Our second ML method

Memorizer: if we've seen *exact same* inputs before, predict corresponding output, else majority

features | labels

Family History	Resting Blood Pressure	Cholesterol	Heart Disease?
Yes	Low	Normal	No
No	Medium	Normal	No
No	Low	Abnormal	Yes
Yes	Medium	Normal	Yes
Yes	High	Abnormal	Yes

training data points

No	Low	Normal	No
No	Medium	Normal	No
Yes	Medium	Abnormal	Yes

testing data points

training error rate =

testing error rate =

Our second ML method

Memorizer: if we've seen *exact same* inputs before, predict corresponding output, else majority

features | labels | predictions

	Family History	Resting Blood Pressure	Cholesterol	Heart Disease?	Model output
training data points	Yes	Low	Normal	No	No
	No	Medium	Normal	No	No
	No	Low	Abnormal	Yes	Yes
	Yes	Medium	Normal	Yes	Yes
	Yes	High	Abnormal	Yes	Yes
testing data points	No	Low	Normal	No	
	No	Medium	Normal	No	
	Yes	Medium	Abnormal	Yes	

training error rate =

testing error rate =

Our second ML method

Memorizer: if we've seen *exact same* inputs before, predict corresponding output, else majority

features | labels | predictions

	Family History	Resting Blood Pressure	Cholesterol	Heart Disease?	Model output
training data points	Yes	Low	Normal	No	No
	No	Medium	Normal	No	No
	No	Low	Abnormal	Yes	Yes
	Yes	Medium	Normal	Yes	Yes
	Yes	High	Abnormal	Yes	Yes
testing data points	No	Low	Normal	No	Yes
	No	Medium	Normal	No	No
	Yes	Medium	Abnormal	Yes	Yes

training error rate =

testing error rate =

Two different learners

- Same goal, two different learners
- From the same data, each learner chooses a ***different*** hypothesis
  - *majority* favors a simpler hypothesis, has ~similar training and test error
  - *memorizer* favors a more complex hypothesis, training error is much lower than test
- Difference is called ***inductive bias***
  - criteria that cause an ML method to favor one hypothesis over another when data isn't enough to determine
  - "bias" sounds bad, but it is ***unavoidable*** — learning means filling in information that isn't in the data, so we have to have a way to choose

Overfitting

- For *majority*, train error $\approx$ test error, while for *memorizer*, train error $\ll$ test error
  - we say that *memorizer* suffers from ***overfitting***
- Overfitting = test error – train error
  - more generally, $P - \hat{P}$ (for min) or $\hat{P} - P$ (for max)
  - lots of overfitting means big difference
  - has nothing to do with whether P or $\hat{P}$ are large or small on their own
- Heuristic: “simpler” models tend to overfit less
  - we’ll try later to measure simplicity — not simple to do!

Underfitting

- On the other hand, *majority* doesn't seem like a great learner — we'll see other methods later that do better
- This is called ***underfitting***
  - overfitting is absent or minimal, and another method could achieve lower error while still limiting overfitting
  - heuristically, we kept the hypothesis *too* simple

Notation

- Data point = $\langle x^{(t)}, y^{(t)} \rangle$
- Feature vector and feature space $x^{(t)} \in \mathcal{X}$
- Label and label space $y^{(t)} \in \mathcal{Y}$
- Hypothesis (classifier) and hypothesis space $h \in \mathcal{H}$
- ***Unknown*** target or true classifier h^*
  - best possible test error, might not even be in $\mathcal{H}$
- Apparent best classifier $\hat{h} \in \mathcal{H}$ given training set (sometimes called ERM, empirical risk minimizer)
  - minimizes training error over $\mathcal{H}$
- ML method: a way of picking a hypothesis given data
  - e.g., majority vote = empirical risk minimization over constant classifiers

Learning goals

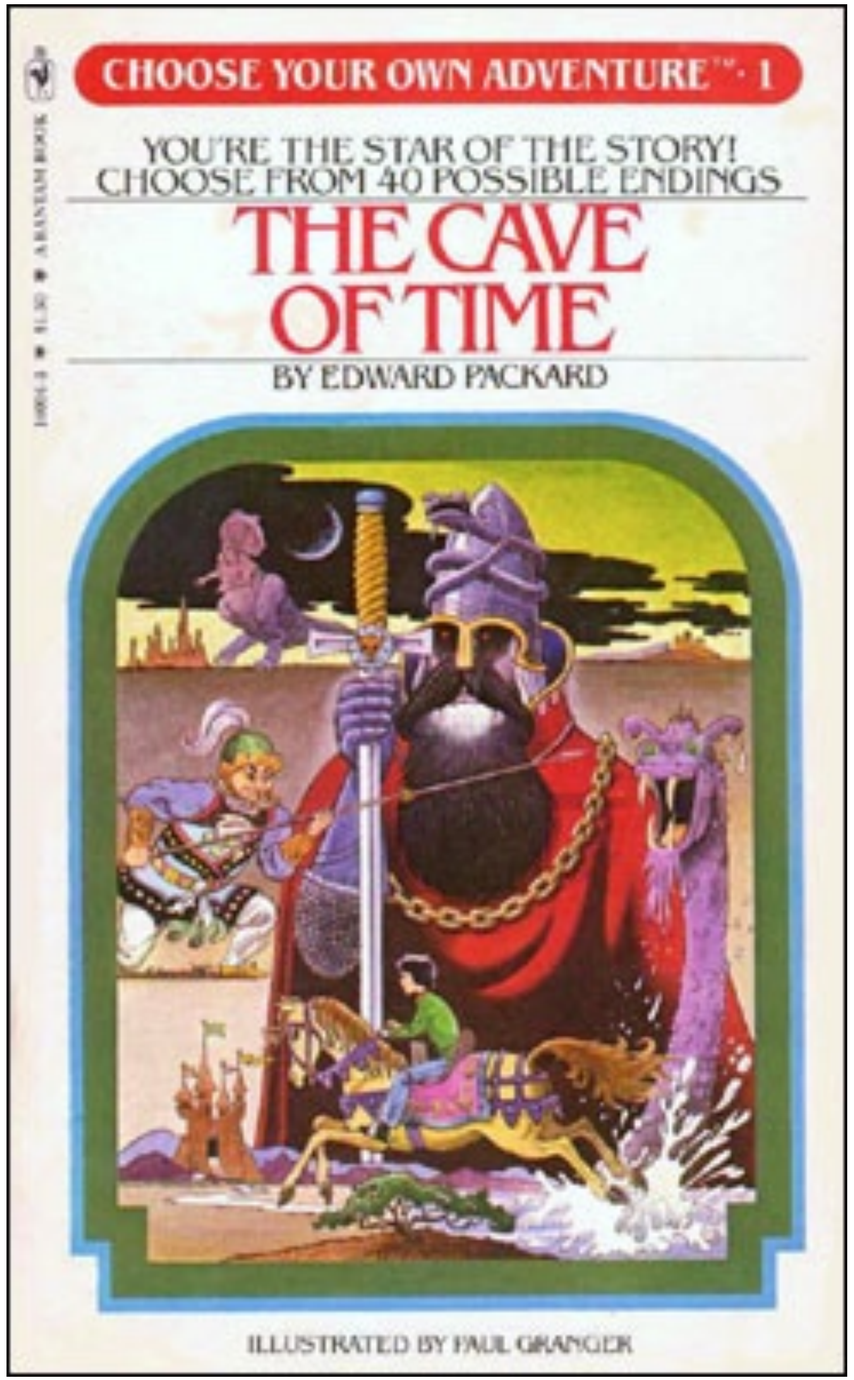
- You should be able to
  - Formulate a well-posed learning problem for a real- world task by identifying the task, performance measure, and training experience
  - Describe the supervised learning paradigm in terms of the type of data needed, the form of prediction, and the structure of the output prediction
  - Explain the difference between memorization and generalization

Decision trees

10-701 Introduction to Machine Learning
Geoff Gordon and Pradeep Ravikumar

(thanks to Matt Gormley and Henry Chai for some content)

Next ML method: decision trees



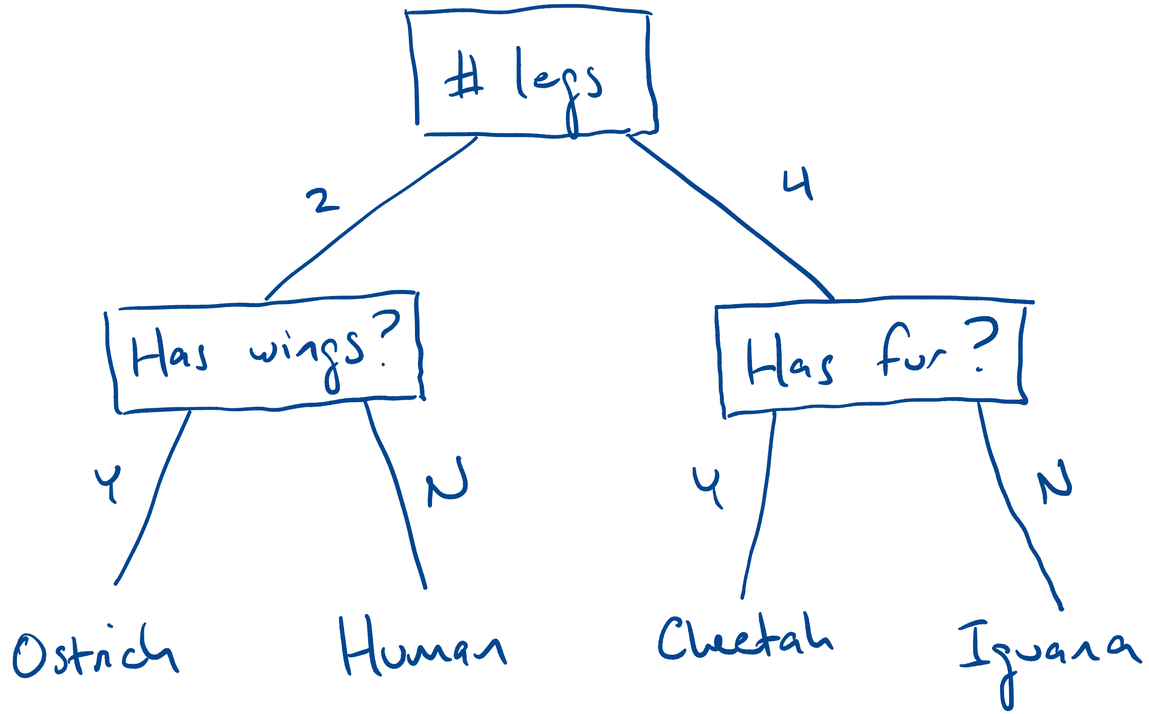
Choose your own adventure!

If you decide to open the creepy sarcophagus, turn to p63

If you decide to back slowly out of the room, turn to p17

If you decide to set up camp for the night, turn to p42

Decision tree

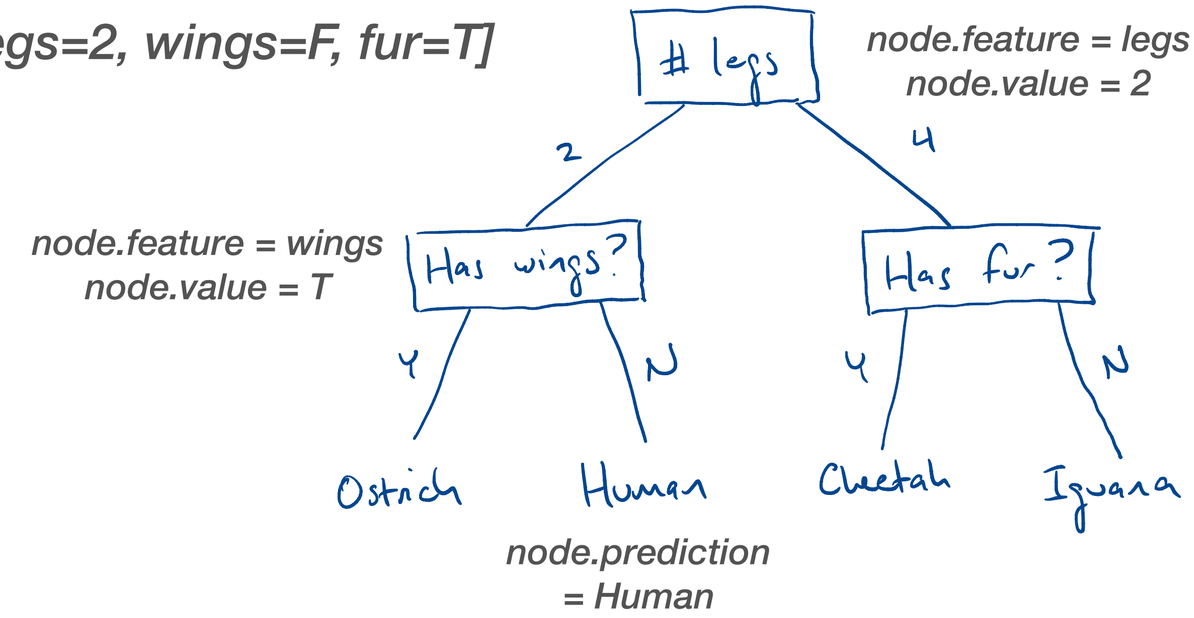


- A question in each node, an answer on each edge
  - answers mutually exclusive and exhaustive
- Each question splits the data into two (or more) pieces
- When we reach a leaf, it makes a prediction

Supplemental reading: Hastie & Tibshirani, ch 9

Decision tree prediction pseudocode

x = [legs=2, wings=F, fur=T]



- function predict(x):
  - node ← root
  - while not leaf(node):
    - test ← (x[node.feature] = node.value)
    - if test:
      - node ← node.left
    - else:
      - node ← node.right
  - return node.prediction

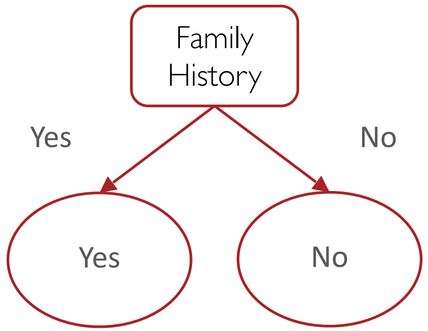
Decision tree example

features

labels

training data points

Family History	Resting Blood Pressure	Cholesterol	Heart Disease?
Yes	Low	Normal	No
No	Medium	Normal	No
No	Low	Abnormal	Yes
Yes	Medium	Normal	Yes
Yes	High	Abnormal	Yes

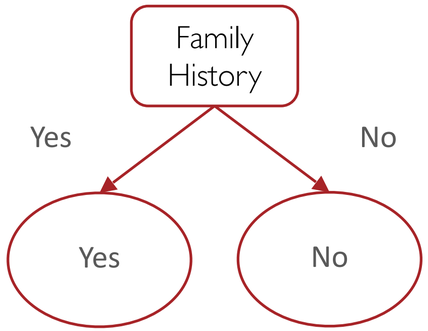


Decision tree example

features labels predictions

training data points

Family History	Resting Blood Pressure	Cholesterol	Heart Disease?	Model output
Yes	Low	Normal	No	Yes
No	Medium	Normal	No	No
No	Low	Abnormal	Yes	No
Yes	Medium	Normal	Yes	Yes
Yes	High	Abnormal	Yes	Yes

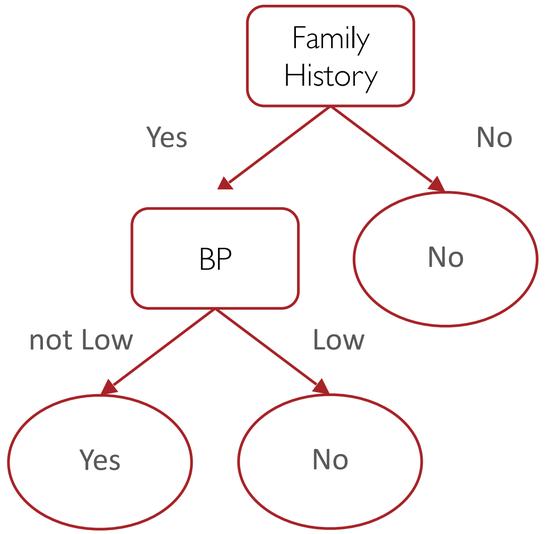


Decision tree example

features | labels | predictions

training data points

Family History	Resting Blood Pressure	Cholesterol	Heart Disease?	Model output
Yes	Low	Normal	No	Yes
No	Medium	Normal	No	No
No	Low	Abnormal	Yes	No
Yes	Medium	Normal	Yes	Yes
Yes	High	Abnormal	Yes	Yes

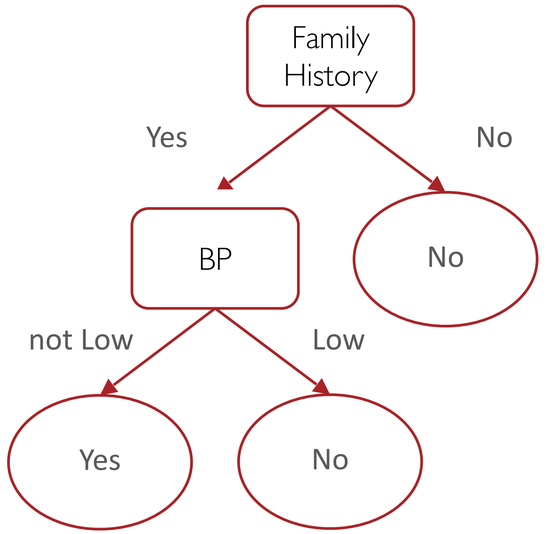


Decision tree example

features labels predictions

training data points

Family History	Resting Blood Pressure	Cholesterol	Heart Disease?	Model output
Yes	Low	Normal	No	No
No	Medium	Normal	No	No
No	Low	Abnormal	Yes	No
Yes	Medium	Normal	Yes	Yes
Yes	High	Abnormal	Yes	Yes

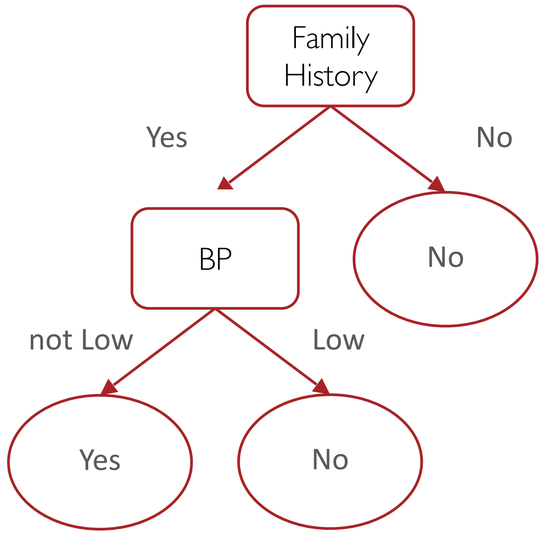


Decision tree example

features | labels | predictions

training data points

Family History	Resting Blood Pressure	Cholesterol	Heart Disease?	Model output
Yes	Low	Normal	No	No
No	Medium	Normal	No	No
No	Low	Abnormal	Yes	No
Yes	Medium	Normal	Yes	Yes
Yes	High	Abnormal	Yes	Yes

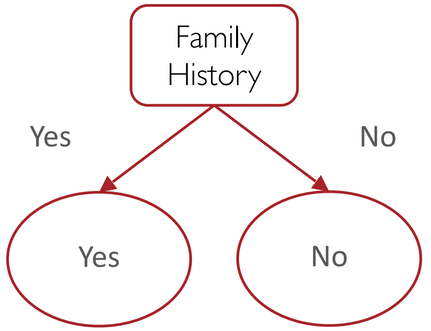


if just one split: "decision stump"

Splitting criterion

Family History	Resting Blood Pressure	Cholesterol	Heart Disease?
Yes	Low	Normal	No
No	Medium	Normal	No
No	Low	Abnormal	Yes
Yes	Medium	Normal	Yes
Yes	High	Abnormal	Yes

- How do we choose which features to split on?
  - example criterion: minimize training error



→ 2 errors total

Yes	Low	Normal	No
Yes	Medium	Normal	Yes
Yes	High	Abnormal	Yes

1 error

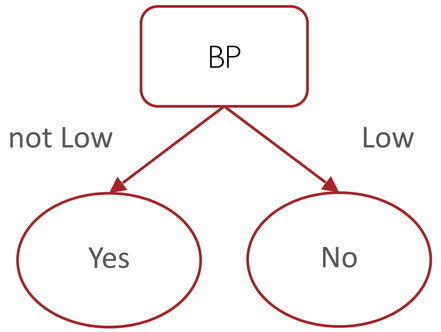
No	Medium	Normal	No
No	Low	Abnormal	Yes

1 error

Splitting criterion

Family History	Resting Blood Pressure	Cholesterol	Heart Disease?
Yes	Low	Normal	No
No	Medium	Normal	No
No	Low	Abnormal	Yes
Yes	Medium	Normal	Yes
Yes	High	Abnormal	Yes

- How do we choose which features to split on?
  - example criterion: minimize training error



→ 2 errors total

No	Medium	Normal	No
Yes	Medium	Normal	Yes
Yes	High	Abnormal	Yes

1 error

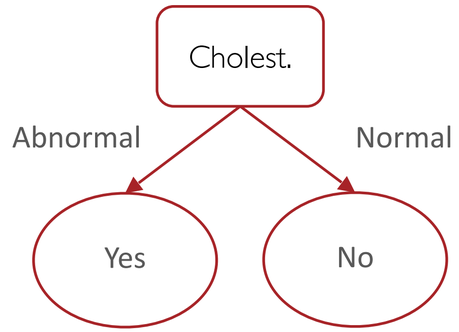
Yes	Low	Normal	No
No	Low	Abnormal	Yes

1 error

Splitting criterion

Family History	Resting Blood Pressure	Cholesterol	Heart Disease?
Yes	Low	Normal	No
No	Medium	Normal	No
No	Low	Abnormal	Yes
Yes	Medium	Normal	Yes
Yes	High	Abnormal	Yes

- How do we choose which features to split on?
  - example criterion: minimize training error



→ 1 error total, ***best!***

No	Low	Abnormal	Yes
Yes	High	Abnormal	Yes

0 errors

Yes	Low	Normal	No
No	Medium	Normal	No
Yes	Medium	Normal	Yes

1 error

Decision tree learning

- We could ask for the decision tree that minimizes error on training set (subject to constraints, e.g., max depth)
  - but this is a difficult combinatorial optimization problem
- Instead, typically use a heuristic search
  - find a good tree but maybe not the absolute minimum
- Most common: greedy top-down search
  - pick the split at the root according to split criterion
  - recursively learn left and right children, using the appropriate subset of the data for each
  - when we stop (e.g., threshold on depth or number of examples in subtree) use *majority* classifier on the leaf

Decision tree learning pseudocode

- function learn(dataset X, labels Y):
  - if stopping_criterion:
    - n ← new leaf, n.prediction = majority(Y)
  - else:
    - splits ← list of allowed split tests # e.g., feature j = value
    - for s in splits:
      - criterion[s] ← evaluate(X, Y, s) # compute split criterion
    - best_s ← argmax(criterion)
    - mask ← apply splits[best_s] to get T/F for each row of X
    - n ← new node
    - n.split = splits[best_s]
    - n.left ← learn(X[mask,:], Y[mask])
    - n.right ← learn(X[~mask,:], Y[~mask])
  - return n

Non-binary features

- Often we have features with more than two values
  - color ∈ {red, green, blue, cyan, magenta, yellow}
  - length ∈ [2cm, 7cm]
- Typically we still use ***binary*** splits when learning a tree
  - decision tree learning works better if we grow gradually
- E.g., [color = cyan] or [length ≤ 4.2cm]
- Might need to test several possible splits per feature
  - but still finitely many (even for continuous features!)

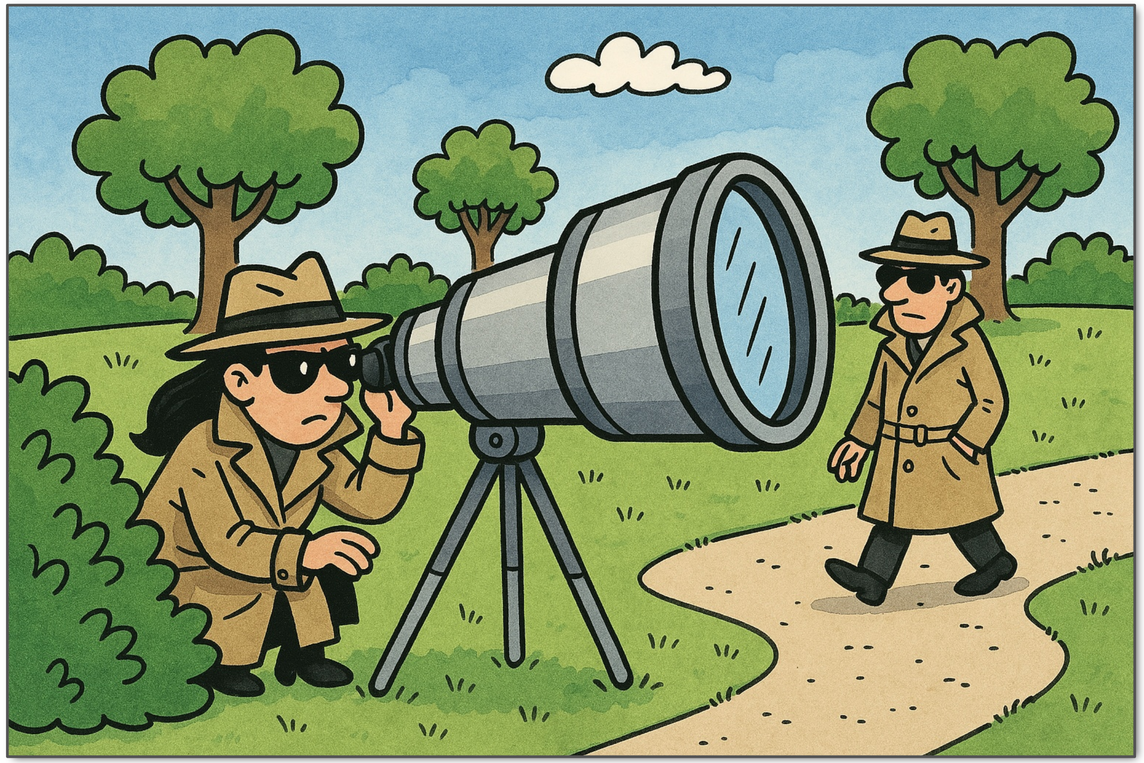
Splitting criteria

- Common criteria
  - minimize training error
  - maximize mutual information → ID3
  - minimize Gini impurity criterion → CART

Splitting criteria

- Common criteria
  - minimize training error
  - maximize mutual information → ID3
  - minimize Gini impurity criterion → CART

Communication channel



- Every morning, Bob finds out a new secret fact: he discovers what the prime minister had for breakfast
- Immediately, he walks in the park, sits at a specific bench, and arranges the rocks next to it in one of four patterns
- Alice observes through a telescope and learns Bob’s message. She informs her superiors.

Code book

meaning	code word	
pancakes	00	
shredded wheat	10	
bagels	01	
caviar	11	

- This channel has a *rate* of 2 bits per day
  - we could speed it up by having more arrangements of rocks, having Bob take multiple walks per day, etc.
- Whatever the rate is, we want to use it efficiently: send as much information as possible per bit
  - for this purpose, A&B set up a *code book,* like the one above for PM's breakfast items

Uniform code book

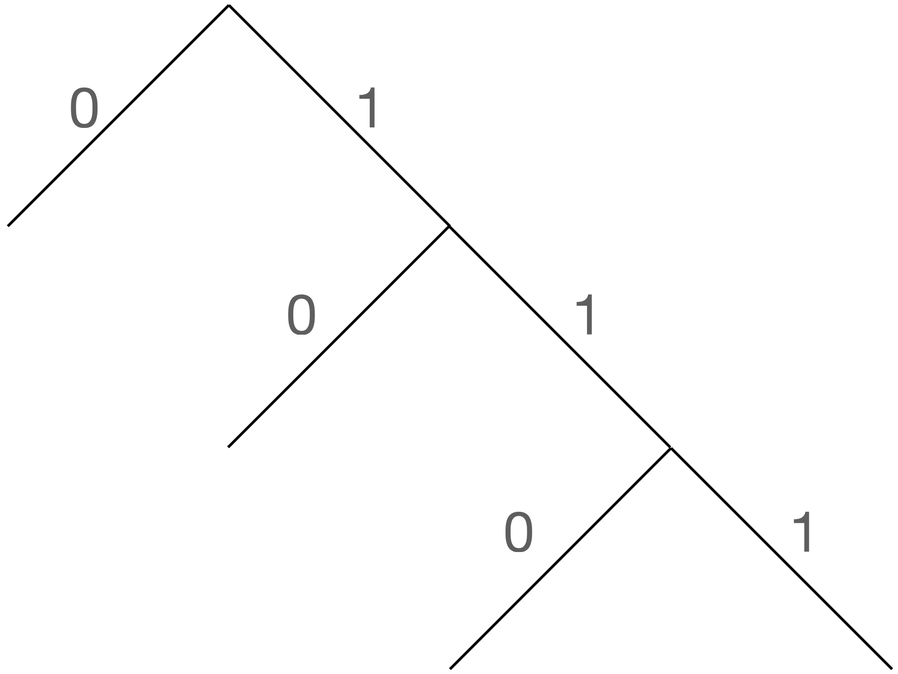
- In this book, all code words are the same length: if there are N possible breakfasts, each is $\lceil \log_2 N \rceil$ bits
- This is optimal in the worst case, which turns out to be when A&B expect all messages to occur equally often (uniform distribution)
- If some events are more common, we might want to send shorter messages for them

Nonuniform code book

meaning	code word	probability
pancakes	0	$\frac{1}{2}$
shredded wheat	10	$\frac{1}{4}$
bagels	110	$\frac{1}{8}$
caviar	111	$\frac{1}{8}$

- An example nonuniform code book (note *prefix property*)

Prefix property



Can guarantee no codeword is a prefix of another by making them paths to leaves in a binary tree

If prefix property failed (e.g., if 11 were a word), we'd be stuck after seeing 11: is this the end or do we have to wait for one more bit?

