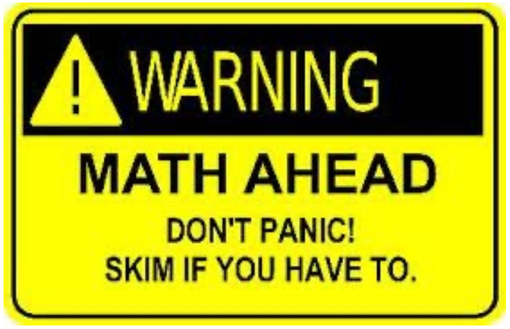


LMP 1210H: Basic Principles of Machine Learning in Biomedical Research

Bo Wang
AI Lead Scientist, PMCC, UHN
CIFAR AI Chair, Vector Institute
Assistant Professor, University of Toronto

UHN Peter Munk Cardiac Centre

Administrative Details

1. Homework
    a. Homework 1 is out! Due: **Feb 2, 9:59am**
    b. Homework 2 will be released on **Feb 2**

2. Final Project
    a. Team: 2-3 students
    b. Proposal due: **Feb 20.**
    c. Visit the office hour before submitting the proposal
    d. Proposal will **not** be graded
    e. Team without proposal submissions **cannot** participate in the final project.

UHN Peter Munk Cardiac Centre

More on the final projects

1. How to form a good team?

   Keywords: Multidisciplinary, Open Communications, Piazza

2. How to write a good proposal?

   Keywords: Concise, Experimental Design, Office Hours

3. How to provide a good presentation?

   Keywords: Clear Structures, Memorable

UHN Peter Munk Cardiac Centre

Recap: K - Nearest Neighbors (KNN)

- Nearest neighbors sensitive to noise or mis-labeled data ("class noise"). Solution?
- Smooth by having k nearest neighbors vote

Algorithm (kNN):

1. Find k examples $\{\mathbf{x}^{(i)}, t^{(i)}\}$ closest to the test instance $\mathbf{x}$
2. Classification output is majority class

$$y = \underset{t^{(z)}}{\operatorname{argmax}} \sum_{i=1}^{k} \mathbb{I}\{t^{(z)} = t^{(i)}\}$$

$\mathbb{I}\{\text{statement}\}$ is the identity function and is equal to one whenever the statement is true. We could also write this as $\delta(t^{(z)}, t^{(i)})$ with $\delta(a, b) = 1$ if $a = b$, 0 otherwise. $\mathbb{I}\{1\}$.

UHN Peter Munk Cardiac Centre

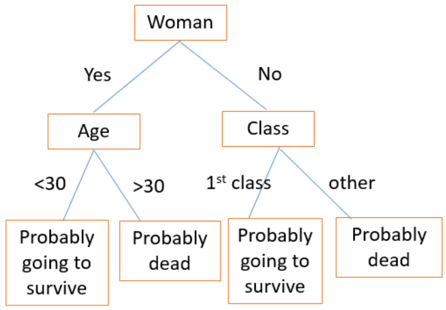
Recap: K - Nearest Neighbors (KNN)

- Simple algorithm that does all its work at test time — in a sense, no learning!
- Can be used for regression too, which we encounter later.
- Can control the complexity by varying k
- Suffers from the Curse of Dimensionality

Decision Trees

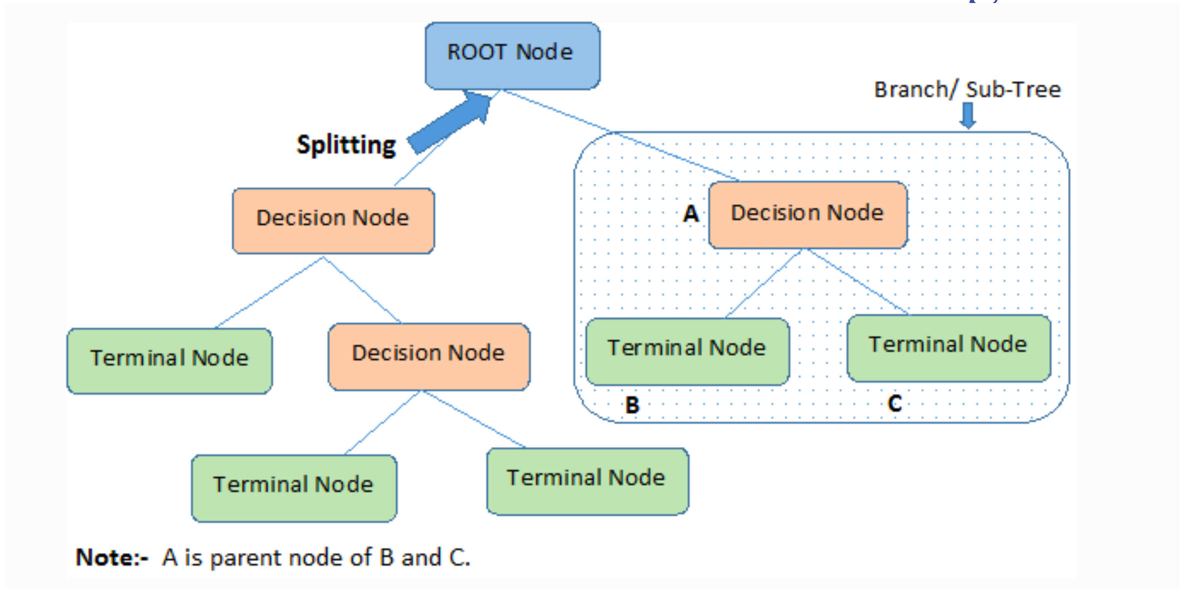
A normal tree

A decision tree



- **Decision Trees**
  - Simple but powerful learning algorithm
  - One of the most widely used learning algorithms in Kaggle competitions
  - Lets us introduce ensembles, a key idea in ML
- Useful information theoretic concepts (entropy, mutual information, etc.)

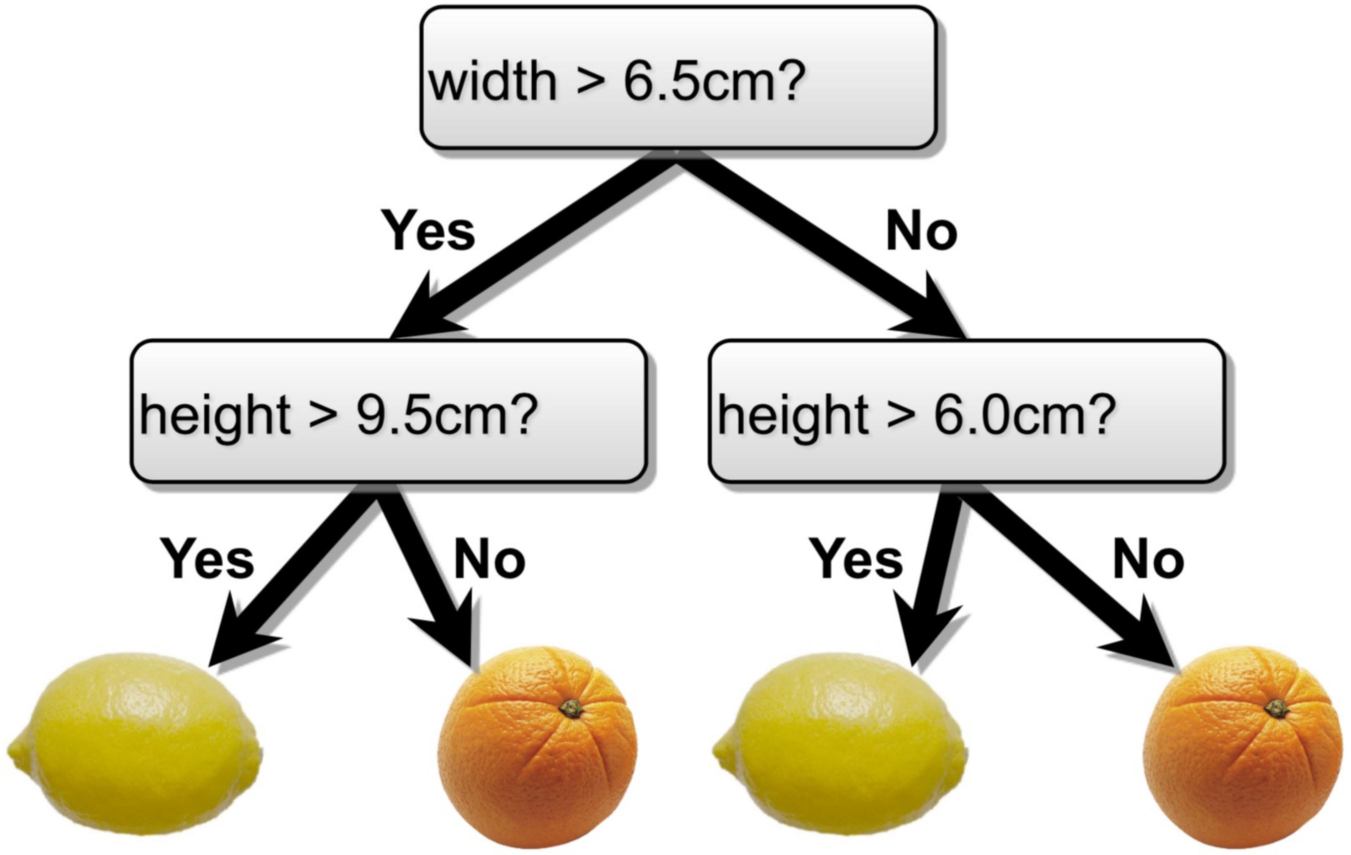
Decision Trees : Basic Terminologies



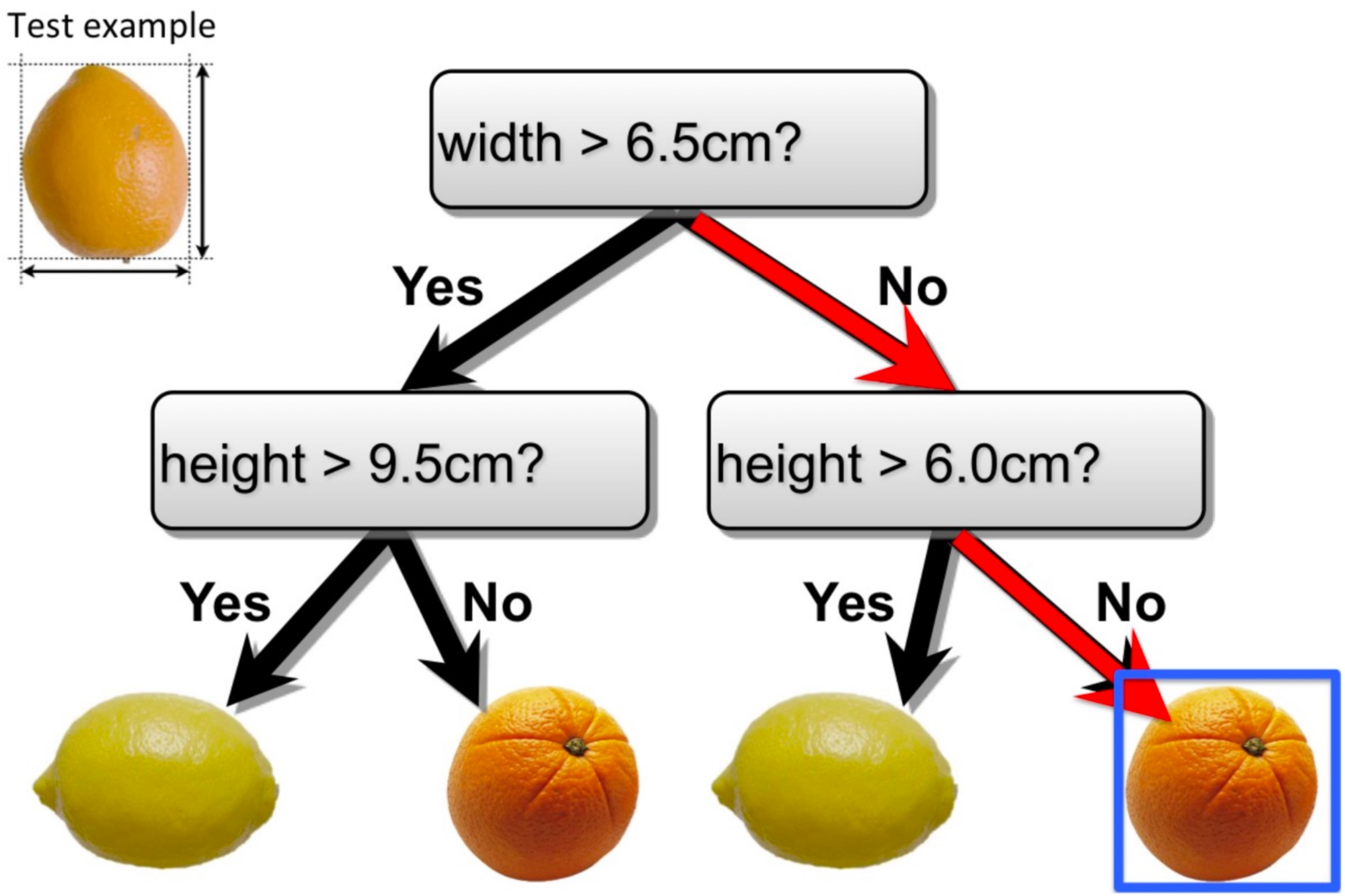
1. **Root Node:** This attribute is used for dividing the data into two or more sets. The feature attribute in this node is selected based on Attribute Selection Techniques.
2. **Branch or Sub-Tree:** A part of the entire decision tree is called a branch or sub-tree.
3. **Splitting:** Dividing a node into two or more sub-nodes based on if-else conditions.
4. **Decision Node:** After splitting the sub-nodes into further sub-nodes, then it is called the decision node.
5. **Leaf or Terminal Node:** This is the end of the decision tree where it cannot be split into further sub-nodes.

