CSCI567 Machine Learning (Fall 2025)

Haipeng Luo

University of Southern California

Oct 3, 2025

Exam 1 Logistics

Date: Friday, Oct 17th

Time: 2:00-4:00pm

Location: will be announced on Piazza

Exam 1 Logistics

Date: Friday, Oct 17th

Time: 2:00-4:00pm

Location: will be announced on Piazza

Individual effort, close-book (no cheat sheet), no calculators or any other electronics, *but need your phone to upload your solutions to Gradescope from 4:00-4:20pm*

Exam 1 Coverage

Coverage: Lectures 1-6.

Exam 1 Coverage

Coverage: Lectures 1-6.

Five problems in total

Exam 1 Coverage

Coverage: Lectures 1-6.

Five problems in total

- one problem of 15 multiple-choice *multiple-answer* questions

Exam 1 Coverage

Coverage: Lectures 1-6.

Five problems in total

- one problem of 15 multiple-choice *multiple-answer* questions
  - 0.5 point for selecting (not selecting) each correct (incorrect) answer

Exam 1 Coverage

Coverage: Lectures 1-6.

Five problems in total

- one problem of 15 multiple-choice *multiple-answer* questions
  - 0.5 point for selecting (not selecting) each correct (incorrect) answer
  - "which of the following is correct?" *does not imply one correct answer*

Exam 1 Coverage

Coverage: Lectures 1-6.

Five problems in total

- one problem of 15 multiple-choice *multiple-answer* questions
  - 0.5 point for selecting (not selecting) each correct (incorrect) answer
  - "which of the following is correct?" *does not imply one correct answer*
- four other homework-like problems, each has a couple sub-problems

Exam 1 Coverage

Coverage: Lectures 1-6.

Five problems in total

- one problem of 15 multiple-choice *multiple-answer* questions
  - 0.5 point for selecting (not selecting) each correct (incorrect) answer
  - "which of the following is correct?" *does not imply one correct answer*
- four other homework-like problems, each has a couple sub-problems
  - linear regression, linear classifiers, backpropagation, kernel, SVM

Tips

Best way to prepare: focus on the **sample exam** (on course website)!

Tips

Best way to prepare: focus on the **sample exam** (on course website)!

- make sure to fully understand those problems (solutions to be posted next week)

Tips

Best way to prepare: focus on the **sample exam** (on course website)!

- make sure to fully understand those problems (solutions to be posted next week)
- no need to remember most formulas from the lectures

Tips

Best way to prepare: focus on the **sample exam** (on course website)!

- make sure to fully understand those problems (solutions to be posted next week)
- no need to remember most formulas from the lectures
- expect to see *variants of these questions*

Tips

Best way to prepare: focus on the **sample exam** (on course website)!

- make sure to fully understand those problems (solutions to be posted next week)
- no need to remember most formulas from the lectures
- expect to see *variants of these questions*
- sample exam might appear challenging, but hopefully the actual one is less so given the practice from the sample exam

Outline

Outline

Support vector machines (SVM)

- most commonly used classification algorithms before deep learning
- works well with the kernel trick
- strong theoretical guarantees

Support vector machines (SVM)

- most commonly used classification algorithms before deep learning
- works well with the kernel trick
- strong theoretical guarantees

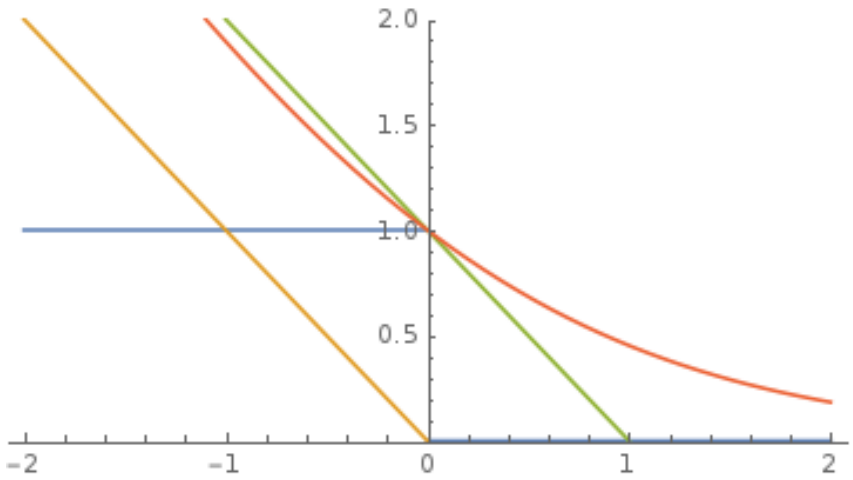
We focus on **binary classification** here.

Primal formulation

In one sentence: linear model with L2 regularized hinge loss.

Primal formulation

In one sentence: linear model with L2 regularized hinge loss. Recall



- perceptron loss $\ell_{\text{perceptron}}(z) = \max\{0, -z\} \rightarrow$ Perceptron
- logistic loss $\ell_{\text{logistic}}(z) = \log(1 + \exp(-z)) \rightarrow$ logistic regression
- hinge loss $\ell_{\text{hinge}}(z) = \max\{0, 1 - z\} \rightarrow$ **SVM**

Primal formulation

For a linear model $(\boldsymbol{w}, b)$, this means

$$\min_{\boldsymbol{w},b} \sum_n \max\left\{0, 1 - y_n(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b)\right\} + \frac{\lambda}{2}\|\boldsymbol{w}\|_2^2$$

Primal formulation

For a linear model $(\boldsymbol{w}, b)$, this means

$$\min_{\boldsymbol{w},b} \sum_n \max\left\{0, 1 - y_n(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b)\right\} + \frac{\lambda}{2}\|\boldsymbol{w}\|_2^2$$

- recall $y_n \in \{-1, +1\}$

Primal formulation

For a linear model $(\boldsymbol{w}, b)$, this means

$$\min_{\boldsymbol{w},b} \sum_n \max\left\{0, 1 - y_n(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b)\right\} + \frac{\lambda}{2}\|\boldsymbol{w}\|_2^2$$

- recall $y_n \in \{-1, +1\}$
- a nonlinear mapping $\boldsymbol{\phi}$ is applied

Primal formulation

For a linear model $(\boldsymbol{w}, b)$, this means

$$\min_{\boldsymbol{w},b} \sum_n \max\left\{0, 1 - y_n(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b)\right\} + \frac{\lambda}{2}\|\boldsymbol{w}\|_2^2$$

- recall $y_n \in \{-1, +1\}$
- a nonlinear mapping $\boldsymbol{\phi}$ is applied
- the bias/intercept term b is used explicitly (think about why after this lecture)

Primal formulation

For a linear model $(\boldsymbol{w}, b)$, this means

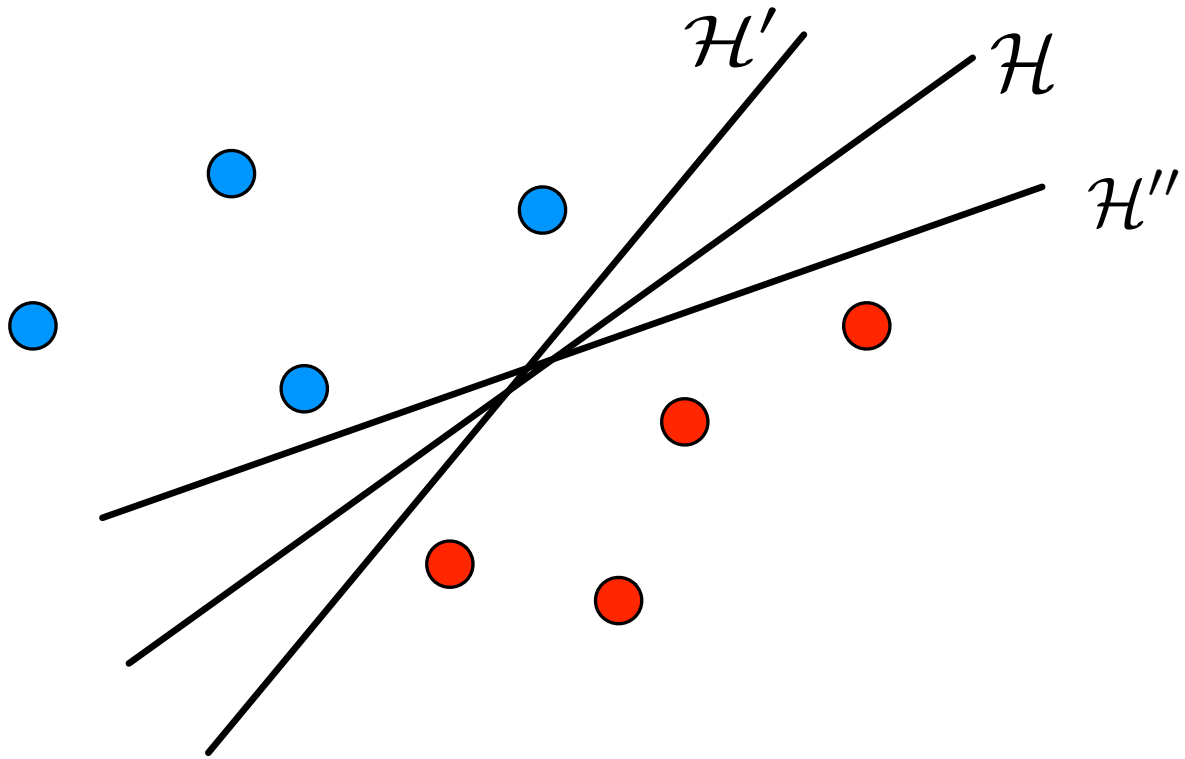
$$\min_{\boldsymbol{w},b} \sum_n \max\left\{0, 1 - y_n(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b)\right\} + \frac{\lambda}{2}\|\boldsymbol{w}\|_2^2$$

- recall $y_n \in \{-1, +1\}$
- a nonlinear mapping $\boldsymbol{\phi}$ is applied
- the bias/intercept term b is used explicitly (think about why after this lecture)

So why L2 regularized hinge loss?

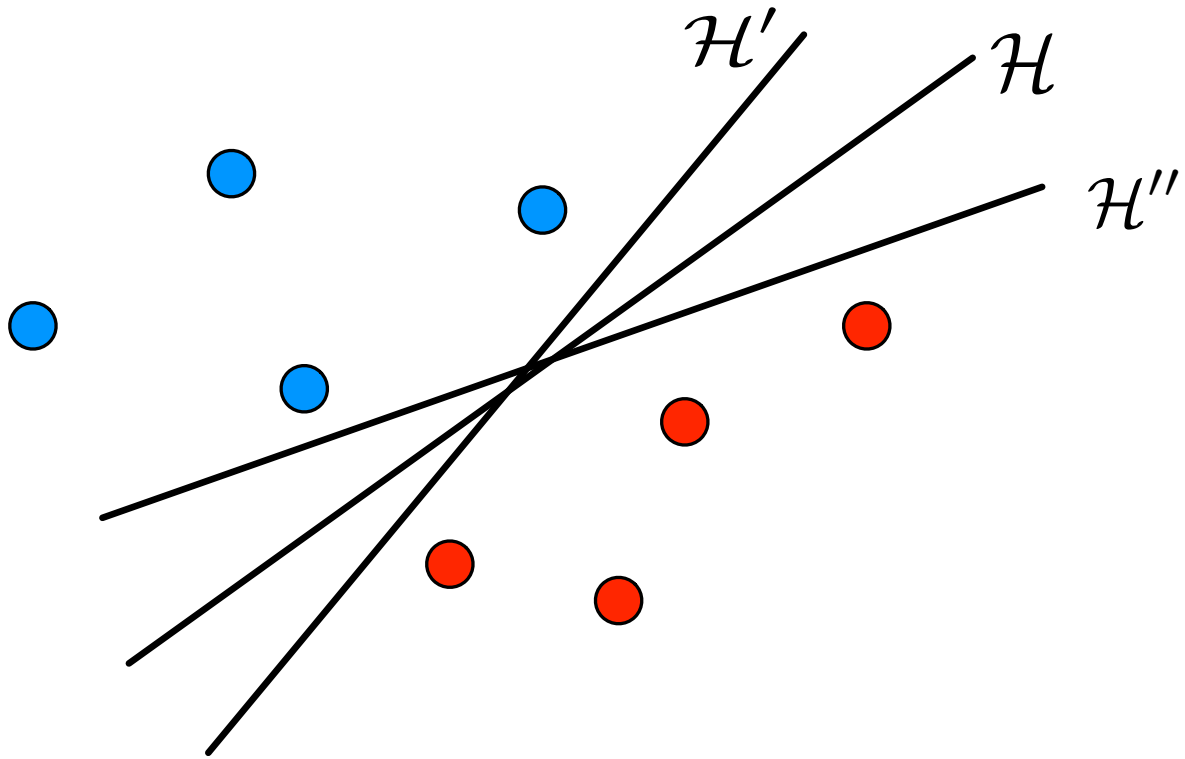
Geometric motivation: separable case

When data is **linearly separable**, there are *infinitely many hyperplanes with zero training error*:



Geometric motivation: separable case

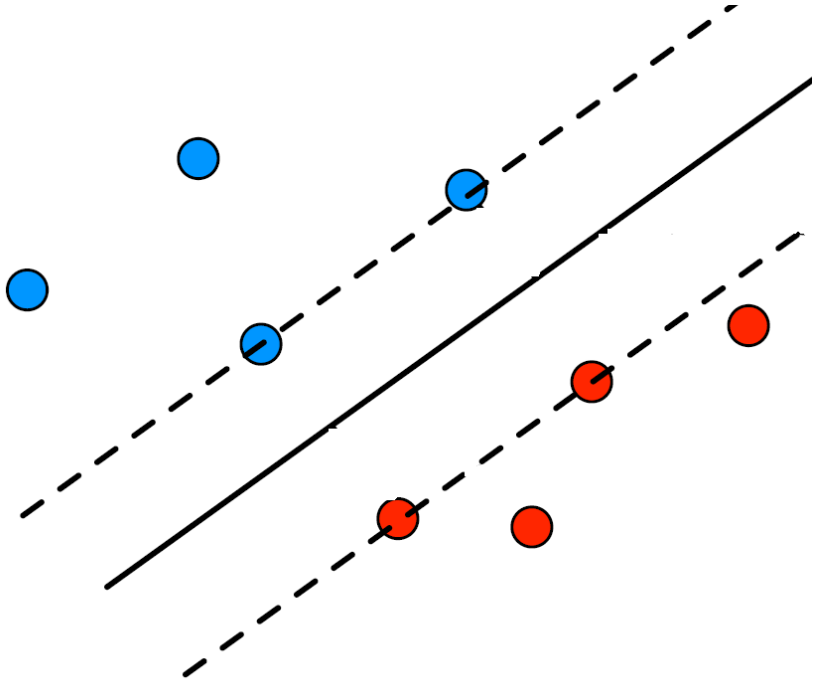
When data is **linearly separable**, there are *infinitely many hyperplanes with zero training error*:



So which one should we choose?

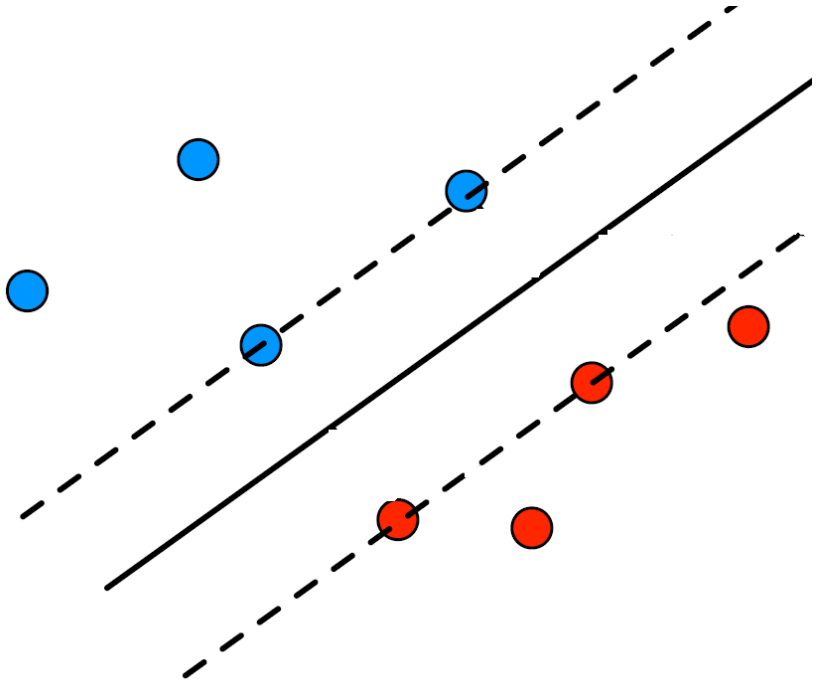
Intuition

The further away from data points the better.



Intuition

The further away from data points the better.



How to formalize this intuition?

Distance to hyperplane

What is the **distance** from a point $\boldsymbol{x}$ to a hyperplane $\{\boldsymbol{x} : \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b = 0\}$?

Distance to hyperplane

What is the **distance** from a point $\boldsymbol{x}$ to a hyperplane $\{\boldsymbol{x} : \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b = 0\}$?

Assume the **projection** is $\boldsymbol{x} - \ell \frac{\boldsymbol{w}}{\|\boldsymbol{w}\|_2}$,

Distance to hyperplane

What is the **distance** from a point $\boldsymbol{x}$ to a hyperplane $\{\boldsymbol{x} : \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b = 0\}$?

Assume the **projection** is $\boldsymbol{x} - \ell \frac{\boldsymbol{w}}{\|\boldsymbol{w}\|_2}$, then

$$0 = \boldsymbol{w}^{\mathrm{T}} \left(\boldsymbol{x} - \ell \frac{\boldsymbol{w}}{\|\boldsymbol{w}\|_2} \right) + b = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} - \ell \|\boldsymbol{w}\| + b$$

and thus $\ell = \frac{\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}+b}{\|\boldsymbol{w}\|_2}$.

Distance to hyperplane

What is the **distance** from a point $\boldsymbol{x}$ to a hyperplane $\{\boldsymbol{x} : \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b = 0\}$?

Assume the **projection** is $\boldsymbol{x} - \ell \frac{\boldsymbol{w}}{\|\boldsymbol{w}\|_2}$, then

$$0 = \boldsymbol{w}^{\mathrm{T}} \left(\boldsymbol{x} - \ell \frac{\boldsymbol{w}}{\|\boldsymbol{w}\|_2} \right) + b = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} - \ell \|\boldsymbol{w}\| + b$$

and thus $\ell = \frac{\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}+b}{\|\boldsymbol{w}\|_2}$.

Therefore the distance is

$$\frac{|\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b|}{\|\boldsymbol{w}\|_2}$$

Distance to hyperplane

What is the **distance** from a point $\boldsymbol{x}$ to a hyperplane $\{\boldsymbol{x} : \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b = 0\}$?

Assume the **projection** is $\boldsymbol{x} - \ell \frac{\boldsymbol{w}}{\|\boldsymbol{w}\|_2}$, then

$$0 = \boldsymbol{w}^{\mathrm{T}} \left(\boldsymbol{x} - \ell \frac{\boldsymbol{w}}{\|\boldsymbol{w}\|_2} \right) + b = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} - \ell \|\boldsymbol{w}\| + b$$

and thus $\ell = \frac{\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}+b}{\|\boldsymbol{w}\|_2}$.

Therefore the distance is

$$\frac{|\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b|}{\|\boldsymbol{w}\|_2}$$

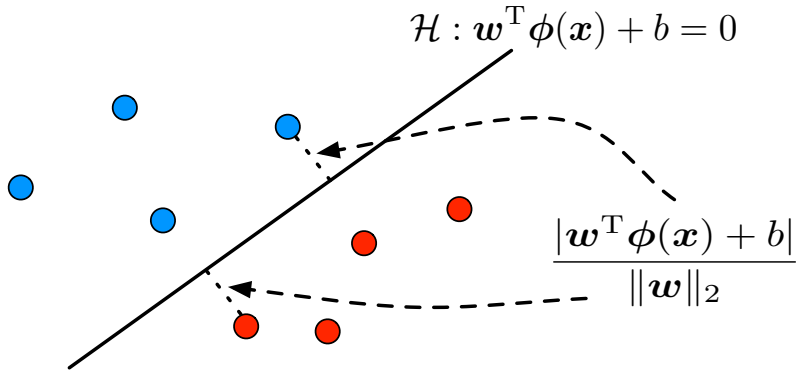
For a hyperplane that correctly classifies $(\boldsymbol{x}, y)$, the distance becomes

$$\frac{y(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b)}{\|\boldsymbol{w}\|_2}$$

Maximizing margin

Margin: the *smallest* distance from all training points to the hyperplane

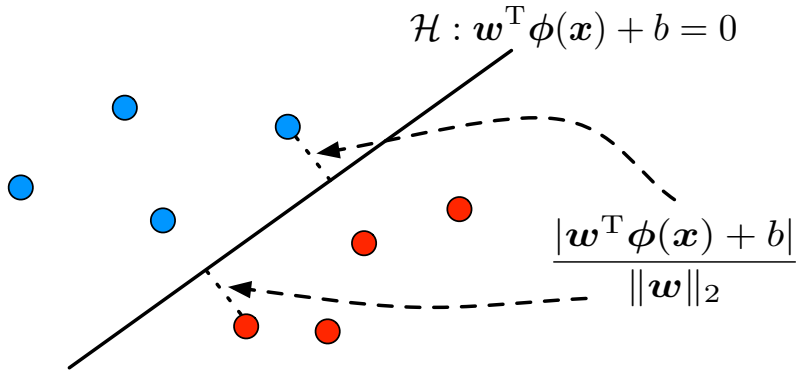
$$\text{MARGIN OF } (\boldsymbol{w},\, b) = \min_n \frac{y_n(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b)}{\|\boldsymbol{w}\|_2}$$



Maximizing margin

Margin: the *smallest* distance from all training points to the hyperplane

$$\text{MARGIN OF } (\boldsymbol{w},\, b) = \min_n \frac{y_n(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b)}{\|\boldsymbol{w}\|_2}$$



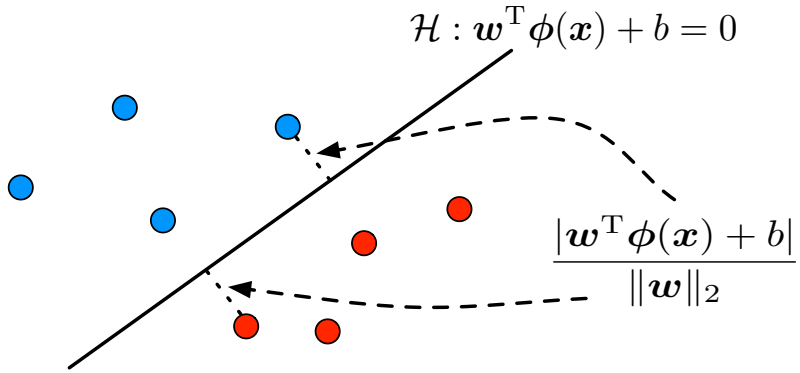
The intuition "**the further away the better**" translates to solving

$$\max_{\boldsymbol{w},b} \quad \min_n \frac{y_n(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b)}{\|\boldsymbol{w}\|_2}$$

Maximizing margin

Margin: the *smallest* distance from all training points to the hyperplane

$$\text{MARGIN OF } (\boldsymbol{w},\, b) = \min_n \frac{y_n(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b)}{\|\boldsymbol{w}\|_2}$$



The intuition "**the further away the better**" translates to solving

$$\max_{\boldsymbol{w},b} \min_n \frac{y_n(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b)}{\|\boldsymbol{w}\|_2} = \max_{\boldsymbol{w},b} \frac{1}{\|\boldsymbol{w}\|_2} \min_n y_n(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b)$$

Rescaling

Note: rescaling $(\boldsymbol{w}, b)$ does not change the hyperplane at all.

