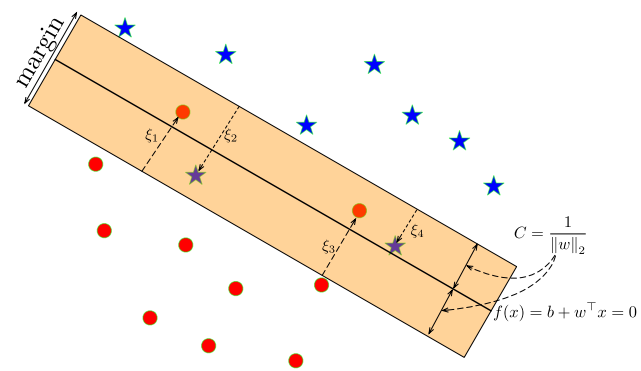
STA414/2104

Statistical Methods for Machine Learning II

Murat A. Erdogdu

Department of Computer Science
Department of Statistical Sciences

Lecture 7



UNIVERSITY OF TORONTO

Today

- Support Vector Machines
- Decision Trees
- HW3 is posted on the course webpage.

Binary Classification with a Linear Model

- Classification: Predict a discrete-valued target
- Binary classification: Targets $t \in \{-1, +1\}$ (previously we had $t \in \{0, +1\}$; here, however, we consider the former case).
- Linear model:

$$z = \mathbf{w}^\top \mathbf{x} + b$$
$$y = sign(z)$$

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

Zero-One Loss

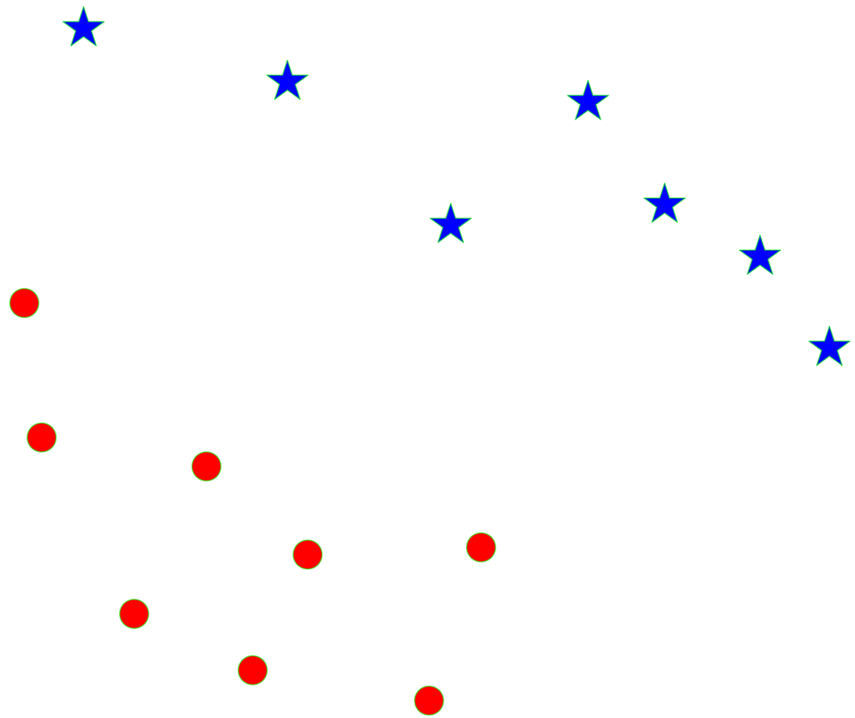
- We can use the $0-1$ loss function, and find the weights that minimize it over data points

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

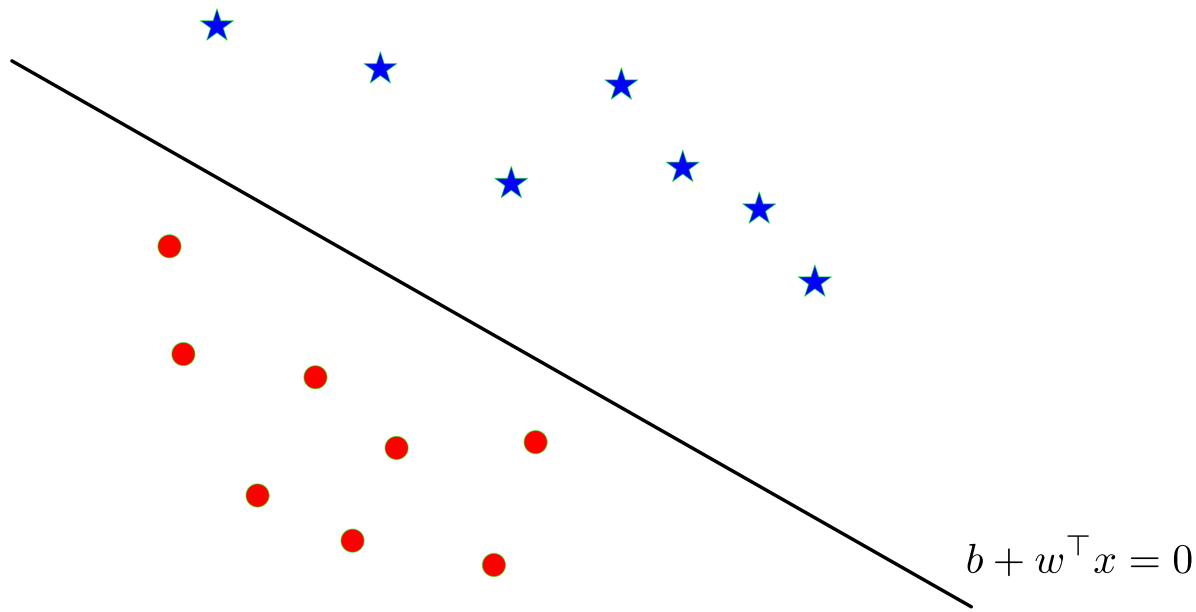
- But minimizing this loss is computationally difficult, and it can't distinguish different hypotheses that achieve the same accuracy.
- We investigated some other loss functions that are easier to minimize, e.g., logistic regression with the cross-entropy loss $\mathcal{L}_{\text{CE}}$.
- Let's consider a different approach, starting from the geometry of binary classifiers.

Separating Hyperplanes

Suppose we are given these data points from two different classes and want to find a linear classifier that separates them.

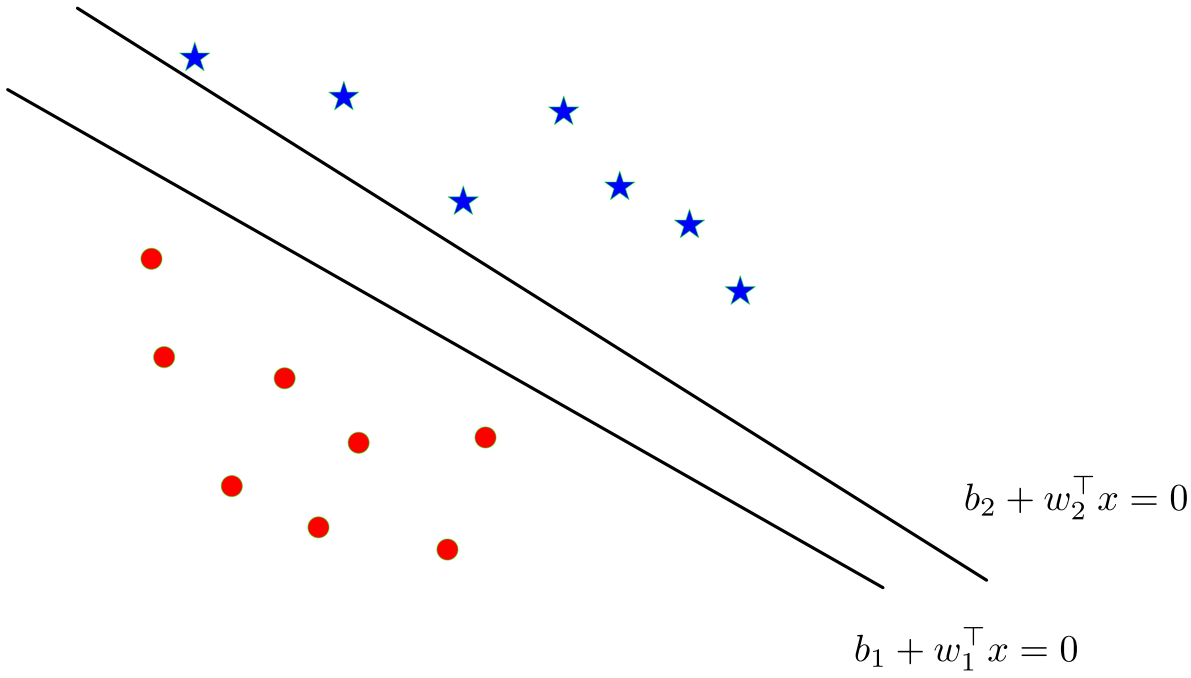


Separating Hyperplanes



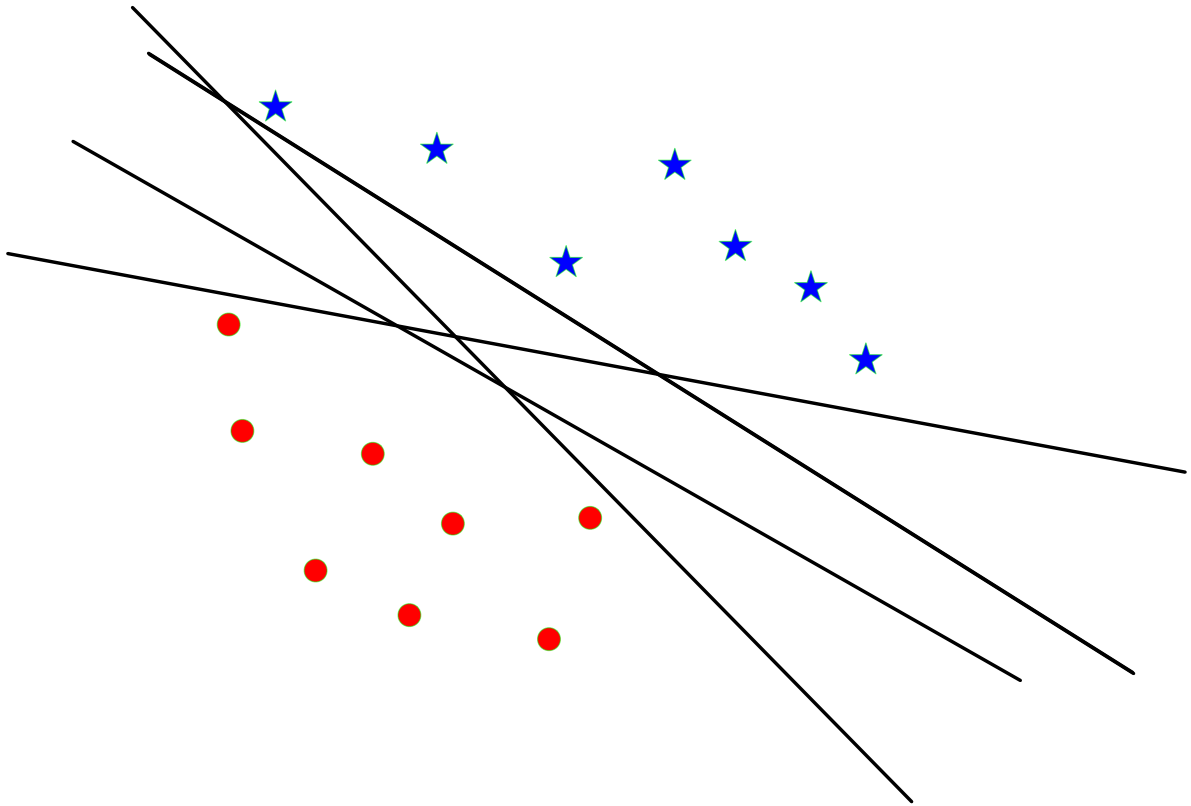
- The decision boundary looks like a line because $\mathsf{x} \in \mathbb{R}^2$, but think about it as a $D - 1$ dimensional hyperplane.
- Recall that a hyperplane is described by points $\mathsf{x} \in \mathbb{R}^D$ such that $f(\mathsf{x}) = \mathsf{w}^\top x + b = 0$.

