



# 10-301/10-601 Introduction to Machine Learning

Machine Learning Department  
School of Computer Science  
Carnegie Mellon University

## Overfitting + k-Nearest Neighbors

Geoff Gordon

Lecture 4

Sep 8, 2025

with thanks to Matt  
Gormley and Henry Chai

# Reminders

- Homework 1 grades released (see Gradescope)
- Homework 1 Resubmission
  - If score < 61.2 points on written, you may resubmit by Wednesday, September 10th 11:59pm (just for HW1) (no extensions)
- Homework 2: Decision Trees
  - Out: Thu, Sep. 4 (see Coursework page on website)
  - Due: Mon, Sep. 15 at 11:59pm
  - Separate Gradescope slots for written and programming
    - Max 10 submissions for programming: debug locally, read FAQ, checklist!

# Q&A

**Q:** Why don't my entropy calculations match those on the slides?

# Q&A

**Q:** Why don't my entropy calculations match those on the slides?

**A:** Remember that  $H(Y)$  is conventionally reported in *bits*, i.e., log base 2.  
 $H(Y) = -P(Y=0) \log_2 P(Y=0) - P(Y=1) \log_2 P(Y=1)$

# Q&A

**Q:** Why don't my entropy calculations match those on the slides?

**A:** Remember that  $H(Y)$  is conventionally reported in *bits*, i.e., log base 2.  
 $H(Y) = -P(Y=0) \log_2 P(Y=0) - P(Y=1) \log_2 P(Y=1)$

**Q:** When and how do we decide to stop growing trees?

# Q&A

**Q:** Why don't my entropy calculations match those on the slides?

**A:** Remember that  $H(Y)$  is conventionally reported in *bits*, i.e., log base 2.  
 $H(Y) = -P(Y=0) \log_2 P(Y=0) - P(Y=1) \log_2 P(Y=1)$

**Q:** When and how do we decide to stop growing trees?

**A:** We'll address this question today. Why might we want to stop growing?

# Q&A

**Q:** What if an attribute can take more than two values?  
Strings, colors, real numbers, ...

# Q&A

**Q:** What if an attribute can take more than two values?  
Strings, colors, real numbers, ...

**A:** Could use 3-way splits: fruit → apple, orange, banana.  
How could this fail on (say) real-valued attributes?  
Common procedure: limit to binary splits



# Q&A

**Q:** What if an attribute can take more than two values?  
Strings, colors, real numbers, ...

**A:** Could use 3-way splits: fruit → apple, orange, banana.  
How could this fail on (say) real-valued attributes?  
Common procedure: limit to binary splits

**Q:** How can we find a good binary split efficiently?

# Q&A

**Q:** What if an attribute can take more than two values?  
Strings, colors, real numbers, ...

**A:** Could use 3-way splits: fruit  $\rightarrow$  apple, orange, banana.  
How could this fail on (say) real-valued attributes?  
Common procedure: limit to binary splits

**Q:** How can we find a good binary split efficiently?

**A:** If an attribute is ordered, there's a clever trick that only considers  $O(L)$  splits where  $L = \#$  of values the attribute takes in the training set. Can you guess what it does?

# Q&A

**Q:** What does decision tree training do if a branch receives no data?

# Q&A

**Q:** What does decision tree training do if a branch receives no data?

**A:** Then we hit the base case and create a leaf node. So the real question is what does majority vote do when there is no data? Of course, there is no majority label, so (if forced to) we could just return one randomly.

# Q&A

**Q:** What does decision tree training do if a branch receives no data?

**A:** Then we hit the base case and create a leaf node. So the real question is what does majority vote do when there is no data? Of course, there is no majority label, so (if forced to) we could just return one randomly.

**Q:** What do we do at test time when we observe a value for a feature that we didn't see at training time?

# Q&A

**Q:** What does decision tree training do if a branch receives no data?

**A:** Then we hit the base case and create a leaf node. So the real question is what does majority vote do when there is no data? Of course, there is no majority label, so (if forced to) we could just return one randomly.

**Q:** What do we do at test time when we observe a value for a feature that we didn't see at training time?

**A:** This really just a variant of the first question. That said, a real DT implementation needs to elegantly handle this case. E.g., we pretend that our tree actually had a branch for this attribute value already — and this branch had no data, so we're in the case above.

# Q&A

**Q:** Which splitting criterion should we use?

# Q&A

**Q:** Which splitting criterion should we use?

**A:** ...



# Empirical comparison: Splitting Criteria

Bluntine & Niblett (1992) compared 4 criteria (random, Gini, mutual information, Marshall) on 12 datasets

## Medical Diagnosis Datasets: (4 of 12)

- **hypo**: data set of 3772 examples records expert opinion on possible hypo- thyroid conditions from 29 real and discrete attributes of the patient such as sex, age, taking of relevant drugs, and hormone readings taken from drug samples.
- **breast**: The classes are reoccurrence or non-reoccurrence of breast cancer sometime after an operation. There are nine attributes giving details about the original cancer nodes, position on the breast, and age, with multi-valued discrete and real values.
- **tumor**: examples of the location of a primary tumor
- **lymph**: from the lymphography domain in oncology. The classes are normal, metastases, malignant, and fibrosis, and there are nineteen attributes giving details about the lymphatics and lymph nodes

Table 1. Properties of the data sets

Data Set	Classes	Attr.s	Training Set	Test Set
hypo	4	29	1000	2772
breast	2	9	200	86
tumor	22	18	237	102
lymph	4	18	103	45
LED	10	7	200	1800
mush	2	22	200	7924
votes	2	17	200	235
votes1	2	16	200	235
iris	3	4	100	50
glass	7	9	100	114
xd6	2	10	200	400
pole	2	4	200	1647

# Empirical comparison: Splitting Criteria

Table 3. Error for different splitting rules (pruned trees).

Data Set	Splitting Rule			
	GINI	Info. Gain	Marsh.	Random
hypo	1.01 $\pm$ 0.29	0.95 $\pm$ 0.22	1.27 $\pm$ 0.47	7.44 $\pm$ 0.53
breast	28.66 $\pm$ 3.87	28.49 $\pm$ 4.28	27.15 $\pm$ 4.22	29.65 $\pm$ 4.97
tumor	60.88 $\pm$ 5.44	62.70 $\pm$ 3.89	61.62 $\pm$ 3.98	67.94 $\pm$ 5.68
lymph	24.44 $\pm$ 6.92	24.00 $\pm$ 6.87	24.33 $\pm$ 5.51	32.33 $\pm$ 11.25
LED	33.77 $\pm$ 3.06	32.89 $\pm$ 2.59	33.15 $\pm$ 4.02	38.18 $\pm$ 4.57
mush	1.44 $\pm$ 0.47	1.44 $\pm$ 0.47	7.31 $\pm$ 2.25	8.77 $\pm$ 4.65
votes	4.47 $\pm$ 0.95	4.57 $\pm$ 0.87	11.77 $\pm$ 3.95	12.40 $\pm$ 4.56
votes1	12.79 $\pm$ 1.48	13.04 $\pm$ 1.65	15.13 $\pm$ 2.89	15.62 $\pm$ 2.73
iris	5.00 $\pm$ 3.08	4.90 $\pm$ 3.08	5.50 $\pm$ 2.59	14.20 $\pm$ 6.77
glass	39.56 $\pm$ 6.20	50.57 $\pm$ 6.73	40.53 $\pm$ 6.41	53.20 $\pm$ 5.01
xd6	22.14 $\pm$ 3.23	22.17 $\pm$ 3.36	22.06 $\pm$ 3.37	31.86 $\pm$ 3.62
pole	15.43 $\pm$ 1.51	15.47 $\pm$ 0.88	15.01 $\pm$ 1.15	26.38 $\pm$ 6.92

Info. Gain is another name  
for *mutual information*

# Experiments: Splitting Criteria

*Table 4.* Difference and significance of error for GINI splitting rule versus others.

Data Set	Splitting Rule		
	Info. Gain	Marsh.	Random
hypo	−0.06 (0.82)	0.26 (0.99)	6.43 (1.00)
breast	−0.17 (0.23)	−1.51 (0.94)	0.99 (0.72)
tumor	1.81 (0.84)	0.74 (0.39)	7.06 (0.99)
lymph	−0.44 (0.83)	0.11 (0.05)	7.89 (0.99)
LED	0.12 (0.17)	6.58	
mush	0.00 (0.50)	5.86	
votes	0.11 (0.55)	7.30	
votes1	0.26 (0.47)	2.34	
iris	−0.10 (0.67)	0.50	
glass	1.01 (0.50)	0.96	
xd6	0.04 (0.11)	−0.07	
pole	0.03 (0.11)	−0.43	

Gini better: +  
Others better: −

**Key Takeaway:**  
GINI criterion  
and Mutual  
Information are  
statistically  
indistinguishable!

Results are formatted as  
A.AA (B.BB) where:

1. A.AA is the **average difference in errors** between the two methods
2. B.BB is the **significance** of the difference according to a two-tailed **paired t-test**

# **INDUCTIVE BIAS (FOR DECISION TREES)**

# Decision Tree Learning Example

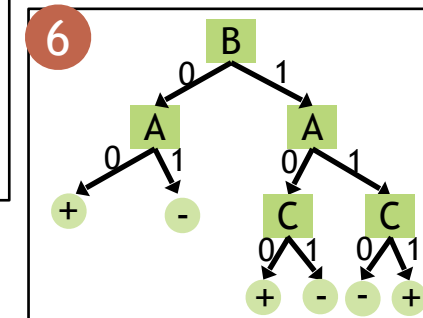
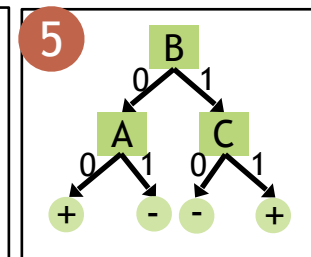
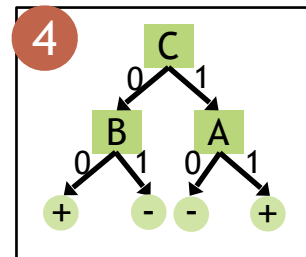
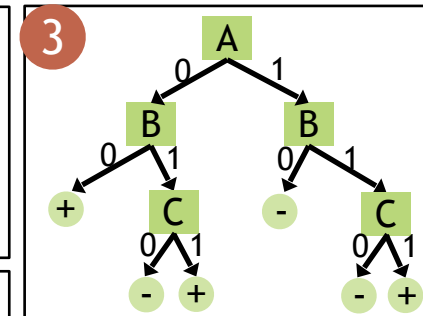
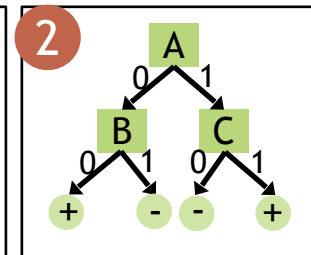
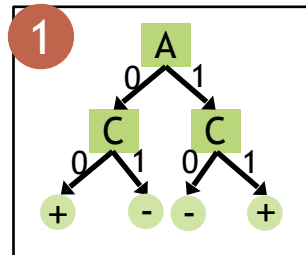
## Dataset:

Output Y, Attributes A, B, C

Y	A	B	C
+	0	0	0
+	0	0	1
-	0	1	0
+	0	1	1
-	1	0	0
-	1	0	1
-	1	1	0
+	1	1	1

## Poll Question 1

Which decision tree would you learn if you used error rate as the splitting criterion?  
(Break ties alphabetically.)



# Decision Tree Learning Example

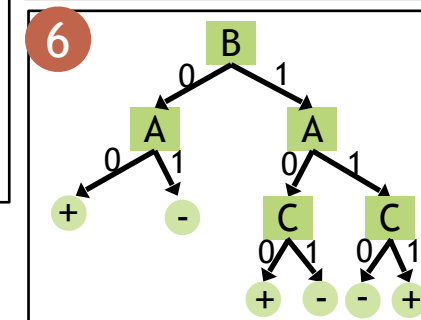
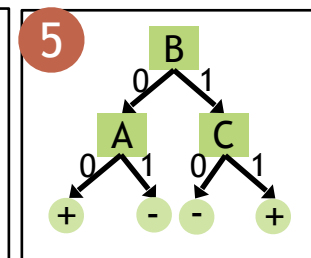
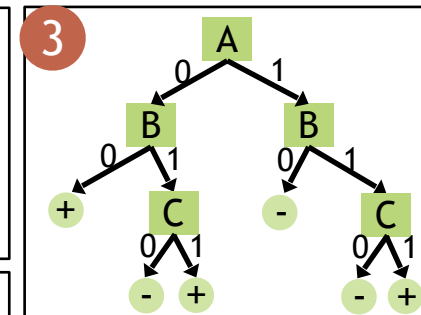
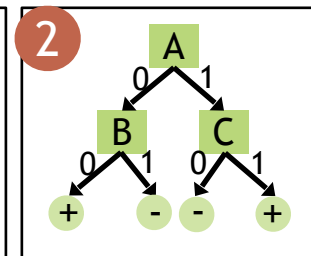
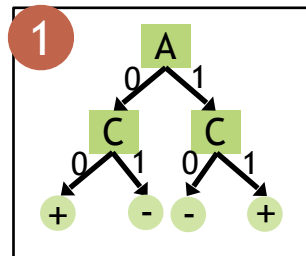
## Dataset:

Output Y, Attributes A, B, C

Y	A	B	C
+	0	0	0
+	0	0	1
-	0	1	0
+	0	1	1
-	1	0	0
-	1	0	1
-	1	1	0
+	1	1	1

## Poll Question 1

Which decision tree would you learn if you used error rate as the splitting criterion?  
(Break ties alphabetically.)



# Decision Tree Learning Example

## Dataset:

Output Y, Attributes A, B, C

Y	A	B	C
0	0	0	0
1	0	0	1
0	1	0	0
1	1	0	1
0	0	1	0
1	0	1	1
0	1	1	0
1	1	1	1

 **WIKIPEDIA**  
The Free Encyclopedia

 **Mr. Yuk**

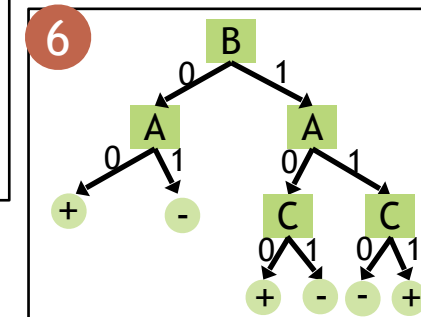
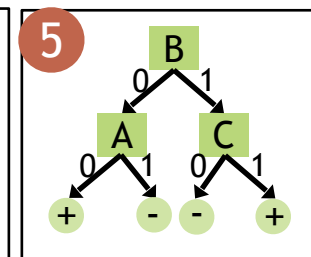
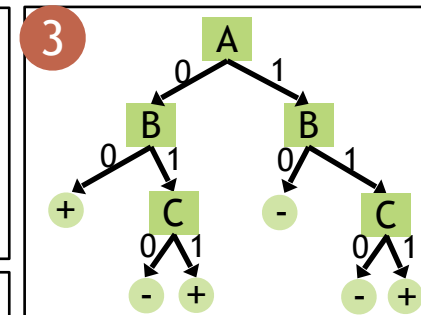
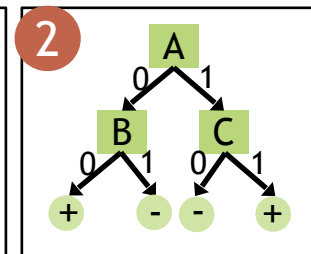
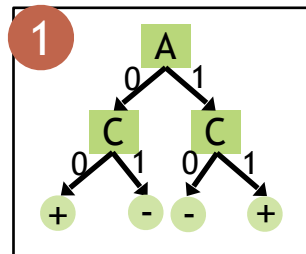
Article [Talk](#)

From Wikipedia, the free encyclopedia

**Mr. Yuk** is a trademarked graphic image, created by [UPMC Children's Hospital of Pittsburgh](#), and widely employed in the [United States](#) in labeling of substances that are [poisonous](#) if ingested.

## Poll Question 1

Which decision tree would you learn if you used error rate as the splitting criterion?  
(Break ties alphabetically.)



# Decision Tree Learning Example

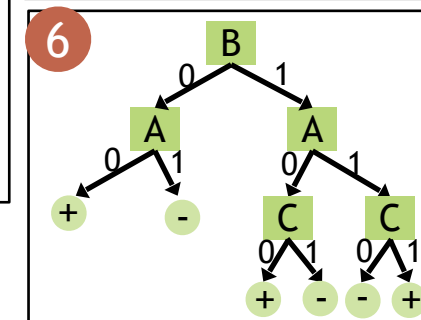
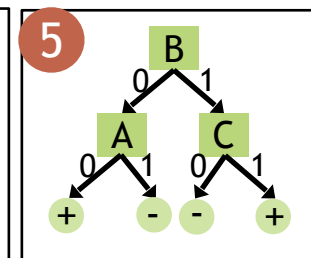
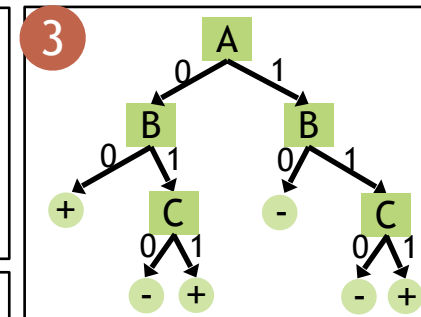
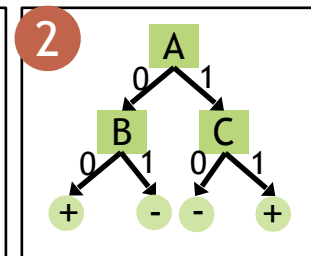
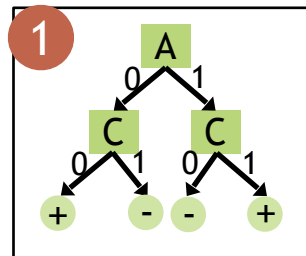
## Dataset:

Output Y, Attributes A, B, C

Y	A	B	C
+	0	0	0
+	0	0	1
-	0	1	0
+	0	1	1
-	1	0	0
-	1	0	1
-	1	1	0
+	1	1	1

## Poll Question 1

Which decision tree would you learn if you used error rate as the splitting criterion?  
(Break ties alphabetically.)





# Decision Tree Learning Example

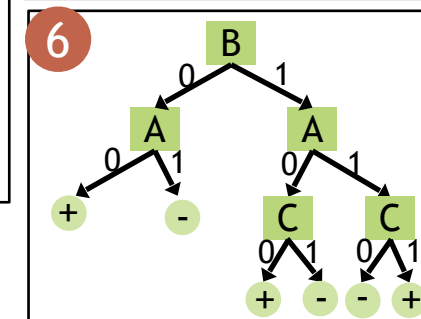
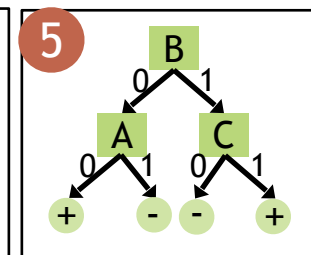
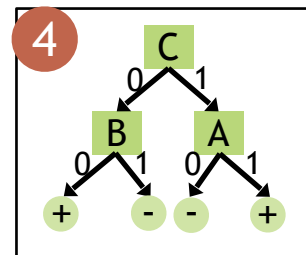
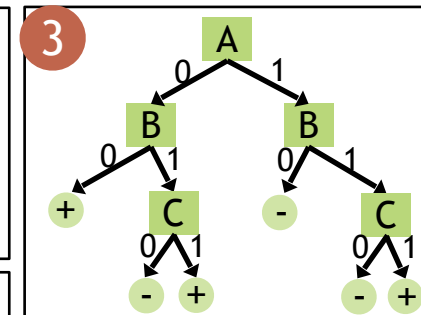
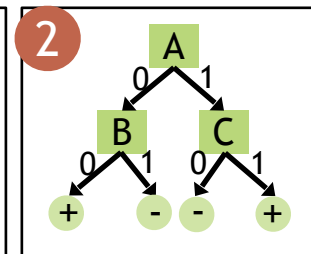
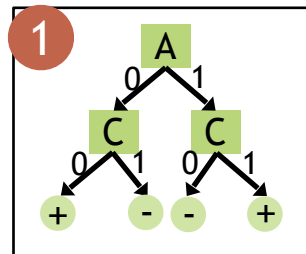
## Dataset:

Output Y, Attributes A, B, C

Y	A	B	C
+	0	0	0
+	0	0	1
-	0	1	0
+	0	1	1
-	1	0	0
-	1	0	1
-	1	1	0
+	1	1	1

## Poll Question 2

Which decision tree is the smallest tree that achieves the lowest possible training error?



# Decision Tree Learning Example

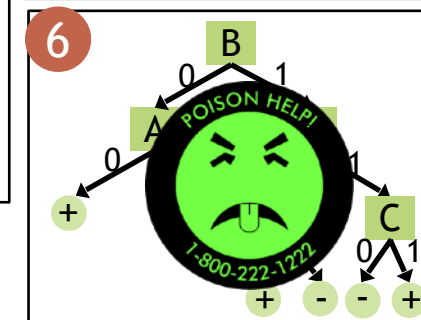
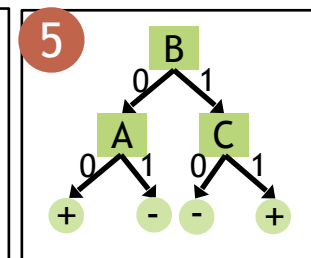
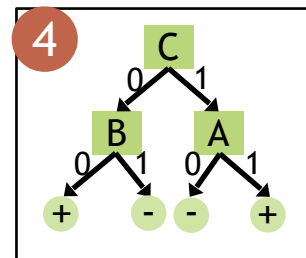
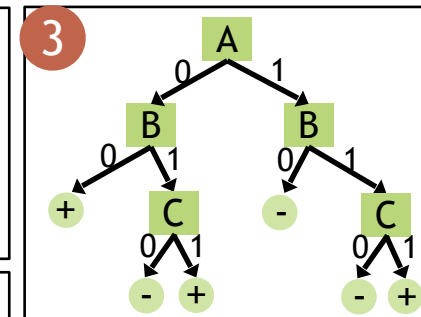
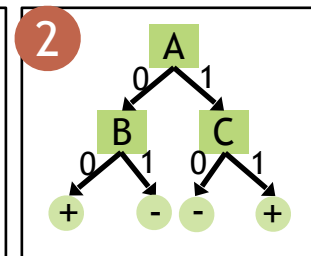
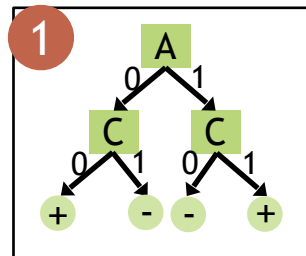
## Dataset:

Output Y, Attributes A, B, C


Y	A	B	C
+	0	0	0
+	0	0	1
-	0	1	0
+	0	1	1
-	1	0	0
-	1	0	1
-	1	1	0
+	1	1	1

## Poll Question 2

Which decision tree is the smallest tree that achieves the lowest possible training error?



# Background: Greedy Search



Start  
State

End  
States

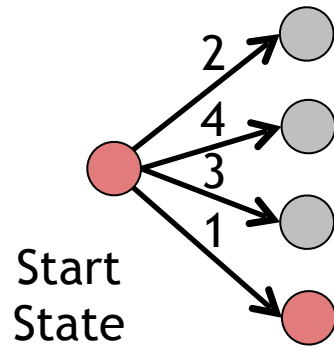
## Goal:

- Search space consists of nodes and weighted edges
- Goal is to find the lowest (total) weight path from root to a leaf

## Greedy Search:

- At each node, select the edge with lowest (immediate) weight
- **Heuristic** method of search (i.e. does *not* necessarily find the best path)
- Computation time: **linear** in max path length

# Background: Greedy Search



End  
States

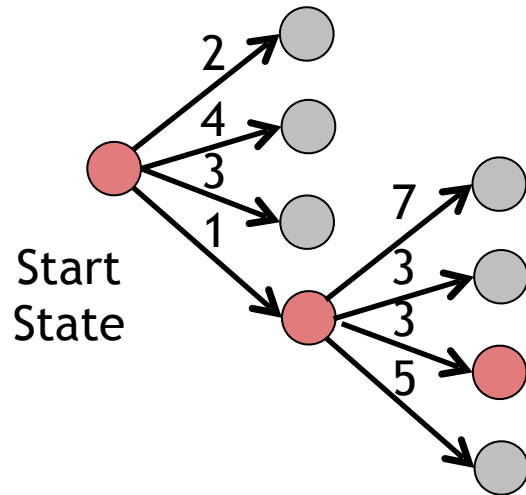
## Goal:

- Search space consists of nodes and weighted edges
- Goal is to find the lowest (total) weight path from root to a leaf

## Greedy Search:

- At each node, select the edge with lowest (immediate) weight
- **Heuristic** method of search (i.e. does *not* necessarily find the best path)
- Computation time: **linear** in max path length

# Background: Greedy Search



End  
States

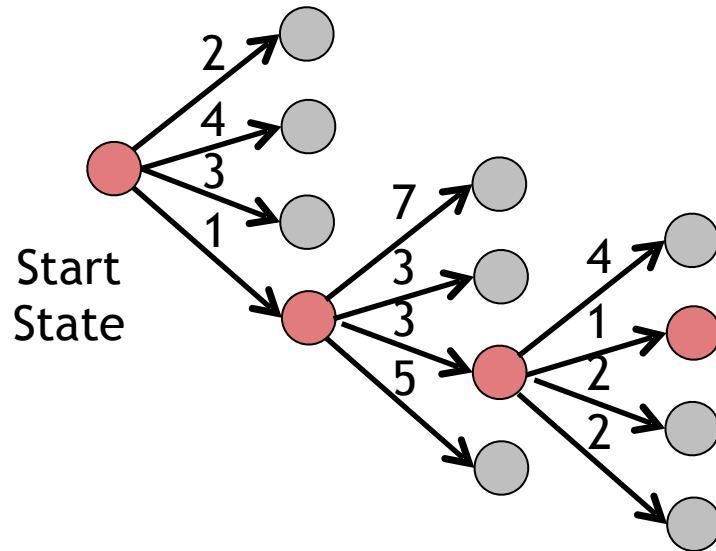
## Goal:

- Search space consists of nodes and weighted edges
- Goal is to find the lowest (total) weight path from root to a leaf

## Greedy Search:

- At each node, select the edge with lowest (immediate) weight
- **Heuristic** method of search (i.e. does *not* necessarily find the best path)
- Computation time: **linear** in max path length

# Background: Greedy Search



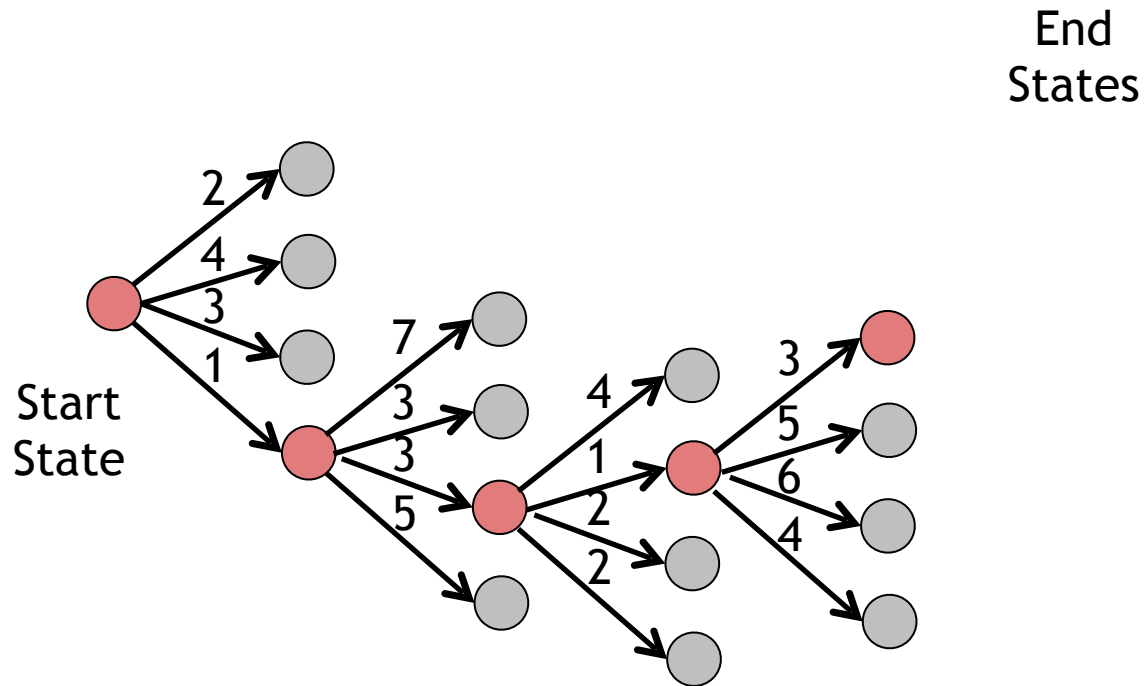
## Goal:

- Search space consists of nodes and weighted edges
- Goal is to find the lowest (total) weight path from root to a leaf

## Greedy Search:

- At each node, select the edge with lowest (immediate) weight
- **Heuristic** method of search (i.e. does *not* necessarily find the best path)
- Computation time: **linear** in max path length

# Background: Greedy Search



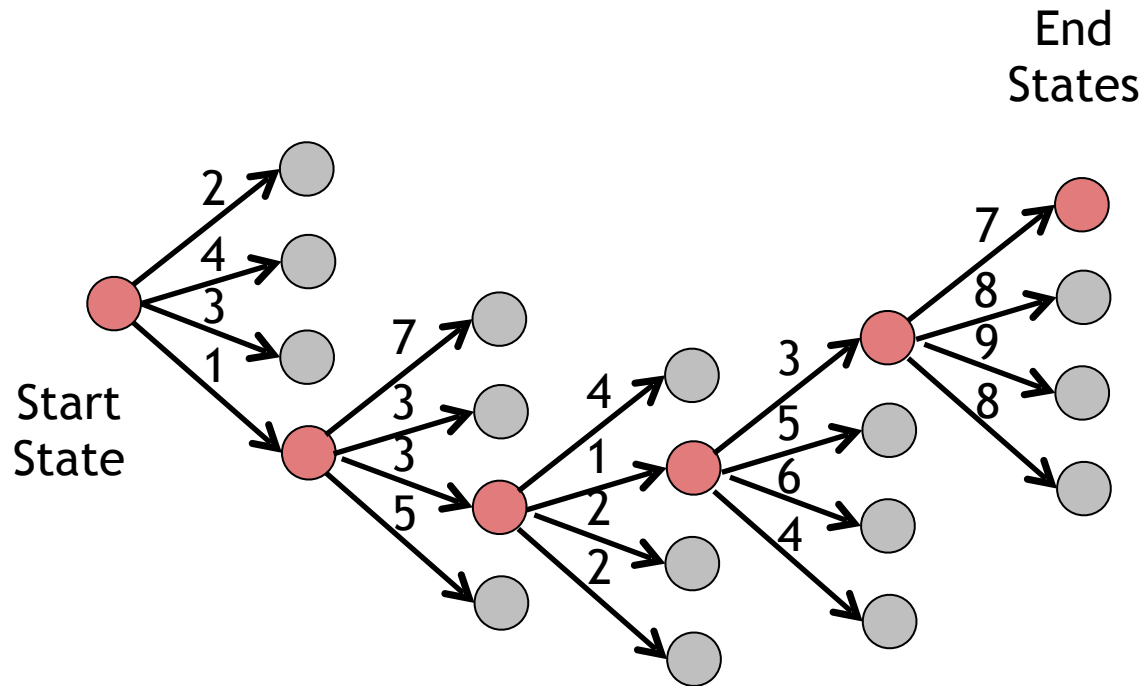
## Goal:

- Search space consists of nodes and weighted edges
- Goal is to find the lowest (total) weight path from root to a leaf

## Greedy Search:

- At each node, select the edge with lowest (immediate) weight
- **Heuristic** method of search (i.e. does *not* necessarily find the best path)
- Computation time: **linear** in max path length

# Background: Greedy Search



## Goal:

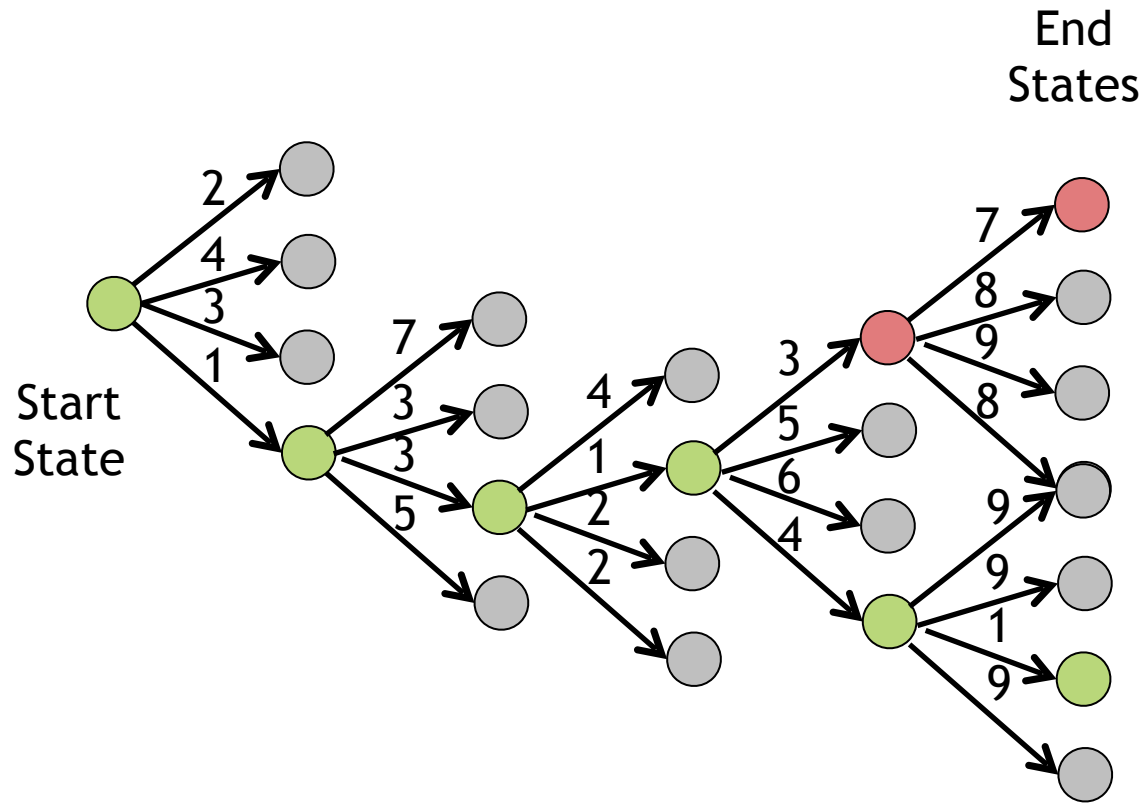
- Search space consists of nodes and weighted edges
- Goal is to find the lowest (total) weight path from root to a leaf

## Greedy Search:

- At each node, select the edge with lowest (immediate) weight
- **Heuristic** method of search (i.e. does *not* necessarily find the best path)
- Computation time: **linear** in max path length



