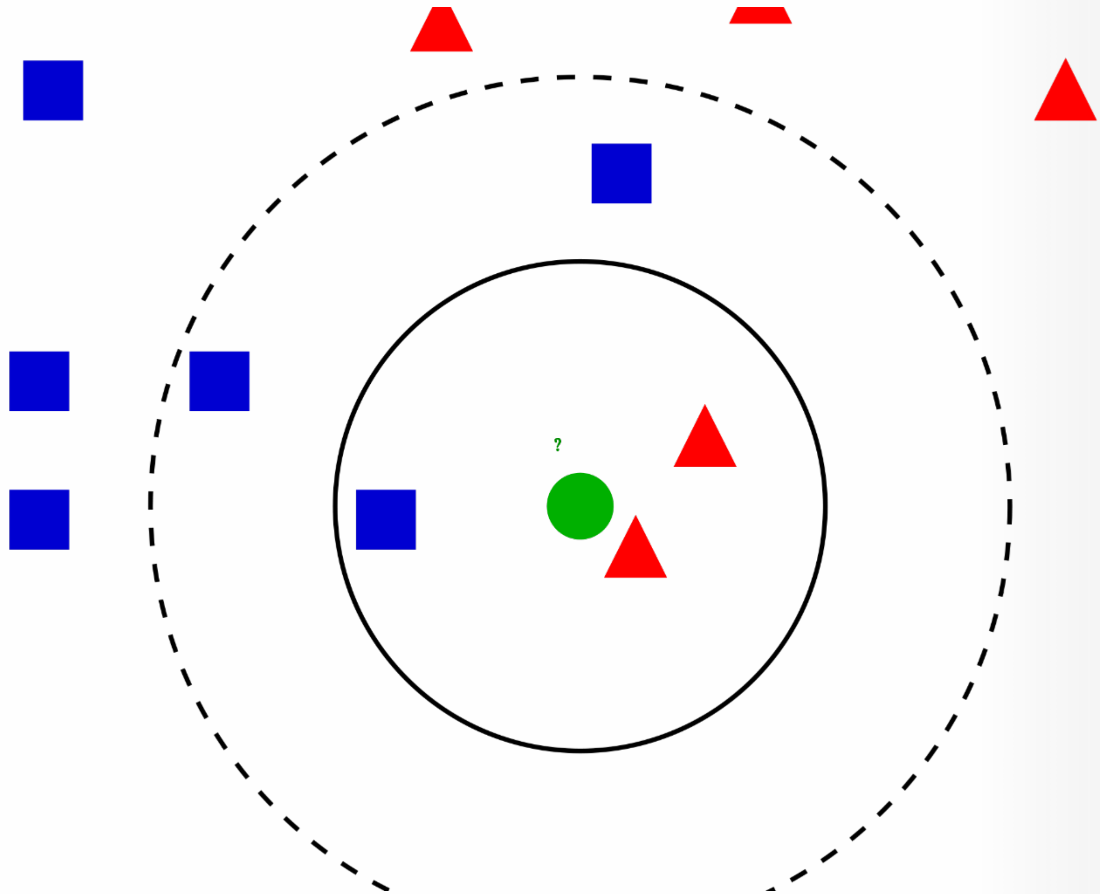
CLASSIFIER NEAREST NEIGHBOR AND DECISION TREE

許志仲 (Chih-Chung Hsu)

Assistant Professor
ACVLab, Institute of Data Science
National Cheng Kung University
https://cchsu.info



Outline

- Introduction to Machine Learning
- Supervised learning
  - Nearest neighbor classifier
  - Decision tree
  - Bayer classifier
    - Two-dim data
    - Multivariant data
  - Support vector machine
  - Ensemble and boosting
- Feature selection
  - PCA and LDA
- Unsupervised learning
  - K-means
  - EM-algorithm
  - Affinity-propagation

Outline

- Introduction to Machine Learning
- Supervised learning
  - Nearest neighbor classifier
  - Decision tree
  - Bayer classifier
    - Two-dim data
    - Multivariant data
  - Support vector machine
  - Ensemble and boosting
- Feature selection
  - PCA and LDA
- Unsupervised learning
  - K-means
  - EM-algorithm
  - Affinity-propagation

Prior vs. Posteriori Knowledge

- To solve a problem, we need an algorithm
  - E.g., sorting
  - A priori knowledge is enough
- For some problem, however, we do not have the a priori knowledge
  - E.g., to tell if an email is spam or not
- The correct answer varies in time and from person to person

Prior vs. Posteriori Knowledge

- To solve a problem, we need an algorithm
  - E.g., sorting
  - A priori knowledge is enough
- For some problem, however, we do not have the a priori knowledge
  - E.g., to tell if an email is spam or not
- The correct answer varies in time and from person to person
- Machine learning algorithms use the a posteriori knowledge to solve problems
- Learnt from examples (as extra input)

Supervised Learning

- Unsupervised:

$$\mathbb{X} = \{\boldsymbol{x}^{(i)}\}_{i=1}^{N}, \text{ where } \boldsymbol{x}^{(i)} \in \mathbb{R}^D$$

  - E.g., $\boldsymbol{x}^{(i)}$ an email
- Supervised:

$$\mathbb{X} = \{(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})\}_{i=1}^{N}, \text{ where } \boldsymbol{x}^{(i)} \in \mathbb{R}^D \text{ and } \boldsymbol{y}^{(i)} \in \mathbb{R}^K,$$

  - E.g., $y^{(i)} \in \{0, 1\}$ a spam label

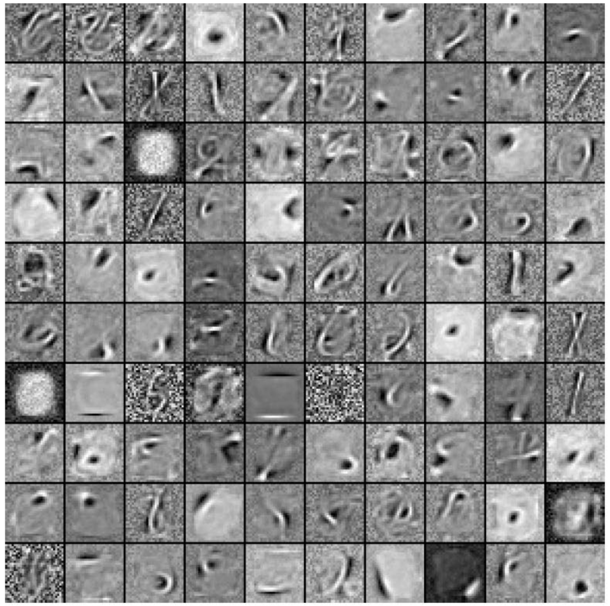
General Type of ML

Supervised learning: learn to predict the labels of future data points

$X \in \mathbb{R}^{N \times D}$:  $\quad$ $\boldsymbol{x}' \in \mathbb{R}^{D}$: 

$\boldsymbol{y} \in \mathbb{R}^{N \times K}$: $[\boldsymbol{e}^{(6)}, \boldsymbol{e}^{(1)}, \boldsymbol{e}^{(9)}, \boldsymbol{e}^{(4)}, \boldsymbol{e}^{(2)}]$ $\quad$ $\boldsymbol{y}' \in \mathbb{R}^{K}$: ?

Unsupervised learning: learn patterns or latent factors in $\boldsymbol{X}$



Steps of Machine Learning

1. Data collection, preprocessing (e.g., integration, cleaning, etc.), and exploration
   1. Split a dataset into the training and testing datasets

Spam Detection as an Example

1. Random split of your past emails and labels
   1. Training dataset: $\mathbb{X} = \{(\boldsymbol{x}^{(i)}, y^{(i)})\}_i$
   2. Testing dataset: $\mathbb{X}' = \{(\boldsymbol{x}'^{(i)}, y'^{(i)})\}_i$

NEAREST NEIGHBOR SEARCH

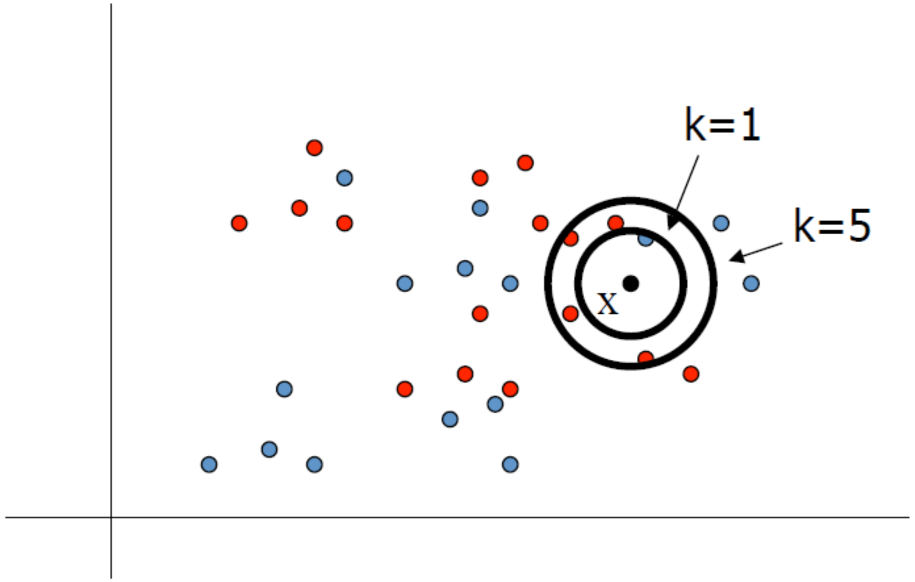
The simplest one!