Decision Trees: A simple example

- **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



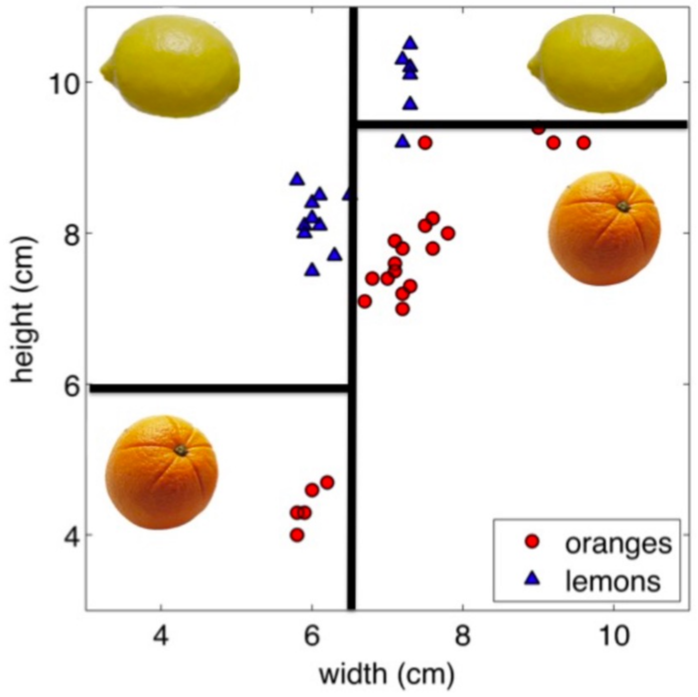
Decision Trees: A simple example

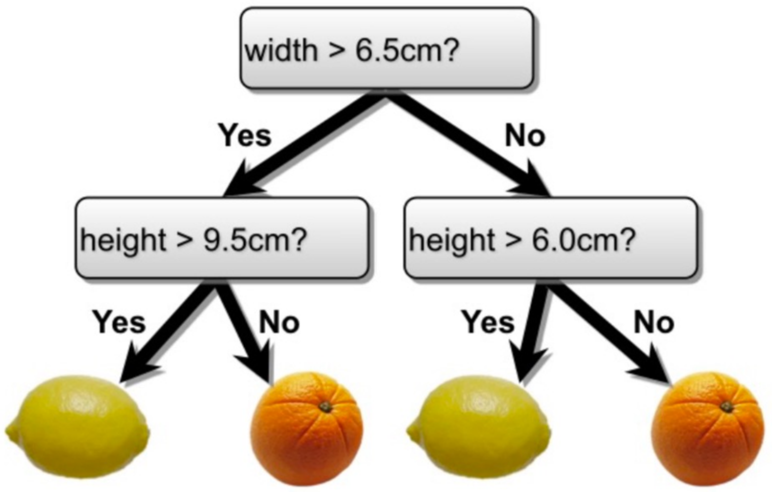


UHN Cardiac Centre

Decision Trees: A simple example

- 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





Decision Trees: Wait or Not

You can tell a lot about a fellow's character by his way of eating jellybeans.

----Ronald Reagan

You can tell a lot about a fellow's character by his way of eating at a restaurant.

----Bo Wang

UHN Peter Munk Cardiac Centre

Decision Trees: Wait or Not

- What if the attributes are discrete?

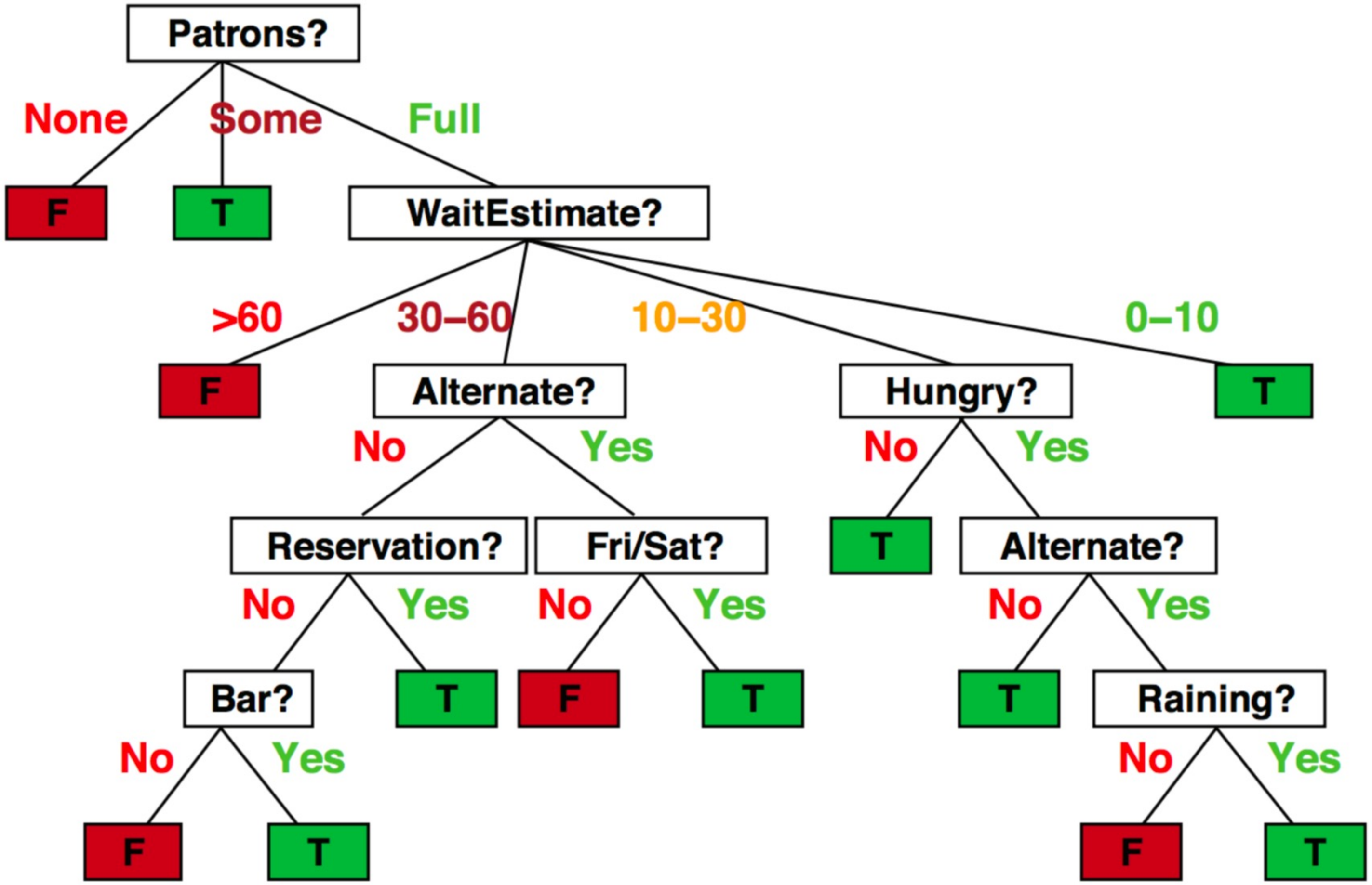
Example	Input Attributes										Goal
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
$\mathbf{x}_1$	Yes	No	No	Yes	Some	\$\$\$	No	Yes	French	0–10	$y_1 =$ Yes
$\mathbf{x}_2$	Yes	No	No	Yes	Full	\$	No	No	Thai	30–60	$y_2 =$ No
$\mathbf{x}_3$	No	Yes	No	No	Some	\$	No	No	Burger	0–10	$y_3 =$ 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

Attributes:

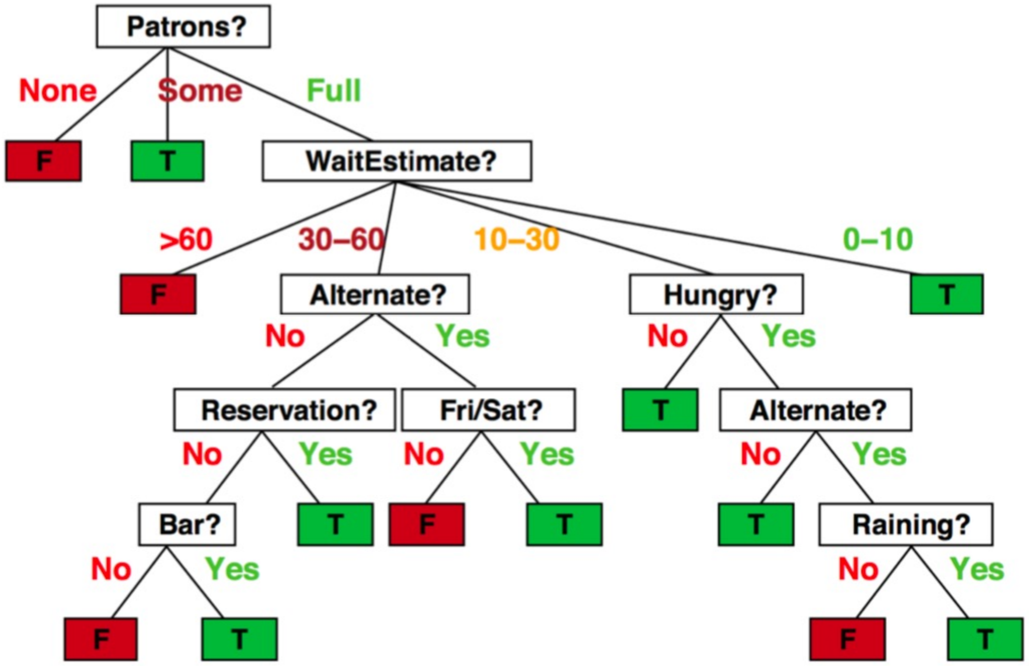
1.	Alternate: whether there is a suitable alternative restaurant nearby.
2.	Bar: whether the restaurant has a comfortable bar area to wait in.
3.	Fri/Sat: true on Fridays and Saturdays.
4.	Hungry: whether we are hungry.
5.	Patrons: how many people are in the restaurant (values are None, Some, and Full).
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).

Decision Trees: Wait or Not

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



Decision Trees: Wait or Not

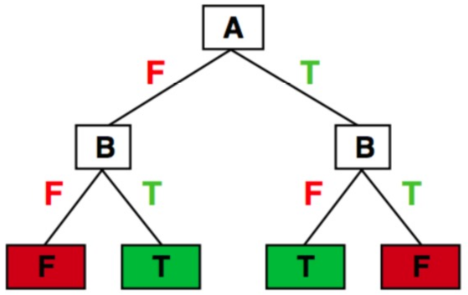


- **Internal nodes** test **attributes**
- **Branching** is determined by **attribute value**
- **Leaf nodes** are **outputs** (predictions)

Decision Trees

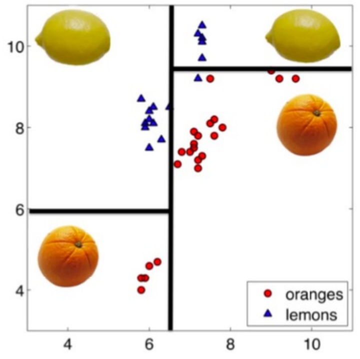
- **Discrete-input, discrete-output case:**
  - Decision trees can express any function of the input attributes
  - Example: For Boolean functions, the truth table row $\rightarrow$ path to leaf

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 (unless f nondeterministic in x) but it probably won't generalize to new examples

Decision Trees



- 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 $\{t^{(m_1)}, \ldots, t^{(m_k)}\}$

How to ***learn*** the trees

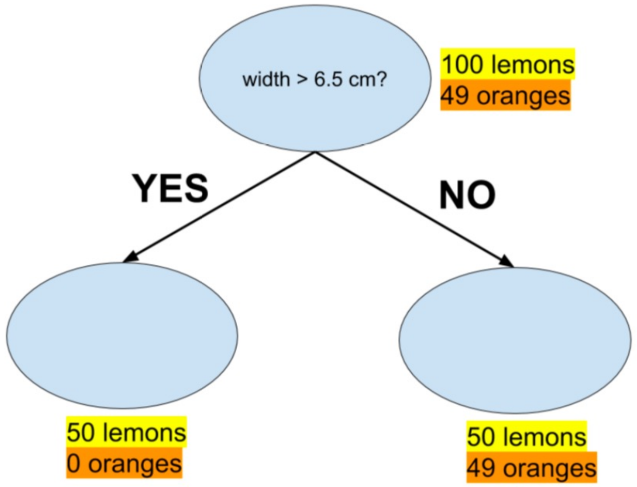
Learning the simplest (smallest) decision tree which correctly classifies training set is an NP complete problem (if you are interested, check: Hyafil & Rivest'76).

- 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: misclassification error
  - Say region R is split in R_1 and R_2 based on loss $L(R)$.
  - Accuracy gain is $L(R) - \frac{|R_1|L(R_1)+|R_2|L(R_2)}{|R_1|+|R_2|}$

How to ***learn*** the trees

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



- Is this split good? Zero accuracy gain

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

- But we have reduced our uncertainty about whether a fruit is a lemon!

