CS-540: Intro to Artificial Intelligence

Decision Tree Learning Continued

Lecturer: Erin Winter

March 2017

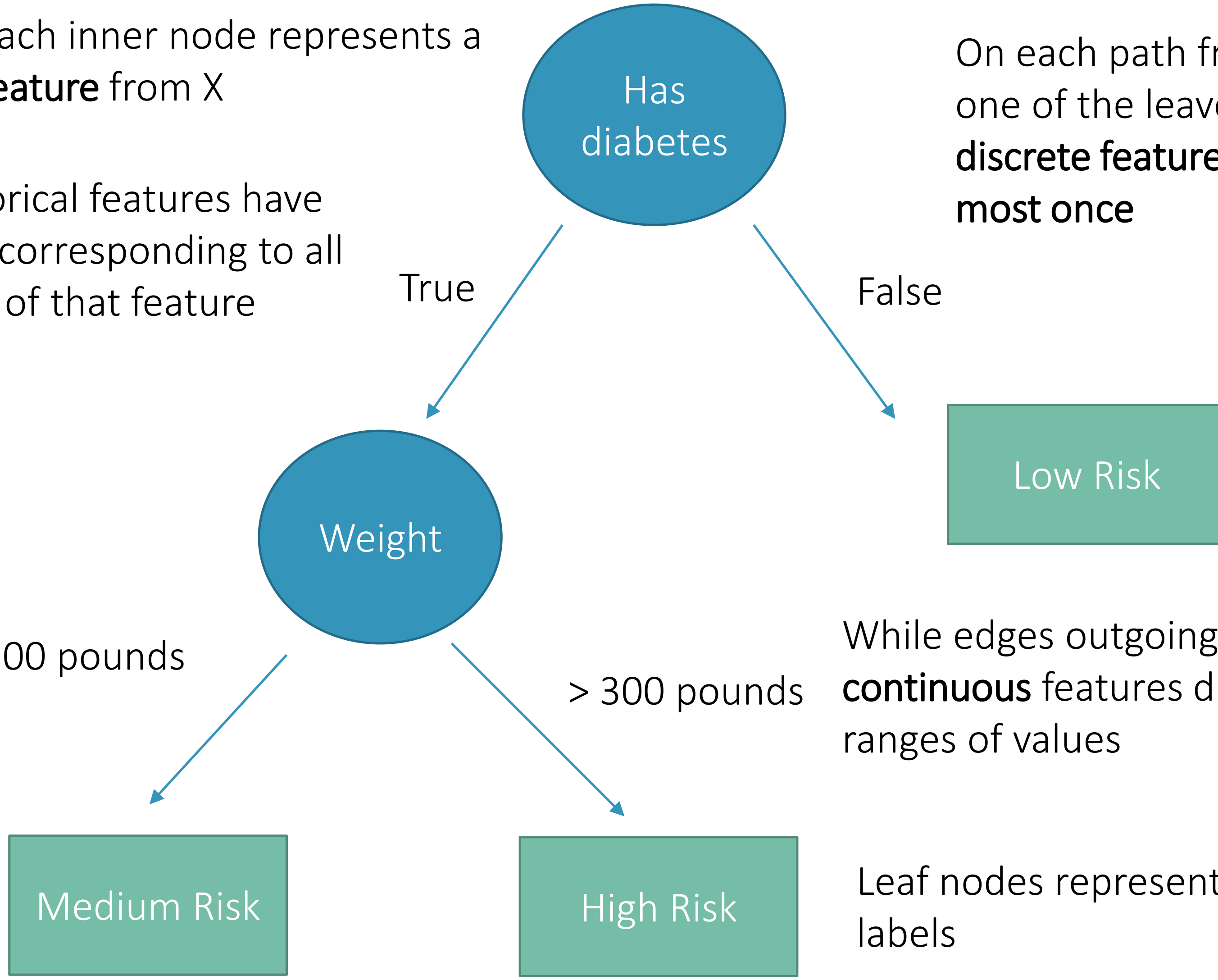
SUN	MON	TUE	WED	THU	FRI	SAT
			1	2	3	4
5	6	7	8	9	HW3 10	11
12	Exam 13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

Anatomy of a Decision Tree

Each inner node represents a **feature** from X

Categorical features have edges corresponding to all values of that feature

On each path from the root to one of the leaves, **each discrete feature appears at most once**



While edges outgoing from **continuous** features diverge on ranges of values

Leaf nodes represent class labels

Information gain, or mutual information

$$I(Y;X) = H(Y) - H(Y \mid X)$$

Information Gain Entropy Conditional Entropy

$$H(Y) = \quad H(Y) = \sum_{i=1}^{k} - p_i \log_2 p_i$$

Entropy

$$H(Y \mid X = v) = \sum_{i=1}^{k} -\Pr(Y = y_i \mid X = v) \log_2 \Pr(Y = y_i \mid X = v)$$

Specific Conditional Entropy

$$H(Y \mid X) = \sum_{v:\text{values of } X} \Pr(X = v) H(Y \mid X = v)$$

Conditional Entropy

Choose the attribute (i.e., feature or question) X that ***maximizes*** $I(Y; X)$

Decision-Tree Algorithm (ID3)

buildtree(*examples*, *attributes*, *default-label*)