Separating Hyperplanes



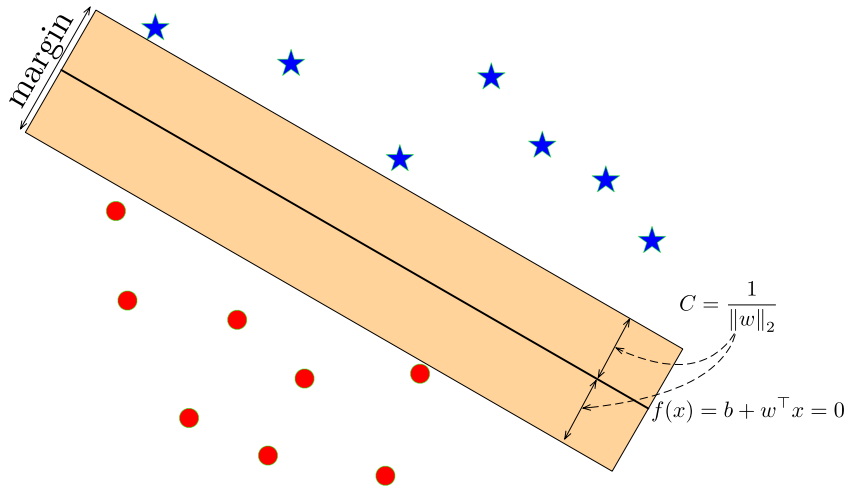
- There are multiple separating hyperplanes, described by different parameters $(\mathbf{w}, b)$.

Separating Hyperplanes



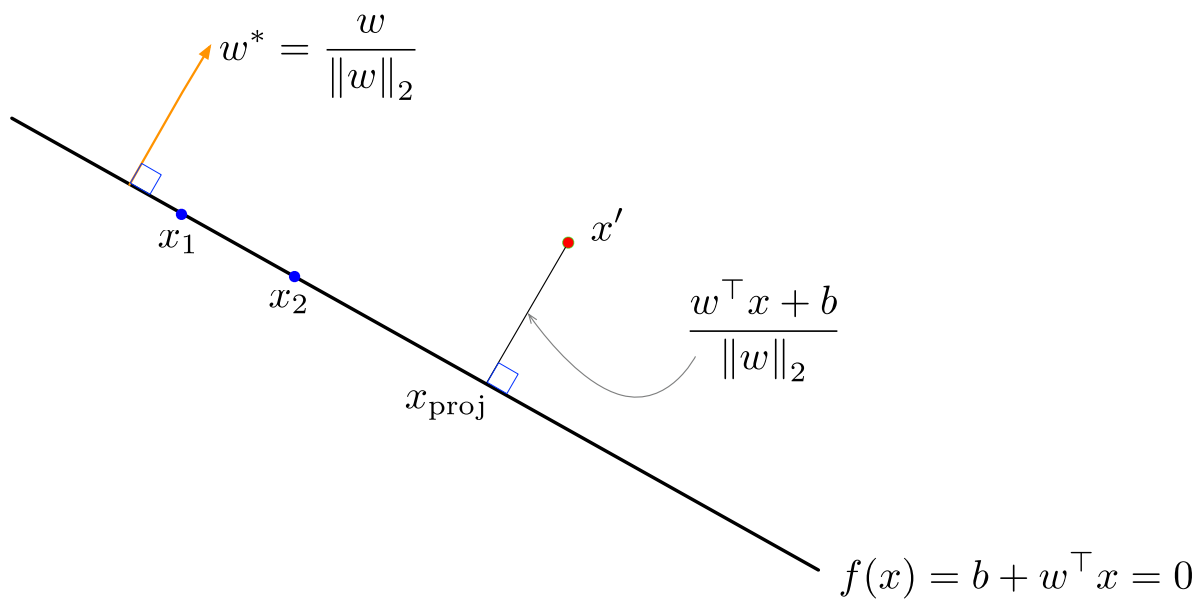
Optimal Separating Hyperplane

Optimal Separating Hyperplane: A hyperplane that separates two classes and maximizes the distance to the closest point from either class, i.e., maximize the **margin** of the classifier.



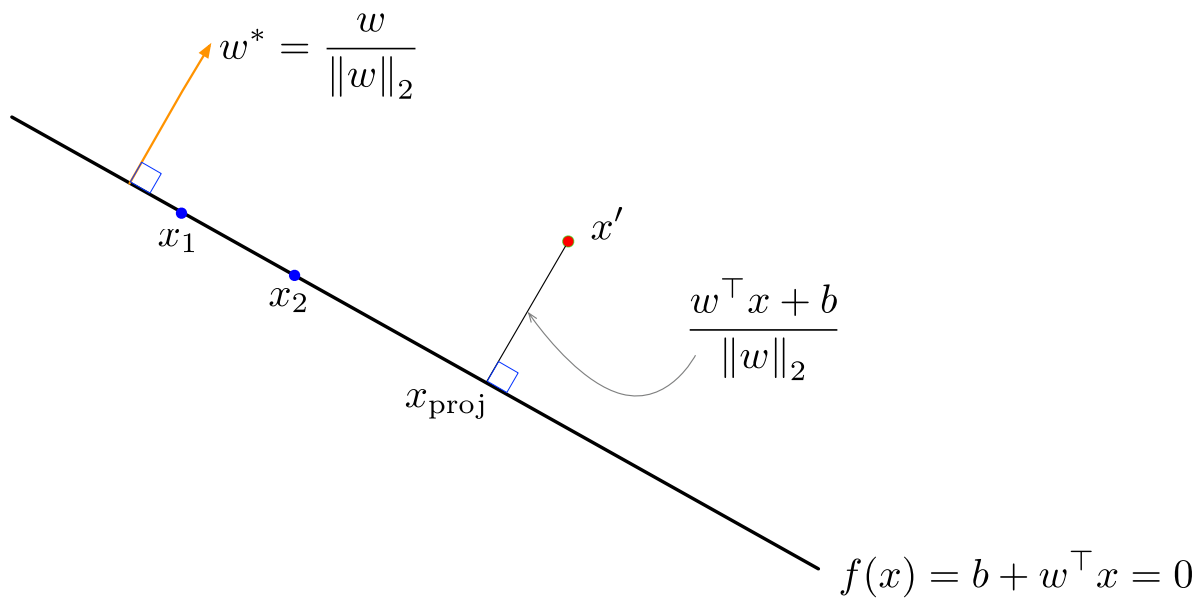
Intuitively, ensuring the decision boundary is not too close to any data points leads to better generalization on the test data.

Geometry of Points and Planes



- Recall that the decision hyperplane is orthogonal (perpendicular) to $\mathbf{w}$.
- The vector $\mathbf{w}^* = \frac{\mathbf{w}}{\|\mathbf{w}\|_2}$ is a unit vector pointing in the same direction as $\mathbf{w}$.
- The same hyperplane could equivalently be defined in terms of $\mathbf{w}^*$.

Geometry of Points and Planes



- Let's compute the distance between a point x' and the hyperplane $H = \{\mathsf{x} : \mathsf{w}^T\mathsf{x} + b = 0\}$
- We can write: $\mathsf{x}' = \mathsf{x}_{\text{proj}} + \mathsf{x}_N$ where $\mathsf{x}_{\text{proj}} \in H$ and $\mathsf{x}_N \in \text{span}(\mathsf{w})$
- Note: $\|\mathsf{x}_N\|_2$ is the distance from x' to H

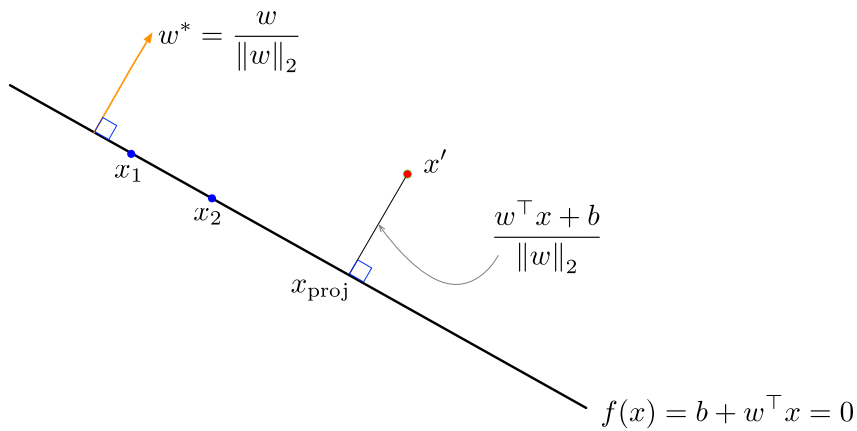
Geometry of Points and Planes

- Since $\mathbf{x}_N \in \text{span}(\mathbf{w})$, can write $\mathbf{x}_N = \lambda \mathbf{w}$ for some λ
- Then:

$$\begin{aligned} \mathbf{w}^T \mathbf{x}' + b &= \mathbf{w}^T (\mathbf{x}_{\text{proj}} + \mathbf{x}_N) + b \\ &= \mathbf{w}^T \mathbf{x}_{\text{proj}} + b + \mathbf{w}^T (\lambda \mathbf{w}) \\ &= 0 + \lambda \|\mathbf{w}\|_2^2 \\ &= \lambda \|\mathbf{w}\| \|\mathbf{w}\| \end{aligned}$$

- Hence, $\|\mathbf{x}_N\| = |\lambda| \|\mathbf{w}\| = \frac{|\mathbf{w}^T \mathbf{x}' + b|}{\|\mathbf{w}\|}$

Geometry of Points and Planes



The (signed) distance of a point x' to the hyperplane is

$$\frac{\mathsf{w}^\top \mathsf{x}' + b}{\|\mathsf{w}\|_2}$$

Maximizing Margin as an Optimization Problem

- Recall: the classification for the i-th data point is correct when

$$\text{sign}(\mathbf{w}^\top \mathbf{x}_i + b) = t_i$$

- This can be rewritten as

$$t_i(\mathbf{w}^\top \mathbf{x}_i + b) > 0$$

- Enforcing a margin of C:

$$t_i \cdot \underbrace{\frac{(\mathbf{w}^\top \mathbf{x}_i + b)}{\|\mathbf{w}\|_2}}_{\text{signed distance}} \geq C$$

Maximizing Margin as an Optimization Problem

Max-margin objective:

$$\max_{\mathbf{w},b} C$$

$$\text{s.t. } \frac{t_i(\mathbf{w}^\top \mathbf{x}_i + b)}{\|\mathbf{w}\|_2} \geq C \qquad i = 1, \ldots, N$$