How to *learn* the trees

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

Entropy: a way to measure uncertainty

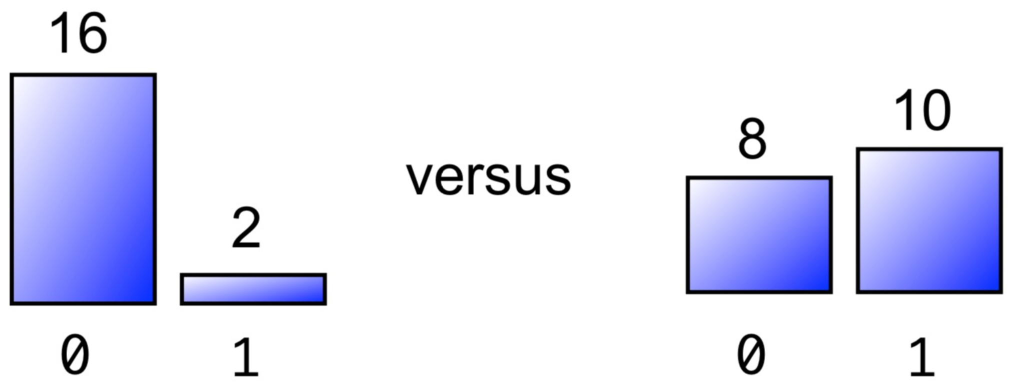
Q: Which coin is more uncertain?

Sequence 1:

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

Sequence 2:

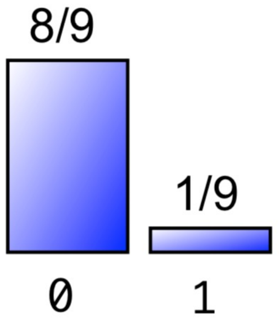
```
0 1 0 1 0 1 1 1 0 1 0 0 1 1 0 1 0 1 ... ?
```

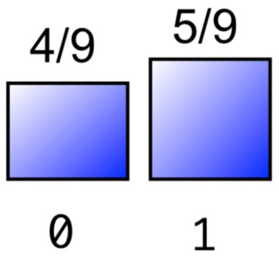


Entropy: a way to measure uncertainty

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

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





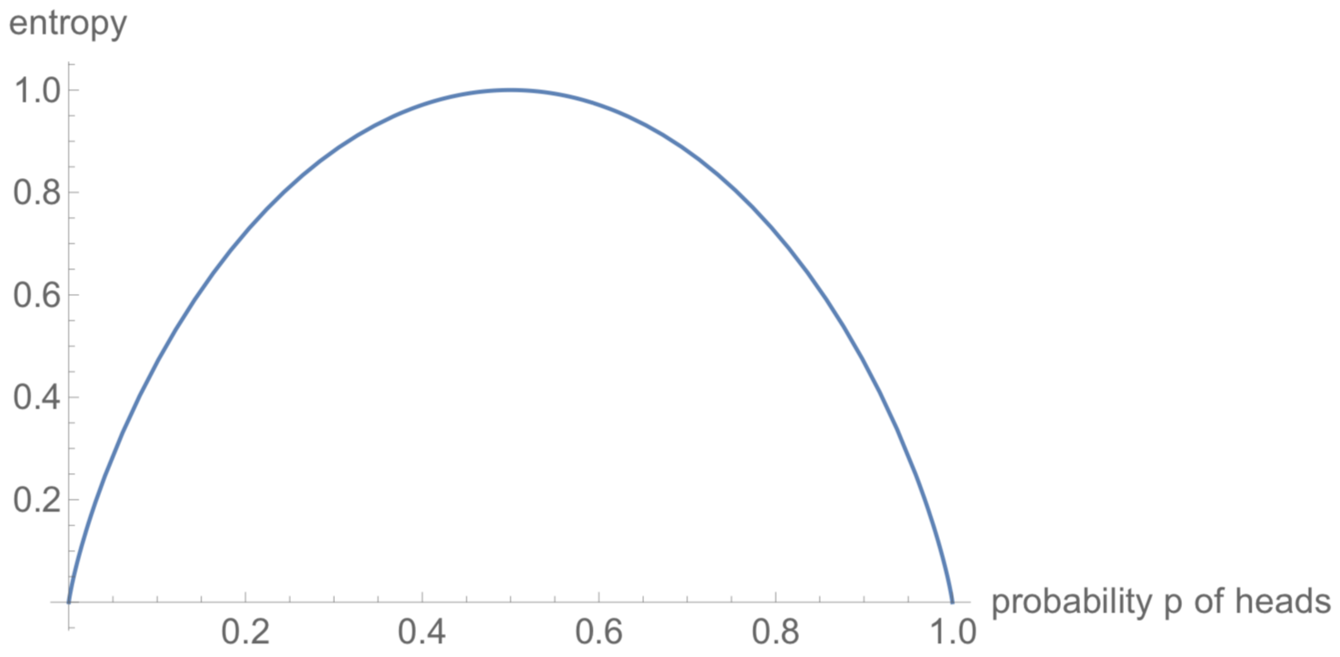
$$-\frac{8}{9}\log_2\frac{8}{9} - \frac{1}{9}\log_2\frac{1}{9} \approx \frac{1}{2}$$

$$-\frac{4}{9}\log_2\frac{4}{9} - \frac{5}{9}\log_2\frac{5}{9} \approx 0.99$$

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

Entropy: a way to measure uncertainty

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



Entropy: a way to measure uncertainty

- **“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

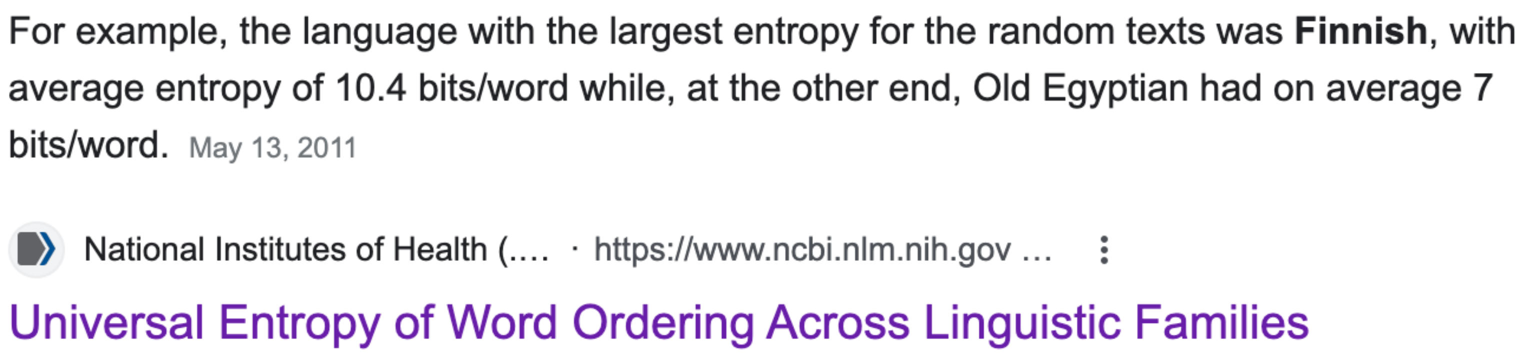
What is the entropy of *English*?

27 characters (A-Z, space).

100,000 words (avg 5.5 characters each)

- Assuming independence between successive characters:
  - uniform character distribution: $\log 27 = 4.75$ bits/character
  - true character distribution: 4.03 bits/character

Fun Fact: Which language has the largest entropy?



For example, the language with the largest entropy for the random texts was **Finnish**, with average entropy of 10.4 bits/word while, at the other end, Old Egyptian had on average 7 bits/word. May 13, 2011

National Institutes of Health (.... · https://www.ncbi.nlm.nih.gov ...

Universal Entropy of Word Ordering Across Linguistic Families

Entropy: a way to measure uncertainty

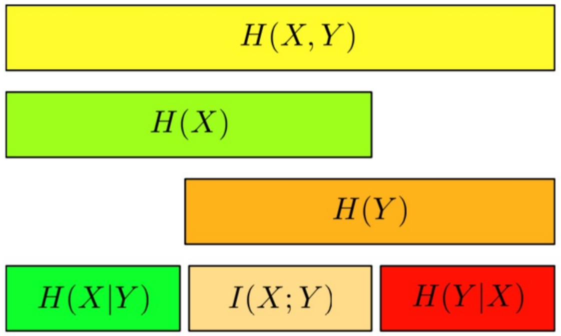
- Example: $X = \{\text{Raining, Not raining}\}$, $Y = \{\text{Cloudy, Not cloudy}\}$

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

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

UHN Peter Munk Cardiac Centre

Entropy: a way to measure uncertainty

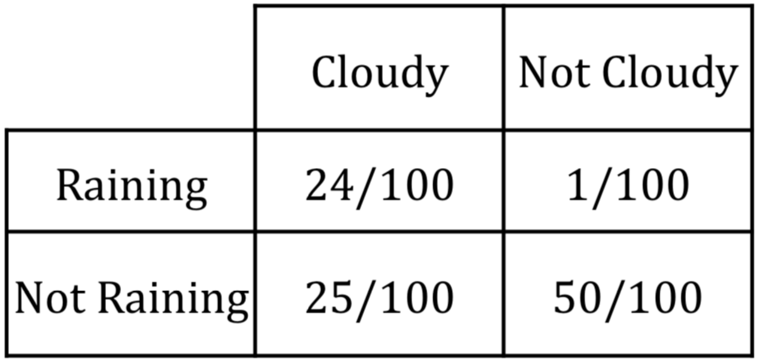


- Some useful properties for the discrete case:
  - H is always non-negative.
  - Chain rule: $H(X,Y) = H(X|Y) + H(Y) = H(Y|X) + H(X)$.
  - If X and Y independent, then X does not tell us anything about Y: $H(Y|X) = H(Y)$.
  - If X and Y independent, then $H(X,Y) = H(X) + H(Y)$.
  - But Y tells us everything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

Exercise: Verify these!

The figure is reproduced from Fig 8.1 of MacKay, Information Theory, Inference, and ... .

Entropy: a way to measure uncertainty



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

- The expected conditional entropy:

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

Entropy: a way to measure uncertainty

- 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

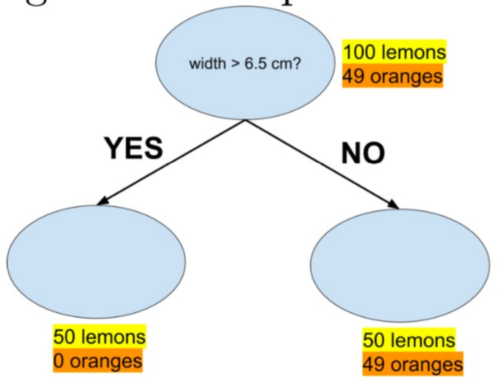
- What is the entropy of cloudiness, given the knowledge of whether or not it is raining?

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

UHN Peter Munk Cardiac Centre

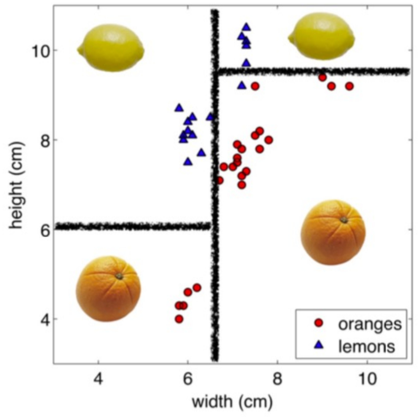
Learn the Decision Trees with Entropy

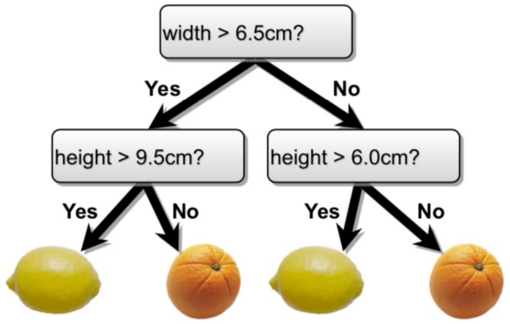
- Information gain measures the informativeness of a variable, which is exactly what we desire in a decision tree attribute!
- What is the information gain of this split?