Nearest Neighbor (NN) Algorithm

- Algorithm Learning Algorithm:
  - Store training examples
- Prediction Algorithm:
  - To classify a new example x by finding the training example (x^i, y^i) that is nearest to x
  - Guess the class $y = y^i$

NN Method

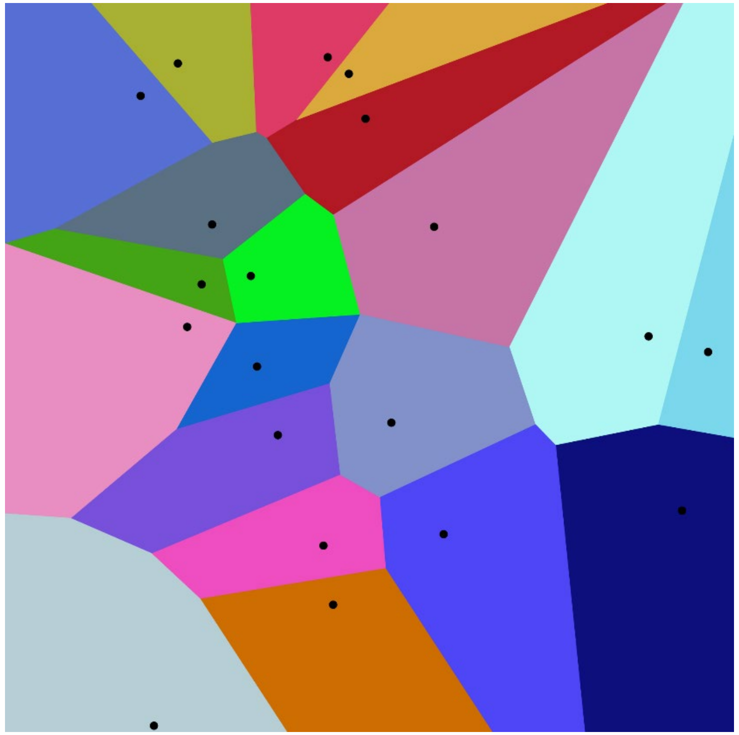
- To classify a new input vector $\mathbf{x}$, examine the k-closest training data points to $\mathbf{x}$ and assign the object to the most frequently occurring class



- common values for k: 3,5

NN Method

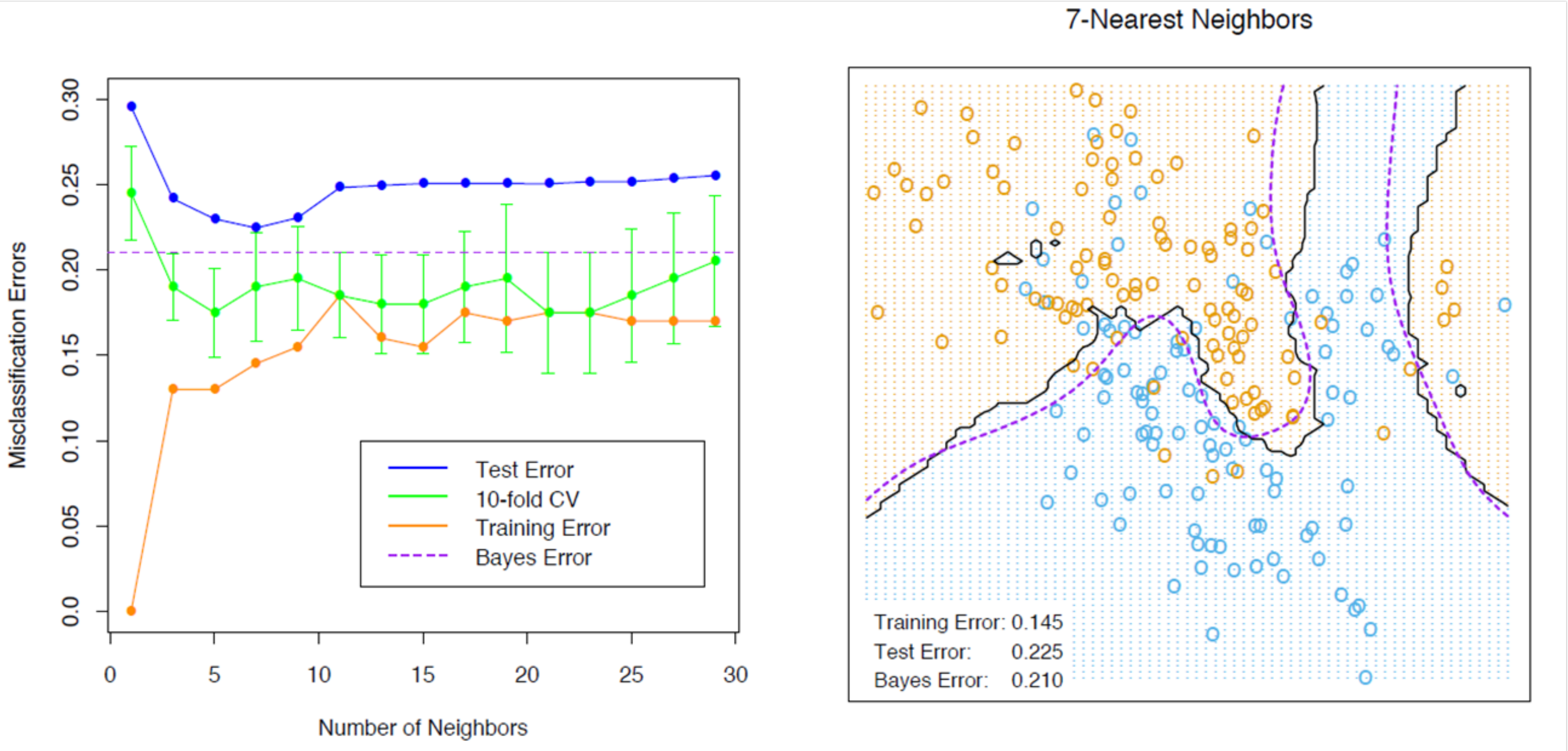
- The nearest neighbor algorithm does not explicitly compute decision boundaries.
  - The decision boundaries form a subset of the Voronoi diagram for the training data.



- 1-NN Decision Surface

- The more examples that are stored, the more complex the decision boundaries can become

NN Method



[Figures from Hastie and Tibshirani, Chapter 13]

NN

- When to Consider
  - Instance map to points in R^n
  - Less than 20 attributes per instance
  - Lots of training data
- Advantages
  - Real-time training (no training actually)
  - Can be very complex target functions
  - Information lossless
- Disadvantages
  - Slow testing
  - Affect easily by irrelevant attributes/outliers

Problems

- Distance measure
  - \- Most common: Euclidean
- Choosing k
  - \- Increasing k reduces variance, increases bias
- For high-dimensional space, problem that the nearest neighbor may not be very close at all!
- Memory-based technique. Must make a pass through the data for each classification. This be prohibitive large data sets.

Distance Measure

- Distance
- Notation: object with p measurements
  - $x^i = (x_1^i, x_2^i, \dots, x_p^i)$
- Most common distance metric is Euclidean distance:
  - $d_E\left(x^i, x^j\right) = \left(\sum_{k=1}^{p}(x_k^i - x_k^j)^2\right)^{1/2}$
- ED makes sense when different measurements are commensurate; each is variable measured in the same units
  - If the measurements are different, say length and weight, it is not clear.

