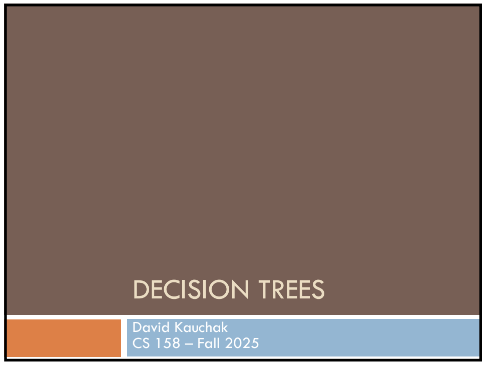
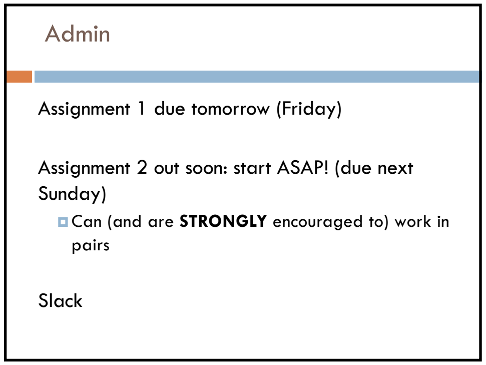


DECISION TREES

David Kauchak
CS 158 – Fall 2025

1



Admin

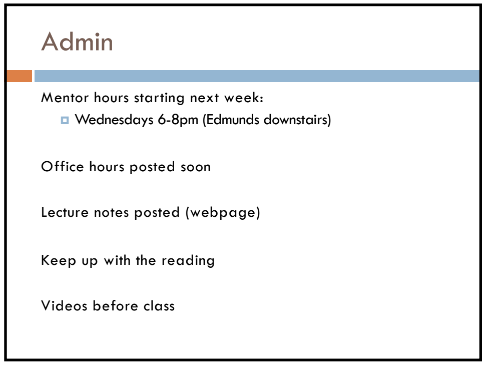
Assignment 1 due tomorrow (Friday)

Assignment 2 out soon: start ASAP! (due next Sunday)

- Can (and are **STRONGLY** encouraged to) work in pairs

Slack

2



Admin

Mentor hours starting next week:

- Wednesdays 6-8pm (Edmunds downstairs)

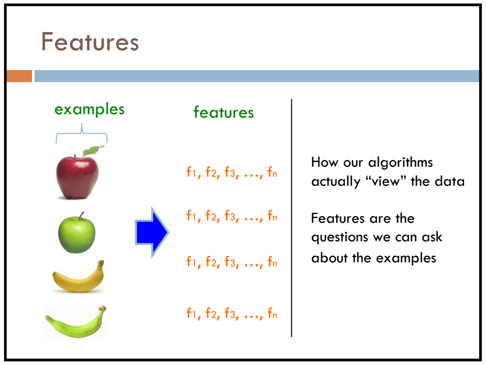
Office hours posted soon

Lecture notes posted (webpage)

Keep up with the reading

Videos before class

3



Features

examples

features

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

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

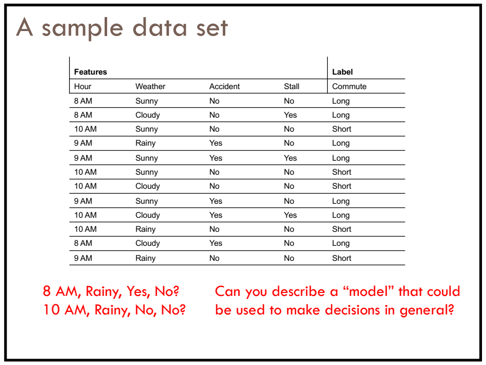
$f_1, f_2, f_3, \ldots, f_n$

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

How our algorithms actually "view" the data

Features are the questions we can ask about the examples

4



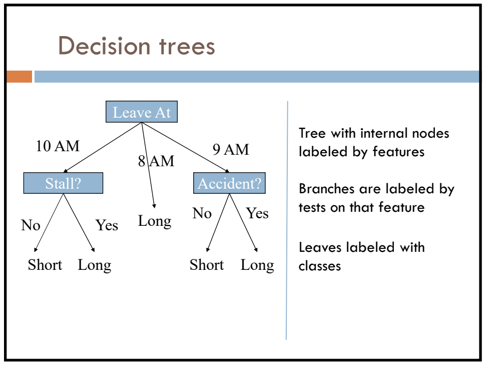
A sample data set

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?

5



Decision trees

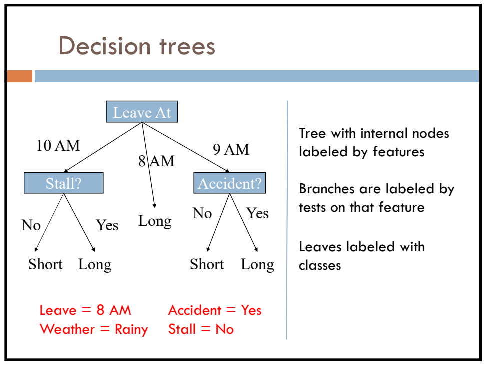
Leave At
10 AM
8 AM
9 AM
Stall?
Accident?
No
Yes
Long
No
Yes
Short
Long
Short
Long

Tree with internal nodes labeled by features

Branches are labeled by tests on that feature

Leaves labeled with classes

6



Decision trees

Leave At
10 AM
8 AM
9 AM
Stall?
Accident?
No
Yes
Long
No
Yes
Short
Long
Short
Long

Tree with internal nodes labeled by features

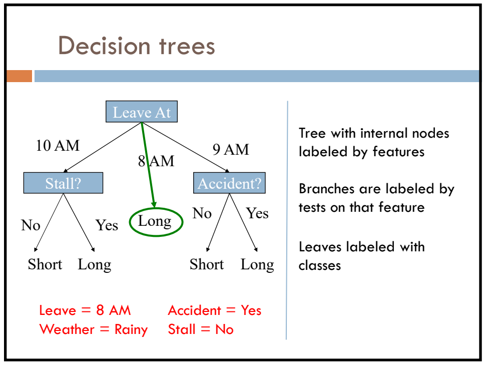
Branches are labeled by tests on that feature

Leaves labeled with classes

Leave = 8 AM
Weather = Rainy

Accident = Yes
Stall = No

7



Decision trees

Leave At
10 AM
8 AM
9 AM
Stall?
Accident?
No
Yes
Long
No
Yes
Short
Long
Short
Long

Tree with internal nodes labeled by features

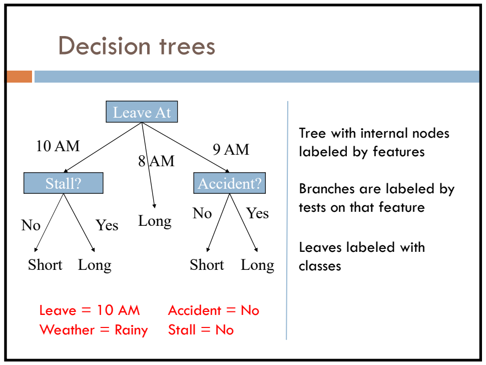
Branches are labeled by tests on that feature

Leaves labeled with classes

Leave = 8 AM
Weather = Rainy

Accident = Yes
Stall = No

8



Decision trees

Leave At

10 AM

8 AM

9 AM

Stall?

Accident?

No

Yes

Long

No

Yes

Short

Long

Short

Long

Tree with internal nodes labeled by features

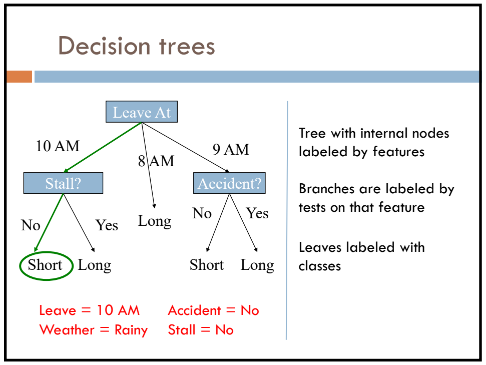
Branches are labeled by tests on that feature

Leaves labeled with classes

Leave = 10 AM
Weather = Rainy

Accident = No
Stall = No

9



Decision trees

Leave At

10 AM

8 AM

9 AM

Stall?

