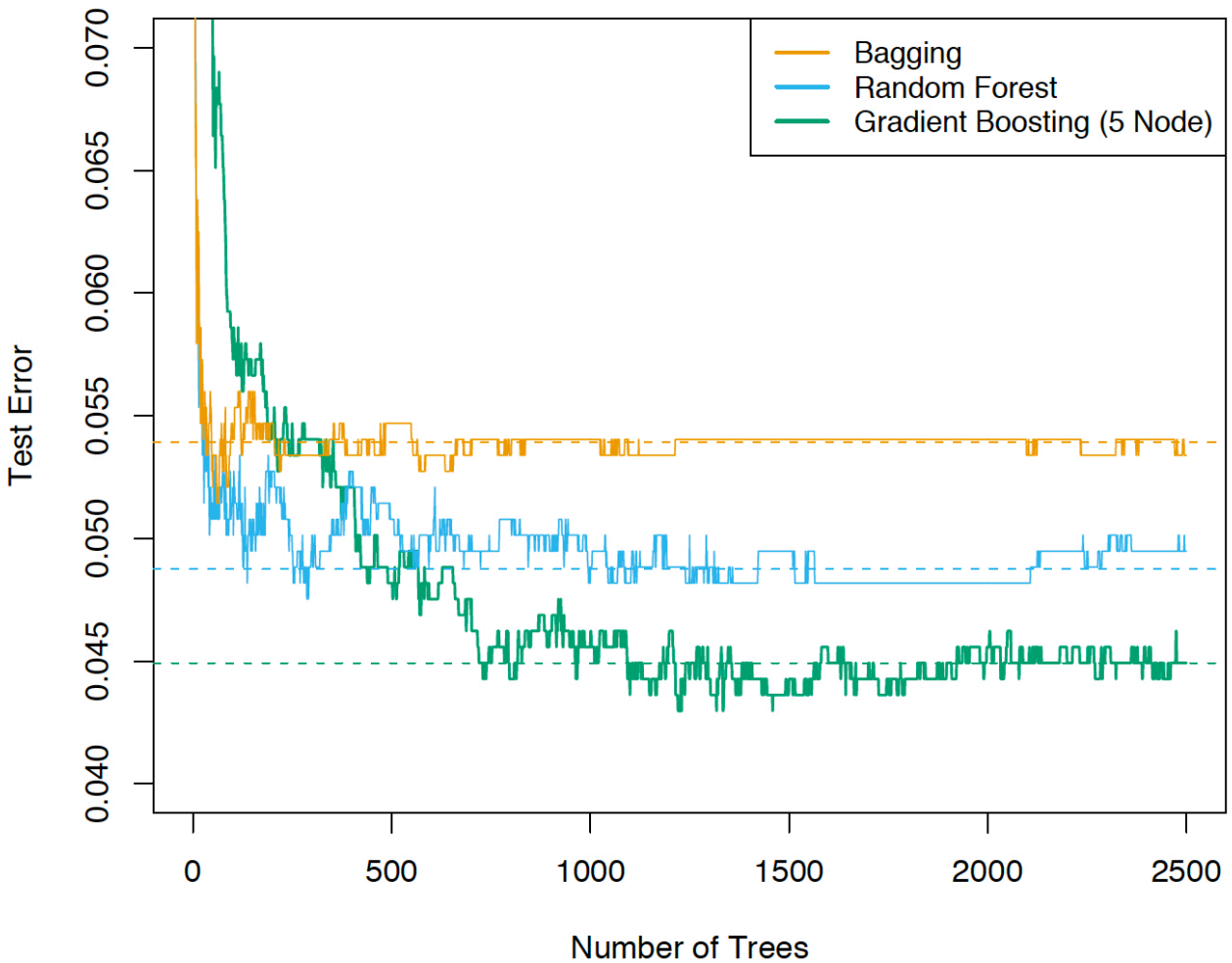
Increasing m, the number of splitting candidates chosen increases the correlation among the trees in the bag