# Background: Greedy Search



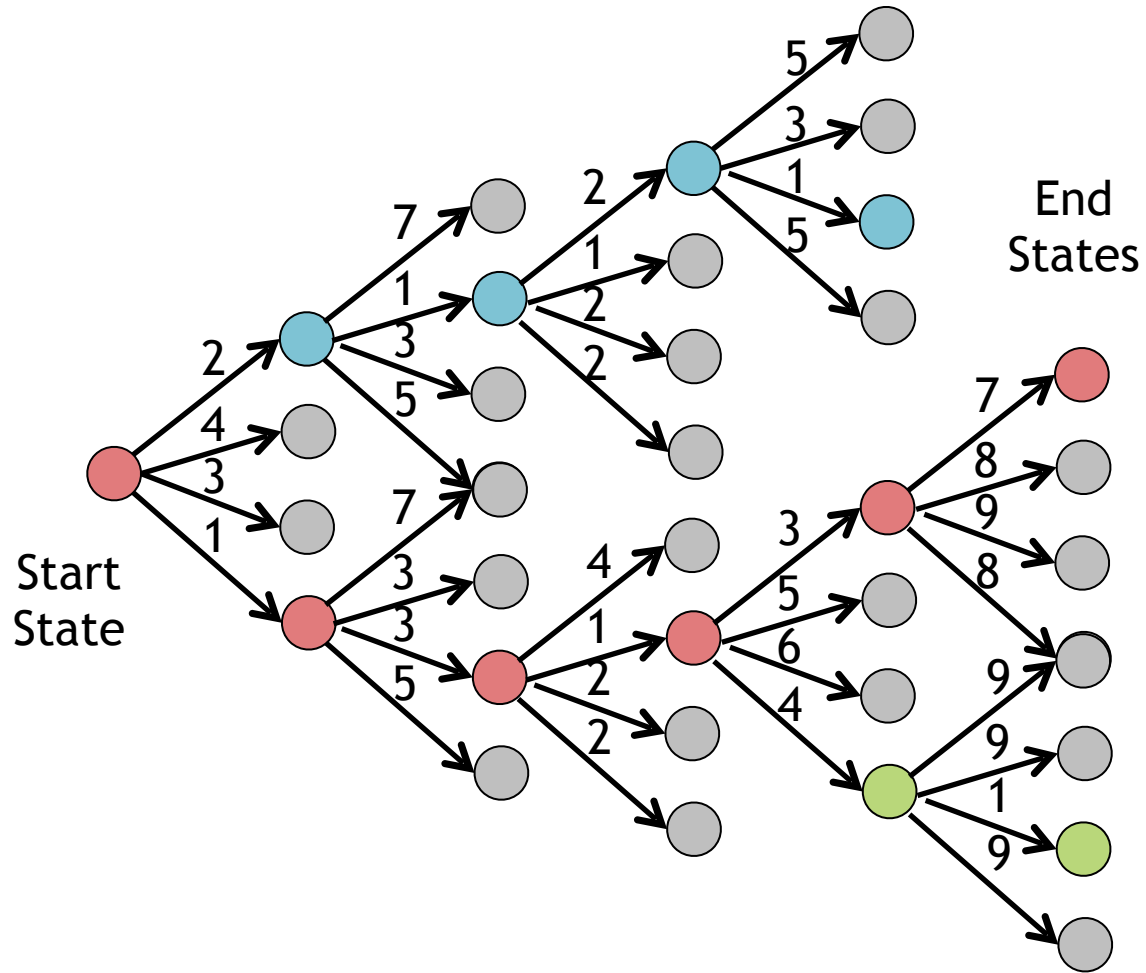
## Goal:

- Search space consists of nodes and weighted edges
- Goal is to find the lowest (total) weight path from root to a leaf

## Greedy Search:

- At each node, selects the edge with lowest (immediate) weight
- **Heuristic** method of search (i.e. does *not* necessarily find the best path)
- Computation time: **linear** in max path length

# Background: Greedy Search



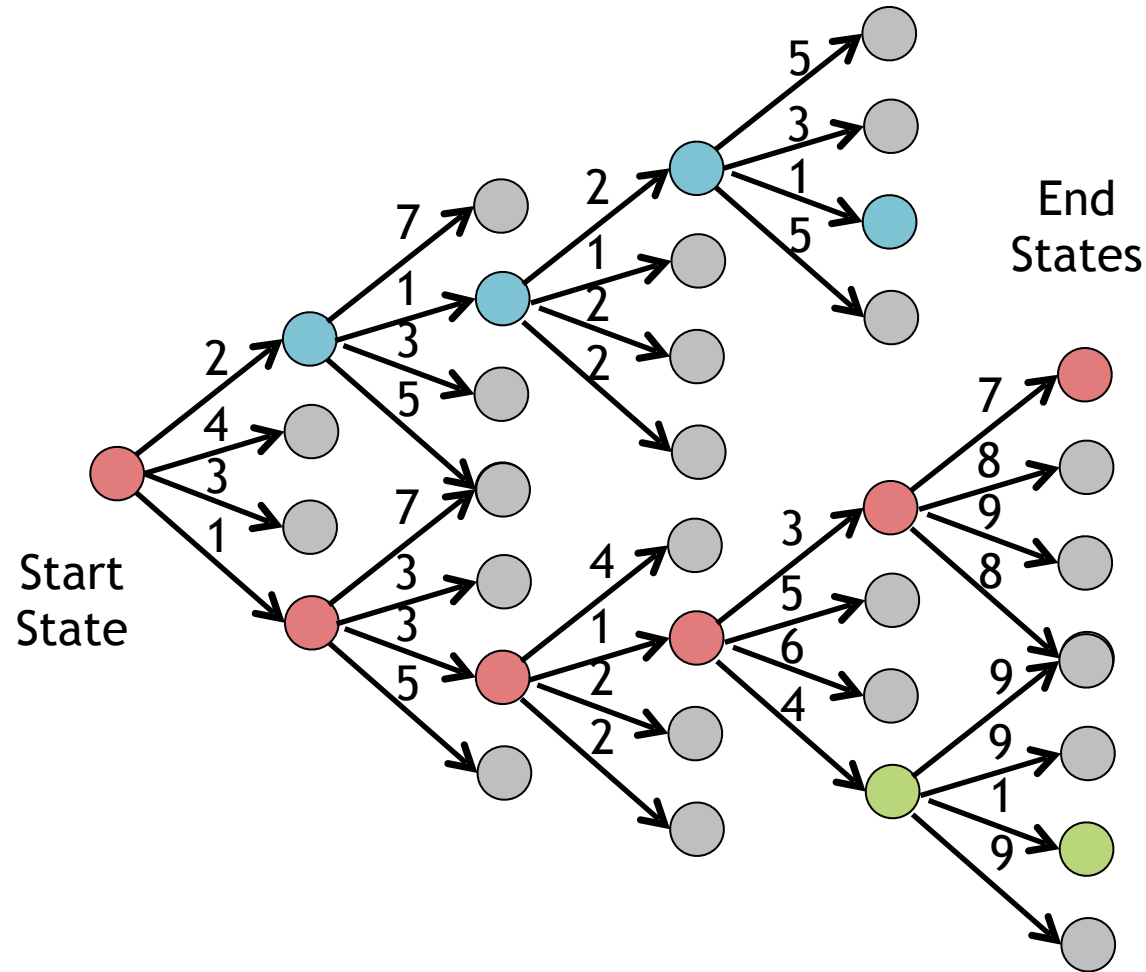
## Goal:

- Search space consists of nodes and weighted edges
- Goal is to find the lowest (total) weight path from root to a leaf

## Greedy Search:

- At each node, selects the edge with lowest (immediate) weight
- **Heuristic** method of search (i.e. does *not* necessarily find the best path)
- Computation time: **linear** in max path length

# Background: Global Search



## Goal:

- Search space consists of nodes and weighted edges
- Goal is to find the lowest (total) weight path from root to a leaf

## Global Search:

- Compute total weight of the path to **every** leaf, pick best
- **Exact** method of search (gauranteed to find the best path)
- Computation time: **exponential** in max path length

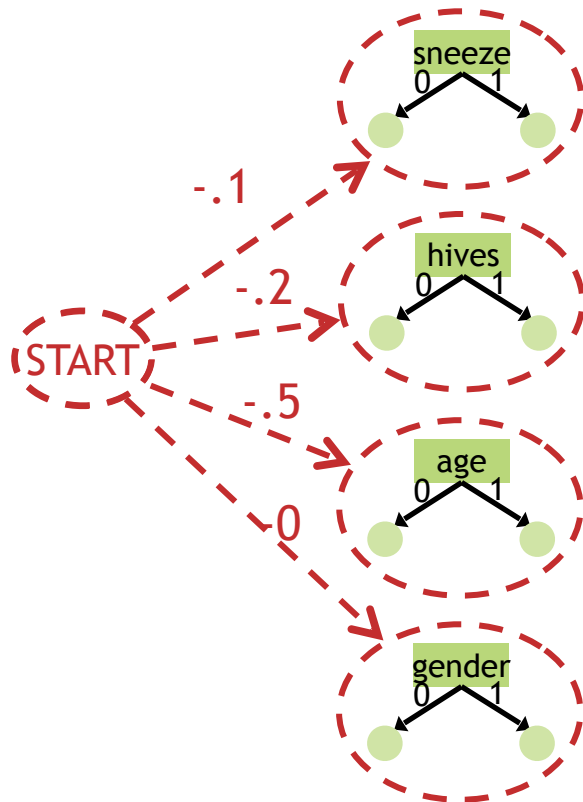
# Decision Tree Learning as Search

1. **search space:** all possible decision trees
2. **node:** single decision tree
3. **edge:** connects one full tree to another, where child has one more split than parent
4. **edge weight:** (negative) splitting criterion
5. **DT learning:** greedy search, maximizing our splitting criterion at each step

(START)

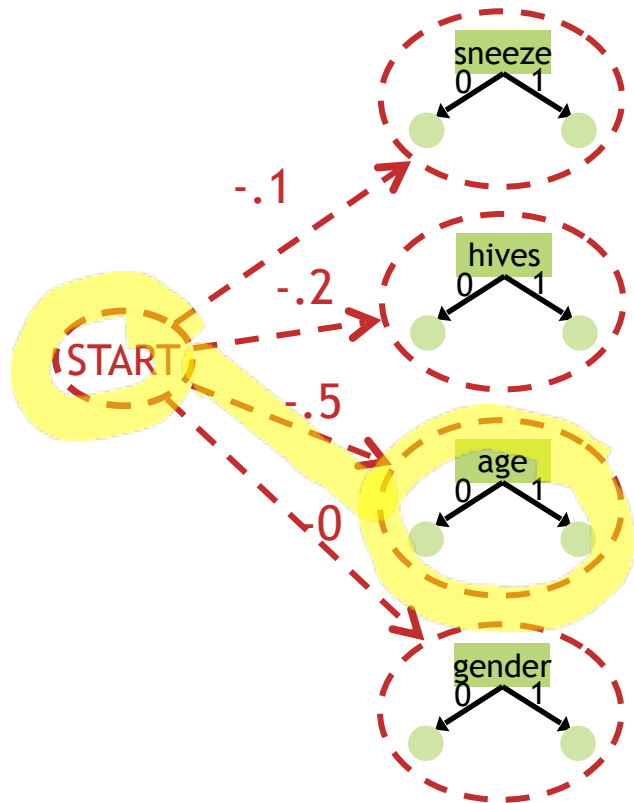
# Decision Tree Learning as Search

1. **search space:** all possible decision trees
2. **node:** single decision tree
3. **edge:** connects one full tree to another, where child has one more split than parent
4. **edge weight:** (negative) splitting criterion
5. **DT learning:** greedy search, maximizing our splitting criterion at each step



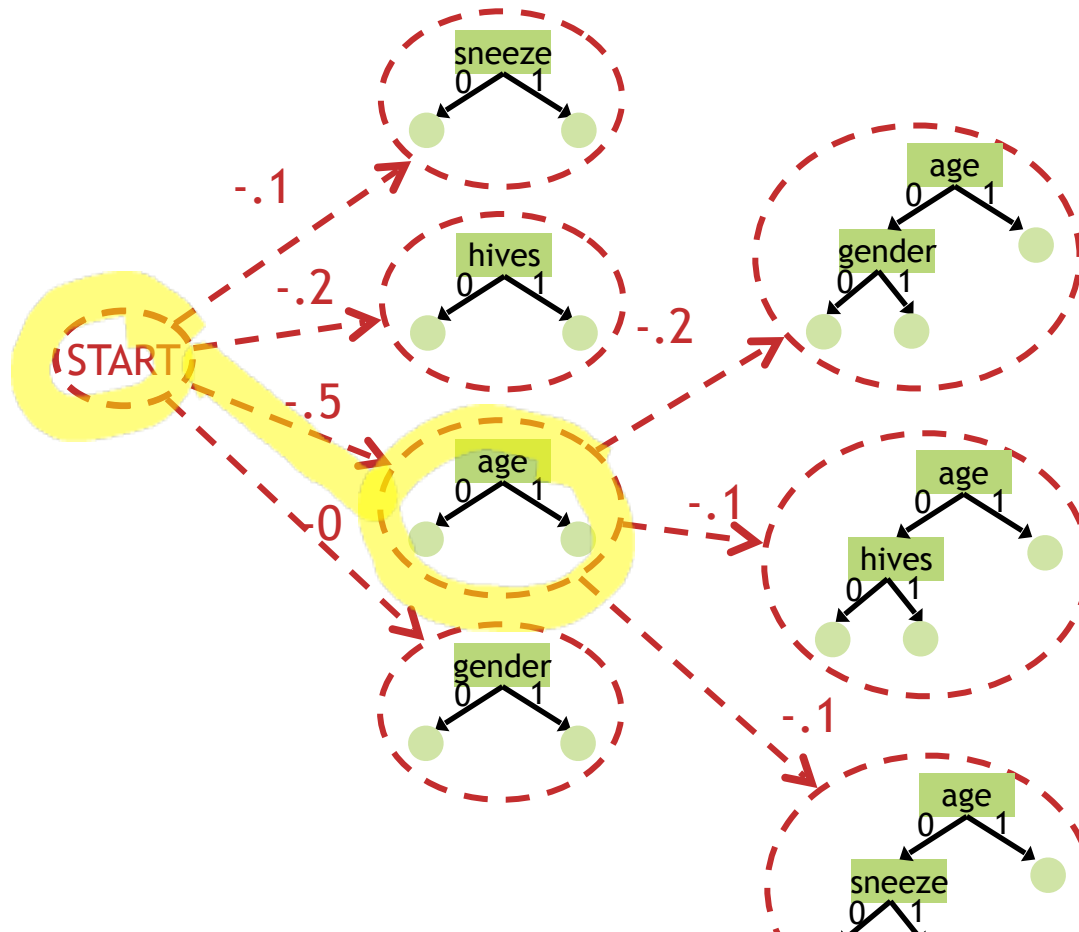
# Decision Tree Learning as Search

1. **search space:** all possible decision trees
2. **node:** single decision tree
3. **edge:** connects one full tree to another, where child has one more split than parent
4. **edge weight:** (negative) splitting criterion
5. **DT learning:** greedy search, maximizing our splitting criterion at each step



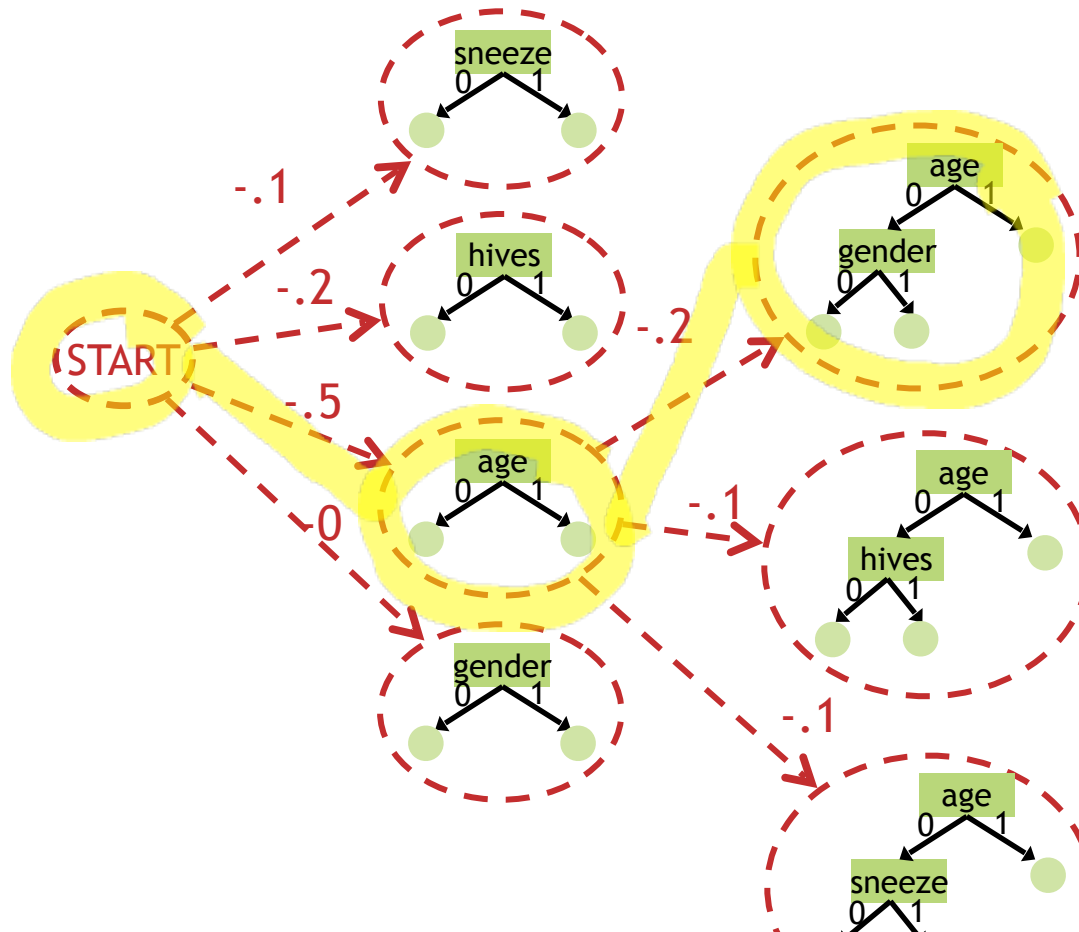
# Decision Tree Learning as Search

1. **search space:** all possible decision trees
2. **node:** single decision tree
3. **edge:** connects one full tree to another, where child has one more split than parent
4. **edge weight:** (negative) splitting criterion
5. **DT learning:** greedy search, maximizing our splitting criterion at each step



# Decision Tree Learning as Search

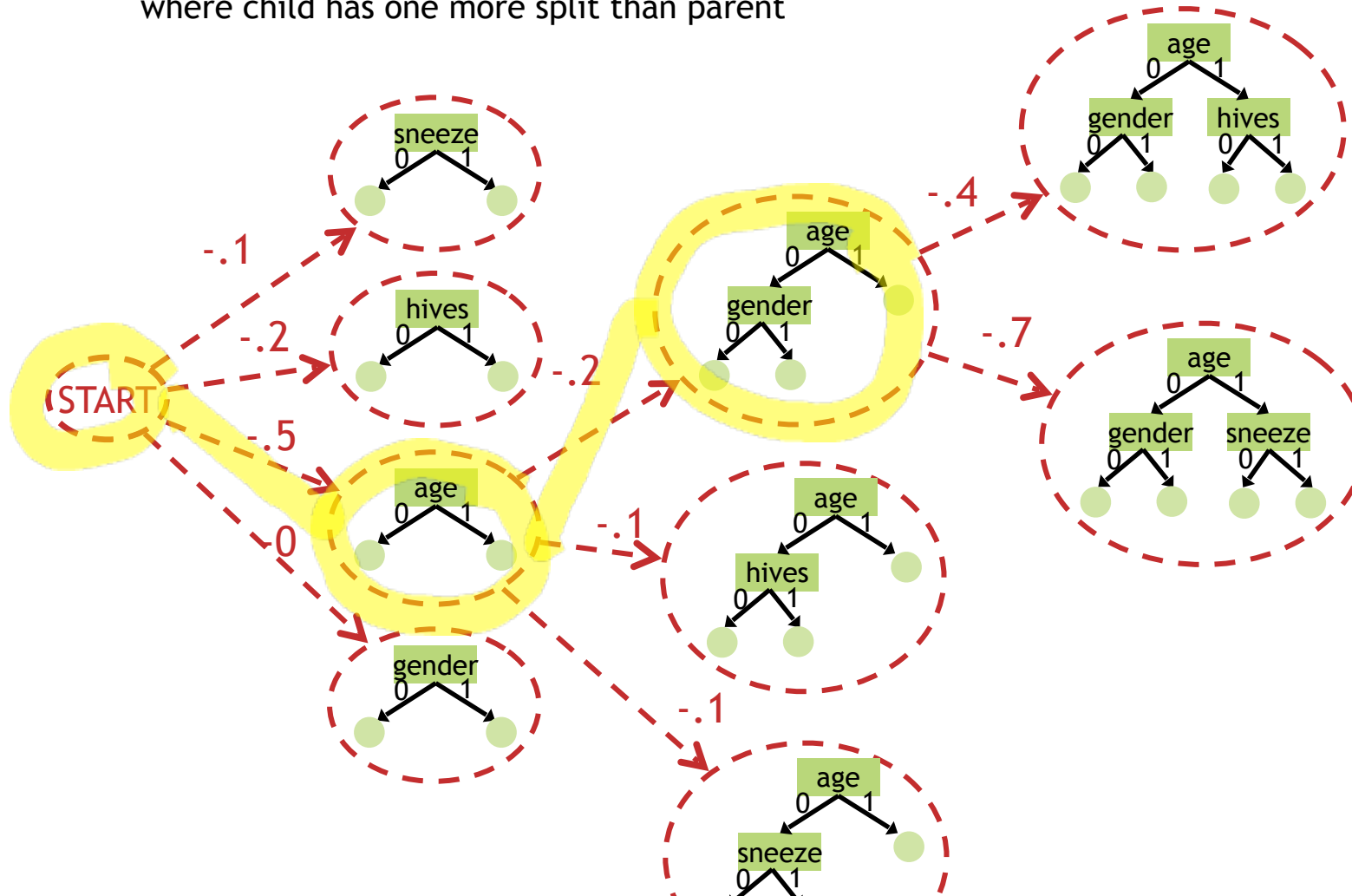
1. **search space:** all possible decision trees
2. **node:** single decision tree
3. **edge:** connects one full tree to another, where child has one more split than parent
4. **edge weight:** (negative) splitting criterion
5. **DT learning:** greedy search, maximizing our splitting criterion at each step





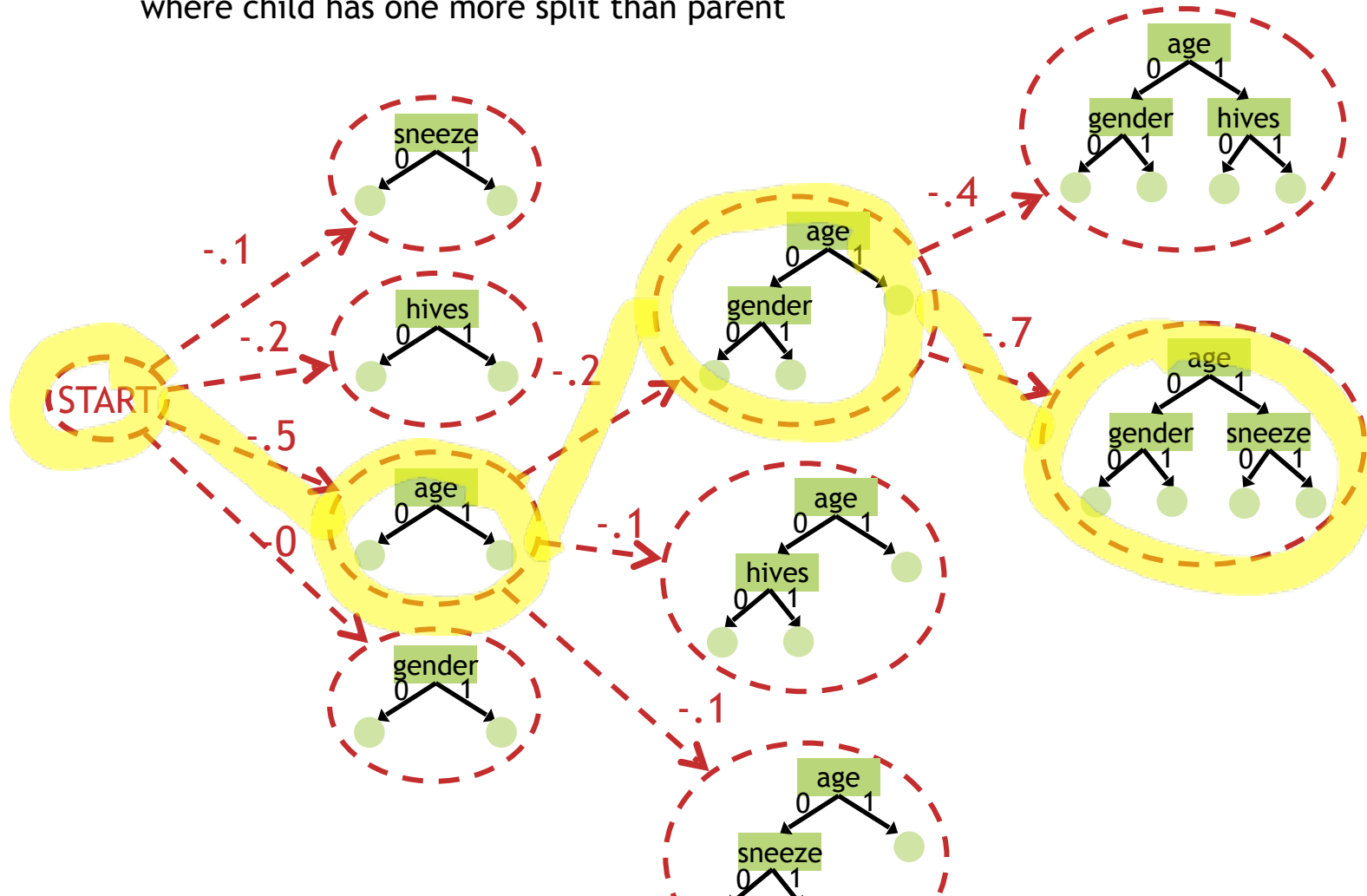
# Decision Tree Learning as Search

1. **search space:** all possible decision trees
2. **node:** single decision tree
3. **edge:** connects one full tree to another, where child has one more split than parent
4. **edge weight:** (negative) splitting criterion
5. **DT learning:** greedy search, maximizing our splitting criterion at each step



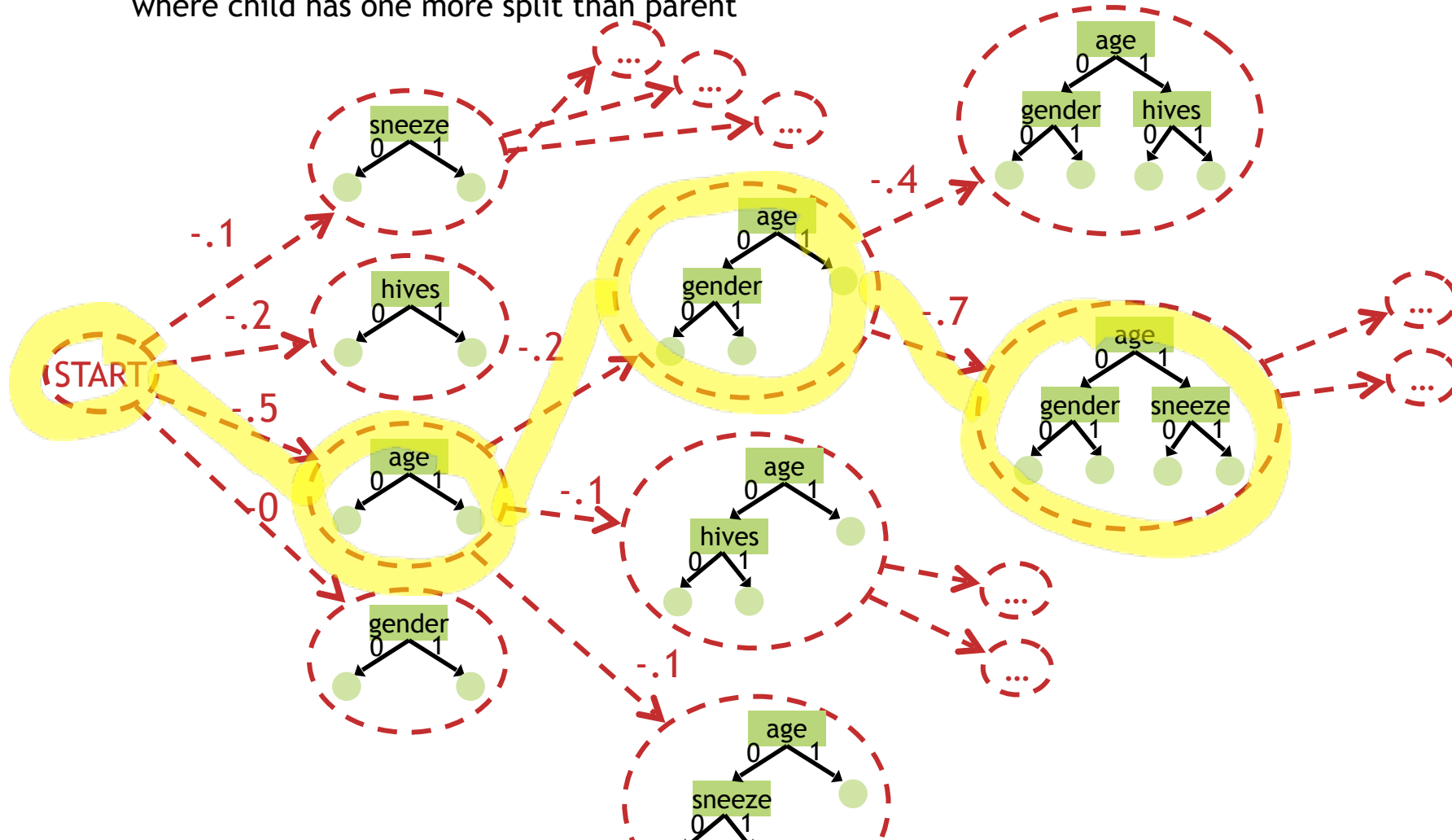
# Decision Tree Learning as Search

1. **search space:** all possible decision trees
2. **node:** single decision tree
3. **edge:** connects one full tree to another, where child has one more split than parent
4. **edge weight:** (negative) splitting criterion
5. **DT learning:** greedy search, maximizing our splitting criterion at each step



# Decision Tree Learning as Search

1. **search space:** all possible decision trees
2. **node:** single decision tree
3. **edge:** connects one full tree to another, where child has one more split than parent
4. **edge weight:** (negative) splitting criterion
5. **DT learning:** greedy search, maximizing our splitting criterion at each step



# Big Question:

# Big Question:

How is it that your  
ML algorithm can  
generalize to  
unseen examples?

# DT: Remarks

ID3 = Decision Tree  
Learning with Mutual  
Information as the  
splitting criterion

**Question:** Which tree does ID3 find?

# DT: Remarks

ID3 = Decision Tree  
Learning with Mutual  
Information as the  
splitting criterion

**Question:** Which tree does ID3 find?

## **Definition:**

We say that the **inductive bias** of a machine learning algorithm is the principal by which it generalizes to unseen examples

# DT: Remarks

ID3 = Decision Tree  
Learning with Mutual  
Information as the  
splitting criterion

**Question:** Which tree does ID3 find?

## **Definition:**

We say that the **inductive bias** of a machine learning algorithm is the principal by which it generalizes to unseen examples

**What is the inductive bias of the ID3 algorithm?**



# DT: Remarks

ID3 = Decision Tree  
Learning with Mutual  
Information as the  
splitting criterion

**Question:** Which tree does ID3 find?

## **Definition:**

We say that the **inductive bias** of a machine learning algorithm is the principal by which it generalizes to unseen examples

**What is the inductive bias of the ID3 algorithm?**

# DT: Remarks

ID3 = Decision Tree  
Learning with Mutual  
Information as the  
splitting criterion

**Question:** Which tree does ID3 find?

## **Definition:**

We say that the **inductive bias** of a machine learning algorithm is the principal by which it generalizes to unseen examples

## **What is the inductive bias of the ID3 algorithm?**

Greedy search for the smallest tree that matches the data with high mutual information attributes near the top

# DT: Remarks

ID3 = Decision Tree  
Learning with Mutual  
Information as the  
splitting criterion

**Question:** Which tree does ID3 find?

## **Definition:**

We say that the **inductive bias** of a machine learning algorithm is the principal by which it generalizes to unseen examples

## **What is the inductive bias of the ID3 algorithm?**

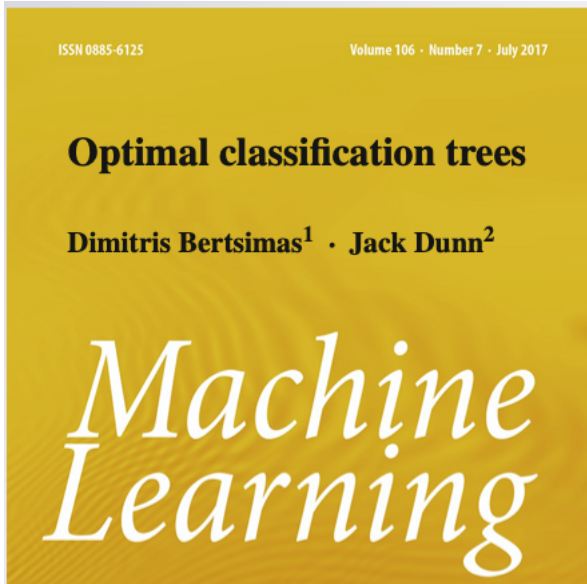
Greedy search for the smallest tree that matches the data with high mutual information attributes near the top

## **Occam's Razor: (restated for ML)**

Prefer the simplest hypothesis that explains the data

# Greedy vs. global search

- 1980s AI researchers: of *course* we can't do global search, that would take exponential time
- 2010s AI researchers: let's try it...



