Massively Parallel Algorithms
Classification & Prediction
Using Random Forests

G. Zachmann

University of Bremen, Germany

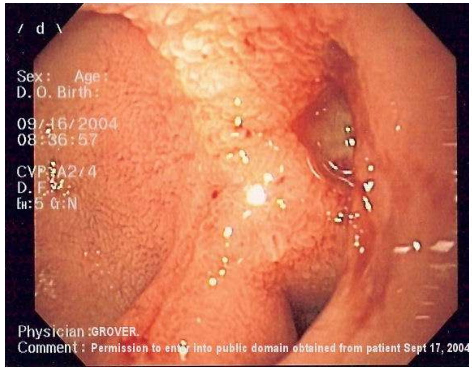
cgvr.cs.uni-bremen.de

Classification Problem Statement

- Given a set of points $\mathcal{L} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \in \mathbb{R}^d$ and for each such point a label $y_i \in \{l_1, l_2, \ldots, l_n\}$
  - Each label represents a class, all points with the same label are in the same class
- Wanted: a method to decide for a *not-yet-seen* point **x** which label it most probably has, i.e., a method to *predict class labels*
  - We say that we learn a classifier C from the training set $\mathcal{L}$:

$$C : \mathbb{R}^d \rightarrow \{l_1, l_2, \ldots, l_n\}$$

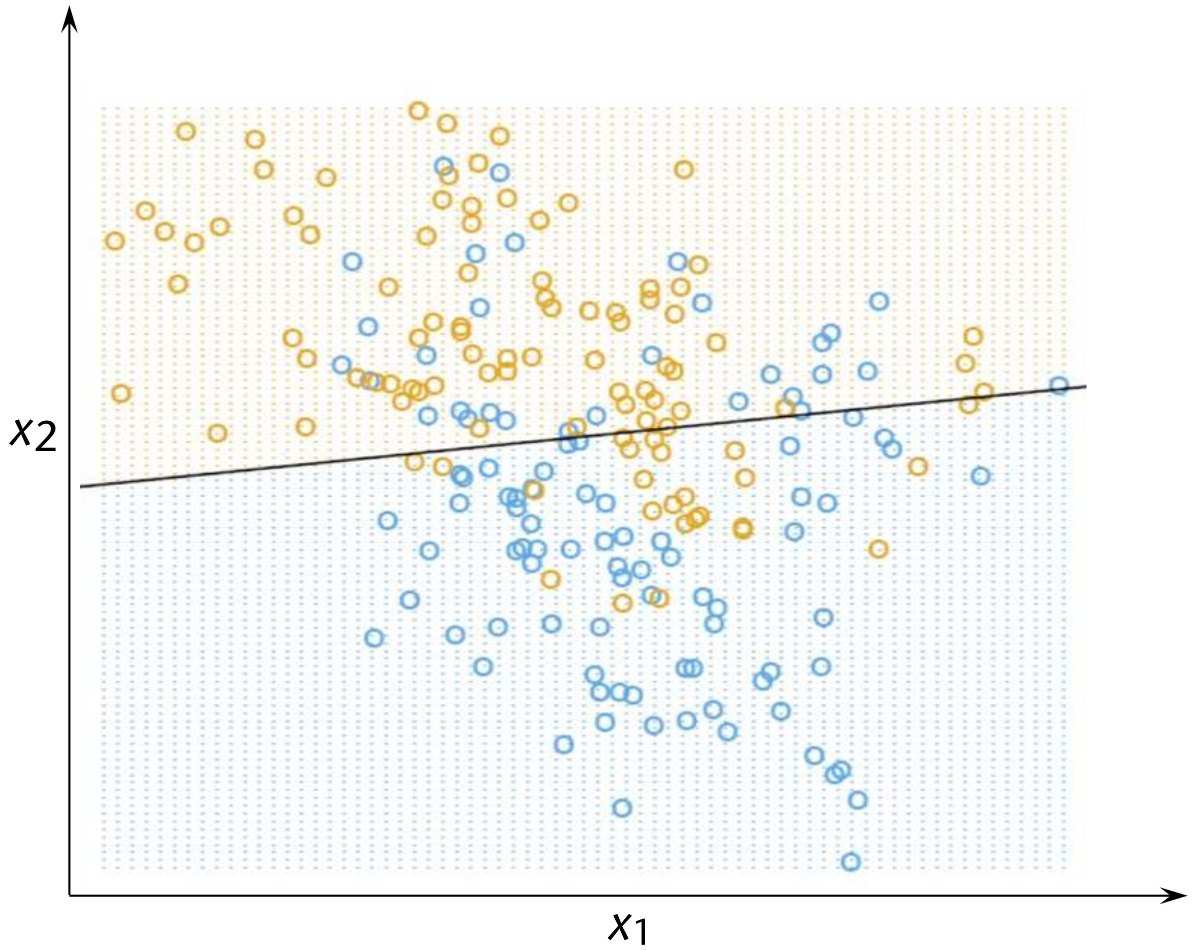
- Typical applications:
  - Computer vision (object recognition, …)
  - Medical diagnosis
  - Credit approval (?)
  - Jurisdiction ?



Ulcer/tumor or not?

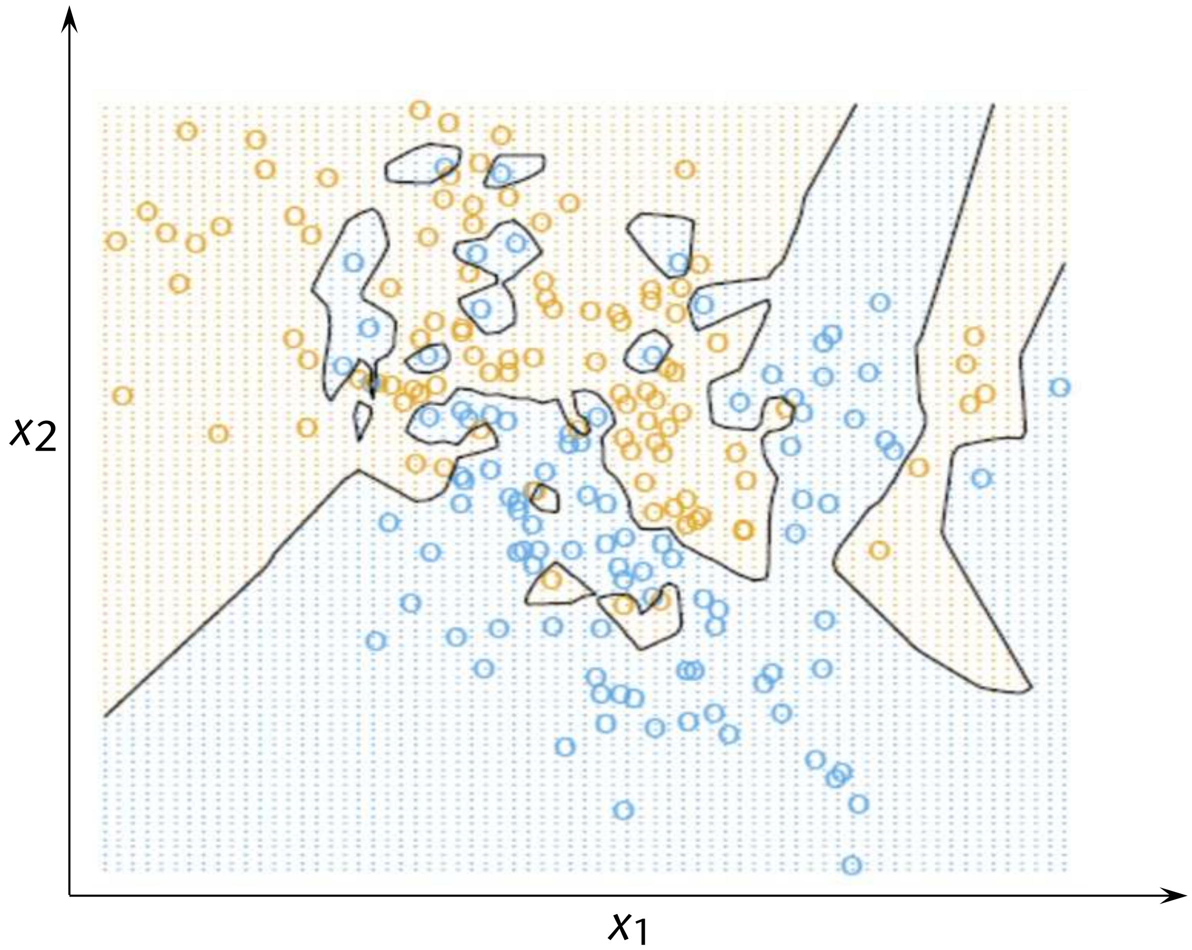
One Possible Solution: Linear Regression

- Assume we have only two classes (e.g., "blue" and "yellow")
- Fit a plane through the data



Another Solution: Nearest Neighbor (NN) Classification

- For the query point **x**, find the nearest neighbor $\mathbf{x}^* \in \{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \in \mathbb{R}^d$
- Assign the class l^* to **x**

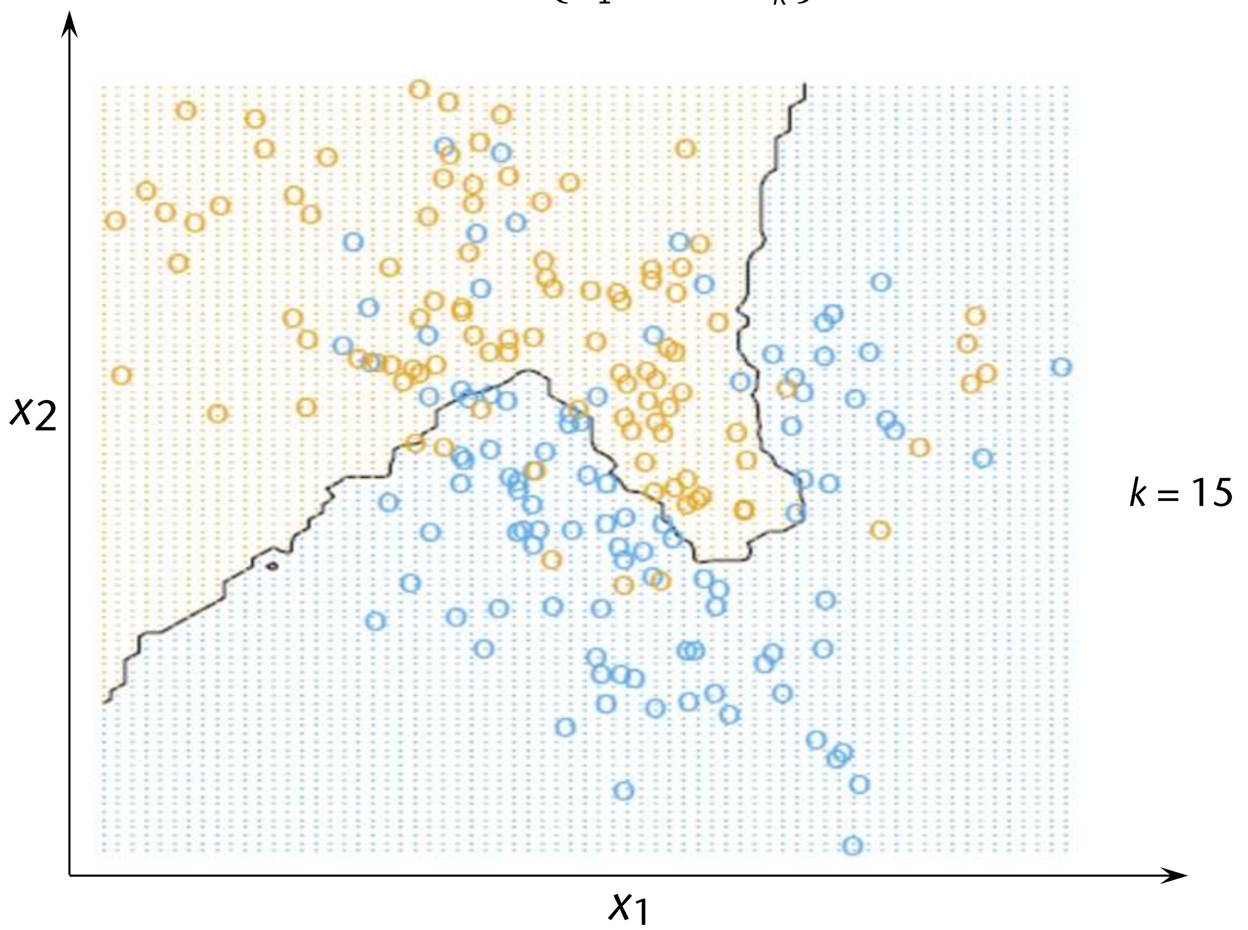


Parallelization

- Trivially:
  - Each thread computes distance $\| \mathbf{x}_i - \mathbf{x} \|$ and stores it in an array
  - All threads perform min reduction
- Can you think of a more clever way?
- What if we have a million queries?

Improvement: *k*-NN Classification

- Instead of the 1 nearest neighbor, find the k nearest neighbors of $\mathbf{x}$, $\{\mathbf{x}_{i_1}, \ldots, \mathbf{x}_{i_k}\} \subset \mathcal{L}$
- Assign the majority of the labels $\{l_{i_1}, \ldots, l_{i_k}\}$ to $\mathbf{x}$



More Terminology

- The coordinates/components $x_{i,j}$ of the points $\mathbf{x}_i$ have special names: independent variables, predictor variables, features, attributes, …
  - Specific name of the $x_{i,j}$ depends on the domain / community
- The space where the $\mathbf{x}_i$ live (i.e., $\mathbb{R}^d$) is called feature space
- The labels y_i are also called target, dependent variable, response variable, …
- The set $\mathcal{L}$ is called the training set / learning set (will become clear later)

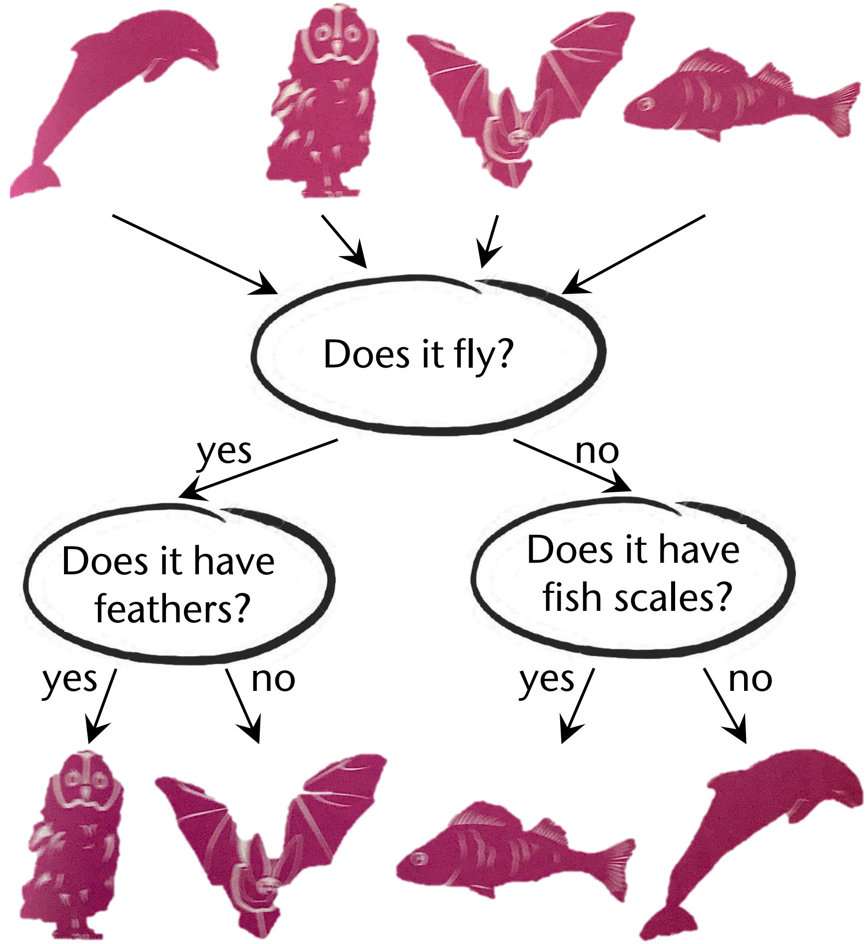
Decision Trees

- Aristotle first described the concept systematically, in order to classify all living things

Branches represent different values of a feature

Each *node* tests one or more feature(s)
This is sometimes called a weak classifier

Leaves represent the classes (decisions)



