CSE419 – Artificial Intelligence and Machine Learning 2018

PhD Furkan Gözükara, Toros University

https://github.com/FurkanGozukara/CSE419_2018

Lecture 3

Decision Trees

Based on Asst. Prof. Dr. David Kauchak (Pomona College) Lecture Slides

Representing examples

examples

What is an example?
How is it represented?

Features

examples

features

$f_1, f_2, f_3, \ldots, f_n$

$f_1, f_2, f_3, \ldots, f_n$

$f_1, f_2, f_3, \ldots, f_n$

$f_1, f_2, f_3, \ldots, f_n$

How our algorithms actually “view” the data

Features are the questions we can ask about the examples

Features

examples

features

red, round, leaf, 3oz, …

green, round, no leaf, 4oz, …

yellow, curved, no leaf, 4oz, …

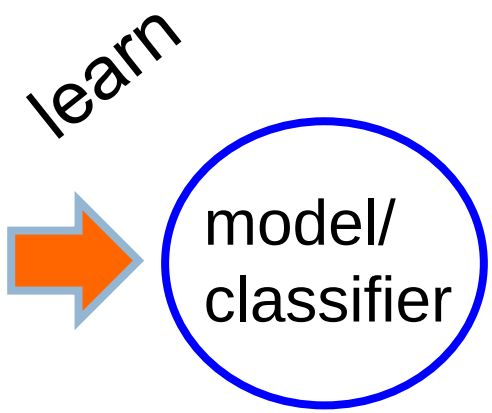
green, curved, no leaf, 5oz, …

How our algorithms actually "view" the data

Features are the questions we can ask about the examples

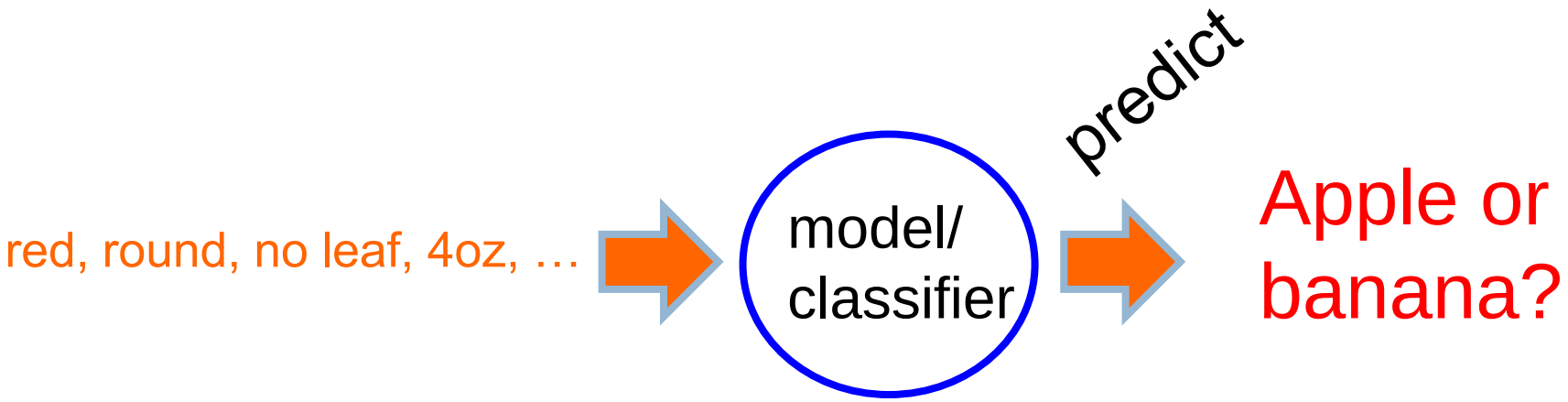
Classification revisited

examples	label
red, round, leaf, 3oz, …	apple
green, round, no leaf, 4oz, …	apple
yellow, curved, no leaf, 4oz, …	banana
green, curved, no leaf, 5oz, …	banana



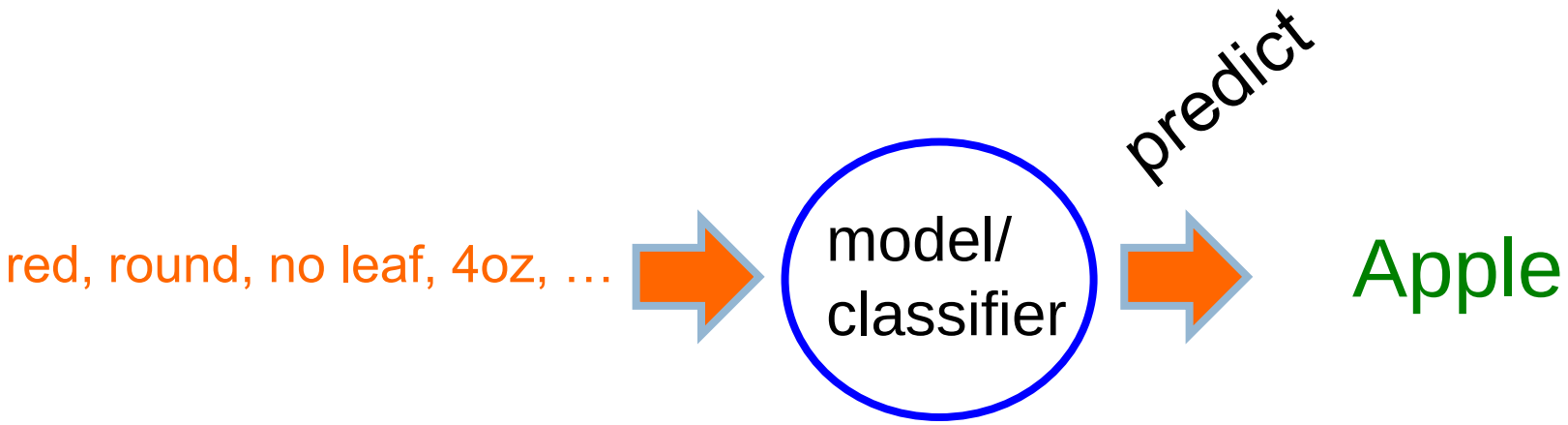
During learning/training/induction, learn a model of what distinguishes apples and bananas *based on the features*

Classification revisited



The model can then classify a new example *based on the features*

Classification revisited



Why?

The model can then classify a new example *based on the features*

Classification revisited

Training data		Test set
examples	label	
red, round, leaf, 3oz, …	apple	
green, round, no leaf, 4oz, …	apple	red, round, no leaf, 4oz, … ?
yellow, curved, no leaf, 4oz, …	banana	
green, curved, no leaf, 5oz, …	banana	

Classification revisited

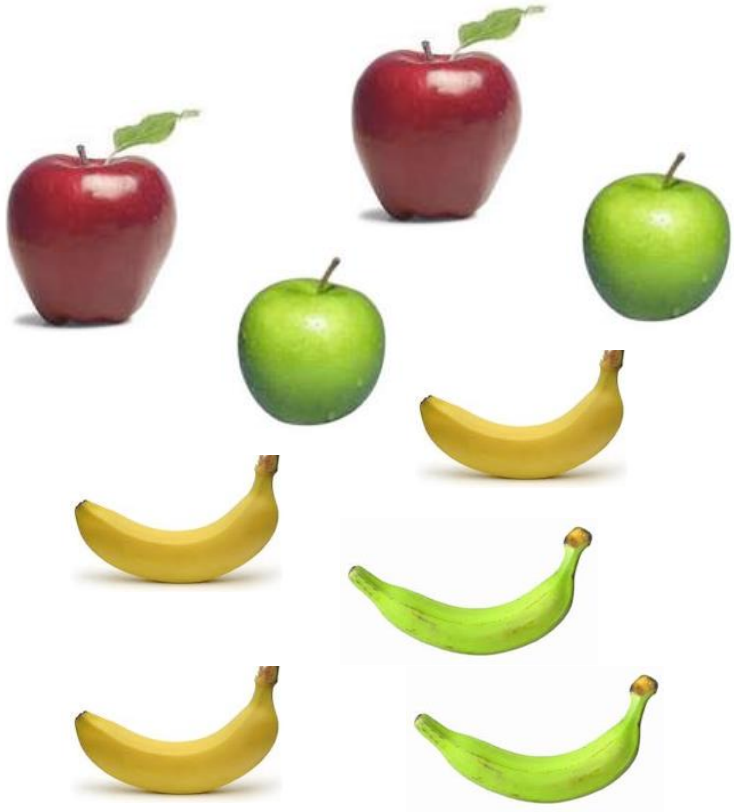
Training data		Test set
examples	label	
red, round, leaf, 3oz, …	apple	
green, round, no leaf, 4oz, …	apple	red, round, no leaf, 4oz, …?
yellow, curved, no leaf, 4oz, …	banana	
green, curved, no leaf, 5oz, …	banana	

Learning is about ***generalizing*** from the training data

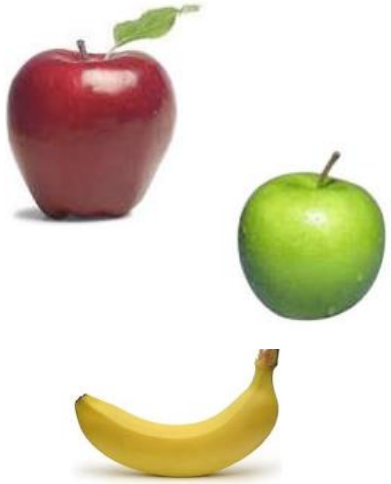
What does this assume about the training and test set?

Past predicts future

Training data

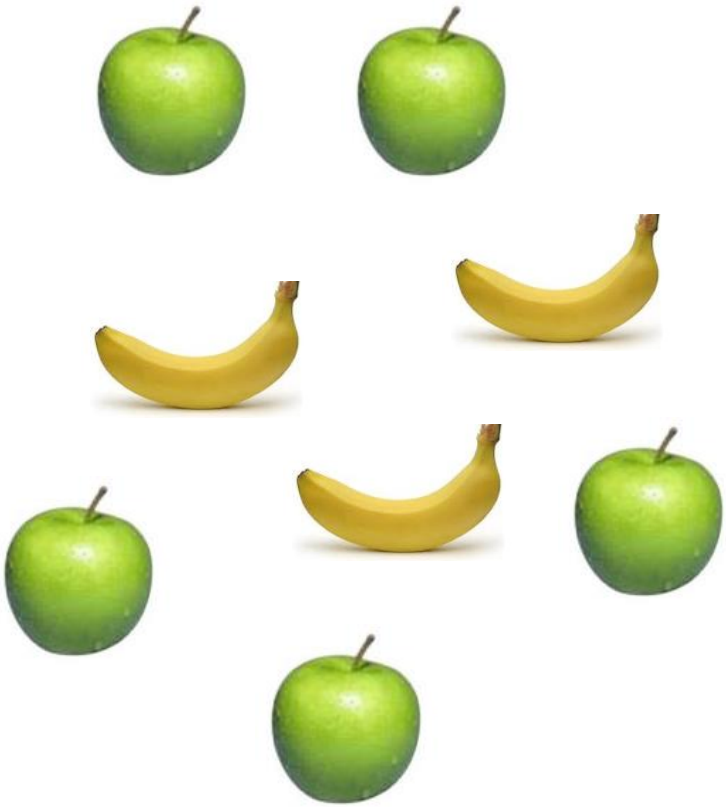


Test set

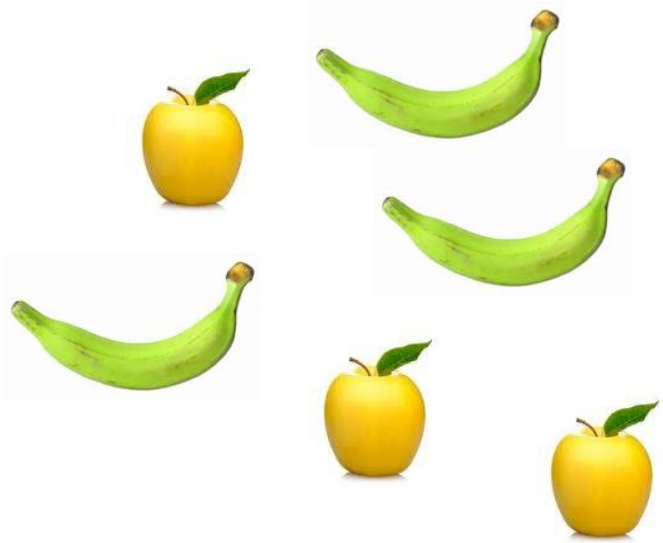


Past predicts future

Training data



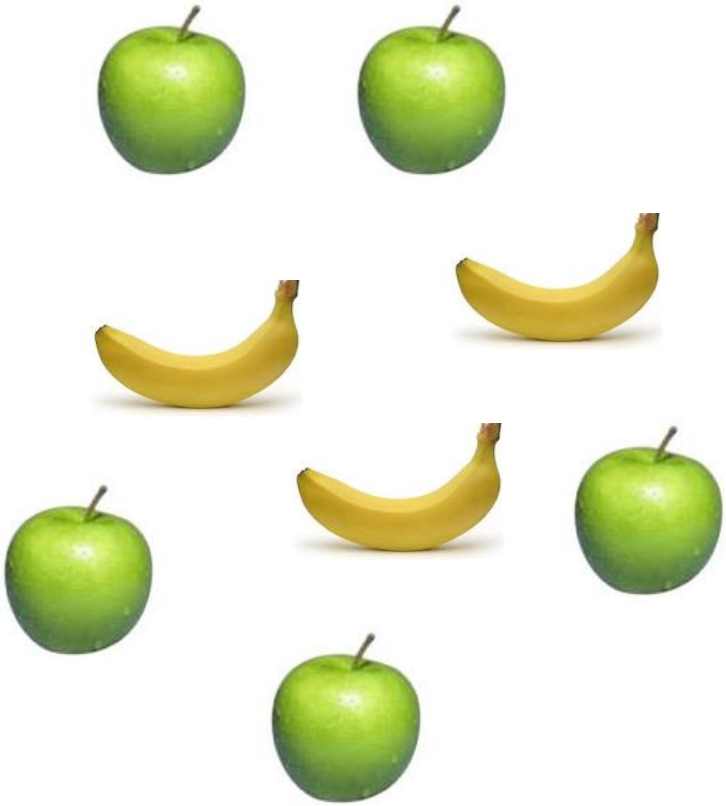
Test set



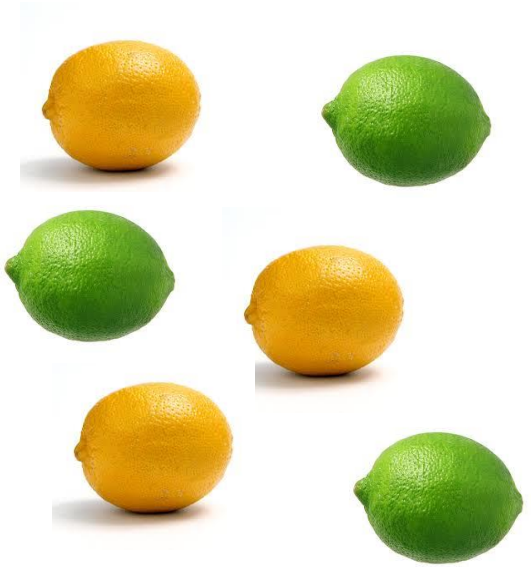
Not always the case, but we'll often assume it is!

Past predicts future

Training data



Test set



Not always the case, but we'll often assume it is!