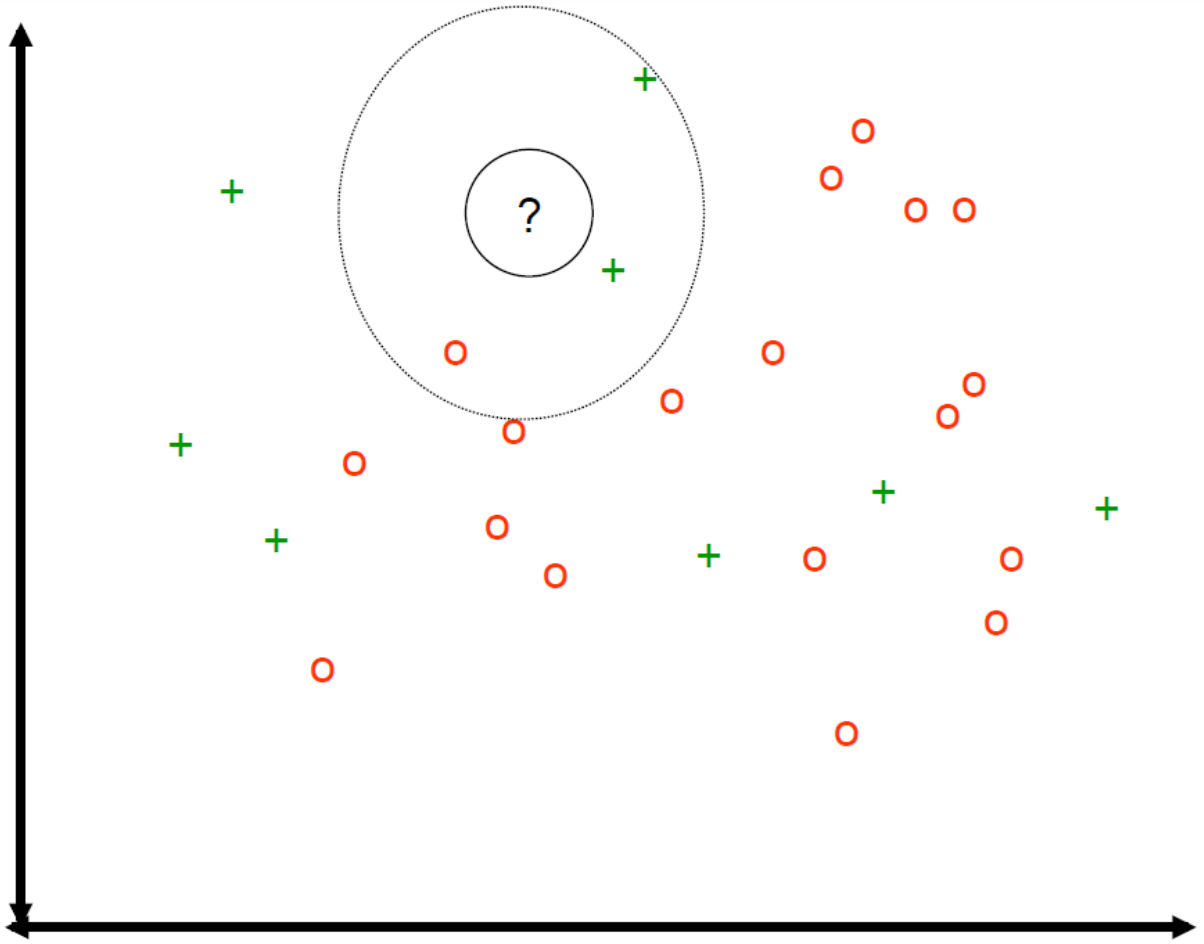
Standardization

- When variables are not commensurate, we can standardize them by dividing by the sample standard deviation. This makes them all equally important.
- The estimate for the standard deviation of x_k:
  - $\widehat{\sigma_k} = (\frac{1}{n}\sum_{i=1}^{n}(x_k^i - \overline{x_k})^2)^{1/2}$
  - where x_k, is the sample mean:
  - $\overline{x_k} = \frac{1}{n}\sum_{i=1}^{n} x_k^i$

Weighted Euclidean distance

- Finally, if we have some idea of the relative importance of each variable, we can weight them:
- $d_{WE}(i,j) = \left(\sum_{k=1}^{p} W_k (x_k^i - x_k^j)^2\right)^{\frac{1}{2}}$

K-NN and irrelevant features

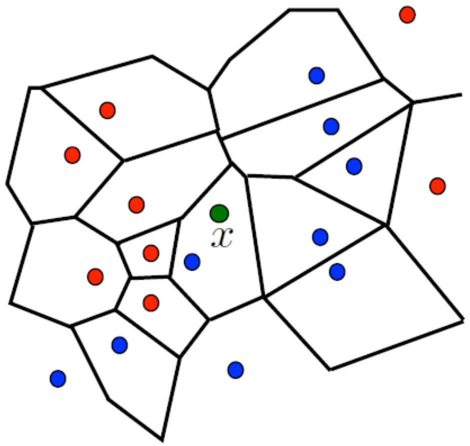


Nearest neighbor problem

- Problem: given sample $S = ((x_1, y_1), \ldots, (x_m, y_m))$,
  - find the nearest neighbor of test point x.
  - general problem extensively studied in computer science.
  - exact vs approximate algorithms.
  - dimensionality N crucial.
  - better algorithms for small intrinsic dimension (e.g., limited doubling dimension).

Efficient Indexing: N=2

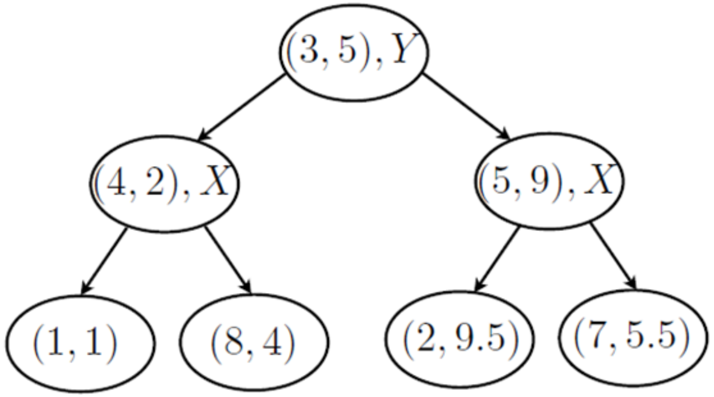
- Algorithm:
  - compute Voronoi diagram in $O(m \log m)$.
  - point location data structure to determine NN.
  - complexity: $O(m)$space, $O(\log m)$time.

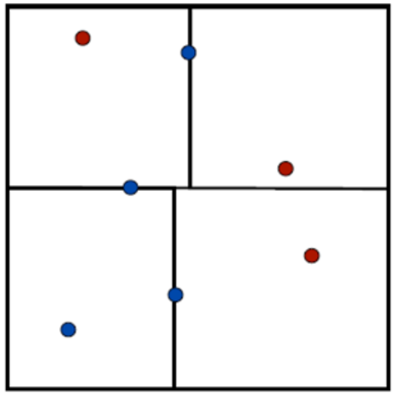


Efficient Indexing: N>2

- Voronoi diagram: size in $O(m^{[n/2]})$.
- Linear algorithm (no pre-processing):
  - Compute distance $||x - x_i||$ for all $i \in [1, m]$.
  - The complexity of distance computation:$\Omega(Nm)$.
  - No additional space needed.
- Tree-based data structures: pre-processing.
  - often used in applications: k-d trees (k-dimensional trees).

Efficient Indexing: N>2 = KDTree



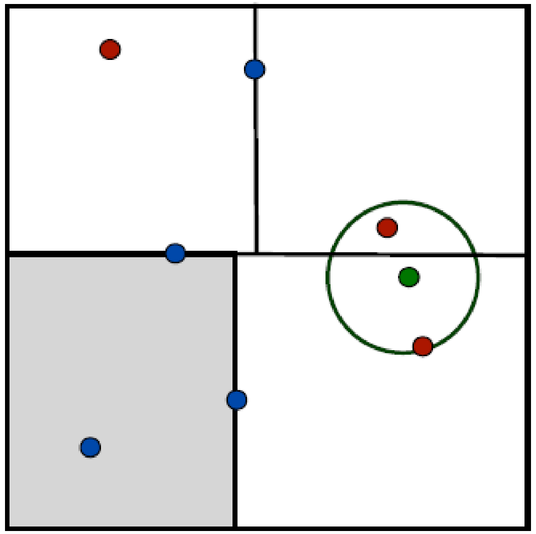


- Construction algorithm
- Algorithm: for each non-leaf node,
  - choose dimension (eg. longest of hyperrectangle).
  - choose pivot (median).
  - split node according to (pivot, dimension).
- →balanced tree, binary space partition

Efficient Indexing: N>2 = KDTree

- Algonthm:
  - find region containing x(starting from root node, move to child node based on node test).
  - save region point x_0 as current best.
  - move up tree and recursively search regions intersecting hypersphere $S(x, ||x - x_0||)$;
    - update current best if current point is closer .
    - restart search with each intersecting sub-tree.
    - move up tree when no more intersecting sub- tree.

Search algorithm



KNN Advantages

- Easy to program
- No optimization or training required
- Classification accuracy can be very good; can outperform more complex models