Rescaling

Note: rescaling $(\boldsymbol{w}, b)$ does not change the hyperplane at all.

We can thus always scale $(\boldsymbol{w}, b)$ s.t. $\min_n y_n(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b) = 1$

Rescaling

Note: rescaling $(\boldsymbol{w}, b)$ does not change the hyperplane at all.

We can thus always scale $(\boldsymbol{w}, b)$ s.t. $\min_n y_n(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b) = 1$

The margin then becomes

$$\begin{aligned}
&\text{MARGIN OF } (\boldsymbol{w},\ b) \\
&= \frac{1}{\|\boldsymbol{w}\|_2} \min_n y_n(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b) \\
&= \frac{1}{\|\boldsymbol{w}\|_2}
\end{aligned}$$

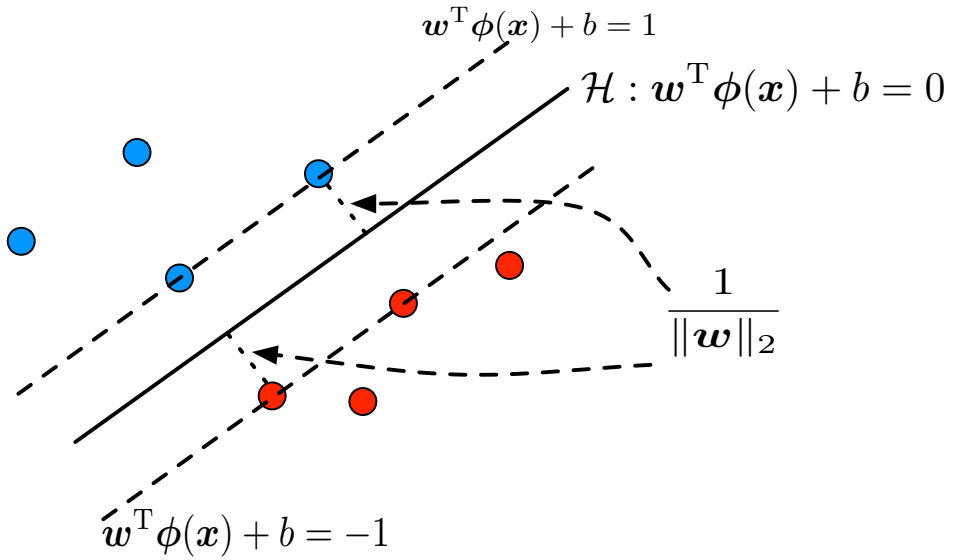
Rescaling

Note: rescaling $(\boldsymbol{w}, b)$ does not change the hyperplane at all.

We can thus always scale $(\boldsymbol{w}, b)$ s.t. $\min_n y_n(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b) = 1$

The margin then becomes

$$\begin{aligned}&\text{MARGIN OF } (\boldsymbol{w},\ b)\\ &= \frac{1}{\|\boldsymbol{w}\|_2} \min_n y_n(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b)\\ &= \frac{1}{\|\boldsymbol{w}\|_2}\end{aligned}$$



Summary for separable data

For a separable training set, we aim to solve

$$\max_{\boldsymbol{w},b} \frac{1}{\|\boldsymbol{w}\|_2} \quad \text{s.t.} \quad \min_n y_n(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b) = 1$$

Summary for separable data

For a separable training set, we aim to solve

$$\max_{\boldsymbol{w},b} \frac{1}{\|\boldsymbol{w}\|_2} \quad \text{s.t.} \quad \min_n y_n(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b) = 1$$

This is equivalent to

$$\begin{aligned} \min_{\boldsymbol{w},b} \quad & \frac{1}{2}\|\boldsymbol{w}\|_2^2 \\ \text{s.t.} \quad & y_n(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b) \geq 1, \quad \forall\, n \end{aligned}$$

Summary for separable data

For a separable training set, we aim to solve

$$\max_{\boldsymbol{w},b} \frac{1}{\|\boldsymbol{w}\|_2} \quad \text{s.t.} \quad \min_n y_n(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b) = 1$$

This is equivalent to

$$\begin{aligned} \min_{\boldsymbol{w},b} \quad & \frac{1}{2}\|\boldsymbol{w}\|_2^2 \\ \text{s.t.} \quad & y_n(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b) \geq 1, \quad \forall\, n \end{aligned}$$

SVM is thus also called *max-margin* classifier. The constraints above are called *hard-margin* constraints.

General non-separable case

If data is not linearly separable, the previous constraint

$$y_n(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b) \geq 1, \quad \forall\ n$$

is obviously *not feasible*.

General non-separable case

If data is not linearly separable, the previous constraint

$$y_n(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b) \geq 1, \quad \forall\ n$$

is obviously *not feasible*.

To deal with this issue, we relax them to **soft-margin** constraints:

$$y_n(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b) \geq 1 - \xi_n, \quad \forall\ n$$

where we introduce **slack variables** $\xi_n \geq 0$.

SVM Primal formulation

We want ξ_n to be as small as possible too.

SVM Primal formulation

We want ξ_n to be as small as possible too. The objective becomes

$$\begin{aligned} \min_{\boldsymbol{w}, b, \{\xi_n\}} \quad & \frac{1}{2}\|\boldsymbol{w}\|_2^2 + C\sum_n \xi_n \\ \text{s.t.} \quad & y_n(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b) \geq 1 - \xi_n, \quad \forall\ n \\ & \xi_n \geq 0, \quad \forall\ n \end{aligned}$$

where C is a hyperparameter to balance the two goals.

Equivalent form

Formulation

$$\begin{aligned} \min_{\boldsymbol{w},b,\{\xi_n\}} \quad & C\sum_n \xi_n + \frac{1}{2}\|\boldsymbol{w}\|_2^2 \\ \text{s.t.} \quad & 1 - y_n(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b) \le \xi_n, \quad \forall\ n \\ & \xi_n \ge 0, \quad \forall\ n \end{aligned}$$

Equivalent form

Formulation

$$\begin{aligned} \min_{\boldsymbol{w},b,\{\xi_n\}} \quad & C\sum_n \xi_n + \frac{1}{2}\|\boldsymbol{w}\|_2^2 \\ \text{s.t.} \quad & 1 - y_n(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b) \leq \xi_n, \quad \forall\ n \\ & \xi_n \geq 0, \quad \forall\ n \end{aligned}$$

is equivalent to

$$\begin{aligned} \min_{\boldsymbol{w},b,\{\xi_n\}} \quad & C\sum_n \xi_n + \frac{1}{2}\|\boldsymbol{w}\|_2^2 \\ \text{s.t.} \quad & \max\left\{0, 1 - y_n(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b)\right\} = \xi_n, \quad \forall\ n \end{aligned}$$

Equivalent form

$$\begin{aligned} \min_{\boldsymbol{w},b,\{\xi_n\}} \quad & C\sum_n \xi_n + \frac{1}{2}\|\boldsymbol{w}\|_2^2 \\ \text{s.t.} \quad & \max\left\{0, 1 - y_n(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b)\right\} = \xi_n, \quad \forall\ n \end{aligned}$$

is equivalent to

$$\min_{\boldsymbol{w},b}\ C\sum_n \max\left\{0, 1 - y_n(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b)\right\} + \frac{1}{2}\|\boldsymbol{w}\|_2^2$$

Equivalent form

$$\min_{\boldsymbol{w},b,\{\xi_n\}} \quad C\sum_n \xi_n + \frac{1}{2}\|\boldsymbol{w}\|_2^2$$
$$\text{s.t.} \quad \max\left\{0, 1 - y_n(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b)\right\} = \xi_n, \quad \forall\ n$$

is equivalent to

$$\min_{\boldsymbol{w},b}\ C\sum_n \max\left\{0, 1 - y_n(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b)\right\} + \frac{1}{2}\|\boldsymbol{w}\|_2^2$$

and

$$\min_{\boldsymbol{w},b}\ \sum_n \max\left\{0, 1 - y_n(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b)\right\} + \frac{\lambda}{2}\|\boldsymbol{w}\|_2^2$$

with $\lambda = 1/C$.

Equivalent form

$$\min_{\boldsymbol{w},b,\{\xi_n\}} \quad C\sum_n \xi_n + \frac{1}{2}\|\boldsymbol{w}\|_2^2$$
$$\text{s.t.} \quad \max\left\{0, 1 - y_n(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b)\right\} = \xi_n, \quad \forall\ n$$

is equivalent to

$$\min_{\boldsymbol{w},b}\ C\sum_n \max\left\{0, 1 - y_n(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b)\right\} + \frac{1}{2}\|\boldsymbol{w}\|_2^2$$

and

$$\min_{\boldsymbol{w},b}\ \sum_n \max\left\{0, 1 - y_n(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b)\right\} + \frac{\lambda}{2}\|\boldsymbol{w}\|_2^2$$

with $\lambda = 1/C$. *This is exactly minimizing L2 regularized hinge loss!*

Optimization

$$\begin{aligned} \min_{\boldsymbol{w},b,\{\xi_n\}} \quad & C\sum_n \xi_n + \frac{1}{2}\|\boldsymbol{w}\|_2^2 \\ \text{s.t.} \quad & 1 - y_n(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b) \leq \xi_n, \quad \forall\ n \\ & \xi_n \geq 0, \quad \forall\ n \end{aligned}$$

- It is a convex (**quadratic** in fact) problem

Optimization

$$\begin{aligned} \min_{\boldsymbol{w},b,\{\xi_n\}} \quad & C\sum_n \xi_n + \frac{1}{2}\|\boldsymbol{w}\|_2^2 \\ \text{s.t.} \quad & 1 - y_n(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b) \le \xi_n, \quad \forall\ n \\ & \xi_n \ge 0, \quad \forall\ n \end{aligned}$$

- It is a convex (**quadratic** in fact) problem
- thus can apply any convex optimization algorithms, e.g. SGD

Optimization

$$\begin{aligned} \min_{\boldsymbol{w},b,\{\xi_n\}} \quad & C\sum_n \xi_n + \frac{1}{2}\|\boldsymbol{w}\|_2^2 \\ \text{s.t.} \quad & 1 - y_n(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b) \le \xi_n, \quad \forall\ n \\ & \xi_n \ge 0, \quad \forall\ n \end{aligned}$$

- It is a convex (**quadratic** in fact) problem
- thus can apply any convex optimization algorithms, e.g. SGD
- there are **more specialized and efficient** algorithms

Optimization

$$\begin{aligned} \min_{\boldsymbol{w},b,\{\xi_n\}} \quad & C\sum_n \xi_n + \frac{1}{2}\|\boldsymbol{w}\|_2^2 \\ \text{s.t.} \quad & 1 - y_n(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b) \le \xi_n, \quad \forall\ n \\ & \xi_n \ge 0, \quad \forall\ n \end{aligned}$$

- It is a convex (**quadratic** in fact) problem
- thus can apply any convex optimization algorithms, e.g. SGD
- there are **more specialized and efficient** algorithms
- but usually we apply kernel trick, which requires solving the *dual problem*

The dual formulation

Similar to Perceptron, it turns out that the primal solution $\boldsymbol{w}^*$ is also a **linear combination of data**

$$\boldsymbol{w}^* = \sum_{n=1}^{\mathsf{N}} \alpha_n^* y_n \boldsymbol{\phi}(\boldsymbol{x}_n)$$

The dual formulation

Similar to Perceptron, it turns out that the primal solution $\boldsymbol{w}^*$ is also a **linear combination of data**

$$\boldsymbol{w}^* = \sum_{n=1}^{\mathsf{N}} \alpha_n^* y_n \boldsymbol{\phi}(\boldsymbol{x}_n)$$

where $\alpha_1^*, \ldots, \alpha_{\mathsf{N}}^*$ are solutions of the **dual formulation of SVM**:

$$\begin{aligned} \max_{\alpha_1,\ldots,\alpha_{\mathsf{N}}} \quad & \sum_n \alpha_n - \frac{1}{2} \sum_{m,n} y_m y_n \alpha_m \alpha_n \boldsymbol{\phi}(\boldsymbol{x}_m)^{\mathrm{T}} \boldsymbol{\phi}(\boldsymbol{x}_n) \\ \text{s.t.} \quad & \sum_n \alpha_n y_n = 0 \quad \text{and} \quad 0 \leq \alpha_n \leq C, \quad \forall\ n \end{aligned}$$

The dual formulation

Similar to Perceptron, it turns out that the primal solution $\boldsymbol{w}^*$ is also a **linear combination of data**

$$\boldsymbol{w}^* = \sum_{n=1}^{\mathsf{N}} \alpha_n^* y_n \boldsymbol{\phi}(\boldsymbol{x}_n)$$

where $\alpha_1^*, \ldots, \alpha_{\mathsf{N}}^*$ are solutions of the **dual formulation of SVM**:

$$\begin{aligned} \max_{\alpha_1,\ldots,\alpha_{\mathsf{N}}} \quad & \sum_n \alpha_n - \frac{1}{2} \sum_{m,n} y_m y_n \alpha_m \alpha_n \boldsymbol{\phi}(\boldsymbol{x}_m)^{\mathrm{T}} \boldsymbol{\phi}(\boldsymbol{x}_n) \\ \text{s.t.} \quad & \sum_n \alpha_n y_n = 0 \quad \text{and} \quad 0 \leq \alpha_n \leq C, \quad \forall\ n \end{aligned}$$

- a **quadratic program**, many efficient optimization algorithms exist

The dual formulation

Similar to Perceptron, it turns out that the primal solution $\boldsymbol{w}^*$ is also a **linear combination of data**

$$\boldsymbol{w}^* = \sum_{n=1}^{\mathsf{N}} \alpha_n^* y_n \boldsymbol{\phi}(\boldsymbol{x}_n)$$

where $\alpha_1^*, \ldots, \alpha_{\mathsf{N}}^*$ are solutions of the **dual formulation of SVM**:

$$\begin{aligned} \max_{\alpha_1,\ldots,\alpha_{\mathsf{N}}} \quad & \sum_n \alpha_n - \frac{1}{2} \sum_{m,n} y_m y_n \alpha_m \alpha_n \boldsymbol{\phi}(\boldsymbol{x}_m)^{\mathrm{T}} \boldsymbol{\phi}(\boldsymbol{x}_n) \\ \text{s.t.} \quad & \sum_n \alpha_n y_n = 0 \quad \text{and} \quad 0 \leq \alpha_n \leq C, \quad \forall\ n \end{aligned}$$

- a **quadratic program**, many efficient optimization algorithms exist
- immediately *kernelizable* by replacing $\boldsymbol{\phi}(\boldsymbol{x}_m)^{\mathrm{T}} \boldsymbol{\phi}(\boldsymbol{x}_n)$ with $k(\boldsymbol{x}_m, \boldsymbol{x}_n)$

Making a prediction

How to efficiently make a prediction $\text{SGN}\left(\boldsymbol{w}^{*\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}) + b^*\right)$ for a new $\boldsymbol{x}$?

Making a prediction

How to efficiently make a prediction $\text{SGN}\left(\boldsymbol{w}^{*\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}) + b^*\right)$ for a new $\boldsymbol{x}$?

- first term

$$\boldsymbol{w}^{*\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}) = \sum_{n=1}^{\mathrm{N}} \alpha_n^* y_n \boldsymbol{\phi}(\boldsymbol{x}_n)^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x})$$

Making a prediction

How to efficiently make a prediction $\text{SGN}\left(\boldsymbol{w}^{*\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}) + b^*\right)$ for a new $\boldsymbol{x}$?

- first term

$$\boldsymbol{w}^{*\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}) = \sum_{n=1}^{\mathrm{N}} \alpha_n^* y_n \boldsymbol{\phi}(\boldsymbol{x}_n)^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}) = \sum_{n=1}^{\mathrm{N}} \alpha_n^* y_n k(\boldsymbol{x}_n, \boldsymbol{x})$$

Making a prediction

How to efficiently make a prediction $\text{SGN}\left(\boldsymbol{w}^{*\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}) + b^*\right)$ for a new $\boldsymbol{x}$?

- first term

$$\boldsymbol{w}^{*\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}) = \sum_{n=1}^{\mathrm{N}} \alpha_n^* y_n \boldsymbol{\phi}(\boldsymbol{x}_n)^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}) = \sum_{n=1}^{\mathrm{N}} \alpha_n^* y_n k(\boldsymbol{x}_n, \boldsymbol{x})$$

- second term (derivation omitted):

$$b^* = y_m - \boldsymbol{w}^{*\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_m)$$

for *any* m *such that* $0 < \alpha_m^* < C$.

Making a prediction

How to efficiently make a prediction $\text{SGN}\left(\boldsymbol{w}^{*\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}) + b^*\right)$ for a new $\boldsymbol{x}$?

- first term

$$\boldsymbol{w}^{*\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}) = \sum_{n=1}^{\mathrm{N}} \alpha_n^* y_n \boldsymbol{\phi}(\boldsymbol{x}_n)^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}) = \sum_{n=1}^{\mathrm{N}} \alpha_n^* y_n k(\boldsymbol{x}_n, \boldsymbol{x})$$

- second term (derivation omitted):

$$b^* = y_m - \boldsymbol{w}^{*\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_m) = y_m - \sum_{n=1}^{\mathrm{N}} \alpha_n^* y_n k(\boldsymbol{x}_n, \boldsymbol{x}_m)$$

for *any* m *such that* $0 < \alpha_m^* < C$.

Making a prediction

How to efficiently make a prediction $\text{SGN}\left(\boldsymbol{w}^{*\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}) + b^*\right)$ for a new $\boldsymbol{x}$?

- first term

$$\boldsymbol{w}^{*\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}) = \sum_{n=1}^{\mathrm{N}} \alpha_n^* y_n \boldsymbol{\phi}(\boldsymbol{x}_n)^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}) = \sum_{n=1}^{\mathrm{N}} \alpha_n^* y_n k(\boldsymbol{x}_n, \boldsymbol{x})$$

- second term (derivation omitted):

$$b^* = y_m - \boldsymbol{w}^{*\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_m) = y_m - \sum_{n=1}^{\mathrm{N}} \alpha_n^* y_n k(\boldsymbol{x}_n, \boldsymbol{x}_m)$$

for *any* m *such that* $0 < \alpha_m^* < C$. (b^* should be precomputed)

A closer look at α_n

Observe:

$$\boldsymbol{w}^* = \sum_{n=1}^{\mathsf{N}} \alpha_n^* y_n \boldsymbol{\phi}(\boldsymbol{x}_n)$$

A closer look at α_n

Observe:

$$\boldsymbol{w}^* = \sum_{n=1}^{\mathsf{N}} \alpha_n^* y_n \boldsymbol{\phi}(\boldsymbol{x}_n) = \sum_{n:\alpha_n^*>0} \alpha_n^* y_n \boldsymbol{\phi}(\boldsymbol{x}_n)$$

A closer look at α_n

Observe:

$$\boldsymbol{w}^* = \sum_{n=1}^{\mathsf{N}} \alpha_n^* y_n \boldsymbol{\phi}(\boldsymbol{x}_n) = \sum_{n:\alpha_n^*>0} \alpha_n^* y_n \boldsymbol{\phi}(\boldsymbol{x}_n)$$

A point with $\alpha_n^* > 0$ is called a "**support vector**".

A closer look at α_n

Observe:

$$\boldsymbol{w}^* = \sum_{n=1}^{\mathsf{N}} \alpha_n^* y_n \boldsymbol{\phi}(\boldsymbol{x}_n) = \sum_{n:\alpha_n^*>0} \alpha_n^* y_n \boldsymbol{\phi}(\boldsymbol{x}_n)$$

A point with $\alpha_n^* > 0$ is called a "**support vector**".