if empty(*examples*) then return *default-label*

if (*examples* all have same label *y*) then return *y*

if empty(*attributes*) then return majority-class of *examples*

***q* = maxInfoGain(*examples, attributes*)**

tree = create-node with attribute *q*

foreach value *v* of attribute *q* do

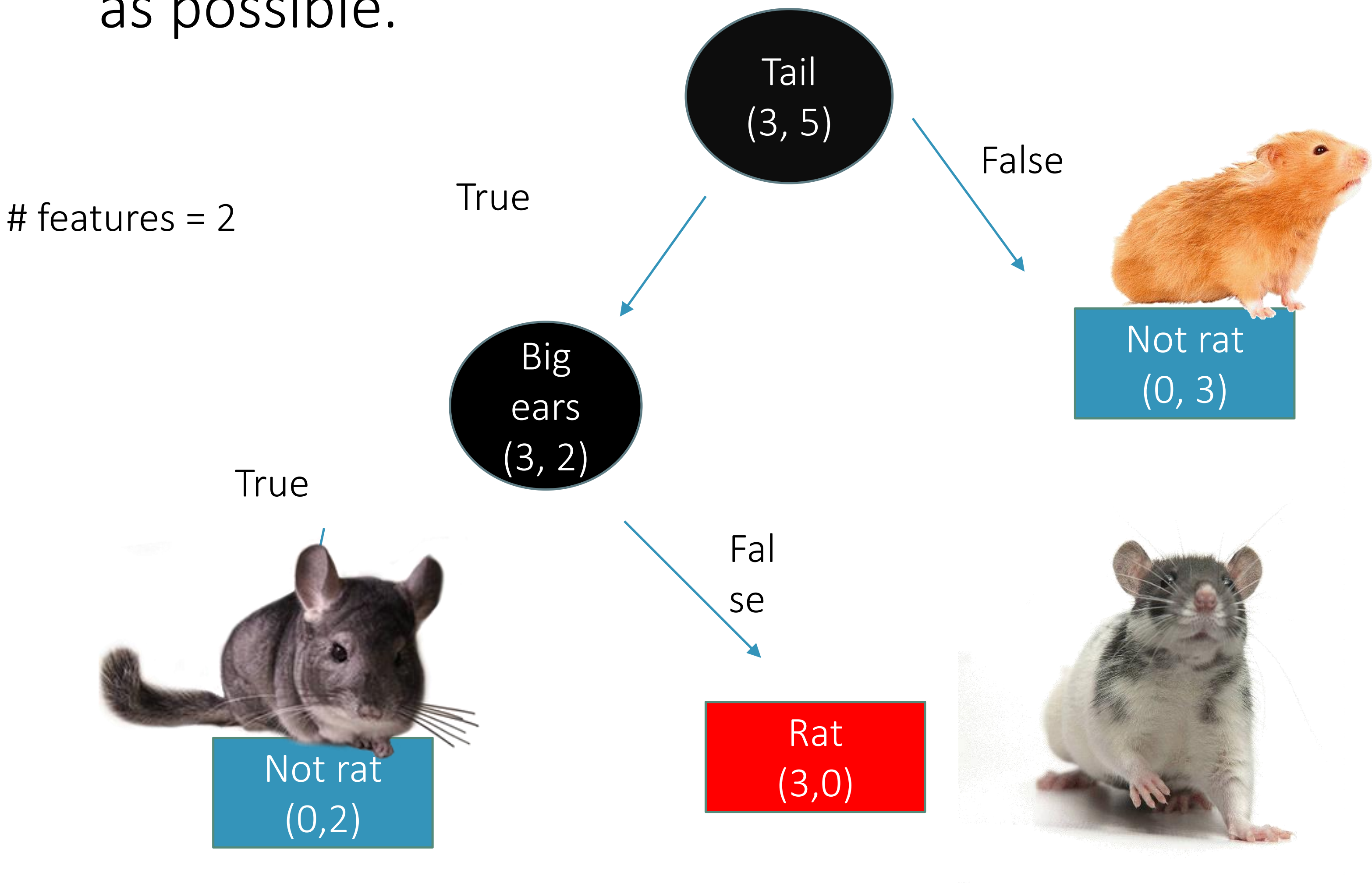
  v-ex = subset of examples with *q* == *v*

  subtree = **buildtree**(*v-ex*, *attributes* - {*q*}, *majority-class*(*examples*))

  add arc from *tree* to *subtree*

return *tree*

The goal of the information gain heuristic is to minimize entropy **of the examples at the leaves** while making the tree as simple (minimum depth) as possible.



Expressiveness of Decision Trees

Assume all attributes are Boolean and all classes are Boolean (i.e., 2 classes)

What is the class of Boolean functions that are possible to represent by decision trees?

Answer: All Boolean functions!

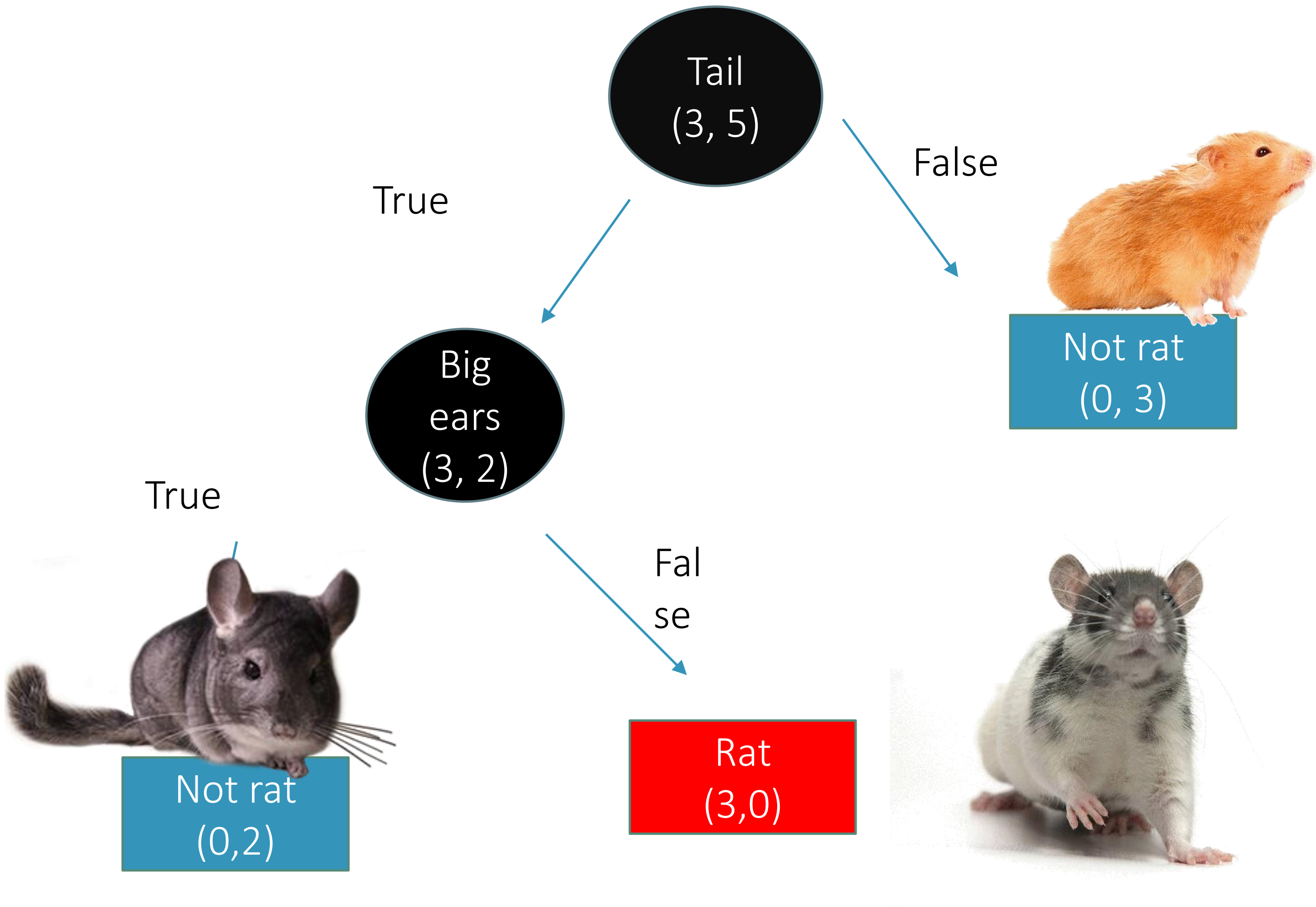
Proof:

1. Take any Boolean function
2. Convert it into a truth table
3. Construct a decision tree in which each row of the truth table corresponds to one path through the decision tree from root to a leaf