- Note: can scale $\mathbf{w}$ and b by any positive value and get the same decision boundary
- This means we can choose to enforce $\|\mathbf{w}\|_2 = r$ for any $r > 0$ without changing the original solution
- Let's add the constraint $\|\mathbf{w}\|_2 = \frac{1}{C}$

$$\max_{\mathbf{w},b} C$$

$$\text{s.t. } \frac{t_i(\mathbf{w}^\top \mathbf{x}_i + b)}{\|\mathbf{w}\|_2} \geq C \qquad i = 1, \ldots, N$$

$$\|\mathbf{w}\|_2 = \frac{1}{C}$$

Maximizing Margin as an Optimization Problem

$$\max_{w,b} C$$

$$\text{s.t. } \frac{t_i(w^\top x_i + b)}{\|w\|_2} \geq C \qquad i = 1, \ldots, N$$

$$\|w\|_2 = \frac{1}{C}$$

Note that if $\|w\|_2 = \frac{1}{C}$ then:

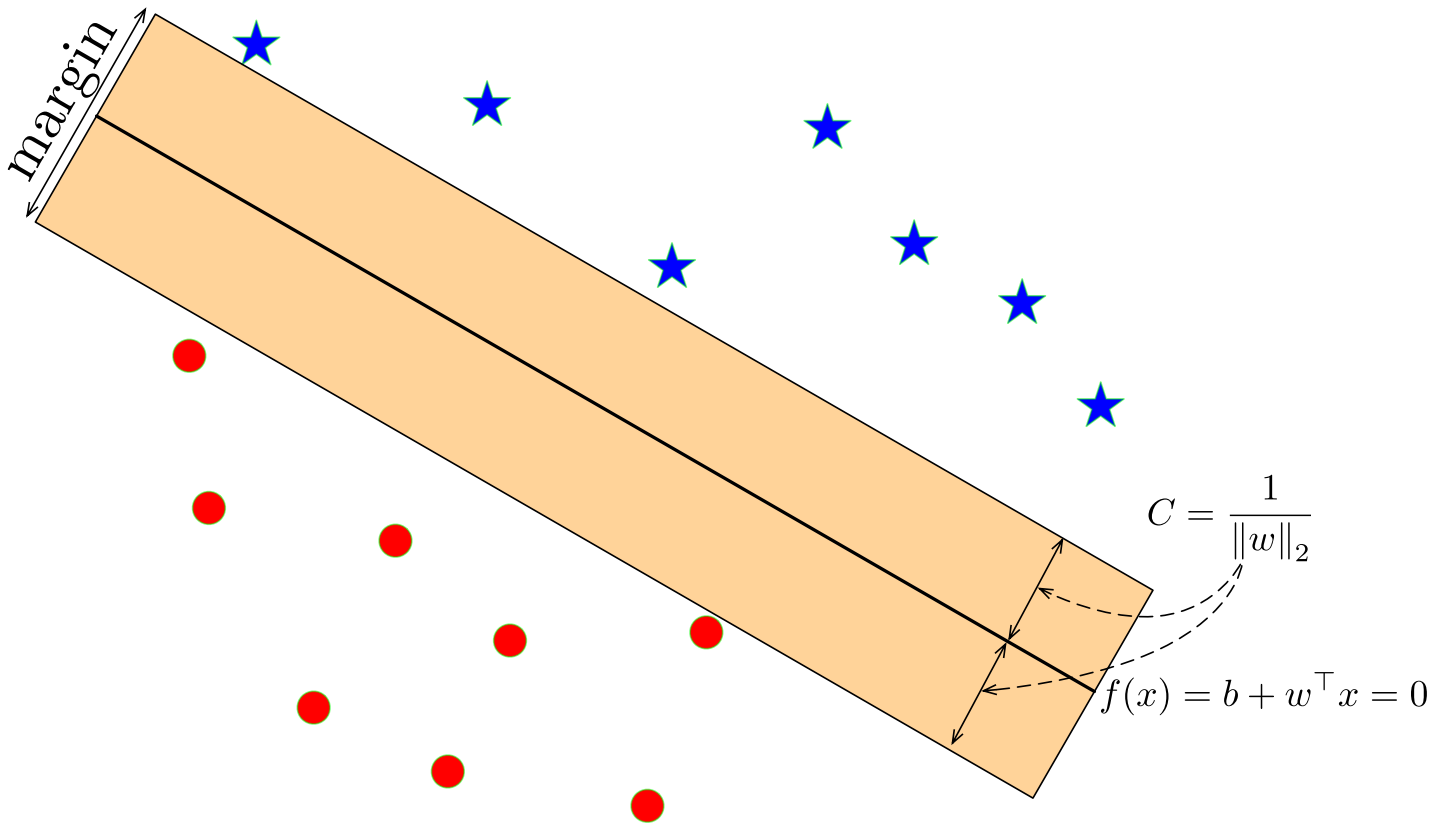
$$\underbrace{\frac{t_i(w^\top x_i + b)}{\|w\|_2} \geq \frac{1}{\|w\|_2}}_{\textbf{geometric margin constraint}} \iff \underbrace{t_i(w^\top x_i + b) \geq 1}_{\textbf{algebraic margin constraint}}$$

Plugging in $C = \frac{1}{\|w\|_2}$, equivalent optimization objective:

$$\min \|w\|_2^2$$

$$\text{s.t. } t_i(w^\top x_i + b) \geq 1 \qquad i = 1, \ldots, N$$

Maximizing Margin as an Optimization Problem

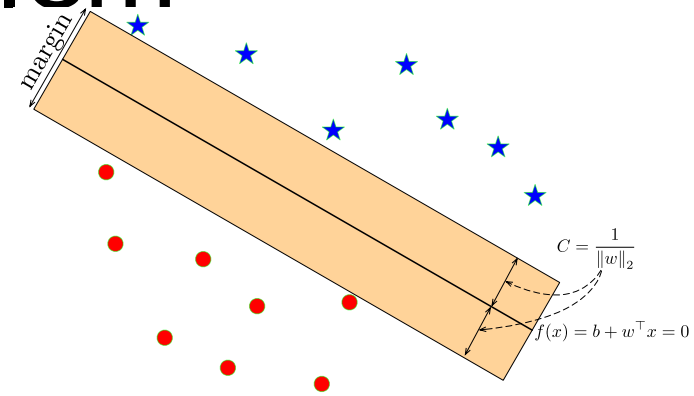


Maximizing Margin as an Optimization Problem

Algebraic max-margin objective:

$$\min_{\mathbf{w},b} \|\mathbf{w}\|_2^2$$

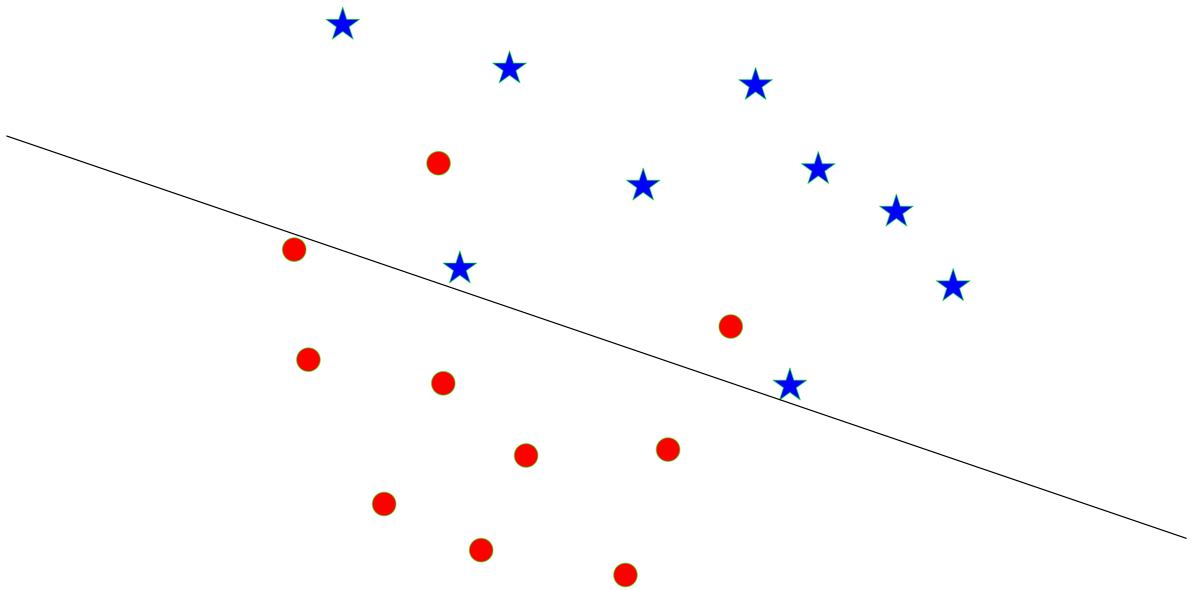
$$\text{s.t. } t_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1 \qquad i = 1, \ldots, N$$



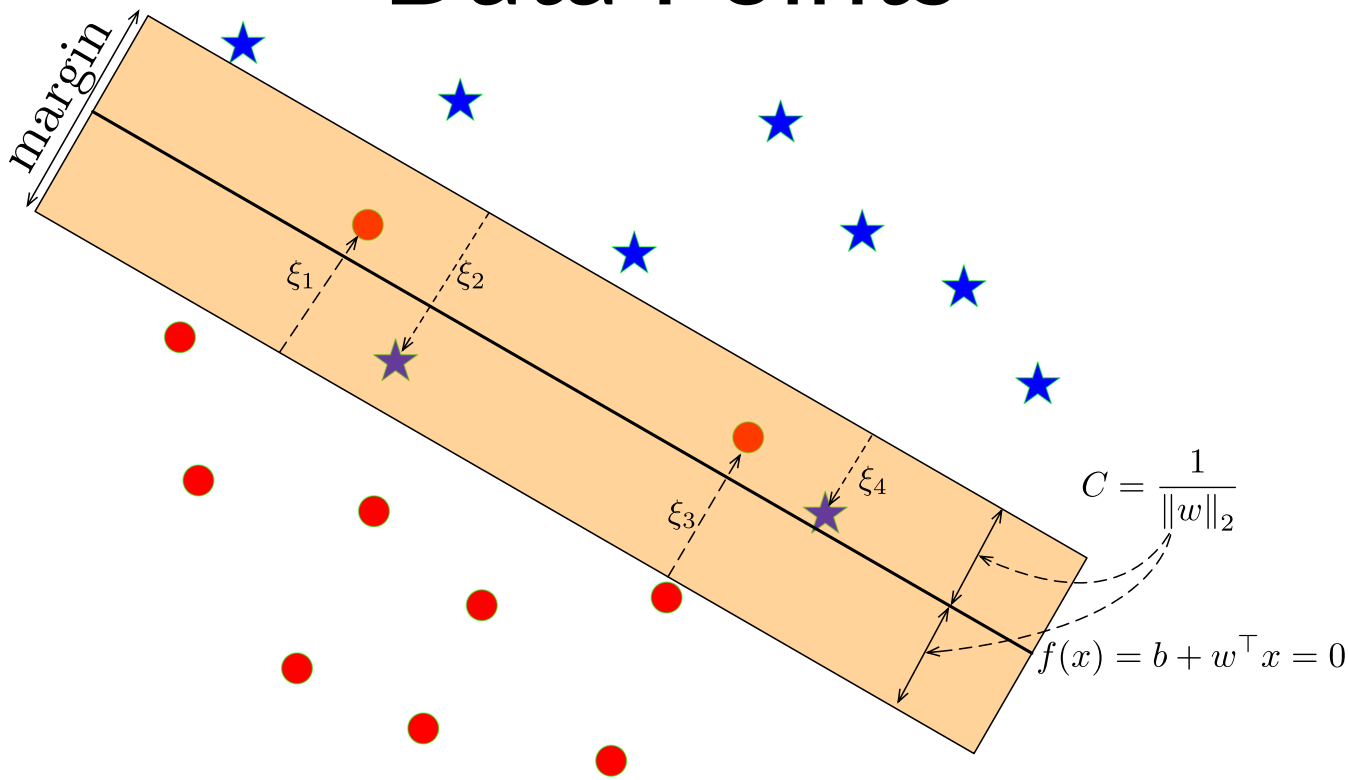
- Observe: if the margin constraint is not tight for $\mathbf{x}_i$, we could remove it from the training set and the optimal $\mathbf{w}$ would be the same.
- The important training examples are the ones with algebraic margin 1, and are called **support vectors**.
- Hence, this algorithm is called the (hard) **Support Vector Machine (SVM)** (or Support Vector Classifier).
- SVM-like algorithms are often called **max-margin** classifiers.