Hence the name **Support Vector Machine** (SVM).

Geometric interpretation of support vectors

A support vector satisfies $\alpha_n^* > 0$ and

$$1 - \xi_n^* - y_n(\boldsymbol{w}^{*\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b^*) = 0$$

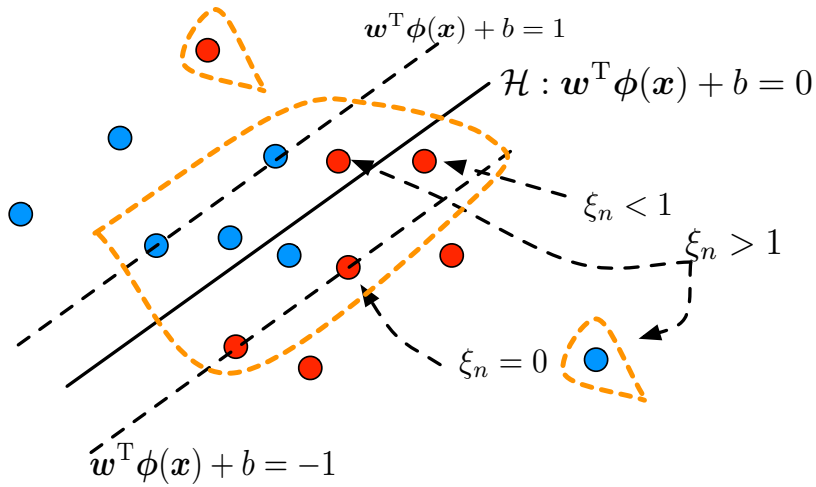
Geometric interpretation of support vectors

A support vector satisfies $\alpha_n^* > 0$ and

$$1 - \xi_n^* - y_n(\boldsymbol{w}^{*\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b^*) = 0$$

When

- $\xi_n^* = 0$, $y_n(\boldsymbol{w}^{*\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b^*) = 1$ and thus the point is $1/\|\boldsymbol{w}^*\|_2$ away from the hyperplane.



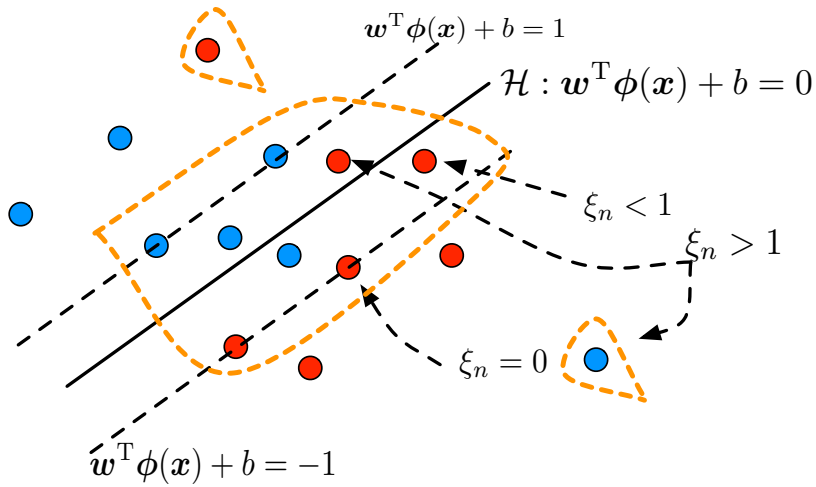
Geometric interpretation of support vectors

A support vector satisfies $\alpha_n^* > 0$ and

$$1 - \xi_n^* - y_n(\boldsymbol{w}^{*\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b^*) = 0$$

When

- $\xi_n^* = 0$, $y_n(\boldsymbol{w}^{*\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b^*) = 1$ and thus the point is $1/\|\boldsymbol{w}^*\|_2$ away from the hyperplane.
- $\xi_n^* < 1$, the point is classified correctly but does not satisfy the large margin constraint.



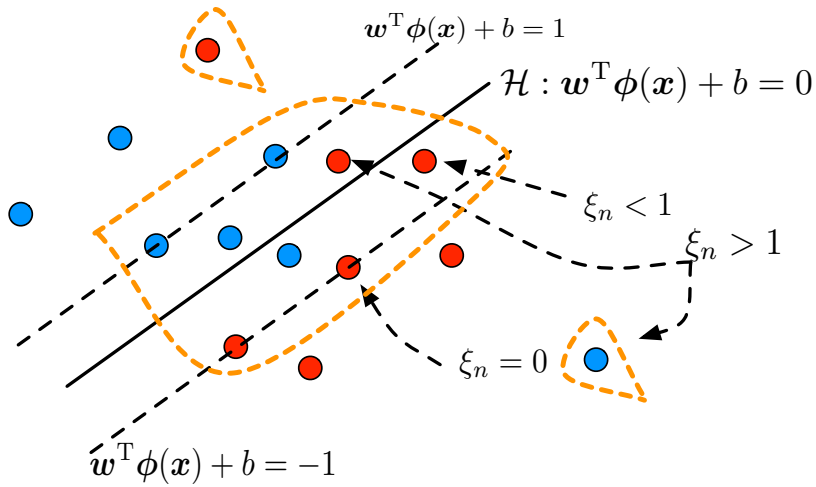
Geometric interpretation of support vectors

A support vector satisfies $\alpha_n^* > 0$ and

$$1 - \xi_n^* - y_n(\boldsymbol{w}^{*\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b^*) = 0$$

When

- $\xi_n^* = 0$, $y_n(\boldsymbol{w}^{*\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b^*) = 1$ and thus the point is $1/\|\boldsymbol{w}^*\|_2$ away from the hyperplane.
- $\xi_n^* < 1$, the point is classified correctly but does not satisfy the large margin constraint.
- $\xi_n^* > 1$, the point is misclassified.



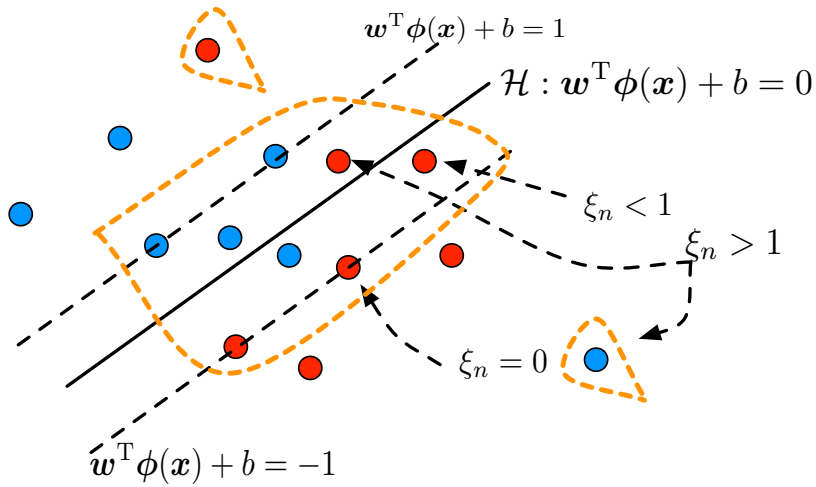
Geometric interpretation of support vectors

A support vector satisfies $\alpha_n^* > 0$ and

$$1 - \xi_n^* - y_n(\boldsymbol{w}^{*\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b^*) = 0$$

When

- $\xi_n^* = 0$, $y_n(\boldsymbol{w}^{*\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b^*) = 1$ and thus the point is $1/\|\boldsymbol{w}^*\|_2$ away from the hyperplane.
- $\xi_n^* < 1$, the point is classified correctly but does not satisfy the large margin constraint.
- $\xi_n^* > 1$, the point is misclassified.



Support vectors (circled with the orange line) are *the only points that matter!*

An example

One drawback of kernel method: **non-parametric**, need to keep all training points potentially

An example

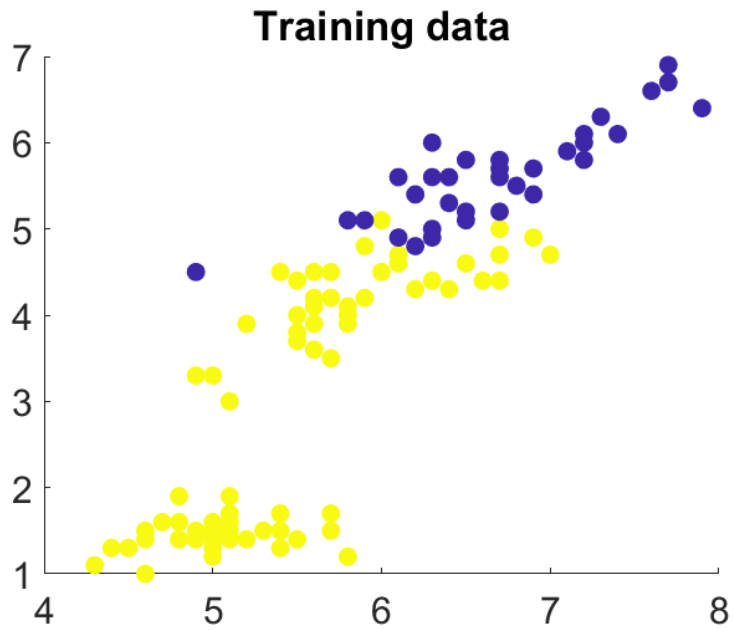
One drawback of kernel method: **non-parametric**, need to keep all training points potentially

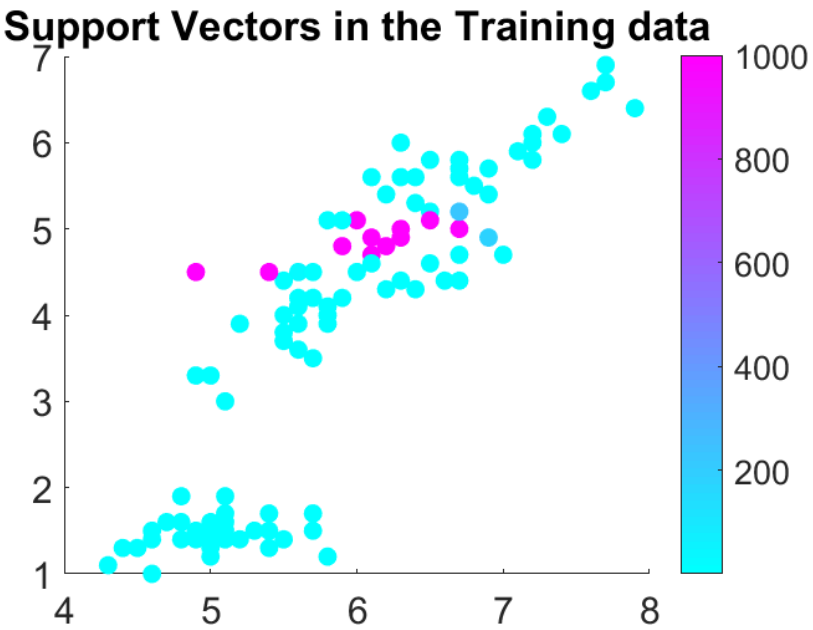
For SVM, very often $\#\text{support vectors} \ll N$

An example

One drawback of kernel method: **non-parametric**, need to keep all training points potentially

For SVM, very often #support vectors $\ll$ N

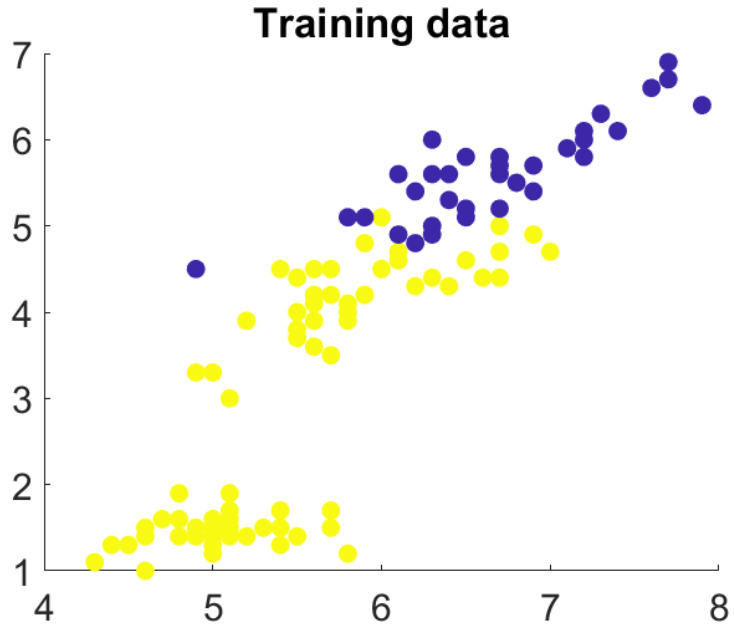


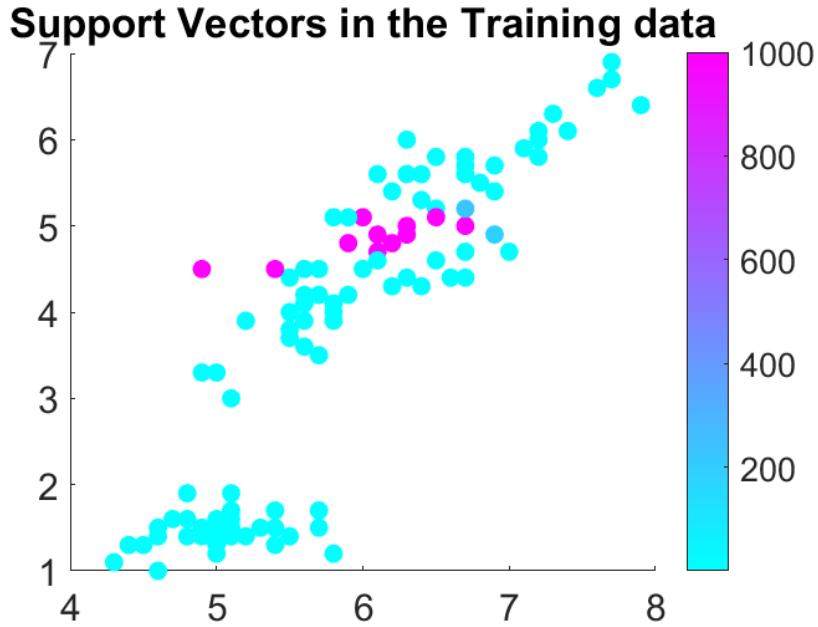


An example

One drawback of kernel method: **non-parametric**, need to keep all training points potentially

For SVM, very often #support vectors $\ll$ N





See also Colab demo.

Summary

SVM: **max-margin linear classifier**

Summary

SVM: **max-margin linear classifier**

Primal (equivalent to minimizing L2 regularized hinge loss):

$$\begin{aligned} \min_{\boldsymbol{w},b,\{\xi_n\}} \quad & C\sum_n \xi_n + \frac{1}{2}\|\boldsymbol{w}\|_2^2 \\ \text{s.t.} \quad & 1 - y_n(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b) \leq \xi_n, \quad \forall\ n \\ & \xi_n \geq 0, \quad \forall\ n \end{aligned}$$

Summary

SVM: **max-margin linear classifier**

Primal (equivalent to minimizing L2 regularized hinge loss):

$$\begin{aligned} \min_{\boldsymbol{w},b,\{\xi_n\}} \quad & C\sum_n \xi_n + \frac{1}{2}\|\boldsymbol{w}\|_2^2 \\ \text{s.t.} \quad & 1 - y_n(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b) \le \xi_n, \quad \forall\ n \\ & \xi_n \ge 0, \quad \forall\ n \end{aligned}$$

Dual (kernelizable, reveals what training points are support vectors):

$$\begin{aligned} \max_{\{\alpha_n\}} \quad & \sum_n \alpha_n - \frac{1}{2}\sum_{m,n} y_m y_n \alpha_m \alpha_n \boldsymbol{\phi}(\boldsymbol{x}_m)^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) \\ \text{s.t.} \quad & \sum_n \alpha_n y_n = 0 \quad \text{and} \quad 0 \le \alpha_n \le C, \quad \forall\ n \end{aligned}$$

Outline

Decision tree

We have seen different ML models for classification/regression:

- linear models, neural nets and other nonlinear models induced by kernels

Decision tree

We have seen different ML models for classification/regression:

- linear models, neural nets and other nonlinear models induced by kernels

Decision tree is yet another one:

Decision tree

We have seen different ML models for classification/regression:

- linear models, neural nets and other nonlinear models induced by kernels

Decision tree is yet another one:

- nonlinear in general

Decision tree

We have seen different ML models for classification/regression:

- linear models, neural nets and other nonlinear models induced by kernels

Decision tree is yet another one:

- nonlinear in general
- works for both classification and regression; we focus on classification

Decision tree

We have seen different ML models for classification/regression:

- linear models, neural nets and other nonlinear models induced by kernels

Decision tree is yet another one:

- nonlinear in general
- works for both classification and regression; we focus on classification
- one key advantage is good interpretability

Decision tree

We have seen different ML models for classification/regression:

- linear models, neural nets and other nonlinear models induced by kernels

Decision tree is yet another one:

- nonlinear in general
- works for both classification and regression; we focus on classification
- one key advantage is good interpretability
- still very popular for small tabular data, especially when used in ensemble (i.e., "forest")

Tree-based models outperform neural nets sometimes

Why do tree-based models still outperform deep learning on tabular data?

Léo Grinsztajn
Soda, Inria Saclay
leo.grinsztajn@inria.fr

Edouard Oyallon
ISIR, CNRS, Sorbonne University

Gaël Varoquaux
Soda, Inria Saclay

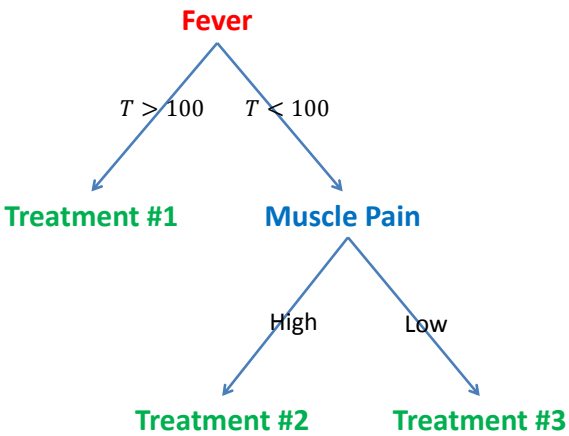
Abstract

While deep learning has enabled tremendous progress on text and image datasets, its superiority on tabular data is not clear. We contribute extensive benchmarks of standard and novel deep learning methods as well as tree-based models such as XGBoost and Random Forests, across a large number of datasets and hyperparameter combinations. We define a standard set of 45 datasets from varied domains with clear characteristics of tabular data and a benchmarking methodology accounting for both fitting models and finding good hyperparameters. Results show that tree-based models remain state-of-the-art on medium-sized data (∼10K samples) even without accounting for their superior speed. To understand this gap, we conduct an empirical investigation into the differing inductive biases of tree-based models and Neural Networks (NNs). This leads to a series of challenges which should guide researchers aiming to build tabular-specific NNs: **1.** be robust to uninformative features, **2.** preserve the orientation of the data, and **3.** be able to easily learn irregular functions. To stimulate research on tabular architectures, we contribute a standard benchmark and raw data for baselines: every point of a 20 000 compute hours hyperparameter search for each learner.