Max. depth	Average accuracy	
	CART (%)	OCT (%)
1	71.0	71.3
2	76.2	78.0
3	78.8	79.5
4	79.8	80.4

## Optimal Classification Tree

Global search for best tree  
(with limits on depth,  
smallest node size)

Our computers are faster  
now, but we still probably  
can't do this for huge trees  
on huge datasets

# **OVERFITTING (FOR DECISION TREES)**

# Overfitting and Underfitting

## Underfitting

- The model...
  - is too simple
  - is unable to capture trends in the data
  - we say: too strong an inductive bias
- *Example:* majority-vote classifier (depth-zero decision tree)
- *Example:* a toddler (that has **not** attended medical school) attempting to do medical diagnosis

## Overfitting

- The model...
  - is too complex
  - fits the noise in the data or “outliers”
  - we say: does not have enough bias
- *Example:* our “memorizer” algorithm responding to an irrelevant attribute
- *Example:* medical student who simply memorizes patient case studies, but does not understand how to apply knowledge to new patients

# Overfitting

- Given a hypothesis  $h$ , its...
  - ...error rate over all training data:  $\text{error}(h, D_{\text{train}})$
  - ...error rate over all test data:  $\text{error}(h, D_{\text{test}})$
  - ...true error over all data:  $\text{error}_{\text{true}}(h)$


# Overfitting

- Given a hypothesis  $h$ , its...
  - ...error rate over all training data:  $\text{error}(h, D_{\text{train}})$
  - ...error rate over all test data:  $\text{error}(h, D_{\text{test}})$
  - ...true error over all data:  $\text{error}_{\text{true}}(h)$
- We say  $h$  overfits the training data if...
$$\text{error}_{\text{true}}(h) \gg \text{error}(h, D_{\text{train}})$$
- Amount of overfitting =
$$\text{error}_{\text{true}}(h) - \text{error}(h, D_{\text{train}})$$



# Overfitting


- Given a hypothesis  $h$ , its...
  - ...error rate over all training data:  $\text{error}(h, D_{\text{train}})$
  - ...error rate over all test data:  $\text{error}(h, D_{\text{test}})$
  - ...true error over all data:  $\text{error}_{\text{true}}(h)$
- We say  $h$  overfits the training data if...
$$\text{error}_{\text{true}}(h) \gg \text{error}(h, D_{\text{train}})$$
- Amount of overfitting =
$$\text{error}_{\text{true}}(h) - \text{error}(h, D_{\text{train}})$$



In practice,  
 $\text{error}_{\text{true}}(h)$  is  
unknown

# Overfitting

- Given a hypothesis  $h$ , its...
  - ...error rate over all training data:  $\text{error}(h, D_{\text{train}})$
  - ...error rate over all test data:  $\text{error}(h, D_{\text{test}})$
  - ...true error over all data:  $\text{error}_{\text{true}}(h)$
- We say  $h$  overfits the training data if...
$$\text{error}(h, D_{\text{test}}) \gg \text{error}(h, D_{\text{train}})$$
- Amount of overfitting =
$$\text{error}(h, D_{\text{test}}) - \text{error}(h, D_{\text{train}})$$



In practice,  
 $\text{error}_{\text{true}}(h)$  is  
unknown

# Overfitting in Decision Tree Learning

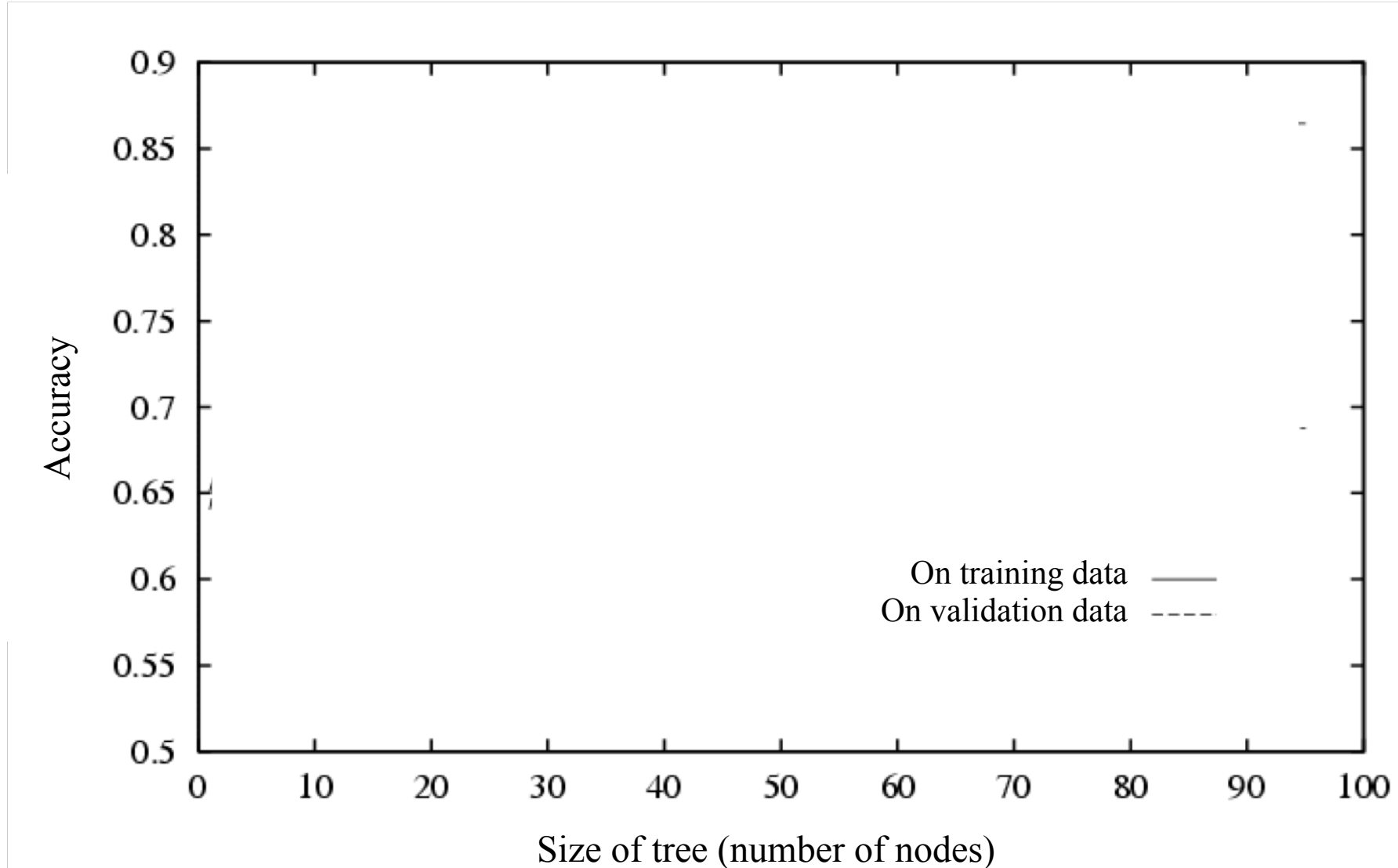


Figure from Tom Mitchell

# Overfitting in Decision Tree Learning

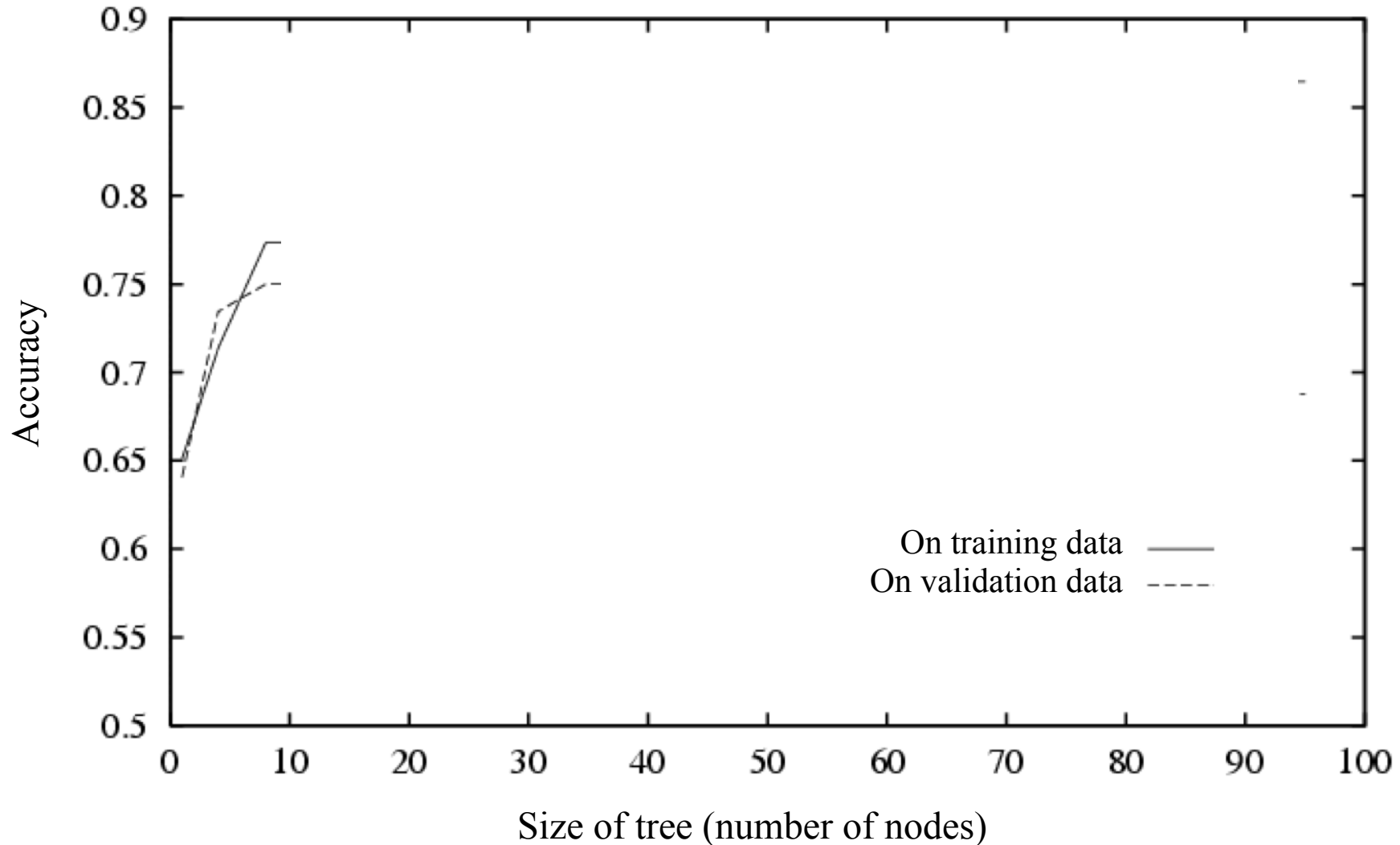


Figure from Tom Mitchell

# Overfitting in Decision Tree Learning

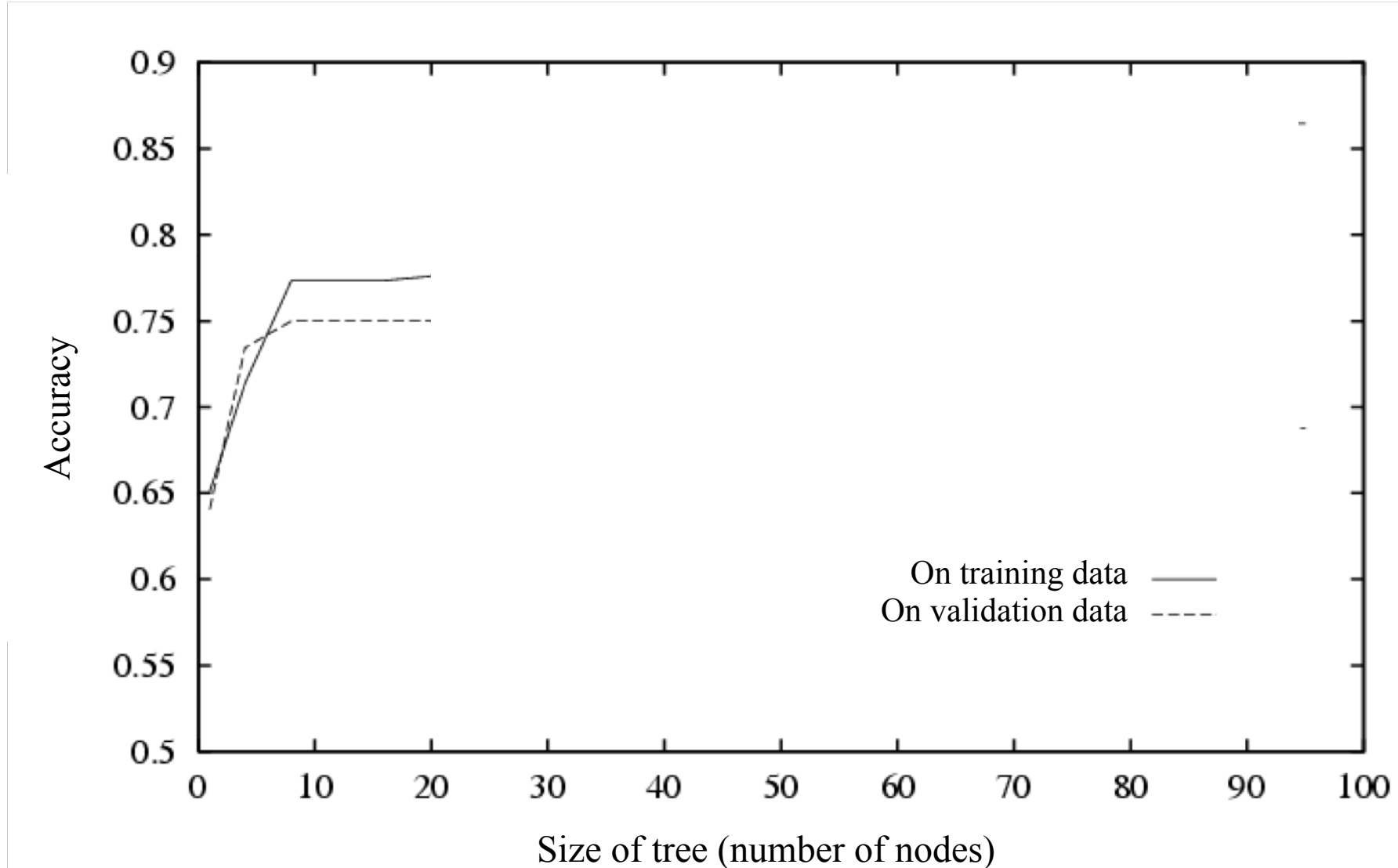


Figure from Tom Mitchell

# Overfitting in Decision Tree Learning

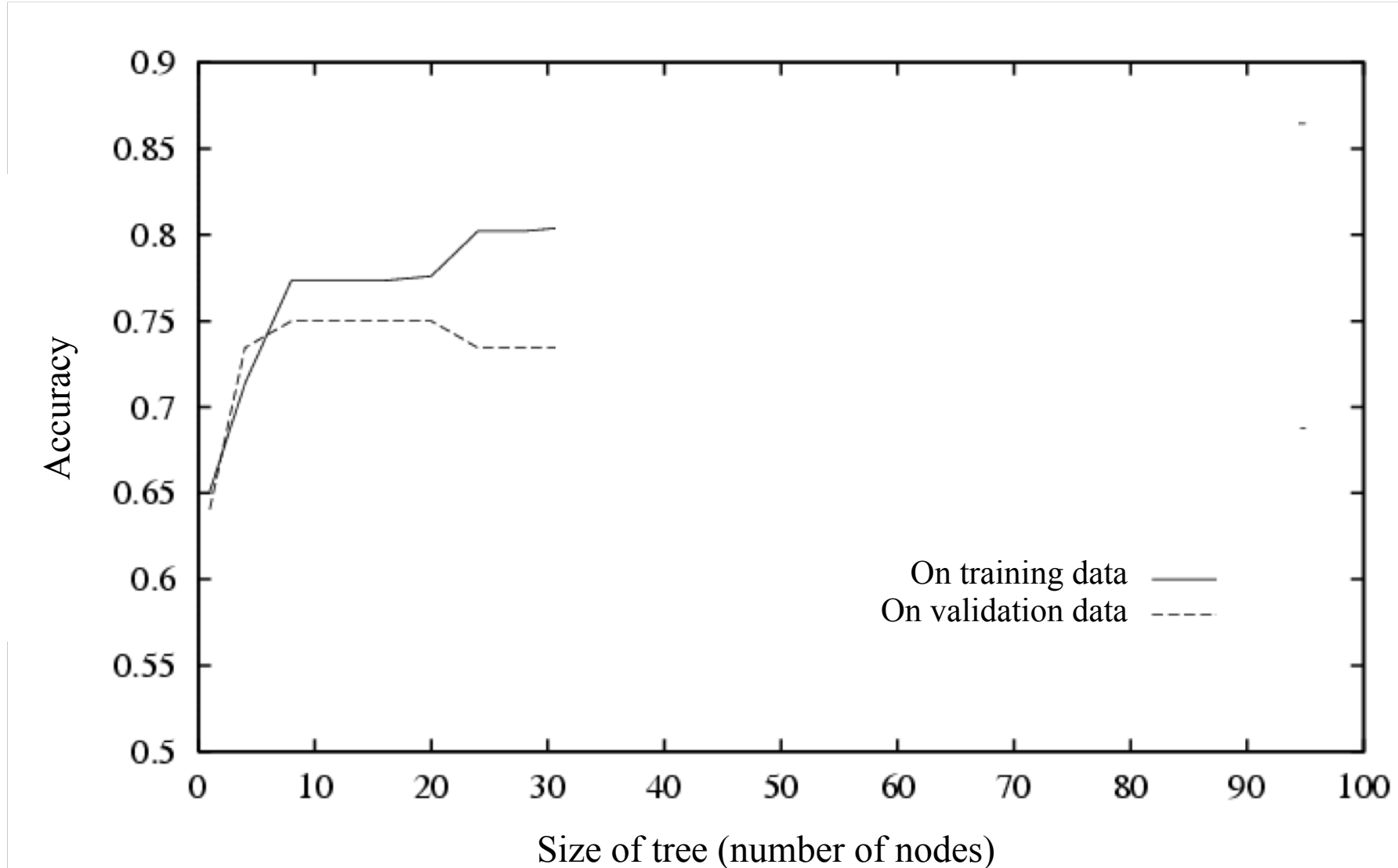


Figure from Tom Mitchell

# Overfitting in Decision Tree Learning

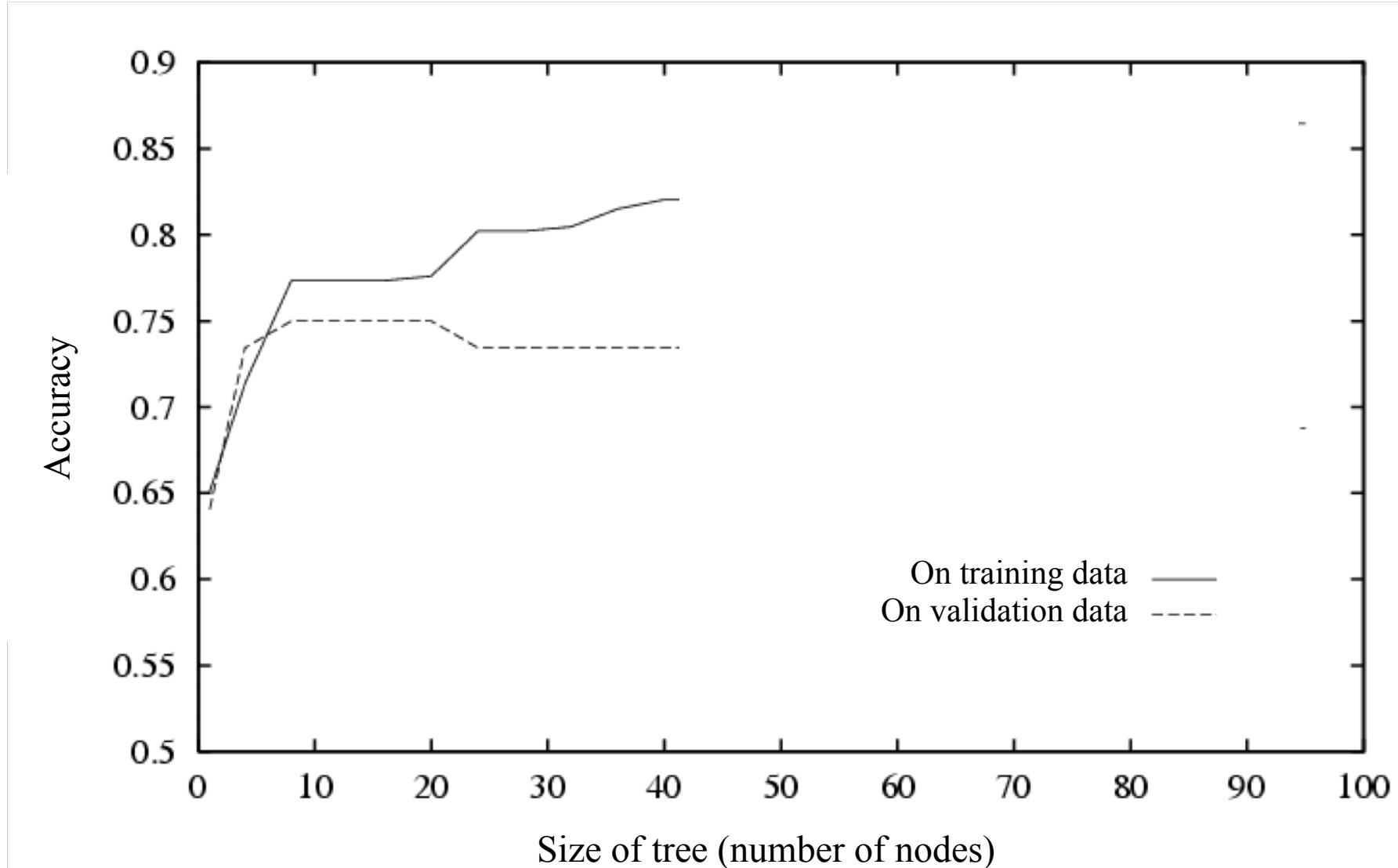


Figure from Tom Mitchell

# Overfitting in Decision Tree Learning

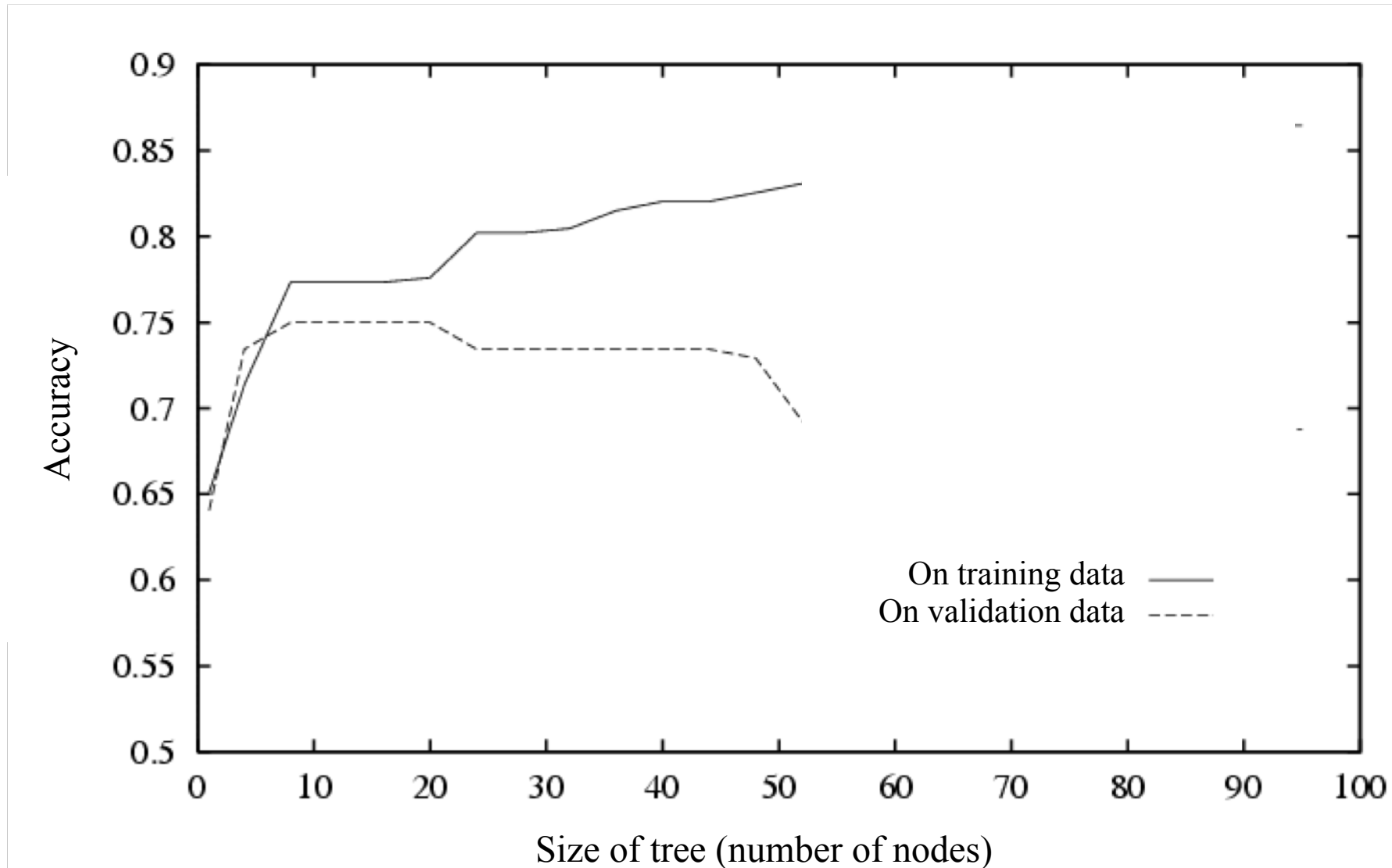


Figure from Tom Mitchell



# Overfitting in Decision Tree Learning

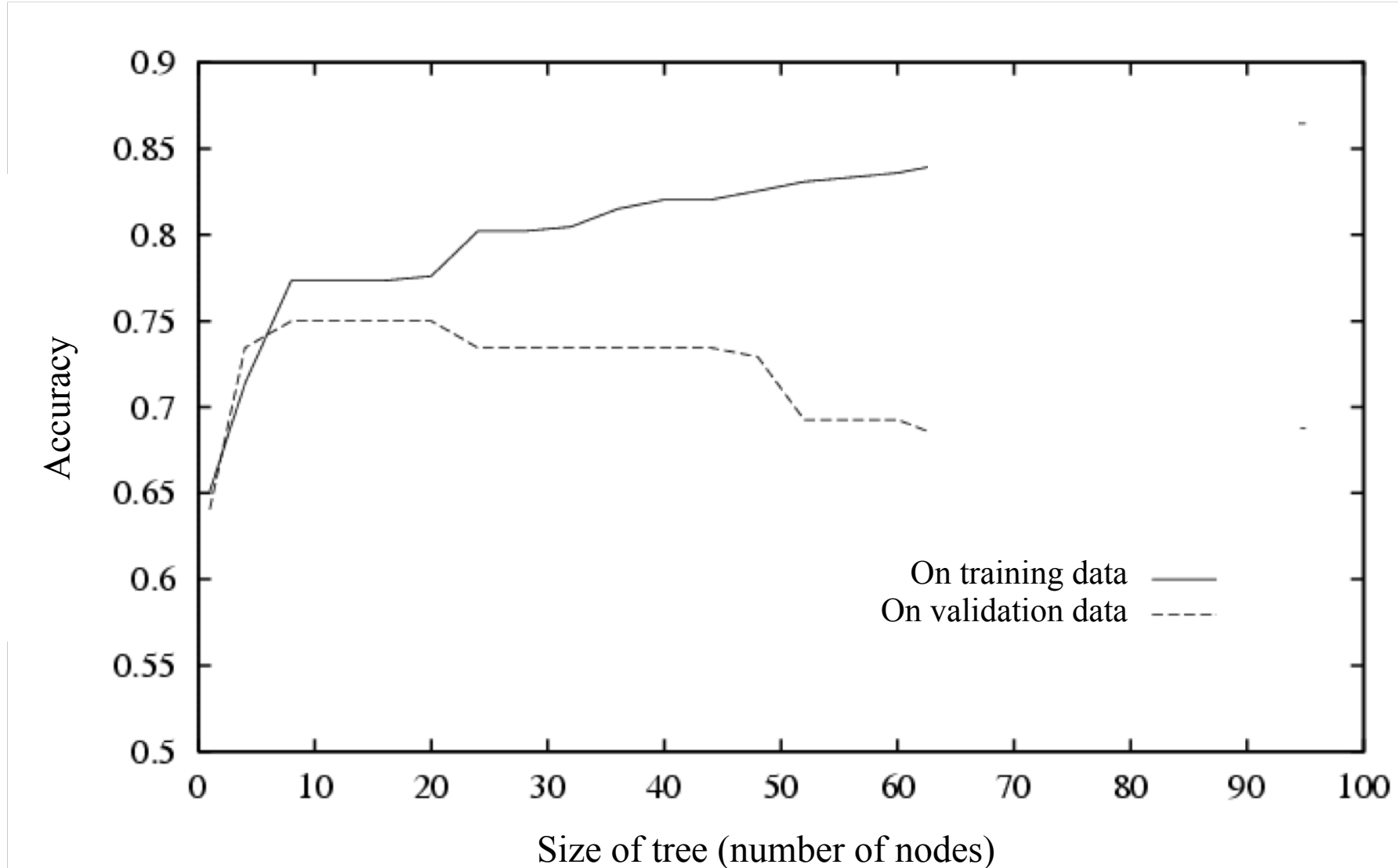


Figure from Tom Mitchell

# Overfitting in Decision Tree Learning

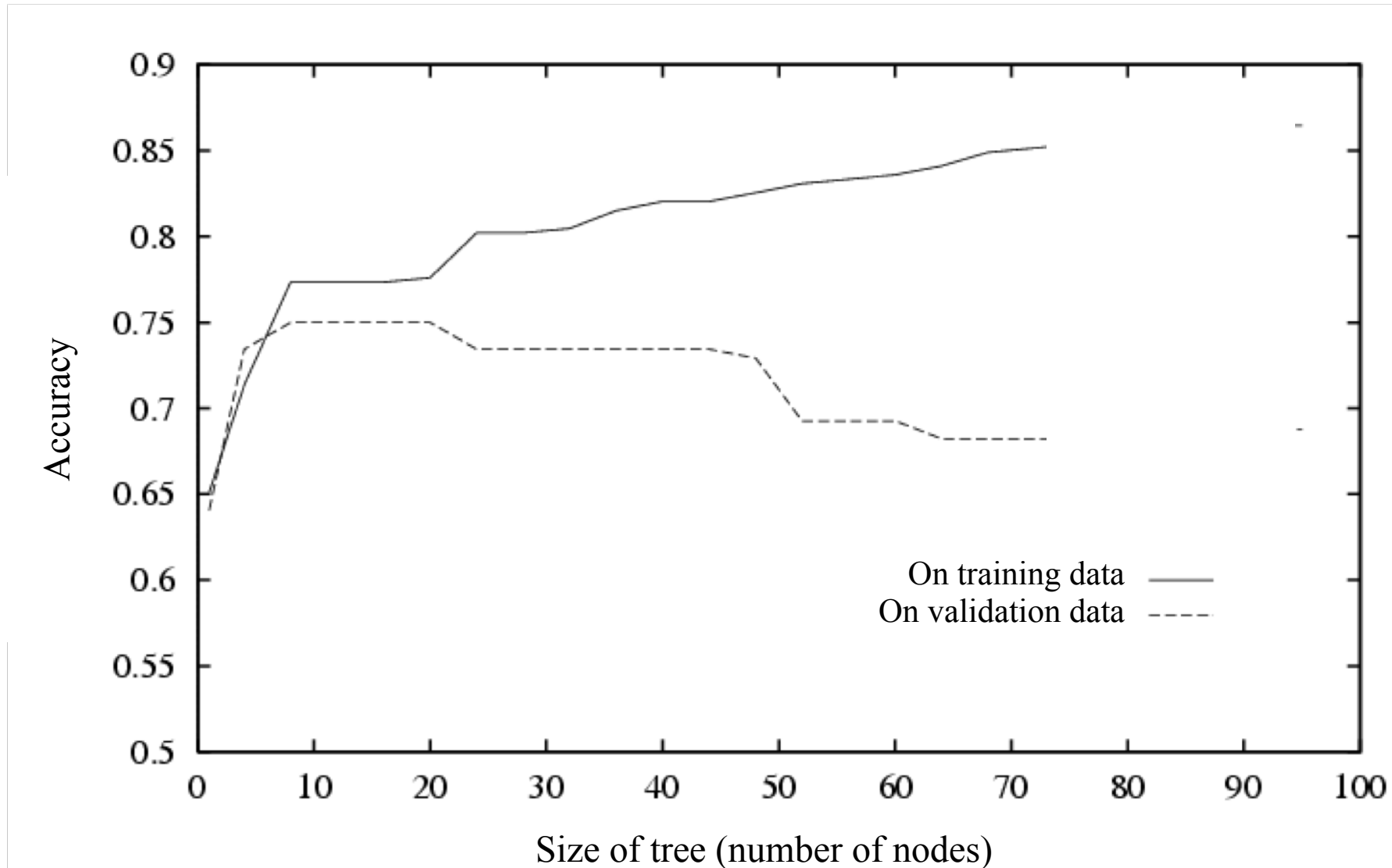


Figure from Tom Mitchell

# Overfitting in Decision Tree Learning

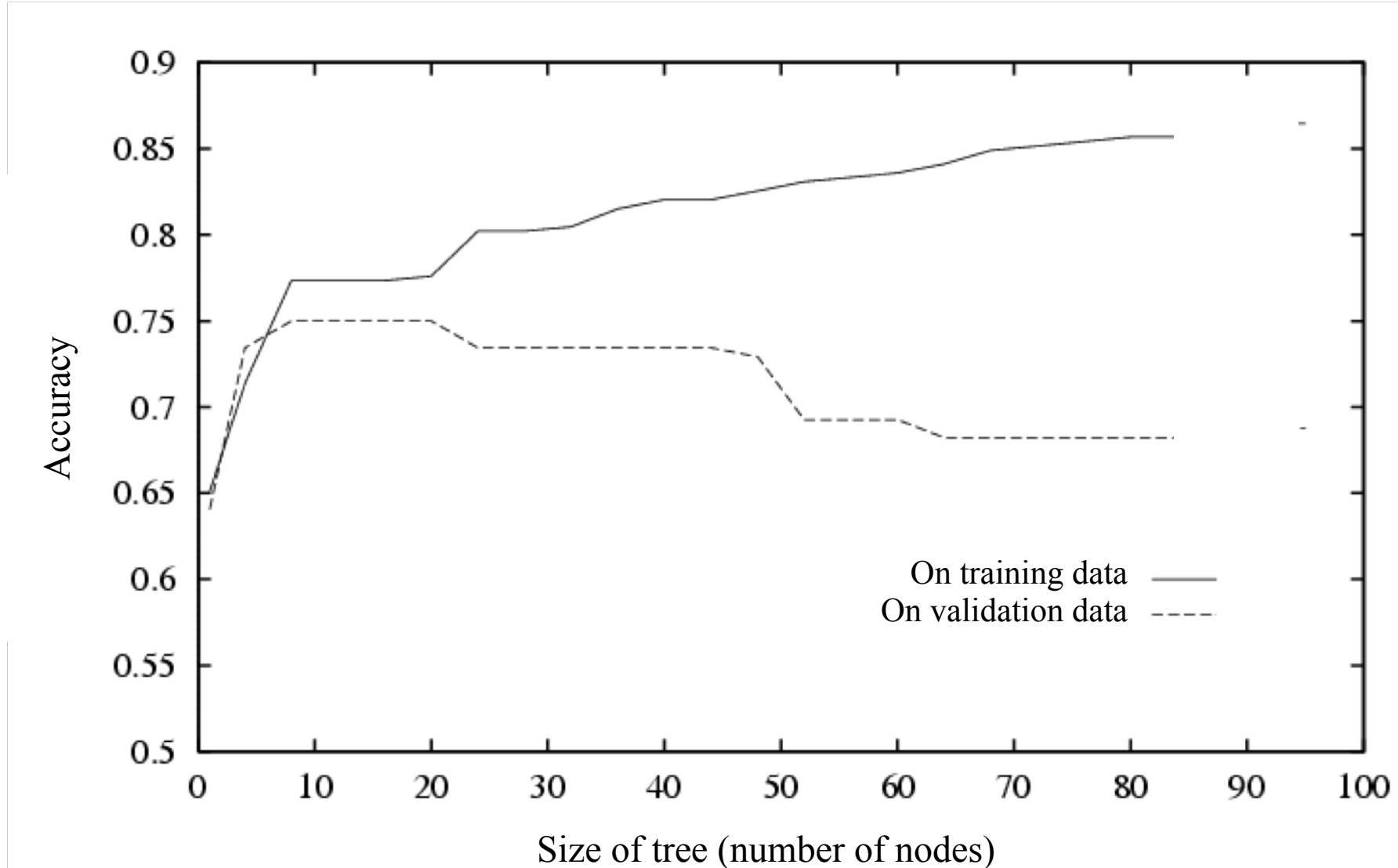


Figure from Tom Mitchell

# Overfitting in Decision Tree Learning

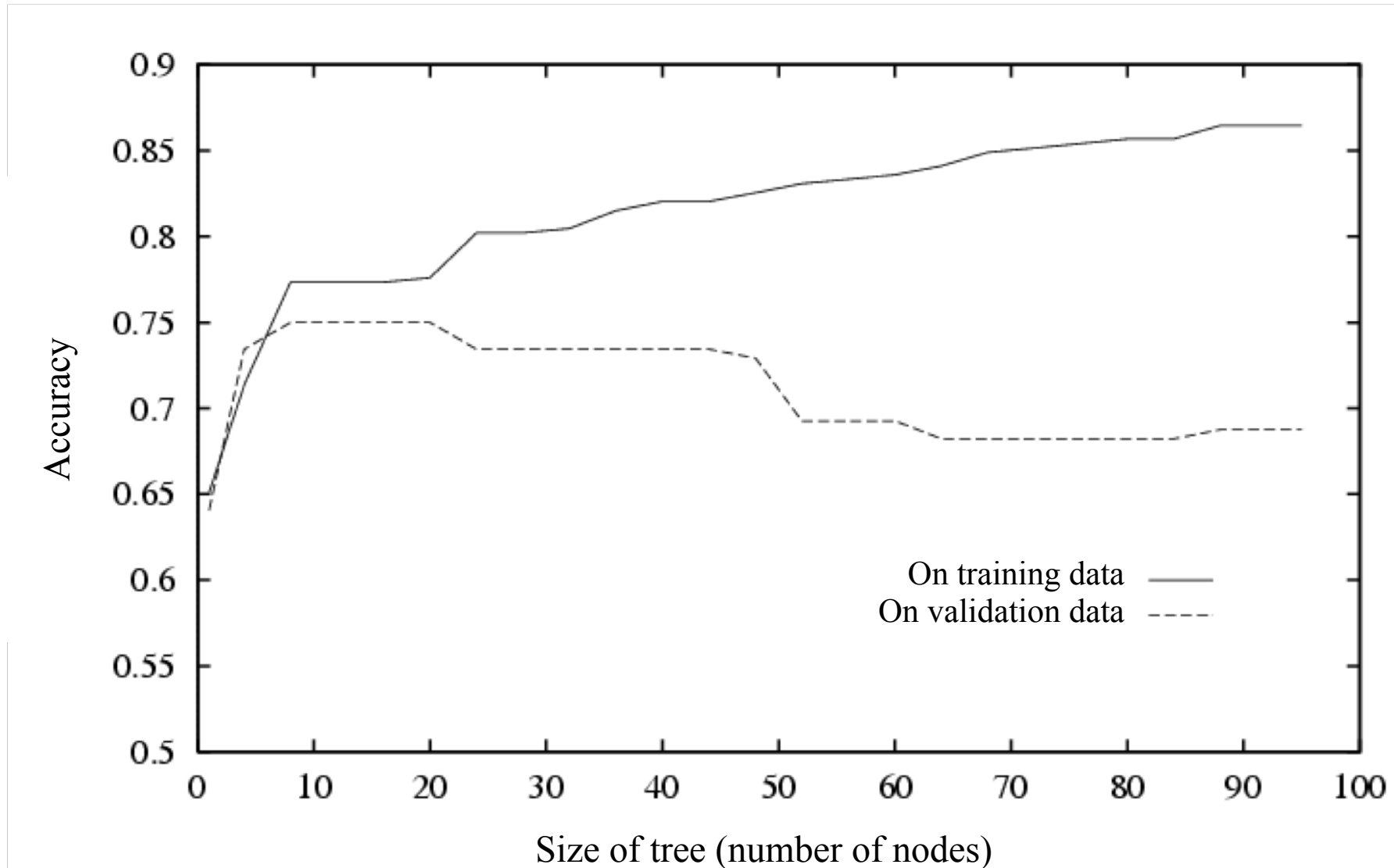


Figure from Tom Mitchell

# Why does overfitting happen?



Expected value of each die = 3.5

# Why does overfitting happen?



Expected value of highest die =  
lower than 3.5?  
the same: still 3.5?  
higher than 3.5?