DECISION TREE CLASSIFIER

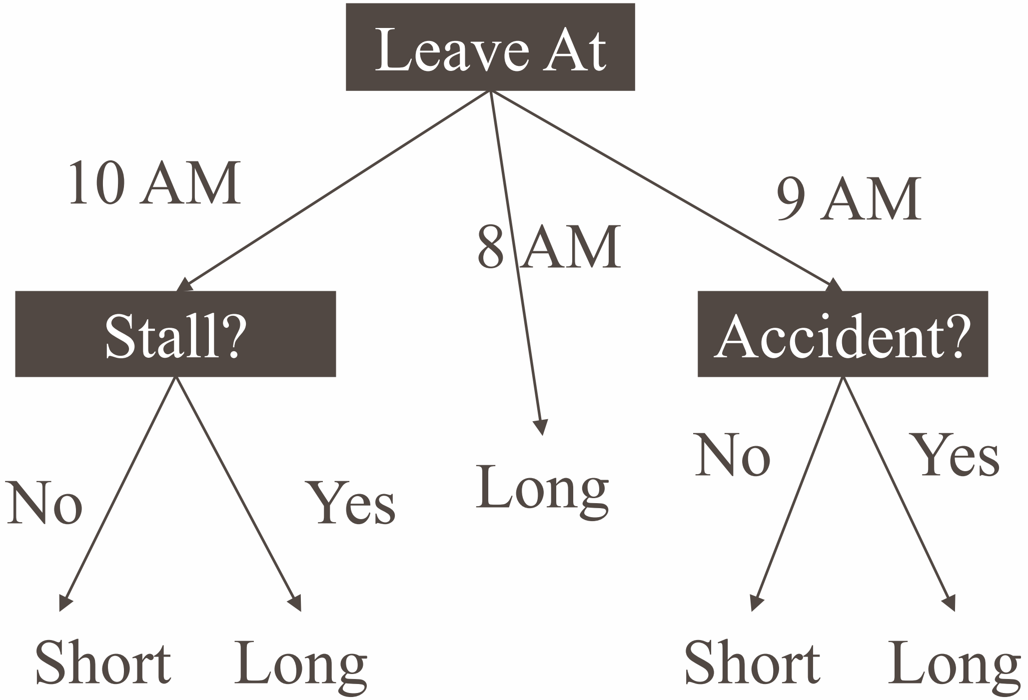
A sample data set

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

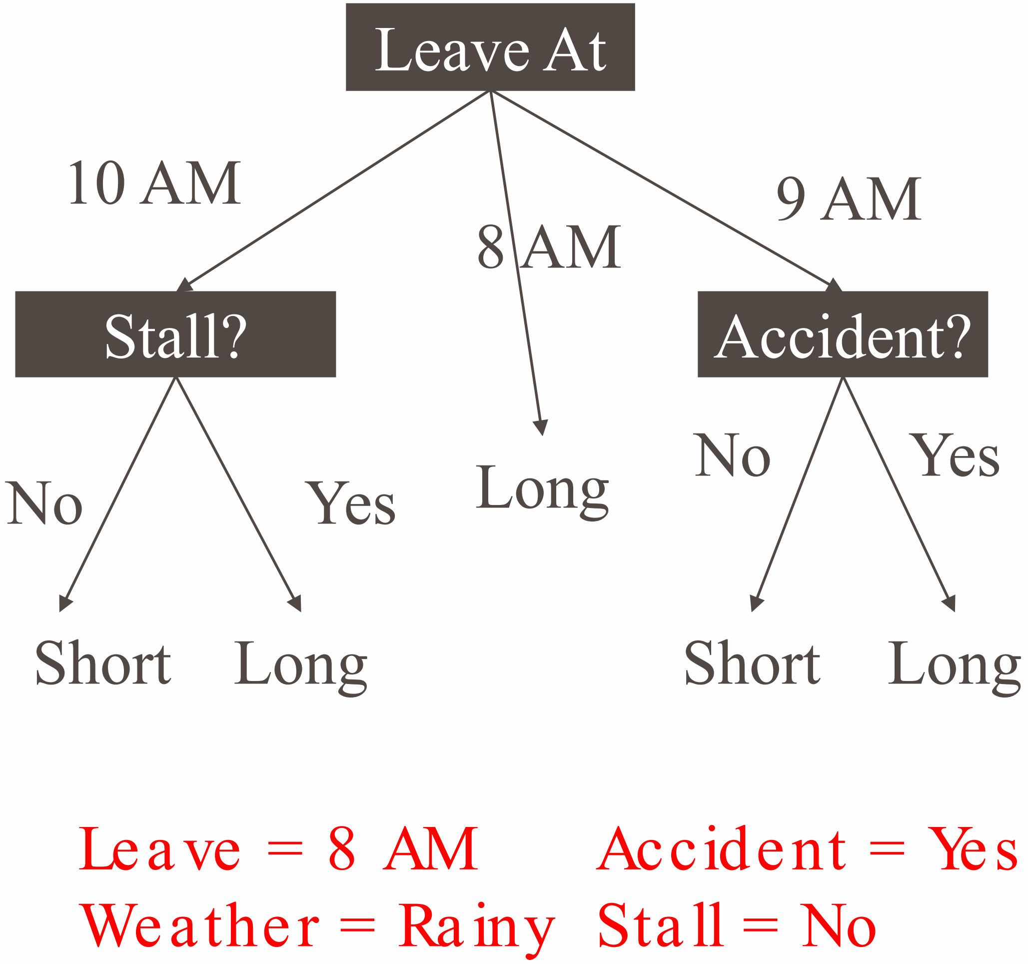


Tree with internal nodes labeled by features

Branches are labeled by tests on that feature

Leaves labeled with classes

Decision trees

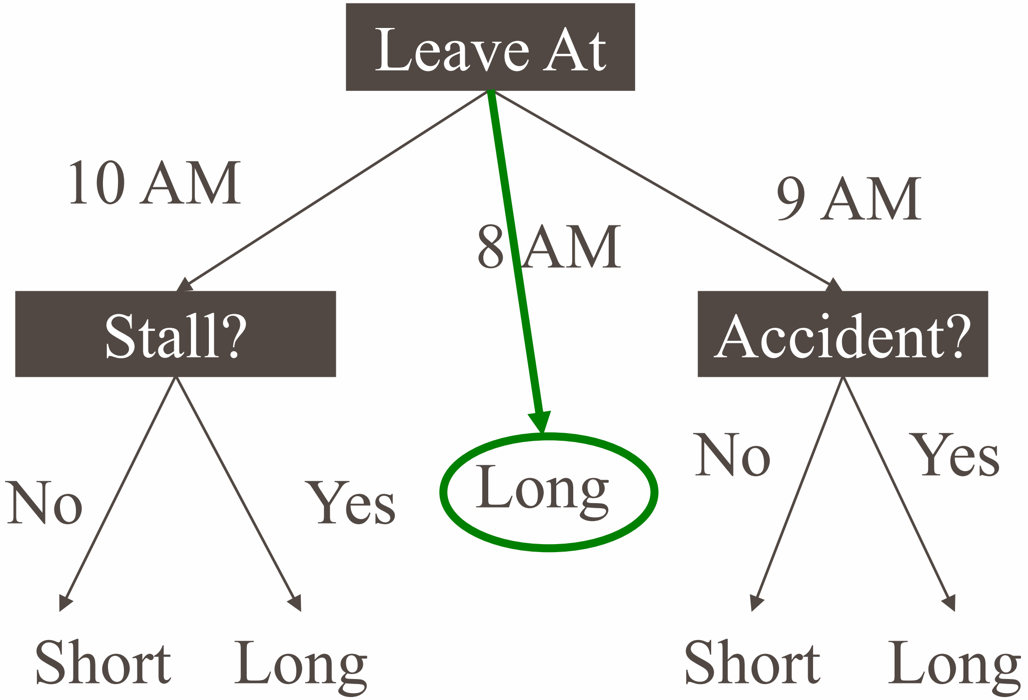


Tree with internal nodes labeled by features

Branches are labeled by tests on that feature

Leaves labeled with classes

Decision trees



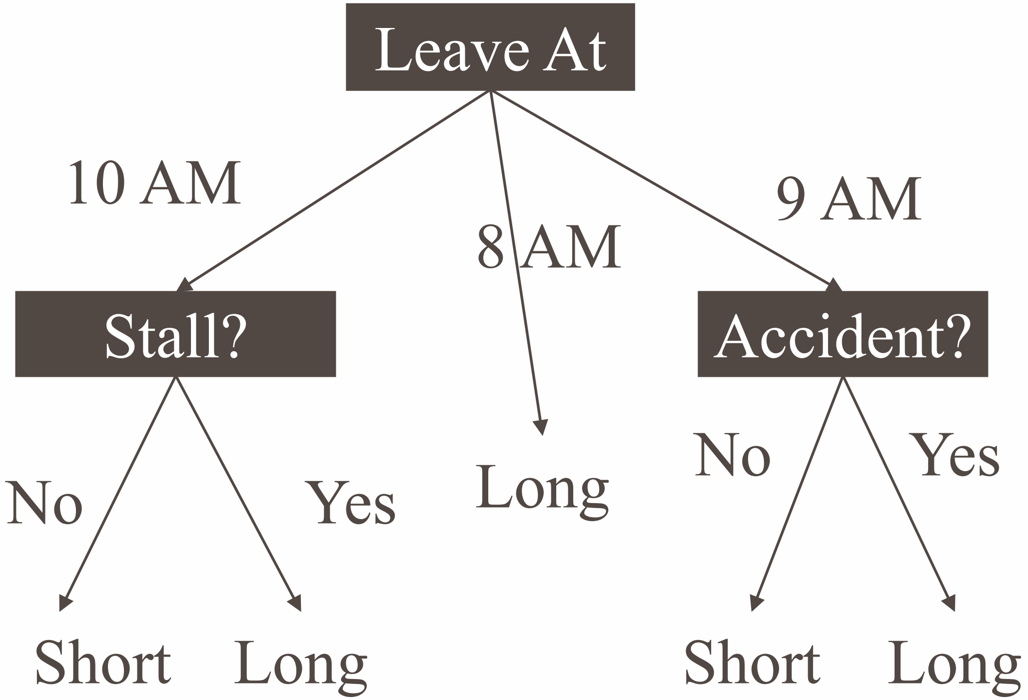
Leave = 8 AM Accident = Yes
Weather = Rainy Stall = No