Accident?

No

Yes

Long

No

Yes

Short

Long

Short

Long

Tree with internal nodes labeled by features

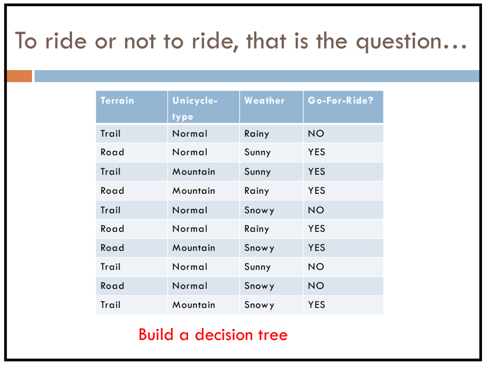
Branches are labeled by tests on that feature

Leaves labeled with classes

Leave = 10 AM
Weather = Rainy

Accident = No
Stall = No

10

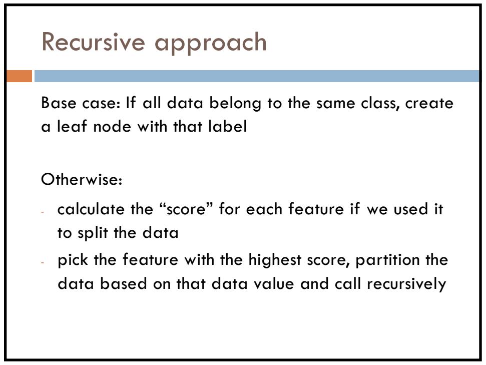


To ride or not to ride, that is the question…

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

11



Recursive approach

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

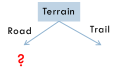
Otherwise:

- 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

12

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


Terrain

Road

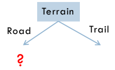
Trail

?

13

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


Terrain

Road

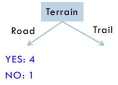
Trail

?

14

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


Terrain

Road

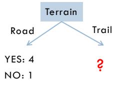
Trail

YES: 4
NO: 1

15

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


Terrain

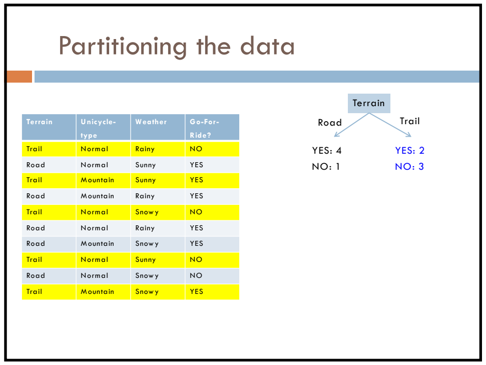
Road

Trail

YES: 4
NO: 1

?

16



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

Terrain

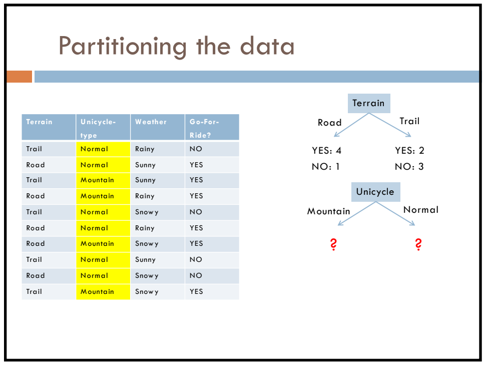
Road

Trail

YES: 4
NO: 1

YES: 2
NO: 3

17



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

Terrain

Road

Trail

YES: 4
NO: 1

YES: 2
NO: 3

Unicycle

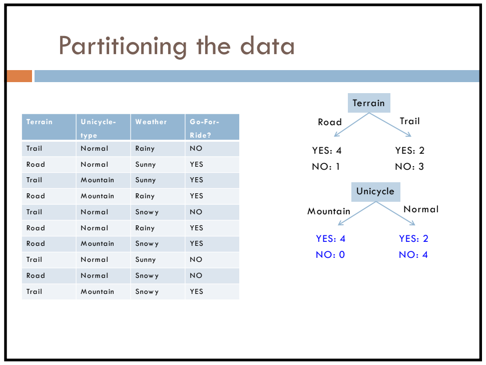
Mountain

Normal

?

?

18



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

Terrain

Road

Trail

YES: 4
NO: 1

YES: 2
NO: 3

Unicycle

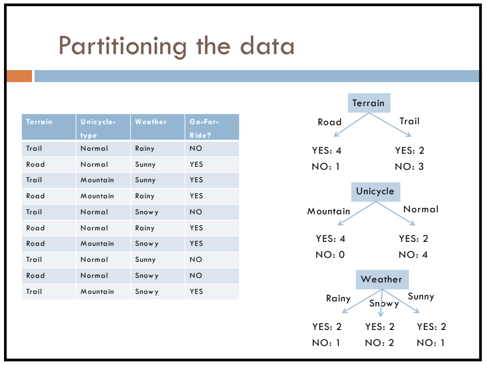
Mountain

Normal

YES: 4
NO: 0

YES: 2
NO: 4

19



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

Terrain

Road

Trail

YES: 4
NO: 1

YES: 2
NO: 3

Unicycle

Mountain

Normal

YES: 4
NO: 0

YES: 2
NO: 4

Weather

Rainy

Snowy

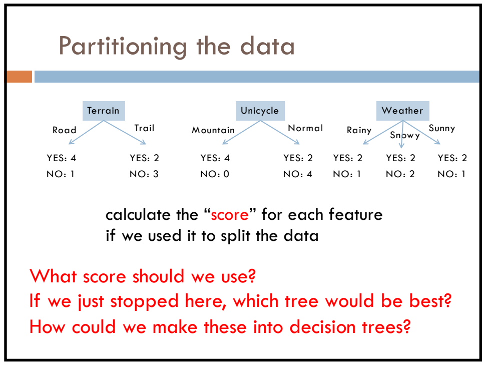
Sunny

YES: 2
NO: 1

YES: 2
NO: 2

YES: 2
NO: 1

20

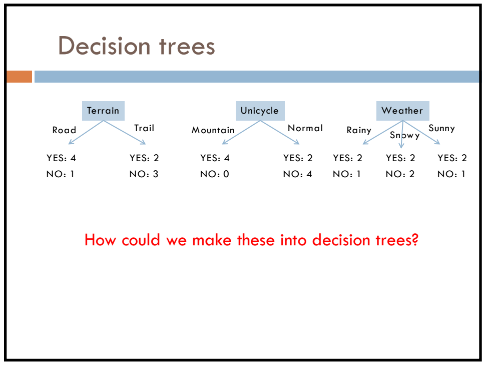


Partitioning the data

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?

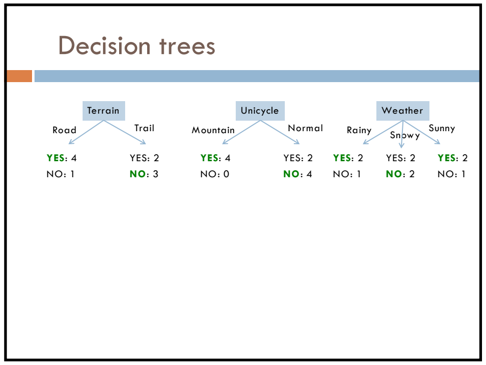
21



Decision trees

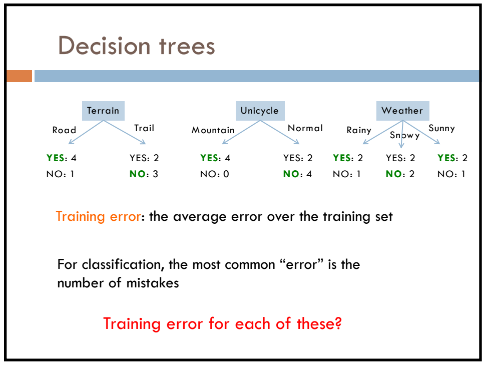
How could we make these into decision trees?

