Statistical Machine Learning
Hilary Term 2020

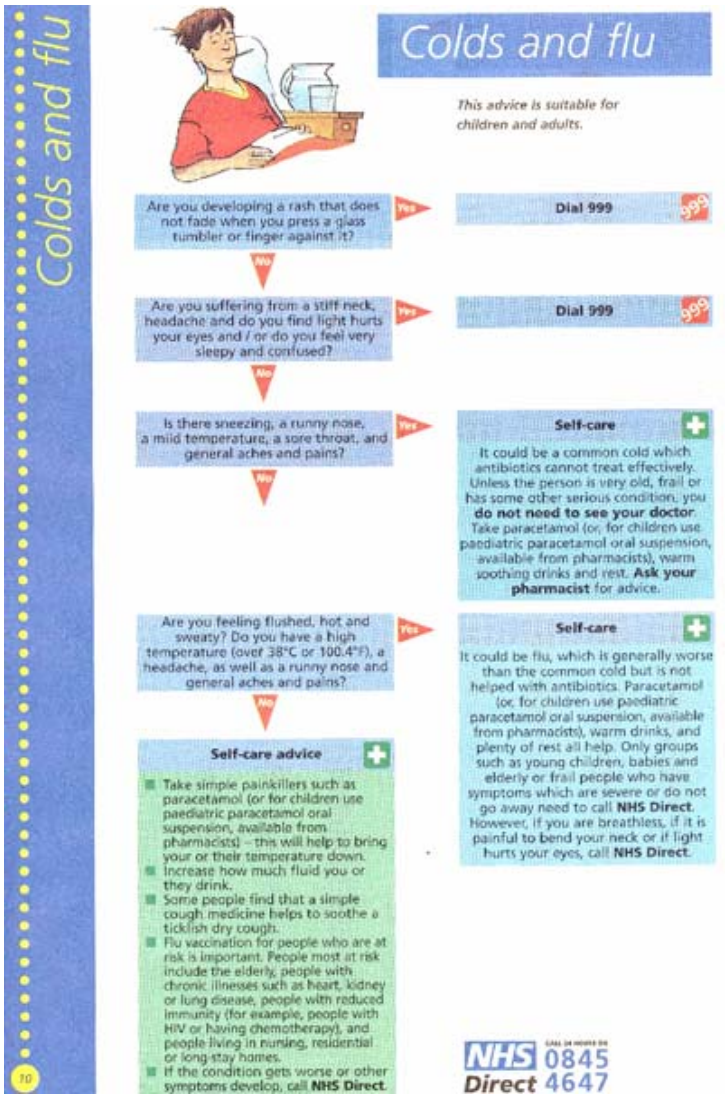
Tom Rainforth
Department of Statistics
University of Oxford

Slide credits and other course material can be found at:
http://www.stats.ox.ac.uk/~palamara/SML20.html

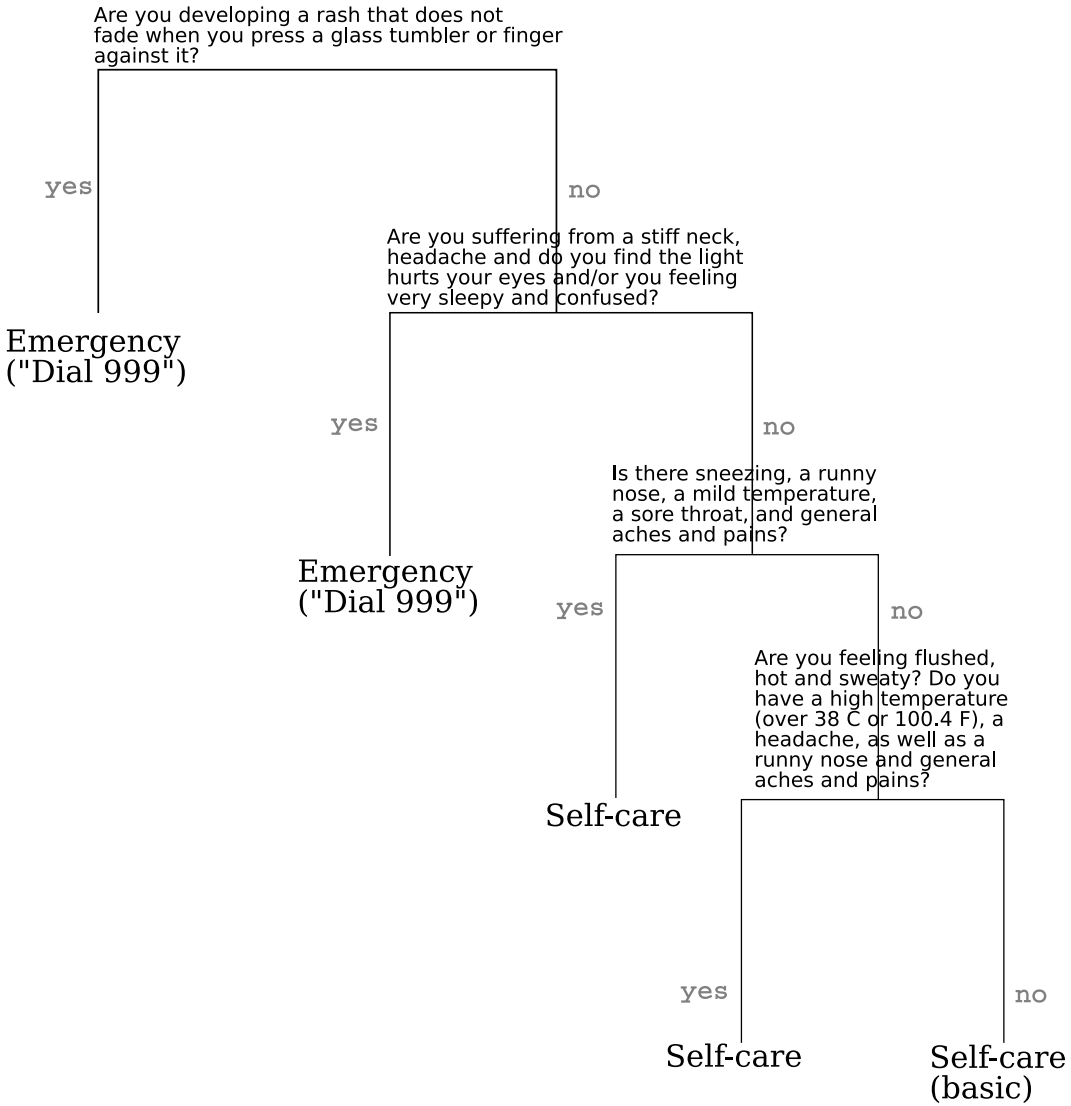
March 5, 2020

Decision Trees

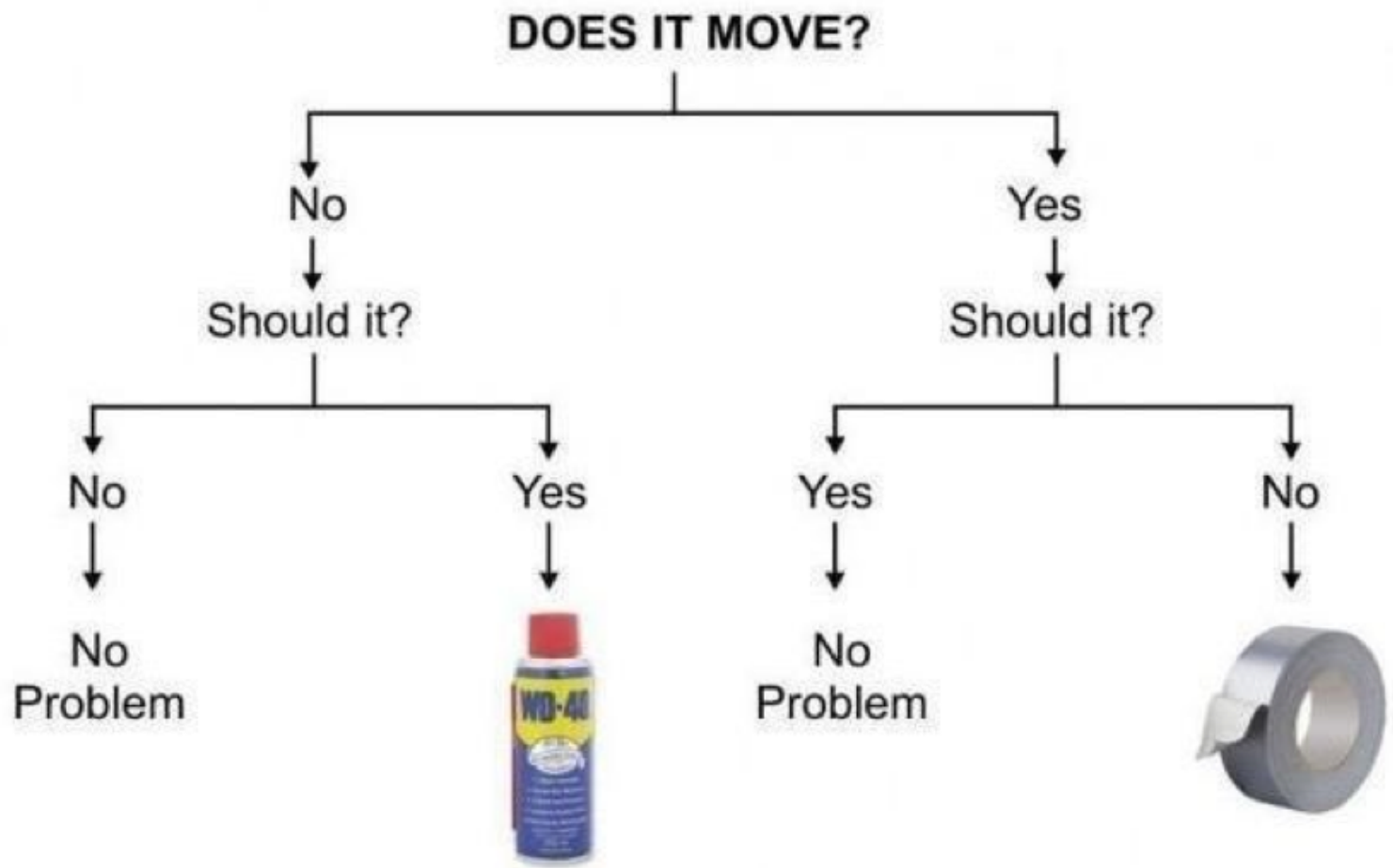
Many decisions are tree-structured



Many decisions are tree-structured



Many decisions are tree-structured



https://www.pinterest.co.uk/pin/281334307952272868/?lp=true

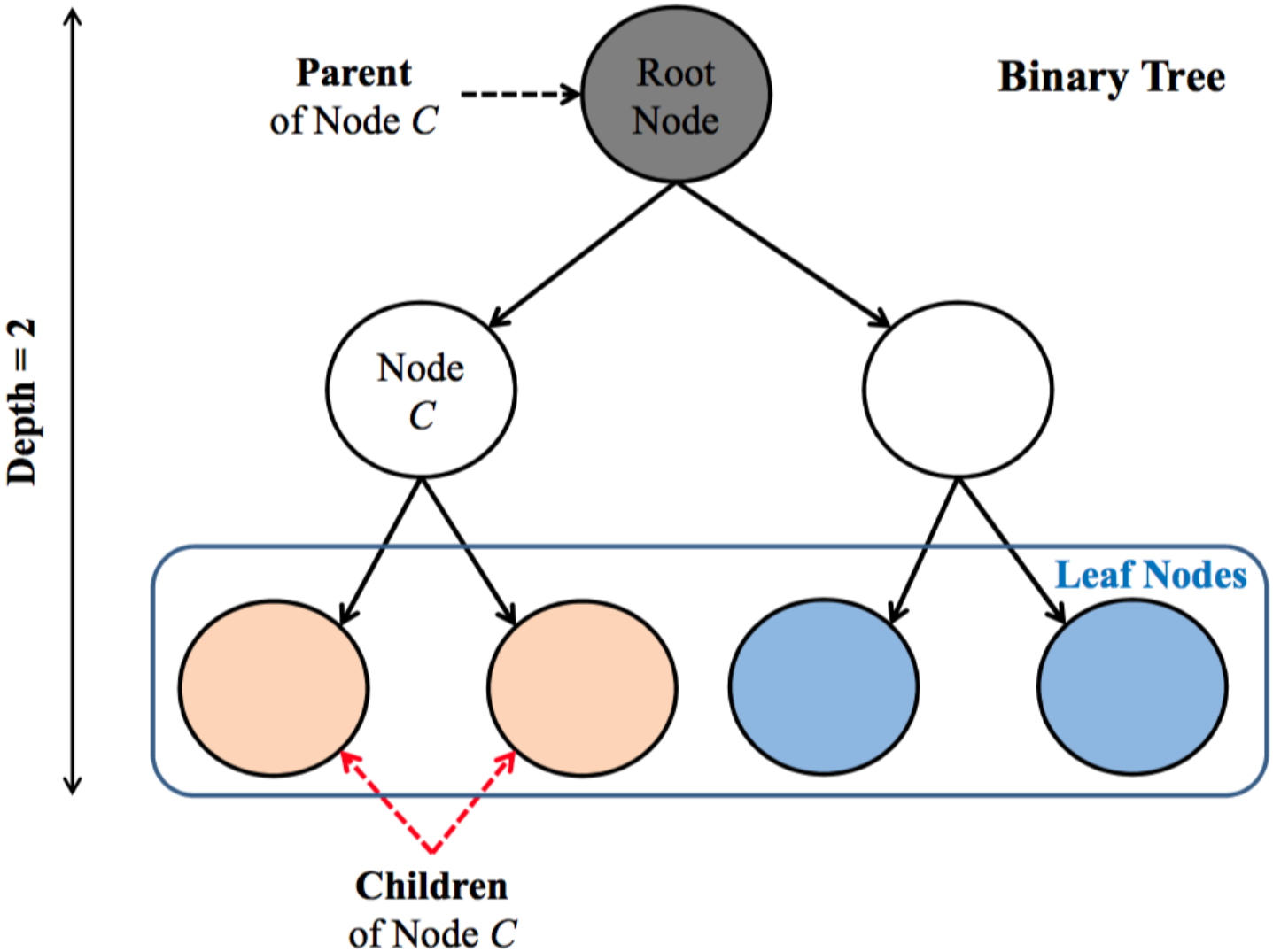
High Level Idea

- Decision trees are predictive models that apply a **hierarchical** series of queries to the input until a “leaf node” is reached
- Predictions are then made by **local leaf models**, e.g. a class label, probabilities over classes, a linear regression model, etc
- Often used as means of constructing human–interpretable predictive models
- Also basis for more powerful machine learning methods like random forests and boosting schemes

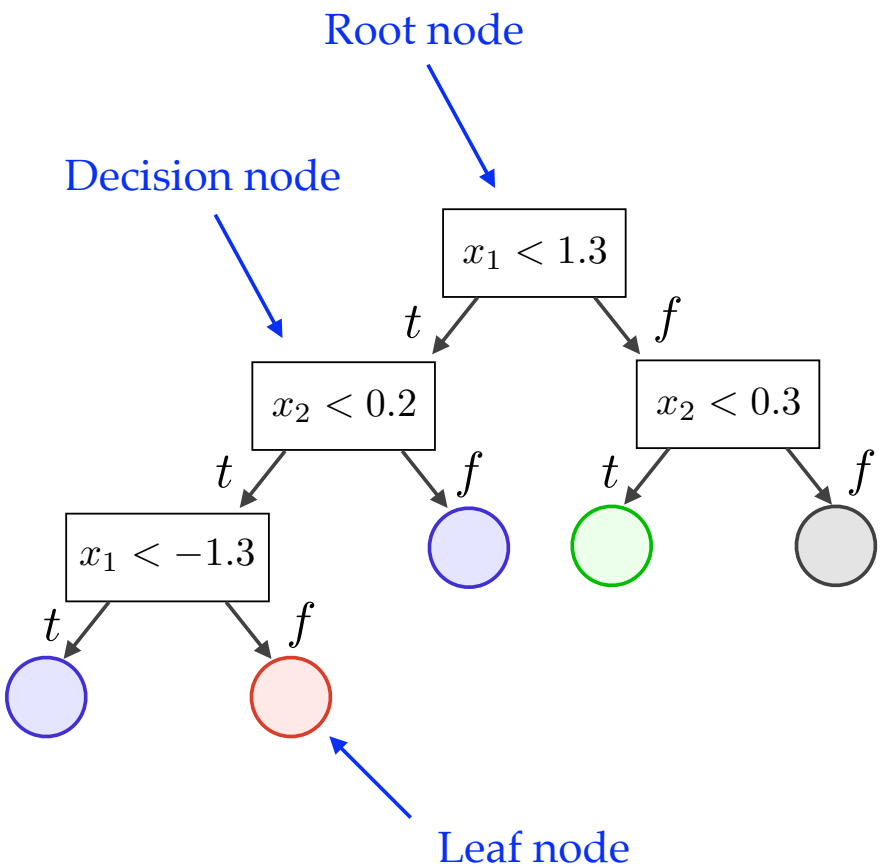
Terminology

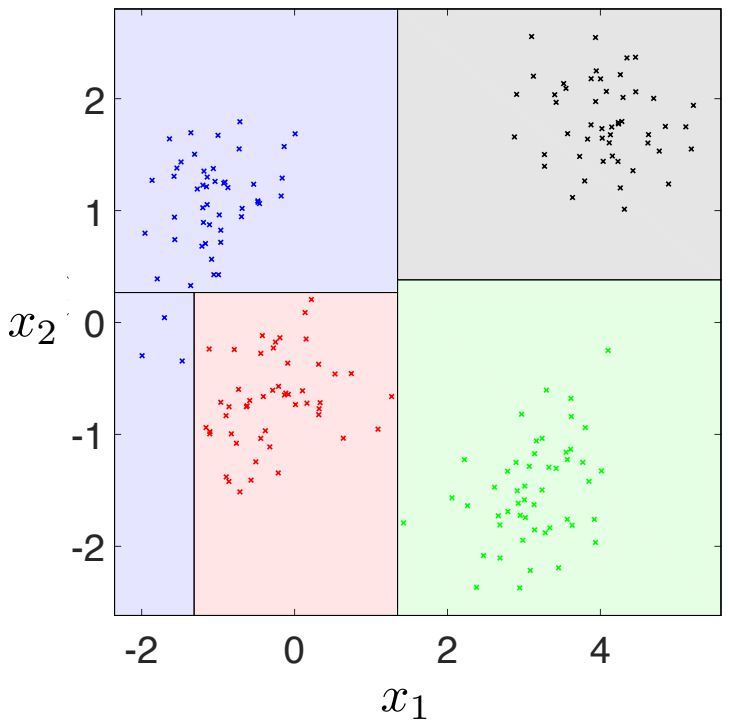
- The **parent** of a node c is the immediate predecessor node.
- The **children** of a node c are the immediate successors of c, equivalently nodes which have c as a parent.
- **Branches** are the edges/arrows connecting the nodes.
- The **root** node is the top node of the tree; the only node without parents.
- The **leaf** nodes are nodes which do not have children.
- **Subtrees** are formed of a node and all is descendants
- A $C-$**ary** tree is a tree where each node (except for leaf nodes) has C children. Usually work with binary trees $(C = 2)$.
- The **depth** of a tree is the maximal length of a path from the root node to a leaf node (also sometimes refer to depth of an individual node).
- **Stumps** (or decision stumps) are trees with just the root node and two leaf nodes.