Random Forests



Increasing m decreases the bias but increases variance in the ensemble

Random Forests



Outline

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

Boosting and AdaBoost

Limitations of Bagging and Random Forests

- Bagging: Significant correlation between trees that are learnt on different training datasets
- Random Forests try to resolve this by doing "feature bagging" but some correlation still remains
- All B trees are given the same weight when taking the average

Boosting methods: Force classifiers to learn on different parts of the feature space, and take their *weighted* average

Boosting

High-level idea: combine a lot of classifiers

- Sequentially construct / identify these 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
- Derive the algorithm

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
- For $t = 1$ to T
  1. Train a weak classifier $h_t(\boldsymbol{x})$ using current 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: $\beta_t = \frac{1}{2} \log \frac{1-\epsilon_t}{\epsilon_t}$
  3. Update weights on training points

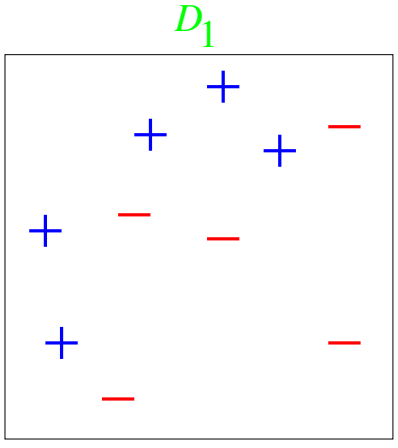
$$w_{t+1}(n) \propto w_t(n) e^{-\beta_t y_n h_t(\boldsymbol{x}_n)}$$

  and normalize them such that $\sum_n w_{t+1}(n) = 1$
- Output the final classifier