22



Decision trees

23



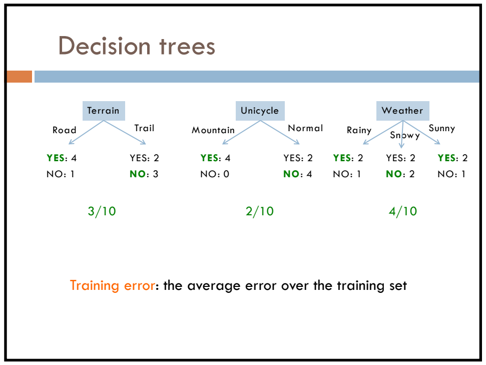
Decision trees

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?

24



Decision trees

Terrain: Road → YES: 4, NO: 1; Trail → YES: 2, NO: 3

3/10

Unicycle: Mountain → YES: 4, NO: 0; Normal → YES: 2, NO: 4

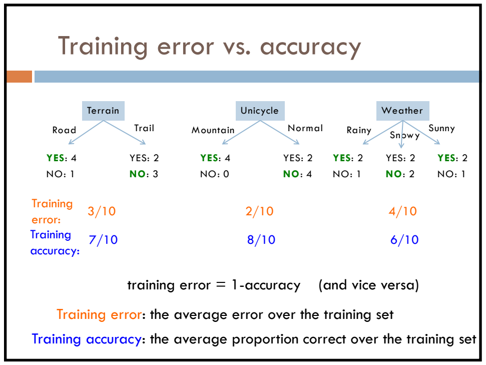
2/10

Weather: Rainy → YES: 2, NO: 1; Snowy → YES: 2, NO: 2; Sunny → YES: 2, NO: 1

4/10

Training error: the average error over the training set

25



Training error vs. accuracy

Terrain: Road → YES: 4, NO: 1; Trail → YES: 2, NO: 3

Unicycle: Mountain → YES: 4, NO: 0; Normal → YES: 2, NO: 4

Weather: Rainy → YES: 2, NO: 1; Snowy → YES: 2, NO: 2; Sunny → YES: 2, NO: 1

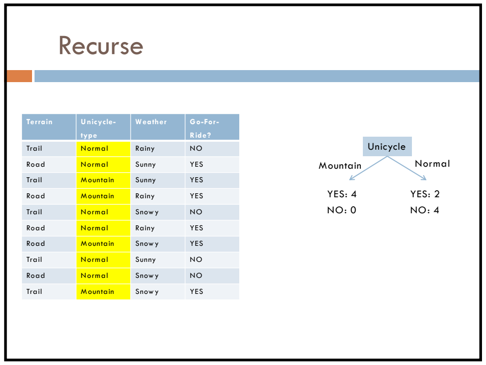
	Terrain	Unicycle	Weather
Training error:	3/10	2/10	4/10
Training accuracy:	7/10	8/10	6/10

training error = 1-accuracy (and vice versa)

Training error: the average error over the training set

Training accuracy: the average proportion correct over the training set

26

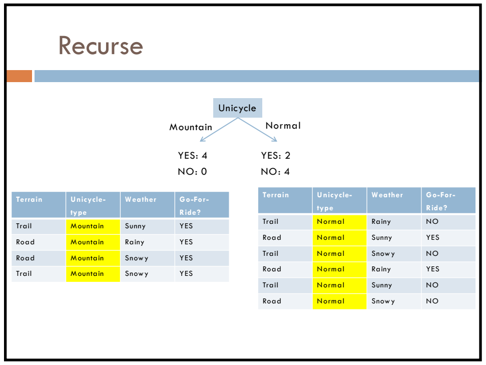


Recurse

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

Unicycle: Mountain → YES: 4, NO: 0; Normal → YES: 2, NO: 4

27



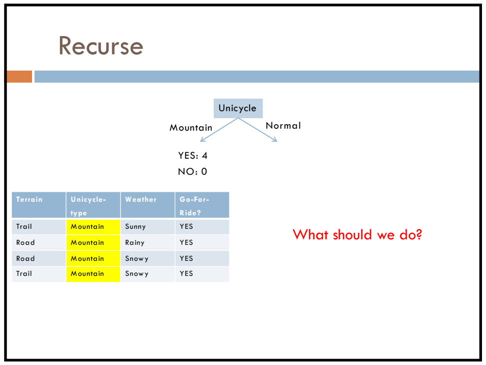
Recurse

Unicycle: Mountain → YES: 4, NO: 0; Normal → YES: 2, NO: 4

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

28



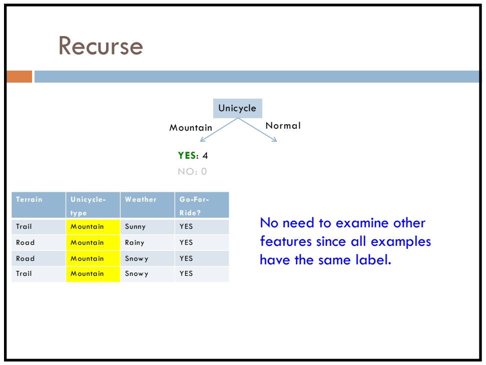
Recurse

Unicycle
Mountain
Normal
YES: 4
NO: 0

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

What should we do?

29



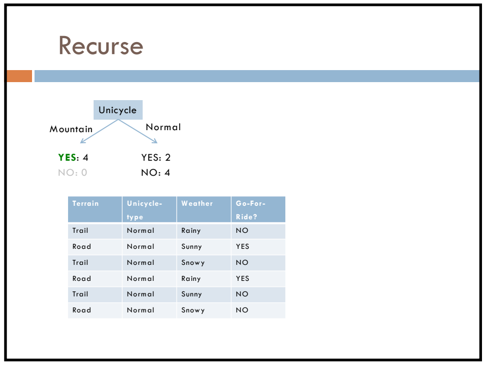
Recurse

Unicycle
Mountain
Normal
YES: 4
NO: 0

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

No need to examine other features since all examples have the same label.