Terminology

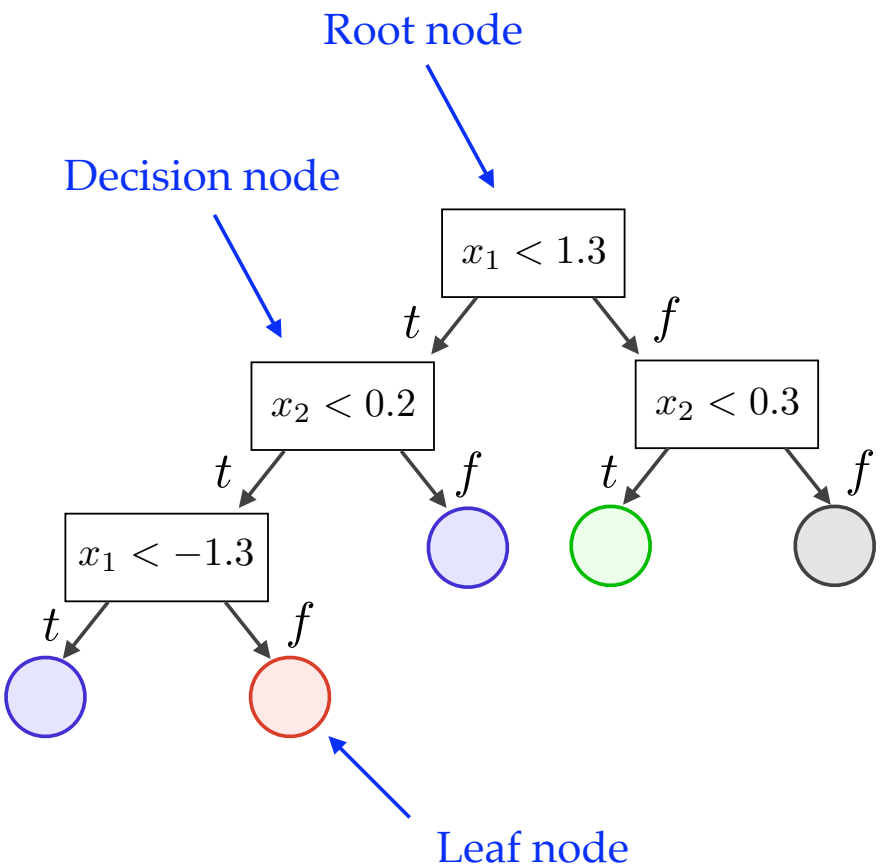


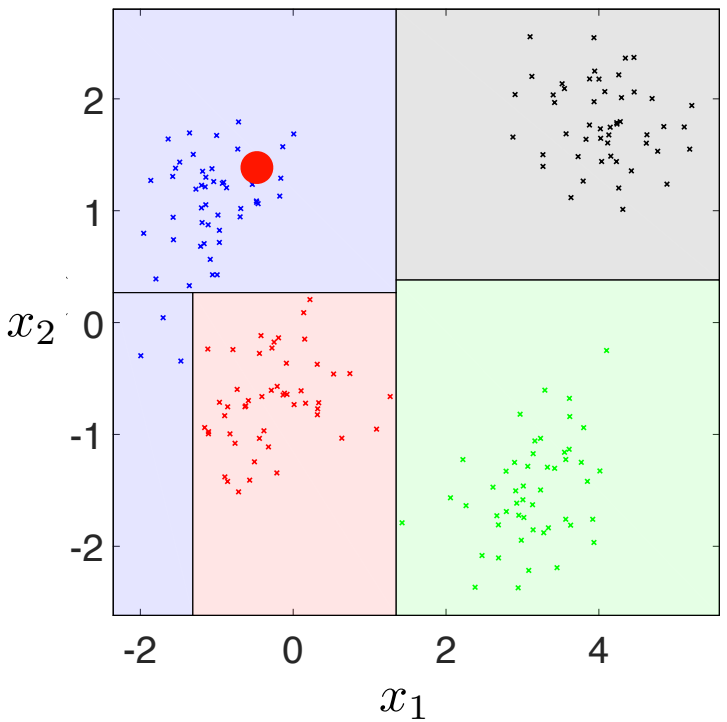
Predicting from a Decision Tree



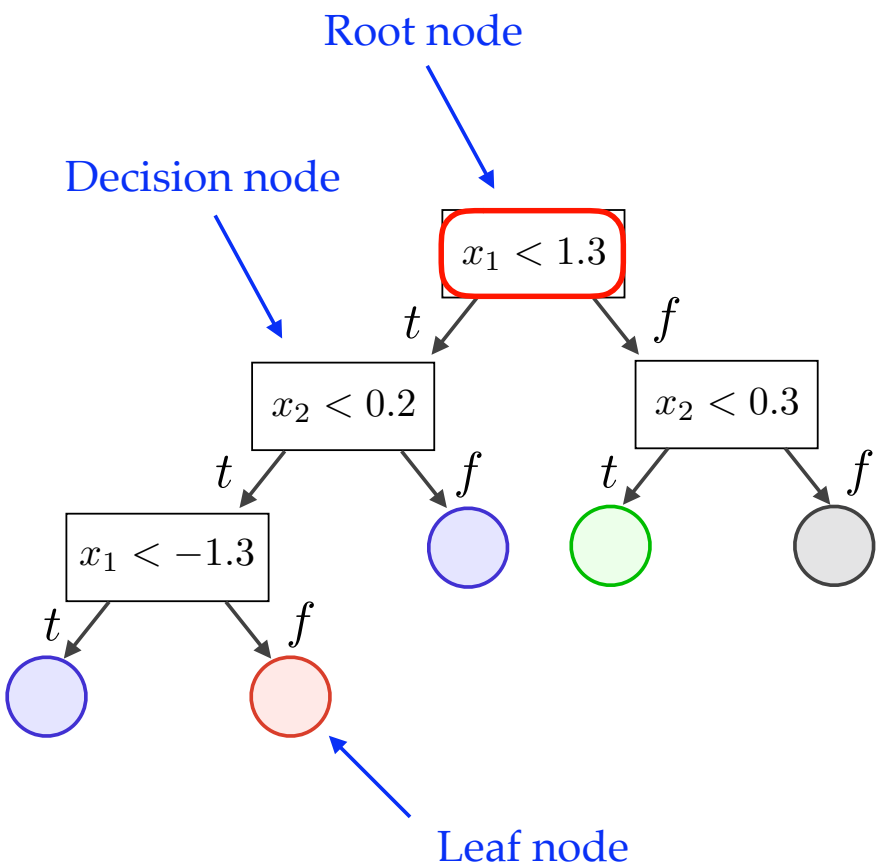


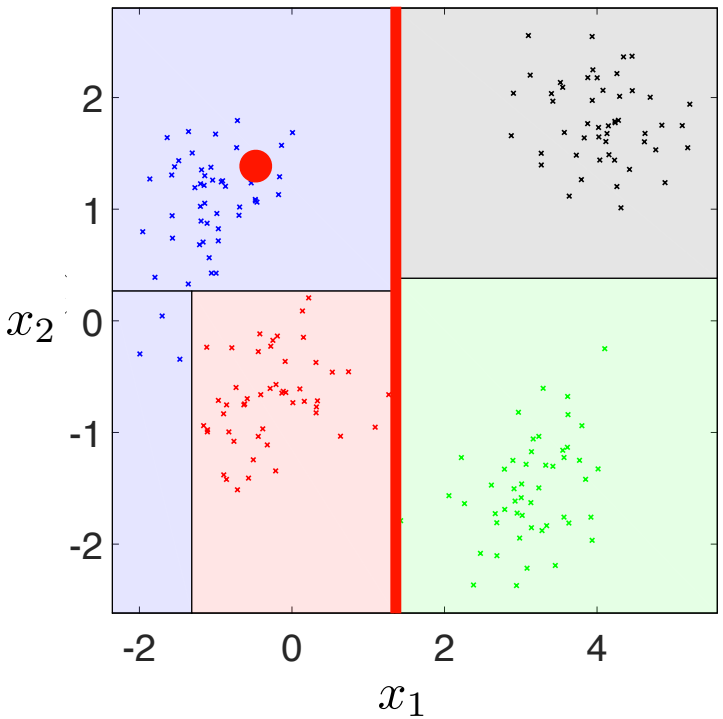
Predicting from a Decision Tree



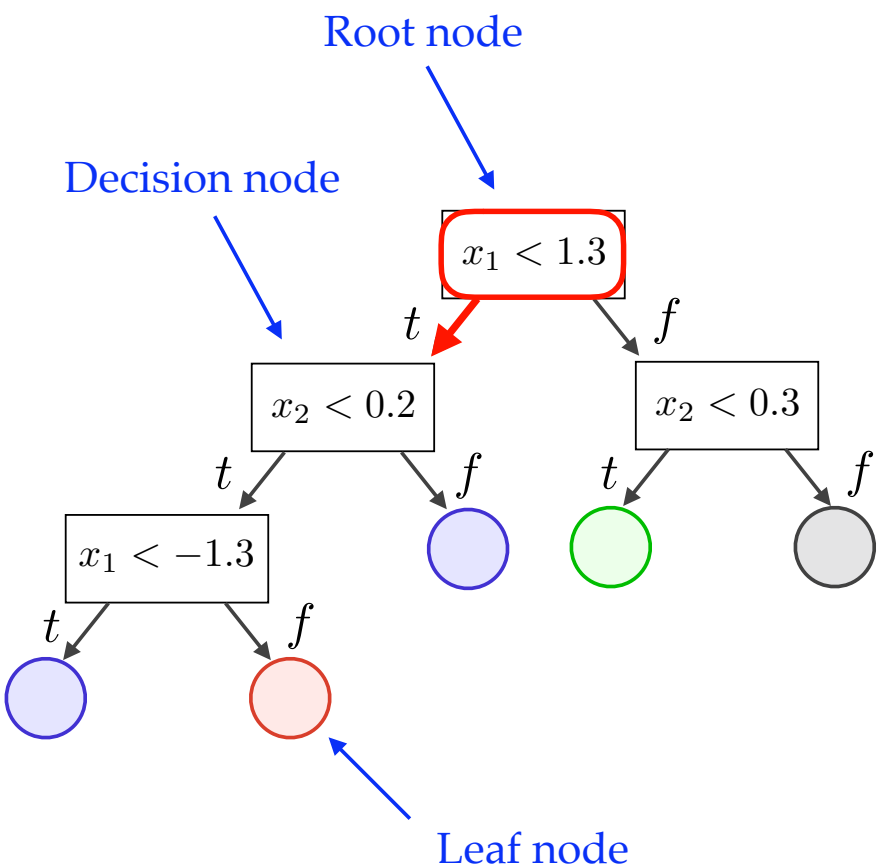


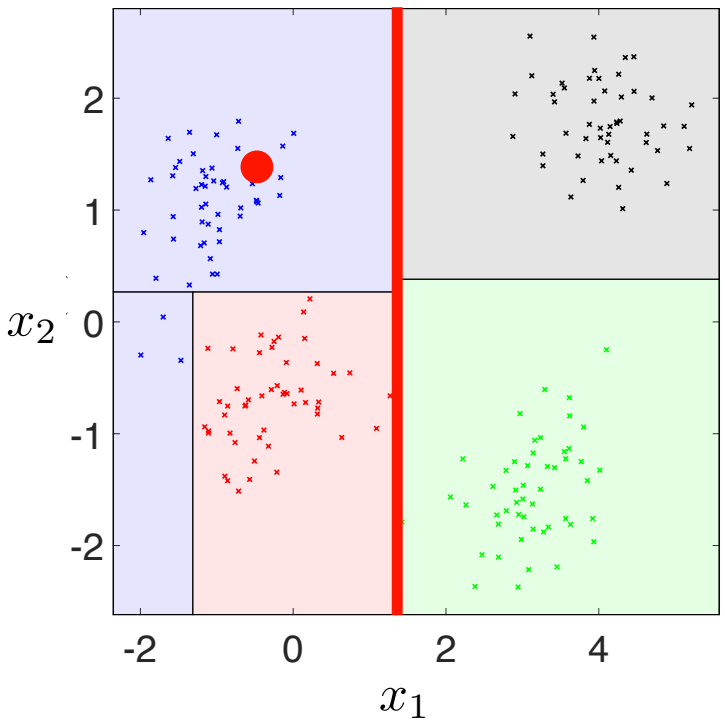
Predicting from a Decision Tree



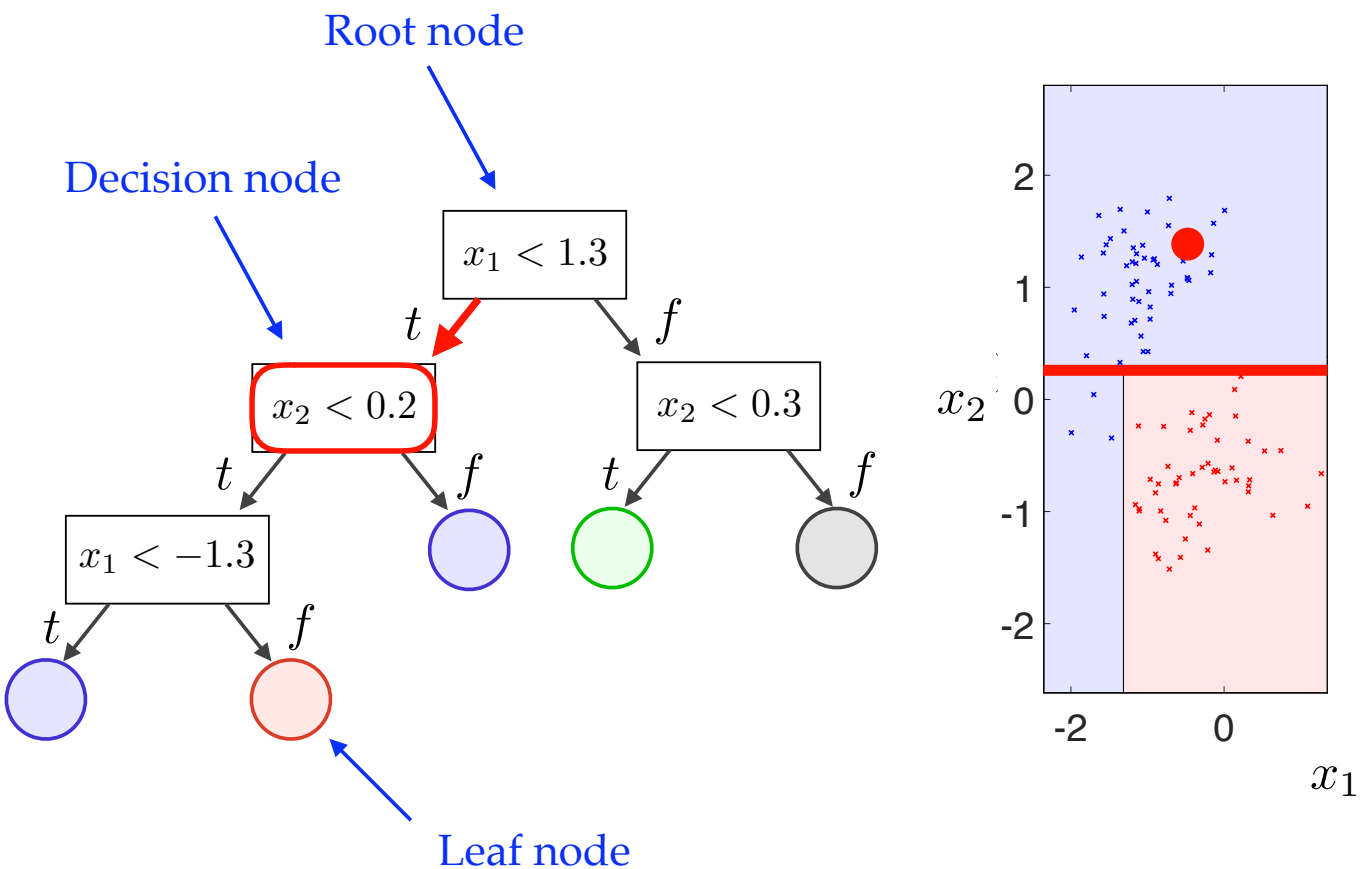


Predicting from a Decision Tree

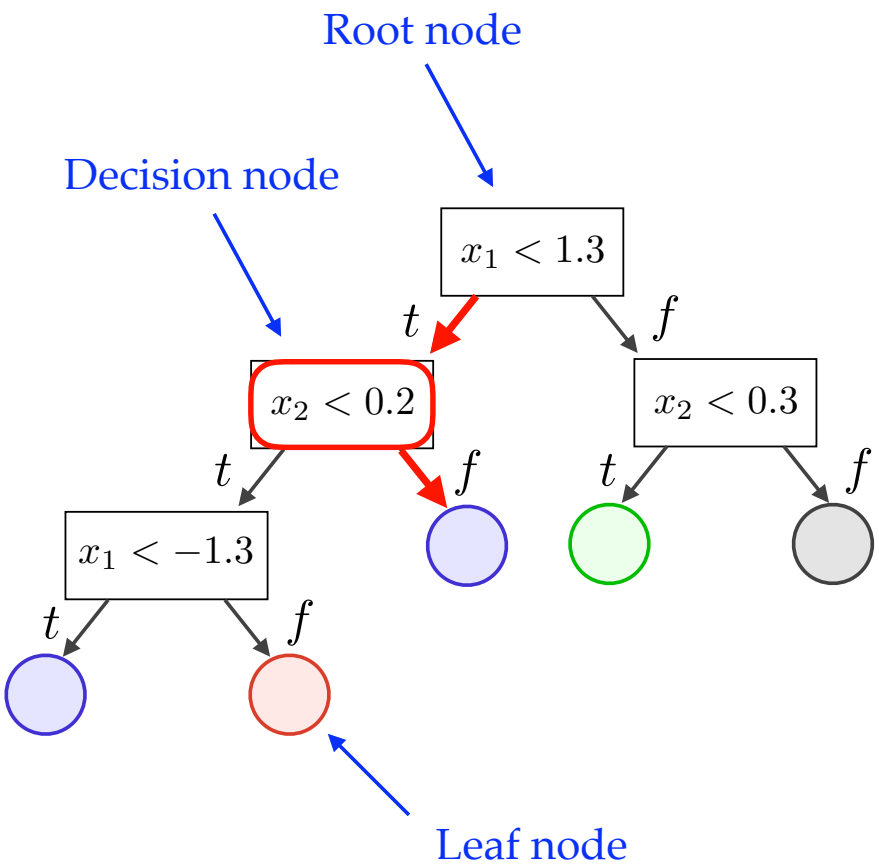


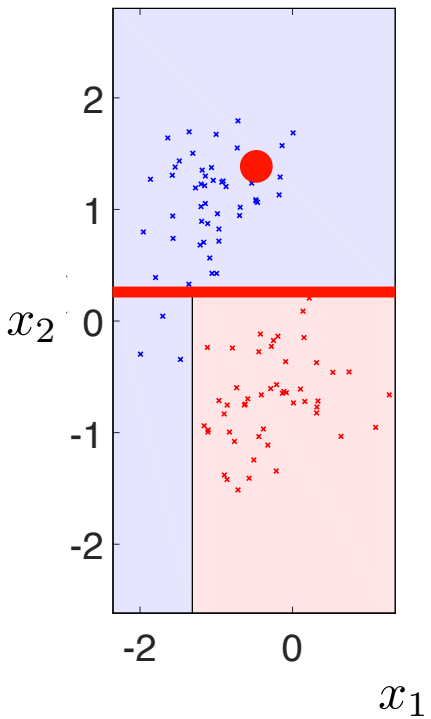


Predicting from a Decision Tree

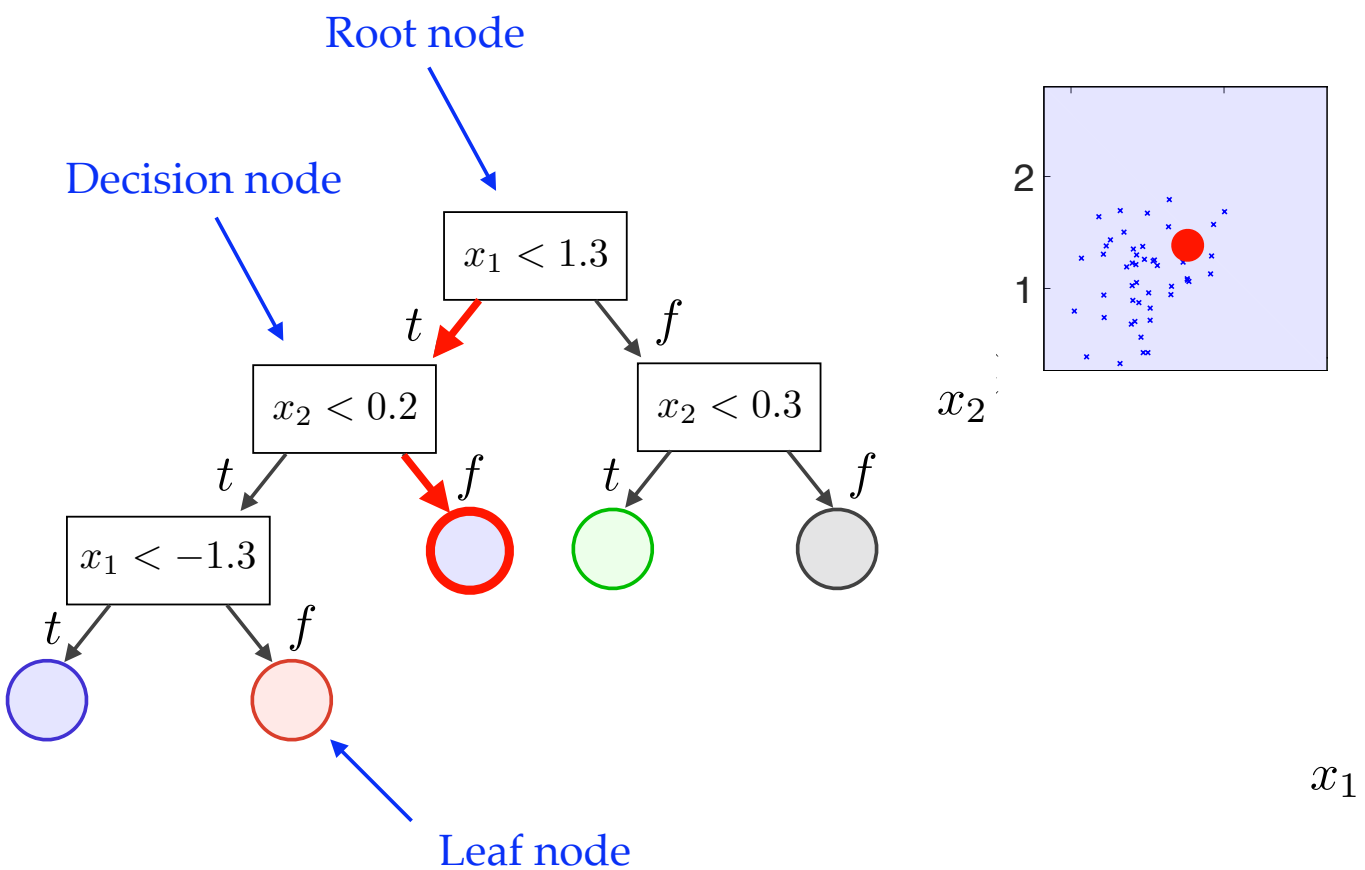


Predicting from a Decision Tree





Predicting from a Decision Tree



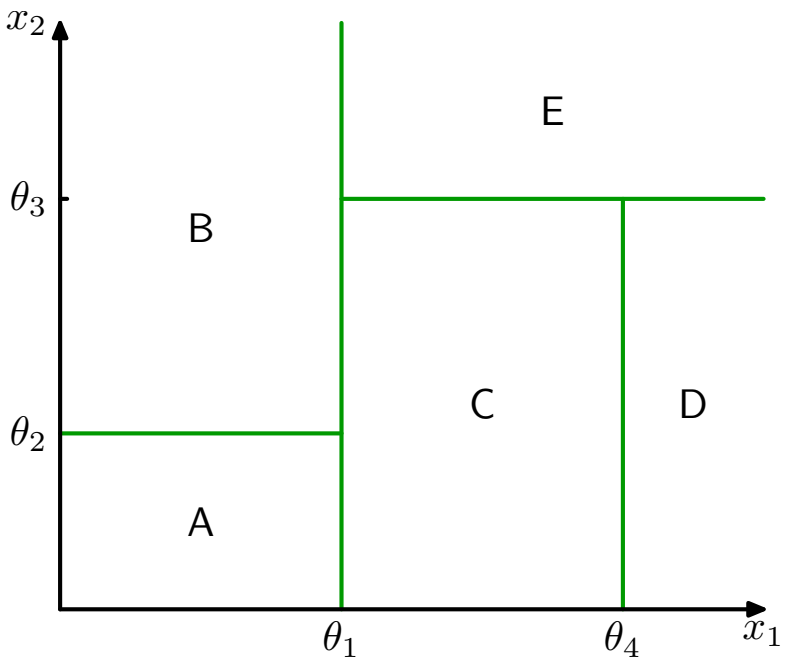
A tree partitions the feature space

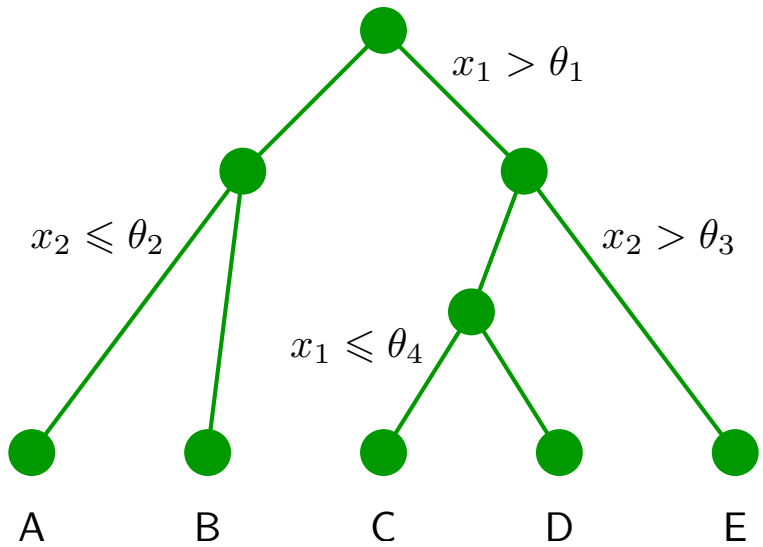
- A Decision Tree is a hierarchically organized structure, with each node splitting the data space into pieces based on value of a feature.
- Equivalent to a partition of overall space $\mathbb{R}^p$ into L disjoint feature regions $\{\mathcal{R}_1^{\text{leaf}}, \ldots, \mathcal{R}_L^{\text{leaf}}\}$, where each $\mathcal{R}_j^{\text{leaf}} \subset \mathbb{R}^p$ and each corresponds to a leaf
- Each leaf has a local decision or predictive model (often just a particular class or value) that is applied for all $x \in \mathcal{R}_j^{\text{leaf}}$.
- Note internal notes also correspond to regions of the space $\mathcal{R}_j$, with the region of a parent node being the union of the regions of its child nodes

$$\mathcal{R}_j = \bigcup_{c \in \text{children}(j)} \mathcal{R}_c$$

- The union of all nodes at a particular depth also corresponds to the full space

A tree partitions the feature space

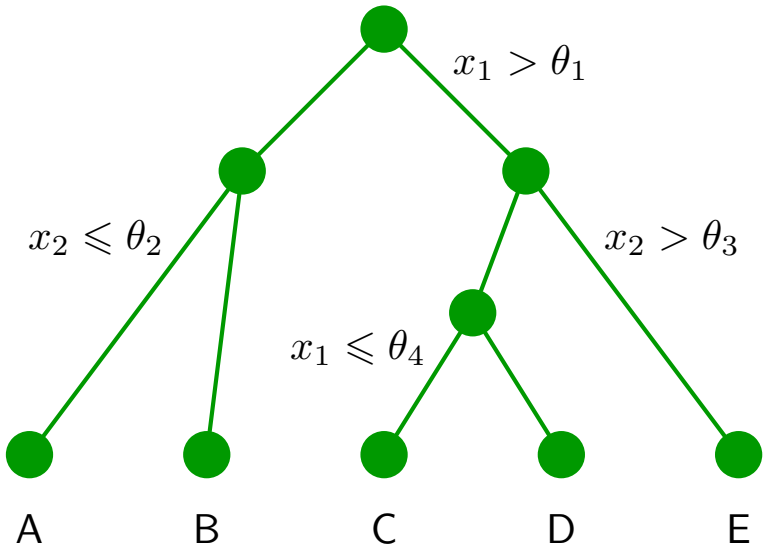




Learning a tree model

Two things to learn:

1. The structure of the tree and associated splitting mechanisms, e.g. $x_1 \leq \theta_4$