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

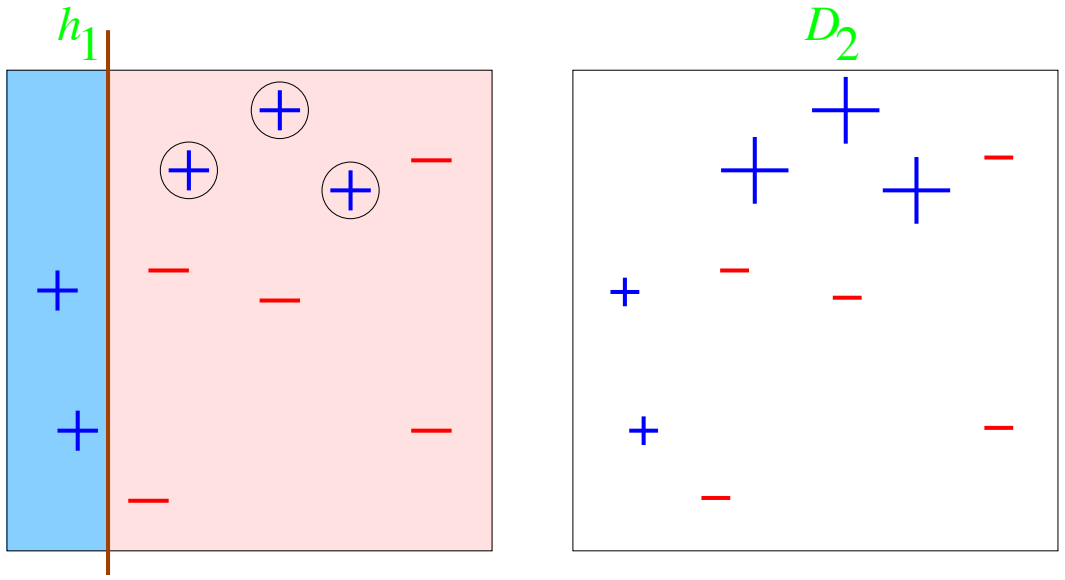
Example

10 data points and 2 features



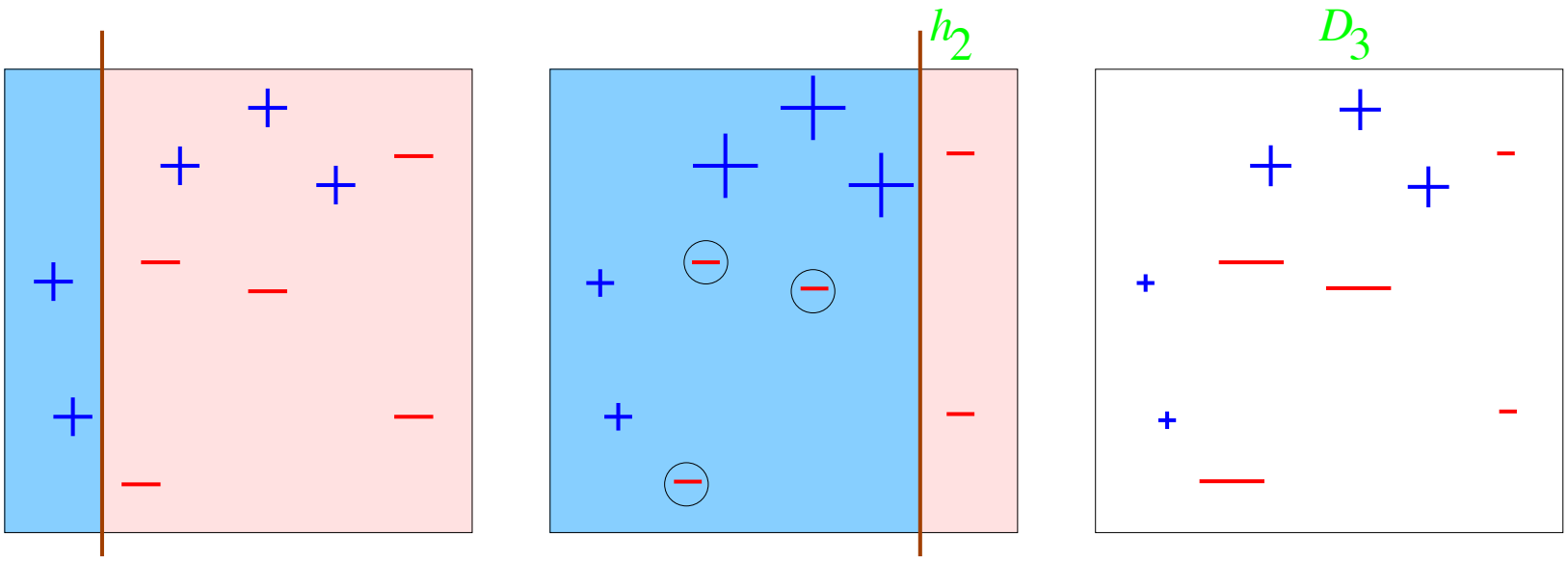
- The data points are clearly not linearly separable
- In the beginning, all data points have equal weights (the size of the data markers "+" or "-")
- Base classifier $h(\cdot)$: horizontal or vertical lines ('decision stumps')
  - Depth-1 decision trees, i.e., classify data based on a single attribute.