30

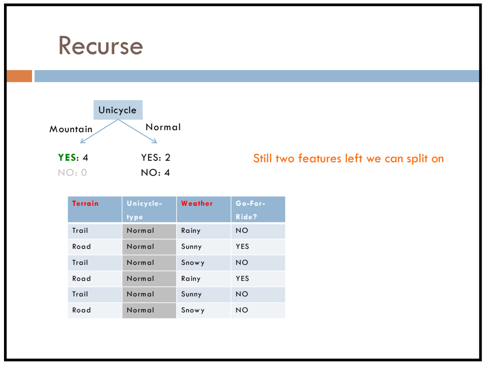


Recurse

Unicycle
Mountain
Normal
YES: 4
NO: 0
YES: 2
NO: 4

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

31



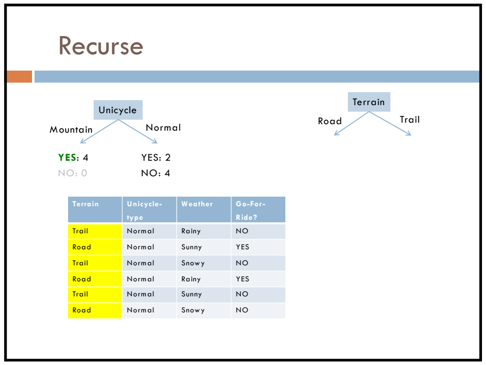
Recurse

Unicycle
Mountain
Normal
YES: 4
NO: 0
YES: 2
NO: 4

Still two features left we can split on

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

32



Recurse

Unicycle

Mountain — YES: 4, NO: 0

Normal — YES: 2, NO: 4

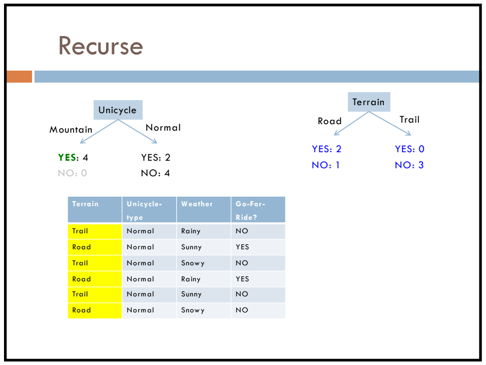
Terrain

Road

Trail

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

33



Recurse

Unicycle

Mountain — YES: 4, NO: 0

Normal — YES: 2, NO: 4

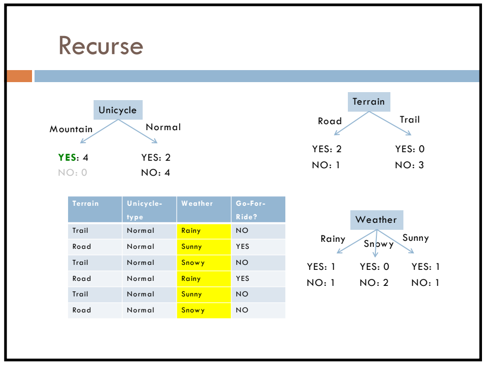
Terrain

Road — YES: 2, NO: 1

Trail — YES: 0, NO: 3

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

34



Recurse

Unicycle

Mountain — YES: 4, NO: 0

Normal — YES: 2, NO: 4

Terrain

Road — YES: 2, NO: 1

Trail — YES: 0, NO: 3

Weather

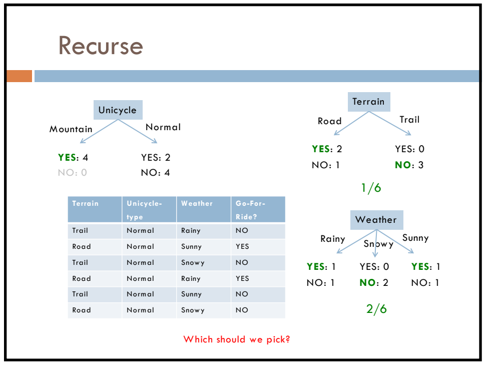
Rainy — YES: 1, NO: 1

Snowy — YES: 0, NO: 2

Sunny — YES: 1, NO: 1

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

35



Recurse

Unicycle

Mountain — YES: 4, NO: 0

Normal — YES: 2, NO: 4

Terrain

Road — YES: 2, NO: 1

Trail — YES: 0, NO: 3

1/6

Weather

Rainy — YES: 1, NO: 1

Snowy — YES: 0, NO: 2

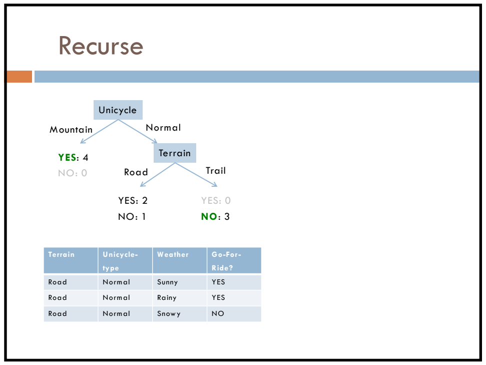
Sunny — YES: 1, NO: 1

2/6

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

Which should we pick?

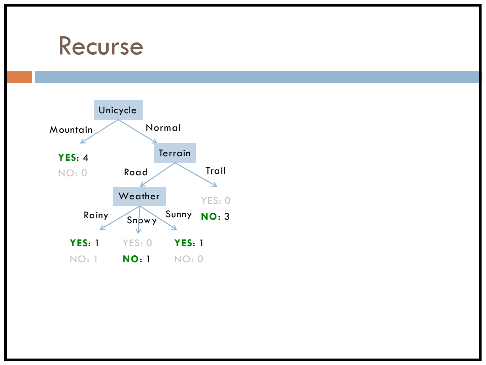
36



Recurse

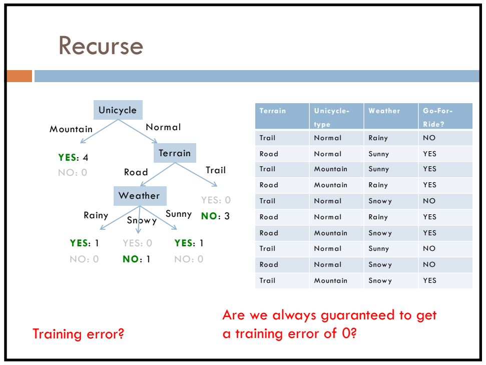
Terrain	Unicycle-type	Weather	Go-For-Ride?
Road	Normal	Sunny	YES
Road	Normal	Rainy	YES
Road	Normal	Snowy	NO

37



Recurse

38



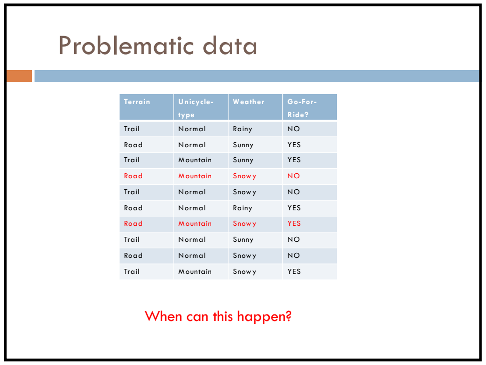
Recurse

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

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

Training error?

39



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?

40

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?

41

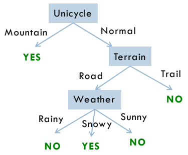
What would the tree look like for…

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

42

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

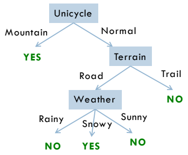


Is that what you would do?

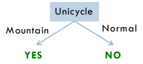
43

What would the tree look like for…

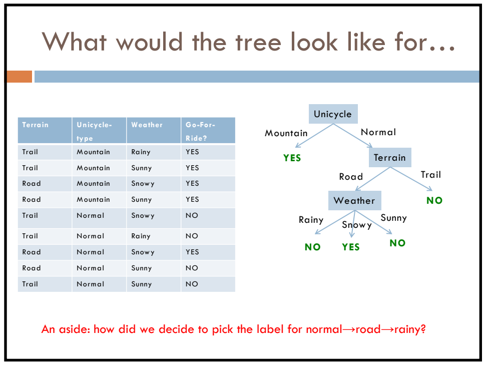
Terrain	Unicycle-type	Weather	Go-For-Ride?
Trail	Mountain	Rainy	YES
Trail	Mountain	Sunny	YES
Road	Mountain	Snowy	YES
Road	Mountain	Sunny	YES
Trail	Normal	Snowy	NO
Trail	Normal	Rainy	NO
Road	Normal	Snowy	YES
Road	Normal	Sunny	NO
Trail	Normal	Sunny	NO



Maybe…



44



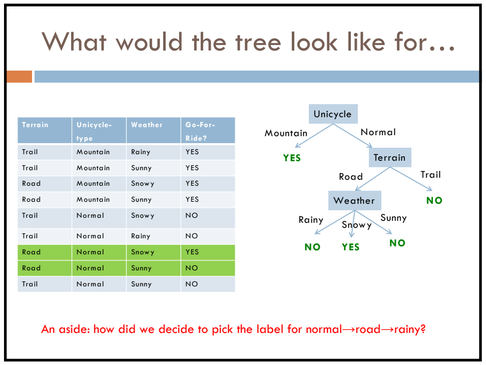
What would the tree look like for…

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

Unicycle
Mountain
Normal
YES
Terrain
Road
Trail
Weather
NO
Rainy
Snowy
Sunny
NO
YES
NO

An aside: how did we decide to pick the label for normal→road→rainy?

45



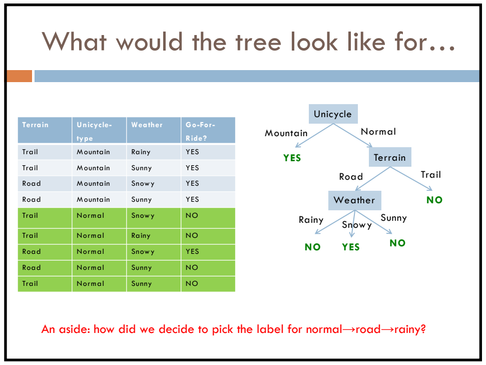
What would the tree look like for…

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

Unicycle
Mountain
Normal
YES
Terrain
Road
Trail
Weather
NO
Rainy
Snowy
Sunny
NO
YES
NO

An aside: how did we decide to pick the label for normal→road→rainy?