Expected value of each die = 3.5

# Why does overfitting happen?



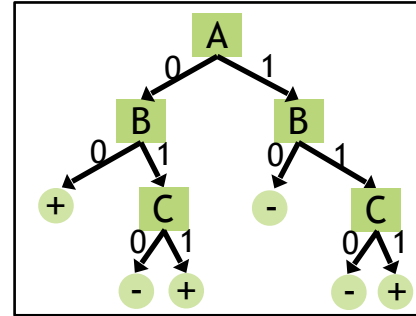
Expected value of each die = 3.5

Expected value of highest die =  
lower than 3.5?  
the same: still 3.5?  
higher than 3.5?

CDF of one die:  $i/6$   
CDF of max of 3 dice:  $(i/6)^3$

# A random experiment

- Fix a decision tree D

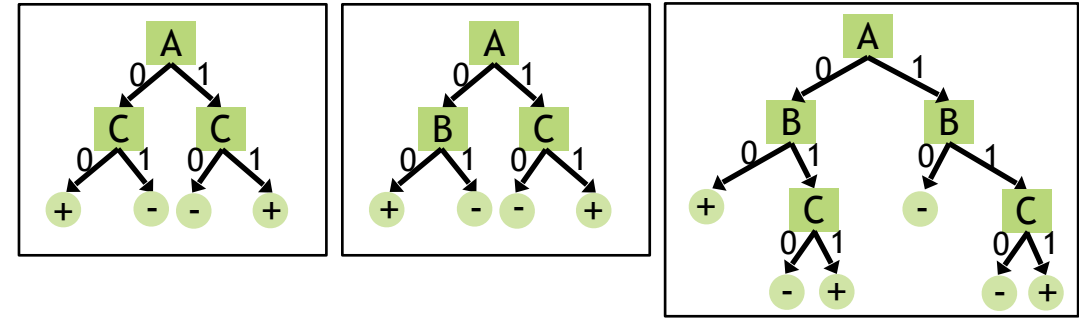


- Pick a random example:  $A = 1, B = 0, C = 1, \text{label} = +$
- Classify it: score 1 if ✓ or 0 if ✗
- Expected value = true accuracy of our tree D
- Same expected value if we pick several examples (a training set) – but lower variance



# A random experiment

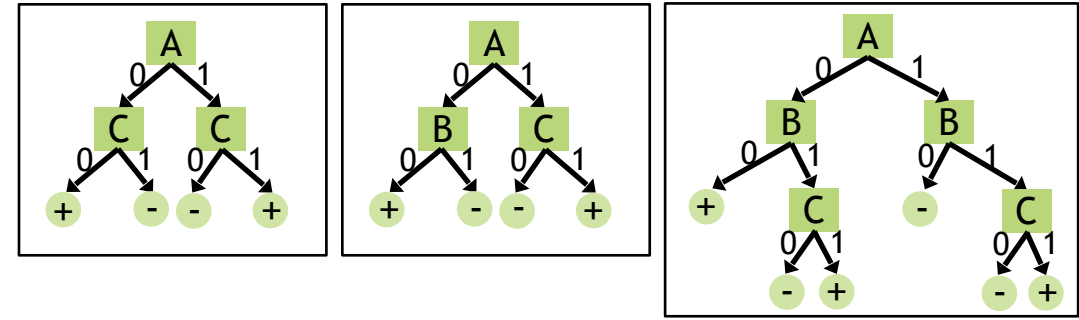
- Fix several trees D1, D2, D3, ...



- Pick random example (or several):  
 $A = 1, B = 0, C = 1, \text{label} = +$
- Classify it with each tree: score 1 if ✓ or 0 if ✗
- Expected values = true accuracies D1, D2, D3

# A random experiment

- Fix several trees D1, D2, D3, ...



- Pick random example (or several):  
 $A = 1, B = 0, C = 1, \text{label} = +$
- Classify it with each tree: score 1 if ✓ or 0 if ✗
- Expected values = true accuracies D1, D2, D3

Now pick the tree with  
highest accuracy (“train”  
from hypothesis set of size 3)

Expected value for winning tree:  
lower than true accuracy?  
equal to true accuracy?  
higher than true accuracy?

# The problem is max

- Any time we optimize on some examples (pick a decision tree structure, train a neural network, ...)
- The *apparent* accuracy of the winner on those examples will be higher than its *true* accuracy
  - “winner’s curse,” “selection bias,” “overfitting”
- If we want to know the true accuracy, need fresh examples (a test set)
  - key is that we didn’t optimize on the fresh examples

# How to Avoid Overfitting?

For Decision Trees...

# How to Avoid Overfitting?

## For Decision Trees...

1. Do not grow tree beyond some **maximum depth** or **minimum node size**
2. Do not split if splitting criterion (e.g. mutual information) is **below some threshold**
3. Stop growing when the split is **not statistically significant**
4. Grow the entire tree, then **prune** (e.g., Reduced Error Pruning)

# Reduced Error Pruning

1. Split data in two: *training* dataset and *validation* dataset
2. Grow the full tree using the *training* dataset
3. Repeatedly prune the tree:
  - Evaluate each split using a *validation* dataset by comparing the validation error rate **with and without** that split
  - (Greedy) remove the split that most decreases the validation error rate
  - Stop if no split improves validation error, otherwise repeat

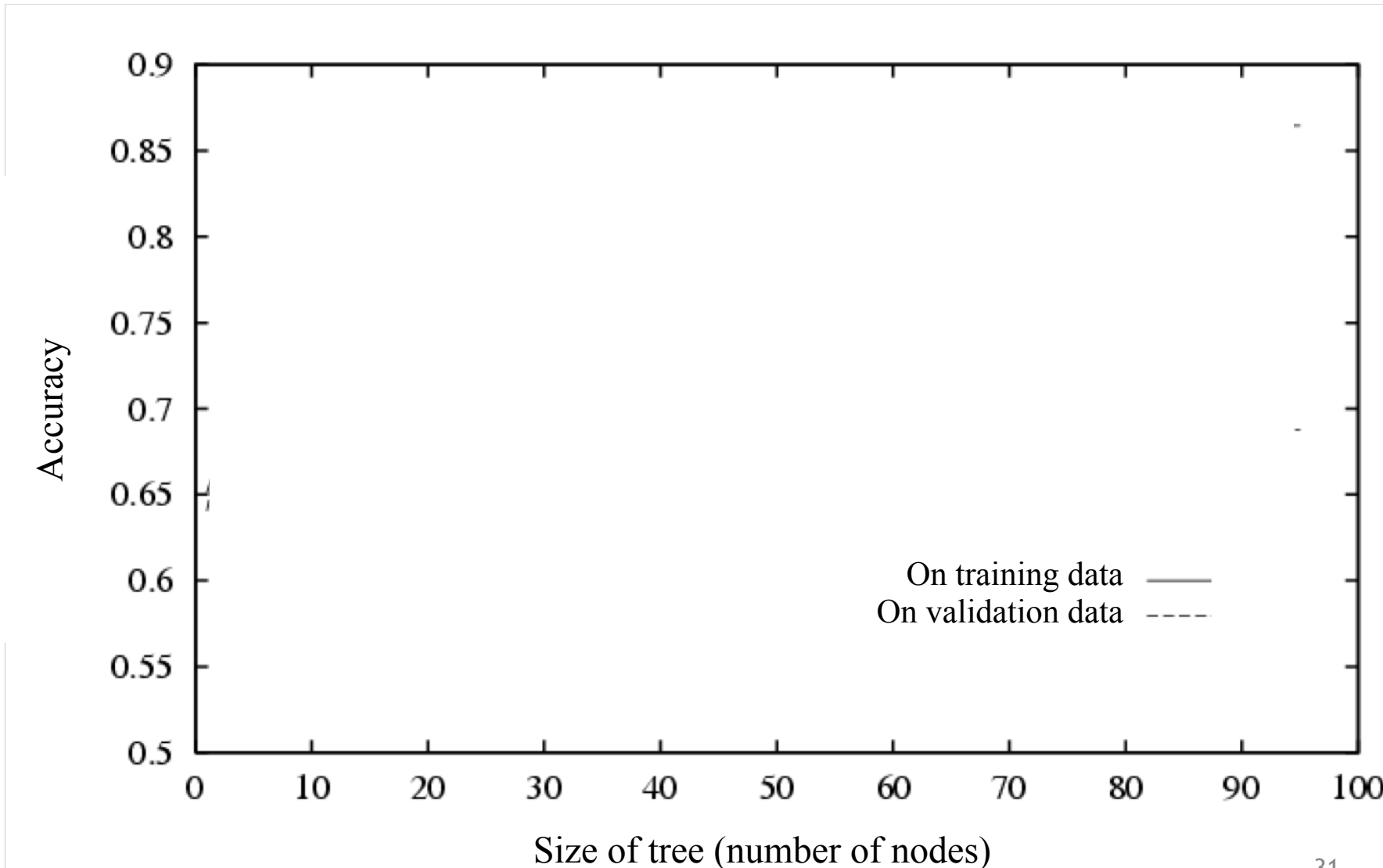


Figure from Tom Mitchell

# Reduced Error Pruning

1. Split data in two: *training* dataset and *validation* dataset
2. Grow the full tree using the *training* dataset
3. Repeatedly prune the tree:
  - Evaluate each split using a *validation* dataset by comparing the validation error rate **with and without** that split
  - (Greedy) remove the split that most decreases the validation error rate
  - Stop if no split improves validation error, otherwise repeat

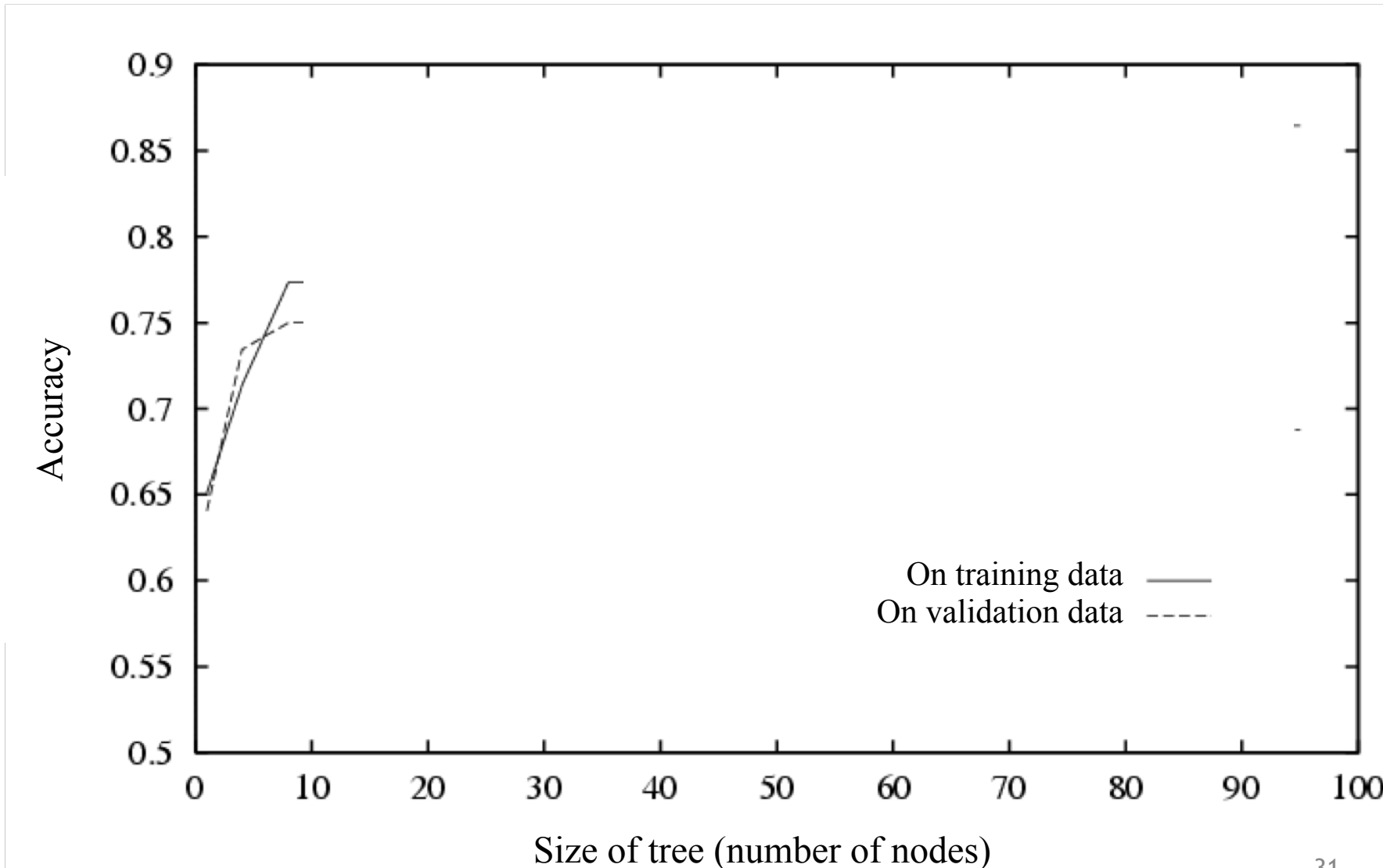


Figure from Tom Mitchell

# Reduced Error Pruning

1. Split data in two: *training* dataset and *validation* dataset
2. Grow the full tree using the *training* dataset
3. Repeatedly prune the tree:
  - Evaluate each split using a *validation* dataset by comparing the validation error rate **with and without** that split
  - (Greedy) remove the split that most decreases the validation error rate
  - Stop if no split improves validation error, otherwise repeat

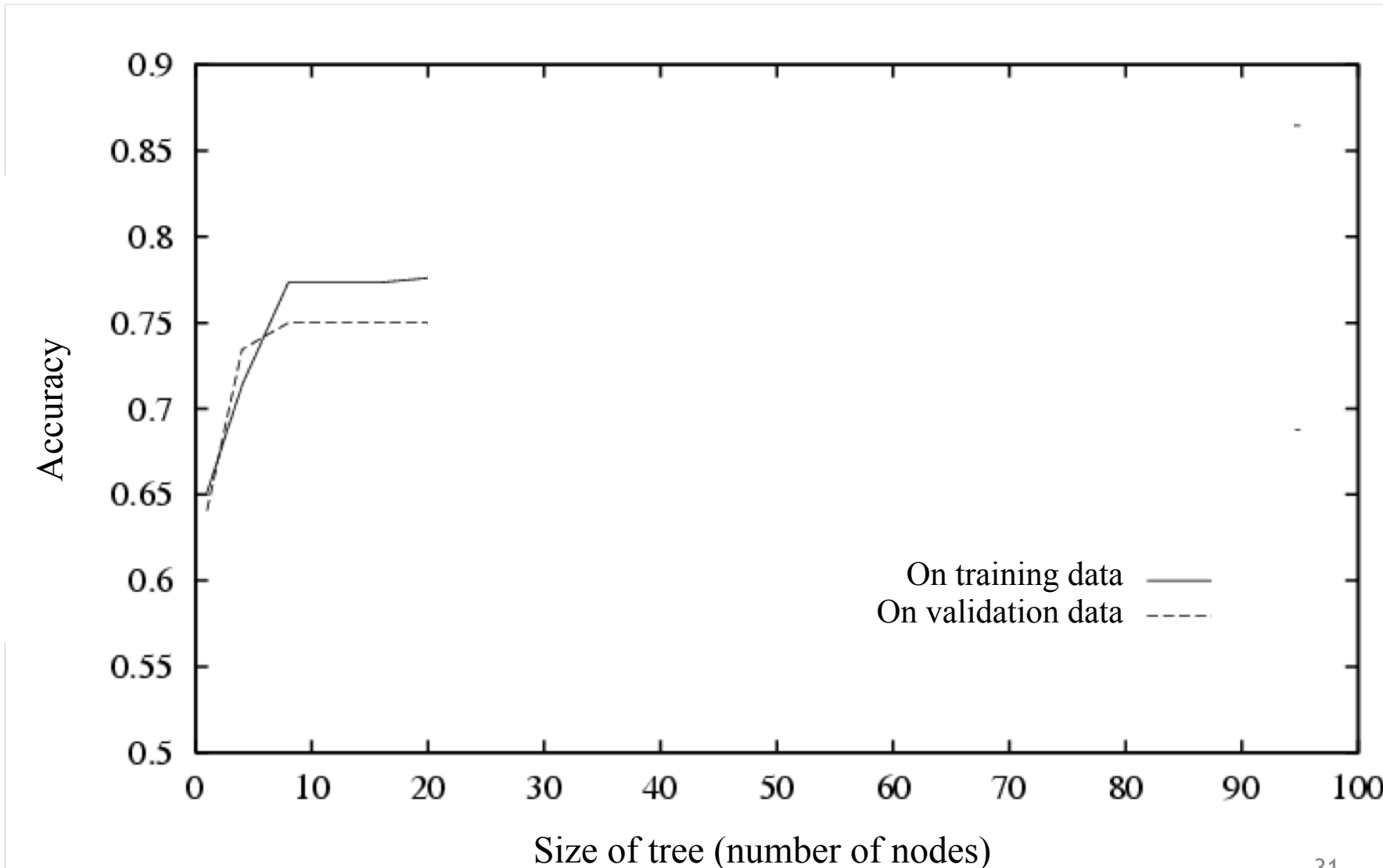


Figure from Tom Mitchell



# Reduced Error Pruning

1. Split data in two:  
*training* dataset and *validation* dataset
2. Grow the full tree using the *training* dataset
3. Repeatedly prune the tree:
  - Evaluate each split using a *validation* dataset by comparing the validation error rate **with and without** that split
  - (Greedy) remove the split that most decreases the validation error rate
  - Stop if no split improves validation error, otherwise repeat

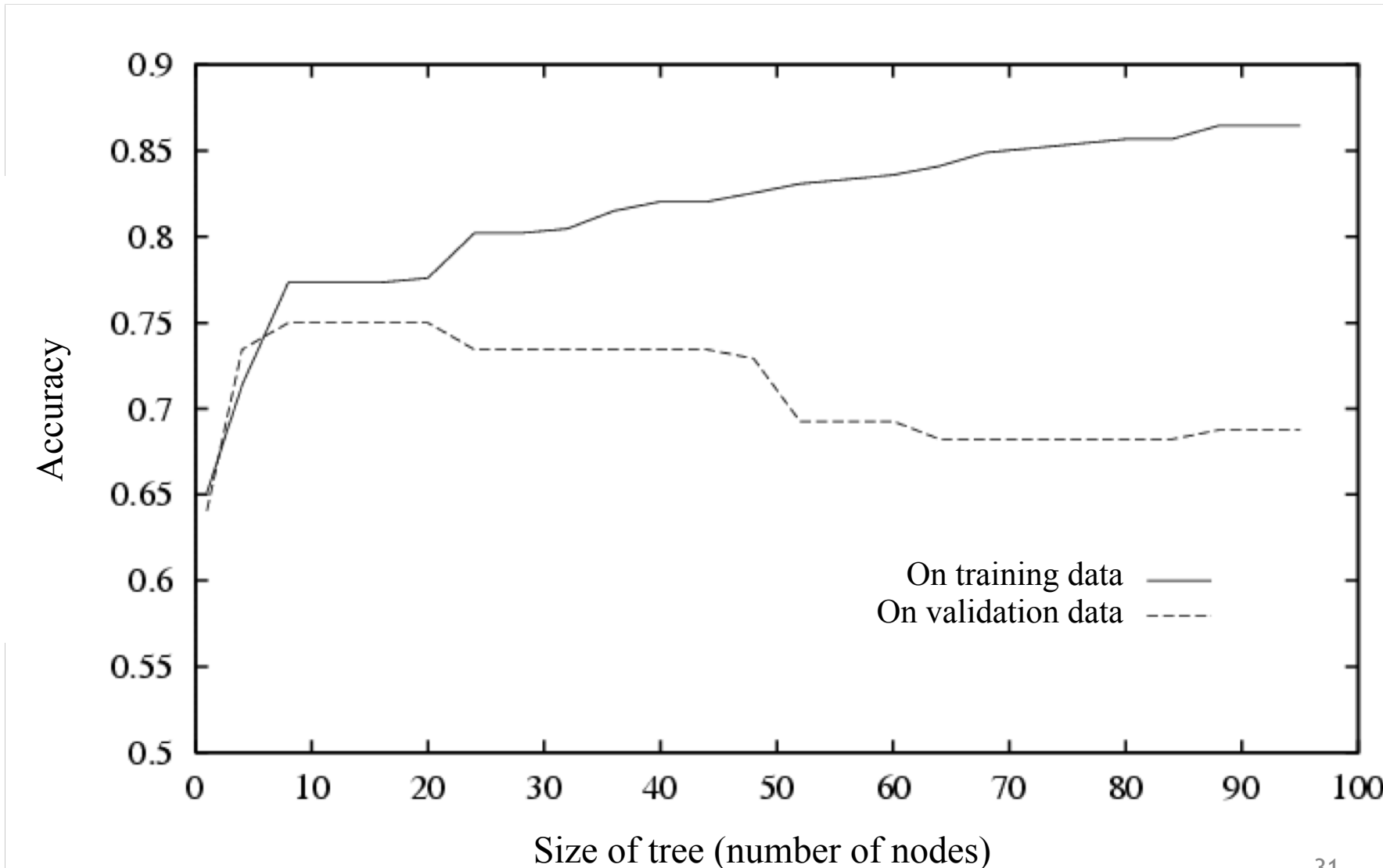


Figure from Tom Mitchell

# Reduced Error Pruning

1. Split data in two:  
*training* dataset and *validation* dataset
2. Grow the full tree using the *training* dataset
3. Repeatedly prune the tree:
  - Evaluate each split using a *validation* dataset by comparing the validation error rate **with and without** that split
  - (Greedy) remove the split that most decreases the validation error rate
  - Stop if no split improves validation error, otherwise repeat

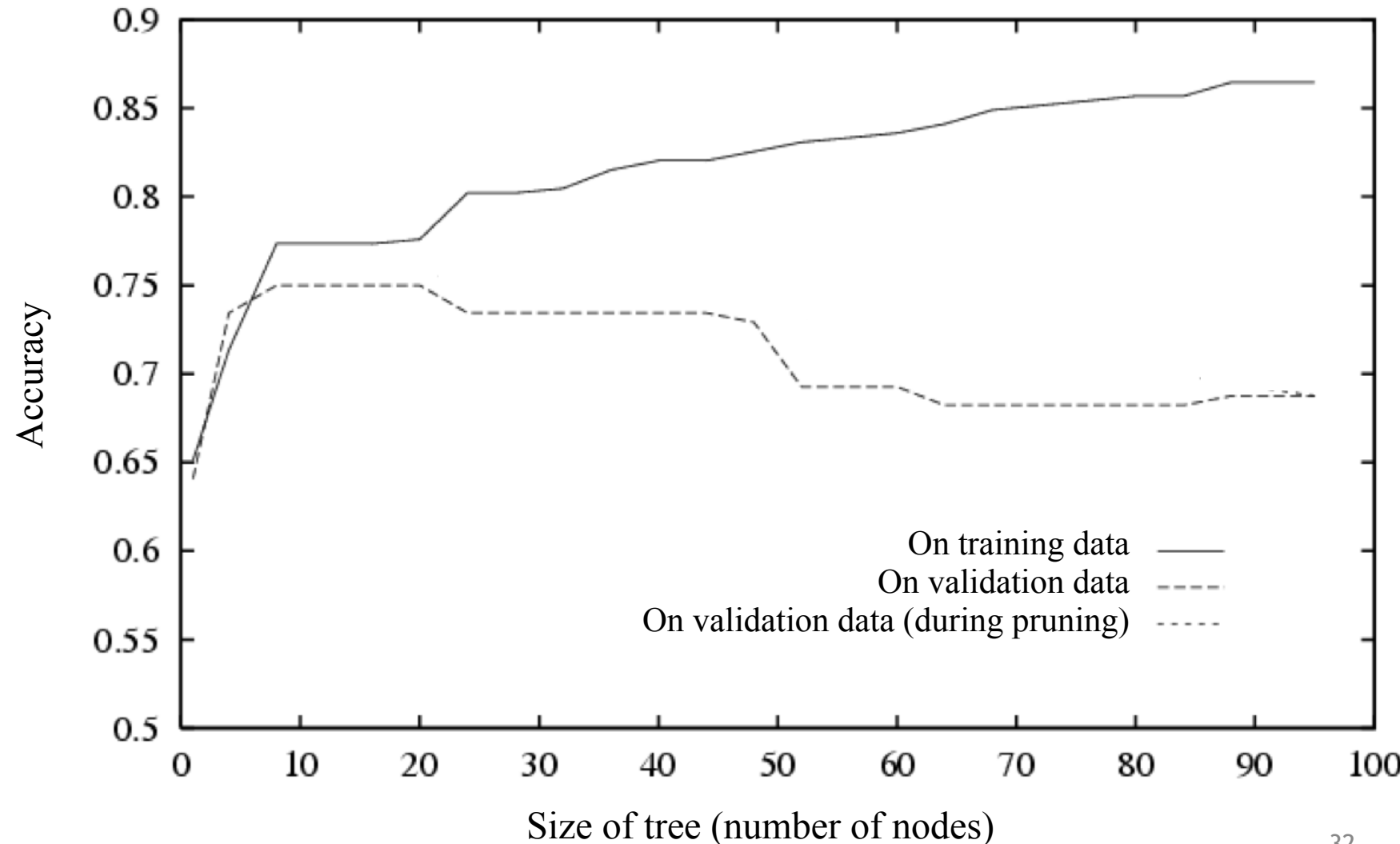


Figure from Tom Mitchell

# Reduced Error Pruning

1. Split data in two:  
*training* dataset and *validation* dataset
2. Grow the full tree using the *training* dataset
3. Repeatedly prune the tree:
  - Evaluate each split using a *validation* dataset by comparing the validation error rate **with and without** that split
  - (Greedy) remove the split that most decreases the validation error rate
  - Stop if no split improves validation error, otherwise repeat

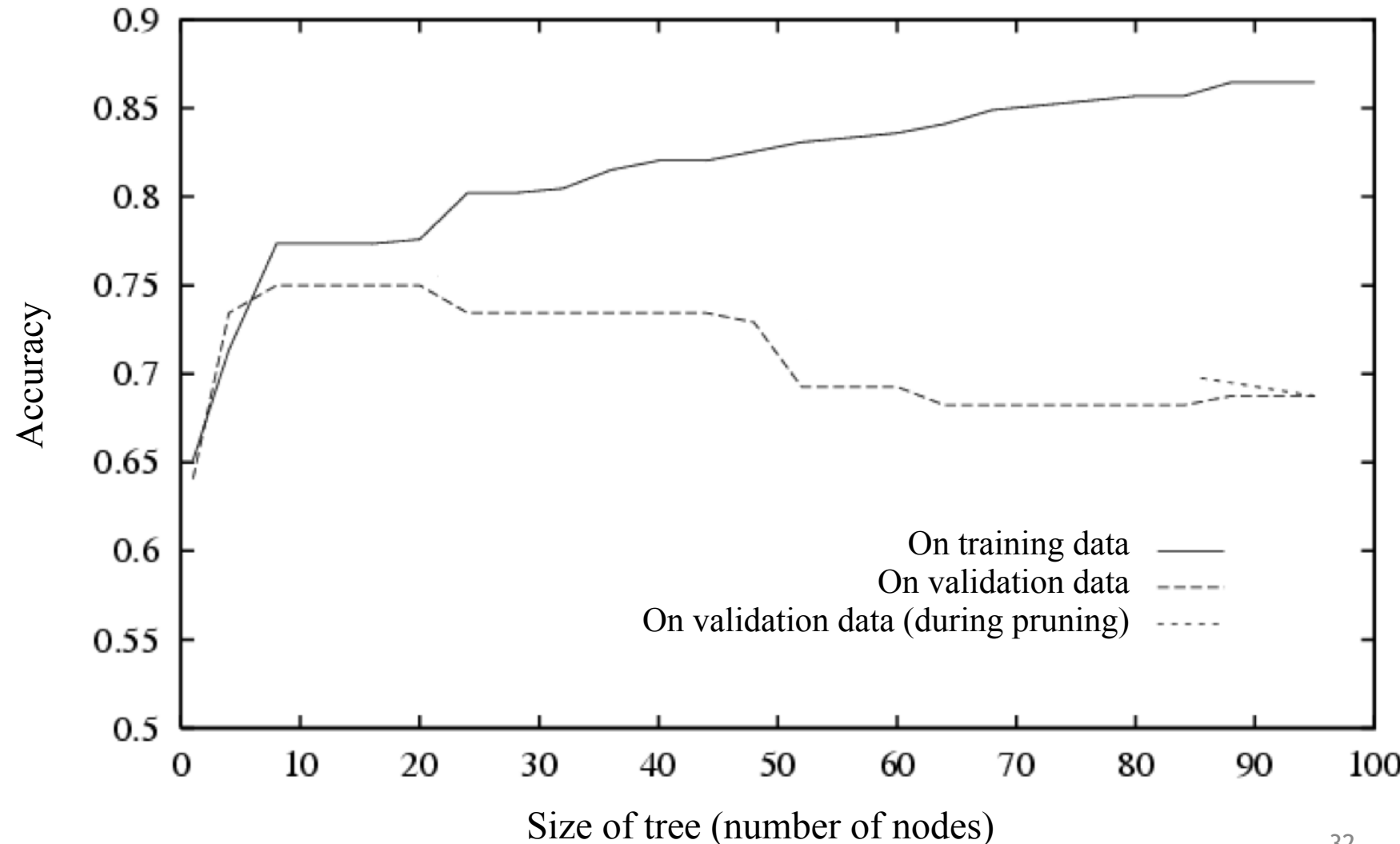


Figure from Tom Mitchell

# Reduced Error Pruning

1. Split data in two:  
*training* dataset and *validation* dataset
2. Grow the full tree using the *training* dataset
3. Repeatedly prune the tree:
  - Evaluate each split using a *validation* dataset by comparing the validation error rate **with and without** that split
  - (Greedy) remove the split that most decreases the validation error rate
  - Stop if no split improves validation error, otherwise repeat

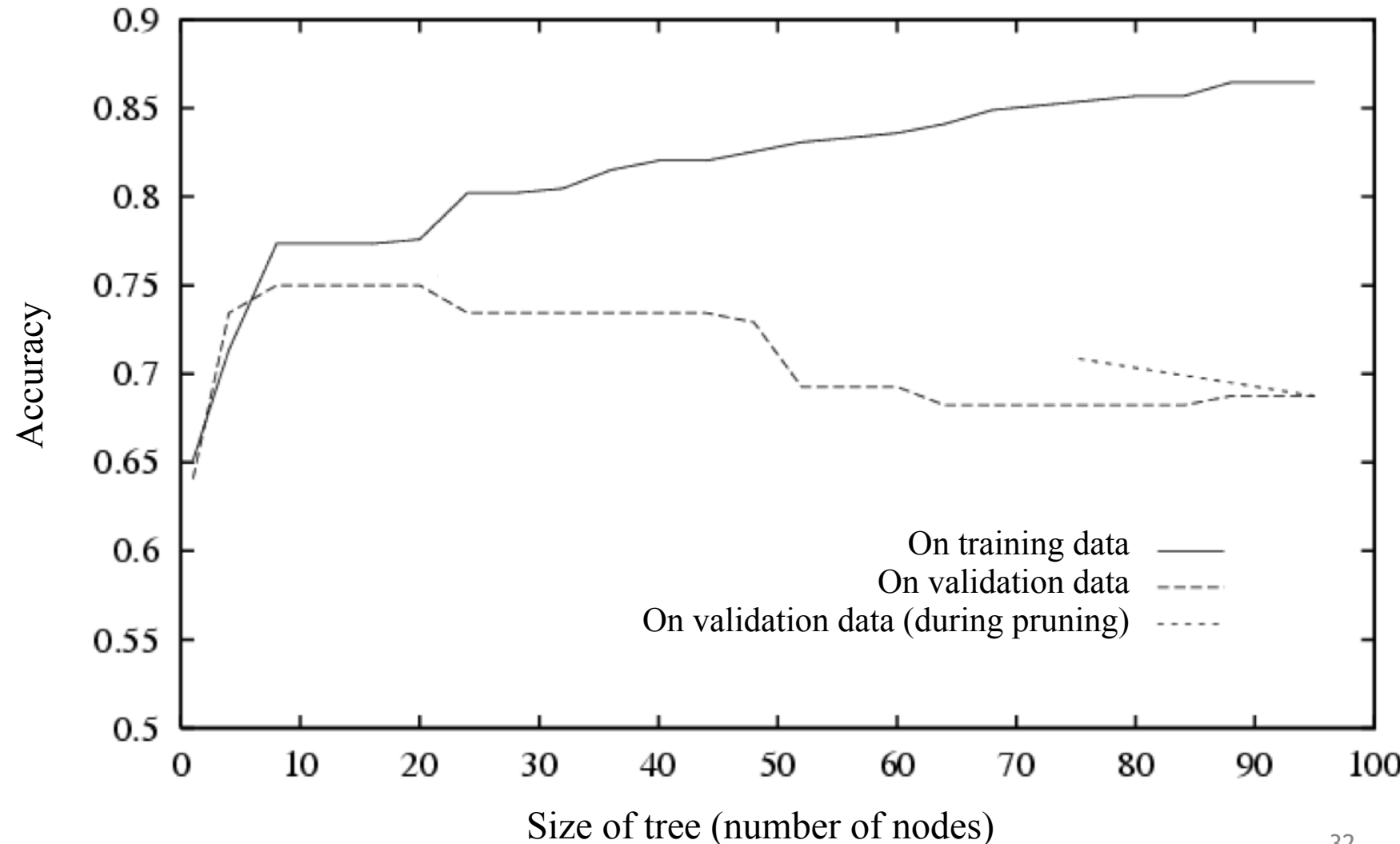


Figure from Tom Mitchell

# Reduced Error Pruning

1. Split data in two:  
*training* dataset and *validation* dataset
2. Grow the full tree using the *training* dataset
3. Repeatedly prune the tree:
  - Evaluate each split using a *validation* dataset by comparing the validation error rate **with and without** that split
  - (Greedy) remove the split that most decreases the validation error rate
  - Stop if no split improves validation error, otherwise repeat