Another Example

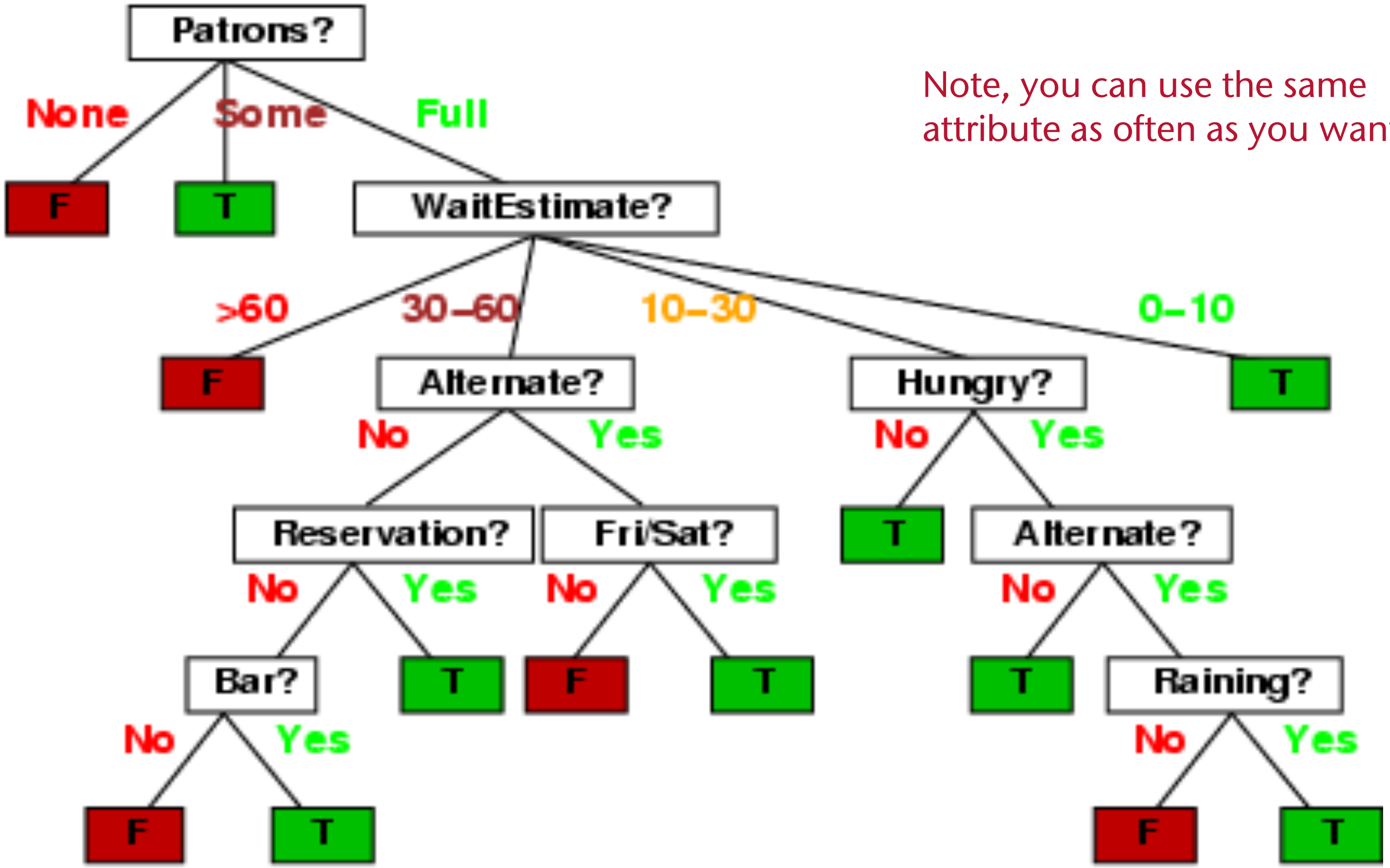
- "Please wait to be seated" ...
- Decide: *wait* or *go* some place else?
- Variables that *could* influence your decision:
  - Alternate: is there an alternative restaurant nearby?
  - Bar: is there a comfortable bar area to wait in?
  - Fri/Sat: is today Friday or Saturday?
  - Hungry: are we hungry?
  - Patrons: number of people in the restaurant (None, Some, Full)
  - Price: price range ($, $$, $$$)
  - Raining: is it raining outside?
  - Reservation: have we made a reservation?
  - Type: kind of restaurant (French, Italian, Thai, Burger)
  - EstimatedWait: estimated waiting time (0-10, 10-30, 30-60, >60)

- You collect data to base your decisions on:

Example	Attributes										Decision
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*Wait*
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

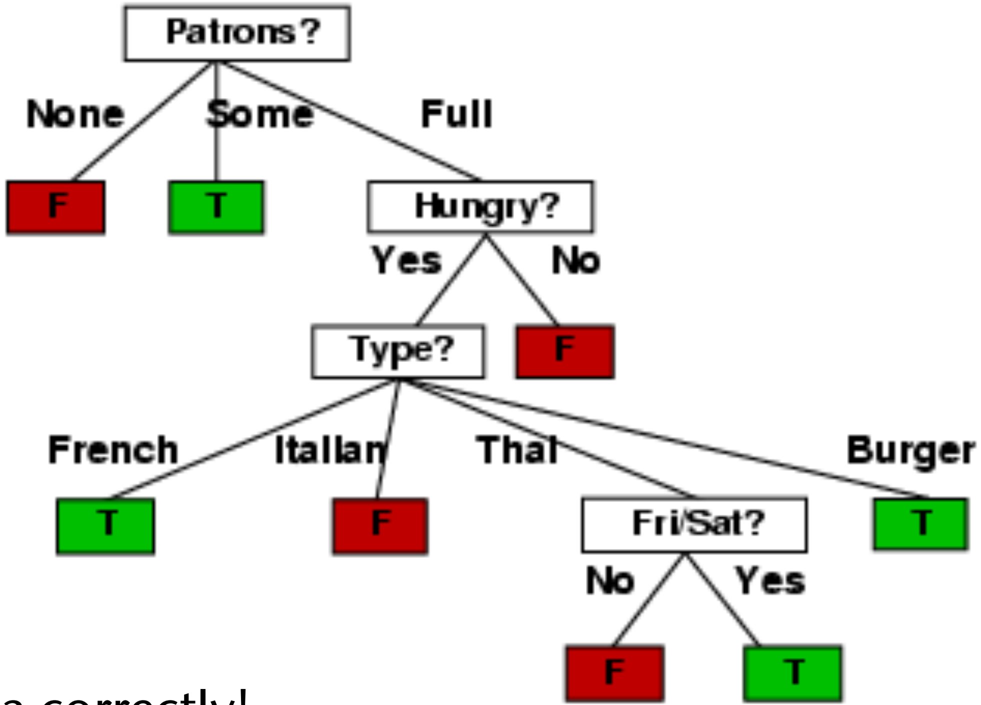
- Feature space: space of all possible feature vectors with all possible combinations of features
  - Here: 10-dimensional, 6 Boolean attributes, 3 discrete attributes, one continuous attribute

- A decision tree that classifies all "training data" correctly:



Note, you can use the same attribute as often as you want

- A better decision tree:



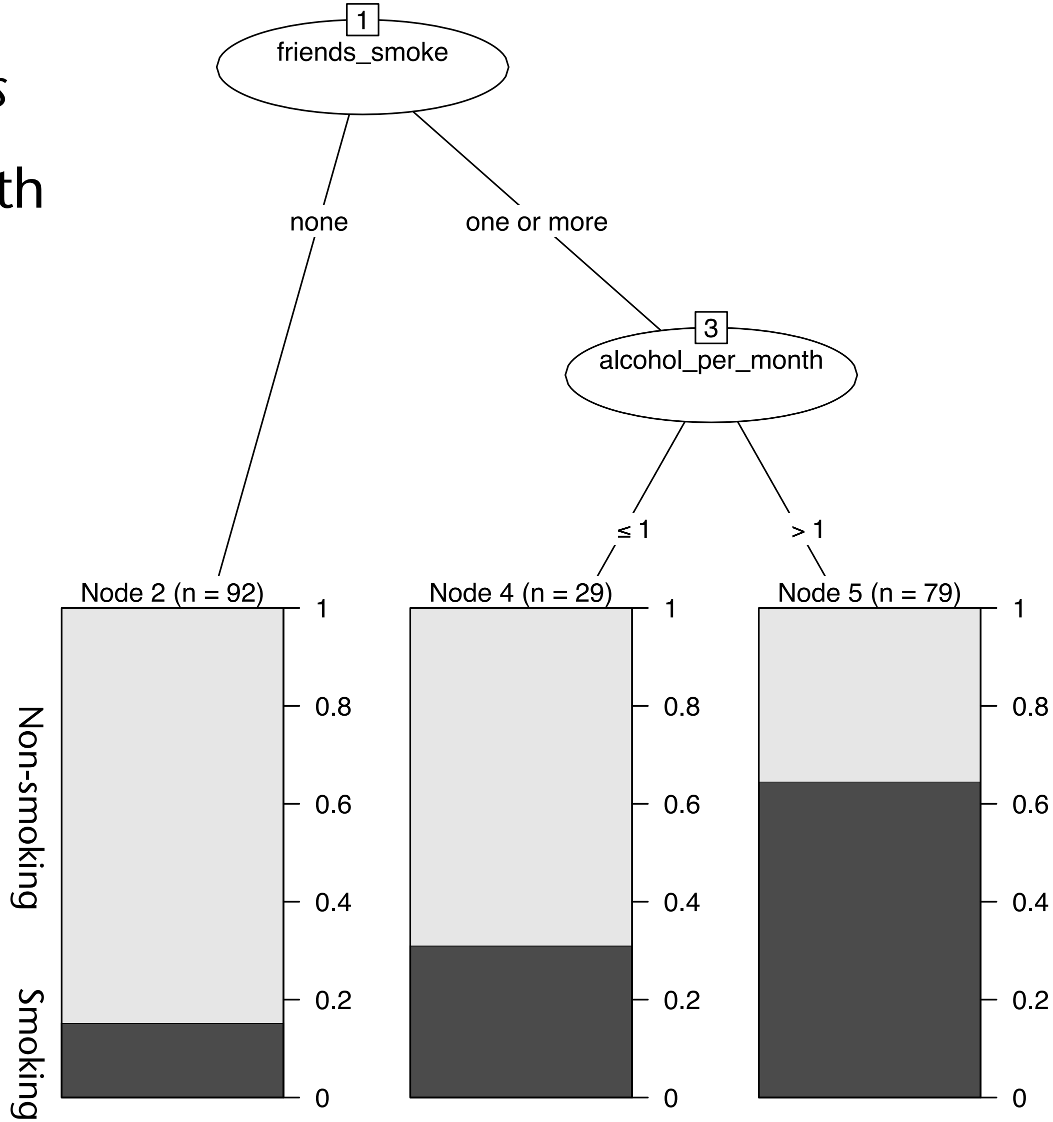
  - Also classifies all training data correctly!
  - Decisions can be made faster
- Questions:
  - How to construct (optimal) decision trees methodically?
  - How well does it generalize/predict? (what is its generalization error?)

Construction (= Learning) of Decision Trees

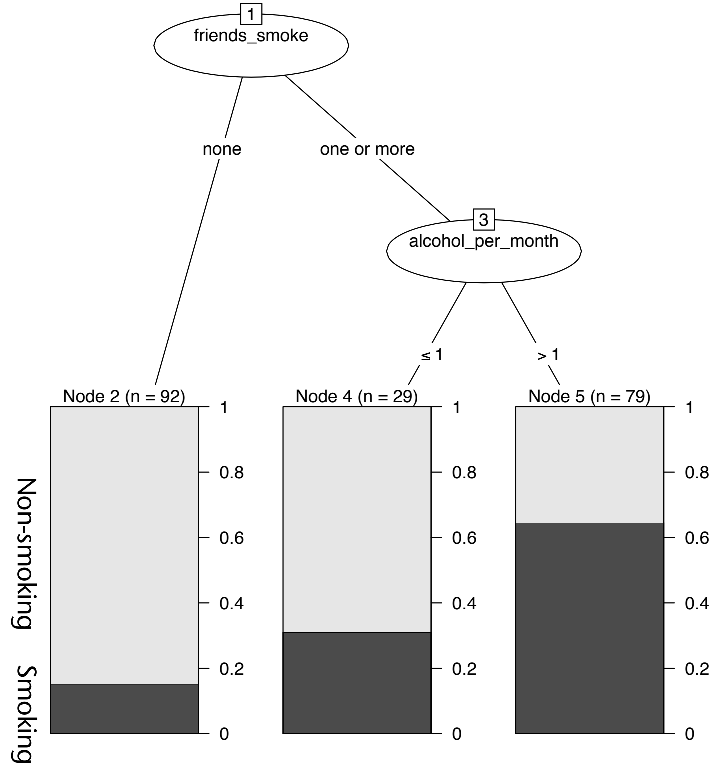
- By way of the following example
- Goal: predict adolescents' intention to smoke within next year
  - Binary response variable *IntentionToSmoke*
- Four predictor variables (= attributes):
  - *LiedToParents* (bool) = subject has ever lied to parents about doing something they would not approve of
  - *FriendsSmoke* (bool) = one or more of the 4 best friends smoke
  - *Age* (int) = subject's current age
  - *AlcoholPerMonth* (int) = # times subject drank alcohol during past month
- Training data:
  - Kitsantas et al.: *Using classification trees to profile adolescent smoking behaviors*. 2007
  - 200 adolescents surveyed

A decision tree

- Root node splits all data points into *two subsets*
- Node 2 = all data points with *FriendsSmoke* = false
- Node 2 contains 92 points, 18% have label "yes", 82% have label "no"
- Ditto for the other nodes

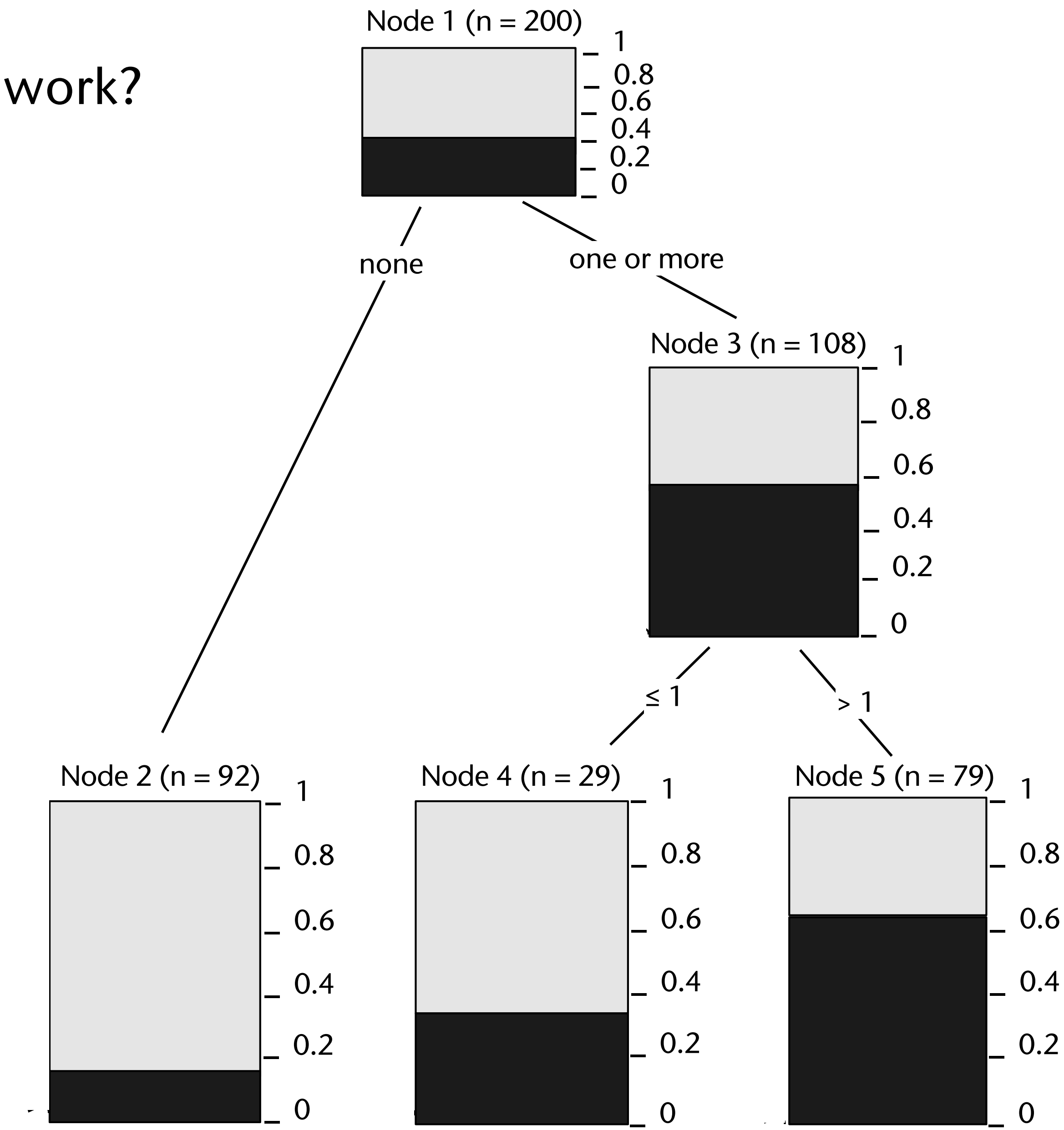


- Observation: a decision tree partitions feature space into rectangular regions (like kd-tree):

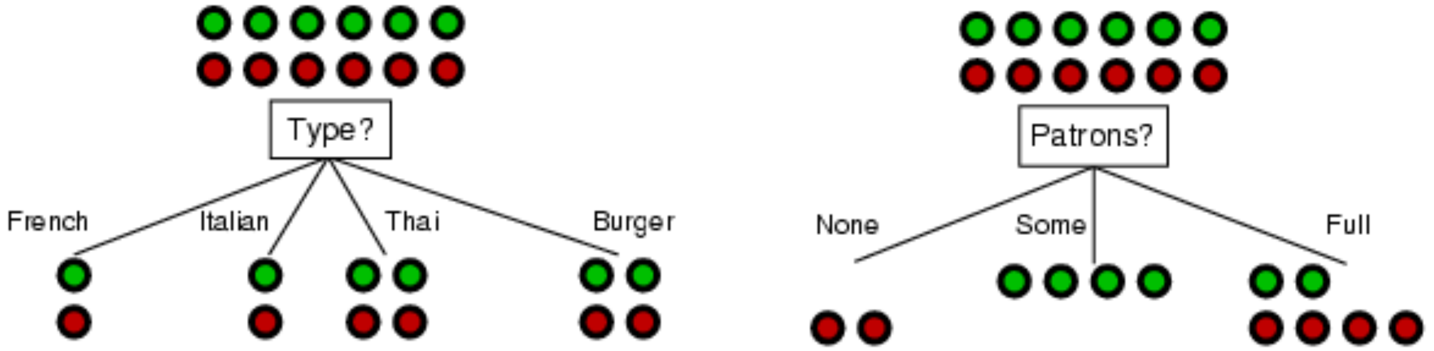


Selection of Splitting Variable and Cutpoint

- Why does our example work?
  - In the root node, *IntentionToSmoke*=yes is 40%
  - In node 2, *IntentionToSmoke*=yes is 18%, while in node 3 *IntentionToSmoke*=yes is 60%
  - So, after first split we can make better predictions

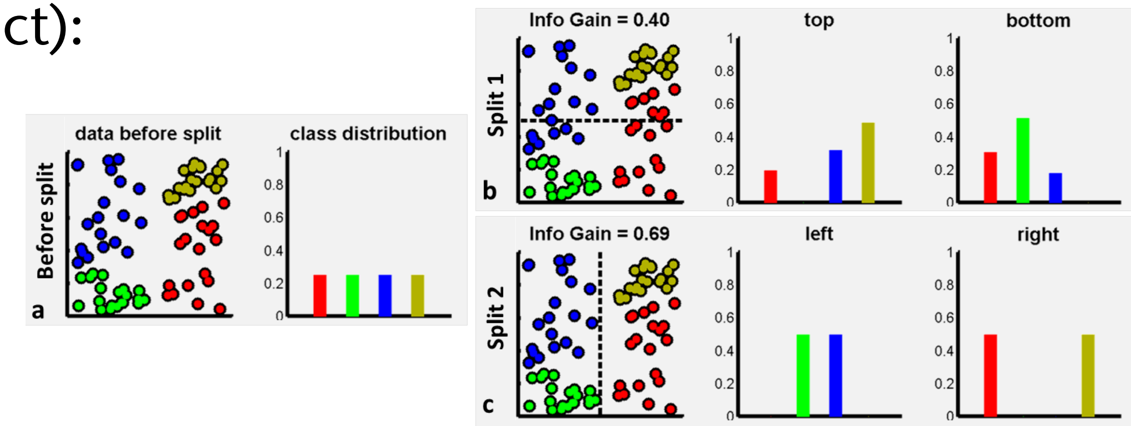


- Ideally, a good attribute (and cutpoint) splits the samples into subsets that are "all positive" or "all negative"
- Example (restaurant):



To wait or not to wait is still at 50%

- Example (abstract):



Goals for Splitting Nodes

- We want:

  (summed "diversity" within children) < ("diversity" in parent)
- Data points should be
  - Homogeneous (by labels) within leaves
  - Different between leaves
- Goal: try to increase purity within subsets
  - *Optimization* goal in each node: find the *attribute* and a *cutpoint* that splits the set of samples into two subsets with *optimal purity*
  - This attribute is the "most discriminative" one for that data (sub-) set
- Question: what is a good measure of purity for two given subsets of our training set?