Tail = F -> Not rat

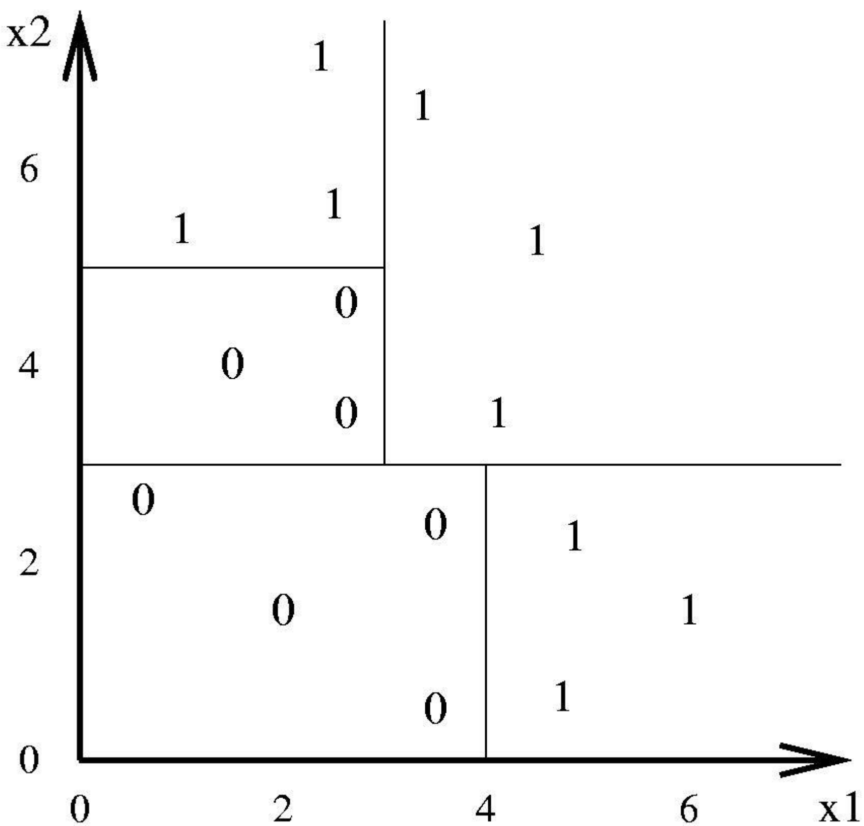
Tail = T ^ Big Ears = T -> Not rat

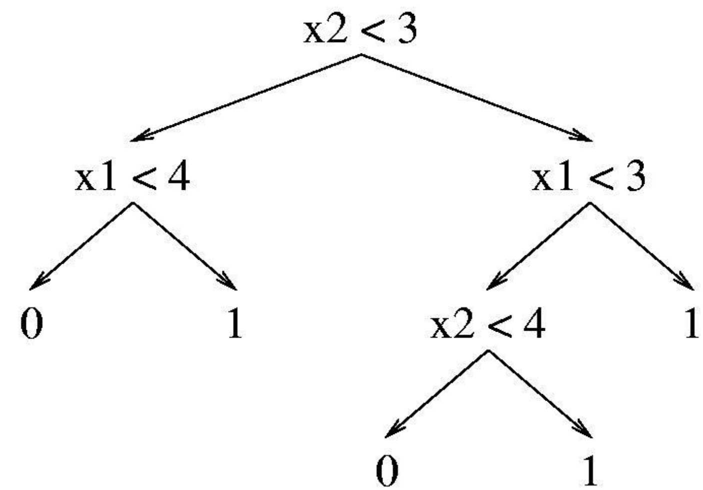
Tail = T ^ Big Ears = F -> Rat



Decision Tree Decision Boundaries

Decision trees divide the feature space into axis-parallel rectangles, and label each rectangle with one of the K classes.





Inductive Bias of Decision Trees

Determined by which algorithm is used, but tree-learning algorithms (ID3 and C4.5) are greedy and have the same preference bias.

Preference Bias: Tree-learning algorithms prefer **shallower trees** to deeper trees when searching the hypothesis space *H* for an *h* that classifies the training data and trees with higher information gain at the root.

Hypothesis Bias: Decision trees learn discrete finite-valued functions, but the entire search space is exponential. Not every possible tree is considered, **may converge to local optimum**.

**Because decision trees can learn any discrete finite-valued function, most consider decision trees to have only preference bias*

Downsides of Max Information Gain as a Heuristic

Information gain is biased **towards tests with many outcomes**

e.g. consider a feature that uniquely identifies each training instance – splitting on this feature would result in many branches, each of which is "pure" (has instances of only one class) – maximal information gain!

This may not lead to the best decision tree

Continuous Features in Decision Trees

What if some (or all) of the features, x_1, x_2, ..., x_k, are continuous?

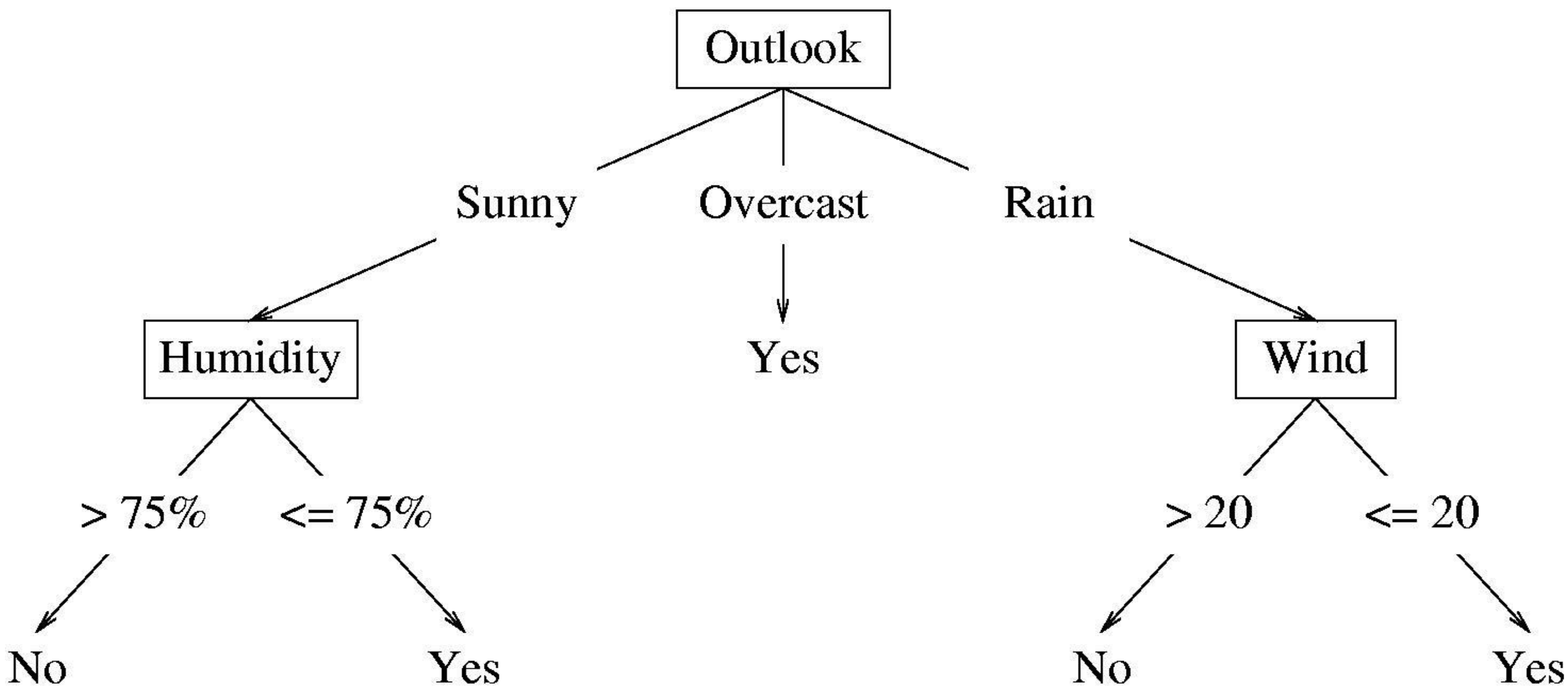
Example: x_1 = height (in inches)