More technically…

We are going to use the *probabilistic model* of learning

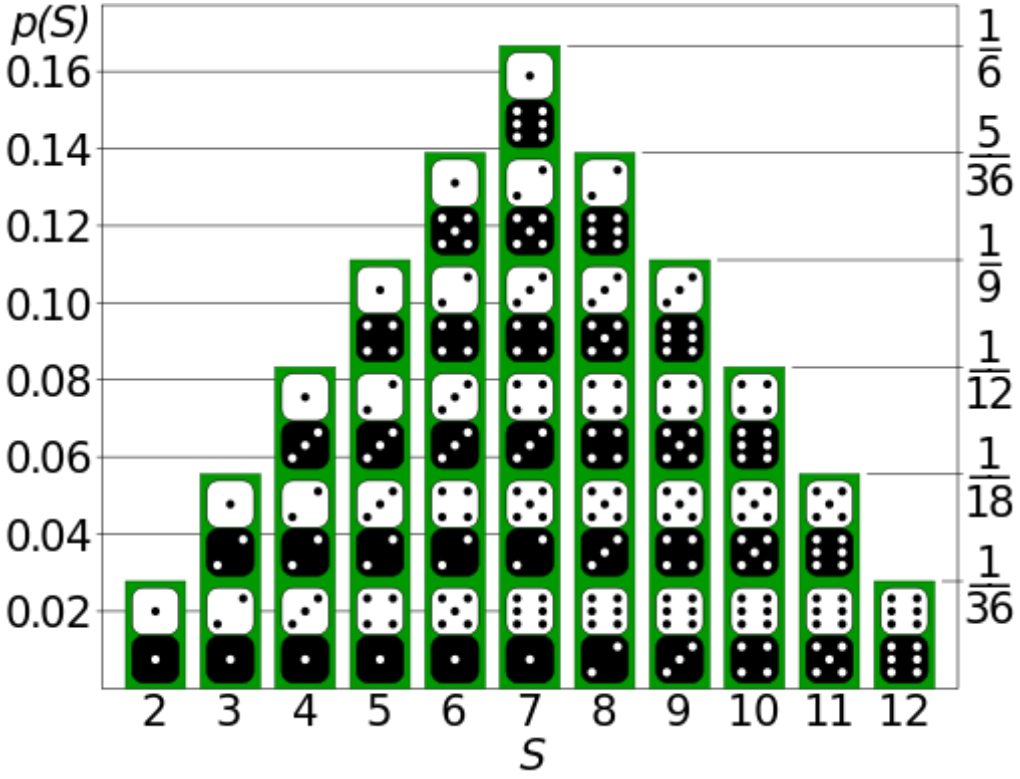
There is some probability distribution over example/label pairs called the *data generating distribution*

Both the training data **and** the test set are generated based on this distribution

What is a probability distribution?

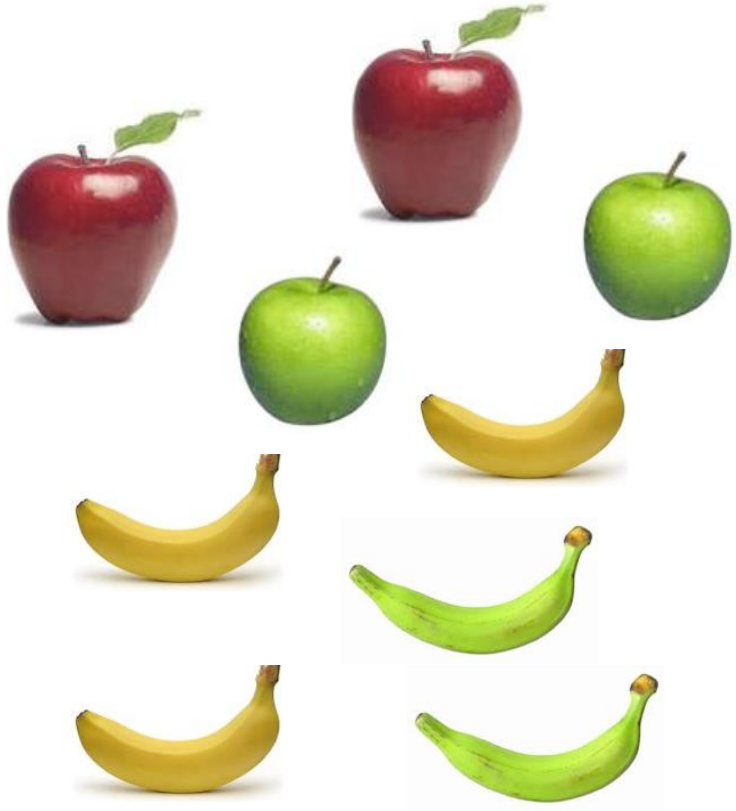
Probability distribution

Describes how likely (i.e. probable) certain events are



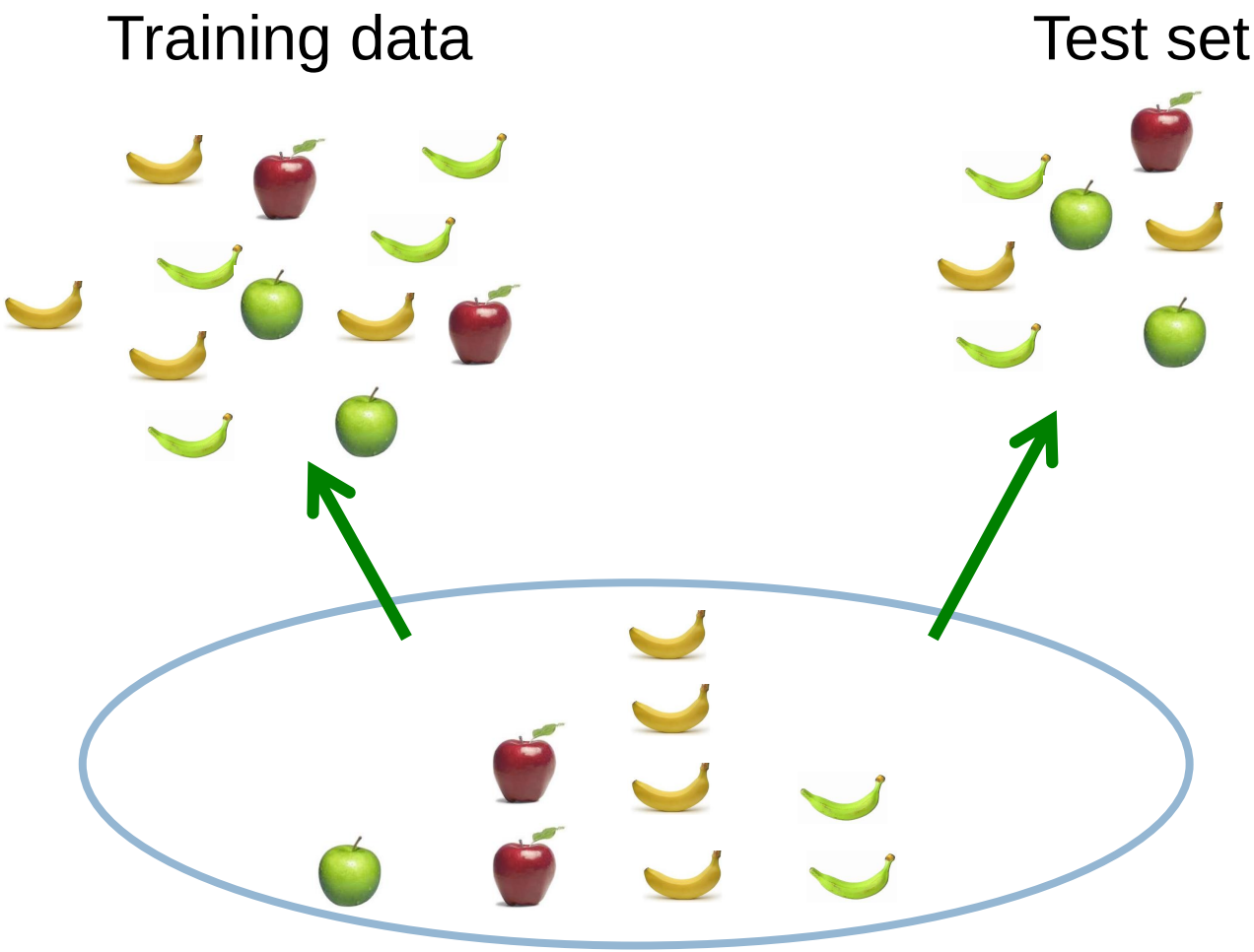
Probability distribution

Training data

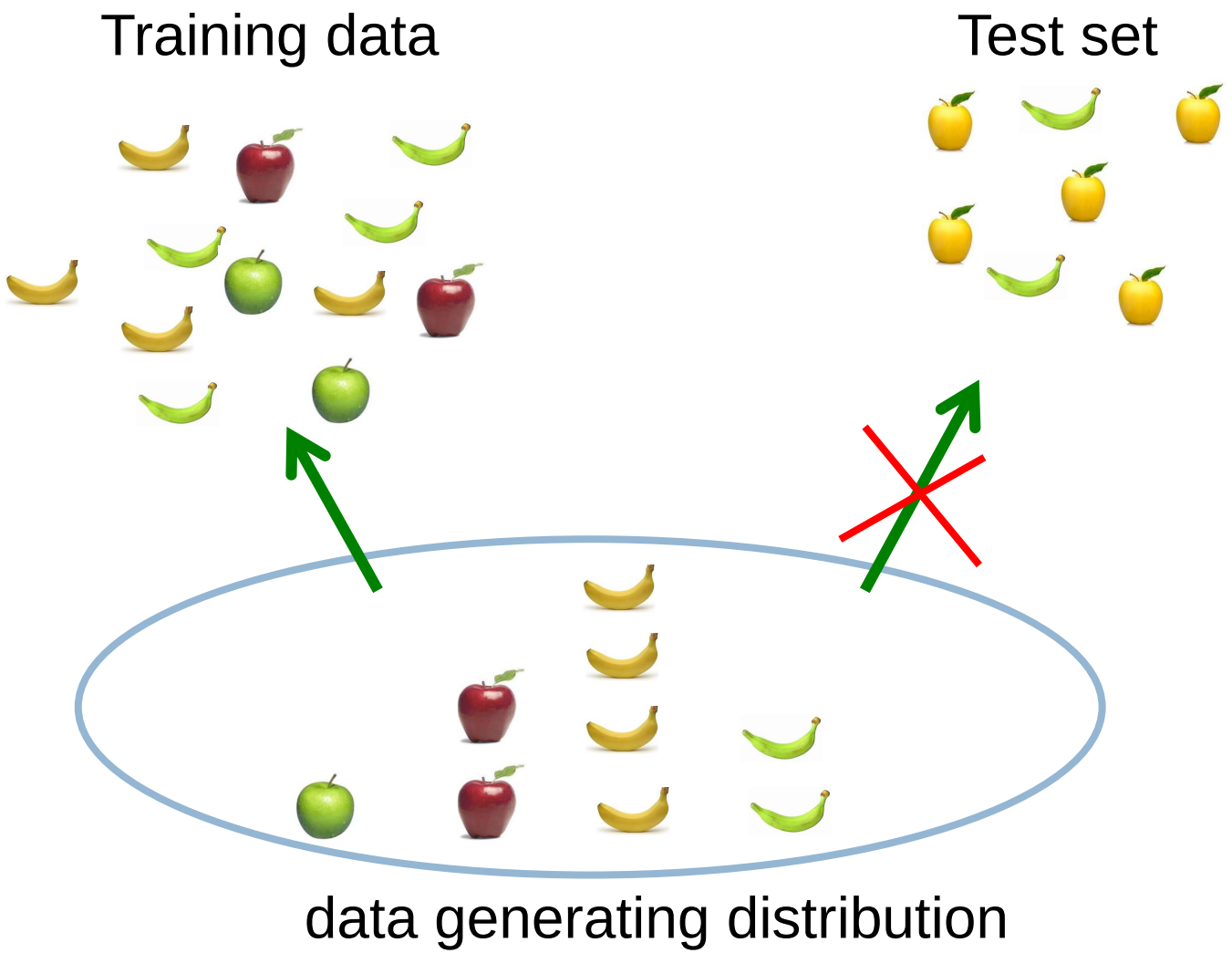


High probability	Low probability
round apples	curved apples
curved bananas	red bananas
apples with leaves	yellow apples
…	…