Non-Separable Data Points

How can we apply the max-margin principle if the data are **not** linearly separable?



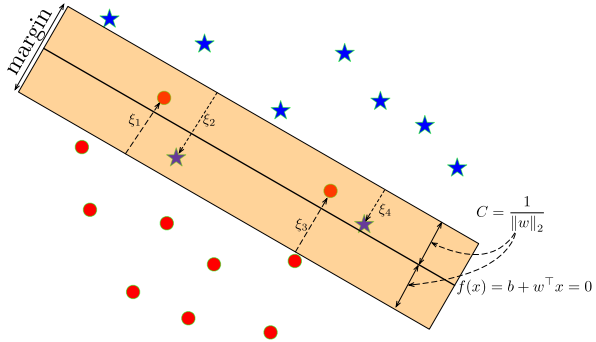
Maximizing Margin for Non-Separable Data Points



Main Idea:

- Allow some points to be within the margin or even be misclassified; we represent this with **slack variables** ξ_i.
- But constrain or penalize the total amount of slack.

Maximizing Margin for Non-Separable Data Points



- **Soft margin constraint**:

$$\frac{t_i(\mathbf{w}^\top \mathbf{x}_i + b)}{\|\mathbf{w}\|_2} \geq C(1 - \xi_i),$$

  for $\xi_i \geq 0$.

- Penalize $\sum_i \xi_i$

Maximizing Margin for Non-Separable Data Points

The **Soft-margin SVM** objective becomes:

$$\begin{aligned} \min_{\mathbf{w},b,\xi} \; & \frac{1}{2}\|\mathbf{w}\|_2^2 + \gamma \sum_{i=1}^{N} \xi_i \\ \text{s.t.} \; & t_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1 - \xi_i \qquad i = 1, \ldots, N \\ & \xi_i \geq 0 \qquad\qquad\qquad\qquad i = 1, \ldots, N \end{aligned}$$

- γ is a hyperparameter that trades off the margin with the amount of slack.
  - For $\gamma = 0$, we'll get $\mathbf{w} = \mathbf{0}$. (Why?)
  - As $\gamma \to \infty$ we get the hard-margin objective.
- Note: it is also possible to constrain $\sum_i \xi_i$ instead of penalizing it.

From Margin Violation to Hinge Loss

Let's simplify the soft margin constraint by eliminating ξ_i. Recall:

$$t_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1 - \xi_i \qquad i = 1, \ldots, N$$
$$\xi_i \geq 0 \qquad i = 1, \ldots, N$$

- Rewrite as $\xi_i \geq 1 - t_i(\mathbf{w}^\top \mathbf{x}_i + b)$.
- **Case 1:** $1 - t_i(\mathbf{w}^\top \mathbf{x}_i + b) \leq 0$
  - The smallest non-negative ξ_i that satisfies the constraint is $\xi_i = 0$.
- **Case 2:** $1 - t_i(\mathbf{w}^\top \mathbf{x}_i + b) > 0$
  - The smallest ξ_i that satisfies the constraint is $\xi_i = 1 - t_i(\mathbf{w}^\top \mathbf{x}_i + b)$.
- Hence, $\xi_i = \max\{0, 1 - t_i(\mathbf{w}^\top \mathbf{x}_i + b)\}$.
- Therefore, the slack penalty can be written as

$$\sum_{i=1}^{N} \xi_i = \sum_{i=1}^{N} \max\{0, 1 - t_i(\mathbf{w}^\top \mathbf{x}_i + b)\}.$$

From Margin Violation to Hinge Loss

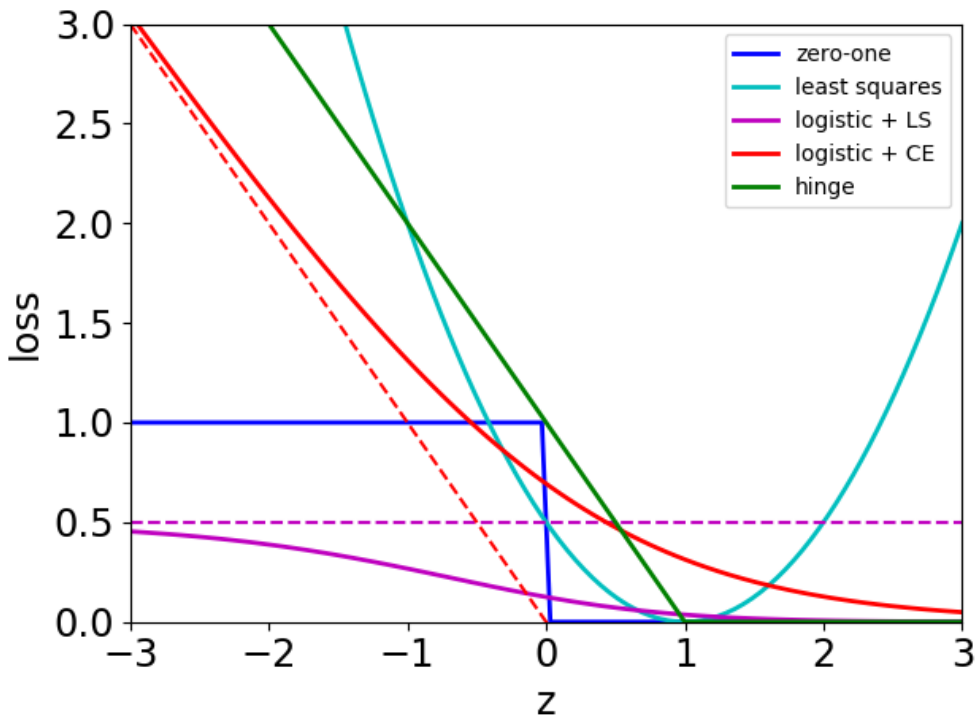
If we write $y_i(\mathsf{w}, b) = \mathsf{w}^\top \mathsf{x} + b$, then the optimization problem can be written as

$$\min_{\mathsf{w},b} \sum_{i=1}^{N} \max\{0, 1 - t_i y_i(\mathsf{w}, b)\} + \frac{1}{2\gamma} \|\mathsf{w}\|_2^2$$

- The loss function $\mathcal{L}_{\mathrm{H}}(y, t) = \max\{0, 1 - ty\}$ is called the **hinge** loss.
- The second term is the L_2-norm of the weights.
- Hence, the soft-margin SVM can be seen as a linear classifier with hinge loss and an L_2 regularizer.

Revisiting Loss Functions for Classification

Hinge loss compared with other loss functions



Dual problem

Soft-margin SVM objective:

$$\min_{\mathbf{w},b,\xi} \frac{1}{2}\|\mathbf{w}\|_2^2 + \gamma \sum_{i=1}^{N} \xi_i$$

$$\text{s.t.} \quad t_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1 - \xi_i \qquad i = 1, \ldots, N$$

$$\xi_i \geq 0 \qquad i = 1, \ldots, N$$

Write Lagrangian

$$L(\mathbf{w}, b, \xi, \alpha, r) = \frac{1}{2}\|\mathbf{w}\|_2^2 + \gamma \sum_{i=1}^{N} \xi_i - \sum_{i=1}^{N} \alpha_i [t_i(\mathbf{w}^\top \mathbf{x}_i + b) - 1 + \xi_i] - \sum_{i=1}^{N} \beta_i \xi_i.$$

Here, α_i's and β_i's are the Lagrange multipliers, constrained to be ≥ 0.

Dual problem

After setting the derivatives with respect to w and b to zero, substituting them back in, and simplifying, we obtain the following dual form of the problem (in hw3):

$$\max_{\alpha} W(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} t_i t_j \alpha_i \alpha_j \mathbf{x}_i^T \mathbf{x}_j$$

$$\text{s.t. } 0 \leq \alpha_i \leq \gamma$$

$$\sum_{i=1}^{N} \alpha_i t_i = 0.$$

- Predict using $\mathbf{w}^T \mathbf{x} + b = \sum_{i=1}^{N} \alpha_i t_i \mathbf{x}_i^T \mathbf{x} + b$.
- Most α_i's will be zero except for the support vectors. Thus, many of the terms in the sum above will be zero.
- Thus, dual formulation provides computational benefits when D ($\mathbf{x} \in \mathbb{R}^D$) is very large.
- Defining $\mathbf{x}^T \mathbf{x}' = k(\mathbf{x}, \mathbf{x}')$, we can kernelize the above formulation.

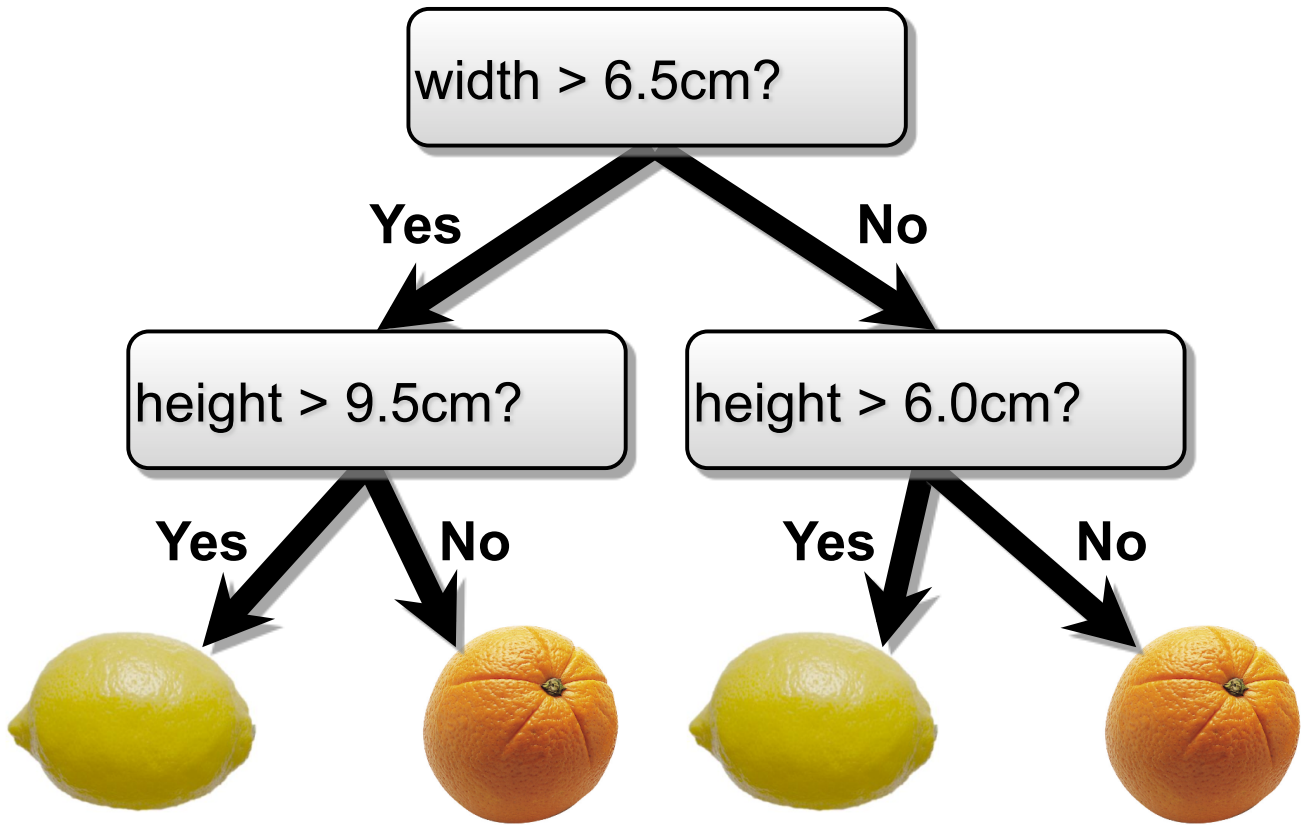
SVMs: What we Left Out

What we left out:

- How to fit $\mathbf{w}$ (or dual):
  - One option: gradient descent on the primal.
  - The SMO (sequential minimal optimization) algorithm, due to John Platt, gives an efficient way of solving the dual problem.
- The "kernel trick" converts it into a powerful nonlinear classifier.
- Classic results from learning theory show that a large margin implies good generalization.

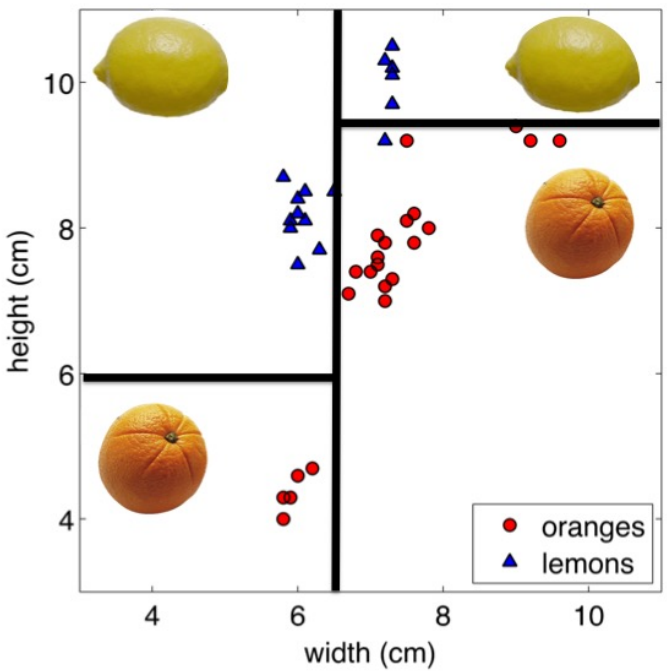
Decision Trees

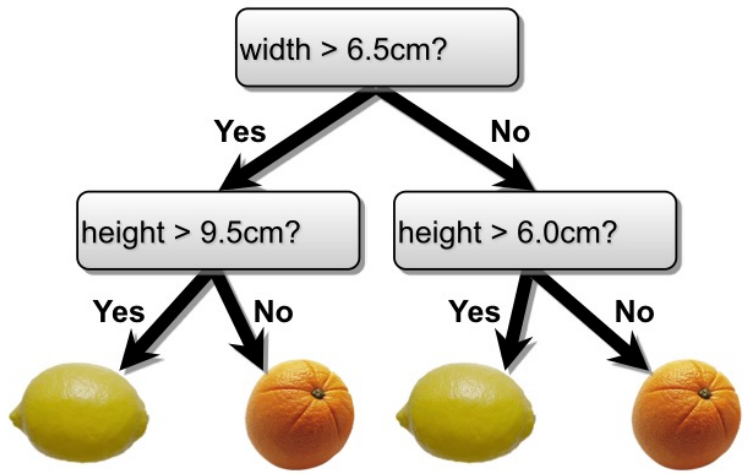
- **Decision trees** make predictions by recursively splitting on different attributes according to a tree structure.
- Example: classifying fruit as an orange or lemon based on height and width



Decision Trees

- For continuous attributes, split based on less than or greater than some threshold
- Thus, input space is divided into regions with boundaries parallel to axes





Example with Discrete Inputs

- What if the attributes are discrete?

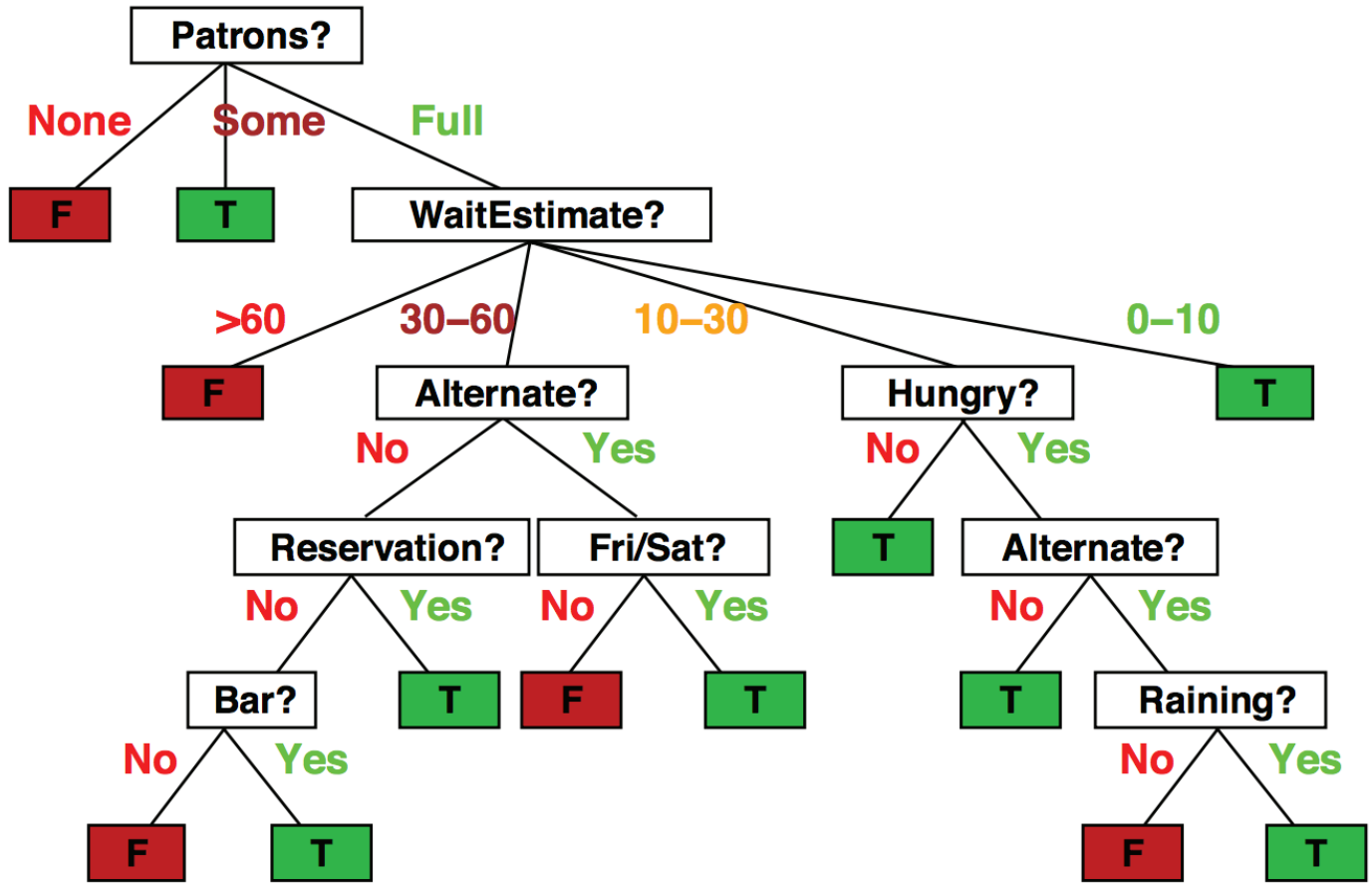
Example	Input Attributes										Goal
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
$\mathbf{x}_1$	*Yes*	*No*	*No*	*Yes*	*Some*	*$$$*	*No*	*Yes*	*French*	*0–10*	$y_1 = Yes$
$\mathbf{x}_2$	*Yes*	*No*	*No*	*Yes*	*Full*	*$*	*No*	*No*	*Thai*	*30–60*	$y_2 = No$
$\mathbf{x}_3$	*No*	*Yes*	*No*	*No*	*Some*	*$*	*No*	*No*	*Burger*	*0–10*	$y_3 = Yes$
$\mathbf{x}_4$	*Yes*	*No*	*Yes*	*Yes*	*Full*	*$*	*Yes*	*No*	*Thai*	*10–30*	$y_4 = Yes$
$\mathbf{x}_5$	*Yes*	*No*	*Yes*	*No*	*Full*	*$$$*	*No*	*Yes*	*French*	*>60*	$y_5 = No$
$\mathbf{x}_6$	*No*	*Yes*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Italian*	*0–10*	$y_6 = Yes$
$\mathbf{x}_7$	*No*	*Yes*	*No*	*No*	*None*	*$*	*Yes*	*No*	*Burger*	*0–10*	$y_7 = No$
$\mathbf{x}_8$	*No*	*No*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Thai*	*0–10*	$y_8 = Yes$
$\mathbf{x}_9$	*No*	*Yes*	*Yes*	*No*	*Full*	*$*	*Yes*	*No*	*Burger*	*>60*	$y_9 = No$
$\mathbf{x}_{10}$	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$$$*	*No*	*Yes*	*Italian*	*10–30*	$y_{10} = No$
$\mathbf{x}_{11}$	*No*	*No*	*No*	*No*	*None*	*$*	*No*	*No*	*Thai*	*0–10*	$y_{11} = No$
$\mathbf{x}_{12}$	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$*	*No*	*No*	*Burger*	*30–60*	$y_{12} = Yes$

1.	Alternate: whether there is a suitable alternative restaurant nearby.
2.	Bar: whether the restaurant has a comfortable bar area to wait in.
3.	Fri/Sat: true on Fridays and Saturdays.
4.	Hungry: whether we are hungry.
5.	Patrons: how many people are in the restaurant (values are None, Some, and Full).
6.	Price: the restaurant's price range ($, $$, $$$).
7.	Raining: whether it is raining outside.
8.	Reservation: whether we made a reservation.
9.	Type: the kind of restaurant (French, Italian, Thai or Burger).
10.	WaitEstimate: the wait estimated by the host (0-10 minutes, 10-30, 30-60, >60).