Nonuniform code book

meaning	code word	probability
pancakes	0	$\frac{1}{2}$
shredded wheat	10	$\frac{1}{4}$
bagels	110	$\frac{1}{8}$
caviar	111	$\frac{1}{8}$

- An example nonuniform code book (note *prefix property*)
- Expected message length = $p_1 b_1 + p_2 b_2 + \ldots + p_N b_N$: here $1\frac{3}{4}$
- But if we were wrong and outcomes really were uniform, expected length would be $(1 + 2 + 3 + 3)/4 = 2\frac{1}{4}$

Entropy

e.g., if $p_i = \frac{1}{4}$, need 2 bits

- Lower bound (Shannon): best code uses $\geq \log_2 \frac{1}{p_i}$ bits on average to send a message whose probability is p_i
  - in fact we can achieve exactly this value in the long run, but have to be slightly cleverer than a fixed code book
  - or hope that all p_i are integer powers of 2
- Expected message length: $-\sum_i p_i \log_2 p_i$, the entropy of distribution p
  - this is binary entropy; if we send digits instead of bits we get base-10 entropy (using $\log_{10}$)
  - also common to report *nats* (using $\ln$)
  - differ by constant factor, e.g., using digits saves factor $\log_2 10$ vs. bits in number of symbols sent

Wrong codebook

- Using a codebook based on q when true distribution is p costs $-\sum_i p_i \log_2 q_i$
  - messages are $-\log_2 q_i$ bits instead of $-\log_2 p_i$ bits
  - e.g., A&B set codebook before knowing true $p(\text{breakfast})$, can't meet up to fix it
- Extra cost is $\sum_i p_i \log_2 \frac{p_i}{q_i}$, the KL divergence or relative entropy between p and q

Information gain

X	Y	p_{XY}
pancakes	left	$\frac{1}{2}$
shredded wheat	left	$\frac{1}{6}$
pancakes	right	$\frac{1}{6}$
shredded wheat	right	$\frac{1}{6}$

$X =$ breakfast,
$Y =$ side of bed

- Suppose A&B both know the joint distribution above, and B wants to tell A what the PM had for breakfast
  - in expectation needs entropy(q) $\approx$ 0.918 bits where q $= \left[\frac{2}{3}, \frac{1}{3}\right]$

Information gain

X	Y	p_{XY}
pancakes	left	$\frac{1}{2}$
shredded wheat	left	$\frac{1}{6}$
pancakes	right	$\frac{1}{6}$
shredded wheat	right	$\frac{1}{6}$

$\left.\vphantom{\frac{1}{2}}\right\}\ \mathsf{p} = \left[\frac{3}{4}, \frac{1}{4}\right]$

- What if B knows that PM got out of bed on left?
  - true distribution is now p, but A doesn't know this; so we pay an extra $D_{KL}(p||q) \approx 0.024$ bits from having to use q's codebook
  - alternate way to say it: if Alice found out that PM got out of bed on left, she would gain 0.024 bits of information

Information gain

X	Y	p_{XY}
pancakes	left	$\frac{1}{2}$
shredded wheat	left	$\frac{1}{6}$
pancakes	right	$\frac{1}{6}$
shredded wheat	right	$\frac{1}{6}$

$\left.\vphantom{\frac{1}{2}}\right\}\ p' = \left[\frac{1}{2}, \frac{1}{2}\right]$

- What if B knows that PM got out of bed on right?
  - true distribution is now p', but A doesn't know this; so we pay an extra $D_{KL}(p' \| q) \approx 0.085$ bits from having to use q's codebook
  - alternate way to say it: if Alice found out that PM got out of bed on right, she would gain 0.085 bits of information

Mutual information

X	Y	p_{XY}
pancakes	left	$\frac{1}{2}$
shredded wheat	left	$\frac{1}{6}$
pancakes	right	$\frac{1}{6}$
shredded wheat	right	$\frac{1}{6}$

- On average, if A finds out left or right each day, she gains 0.024 bits on $\frac{2}{3}$ of days, and 0.085 bits on $\frac{1}{3}$ of days
- These 0.044 bits are the expected information gain from finding out side-of-bed — also called the *mutual information* between side-of-bed and breakfast
- Hard to see from this expression, but MI is symmetric

Mutual information as a splitting criterion

- When we run a decision tree, each example sorts to some leaf l and has true label y
- A good decision tree has high mutual information between l and y (knowing l is helpful in predicting y)
- Splitting criterion: which split increases this mutual information the most, as measured on the training set?

Decision Tree — Training

Suppose you're trying to predict whether a house is located in ***San Francisco*** or ***New York*** based on the following features:

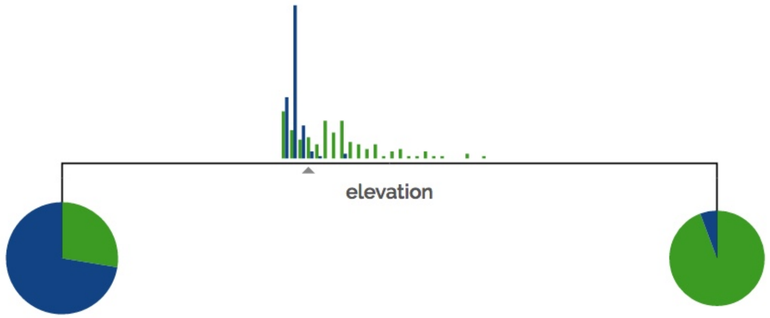
- price
- sq ft
- price per sq ft
- year built
- elevation
- # bathrooms
- # bedrooms

slide credit: Ameet Talwalkar

Decision Tree — Training

Suppose you're trying to predict whether a house is located in ***San Francisco*** or ***New York*** based on the following features:

- price
- sq ft
- price per sq ft
- year built
- elevation
- # bathrooms
- # bedrooms

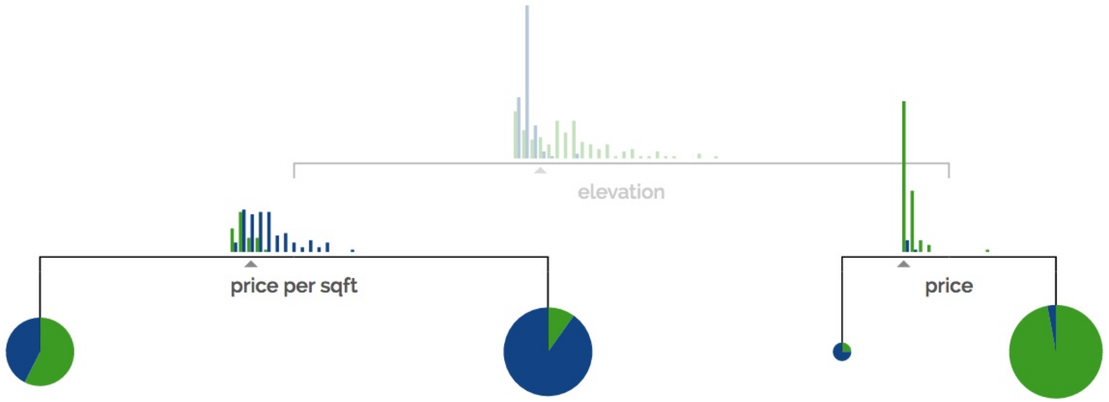


slide credit: Ameet Talwalkar

Decision Tree — Training

Suppose you're trying to predict whether a house is located in ***San Francisco*** or ***New York*** based on the following features:

- price
- sq ft
- price per sq ft
- year built
- elevation
- # bathrooms
- # bedrooms

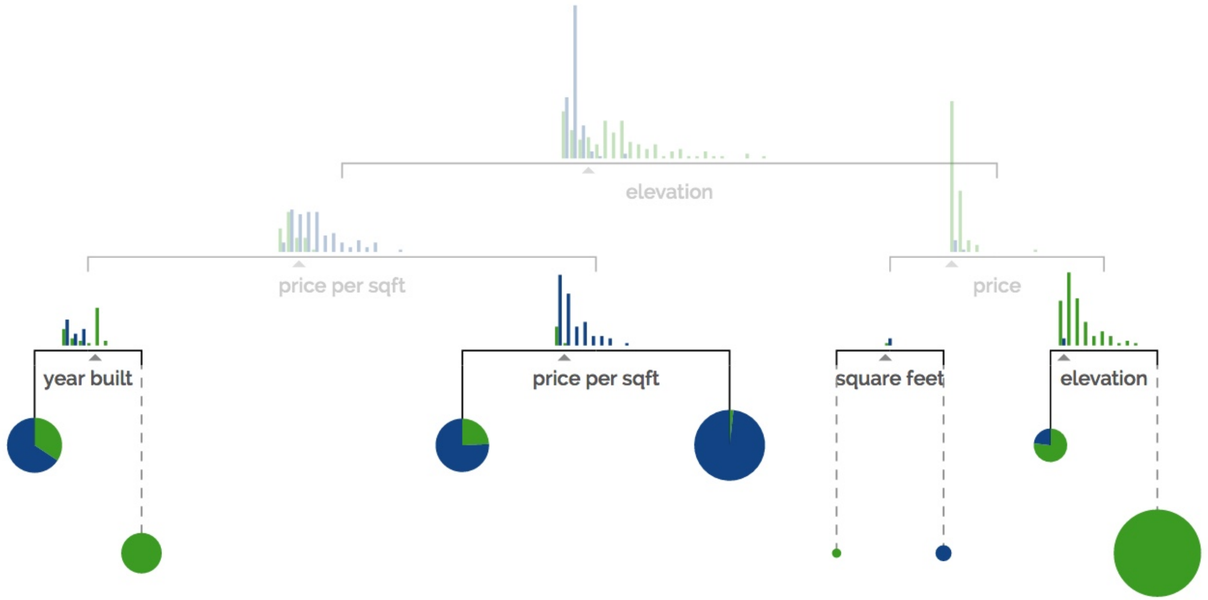


slide credit: Ameet Talwalkar

Decision Tree — Training

Suppose you're trying to predict whether a house is located in ***San Francisco*** or ***New York*** based on the following features:

- price
- sq ft
- price per sq ft
- year built
- elevation
- # bathrooms
- # bedrooms

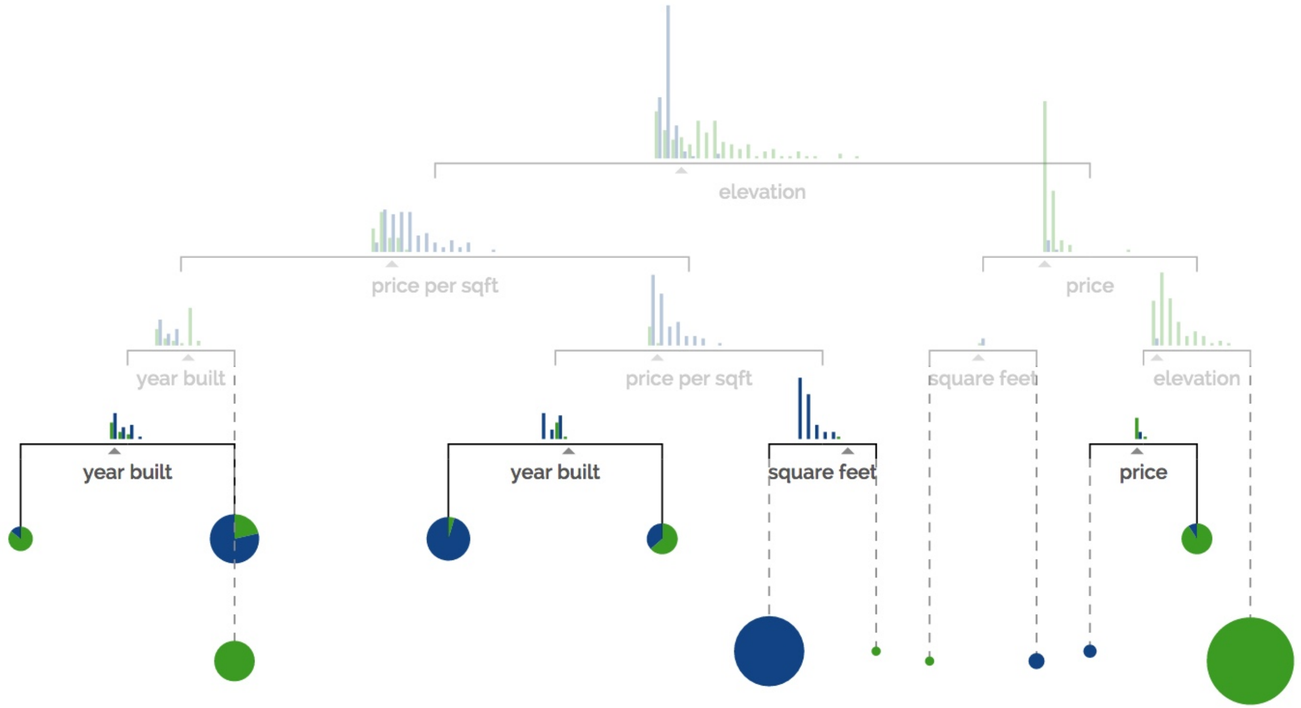


slide credit: Ameet Talwalkar

Decision Tree — Training

Suppose you're trying to predict whether a house is located in ***San Francisco*** or ***New York*** based on the following features:

- price
- sq ft
- price per sq ft
- year built
- elevation
- # bathrooms
- # bedrooms

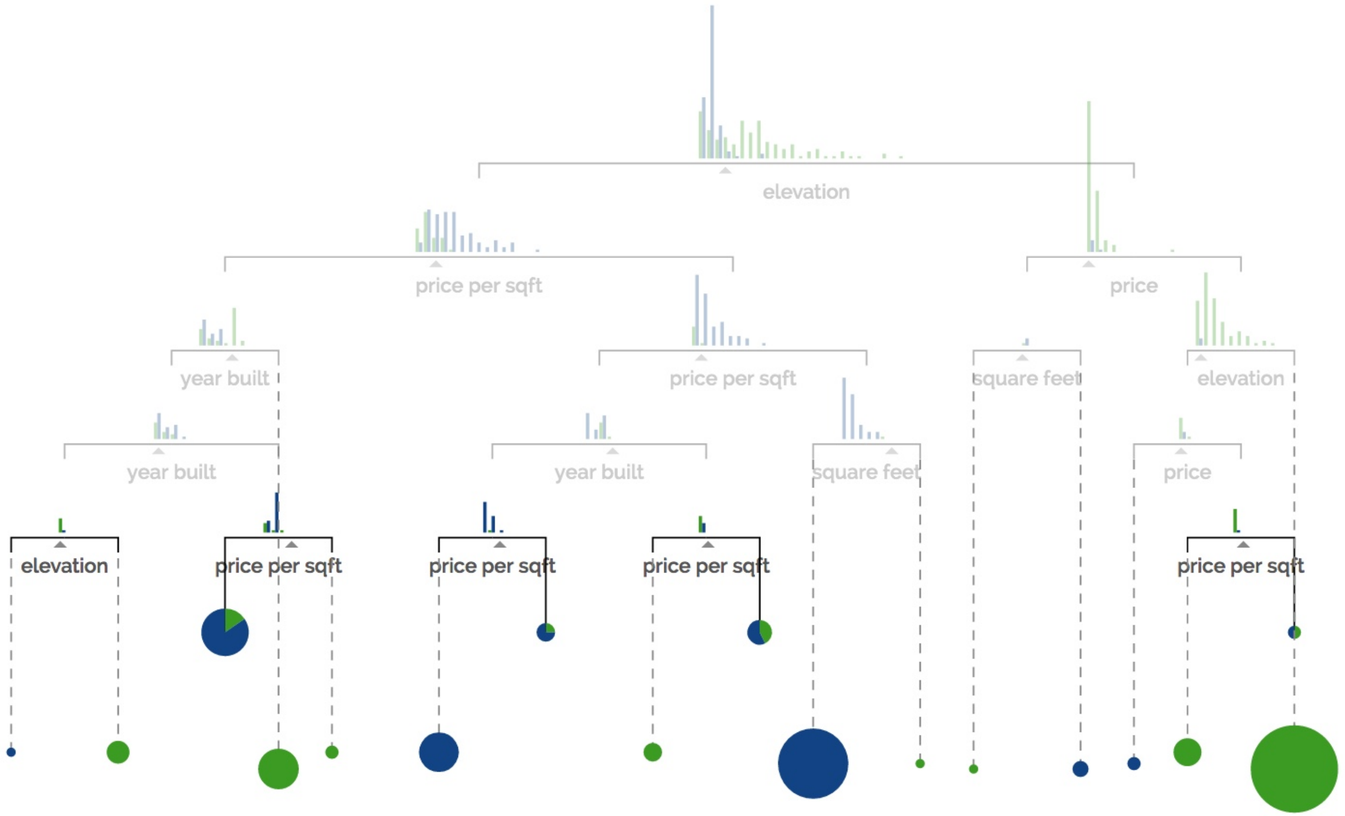


slide credit: Ameet Talwalkar

Decision Tree — Training

Suppose you're trying to predict whether a house is located in ***San Francisco*** or ***New York*** based on the following features:

- price
- sq ft
- price per sq ft
- year built
- elevation
- # bathrooms
- # bedrooms

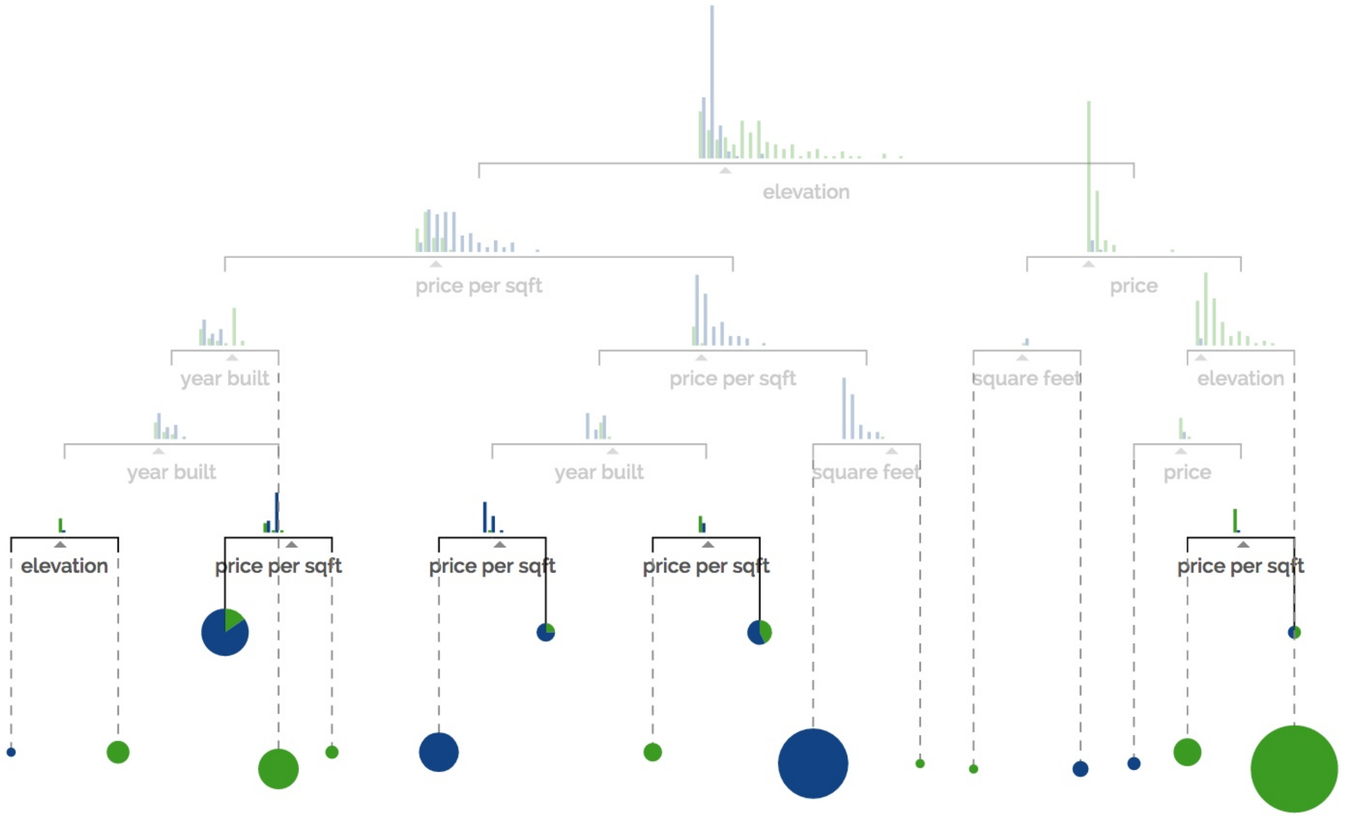


slide credit: Ameet Talwalkar

Decision Tree — Prediction

Suppose you're trying to predict whether a house is located in ***San Francisco*** or ***New York*** based on the following features:

- price
- sq ft
- price per sq ft
- year built
- elevation
- # bathrooms
- # bedrooms

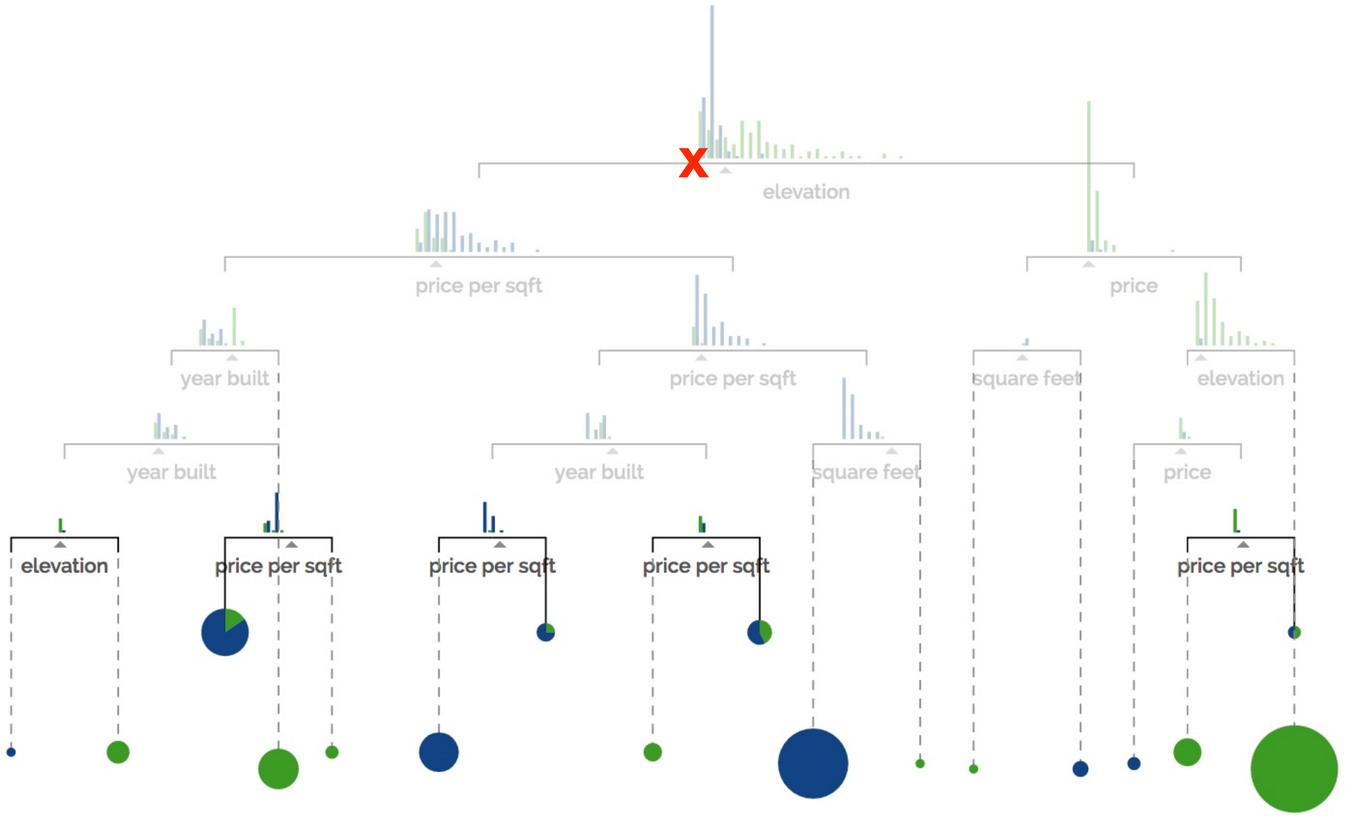


slide credit: Ameet Talwalkar

Decision Tree — Prediction

Suppose you're trying to predict whether a house is located in ***San Francisco*** or ***New York*** based on the following features:

- price
- sq ft
- price per sq ft
- year built
- elevation
- # bathrooms
- # bedrooms

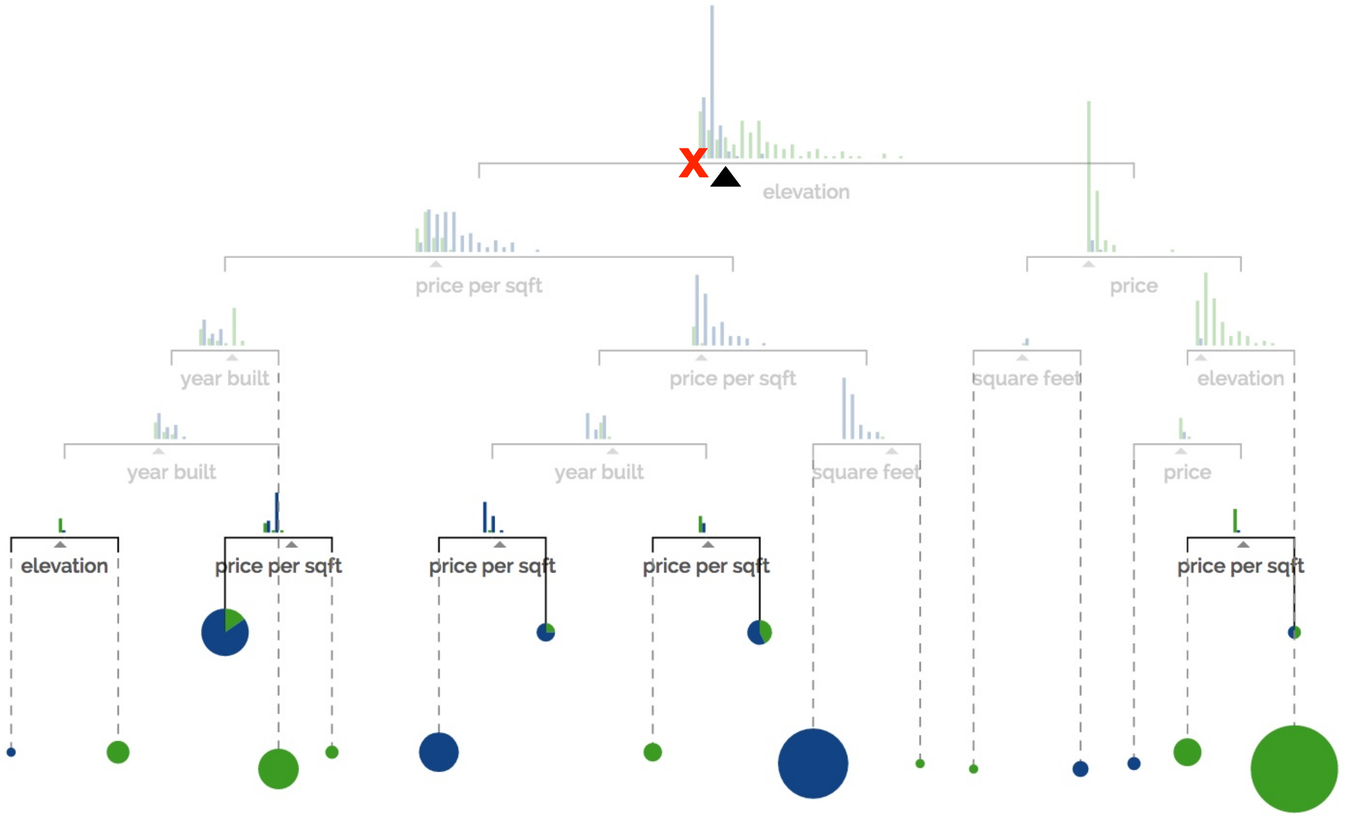


slide credit: Ameet Talwalkar

Decision Tree — Prediction

Suppose you're trying to predict whether a house is located in ***San Francisco*** or ***New York*** based on the following features:

- price
- sq ft
- price per sq ft
- year built
- elevation
- # bathrooms
- # bedrooms