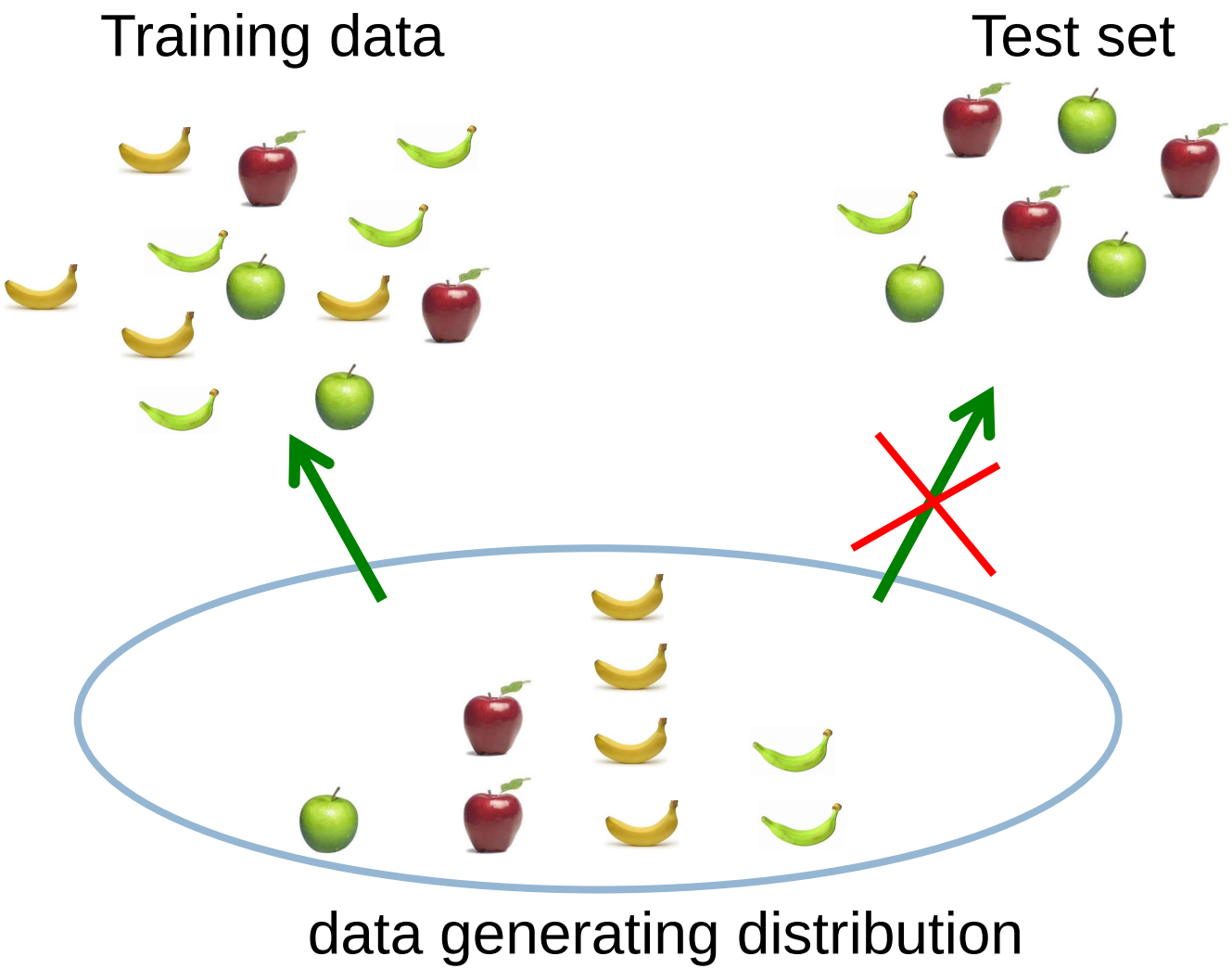
data generating distribution



data generating distribution



data generating distribution



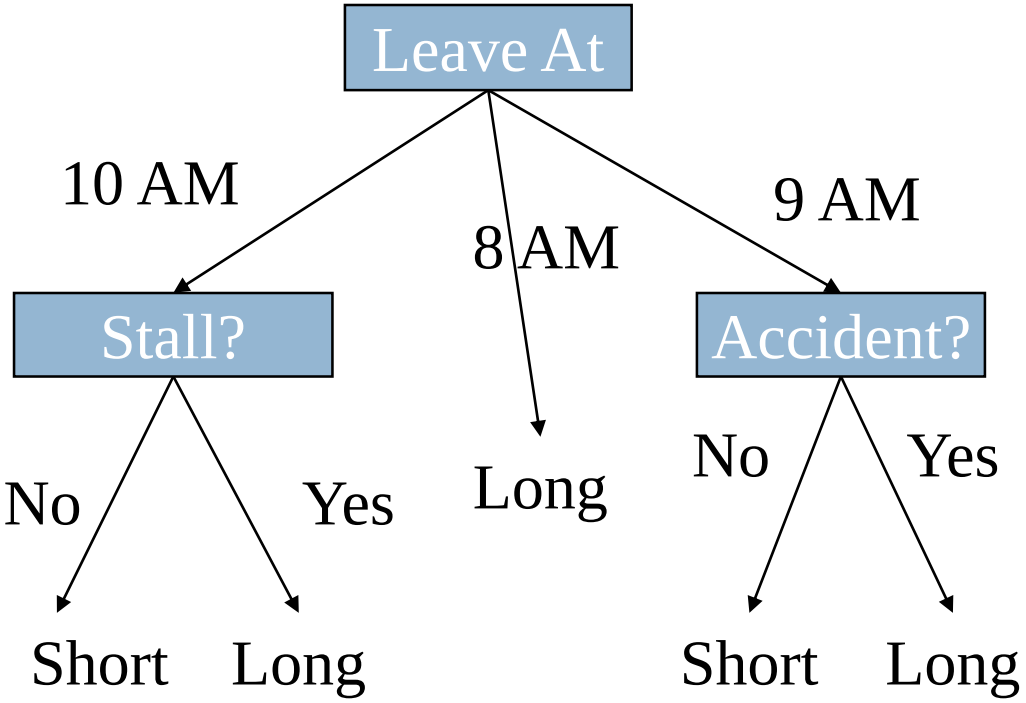
A sample data set

Features				**Label**
Hour	Weather	Accident	Stall	Commute
8 AM	Sunny	No	No	Long
8 AM	Cloudy	No	Yes	Long
10 AM	Sunny	No	No	Short
9 AM	Rainy	Yes	No	Long
9 AM	Sunny	Yes	Yes	Long
10 AM	Sunny	No	No	Short
10 AM	Cloudy	No	No	Short
9 AM	Sunny	Yes	No	Long
10 AM	Cloudy	Yes	Yes	Long
10 AM	Rainy	No	No	Short
8 AM	Cloudy	Yes	No	Long
9 AM	Rainy	No	No	Short

8 AM, Rainy, Yes, No?
10 AM, Rainy, No, No?

Can you describe a “model” that could be used to make decisions in general?

Decision trees

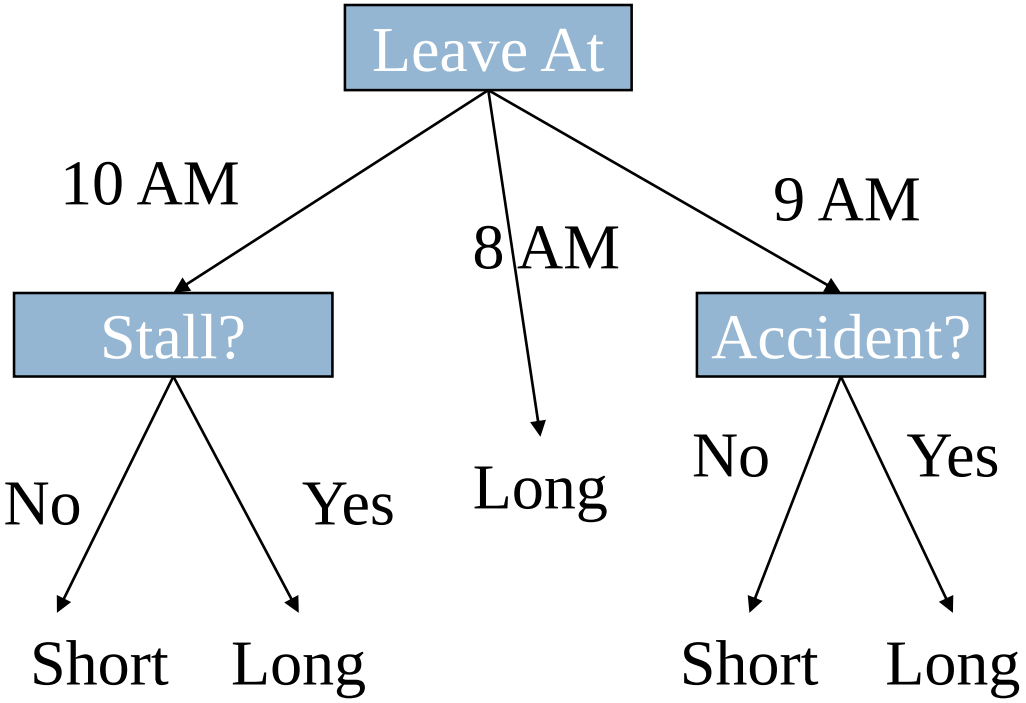


Tree with internal nodes labeled by features

Branches are labeled by tests on that feature

Leaves labeled with classes

Decision trees



Leave = 8 AM
Weather = Rainy

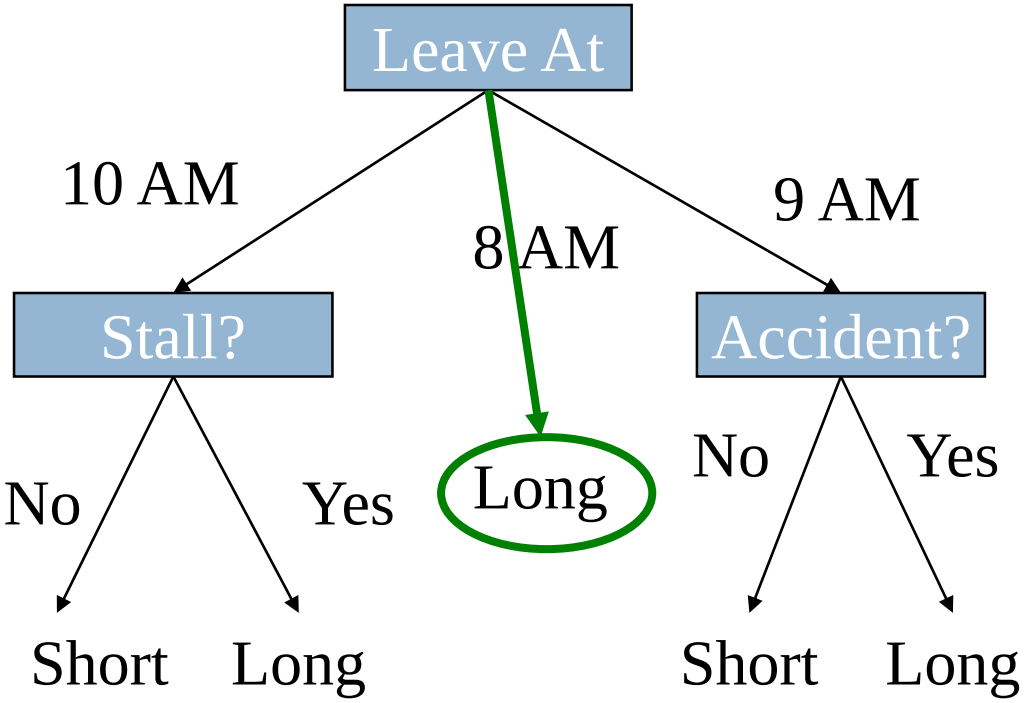
Accident = Yes
Stall = No

Tree with internal nodes labeled by features

Branches are labeled by tests on that feature

Leaves labeled with classes

Decision trees



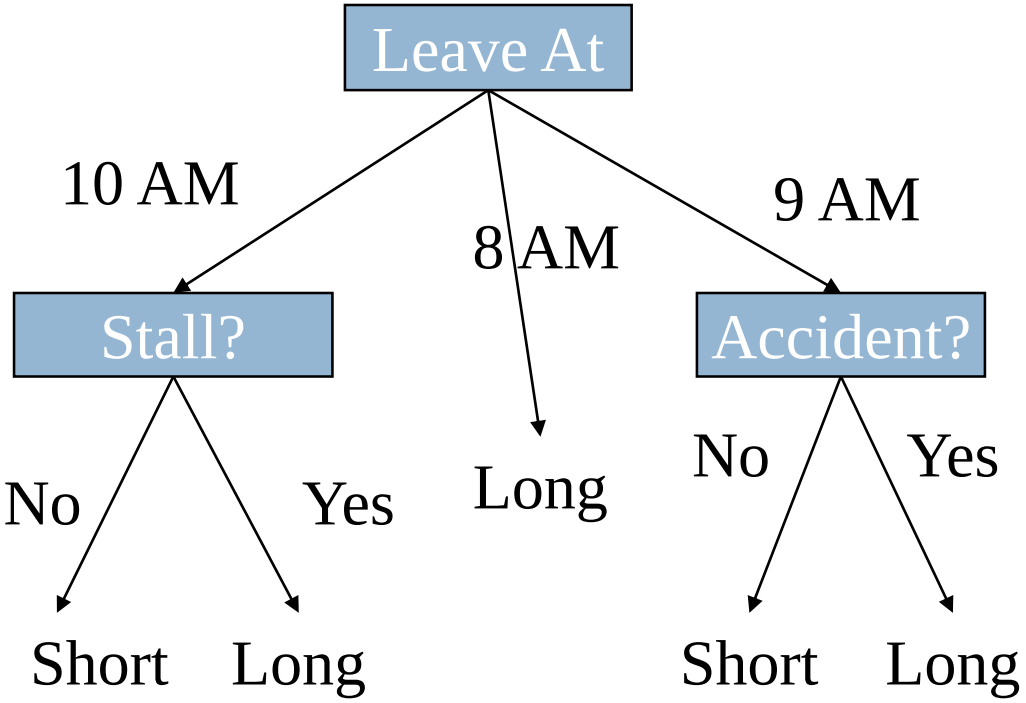
Leave = 8 AM
Weather = Rainy
Accident = Yes
Stall = No

Tree with internal nodes labeled by features

Branches are labeled by tests on that feature

Leaves labeled with classes

Decision trees



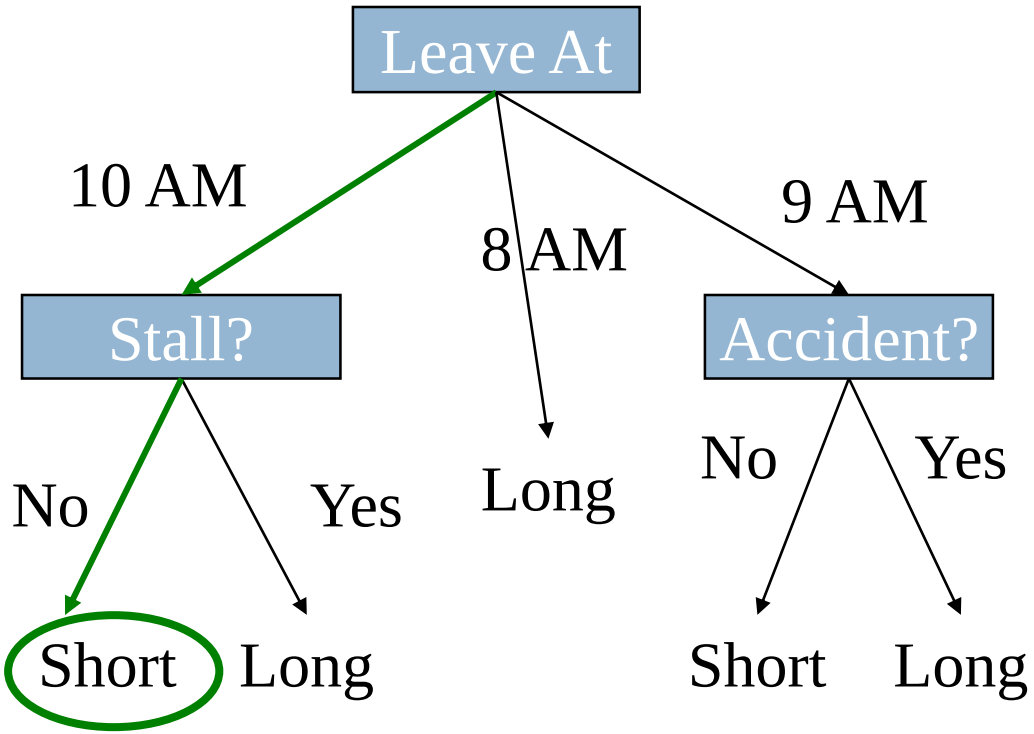
Leave = 10 AM Accident = No
Weather = Rainy Stall = No

Tree with internal nodes labeled by features

Branches are labeled by tests on that feature

Leaves labeled with classes

Decision trees



Leave = 10 AM Accident = No
Weather = Rainy Stall = No

Tree with internal nodes labeled by features

Branches are labeled by tests on that feature

Leaves labeled with classes

To ride or not to ride, that is the question…

Terrain	Unicycle-type	Weather	Go-For-Ride?
Trail	Normal	Rainy	NO
Road	Normal	Sunny	YES
Trail	Mountain	Sunny	YES
Road	Mountain	Rainy	YES
Trail	Normal	Snowy	NO
Road	Normal	Rainy	YES
Road	Mountain	Snowy	YES
Trail	Normal	Sunny	NO
Road	Normal	Snowy	NO
Trail	Mountain	Snowy	YES

Build a decision tree

Recursive approach

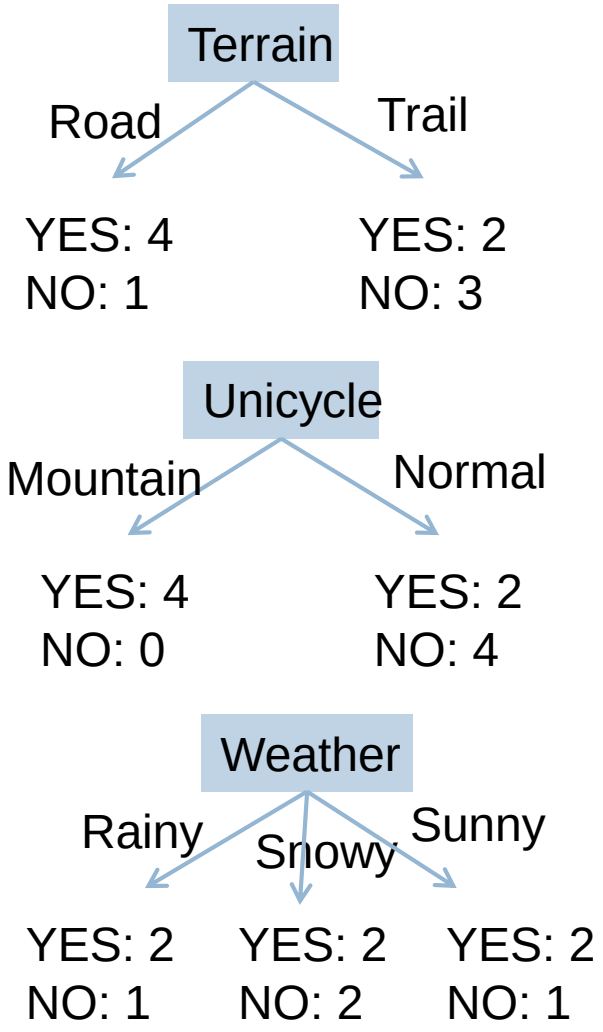
Base case: If all data belong to the same class, create a leaf node with that label

Otherwise:

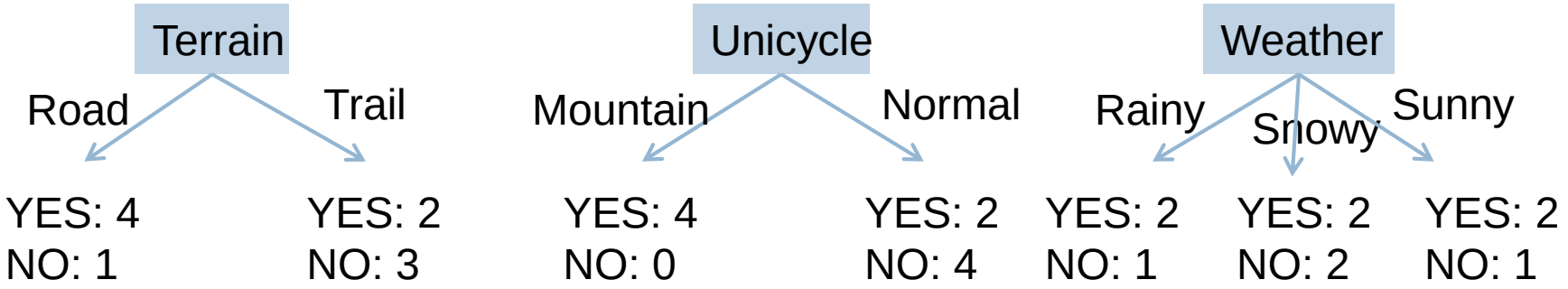
- calculate the “score” for each feature if we used it to split the data
- pick the feature with the highest score, partition the data based on that data value and call recursively

Partitioning the data

Terrain	Unicycle-type	Weather	Go-For-Ride?
Trail	Normal	Rainy	NO
Road	Normal	Sunny	YES
Trail	Mountain	Sunny	YES
Road	Mountain	Rainy	YES
Trail	Normal	Snowy	NO
Road	Normal	Rainy	YES
Road	Mountain	Snowy	YES
Trail	Normal	Sunny	NO
Road	Normal	Snowy	NO
Trail	Mountain	Snowy	YES



Partitioning the data

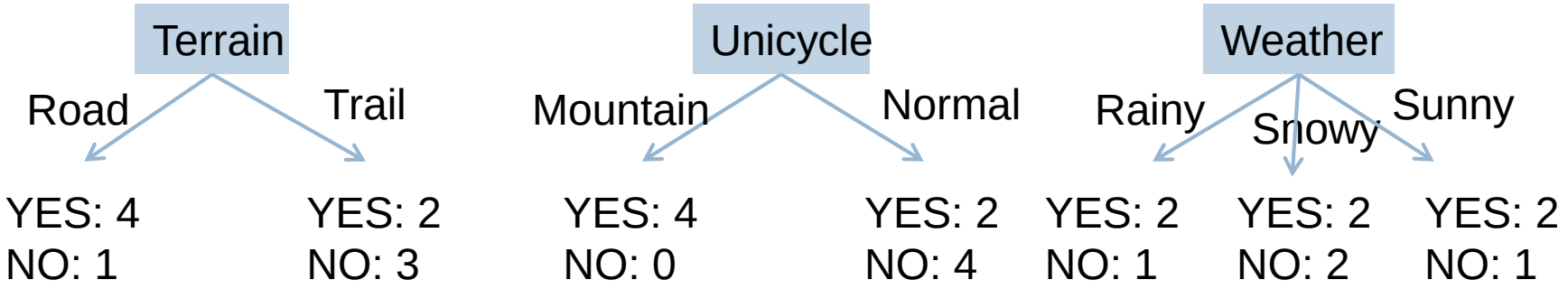


calculate the "score" for each feature if we used it to split the data

What score should we use?

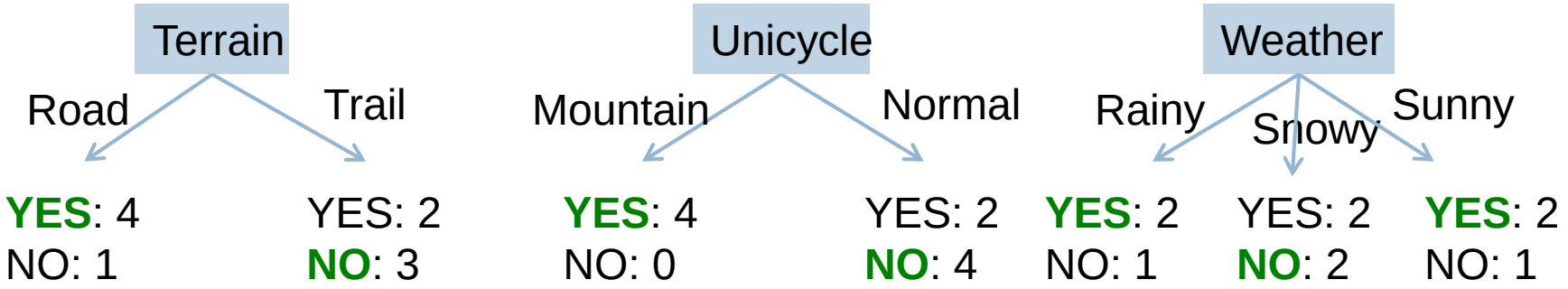
If we just stopped here, which tree would be best? How could we make these into decision trees?

Decision trees

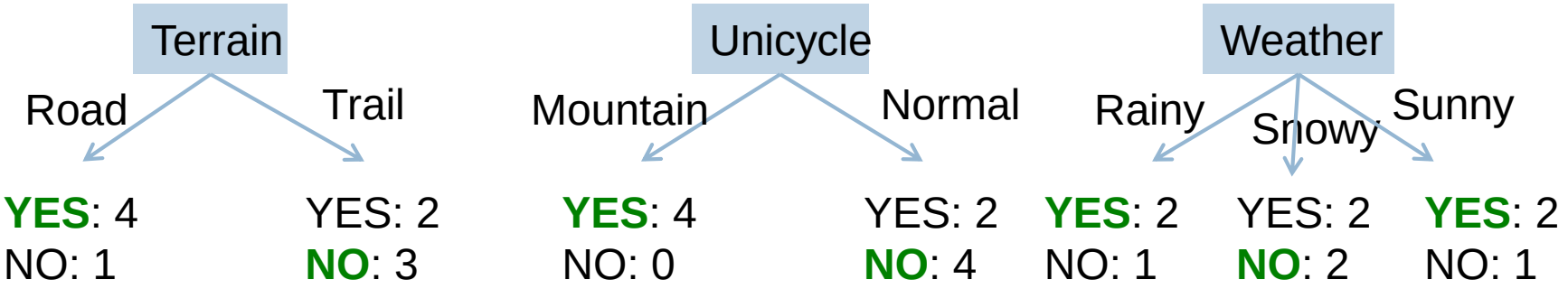


How could we make these into decision trees?

Decision trees



Decision trees

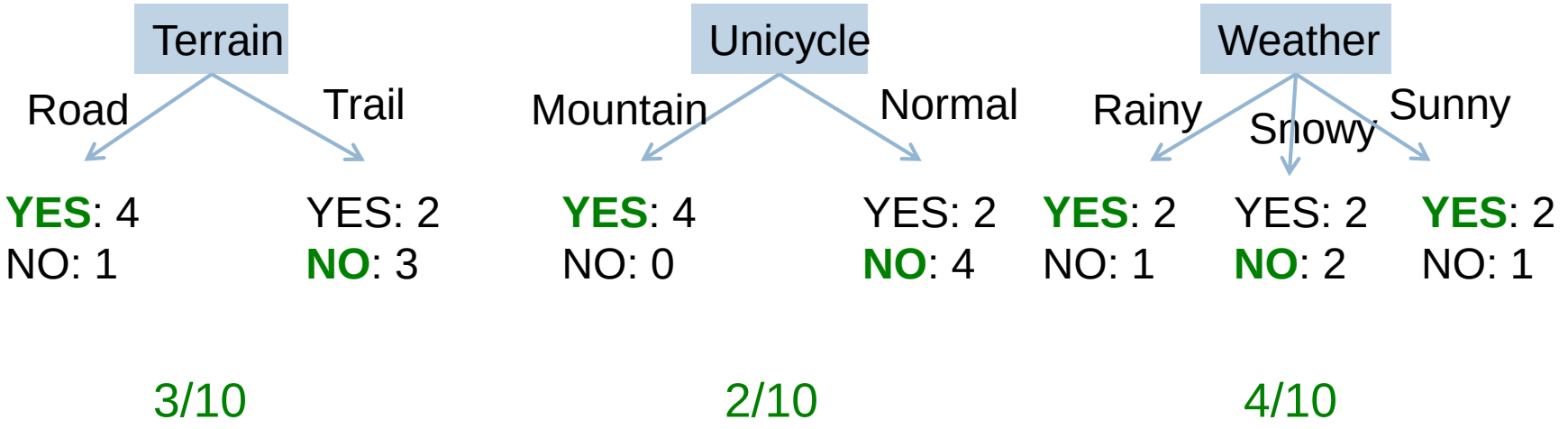


Training error: the average error over the training set

For classification, the most common "error" is the number of mistakes

Training error for each of these?

Decision trees



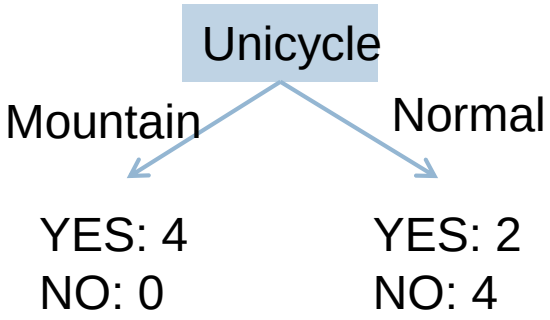
3/10

2/10

4/10

Training error: the average error over the training set

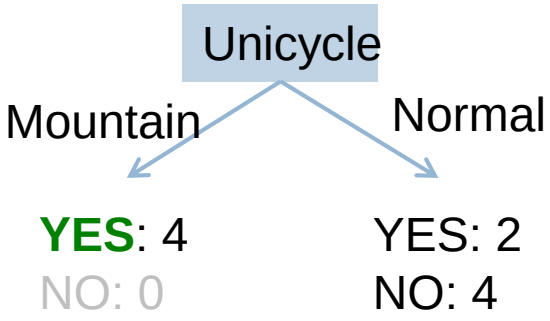
Recurse

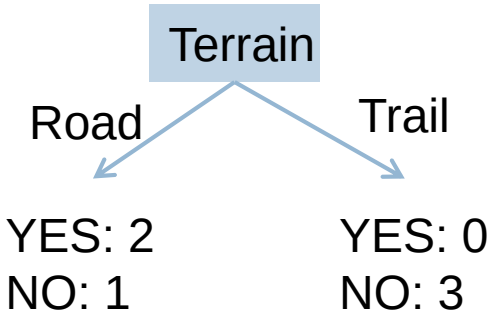


Terrain	Unicycle-type	Weather	Go-For-Ride?
Trail	Mountain	Sunny	YES
Road	Mountain	Rainy	YES
Road	Mountain	Snowy	YES
Trail	Mountain	Snowy	YES

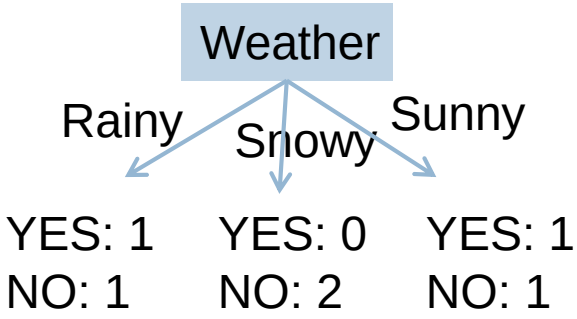
Terrain	Unicycle-type	Weather	Go-For-Ride?
Trail	Normal	Rainy	NO
Road	Normal	Sunny	YES
Trail	Normal	Snowy	NO
Road	Normal	Rainy	YES
Trail	Normal	Sunny	NO
Road	Normal	Snowy	NO

Recurse

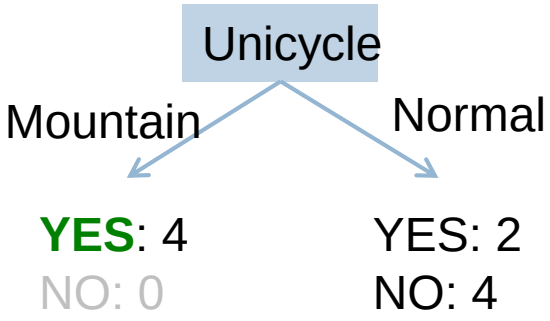




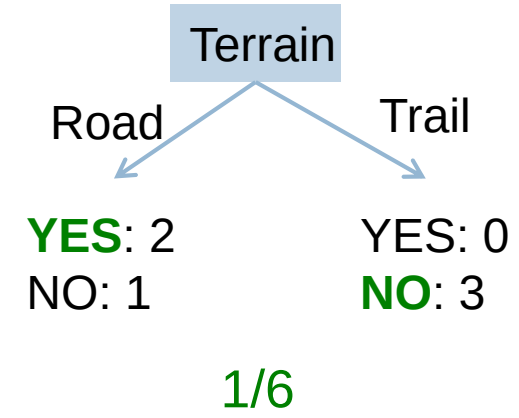
Terrain	Unicycle-type	Weather	Go-For-Ride?
Trail	Normal	Rainy	NO
Road	Normal	Sunny	YES
Trail	Normal	Snowy	NO
Road	Normal	Rainy	YES
Trail	Normal	Sunny	NO
Road	Normal	Snowy	NO

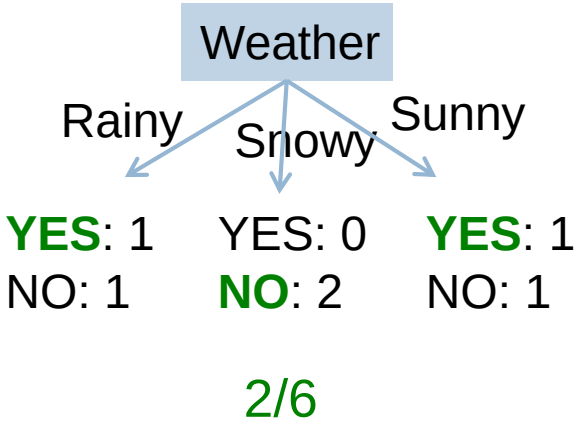


Recurse

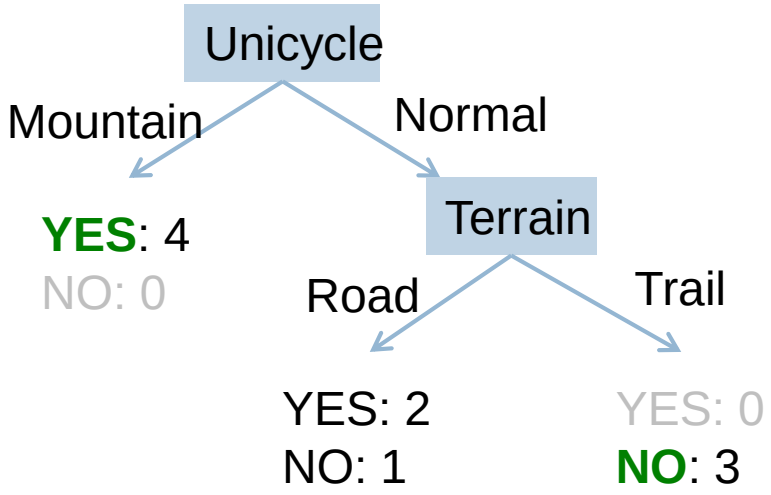


Terrain	Unicycle-type	Weather	Go-For-Ride?
Trail	Normal	Rainy	NO
Road	Normal	Sunny	YES
Trail	Normal	Snowy	NO
Road	Normal	Rainy	YES
Trail	Normal	Sunny	NO
Road	Normal	Snowy	NO