- 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: $H(Y|B = \text{left}) = 0$, $H(Y|B = \text{right}) \approx 1$.
- $$\begin{aligned} IG(Y|B) &= H(Y) - H(Y|B) \\ &= H(Y) - \{H(Y|B{=}\text{left})\mathbb{P}(B{=}\text{left}) + H(Y|B{=}\text{right})\mathbb{P}(B{=}\text{right})\} \\ &\approx 0.91 - (0 \cdot \tfrac{1}{3} + 1 \cdot \tfrac{2}{3}) \approx 0.24 > 0 \end{aligned}$$

Learn the Decision Trees with Entropy



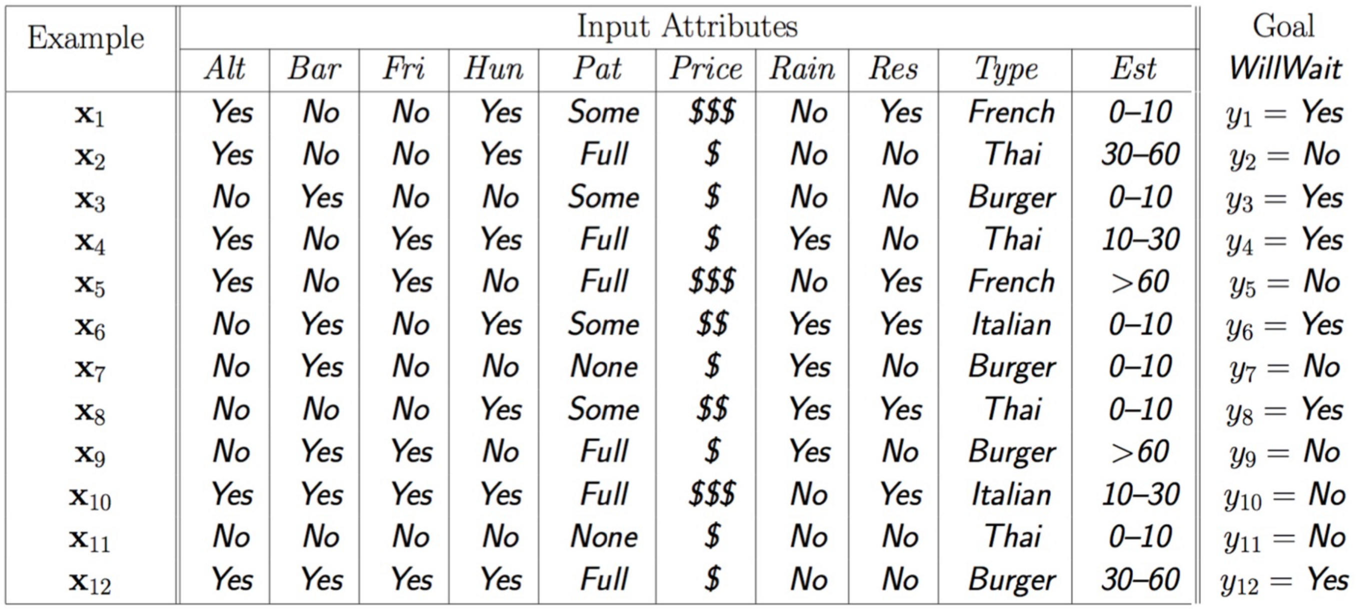


- At each level, one must choose:
  1. Which variable to split.
  2. Possibly where to split it.
- Choose them based on how much information we would gain from the decision! (choose attribute that gives the **best** gain)

Learn the Decision Trees with Entropy

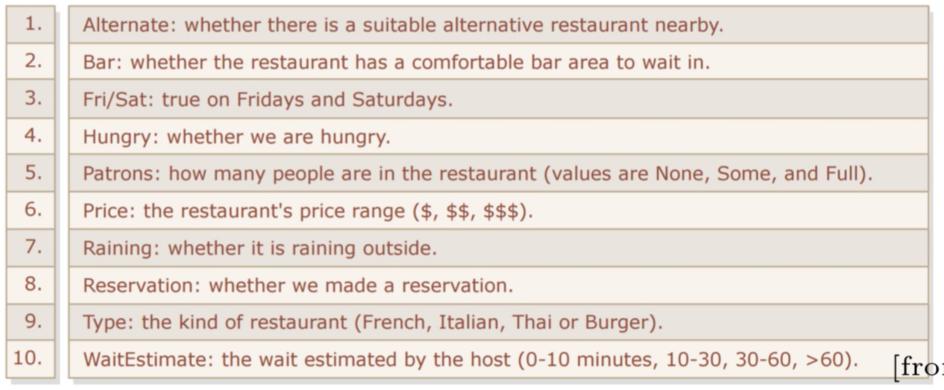
- Simple, greedy, recursive approach, builds up tree node-by-node
- Start with empty decision tree and complete training set
    - Split on the most informative attribute, partitioning dataset
    - Recurse on subpartitions
- Possible termination condition: end if all examples in current subpartition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

Get Back to the Restaurant: Wait or Not



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:

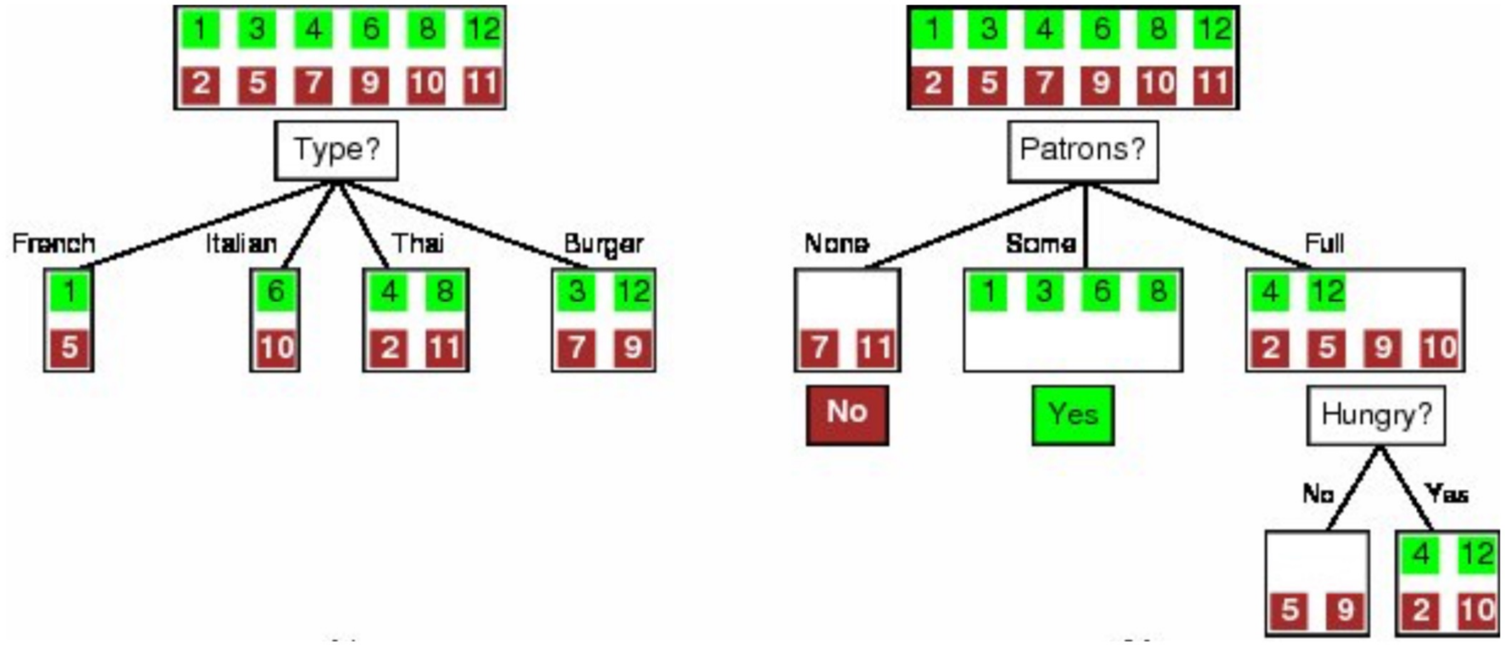


1.	Alternate: whether there is a suitable alternative restaurant nearby.
2.	Bar: whether the restaurant has a comfortable bar area to wait in.
3.	Fri/Sat: true on Fridays and Saturdays.
4.	Hungry: whether we are hungry.
5.	Patrons: how many people are in the restaurant (values are None, Some, and Full).
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).

[from: Russell & Norvig]

Get Back to the Restaurant: Wait or Not

Which attribute is better?



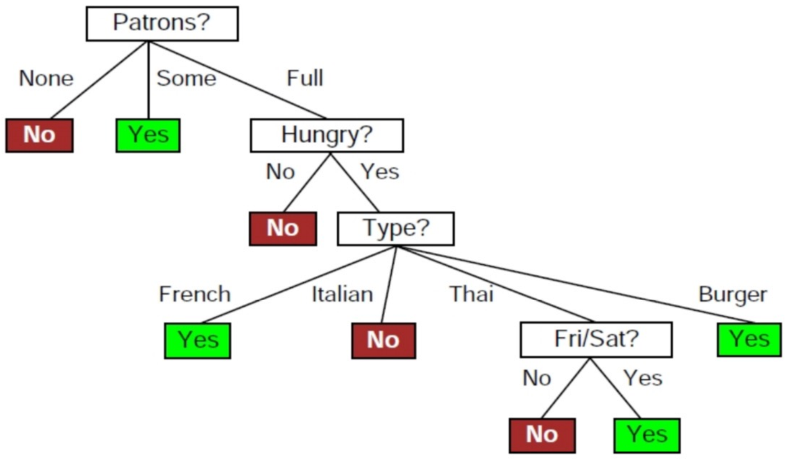
$$IG(Y) = H(Y) - H(Y|X)$$

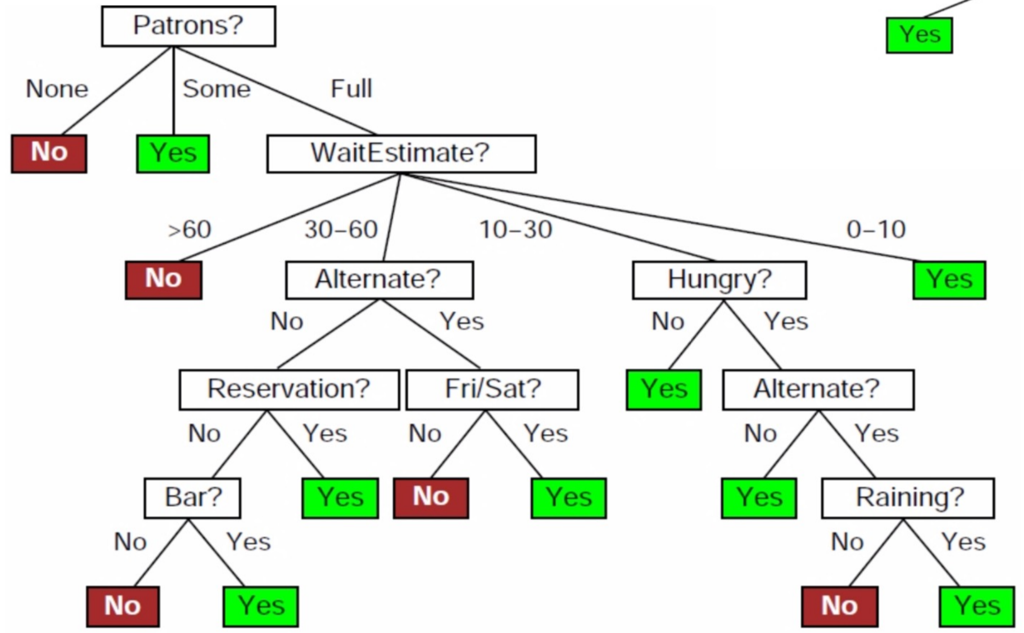
$$IG(type) = 1 - \left[\frac{2}{12}H(Y|\text{Fr.}) + \frac{2}{12}H(Y|\text{It.}) + \frac{4}{12}H(Y|\text{Thai}) + \frac{4}{12}H(Y|\text{Bur.})\right] = 0$$

$$IG(Patrons) = 1 - \left[\frac{2}{12}H(0,1) + \frac{4}{12}H(1,0) + \frac{6}{12}H(\frac{2}{6},\frac{4}{6})\right] \approx 0.541$$

Get Back to the Restaurant: Wait or Not

Which tree is better?

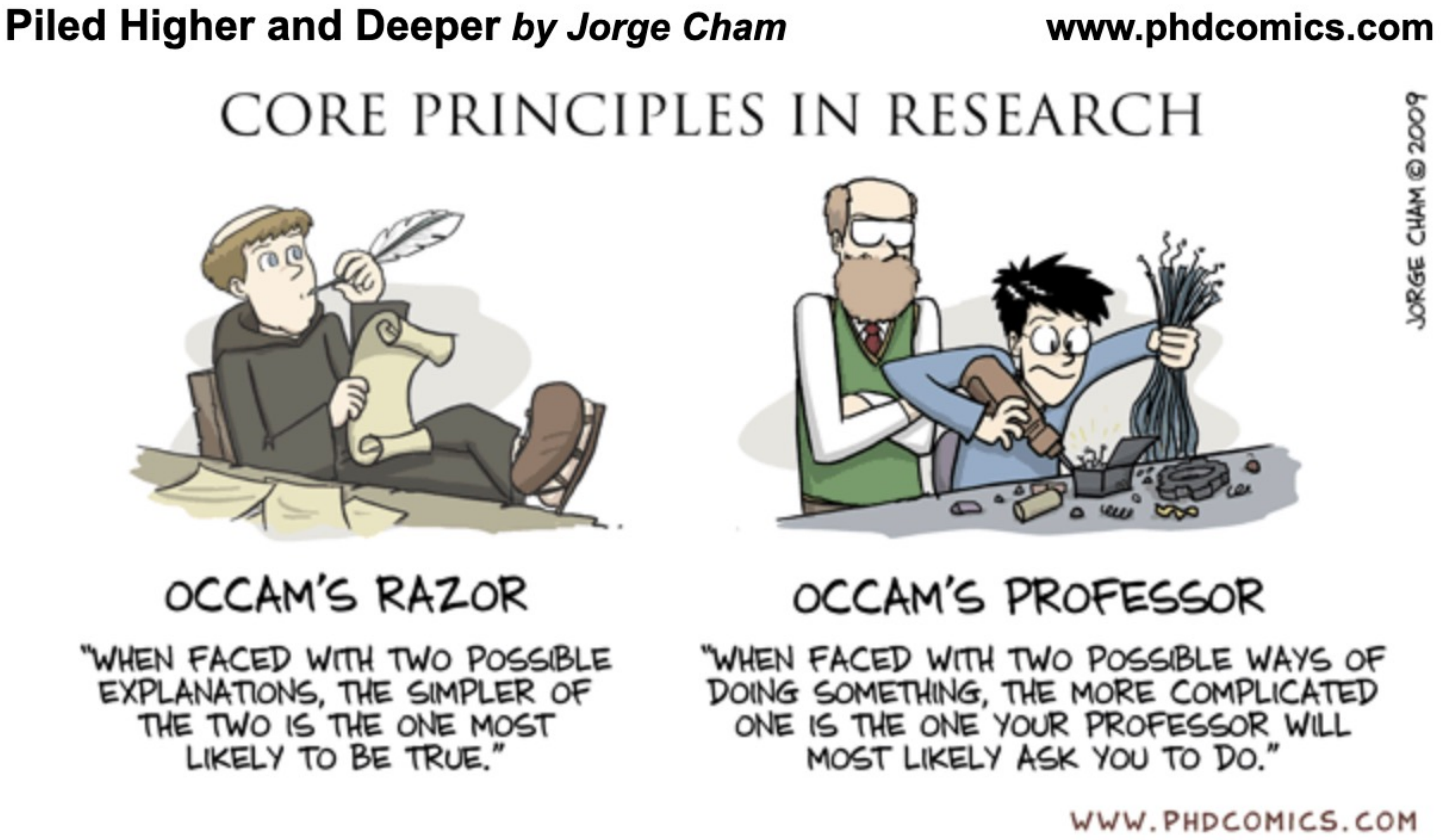




Model Selection: Which Tree is Better?