Tree with internal nodes labeled by features

Branches are labeled by tests on that feature

Leaves labeled with classes

Decision trees



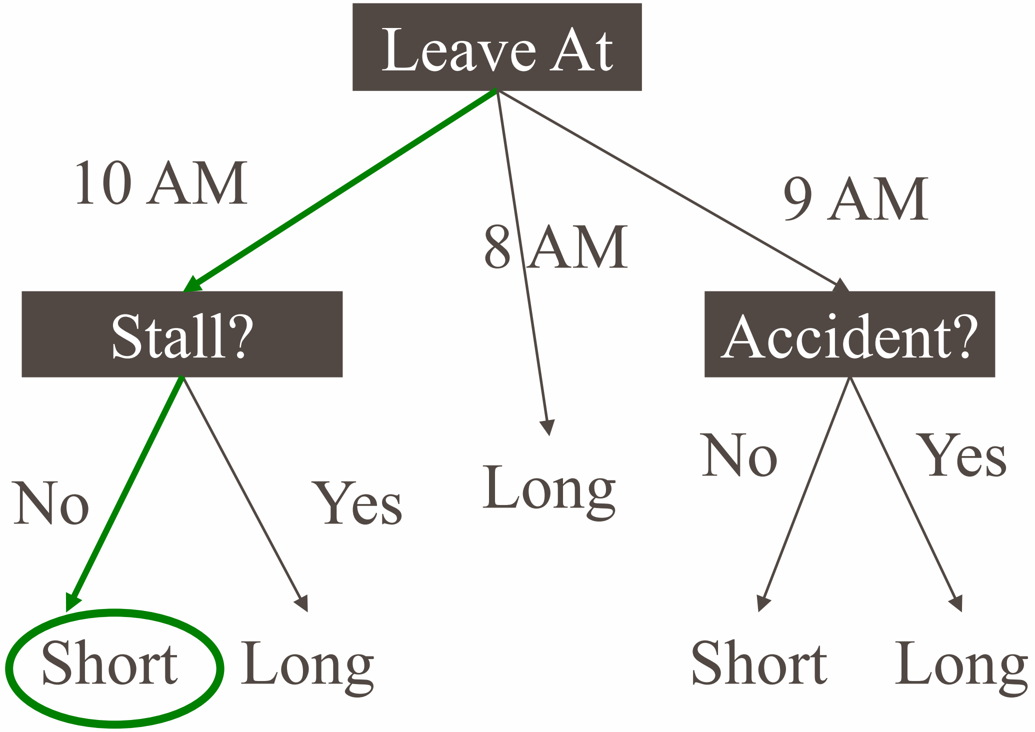
Leave = 10 AM   Accident = No
Weather = Rainy   Stall = No

Tree with internal nodes labeled by features

Branches are labeled by tests on that feature

Leaves labeled with classes

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

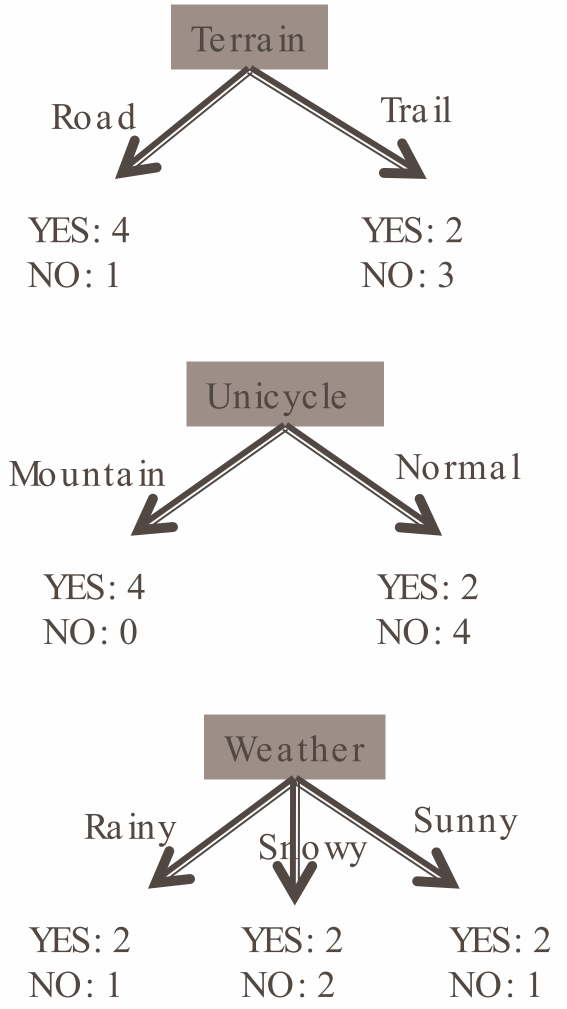
Recursive approach

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

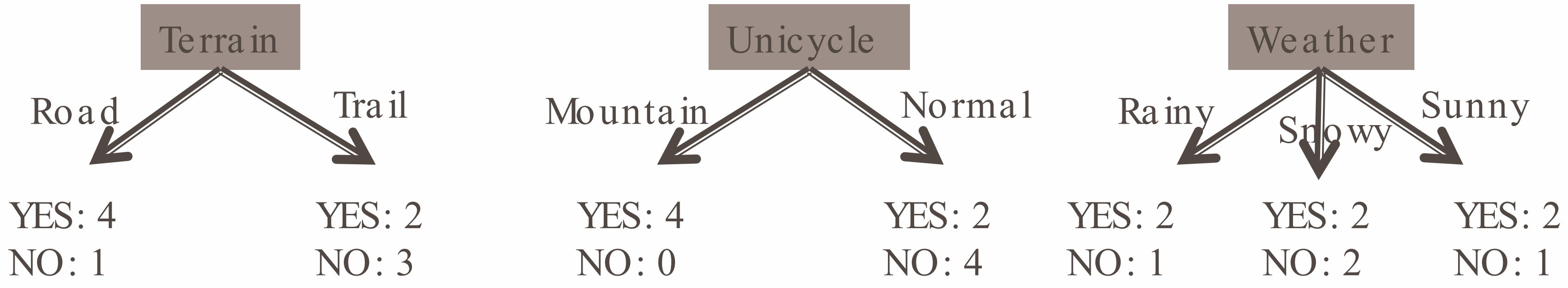
- Otherwise:
  - calculate the “score” for each feature if we used it to split the data
  - pick the feature with the highest score, partition the data based on that data value and call recursively

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



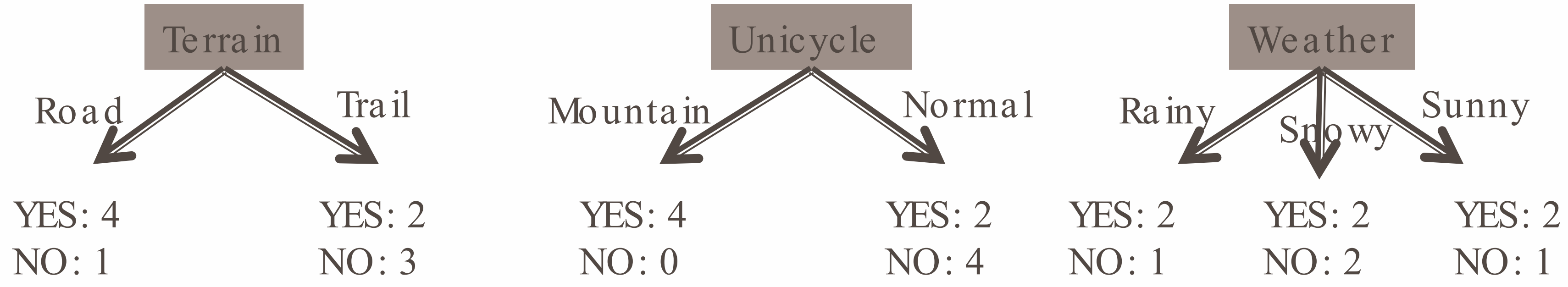
Partitioning the data



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

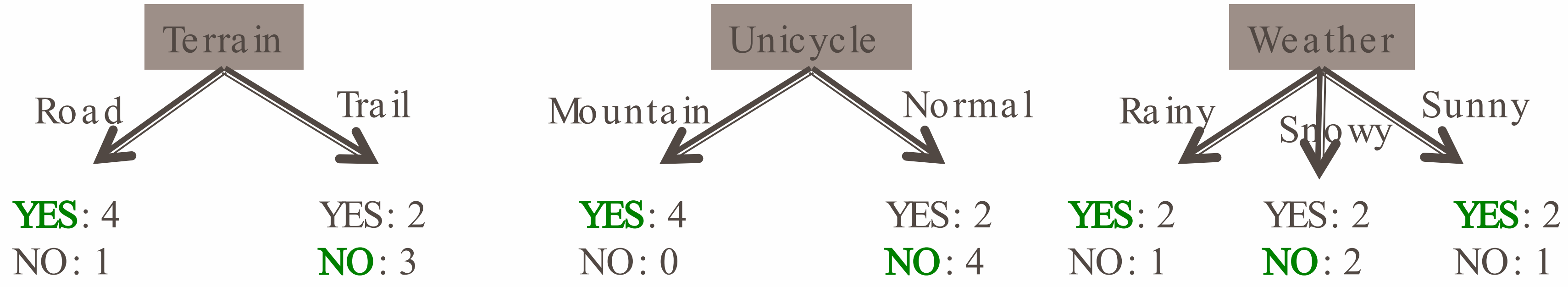
What score should we use?
If we just stopped here, which tree would be best? How could we make these into decision trees?