46



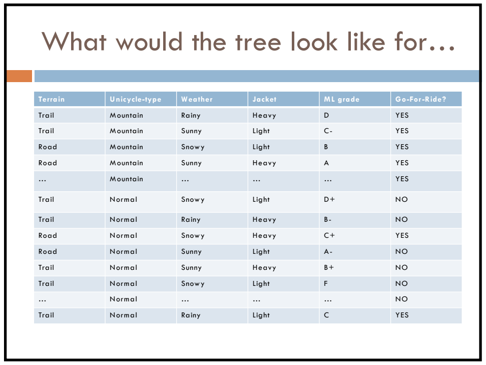
What would the tree look like for…

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

Unicycle
Mountain
Normal
YES
Terrain
Road
Trail
Weather
NO
Rainy
Snowy
Sunny
NO
YES
NO

An aside: how did we decide to pick the label for normal→road→rainy?

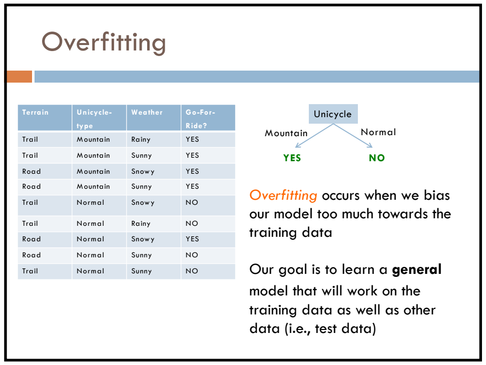
47



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

48



Overfitting

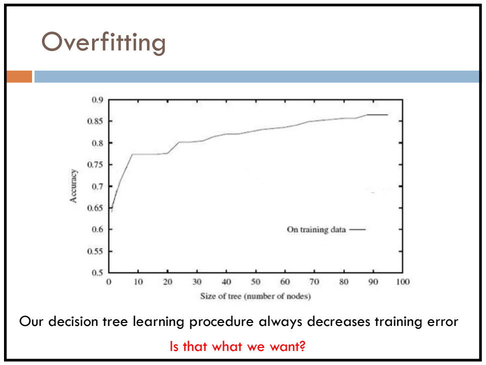
Terrain	Unicycle-type	Weather	Go-For-Ride?
Trail	Mountain	Rainy	YES
Trail	Mountain	Sunny	YES
Road	Mountain	Snowy	YES
Road	Mountain	Sunny	YES
Trail	Normal	Snowy	NO
Trail	Normal	Rainy	NO
Road	Normal	Snowy	YES
Road	Normal	Sunny	NO
Trail	Normal	Sunny	NO

Unicycle
Mountain
Normal
YES
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)

49

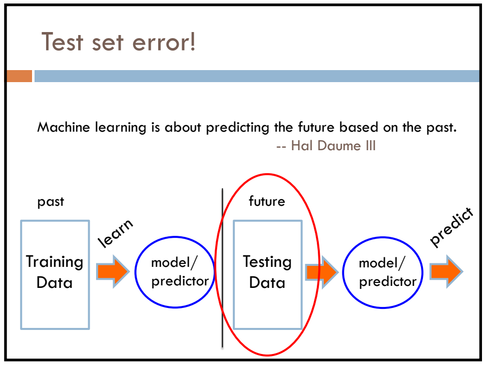


Overfitting

Our decision tree learning procedure always decreases training error

Is that what we want?

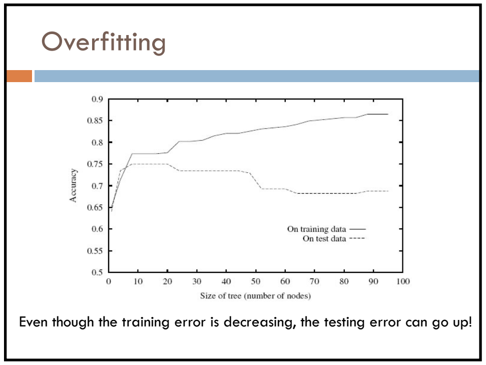
50



Test set error!

Machine learning is about predicting the future based on the past.
-- Hal Daume III

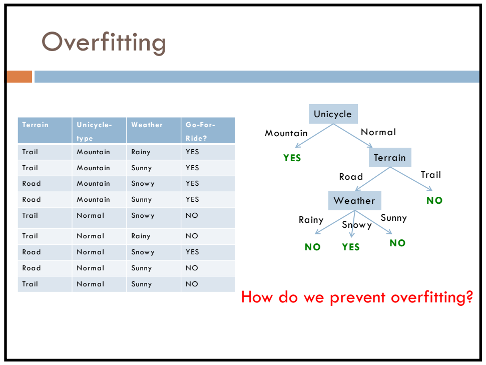
51



Overfitting

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

52



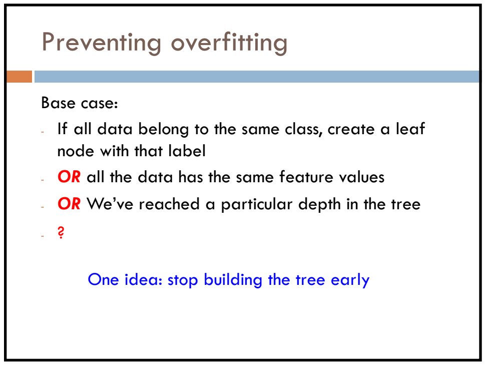
Overfitting

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

Unicycle
Mountain
Normal
YES
Terrain
Road
Trail
NO
Weather
Rainy
Snowy
Sunny
NO
YES
NO

How do we prevent overfitting?

53



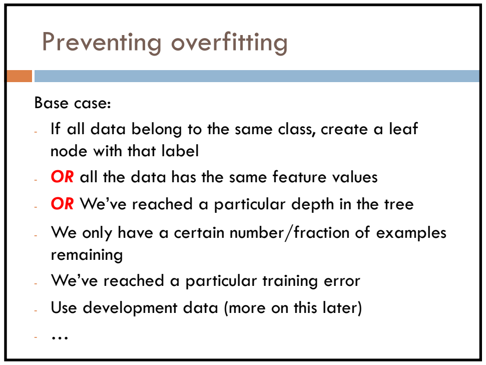
Preventing overfitting

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

54

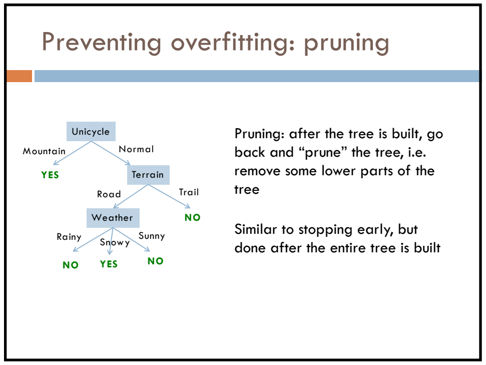


Preventing overfitting

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)
- …

55



Preventing overfitting: pruning

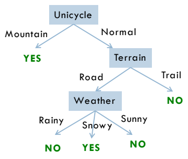
Unicycle
Mountain
Normal
YES
Terrain
Road
Trail
NO
Weather
Rainy
Snowy
Sunny
NO
YES
NO

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

56

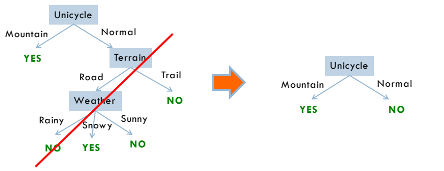
Preventing overfitting: pruning



Build the full tree

57

Preventing overfitting: pruning

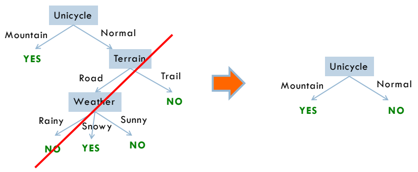


Build the full tree

Prune back leaves that are too specific

58

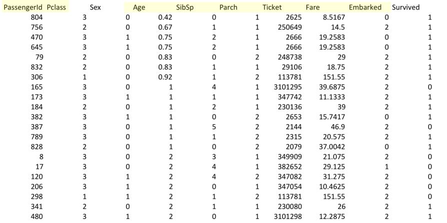
Preventing overfitting: pruning



Pruning criterion?