Digression: Information Gain in Politics/Journalism

- Politician *X* is accused of doing something wrong
- He is asked (e.g., by journalists): "Did you do it?"
- The opposition (assuming *X* is a member of the ruling party) is asked: "Do you think he did it?"
- The answers are reported in the news …
- What information do you gain?

Information Gain

- Enter the information theoretic concept of information gain
- Imagine different events:
  - The outcome of rolling a dice = 6
  - The outcome of rolling a *biased* dice = 6
  - Each situation has a different amount of uncertainty whether or not the event will occur
- Information = *amount of reduction in uncertainty* (= amount of surprise if a specific outcome occurs)

- Quiz:
  - I am thinking of an integer number in [1,100]
  - How many yes/no questions do you need at most to find it out?
  - Answer: $\lceil \log_2 100 \rceil = 7$
- Definition Information Value:
  - Given a set S, the maximum amount of work required to determine a *specific* element in S by traversing a decision tree is

$$\log_2 |S|$$

  - Call this value the information value of being *told* the element, rather than having to *work* for it (by asking binary questions)

- Let Y be a random variable; we make one observation of the variable Y (e.g., we draw a random ball out of a box) $\longrightarrow$ value y
- The information we obtain if event "$Y = y$" occurs, i.e., the *information value* of that event, is

$$I[Y = y] = \log_2 \left(\frac{\#\text{ balls in box}}{\#\text{ y's in box}} \right) = \log_2 \frac{1}{p(y)} = -\log p(y)$$

  - "If the probability of this event happening is small and it does happen, then the information value is large"
- Examples:
  - Observing the outcome of coin flip $\longrightarrow \; I = -\log \frac{1}{2} = 1$
  - Observing the outcome of dice == $x \longrightarrow \; I = -\log \frac{1}{6} = 2.58$

Entropy

- A random variable Y (= experiment) can assume different values $y_1, \ldots, y_n$ (i.e., the experiment can have different outcomes)
- What is the *average* information we obtain by observing the random variable?
  - In other words: if I pick a value y_i at random, according to their respective probabilities – what is the *average* number of yes/no questions you need to ask to determine it?
  - In probabilistic terms: what is the *expected amount of information*? $\longrightarrow$ captured by the notion of entropy
- Definition: Entropy

  Let Y be a random variable. The entropy of Y is

$$H(Y) = E[I(Y)] = \sum_i p(y_i) I[Y = y_i] = -\sum_i p(y_i) \log p(y_i)$$

  Units = bits

- Interpretation: The number of yes/no questions (= bits) needed *on average* to pin down the value of *y* in a random drawing

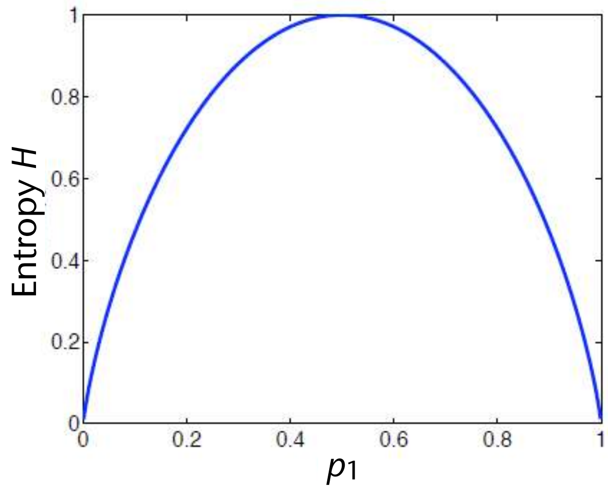
- Example: if *Y* can assume 8 values, and all are equally likely, then

$$H(Y) = -\sum_{i=1}^{8} \frac{1}{8} \log \frac{1}{8} = \log 2^3 = 3 \text{ bits}$$

- In general, if there are k different possible outcomes, then

$$H(Y) \leq \log k$$

  - Equality holds when all outcomes are equally likely
- With $k = 2$ (two outcomes), entropy looks like this ($p_1 + p_2 = 1$) :
- The more the probability distribution deviates from uniformity, the lower the entropy
- *Entropy* measures the *impurity*:



Balls-in-bin model

This distribution is less uniform =
Entropy is lower =
The node is purer

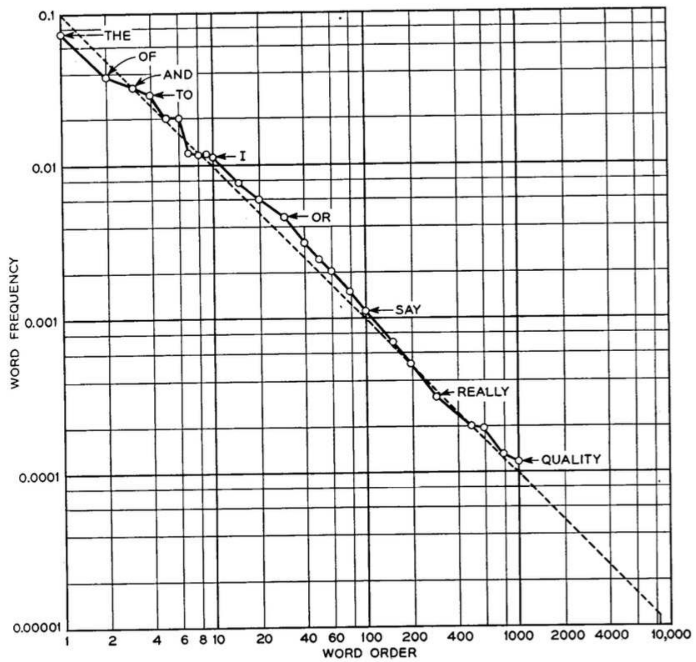
Examples

- Entropy of printed English
  - Let L = random variable, values = letters, picked randomly from a random English text
  - $H(L) = -p(\text{'}E\text{'}) \log p(\text{'}E\text{'}) - p(\text{'}T\text{'}) \log p(\text{'}T\text{'}) - p(\text{'}A\text{'}) \log p(\text{'}A\text{'}) - \ldots$

    $= 4.175$ bits

- Entropy of English words
  - Statistics of large English texts show $p_k \approx 0.1\frac{1}{k}$ where p_k = probability of word of rank k, up to rank 10 000 (Zipf's law)
  - Thus,

$$H \approx \sum_{k=1}^{10\,000} \frac{0.1}{k} \log_2\left(\frac{k}{0.1}\right) = 9.36 \text{ bits}$$

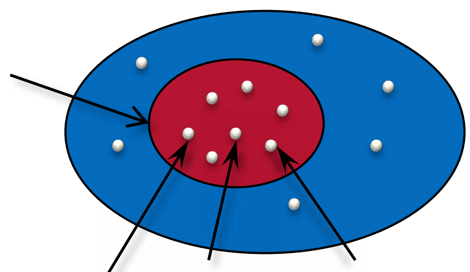


Conditional Entropy

- Now consider a random variable Y (e.g., the different classes/labels) with an attribute X (e.g., the first variable, $x_{i,1}$, of the data points, $\mathbf{x}_i$)
  - With every drawing of Y, we also get a value for the associated attribute X
- Assume that X is discrete, i.e., $x_i \in \{1, 2, ..., z\}$
- Now consider only outcomes of Y that fulfill some *condition*, e.g., $x_i = 1$
- The entropy of Y, provided that it assumes only values with $x_i = 1$:

$$H(Y|x_i = 1) = -\sum_i p(y_i|x_i = 1) \log \underbrace{p(y_i|x_i = 1)}$$

Probability of y_i occurring as a value of Y, where we draw Y only from the subset that contains only data points that have attribute $x_i = 1$

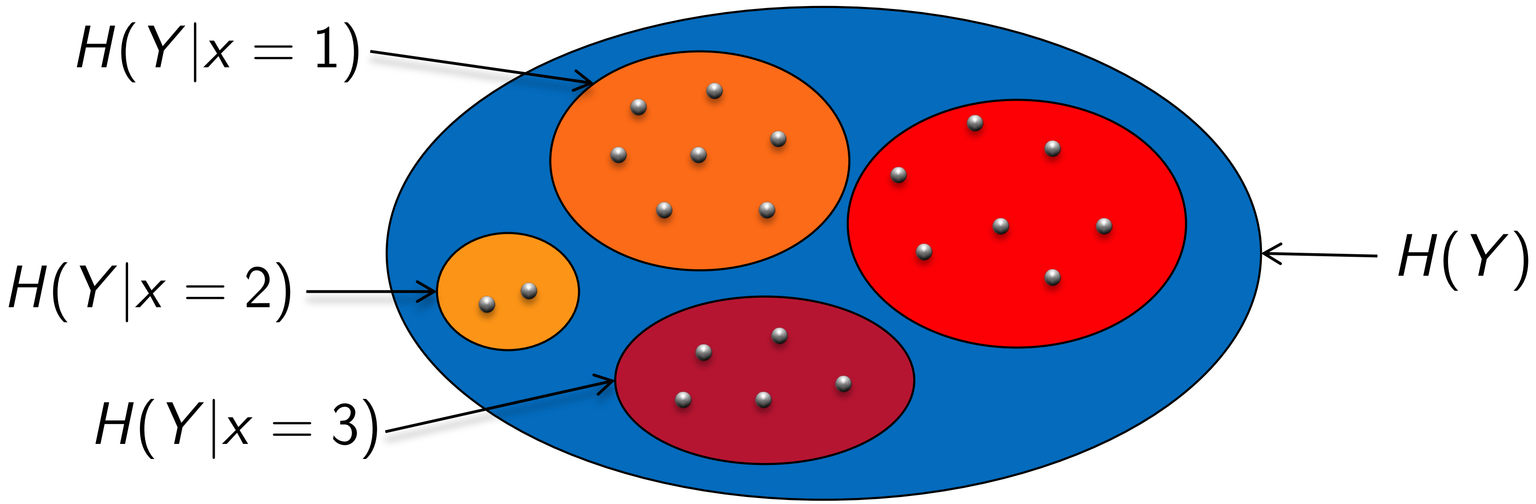


- Overall conditional entropy:

Probability that the attribute X has value k

$$H(Y|X) = \sum_{k=1}^{z} p(x=k) \cdot H(Y|x=k)$$

$$= -\sum_{k=1}^{z} p(x=k) \sum_{i} p(y_i|x_i=k) \log p(y_i|x_i=k)$$



Information Gain

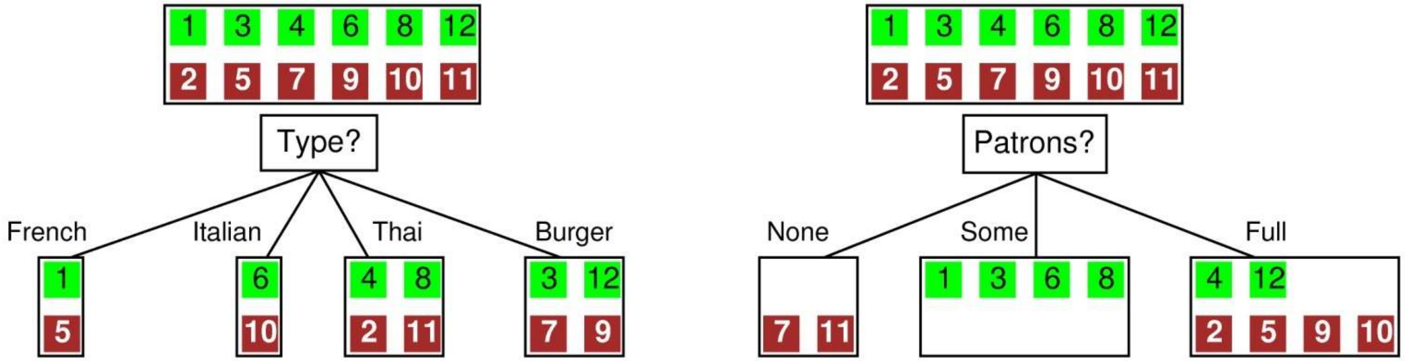
- How much information do we gain *if we disclose (or choose) the value of one attribute X*?
  - Disclosure $\longrightarrow$ splitting of the set of all data points into subsets
- Information gain = (information *before* split) – (information *after* split) = *reduction of uncertainty regarding label y* by learning value of attribute *X*
- The information gained by a split in a node of a decision tree:

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

  - Hopefully / usually $H(Y|X) < H(Y)$
- Goal: choose the attribute with the largest *IG*
  - In case of scalar attributes, also choose the *optimal cutpoint*
    - In doing so, we basically convert the scalar attribute into a binary one (at that node!)

Example

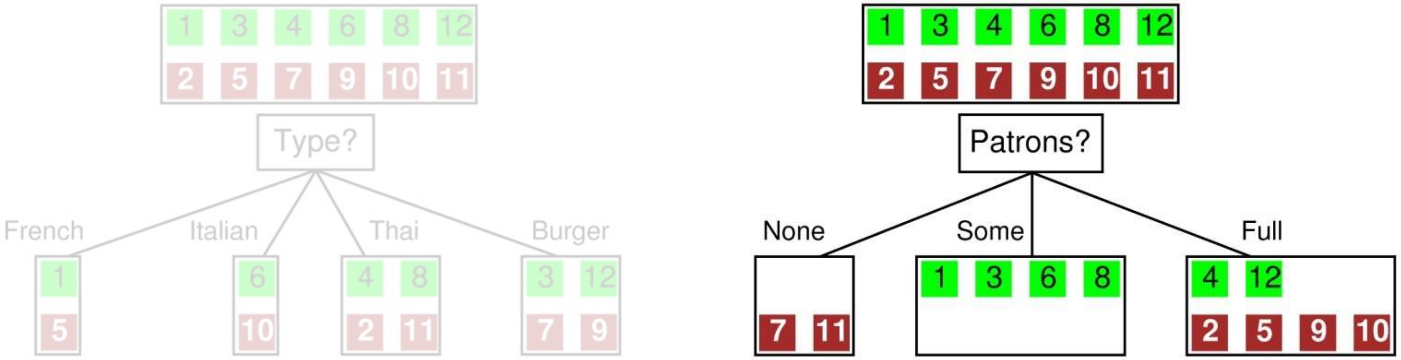
- Consider 2 options to split the root node of the restaurant example