Example

Many decisions are made based on some tree structure

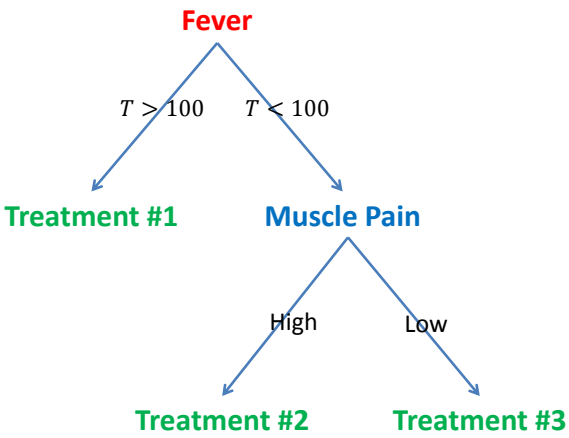
Medical treatment



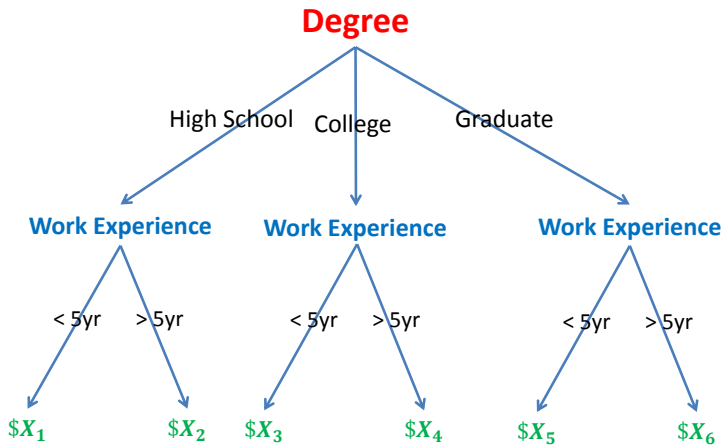
Example

Many decisions are made based on some tree structure

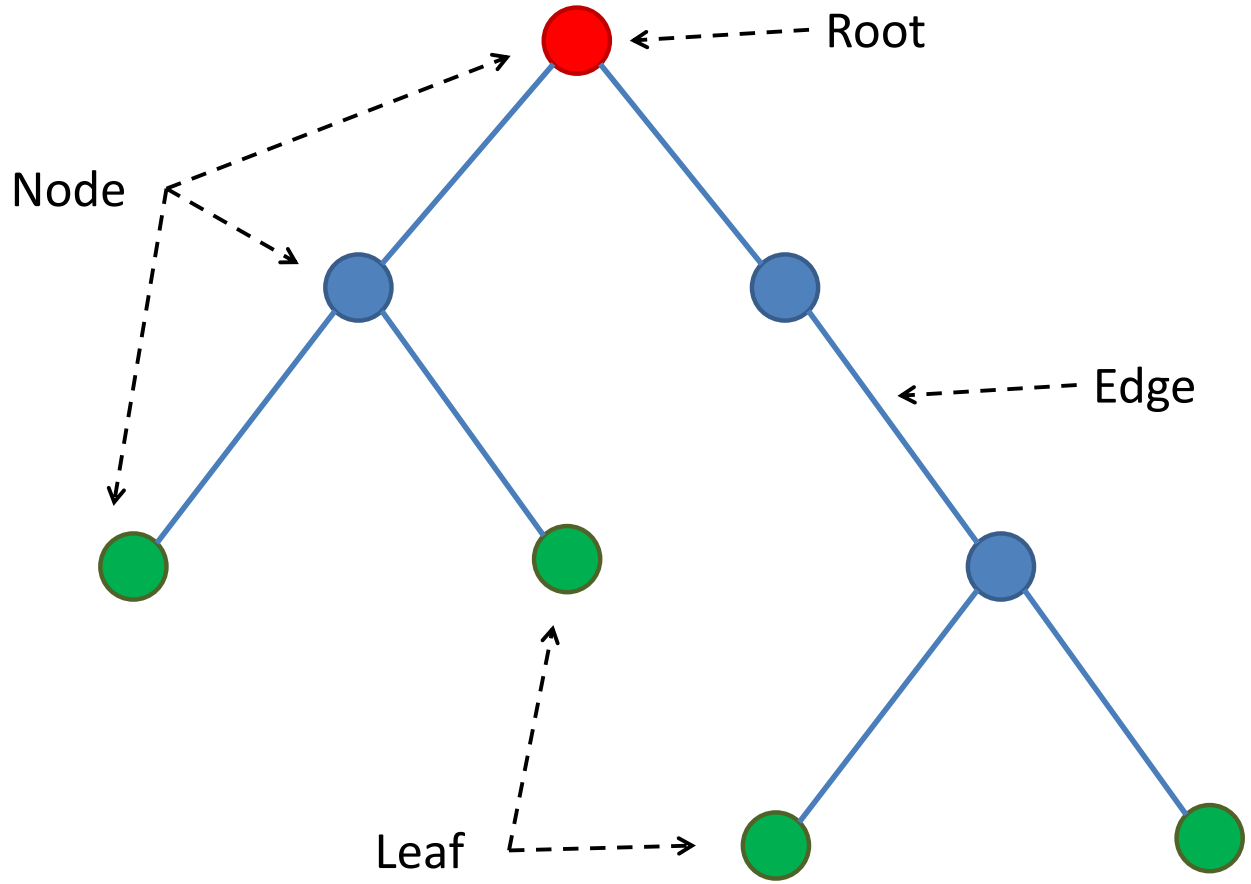
Medical treatment



Salary in a company

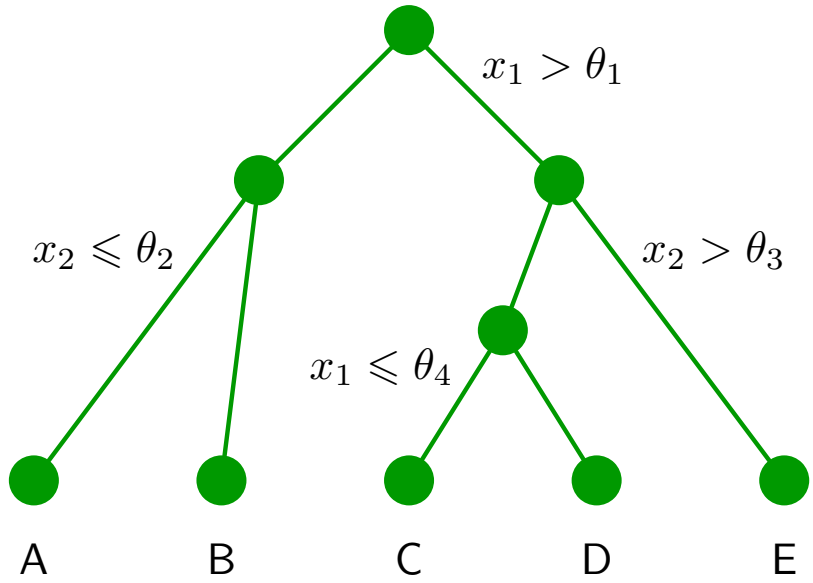


Tree terminology



A more abstract example of decision trees

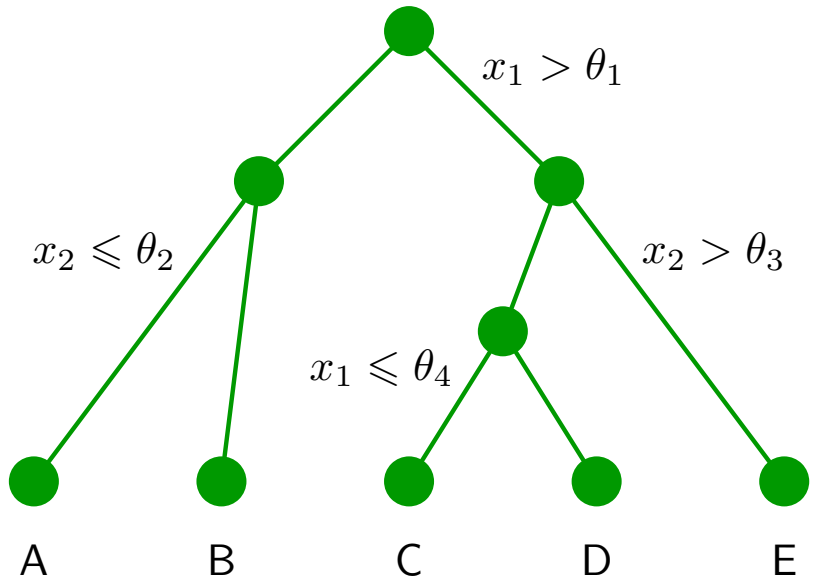
Input: $\boldsymbol{x} = (x_1, x_2)$



A more abstract example of decision trees

Input: $\boldsymbol{x} = (x_1, x_2)$

Output: $f(\boldsymbol{x})$ determined naturally by traversing the tree

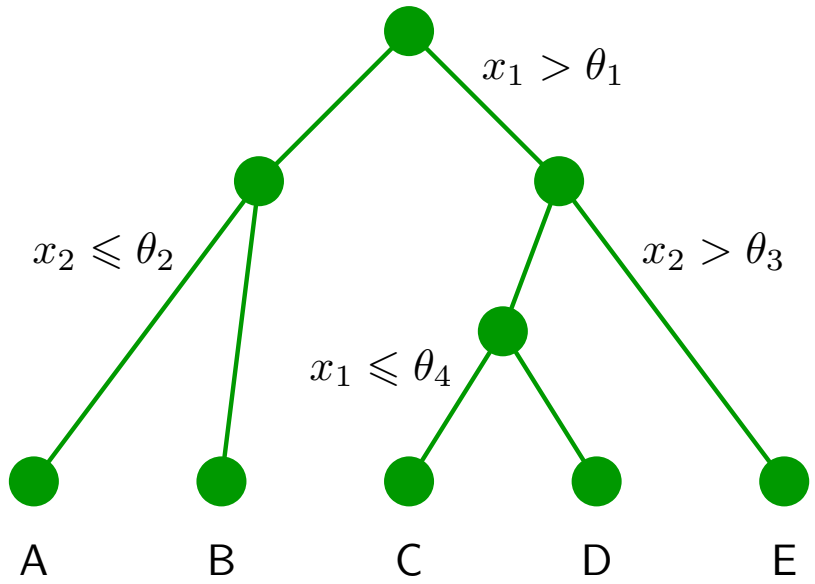


A more abstract example of decision trees

Input: $\boldsymbol{x} = (x_1, x_2)$

Output: $f(\boldsymbol{x})$ determined naturally by traversing the tree

- start from the root

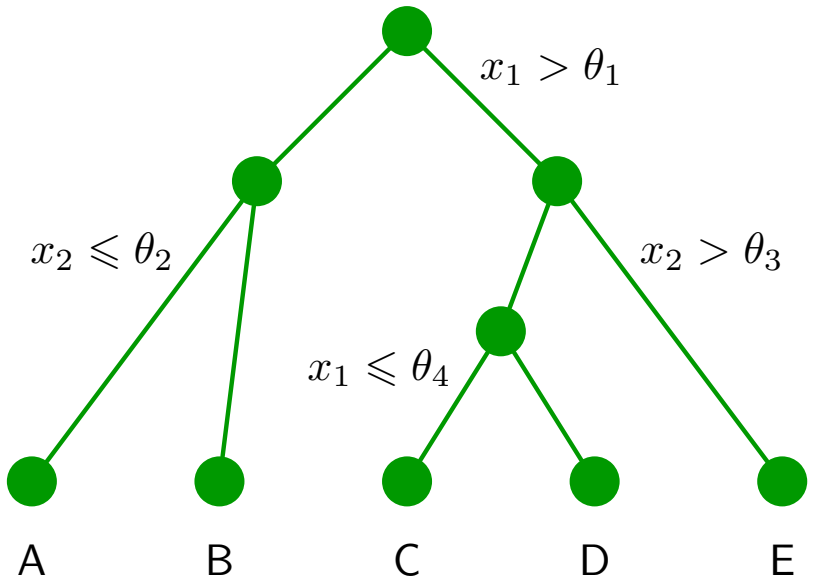


A more abstract example of decision trees

Input: $\boldsymbol{x} = (x_1, x_2)$

Output: $f(\boldsymbol{x})$ determined naturally by traversing the tree

- start from the root
- test at each node to decide which child to visit next

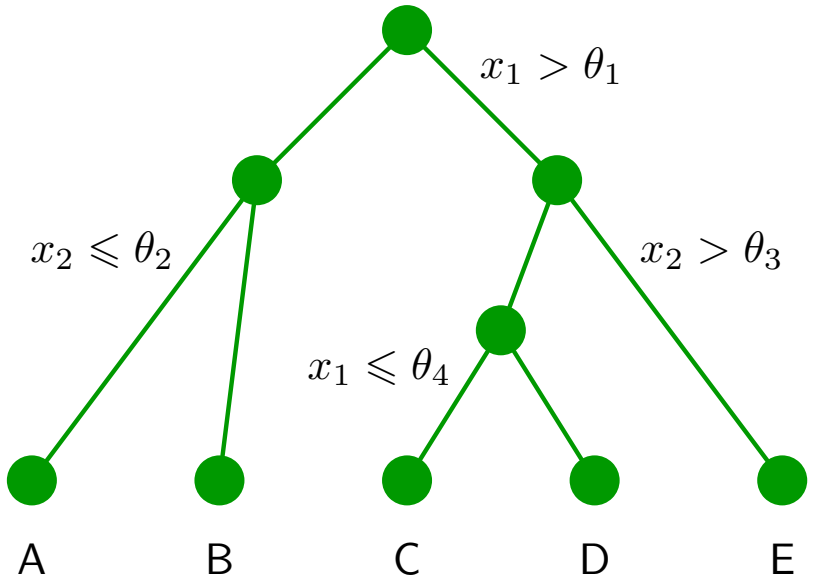


A more abstract example of decision trees

Input: $\boldsymbol{x} = (x_1, x_2)$

Output: $f(\boldsymbol{x})$ determined naturally by traversing the tree

- start from the root
- test at each node to decide which child to visit next
- finally the leaf gives the prediction $f(\boldsymbol{x})$

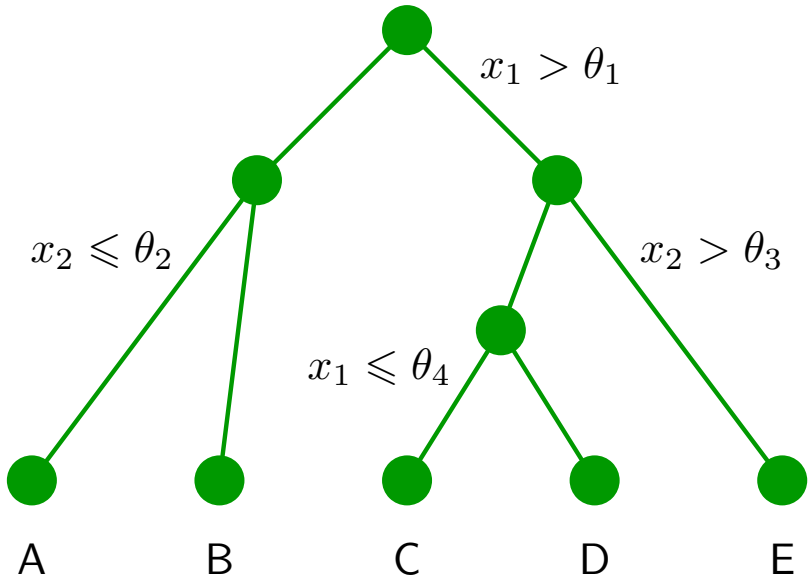


A more abstract example of decision trees

Input: $\boldsymbol{x} = (x_1, x_2)$

Output: $f(\boldsymbol{x})$ determined naturally by traversing the tree

- start from the root
- test at each node to decide which child to visit next
- finally the leaf gives the prediction $f(\boldsymbol{x})$



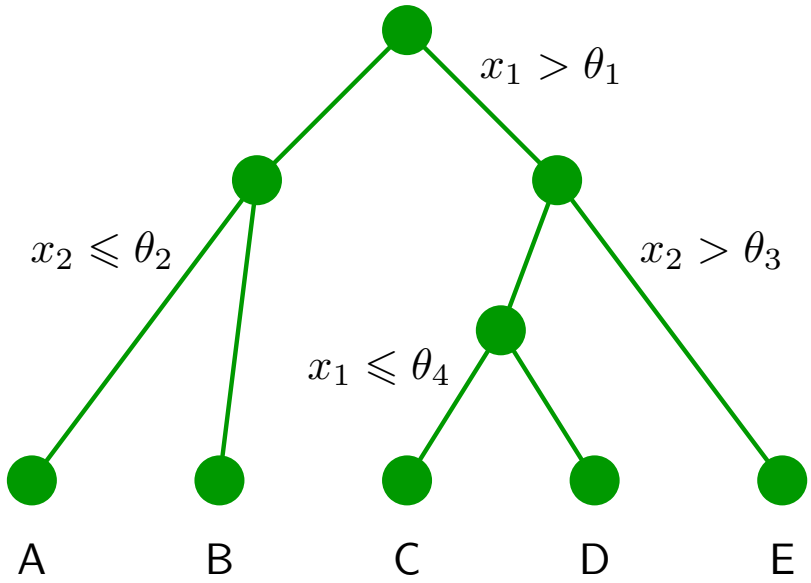
For example, $f((\theta_1 - 1, \theta_2 + 1)) = \mathsf{B}$

A more abstract example of decision trees

Input: $\boldsymbol{x} = (x_1, x_2)$

Output: $f(\boldsymbol{x})$ determined naturally by traversing the tree

- start from the root
- test at each node to decide which child to visit next
- finally the leaf gives the prediction $f(\boldsymbol{x})$

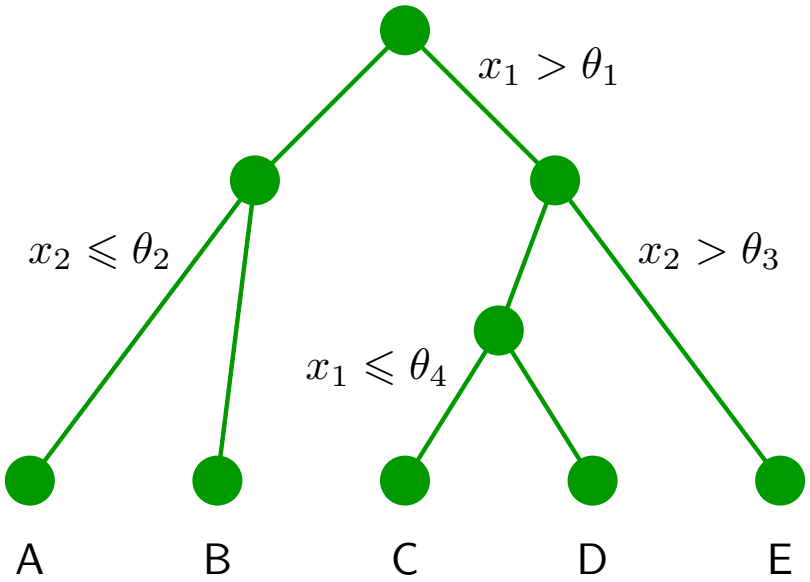


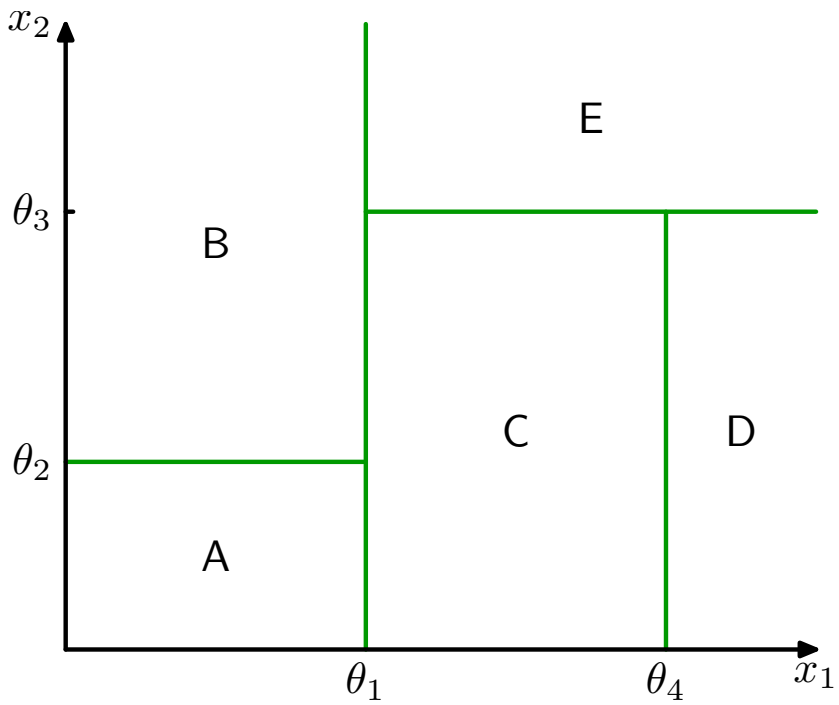
For example, $f((\theta_1 - 1, \theta_2 + 1)) = \text{B}$

Complex to formally write down, but easy to represent pictorially or as codes.