59

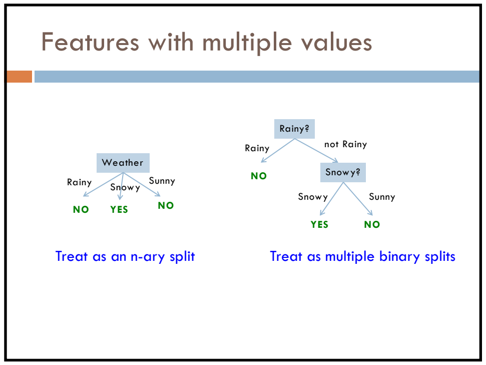
Handling non-binary attributes



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?

60

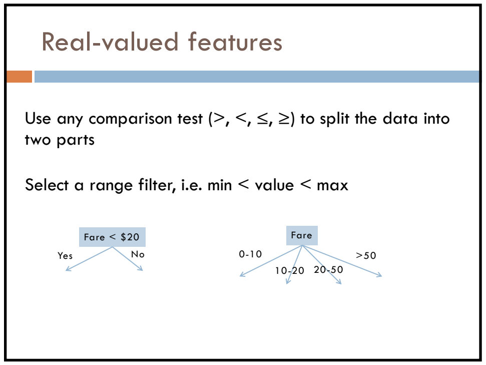


Features with multiple values

Treat as an n-ary split

Treat as multiple binary splits

61

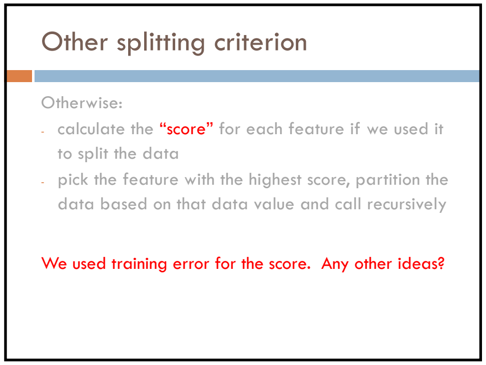


Real-valued features

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

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

62



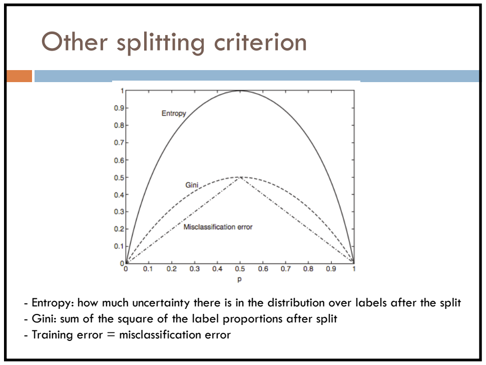
Other splitting criterion

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?

63



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

64

Decision trees

Good? Bad?

65

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

66

Decision trees: the bad

Be careful with features with lots of values if you're not doing binary splits

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

Which feature would be at the top here?

67

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

68

Final DT algorithm

DT_train(data):

Base cases:

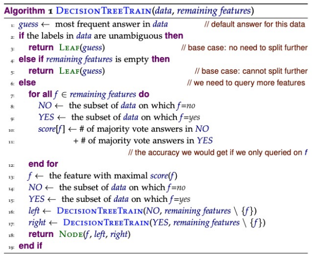
1. If all data belong to the same class, pick that label
2. If all the data have the same feature values, pick majority label (if tie, parent majority)
3. If we're out of features to examine, pick majority label (if tie, parent majority)
4. If the we don't have any data left, pick majority label of *parent*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

Otherwise (i.e. if none of the base cases apply):

- 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, e.g. data_left and data_right
- Recurse, i.e. DT_train(data_left) and DT_train(data_right)
- Make tree with feature as the splitting criterion with the decision trees returned from the recursive calls as the children

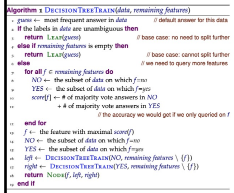
69

Pseudocode (from the book)



```
Algorithm 1 DecisionTreeTrain(data, remaining features)
guess ← most frequent answer in data                 // default answer for this data
if the labels in data are unambiguous then
   return Leaf(guess)                                // base case: no need to split further
else if remaining features is empty then
   return Leaf(guess)                                // base case: cannot split further
else                                                 // we need to query more features
   for all $f \in$ remaining features do
      NO ← the subset of data on which $f$=no
      YES ← the subset of data on which $f$=yes
      score[$f$] ← # of majority vote answers in NO
                  + # of majority vote answers in YES
                               // the accuracy we would get if we only queried on f
   end for
   $f$ ← the feature with maximal score($f$)
   NO ← the subset of data on which $f$=no
   YES ← the subset of data on which $f$=yes
   left ← DecisionTreeTrain(NO, remaining features $\setminus \{f\}$)
   right ← DecisionTreeTrain(YES, remaining features $\setminus \{f\}$)
   return Node($f$, left, right)
end if
```

70

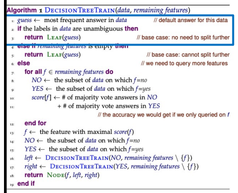


```
Algorithm 1 DecisionTreeTrain(data, remaining features)
guess ← most frequent answer in data                 // default answer for this data
if the labels in data are unambiguous then
   return Leaf(guess)                                // base case: no need to split further
else if remaining features is empty then
   return Leaf(guess)                                // base case: cannot split further
else                                                 // we need to query more features
   for all $f \in$ remaining features do
      NO ← the subset of data on which $f$=no
      YES ← the subset of data on which $f$=yes
      score[$f$] ← # of majority vote answers in NO
                  + # of majority vote answers in YES
                               // the accuracy we would get if we only queried on f
   end for
   $f$ ← the feature with maximal score($f$)
   NO ← the subset of data on which $f$=no
   YES ← the subset of data on which $f$=yes
   left ← DecisionTreeTrain(NO, remaining features $\setminus \{f\}$)
   right ← DecisionTreeTrain(YES, remaining features $\setminus \{f\}$)
   return Node($f$, left, right)
end if
```

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 (if tie, parent majority)
3. If we're out of features to examine, pick majority label (if tie, parent majority)
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

71



```
Algorithm 1 DecisionTreeTrain(data, remaining features)
guess ← most frequent answer in data                 // default answer for this data
if the labels in data are unambiguous then
   return Leaf(guess)                                // base case: no need to split further
else if remaining features is empty then
   return Leaf(guess)                                // base case: cannot split further
else                                                 // we need to query more features
   for all $f \in$ remaining features do
      NO ← the subset of data on which $f$=no
      YES ← the subset of data on which $f$=yes
      score[$f$] ← # of majority vote answers in NO
                  + # of majority vote answers in YES
                               // the accuracy we would get if we only queried on f
   end for
   $f$ ← the feature with maximal score($f$)
   NO ← the subset of data on which $f$=no
   YES ← the subset of data on which $f$=yes
   left ← DecisionTreeTrain(NO, remaining features $\setminus \{f\}$)
   right ← DecisionTreeTrain(YES, remaining features $\setminus \{f\}$)
   return Node($f$, left, right)
end if
```

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 (if tie, parent majority)
3. If we're out of features to examine, pick majority label (if tie, parent majority)
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