2. The leaf predictive models $(A, B, \ldots)$.



Key questions:

- What makes a good tree?
- How can learn trees from data?

Tree Training – High Level Idea

Algorithm 1 TRAINTREE (Training Data D)

1: **if** any stopping criteria are met **then**
2: **return** LEAFNODE(D)
3: **else**
4: Search over possible splits and chose the best one
5: Separate D into newly created child nodes, giving $D_1, \ldots, D_C$
6: **for** $c = 1 : C$ **do**
7: child-sub-tree$_c \leftarrow$ TRAINTREE(D_c)
8: **end for**
9: **return** Subtree with current node as root, chosen split, and child sub-trees
10: **end if**

- For classification the main stopping criteria is that we stop splitting if a node is pure, i.e. all of the samples it contains are of the same class. Other criteria are discussed later
- LEAFNODE(D) produces a local leaf predictor from local data D

Leaf Predictors

Let's start by thinking how to setup the leaf predictors in a classification tree:

- Assume that we are given a dataset $D = (x_1, y_1), \ldots, (x_n, y_n)$, where $x_i \in \mathbb{R}, y_i \in Y = \{1, \ldots, m\}$,
- For a given structure and splits, we can minimize the empirical risk for the overall tree by separately minimizing the empirical risk for each leaf
- The most common predictive model for each leaf is use the empirical probability of each class (i.e. relative frequency in the training data).
- The estimated probability of each class k in region $\mathcal{R}_j$ is now simply:

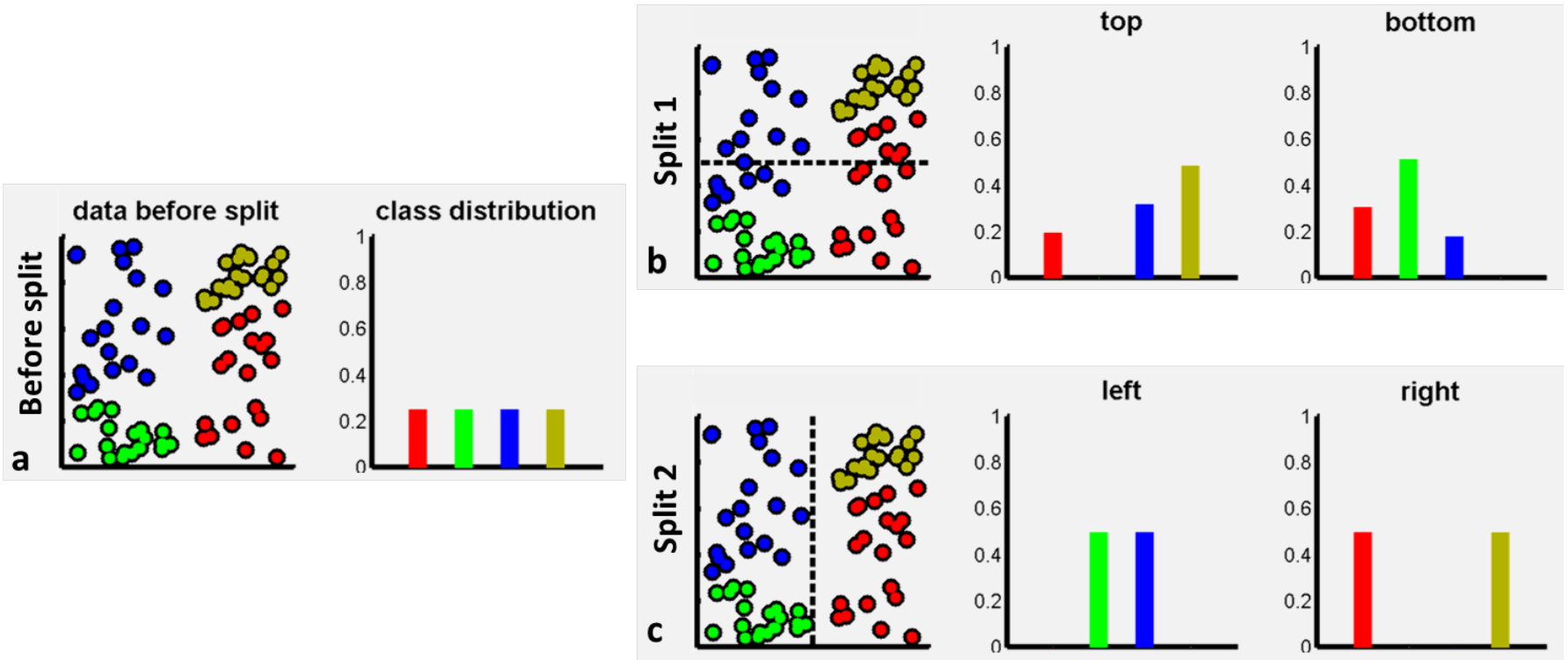
$$\hat{p}_{j,k} = \frac{\sum_i \mathbb{I}(y_i = k) \cdot \mathbb{I}(x_i \in \mathcal{R}_j)}{\sum_i \mathbb{I}(x_i \in \mathcal{R}_j)}$$

- Equivalently, we can express this in terms of the data $D_j \subseteq D$ at node j

$$\hat{p}_{j,k} = \frac{1}{n_j} \sum_{i \in D_j} \mathbb{I}(y_i = k)$$

where n_j is the number of training samples at the node

What makes a good split?