- 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
- "Occam's Razor": find the simplest hypothesis that fits the observations
  - Useful principle, but hard to formalize (how to define simplicity?)
  - See Domingos, 1999, "The role of Occam's razor in knowledge discovery"
- We desire small trees with informative nodes near the root

Model Selection: Occam's Razor


Piled Higher and Deeper by Jorge Cham
www.phdcomics.com
CORE PRINCIPLES IN RESEARCH
JORGE CHAM © 2009
OCCAM'S RAZOR
"WHEN FACED WITH TWO POSSIBLE EXPLANATIONS, THE SIMPLER OF THE TWO IS THE ONE MOST LIKELY TO BE TRUE."
OCCAM'S PROFESSOR
"WHEN FACED WITH TWO POSSIBLE WAYS OF DOING SOMETHING, THE MORE COMPLICATED ONE IS THE ONE YOUR PROFESSOR WILL MOST LIKELY ASK YOU TO DO."
WWW.PHDCOMICS.COM

title: "Core Principles" - originally published 10/12/2009

UHN Cardiac Centre

Decision Trees: Pitfalls

- Problems:
  - You have exponentially less data at lower levels
  - A large 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 other criteria used to measure the quality of a split, e.g., Gini index
- Trees can be pruned in order to make them less complex
- Decision trees can also be used for regression on real-valued outputs. Choose splits to minimize squared error, rather than maximize information gain.

Decision Trees v.s. KNN

Advantages of decision trees over k-NN

- Good with discrete attributes
- Easily deals with missing values (just treat as another value)
- Robust to scale of inputs; only depends on ordering
- Good when there are lots of attributes, but only a few are important
- Fast at test time
- More interpretable

Decision Trees v.s. KNN

Advantages of k-NN over decision trees

- Able to handle attributes/features that interact in complex ways
- Can incorporate interesting distance measures, e.g., shape contexts.

Decision Trees in Healthcare

Fun Facts:

Decision trees is the most widely-adopted method in healthcare.

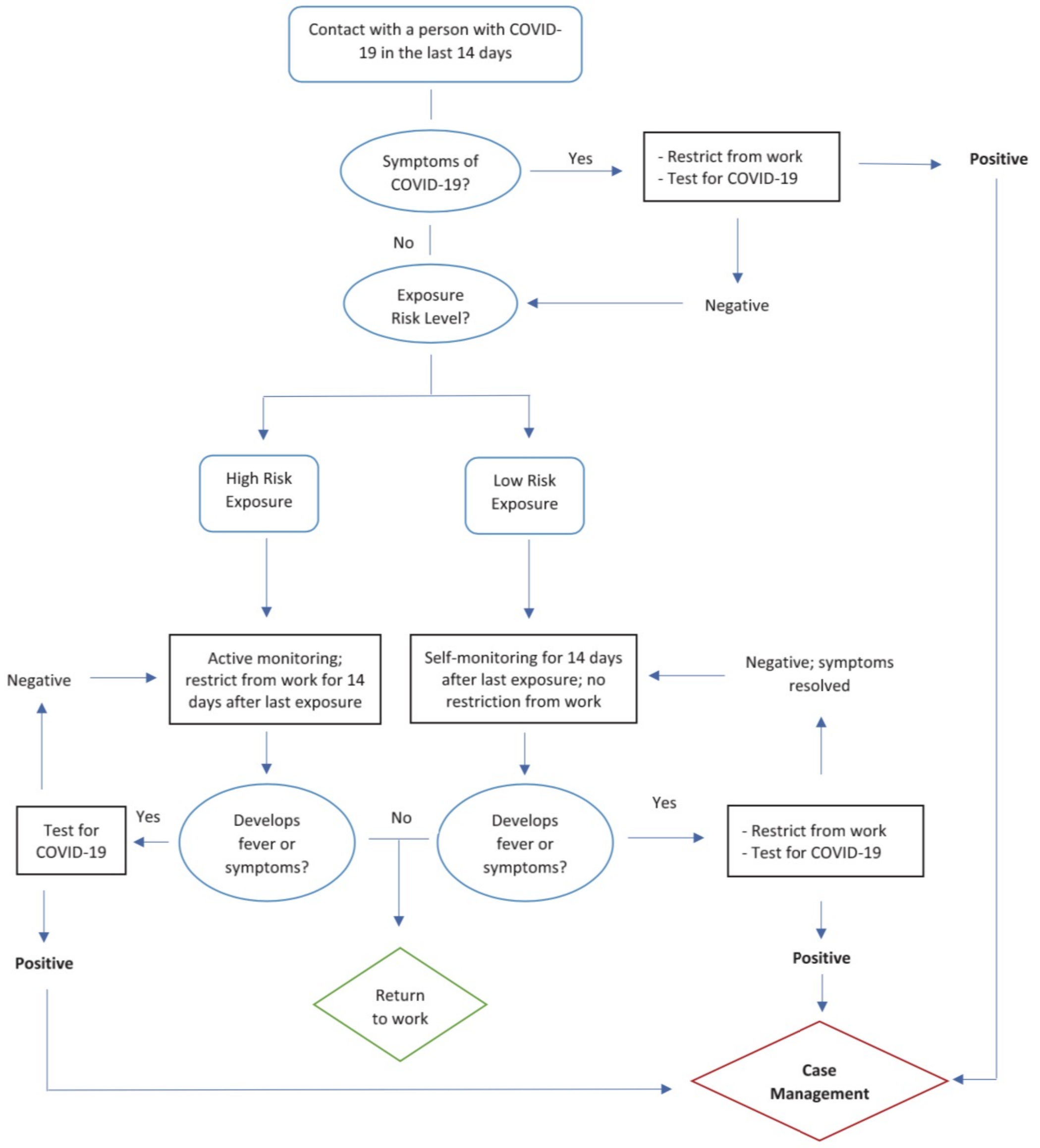
Class Discussion:

Name one application of decision trees in healthcare and discuss its pros and cons.

Decision Trees in COVID Management

Figure: Flowchart for management of HCWs with exposure to a person with COVID-19

Source: https://www.cdc.gov/



Contact with a person with COVID-19 in the last 14 days
Symptoms of COVID-19?
Yes
- Restrict from work
- Test for COVID-19
Positive
No
Negative
Exposure Risk Level?
High Risk Exposure
Low Risk Exposure
Active monitoring; restrict from work for 14 days after last exposure
Self-monitoring for 14 days after last exposure; no restriction from work
Negative
Negative; symptoms resolved
Test for COVID-19
Yes
Develops fever or symptoms?
No
Develops fever or symptoms?
Yes
- Restrict from work
- Test for COVID-19
Positive
Return to work
Positive
Case Management

Peter Mu
Cardiac
Centre

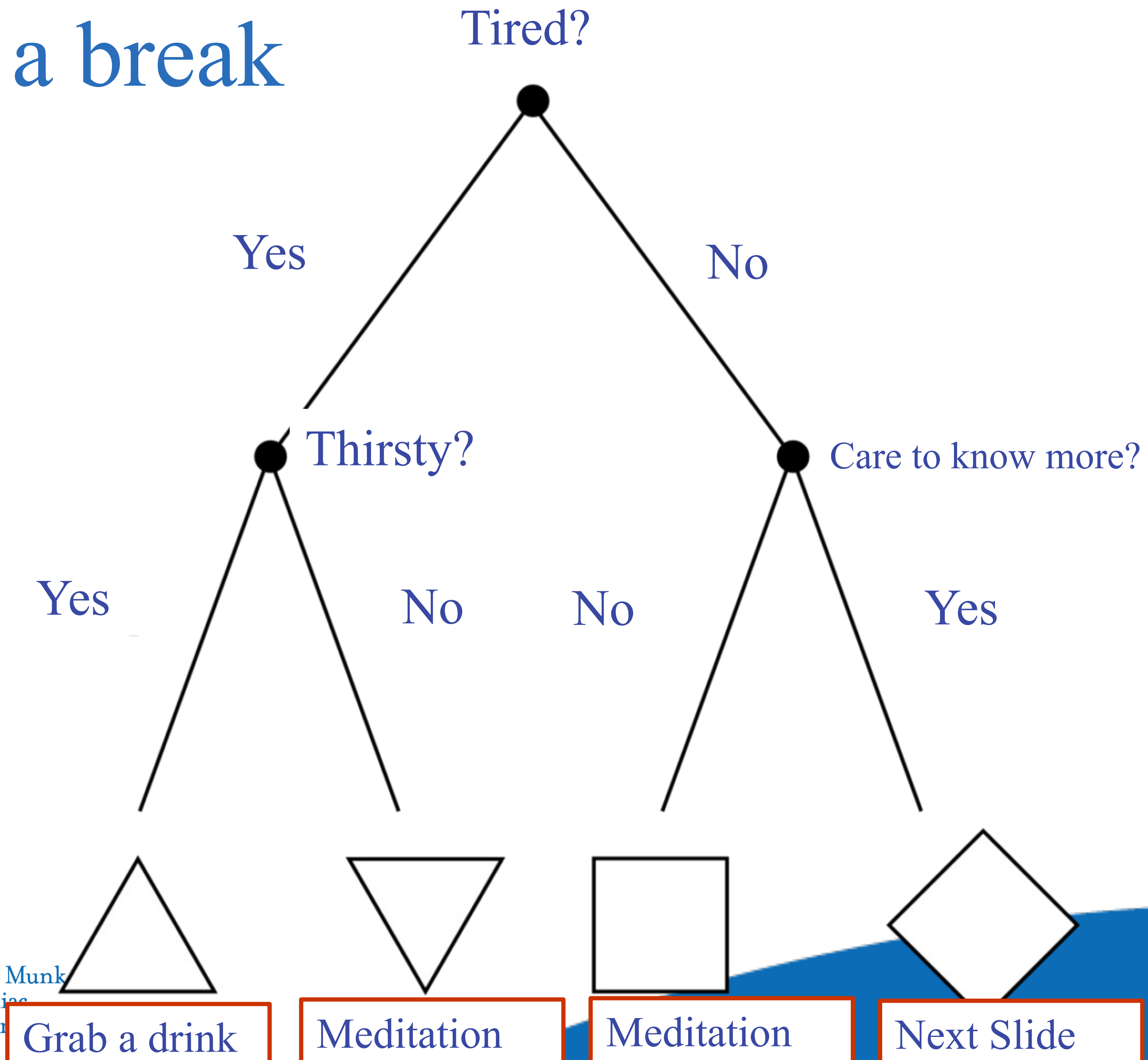
Decision Trees : Take-Home Messages

1. Decision Trees is simple and interpretable.

2. Decision Trees uses information gains to ***learn*** to split the trees.

3. Decision Trees tends to overfit

Take a break

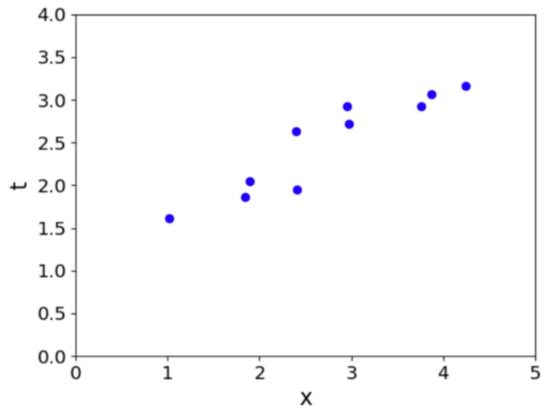


UHN Peter Munk Cardiac Centre

Linear Models

- So far, we have talked about *procedures* for learning.
  - KNN and decision trees.
- For the remainder of this course, we will take a more modular approach:
  - choose a model describing the relationships between variables of interest
  - define a loss function quantifying how bad the fit to the data is
  - choose a regularizer saying how much we prefer different candidate models (or explanations of data)
  - fit the model that minimizes the loss function and satisfy the constraint/penalty imposed by the regularizer, possibly using an optimization algorithm
- Mixing and matching these modular components gives us a lot of new ML methods.

Linear Models -- Problem Setup



Recall that in supervised learning:

- There is target $t \in \mathcal{T}$ (also called response, outcome, output, class)
- There are features $\mathbf{x} \in \mathcal{X}$ (also called inputs and covariates)
- Objective is to learn a function $f : \mathcal{X} \to \mathcal{T}$ such that

$$t \approx y = f(x)$$

based on some data $\mathcal{D} = \{(\mathbf{x}^{(i)}, t^{(i)}) \text{ for } i = 1, 2, ..., N\}$.