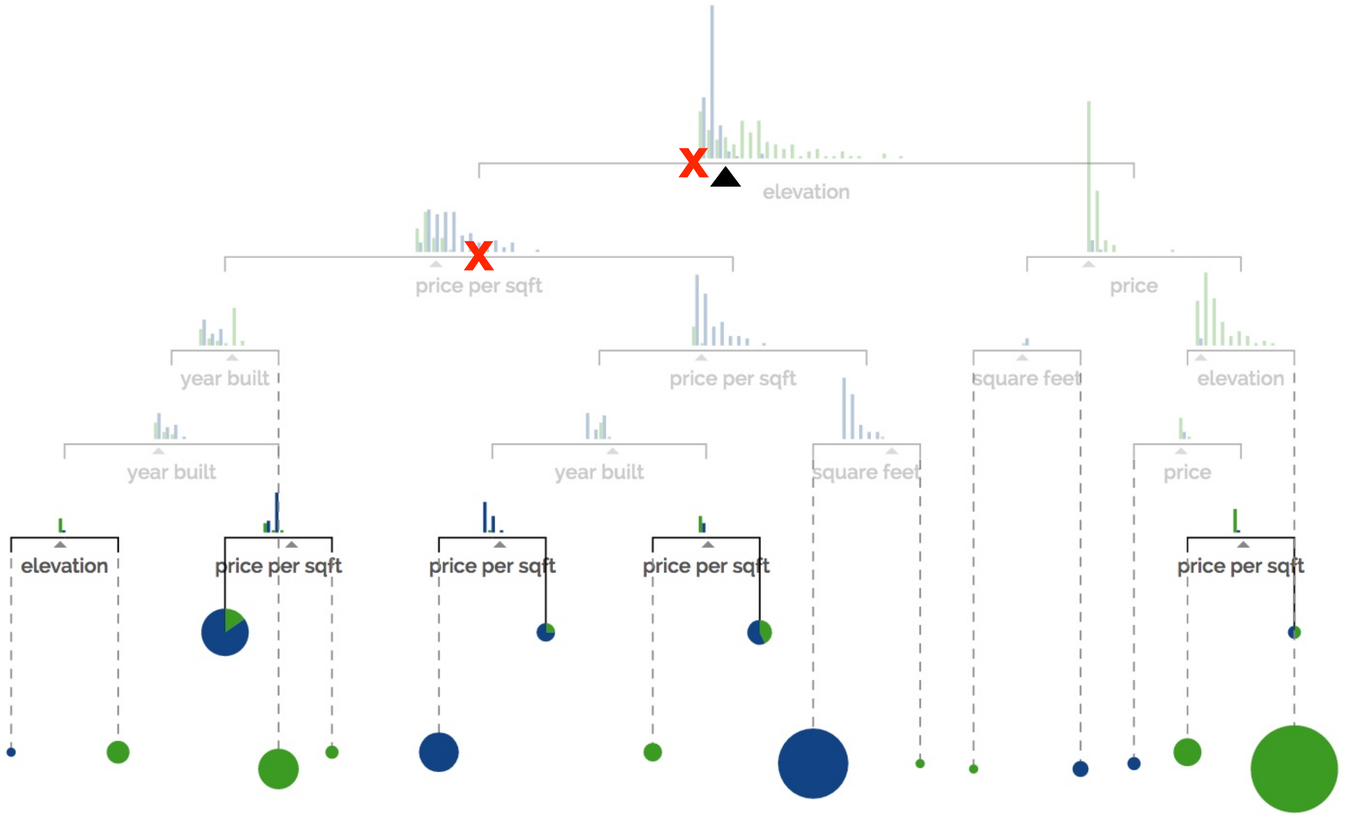


slide credit: Ameet Talwalkar

Decision Tree — Prediction

Suppose you're trying to predict whether a house is located in *San Francisco* or *New York* based on the following features:

- price
- sq ft
- price per sq ft
- year built
- elevation
- # bathrooms
- # bedrooms

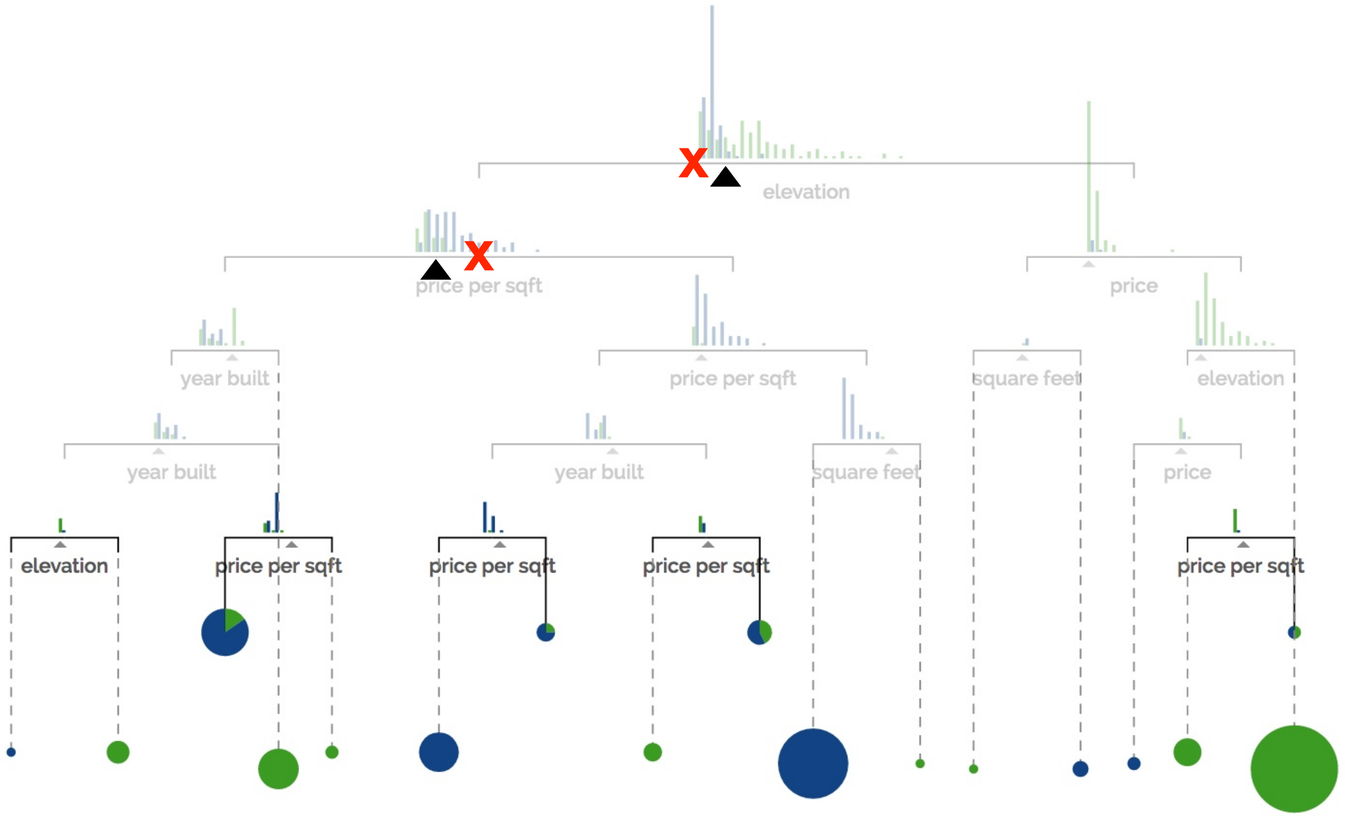


slide credit: Ameet Talwalkar

Decision Tree — Prediction

Suppose you're trying to predict whether a house is located in ***San Francisco*** or ***New York*** based on the following features:

- price
- sq ft
- price per sq ft
- year built
- elevation
- # bathrooms
- # bedrooms

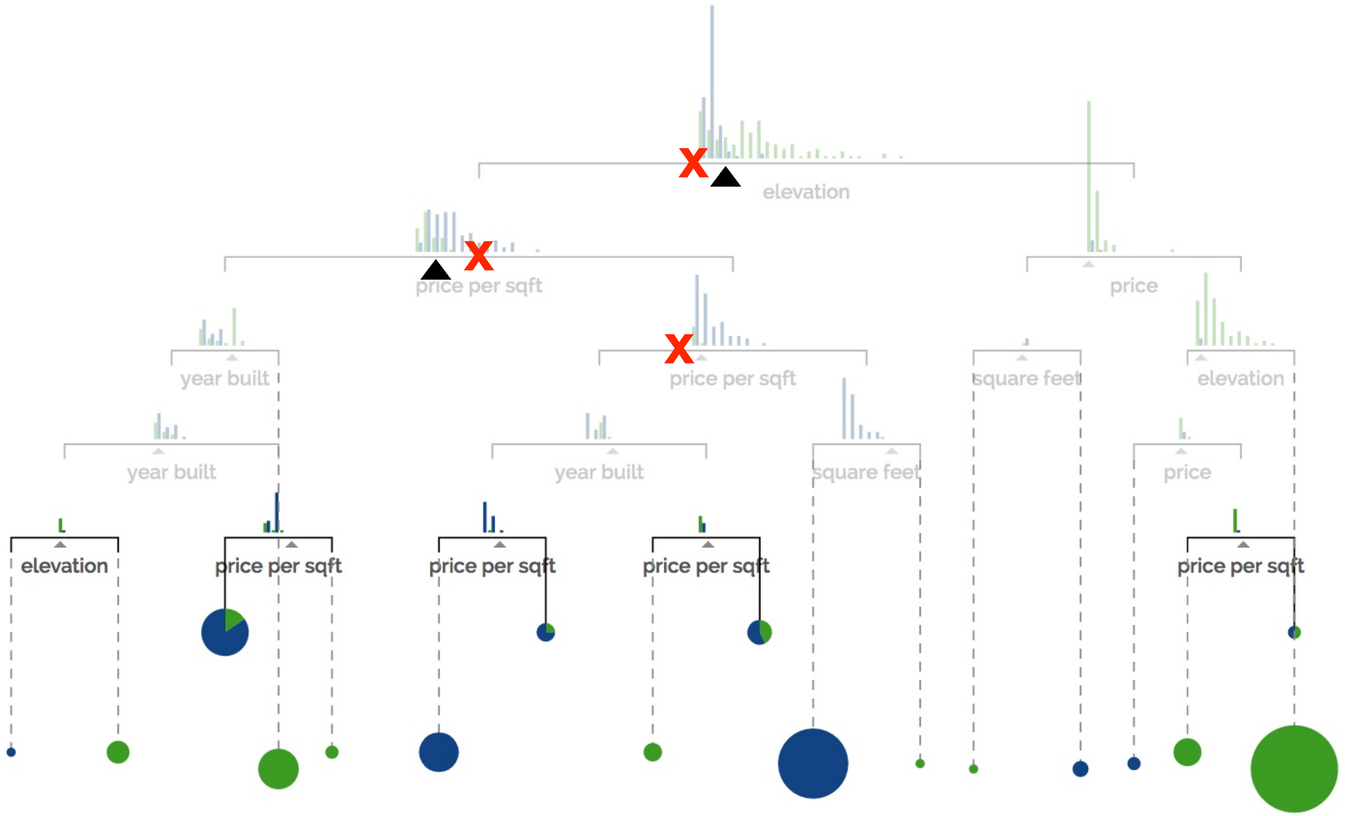


slide credit: Ameet Talwalkar

Decision Tree — Prediction

Suppose you're trying to predict whether a house is located in ***San Francisco*** or ***New York*** based on the following features:

- price
- sq ft
- price per sq ft
- year built
- elevation
- # bathrooms
- # bedrooms

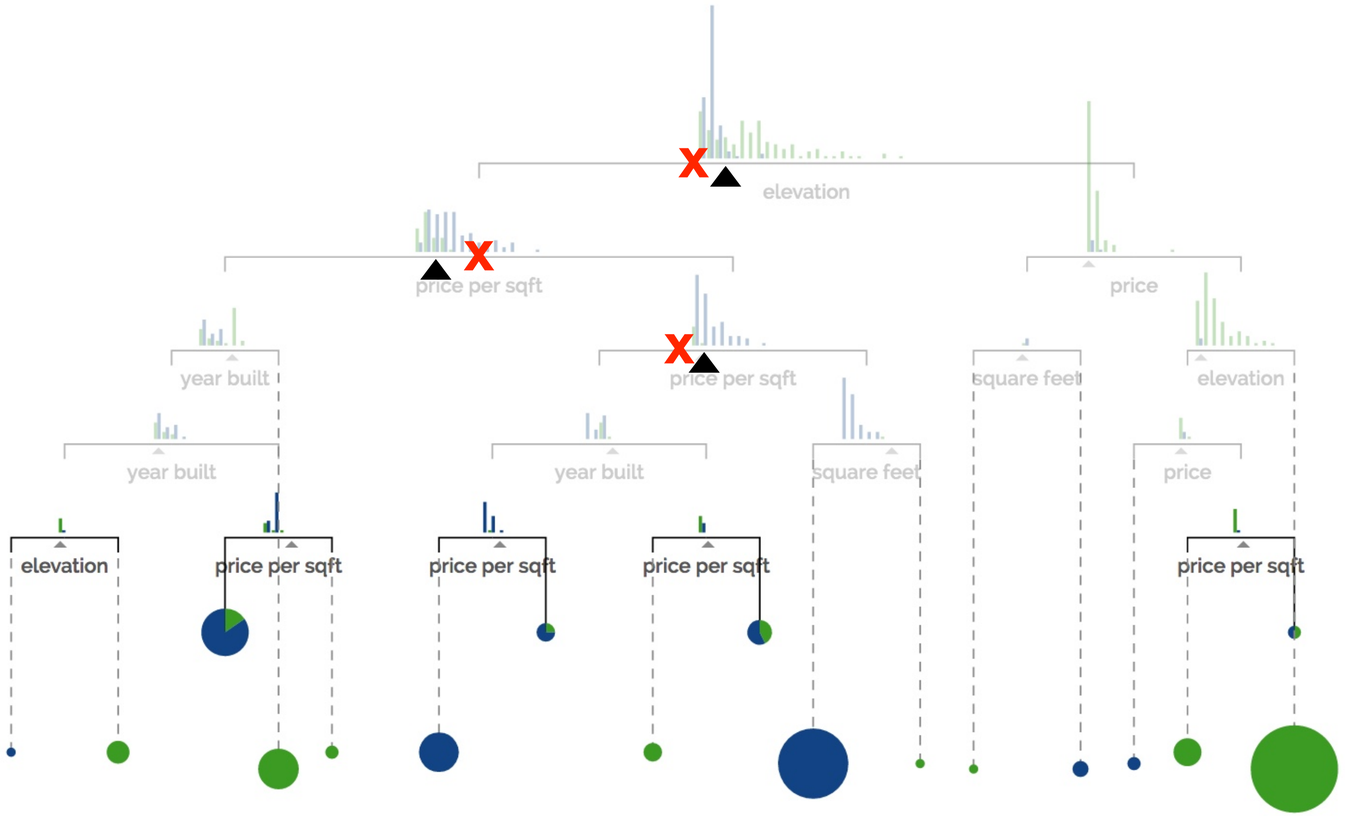


slide credit: Ameet Talwalkar

Decision Tree — Prediction

Suppose you're trying to predict whether a house is located in ***San Francisco*** or ***New York*** based on the following features:

- price
- sq ft
- price per sq ft
- year built
- elevation
- # bathrooms
- # bedrooms

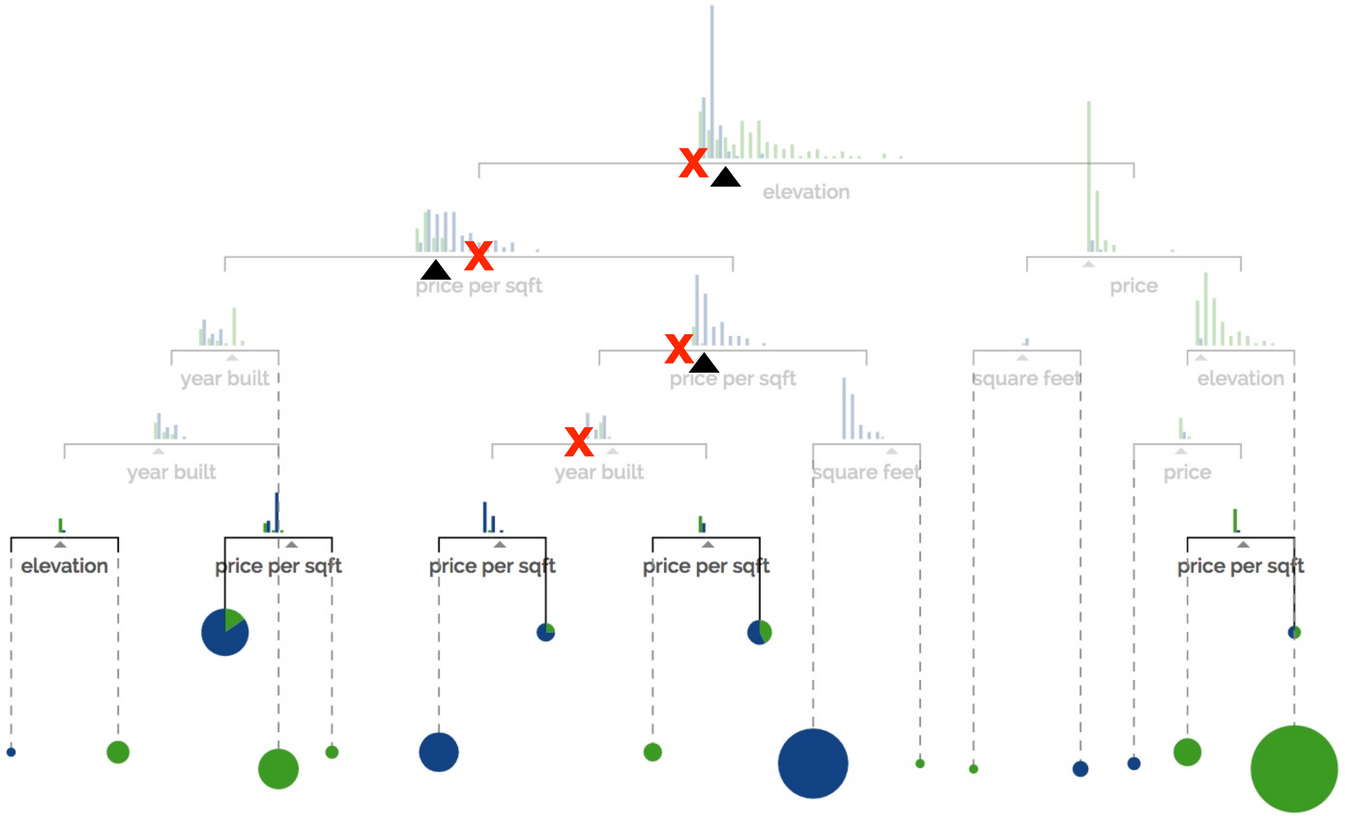


slide credit: Ameet Talwalkar

Decision Tree — Prediction

Suppose you're trying to predict whether a house is located in ***San Francisco*** or ***New York*** based on the following features:

- price
- sq ft
- price per sq ft
- year built
- elevation
- # bathrooms
- # bedrooms

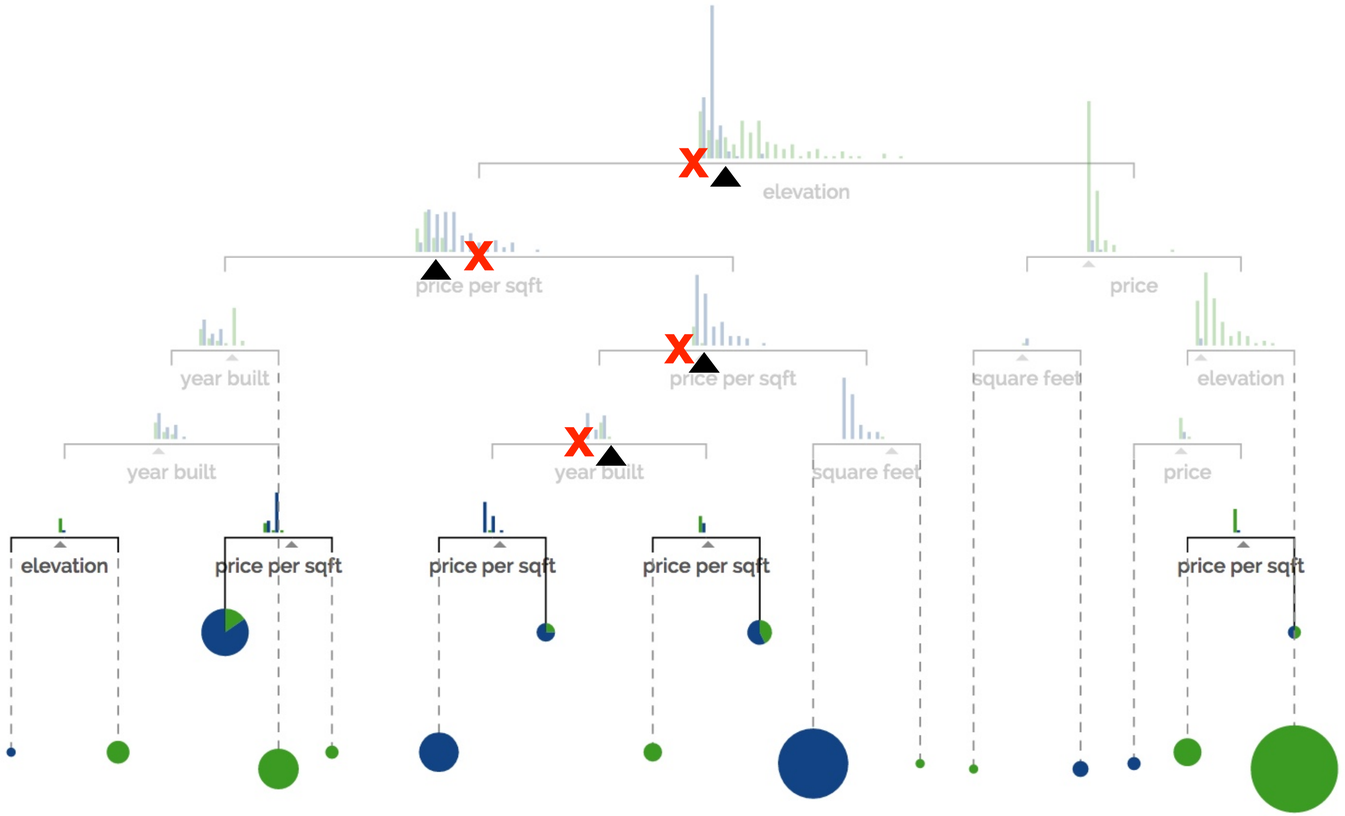


slide credit: Ameet Talwalkar

Decision Tree — Prediction

Suppose you're trying to predict whether a house is located in ***San Francisco*** or ***New York*** based on the following features:

- price
- sq ft
- price per sq ft
- year built
- elevation
- # bathrooms
- # bedrooms

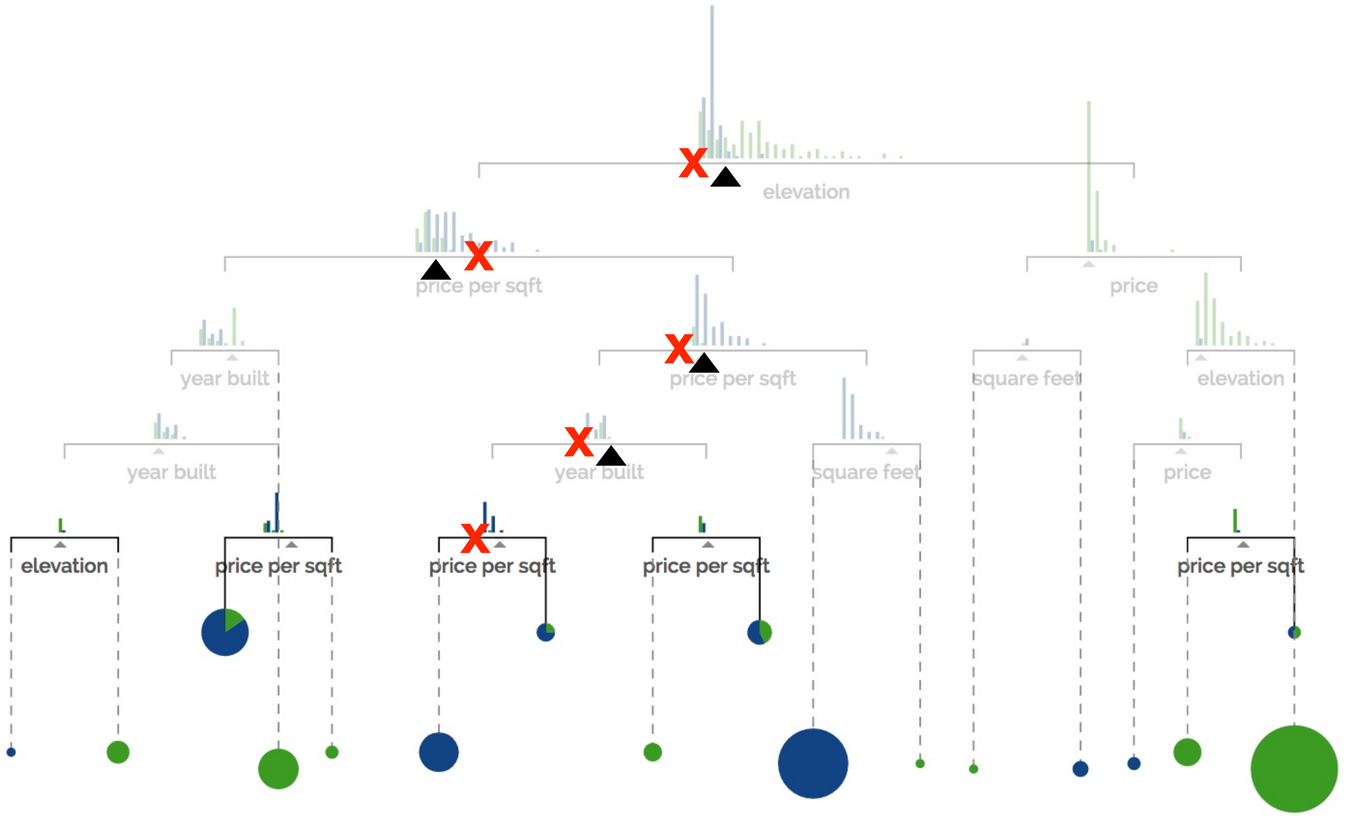


slide credit: Ameet Talwalkar

Decision Tree — Prediction

Suppose you're trying to predict whether a house is located in ***San Francisco*** or ***New York*** based on the following features:

- price
- sq ft
- price per sq ft
- year built
- elevation
- # bathrooms
- # bedrooms

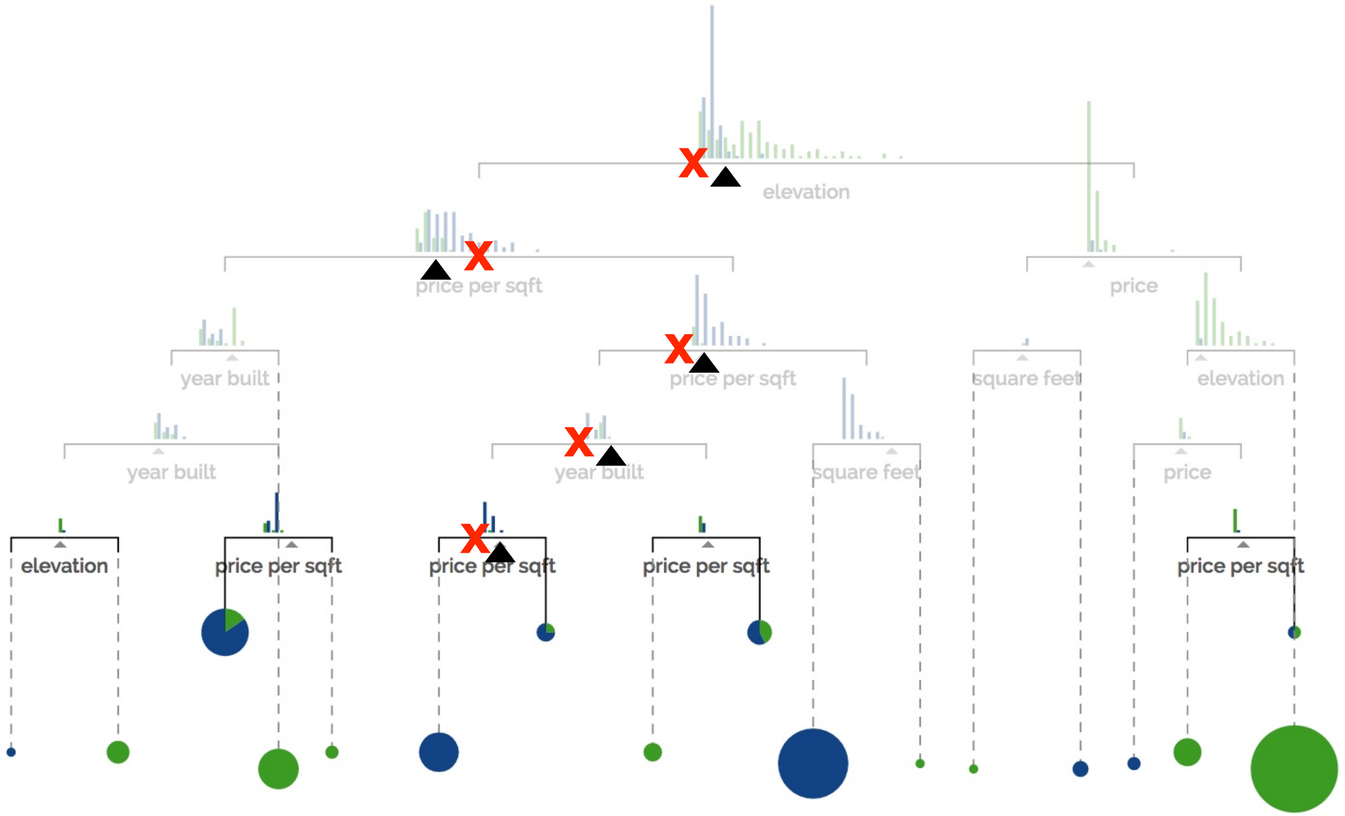


slide credit: Ameet Talwalkar

Decision Tree — Prediction

Suppose you're trying to predict whether a house is located in ***San Francisco*** or ***New York*** based on the following features:

- price
- sq ft
- price per sq ft
- year built
- elevation
- # bathrooms
- # bedrooms

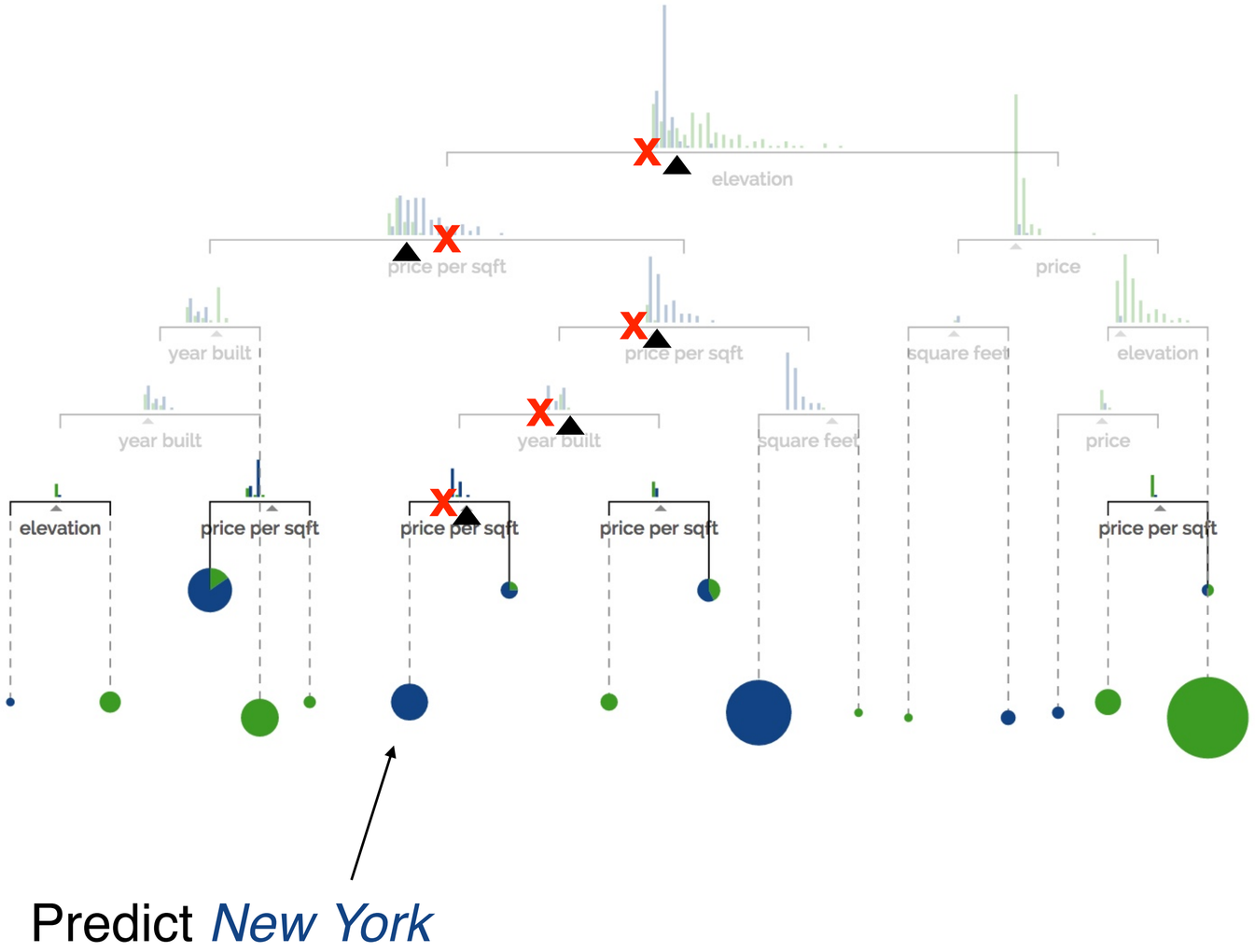


slide credit: Ameet Talwalkar

Decision Tree — Prediction

Suppose you're trying to predict whether a house is located in *San Francisco* or *New York* based on the following features:

- price
- sq ft
- price per sq ft
- year built
- elevation
- # bathrooms
- # bedrooms

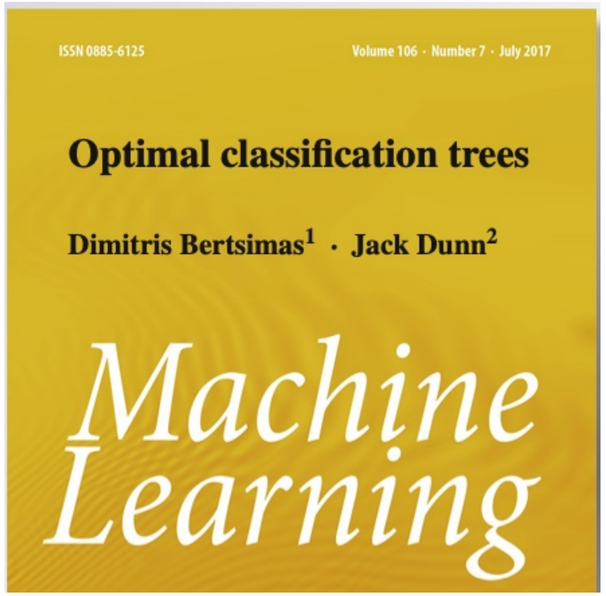


Predict *New York*

slide credit: Ameet Talwalkar

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…



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

Optimal **C**lassification **T**ree

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

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

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

Table from Bluntine & Niblett (1992)

Empirical comparison: Splitting Criteria

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

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

Info. Gain is another name for *mutual information*

Table from Bluntine & Niblett (1992)

Experiments: Splitting Criteria

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

	Splitting Rule		
Data Set	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)	[illegible]	[illegible]
mush	0.00 (0.50)	5.86	[illegible]
votes	0.11 (0.55)	7.30	[illegible]
votes1	0.26 (0.47)	2.34	[illegible]
iris	−0.10 (0.67)	0.50	[illegible]
glass	1.01 (0.50)	0.96	[illegible]
xd6	0.04 (0.11)	−0.07	[illegible]
pole	0.03 (0.11)	−0.43	[illegible]

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**

Table from Bluntine & Niblett (1992)