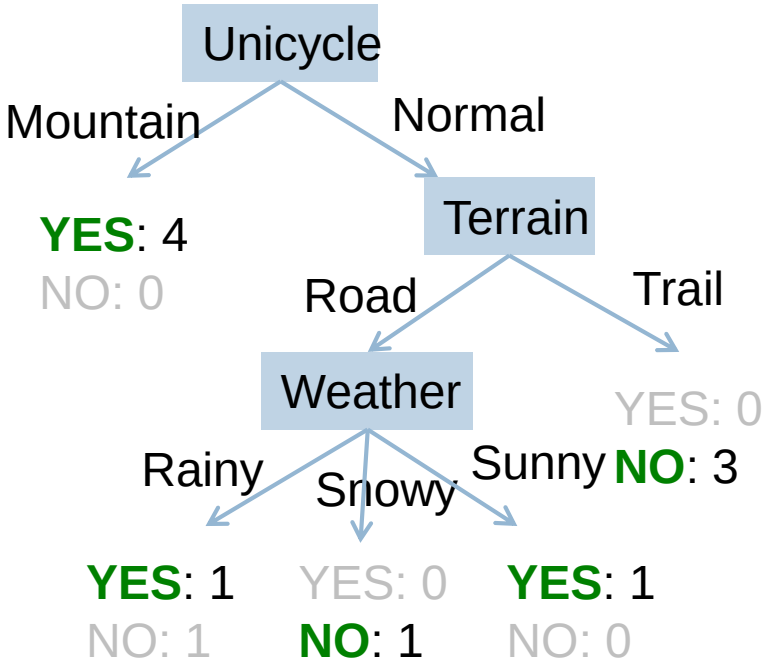


Recurse



Terrain	Unicycle-type	Weather	Go-For-Ride?
Road	Normal	Sunny	YES
Road	Normal	Rainy	YES
Road	Normal	Snowy	NO

Recurse


Unicycle
Mountain
Normal
YES: 4
NO: 0
Terrain
Road
Trail
Weather
YES: 0
NO: 3
Rainy
Snowy
Sunny
YES: 1
NO: 1
YES: 0
NO: 1
YES: 1
NO: 0

Building decision trees

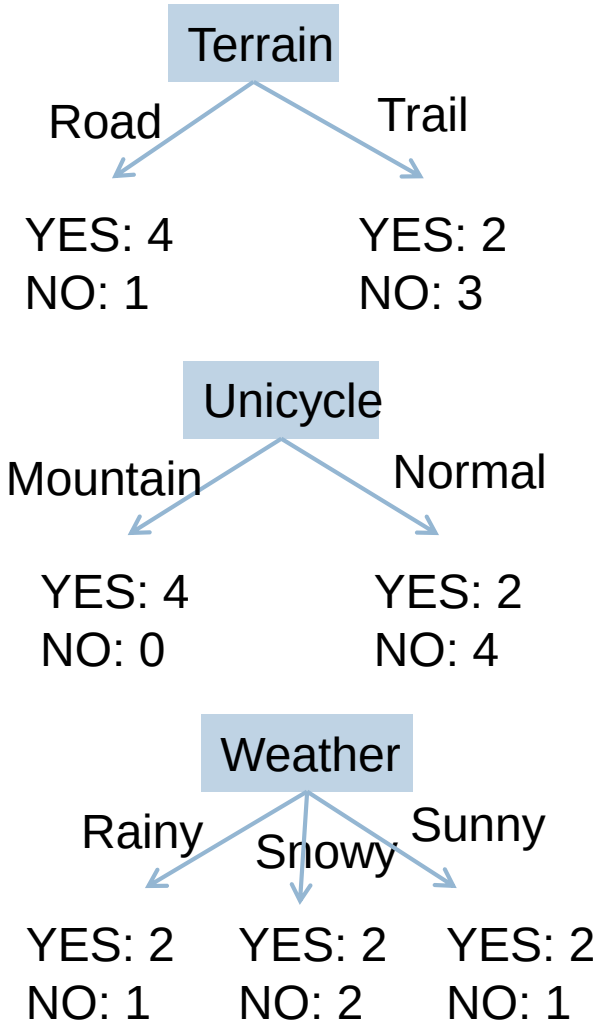
Base case: If all data belong to the same class, create a leaf node with that label

Otherwise:

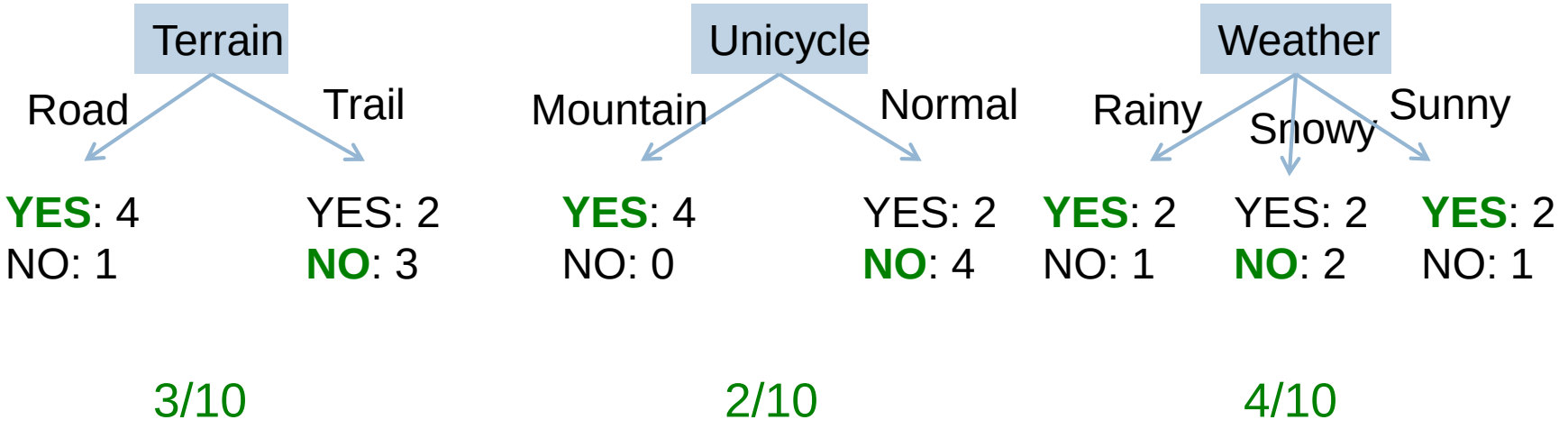
- calculate the “score” for each feature if we used it to split the data
- pick the feature with the highest score, partition the data based on that data value and call recursively

Partitioning the data

Terrain	Unicycle-type	Weather	Go-For-Ride?
Trail	Normal	Rainy	NO
Road	Normal	Sunny	YES
Trail	Mountain	Sunny	YES
Road	Mountain	Rainy	YES
Trail	Normal	Snowy	NO
Road	Normal	Rainy	YES
Road	Mountain	Snowy	YES
Trail	Normal	Sunny	NO
Road	Normal	Snowy	NO
Trail	Mountain	Snowy	YES



Decision trees



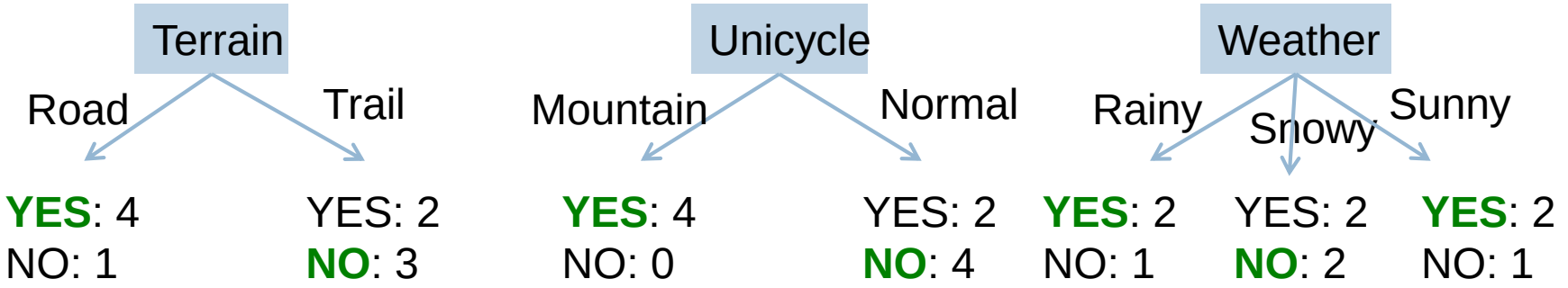
3/10

2/10

4/10

Training error: the average error over the training set

Training error vs. accuracy



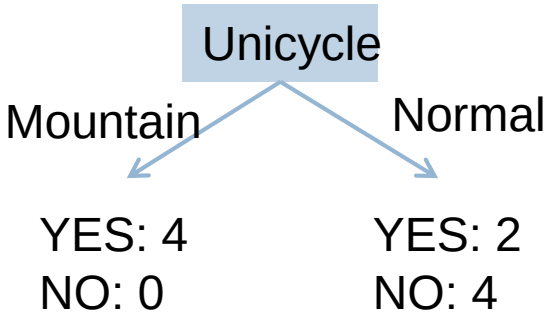
Training error:	3/10	2/10	4/10
Training accuracy:	7/10	8/10	6/10

training error = 1-accuracy (and vice versa)

Training error: the average error over the training set

Training accuracy: the average percent correct over the training set

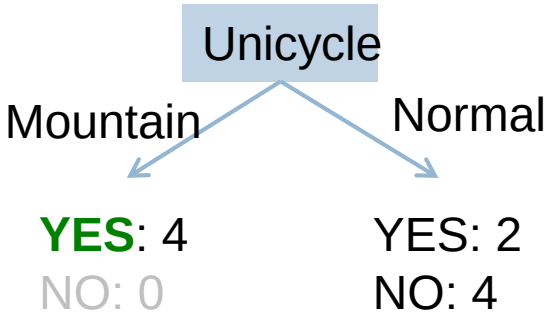
Recurse

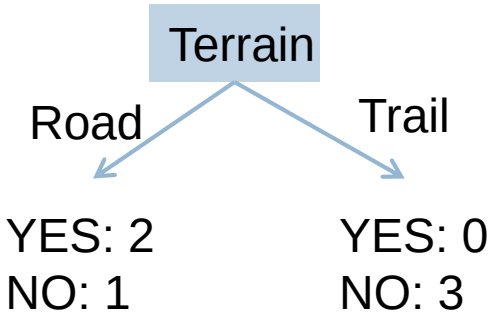


Terrain	Unicycle-type	Weather	Go-For-Ride?
Trail	Mountain	Sunny	YES
Road	Mountain	Rainy	YES
Road	Mountain	Snowy	YES
Trail	Mountain	Snowy	YES

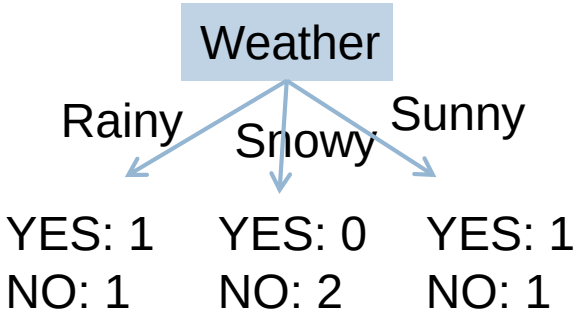
Terrain	Unicycle-type	Weather	Go-For-Ride?
Trail	Normal	Rainy	NO
Road	Normal	Sunny	YES
Trail	Normal	Snowy	NO
Road	Normal	Rainy	YES
Trail	Normal	Sunny	NO
Road	Normal	Snowy	NO

Recurse

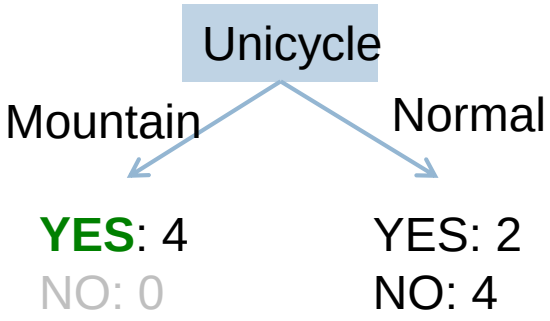




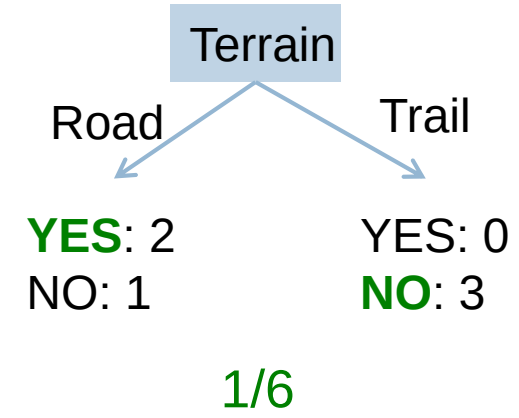
Terrain	Unicycle-type	Weather	Go-For-Ride?
Trail	Normal	Rainy	NO
Road	Normal	Sunny	YES
Trail	Normal	Snowy	NO
Road	Normal	Rainy	YES
Trail	Normal	Sunny	NO
Road	Normal	Snowy	NO

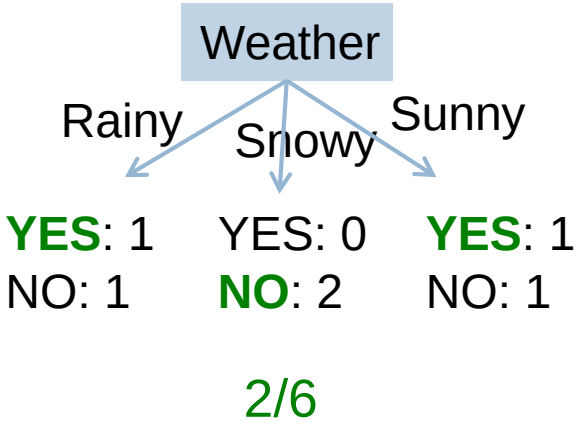


Recurse

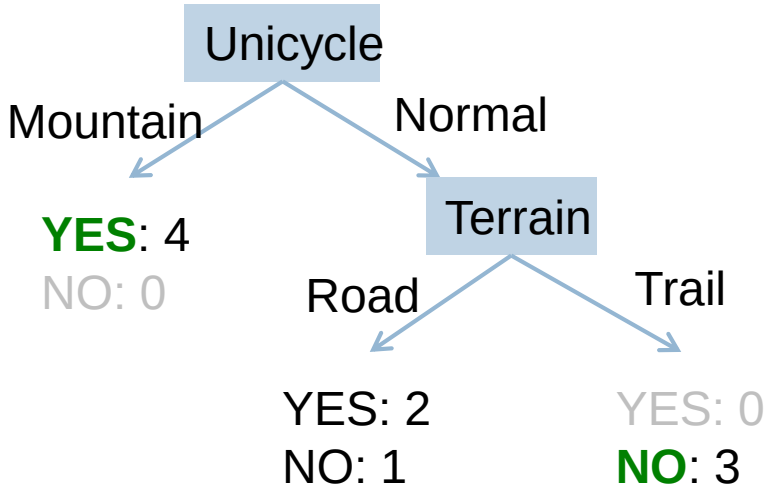


Terrain	Unicycle-type	Weather	Go-For-Ride?
Trail	Normal	Rainy	NO
Road	Normal	Sunny	YES
Trail	Normal	Snowy	NO
Road	Normal	Rainy	YES
Trail	Normal	Sunny	NO
Road	Normal	Snowy	NO