- Labels of random variable $Y \in \{$ "yes", "no" $\}$
- Entropy at the root node:

$$H(Y) = p(y = \text{“yes”}) \log \frac{1}{p(y = \text{“yes”})} + p(y = \text{“no”}) \log \frac{1}{p(y = \text{“no”})}$$

$$= \frac{1}{2} \log 2 + \frac{1}{2} \log 2 = 1$$



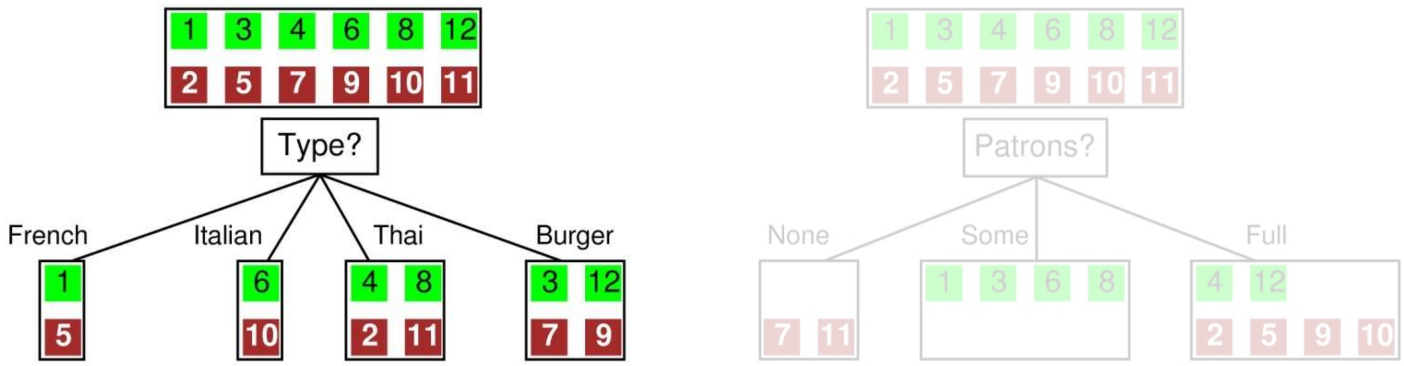
- Conditional entropy for split by #patrons:

$$H(Y \mid n) = p(n=\text{“full”})\cdot H(Y|n=\text{“full”})+ p(n=\text{“some”})\cdot H(Y|n=\text{“some”})+ p(n=\text{“none”})\cdot H(Y|n=\text{“none”})$$

where n = the attribute "#patrons" ∈ { "none", "some", "full" }

$$H(Y|n) = -\frac{6}{12}\big(p(y=\text{“no”})\log p(y=\text{“no”}) + p(y=\text{“yes”})\log p(y=\text{“yes”})\big) -\frac{4}{12}\big(p(y=\text{“no”})\log p(y=\text{“no”}) + p(y=\text{“yes”})\log p(y=\text{“yes”})\big) -\frac{2}{12}\big(p(y=\text{“no”})\log p(y=\text{“no”}) + p(y=\text{“yes”})\log p(y=\text{“yes”})\big)$$

$$H(Y|n) = \frac{6}{12}\Big(\frac{4}{6}\log\frac{6}{4} + \frac{2}{6}\log\frac{6}{2}\Big) + \frac{4}{12}(0\log 0 + 1\log 1) + \frac{2}{12}(1\log 1 + 0\log 0)$$



- Conditional entropy for split by restaurant type:

$$
\begin{aligned}
H(Y|\text{type}) = \; & -\frac{2}{12}\left(p(y=\text{“no”})\log p(y=\text{“no”}) + p(y=\text{“yes”})\log p(y=\text{“yes”})\right) \\
& -\frac{2}{12}\left(p(y=\text{“no”})\log p(y=\text{“no”}) + p(y=\text{“yes”})\log p(y=\text{“yes”})\right) \\
& -\frac{4}{12}\left(p(y=\text{“no”})\log p(y=\text{“no”}) + p(y=\text{“yes”})\log p(y=\text{“yes”})\right) \\
& -\frac{4}{12}\left(p(y=\text{“no”})\log p(y=\text{“no”}) + p(y=\text{“yes”})\log p(y=\text{“yes”})\right)
\end{aligned}
$$

$$
H(Y|\text{type}) = 2\cdot\frac{2}{12}\left(\frac{1}{2}\log\frac{2}{1} + \frac{1}{2}\log\frac{2}{1}\right) \; + \; 2\cdot\frac{4}{12}\left(\frac{2}{4}\log\frac{4}{2} + \frac{2}{4}\log\frac{4}{2}\right)
$$

- Compare the information gains:

$$\begin{aligned} IG(Y, \#\text{patrons}) &= H(Y) - H(Y|\#\text{patrons}) \\ &= 1 - 0.585 \end{aligned}$$

$$\begin{aligned} IG(Y, \text{type}) &= H(Y) - H(Y|\text{type}) \\ &= 1 - 1 \end{aligned}$$

  - Result: learning the value of the attribute "#patrons" gives us more information about the label of Y

- Compute the *IG* obtained by a split induced by *each attribute*
  - In the restaurant case, the optimum is achieved by the attribute "#patrons" for splitting the set of data points at the root

Another Example

- Given the following data points in the parent node:

Attrib.	0	3	7	2	3	2	8	6	1	3
Label	G	G	R	G	G	G	R	R	G	R

  - Entropy: $H = -\frac{4}{10}\log_2\frac{4}{10} - \frac{6}{10}\log_2\frac{6}{10} = 0.97$

- One way to split them:

Attrib.	0	3	7	2	3 |	2	8	6	1	3
Label	G	G	R	G	G |	G	R	R	G	G

  - Entropies: $H_L = -\frac{4}{5}\log_2\frac{4}{5} - \frac{1}{5}\log_2\frac{1}{5} = 0.72$

    $H_R = -\frac{3}{5}\log_2\frac{3}{5} - \frac{2}{5}\log_2\frac{2}{5} = 0.97$

    $$\Big\} \; H_{\text{after}} = \frac{5}{10}H_L + \frac{5}{10}H_R = 0.85$$

  - Information gain: $IG = H_{\text{before}} - H_{\text{after}} = 0.03$

Slight bug in the numbers!

- Another way to split them:

Attrib.	0	1	2	2	3	3	3	6	7	8
Label	G	G	G	G	G	G	R	R	R	R

- Entropies:

$$\left.\begin{aligned} H_L &= -\frac{4}{4}\log_2\frac{4}{4} - \frac{0}{4}\log_2\frac{0}{4} = 0 \\ H_R &= -\frac{2}{6}\log_2\frac{2}{6} - \frac{4}{6}\log_2\frac{4}{6} = 0.92 \end{aligned}\right\} H_{\text{after}} = \frac{4}{10}H_L + \frac{6}{10}H_R = 0.55$$

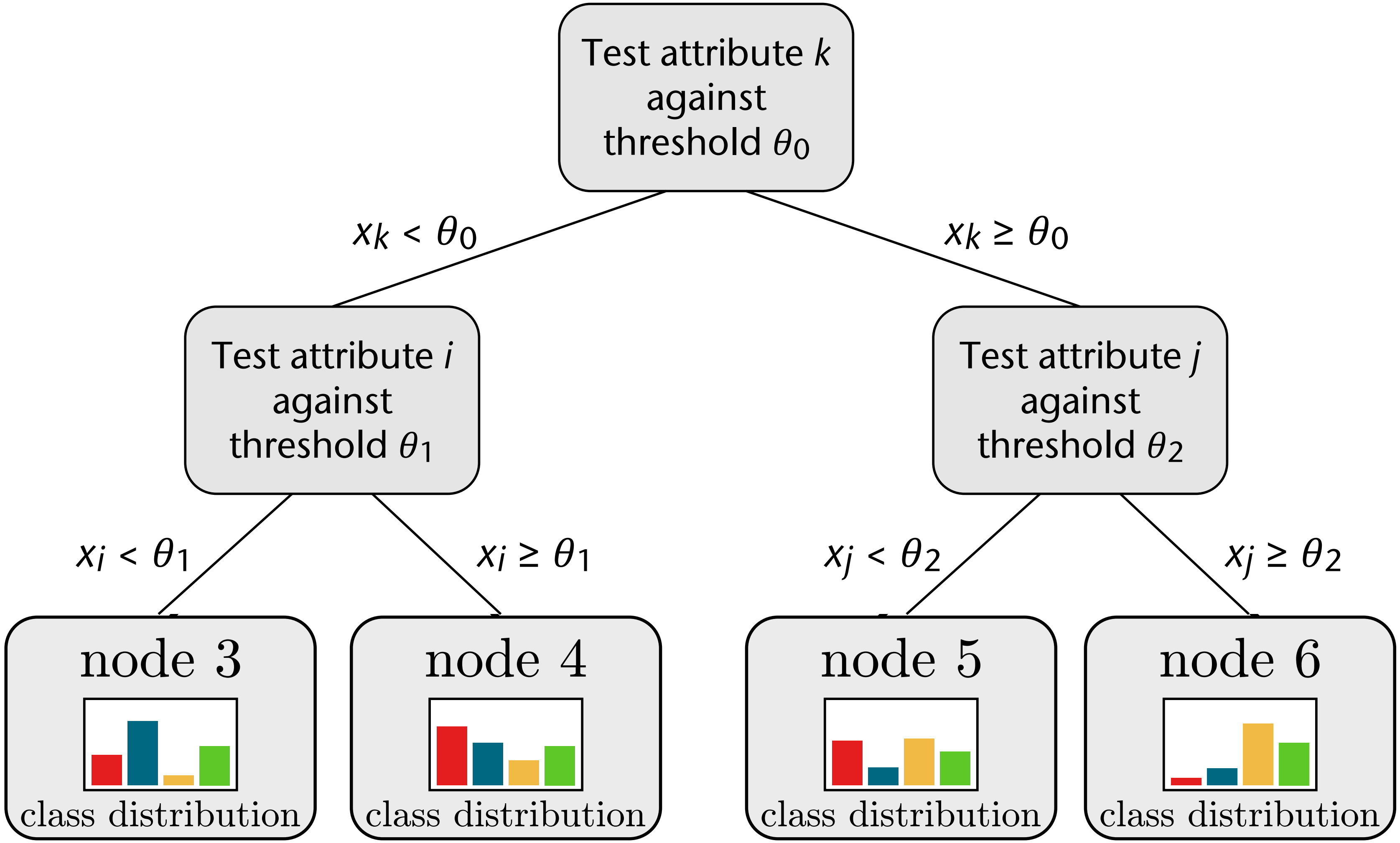
- Information gain: $IG = H_{\text{before}} - H_{\text{after}} = 0.42$

Bits and Pieces

- If there are no attributes left:
  - Can happen during learning of the decision tree, when a node contains data points with same attribute values but different labels
  - This constitutes error / noise in the training data
  - Stop construction here, use majority vote (i.e., discard erroneous point)
- If there are leaves with no data points:
  - While classifying a new data point
  - Just choose the majority vote of the parent node

Classification at Runtime

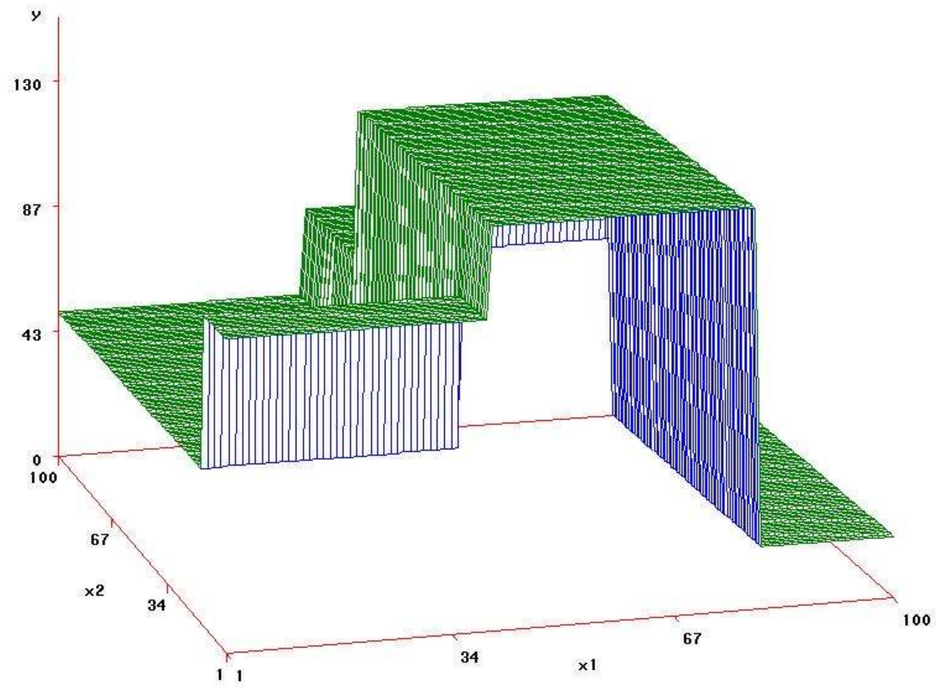
- Given an (unseen) data point **x**, traverse the tree, testing one of its attributes at each node



Expressiveness of Decision Trees

- Assume all variables (attributes and labels) are Boolean
- What is the set of Boolean functions that can be represented by a decision tree?
- Answer: all Boolean functions!
- Proof:
  - Given any Boolean function
  - Convert it to a truth table
  - Consider each row as a data point, output of the fct = label of data point
  - Construct a DT over all data points / rows

- If *Y* is a discrete, numerical variable, then DTs can be regarded as piecewise constant functions over the feature space:

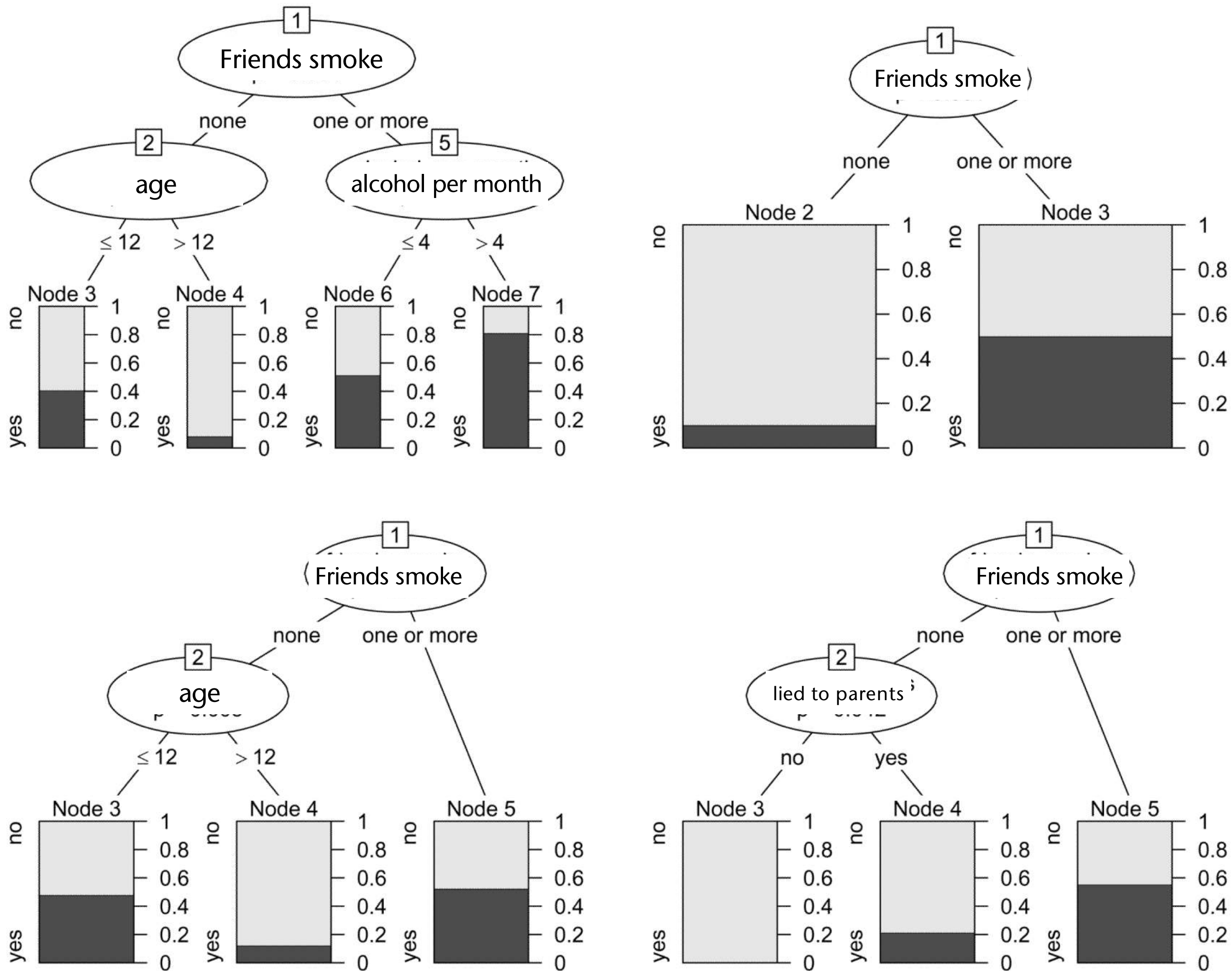


- DTs can approximate *any* function

Problems of Decision Trees

- Error propagation:
  - Learning a DT is based on a series of *local* decisions
  - What happens, if one of the nodes implements the wrong decision? (e.g., because of an outlier)
  - The whole subtree will be wrong!
- Overfitting: in general, it means the classifier performs extremely well on the training data, but very poorly on unseen data ⟶ low generalization capability
  - When overfitting occurs, the DT has "learned the noise in the data"

Example for the instability of single decision trees



"The Wisdom of Crowds"

[James Surowiecki, 2004]

- Francis Galton’s experience at the 1906 West of England Fat Stock and Poultry Exhibition
- Jack Treynor’s jelly-beans-in-the-jar experiment (1987)
  - Only 1 of 56 students' guesses came closer to the truth than the average of the class’ guesses
- Who Wants to Be a Millionaire?
  - Call an expert? ⟶ 65% correct
  - Ask the audience? ⟶ 91% correct

Example (Thought Experiment)

- "Which person from the following list was *not* a member of the Monkees?"