- Good splits will produce children with better predictive power
- Equivalently, good splits will produce purer nodes (i.e. nodes with concentrated distribution on their classes)

Adapted from: Criminisi et al, Decision Forests, 2012

Entropy of a Node

- We can evaluate the predictive power of a node through the **entropy** over classes

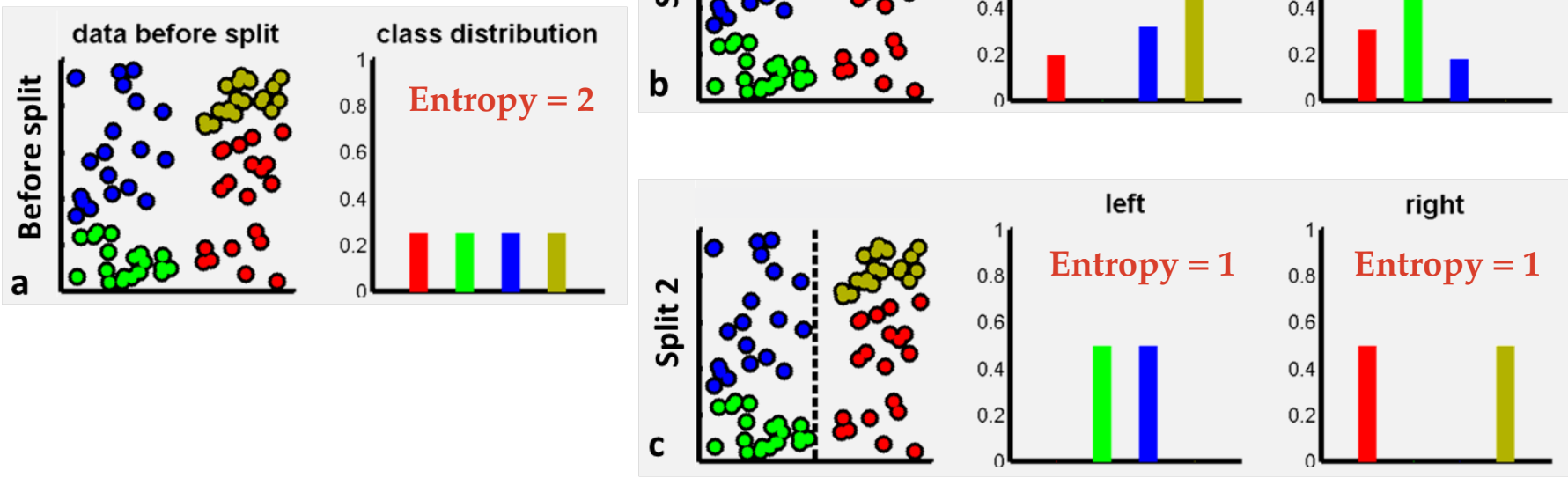
$$H[Y] = -\sum_{k=1}^{K} P(Y = k) \log_2 P(Y = k)$$

- Given data D_j at node j, we can evaluate this entropy based on the empirical entropy of the training data at that node

$$H_j = -\sum_{k=1}^{K} \hat{p}_{j,k} \log_2 \hat{p}_{j,k}$$

- This is effectively the entropy of the predictive model if this node was turned into a leaf
- Purer nodes will have lower entropy

What makes a good split?



Adapted from: Criminisi et al, Decision Forests, 2012

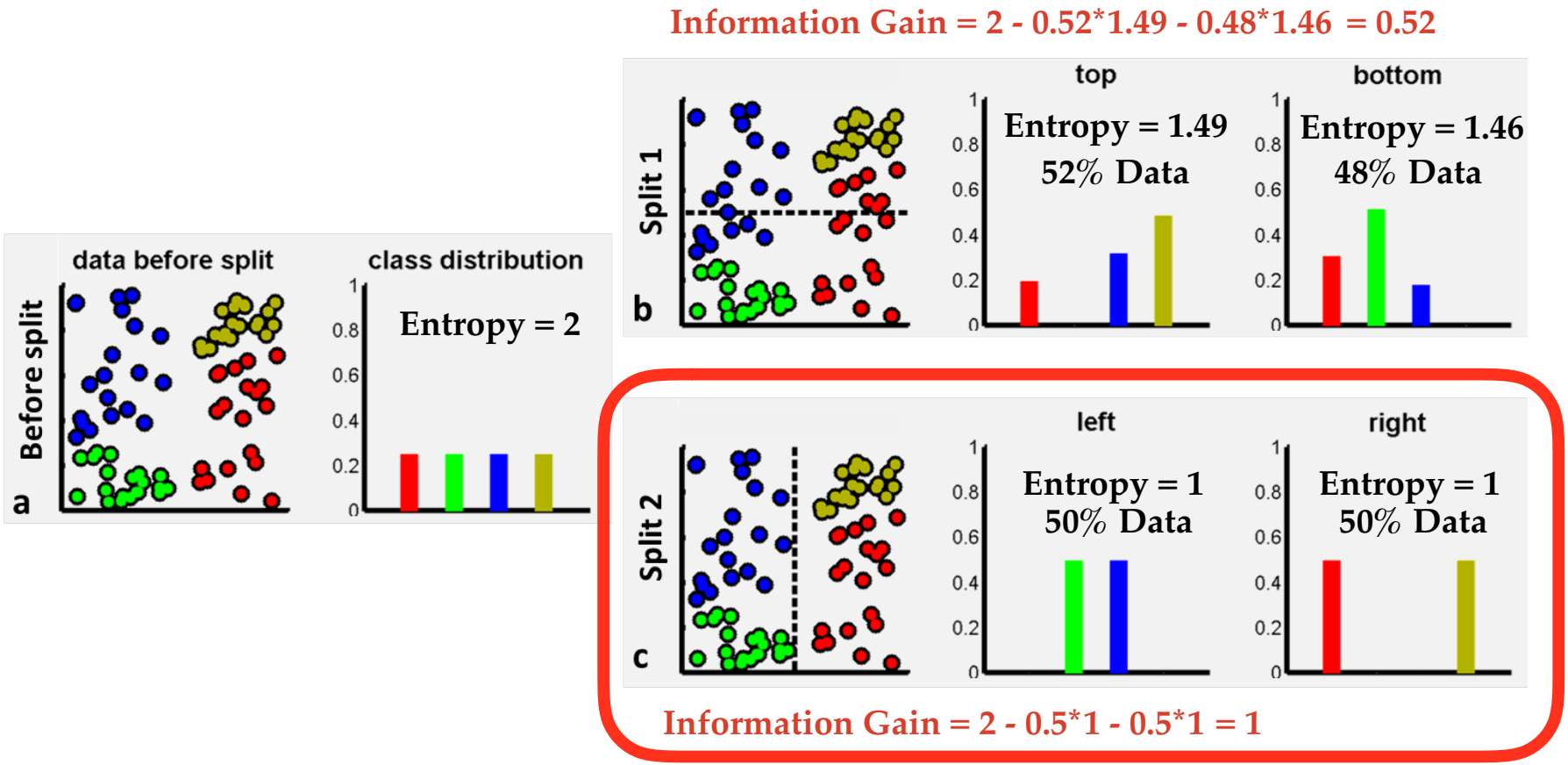
Information Gain

- A **split criterion** is a way to measure how good a hypothetical split will be
- We can use it to decide between possible splits when training a tree
- One common split criterion is to use node entropy to calculate the **information gain** of a split
- The information gain of a split is the reduction in entropy in the predictive distribution from adding the split, where we assume that new inputs take each branch in proportion to the number of training samples that follow the branch
- Using n_j to represent the number of training samples in node j, we have

$$\text{InfoGain} = H_j - \sum_{c \in \text{Children}(j)} \frac{n_c}{n_j} H_c$$

  where we note that $n_j = \sum_{c \in \text{Children}(j)} n_c$
- Gaining information reduces uncertainty in our predictions

What makes a good split?



Adapted from: Criminisi et al, Decision Forests, 2012

Other Split Criteria

- Information gain is not the only commonly used split criterion
- Other approaches use a different measure of uncertainty to entropy, before looking in the gain in that uncertainty metric in an analogous manner to information gain
- In general, we thus have that the gain for a hypothetical split node of j under uncertainty metric $B(\cdot)$ is

$$\text{Gain} = B(D_j) - \frac{1}{n_j} \sum_{c \in \text{Children}(j)} n_c B(D_c)$$

- Note that when we are choosing a split, $B(D_j)$ is fixed, but different splits will give different partitions in the children and thus different gains
  - Any split that partitions the data in same way has the same gain
- $B(D_j)$ effectively represents how bad node j is: splits with high gains produce children with low uncertainties

Uncertainty Metrics

- Possible choices for $B(\cdot)$ include (remember $\hat{p}_{j,k}$ is the empirical probability of class of k for node j)
  - **Entropy** (measured in bits):

$$B(D_j) = -\sum_{k=1}^{K} \hat{p}_{j,k} \log_2 \hat{p}_{j,k}$$

  - **Gini Impurity** (sometimes called Gini index):

$$B(D_j) = 1 - \sum_{k=1}^{K} \hat{p}_{j,k}^2$$

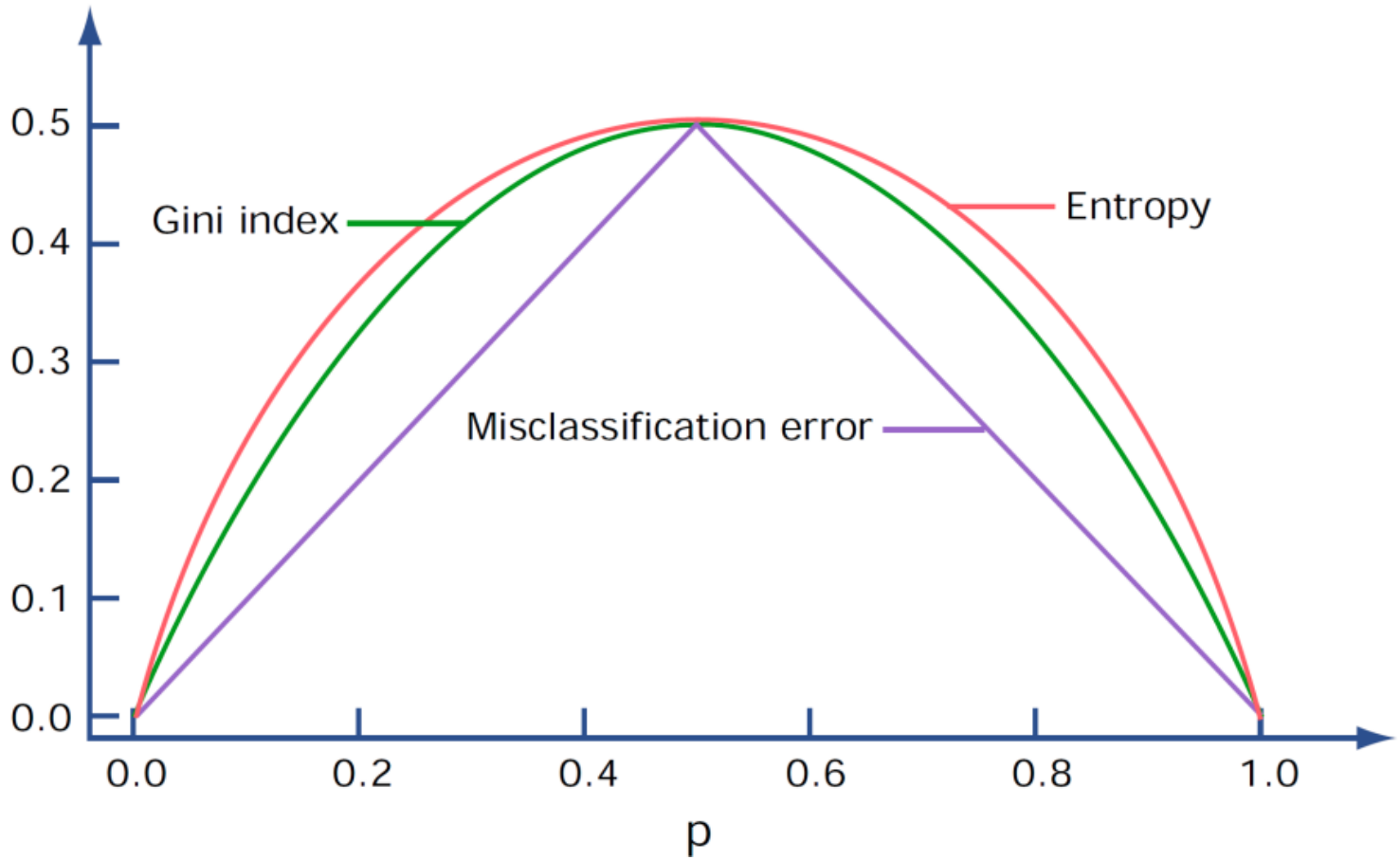
  - **Misclassification error** (rarely used in practice):

$$B(D_j) = 1 - \max_{k \in \{1,\ldots,K\}} \hat{p}_{j,k}$$

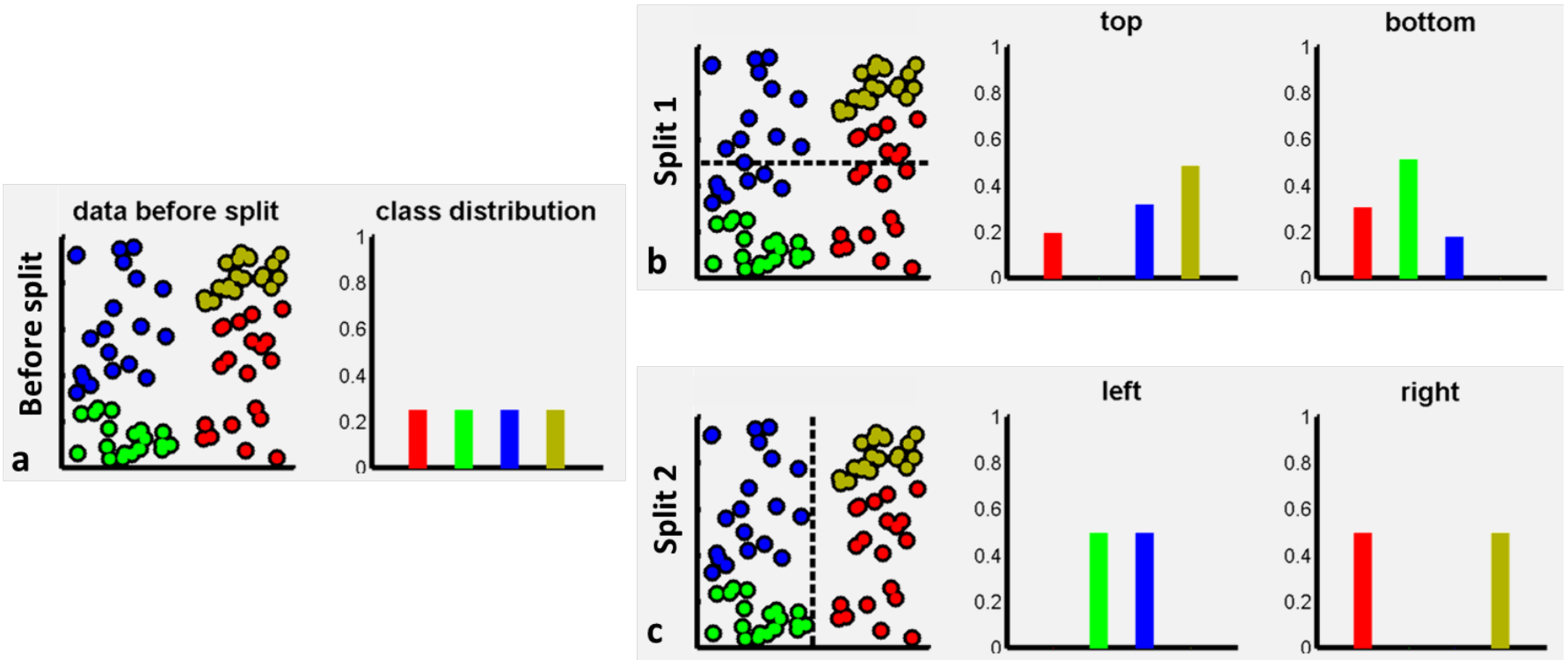
- **C4.5** Tree algorithm: Classification uses entropy to measure uncertainty.
- **CART** (class. and regression tree) algorithm: Classification uses Gini.