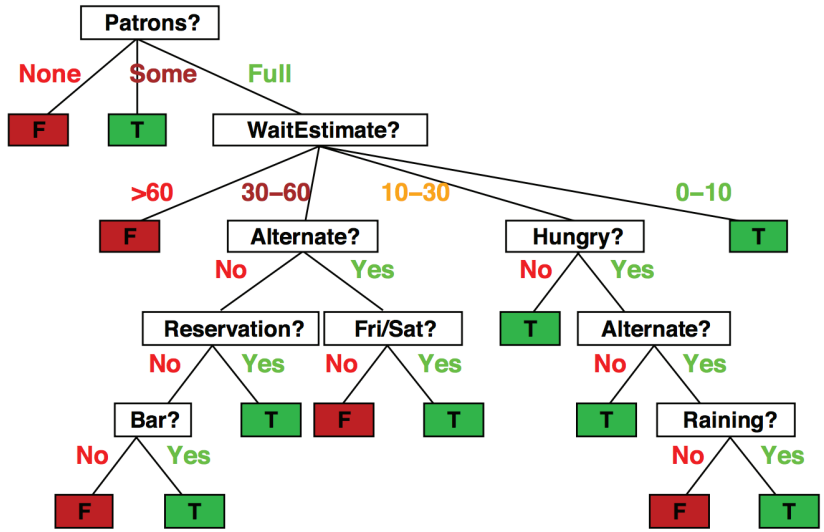
Attributes.

Decision Tree: Example with Discrete Inputs

- Possible tree to decide whether to wait (T) or not (F)

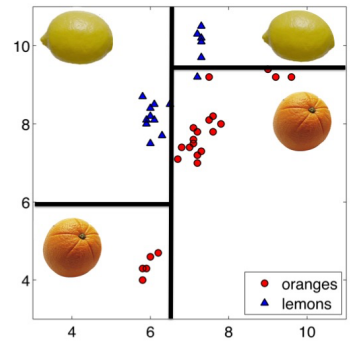


Decision Trees



- **Internal nodes** test **attributes**
- **Branching** is determined by **attribute value**
- **Leaf nodes** are **outputs** (predictions)

Decision Tree: Classification and Regression



- Each path from root to a leaf defines a region R_m of input space
- Let $\{(x_{m_1}, t_{m_1}), \ldots, (x_{m_k}, t_{m_k})\}$ be the training examples that fall into R_m
- **Classification tree**:
  - discrete output
  - leaf value y_m typically set to the most common value in $\{t_{m_1}, \ldots, t_{m_k}\}$
- **Regression tree**:
  - continuous output
  - leaf value y_m typically set to the mean value in $\{t_{m_1}, \ldots, t_{m_k}\}$

Note: We will focus on classification

How do we Learn a DecisionTree?

- How do we construct a useful decision tree?

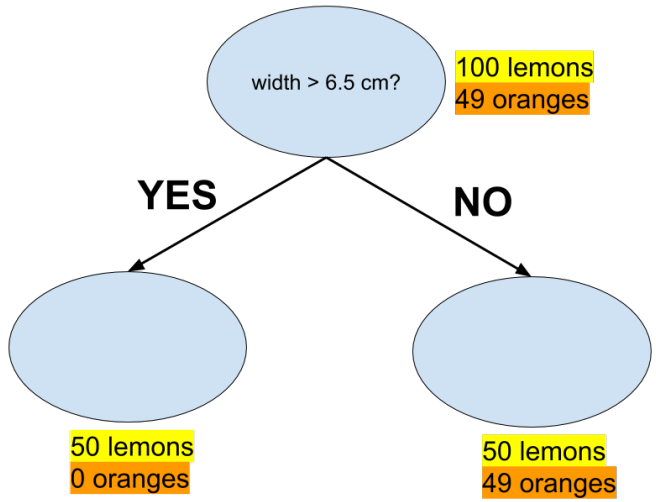
Learning Decision Trees

- Resort to a **greedy heuristic**! Start with empty decision tree and complete training set
  - Split on the "best" attribute, i.e. partition dataset
  - Recurse on subpartitions

- When should we stop?
- Which attribute is the "best" (and where should we split, if continuous)?
  - Choose based on accuracy?
  - Loss $L(R)$: misclassification rate
  - Say region R is split in R_1 and R_2 based on loss $L(R)$.
  - Accuracy gain is $L(R) - \frac{|R_1|L(R_1)+|R_2|L(R_2)}{|R_1|+|R_2|}$.

Choosing a Good Split

- Why isn't accuracy a good measure?
- Classify by the **majority**, loss is misclassification rate.



- Is this split good? Zero accuracy gain (using misclassification rate)

$$L(R) - \frac{|R_1|L(R_1) + |R_2|L(R_2)}{|R_1| + |R_2|} = \frac{49}{149} - \frac{50 \times 0 + 99 \times \frac{49}{99}}{149}$$

- But we've reduced our uncertainty about whether a fruit is a lemon

Choosing a Good Split

- How can we quantify uncertainty in prediction for a given leaf node?
  - All examples in leaf have same class: good, low uncertainty
  - Each class has same amount of examples in leaf: bad, high uncertainty
- **Idea:** Use counts at leaves to define probability distributions, and use information theory to measure uncertainty

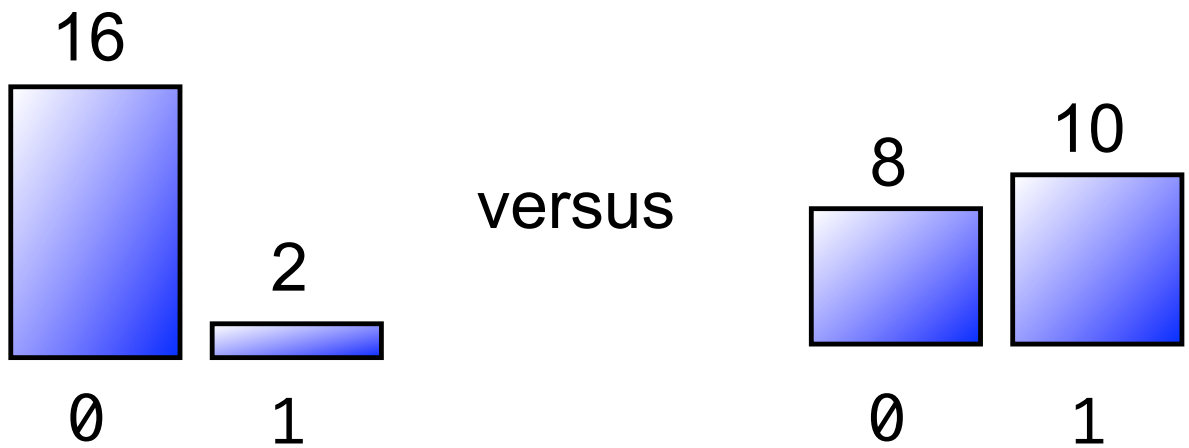
We Flip Two Different Coins

Sequence 1:

`0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 ... ?`

Sequence 2:

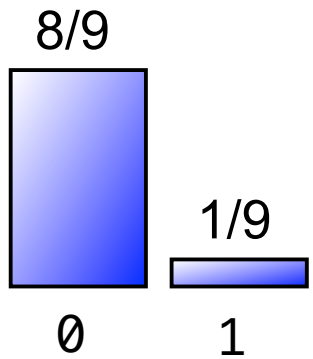
`0 1 0 1 0 1 1 1 0 1 0 0 1 1 0 1 0 1 ... ?`

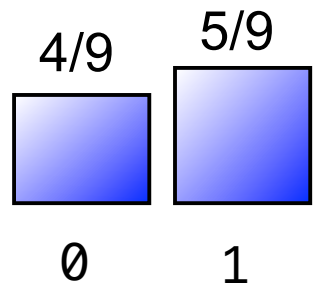


Quantifying Uncertainty

Entropy is a measure of expected "surprise": How uncertain are we of the value of a draw from this distribution?

$$H(X) = -\mathbb{E}_{X \sim p}[\log_2 p(X)] = -\sum_{x \in X} p(x) \log_2 p(x)$$



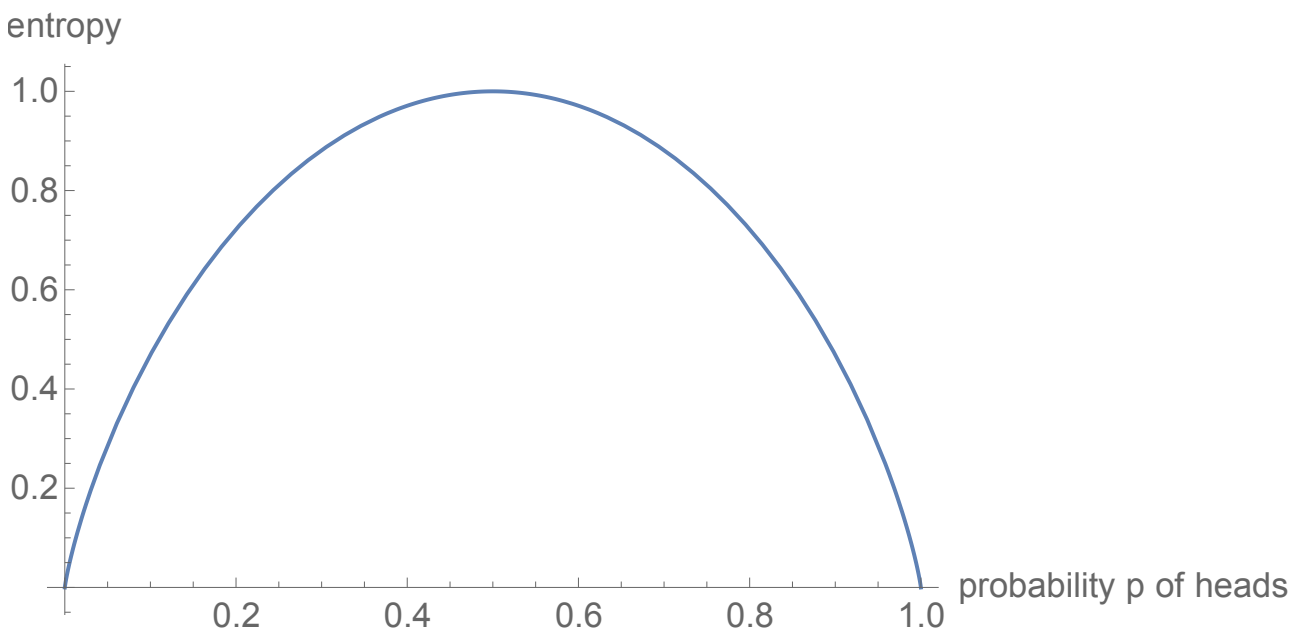


$$-\frac{8}{9}\log_2\frac{8}{9} - \frac{1}{9}\log_2\frac{1}{9} \approx \frac{1}{2} \qquad -\frac{4}{9}\log_2\frac{4}{9} - \frac{5}{9}\log_2\frac{5}{9} \approx 0.99$$

- We treat frequency as probabiliy.
- Averages over information content of each observation
- Unit = **bits** (based on the base of logarithm)
- A fair coin flip has 1 bit of entropy

Quantifying Uncertainty

$$H(X) = -\sum_{x \in X} p(x) \log_2 p(x)$$



Entropy

- **“High Entropy”**:
  - Variable has a uniform like distribution
  - Flat histogram
  - Values sampled from it are less predictable
- **“Low Entropy"**
  - Distribution of variable has peaks and valleys
  - Histogram has lows and highs
  - Values sampled from it are more predictable