Use nodes of the form: $\boldsymbol{x_1} > \boldsymbol{t_1}$? where t_1 is a threshold

- How many thresholds?
  - *Between examples that are classified differently*
- How to set threshold values?
  - *Midpoint between 2 consecutive examples' values*

Decision Tree Hypothesis Space

the features are continuous, internal nodes may test the value of a feature against a threshol



Computing candidate thresholds

Sort examples by increasing feature value for continuous feature of interest

Find pairs of *consecutive* examples that have *different* class labels, and define a candidate threshold as the average of these two examples' feature values

	X1	X2	Class
x2	2.2	1.5	T
x4	2.1	1.9	T
x3	3.9	3.5	F
x5	1.1	4	T
x1	0.5	4.5	F

$$\text{Candidate Thresholds} = \{\frac{1.9+3.5}{2} = 2.7, \frac{3.5+4}{2} = 3.75, \frac{4+4.5}{2} = 4.25\}$$

Learning Continuous Features

buildtree(*examples*, *attributes*, *default-label*)

if empty(*examples*) then return *default-label*

if (*examples* all have same label *y*) then return *y*

if empty(*attributes*) then return majority-class of *examples*

q = maxInfoGain(*examples, attributes*) compute candidate splits for continuous attributes + find info gain of **best** candidate threshold, only consider best threshold when comparing info gain to that of other feature splits

tree = create-node with attribute *q*

foreach value *v* of attribute *q* do

  v-ex = subset of examples with *q* == *v*

  subtree = **buildtree**(*v-ex*, *attributes* - {*q*}, *majority-class*(*examples*))

  add arc from *tree* to *subtree*

return *tree*

Evaluating Performance

How might the performance be evaluated?

Predictive accuracy of classifier

Speed of learner

Speed of classifier

*Space requirements**

These are project specific implementation details and less of a problem when a model is optimized.

Extensions to Decision Tree Learning: **Missing Data**

Ex: d = [height, weight, blood type]

y_i = [unknown, 300, A+]

During learning: replace with ***most likely*** *value* or use ***Unknown*** as a value

During classification: follow arcs for *all values* and weight each by the frequency of the examples along that arc

Training Set Error

For each example in the training set, use the decision tree to see what class it predicts

What % of the examples does the decision tree's prediction *disagree* with the known *true* value?

This quantity is called the *Training Set Error*

The smaller the better

But why are we doing learning anyway?

More important to assess how well the decision tree predicts output for *future* data

Error = incorrect/(correct + incorrect)
Accuracy = correct/(correct + incorrect)

Test Set Error

Partition labeled data into training and test sets, learn model on the training set

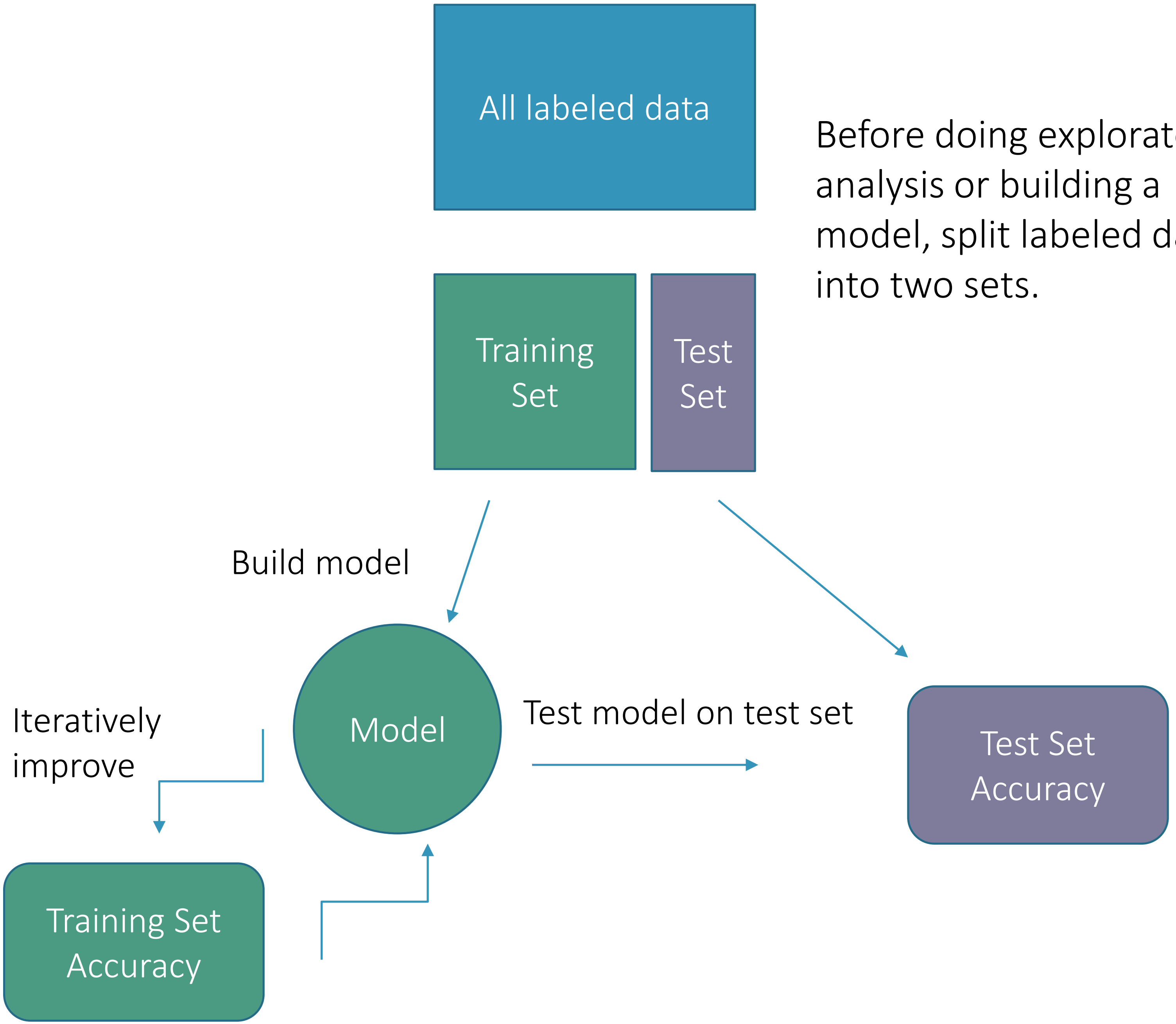
But once learned, we see how well the tree predicts that data: % classified incorrectly