Different measures of uncertainty

Different uncertainty metrics for single split of binary classification function as proportion of points p of the first class:



Why misclassification error is bad uncertainty metric

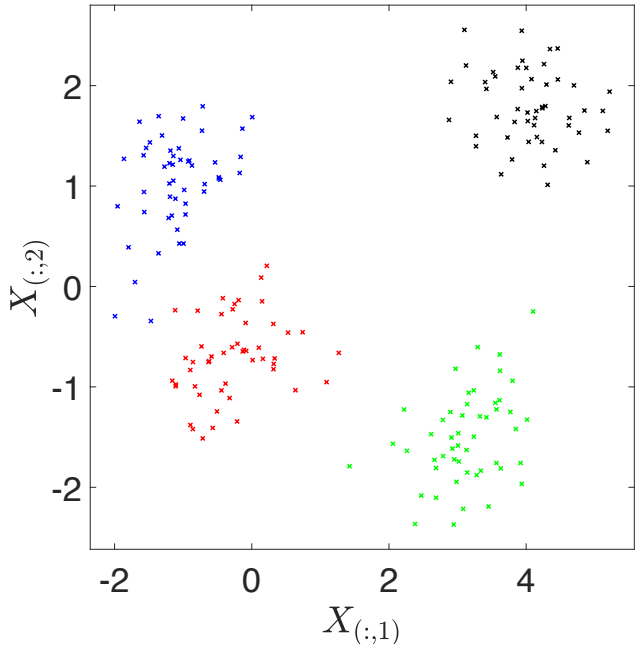


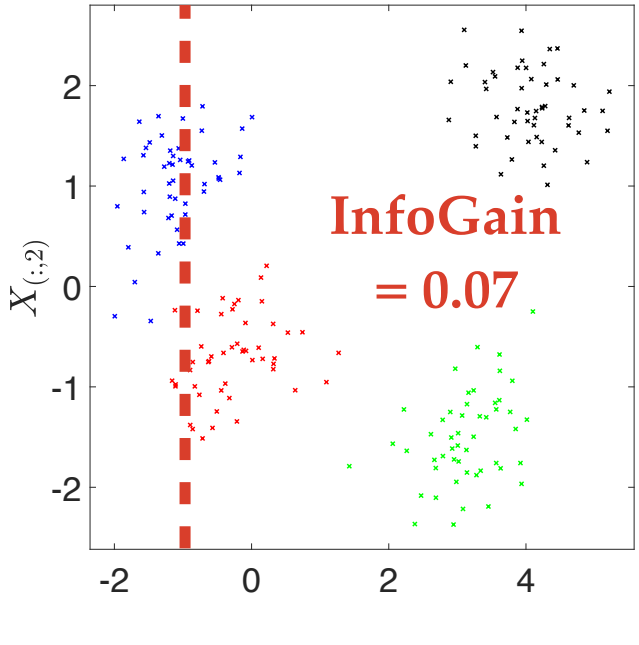
Misclassification error gives the same gain of 0.25 to both these splits

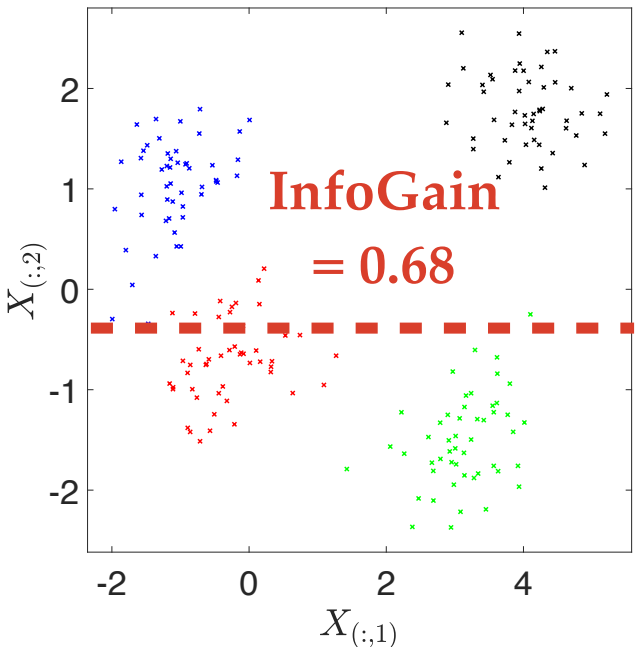
Adapted from: Criminisi et al, Decision Forests, 2012

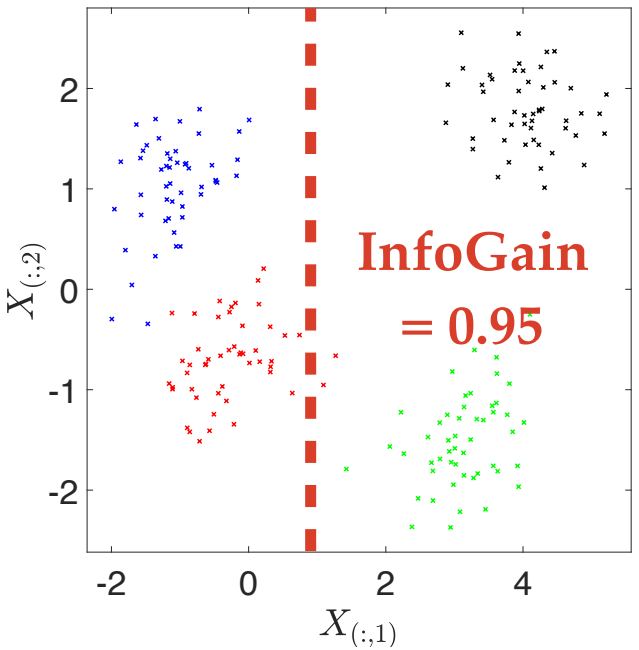
Splitting a Node: Exhaustive Search

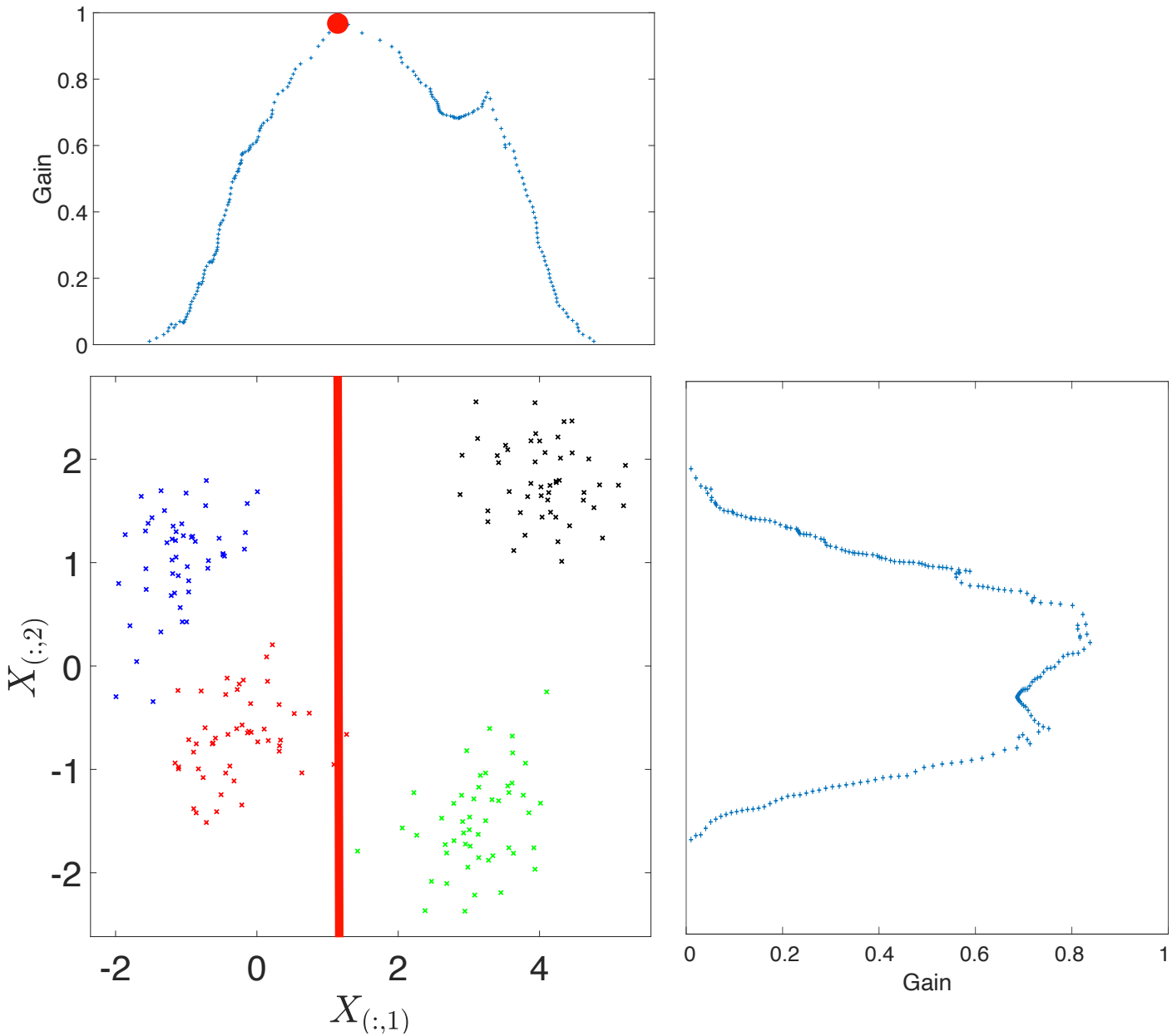
- Though some alternatives exist, most training approaches only split based on a single feature (i.e. our splits are “axis-aligned”)
- The standard approach is to perform exhaustive search over the set of possible splits and pick the one with the highest gain
- For categorical features, we can either create a new child for each possible realization of the variable, or group realizations together to reduce the number of children (e.g. to create a binary split)
- For numerical features, we consider binary splits placed halfway between consecutive feature values. This ensures the number of allowed splits is finite.
- From a predictive perspective, binary splits are always best, but using larger numbers of children can make the tree more user–friendly
  - Note that any tree with multi-way splits can be reformulated into a binary tree by repeatedly splitting on the same variable


2
1
0
-1
-2
-2
0
2
4
$X_{(:,2)}$
$X_{(:,1)}$


InfoGain
= 0.07
2
1
0
-1
-2
-2
0
2
4
$X_{(:,2)}$

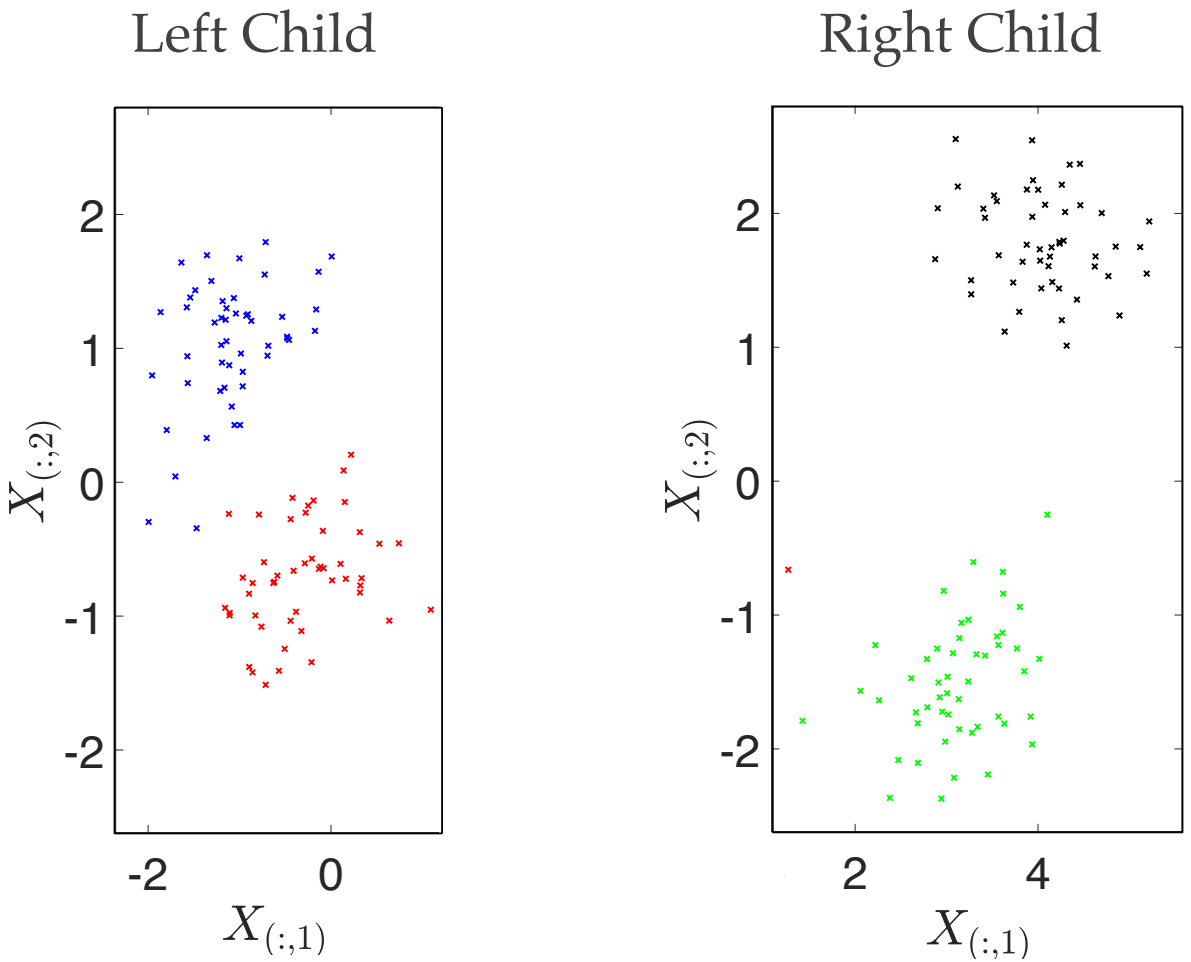

InfoGain
= 0.68
2
1
0
-1
-2
-2
0
2
4
$X_{(:,2)}$
$X_{(:,1)}$


InfoGain
= 0.95
2
1
0
-1
-2
-2
0
2
4
$X_{(:,2)}$
$X_{(:,1)}$


Gain
1
0.8
0.6
0.4
0.2
0
$X_{(:,2)}$
$X_{(:,1)}$
2
1
0
-1
-2
-2
0
2
4
0
0.2
0.4
0.6
0.8
1
Gain

Recursively Repeat Process for New Nodes

Once we have split a node, we simply call the same tree training algorithm on each of the new child nodes, leading to a recursive, self-similar training scheme



Tree Training Algorithm Recap

Algorithm 2 TRAINTREE (Training Data D)

1: **if** any stopping criteria are met **then**
2: **return** LEAFNODE(D)
3: **else**
4: Search over possible splits and chose the best one
5: Separate D into newly created child nodes, giving $D_1, \ldots, D_C$
6: **for** $c = 1 : C$ **do**
7: child-sub-tree$_c \leftarrow$ TRAINTREE(D_c)
8: **end for**
9: **return** Subtree with current node as root, chosen split, and child sub-trees
10: **end if**

Example: A tree model for deciding where to eat

Decide whether to wait for a table at a restaurant, based on the following attributes (Example from Russell and Norvig, AIMA)

- 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)
- Wait Estimate: estimated waiting time (0-10, 10-30, 30-60, >60)

Example: A tree model for deciding where to eat

Choosing a restaurant

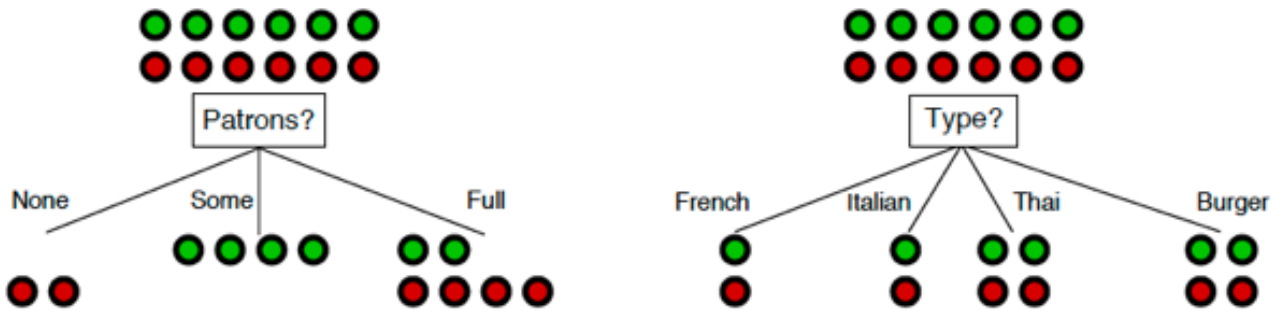
(Example from Russell & Norvig, AIMA)

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

Classification of examples is positive (T) or negative (F)

Start with root node containing all data

Which attribute to split?