Linear Models -- Problem Setup

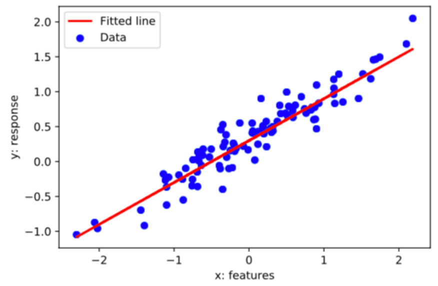
- Model: In linear regression, we use linear functions of the inputs $\mathbf{x} = (x_1, \ldots, x_D)$ to make predictions y of the target value t:

$$y = f(\mathbf{x}) = \sum_j w_j x_j + b$$

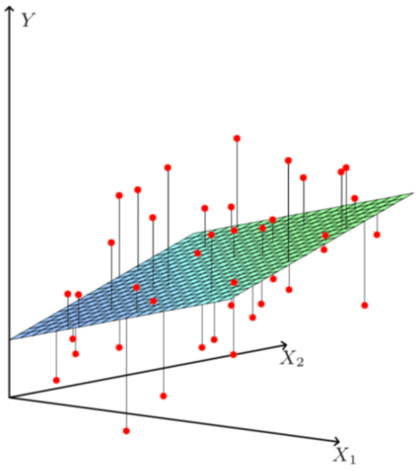
  - y is the prediction
  - $\mathbf{w}$ is the weights
  - b is the bias (or intercept) (do not confuse with the bias-variance tradeoff in the next lecture)
- $\mathbf{w}$ and b together are the parameters
- We hope that our prediction is close to the target: $y \approx t$.

Peter Munk Cardiac Centre

Linear Models -- Problem Setup



- If we have only 1 feature: $y = wx + b$ where $w, x, b \in \mathbb{R}$.
- y is linear in x.



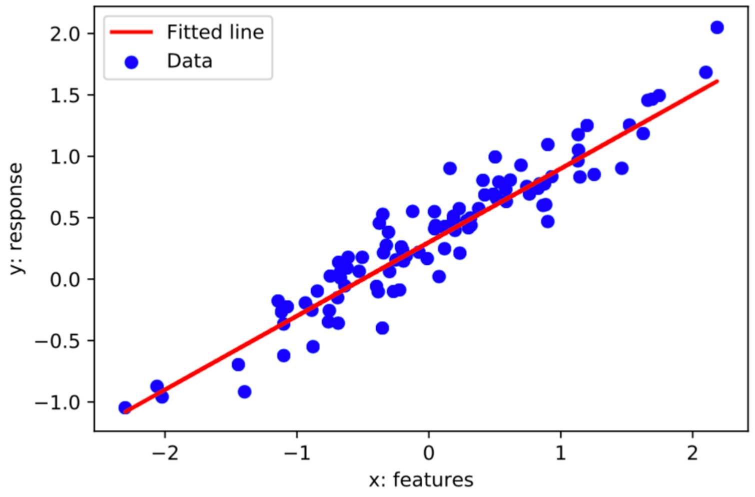
- If we have D features: $y = \mathbf{w}^\top \mathbf{x} + b$ where $\mathbf{w}, \mathbf{x} \in \mathbb{R}^D$, $b \in \mathbb{R}$
- y is linear in $\mathbf{x}$.

Relation between the prediction y and inputs $\mathbf{x}$ is linear in both cases.

Linear Models -- Problem Setup

We have a dataset $\mathcal{D} = \{(\mathbf{x}^{(i)}, t^{(i)}) \text{ for } i = 1, 2, ..., N\}$ where,

- $\mathbf{x}^{(i)} = (x_1^{(i)}, x_2^{(i)}, ..., x_D^{(i)})^\top \in \mathbb{R}^D$ are the inputs, e.g., age, height.
- $t^{(i)} \in \mathbb{R}$ is the target or response (e.g. income),
- predict $t^{(i)}$ with a linear function of $\mathbf{x}^{(i)}$:



- $t^{(i)} \approx y^{(i)} = \mathbf{w}^\top \mathbf{x}^{(i)} + b$
- Find the "best" line $(\mathbf{w}, b)$.
- $\underset{(\mathbf{w},b)}{\text{minimize}} \sum_{i=1}^{N} \mathcal{L}(y^{(i)}, t^{(i)})$

Linear Models -- Problem Setup

- How to quantify the quality of the fit to data?
- A loss function $\mathcal{L}(y, t)$ defines how bad it is if, for some example $\mathbf{x}$, the algorithm predicts y, but the target is actually t.
- Squared error loss function:

$$\mathcal{L}(y, t) = \tfrac{1}{2}(y - t)^2$$

- $y - t$ is the residual, and we want to make its magnitude small
- The $\frac{1}{2}$ factor is just to make the calculations convenient.
- Cost function: loss function averaged over all training examples

$$\begin{aligned}\mathcal{J}(\mathbf{w}, b) &= \frac{1}{2N}\sum_{i=1}^{N}\left(y^{(i)} - t^{(i)}\right)^2 \\ &= \frac{1}{2N}\sum_{i=1}^{N}\left(\mathbf{w}^\top \mathbf{x}^{(i)} + b - t^{(i)}\right)^2\end{aligned}$$

- The terminology is not universal. Some might call "loss" *pointwise loss* and the "cost function" the *empirical loss* or *average loss*.

Linear Models -- Vectorization

- We can organize all the training examples into a design matrix $\mathbf{X}$ with one row per training example, and all the targets into the target vector $\mathbf{t}$.

$$\mathbf{X} = \begin{pmatrix} \mathbf{x}^{(1)\top} \\ \mathbf{x}^{(2)\top} \\ \mathbf{x}^{(3)\top} \end{pmatrix} = \begin{array}{ccccc} & \text{one feature across all training examples} & & & \\ 8 & 0 & 3 & 0 & \\ 6 & -1 & 5 & 3 & \text{one training example (vector)} \\ 2 & 5 & -2 & 8 & \end{array}$$

- Computing the predictions for the whole dataset:

$$\mathbf{X}\mathbf{w} + b\mathbf{1} = \begin{pmatrix} \mathbf{w}^T\mathbf{x}^{(1)} + b \\ \vdots \\ \mathbf{w}^T\mathbf{x}^{(N)} + b \end{pmatrix} = \begin{pmatrix} y^{(1)} \\ \vdots \\ y^{(N)} \end{pmatrix} = \mathbf{y}$$

UHN Peter Munk Cardiac Centre

Linear Models -- Vectorization

- Computing the squared error cost across the whole dataset:

$$\mathbf{y} = \mathbf{X}\mathbf{w} + b\mathbf{1}$$
$$\mathcal{J} = \frac{1}{2N}\|\mathbf{y} - \mathbf{t}\|^2$$

- Note that sometimes we may use $\mathcal{J} = \frac{1}{2}\|\mathbf{y} - \mathbf{t}\|^2$, without normalizer. That would correspond to the sum of losses, and not the average loss. The minimizer does not depend on N.
- We can also add a column of 1s to the design matrix, combine the bias and the weights, and conveniently write

$$\mathbf{X} = \begin{bmatrix} 1 & [\mathbf{x}^{(1)}]^\top \\ 1 & [\mathbf{x}^{(2)}]^\top \\ 1 & \vdots \end{bmatrix} \in \mathbb{R}^{N \times D+1} \quad \text{and} \quad \mathbf{w} = \begin{bmatrix} b \\ w_1 \\ w_2 \\ \vdots \end{bmatrix} \in \mathbb{R}^{D+1}$$

Then, our predictions reduce to $\mathbf{y} = \mathbf{X}\mathbf{w}$.

UHN Peter Munk Cardiac Centre

Linear Models -- Vectorization

- Computing the prediction using a for loop:

```
y = b
for j in range(M):
    y += w[j] * x[j]
```

- For-loops in Python are slow, so we vectorize algorithms by expressing them in terms of vectors and matrices.

$$\mathbf{w} = (w_1, \ldots, w_D)^T \qquad \mathbf{x} = (x_1, \ldots, x_D)$$

$$y = \mathbf{w}^T \mathbf{x} + b$$

- This is simpler and much faster:

```
y = np.dot(w, x) + b
```

Linear Models -- Optimization

- We defined a cost function. This is what we would like to minimize.
- Recall from your calculus class: minimum of a smooth function (if it exists) occurs at a critical point, i.e., point where the derivative is zero.
- Multivariate generalization: set the partial derivatives to zero (or equivalently the gradient).
- We would like to find a point where the gradient is (close to) zero. How can we do it?
    - Sometimes it is possible to directly find the parameters that make the gradient zero in a closed-form. We call this the direct solution.
    - We may also use optimization techniques that iteratively get us closer to the solution. We will get back to this soon.

Optimization : What to do if we are first-year undergrad

- Partial derivatives: derivatives of a multivariate function with respect to one of its arguments.

$$\frac{\partial}{\partial x_1} f(x_1, x_2) = \lim_{h \to 0} \frac{f(x_1 + h, x_2) - f(x_1, x_2)}{h}$$

- To compute, take the single variable derivatives, pretending the other arguments are constant.
- Example: partial derivatives of the prediction y

$$\begin{aligned}\frac{\partial y}{\partial w_j} &= \frac{\partial}{\partial w_j}\left[\sum_{j'} w_{j'} x_{j'} + b\right] \\ &= x_j \\ \frac{\partial y}{\partial b} &= \frac{\partial}{\partial b}\left[\sum_{j'} w_{j'} x_{j'} + b\right] \\ &= 1\end{aligned}$$

UHN Peter Munk Cardiac Centre

Optimization : What to do if we are first-year undergrad

- Chain rule for derivatives:

$$
\begin{aligned}
\frac{\partial \mathcal{L}}{\partial w_j} &= \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}y}\frac{\partial y}{\partial w_j} \\
&= \frac{\mathrm{d}}{\mathrm{d}y}\left[\frac{1}{2}(y-t)^2\right]\cdot x_j \\
&= (y-t)x_j \\
\frac{\partial \mathcal{L}}{\partial b} &= y-t
\end{aligned}
$$

- Cost derivatives (average over data points):

$$
\begin{aligned}
\frac{\partial \mathcal{J}}{\partial w_j} &= \frac{1}{N}\sum_{i=1}^{N}(y^{(i)}-t^{(i)})\,x_j^{(i)} \\
\frac{\partial \mathcal{J}}{\partial b} &= \frac{1}{N}\sum_{i=1}^{N}y^{(i)}-t^{(i)}
\end{aligned}
$$

Optimization : What to do if we are first-year undergrad

- The minimum must occur at a point where the partial derivatives are zero, i.e.,

$$\frac{\partial \mathcal{J}}{\partial w_j} = 0 \quad (\forall j), \qquad \frac{\partial \mathcal{J}}{\partial b} = 0.$$

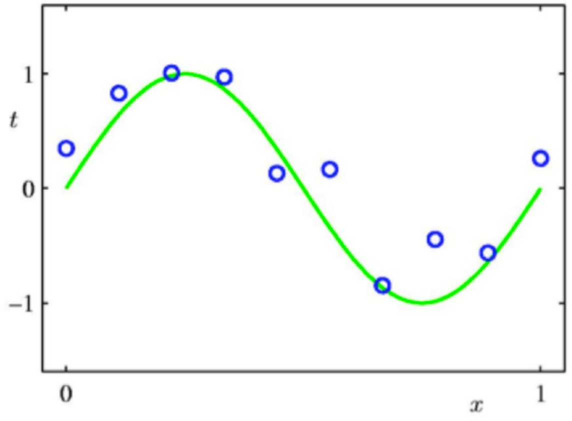
- If $\partial \mathcal{J} / \partial w_j \neq 0$, you could reduce the cost by changing w_j.
- This turns out to give a system of linear equations, which we can solve efficiently. **Full derivation in the preliminaries.pdf.**
- Optimal weights:

$$\mathbf{w}^{\mathrm{LS}} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{t}$$

- Linear regression is one of only a handful of models in this course that permit direct solution.

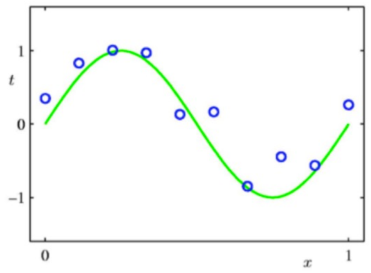
Linear Models – Feature Mapping

- The relation between the input and output may not be linear.



- We can still use linear regression by mapping the input feature to another space using feature mapping (or basis expansion) $\boldsymbol{\psi}(\mathbf{x}) : \mathbb{R}^D \to \mathbb{R}^d$ and treat the mapped feature (in $\mathbb{R}^d$) as the input of a linear regression procedure.
- Let us see how it works when $\mathbf{x} \in \mathbb{R}$ and we use polynomial feature mapping.

Linear Models – Feature Mapping



Fit the data using a degree-M polynomial function of the form:

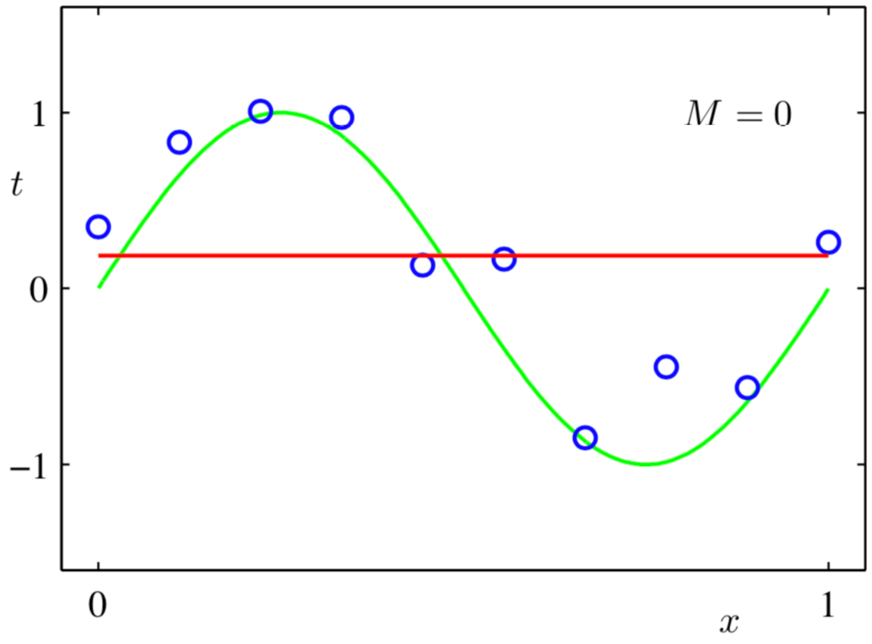
$$y = w_0 + w_1 x + w_2 x^2 + \ldots + w_M x^M = \sum_{i=0}^{M} w_i x^i$$

- Here the feature mapping is $\boldsymbol{\psi}(x) = [1, x, x^2, \ldots]^\top$.
- We can still use least squares to find $\mathbf{w}$ since $y = \boldsymbol{\psi}(x)^\top \mathbf{w}$ is linear in $w_0, w_1, \ldots$.
- In general, $\boldsymbol{\psi}$ can be any function. Another example: $\boldsymbol{\psi} = [1, \sin(2\pi x), \cos(2\pi x), \sin(4\pi x), \cos(4\pi x), \sin(6\pi x), \cos(6\pi x), \cdots]^\top$.