This is a good simulation of what happens when we try to predict future data

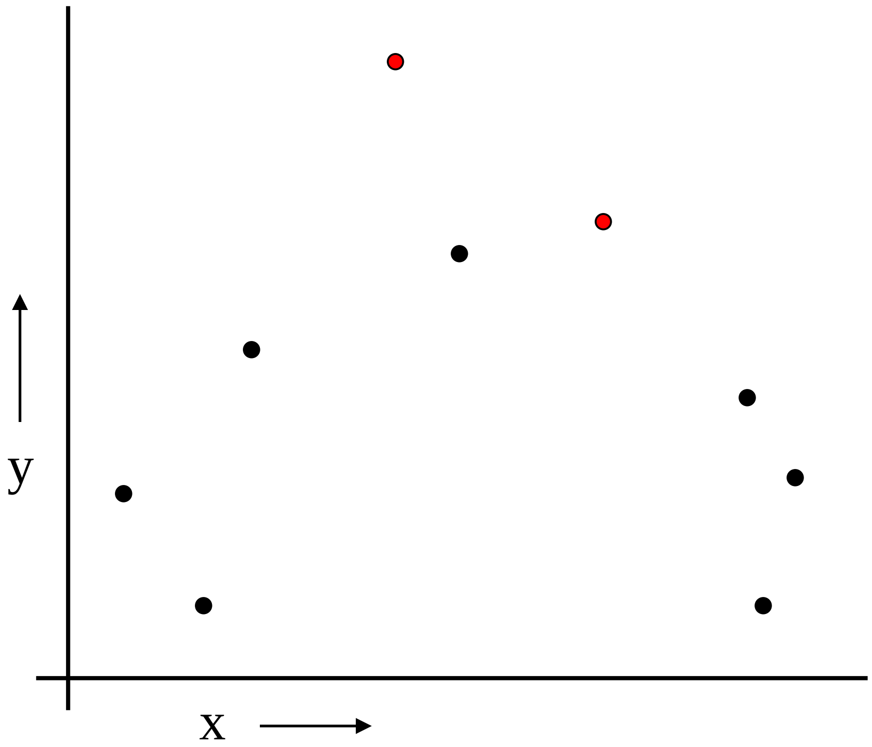
Called the *Test Set Error*

Ultimately, a **model is only as good as its ability to generalize to data outside of the training set**

Before doing exploratory analysis or building a model, split labeled data into two sets.



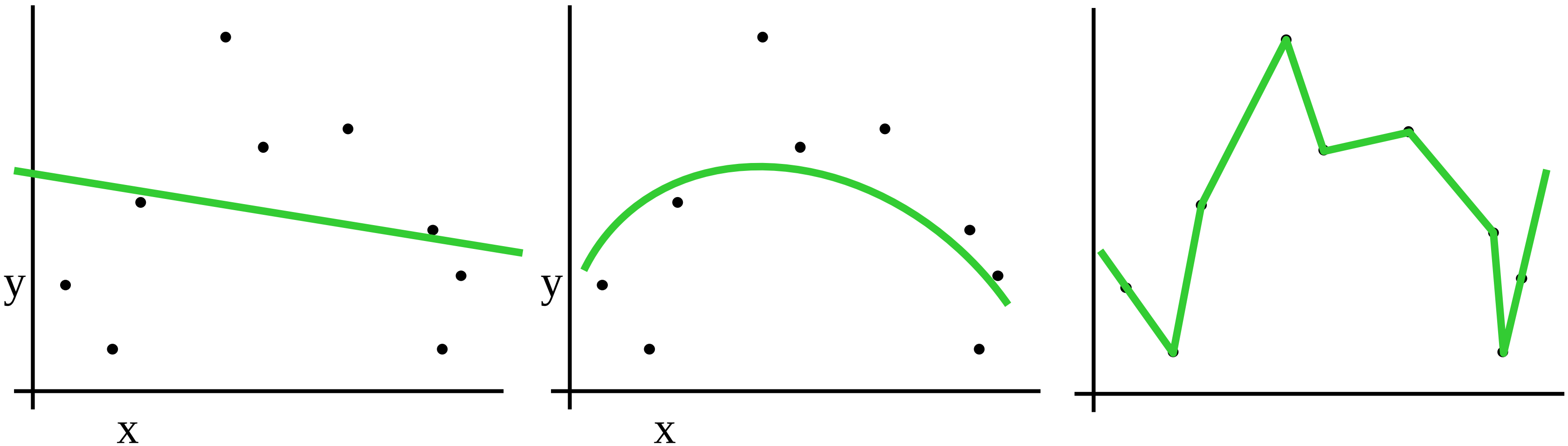
A Regression Problem



$y = f(x)$ + noise

Can we learn f from this data?

Which is Best?



Linear f Quadratic f Piecewise Linear f

An algorithm that **maximizes fit** by **minimizing error over training data** would learn a model that fits the noise in the data.

Regression: Polynomial Fit

- The degree *d* (complexity of the model) is important

$$f(x) = c_d x^d + c_{d-1} x^{d-1} + \cdots + c_1 x + c_0$$

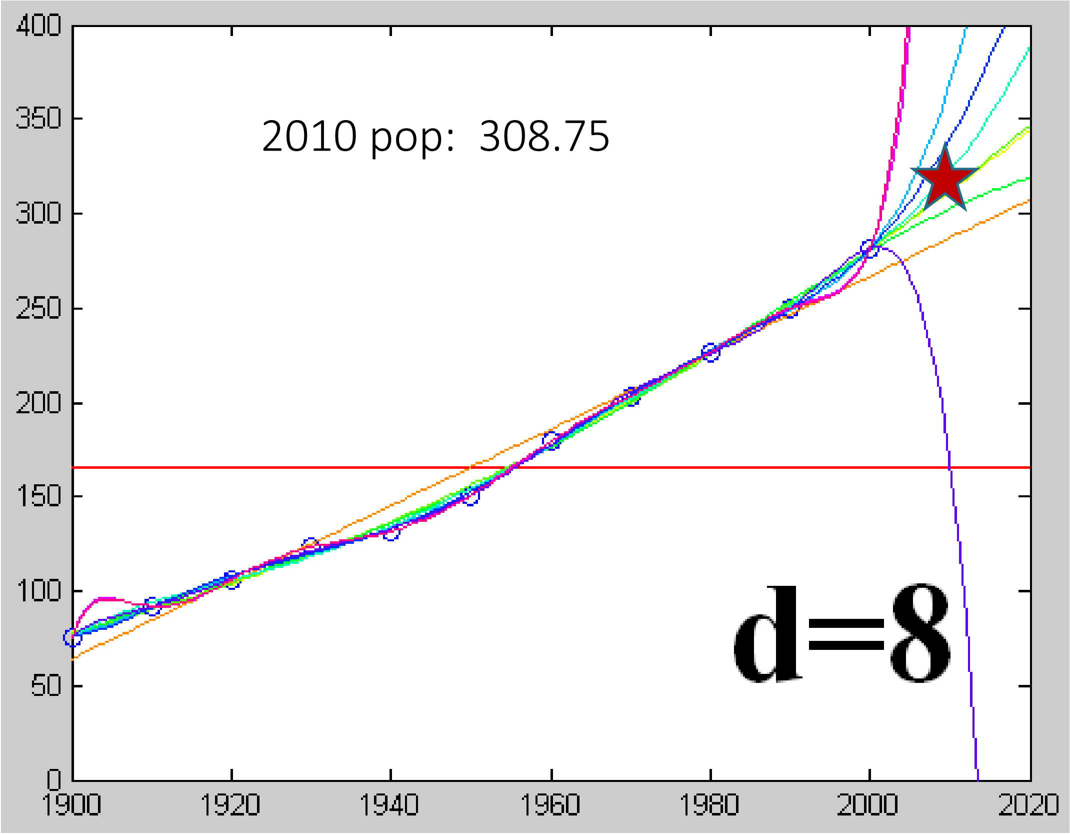
- Fit (= learn) coefficients c_d, … c_0 to minimize Mean Squared Error (MSE) on training data