The decision boundary

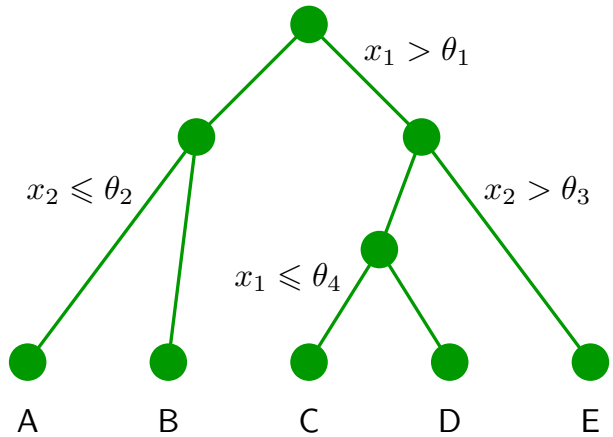
Corresponds to a classifier with boundaries:





Parameters

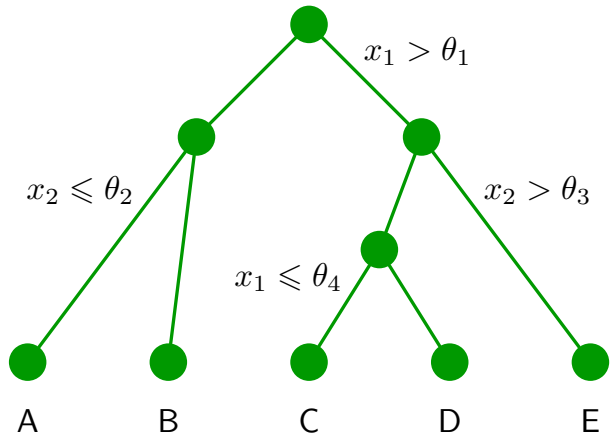
Parameters to learn for a decision tree:



Parameters

Parameters to learn for a decision tree:

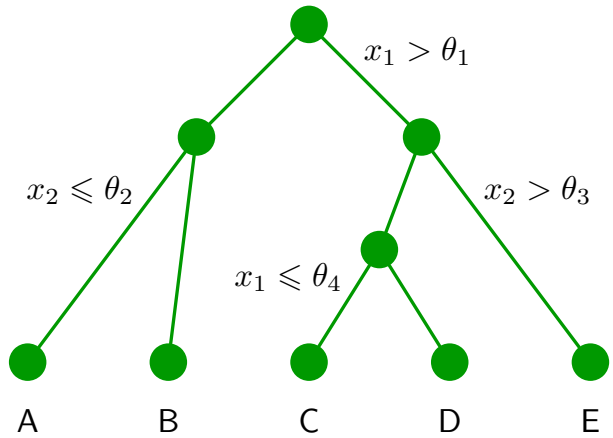
- the structure of the tree, such as the depth, #branches, #nodes, etc



Parameters

Parameters to learn for a decision tree:

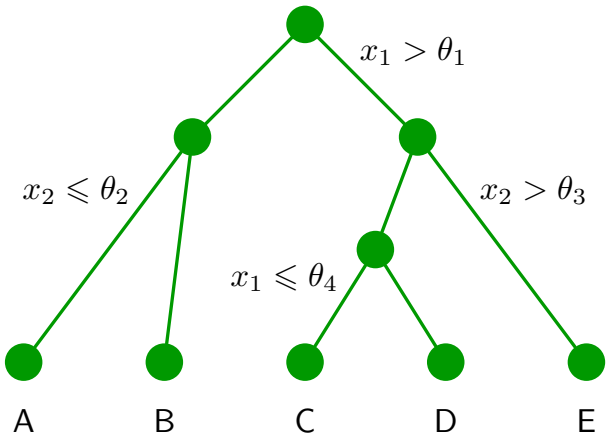
- the structure of the tree, such as the depth, #branches, #nodes, etc
  - some of them are sometimes considered as hyperparameters



Parameters

Parameters to learn for a decision tree:

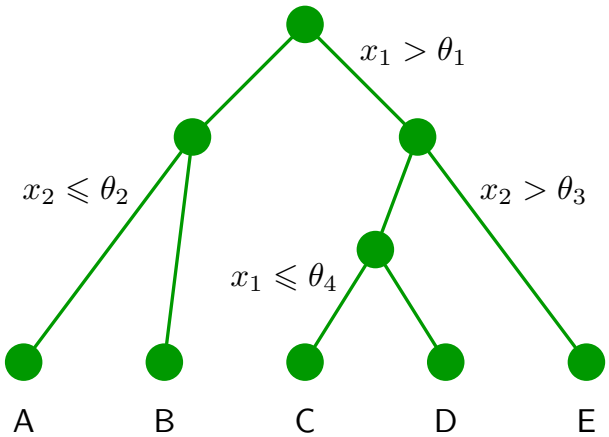
- the structure of the tree, such as the depth, #branches, #nodes, etc
  - some of them are sometimes considered as hyperparameters
  - unlike typical neural nets, the structure of a tree is *not fixed in advance, but learned from data*



Parameters

Parameters to learn for a decision tree:

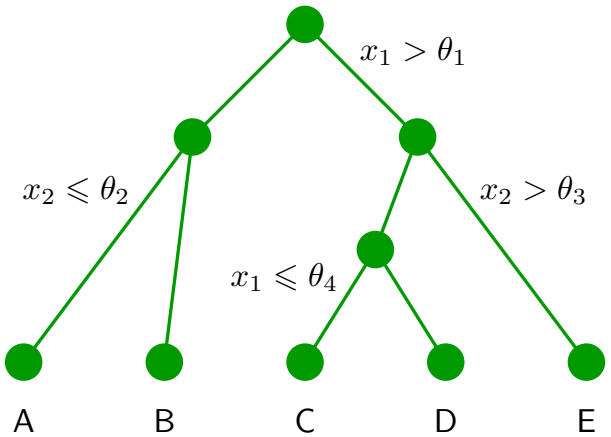
- the structure of the tree, such as the depth, #branches, #nodes, etc
  - some of them are sometimes considered as hyperparameters
  - unlike typical neural nets, the structure of a tree is *not fixed in advance, but learned from data*
- the test at each internal node



Parameters

Parameters to learn for a decision tree:

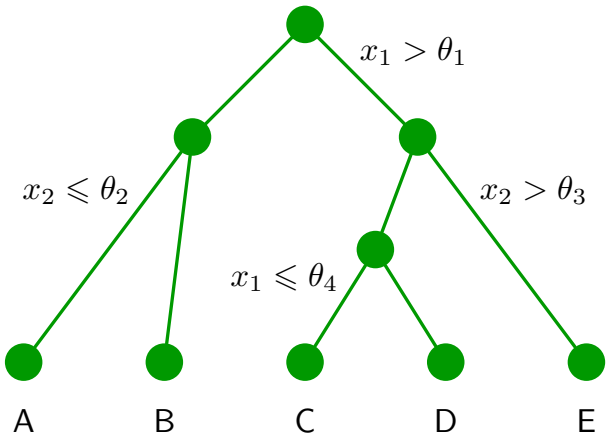
- the structure of the tree, such as the depth, #branches, #nodes, etc
  - some of them are sometimes considered as hyperparameters
  - unlike typical neural nets, the structure of a tree is *not fixed in advance, but learned from data*
- the test at each internal node
  - which feature(s) to test on?



Parameters

Parameters to learn for a decision tree:

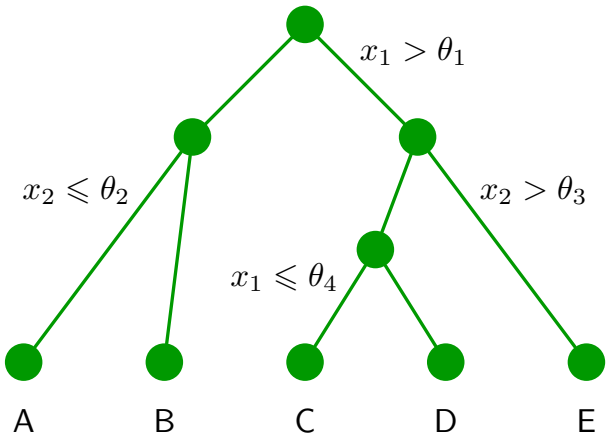
- the structure of the tree, such as the depth, #branches, #nodes, etc
  - some of them are sometimes considered as hyperparameters
  - unlike typical neural nets, the structure of a tree is *not fixed in advance, but learned from data*
- the test at each internal node
  - which feature(s) to test on?
  - if the feature is continuous, what threshold $(\theta_1, \theta_2, \ldots)$?



Parameters

Parameters to learn for a decision tree:

- the structure of the tree, such as the depth, #branches, #nodes, etc
  - some of them are sometimes considered as hyperparameters
  - unlike typical neural nets, the structure of a tree is *not fixed in advance, but learned from data*
- the test at each internal node
  - which feature(s) to test on?
  - if the feature is continuous, what threshold $(\theta_1, \theta_2, \ldots)$?



- the value/prediction of the leaves (A, B, ...)

Learning the parameters

So how do we *learn all these parameters?*

Learning the parameters

So how do we *learn all these parameters?*

Recall typical approach is to find the parameters that minimize some loss.

Learning the parameters

So how do we *learn all these parameters?*

Recall typical approach is to find the parameters that minimize some loss.

This is unfortunately *not feasible for trees*

Learning the parameters

So how do we *learn all these parameters?*

Recall typical approach is to find the parameters that minimize some loss.

This is unfortunately *not feasible for trees*

- For Z nodes, there are roughly $\#\text{features}^Z$ different ways to decide "which feature to test on each node", which is *a lot*.

Learning the parameters

So how do we *learn all these parameters?*

Recall typical approach is to find the parameters that minimize some loss.

This is unfortunately *not feasible for trees*

- For Z nodes, there are roughly $\#\text{features}^Z$ different ways to decide "which feature to test on each node", which is *a lot*.
- enumerating all these configurations to find the one that minimizes some loss is too computationally expensive.

Learning the parameters

So how do we *learn all these parameters?*

Recall typical approach is to find the parameters that minimize some loss.

This is unfortunately *not feasible for trees*

- For Z nodes, there are roughly $\#\text{features}^Z$ different ways to decide "which feature to test on each node", which is *a lot*.
- enumerating all these configurations to find the one that minimizes some loss is too computationally expensive.

Instead, we turn to some **greedy top-down approach**.

A running example

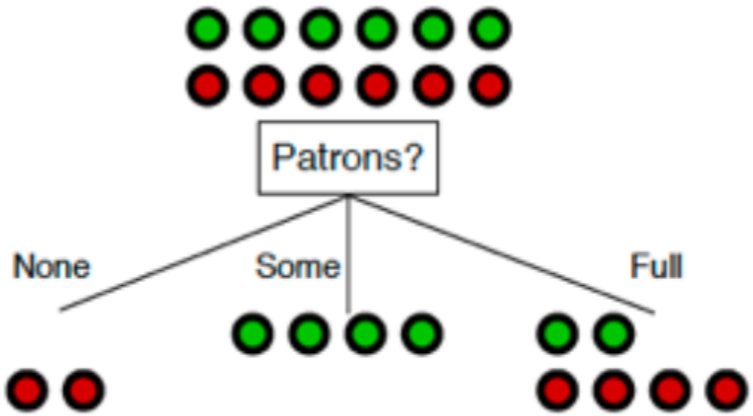
[Russell & Norvig, AIMA]

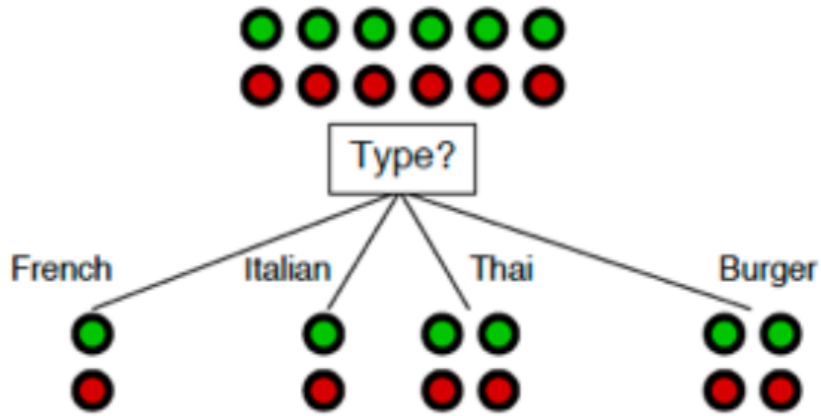
- predict whether a customer will wait for a table at a restaurant
- 12 training examples
- 10 features (all discrete)

Example	Attributes										Target
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
X_1	T	F	F	T	Some	$$$	F	T	French	0–10	T
X_2	T	F	F	T	Full	$	F	F	Thai	30–60	F
X_3	F	T	F	F	Some	$	F	F	Burger	0–10	T
X_4	T	F	T	T	Full	$	F	F	Thai	10–30	T
X_5	T	F	T	F	Full	$$$	F	T	French	>60	F
X_6	F	T	F	T	Some	$$	T	T	Italian	0–10	T
X_7	F	T	F	F	None	$	T	F	Burger	0–10	F
X_8	F	F	F	T	Some	$$	T	T	Thai	0–10	T
X_9	F	T	T	F	Full	$	T	F	Burger	>60	F
X_{10}	T	T	T	T	Full	$$$	F	T	Italian	10–30	F
X_{11}	F	F	F	F	None	$	F	F	Thai	0–10	F
X_{12}	T	T	T	T	Full	$	F	F	Burger	30–60	T

First step: how to build the root?

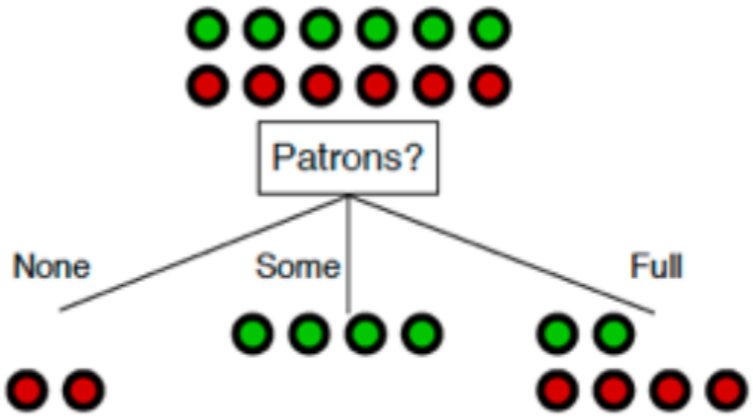
I.e., which feature should we test at the root? Examples:

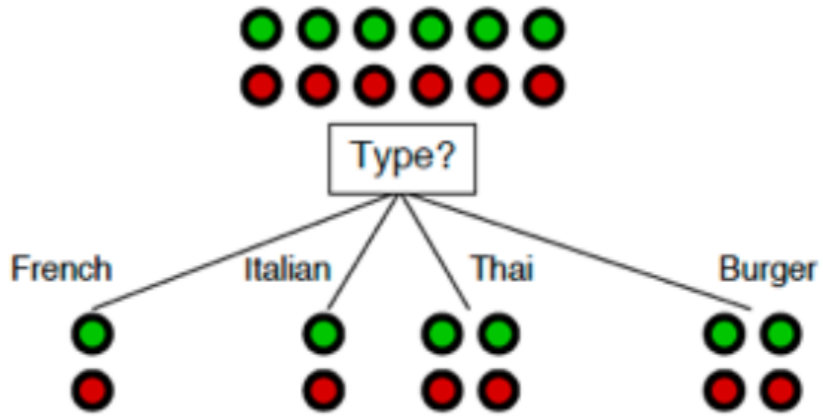




First step: how to build the root?

I.e., which feature should we test at the root? Examples:

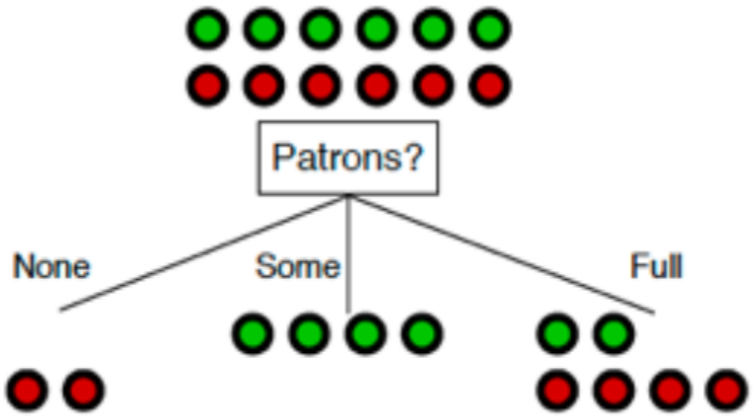


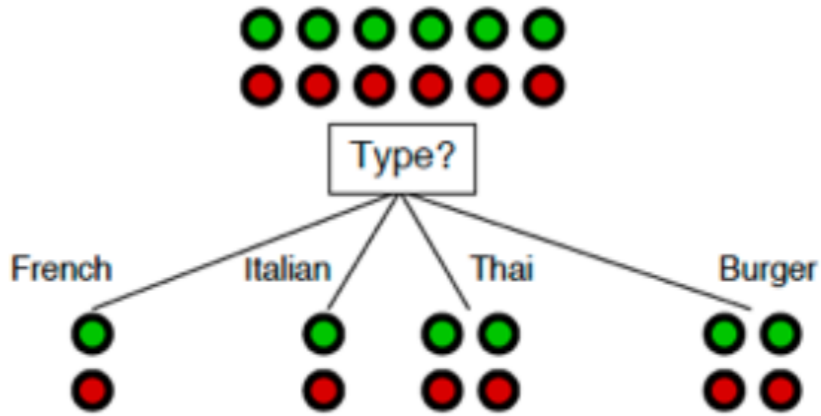


Which split is better?

First step: how to build the root?

I.e., which feature should we test at the root? Examples:



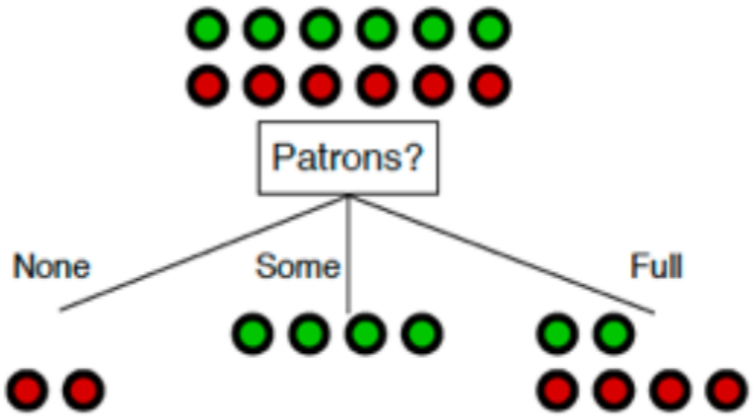


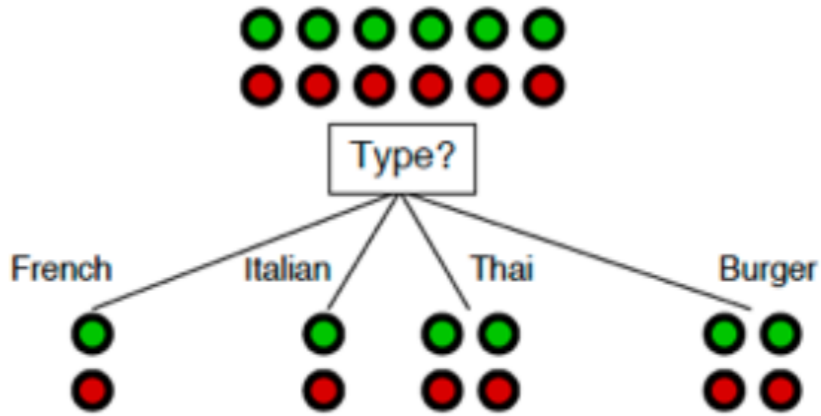
Which split is better?

- intuitively "patrons" is a better feature since it leads to "more pure" or "more certain" children

First step: how to build the root?

I.e., which feature should we test at the root? Examples:





Which split is better?

- intuitively "patrons" is a better feature since it leads to "more pure" or "more certain" children
- how to quantify this intuition?

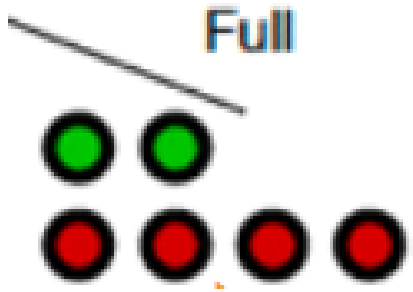
Measure of uncertainty of a node

It should be **a function of the distribution of classes**

Measure of uncertainty of a node

It should be **a function of the distribution of classes**

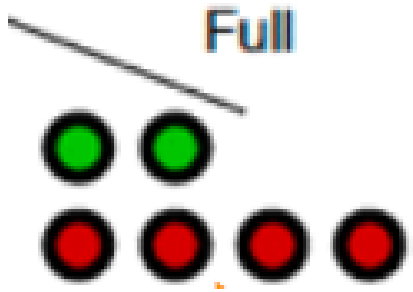
- e.g. a node with 2 positive and 4 negative examples can be summarized by a distribution P with $P(Y = +1) = 1/3$ and $P(Y = -1) = 2/3$



Measure of uncertainty of a node

It should be **a function of the distribution of classes**

- e.g. a node with 2 positive and 4 negative examples can be summarized by a distribution P with $P(Y = +1) = 1/3$ and $P(Y = -1) = 2/3$



One classic uncertainty measure of a distribution is its *(Shannon) entropy*:

$$H(P) = -\sum_{k=1}^{C} P(Y = k) \log P(Y = k)$$

Properties of entropy

$$H(P) = -\sum_{k=1}^{C} P(Y = k) \log P(Y = k)$$

- the base of log can be 2, e or 10

Properties of entropy

$$H(P) = -\sum_{k=1}^{C} P(Y = k) \log P(Y = k)$$

- the base of log can be 2, e or 10
- always non-negative

Properties of entropy

$$H(P) = -\sum_{k=1}^{\mathsf{C}} P(Y = k) \log P(Y = k)$$

- the base of log can be 2, e or 10
- always non-negative
- it's the *smallest codeword length to encode symbols drawn from* P

Properties of entropy

$$H(P) = -\sum_{k=1}^{\mathsf{C}} P(Y = k) \log P(Y = k)$$

- the base of log can be 2, e or 10
- always non-negative
- it's the *smallest codeword length to encode symbols drawn from* P
- maximized if P is uniform ($\max = \log \mathsf{C}$): most uncertain case

Properties of entropy

$$H(P) = -\sum_{k=1}^{\mathsf{C}} P(Y = k) \log P(Y = k)$$

- the base of log can be 2, e or 10
- always non-negative
- it's the *smallest codeword length to encode symbols drawn from* P
- maximized if P is uniform (max $= \log \mathsf{C}$): most uncertain case
- minimized if P focuses on one class (min $= 0$): most certain case
  - e.g. $P = (1, 0, \ldots, 0)$