Decision trees

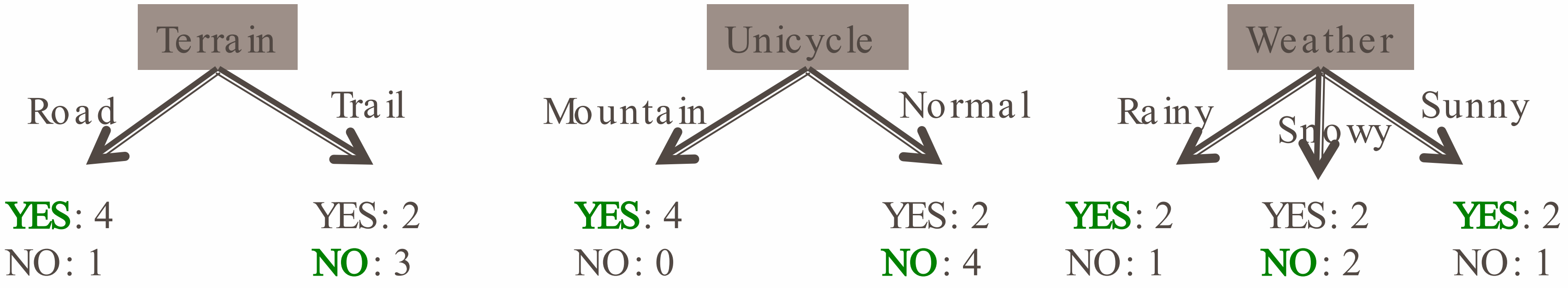


How could we make these into decision trees?

Decision trees



Decision trees

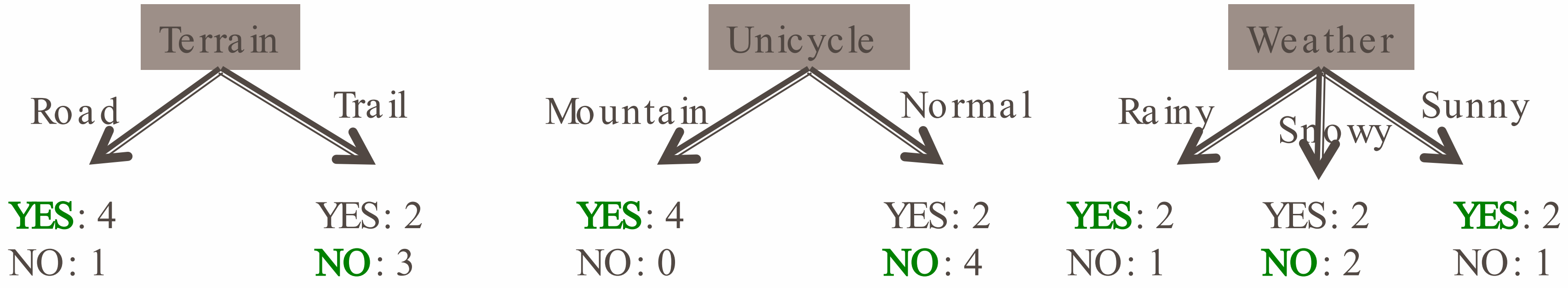


Training error: the average error over the training set

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

Unicycle

Mountain

Normal

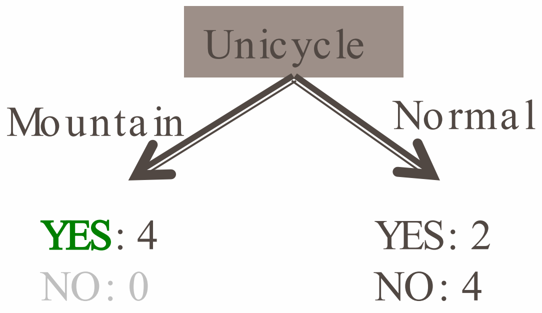
YES: 4
NO: 0

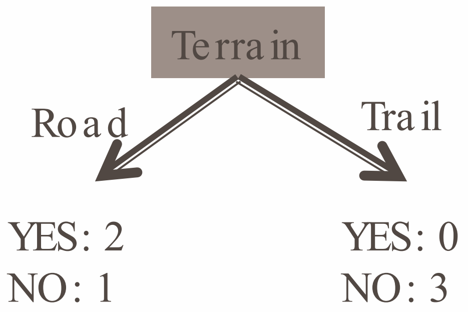
YES: 2
NO: 4

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

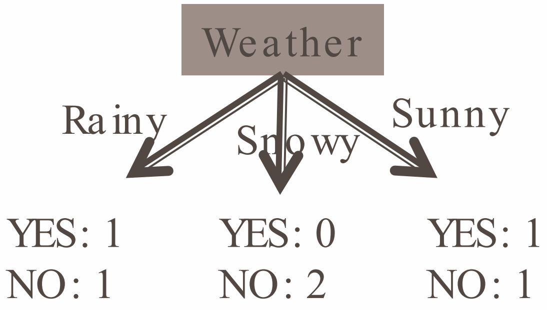
Terrain	Unicycle-type	Weather	Go-For-Ride?
Trail	Normal	Rainy	NO
Road	Normal	Sunny	YES
Trail	Normal	Snowy	NO
Road	Normal	Rainy	YES
Trail	Normal	Sunny	NO
Road	Normal	Snowy	NO

Recurse

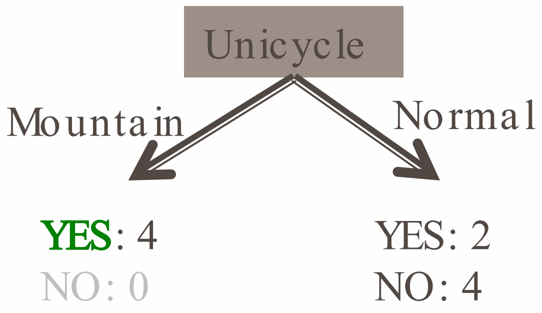


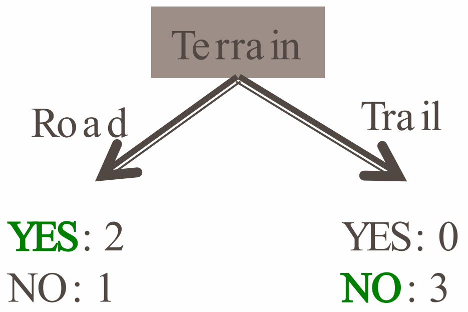


Terrain	Unicycle-type	Weather	Go-For-Ride?
Trail	Normal	Rainy	NO
Road	Normal	Sunny	YES
Trail	Normal	Snowy	NO
Road	Normal	Rainy	YES
Trail	Normal	Sunny	NO
Road	Normal	Snowy	NO