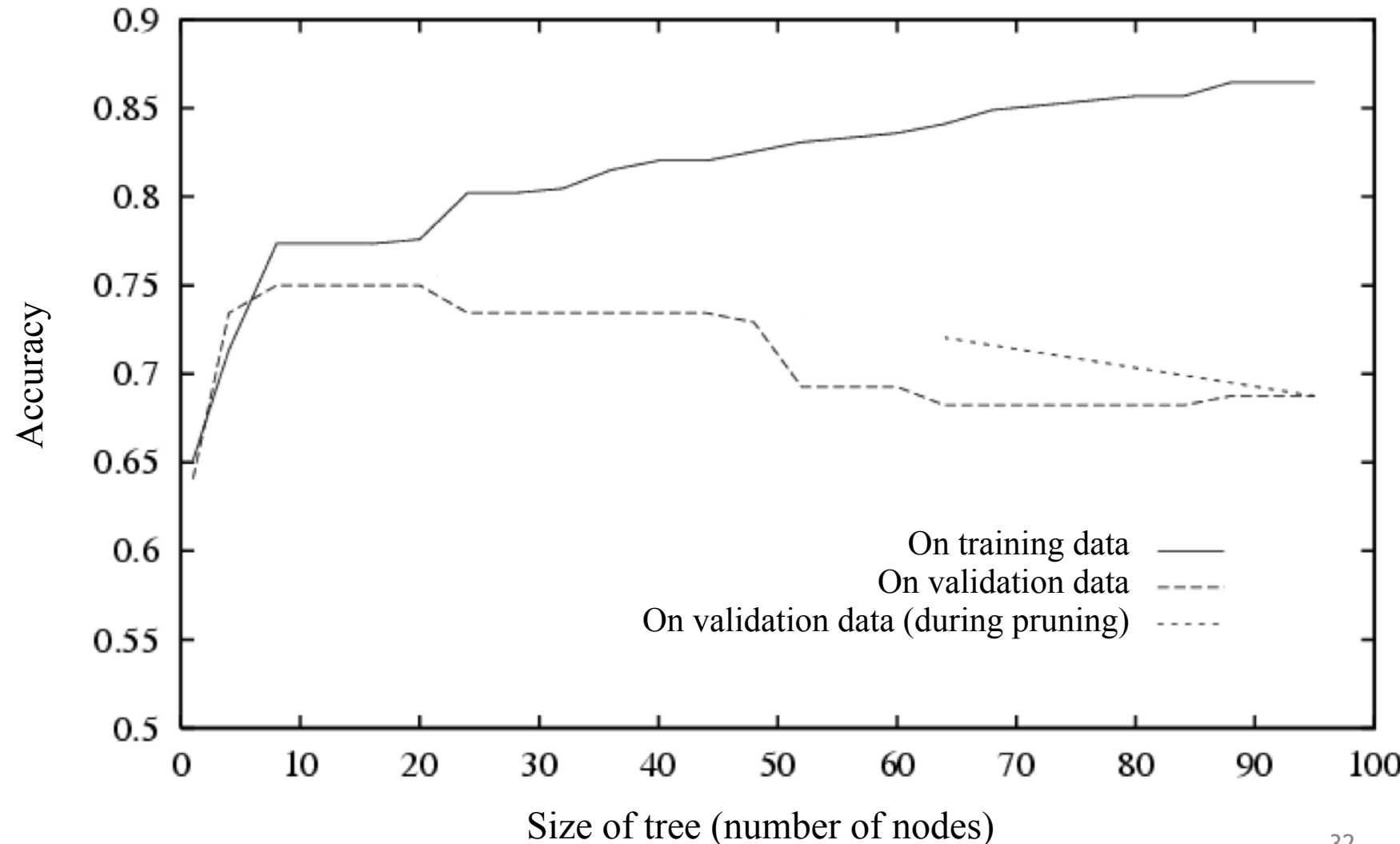


Figure from Tom Mitchell

# Reduced Error Pruning

1. Split data in two:  
*training* dataset and *validation* dataset
2. Grow the full tree using the *training* dataset
3. Repeatedly prune the tree:
  - Evaluate each split using a *validation* dataset by comparing the validation error rate **with and without** that split
  - (Greedy) remove the split that most decreases the validation error rate
  - Stop if no split improves validation error, otherwise repeat

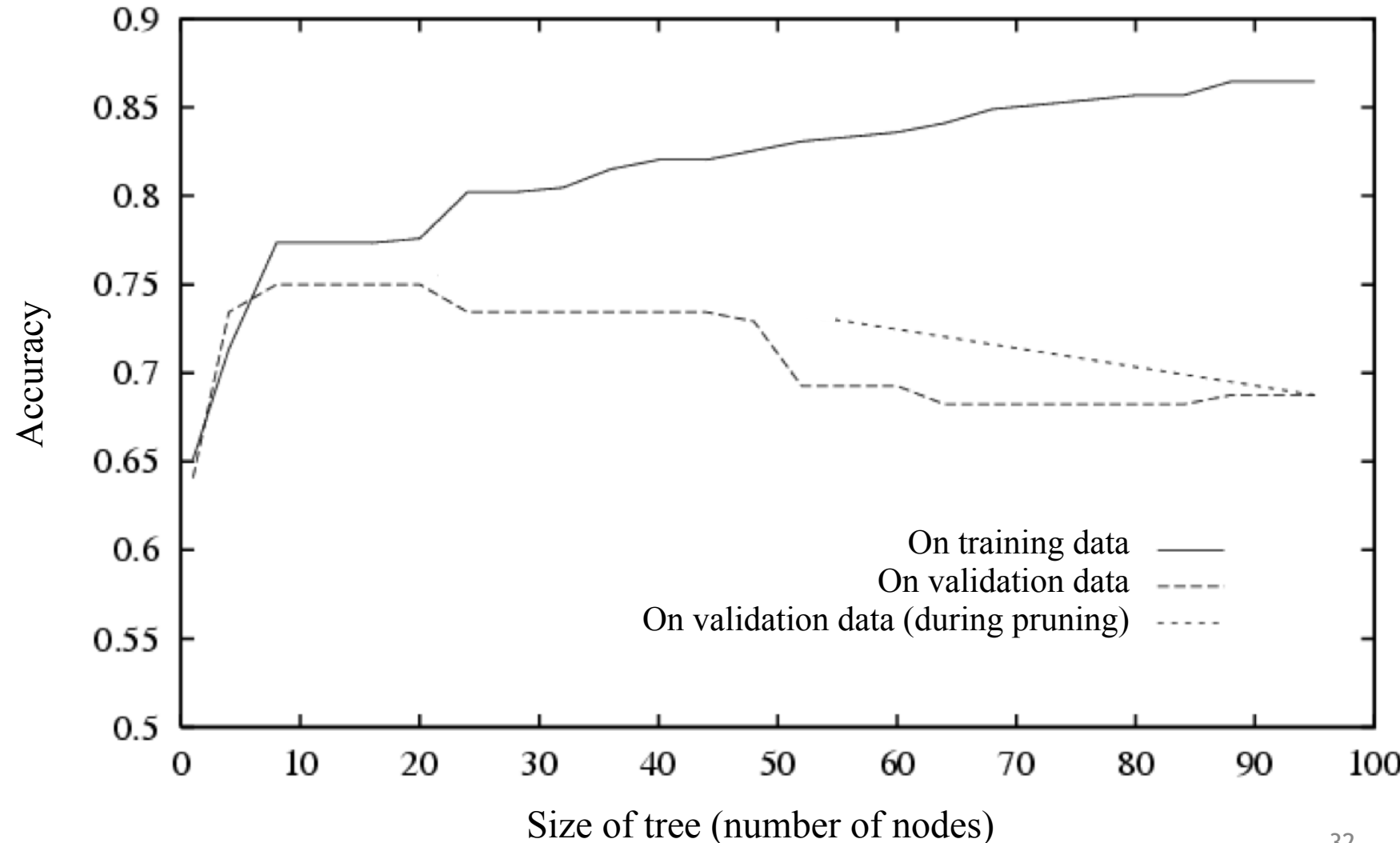


Figure from Tom Mitchell

# Reduced Error Pruning

1. Split data in two:  
*training* dataset and *validation* dataset
2. Grow the full tree using the *training* dataset
3. Repeatedly prune the tree:
  - Evaluate each split using a *validation* dataset by comparing the validation error rate **with and without** that split
  - (Greedy) remove the split that most decreases the validation error rate
  - Stop if no split improves validation error, otherwise repeat

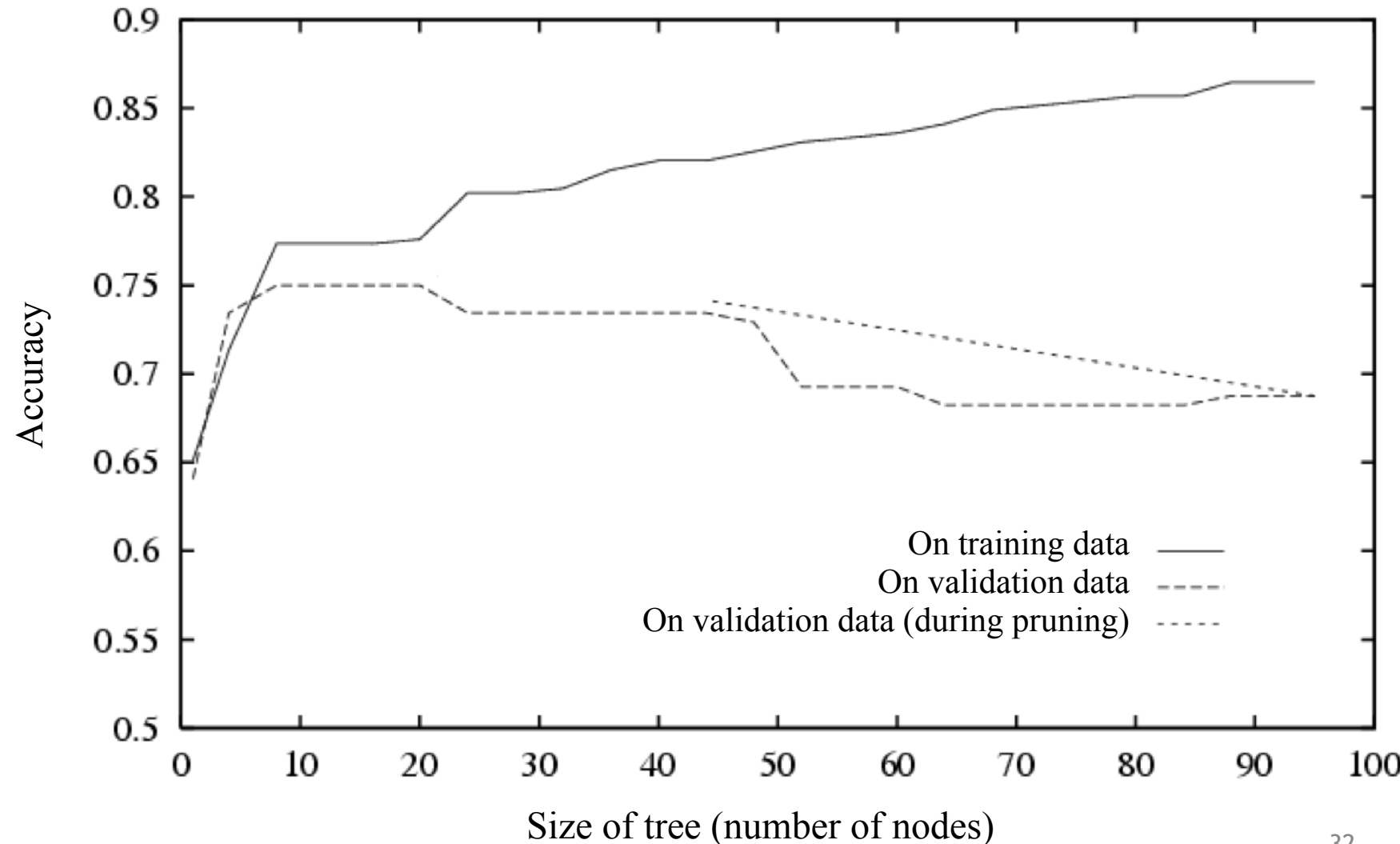


Figure from Tom Mitchell

# Reduced Error Pruning

1. Split data in two:  
*training* dataset and *validation* dataset
2. Grow the full tree using the *training* dataset
3. Repeatedly prune the tree:
  - Evaluate each split using a *validation* dataset by comparing the validation error rate **with and without** that split
  - (Greedy) remove the split that most decreases the validation error rate
  - Stop if no split improves validation error, otherwise repeat

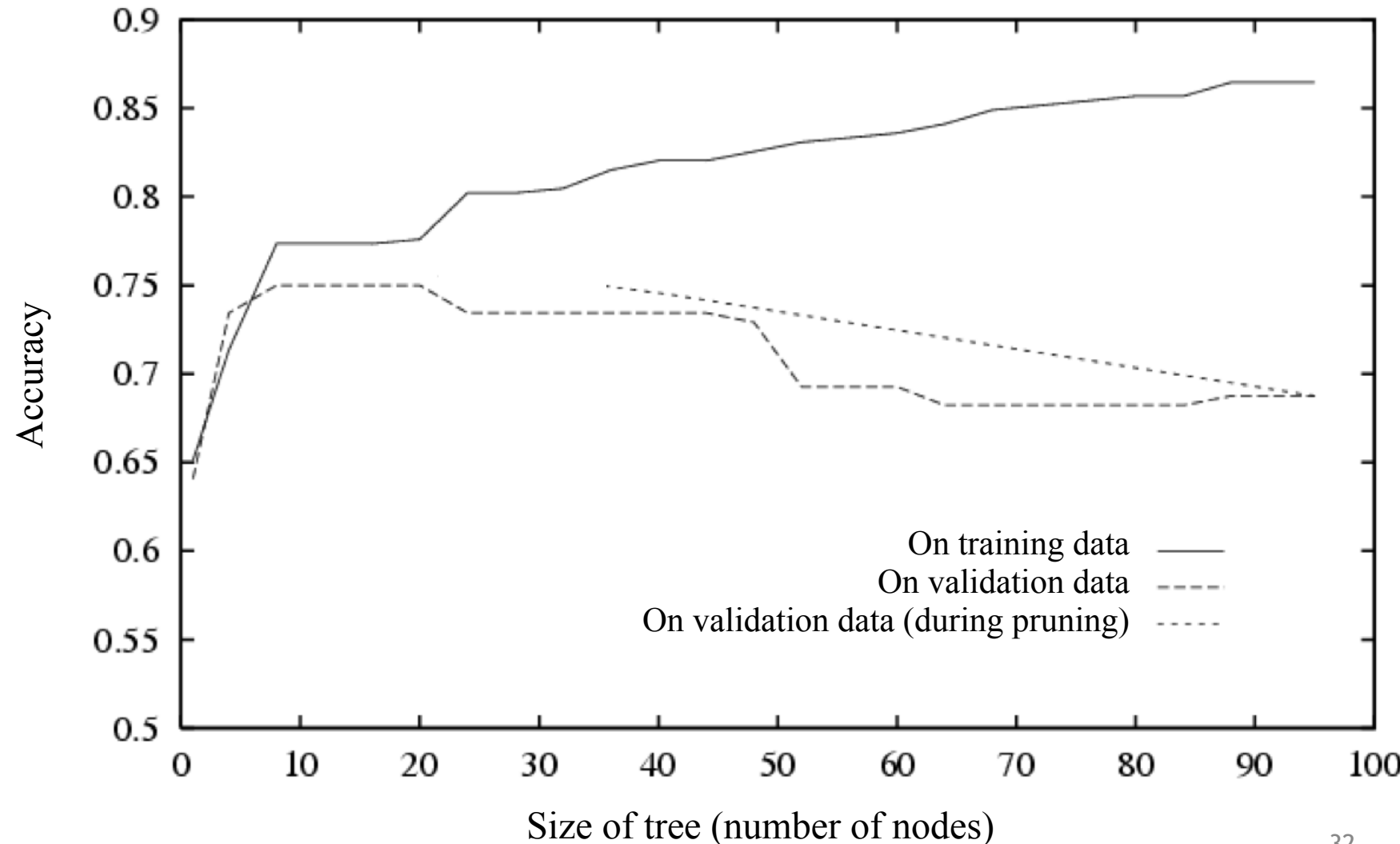


Figure from Tom Mitchell



# Reduced Error Pruning

1. Split data in two:  
*training* dataset and *validation* dataset
2. Grow the full tree using the *training* dataset
3. Repeatedly prune the tree:
  - Evaluate each split using a *validation* dataset by comparing the validation error rate **with and without** that split
  - (Greedy) remove the split that most decreases the validation error rate
  - Stop if no split improves validation error, otherwise repeat

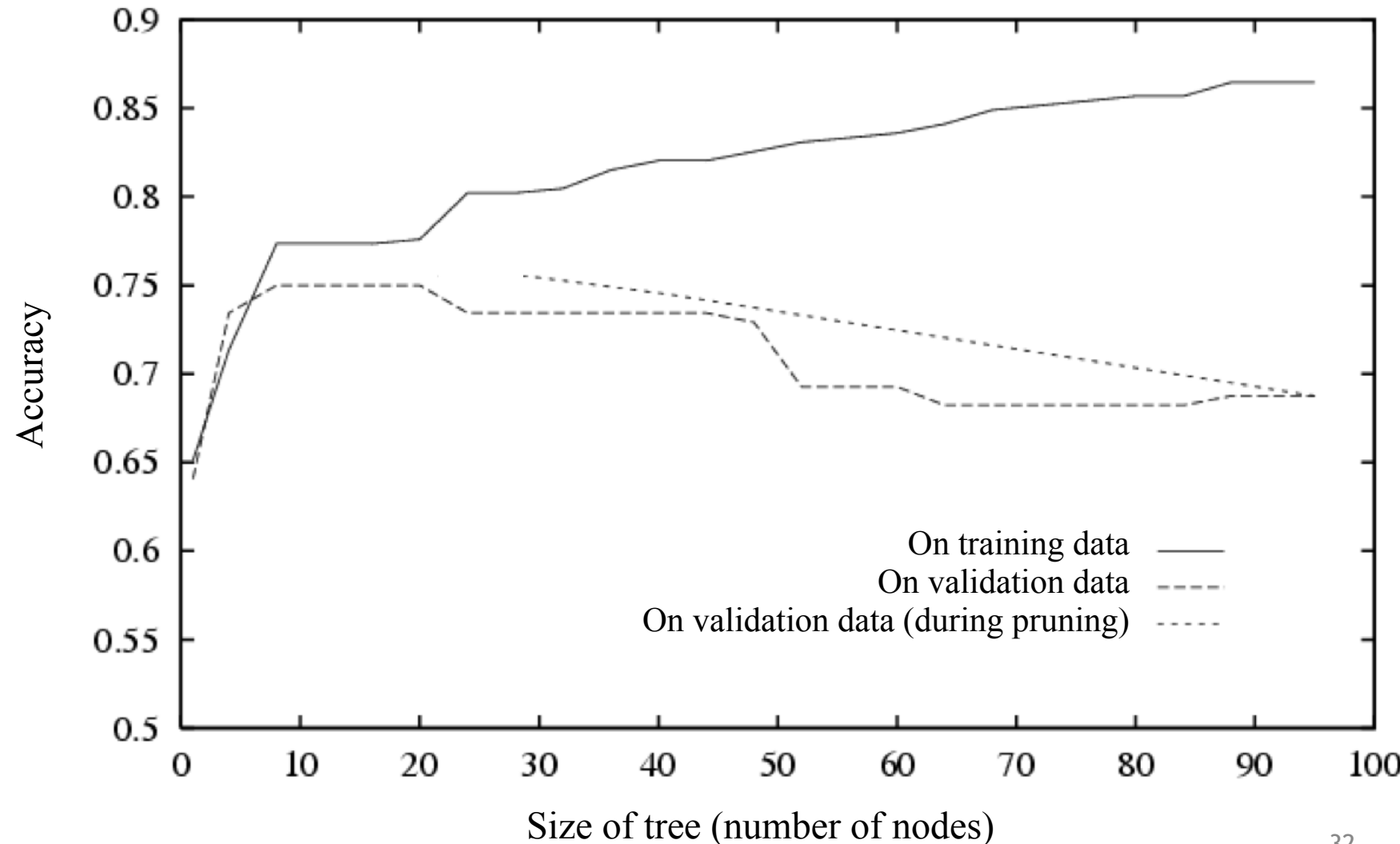


Figure from Tom Mitchell

# Reduced Error Pruning

1. Split data in two:  
*training* dataset and *validation* dataset
2. Grow the full tree using the *training* dataset
3. Repeatedly prune the tree:
  - Evaluate each split using a *validation* dataset by comparing the validation error rate **with and without** that split
  - (Greedy) remove the split that most decreases the validation error rate
  - Stop if no split improves validation error, otherwise repeat

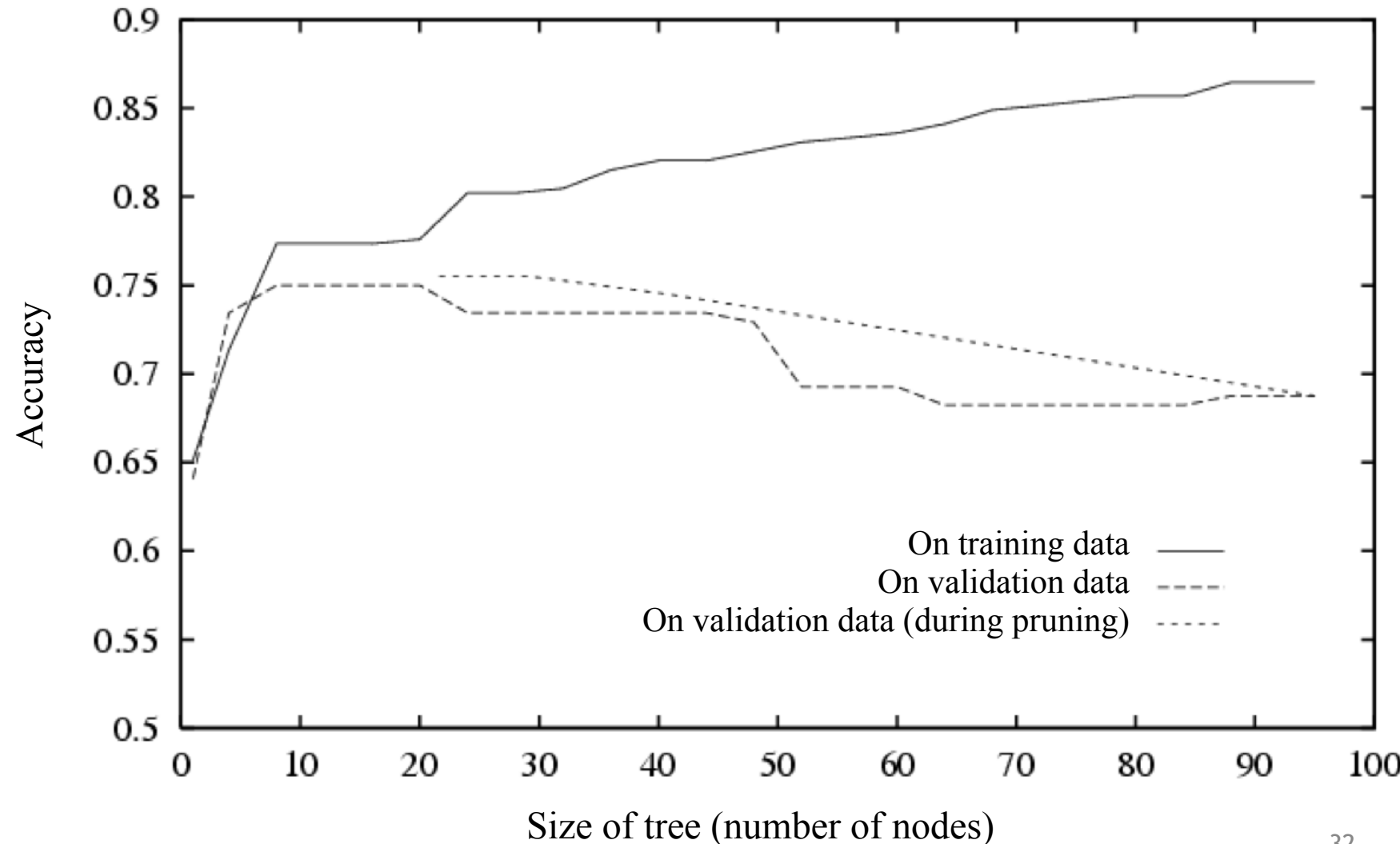
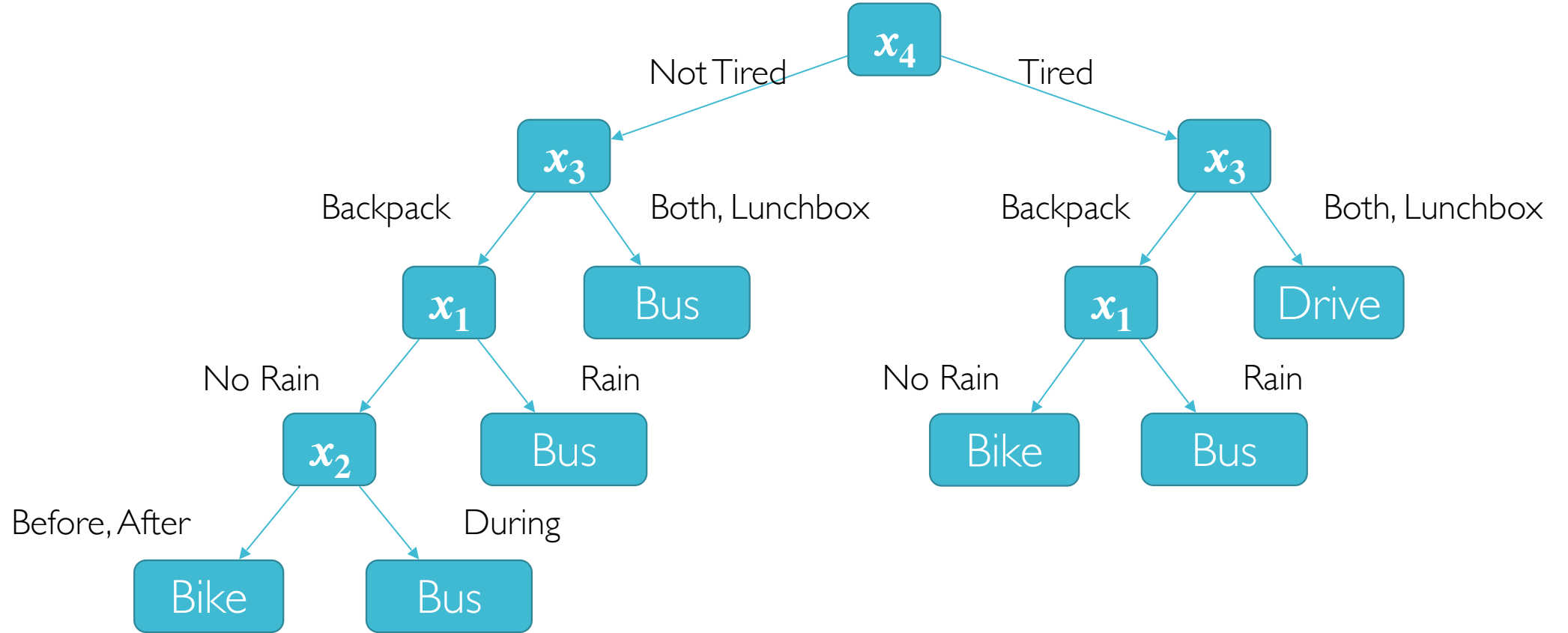
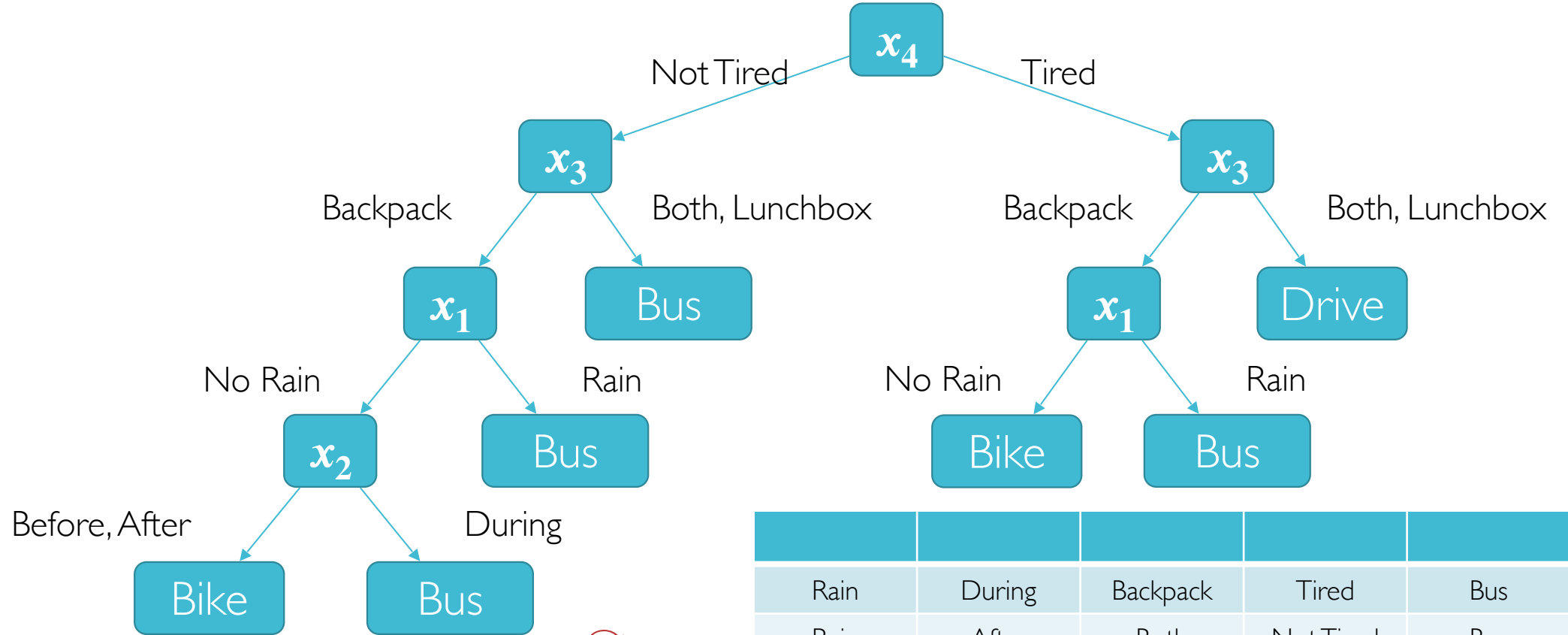


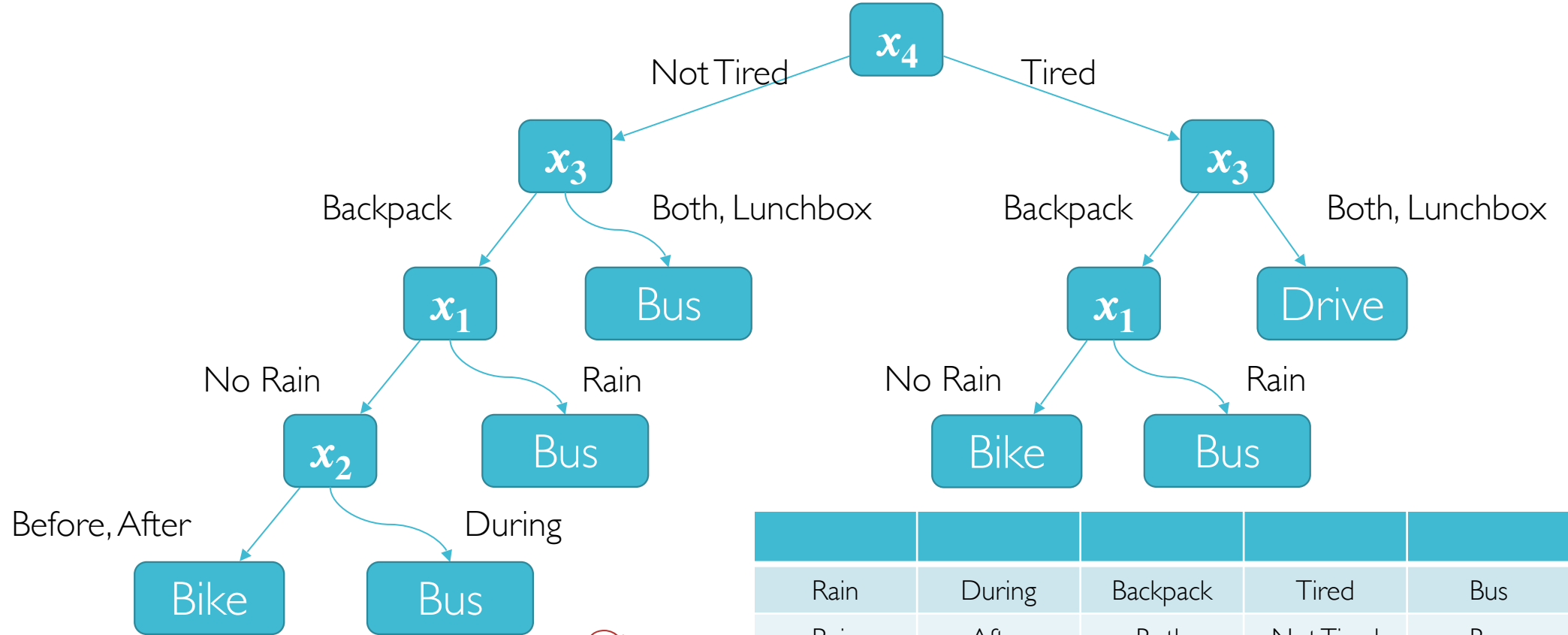
Figure from Tom Mitchell





$\mathcal{D}_{val} =$

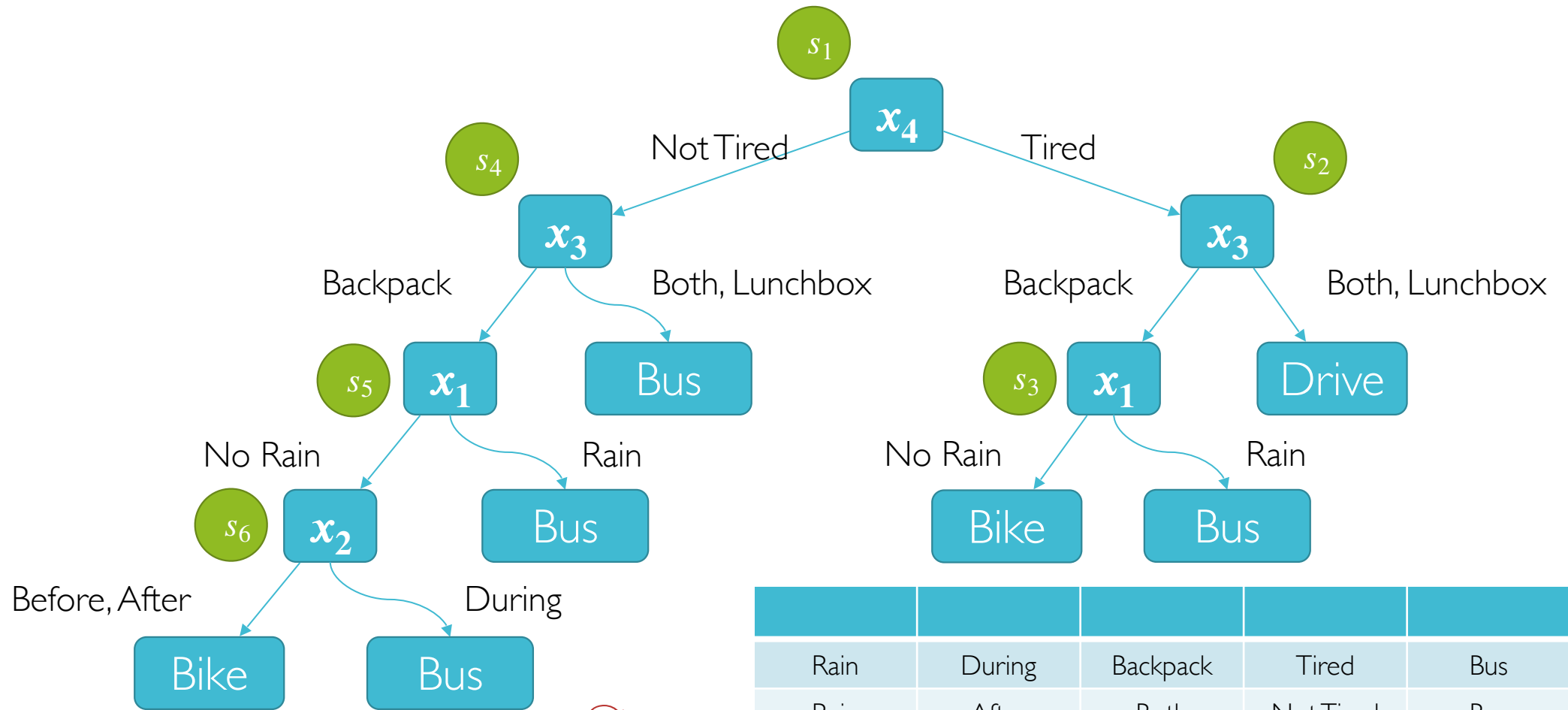
Rain	During	Backpack	Tired	Bus
Rain	After	Both	Not Tired	Bus
No Rain	Before	Backpack	Not Tired	Bus
No Rain	During	Lunchbox	Tired	Drive
No Rain	After	Lunchbox	Tired	Drive