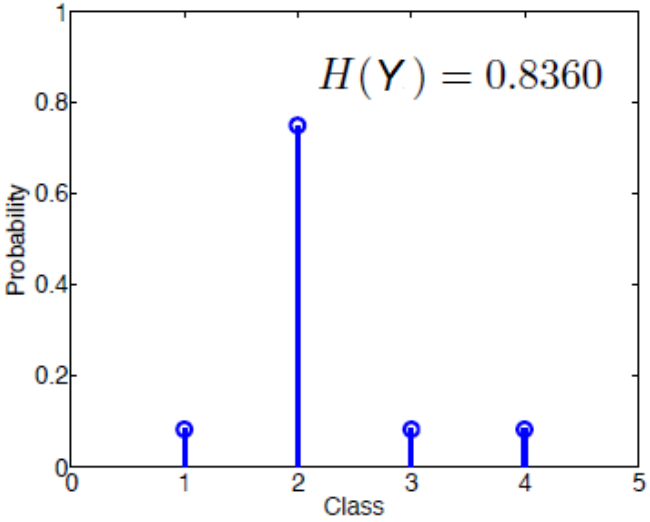
Properties of entropy

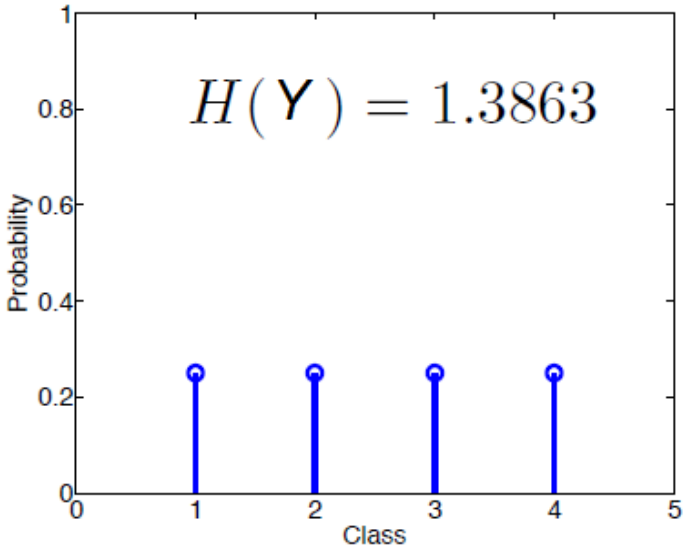
$$H(P) = -\sum_{k=1}^{\mathsf{C}} P(Y = k) \log P(Y = k)$$

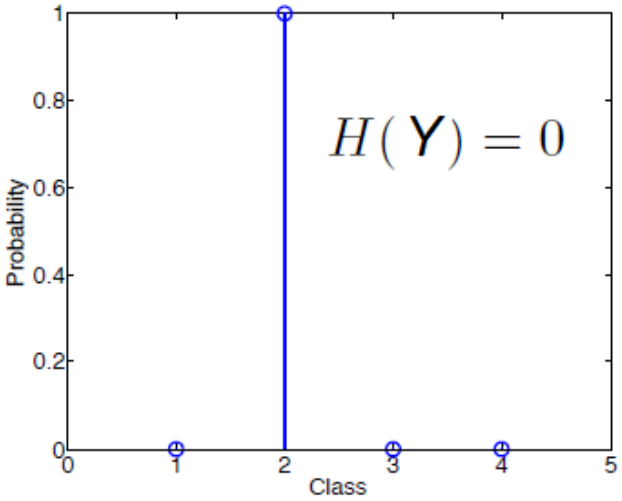
- the base of log can be 2, e or 10
- always non-negative
- it's the *smallest codeword length to encode symbols drawn from* P
- maximized if P is uniform (max $= \log \mathsf{C}$): most uncertain case
- minimized if P focuses on one class (min $= 0$): most certain case
  - e.g. $P = (1, 0, \ldots, 0)$
  - $0 \log 0$ is defined naturally as $\lim_{z \to 0+} z \log z = 0$

Examples of computing entropy

With base e and 4 classes:







Another example

Entropy in each child if root tests on "patrons"

For "None" branch

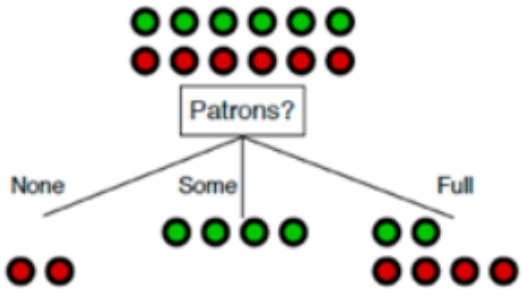
$$-\left(\frac{0}{0+2}\log\frac{0}{0+2}+\frac{2}{0+2}\log\frac{2}{0+2}\right)=0$$

For "Some" branch

$$-\left(\frac{4}{4+0}\log\frac{4}{4+0}+\frac{0}{4+0}\log\frac{0}{4+0}\right)=0$$

For "Full" branch

$$-\left(\frac{2}{2+4}\log\frac{2}{2+4}+\frac{4}{2+4}\log\frac{4}{2+4}\right)\approx 0.9$$



Another example

Entropy in each child if root tests on "patrons"

For "None" branch

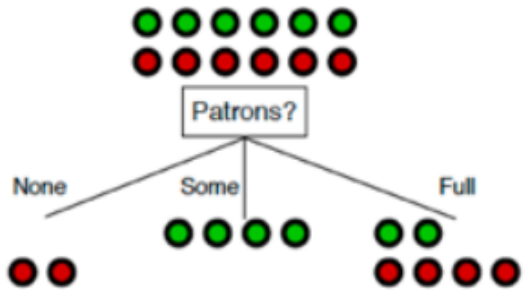
$$-\left(\frac{0}{0+2}\log\frac{0}{0+2}+\frac{2}{0+2}\log\frac{2}{0+2}\right)=0$$

For "Some" branch

$$-\left(\frac{4}{4+0}\log\frac{4}{4+0}+\frac{0}{4+0}\log\frac{0}{4+0}\right)=0$$

For "Full" branch

$$-\left(\frac{2}{2+4}\log\frac{2}{2+4}+\frac{4}{2+4}\log\frac{4}{2+4}\right)\approx 0.9$$



So how good is choosing "patrons" overall?

Another example

Entropy in each child if root tests on "patrons"

For "None" branch

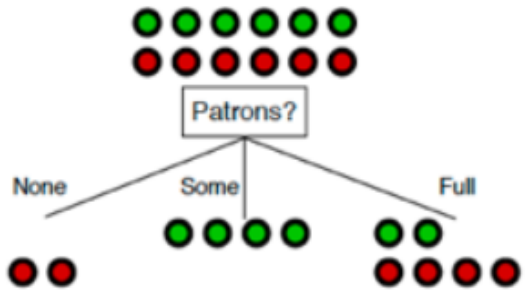
$$-\left(\frac{0}{0+2}\log\frac{0}{0+2}+\frac{2}{0+2}\log\frac{2}{0+2}\right)=0$$

For "Some" branch

$$-\left(\frac{4}{4+0}\log\frac{4}{4+0}+\frac{0}{4+0}\log\frac{0}{4+0}\right)=0$$

For "Full" branch

$$-\left(\frac{2}{2+4}\log\frac{2}{2+4}+\frac{4}{2+4}\log\frac{4}{2+4}\right)\approx 0.9$$



So how good is choosing "patrons" overall?

Very naturally, we take the weighted average of entropy:

$$\frac{2}{12}\times 0+\frac{4}{12}\times 0+\frac{6}{12}\times 0.9=0.45$$

Measure of uncertainty of a split

Suppose we split based on a discrete feature A, the uncertainty can be measured by the **conditional entropy**:

$$H(Y \mid A)$$

Measure of uncertainty of a split

Suppose we split based on a discrete feature A, the uncertainty can be measured by the **conditional entropy**:

$$
\begin{aligned}
& H(Y \mid A) \\
&= \sum_a P(A = a) H(Y \mid A = a)
\end{aligned}
$$

Measure of uncertainty of a split

Suppose we split based on a discrete feature A, the uncertainty can be measured by the **conditional entropy**:

$$
\begin{aligned}
& H(Y \mid A) \\
&= \sum_a P(A=a) H(Y \mid A=a) \\
&= \sum_a P(A=a) \left(-\sum_{k=1}^{\mathsf{C}} P(Y \mid A=a) \log P(Y \mid A=a) \right)
\end{aligned}
$$

Measure of uncertainty of a split

Suppose we split based on a discrete feature A, the uncertainty can be measured by the **conditional entropy**:

$$
\begin{aligned}
& H(Y \mid A) \\
&= \sum_a P(A=a) H(Y \mid A=a) \\
&= \sum_a P(A=a) \left(-\sum_{k=1}^{\mathsf{C}} P(Y \mid A=a) \log P(Y \mid A=a) \right) \\
&= \sum_a \text{“fraction of example at node } A=a\text{”} \times \text{“entropy at node } A=a\text{”}
\end{aligned}
$$

Measure of uncertainty of a split

Suppose we split based on a discrete feature A, the uncertainty can be measured by the **conditional entropy**:

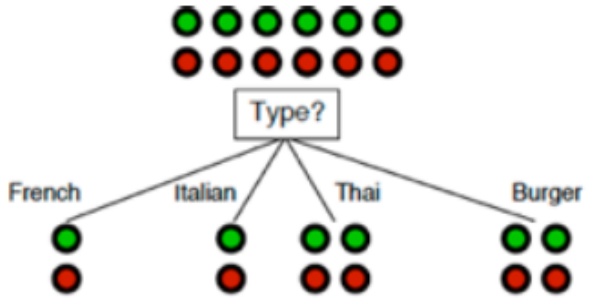
$$
\begin{aligned}
& H(Y \mid A) \\
&= \sum_a P(A=a) H(Y \mid A=a) \\
&= \sum_a P(A=a) \left(-\sum_{k=1}^{\mathsf{C}} P(Y \mid A=a) \log P(Y \mid A=a) \right) \\
&= \sum_a \text{“fraction of example at node } A=a\text{”} \times \text{“entropy at node } A=a\text{”}
\end{aligned}
$$

Pick the feature that leads to the smallest conditional entropy.

Deciding the root

For "French" branch

$$-\left(\frac{1}{1+1}\log\frac{1}{1+1}+\frac{1}{1+1}\log\frac{1}{1+1}\right)=1$$



For "Italian" branch

$$-\left(\frac{1}{1+1}\log\frac{1}{1+1}+\frac{1}{1+1}\log\frac{1}{1+1}\right)=1$$

For "Thai" and "Burger" branches

$$-\left(\frac{2}{2+2}\log\frac{2}{2+2}+\frac{2}{2+2}\log\frac{2}{2+2}\right)=1$$

Deciding the root

For "French" branch

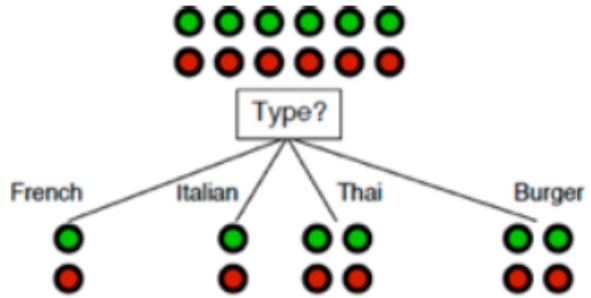
$$-\left(\frac{1}{1+1}\log\frac{1}{1+1}+\frac{1}{1+1}\log\frac{1}{1+1}\right)=1$$

For "Italian" branch

$$-\left(\frac{1}{1+1}\log\frac{1}{1+1}+\frac{1}{1+1}\log\frac{1}{1+1}\right)=1$$

For "Thai" and "Burger" branches

$$-\left(\frac{2}{2+2}\log\frac{2}{2+2}+\frac{2}{2+2}\log\frac{2}{2+2}\right)=1$$



The conditional entropy is $\frac{2}{12}\times 1+\frac{2}{12}\times 1+\frac{4}{12}\times 1+\frac{4}{12}\times 1=1>0.45$

Deciding the root

For "French" branch

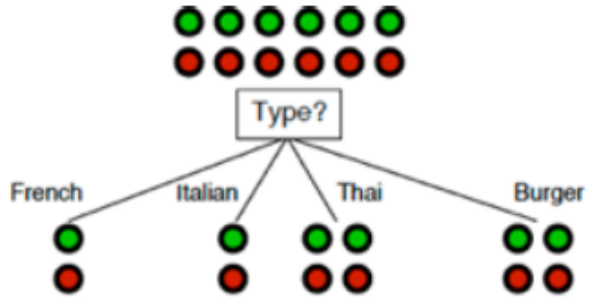
$$-\left(\frac{1}{1+1}\log\frac{1}{1+1}+\frac{1}{1+1}\log\frac{1}{1+1}\right)=1$$

For "Italian" branch

$$-\left(\frac{1}{1+1}\log\frac{1}{1+1}+\frac{1}{1+1}\log\frac{1}{1+1}\right)=1$$

For "Thai" and "Burger" branches

$$-\left(\frac{2}{2+2}\log\frac{2}{2+2}+\frac{2}{2+2}\log\frac{2}{2+2}\right)=1$$



The conditional entropy is $\frac{2}{12}\times 1+\frac{2}{12}\times 1+\frac{4}{12}\times 1+\frac{4}{12}\times 1=1>0.45$

So splitting with "patrons" is better than splitting with "type".

Deciding the root

For "French" branch

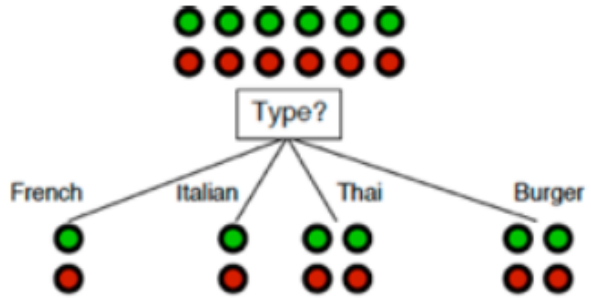
$$-\left(\frac{1}{1+1}\log\frac{1}{1+1}+\frac{1}{1+1}\log\frac{1}{1+1}\right)=1$$

For "Italian" branch

$$-\left(\frac{1}{1+1}\log\frac{1}{1+1}+\frac{1}{1+1}\log\frac{1}{1+1}\right)=1$$

For "Thai" and "Burger" branches

$$-\left(\frac{2}{2+2}\log\frac{2}{2+2}+\frac{2}{2+2}\log\frac{2}{2+2}\right)=1$$



The conditional entropy is $\frac{2}{12}\times 1+\frac{2}{12}\times 1+\frac{4}{12}\times 1+\frac{4}{12}\times 1=1>0.45$

So splitting with "patrons" is better than splitting with "type".

In fact by similar calculation "patrons" is the best split among all features.

Deciding the root

For "French" branch

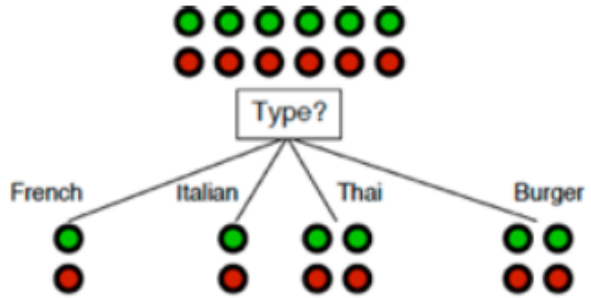
$$-\left(\frac{1}{1+1}\log\frac{1}{1+1}+\frac{1}{1+1}\log\frac{1}{1+1}\right)=1$$

For "Italian" branch

$$-\left(\frac{1}{1+1}\log\frac{1}{1+1}+\frac{1}{1+1}\log\frac{1}{1+1}\right)=1$$

For "Thai" and "Burger" branches

$$-\left(\frac{2}{2+2}\log\frac{2}{2+2}+\frac{2}{2+2}\log\frac{2}{2+2}\right)=1$$



The conditional entropy is $\frac{2}{12}\times 1+\frac{2}{12}\times 1+\frac{4}{12}\times 1+\frac{4}{12}\times 1=1>0.45$

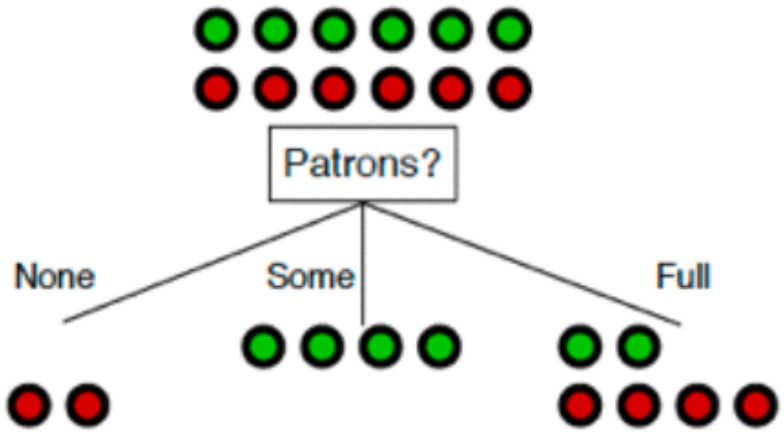
So splitting with "patrons" is better than splitting with "type".

In fact by similar calculation "patrons" is the best split among all features.

We are now done with building the root (this is also called a **stump**).

Repeat recursively

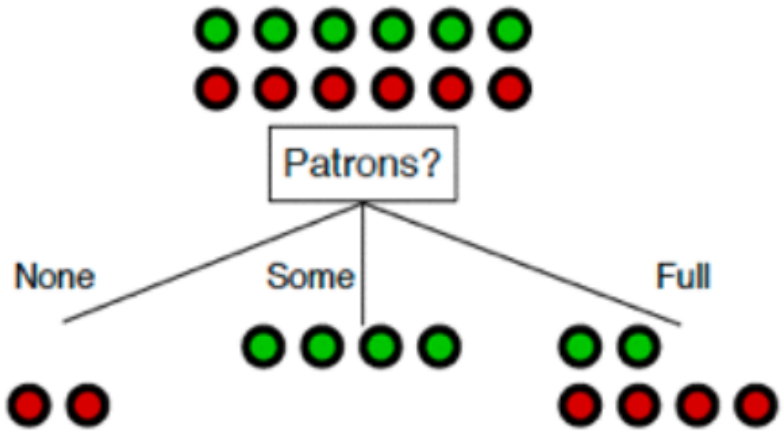
Split each child in the same way.



Repeat recursively

Split each child in the same way.

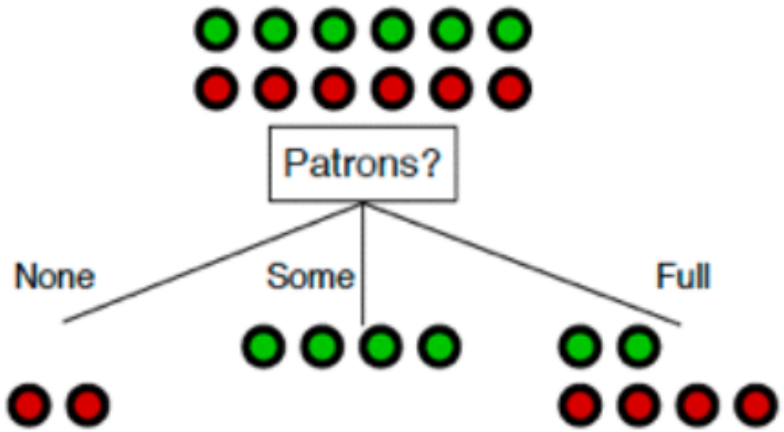
- but no need to split children "none" and "some": they are pure already and become leaves



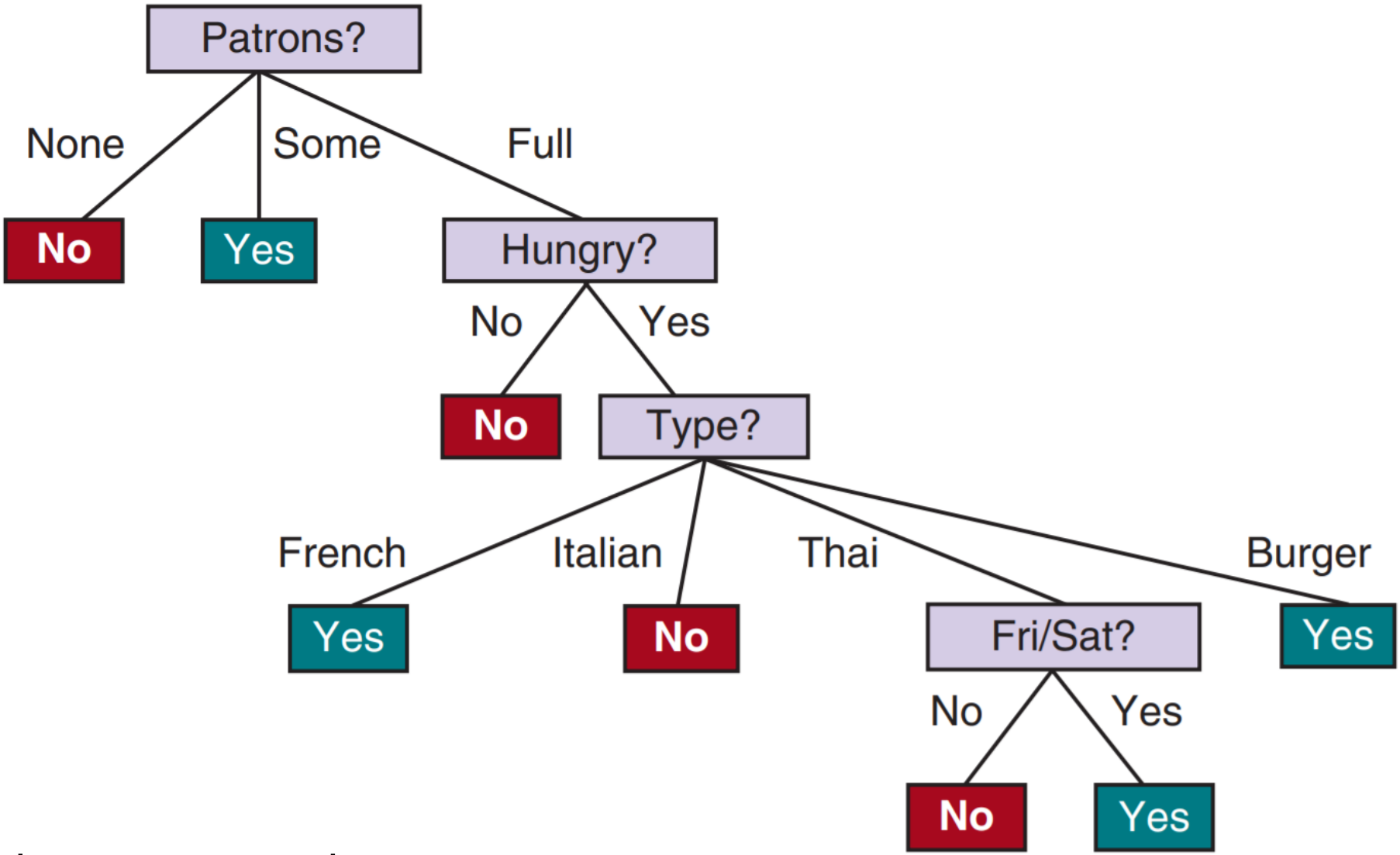
Repeat recursively

Split each child in the same way.