Round 1: $t = 1$



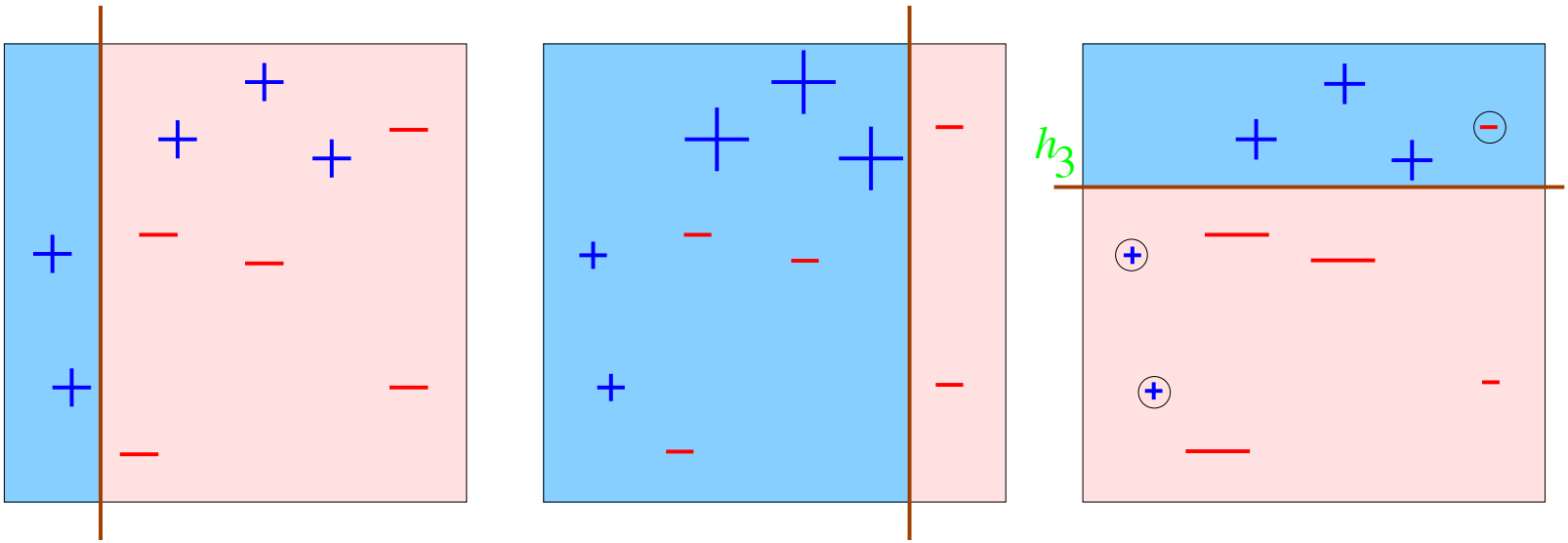
- 3 misclassified (with circles): $\epsilon_1 = 0.3 \rightarrow \beta_1 = 0.42$.
- Recompute the weights; the 3 misclassified data points receive larger weights

Round 2: $t = 2$



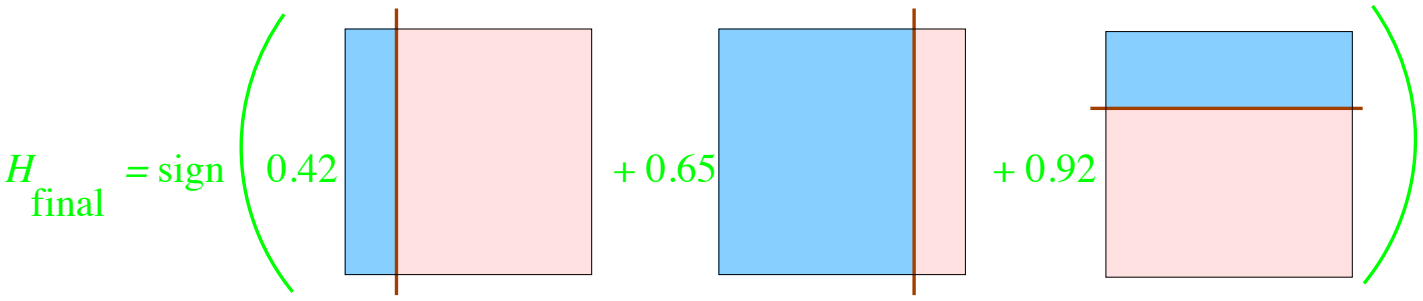
- 3 misclassified (with circles): $\epsilon_2 = 0.21 \rightarrow \beta_2 = 0.65$.
  $\epsilon_2 < 0.3$ as those 3 misclassified data points have weights < 0.1
- 3 newly misclassified data points get larger weights
- Data points classified correctly in both rounds have small weights

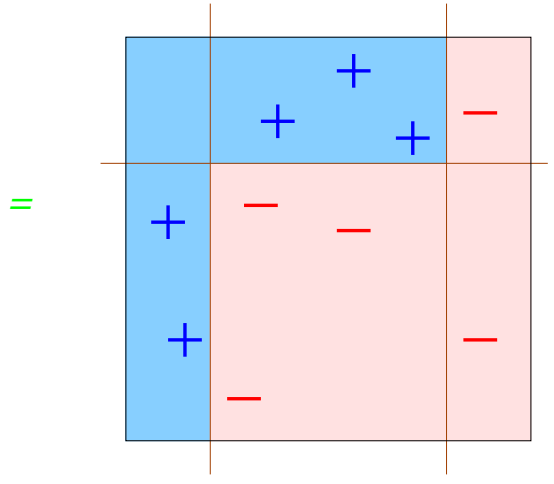
Round 3: $t = 3$



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

Final classifier: Combining 3 classifiers





- All data points are now classified correctly!