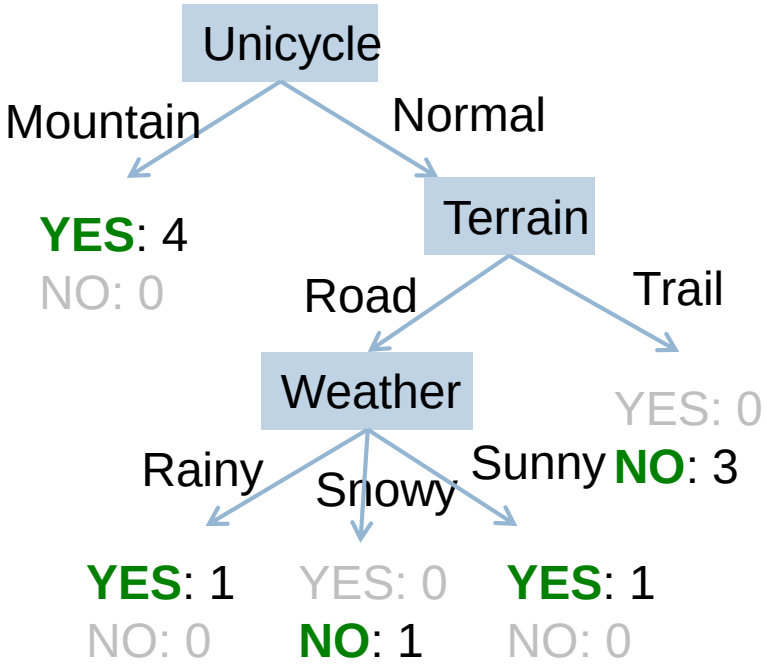


Recurse

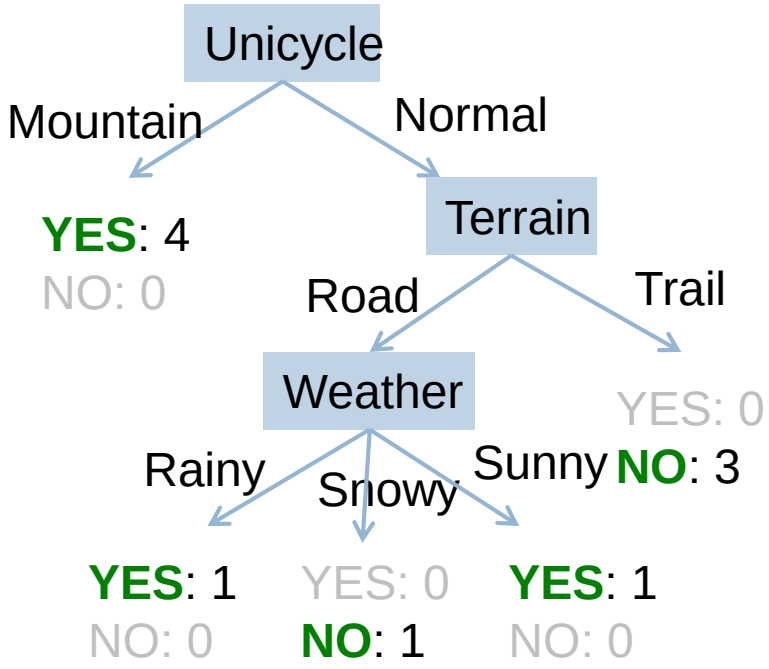


Terrain	Unicycle-type	Weather	Go-For-Ride?
Road	Normal	Sunny	YES
Road	Normal	Rainy	YES
Road	Normal	Snowy	NO

Recurse


Unicycle
Mountain
Normal
YES: 4
NO: 0
Terrain
Road
Trail
Weather
YES: 0
NO: 3
Rainy
Snowy
Sunny
YES: 1
NO: 0
YES: 0
NO: 1
YES: 1
NO: 0

Recurse



Terrain	Unicycle-type	Weather	Go-For-Ride?
Trail	Normal	Rainy	NO
Road	Normal	Sunny	YES
Trail	Mountain	Sunny	YES
Road	Mountain	Rainy	YES
Trail	Normal	Snowy	NO
Road	Normal	Rainy	YES
Road	Mountain	Snowy	YES
Trail	Normal	Sunny	NO
Road	Normal	Snowy	NO
Trail	Mountain	Snowy	YES

Training error?

Are we always guaranteed to get a training error of 0?

Problematic data

Terrain	Unicycle-type	Weather	Go-For-Ride?
Trail	Normal	Rainy	NO
Road	Normal	Sunny	YES
Trail	Mountain	Sunny	YES
Road	Mountain	Snowy	NO
Trail	Normal	Snowy	NO
Road	Normal	Rainy	YES
Road	Mountain	Snowy	YES
Trail	Normal	Sunny	NO
Road	Normal	Snowy	NO
Trail	Mountain	Snowy	YES

When can this happen?

Recursive approach

Base case: If all data belong to the same class, create a leaf node with that label ***OR*** all the data has the same feature values

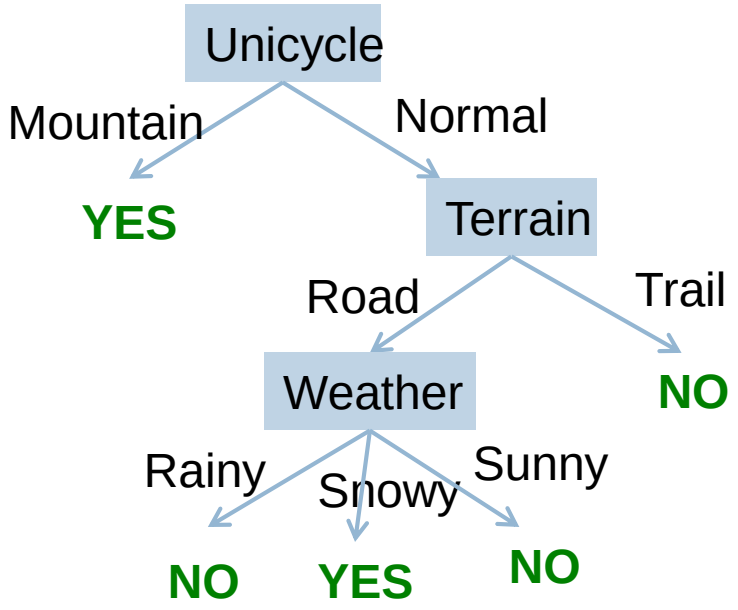
Do we always want to go all the way to the bottom?

What would the tree look like for…

Terrain	Unicycle-type	Weather	Go-For-Ride?
Trail	Mountain	Rainy	YES
Trail	Mountain	Sunny	YES
Road	Mountain	Snowy	YES
Road	Mountain	Sunny	YES
Trail	Normal	Snowy	NO
Trail	Normal	Rainy	NO
Road	Normal	Snowy	YES
Road	Normal	Sunny	NO
Trail	Normal	Sunny	NO

What would the tree look like for…

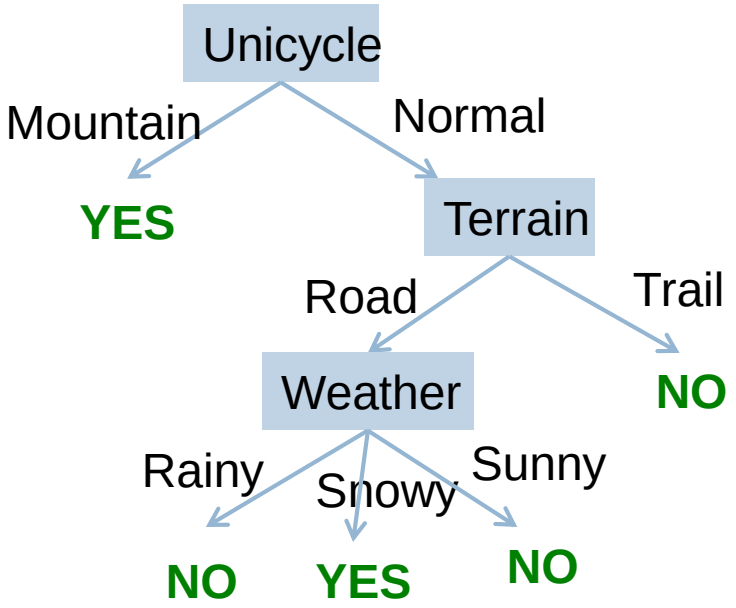
Terrain	Unicycle-type	Weather	Go-For-Ride?
Trail	Mountain	Rainy	YES
Trail	Mountain	Sunny	YES
Road	Mountain	Snowy	YES
Road	Mountain	Sunny	YES
Trail	Normal	Snowy	NO
Trail	Normal	Rainy	NO
Road	Normal	Snowy	YES
Road	Normal	Sunny	NO
Trail	Normal	Sunny	NO



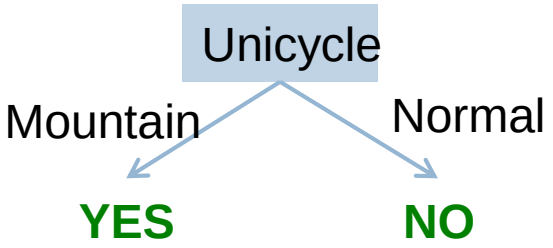
Is that what you would do?

What would the tree look like for…

Terrain	Unicycle-type	Weather	Go-For-Ride?
Trail	Mountain	Rainy	YES
Trail	Mountain	Sunny	YES
Road	Mountain	Snowy	YES
Road	Mountain	Sunny	YES
Trail	Normal	Snowy	NO
Trail	Normal	Rainy	NO
Road	Normal	Snowy	YES
Road	Normal	Sunny	NO
Trail	Normal	Sunny	NO



Maybe…

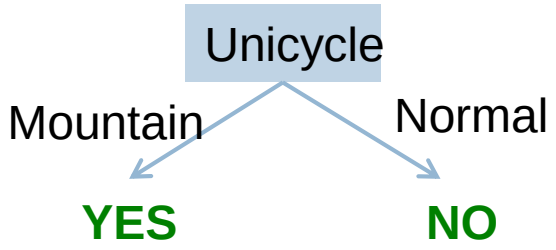


What would the tree look like for…

Terrain	Unicycle-type	Weather	Jacket	ML grade	Go-For-Ride?
Trail	Mountain	Rainy	Heavy	D	YES
Trail	Mountain	Sunny	Light	C-	YES
Road	Mountain	Snowy	Light	B	YES
Road	Mountain	Sunny	Heavy	A	YES
…	Mountain	…	…	…	YES
Trail	Normal	Snowy	Light	D+	NO
Trail	Normal	Rainy	Heavy	B-	NO
Road	Normal	Snowy	Heavy	C+	YES
Road	Normal	Sunny	Light	A-	NO
Trail	Normal	Sunny	Heavy	B+	NO
Trail	Normal	Snowy	Light	F	NO
…	Normal	…	…	…	NO
Trail	Normal	Rainy	Light	C	YES

Overfitting

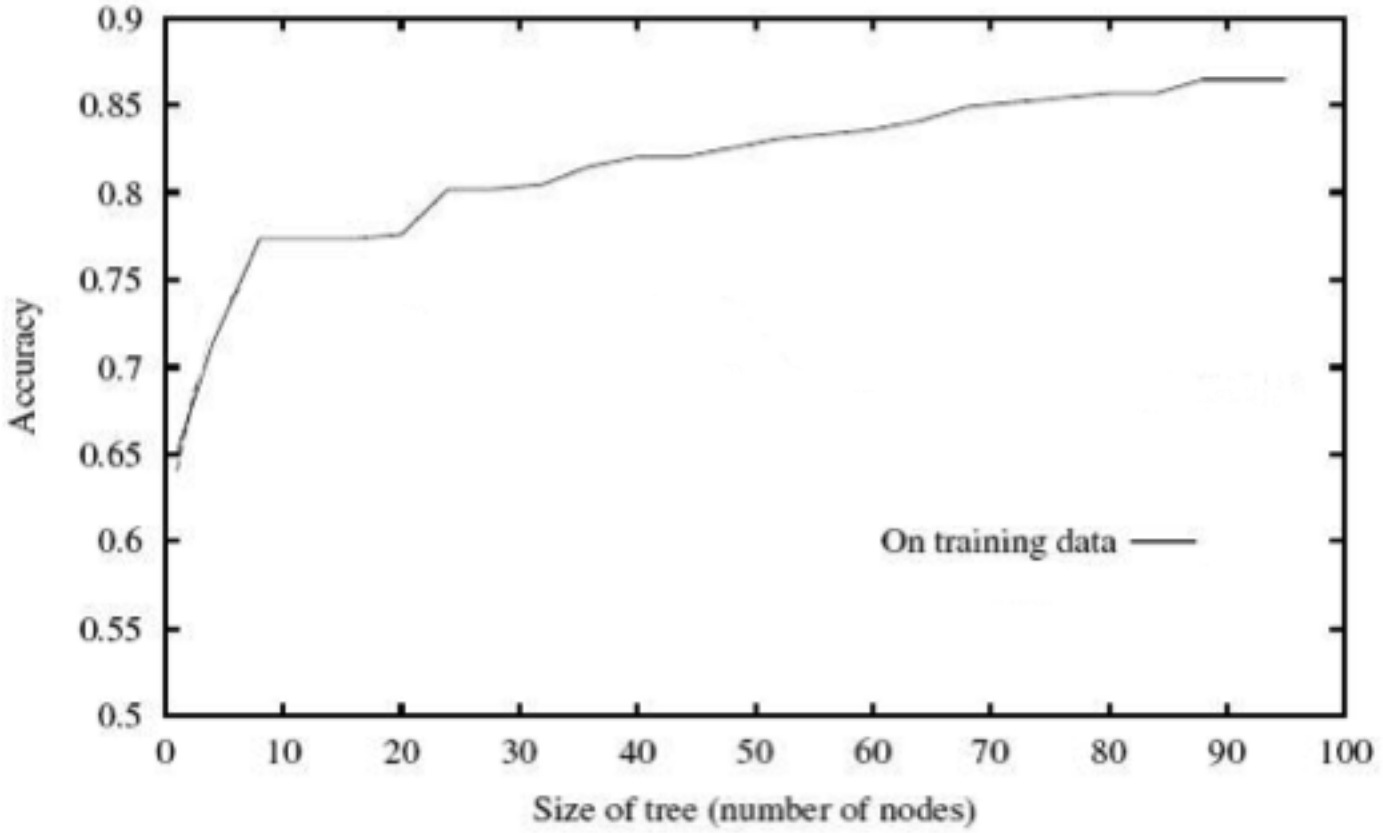
Terrain	Unicycle-type	Weather	Go-For-Ride?
Trail	Mountain	Rainy	YES
Trail	Mountain	Sunny	YES
Road	Mountain	Snowy	YES
Road	Mountain	Sunny	YES
Trail	Normal	Snowy	NO
Trail	Normal	Rainy	NO
Road	Normal	Snowy	YES
Road	Normal	Sunny	NO
Trail	Normal	Sunny	NO



Overfitting occurs when we bias our model too much towards the training data

Our goal is to learn a **general** model that will work on the training data as well as other data (i.e. test data)

Overfitting



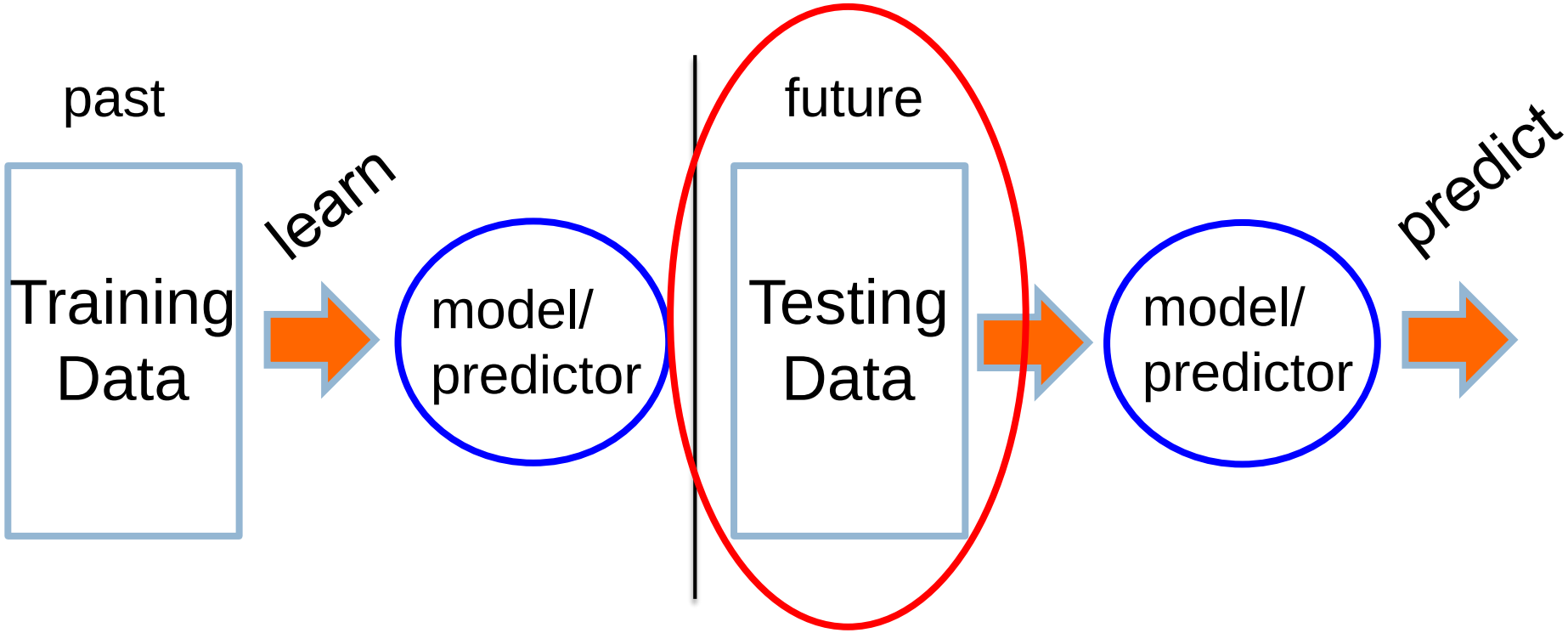
Our decision tree learning procedure always decreases training error

Is that what we want?