- but no need to split children "none" and "some": they are pure already and become leaves
- for "full", repeat, focusing on those 6 examples:



	Alt	Bar	Fri	Hun	Pat	Price	Rain	Res	Type	Est	WillWait
X_1	T	F	F	T	Some	$$$	F	T	French	0–10	T
X_2	T	F	F	T	Full	$	F	F	Thai	30–60	F
X_3	F	T	F	F	Some	$	F	F	Burger	0–10	T
X_4	T	F	T	T	Full	$	F	F	Thai	10–30	T
X_5	T	F	T	F	Full	$$$	F	T	French	>60	F
X_6	F	T	F	T	Some	$$	T	T	Italian	0–10	T
X_7	F	T	F	F	None	$	T	F	Burger	0–10	F
X_8	F	F	F	T	Some	$$	T	T	Thai	0–10	T
X_9	F	T	T	F	Full	$	T	F	Burger	>60	F
X_{10}	T	T	T	T	Full	$$$	F	T	Italian	10–30	F
X_{11}	F	F	F	F	None	$	F	F	Thai	0–10	F
X_{12}	T	T	T	T	Full	$	F	F	Burger	30–60	T



Again, very easy to interpret.

Random Forests

Random forest is an **ensemble** of trees:

Random Forests

Random forest is an **ensemble** of trees:

- each tree is built using *a bootstrapped dataset* (that is, a set of points randomly sampled from the training set with replacement)

Random Forests

Random forest is an **ensemble** of trees:

- each tree is built using *a bootstrapped dataset* (that is, a set of points randomly sampled from the training set with replacement)
- each split of each tree is selected from *a random subset of features*

Random Forests

Random forest is an **ensemble** of trees:

- each tree is built using *a bootstrapped dataset* (that is, a set of points randomly sampled from the training set with replacement)
- each split of each tree is selected from *a random subset of features*
- final prediction is the *plurality vote* of all tress (for classification tasks) or the *averaged prediction* of all trees (for regression tasks)

Random Forests

Random forest is an **ensemble** of trees:

- each tree is built using *a bootstrapped dataset* (that is, a set of points randomly sampled from the training set with replacement)
- each split of each tree is selected from *a random subset of features*
- final prediction is the *plurality vote* of all tress (for classification tasks) or the *averaged prediction* of all trees (for regression tasks)
- much better performance than a single tree, trivially parallelizable!

Outline

Introduction

Boosting (an even more powerful/general ensemble method):

- is a meta-algorithm, which takes a base algorithm (classification, regression, ranking, etc) as input and boosts its accuracy

Introduction

Boosting (an even more powerful/general ensemble method):

- is a meta-algorithm, which takes a base algorithm (classification, regression, ranking, etc) as input and boosts its accuracy
- main idea: combine weak "rules of thumb" (e.g. 51% accuracy) to form a highly accurate predictor (e.g. 99% accuracy)

Introduction

Boosting (an even more powerful/general ensemble method):

- is a meta-algorithm, which takes a base algorithm (classification, regression, ranking, etc) as input and boosts its accuracy
- main idea: combine weak "rules of thumb" (e.g. 51% accuracy) to form a highly accurate predictor (e.g. 99% accuracy)
- works very well in practice (especially in combination with trees)

Introduction

Boosting (an even more powerful/general ensemble method):

- is a meta-algorithm, which takes a base algorithm (classification, regression, ranking, etc) as input and boosts its accuracy
- main idea: combine weak "rules of thumb" (e.g. 51% accuracy) to form a highly accurate predictor (e.g. 99% accuracy)
- works very well in practice (especially in combination with trees)
- often is *resistant to overfitting*

Introduction

Boosting (an even more powerful/general ensemble method):

- is a meta-algorithm, which takes a base algorithm (classification, regression, ranking, etc) as input and boosts its accuracy
- main idea: combine weak "rules of thumb" (e.g. 51% accuracy) to form a highly accurate predictor (e.g. 99% accuracy)
- works very well in practice (especially in combination with trees)
- often is *resistant to overfitting*
- has strong theoretical guarantees

Introduction

Boosting (an even more powerful/general ensemble method):

- is a meta-algorithm, which takes a base algorithm (classification, regression, ranking, etc) as input and boosts its accuracy
- main idea: combine weak "rules of thumb" (e.g. 51% accuracy) to form a highly accurate predictor (e.g. 99% accuracy)
- works very well in practice (especially in combination with trees)
- often is *resistant to overfitting*
- has strong theoretical guarantees

We again focus on binary classification.

A simple example

Email spam detection:

A simple example

Email spam detection:

- given a training set like:
  - ("Want to make money fast? ...", **spam**)
  - ("Viterbi Research Gist ...", **not spam**)

A simple example

Email spam detection:

- given a training set like:
  - ("Want to make money fast? ...", **spam**)
  - ("Viterbi Research Gist ...", **not spam**)
- first obtain a classifier by applying a base algorithm, which can be a rather simple/weak one, like decision stumps:
  - e.g. contains the word "money" $\Rightarrow$ spam

A simple example

Email spam detection:

- given a training set like:
  - ("Want to make money fast? ...", **spam**)
  - ("Viterbi Research Gist ...", **not spam**)
- first obtain a classifier by applying a base algorithm, which can be a rather simple/weak one, like decision stumps:
  - e.g. contains the word "money" $\Rightarrow$ spam
- reweight the examples so that "difficult" ones get more attention
  - e.g. spam that doesn't contain the word "money"

A simple example

Email spam detection:

- given a training set like:
  - ("Want to make money fast? ...", **spam**)
  - ("Viterbi Research Gist ...", **not spam**)
- first obtain a classifier by applying a base algorithm, which can be a rather simple/weak one, like decision stumps:
  - e.g. contains the word "money" $\Rightarrow$ spam
- reweight the examples so that "difficult" ones get more attention
  - e.g. spam that doesn't contain the word "money"
- obtain another classifier by applying the same base algorithm:
  - e.g. empty "to address" $\Rightarrow$ spam

A simple example

Email spam detection:

- given a training set like:
  - ("Want to make money fast? ...", **spam**)
  - ("Viterbi Research Gist ...", **not spam**)
- first obtain a classifier by applying a base algorithm, which can be a rather simple/weak one, like decision stumps:
  - e.g. contains the word "money" $\Rightarrow$ spam
- reweight the examples so that "difficult" ones get more attention
  - e.g. spam that doesn't contain the word "money"
- obtain another classifier by applying the same base algorithm:
  - e.g. empty "to address" $\Rightarrow$ spam
- repeat ...

A simple example

Email spam detection:

- given a training set like:
  - ("Want to make money fast? ...", **spam**)
  - ("Viterbi Research Gist ...", **not spam**)
- first obtain a classifier by applying a base algorithm, which can be a rather simple/weak one, like decision stumps:
  - e.g. contains the word "money" $\Rightarrow$ spam
- reweight the examples so that "difficult" ones get more attention
  - e.g. spam that doesn't contain the word "money"
- obtain another classifier by applying the same base algorithm:
  - e.g. empty "to address" $\Rightarrow$ spam
- repeat ...
- final classifier is the (weighted) majority vote of all weak classifiers

The base algorithm

A **base algorithm** $\mathcal{A}$ (also called weak learning algorithm/oracle) takes a training set S weighted by D as input, and outputs classifier $h \leftarrow \mathcal{A}(S, D)$

The base algorithm

A **base algorithm** $\mathcal{A}$ (also called weak learning algorithm/oracle) takes a training set S weighted by D as input, and outputs classifier $h \leftarrow \mathcal{A}(S, D)$

- this can be any off-the-shelf classification algorithm (e.g. decision trees, logistic regression, neural nets, etc)

The base algorithm

A **base algorithm** $\mathcal{A}$ (also called weak learning algorithm/oracle) takes a training set S weighted by D as input, and outputs classifier $h \leftarrow \mathcal{A}(S, D)$

- this can be any off-the-shelf classification algorithm (e.g. decision trees, logistic regression, neural nets, etc)
- many algorithms can deal with a weighted training set (e.g. for algorithm that minimizes some loss, we can simply replace "total loss" by "weighted total loss")

The base algorithm

A **base algorithm** $\mathcal{A}$ (also called weak learning algorithm/oracle) takes a training set S weighted by D as input, and outputs classifier $h \leftarrow \mathcal{A}(S, D)$

- this can be any off-the-shelf classification algorithm (e.g. decision trees, logistic regression, neural nets, etc)
- many algorithms can deal with a weighted training set (e.g. for algorithm that minimizes some loss, we can simply replace "total loss" by "weighted total loss")
- even if it's not obvious how to deal with weight directly, we can always resample according to D to create a new unweighted dataset

Boosting Algorithms

Given:

- a training set S
- a base algorithm $\mathcal{A}$

Boosting Algorithms

Given:

- a training set S
- a base algorithm $\mathcal{A}$

Two things to specify a boosting algorithm:

- how to **reweight** the examples?
- how to **combine** all the weak classifiers?

Boosting Algorithms

Given:

- a training set S
- a base algorithm $\mathcal{A}$

Two things to specify a boosting algorithm:

- how to **reweight** the examples?
- how to **combine** all the weak classifiers?

Focus on AdaBoost, one of the most successful boosting algorithms.

The AdaBoost Algorithm

Given a training set S and a base algorithm $\mathcal{A}$, initialize D_1 to be uniform

The AdaBoost Algorithm

Given a training set S and a base algorithm $\mathcal{A}$, initialize D_1 to be uniform

For $t = 1, \ldots, T$

- obtain a weak classifier $h_t \leftarrow \mathcal{A}(S, D_t)$

The AdaBoost Algorithm

Given a training set S and a base algorithm $\mathcal{A}$, initialize D_1 to be uniform

For $t = 1, \ldots, T$

- obtain a weak classifier $h_t \leftarrow \mathcal{A}(S, D_t)$
- calculate the importance of h_t as

$$\beta_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right) \qquad (\beta_t > 0 \Leftrightarrow \epsilon_t < 0.5)$$

where $\epsilon_t = \sum_{n:h_t(\boldsymbol{x}_n) \neq y_n} D_t(n)$ is the weighted error of h_t.

The AdaBoost Algorithm

Given a training set S and a base algorithm $\mathcal{A}$, initialize D_1 to be uniform

For $t = 1, \ldots, T$

- obtain a weak classifier $h_t \leftarrow \mathcal{A}(S, D_t)$
- calculate the importance of h_t as

$$\beta_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right) \qquad (\beta_t > 0 \Leftrightarrow \epsilon_t < 0.5)$$

  where $\epsilon_t = \sum_{n: h_t(\boldsymbol{x}_n) \neq y_n} D_t(n)$ is the weighted error of h_t.

- update distributions

$$D_{t+1}(n) \propto D_t(n) e^{-\beta_t y_n h_t(\boldsymbol{x}_n)} = \begin{cases} D_t(n) e^{-\beta_t} & \text{if } h_t(x_n) = y_n \\ D_t(n) e^{\beta_t} & \text{else} \end{cases}$$

The AdaBoost Algorithm

Given a training set S and a base algorithm $\mathcal{A}$, initialize D_1 to be uniform

For $t = 1, \ldots, T$

- obtain a weak classifier $h_t \leftarrow \mathcal{A}(S, D_t)$
- calculate the importance of h_t as

$$\beta_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right) \qquad (\beta_t > 0 \Leftrightarrow \epsilon_t < 0.5)$$

  where $\epsilon_t = \sum_{n:h_t(\boldsymbol{x}_n) \neq y_n} D_t(n)$ is the weighted error of h_t.

- update distributions

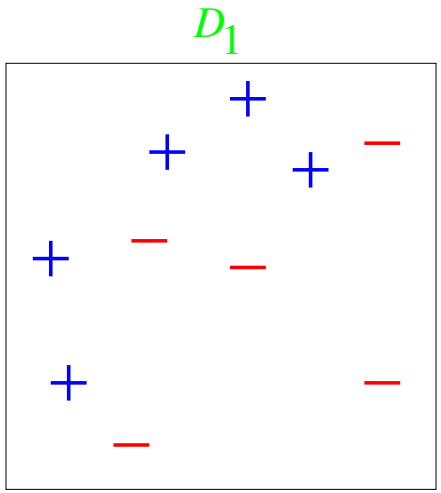
$$D_{t+1}(n) \propto D_t(n) e^{-\beta_t y_n h_t(\boldsymbol{x}_n)} = \begin{cases} D_t(n) e^{-\beta_t} & \text{if } h_t(x_n) = y_n \\ D_t(n) e^{\beta_t} & \text{else} \end{cases}$$

Output the final classifier $H(\boldsymbol{x}) = \text{sgn}\left(\sum_{t=1}^{T} \beta_t h_t(\boldsymbol{x}) \right)$

Example

10 data points in $\mathbb{R}^2$

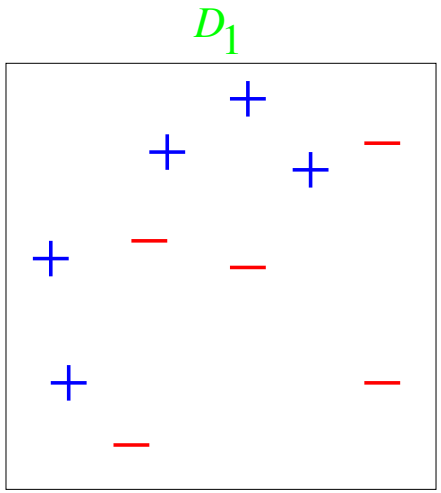
The size of $+$ or - indicates the weight, which starts from uniform D_1



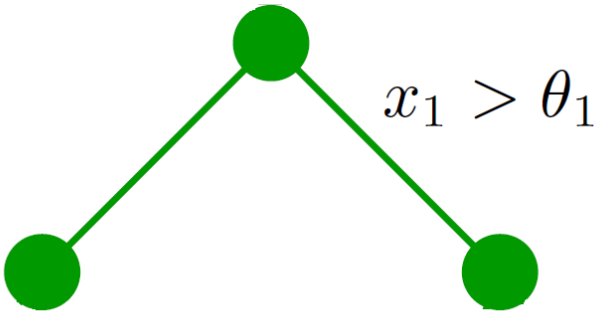
Example

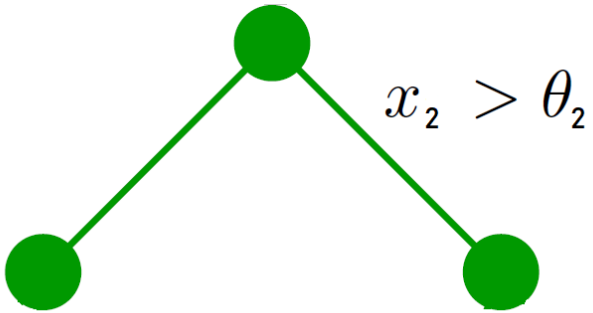
10 data points in $\mathbb{R}^2$

The size of $+$ or - indicates the weight, which starts from uniform D_1



Base algorithm is decision stump:

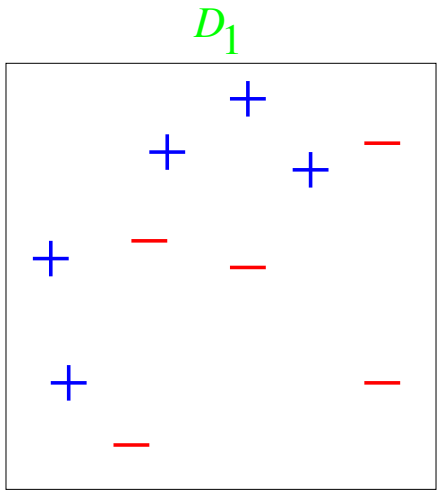




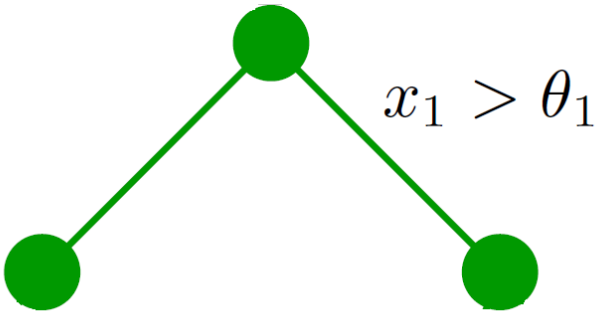
Example

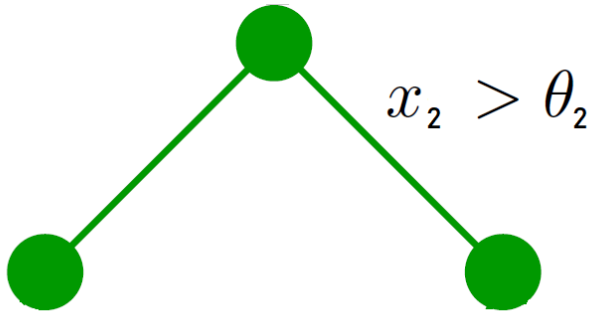
10 data points in $\mathbb{R}^2$

The size of + or - indicates the weight, which starts from uniform D_1



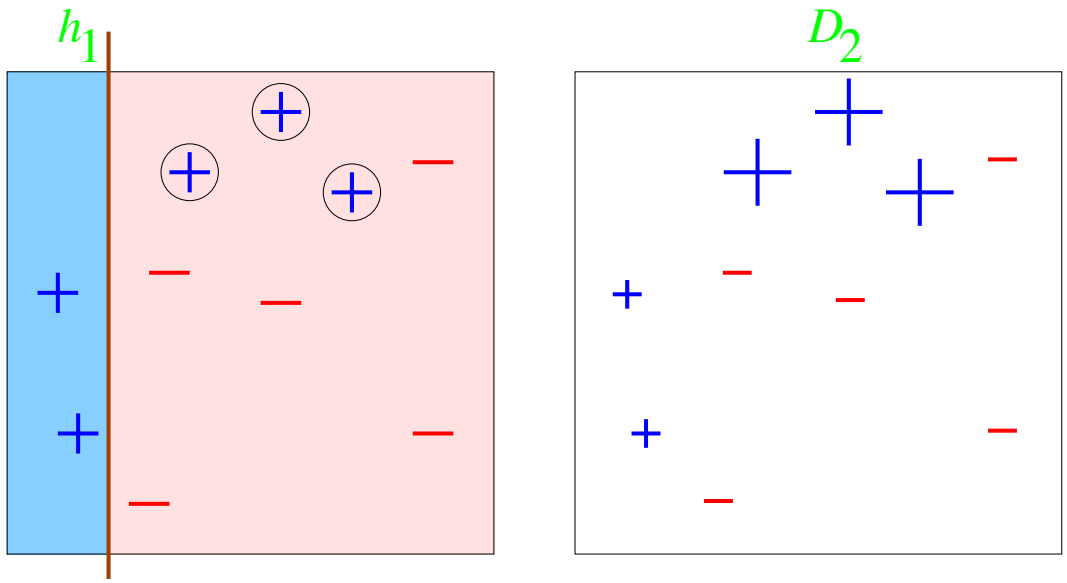
Base algorithm is decision stump:





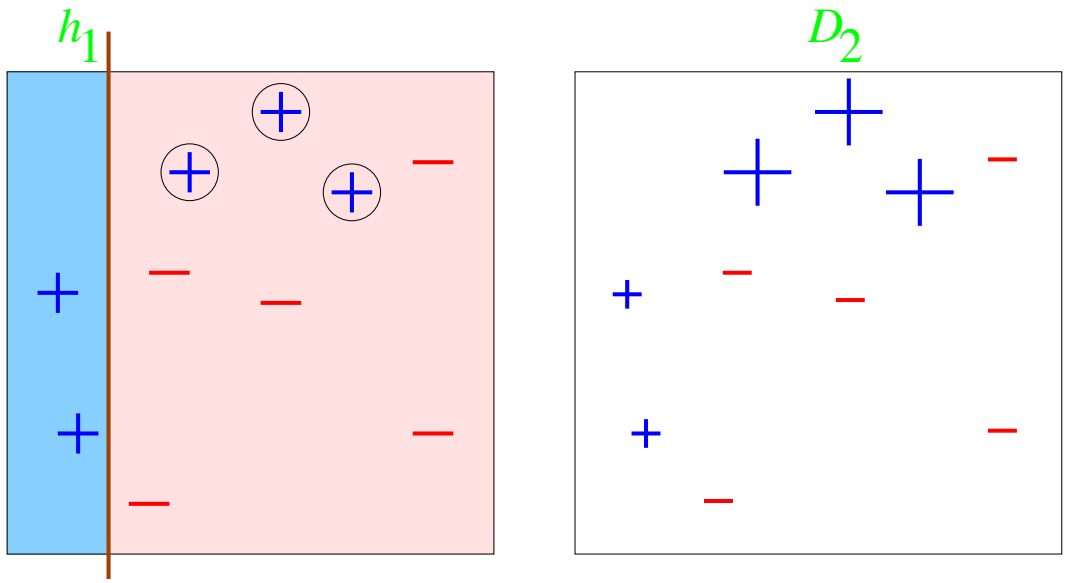
Observe that *no stump can predict very accurately for this dataset*