UHN Peter Munk Cardiac Centre

Linear Models – Feature Mapping

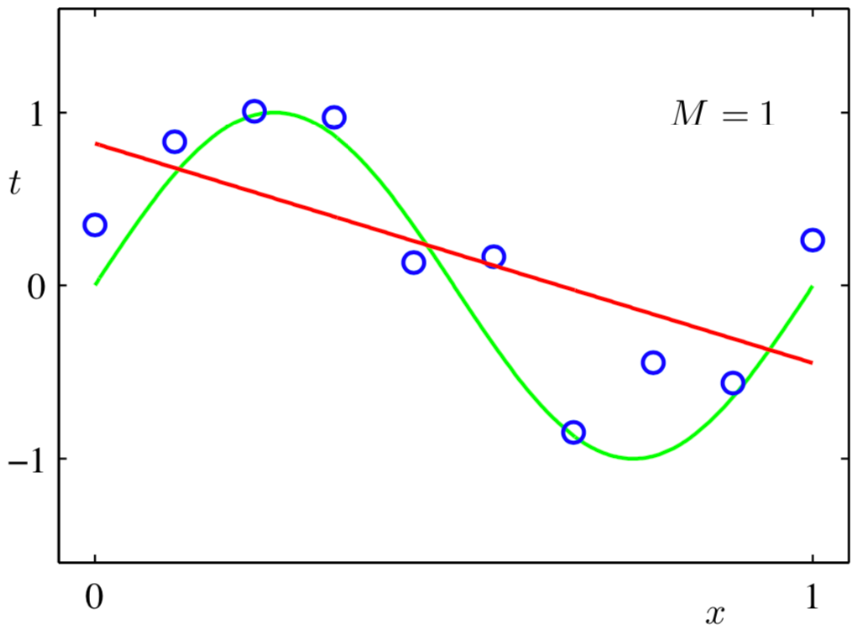
$$y = w_0$$



-Pattern Recognition and Machine Learning, Christopher Bishop.

Linear Models – Feature Mapping

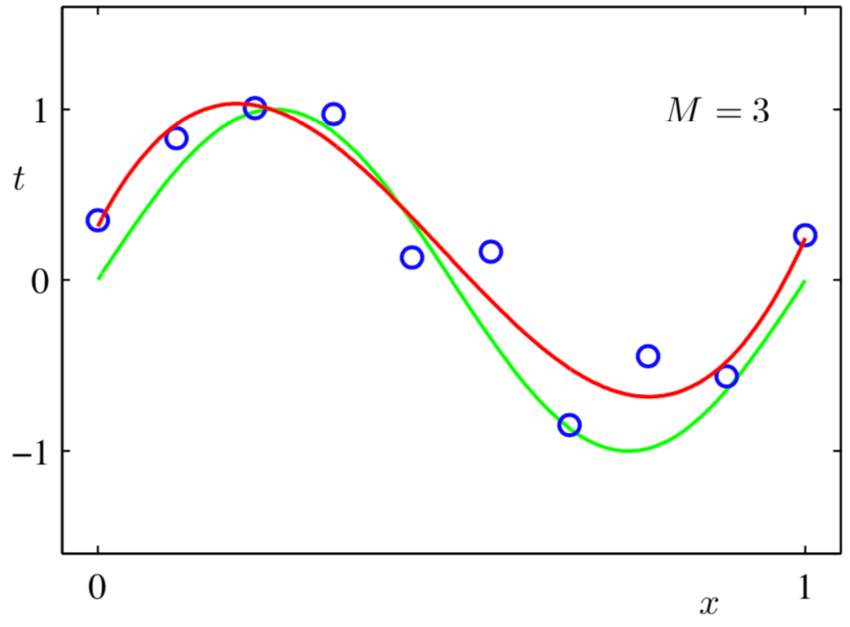
$$y = w_0 + w_1 x$$



-Pattern Recognition and Machine Learning, Christopher Bishop.

Linear Models – Feature Mapping

$$y = w_0 + w_1 x + w_2 x^2 + w_3 x^3$$

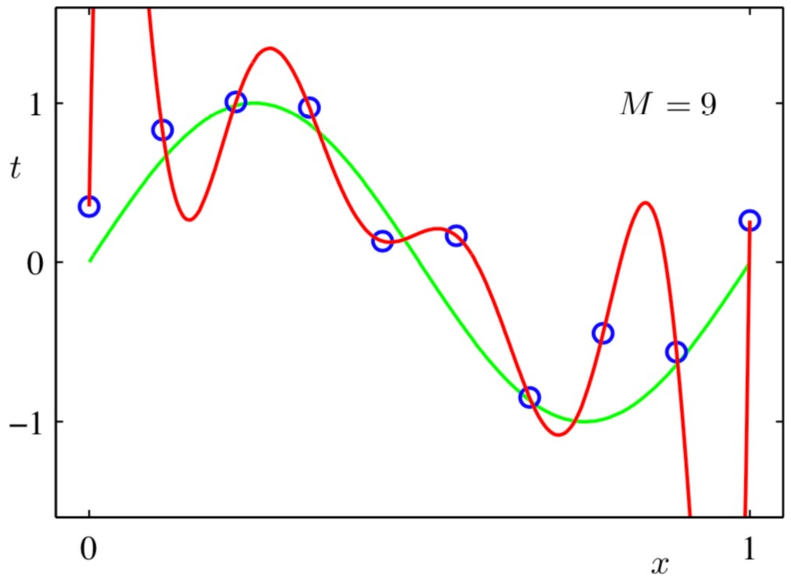


-Pattern Recognition and Machine Learning, Christopher Bishop.

UHN Peter Munk Cardiac Centre

Linear Models – Feature Mapping

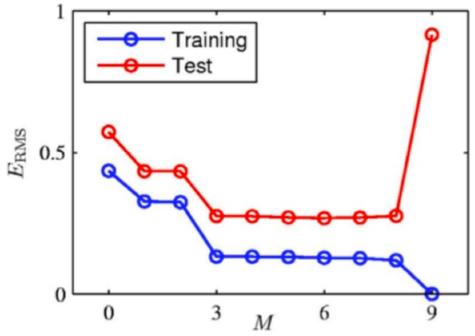
$$y = w_0 + w_1 x + w_2 x^2 + w_3 x^3 + \ldots + w_9 x^9$$

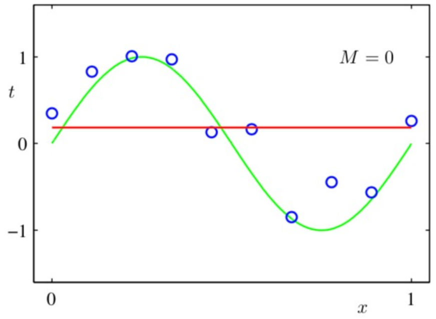


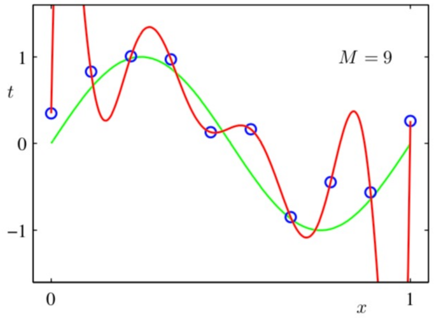
-Pattern Recognition and Machine Learning, Christopher Bishop.

Linear Models – Model Selection

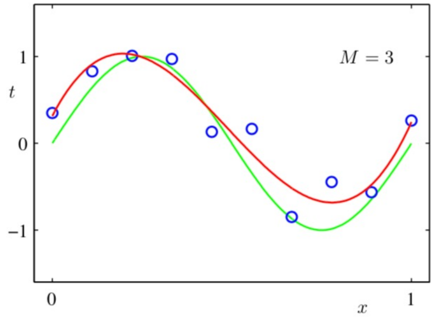
Underfitting (M=0): model is too simple — does not fit the data.
Overfitting (M=9): model is too complex — fits perfectly.





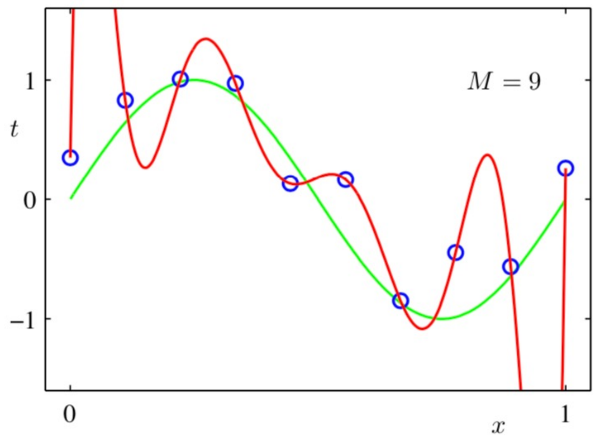


Good model (M=3): Achieves small test error (generalizes well).



Linear Models – Model Selection

	$M=0$	$M=1$	$M=3$	$M=9$
$w_0^\star$	0.19	0.82	0.31	0.35
$w_1^\star$		-1.27	7.99	232.37
$w_2^\star$			-25.43	-5321.83
$w_3^\star$			17.37	48568.31
$w_4^\star$				-231639.30
$w_5^\star$				640042.26
$w_6^\star$				-1061800.52
$w_7^\star$				1042400.18
$w_8^\star$				-557682.99
$w_9^\star$				125201.43



- As M increases, the magnitude of coefficients gets larger.
- For $M=9$, the coefficients have become finely tuned to the data.
- Between data points, the function exhibits large oscillations.

Linear Models – Model Selection

- The degree of the polynomial M controls the complexity of the model.
- The value of M is a hyperparameter for polynomial expansion, just like k in KNN. We can tune it using a validation set.
- Restricting the number of parameters of a model (M here) is a crude approach to control the complexity of the model.
- A better solution: keep the number of parameters of the model large, but enforce "simpler" solutions within the same space of parameters.
- This is done through regularization or penalization.
    - Regularizer (or penalty): a function that quantifies how much we prefer one hypothesis vs. another
- Q: How?!

Linear Models -- Regularization

- We can encourage the weights to be small by choosing as our regularizer the ℓ_2 (or L^2) penalty.

$$\mathcal{R}(\mathbf{w}) = \tfrac{1}{2}\|\mathbf{w}\|_2^2 = \frac{1}{2}\sum_j w_j^2.$$

  - Note: To be precise, we are regularizing the *squared* ℓ_2 norm.

- The regularized cost function makes a tradeoff between fit to the data and the norm of the weights:

$$\mathcal{J}_{\text{reg}}(\mathbf{w}) = \mathcal{J}(\mathbf{w}) + \lambda\mathcal{R}(\mathbf{w}) = \mathcal{J}(\mathbf{w}) + \frac{\lambda}{2}\sum_j w_j^2.$$

- The basic idea is that "simpler" functions have smaller ℓ_2-norm of their weights $\mathbf{w}$, and we prefer them to functions with larger ℓ_2-norms.
- If you fit training data poorly, $\mathcal{J}$ is large. If your optimal weights have high values, $\mathcal{R}$ is large.
- Large λ penalizes weight values more.
- Here, λ is a hyperparameter that we can tune with a validation set.

Peter Munk Cardiac Centre

Linear Models -- Regularization

For the least squares problem, we have $\mathcal{J}(\mathbf{w}) = \frac{1}{2N}\|\mathbf{X}\mathbf{w} - \mathbf{t}\|^2$.