Underfitting and Overfitting in Decision Tree Learning

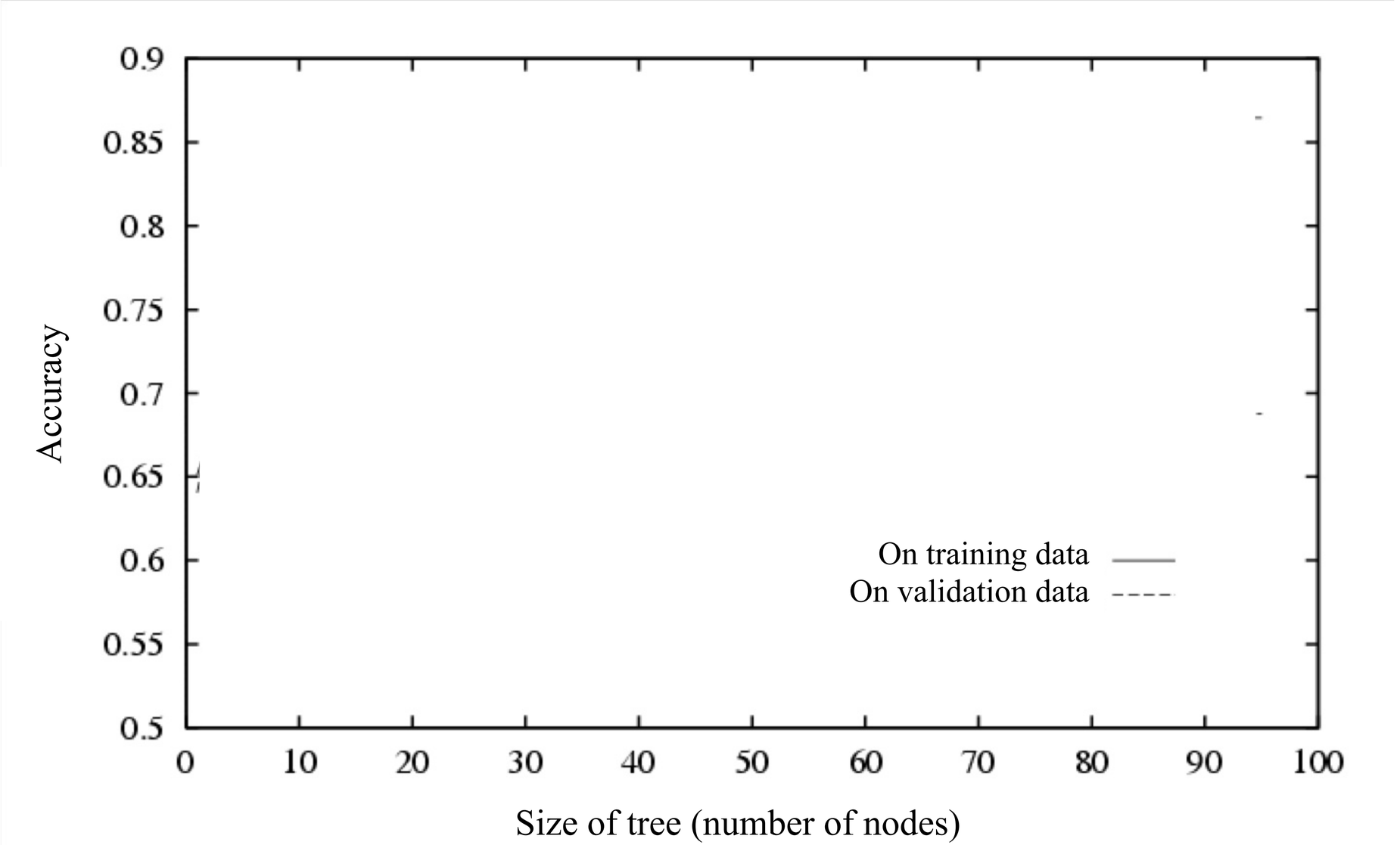


Figure from Tom Mitchell

Underfitting and Overfitting in Decision Tree Learning

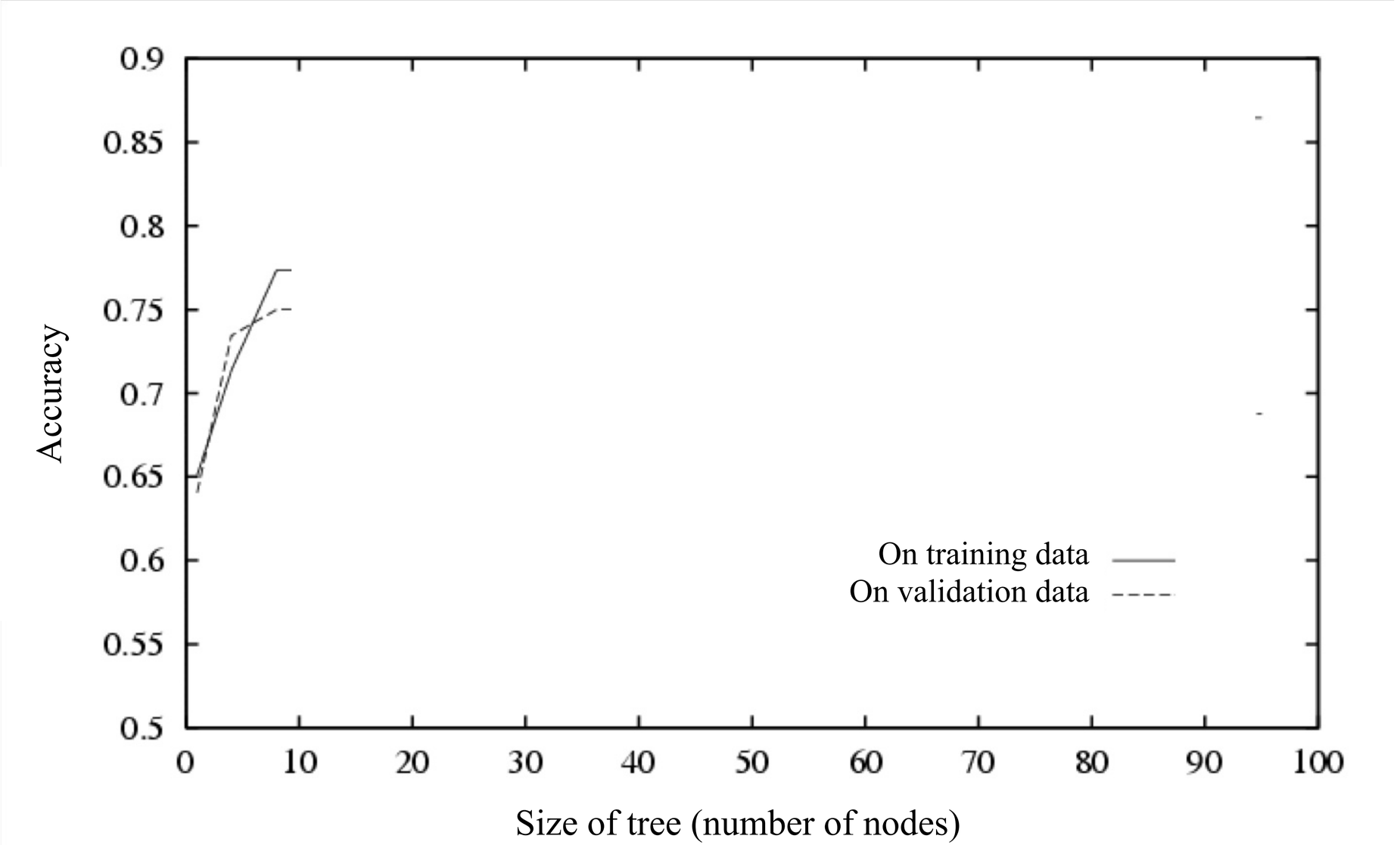


Figure from Tom Mitchell

Underfitting and Overfitting in Decision Tree Learning

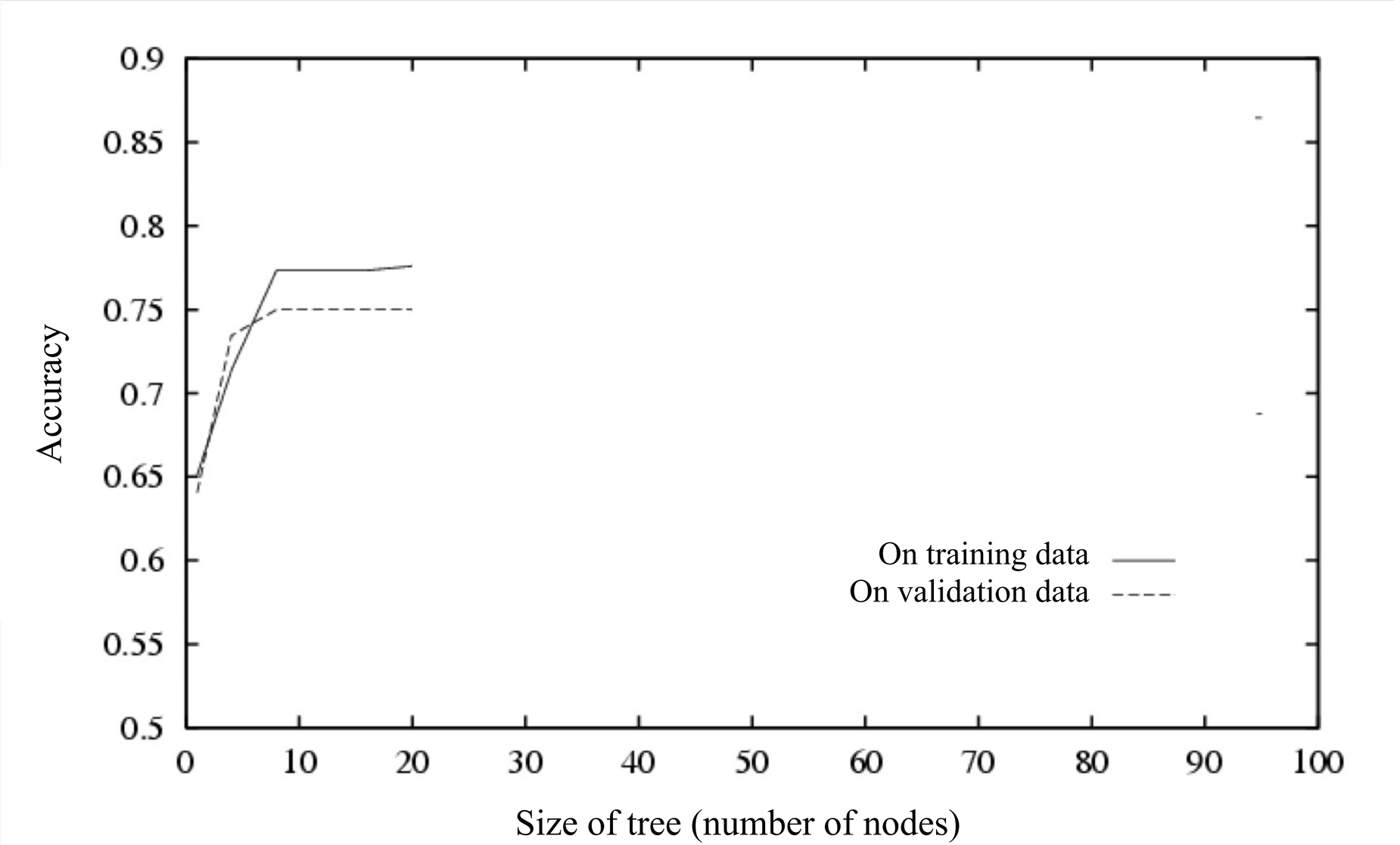


Figure from Tom Mitchell

Underfitting and Overfitting in Decision Tree Learning

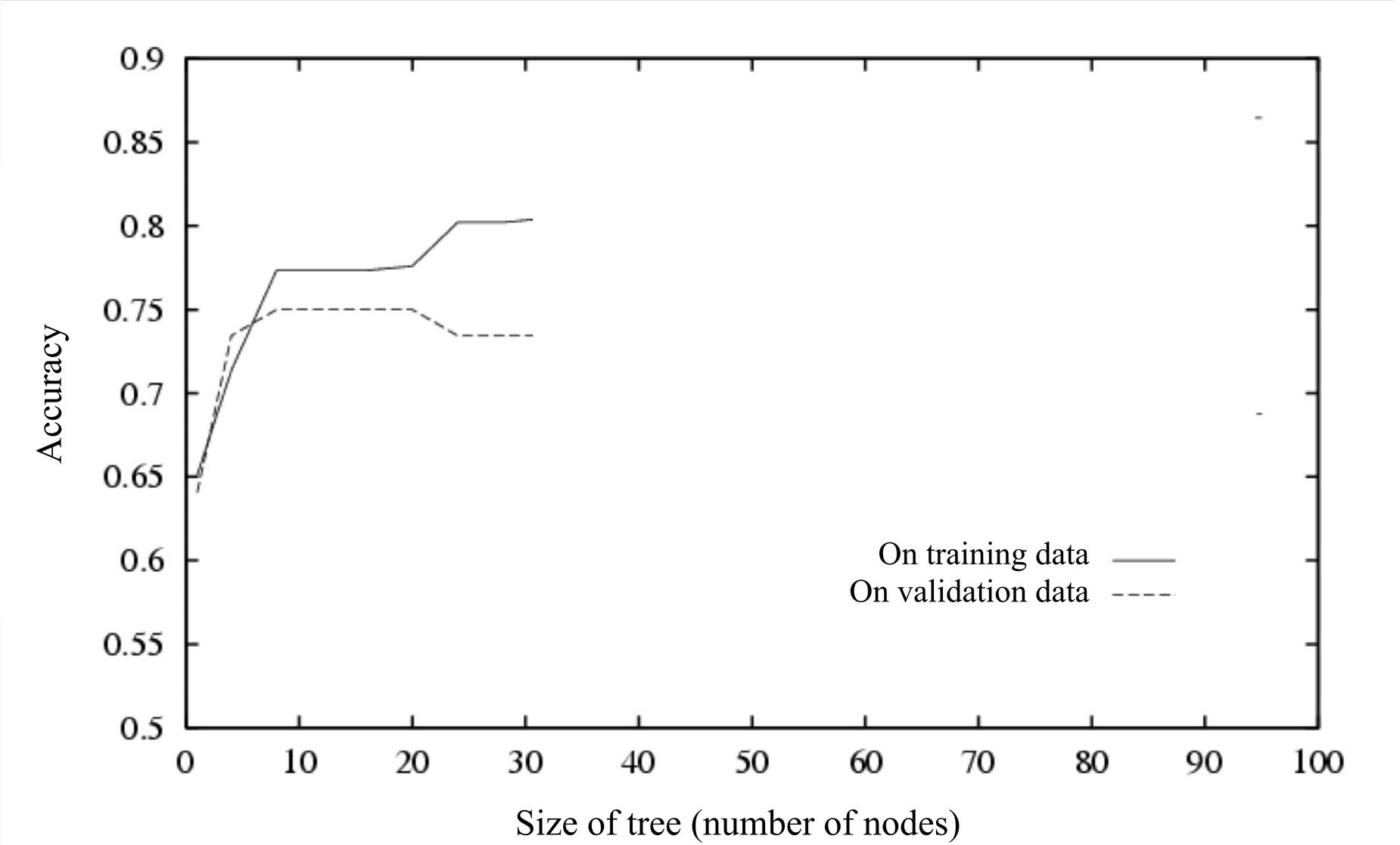


Figure from Tom Mitchell

Underfitting and Overfitting in Decision Tree Learning

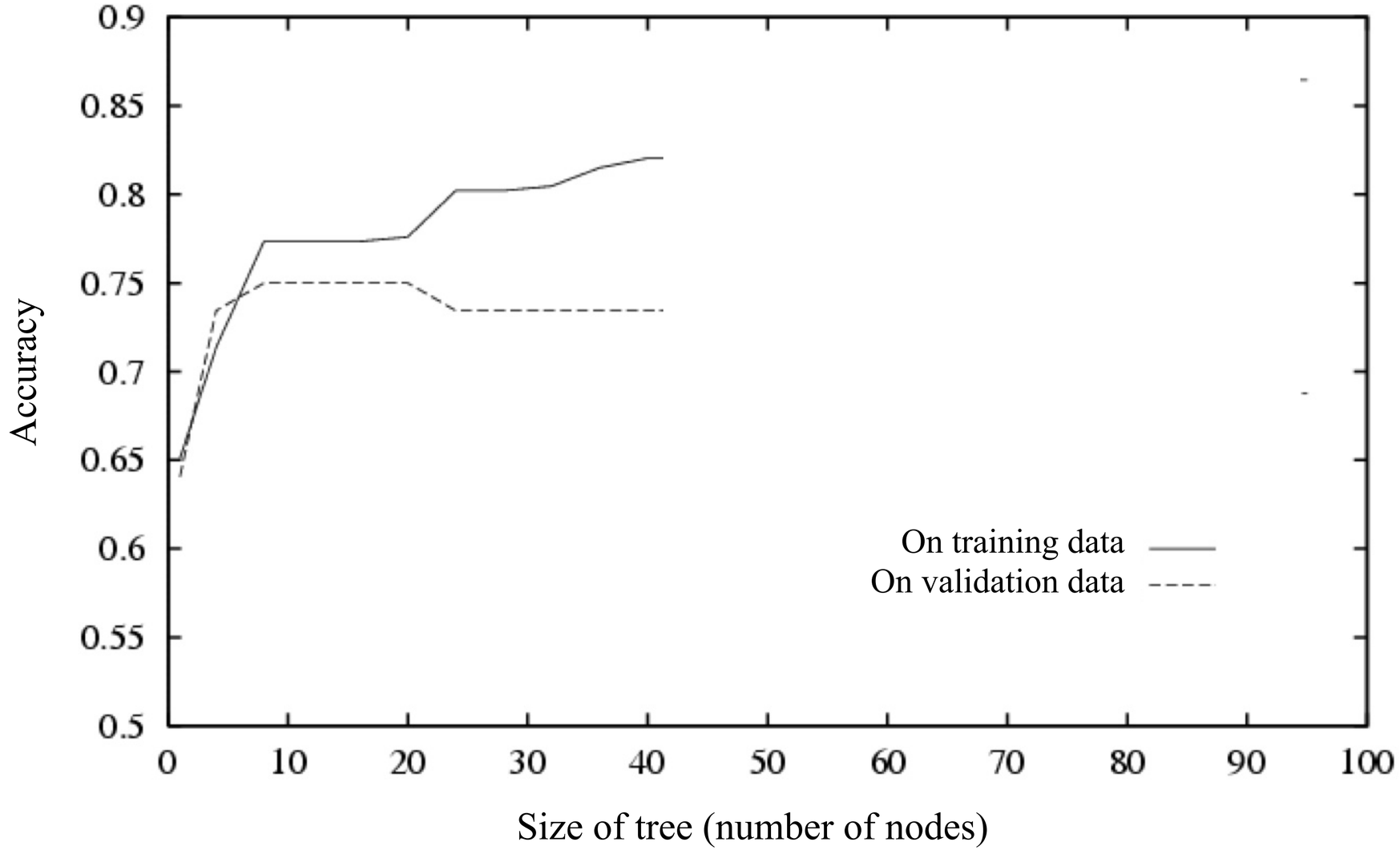


Figure from Tom Mitchell

Underfitting and Overfitting in Decision Tree Learning

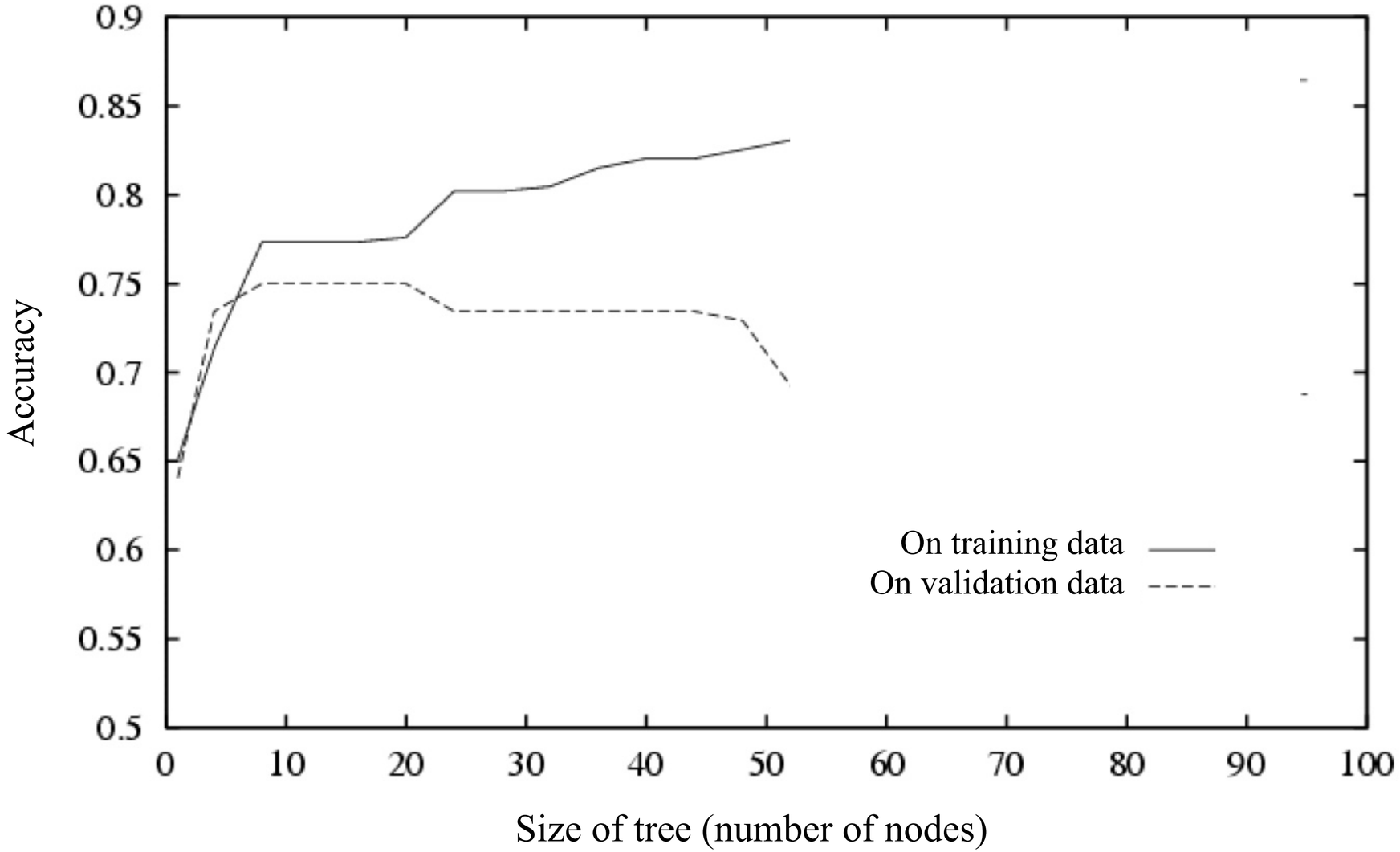


Figure from Tom Mitchell

Underfitting and Overfitting in Decision Tree Learning

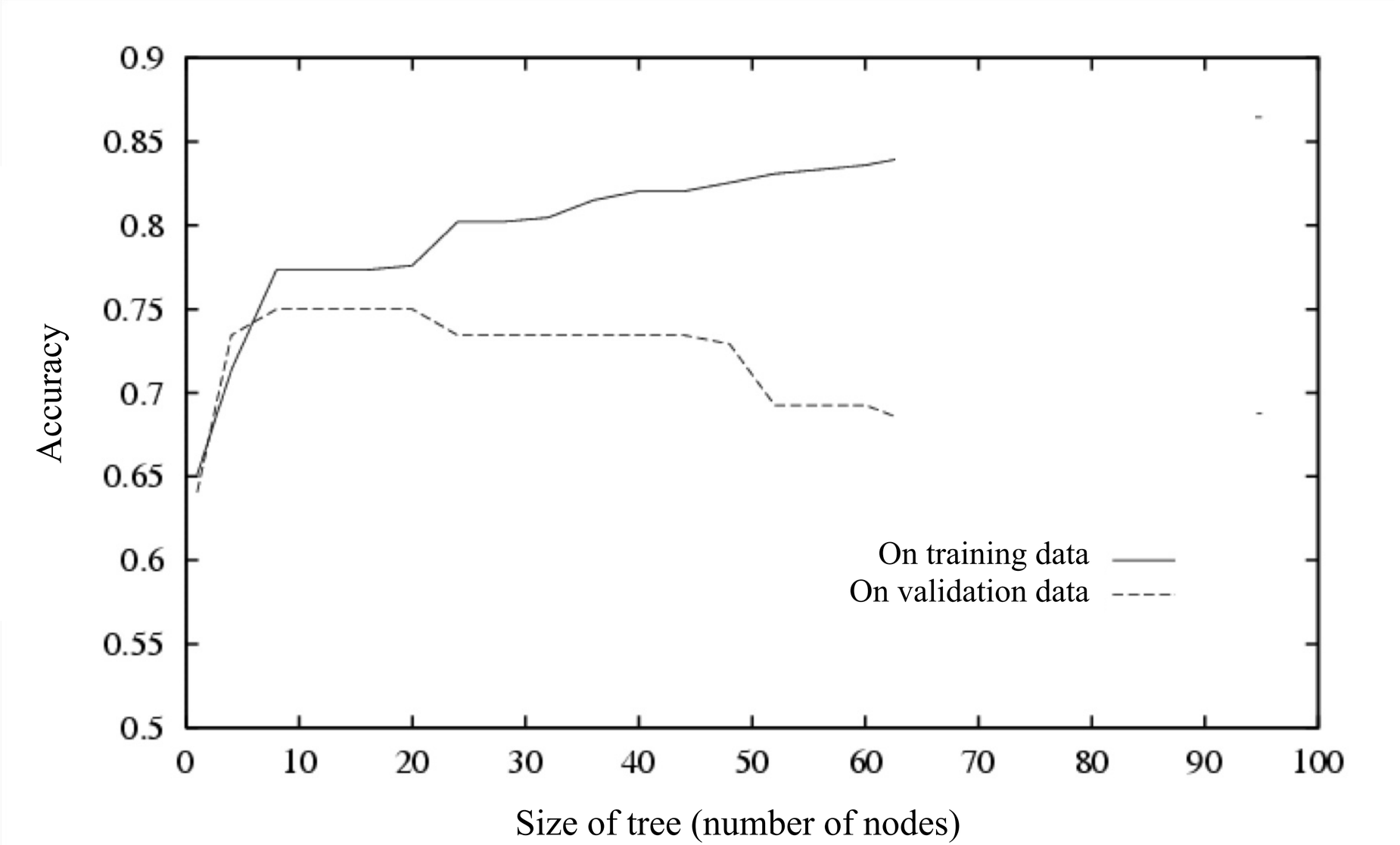


Figure from Tom Mitchell

Underfitting and Overfitting in Decision Tree Learning

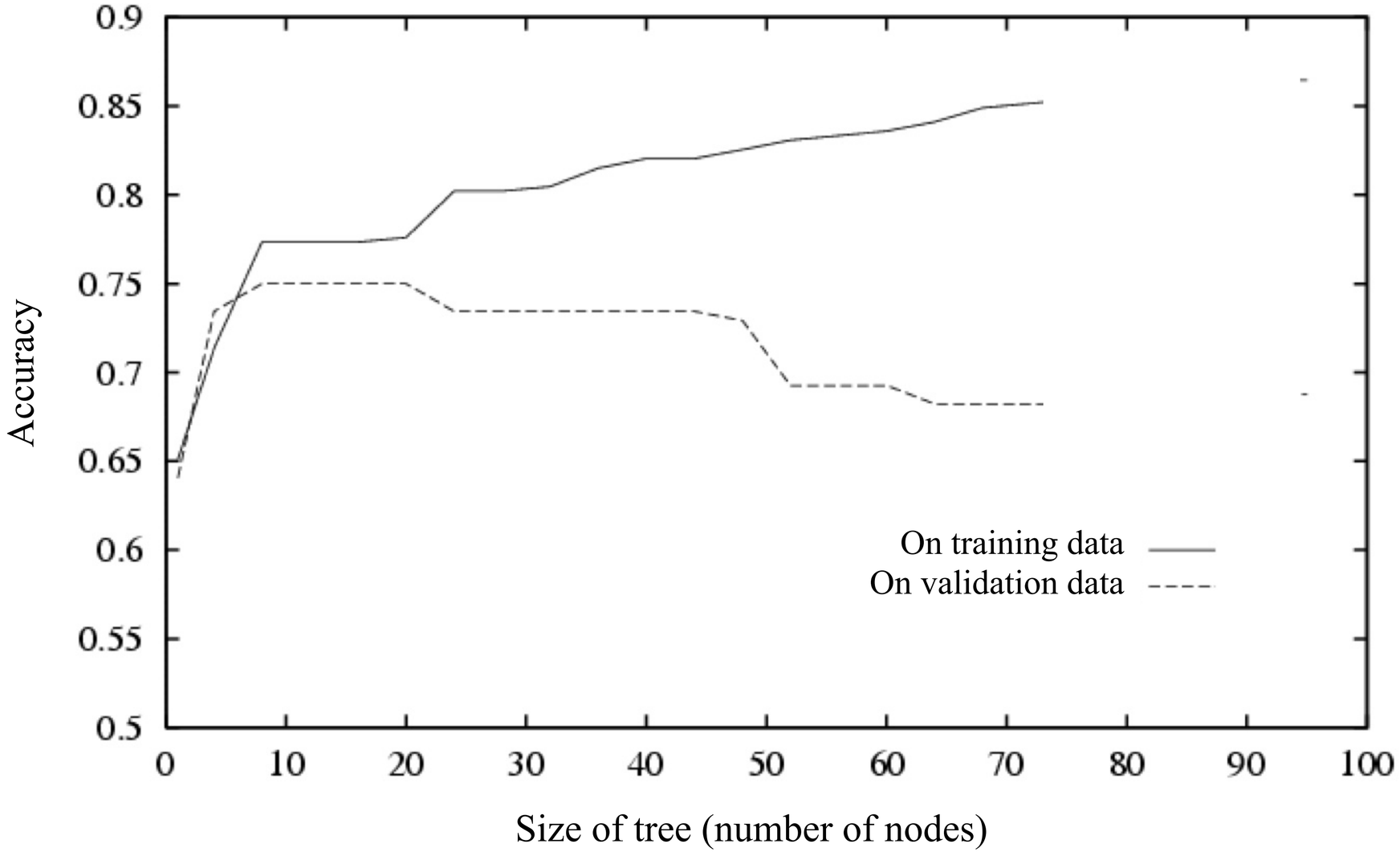


Figure from Tom Mitchell

Underfitting and Overfitting in Decision Tree Learning

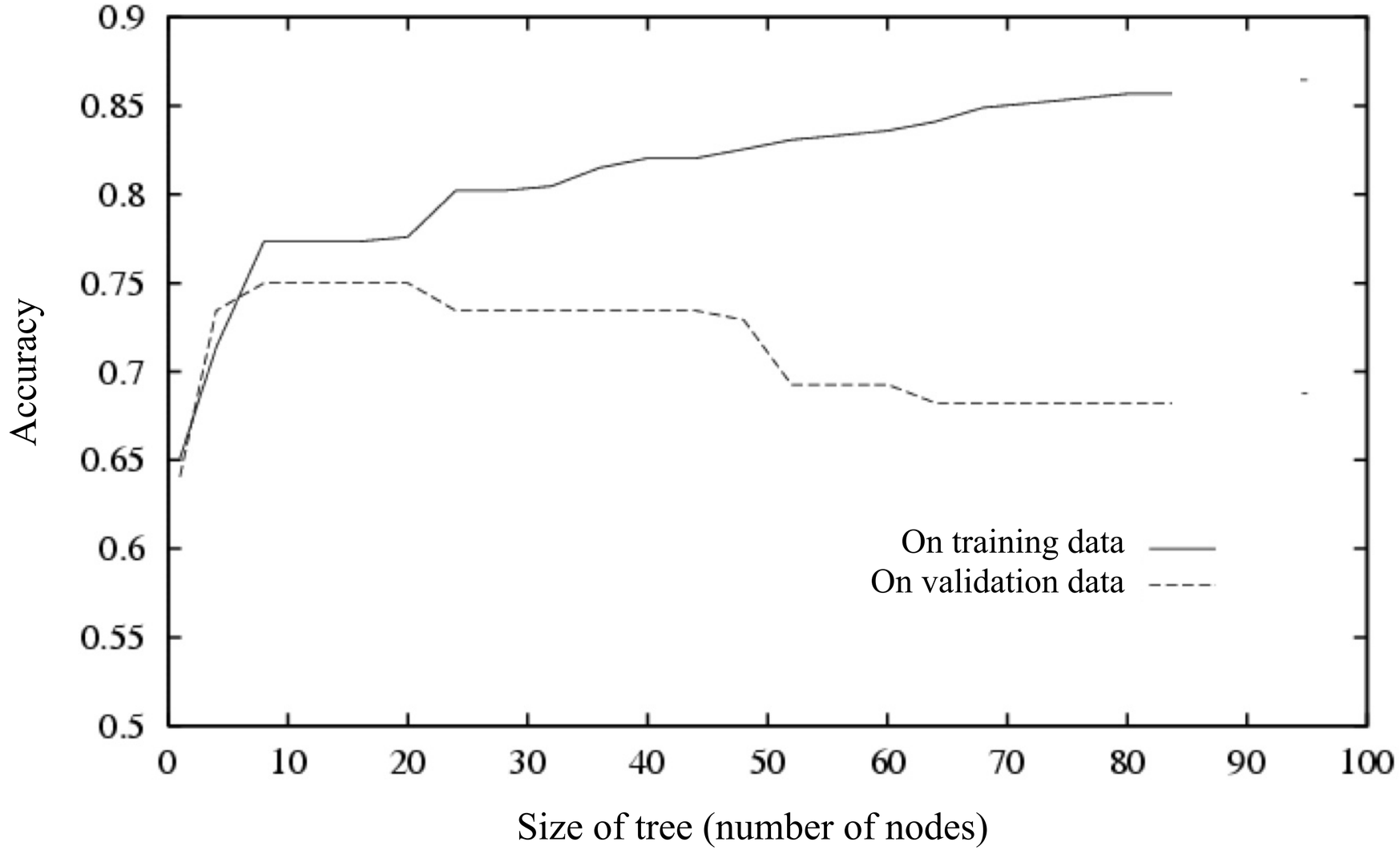


Figure from Tom Mitchell

Underfitting and Overfitting in Decision Tree Learning

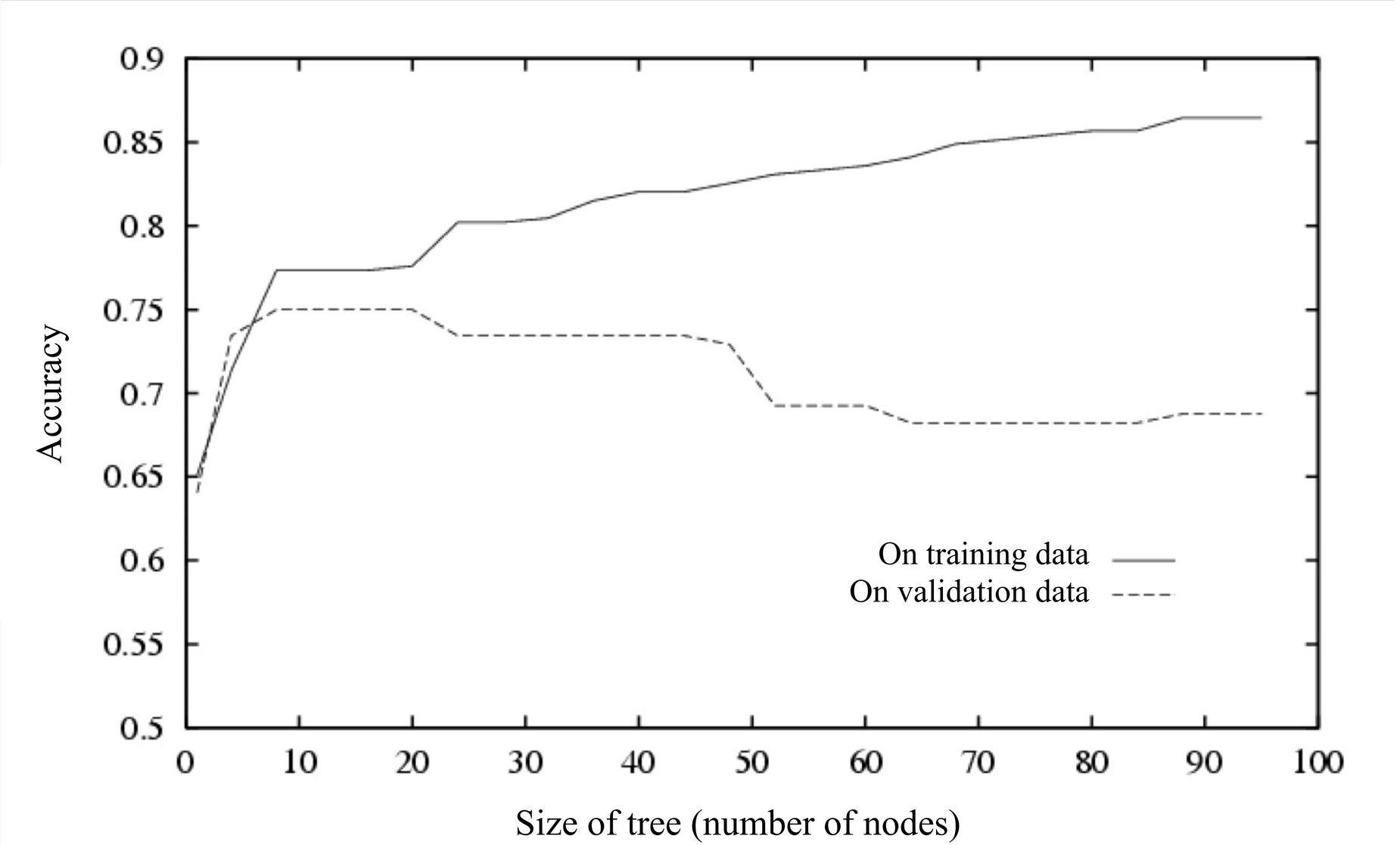


Figure from Tom Mitchell

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?

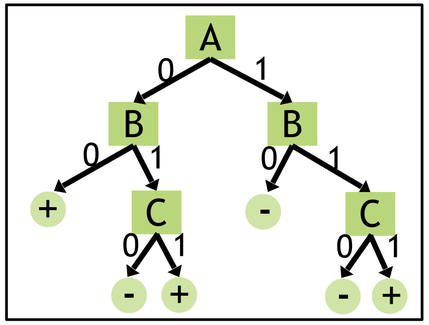
Expected value of highest die =
lower than 3.5?
the same: still 3.5?
higher than 3.5?

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

Expected value of each die = 3.5

A random experiment

- *Fix* a decision tree D

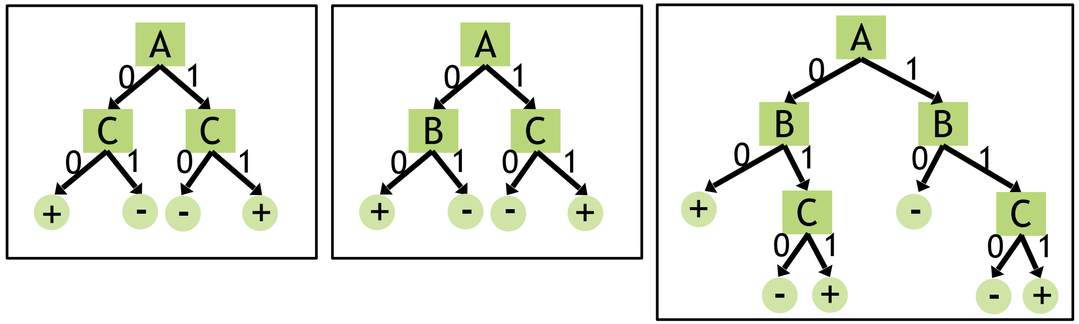


- Pick a random example: A = 1, B = 0, C = 1, label = +
- Classify it: score 1 if ✓ or 0 if ✗
- Expected value = true accuracy of our tree D

- Same expected value if we pick several examples (a training set) — but lower variance

A random experiment

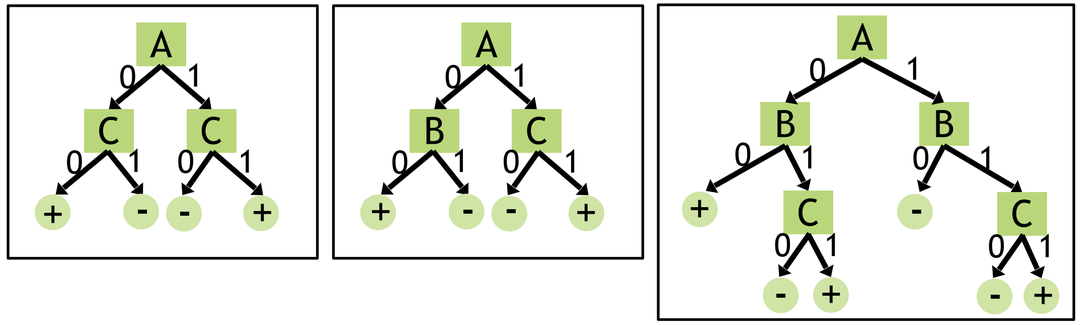
- *Fix* several trees D1, D2, D3, …



- Pick random example (or several):
  A = 1, B = 0, C = 1, 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, 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
   - (Greedily) remove the split that most decreases the validation error rate
   - Stop if no split improves validation error, otherwise repeat

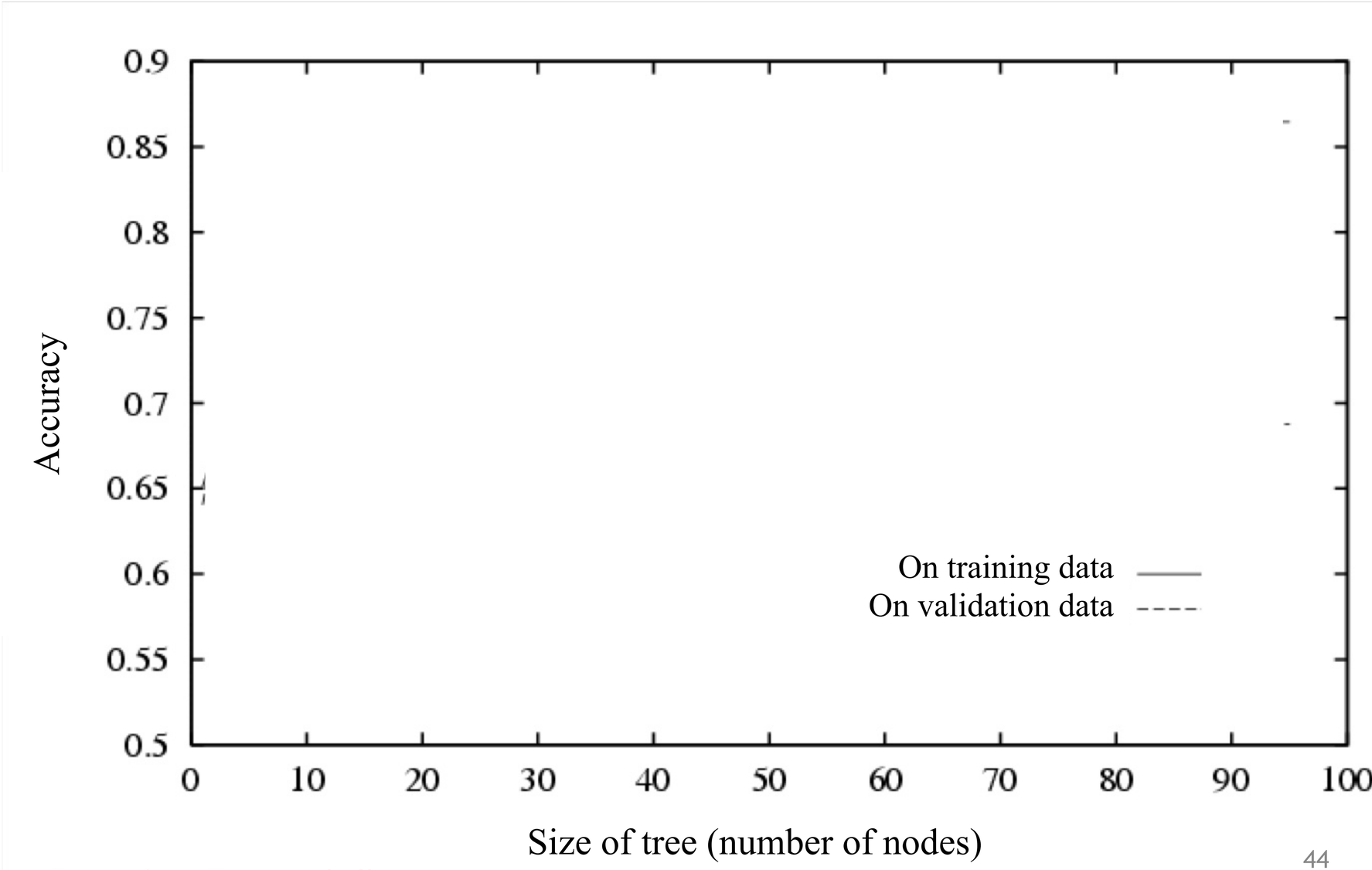


Figure from Tom Mitchell

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
   - (Greedily) remove the split that most decreases the validation error rate
   - Stop if no split improves validation error, otherwise repeat

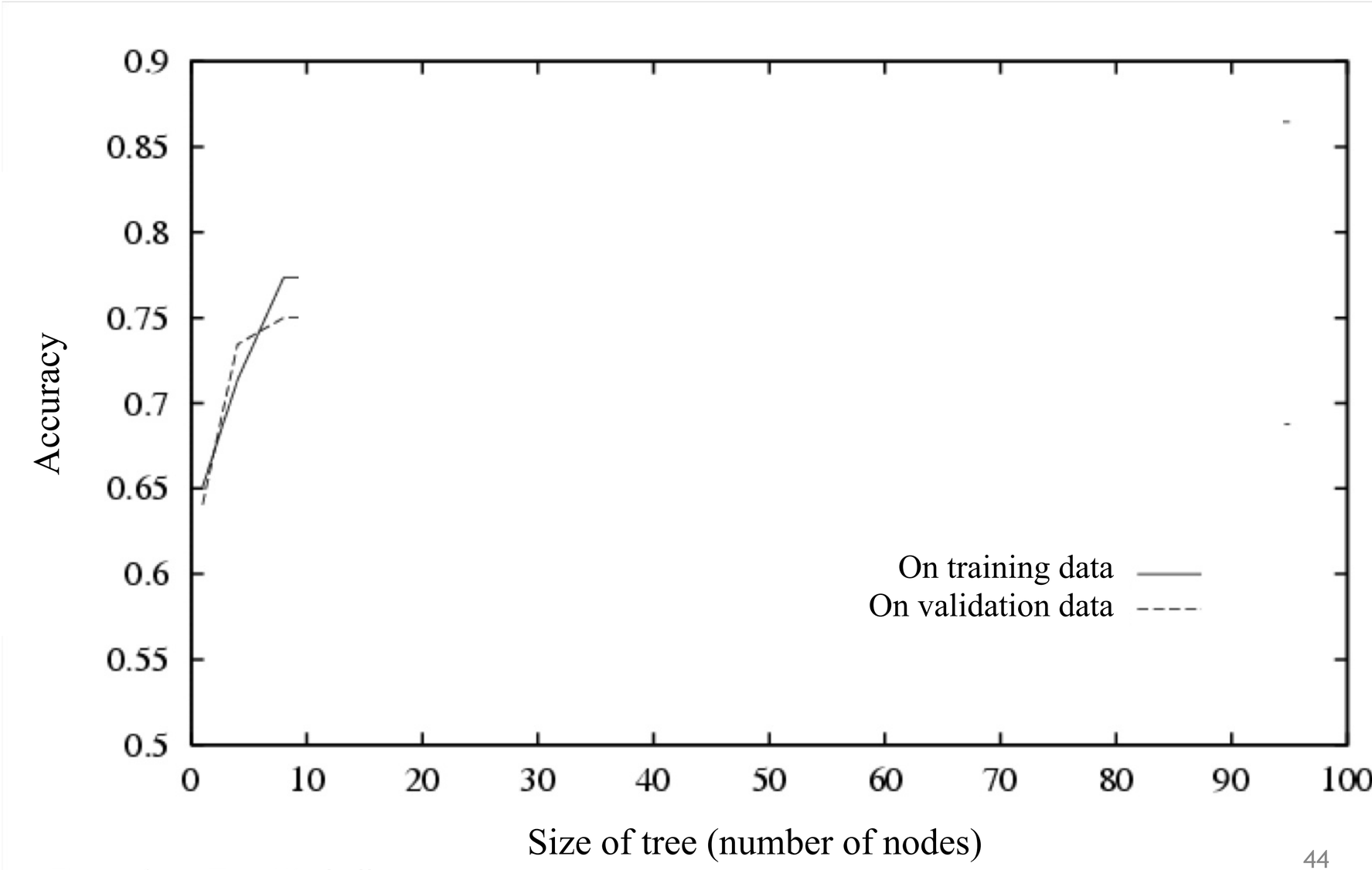


Figure from Tom Mitchell

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
   - (Greedily) remove the split that most decreases the validation error rate
   - Stop if no split improves validation error, otherwise repeat