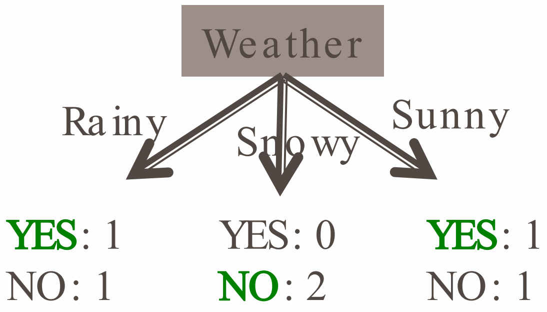
Recurse





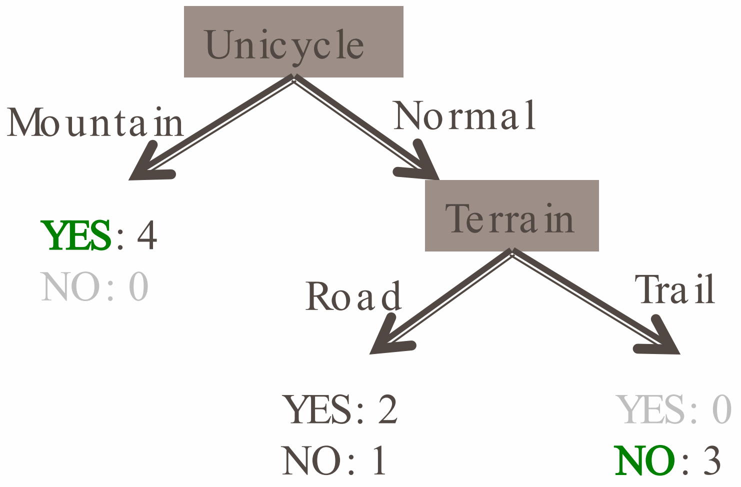
1/ 6

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



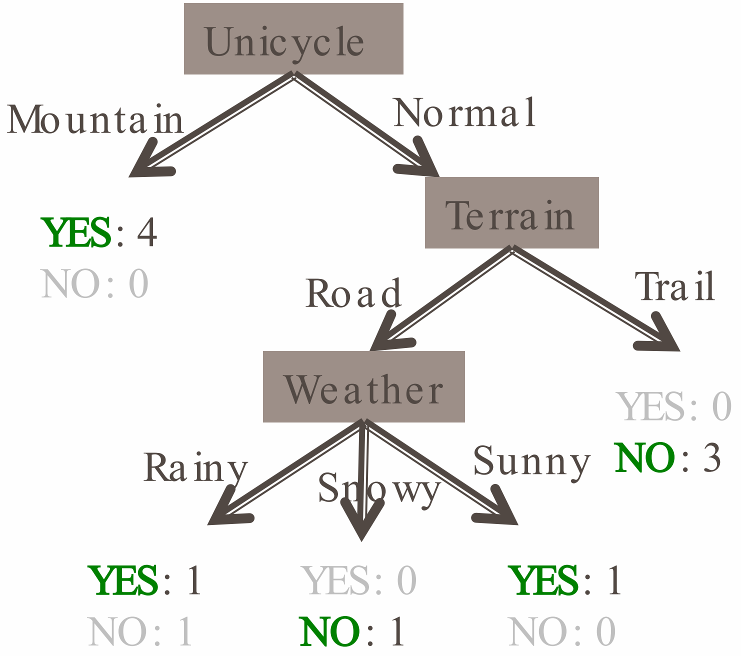
2/ 6

Recurse



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

Recurse



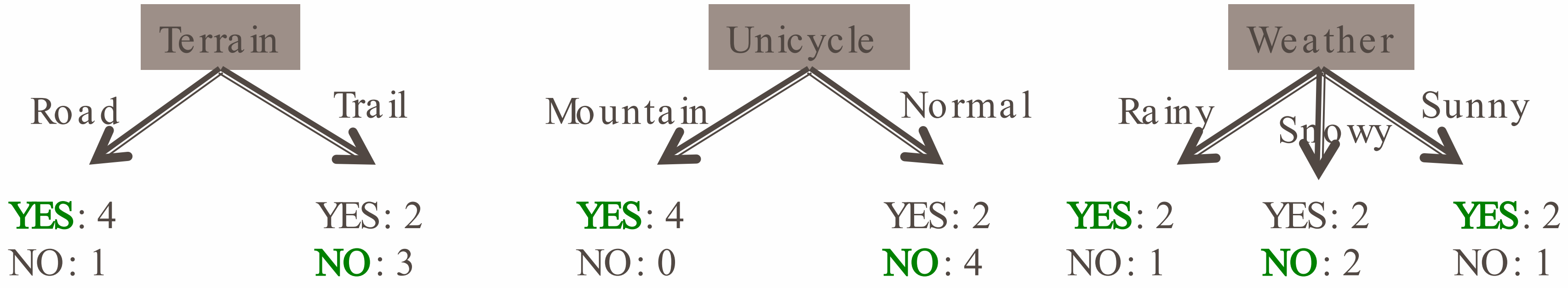
Unicycle
Mountain
Normal
YES: 4
NO: 0
Terrain
Road
Trail
Weather
YES: 0
NO: 3
Rainy
Snowy
Sunny
YES: 1
NO: 1
YES: 0
NO: 1
YES: 1
NO: 0

On-site Practice

- Based on the information on the Moodle, please finish the requirements below:
  - Draw the decision manually like pp. 46
  - Write a flowchart of the programming (block-diagram) of“decision tree”
    - Can be powerpoint / word/ visio

BUILD A DECISION TREE WITH CRITERION

Decision trees

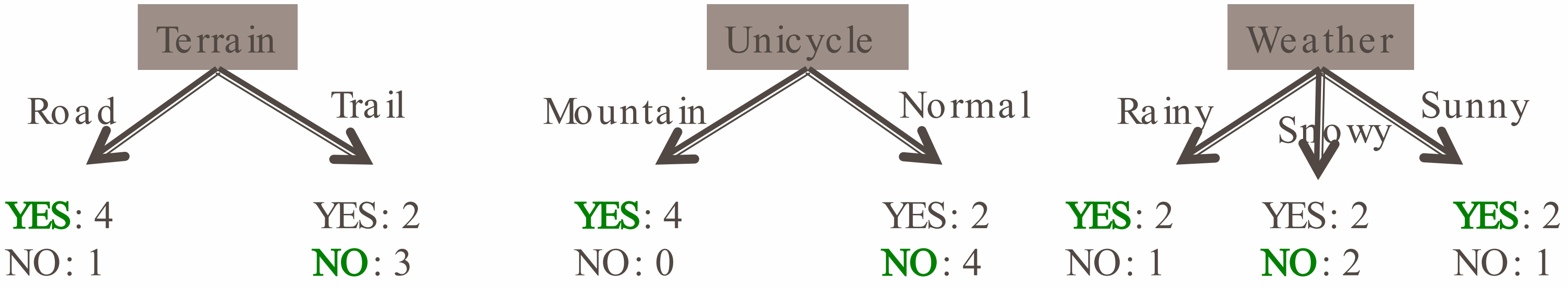


3/ 10 | 2/ 10 | 4/ 10

Training error: the average error over the training set

ACVLab

Training error vs. accuracy



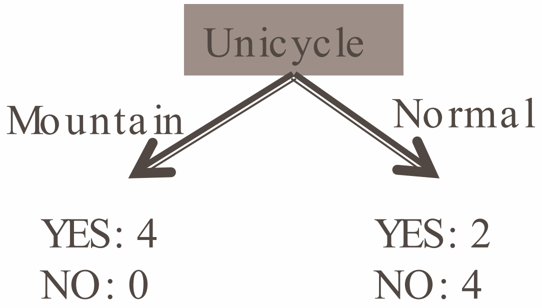
Training error:	3/ 10	2/ 10	4/ 10
Training accuracy:	7/ 10	8/ 10	6/ 10

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

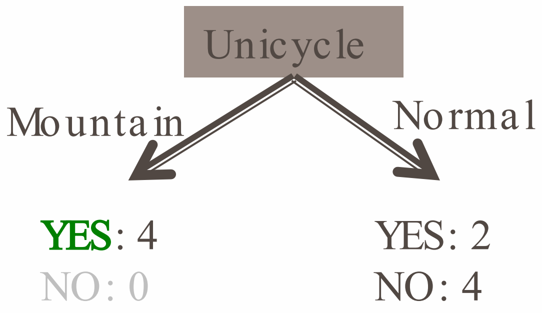
Recurse

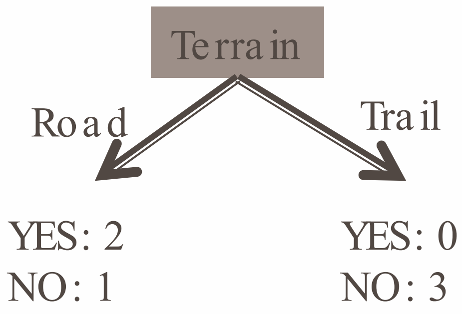


Terrain	Unicycle-type	Weather	Go-For-Ride?
Trail	Mountain	Sunny	YES
Road	Mountain	Rainy	YES
Road	Mountain	Snowy	YES
Trail	Mountain	Snowy	YES

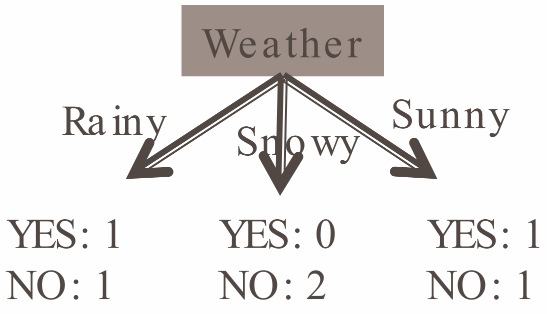
Terrain	Unicycle-type	Weather	Go-For-Ride?
Trail	Normal	Rainy	NO
Road	Normal	Sunny	YES
Trail	Normal	Snowy	NO
Road	Normal	Rainy	YES
Trail	Normal	Sunny	NO
Road	Normal	Snowy	NO

Recurse

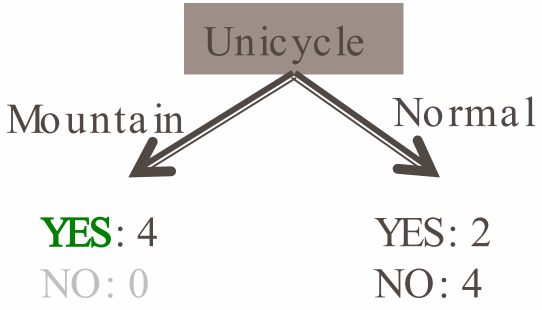


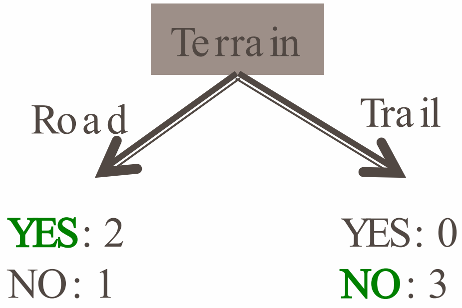


Terrain	Unicycle-type	Weather	Go-For-Ride?
Trail	Normal	Rainy	NO
Road	Normal	Sunny	YES
Trail	Normal	Snowy	NO
Road	Normal	Rainy	YES
Trail	Normal	Sunny	NO
Road	Normal	Snowy	NO