Patrons? is a better choice—gives information about the classification

Patron vs. Type?

By choosing Patron, we end up with a partition (3 branches) with smaller entropy, ie, smaller uncertainty (0.45 bit)

By choosing Type, we end up with uncertainty of 1 bit.

Thus, we choose Patron over Type.

Uncertainty if we go with "Patron"

For "None" branch

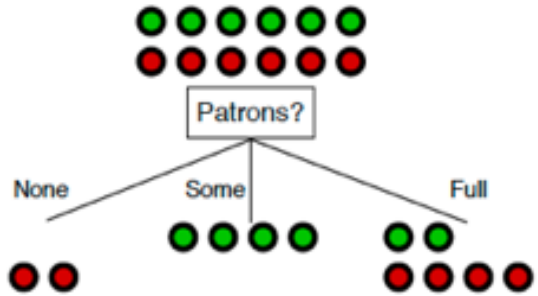
$$-\left(\frac{0}{0+2}\log\frac{0}{0+2}+\frac{2}{0+2}\log\frac{2}{0+2}\right)=0$$

For "Some" branch

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

For "Full" branch

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



For choosing "Patrons"

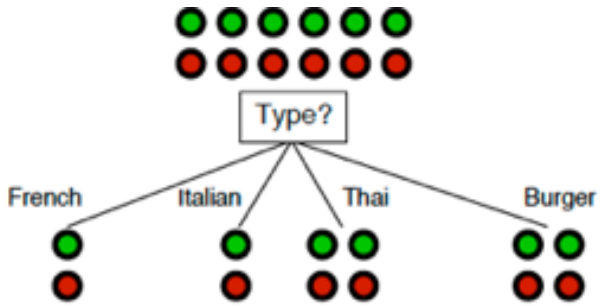
weighted average of each branch: this quantity is called conditional entropy

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

Conditional entropy for Type

For "French" branch

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



For "Italian" branch

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

For "Thai" and "Burger" branches

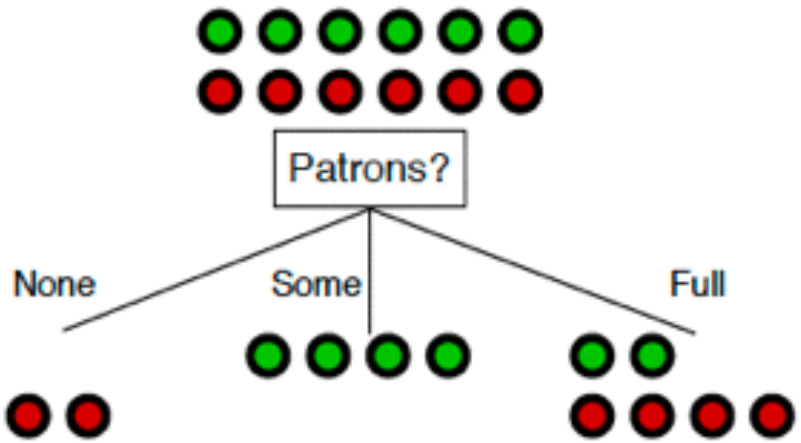
$$-\left(\frac{2}{2+2}\log\frac{2}{2+2}+\frac{2}{2+2}\log\frac{2}{2+2}\right)=1$$

For choosing "Type"

weighted average of each branch:

$$\frac{2}{12}*1+\frac{2}{12}*1+\frac{4}{12}*1+\frac{4}{12}*1=1$$

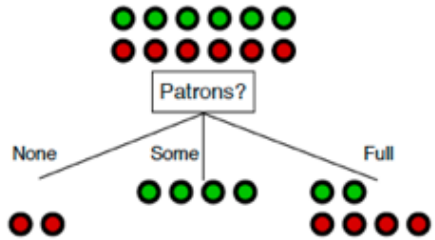
Do we split on “Non” or “Some”?



No, we do not

The decision is deterministic, as seen from the training data

next split?

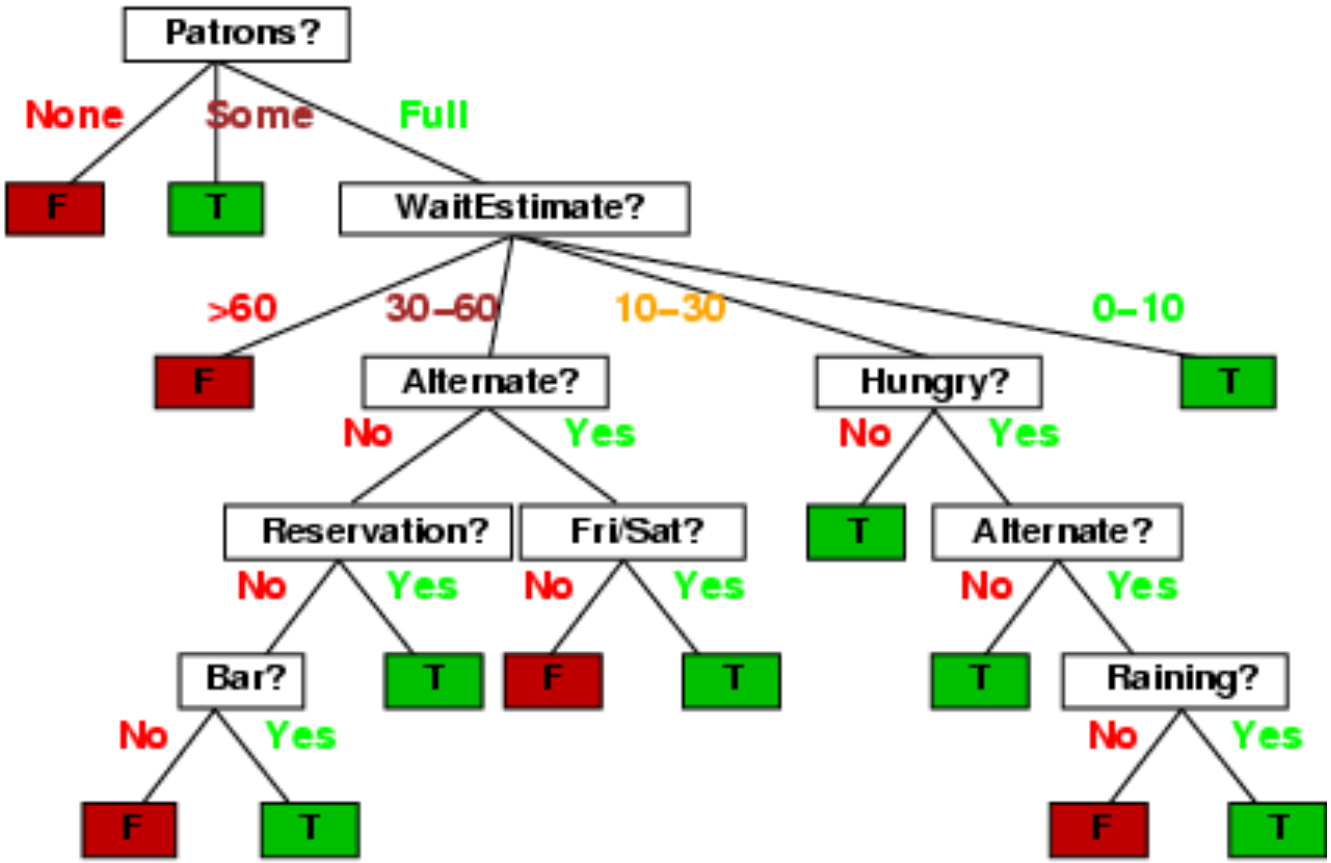


We will look only at the 6 instances with Patrons == Full

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

Classification of examples is positive (T) or negative (F)

Greedily, we build



An Algorithm for Classification Trees

Assume numerical features for simplicity (see Section 9.2.4 in ESL for categorical).

1. Start with root node with all data $\mathcal{R}_1 = \mathcal{X} = \mathbb{R}^p$.
2. For each feature $m = 1, \ldots, p$, for each value $v \in \mathbb{R}$ that we can split on:
   1. Split data set:
   $$I_< = \{i : x_i^{(m)} < v\} \qquad I_> = \{i : x_i^{(m)} \geq v\}$$
   2. Estimate class probabilities for each hypothetical child node:
   $$\hat{p}_< = \frac{\sum_{i \in I_<} y_i}{|I_<|} \qquad \hat{p}_> = \frac{\sum_{i \in I_>} y_i}{|I_>|}$$
   3. Compute the **gain** of the proposed split, e.g., using entropy, (note we can ignore the constant term)
   $$\text{Gain} = \text{constant} - \left(\frac{|I_<|}{|I_<| + |I_>|} B(\hat{p}_<) + \frac{|I_>|}{|I_<| + |I_>|} B(\hat{p}_>) \right)$$
3. Choose split, i.e., feature m and value v, with biggest gain
4. Recurse on both children, with datasets $(x_i, y_i)_{i \in I_<}$ and $(x_i, y_i)_{i \in I_>}$.

Computational Considerations

Numerical Features

- We could split on any feature, with any threshold
- However, for a given feature, the only split points we need to consider are the the n values in the training data for this feature.
- If we sort each feature by these n values, we can quickly compute our impurity metric of interest (cross entropy or others), skipping values where labels are unchanged.
  - This takes $O(p\, n \log n)$ time (sorting n elements takes $O(n \log n)$ steps).

Categorical Features

- Assuming q distinct categories, there are $2^{q-1} - 1$ possible binary partitions we can consider.
- If this becomes prohibitively large, we can replace the exhaustive search with drawing a large number of sample partitions and choosing the best one

Regression Trees

- Regression trees are almost exactly the same as classification trees. The only differences are
  1. They have different local leaf models based on predicting a continuous output $y \in \mathbb{R}$
  2. These use different split criteria
- The most common choice of predictive leaf model is just to have a constant output prediction $\hat{y}_j$ such that their overall predictive mapping can be expressed in the form

$$\hat{f}(x) = \sum_{j=1}^{L} \hat{y}_j \cdot \mathbb{I}(x \in \mathcal{R}_j)$$

- Using the squared loss, the optimal parameters are the sample means:

$$\hat{y}_j = \bar{y}_j \triangleq \frac{1}{n_j} \sum_{i \in D_j} y_i = \frac{\sum_{i=1}^{n} y_i \cdot \mathbb{I}(x_i \in \mathcal{R}_j)}{\sum_{i=1}^{n} \mathbb{I}(x_i \in \mathcal{R}_j)}$$

- Other losses, such as the L_1 loss (producing the median), are also viable
- Some algorithms make use of more complicated local leaf models, such as a linear regression or even a Gaussian process

Split Criteria for Regression Trees

- Regression tree split criteria are based on reducing a loss metric in the same manner as for classification:

$$\text{Gain} = B(D_j) - \frac{1}{n_j} \sum_{c \in \text{Children}(j)} n_c B(D_c)$$

- The difference is that $B(\cdot)$ now represents an empirical error

Split Criteria for Regression Trees (2)

- The most common choice for $B(\cdot)$ is the empirical mean squared error

$$B_{\text{mse}}(D_j) = \frac{1}{n_j} \sum_{i \in D_j} (y_i - \hat{f}_j(x_i))^2$$

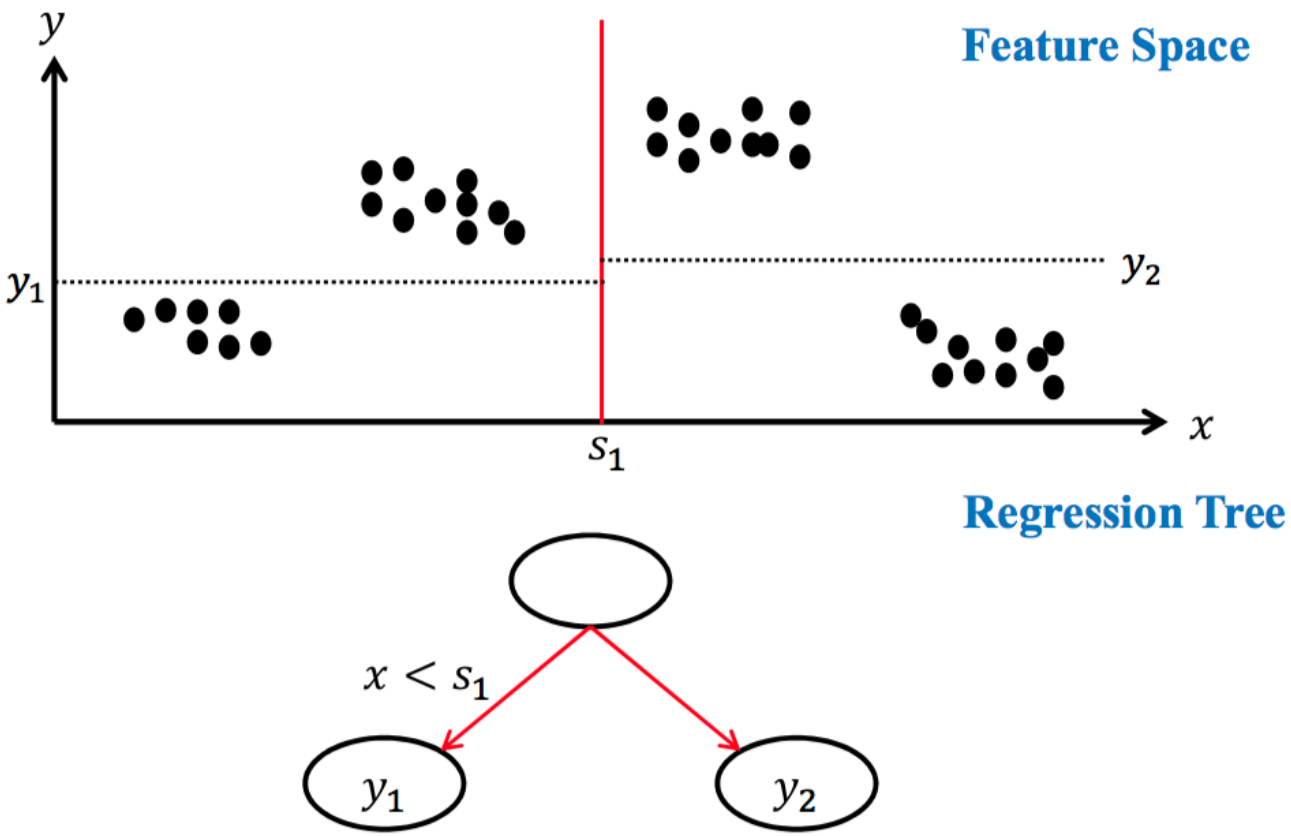
  where $\hat{f}_j(x)$ is the predictive model that would be employed if the node were a leaf.
- For the common choice of a constant predictive model, we have