$\mathcal{D}_{val} =$

$$err(h, \mathcal{D}_{val}) = 0.2$$

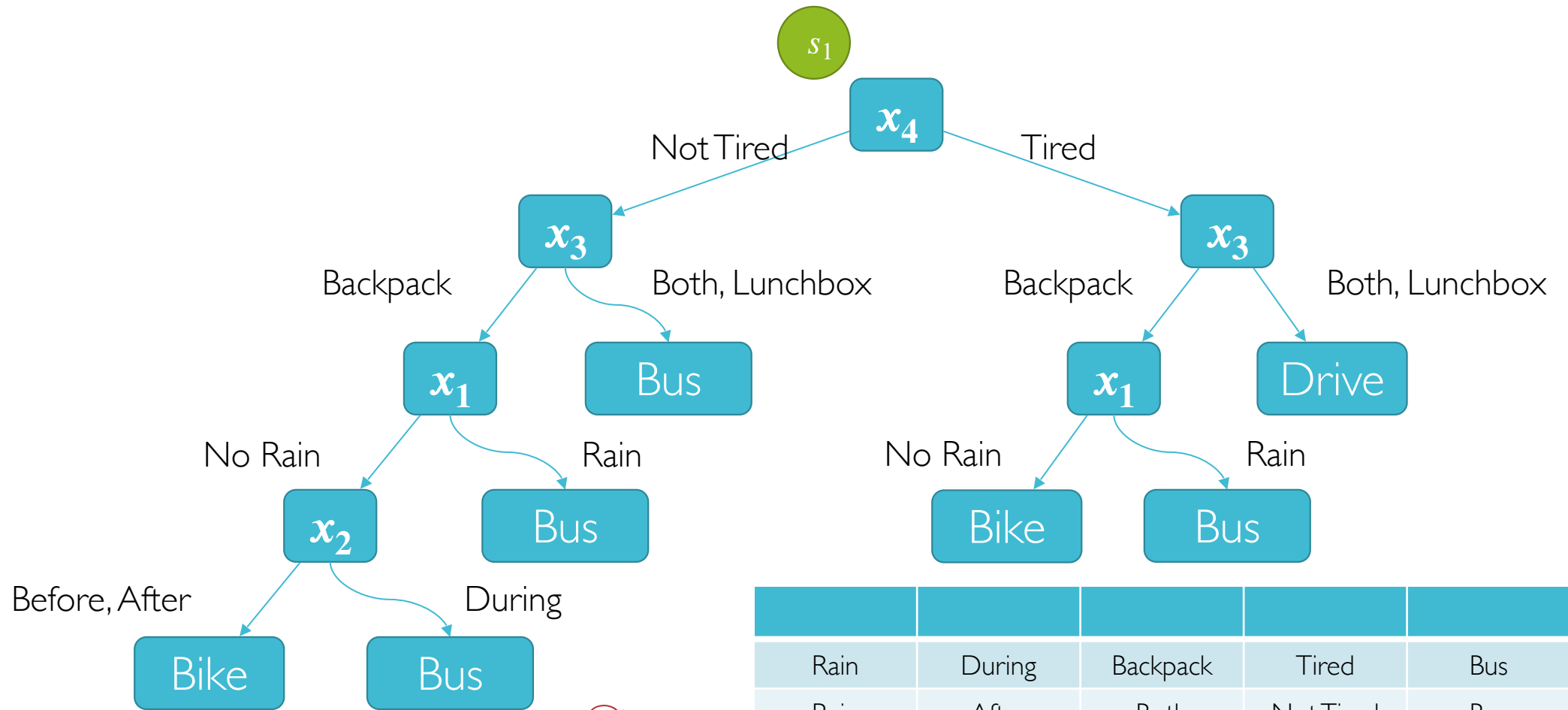
Rain	During	Backpack	Tired	Bus
Rain	After	Both	Not Tired	Bus
No Rain	Before	Backpack	Not Tired	Bus
No Rain	During	Lunchbox	Tired	Drive
No Rain	After	Lunchbox	Tired	Drive



$\mathcal{D}_{val} =$

$$err(h, \mathcal{D}_{val}) = 0.2$$

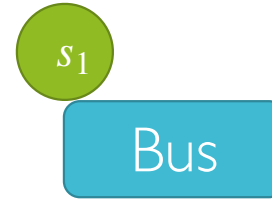
Rain	During	Backpack	Tired	Bus
Rain	After	Both	Not Tired	Bus
No Rain	Before	Backpack	Not Tired	Bus
No Rain	During	Lunchbox	Tired	Drive
No Rain	After	Lunchbox	Tired	Drive



$\mathcal{D}_{val} =$

$$err(h - s_1, \mathcal{D}_{val})$$

Rain	During	Backpack	Tired	Bus
Rain	After	Both	Not Tired	Bus
No Rain	Before	Backpack	Not Tired	Bus
No Rain	During	Lunchbox	Tired	Drive
No Rain	After	Lunchbox	Tired	Drive

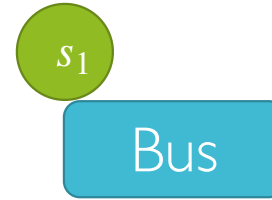


$$err(h - s_1, \mathcal{D}_{val})$$

$$\mathcal{D}_{val} =$$

Rain	During	Backpack	Tired	Bus
Rain	After	Both	Not Tired	Bus
No Rain	Before	Backpack	Not Tired	Bus
No Rain	During	Lunchbox	Tired	Drive
No Rain	After	Lunchbox	Tired	Drive

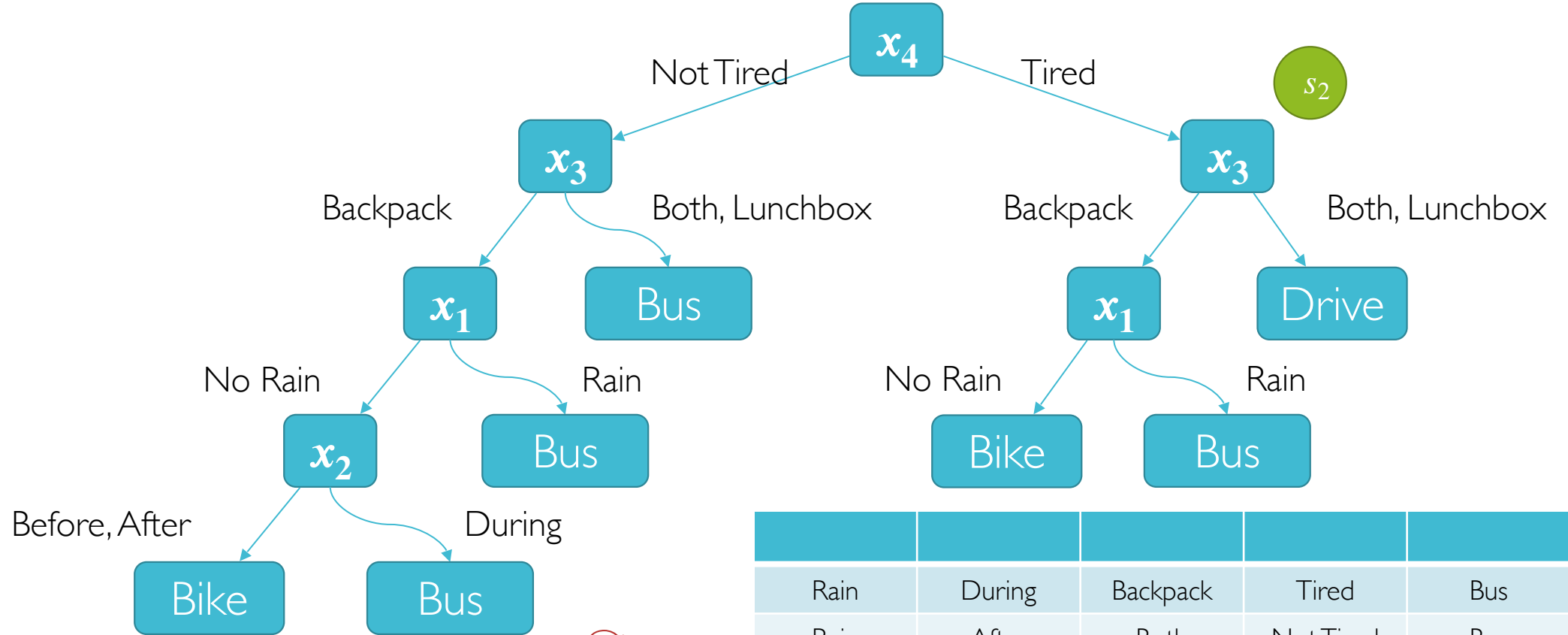




$$err(h - s_1, \mathcal{D}_{val}) = 0.4$$

$\mathcal{D}_{val} =$

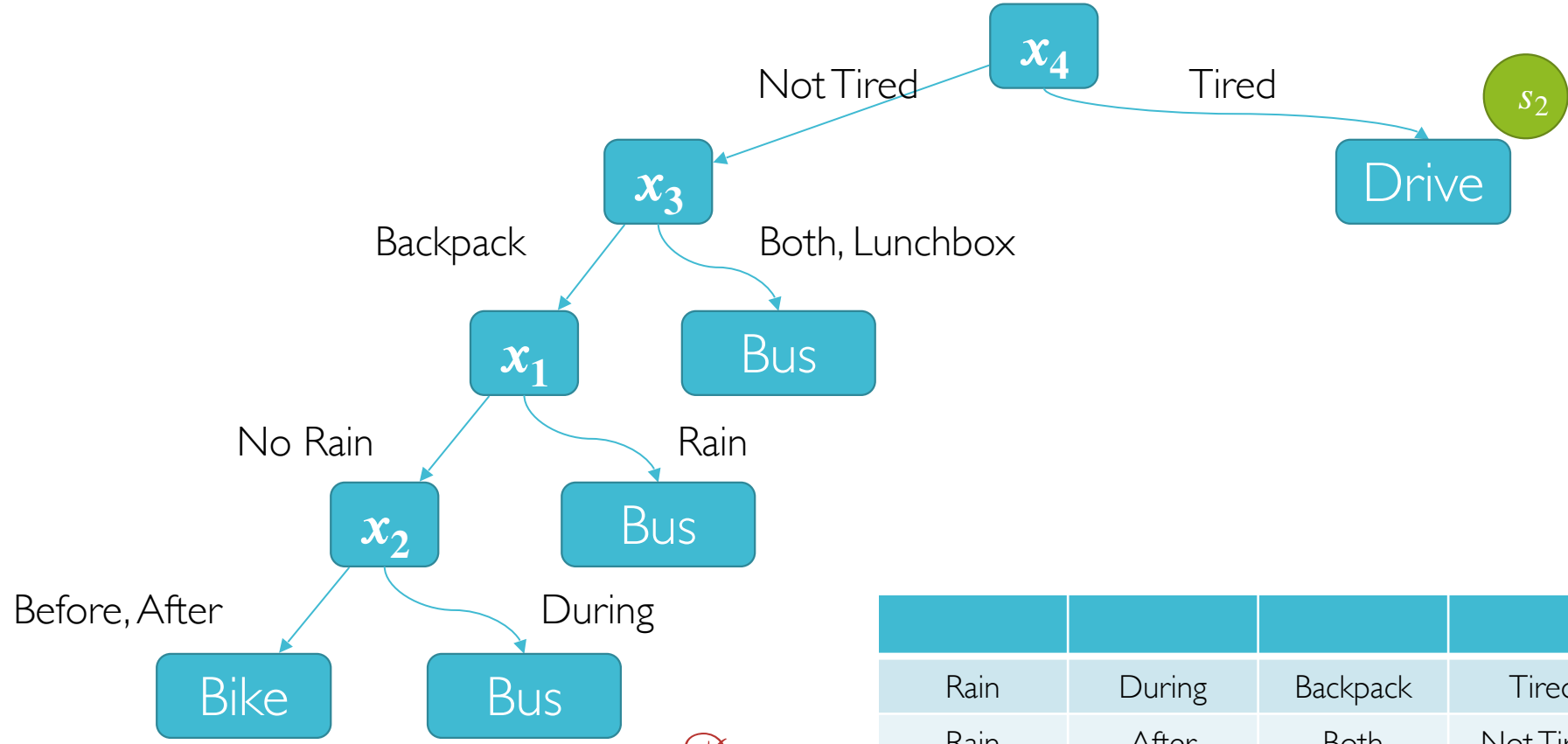
Rain	During	Backpack	Tired	Bus
Rain	After	Both	Not Tired	Bus
No Rain	Before	Backpack	Not Tired	Bus
No Rain	During	Lunchbox	Tired	Drive
No Rain	After	Lunchbox	Tired	Drive



$\mathcal{D}_{val} =$

$$err(h - s_2, \mathcal{D}_{val})$$

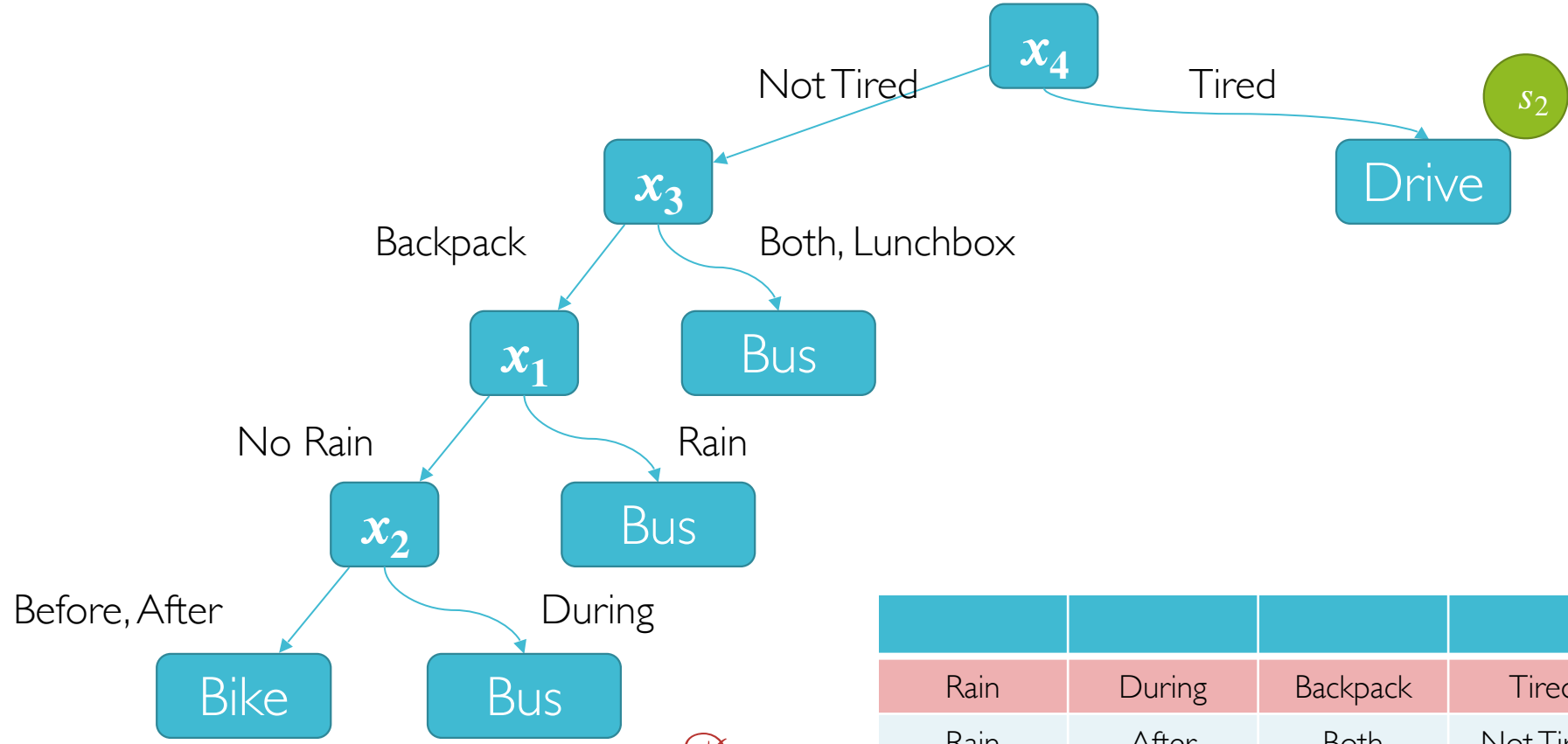
Rain	During	Backpack	Tired	Bus
Rain	After	Both	Not Tired	Bus
No Rain	Before	Backpack	Not Tired	Bus
No Rain	During	Lunchbox	Tired	Drive
No Rain	After	Lunchbox	Tired	Drive



$\mathcal{D}_{val} =$

$err(h - s_2, \mathcal{D}_{val})$

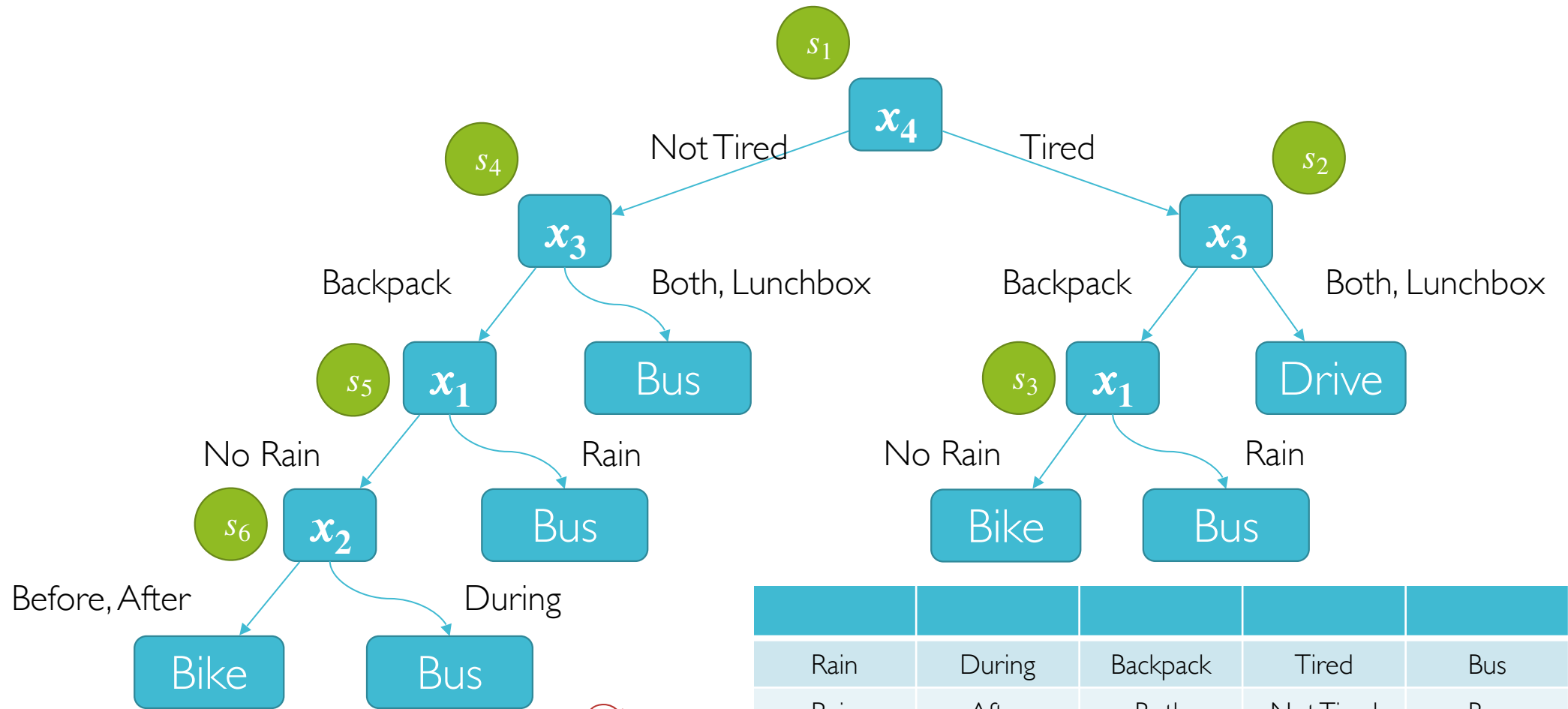
Rain	During	Backpack	Tired	Bus
Rain	After	Both	Not Tired	Bus
No Rain	Before	Backpack	Not Tired	Bus
No Rain	During	Lunchbox	Tired	Drive
No Rain	After	Lunchbox	Tired	Drive



$\mathcal{D}_{val} =$

$$err(h - s_2, \mathcal{D}_{val}) = 0.4$$

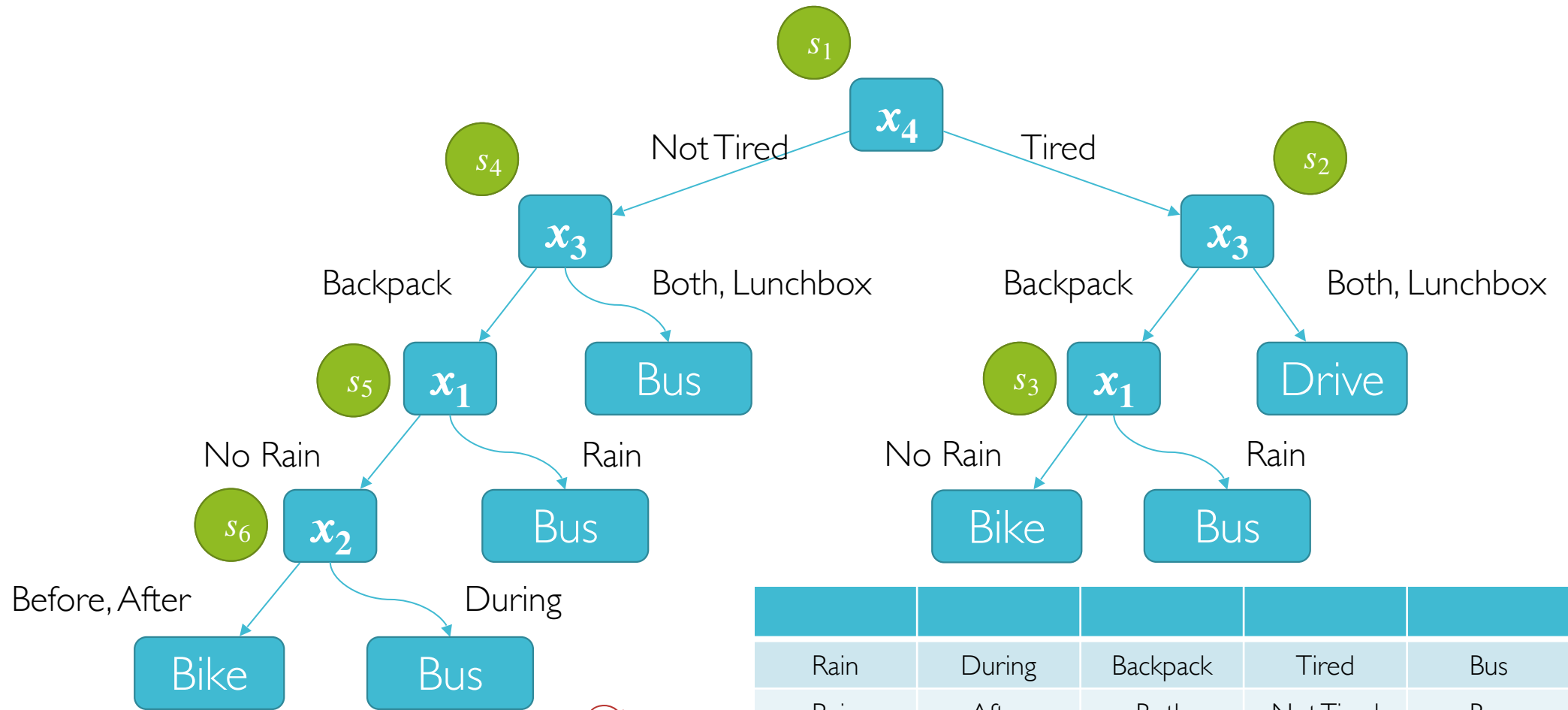
Rain	During	Backpack	Tired	Bus
Rain	After	Both	Not Tired	Bus
No Rain	Before	Backpack	Not Tired	Bus
No Rain	During	Lunchbox	Tired	Drive
No Rain	After	Lunchbox	Tired	Drive



$\mathcal{D}_{val} =$

$s$	$s_1$	$s_2$	$s_3$	$s_4$	$s_5$	$s_6$
$\text{err}(h - s, \mathcal{D}_{val})$	0.4	0.4	0.4	0	0	0.2

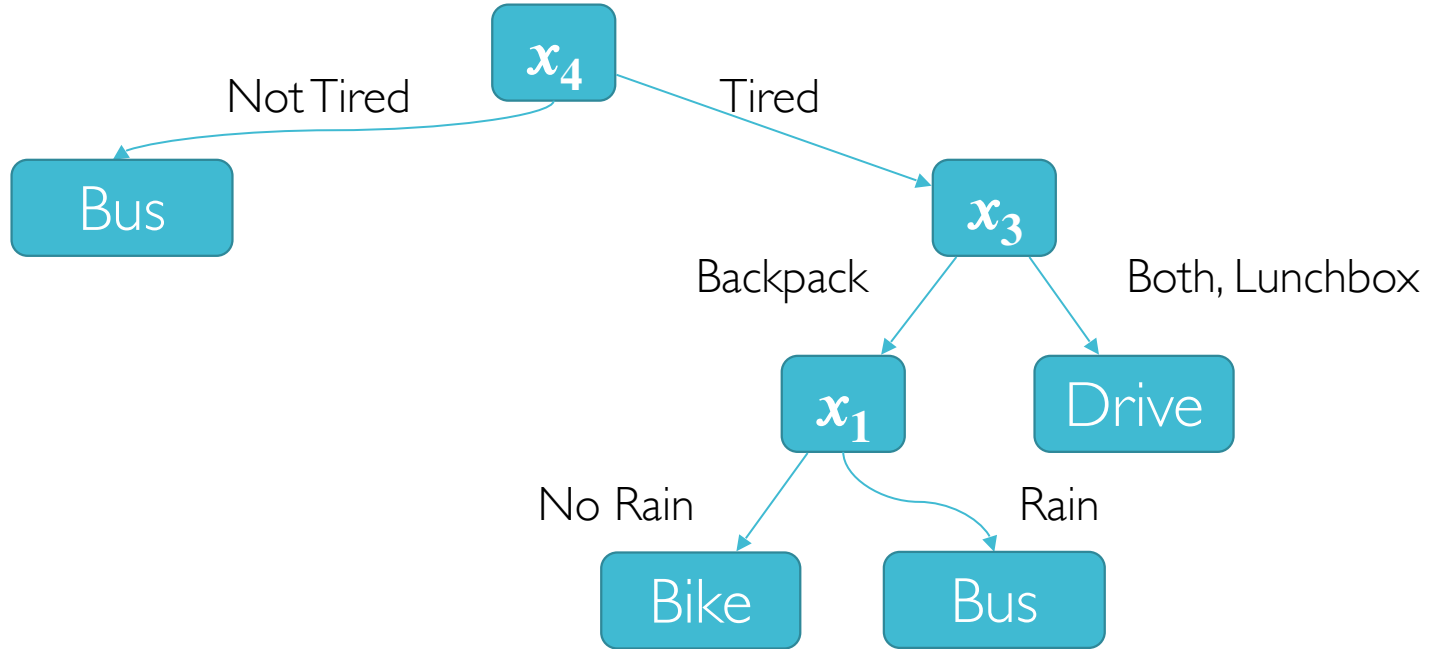
Rain	During	Backpack	Tired	Bus
Rain	After	Both	Not Tired	Bus
No Rain	Before	Backpack	Not Tired	Bus
No Rain	During	Lunchbox	Tired	Drive
No Rain	After	Lunchbox	Tired	Drive



$s$	$s_1$	$s_2$	$s_3$	$s_4$	$s_5$	$s_6$
$\text{err}(h - s, \mathcal{D}_{\text{val}})$	0.4	0.4	0.4	0	0	0.2

$\mathcal{D}_{\text{val}} =$

Rain	During	Backpack	Tired	Bus
Rain	After	Both	Not Tired	Bus
No Rain	Before	Backpack	Not Tired	Bus
No Rain	During	Lunchbox	Tired	Drive
No Rain	After	Lunchbox	Tired	Drive

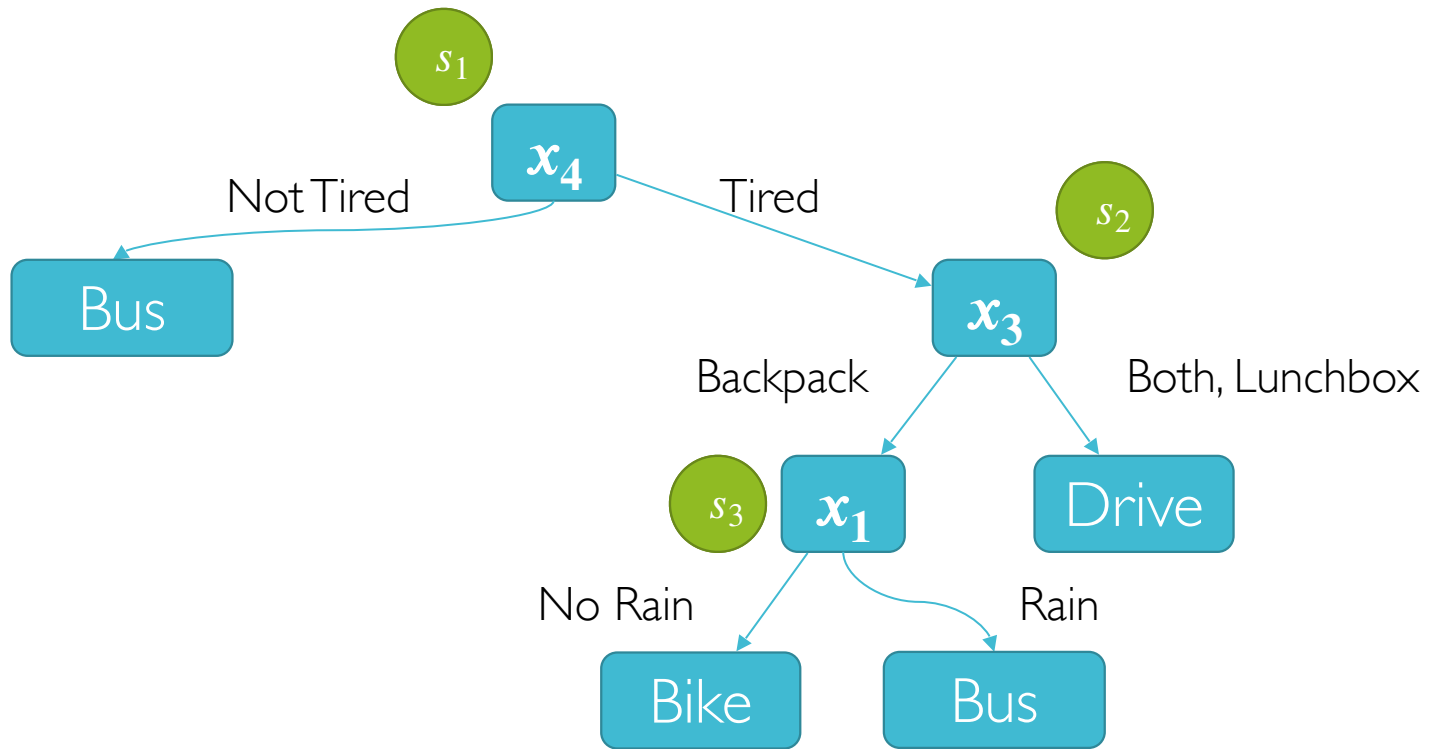


$\mathcal{D}_{val} =$

Rain	During	Backpack	Tired	Bus
Rain	After	Both	Not Tired	Bus
No Rain	Before	Backpack	Not Tired	Bus
No Rain	During	Lunchbox	Tired	Drive
No Rain	After	Lunchbox	Tired	Drive

$$err(h, \mathcal{D}_{val}) = 0$$

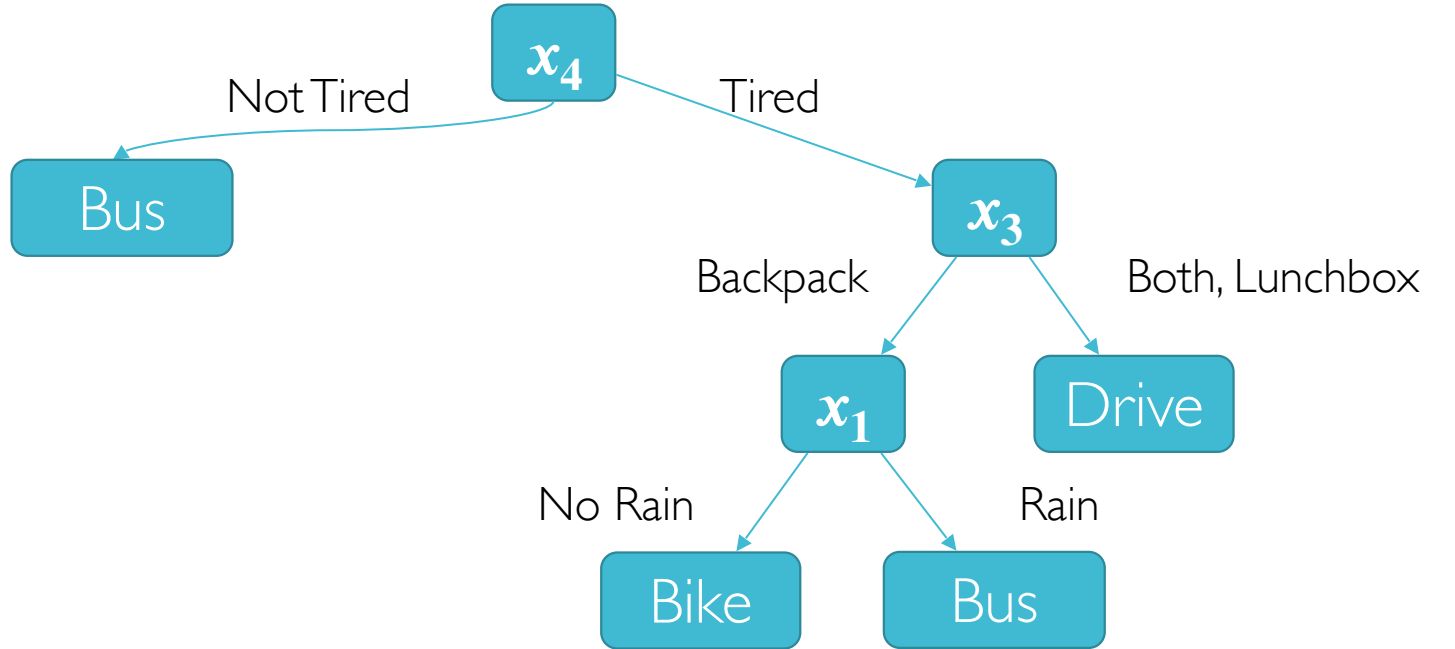
	$s_1$	$s_2$	$s_3$
$\text{err}(h - s, \mathcal{D}_{\text{val}})$	0.4	0.2	0.2



$\mathcal{D}_{\text{val}} =$

Rain	During	Backpack	Tired	Bus
Rain	After	Both	Not Tired	Bus
No Rain	Before	Backpack	Not Tired	Bus
No Rain	During	Lunchbox	Tired	Drive
No Rain	After	Lunchbox	Tired	Drive





# Getting real

- So far we've been doing
  - discrete
  - binary classification
- Either features or labels (or both!) could have more than two values

## How to handle ordered features (e.g., reals)

Cholesterol	Heart Disease?
174	No
155	No
163	Yes
233	Yes
197	Yes
181	No
244	Yes
245	No
178	No
221	Yes

# How to handle ordered features (e.g., reals)

Cholesterol	Heart Disease?
174	No
155	No
163	Yes
233	Yes
197	Yes
181	No
244	Yes
245	No
178	No
221	Yes



Cholesterol	Heart Disease?
155	No
163	Yes
174	No
178	No
181	No
197	Yes
221	Yes
233	Yes
244	Yes
245	No

# How to handle ordered features (e.g., reals)

Cholesterol	Heart Disease?
174	No
155	No
163	Yes
233	Yes
197	Yes
181	No
244	Yes
245	No
178	No
221	Yes



Cholesterol	Heart Disease?
155	No
163	Yes
174	No
178	No
181	No
197	Yes
221	Yes
233	Yes
244	Yes
245	No