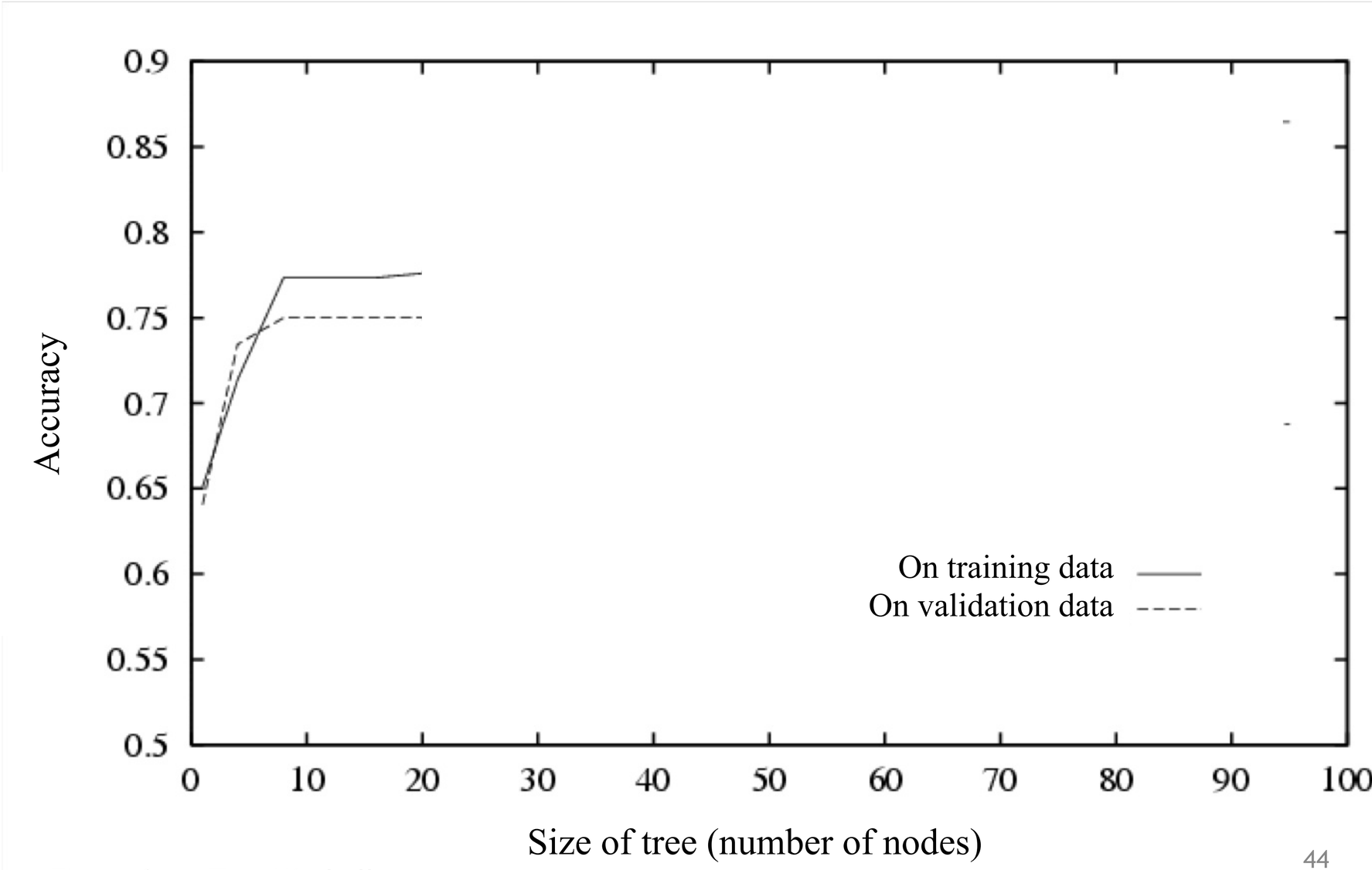


Figure from Tom Mitchell

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
   - (Greedily) remove the split that most decreases the validation error rate
   - Stop if no split improves validation error, otherwise repeat

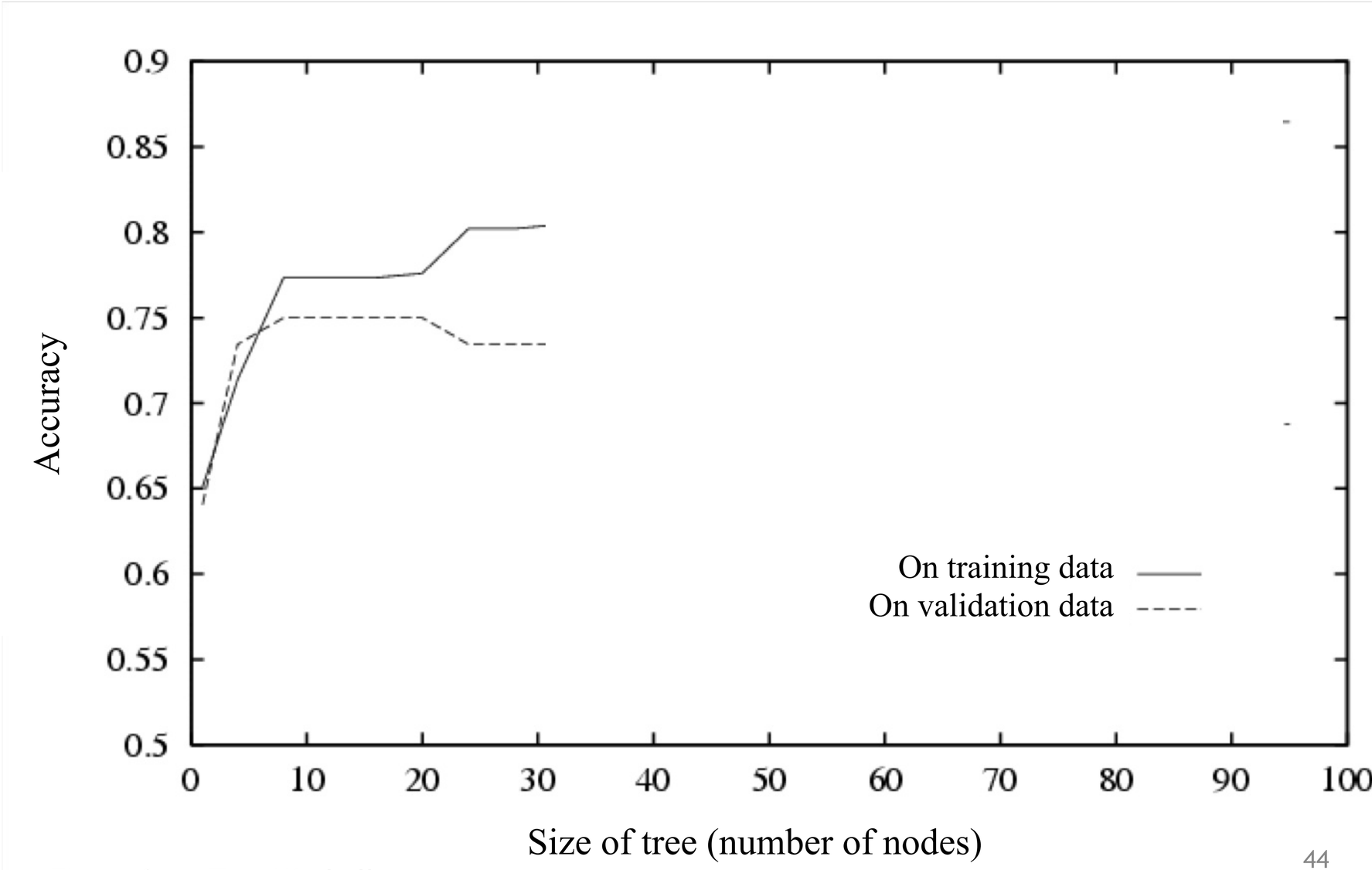


Figure from Tom Mitchell

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
   - (Greedily) remove the split that most decreases the validation error rate
   - Stop if no split improves validation error, otherwise repeat

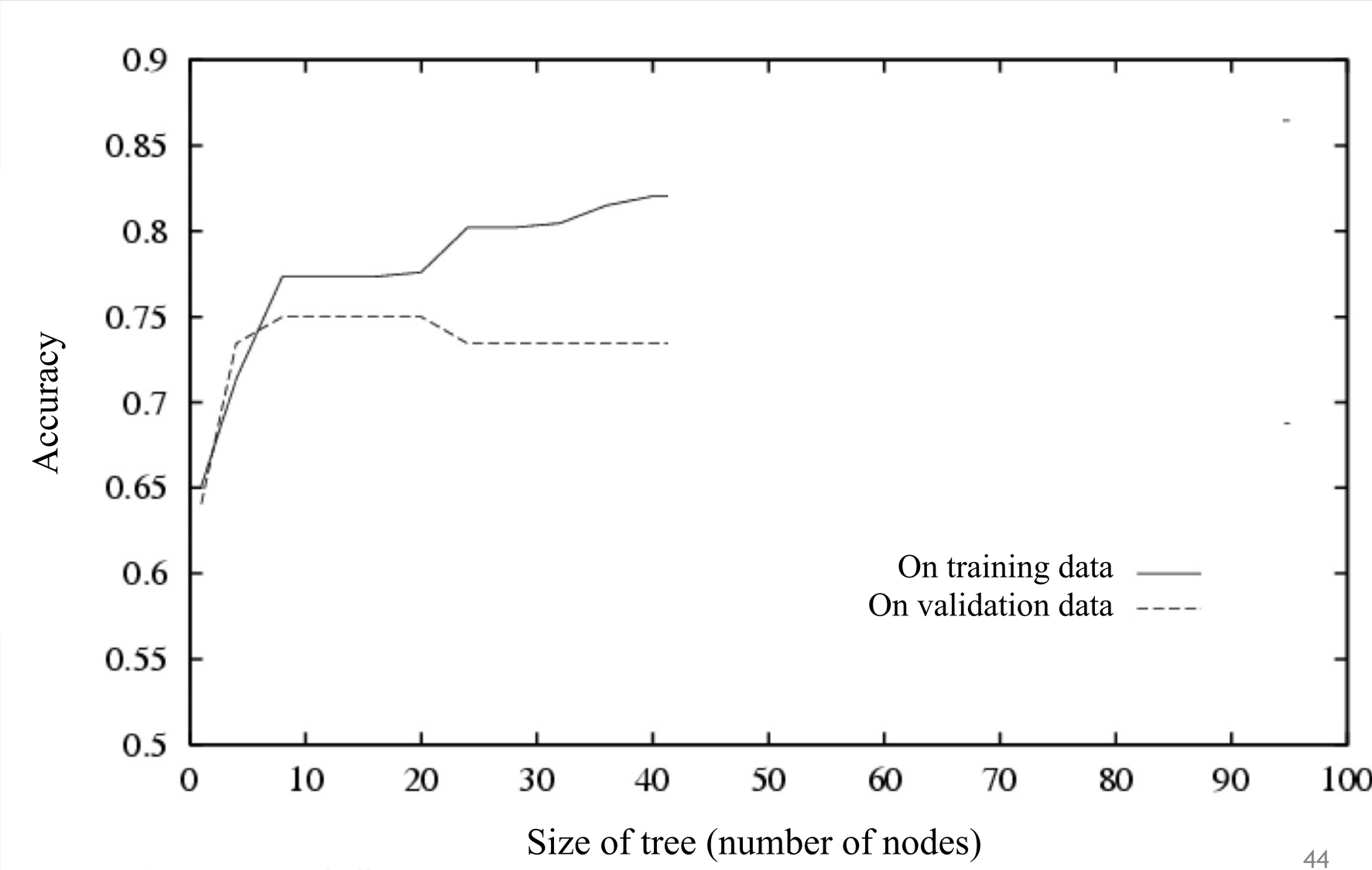


Figure from Tom Mitchell

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
   - (Greedily) remove the split that most decreases the validation error rate
   - Stop if no split improves validation error, otherwise repeat

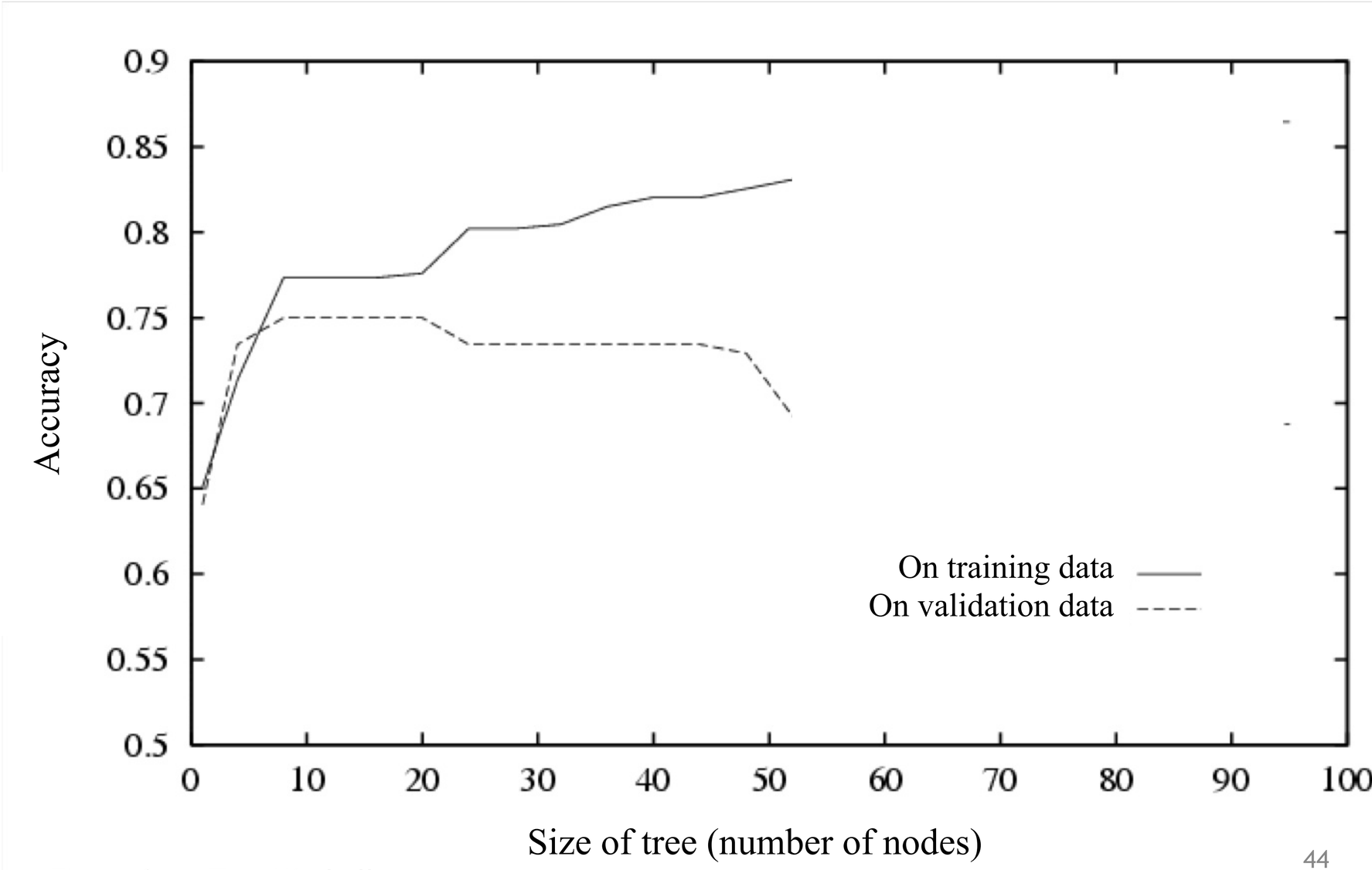


Figure from Tom Mitchell

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
   - (Greedily) remove the split that most decreases the validation error rate
   - Stop if no split improves validation error, otherwise repeat

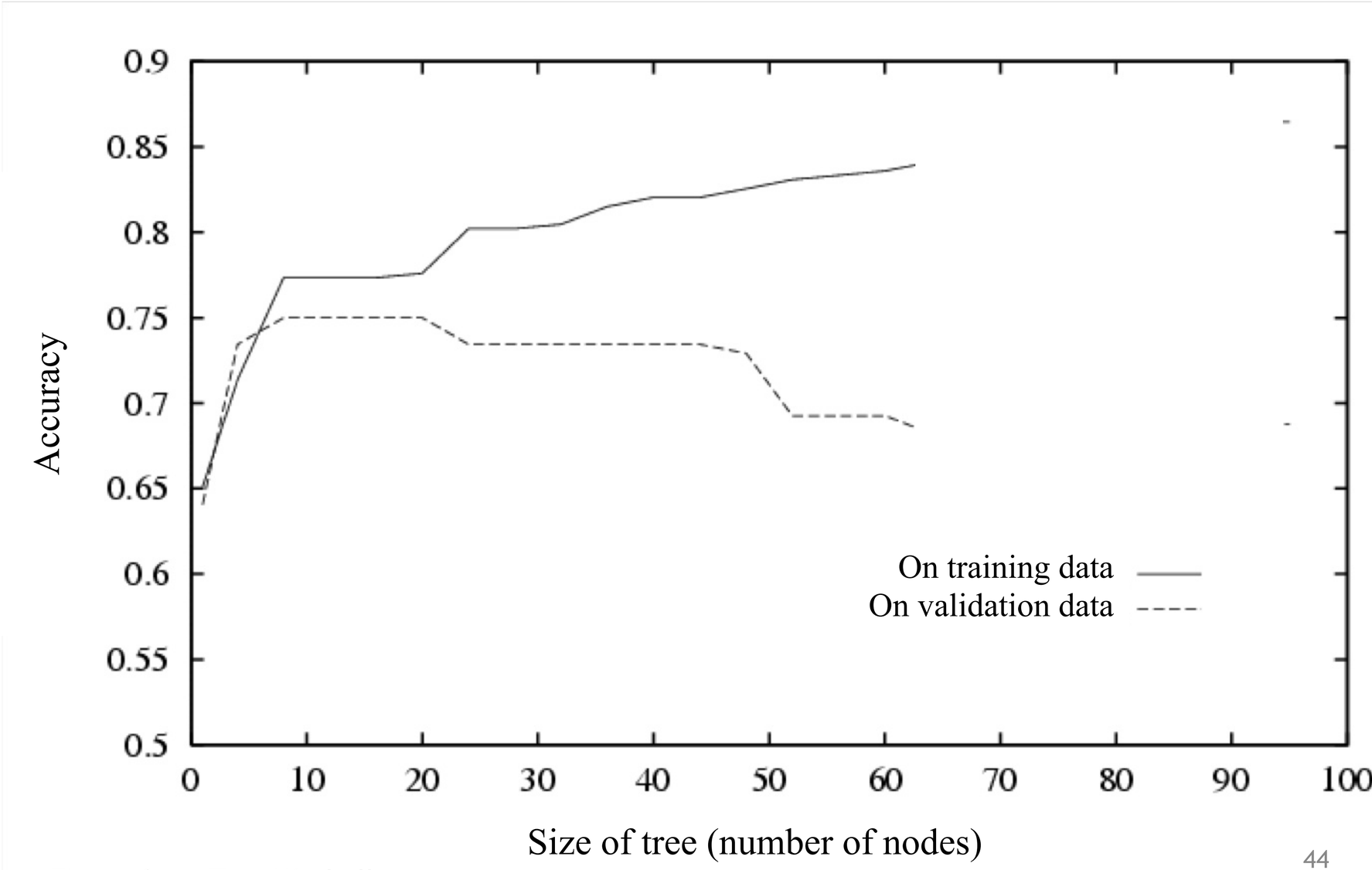


Figure from Tom Mitchell

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
   - (Greedily) remove the split that most decreases the validation error rate
   - Stop if no split improves validation error, otherwise repeat

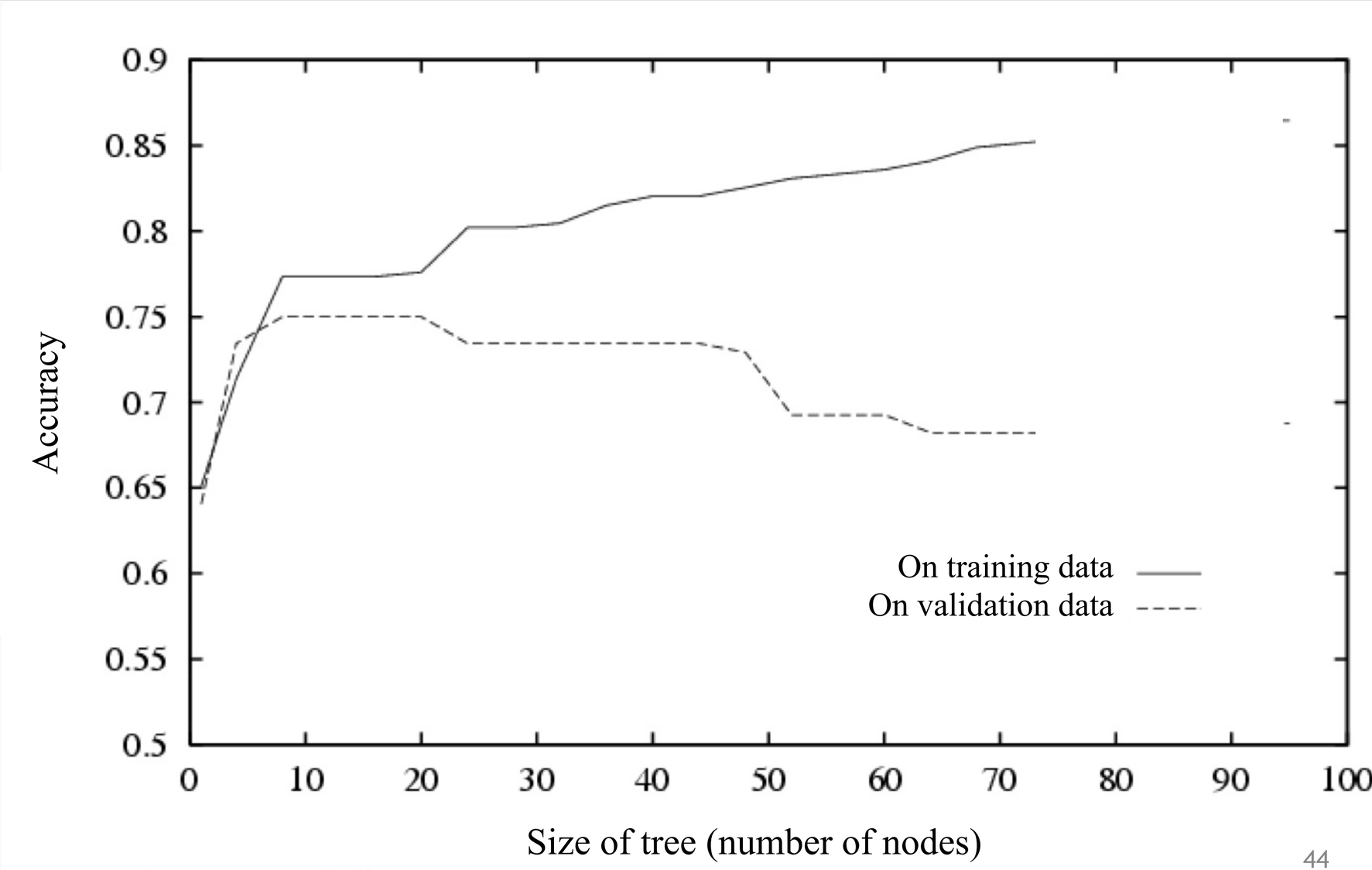


Figure from Tom Mitchell

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
   - (Greedily) remove the split that most decreases the validation error rate
   - Stop if no split improves validation error, otherwise repeat

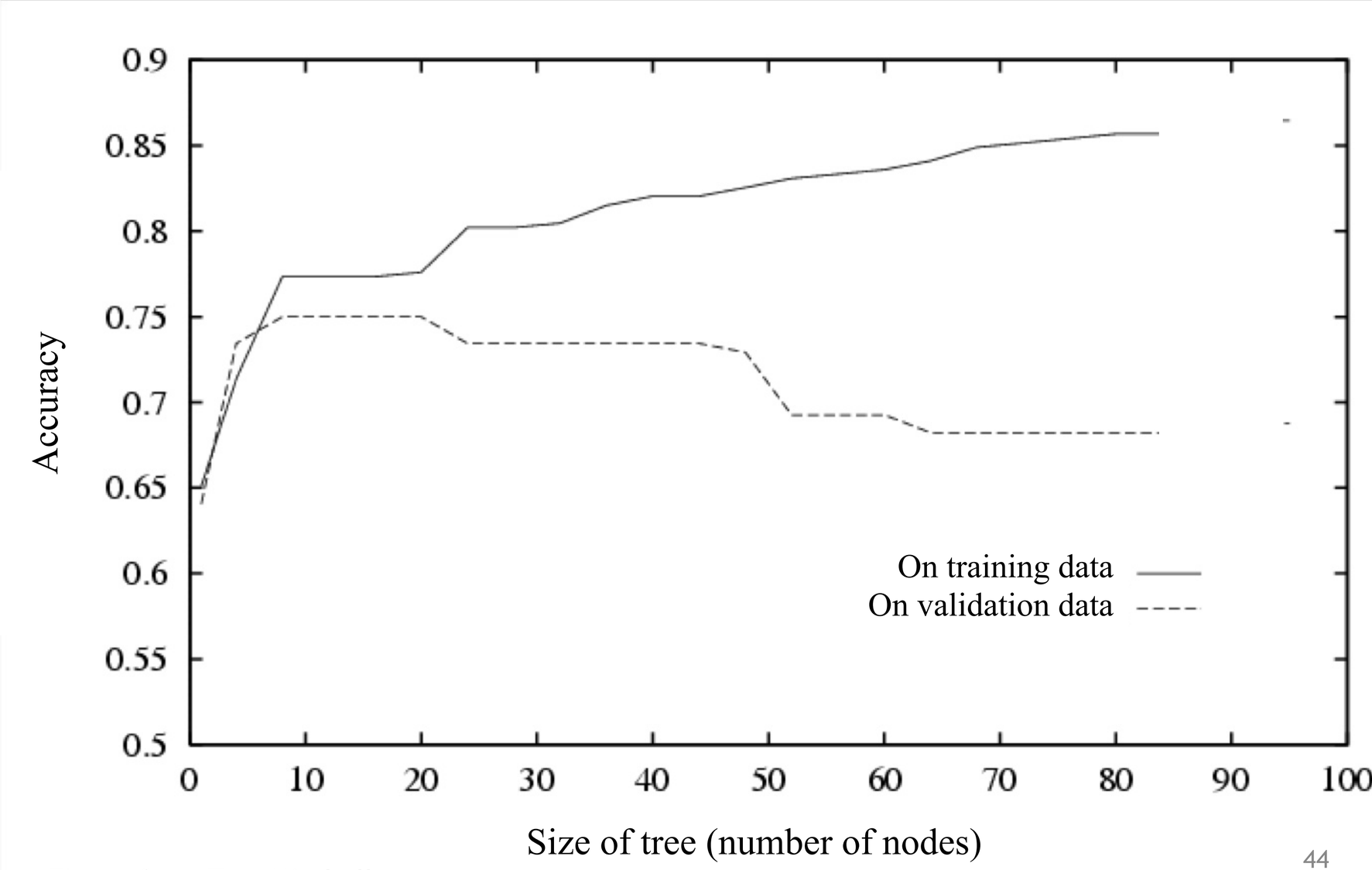


Figure from Tom Mitchell

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
   - (Greedily) remove the split that most decreases the validation error rate
   - Stop if no split improves validation error, otherwise repeat

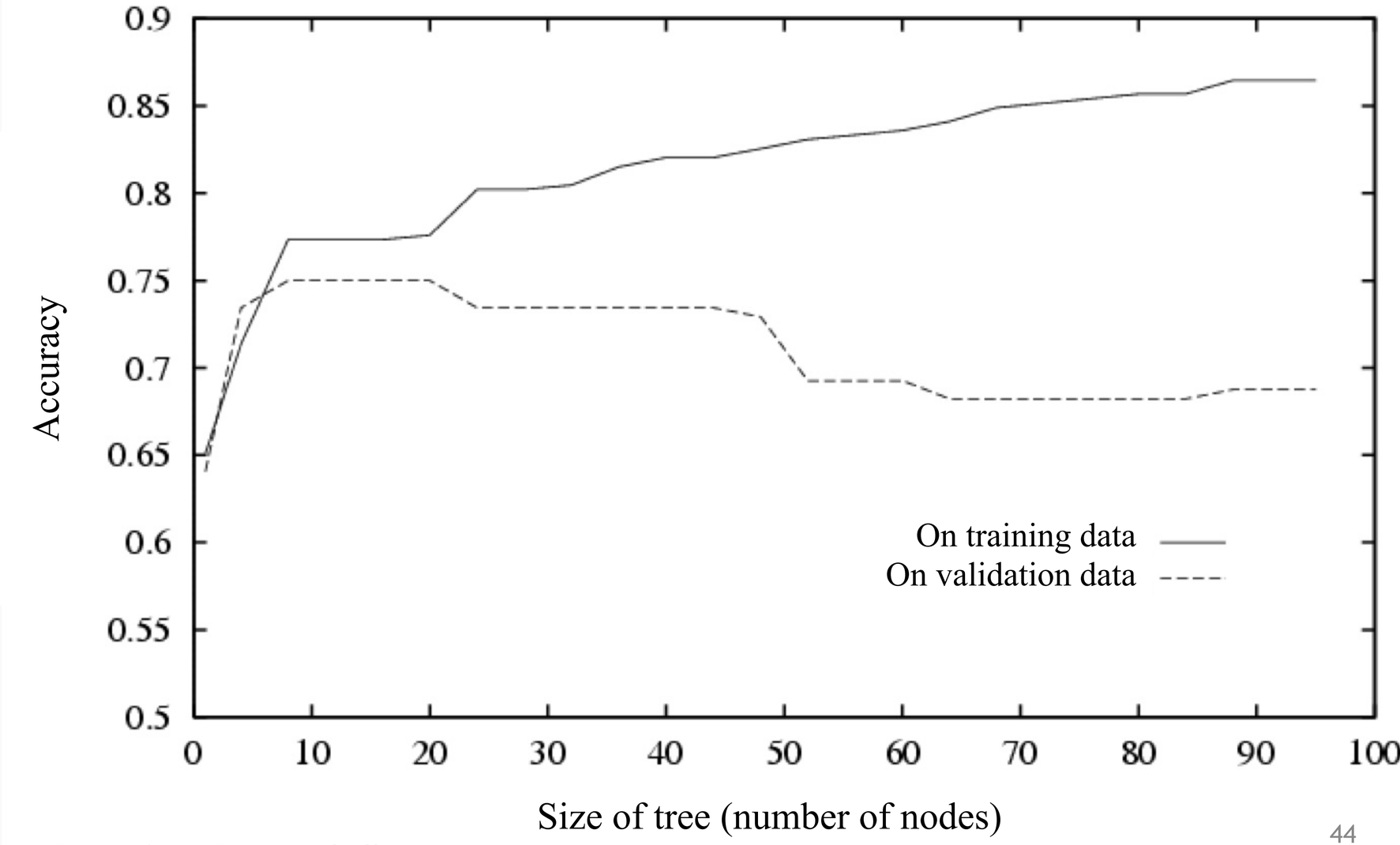


Figure from Tom Mitchell

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
   - (Greedily) remove the split that most decreases the validation error rate
   - Stop if no split improves validation error, otherwise repeat