72

Algorithm 1 DecisionTreeTrain(*data, remaining features*)

```
guess ← most frequent answer in data                 // default answer for this data
if the labels in data are unambiguous then
    return Leaf(guess)                                // base case: no need to split further
else if remaining features is empty then
    return Leaf(guess)                                // base case: cannot split further
else                                                  // we need to query more features
    for all f ∈ remaining features do
        NO ← the subset of data on which f=no
        YES ← the subset of data on which f=yes
        score[f] ← # of majority vote answers in NO
                     + # of majority vote answers in YES
                                   // the accuracy we would get if we only queried on f
    end for
    f ← the feature with maximal score(f)
    NO ← the subset of data on which f=no
    YES ← the subset of data on which f=yes
    left ← DecisionTreeTrain(NO, remaining features \ {f})
    right ← DecisionTreeTrain(YES, remaining features \ {f})
    return Node(f, left, right)
end if
```

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 (if tie, parent majority)
3. If we're out of features to examine, pick majority label (if tie, parent majority)
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

73

Algorithm 1 DecisionTreeTrain(*data, remaining features*)

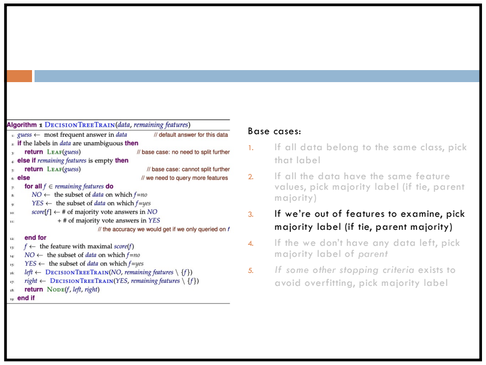
```
guess ← most frequent answer in data                 // default answer for this data
if the labels in data are unambiguous then
    return Leaf(guess)                                // base case: no need to split further
else if remaining features is empty then
    return Leaf(guess)                                // base case: cannot split further
else                                                  // we need to query more features
    for all f ∈ remaining features do
        NO ← the subset of data on which f=no
        YES ← the subset of data on which f=yes
        score[f] ← # of majority vote answers in NO
                     + # of majority vote answers in YES
                                   // the accuracy we would get if we only queried on f
    end for
    f ← the feature with maximal score(f)
    NO ← the subset of data on which f=no
    YES ← the subset of data on which f=yes
    left ← DecisionTreeTrain(NO, remaining features \ {f})
    right ← DecisionTreeTrain(YES, remaining features \ {f})
    return Node(f, left, right)
end if
```

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 (if tie, parent majority)
3. If we're out of features to examine, pick majority label (if tie, parent majority)
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

Not in this pseudocode!

74



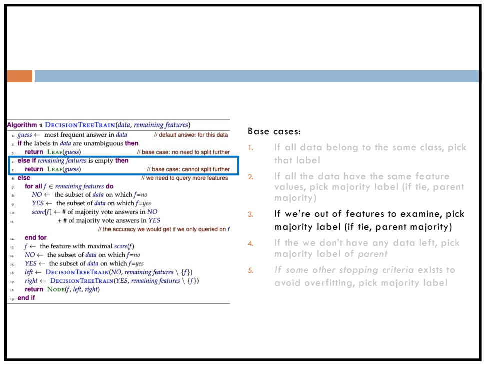
Algorithm 1 DecisionTreeTrain(*data, remaining features*)

```
guess ← most frequent answer in data                 // default answer for this data
if the labels in data are unambiguous then
    return Leaf(guess)                                // base case: no need to split further
else if remaining features is empty then
    return Leaf(guess)                                // base case: cannot split further
else                                                  // we need to query more features
    for all f ∈ remaining features do
        NO ← the subset of data on which f=no
        YES ← the subset of data on which f=yes
        score[f] ← # of majority vote answers in NO
                     + # of majority vote answers in YES
                                   // the accuracy we would get if we only queried on f
    end for
    f ← the feature with maximal score(f)
    NO ← the subset of data on which f=no
    YES ← the subset of data on which f=yes
    left ← DecisionTreeTrain(NO, remaining features \ {f})
    right ← DecisionTreeTrain(YES, remaining features \ {f})
    return Node(f, left, right)
end if
```

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 (if tie, parent majority)
3. If we're out of features to examine, pick majority label (if tie, parent majority)
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

75



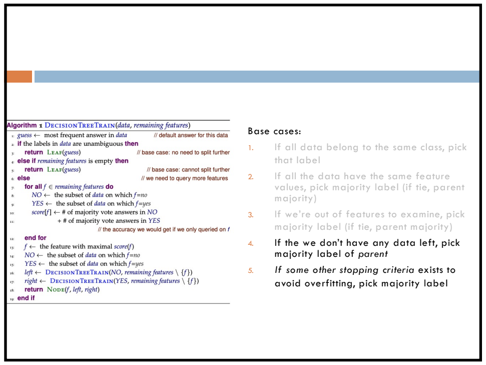
Algorithm 1 DecisionTreeTrain(*data, remaining features*)

```
guess ← most frequent answer in data                 // default answer for this data
if the labels in data are unambiguous then
    return Leaf(guess)                                // base case: no need to split further
else if remaining features is empty then
    return Leaf(guess)                                // base case: cannot split further
else                                                  // we need to query more features
    for all f ∈ remaining features do
        NO ← the subset of data on which f=no
        YES ← the subset of data on which f=yes
        score[f] ← # of majority vote answers in NO
                     + # of majority vote answers in YES
                                   // the accuracy we would get if we only queried on f
    end for
    f ← the feature with maximal score(f)
    NO ← the subset of data on which f=no
    YES ← the subset of data on which f=yes
    left ← DecisionTreeTrain(NO, remaining features \ {f})
    right ← DecisionTreeTrain(YES, remaining features \ {f})
    return Node(f, left, right)
end if
```

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 (if tie, parent majority)
3. If we're out of features to examine, pick majority label (if tie, parent majority)
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

76

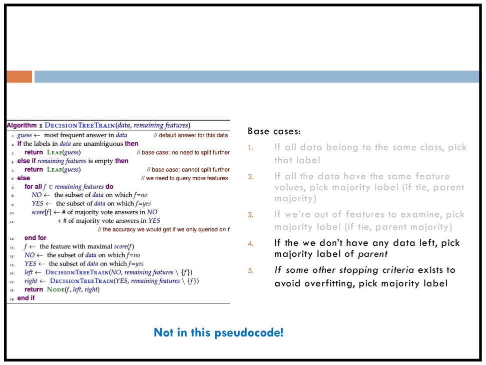


```
Algorithm 1 DecisionTreeTrain(data, remaining features)
guess ← most frequent answer in data                  // default answer for this data
if the labels in data are unambiguous then
    return Leaf(guess)                                // base case: no need to split further
else if remaining features is empty then
    return Leaf(guess)                                // base case: cannot split further
else                                                  // we need to query more features
    for all f ∈ remaining features do
        NO ← the subset of data on which f=no
        YES ← the subset of data on which f=yes
        score[f] ← # of majority vote answers in NO
                   + # of majority vote answers in YES
                                 // the accuracy we would get if we only queried on f
    end for
    f ← the feature with maximal score(f)
    NO ← the subset of data on which f=no
    YES ← the subset of data on which f=yes
    left ← DecisionTreeTrain(NO, remaining features \ {f})
    right ← DecisionTreeTrain(YES, remaining features \ {f})
    return Node(f, left, right)
end if
```

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 (if tie, parent majority)
3. If we're out of features to examine, pick majority label (if tie, parent majority)
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

77



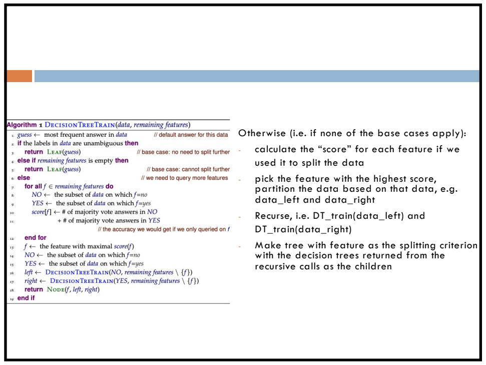
```
Algorithm 1 DecisionTreeTrain(data, remaining features)
guess ← most frequent answer in data                  // default answer for this data
if the labels in data are unambiguous then
    return Leaf(guess)                                // base case: no need to split further
else if remaining features is empty then
    return Leaf(guess)                                // base case: cannot split further
else                                                  // we need to query more features
    for all f ∈ remaining features do
        NO ← the subset of data on which f=no
        YES ← the subset of data on which f=yes
        score[f] ← # of majority vote answers in NO
                   + # of majority vote answers in YES
                                 // the accuracy we would get if we only queried on f
    end for
    f ← the feature with maximal score(f)
    NO ← the subset of data on which f=no
    YES ← the subset of data on which f=yes
    left ← DecisionTreeTrain(NO, remaining features \ {f})
    right ← DecisionTreeTrain(YES, remaining features \ {f})
    return Node(f, left, right)
end if
```

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 (if tie, parent majority)
3. If we're out of features to examine, pick majority label (if tie, parent majority)
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

Not in this pseudocode!