Entropy of a Joint Distribution

- Example: $X = \{\text{Raining, Not raining}\}$, $Y = \{\text{Cloudy, Not cloudy}\}$

	Cloudy	Not Cloudy
Raining	24/100	1/100
Not Raining	25/100	50/100

$$
\begin{aligned}
H(X,Y) &= -\sum_{x\in X}\sum_{y\in Y} p(x,y)\log_2 p(x,y) \\
&= -\frac{24}{100}\log_2\frac{24}{100} - \frac{1}{100}\log_2\frac{1}{100} - \frac{25}{100}\log_2\frac{25}{100} - \frac{50}{100}\log_2\frac{50}{100} \\
&\approx 1.56\text{bits}
\end{aligned}
$$

Specific Conditional Entropy

- Example: $X = \{\text{Raining, Not raining}\}$, $Y = \{\text{Cloudy, Not cloudy}\}$

	Cloudy	Not Cloudy
Raining	24/100	1/100
Not Raining	25/100	50/100

- What is the entropy of cloudiness Y, **given that it is raining**?

$$\begin{aligned} H(Y|X = \text{raining}) &= -\sum_{y \in Y} p(y|\text{raining}) \log_2 p(y|\text{raining}) \\ &= -\frac{24}{25} \log_2 \frac{24}{25} - \frac{1}{25} \log_2 \frac{1}{25} \\ &\approx 0.24\text{bits} \end{aligned}$$

- We used: $p(y|x) = \frac{p(x,y)}{p(x)}$, and $p(x) = \sum_y p(x, y)$ (sum in a row)

Conditional Entropy

	Cloudy	Not Cloudy
Raining	24/100	1/100
Not Raining	25/100	50/100

- The expected conditional entropy:

$$
\begin{aligned}
H(Y|X) &= \mathbb{E}_{x\sim p(x)}[H(Y|X=x)] \\
&= \sum_{x\in X} p(x)H(Y|X=x) \\
&= -\sum_{x\in X}\sum_{y\in Y} p(x,y)\log_2 p(y|x) \\
&= -\mathbb{E}_{(X,Y)\sim p(x,y)}[\log_2 p(Y|X)]
\end{aligned}
\tag{1}
$$

Conditional Entropy

- Example: $X = \{\text{Raining, Not raining}\}$, $Y = \{\text{Cloudy, Not cloudy}\}$

	Cloudy	Not Cloudy
Raining	24/100	1/100
Not Raining	25/100	50/100

- What is the entropy of cloudiness, given the knowledge of whether or not it is raining?

$$\begin{aligned} H(Y|X) &= \sum_{x \in X} p(x) H(Y|X = x) \\ &= \frac{1}{4} H(\text{cloudyness}|\text{is raining}) + \frac{3}{4} H(\text{cloudyness}|\text{not raining}) \\ &\approx 0.75 \text{ bits} \end{aligned}$$

Conditional Entropy

- Some useful properties for the discrete case:
  - H is always non-negative.
  - Chain rule: $H(X, Y) = H(X|Y) + H(Y) = H(Y|X) + H(X)$
  - If X and Y independent, then X doesn't tell us anything about Y: $H(Y|X) = H(Y)$
  - But Y tells us everything about Y: $H(Y|Y) = 0$
  - By knowing X, we can only decrease uncertainty about Y: $H(Y|X) \leq H(Y)$

Verify these.

Information Gain

	Cloudy	Not Cloudy
Raining	24/100	1/100
Not Raining	25/100	50/100

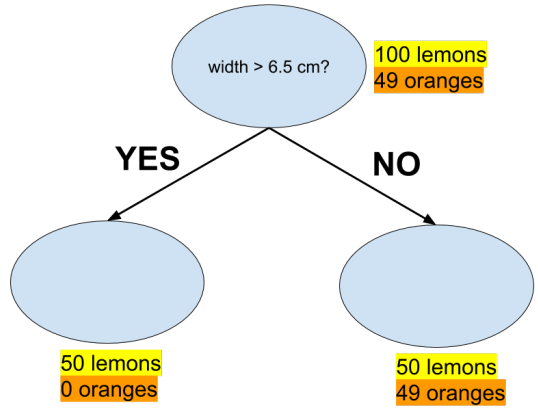
- How much information about cloudiness do we get by discovering whether it is raining?

$$\begin{aligned} IG(Y|X) &= H(Y) - H(Y|X) \\ &\approx 0.25 \text{ bits} \end{aligned}$$

- This is called the **information gain** in Y due to X, or the **mutual information** of Y and X
- If X is completely uninformative about Y: $IG(Y|X) = 0$
- If X is completely informative about Y: $IG(Y|X) = H(Y)$

Revisiting Our Original Example

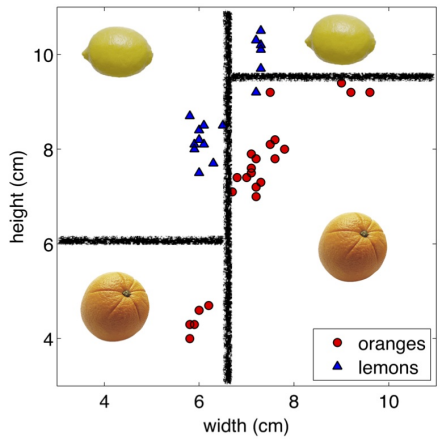
- Information gain measures the informativeness of a variable, which is exactly what we desire in a decision tree attribute!
- What is the information gain of this split?

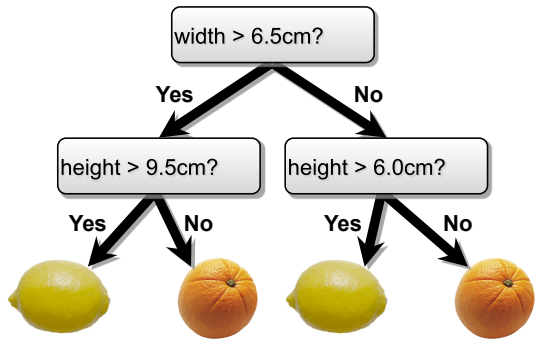


- Let Y be r.v. denoting lemon or orange, B be r.v. denoting whether left or right split taken, and treat counts as probabilities.
- Root entropy: $H(Y) = -\frac{49}{149}\log_2(\frac{49}{149}) - \frac{100}{149}\log_2(\frac{100}{149}) \approx 0.91$
- Leafs entropy: $H(Y|B = \text{left}) = 0$, $H(Y|B = \text{right}) \approx 1$.
- $IG(Y|B) = H(Y) - H(Y|B)$

$$= H(Y) - \{H(Y|B=\text{left})\mathbb{P}(B=\text{left}) + H(Y|B=\text{right})\mathbb{P}(B=\text{right})\}$$
$$\approx 0.91 - (0 \cdot \tfrac{1}{3} + 1 \cdot \tfrac{2}{3}) \approx 0.24 > 0$$

Constructing Decision Trees





- At each level, one must choose:
  1. Which variable to split.
  2. Possibly where to split it.
- Choose them based on how much information we would gain from the decision! (choose attribute that gives the **best** gain)

Decision Tree Construction Algorithm

- Simple, greedy, recursive approach, builds up tree node-by-node
- Start with empty decision tree and complete training set
  - Split on the most informative attribute, partitioning dataset
  - Recurse on subpartitions
- Possible termination condition: end if all examples in current subpartition share the same class

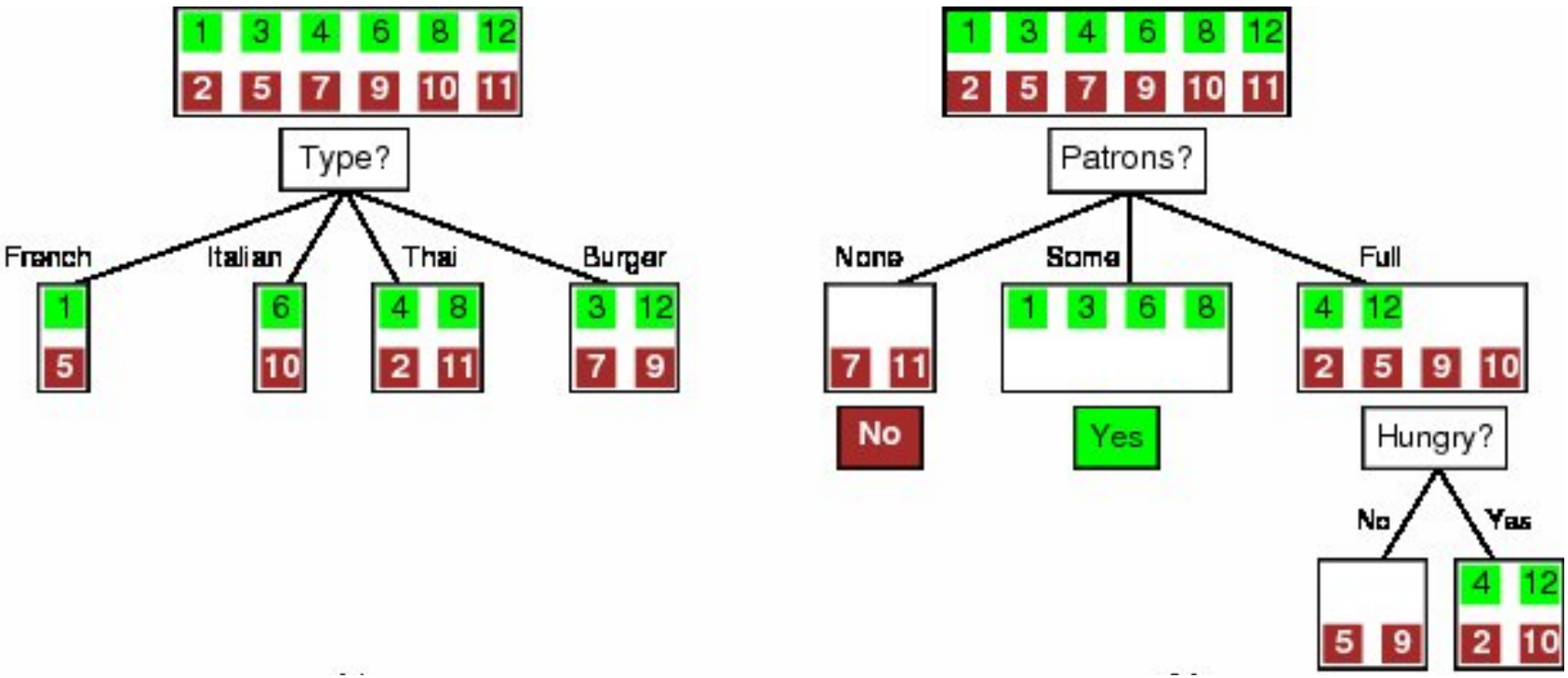
Back to Our Example

Example	Input Attributes										Goal
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
$\mathbf{x}_1$	*Yes*	*No*	*No*	*Yes*	*Some*	*$$$*	*No*	*Yes*	*French*	*0–10*	$y_1 = Yes$
$\mathbf{x}_2$	*Yes*	*No*	*No*	*Yes*	*Full*	*$*	*No*	*No*	*Thai*	*30–60*	$y_2 = No$
$\mathbf{x}_3$	*No*	*Yes*	*No*	*No*	*Some*	*$*	*No*	*No*	*Burger*	*0–10*	$y_3 = Yes$
$\mathbf{x}_4$	*Yes*	*No*	*Yes*	*Yes*	*Full*	*$*	*Yes*	*No*	*Thai*	*10–30*	$y_4 = Yes$
$\mathbf{x}_5$	*Yes*	*No*	*Yes*	*No*	*Full*	*$$$*	*No*	*Yes*	*French*	*>60*	$y_5 = No$
$\mathbf{x}_6$	*No*	*Yes*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Italian*	*0–10*	$y_6 = Yes$
$\mathbf{x}_7$	*No*	*Yes*	*No*	*No*	*None*	*$*	*Yes*	*No*	*Burger*	*0–10*	$y_7 = No$
$\mathbf{x}_8$	*No*	*No*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Thai*	*0–10*	$y_8 = Yes$
$\mathbf{x}_9$	*No*	*Yes*	*Yes*	*No*	*Full*	*$*	*Yes*	*No*	*Burger*	*>60*	$y_9 = No$
$\mathbf{x}_{10}$	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$$$*	*No*	*Yes*	*Italian*	*10–30*	$y_{10} = No$
$\mathbf{x}_{11}$	*No*	*No*	*No*	*No*	*None*	*$*	*No*	*No*	*Thai*	*0–10*	$y_{11} = No$
$\mathbf{x}_{12}$	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$*	*No*	*No*	*Burger*	*30–60*	$y_{12} = Yes$