$$MSE = \frac{1}{n} \sum_{i=1}^{n} \Big(\underbrace{y_i}_{\text{label}} - \underbrace{f(x_i)}_{\text{prediction}} \Big)^2$$

Overfitting

As *d* increases, MSE on *training* data improves, but **prediction** on *test* data **worsens**

degree=0 MSE=4181.451643
degree=1 MSE=79.600506
degree=2 MSE=9.346899
degree=3 MSE=9.289570
degree=4 MSE=7.420147
degree=5 MSE=5.310130
degree=6 MSE=2.493168
degree=7 MSE=2.278311
degree=8 MSE=1.257978
degree=9 MSE=0.001433
degree=10 MSE=0.000000



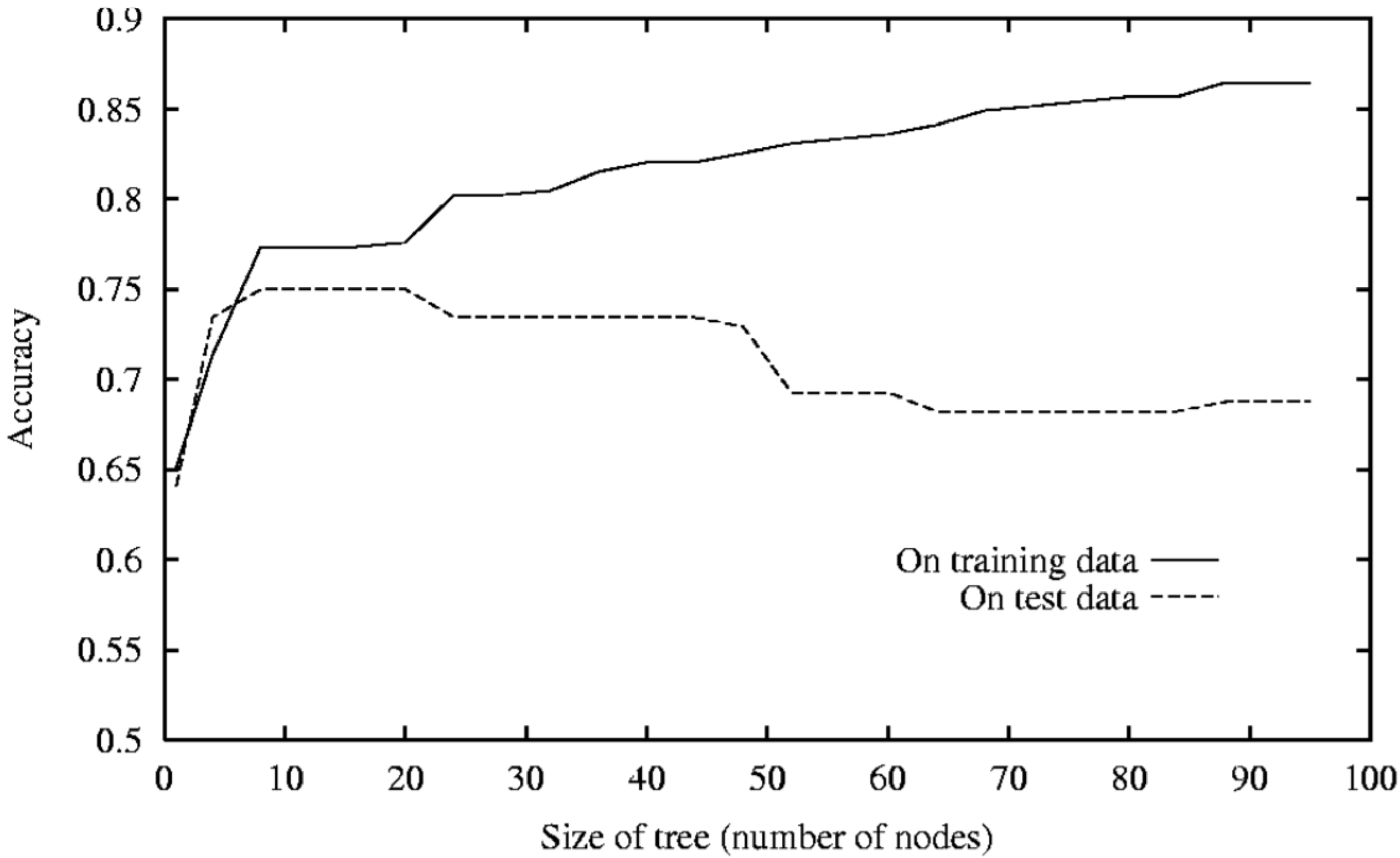
Overfitting a Decision Tree

In general, **overfitting means finding “meaningless” regularity in data**

Noisy Data: "noise" could be in the examples:

- examples have the same attribute values, but different classifications
- classification is wrong
- attribute values are incorrect because of errors getting or preprocessing the data
- irrelevant attributes

Decision Tree Overfitting



Overfitting a Decision Tree

Five inputs, all binary, are generated in all 32 possible combinations

Output $\boldsymbol{y}$ = copy of $\boldsymbol{e}$, except a random 25% of the records have $\boldsymbol{y}$ set to the *opposite* of $\boldsymbol{e}$

32 records

a	***b***	***c***	***d***	***e***	***y***
0	0	0	0	0	0
0	0	0	0	1	0
0	0	0	1	0	0
0	0	0	1	1	1
0	0	1	0	0	1
:	:	:	:	:	:
1	1	1	1	1	1