78

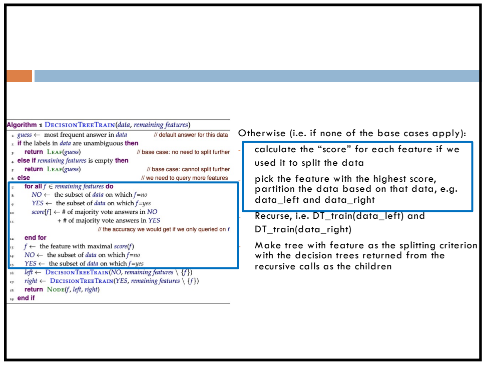


```
Algorithm 1 DecisionTreeTrain(data, remaining features)
guess ← most frequent answer in data                  // default answer for this data
if the labels in data are unambiguous then
    return Leaf(guess)                                // base case: no need to split further
else if remaining features is empty then
    return Leaf(guess)                                // base case: cannot split further
else                                                  // we need to query more features
    for all f ∈ remaining features do
        NO ← the subset of data on which f=no
        YES ← the subset of data on which f=yes
        score[f] ← # of majority vote answers in NO
                   + # of majority vote answers in YES
                                 // the accuracy we would get if we only queried on f
    end for
    f ← the feature with maximal score(f)
    NO ← the subset of data on which f=no
    YES ← the subset of data on which f=yes
    left ← DecisionTreeTrain(NO, remaining features \ {f})
    right ← DecisionTreeTrain(YES, remaining features \ {f})
    return Node(f, left, right)
end if
```

Otherwise (i.e. if none of the base cases apply):

- 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, e.g. data_left and data_right
- Recurse, i.e. DT_train(data_left) and DT_train(data_right)
- Make tree with feature as the splitting criterion with the decision trees returned from the recursive calls as the children

79

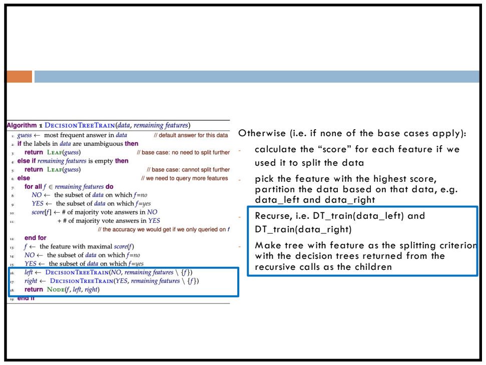


```
Algorithm 1 DecisionTreeTrain(data, remaining features)
guess ← most frequent answer in data                  // default answer for this data
if the labels in data are unambiguous then
    return Leaf(guess)                                // base case: no need to split further
else if remaining features is empty then
    return Leaf(guess)                                // base case: cannot split further
else                                                  // we need to query more features
    for all f ∈ remaining features do
        NO ← the subset of data on which f=no
        YES ← the subset of data on which f=yes
        score[f] ← # of majority vote answers in NO
                   + # of majority vote answers in YES
                                 // the accuracy we would get if we only queried on f
    end for
    f ← the feature with maximal score(f)
    NO ← the subset of data on which f=no
    YES ← the subset of data on which f=yes
    left ← DecisionTreeTrain(NO, remaining features \ {f})
    right ← DecisionTreeTrain(YES, remaining features \ {f})
    return Node(f, left, right)
end if
```

Otherwise (i.e. if none of the base cases apply):

- 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, e.g. data_left and data_right
- Recurse, i.e. DT_train(data_left) and DT_train(data_right)
- Make tree with feature as the splitting criterion with the decision trees returned from the recursive calls as the children

80

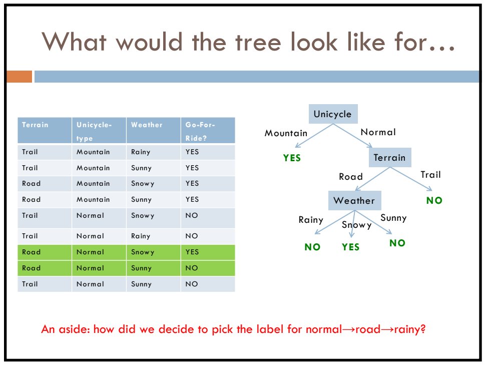


```
Algorithm 1 DecisionTreeTrain(data, remaining features)
guess ← most frequent answer in data                  // default answer for this data
if the labels in data are unambiguous then
   return Leaf(guess)                                 // base case: no need to split further
else if remaining features is empty then
   return Leaf(guess)                                 // base case: cannot split further
else                                                  // we need to query more features
   for all $f \in$ remaining features do
      NO ← the subset of data on which $f$=no
      YES ← the subset of data on which $f$=yes
      score[$f$] ← # of majority vote answers in NO
                  + # of majority vote answers in YES
                              // the accuracy we would get if we only queried on $f$
   end for
   $f$ ← the feature with maximal score($f$)
   NO ← the subset of data on which $f$=no
   YES ← the subset of data on which $f$=yes
   left ← DecisionTreeTrain(NO, remaining features \ {$f$})
   right ← DecisionTreeTrain(YES, remaining features \ {$f$})
   return Node($f$, left, right)
end if
```

Otherwise (i.e. if none of the base cases apply):

- 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, e.g. data_left and data_right
- Recurse, i.e. DT_train(data_left) and DT_train(data_right)
- Make tree with feature as the splitting criterion with the decision trees returned from the recursive calls as the children

81

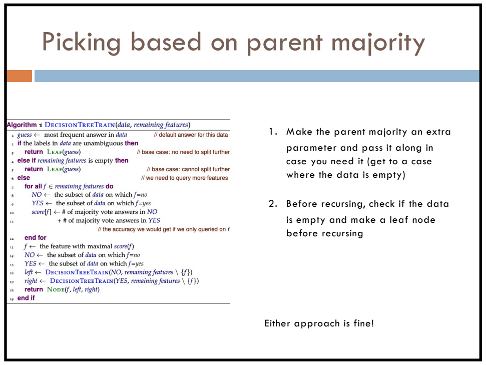


What would the tree look like for…

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

An aside: how did we decide to pick the label for normal→road→rainy?

82



Picking based on parent majority

```
Algorithm 1 DecisionTreeTrain(data, remaining features)
guess ← most frequent answer in data                  // default answer for this data
if the labels in data are unambiguous then
   return Leaf(guess)                                 // base case: no need to split further
else if remaining features is empty then
   return Leaf(guess)                                 // base case: cannot split further
else                                                  // we need to query more features
   for all $f \in$ remaining features do
      NO ← the subset of data on which $f$=no
      YES ← the subset of data on which $f$=yes
      score[$f$] ← # of majority vote answers in NO
                  + # of majority vote answers in YES
                              // the accuracy we would get if we only queried on $f$
   end for
   $f$ ← the feature with maximal score($f$)
   NO ← the subset of data on which $f$=no
   YES ← the subset of data on which $f$=yes
   left ← DecisionTreeTrain(NO, remaining features \ {$f$})
   right ← DecisionTreeTrain(YES, remaining features \ {$f$})
   return Node($f$, left, right)
end if
```

1. Make the parent majority an extra parameter and pass it along in case you need it (get to a case where the data is empty)
2. Before recursing, check if the data is empty and make a leaf node before recursing

Either approach is fine!

83