←  $x_3 < 189$

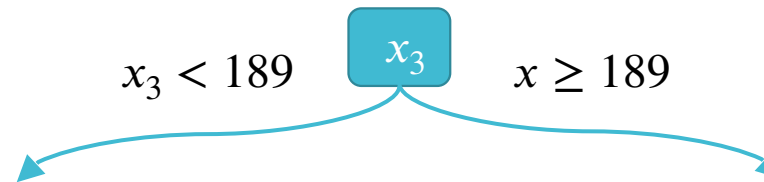
# How to handle ordered features (e.g., reals)

Cholesterol	Heart Disease?
174	No
155	No
163	Yes
233	Yes
197	Yes
181	No
244	Yes
245	No
178	No
221	Yes



Cholesterol	Heart Disease?
155	No
163	Yes
174	No
178	No
181	No
197	Yes
221	Yes
233	Yes
244	Yes
245	No

←  $x_3 < 189$



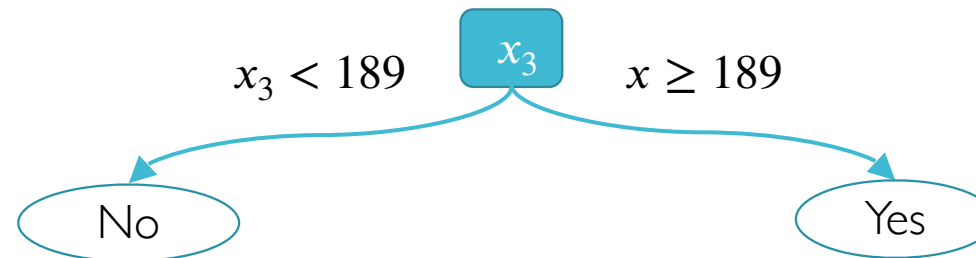
# How to handle ordered features (e.g., reals)

Cholesterol	Heart Disease?
174	No
155	No
163	Yes
233	Yes
197	Yes
181	No
244	Yes
245	No
178	No
221	Yes



Cholesterol	Heart Disease?
155	No
163	Yes
174	No
178	No
181	No
197	Yes
221	Yes
233	Yes
244	Yes
245	No

←  $x_3 < 189$



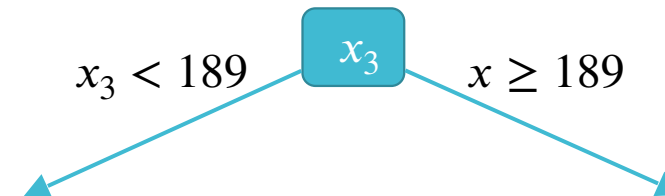
# How to handle ordered features (e.g., reals)

Cholesterol	Heart Disease?
174	No
155	No
163	Yes
233	Yes
197	Yes
181	No
244	Yes
245	No
178	No
221	Yes



Cholesterol	Heart Disease?
155	No
163	Yes
174	No
178	No
181	No
197	Yes
221	Yes
233	Yes
244	Yes
245	No

$x_3 < 189$





# How to handle ordered features (e.g., reals)

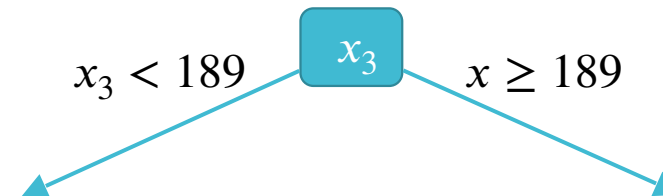
Cholesterol	Heart Disease?
174	No
155	No
163	Yes
233	Yes
197	Yes
181	No
244	Yes
245	No
178	No
221	Yes



Cholesterol	Heart Disease?
155	No
163	Yes
174	No
178	No
181	No
197	Yes
221	Yes
233	Yes
244	Yes
245	No

←  $x_3 < 189$

←  $x_3 < 244.5$



# How to handle ordered features (e.g., reals)

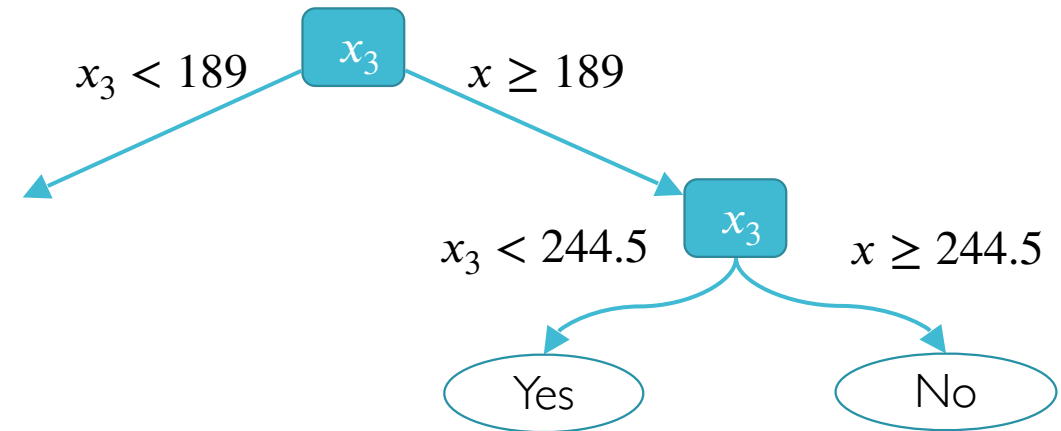
Cholesterol	Heart Disease?
174	No
155	No
163	Yes
233	Yes
197	Yes
181	No
244	Yes
245	No
178	No
221	Yes



Cholesterol	Heart Disease?
155	No
163	Yes
174	No
178	No
181	No
197	Yes
221	Yes
233	Yes
244	Yes
245	No

$x_3 < 189$

$x_3 < 244.5$



# How to handle ordered features (e.g., reals)

Cholesterol	Heart Disease?
174	No
155	No
163	Yes
233	Yes
197	Yes
181	No
244	Yes
245	No
178	No
221	Yes

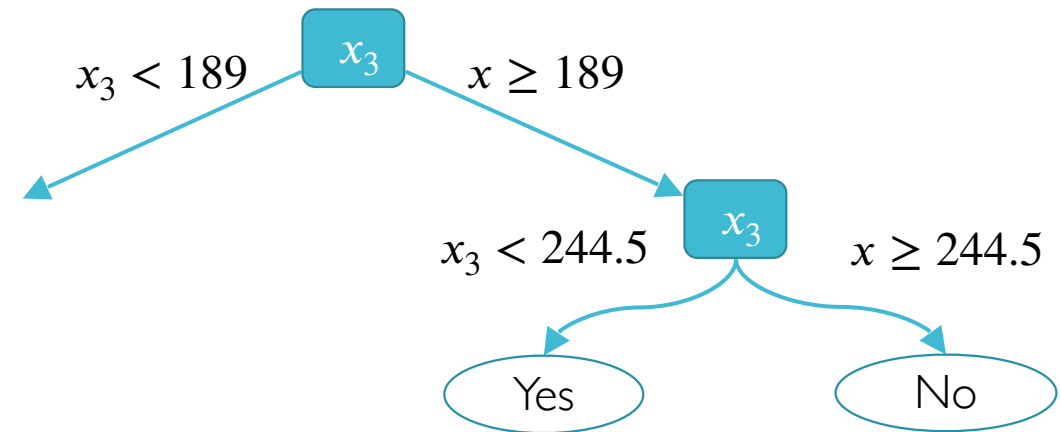


Cholesterol	Heart Disease?
155	No
163	Yes
174	No
178	No
181	No
197	Yes
221	Yes
233	Yes
244	Yes
245	No

←  $x_3 < 168.5$

←  $x_3 < 189$

←  $x_3 < 244.5$



# How to handle ordered features (e.g., reals)

Cholesterol	Heart Disease?
174	No
155	No
163	Yes
233	Yes
197	Yes
181	No
244	Yes
245	No
178	No
221	Yes

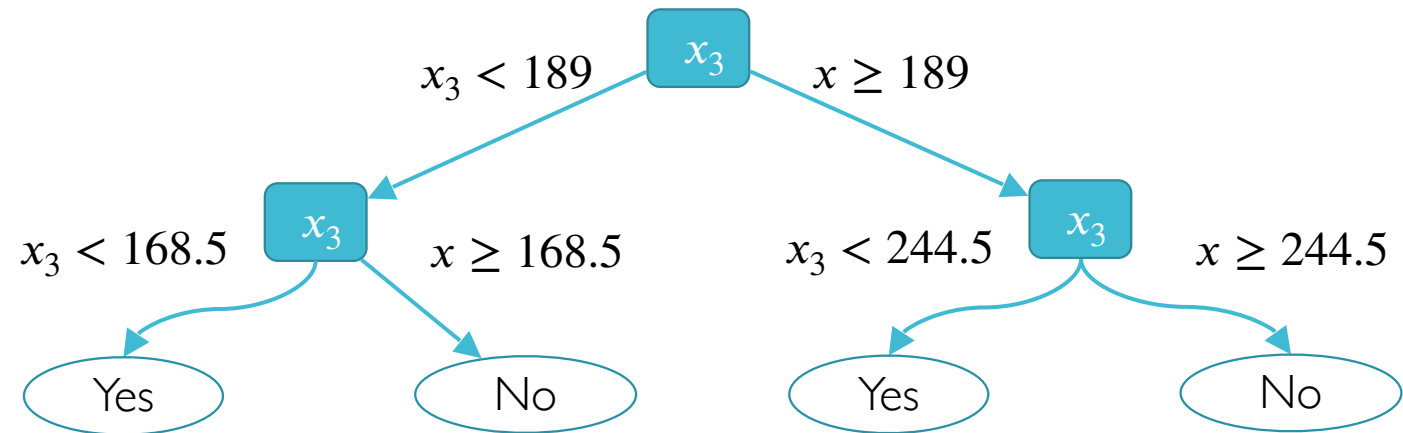


Cholesterol	Heart Disease?
155	No
163	Yes
174	No
178	No
181	No
197	Yes
221	Yes
233	Yes
244	Yes
245	No

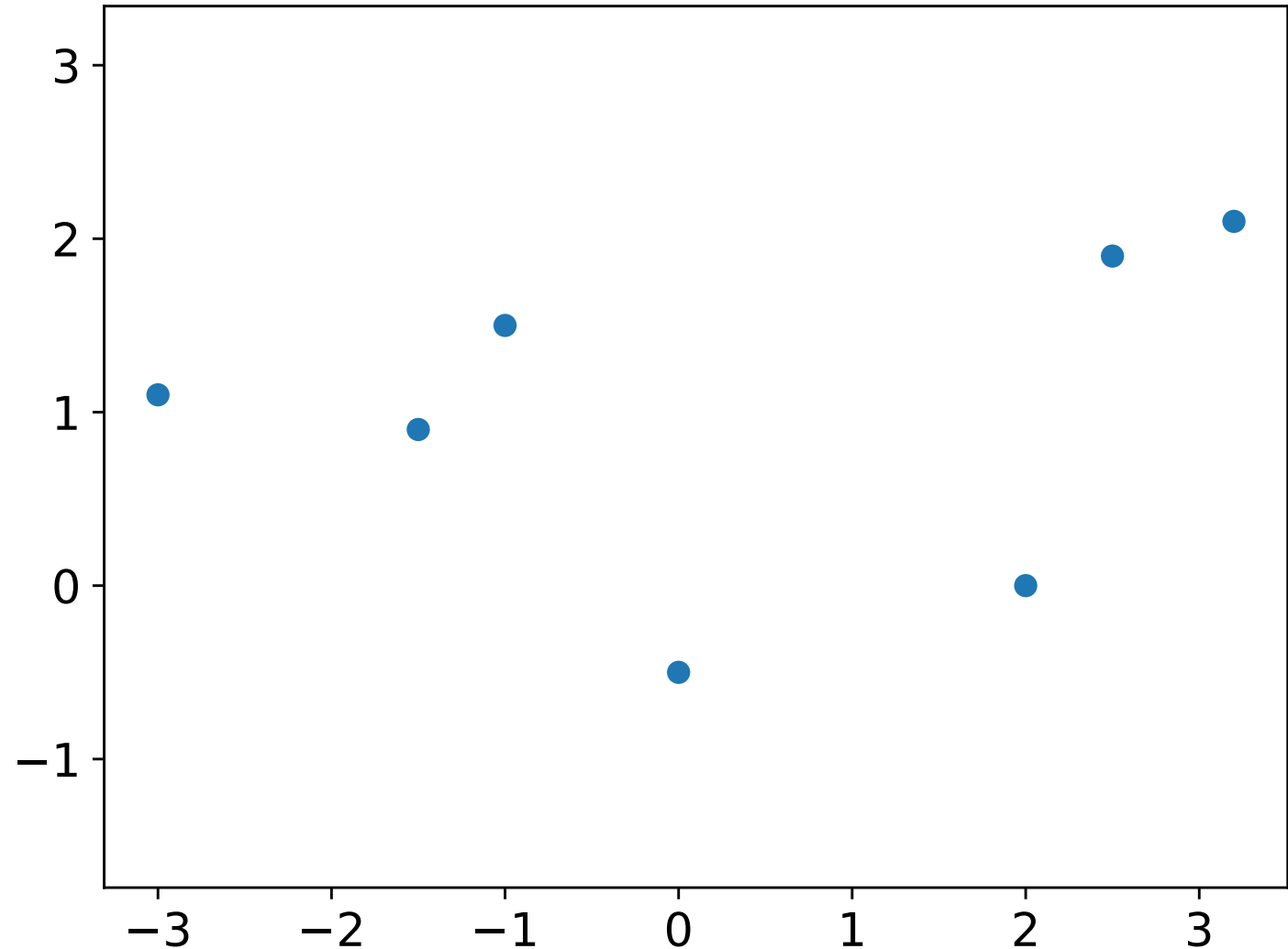
←  $x_3 < 168.5$

←  $x_3 < 189$

←  $x_3 < 244.5$



## How to handle real outputs: regression tree



- In each leaf: predict mean
- Splitting criterion: prediction error  $\sum_{\text{examples } i} (y_i - \hat{y}_i)^2$

# Decision Trees (DTs) in the Wild

- DTs are one of the most popular classification methods for practical applications
  - Reason #1: The learned representation is **easy to explain** a non-ML person
  - Reason #2: They are **efficient** in both computation and memory

# Decision Trees (DTs) in the Wild

- DTs are one of the most popular classification methods for practical applications
  - Reason #1: The learned representation is **easy to explain** a non-ML person
  - Reason #2: They are **efficient** in both computation and memory
- DTs can be applied to a wide variety of problems including **classification, regression, density estimation, etc.**

# Decision Trees (DTs) in the Wild

- DTs are one of the most popular classification methods for practical applications
  - Reason #1: The learned representation is **easy to explain** a non-ML person
  - Reason #2: They are **efficient** in both computation and memory
- DTs can be applied to a wide variety of problems including **classification, regression, density estimation, etc.**
- **Applications of DTs include...**
  - medicine, molecular biology, text classification, manufacturing, astronomy, agriculture, and many others



# Decision Trees (DTs) in the Wild

- DTs are one of the most popular classification methods for practical applications
  - Reason #1: The learned representation is **easy to explain** a non-ML person
  - Reason #2: They are **efficient** in both computation and memory
- DTs can be applied to a wide variety of problems including **classification, regression, density estimation, etc.**
- **Applications of DTs include...**
  - medicine, molecular biology, text classification, manufacturing, astronomy, agriculture, and many others
- **Decision Forests** learn many DTs from random subsets of features; the result is a very powerful example of an **ensemble method** (discussed later in the course)

# DT Learning Objectives

*You should be able to...*

1. Formalize a learning problem (e.g., learning a Decision Tree) by identifying the input space, output space, hypothesis space, and target function
2. Implement Decision Tree training and prediction
3. Use different splitting criteria for Decision Trees and be able to define entropy, conditional entropy, and mutual information / information gain
4. Describe the inductive bias of a decision tree
5. Explain the difference between true error and training error
6. Explain the difference between memorization and generalization
7. Judge whether a decision tree is underfitting or overfitting
8. Implement a pruning or early stopping method to combat overfitting in Decision Tree learning

# **REAL VALUED ATTRIBUTES**



# Fisher Iris Dataset

Fisher (1936) used 150 measurements of flowers from 3 different species: Iris setosa (0), Iris virginica (1), Iris versicolor (2) collected by Anderson (1936)

N=7 (here)  
N=150 (total)

Species	Sepal Length	Sepal Width	Petal Length	Petal Width
0	4.3	3.0	1.1	0.1
0	4.9	3.6	1.4	0.1
0	5.3	3.7	1.5	0.2
1	4.9	2.4	3.3	1.0
1	5.7	2.8	4.1	1.3
1	6.3	3.3	4.7	1.6
1	6.7	3.0	5.0	1.7

M=4

# Fisher Iris Dataset

Fisher (1936) used 150 measurements of flowers from 3 different species: Iris setosa (0), Iris virginica (1), Iris versicolor (2) collected by Anderson (1936)

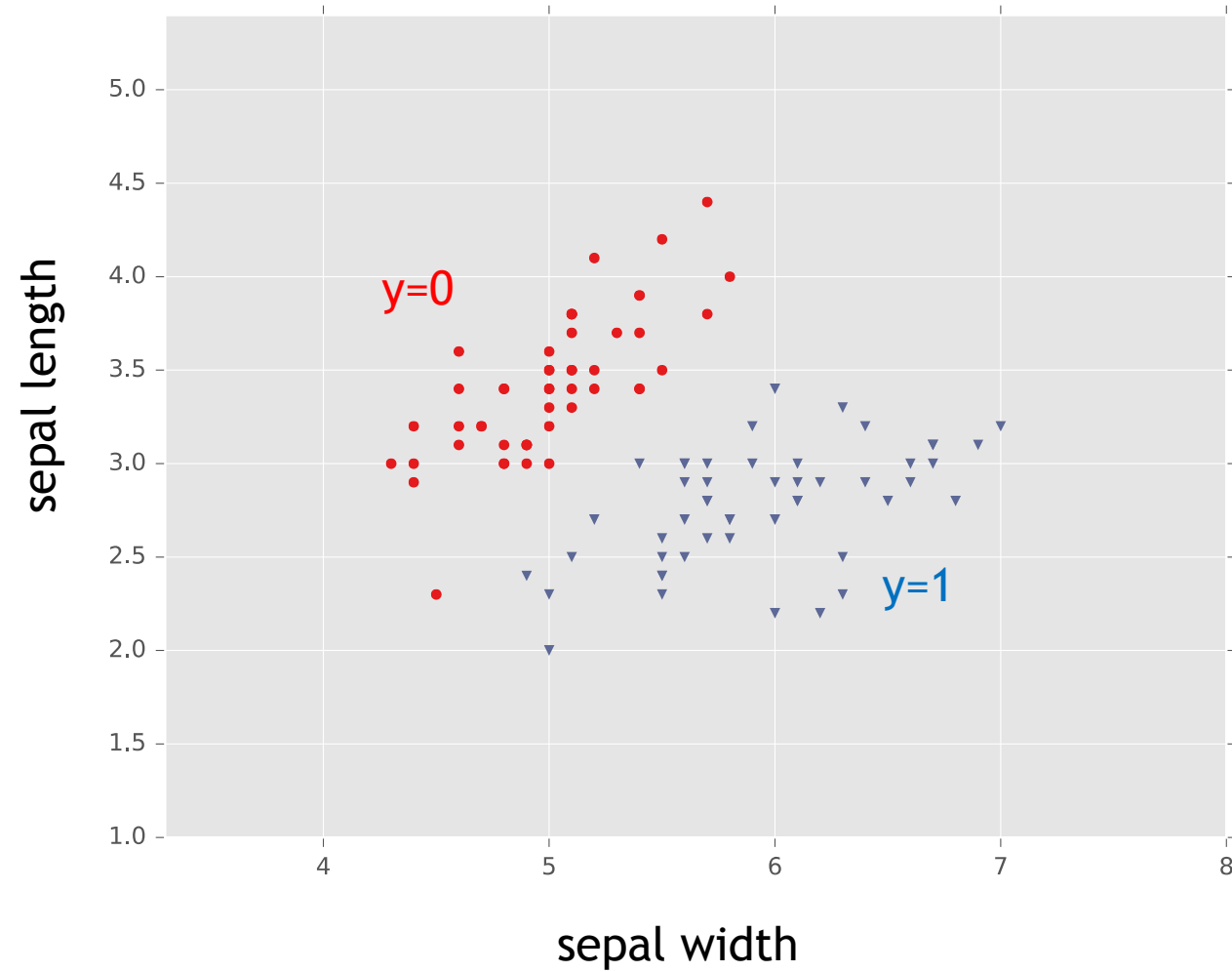
N=7 (here)  
N=150 (total)

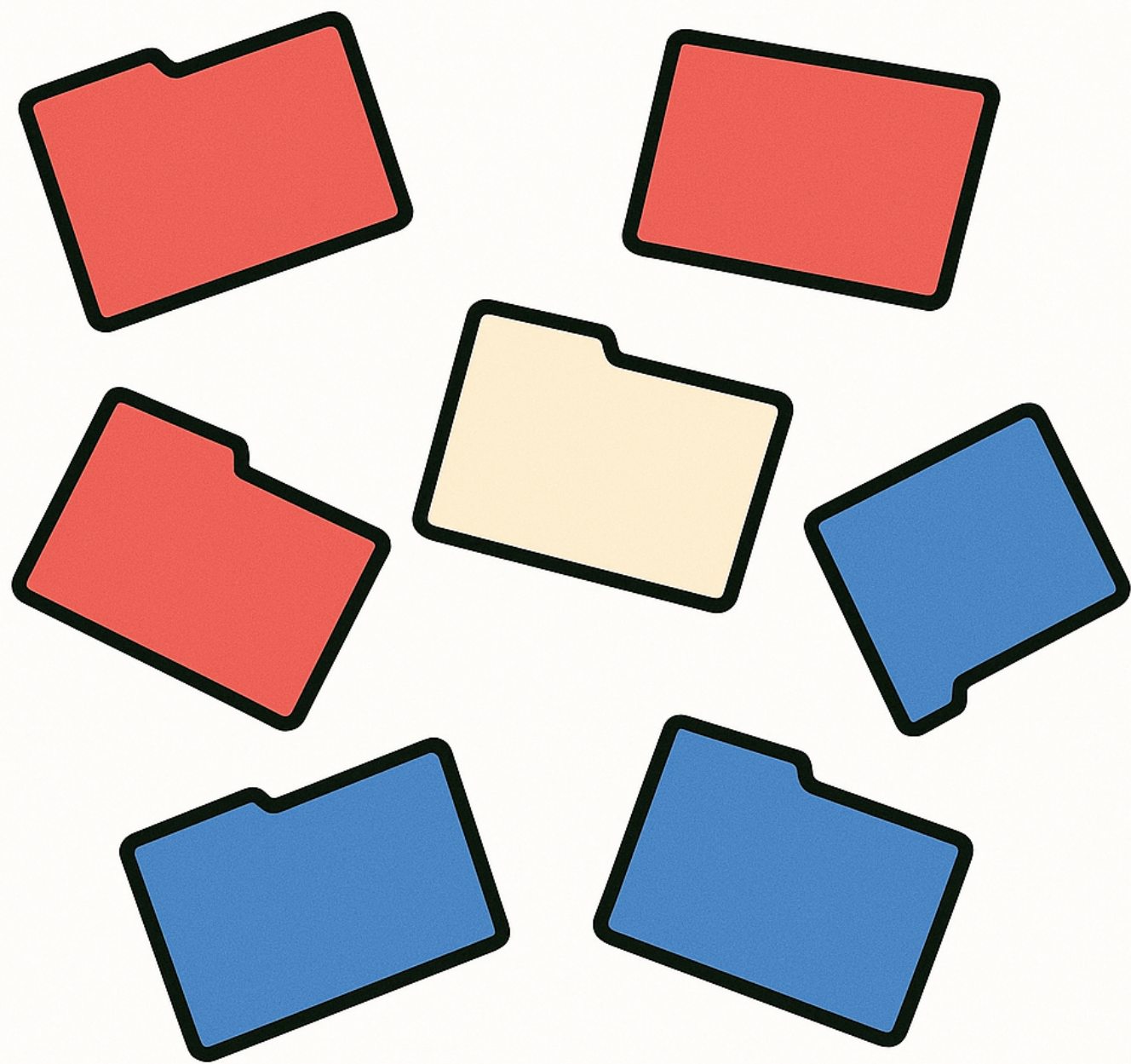
Species	Sepal Length	Sepal Width
0	4.3	3.0
0	4.9	3.6
0	5.3	3.7
1	4.9	2.4
1	5.7	2.8
1	6.3	3.3
1	6.7	3.0

Deleted two of the four features, so that input space is 2D

$M=2$

# Fisher Iris Dataset







# Classification & Real-Valued Features

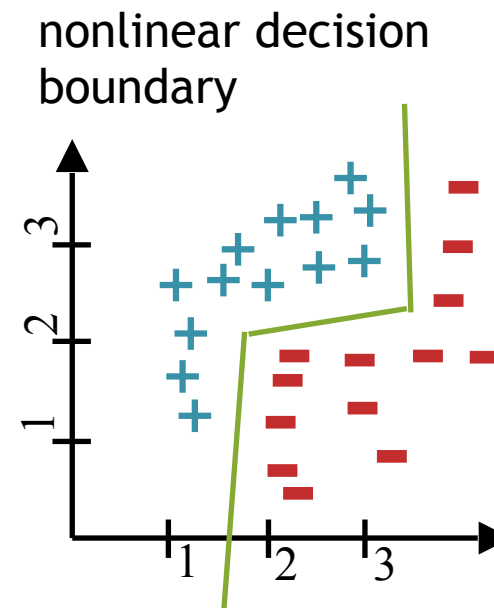
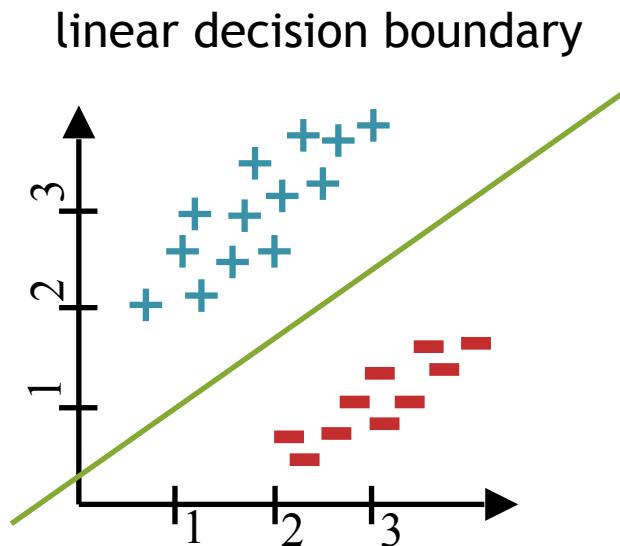
**Def:** Hypothesis (aka. Decision Rule) for Binary Classification

$$h : \mathbb{R}^M \rightarrow \{ +, - \}$$

**Train time:** learn  $h$  by looking only at  $N$  training examples

**Test time:** Given new features  $\mathbf{x}$ , predict  $\hat{y} = h(\mathbf{x})$ , check  $\hat{y} = y$ ?

**Ex:** Decision Boundaries (2D Binary Classification)



# Classification & Real-Valued Features

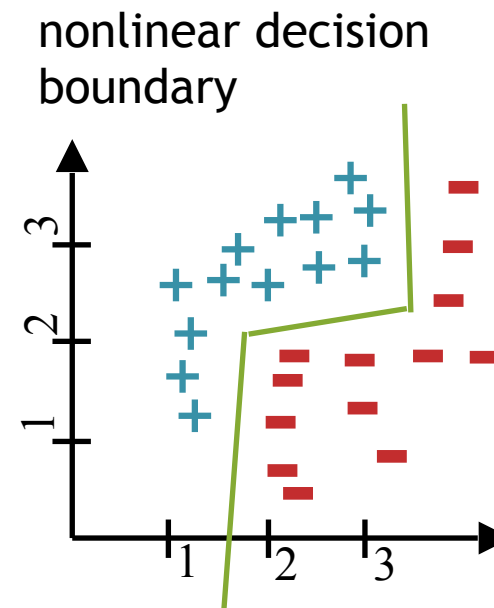
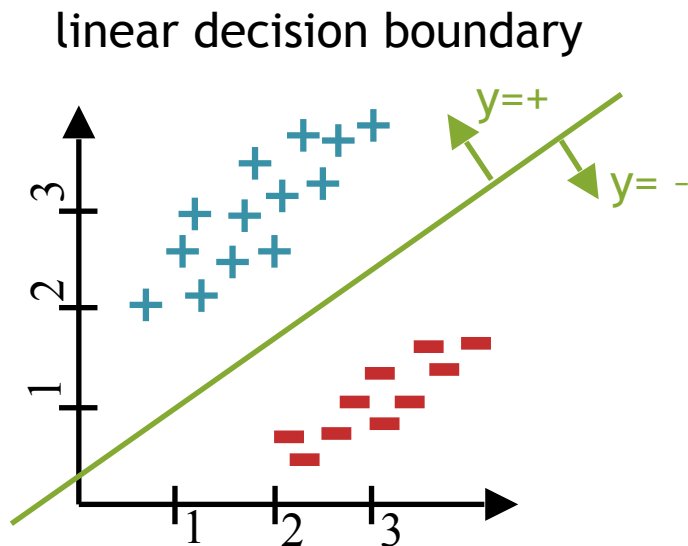
**Def: Hypothesis (aka. Decision Rule) for Binary Classification**

$$h : \mathbb{R}^M \rightarrow \{ +, - \}$$

**Train time:** learn  $h$  by looking only at  $N$  training examples

**Test time:** Given new features  $\mathbf{x}$ , predict  $\hat{y} = h(\mathbf{x})$ , check  $\hat{y} = y$ ?

**Ex: Decision Boundaries (2D Binary Classification)**



# Classification & Real-Valued Features

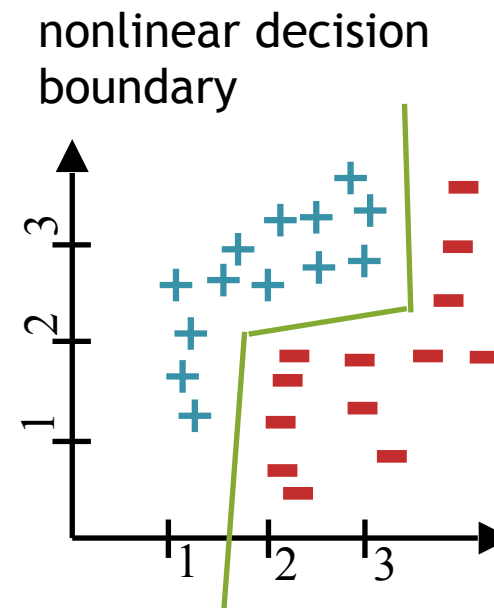
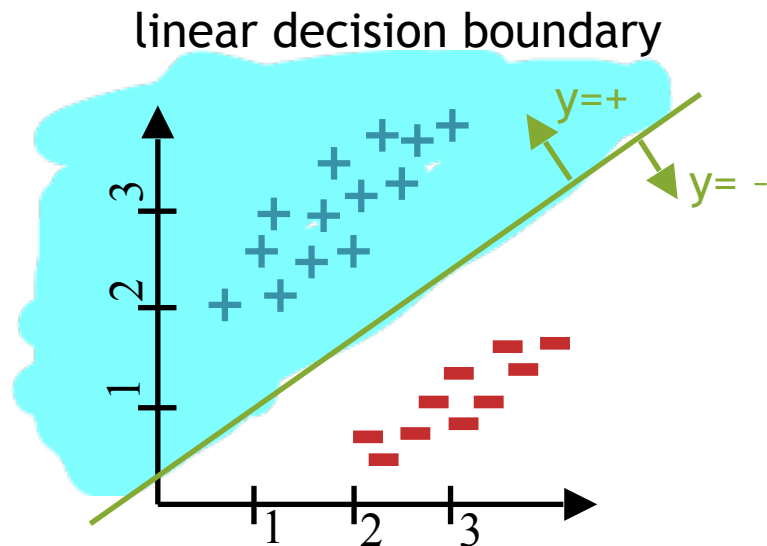
**Def:** Hypothesis (aka. Decision Rule) for Binary Classification

$$h : \mathbb{R}^M \rightarrow \{ +, - \}$$

**Train time:** learn  $h$  by looking only at  $N$  training examples

**Test time:** Given new features  $\mathbf{x}$ , predict  $\hat{y} = h(\mathbf{x})$ , check  $\hat{y} = y$ ?

**Ex:** Decision Boundaries (2D Binary Classification)



# Classification & Real-Valued Features

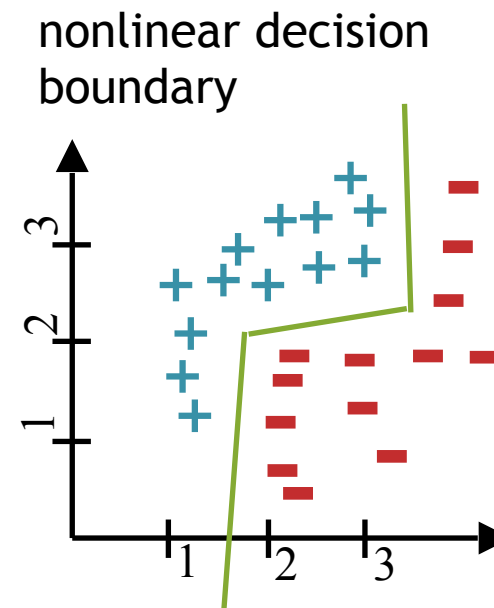
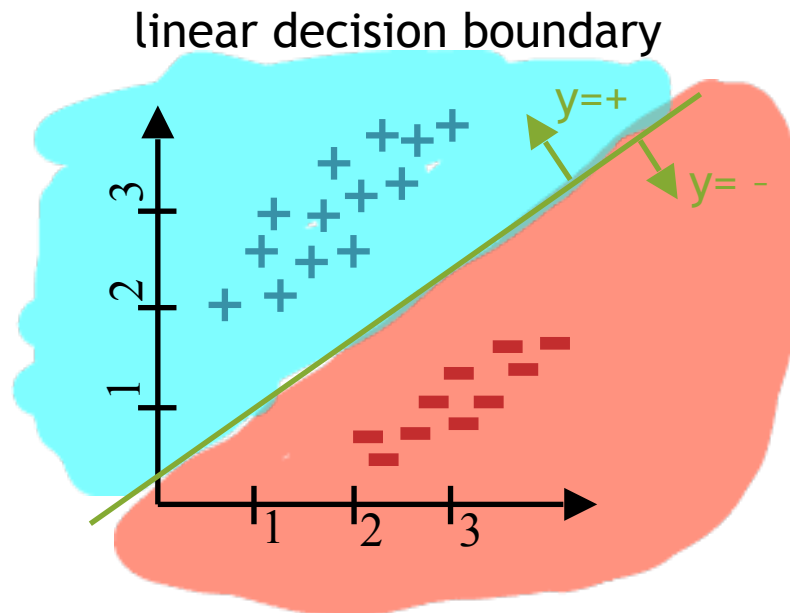
**Def: Hypothesis (aka. Decision Rule) for Binary Classification**