Training set

Overfitting a Decision Tree

The **test set** is constructed similarly

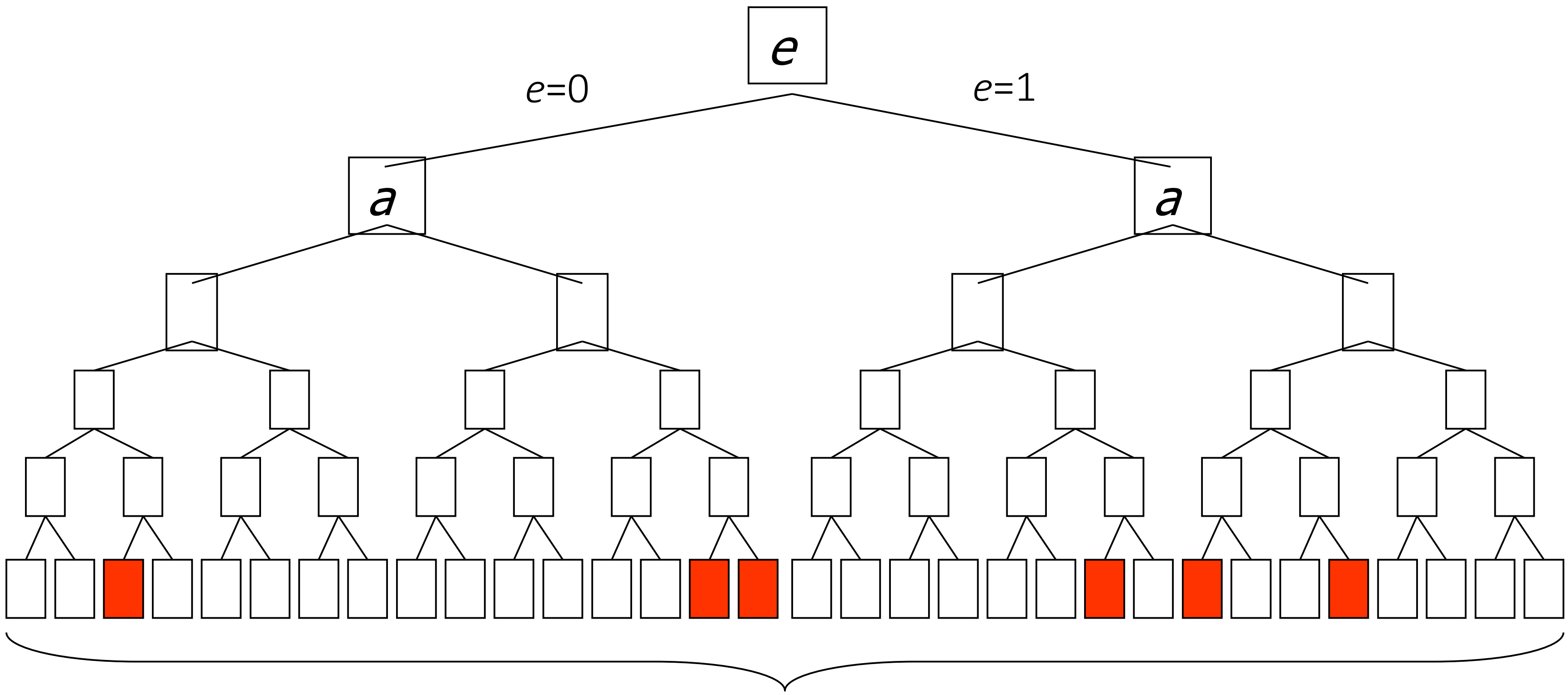
- $\boldsymbol{y} = \boldsymbol{e}$, but 25% the time we corrupt it by $\boldsymbol{y} = \mathbf{1} - \boldsymbol{e}$
- Assume the corruption in training and test sets are *independent*

The training and test sets are the same, except

- Some *y*'s are corrupted in training, but not in test
- Some *y*'s are corrupted in test, but not in training

Overfitting a Decision Tree

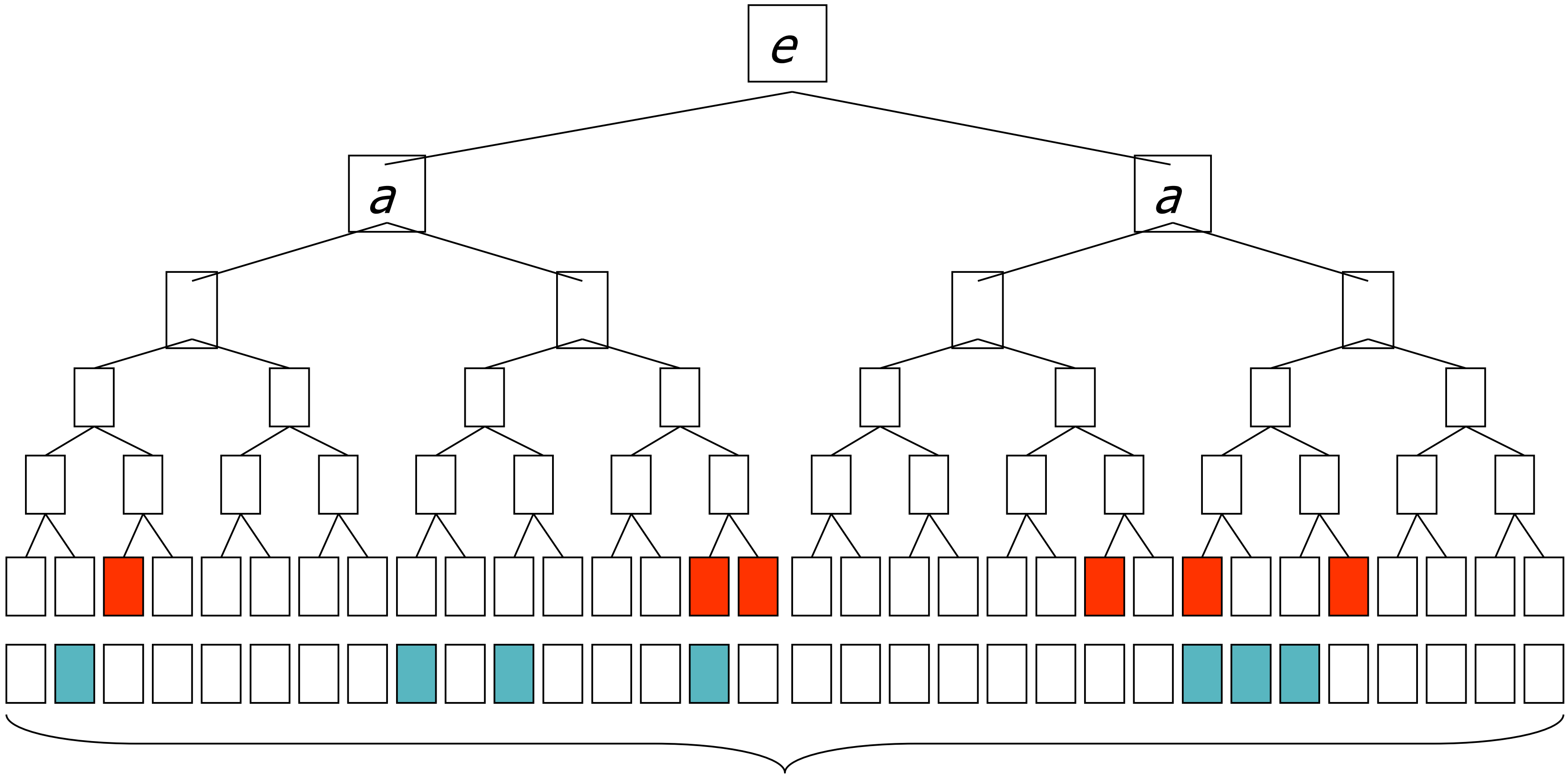
Suppose we build a full tree on the **training set**