  (A) Peter Tork (C) Roger Noll

  (B) Davy Jones (D) Michael Nesmith

  - (BTW: Monkeys are a 1960s pop band, comprising 3 band members)
- Correct answer: the non-Monkee is Roger Noll
- Now imagine a crowd of 100 people with this distributed knowledge:

  7 know 3 of the Monkees

  10 know 2 of the Monkees

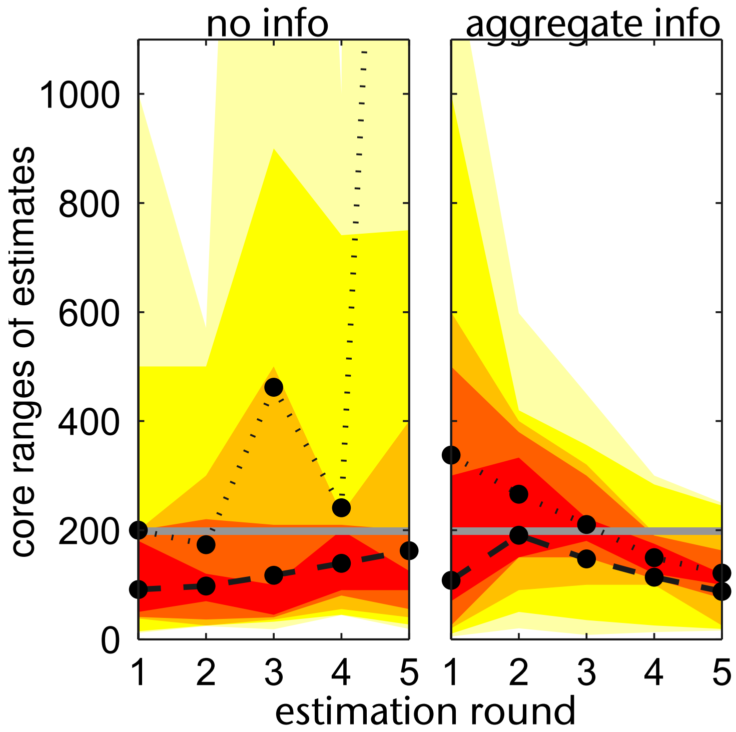
  15 know 1 of the Monkees

  68 have no clue
- So "Noll" will garner, on average, 34 votes versus 22 votes for each of the other choices
  - (68/4 + (15/3)/3*3 + (10/3)/2*3 + 7 = 34)

- Implication: one should not spend energy trying to identify an expert within a group, but instead rely on the group's collective wisdom
- Counter example:
  - Kindergartners guessing the weight of a Boeing 747
- Prerequisites for crowd wisdom to emerge:
  - Some knowledge of the truth must reside with some group members (⟶ *weak classifiers*)
  - Opinions must be independent
  - Knowledge must be objective (no subjective opinions)
  - Works best for quantifiable things (need to calculate the average) ("if you can count it, you can crowd it")

Digression: the Stupidity of Crowds

- Examples:
  - Financial crisis in 2008
  - Bubble formation in social networks
- Social experiment (N = 144) [2011]:
  - Several estimation tasks (country's population, etc.)
  - Conditions:
    - No info: subjects had no information about other participants' guesses
    - Aggregate info: subjects could reconsider their estimate after gaining some information about the estimates of others
  - *Social influence effect*: diversity diminishes, but collective error does not
  - *Confidence effect*: subjects become more certain about their guesses



Bremen

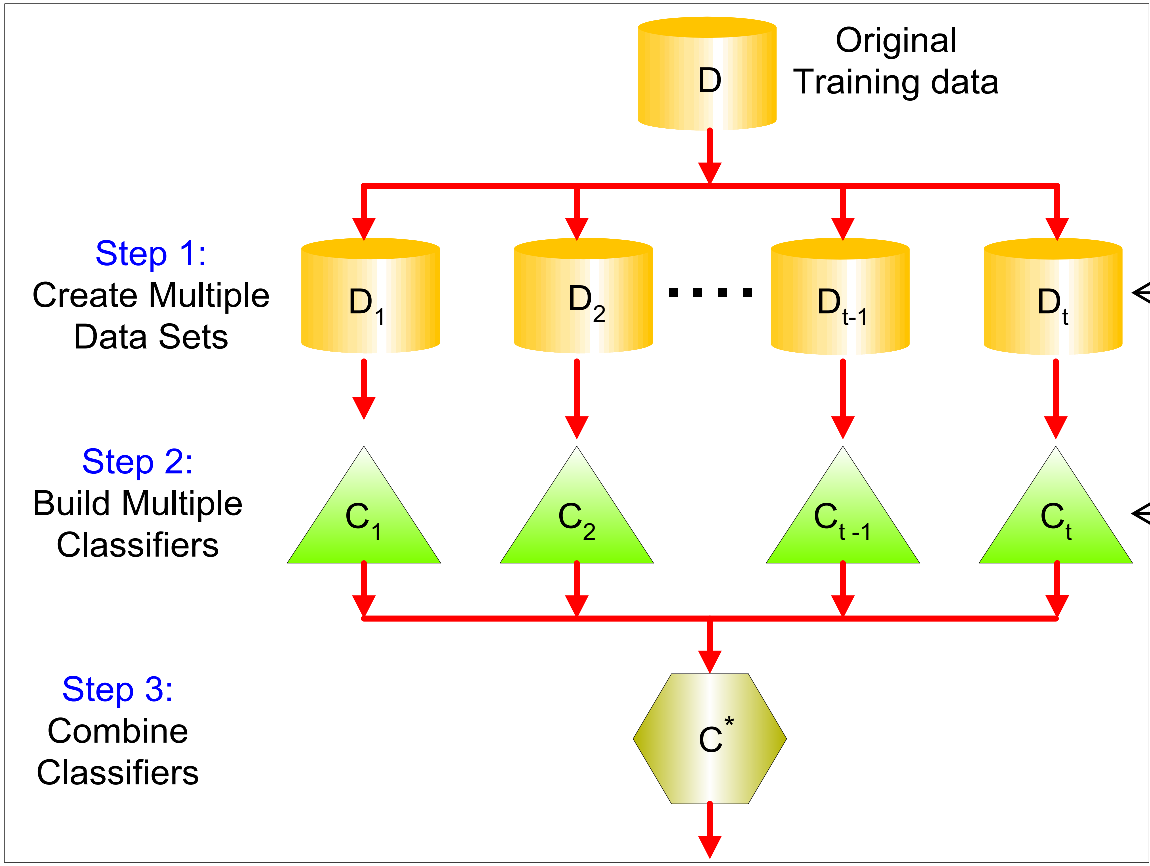
Digression: Francis Galton

- Cousin of Charles Darwin
- "Father" of statistics
- Incidentally, he also invented finger printing
- He also published the "scientific" way to cut cakes in Nature 1906:

Numberphile.com

The Random Forest (RF) Method

- One kind of so-called ensemble (of experts) methods
- Idea: predict class label for unseen data by *aggregating* a set of predictions (= classifiers learned from the training data)



Must encode the same distribution as the original data set *D*!

Each classifier = one decision tree

Randomizations During the Construction of RF's

- Generating the data sets for learning multiple trees:
  - Generate a number of random sub-sets $\mathcal{L}_1, \mathcal{L}_2, \ldots$ from the original training data $\mathcal{L}$, $\mathcal{L}_i \subset \mathcal{L}$. There are basically two methods:

  1. Bootstrapping: randomly draw samples from $\mathcal{L}$, with replacement, size of new data = size of original data set; or,
  2. Subsampling: randomly draw samples from $\mathcal{L}$, without replacement, size of new data < size of original data set

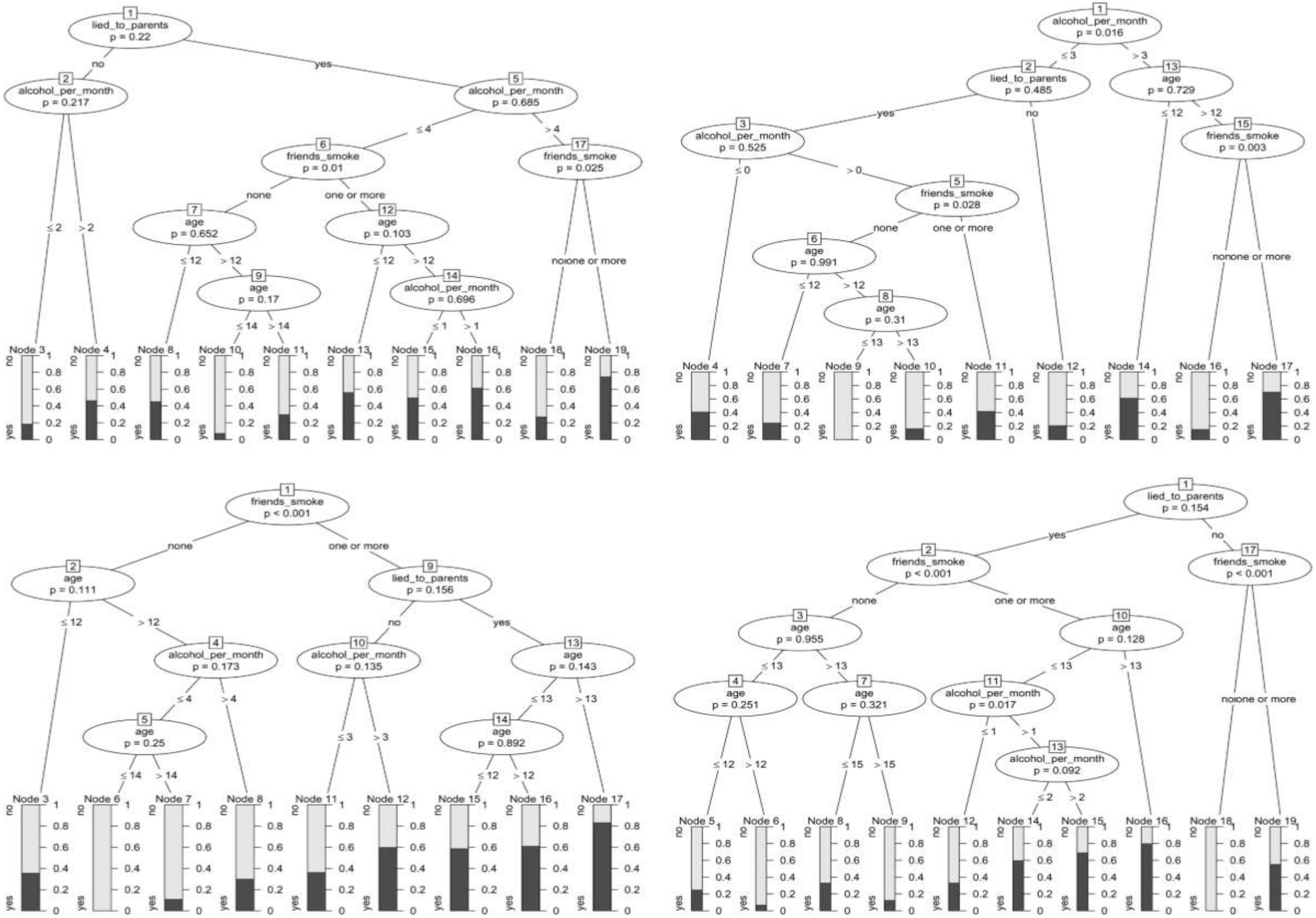
  - New data sets reflect the *same* random process as the orig. data, but they differ slightly from each other and the original set due to random variation
  - Resulting trees can differ substantially (see earlier slide)

- Growing the trees:
  - At *each node*, a random subset of attributes (= predictor variables/ features) is preselected; *only from those,* the one with the best information gain is chosen
    - NB: an individual tree is *not just a DT over a subspace of feature space*!
  - Each tree is grown without any stopping criterion, i.e., until each leaf contains data points of only *one single* class
- Naming convention for 2 essential parameters:
  - Number of trees = *ntree*
  - Size of random subset of variables/attributes at each node = *mtry*
- Rules of thumb:
  - *ntree* = 100 … 300
  - *mtry* = sqrt(d) , with d = dimensions of the feature space

- The learning algorithm:

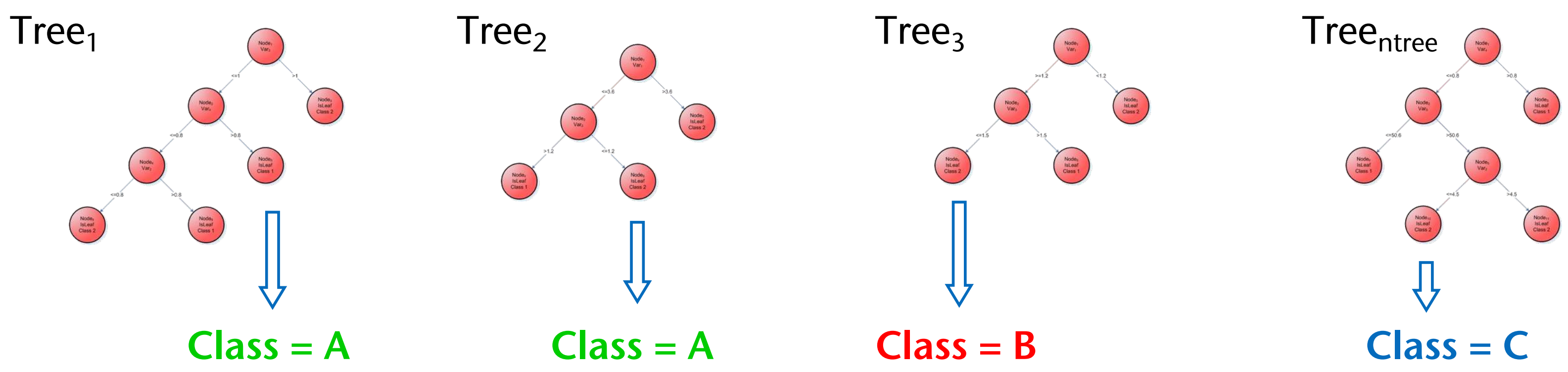
```
input: learning set L
for t = 1...ntree:
   build subset Lt from L by random sampling
   learn tree Tt from Lt:
      at each node:
         randomly choose mtry features
         compute best split from only those features
      grow each tree until leaves are perfectly pure
```

A Random Forest Example for the Smoking Data Set



Using a Random Forest for Classification

- With a new data point:
  - Traverse each tree individually using that point
  - Gives *ntree* many class labels



  - Take majority of those class labels
- Sometimes, if labels are cardinal numbers, (weighted) averaging makes sense

Why Does it Work?

- Make following assumptions:
  - The RF has *ntree* many trees (classifiers)
  - Each tree has an error rate of ε
  - All trees are perfectly independent! (no correlation among trees)
- Probability that the RF makes a wrong prediction:

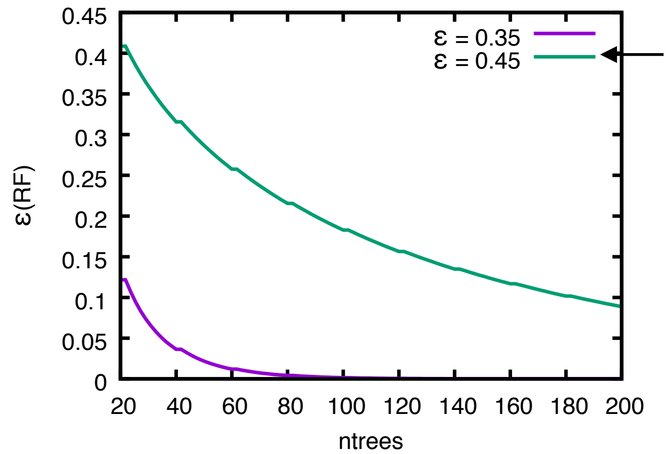
$$\varepsilon_{\text{RF}} = \sum_{i=\left\lceil \frac{ntree}{2} \right\rceil}^{ntree} \binom{ntree}{i} \varepsilon^i (1-\varepsilon)^{(ntree-i)}$$

- Example:

  individual error rate ε= 0.35 ,

  ntree = 60 ⟶

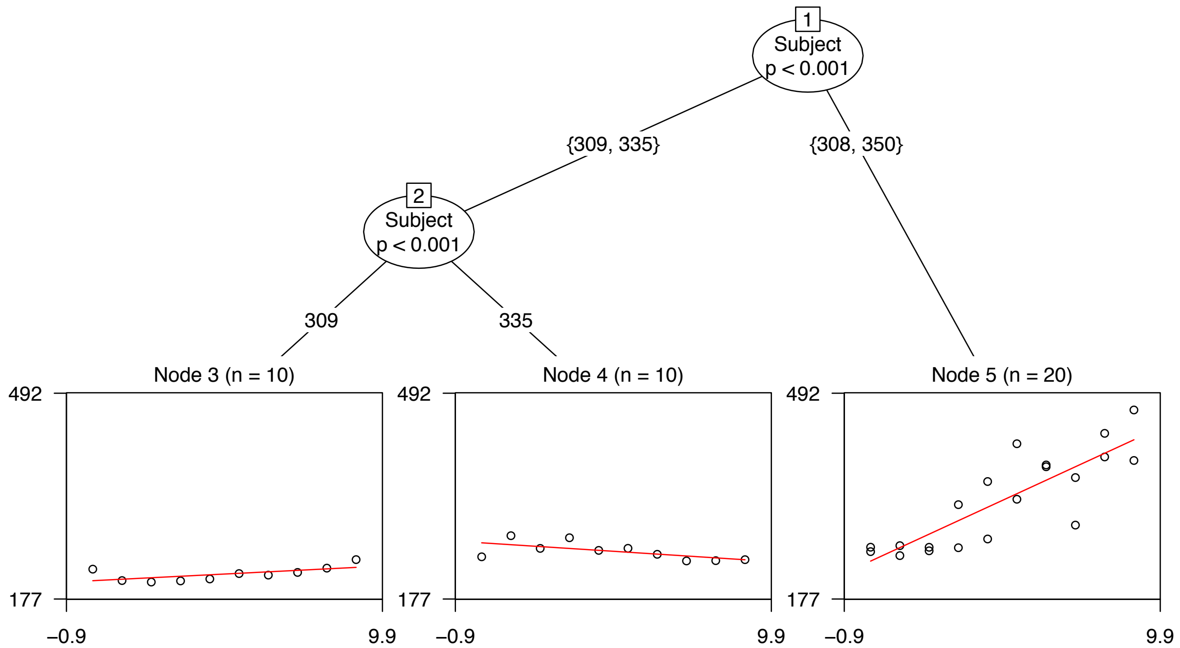
  error rate of RF $\varepsilon_{\text{RF}} \approx 0.01$



Each DT is only 5% better than randomly choosing the class

Variants of Random Forests

- Regression trees:
  - Variable Y (dependent variable) is continuous
    - I.e., no longer a class label
  - Goal is to learn a function $\mathbb{R}^d \rightarrow \mathbb{R}$ that generalizes the training data
  - Example:



- Extremely randomized trees (ERTs):
  - Do not find the optimal threshold for splitting the training set
  - Instead, just pick a random value in the interval of the feature's values
- Oblique random forests:
  - Do not test just one feature
  - Instead, test a linear combination of $k = mtry$ features: $\begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_k \end{pmatrix} \begin{pmatrix} x_{i_1} \\ x_{i_2} \\ \vdots \\ x_{i_k} \end{pmatrix} < \theta$
  - Variant "Forest-RC":
    - Randomly choose l different vectors of coefficients $a_i \in [-1,1]$, $i = 1,\ldots,k$
    - Pick that vector of a_i's that maximizes information gain
- Random ferns:
  - All nodes on the same level within a tree test the same attribute against the same threshold
  - Advantage: all decision tests at runtime can be done in parallel
  - Disadvantage: need deeper trees