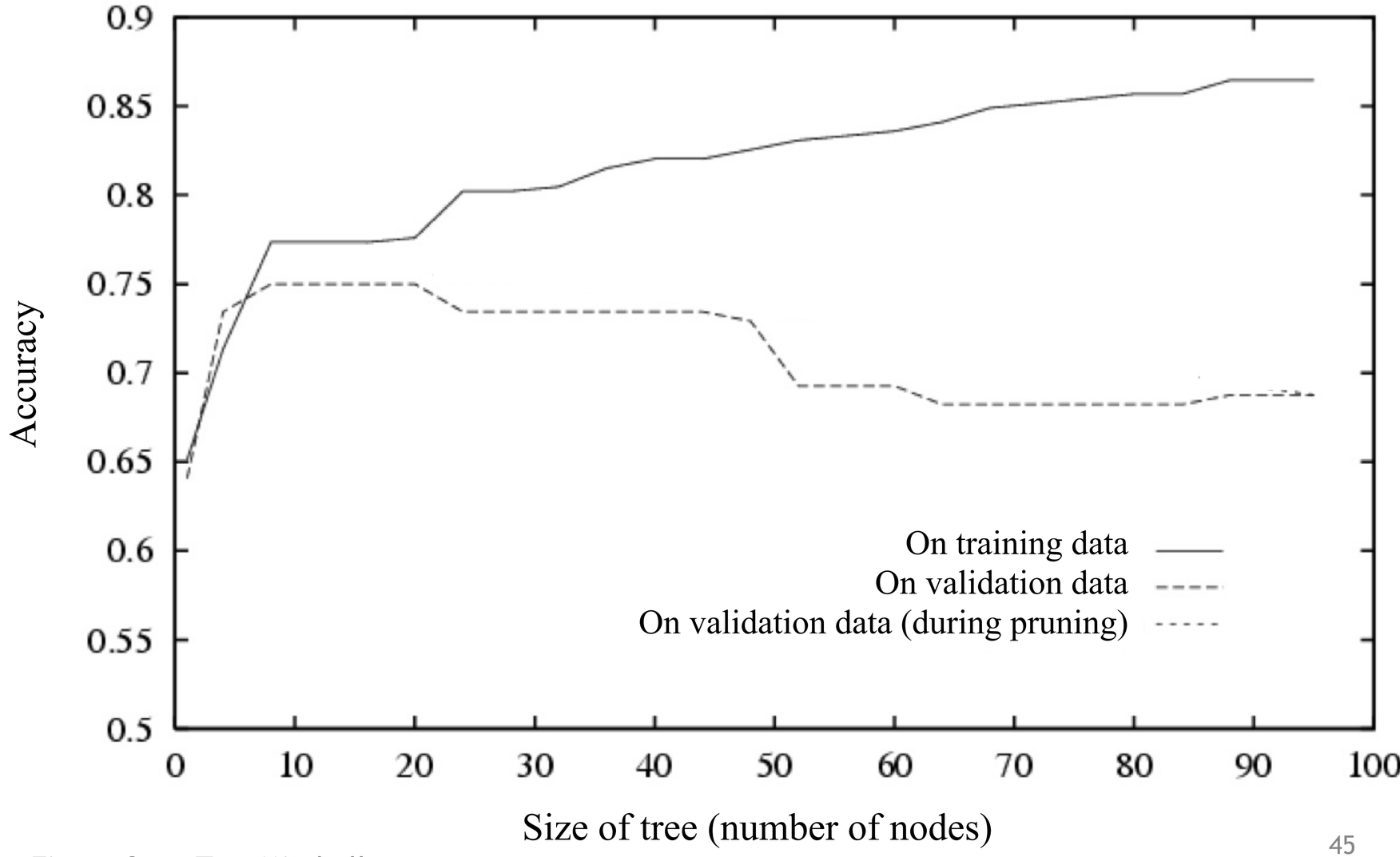


Figure from Tom Mitchell

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
   - (Greedily) remove the split that most decreases the validation error rate
   - Stop if no split improves validation error, otherwise repeat

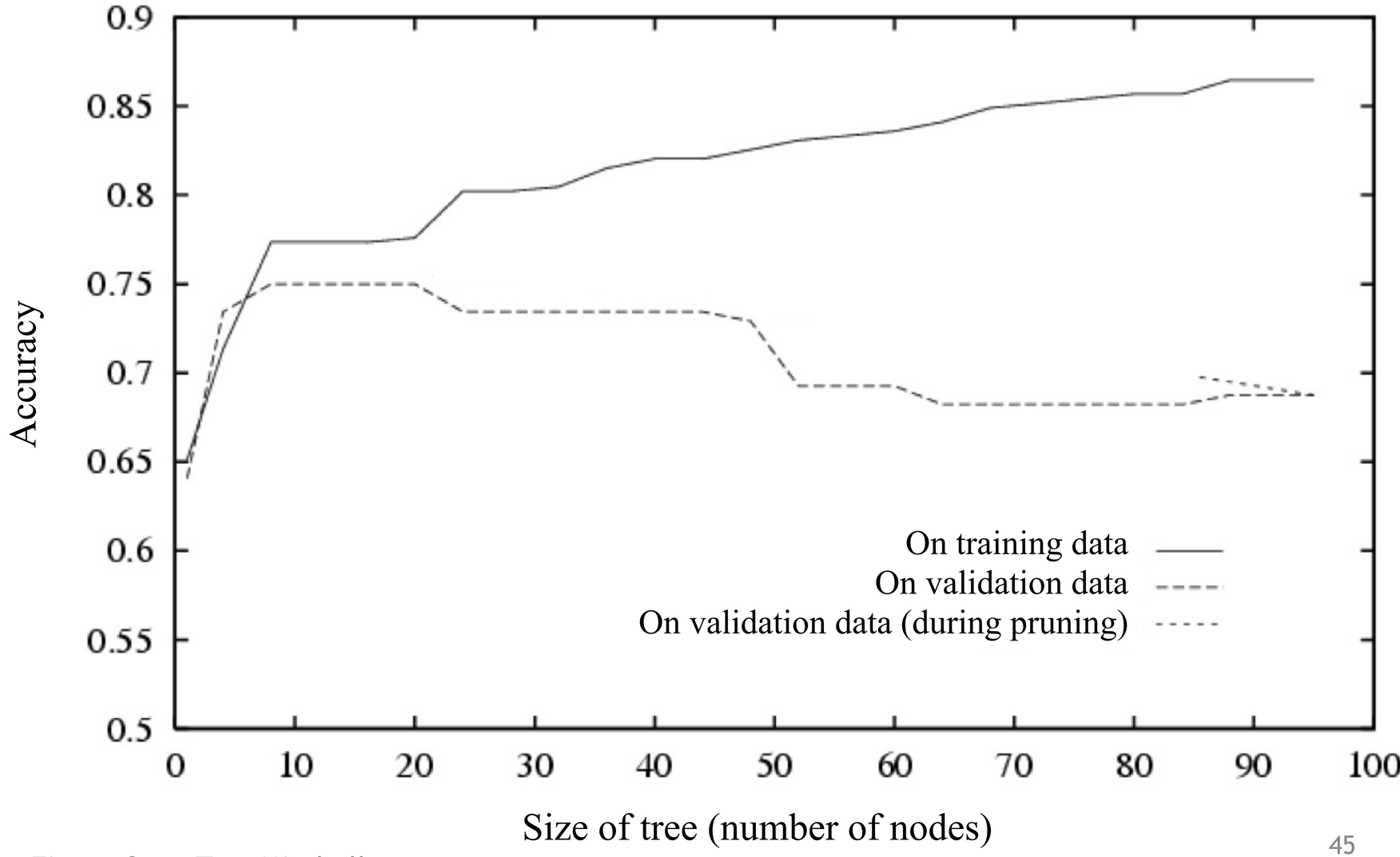


Figure from Tom Mitchell

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
   - (Greedily) remove the split that most decreases the validation error rate
   - Stop if no split improves validation error, otherwise repeat

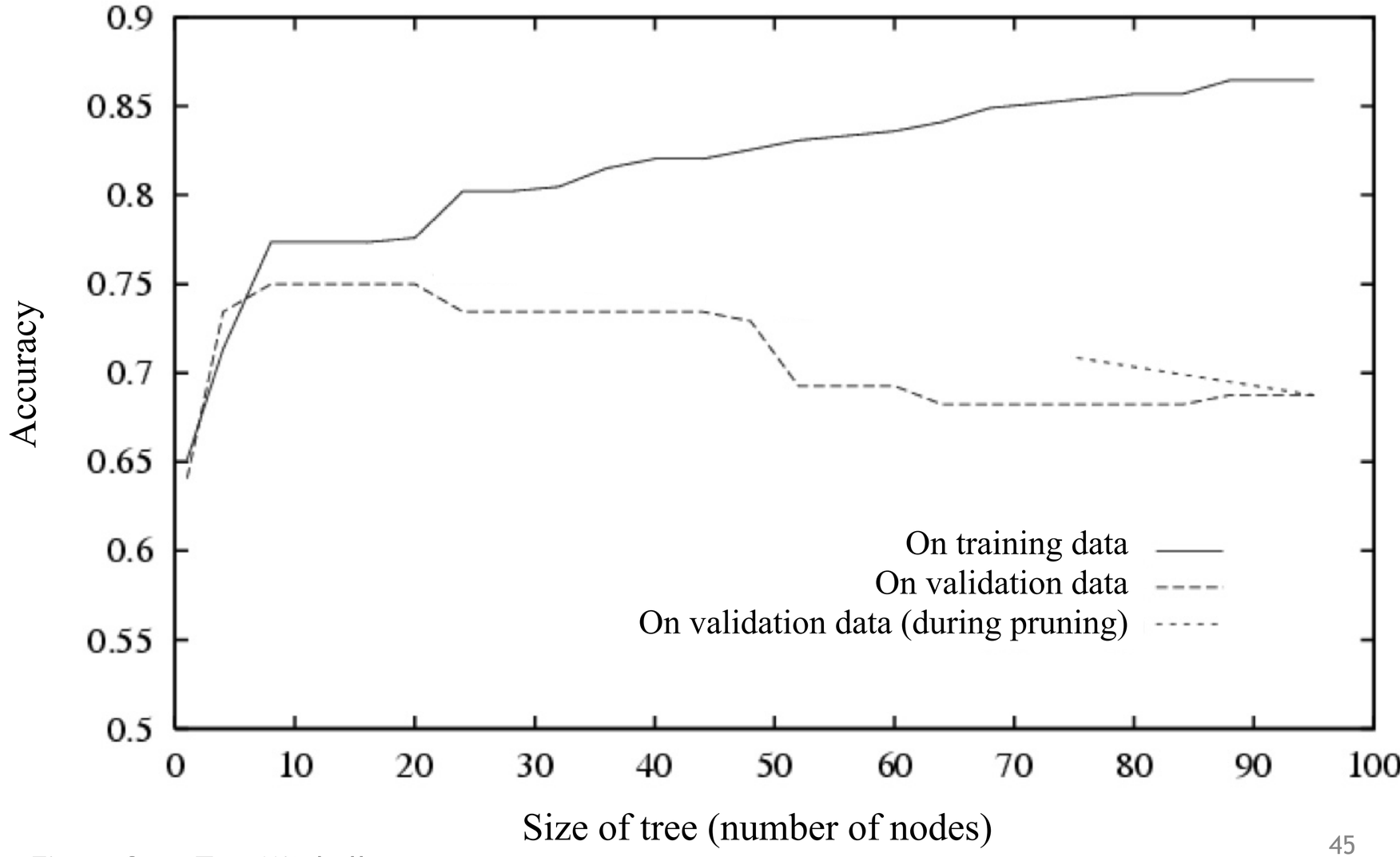


Figure from Tom Mitchell

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
   - (Greedily) remove the split that most decreases the validation error rate
   - Stop if no split improves validation error, otherwise repeat

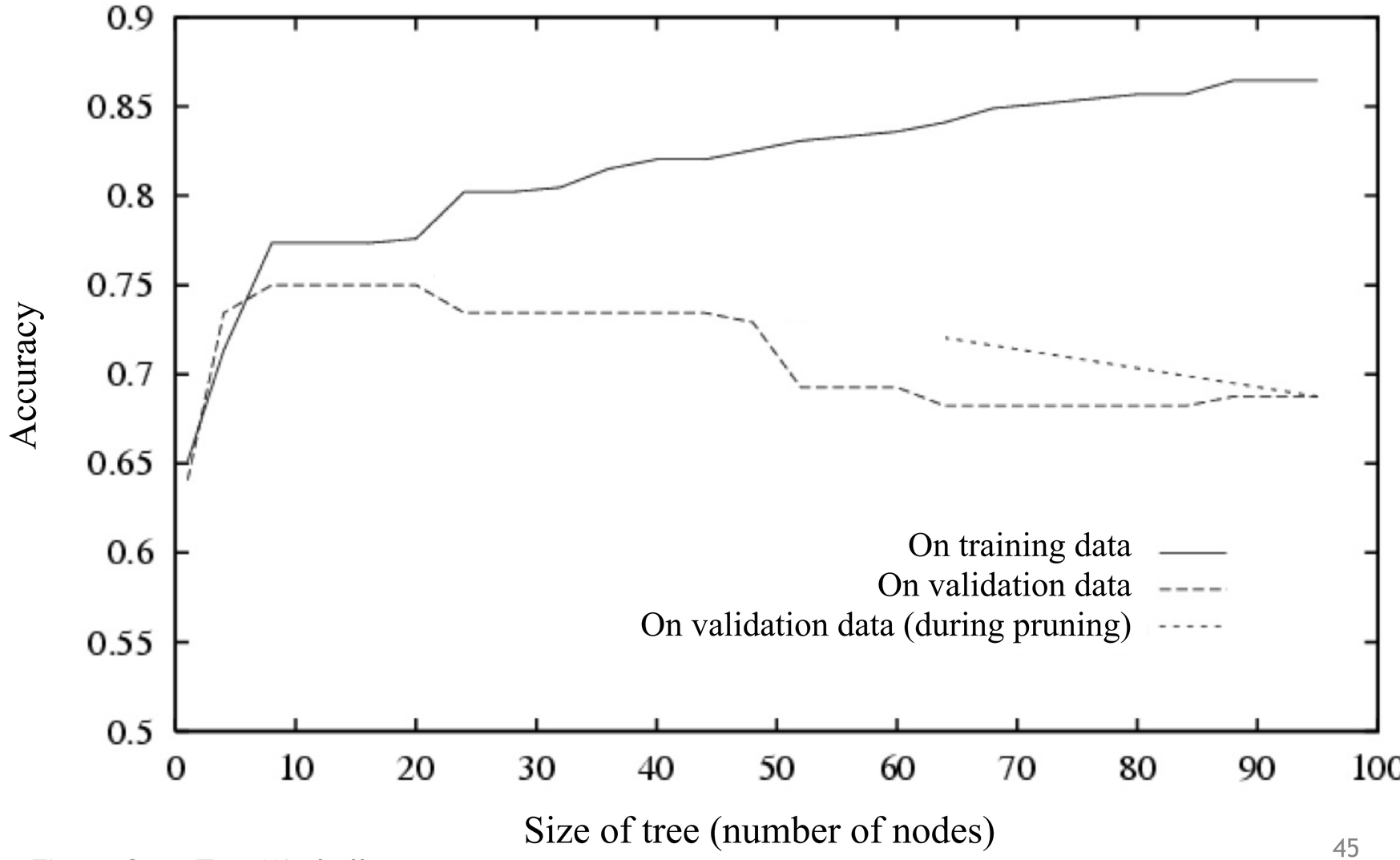


Figure from Tom Mitchell

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
   - (Greedily) remove the split that most decreases the validation error rate
   - Stop if no split improves validation error, otherwise repeat

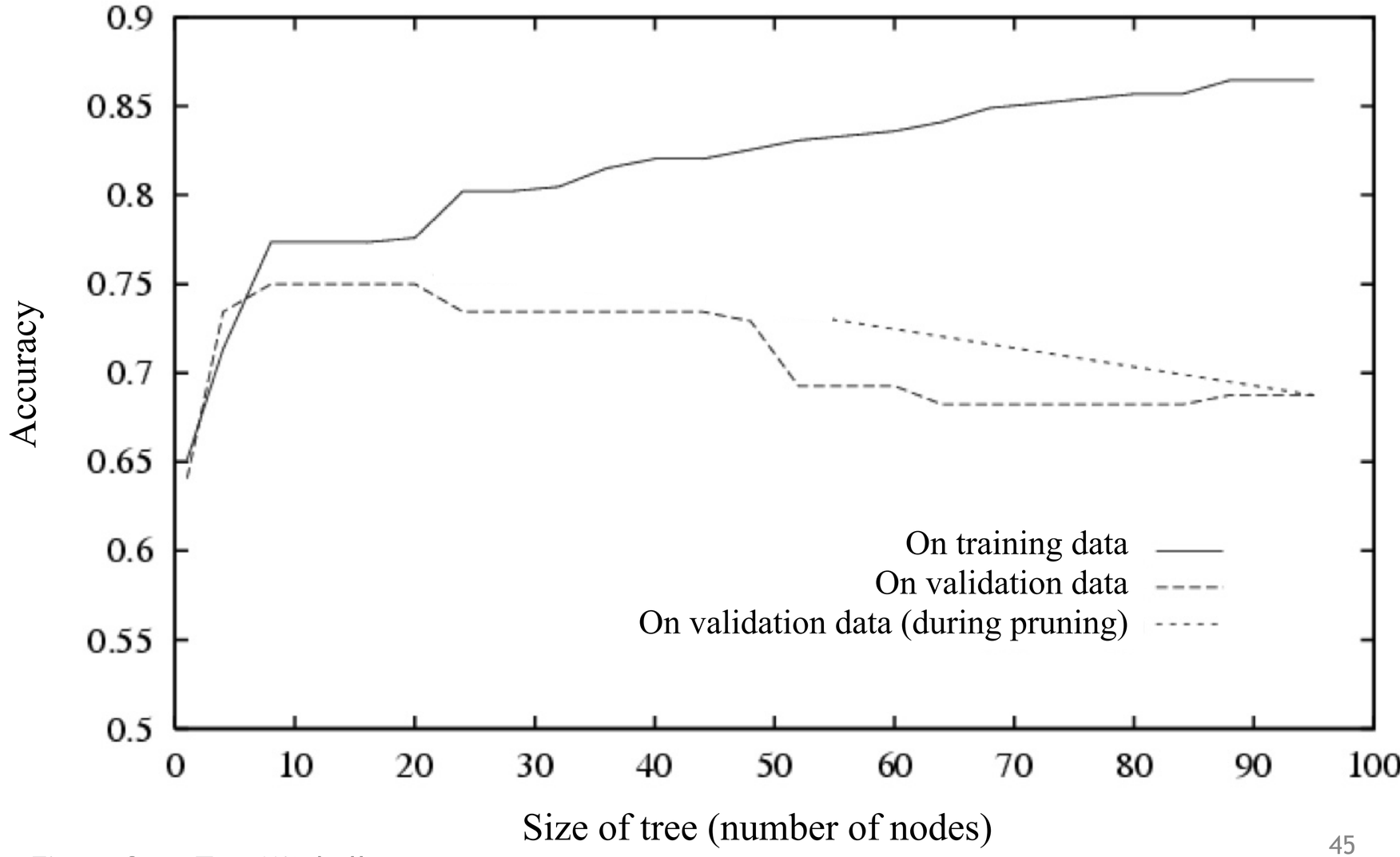


Figure from Tom Mitchell

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
   - (Greedily) remove the split that most decreases the validation error rate
   - Stop if no split improves validation error, otherwise repeat

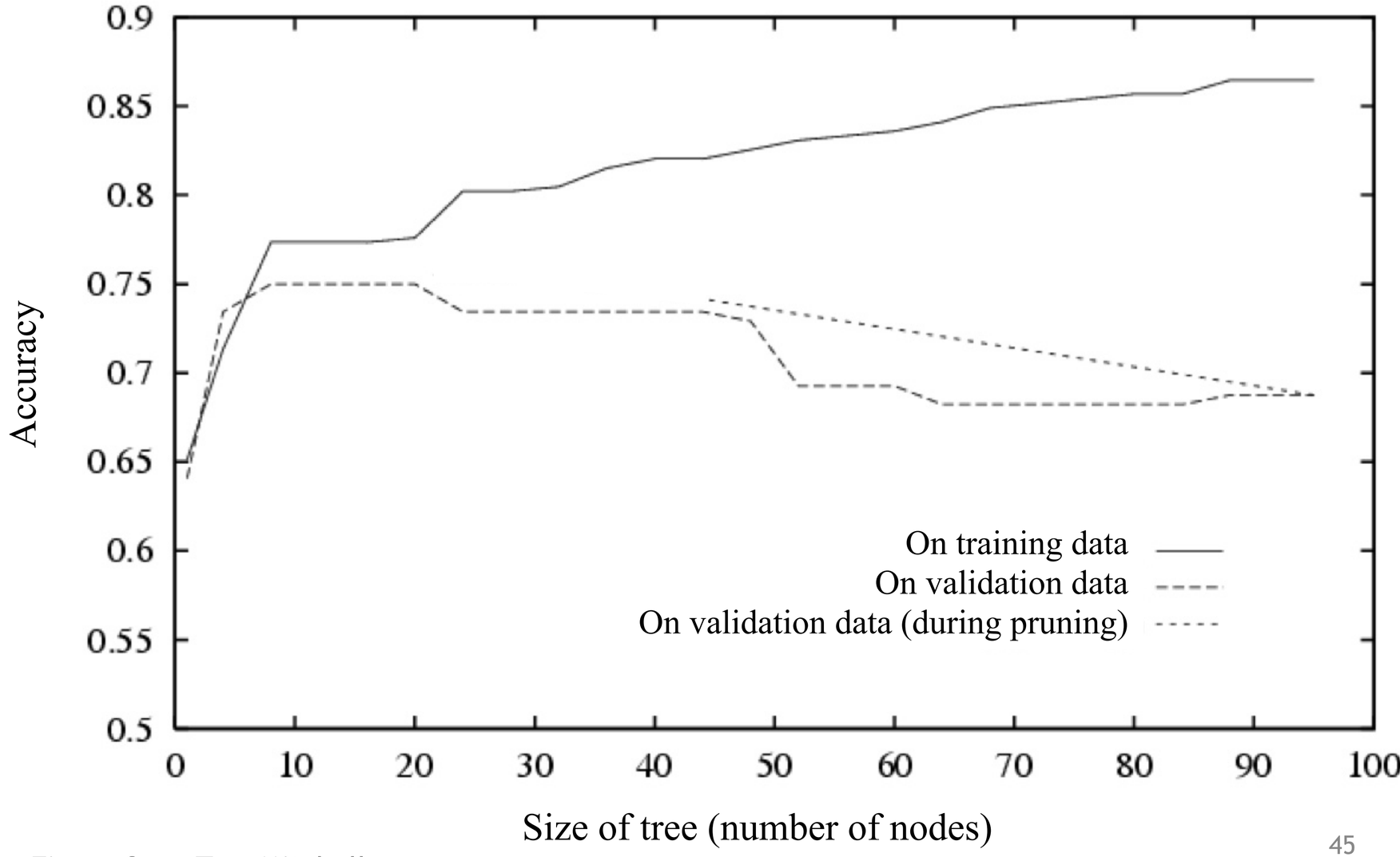


Figure from Tom Mitchell

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
   - (Greedily) remove the split that most decreases the validation error rate
   - Stop if no split improves validation error, otherwise repeat

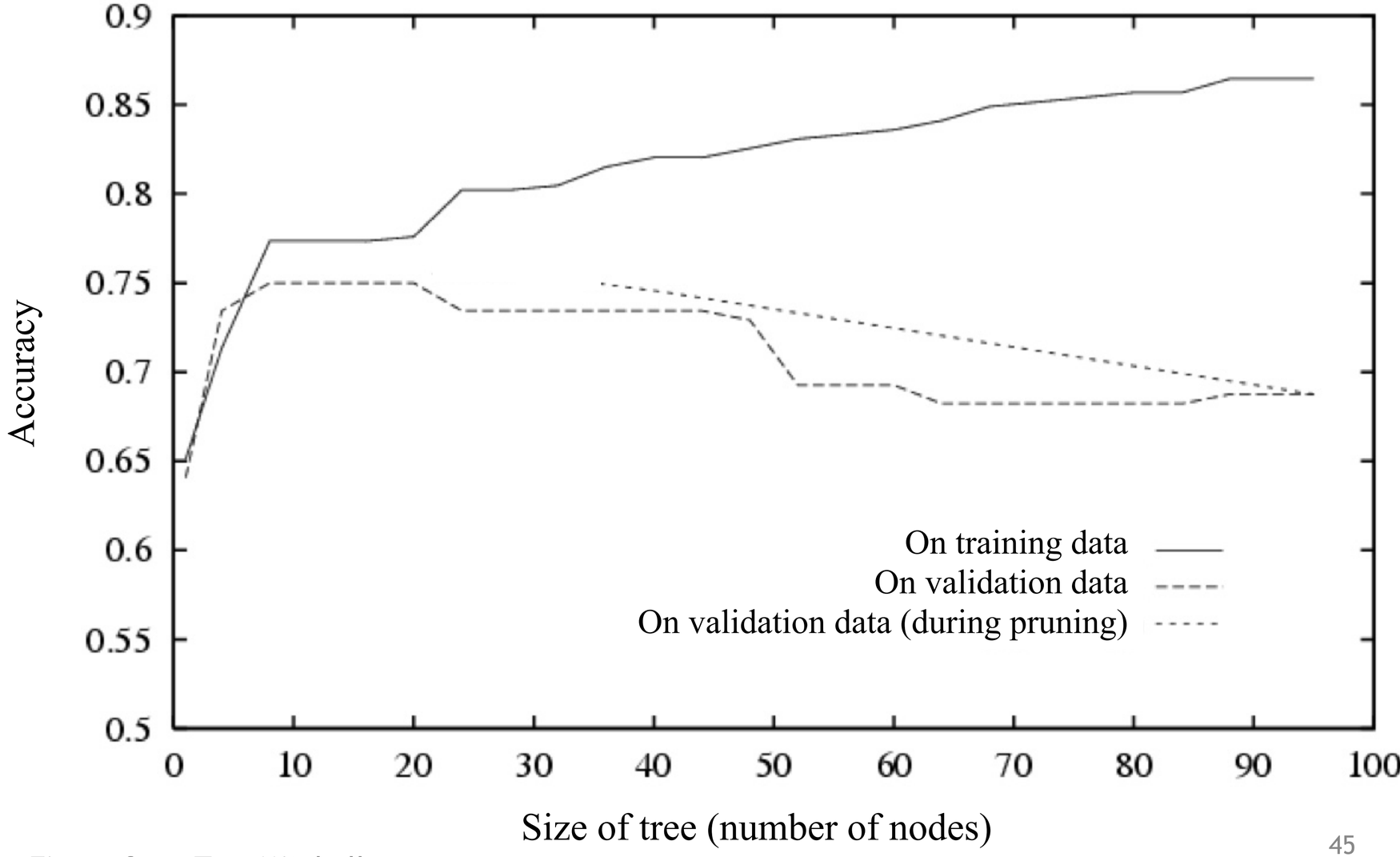


Figure from Tom Mitchell

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
   - (Greedily) remove the split that most decreases the validation error rate
   - Stop if no split improves validation error, otherwise repeat