Training set accuracy = 100% (all leaf nodes contain exactly 1 example)
25% of these training leaf node labels will be corrupted ($\neq$ ***e***)

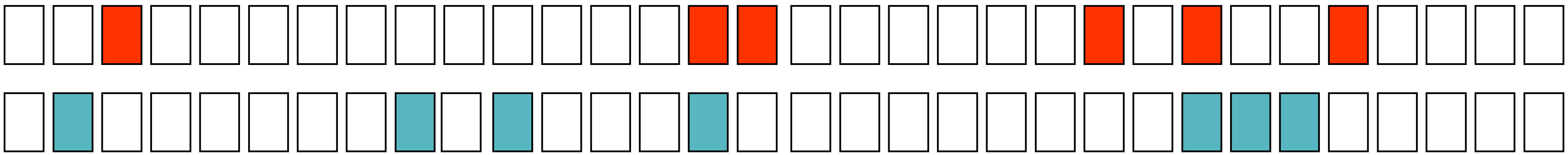
Overfitting a Decision Tree

Next, classify the test data with the tree



25% of the test examples are corrupted – independent of training data

Overfitting a Decision Tree



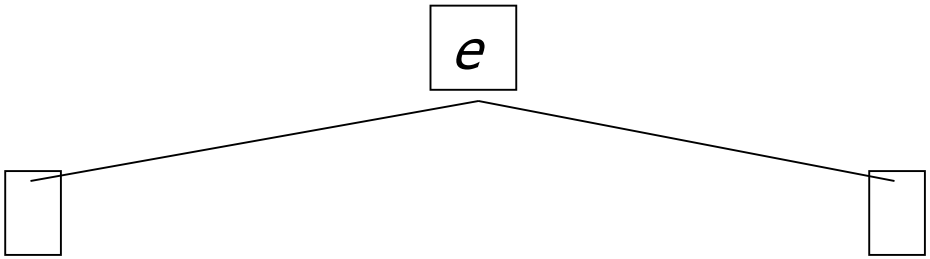
On average:

- ¾ training data *uncorrupted*
  - ¾ of these are uncorrupted in test – correct labels
  - ¼ of these are corrupted in test – wrong
- ¼ training data *corrupted*
  - ¾ of these are uncorrupted in test – wrong
  - ¼ of these are also corrupted in test – correct labels

Test accuracy = (¾ * ¾) + (¼ * ¼) = 5/8 = **62.5%**

Overfitting a Decision Tree

The tree would be:



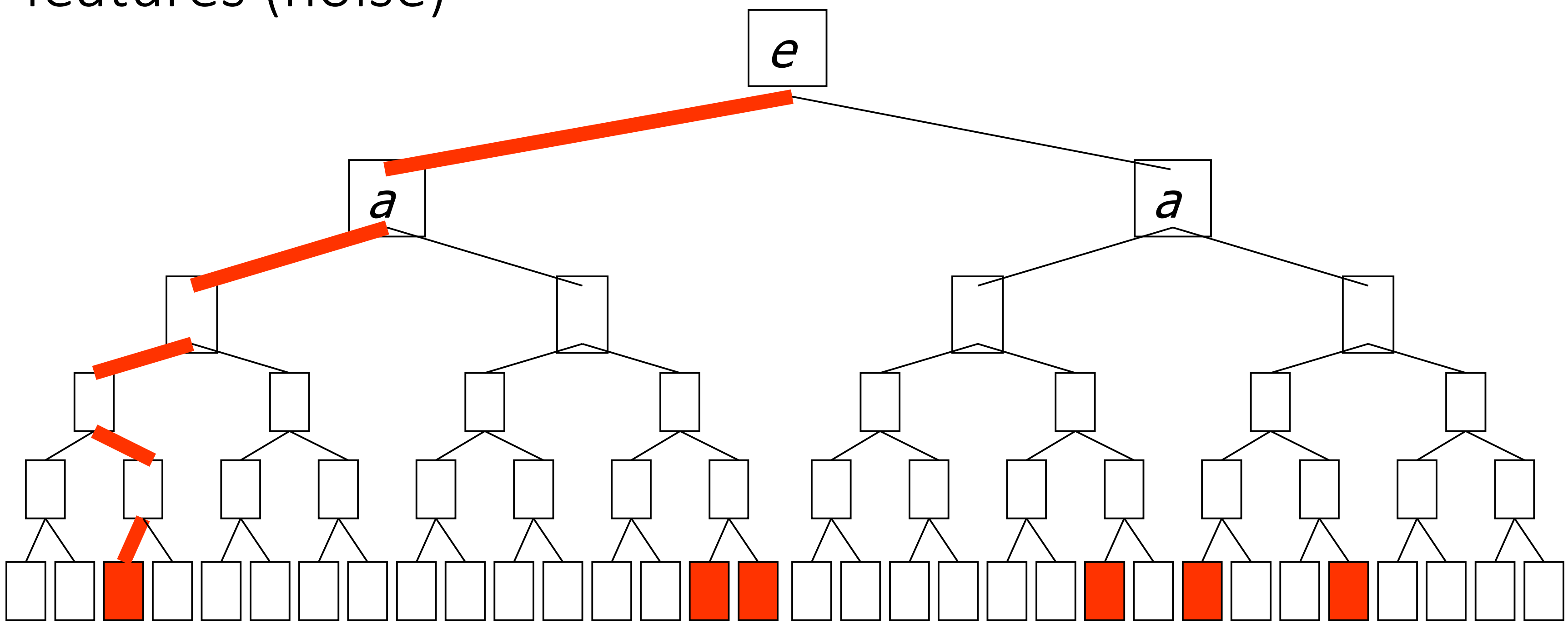
In training data, about ¾ ***y***'s are 0 here. Majority vote predicts ***y*** = 0

In training data, about ¾ ***y***'s are 1 here. Majority vote predicts ***y*** = 1

In test data, ¼ ***y***'s are different from ***e*** because they were corrupted, and ¾ ***y***'s will be correct, so **test set accuracy = 75%**, which is *better* than when using more (meaningless) attributes (= 62.5%)

Overfitting a Decision Tree

Hence, the full tree ***overfit*** by learning meaningless features (noise)



Can we recognize irrelevant features without knowing the underlying f?

Avoiding Overfitting: **Pruning**

In overfitted trees, *irrelevant features* confound the true distinguishing features

Pruning with a ***Tuning Set***

1. Randomly split the **training data** into TRAIN and TUNE, say 70% and 30%
2. Build a full tree using *only* the TRAIN set
3. Prune the tree using the TUNE set