$$\hat{f}_j(x) = \bar{y}_j = \frac{1}{n_j} \sum_{i \in D_j} y_i$$

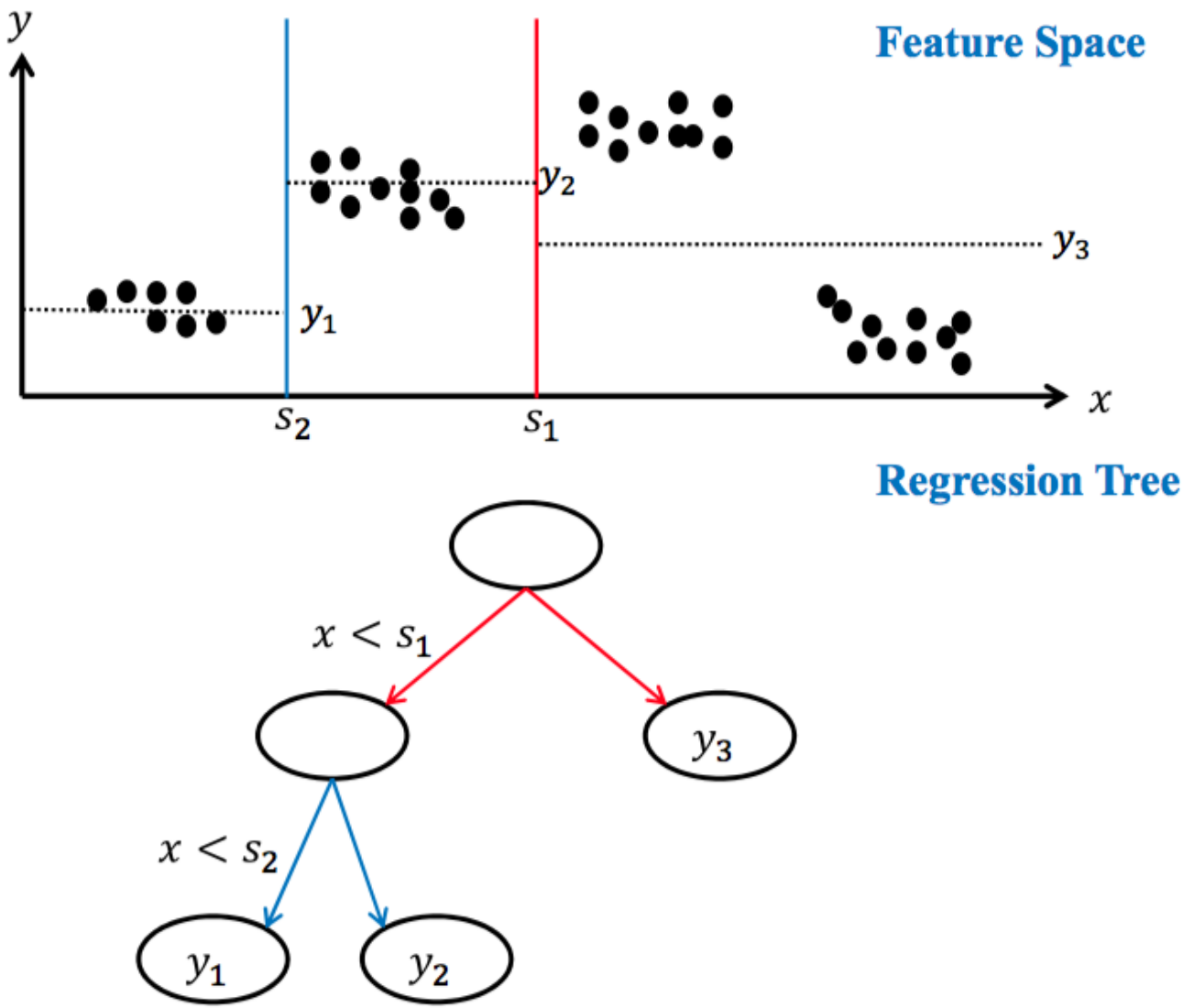
  where we note this is independent of the x
- Combing these choices, gives the following gain

$$\text{Gain} = \text{constant} - \frac{1}{n_j} \sum_{c \in \text{Children}(j)} \sum_{i \in D_c} (y_i - \bar{y}_c)^2$$

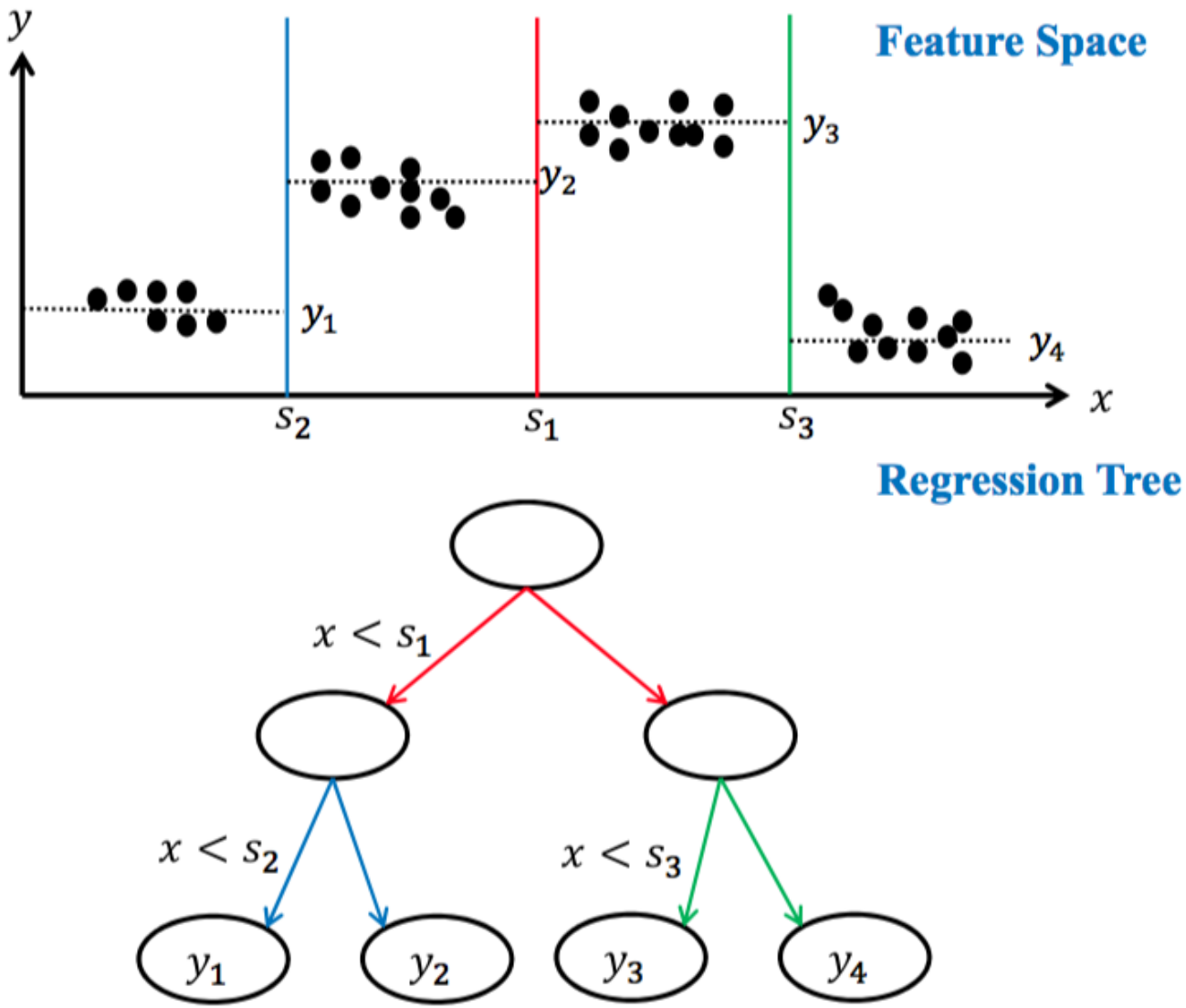
Example of Regression Trees



Example of Regression Trees

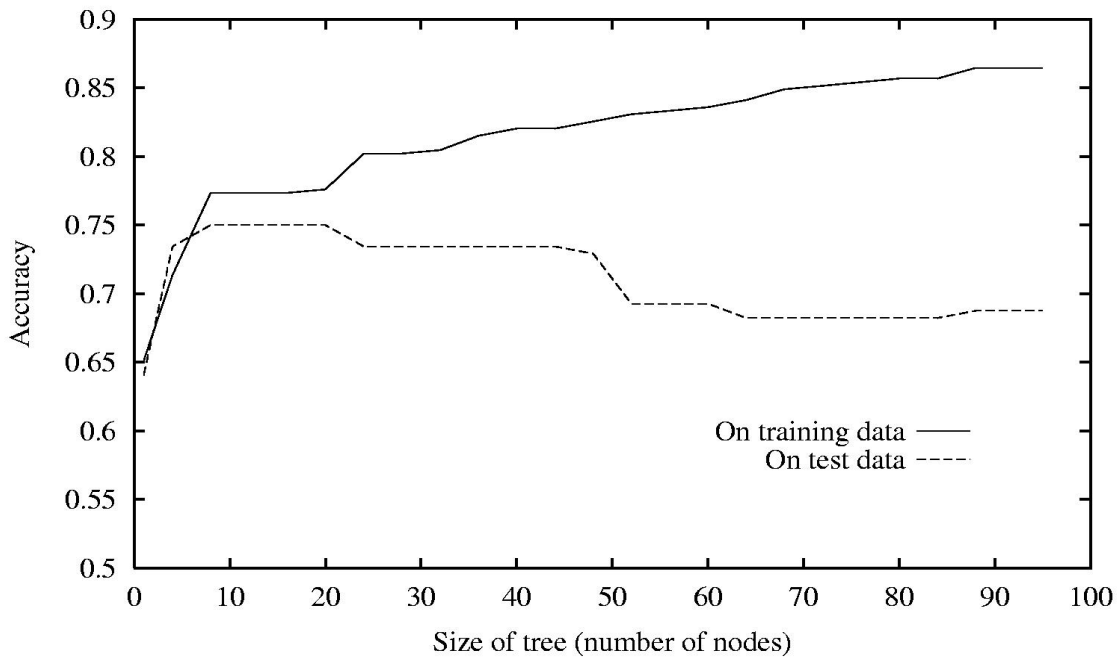


Example of Regression Trees



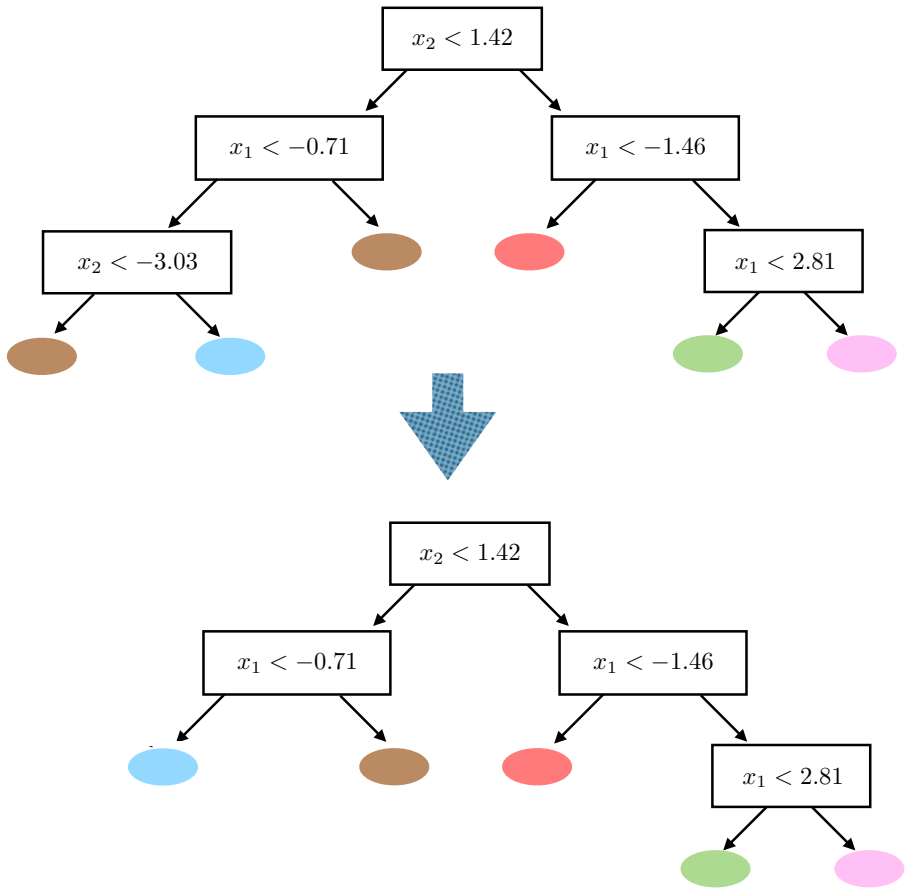
Regularization

- We need to be careful to appropriately regularize the tree, e.g. by introducing an appropriate maximum tree depth as a stopping criterion.
- If the tree is too deep, we can overfit, if the tree is too shallow, we underfit
- Max depth is a hyper-parameter that should be tuned by the data.
- Can also have other early stopping criteria such as not splitting if the current node exceeds a minimum number of datapoints at a node or a minimum node error, or if a minimum level of gain is not achieved when we try and split



Pruning

- An alternative strategy is to create an overly deep tree, and then prune it
- This involves collapsing down some nodes to a single leaf node
- Start at the leaves and step upwards deciding whether to collapse based on some metric
- Produces smaller tree that is less prone to overfitting
- However, pruning can be computational expensive



Regularized Objectives

- In general, can also use a regularized objective, e.g.

$$R^{\text{emp}}(T) + \lambda \text{size}(T)$$

- Early stopping: grow the tree from scratch and stop once the criterion objective starts to increase.
- Pruning: first grow the full tree and prune nodes (starting at leaves), until the objective starts to increase.
- Pruning is preferred as the choice of tree is less sensitive to "wrong" choices of split points and variables to split on in the first stages of tree fitting.
  - Horizon issue where no single split is helpful, but multiple splits are (e.g. XOR)
- Can use cross-validation to determine optimal λ.

Summary of learning trees

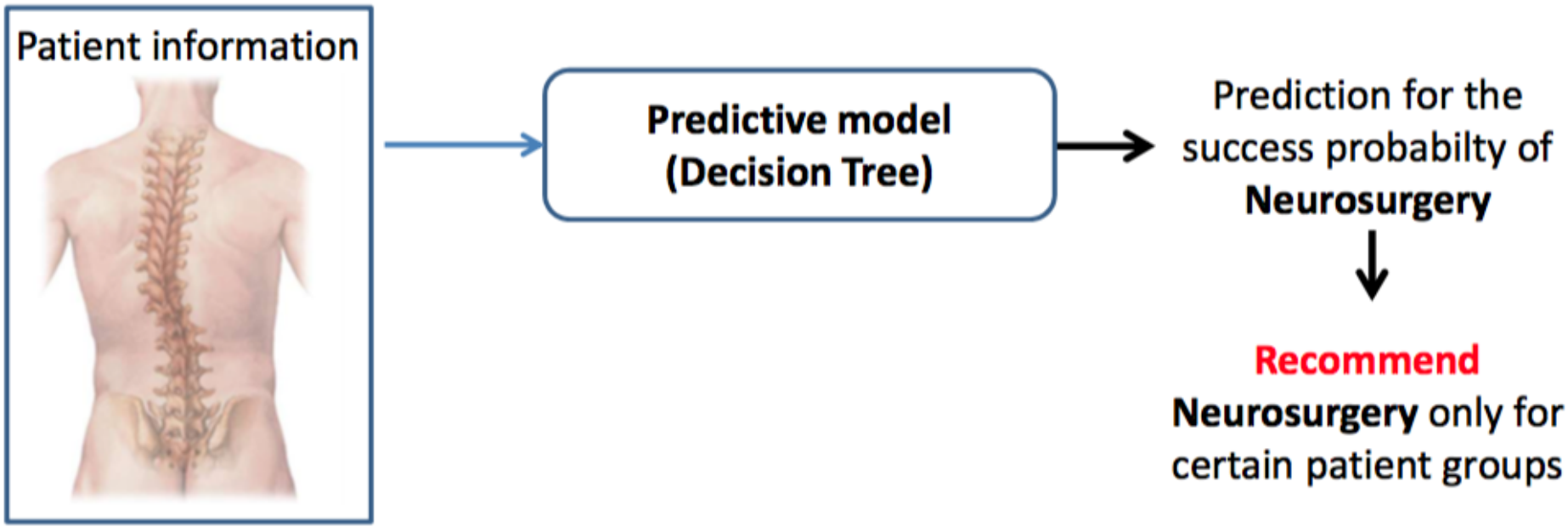
Advantages

- Easily interpretable by humans (as long as the tree is not too big)
- Computationally efficient
- Handle both numerical and categorical data
- Can produce arbitrarily complex predictors given enough data
- Do not require the data to be stored even though they are non-parametric
- Building block for various ensemble methods (more on this next lecture)