Features of Random Forests

- "Small *n*, large *d*"
  - RFs are well-suited for problems with many more variables (d = dimensions in the feature space) than observations / training data (n)
- Nonlinear function approximation
  - RFs can approximate *any* unknown function
- RF's can solve the "XOR problem"
  - In an XOR truth table, the two variables show no effect at all
    - With either split variable, the information gain is 0
  - But there is a perfect interaction between the two variables
  - Random selection of $mtry < d$ variables can help in such cases

Tips and Tricks

- Out-of-bag error estimation:
  - For each tree T_i, a training data set $\mathcal{L}_i \subset \mathcal{L}$ was used
  - Use $\mathcal{L} \setminus \mathcal{L}_i$ (the out-of-bag data set) to test the prediction accuracy
- Handling of missing values:
  - Occasionally, some data points contain a missing value for one or more of its variables (e.g., because the corresponding measuring instrument had a malfunction)
  - When information gain is computed, ignore those data points with a missing value at the currently evaluated variable
  - During splitting, use a *surrogate* that best predicts the values of the splitting variable (in case of a missing value)
    - Assume data point has class label l, its m-th variable is missing: compute median of m-th variable of all data points in class l, use this as surrogate for all missing m-th variables of all data points

- Randomness:
  - Random forests are truly random
  - Consequence: when you build two RFs with the same training data, you get slightly different classifiers/predictors
    - Fix the random seed, if you need reproducible RFs
  - Suggestion: if you observe that two RFs over the same training data (with different random seeds) produce noticeably different prediction results, and different variable importance rankings, then you should adjust the parameters *ntree* and *mtry*

Remarks on RFs

- Do random forests overfit?
  - The evidence is inconclusive (with some data sets it seems like they could, with other data sets it doesn't)
  - If you suspect overfitting: try to build the individual trees of the RF to a smaller depth, i.e., not up to completely pure leaves
- Better explainability than CNN’s:
  - RF’s can provide information on which variables/features are important for the decision making (and which are unimportant)

Parallel Construction of Random Forests

- Naïve method: one thread per tree (not massively parallel)
- Better method: one thread per node
- In the following: "data point" actually means "index into data point array", i.e., threads always work with indices only
- General idea:
  - Build all trees breadth-first
  - In each iteration, each thread
    - gets a task = node of one of the DT's, and a list of data points,
    - determines input variable *i* and cutpoint θ for optimal split
  - Produces two new lists and allocates child nodes

The Algorithm in More Detail

```
create ntrees many subsets of training data
assign these subsets to the root nodes
while there are still inner nodes:
  repeat mtry many times:
    pick a random feature i
    sort all data points by value of feature i
    initialize left/right histograms, left h. = empty
    loop with k over data points left to right:
      conceptually move data point k from right to
        left subset ⟶ new θ
      update left/right histograms
      compute new information gain (IG)
      if new IG is better:
        save new θ and IG
    if feature i yields better IG:
      save new feature index i, θ, IG
  create child nodes
  split input data points by feature i and θ
    and create two subsets, one for each child node
```

Executed in parallel

Achieving Higher Parallelism

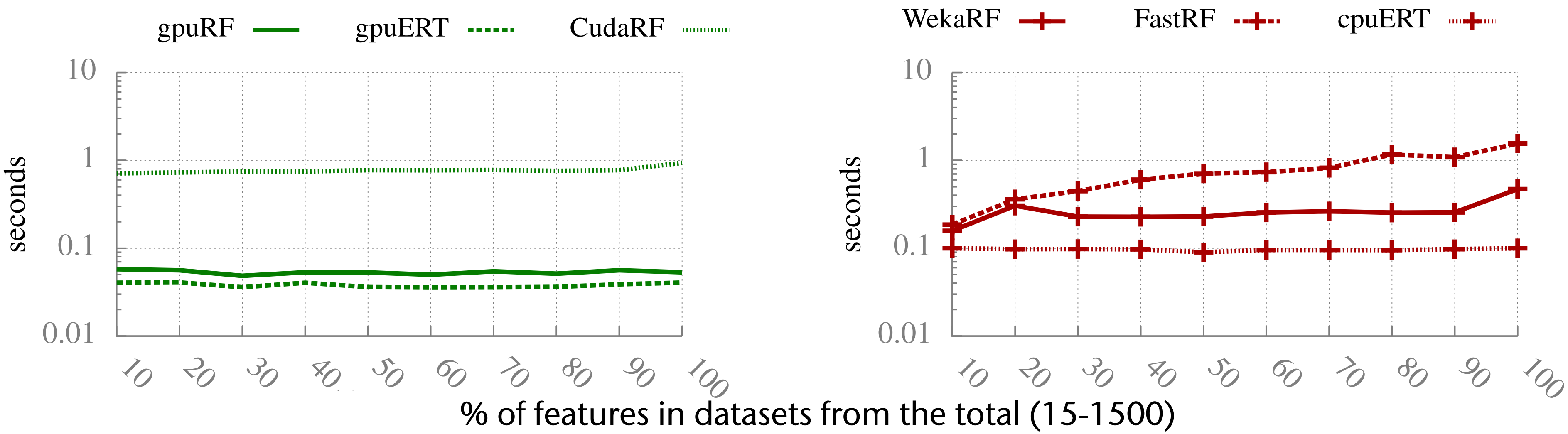
- At each node: calculate IG for *mtry* many features and a fixed number, *s*, of potential cutpoints
- Let n = # inner nodes on the current level
- Launch n blocks of $s \times mtry$ many threads
  - Each thread computes the IG for one specific node, one specific feature i, one specific cutpoint θ
  - Output is a matrix of IG's per node, pick the maximum for the split
    - Segmented max-scan over array of $n \times s \times mtry$ elements, n segments, one segment = $s \times mtry$ many IG values
  - Advantage: all threads in a block work on the same set of data points $\longrightarrow$ load into shared memory

Updating the Histogram

- Given two sets of data points (left and right), and the associated histograms h_l and h_r
- Move one data point from right to left, let $y \in \{y_1, y_2, \ldots, y_l\}$ be its label
- The update method:

```
updateHistograms( hl, hr, y ):
  hr[ y ] -= 1
  hl[ y ] += 1
```

Training Time Depending on Size of Dataset



gpuRF / gpuERT: the presented method for training RF and ERT on the GPU;
cpuERT: same algorithm, but implemented on the CPU running 32 threads;
CudaRF: older method on GPU
WekaRF, FastRF: multi-threaded CPU versions

Dataset	Nr Instances	Nr Features	mtry	Nr Missing values
Adult	32561	14	4	4262
Mushroom	8124	22	5	2480
Spambase	4601	57	6	0
Kr-vs-kp	3196	36	6	0
Eula-Freq	996	1268	11	0
Breast-Cancer-Wis	569	30	5	0
Skin-Disorder	462	1669	11	0
House-Votes	435	16	5	392

Some Code Optimization Tricks (not Only for GPU's)

- Instead of

```
if ( x[i] < threshold )
   child node ptr = left child ptr
else
   child node ptr = right child ptr
```

use

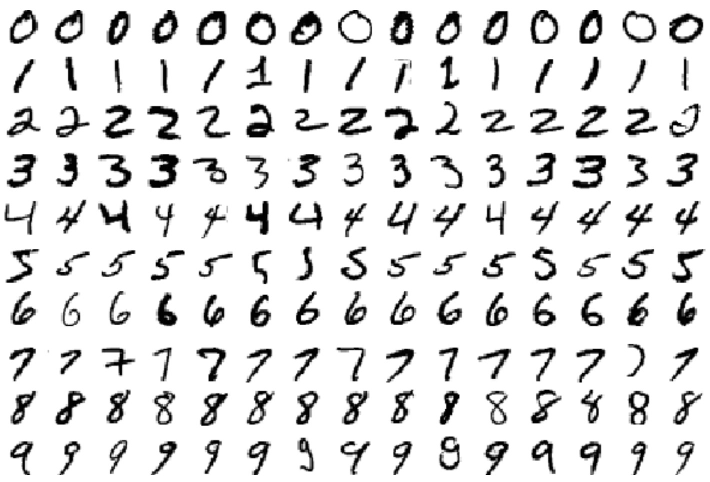
```
child node ptr = left child ptr +
                 static_cast<int>( x[i] >= threshold )
```

- Use half-precision floats for storing the training data set
  - FP16 = 16-bit floating point type **`half`** (since CUDA 7.5)
  - Increases bandwidth, allows 2 × data in shared memory
  - Lower precision is OK, since data set contains noise anyways

- Use `__log2f()` instead of `log2f()`
  - Less precision, but faster
  - Loss in precision does not matter here, because of all the other randomizations
- Use `__fdividef(x,y)` instead of division operator (`x/y`)
  - Twice as fast

Application: Handwritten Digit Recognition

- Data set:
  - Images of handwritten digits
  - 10 classes
  - Normalization: 20x20 pixels, binary images



MNIST data set

- Naïve feature vectors (data points):
  - Each pixel = one variable ⟶ 400-dim. feature space over {0,1}
  - Recognition rate: ~ 70-80 %
- Better feature vectors by *domain knowledge*:
  - For each pixel I(*i*,*j*) compute:

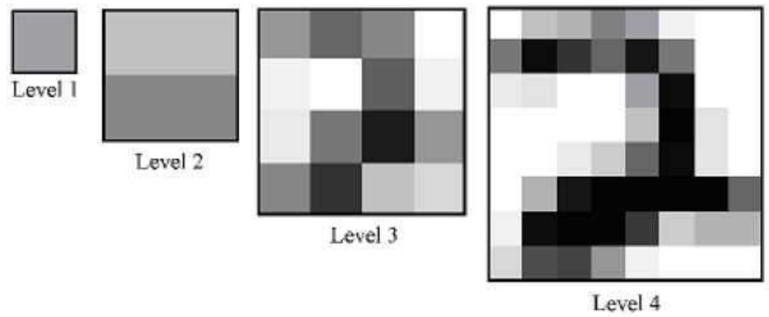
$$H(i,j) = I(i,j) \wedge I(i,j+2)$$
$$V(i,j) = I(i,j) \wedge I(i+2,j)$$
$$N(i,j) = I(i,j) \wedge I(i+2,j+2)$$
$$S(i,j) = I(i,j) \wedge I(i+2,j-2)$$

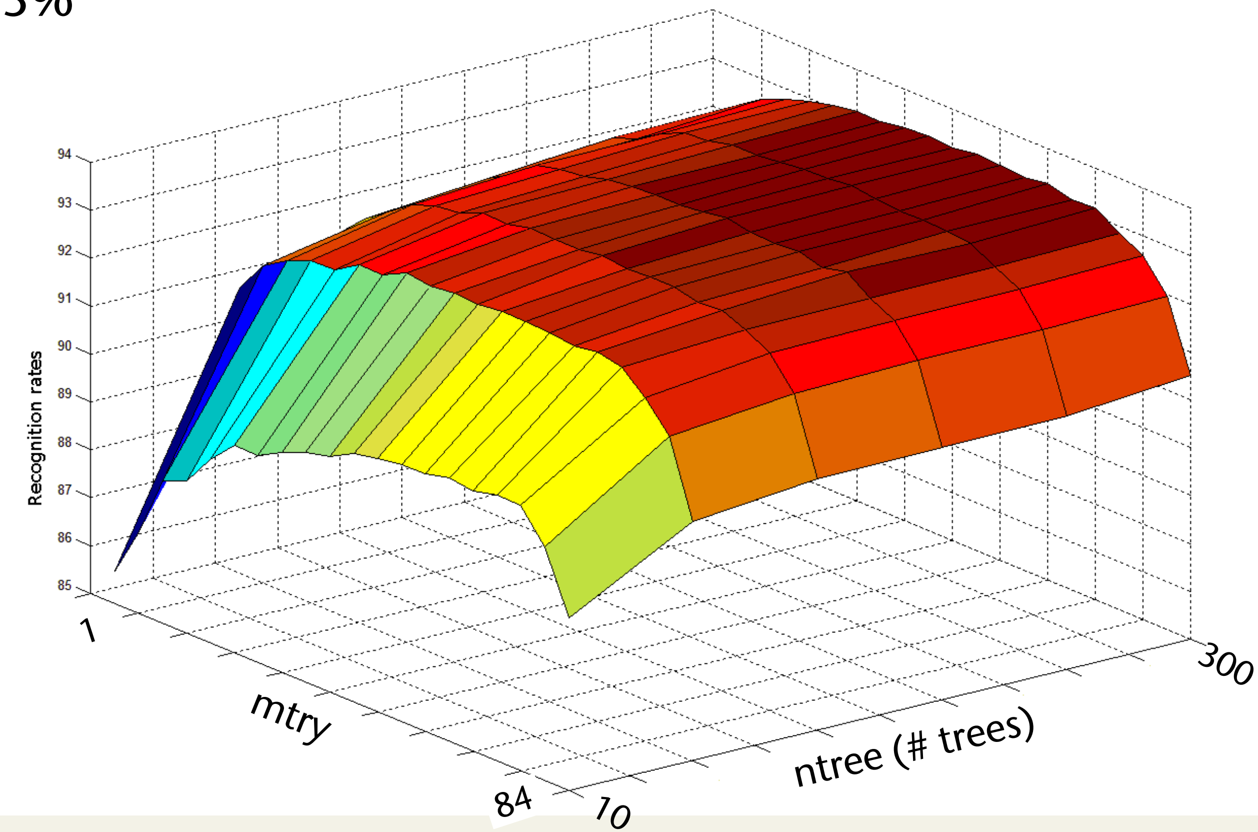
and a few more …

- Feature vector for an image = (all pixels $I(i,j)$, all $H(i,j)$, $V(i,j)$, …)
- Feature space = ca. 1400-dimensional = 1400 variables per data point

- Classification accuracy = ~93%
  - (NB: it was a precursor of random forests)

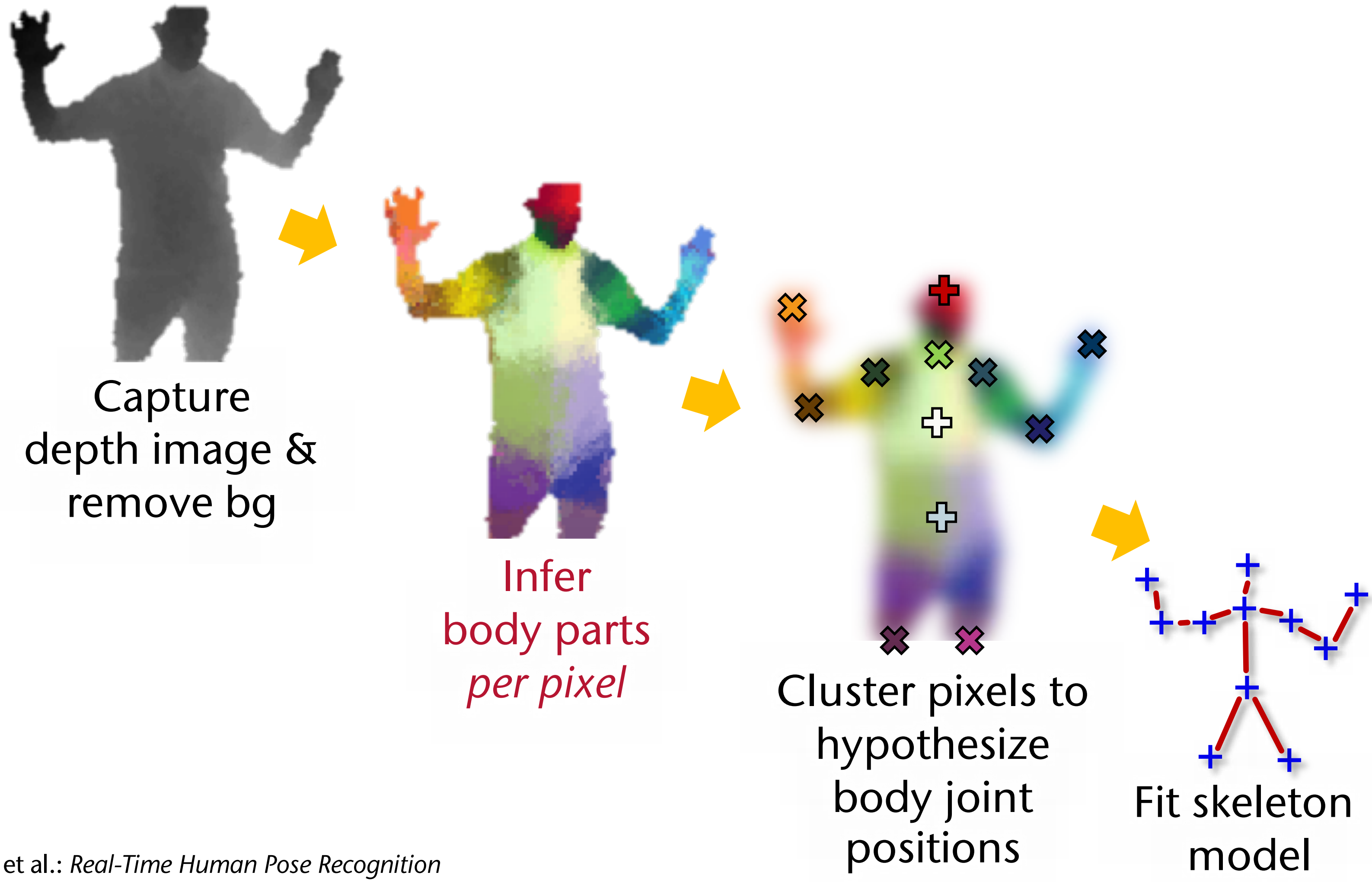
- Other experiments on handwritten digit recognition:
  - Feature vector = all pixels of an image pyramid
  - Recognition rate: ~ 93%
  - Dependence of recognition rate on *ntree* and *mtry*:





Body Tracking Using Depth Images (Kinect)

- The tracking / data flow pipeline:

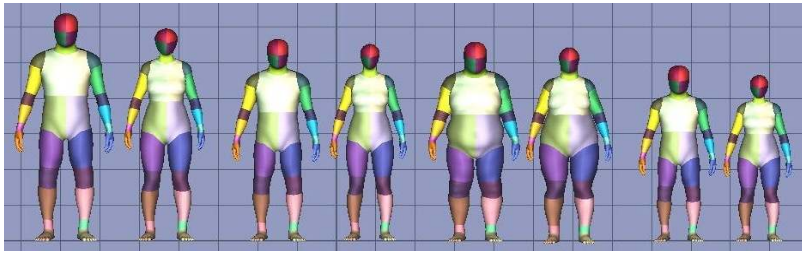


[Shotton et al.: *Real-Time Human Pose Recognition in Parts from Single Depth Images*; CVPR 2011]

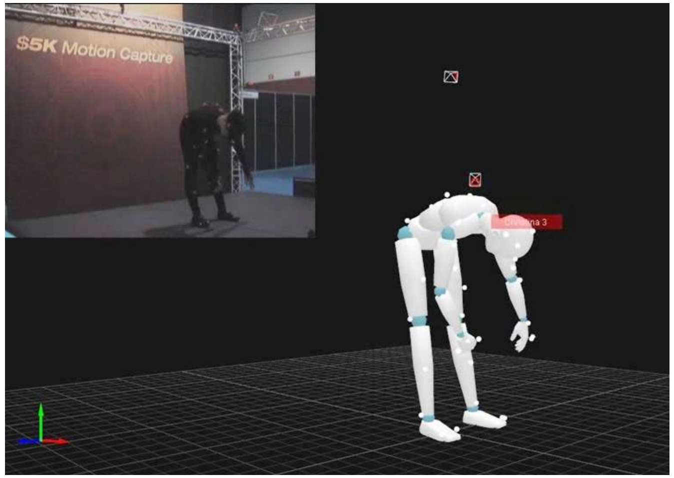
The Training Data

Record mocap
500k frames
distilled to 100k poses

Retarget to several models

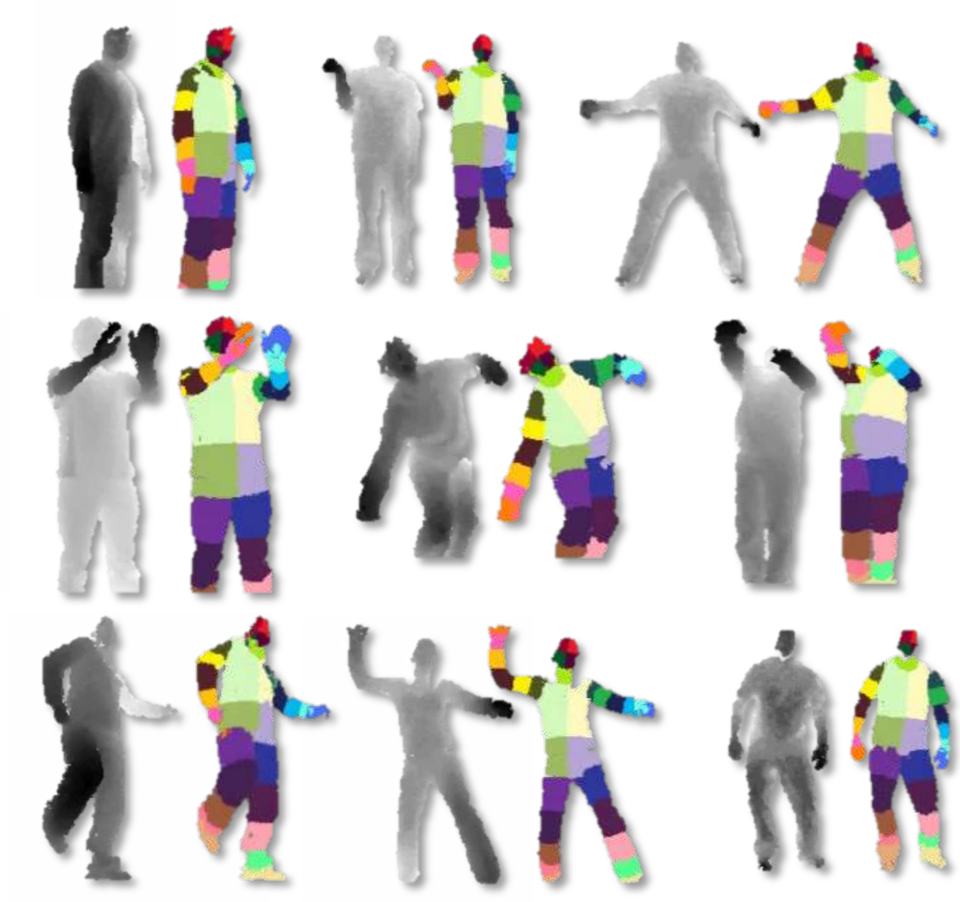


Render models: store depth & body part ID



Synthetic and Real Data

synthetic
(train & test)

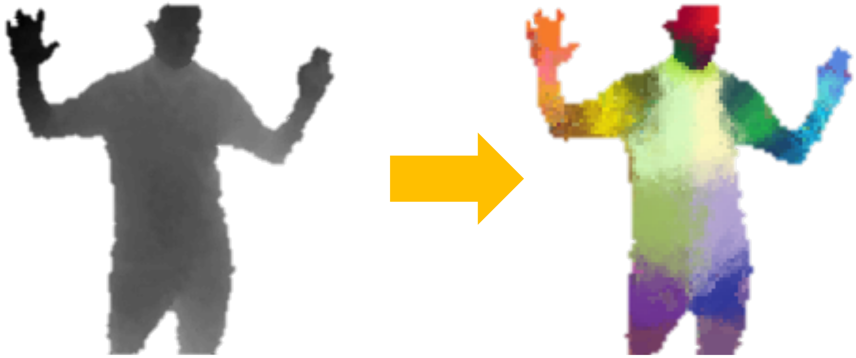


real
(test)

For each pixel in the synthetic depth image, we know its correct class (= label).
Sometimes, such data is also called ground truth data.
For the real test data, the pixels have been *hand labeled*.

Classifying Pixels

- Goal: for each pixel determine the most likely body part (head, shoulder, knee, etc.) it belongs to
- Classifying pixels = compute probability $P(c_\mathbf{x})$ for pixel $\mathbf{x} = (x,y)$, where $c_\mathbf{x}$ = body part
- Task: learn classifier that returns the most likely body part class $c_\mathbf{x}$ for every pixel $\mathbf{x}$
- Idea: consider a neighborhood around **x** (moving window)



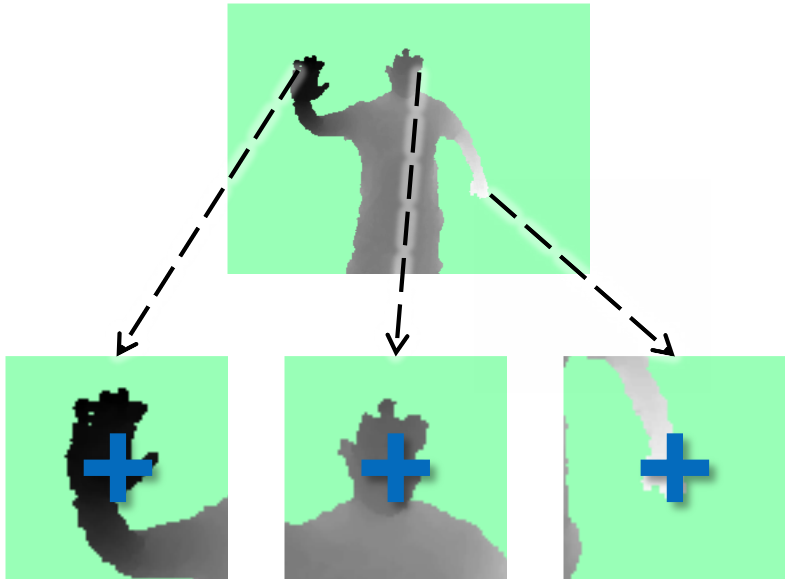


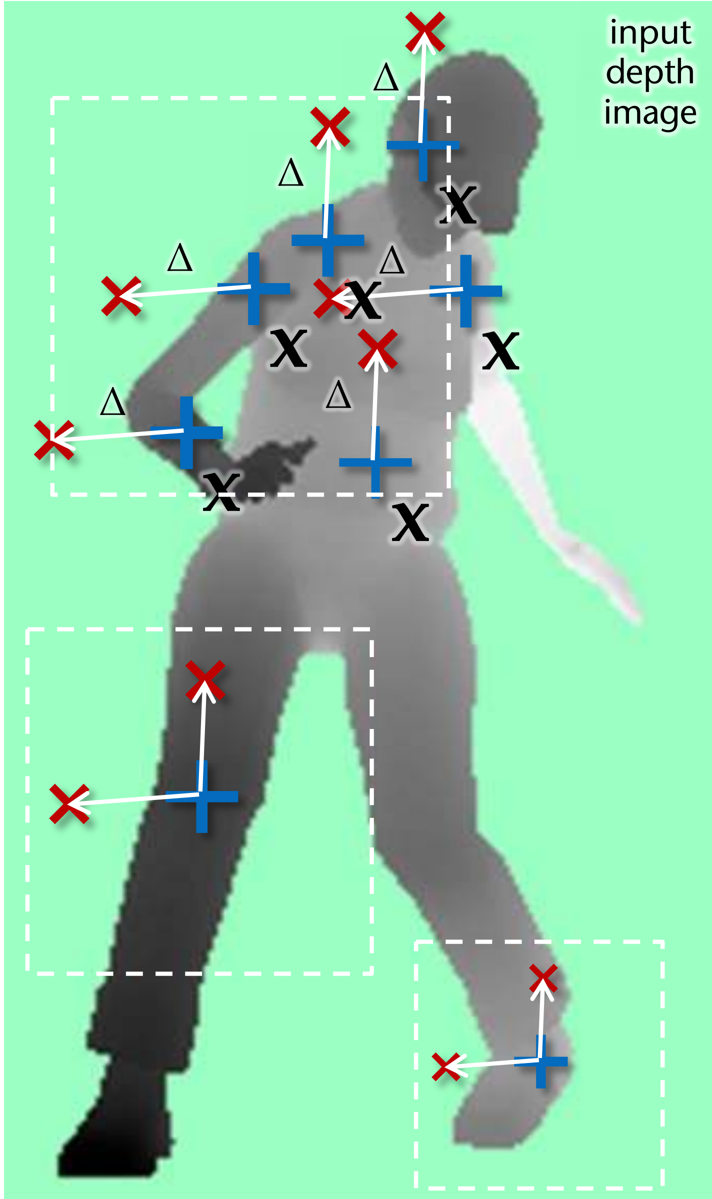
Image windows move with classifier

Fast Depth Image Features

- For a given pixel, consider all depth comparisons inside a window
- The *feature vector* for a pixel **x** are all *feature variables* obtained by all possible depth comparisons inside the window:

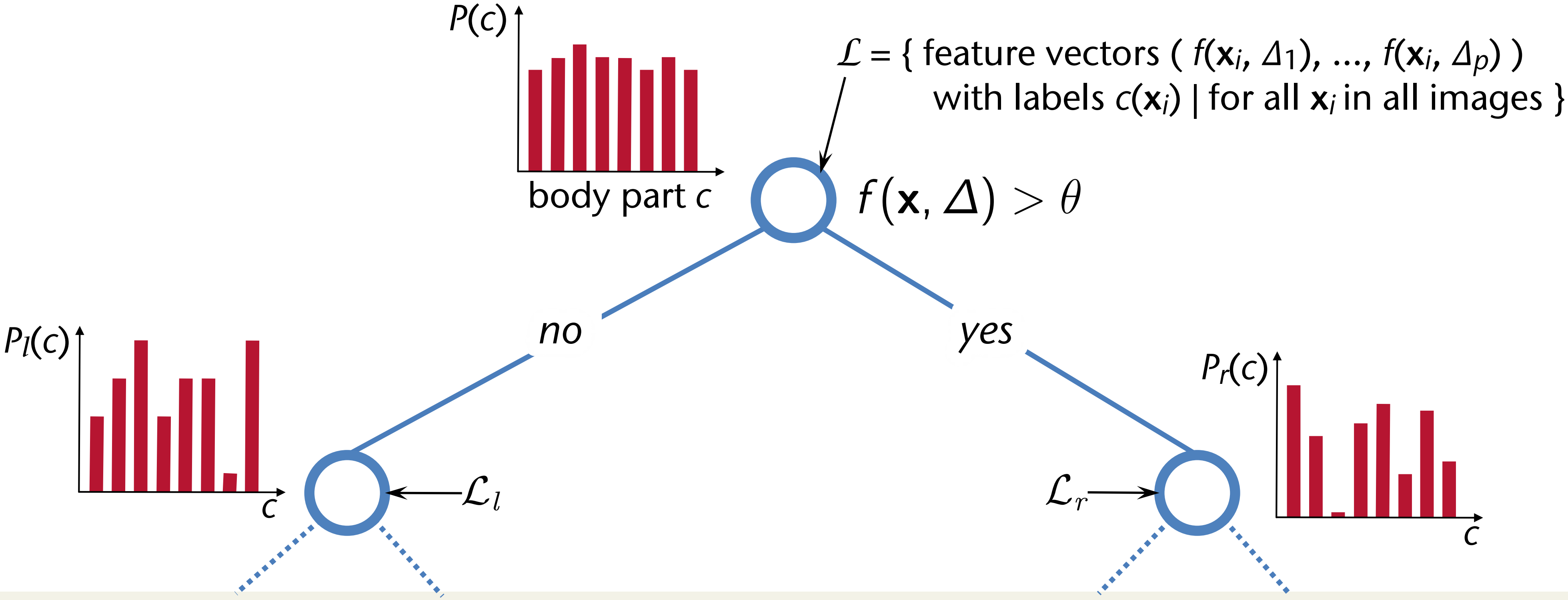
$$f(\mathbf{x}, \Delta) = D(\mathbf{x}) - D\left(\mathbf{x} + \frac{\Delta}{D(\mathbf{x})}\right)$$

  where D = depth image,
  $\Delta = (\Delta_x, \Delta_y)$ = offset vector,
  and D(background) = large constant
  - Note: scale Δ by 1/depth of **x**, so that the window shrinks with distance
- Features are very fast to compute



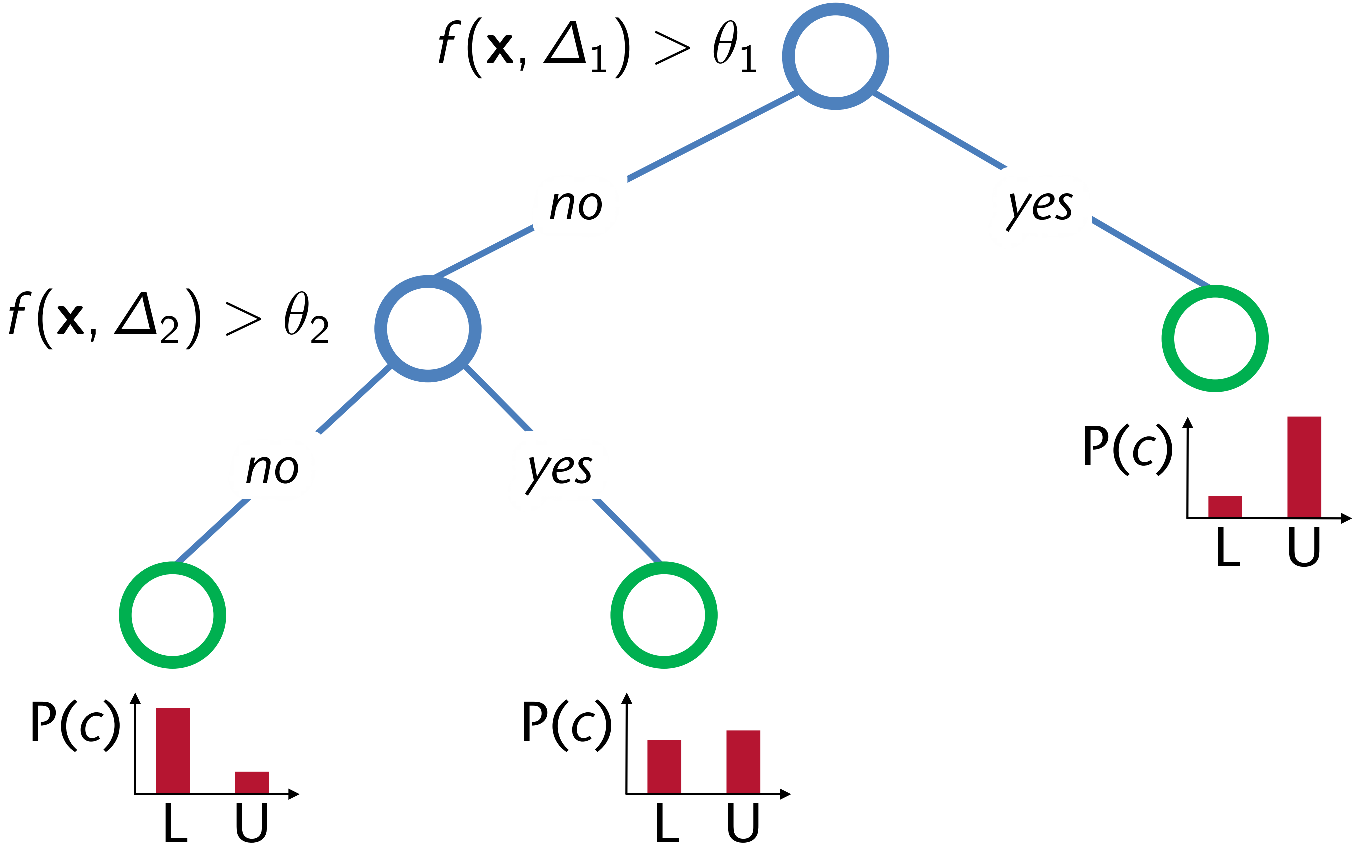
Training of a Single Decision Tree

- Conceptually, the training set $\mathcal{L}$ = { all feature vectors (= all $f(\mathbf{x}, \Delta)$) of all pixels of all training images }, together with the correct labels (= body part)
- Training a decision tree amounts to finding Δ and θ such that the information gain is maximized