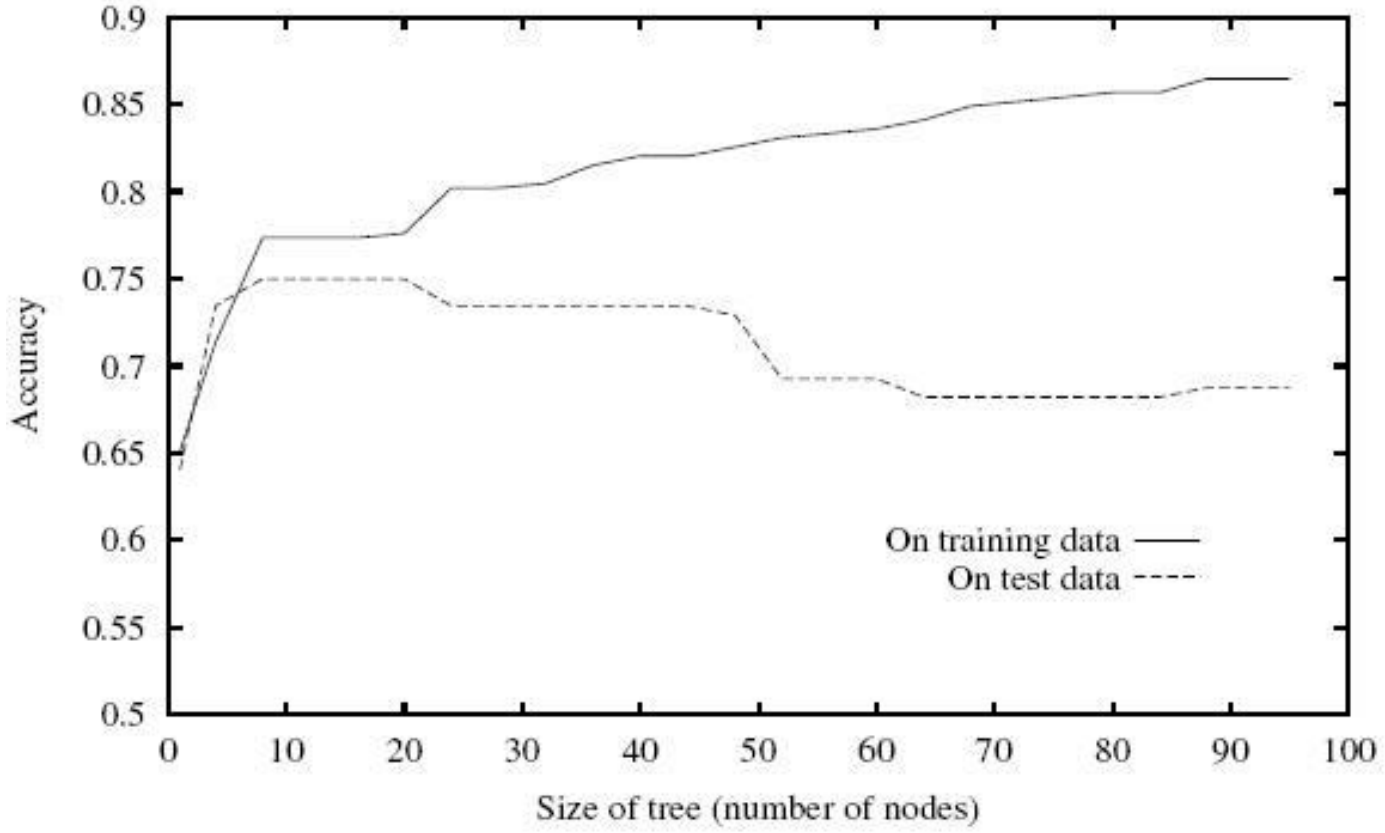
Test set error!

Machine learning is about predicting the future based on the past.

-- Hal Daume III



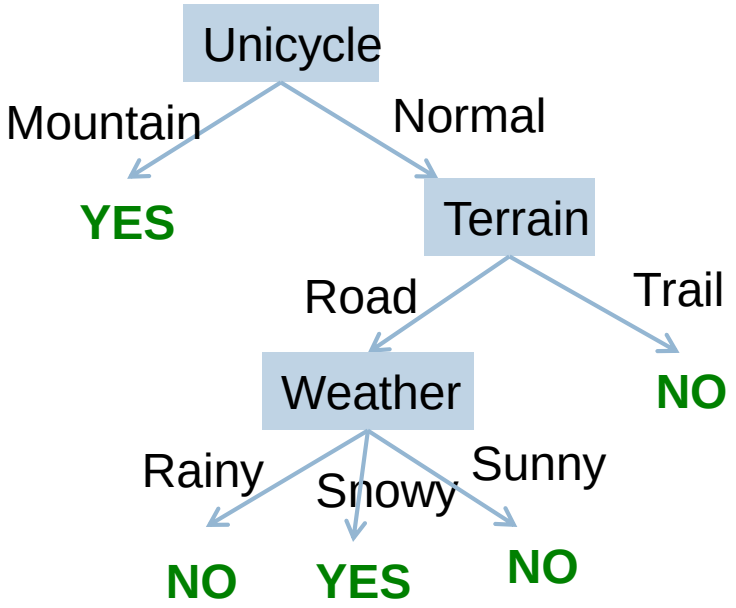
Overfitting



Even though the training error is decreasing, the testing error can go up!

Overfitting

Terrain	Unicycle-type	Weather	Go-For-Ride?
Trail	Mountain	Rainy	YES
Trail	Mountain	Sunny	YES
Road	Mountain	Snowy	YES
Road	Mountain	Sunny	YES
Trail	Normal	Snowy	NO
Trail	Normal	Rainy	NO
Road	Normal	Snowy	YES
Road	Normal	Sunny	NO
Trail	Normal	Sunny	NO



How do we prevent overfitting?

Preventing overfitting

Base case: If all data belong to the same class, create a leaf node with that label ***OR*** all the data has the same feature values ***OR***

- We've reached a particular depth in the tree
- ?

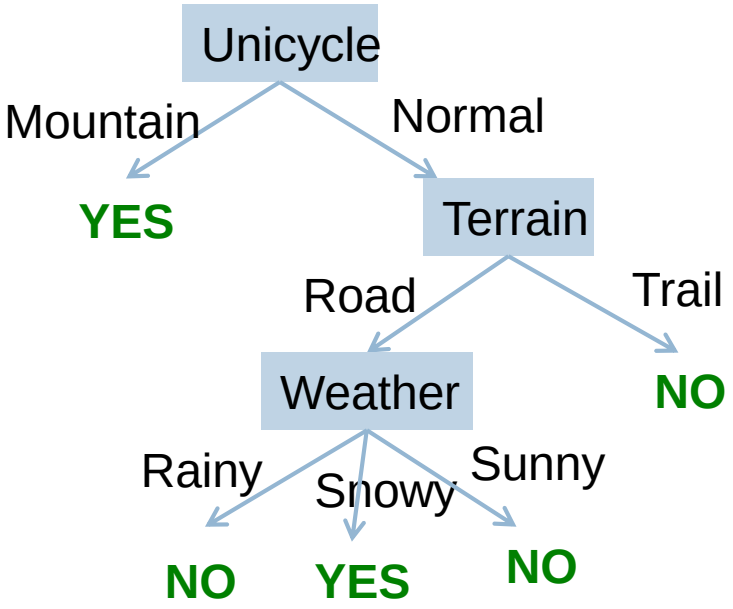
One idea: stop building the tree early

Preventing overfitting

Base case: If all data belong to the same class, create a leaf node with that label ***OR*** all the data has the same feature values ***OR***

- We've reached a particular depth in the tree
- We only have a certain number/fraction of examples remaining
- We've reached a particular training error
- Use development data (more on this later)
- …

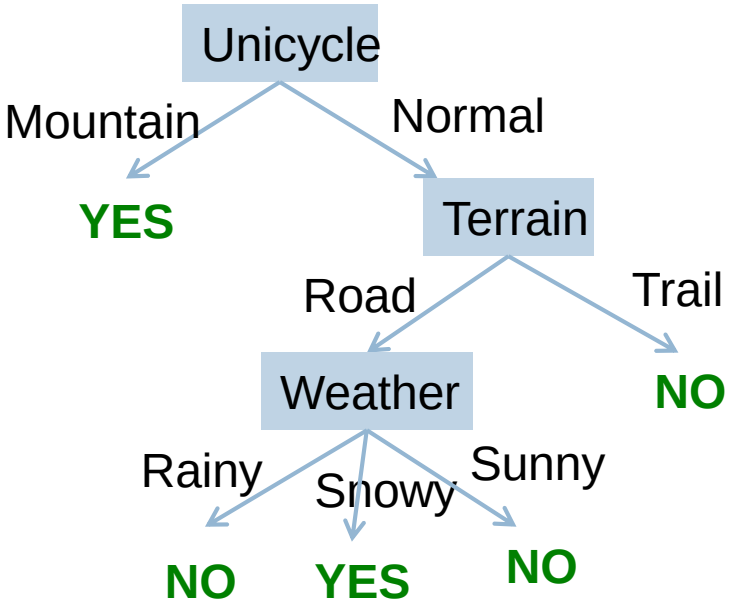
Preventing overfitting: pruning



Pruning: after the tree is built, go back and “prune” the tree, i.e. remove some lower parts of the tree

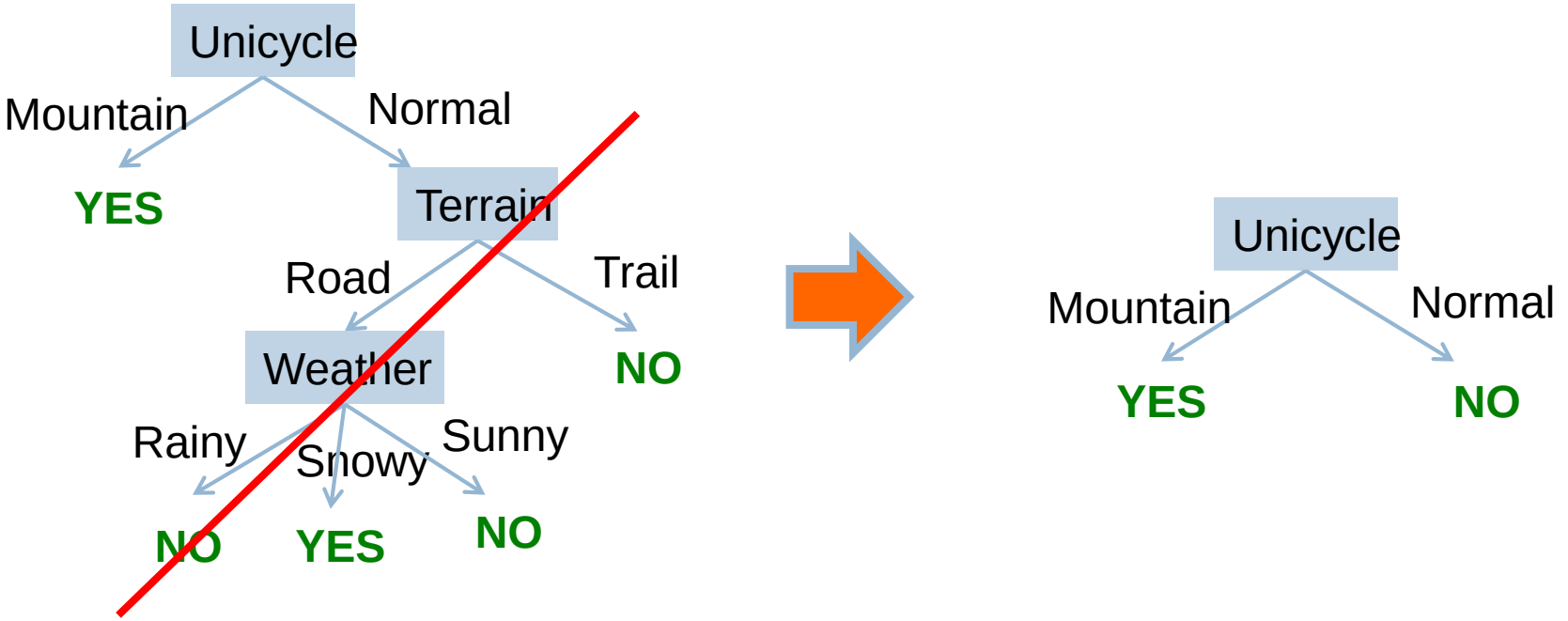
Similar to stopping early, but done after the entire tree is built

Preventing overfitting: pruning



Build the full tree

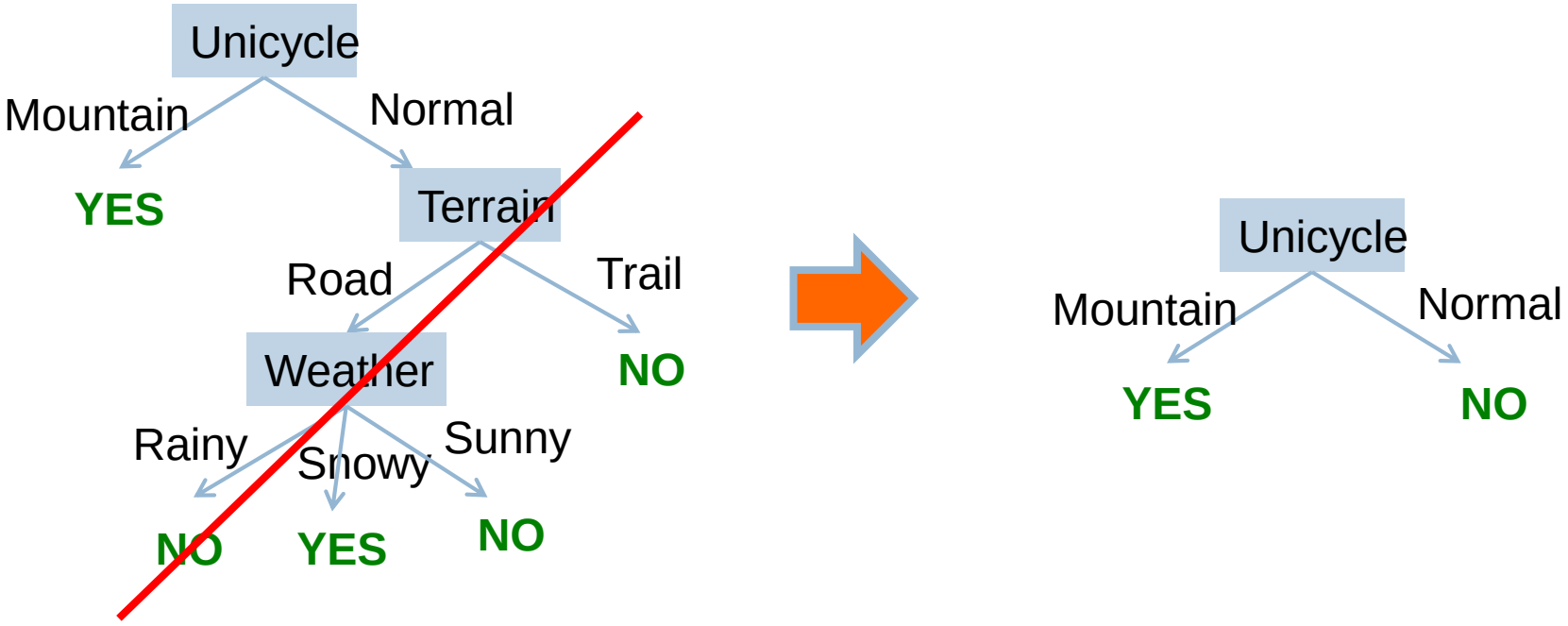
Preventing overfitting: pruning



Build the full tree

Prune back leaves that are too specific

Preventing overfitting: pruning



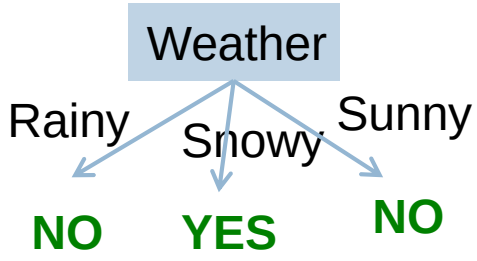
Pruning criterion?