Why does AdaBoost work?

It 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.

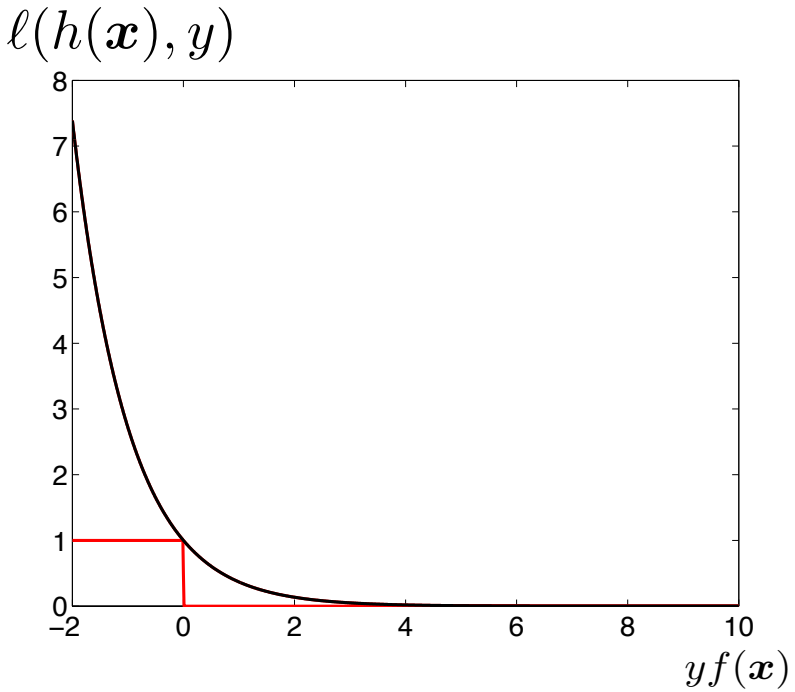
Surrogate loss

$0-1$ 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

Suppose a classifier $f_{t-1}(\boldsymbol{x})$, and want to add a weak learner $h_t(\boldsymbol{x})$

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

Note: $h_t(\cdot)$ outputs -1 or 1, as does sign $[f_{t-1}(\cdot)]$

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

Summary

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

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

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

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

How to choose β_t?

Summary

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

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

$$+ e^{-\beta_t} \sum_n w_t(n)$$

What term(s) must we optimize?

We need to minimize the entire objective function with respect to 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:

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

We then need to compute the weights for the above classifier as:

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

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

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

Meta-Algorithm

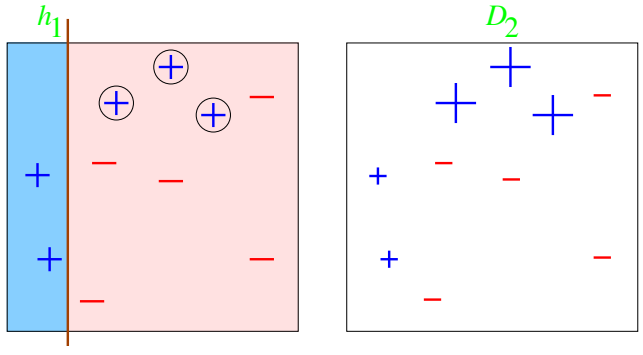
Note that the AdaBoost algorithm itself never specifies how we would get $h_t^*(\boldsymbol{x})$ as long as it minimizes the weighted classification error

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

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

We can use decision stumps

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!

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 (mathematically, it should maximize the gradient of the residuals calculated with this loss function).
  - 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 step size (β) of each learner update.

Ensemble Methods: Summary

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

Bagging trains several classifiers on random subsets of the training data. Random forests train several decision trees that are constrained to split on random subsets of data *features*.

- Prevents some correlation between trees due to dominant features.
- All classifiers are weighted equally in making the final decision.

Boosting sequentially adds weak classifiers, increasing the weight of "hard" data points at each step.

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