Disadvantages

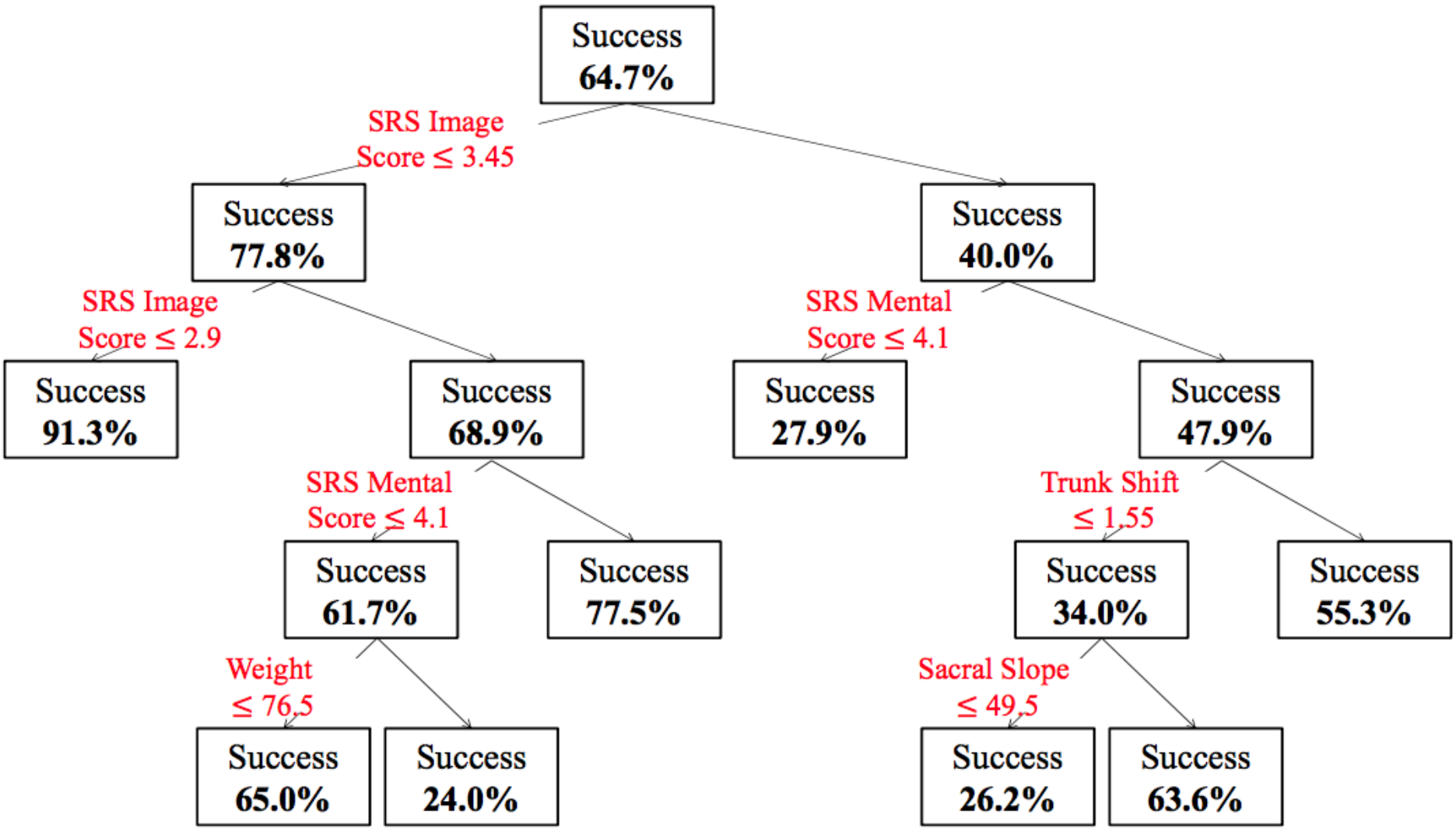
- Rather crude predictive model compared with more advanced techniques
- Training can suffer from horizon issues (think about an XOR problem)
- Limited theoretical underpinning
- Can be difficult to find right level of regularization (e.g. max depth)
- Unstable: small changes in input data lead to very different trees

Example: Neurosurgery



Type	Explanation	Note
Patient	1,449 patients with neurosurgery	2 year follow-up
Feature	91 Features (61 Continuous / 30 Binary)	
Label	MCID 1: 938 Patients (64.7%) MCID 0: 511 Patients (35.3%)	

Example: Neurosurgery

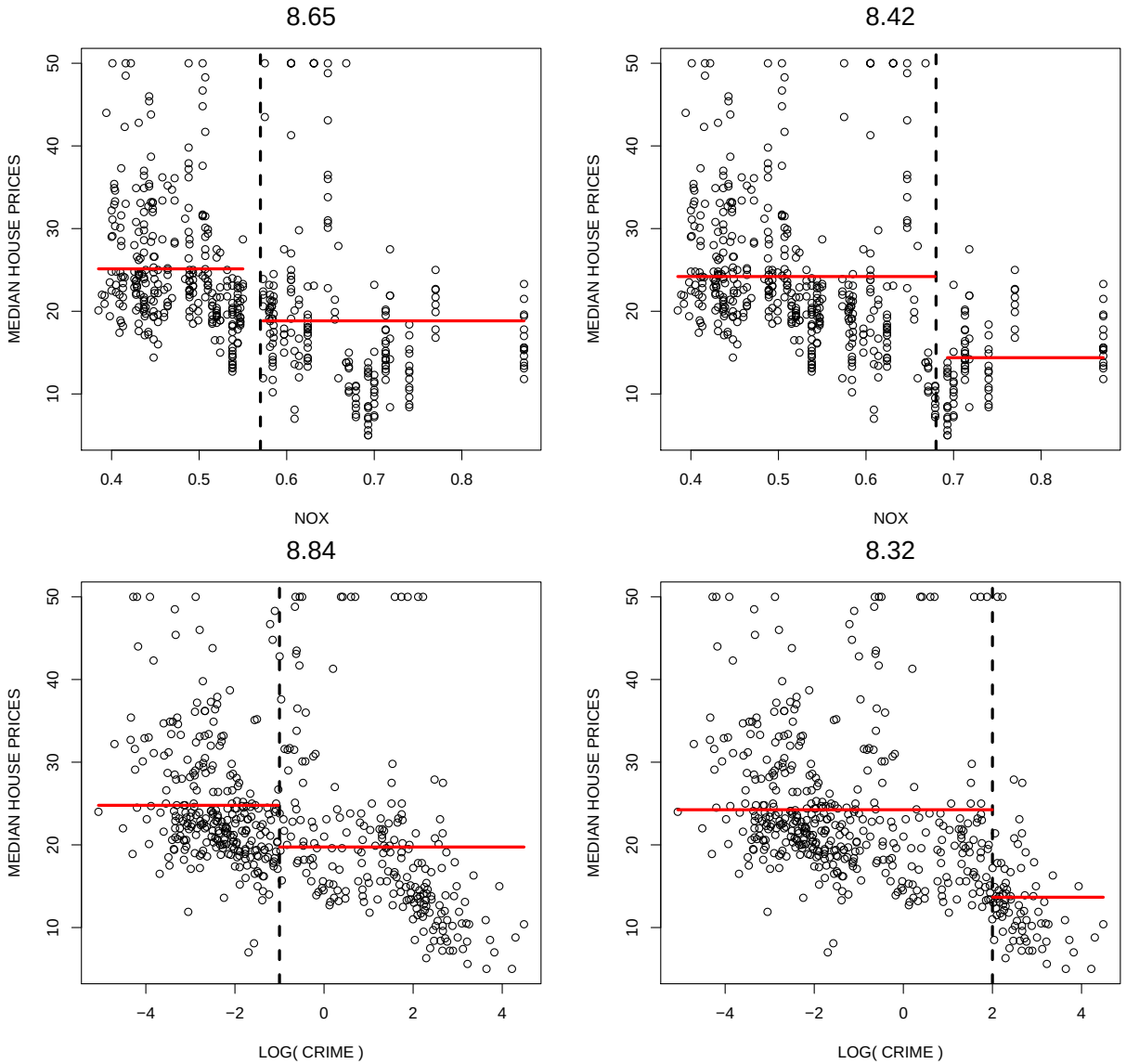


Example: Boston Housing Data

```
crim    per capita crime rate by town
nox     nitric oxides concentration (parts per 10 million)
rm      average number of rooms per dwelling
dis     weighted distances to five Boston employment centres
lstat   percentage of lower status of the population
... (6 more features)
```

- Predict median house value.

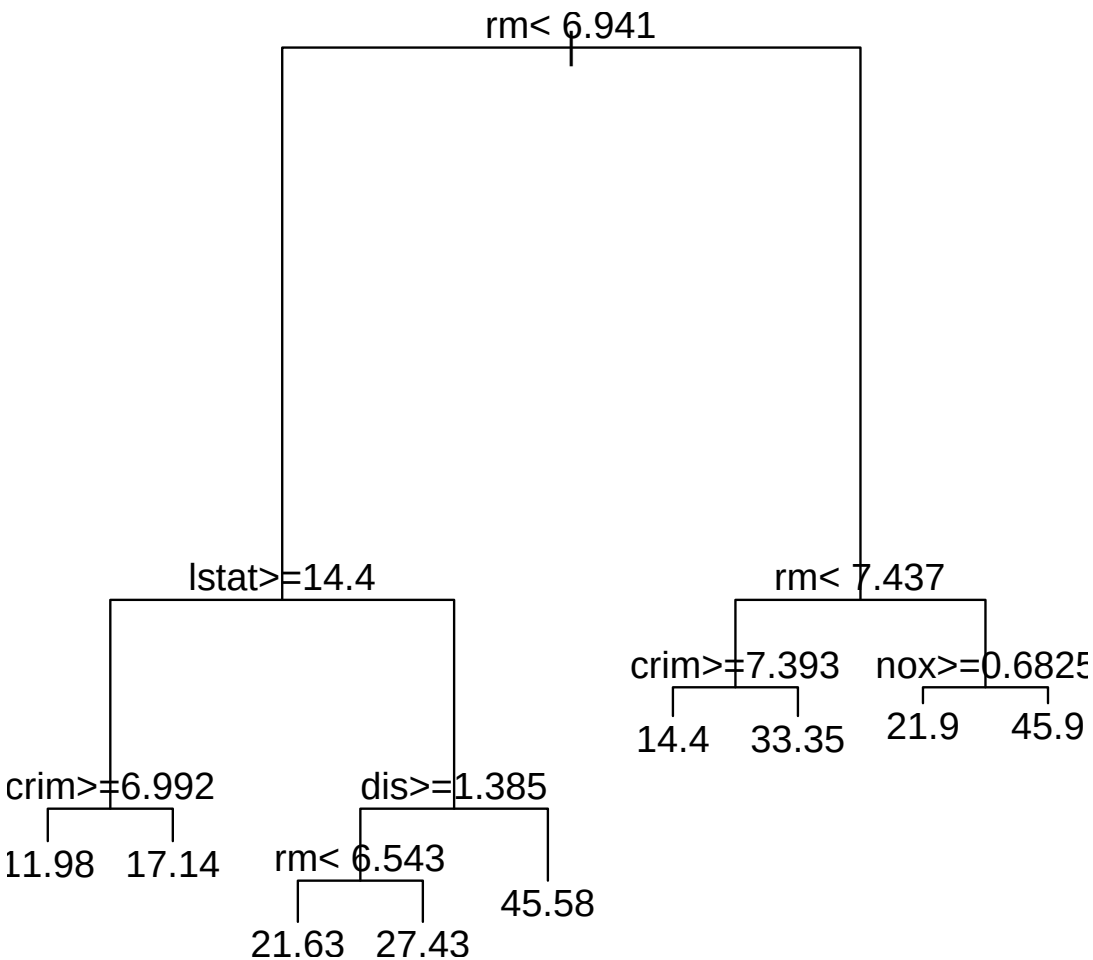
Example: Boston Housing Data



Different possible splits (features and thresholds) result in different quality measures.

Example: Boston Housing Data

- Overall, the best first split is on variable `rm`, average number of rooms per dwelling.
- Final tree contains predictions in leaf nodes.



Example: Pima Indians Diabetes Dataset

Goal: predict whether or not a patient has diabetes.

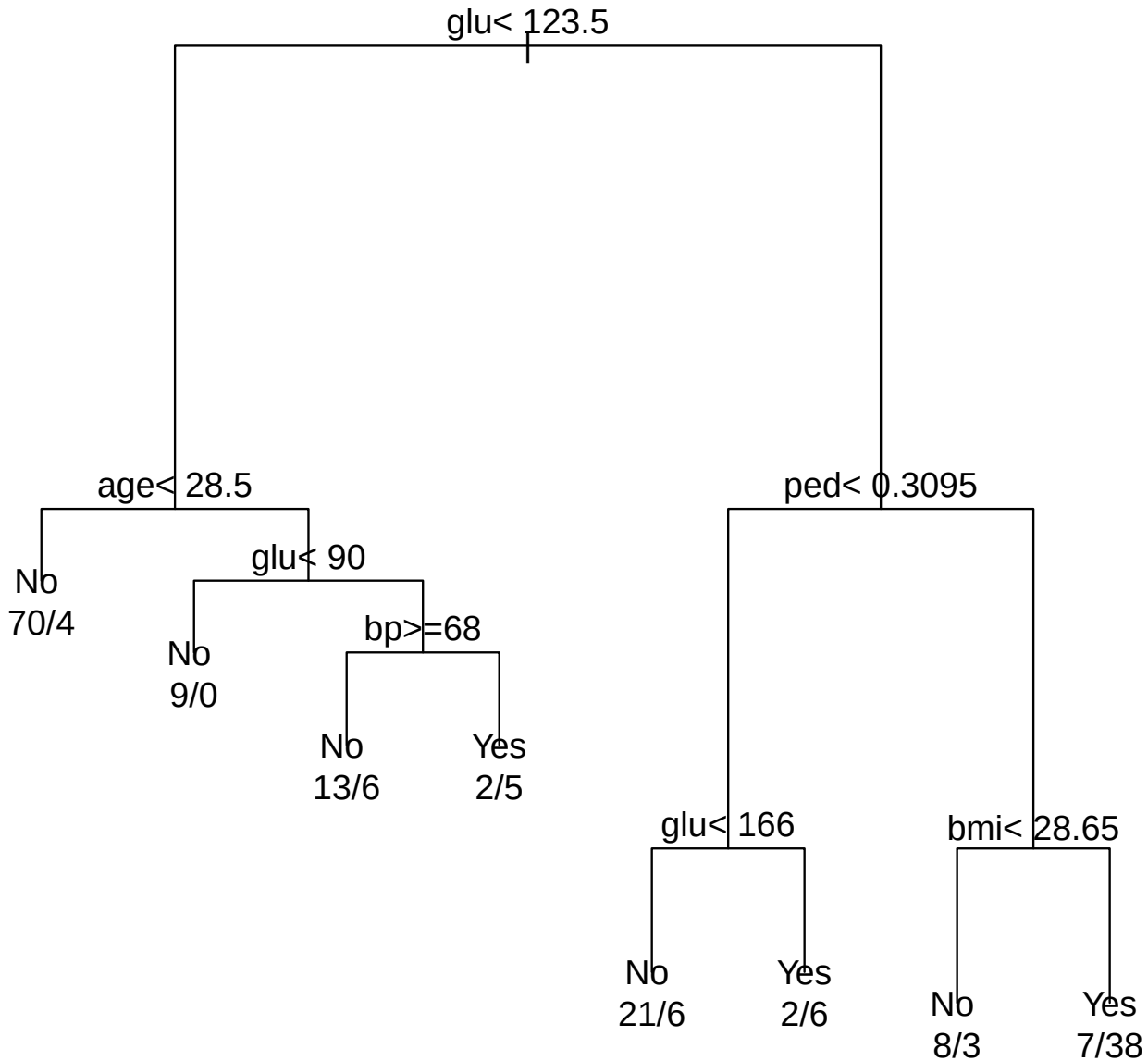
```
> library(rpart)
> library(MASS)
> data(Pima.tr)
> rp <- rpart(Pima.tr[,8] ~ ., data=Pima.tr[,-8])
> rp
n= 200

node), split, n, loss, yval, (yprob)
* denotes terminal node

1) root 200 68 No (0.66000000 0.34000000)
2) glu< 123.5 109 15 No (0.86238532 0.13761468)
4) age< 28.5 74  4 No (0.94594595 0.05405405) *
5) age>=28.5 35 11 No (0.68571429 0.31428571)
10) glu< 90 9  0 No (1.00000000 0.00000000) *
11) glu>=90 26 11 No (0.57692308 0.42307692)
22) bp>=68 19  6 No (0.68421053 0.31578947) *
23) bp< 68 7  2 Yes (0.28571429 0.71428571) *
3) glu>=123.5 91 38 Yes (0.41758242 0.58241758)
6) ped< 0.3095 35 12 No (0.65714286 0.34285714)
12) glu< 166 27  6 No (0.77777778 0.22222222) *
13) glu>=166 8  2 Yes (0.25000000 0.75000000) *
7) ped>=0.3095 56 15 Yes (0.26785714 0.73214286)
14) bmi< 28.65 11  3 No (0.72727273 0.27272727) *
15) bmi>=28.65 45  7 Yes (0.15555556 0.84444444) *
```

Example: Pima Indians Diabetes Dataset

```
> plot(rp,margin=0.1); text(rp,use.n=T)
```



Two possible trees.

```
> rp1 <- rpart(Pima.tr[,8] ~ ., data=Pima.tr[,-8])
> plot(rp1);text(rp1)
```

```
> rp2 <- rpart(Pima.tr[,8] ~ ., data=Pima.tr[,-8],
control=rpart.control(cp=0.05))
> plot(rp2);text(rp2)
```