$$h : \mathbb{R}^M \rightarrow \{ +, - \}$$

**Train time:** learn  $h$  by looking only at  $N$  training examples

**Test time:** Given new features  $\mathbf{x}$ , predict  $\hat{y} = h(\mathbf{x})$ , check  $\hat{y} = y$ ?

**Ex: Decision Boundaries (2D Binary Classification)**



# Classification & Real-Valued Features

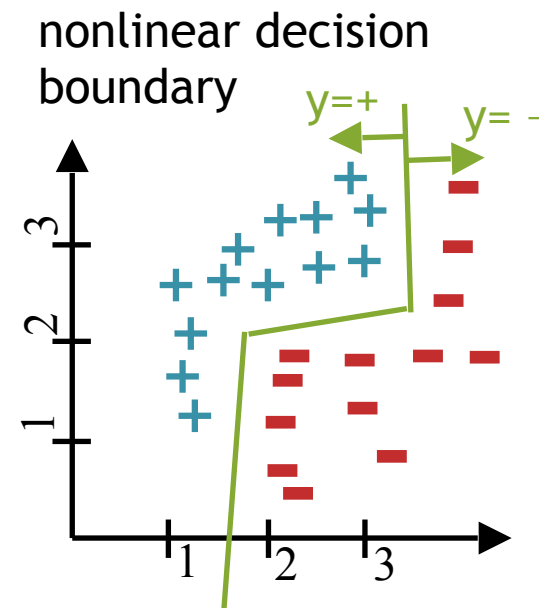
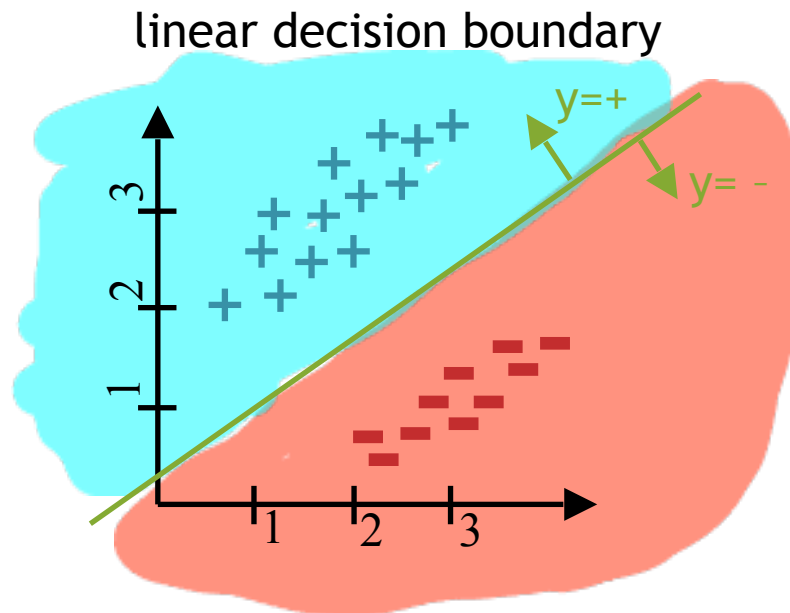
**Def: Hypothesis (aka. Decision Rule) for Binary Classification**

$$h : \mathbb{R}^M \rightarrow \{ +, - \}$$

**Train time:** learn  $h$  by looking only at  $N$  training examples

**Test time:** Given new features  $\mathbf{x}$ , predict  $\hat{y} = h(\mathbf{x})$ , check  $\hat{y} = y$ ?

**Ex: Decision Boundaries (2D Binary Classification)**



# Classification & Real-Valued Features

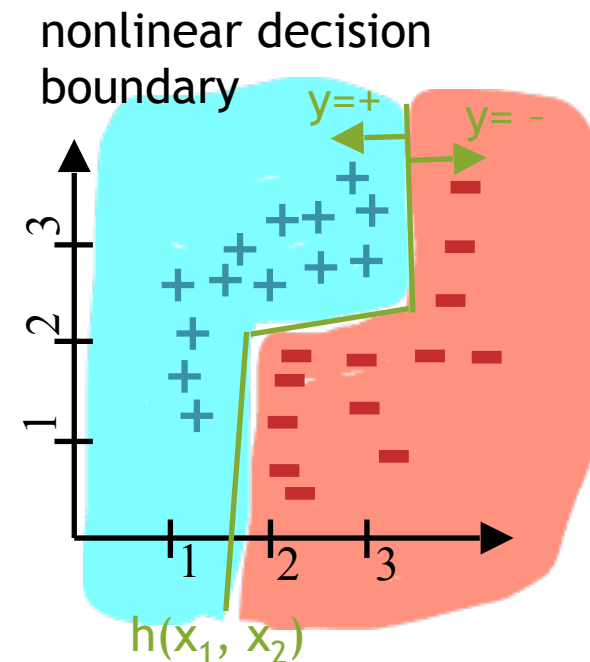
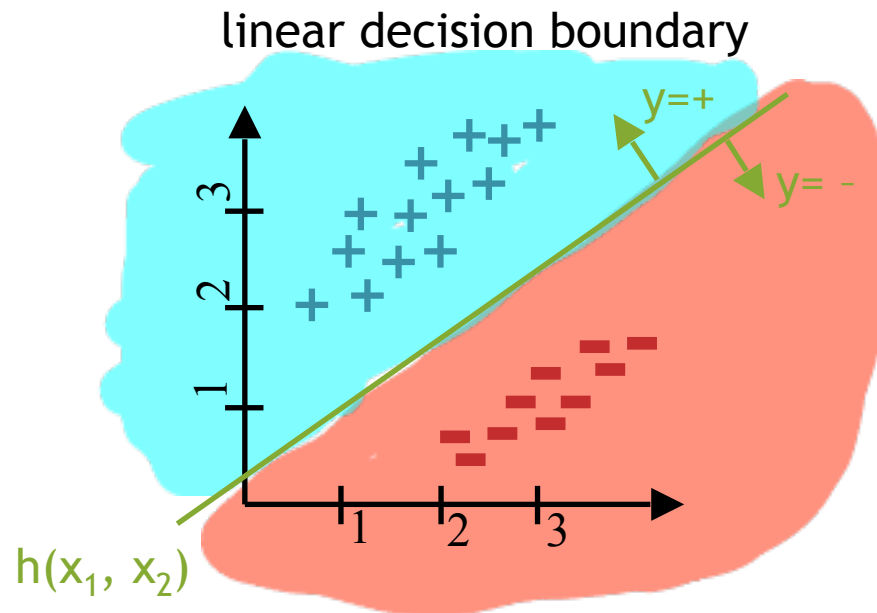
**Def: Hypothesis (aka. Decision Rule) for Binary Classification**

$$h : \mathbb{R}^M \rightarrow \{ +, - \}$$

**Train time:** learn  $h$  by looking only at  $N$  training examples

**Test time:** Given new features  $\mathbf{x}$ , predict  $\hat{y} = h(\mathbf{x})$ , check  $\hat{y} = y$ ?

**Ex: Decision Boundaries (2D Binary Classification)**



# K-NEAREST NEIGHBORS

# Nearest Neighbor: Algorithm

```
def train( $\mathcal{D}$ ):
```

```
    Store  $\mathcal{D}$ 
```

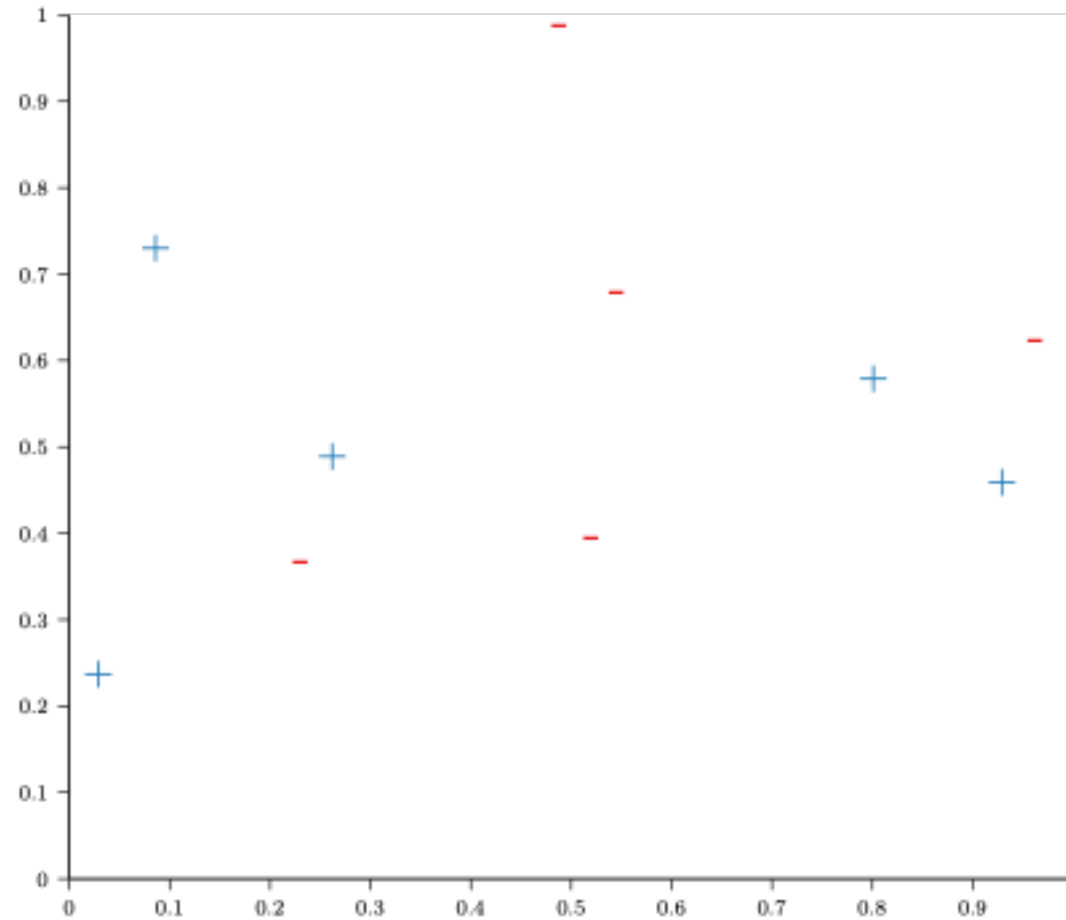
```
def h( $x'$ ):
```

```
    Let  $x^{(i)}$  = the point in  $\mathcal{D}$  that is nearest to  $x'$ 
```

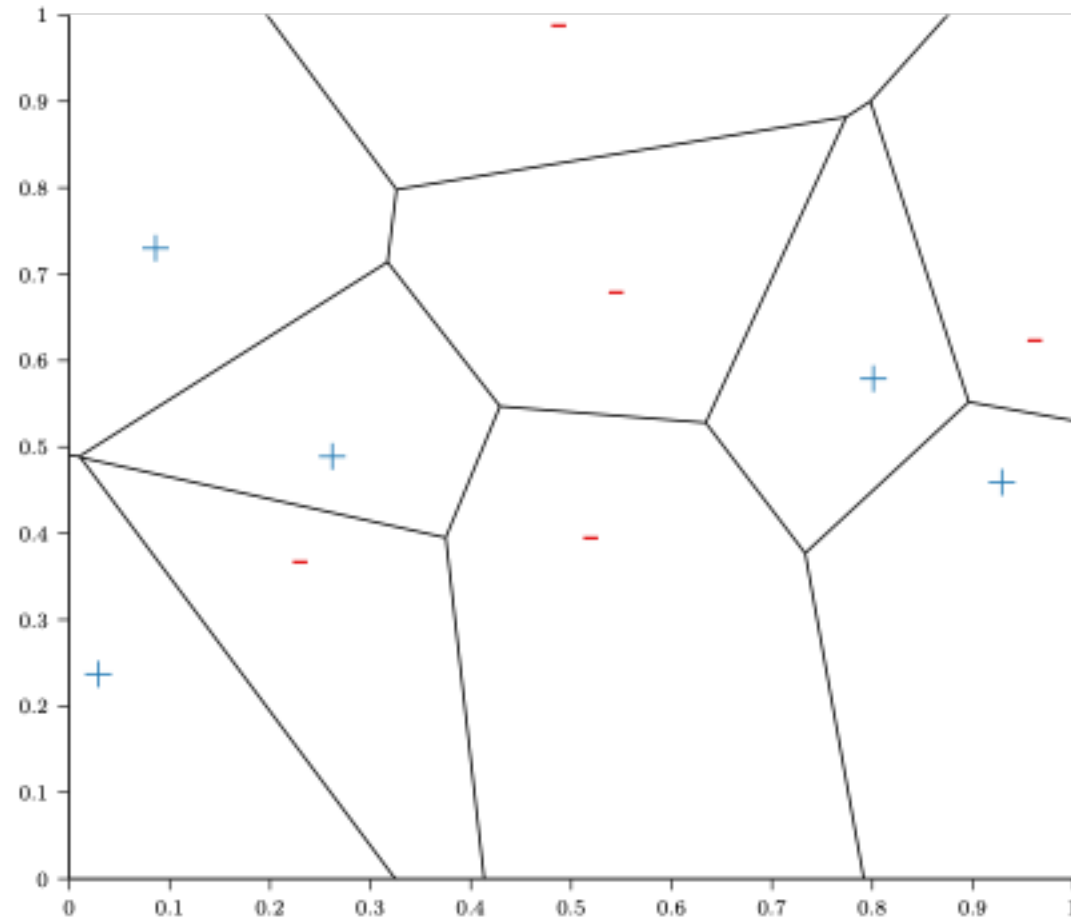
```
    return  $y^{(i)}$ 
```



# Nearest Neighbor: Example

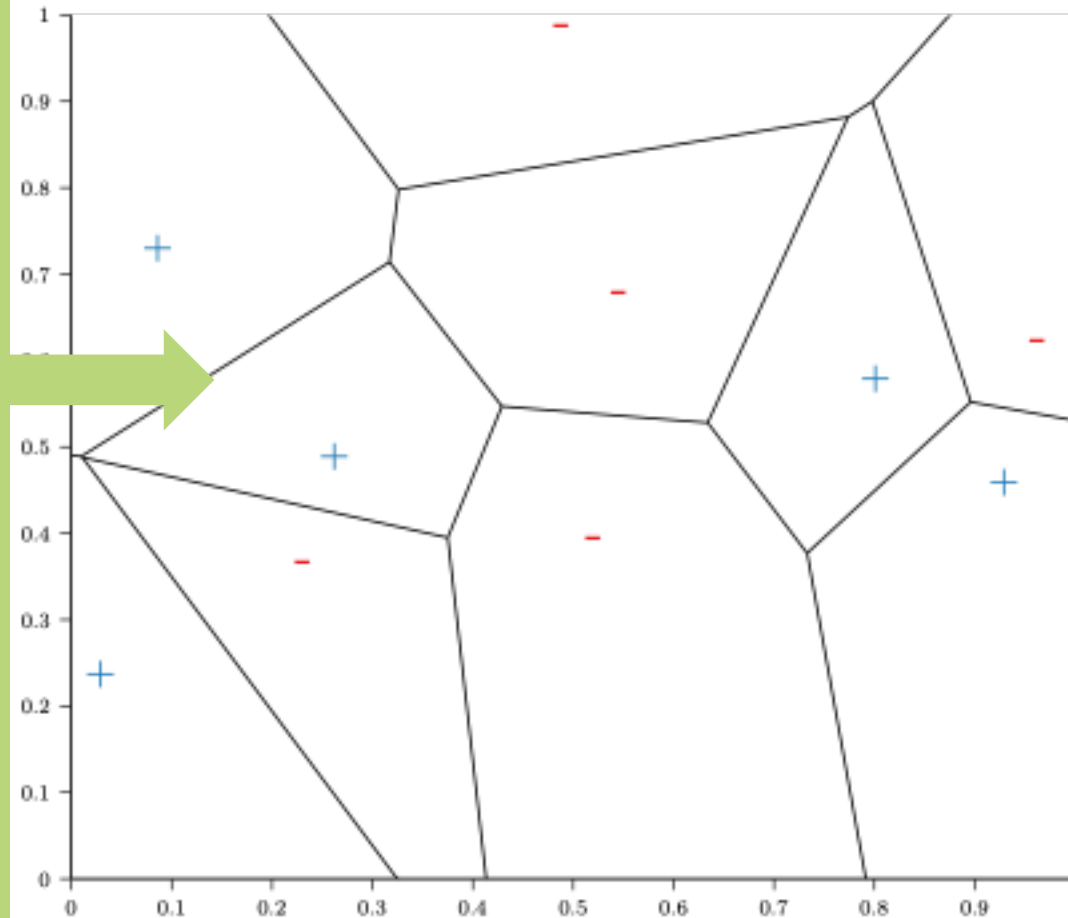


# Nearest Neighbor: Example

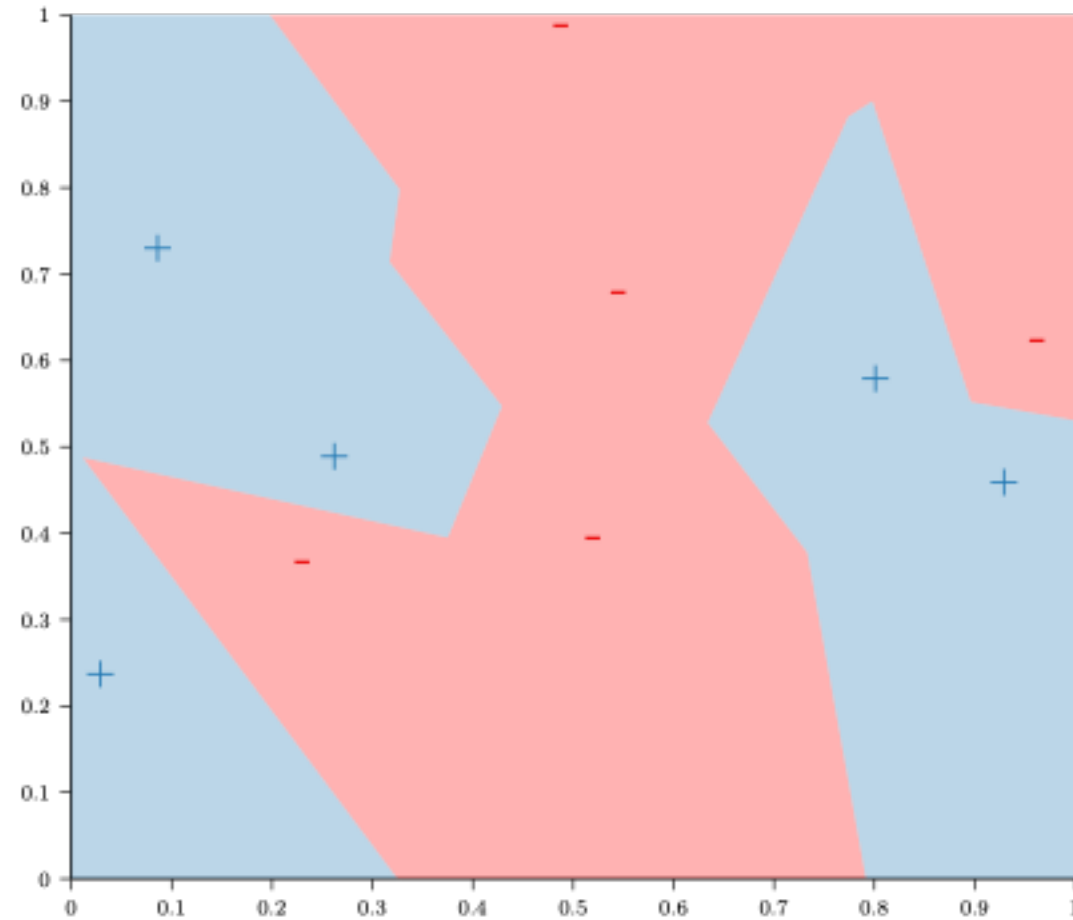


# Nearest Neighbor: Example

- This is a **Voronoi diagram**
- Each **cell** contain one of our training examples
- **All points within a cell** are closer to that training example, than to any other training example
- **Points on the Voronoi line segments** are equidistant to one or more training examples



# Nearest Neighbor: Example



# The Nearest Neighbor Model

- Requires no training!
- Always has zero training error!
  - *A data point is always its own nearest neighbor*

# k-Nearest Neighbors: Algorithm

```
def set_hyperparameters(k, d):
```

```
    Store k
```

```
    Store  $d(\cdot, \cdot)$ 
```

```
def train( $\mathcal{D}$ ):
```

```
    Store  $\mathcal{D}$ 
```

```
def h( $x'$ ):
```

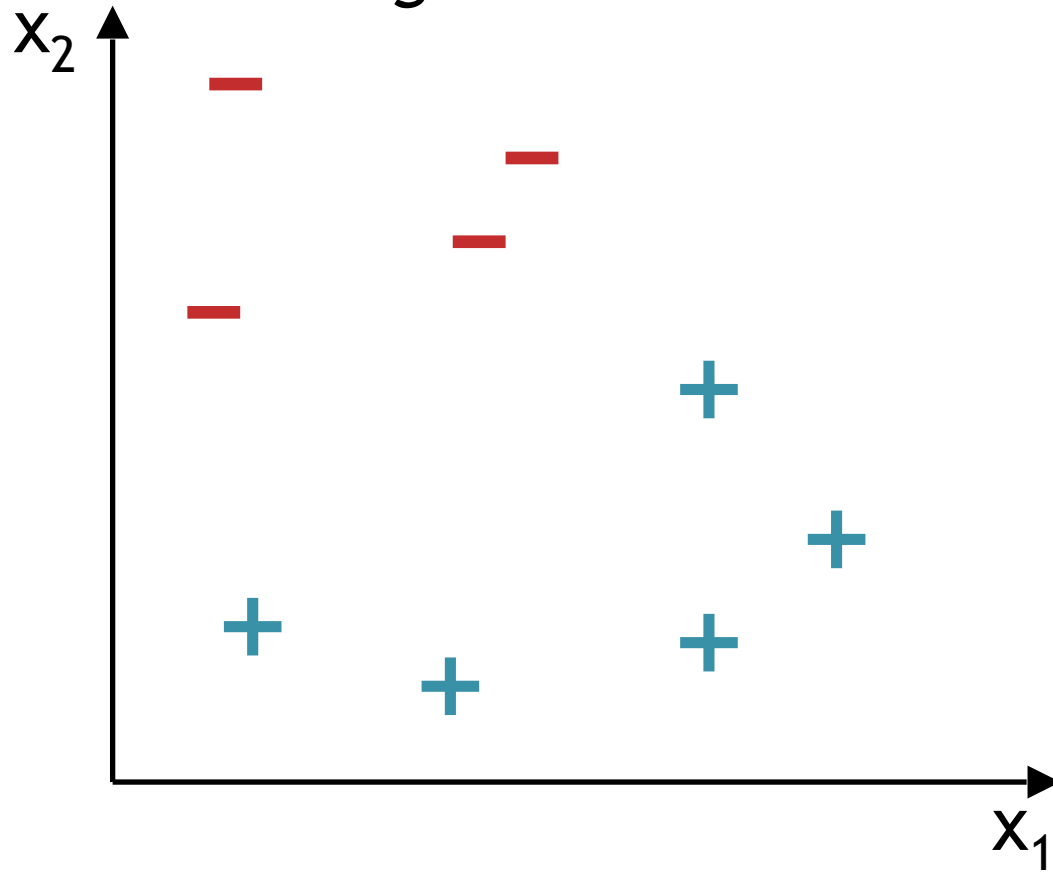
```
    Let  $S$  = the set of  $k$  points in  $\mathcal{D}$  nearest to  $x'$   
             according to distance function  $d(u, v)$ 
```

```
    Let  $v$  = majority_vote( $S$ )
```

```
    return  $v$ 
```

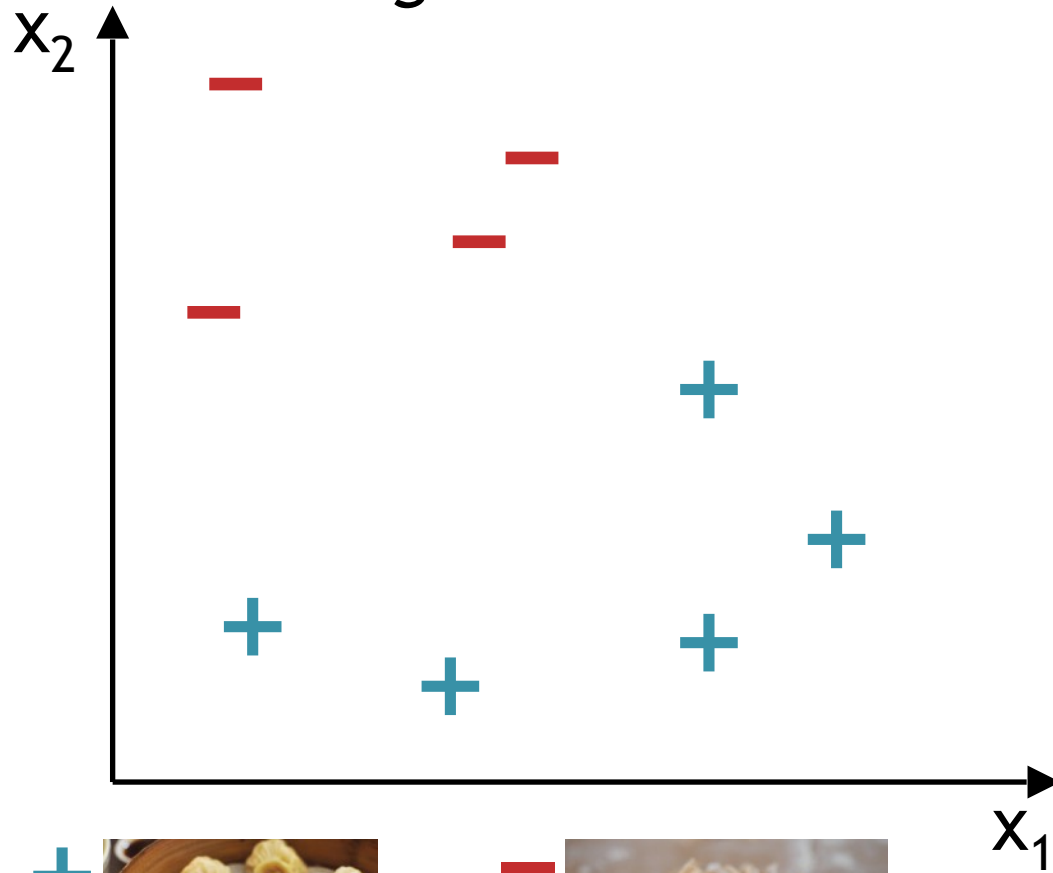
# k-Nearest Neighbors

*Suppose we have the training dataset below.*



# k-Nearest Neighbors

*Suppose we have the training dataset below.*



+



-

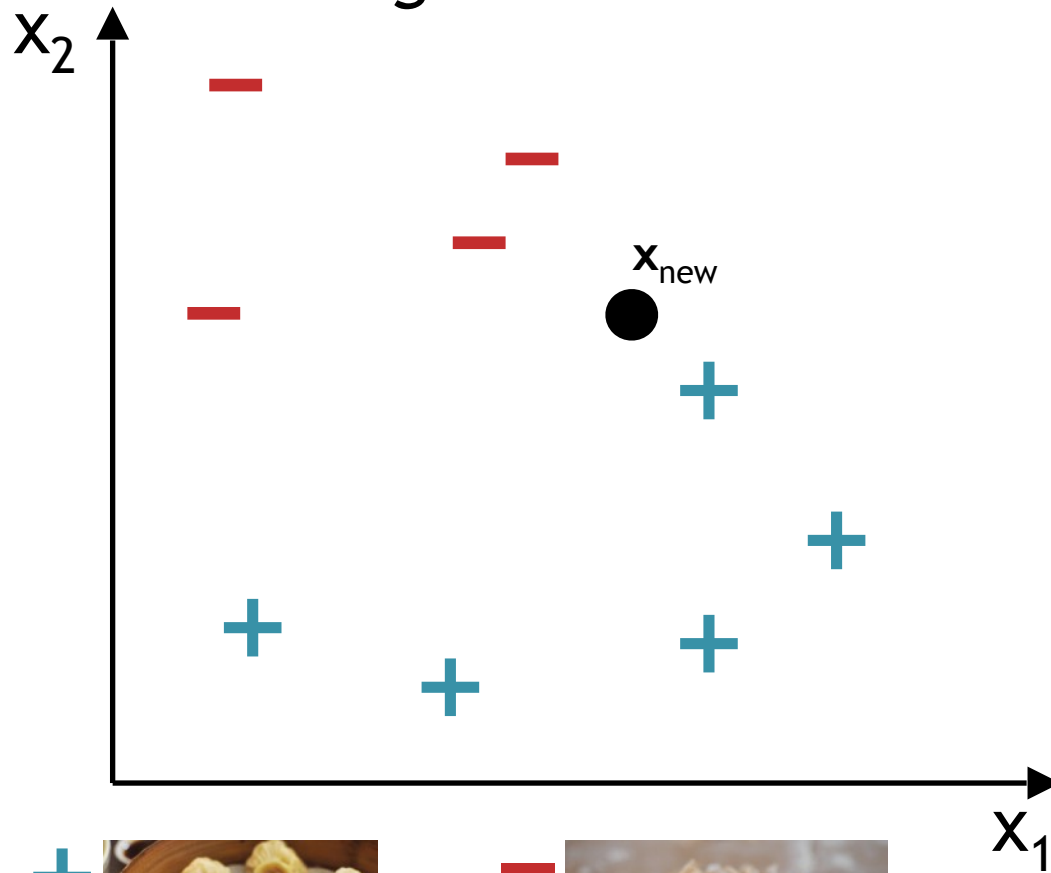




# k-Nearest Neighbors

*Suppose we have the training dataset below.*

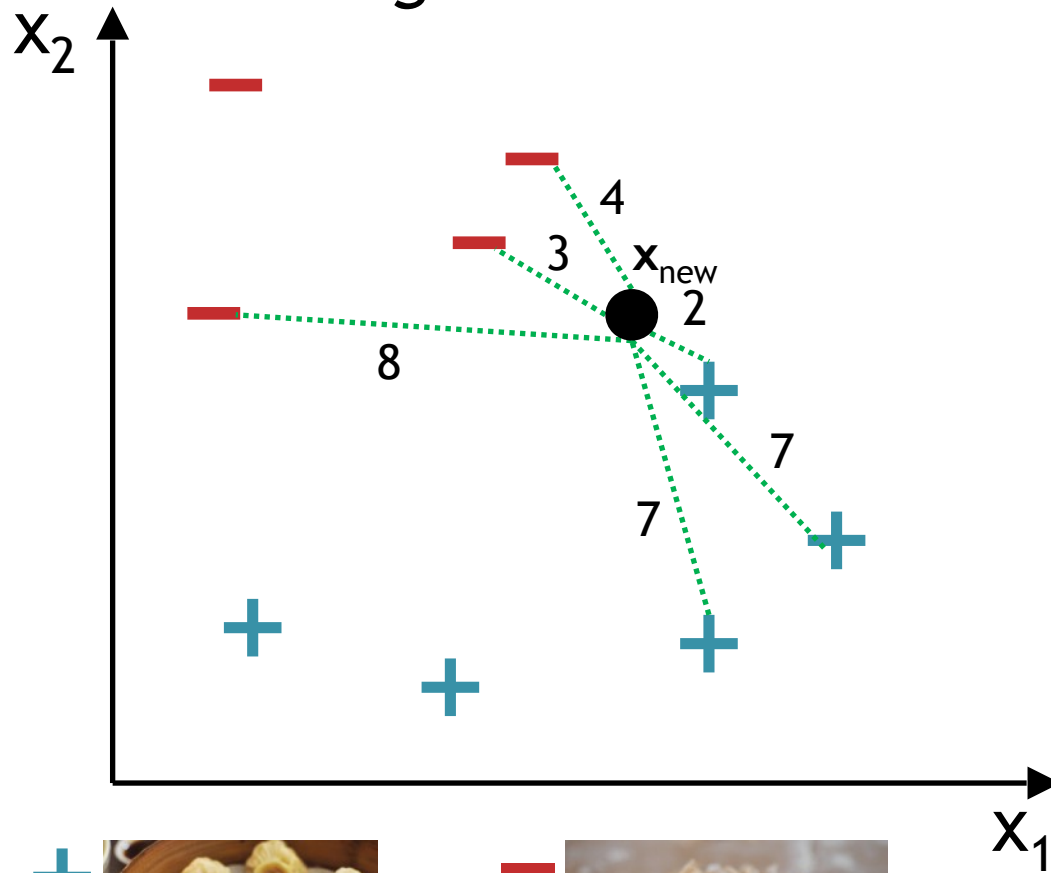
*How should we label the new point?*



# k-Nearest Neighbors

*Suppose we have the training dataset below.*

*How should we label the new point?*



+

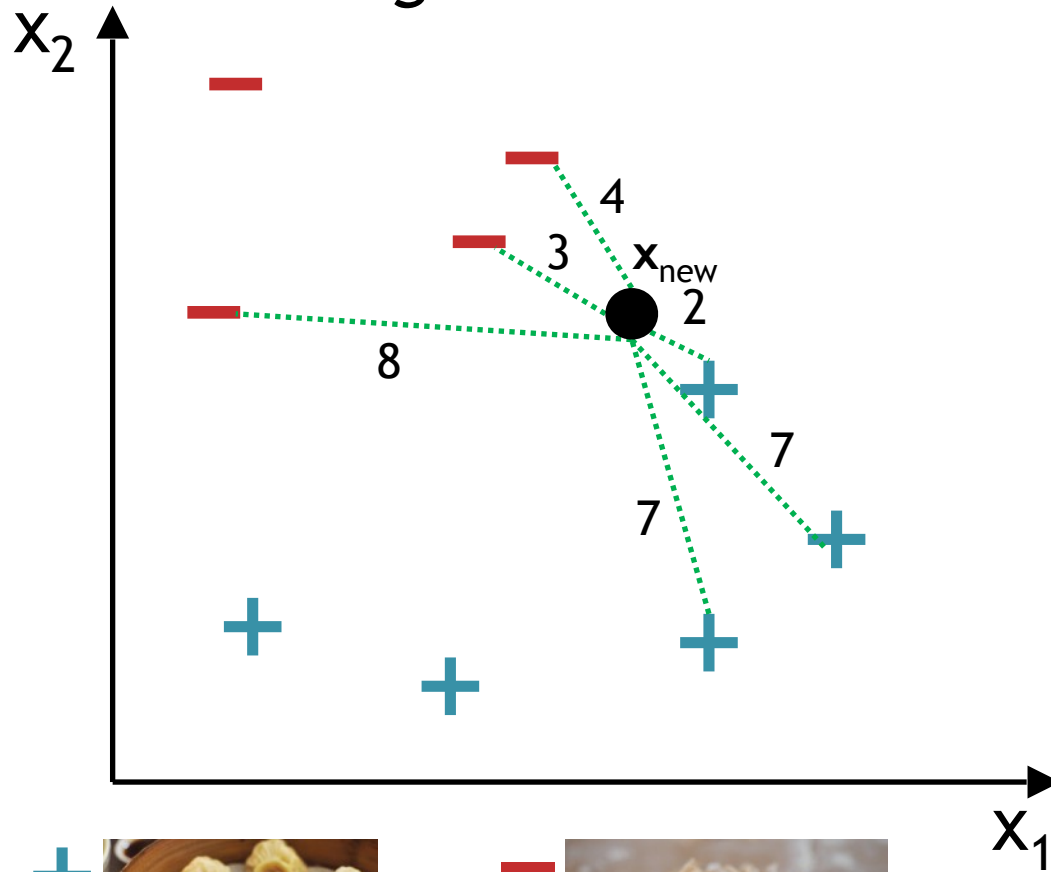


-



# k-Nearest Neighbors

*Suppose we have the training dataset below.*



*How should we label the new point?*

It depends on  $k$ :

if  $k=1$ ,  $h(\mathbf{x}_{\text{new}}) = +1$

if  $k=3$ ,  $h(\mathbf{x}_{\text{new}}) = -1$

if  $k=5$ ,  $h(\mathbf{x}_{\text{new}}) = +1$

# KNN: Remarks

## Distance Functions:

- KNN requires a **distance function**

$$d : \mathbb{R}^M \times \mathbb{R}^M \rightarrow \mathbb{R}$$

- The most common choice is **Euclidean distance**

$$d(\mathbf{u}, \mathbf{v}) = \sqrt{\sum_{m=1}^M (u_m - v_m)^2}$$

- But there are other choices (e.g. **Manhattan distance**)

$$d(\mathbf{u}, \mathbf{v}) = \sum_{m=1}^M |u_m - v_m|$$

# KNN: Computational Efficiency

- $N$  training examples, each one w/  $M$  features
- Computational complexity when  $k=1$ :

Task	Naive	Smart data structure: ball tree, kd-tree, neighborhood graph ( <i>exact NN</i> )	Smart data structure ( <i>approximate NN</i> )
Train	$O(MN)$ or $O(1)$	$O(N \log N)$ if $M=1,2$ ; in general $O(MN^2)$ worst case	$O(MN \log N)$
Predict (one test example)	$O(MN)$	$O(\log N)$ if $M=1,2$ ; in general $O(MN)$ worst case	$O(M \log N)$

**Problem:** Exact can be fast for small  $M$ , but slow for large  $M$

**If approximate is good enough:** can still be very fast!

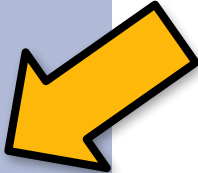
# KNN: Theoretical Guarantees

Cover & Hart (1967)

Let  $h(x)$  be a Nearest Neighbor ( $k=1$ ) binary classifier. As the number of training examples  $N$  goes to infinity...

$$\text{error}_{\text{true}}(h) < 2 \times \text{Bayes Error Rate}$$

“In this sense, it may be said that half the classification information in an infinite sample set is contained in the nearest neighbor.”



**very informally,**  
Bayes Error Rate =  
*‘the best you could  
possibly do’*