Handling non-binary attributes

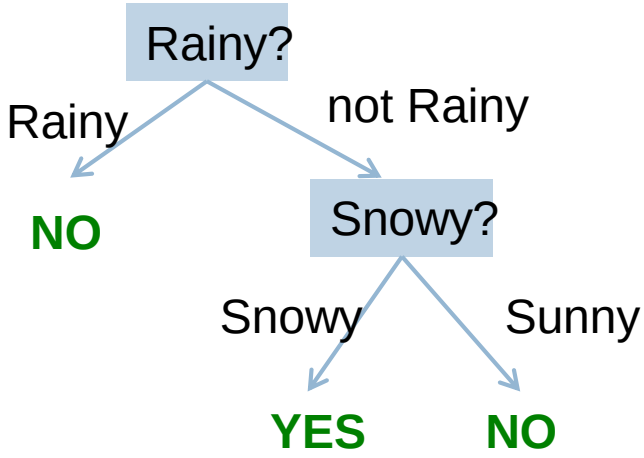
PassengerId	Pclass	Sex	Age	SibSp	Parch	Ticket	Fare	Embarked	Survived
804	3	0	0.42	0	1	2625	8.5167	0	1
756	2	0	0.67	1	1	250649	14.5	2	1
470	3	1	0.75	2	1	2666	19.2583	0	1
645	3	1	0.75	2	1	2666	19.2583	0	1
79	2	0	0.83	0	2	248738	29	2	1
832	2	0	0.83	1	1	29106	18.75	2	1
306	1	0	0.92	1	2	113781	151.55	2	1
165	3	0	1	4	1	3101295	39.6875	2	0
173	3	1	1	1	1	347742	11.1333	2	1
184	2	0	1	2	1	230136	39	2	1
382	3	1	1	0	2	2653	15.7417	0	1
387	3	0	1	5	2	2144	46.9	2	0
789	3	0	1	1	2	2315	20.575	2	1
828	2	0	1	0	2	2079	37.0042	0	1
8	3	0	2	3	1	349909	21.075	2	0
17	3	0	2	4	1	382652	29.125	1	0
120	3	1	2	4	2	347082	31.275	2	0
206	3	1	2	0	1	347054	10.4625	2	0
298	1	1	2	1	2	113781	151.55	2	0
341	2	0	2	1	1	230080	26	2	1
480	3	1	2	0	1	3101298	12.2875	2	1

What do we do with features that have multiple values?
Real-values?

Features with multiple values



Treat as an n-ary split

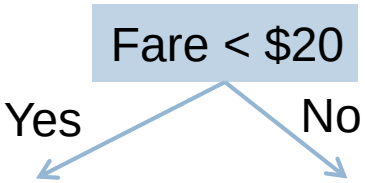


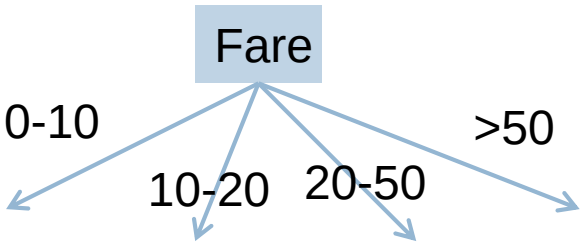
Treat as multiple binary splits

Real-valued features

Use any comparison test (>, <, ≤, ≥) to split the data into two parts

Select a range filter, i.e. min < value < max





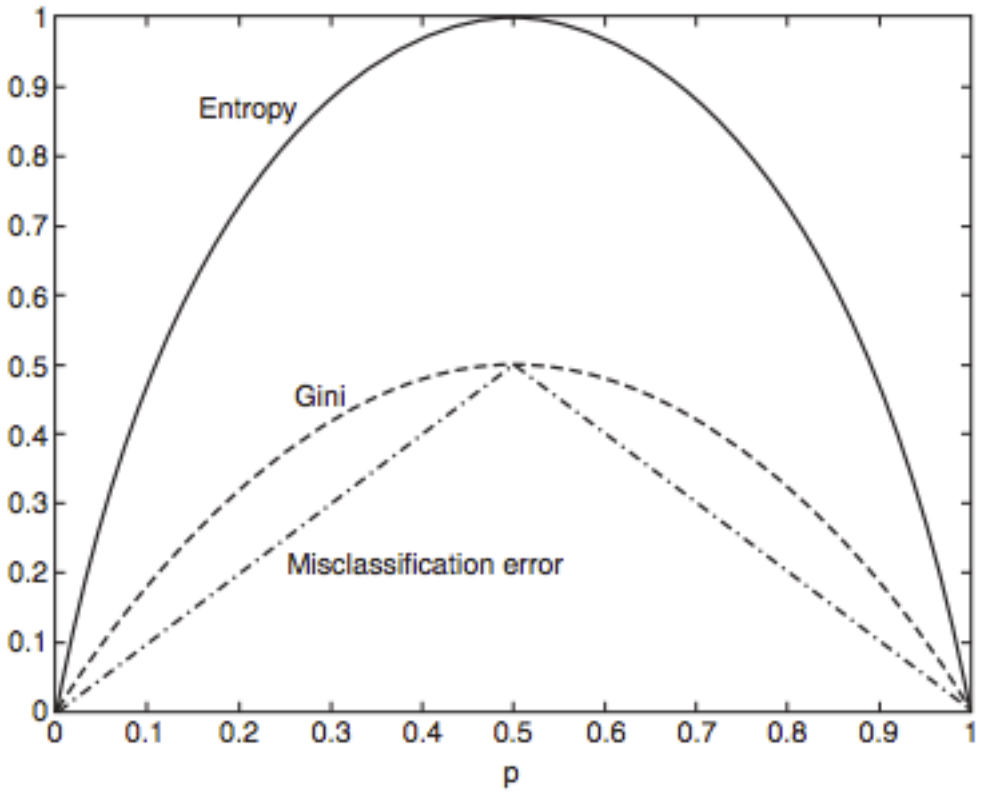
Other splitting criterion

Otherwise:

- calculate the "score" for each feature if we used it to split the data
- pick the feature with the highest score, partition the data based on that data value and call recursively

We used training error for the score. Any other ideas?

Other splitting criterion



- Entropy: how much uncertainty there is in the distribution over labels after the split
- Gini: sum of the square of the label proportions after split
- Training error = misclassification error

Decision trees

Good? Bad?

Decision trees: the good

Very intuitive and easy to interpret

Fast to run and fairly easy to implement (Assignment 2 ☺)

Historically, perform fairly well (especially with a few more tricks we'll see later on)

No prior assumptions about the data

Decision trees: the bad

Be careful with features with lots of values

Which feature would be at the top here?

ID	Terrain	Unicycle-type	Weather	Go-For-Ride?
1	Trail	Normal	Rainy	NO
2	Road	Normal	Sunny	YES
3	Trail	Mountain	Sunny	YES
4	Road	Mountain	Rainy	YES
5	Trail	Normal	Snowy	NO
6	Road	Normal	Rainy	YES
7	Road	Mountain	Snowy	YES
8	Trail	Normal	Sunny	NO
9	Road	Normal	Snowy	NO
10	Trail	Mountain	Snowy	YES

Decision trees: the bad

Can be problematic (slow, bad performance) with large numbers of features

Can’t learn some very simple data sets (e.g. some types of linearly separable data)

Pruning/tuning can be tricky to get right

Final DT algorithm

Base cases:

1. If all data belong to the same class, pick that label
2. If all the data have the same feature values, pick majority label
3. If we're out of features to examine, pick majority label
4. If the we don't have any data left, pick majority label of *parent*
5. *If some other stopping criteria* exists to avoid overfitting, pick majority label

Otherwise:

- calculate the "score" for each feature if we used it to split the data
- pick the feature with the highest score, partition the data based on that data value and call recursively

Alternative Score Calculation – Information Gain

- Why information gain is better over accuracy?
  - Decision trees are generally prone to over-fitting and accuracy doesn't generalize well to unseen data.
  - One advantage of information gain is that -- due to the factor $-p*\log(p)$ in the entropy definition -- leafs with a small number of instances are assigned less weight ($\lim p \to 0 + p*\log(p) = 0$) and it favors dividing data into bigger but homogeneous groups.
  - This approach is usually more stable and also chooses the most impactful features close to the root of the tree.

To illustrate, imagine the task of learning to classify first-names into male/female groups. That is given a list of names each labeled with either `m` or `f`, we want to learn a model that fits the data and can be used to predict the gender of a new unseen first-name.

```
name       gender
----------------        Now we want to predict
Ashley        f             the gender of "Amro" (my name)
Brian         m
Caroline      f
David         m
```

First step is deciding what **features** of the data are relevant to the target class we want to predict. Some example features include: first/last letter, length, number of vowels, does it end with a vowel, etc.. So after feature extraction, our data looks like:

```
# name     ends-vowel  num-vowels   length   gender
# ------------------------------------------------
Ashley         1          3           6         f
Brian          0          2           5         m
Caroline       1          4           8         f
David          0          2           5         m
```

The goal is to build a decision tree. An example of a tree would be:

```
length<7
|   num-vowels<3: male
|   num-vowels>=3
|   |   ends-vowel=1: female
|   |   ends-vowel=0: male
length>=7
|   length=5: male
```

basically each node represent a test performed on a single attribute, and we go left or right depending on the result of the test. We keep traversing the tree until we reach a leaf node which contains the class prediction (`m` or `f`)

So if we run the name *Amro* down this tree, we start by testing "*is the length<7?*" and the answer is *yes*, so we go down that branch. Following the branch, the next test "*is the number of vowels<3?*" again evaluates to *true*. This leads to a leaf node labeled `m` , and thus the prediction is *male* (which I happen to be, so the tree predicted the outcome correctly).

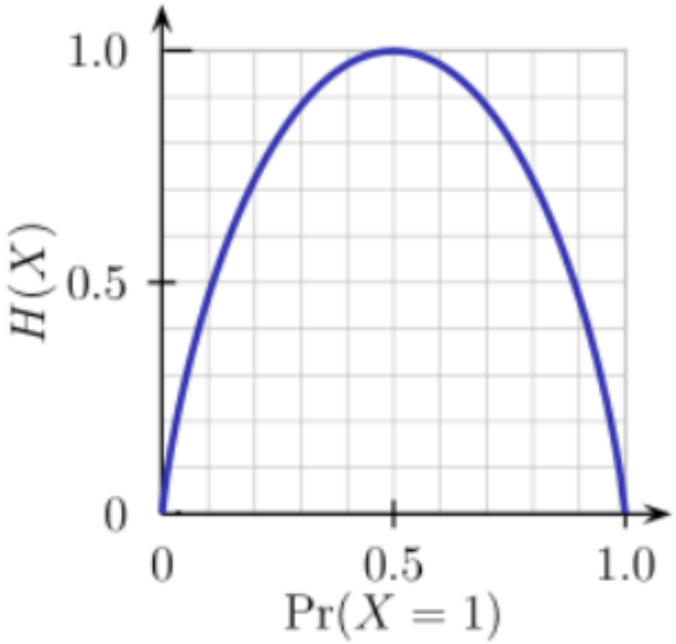
The decision tree is built in a top-down fashion, but the question is how do you choose which attribute to split at each node? The answer is find the feature that best splits the target class into the purest possible children nodes (ie: nodes that don't contain a mix of both male and female, rather pure nodes with only one class).

This measure of *purity* is called the **information**. It represents the expected amount of information that would be needed to specify whether a new instance (first-name) should be classified male or female, given the example that reached the node. We calculate it based on the number of male and female classes at the node.

Entropy on the other hand is a measure of *impurity* (the opposite). It is defined for a binary class with values `a` / `b` as:

```
Entropy = - p(a)*log(p(a)) - p(b)*log(p(b))
```

This binary entropy function is depicted in the figure below (random variable can take one of two values). It reaches its maximum when the probability is `p=1/2`, meaning that `p(X=a)=0.5` or similarly `p(X=b)=0.5` having a 50%/50% chance of being either `a` or `b` (uncertainty is at a maximum). The entropy function is at zero minimum when probability is `p=1` or `p=0` with complete certainty (`p(X=a)=1` or `p(X=a)=0` respectively, latter implies `p(X=b)=1`).



Of course the definition of entropy can be generalized for a discrete random variable X with N outcomes (not just two):

$$H(X) = -\sum_{i=1}^{n} p(x_i) \log_b p(x_i)$$

(the `Log` in the formula is usually taken as the logarithm to the base 2)

Back to our task of name classification, lets look at an example. Imagine at some point during the process of constructing the tree, we were considering the following split:

```
     ends-vowel
      [9m,5f]          <--- the [..,..] notation represents the class
     /        \             distribution of instances that reached a node
   =1          =0
 -------     -------
 [3m,4f]     [6m,1f]
```

As you can see, before the split we had 9 males and 5 females, i.e. `P(m)=9/14` and `P(f)=5/14`. According to the definition of entropy:

```
Entropy_before = - (5/14)*log2(5/14) - (9/14)*log2(9/14) = 0.9403
```

Next we compare it with the entropy computed after considering the split by looking at two child branches. In the left branch of `ends-vowel=1`, we have:

```
Entropy_left = - (3/7)*log2(3/7) - (4/7)*log2(4/7) = 0.9852
```

and the right branch of `ends-vowel=0`, we have:

```
Entropy_right = - (6/7)*log2(6/7) - (1/7)*log2(1/7) = 0.5917
```

We combine the left/right entropies using the number of instances down each branch as weight factor (7 instances went left, and 7 instances went right), and get the final entropy after the split:

```
Entropy_after = 7/14*Entropy_left + 7/14*Entropy_right = 0.7885
```

Information Gain Example

Now by comparing the entropy before and after the split, we obtain a measure of **information gain**, or how much information we gained by doing the split using that particular feature:

```
Information_Gain = Entropy_before - Entropy_after = 0.1518
```

You can interpret the above calculation as following: by doing the split with the `end-vowels` *feature, we were able to reduce uncertainty in the sub-tree prediction outcome by a small amount of 0.1518 (measured in bits as units of information).*

At each node of the tree, this calculation is performed for every feature, and the feature with the *largest information gain* is chosen for the split in a greedy manner (thus favoring features that produce *pure* splits with low uncertainty/entropy). This process is applied recursively from the root-node down, and stops when a leaf node contains instances all having the same class (no need to split it further).

Note that I skipped over some details which are beyond the scope of this post, including how to handle numeric features, missing values, overfitting and pruning trees, etc..

Source of Information Gain Example > https://stackoverflow.com/questions/1859554/what-is-entropy-and-information-gain