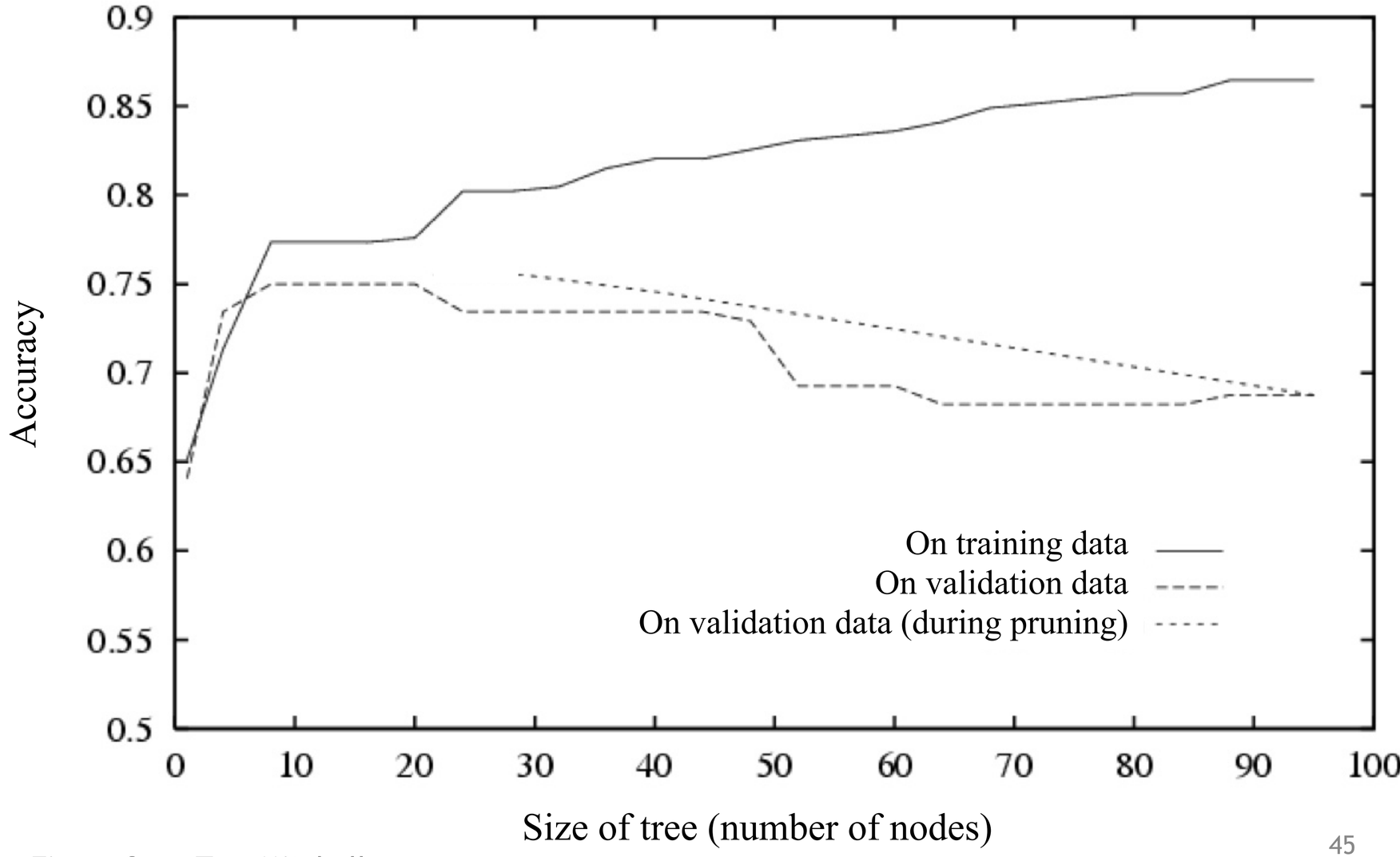


Figure from Tom Mitchell

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
   - (Greedily) remove the split that most decreases the validation error rate
   - Stop if no split improves validation error, otherwise repeat

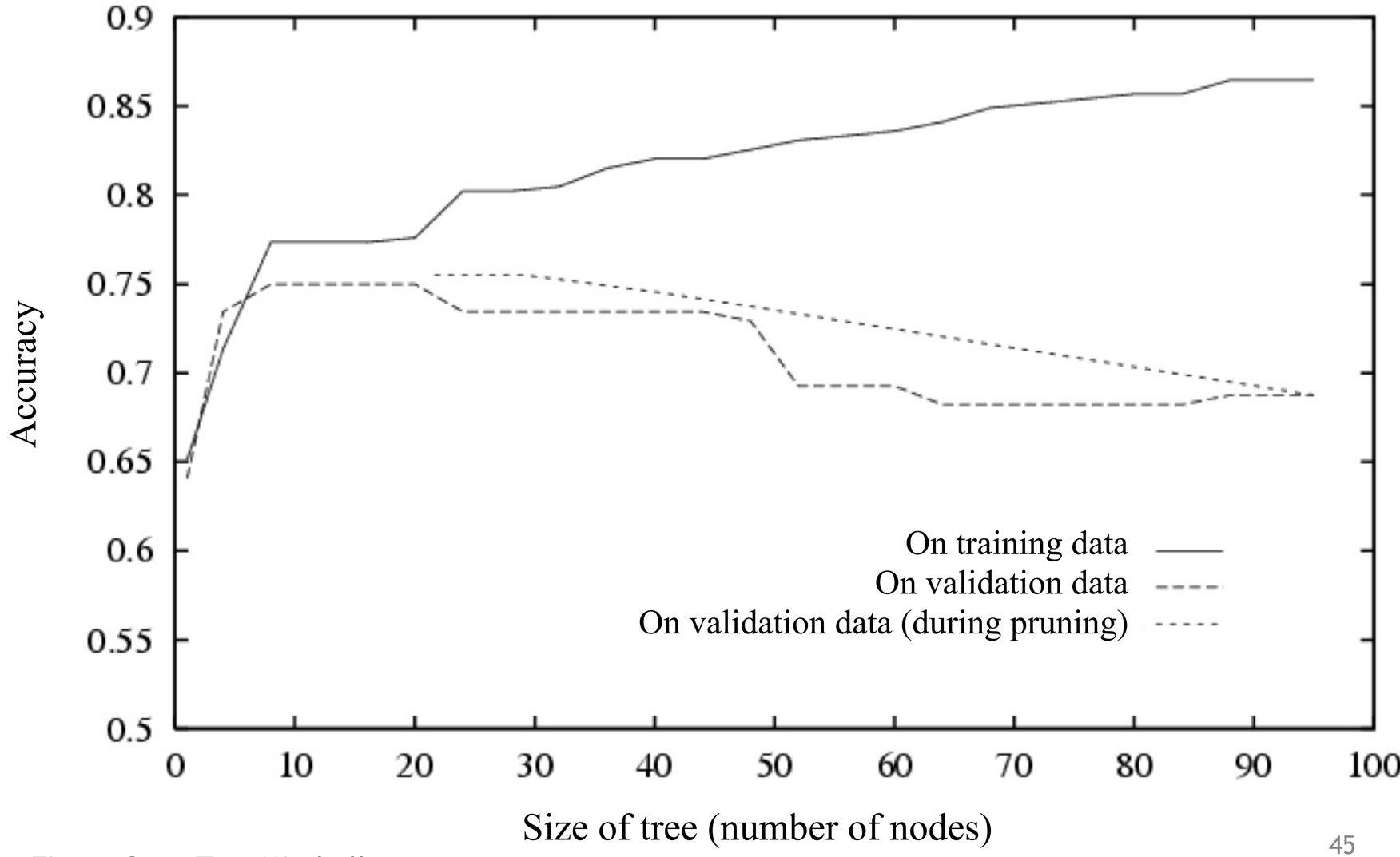
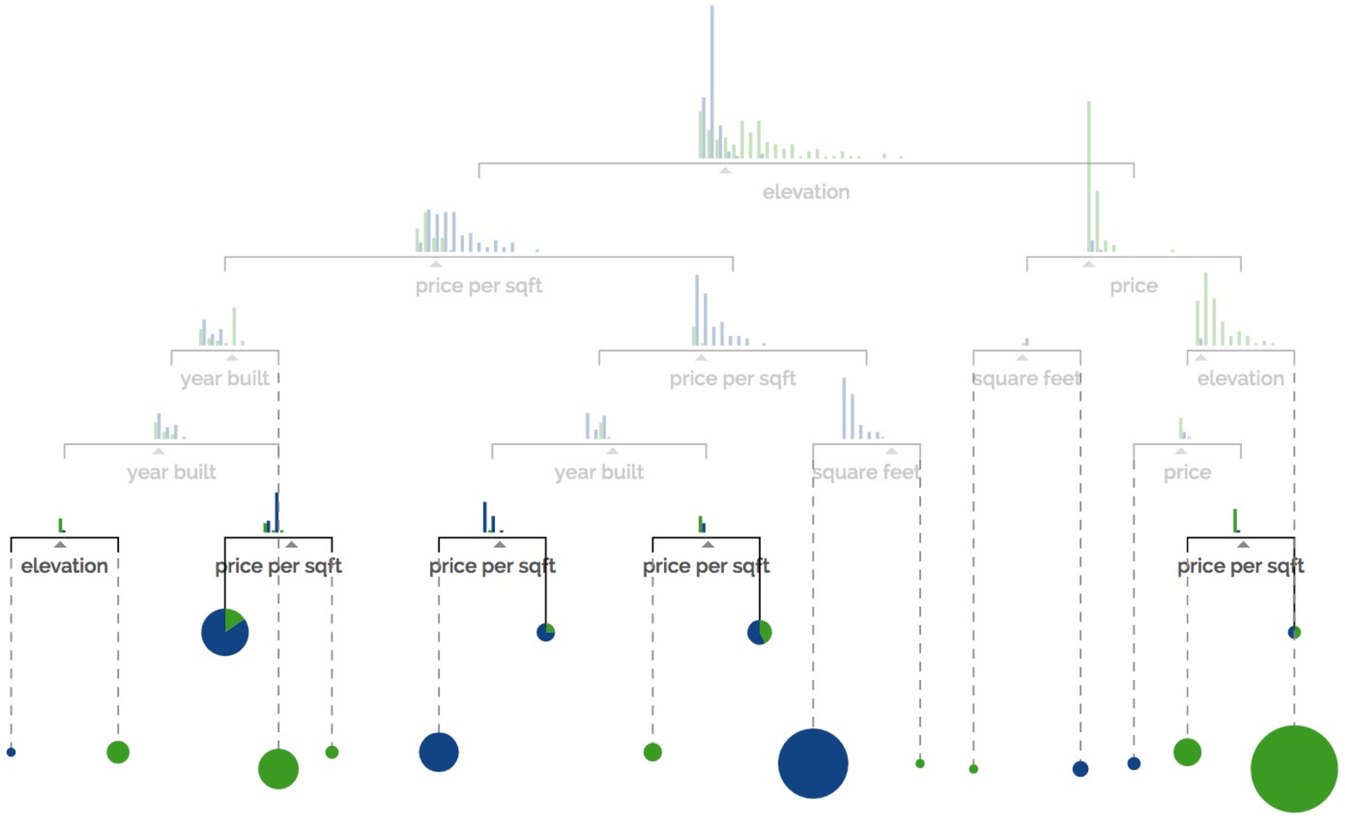


Figure from Tom Mitchell

Decision Tree — Pruning

Suppose you're trying to predict whether a house is located in ***San Francisco*** or ***New York*** based on the following features:

- price
- sq ft
- price per sq ft
- year built
- elevation
- # bathrooms
- # bedrooms

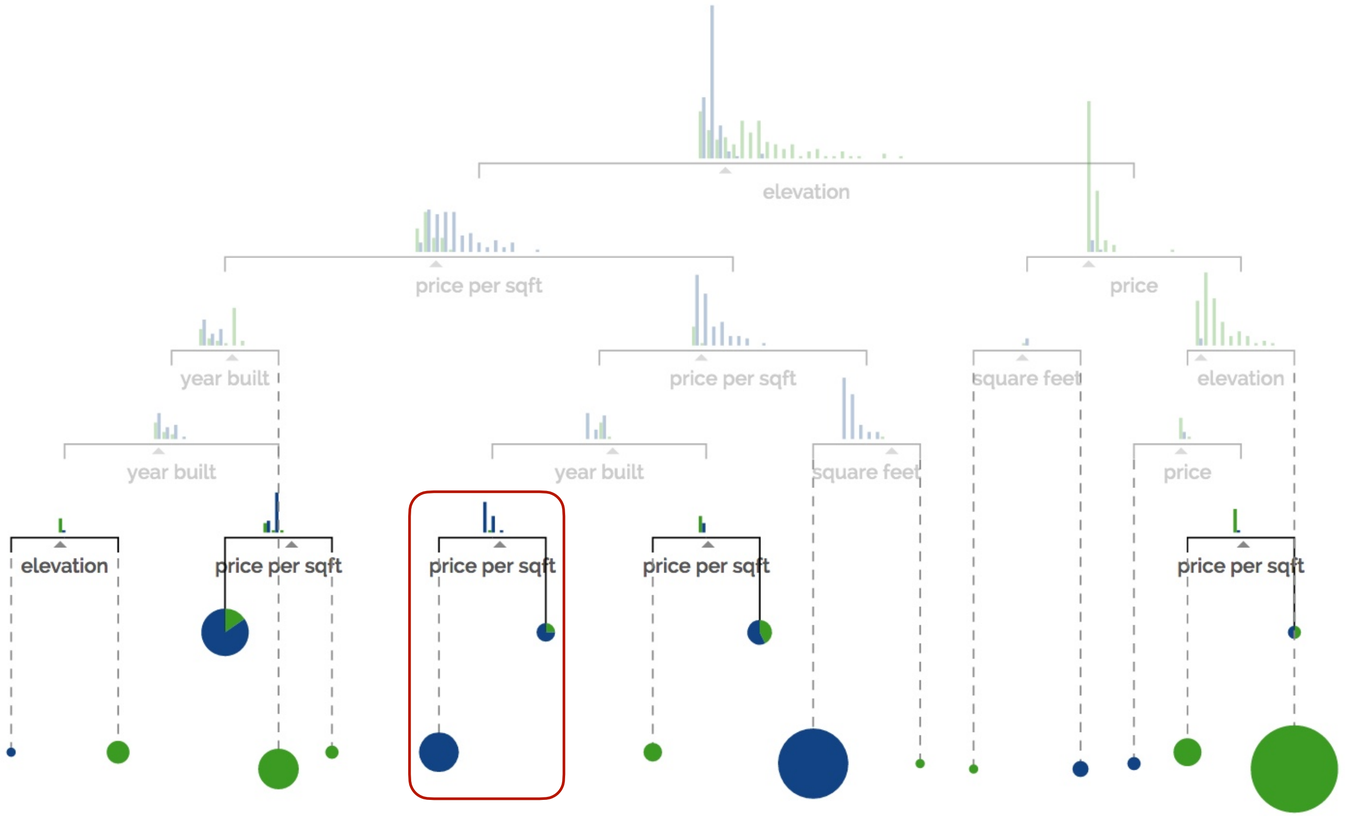


slide credit: Ameet Talwalkar

Decision Tree — Pruning

Suppose you're trying to predict whether a house is located in ***San Francisco*** or ***New York*** based on the following features:

- price
- sq ft
- price per sq ft
- year built
- elevation
- # bathrooms
- # bedrooms

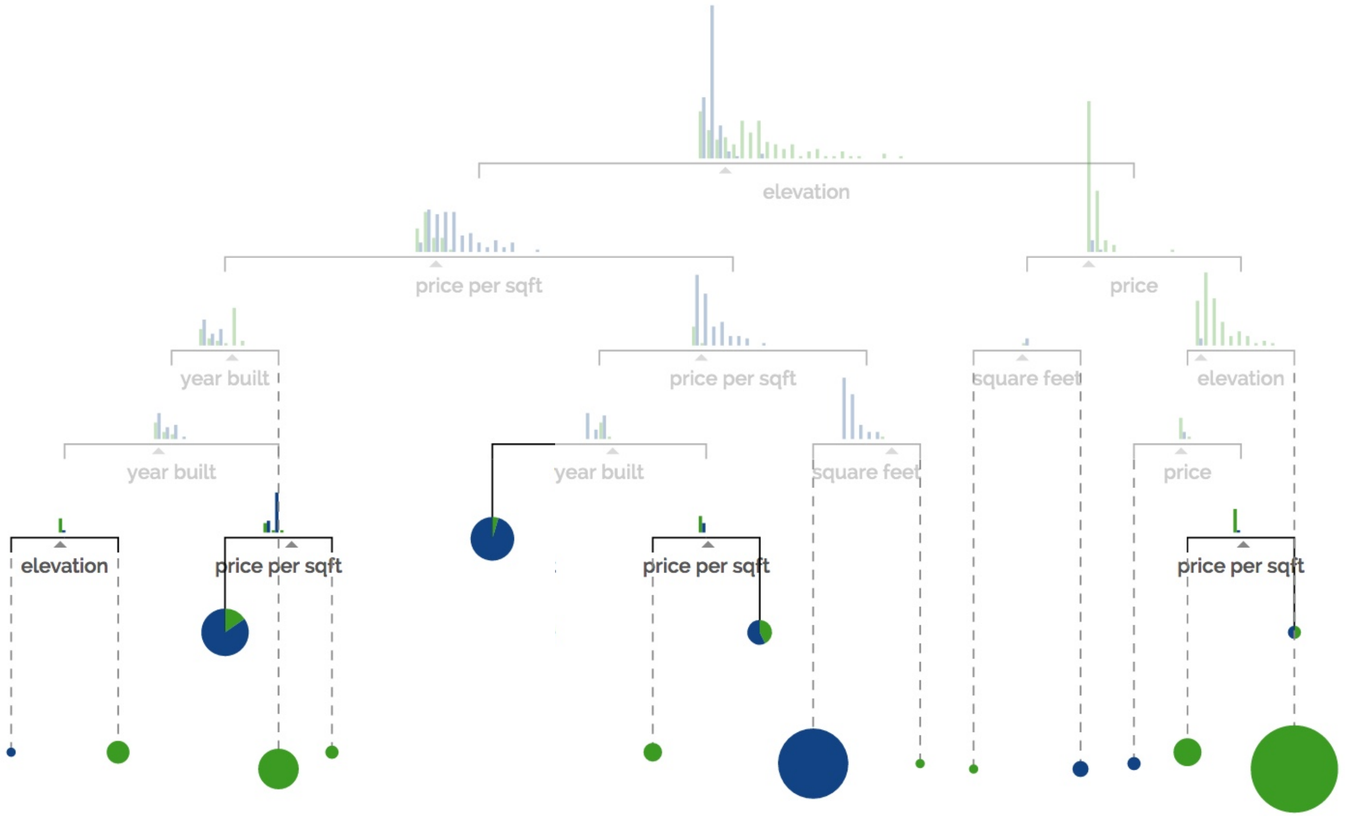


slide credit: Ameet Talwalkar

Decision Tree — Pruning

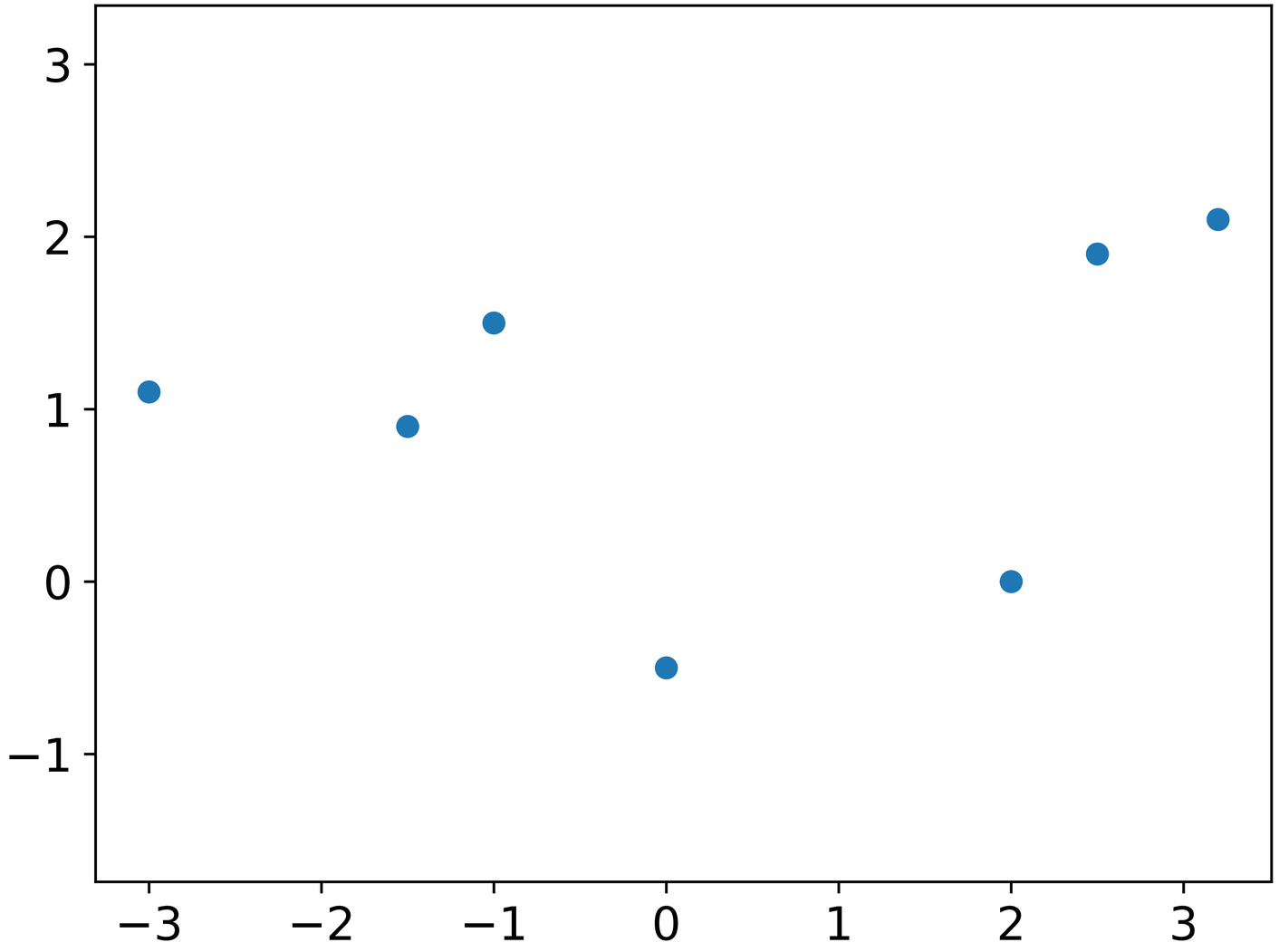
Suppose you're trying to predict whether a house is located in ***San Francisco*** or ***New York*** based on the following features:

- price
- sq ft
- price per sq ft
- year built
- elevation
- # bathrooms
- # bedrooms



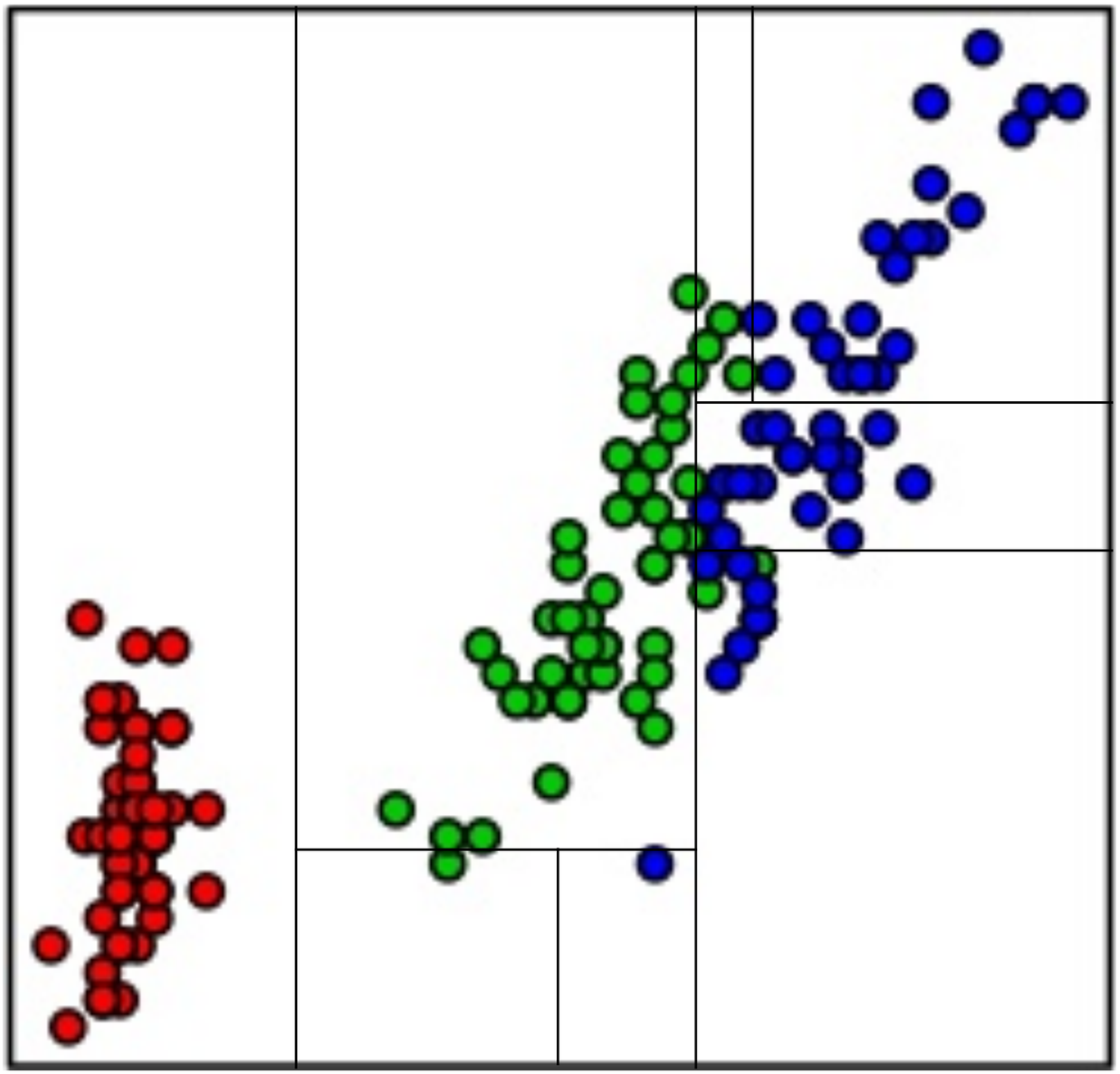
slide credit: Ameet Talwalkar

How to handle real outputs: regression tree

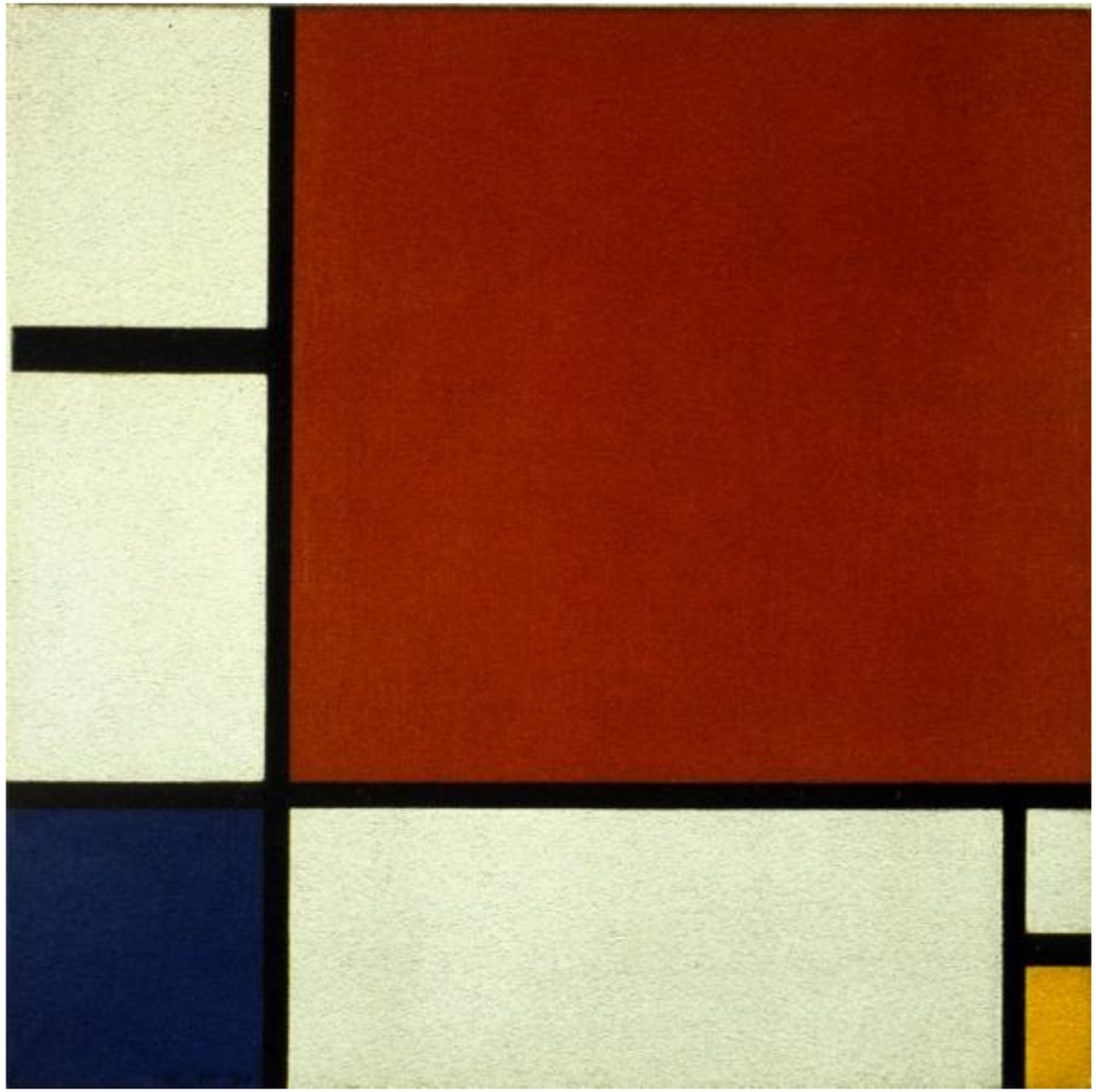


- In each leaf: predict mean
- Splitting criterion: prediction error $\sum_{\text{examples } i} (y_i - \hat{y}_i)^2$

Tree divides input space into nested regions



Decision tree as art



Composition II in Red, Blue, and Yellow
Piet Mondrian, 1930

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