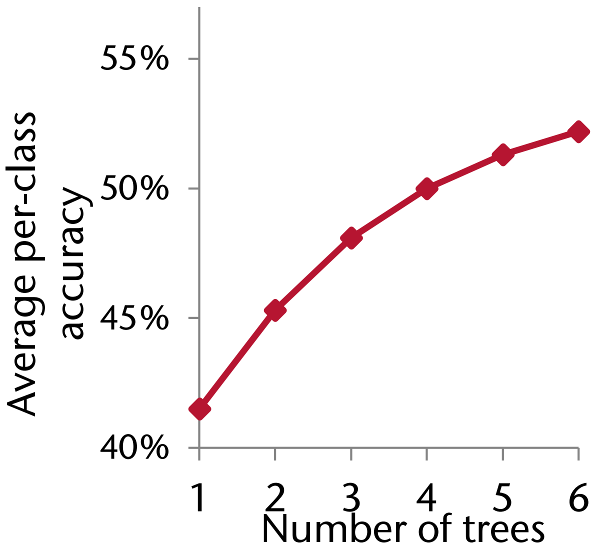
Classification of a Pixel at Runtime

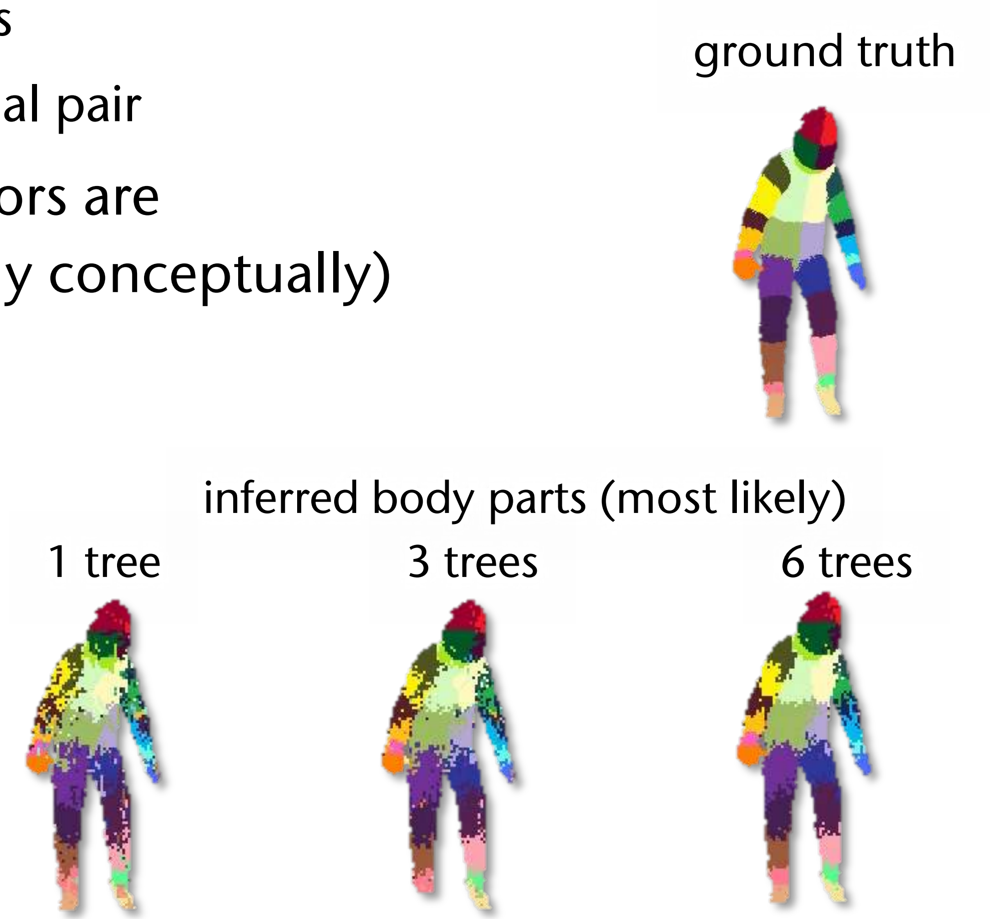
- Toy example: distinguish lower (L) and upper (U) parts of the body
- Note: each node only needs to store Δ and θ !
- For every pixel $\mathbf{x}$ in the depth image,
  we traverse the DT:



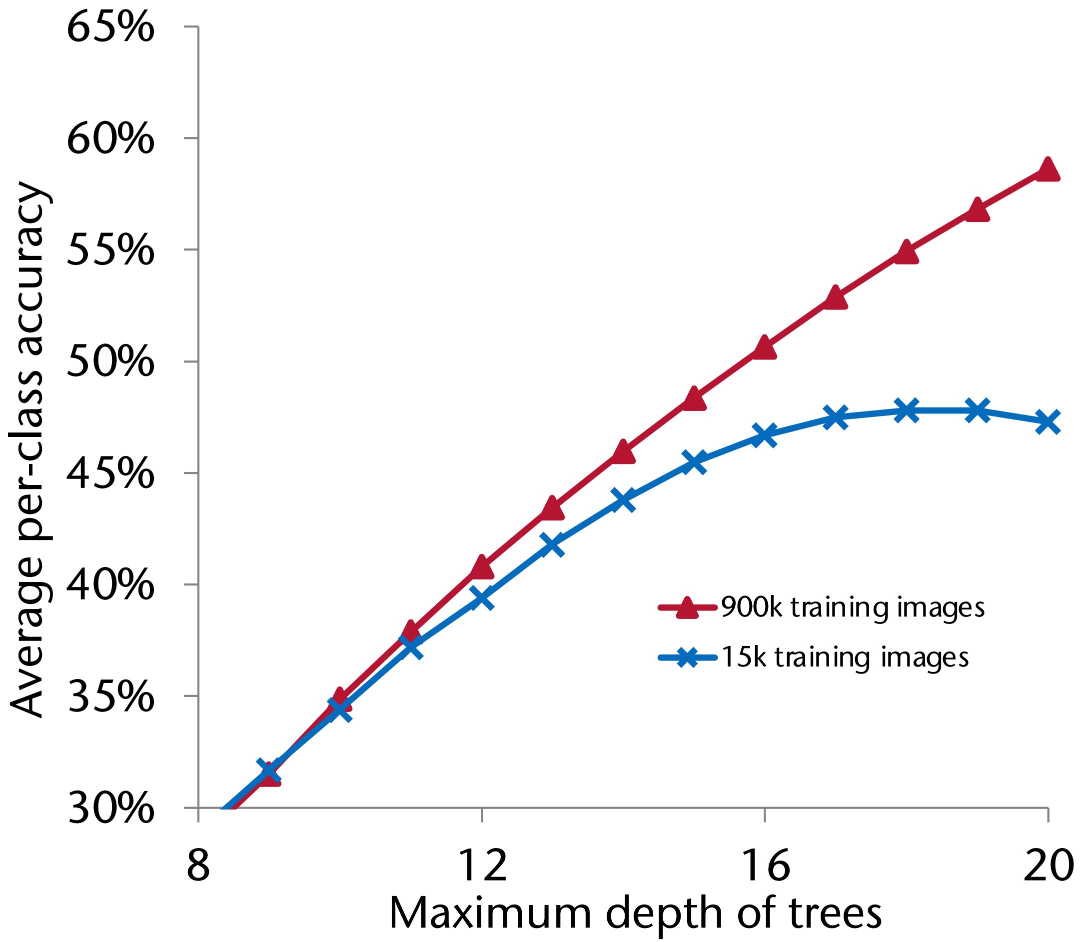
Training a Random Forest

- Train *ntree* many trees, for each one introduce lots of randomization:
  - Random subset of pixels of the training images (~ 2000)
  - At each node to be trained, choose a random set of *mtry* many Δ values
  - Optimize θ for each Δ, pick optimal pair
- Note: the complete feature vectors are never explicitly constructed (only conceptually)

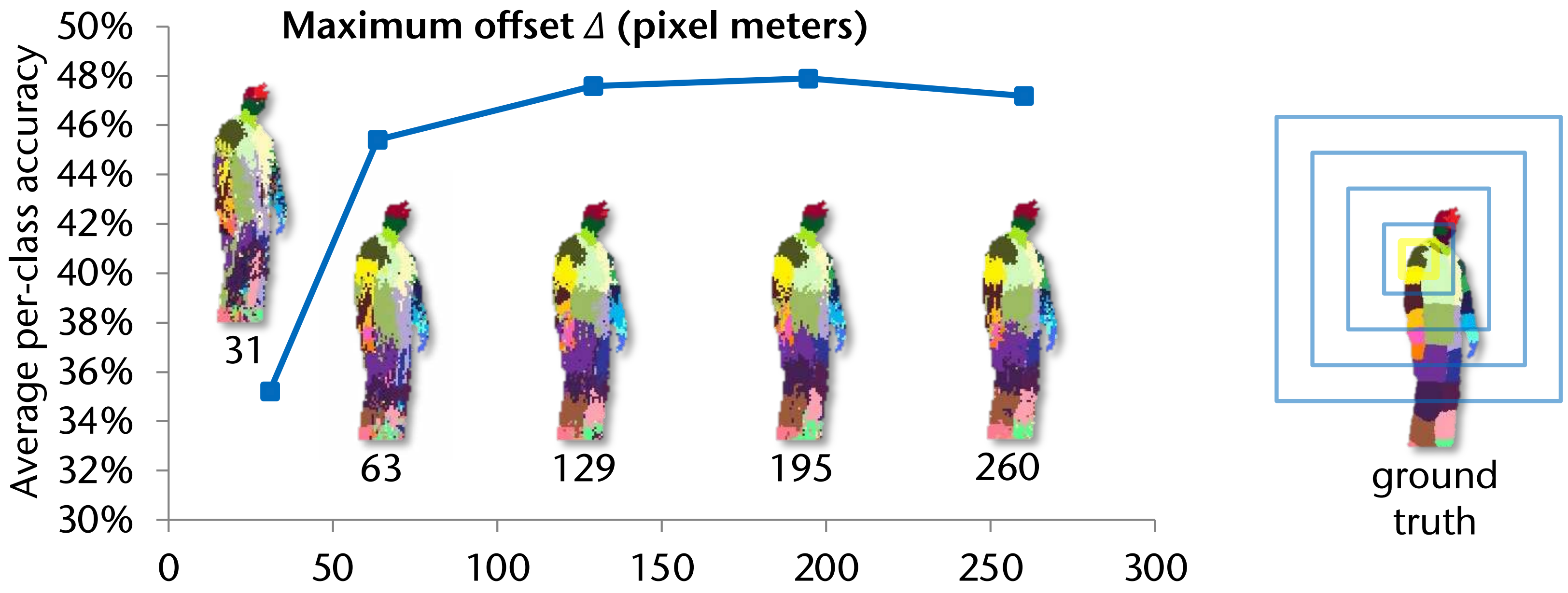


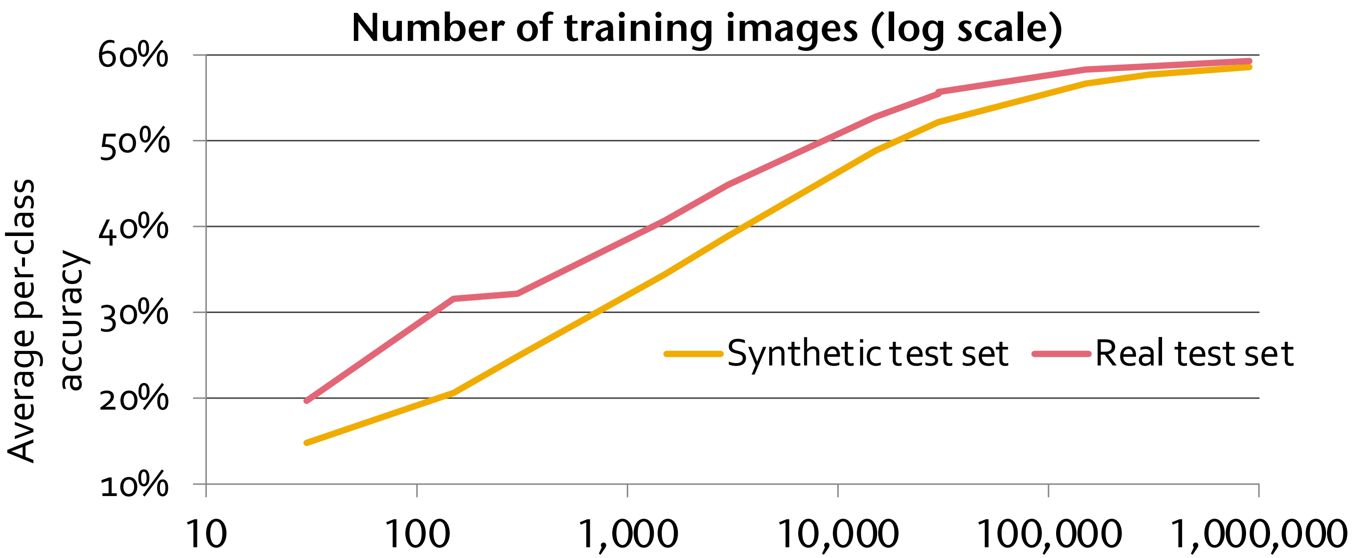


- Depth of trees: check whether it is really best to grow all DTs in the RF to their maximum depth



More Parameters

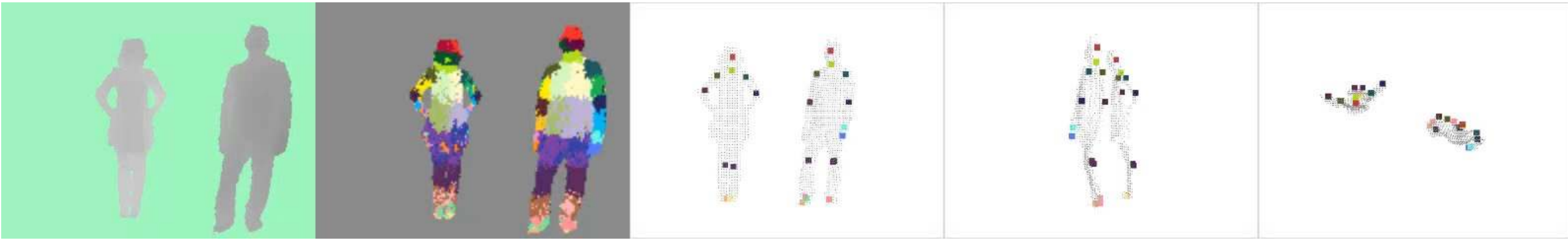


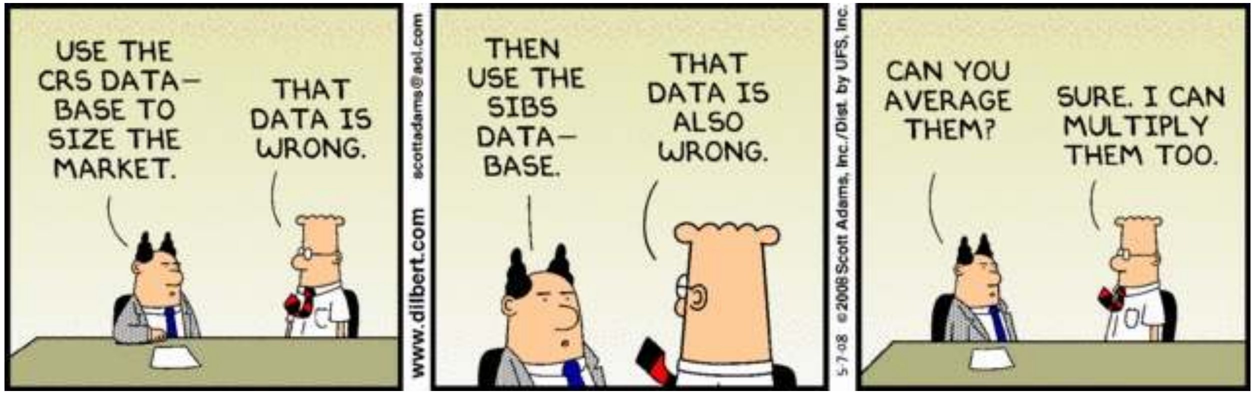


Video

Input depth image (bg removed)

Inferred body parts posterior




USE THE CRS DATA-BASE TO SIZE THE MARKET.
THAT DATA IS WRONG.
www.dilbert.com scottadams@aol.com
THEN USE THE SIBS DATA-BASE.
THAT DATA IS ALSO WRONG.
5-7-08 ©2008 Scott Adams, Inc./Dist. by UFS, Inc.
CAN YOU AVERAGE THEM?
SURE. I CAN MULTIPLY THEM TOO.


OUR CONSULTANT HAS BEEN MINING DATA ALL DAY.
THE RESULTS ARE QUITE SHOCKING.
www.dilbert.com scottadams@aol.com
ACCORDING TO THE DATA, SALES ARE ALWAYS HIGHEST WHEN I DO THIS...
1/5/00 © 1999 United Feature Syndicate, Inc.


SOMEONE SENT ME ANOTHER ANONYMOUS EMAIL WITH A LINK TO AN ARTICLE ABOUT THE WORLD'S WORST BOSSES.
Dilbert.com DilbertCartoonist@gmail.com
I GET ONE OF THOSE EMAILS EVERY TIME I LEAVE YOUR CUBICLE. DID YOU THINK I WOULDN'T NOTICE THE CORRELATION?
11-28-11 ©2011 Scott Adams, Inc.
CORRELATION DOES NOT IMPLY CAUSATION.