- When $\lambda > 0$ (with regularization), regularized cost gives

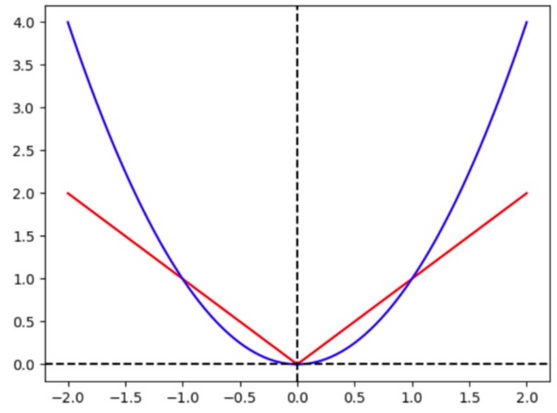
$$\begin{aligned} \mathbf{w}_{\lambda}^{\text{Ridge}} = \underset{\mathbf{w}}{\operatorname{argmin}}\, \mathcal{J}_{\text{reg}}(\mathbf{w}) &= \underset{\mathbf{w}}{\operatorname{argmin}}\, \frac{1}{2N}\|\mathbf{X}\mathbf{w} - \mathbf{t}\|_2^2 + \frac{\lambda}{2}\|\mathbf{w}\|_2^2 \\ &= (\mathbf{X}^T\mathbf{X} + \lambda N\mathbf{I})^{-1}\mathbf{X}^T\mathbf{t} \end{aligned}$$

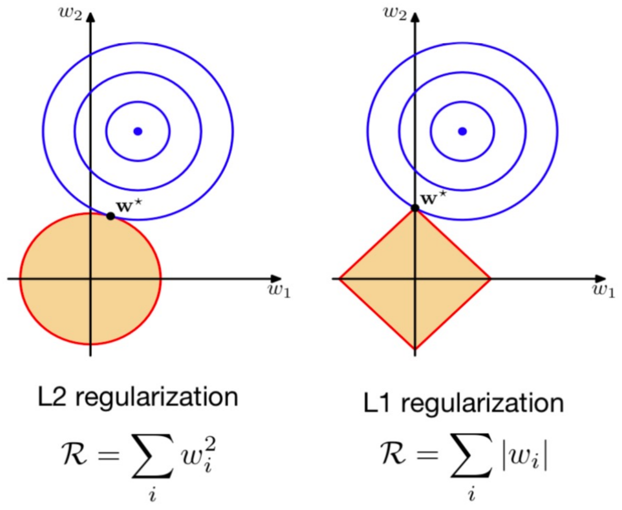
- The case $\lambda = 0$ (no regularization) reduces to least squares solution!
- Q: What happens when $\lambda \to \infty$?
- Note that it is also common to formulate this problem as $\operatorname{argmin}_{\mathbf{w}} \|\mathbf{X}\mathbf{w} - \mathbf{t}\|_2^2 + \frac{\lambda}{2}\|\mathbf{w}\|_2^2$ in which case the solution is $\mathbf{w}_{\lambda}^{\text{Ridge}} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{t}$.

UHN Peter Munk Cardiac Centre

Linear Models -- Regularization

- The ℓ_1 norm, or sum of absolute values, is another regularizer that encourages weights to be exactly zero. (How can you tell?)
- We can design regularizers based on whatever property we'd like to encourage.





— Bishop, *Pattern Recognition and Machine Learning*

Optimization :
A more generalized approach

- Now let's see a second way to minimize the cost function which is more broadly applicable: gradient descent.
- Gradient descent is an iterative algorithm, which means we apply an update repeatedly until some criterion is met.
- We initialize the weights to something reasonable (e.g. all zeros) and repeatedly adjust them in the direction of steepest descent.

Gradient Descent

- Observe:
  - if $\partial \mathcal{J}/\partial w_j > 0$, then increasing w_j increases $\mathcal{J}$.
  - if $\partial \mathcal{J}/\partial w_j < 0$, then increasing w_j decreases $\mathcal{J}$.
- The following update decreases the cost function:

$$\begin{aligned} w_j &\leftarrow w_j - \alpha \frac{\partial \mathcal{J}}{\partial w_j} \\ &= w_j - \frac{\alpha}{N} \sum_{i=1}^{N} (y^{(i)} - t^{(i)})\, x_j^{(i)} \end{aligned}$$

- α is a learning rate. The larger it is, the faster $\mathbf{w}$ changes.
  - We'll see later how to tune the learning rate, but values are typically small, e.g. 0.01 or 0.0001

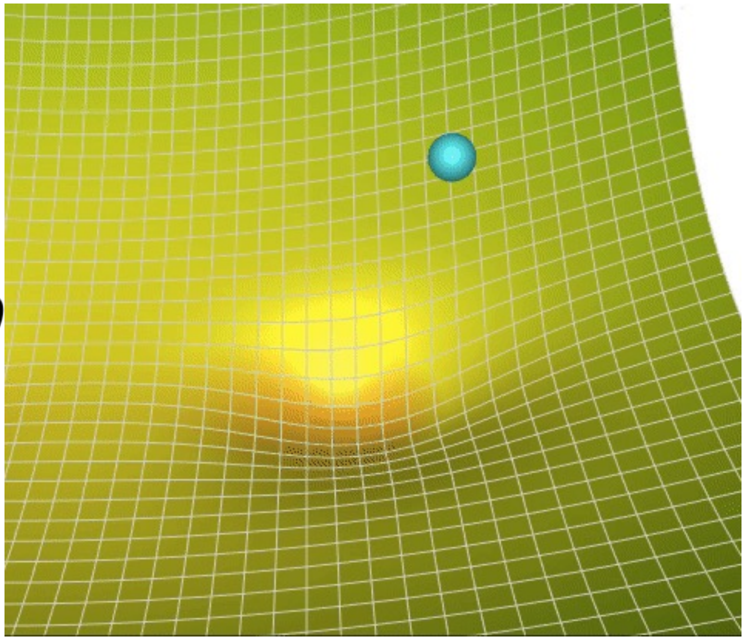
Gradient Descent

- Goal: we want to minimize a specific objective function (cost or loss):

$$\mathcal{J}(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N} \mathcal{J}^{(i)}(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N} \mathcal{L}(y(\mathbf{x}^{(i)}, \boldsymbol{\theta}), t^{(i)}).$$

- Step 1: Calculate Gradient (by linearity)

$$\nabla \mathcal{J}(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N} \nabla \mathcal{J}^{(i)}(\boldsymbol{\theta}$$



- Step 2: Update the parameters:

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \alpha \nabla \mathcal{J}(\boldsymbol{\theta})$$

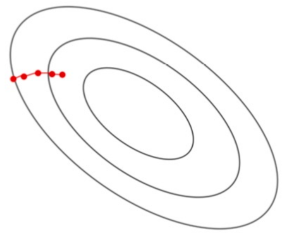
Source: https://towardsdatascience.com/a-visual-explanation-of-gradient-descent-methods-momentum-adagrad-rmsprop-adam-f898b102325c

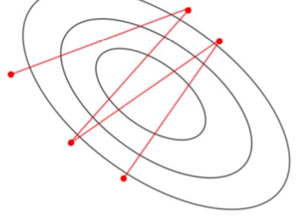
Gradient Descent

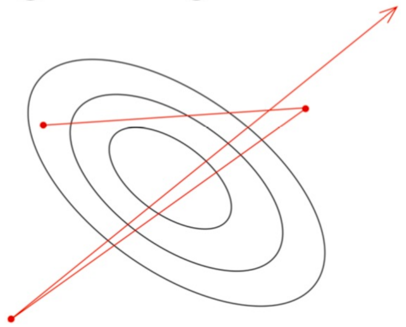
- Even for linear regression, where there is a direct solution, we sometimes need to use GD.
- Why gradient descent, if we can find the optimum directly?
  - GD can be applied to a much broader set of models
  - GD can be easier to implement than direct solutions
  - For regression in high-dimensional spaces, GD is more efficient than direct solution
    - Linear regression solution: $(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{t}$
    - matrix inversion is an $\mathcal{O}(D^3)$ algorithm
    - each GD update costs $O(ND)$
    - Huge difference if $D \gg 1$

Gradient Descent

- In gradient descent, the learning rate α is a hyperparameter we need to tune. Here are some things that can go wrong:


α too small: slow progress

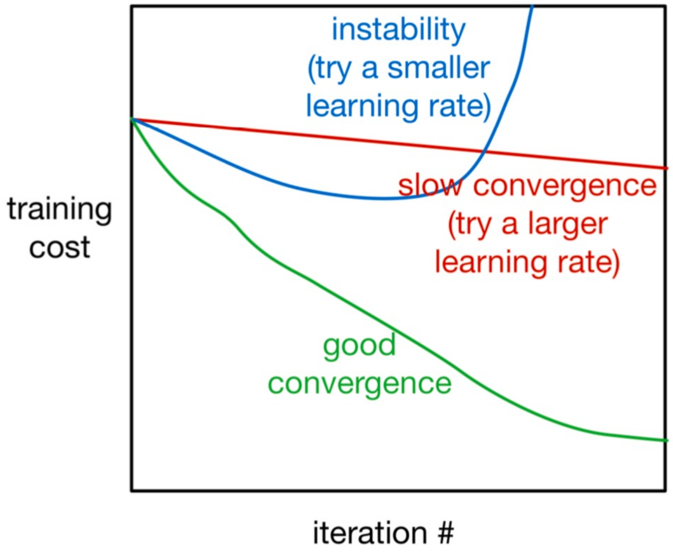

α too large: oscillations


α much too large: instability

- Good values are typically between 0.001 and 0.1. You should do a grid search if you want good performance (i.e. try $0.1, 0.03, 0.01, \ldots$).

Gradient Descent

- To diagnose optimization problems, it's useful to look at training curves: plot the training cost as a function of iteration.



- Warning: it's very hard to tell from the training curves whether an optimizer has converged. They can reveal major problems, but they can't guarantee convergence.

Linear Models

Why vectorize?

- The equations, and the code, will be simpler and more readable. Gets rid of dummy variables/indices!
- Vectorized code is much faster
  - Cut down on Python interpreter overhead
  - Use highly optimized linear algebra libraries
  - Matrix multiplication is very fast on a Graphics Processing Unit (GPU)

A general theme

Linear regression exemplifies recurring themes of this course:

- choose a model and a loss function
- formulate an optimization problem
- solve the minimization problem using one of two strategies
  - direct solution (set derivatives to zero)
  - gradient descent (see appendix)
- vectorize the algorithm, i.e. represent in terms of linear algebra
- make a linear model more powerful using features
- improve the generalization by adding a regularizer

Linear Model: Take-Home Messages

1. Linear Model is simple and interpretable.

2. Regularization on weights can help with overfitting.

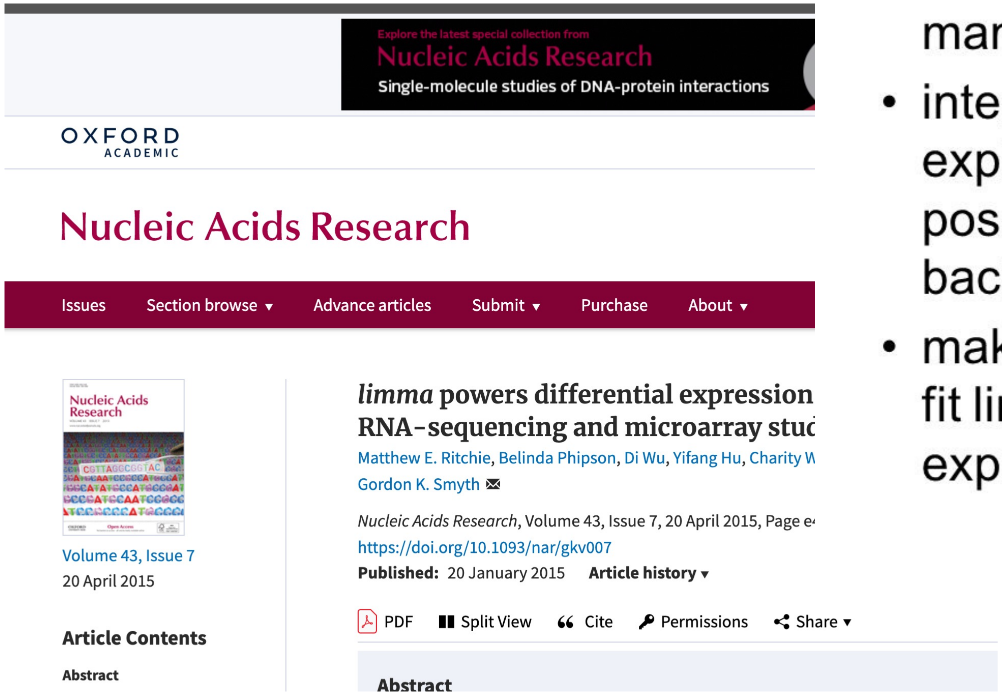
3. Gradient descent is a general way to solve optimization problems in machine learning.

4. Linear model may fail when linearity assumption does not hold.

Linear Models in Medicine

LIMMA: Linear models for microarray data

Nearly 20K citations!



Explore the latest special collection from

Nucleic Acids Research

Single-molecule studies of DNA-protein interactions

OXFORD ACADEMIC

Nucleic Acids Research

limma powers differential expression analyses for RNA-sequencing and microarray stud

Matthew E. Ritchie, Belinda Phipson, Di Wu, Yifang Hu, Charity W

Gordon K. Smyth

Nucleic Acids Research, Volume 43, Issue 7, 20 April 2015, Page e

https://doi.org/10.1093/nar/gkv007

Published: 20 January 2015 **Article history** ▾

Volume 43, Issue 7

20 April 2015

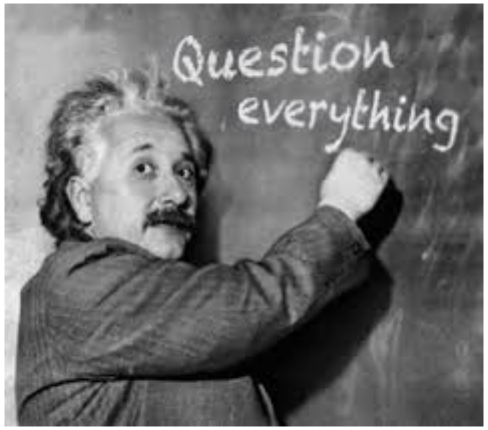
Article Contents

Abstract

Abstract

My experience

- limma has excellent documentation and many examples
- integration with preprocessing and exploratory data analysis makes it possible to test different options for background subtraction and normalization
- makes it possible for a non-statistician to fit linear models and find differentially expressed genes



UHN Peter Munk Cardiac Centre