Round 1: $t = 1$



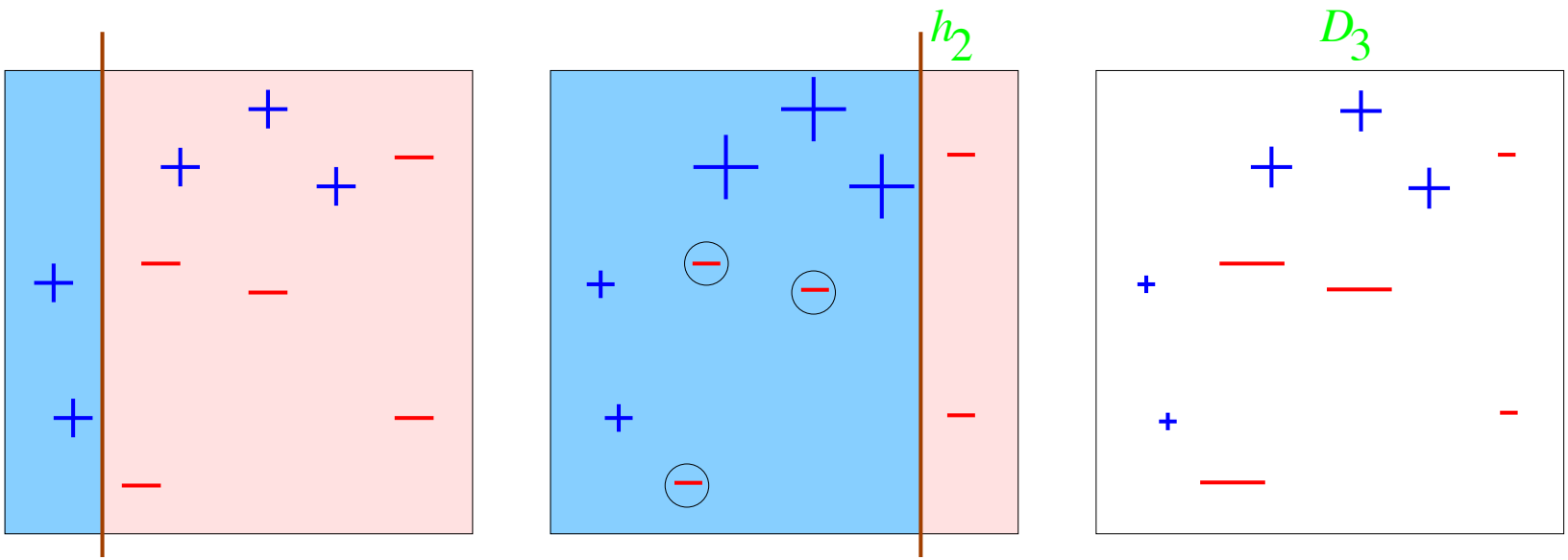
- 3 misclassified (circled): $\epsilon_1 = 0.3 \rightarrow \beta_1 = \frac{1}{2} \ln \left(\frac{1-\epsilon_t}{\epsilon_t} \right) \approx 0.42.$

Round 1: $t = 1$



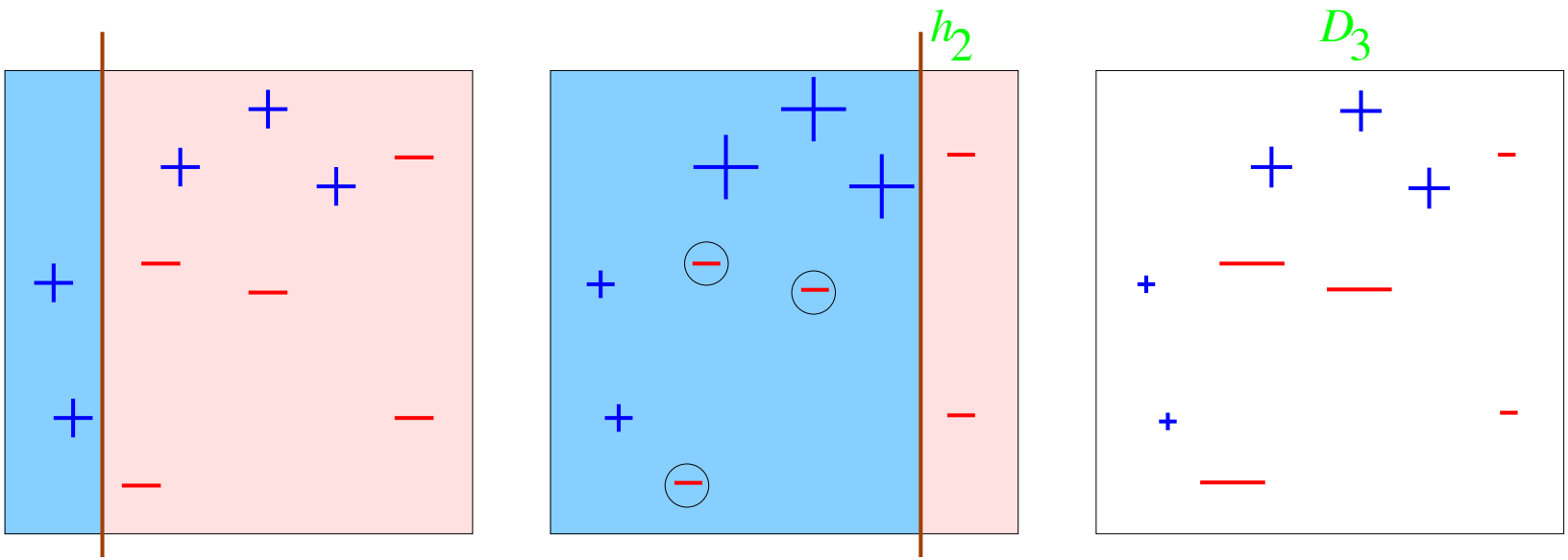
- 3 misclassified (circled): $\epsilon_1 = 0.3 \rightarrow \beta_1 = \frac{1}{2} \ln \left(\frac{1-\epsilon_t}{\epsilon_t} \right) \approx 0.42.$
- D_2 puts more weights on those examples

Round 2: $t = 2$



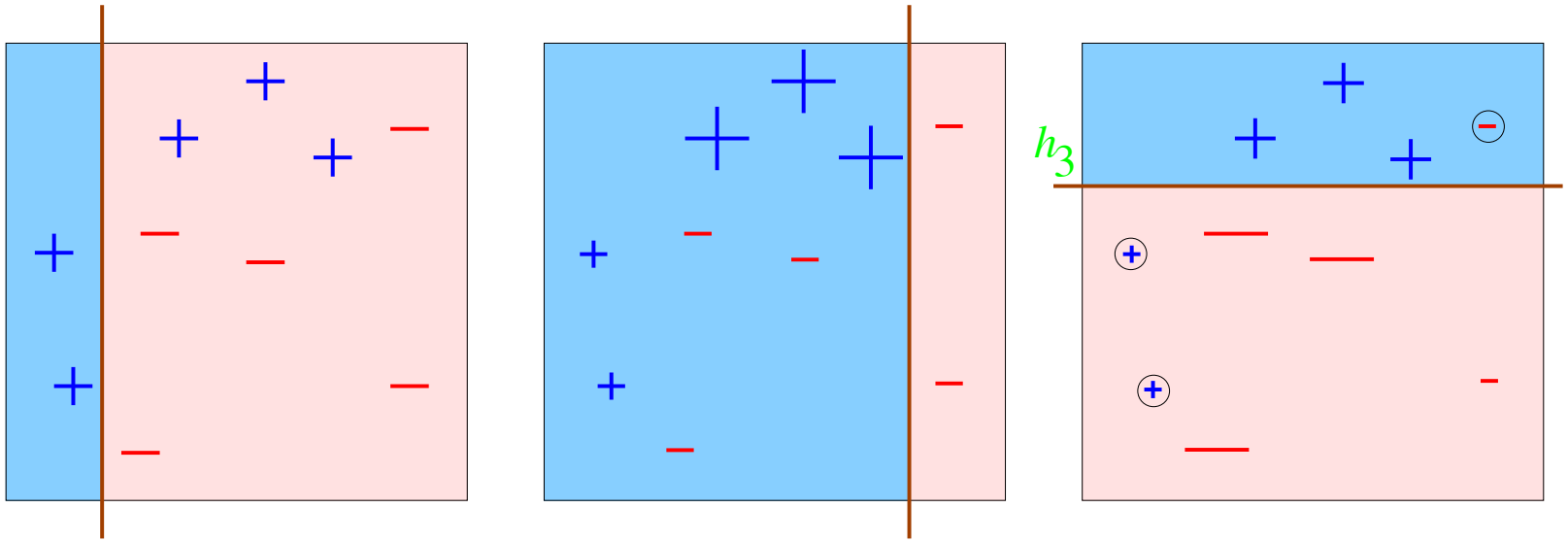
- 3 misclassified (circled): $\epsilon_2 = 0.21 \rightarrow \beta_2 = 0.65$.

Round 2: $t = 2$



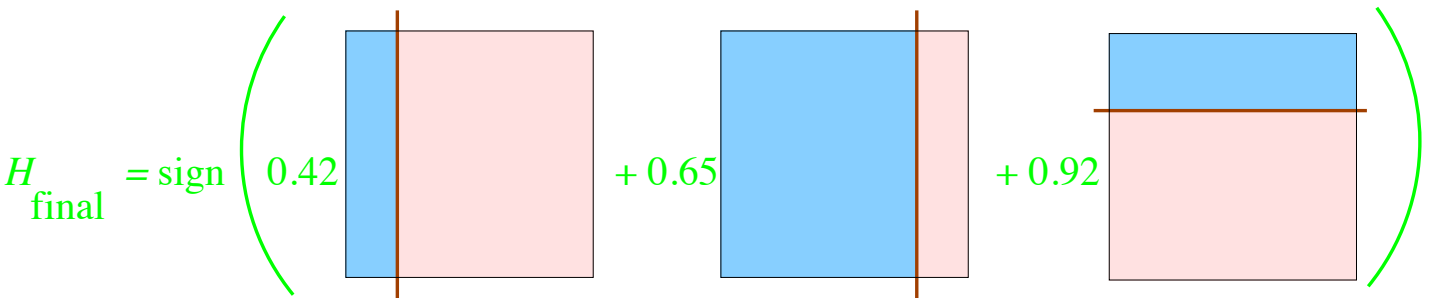
- 3 misclassified (circled): $\epsilon_2 = 0.21 \rightarrow \beta_2 = 0.65$.
- D_3 puts more weights on those examples

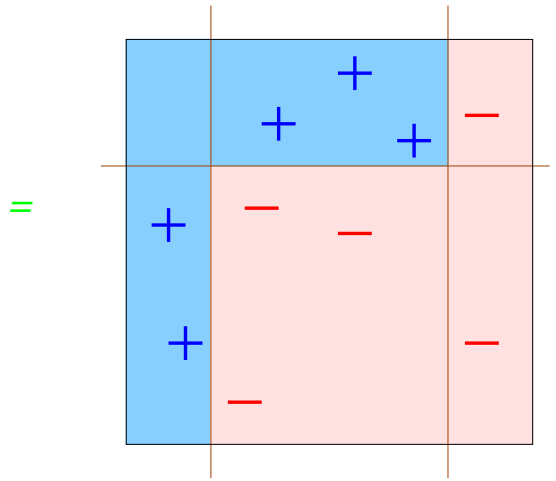
Round 3: $t = 3$



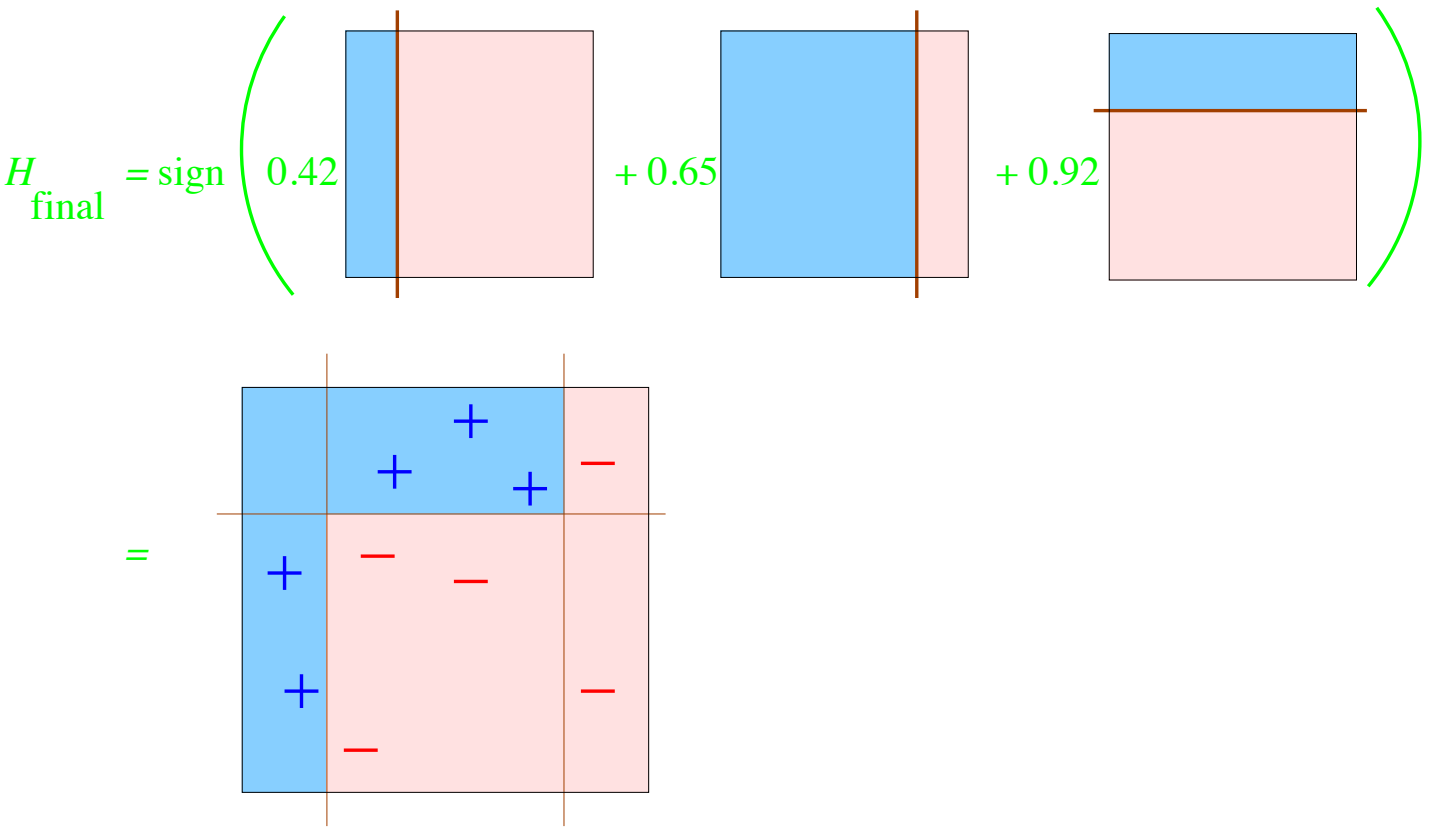
- again 3 misclassified (circled): $\epsilon_3 = 0.14 \rightarrow \beta_3 = 0.92$.

Final classifier: combining 3 classifiers





Final classifier: combining 3 classifiers



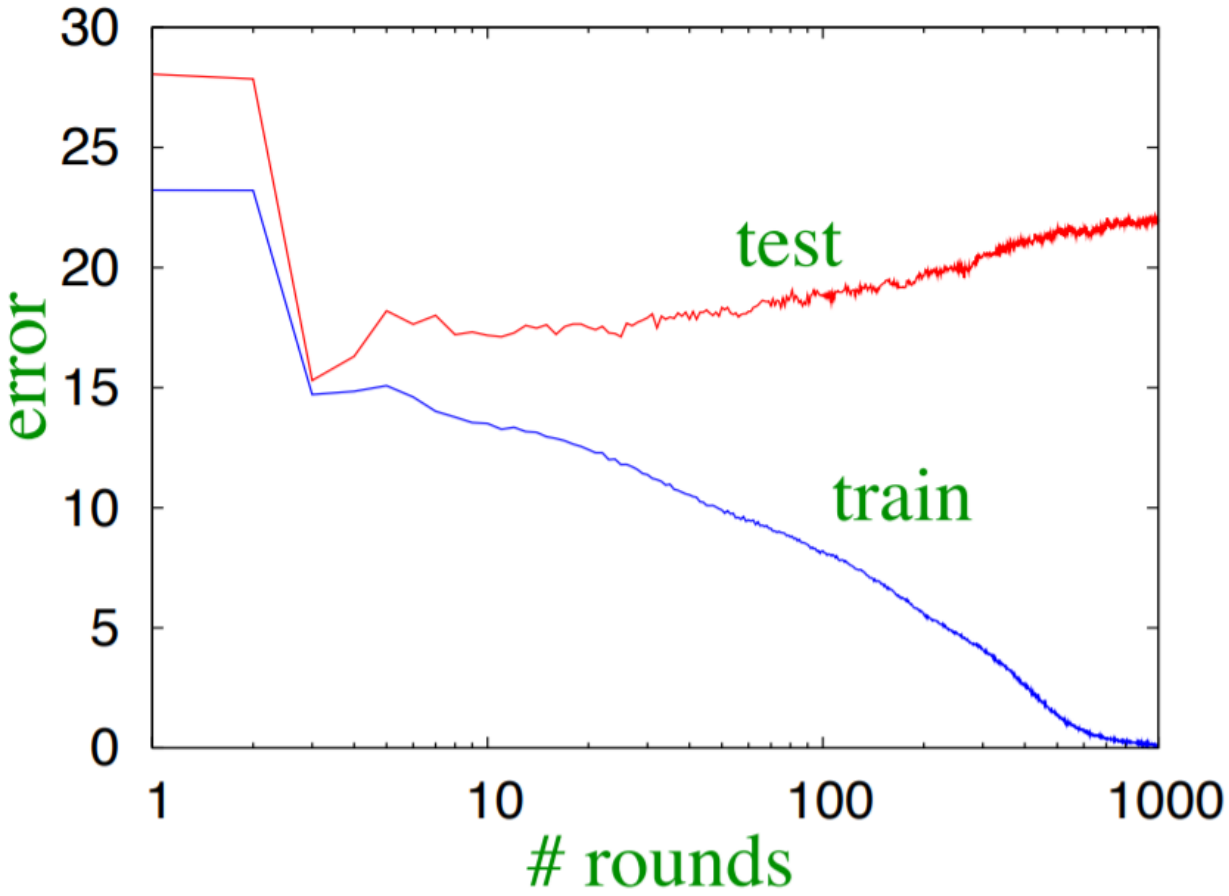
All data points are now classified correctly, even though each weak classifier makes 3 mistakes.

Overfitting

When T is large, the model is very complicated and overfitting can happen

Overfitting

When T is large, the model is very complicated and overfitting can happen



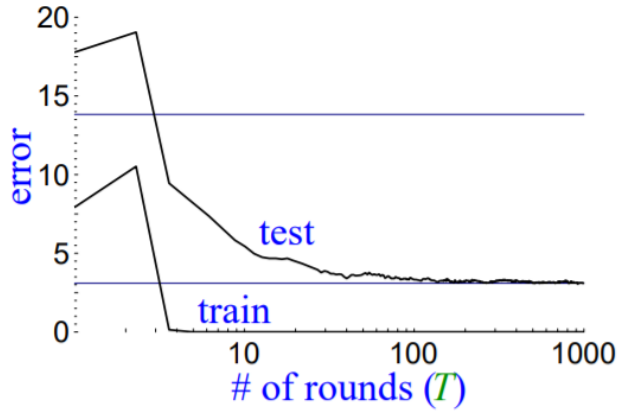
(boosting "stumps" on heart-disease dataset)

Resistance to overfitting

However, *very often AdaBoost is resistant to overfitting*

Resistance to overfitting

However, *very often AdaBoost is resistant to overfitting*



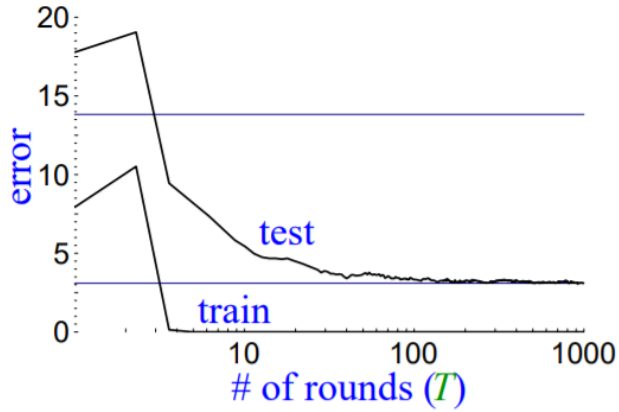
(boosting C4.5 on "letter" dataset)

- test error does not increase, even after 1000 rounds
  - (total size $>$ 2,000,000 nodes)
- test error continues to drop even after training error is zero!

	# rounds		
	5	100	1000
train error	0.0	0.0	0.0
test error	8.4	3.3	3.1

Resistance to overfitting

However, *very often AdaBoost is resistant to overfitting*



(boosting C4.5 on "letter" dataset)

- test error does not increase, even after 1000 rounds
  - (total size $>$ 2,000,000 nodes)
- test error continues to drop even after training error is zero!

	# rounds		
	5	100	1000
train error	0.0	0.0	0.0
test error	8.4	3.3	3.1

Used to be a mystery, but by now rigorous theory has been developed to explain this phenomenon.