Attributes:

1. Alternate: whether there is a suitable alternative restaurant nearby.
2. Bar: whether the restaurant has a comfortable bar area to wait in.
3. Fri/Sat: true on Fridays and Saturdays.
4. Hungry: whether we are hungry.
5. Patrons: how many people are in the restaurant (values are None, Some, and Full).
6. Price: the restaurant's price range ($, $$, $$$).
7. Raining: whether it is raining outside.
8. Reservation: whether we made a reservation.
9. Type: the kind of restaurant (French, Italian, Thai or Burger).
10. WaitEstimate: the wait estimated by the host (0-10 minutes, 10-30, 30-60, >60).

Attribute Selection

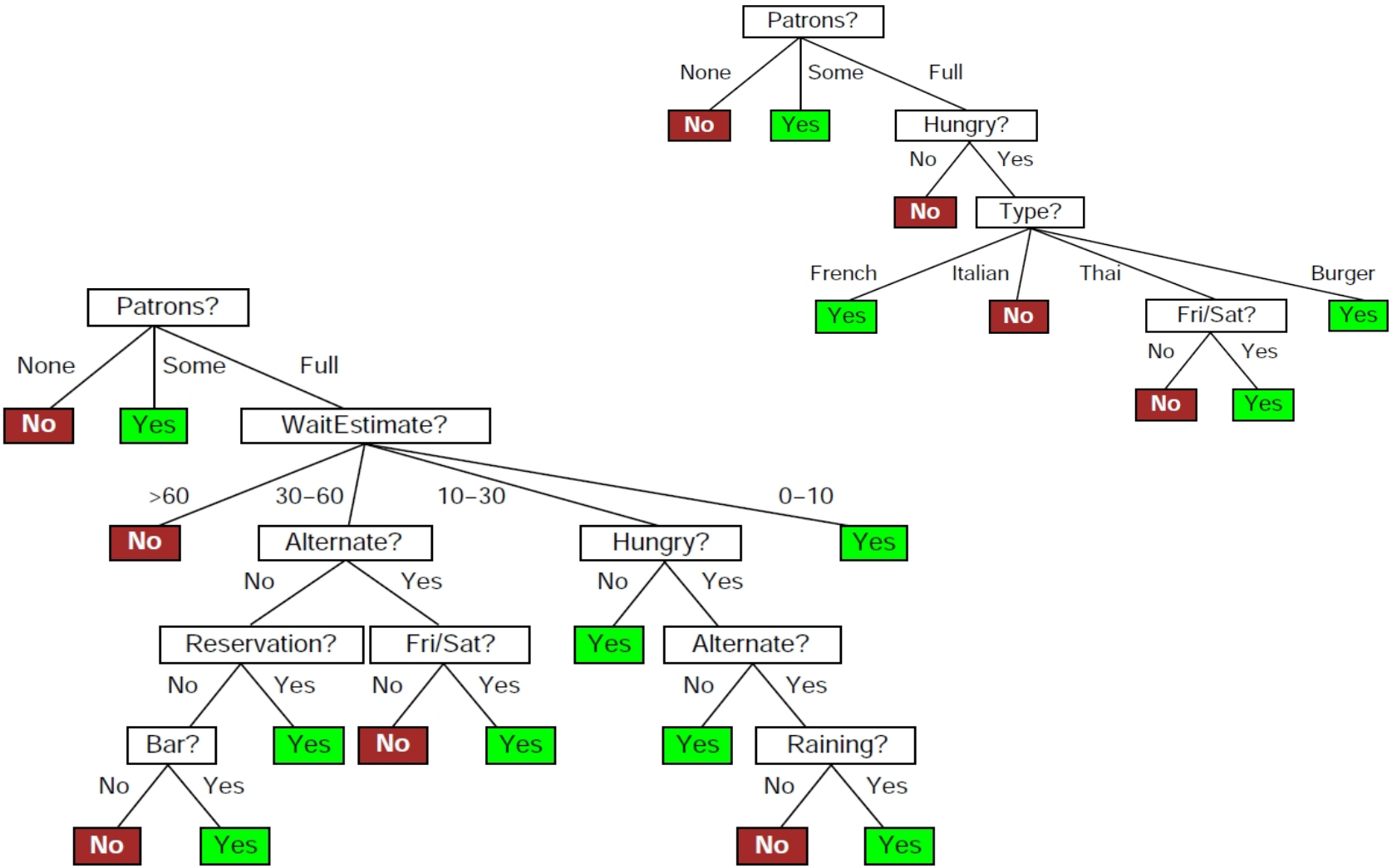


$$IG(Y) = H(Y) - H(Y|X)$$

$$IG(type) = 1 - \left[\frac{2}{12}H(Y|\text{Fr.}) + \frac{2}{12}H(Y|\text{It.}) + \frac{4}{12}H(Y|\text{Thai}) + \frac{4}{12}H(Y|\text{Bur.})\right] = 0$$

$$IG(Patrons) = 1 - \left[\frac{2}{12}H(Y|\text{None}) + \frac{4}{12}H(Y|\text{Some}) + \frac{6}{12}H(Y|\text{Full})\right] \approx 0.541$$

Which Tree is Better?



What Makes a Good Tree?

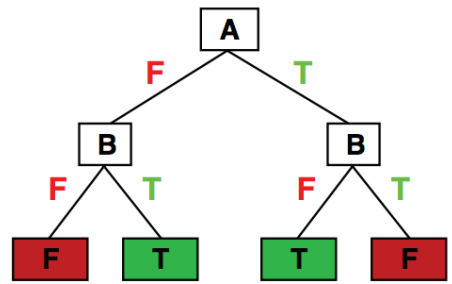
- Not too small: need to handle important but possibly subtle distinctions in data

- Not too big:
  - Computational efficiency (avoid redundant, spurious attributes)
  - Avoid over-fitting training examples
  - Human interpretability

Expressiveness

- **Discrete-input, discrete-output case**:
  - Decision trees can express any function of the input attributes
  - E.g., for Boolean functions, truth table row → path to leaf:

A	B	A xor B
F	F	F
F	T	T
T	F	T
T	T	F



- **Continuous-input, continuous-output case**:
  - Can approximate any function arbitrarily closely
- Trivially, there is a consistent decision tree for any training set w/ one path to leaf for each example but it probably won't generalize to new examples

[Slide credit: S. Russell]

Decision Tree Miscellany

- They produce highly interpretable models.

- Problems:
  - You have exponentially less data at lower levels
  - Too big of a tree can overfit the data
  - Greedy algorithms don't necessarily yield the global optimum
  - Mistakes at top-level propagate down tree

- Handling continuous attributes
  - Split based on a threshold, chosen to maximize information gain. There are efficient ways of doing this, which we don't cover in the lecture.

- Decision trees can also be used for regression on real-valued outputs. Choose splits to minimize squared error, rather than maximize information gain.

Ensemble methods: Brief overview

- An **ensemble** of predictors is a set of predictors whose individual decisions are combined in some way to predict new examples
  - E.g., (possibly weighted) majority vote
- For this to be nontrivial, the learned hypotheses must differ somehow, e.g.
  - Different algorithm
  - Different choice of hyperparameters
  - Trained on different data
  - Trained with different weighting of the training examples
- Ensembles are usually easy to implement. The hard part is deciding what kind of ensemble you want, based on your goals.

Bagging: Motivation

- Suppose we could somehow sample m independent training sets $\{\mathcal{D}_i\}_{i=1}^m$ from p_{dataset}.
- We could then learn a predictor $h_i := h_{\mathcal{D}_i}$ based on each one, and take the average $h = \frac{1}{m}\sum_{i=1}^m h_i$.
- How does this affect the terms of the expected loss (recall Bias-Variance trade off)?
  - **Bias: unchanged**, since the averaged prediction has the same expectation

$$\mathbb{E}_{\mathcal{D}_1,\ldots,\mathcal{D}_m \overset{iid}{\sim} p_{\text{dataset}}}[h(x)] = \frac{1}{m}\sum_{i=1}^m \mathbb{E}_{\mathcal{D}_i \sim p_{\text{dataset}}}[h_i(x)] = \mathbb{E}_{\mathcal{D}\sim p_{\text{dataset}}}[h_{\mathcal{D}}(x)]$$

  - **Variance: reduced**, since we're averaging over independent samples

$$\operatorname*{Var}_{\mathcal{D}_1,\ldots,\mathcal{D}_m}[h(x)] = \frac{1}{m^2}\sum_{i=1}^m \operatorname*{Var}_{\mathcal{D}_i}[h_i(x)] = \frac{1}{m}\operatorname*{Var}_{\mathcal{D}}[h_{\mathcal{D}}(x)].$$

Bagging: The Idea

- In practice, we don't have access to the underlying data generating distribution p_{sample}.
- It is expensive to independently collect many datasets.
- Solution: **bootstrap aggregation**, or **bagging**.
  - Take a single dataset $\mathcal{D}$ with n examples.
  - Generate m new datasets, each by sampling n training examples from $\mathcal{D}$, with replacement.
  - Average the predictions of models trained on each of these datasets.

Bagging: The Idea

- Problem: the datasets are not independent, so we don't get the $1/m$ variance reduction.
  - Possible to show that if the sampled predictions have variance σ^2 and correlation ρ, then

$$\text{Var}\left(\frac{1}{m}\sum_{i=1}^{m} h_i(x)\right) = \frac{1}{m}(1-\rho)\sigma^2 + \rho\sigma^2.$$

- Ironically, it can be advantageous to introduce *additional* variability into your algorithm, as long as it reduces the correlation between samples.
  - Intuition: you want to invest in a diversified portfolio, not just one stock.
  - Can help to use average over multiple algorithms, or multiple configurations of the same algorithm.

Random Forests

- **Random forests** = bagged decision trees, with one extra trick to decorrelate the predictions
- When choosing each node of the decision tree, choose a random set of d input features, and only consider splits on those features
- This extra randomness helps the algorithm generalize better.
- Random forests are probably the best black-box machine learning algorithm — they often work well with no tuning whatsoever.
  - one of the most widely used algorithms in Kaggle competitions