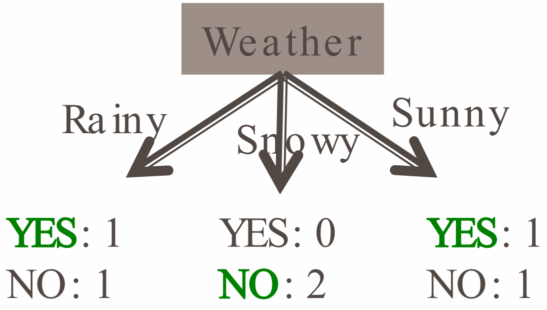
Recurse





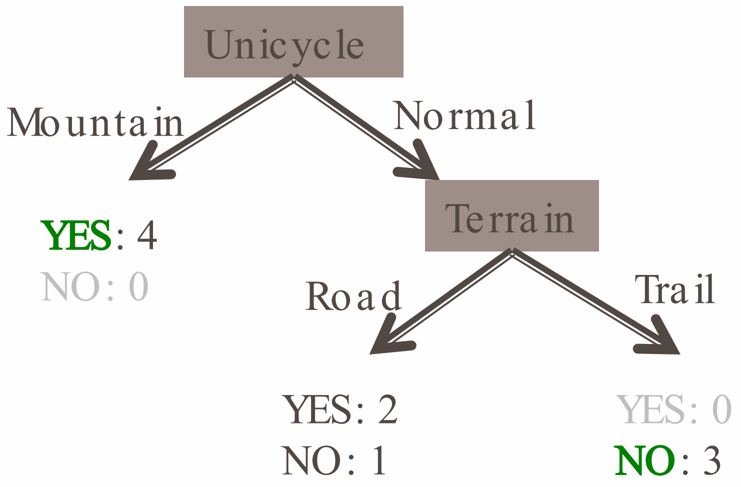
1/ 6

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



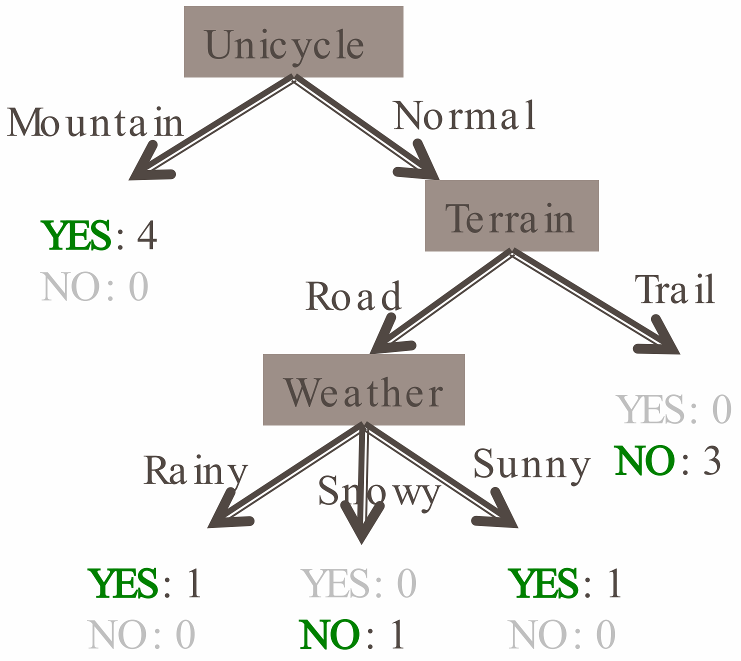
2/ 6

Recurse

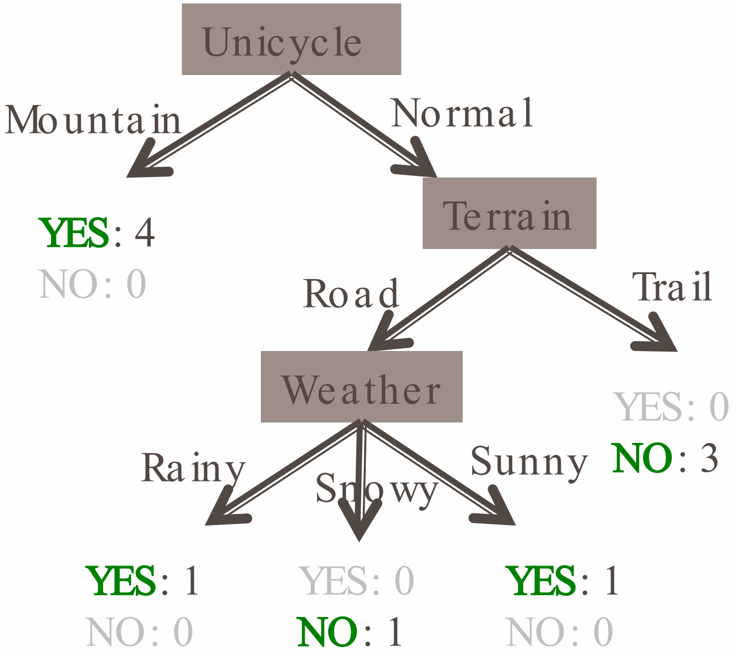


Terrain	Unicycle-type	Weather	Go-For-Ride?
Road	Normal	Sunny	YES
Road	Normal	Rainy	YES
Road	Normal	Snowy	NO

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

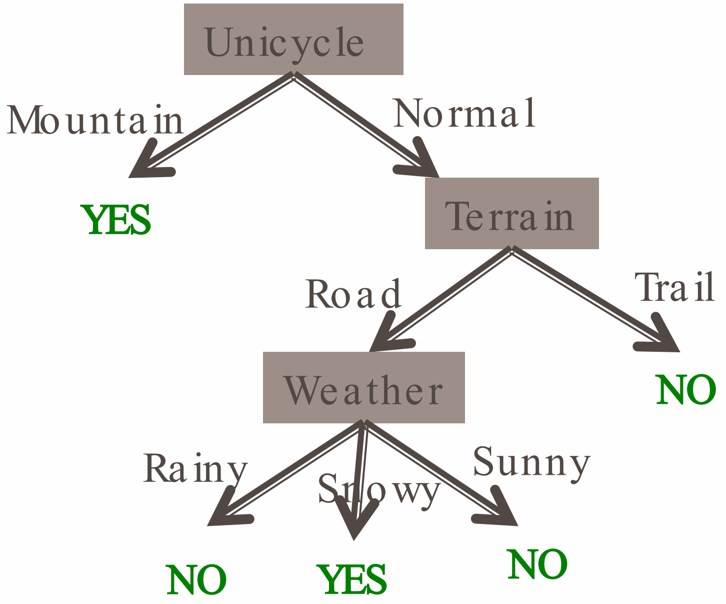
Do we always want to go all the way to the bottom?

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…

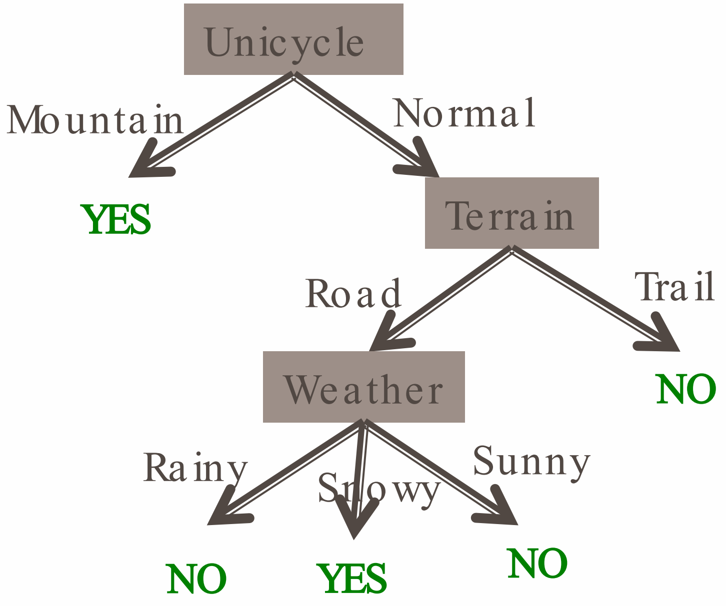
Terrain	Unicycle-type	Weather	Go-For-Ride?
Trail	Mountain	Rainy	YES
Trail	Mountain	Sunny	YES
Road	Mountain	Snowy	YES
Road	Mountain	Sunny	YES
Trail	Normal	Snowy	NO
Trail	Normal	Rainy	NO
Road	Normal	Snowy	YES
Road	Normal	Sunny	NO
Trail	Normal	Sunny	NO



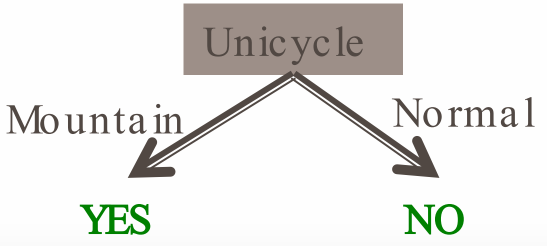
Is that what you would do?

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…

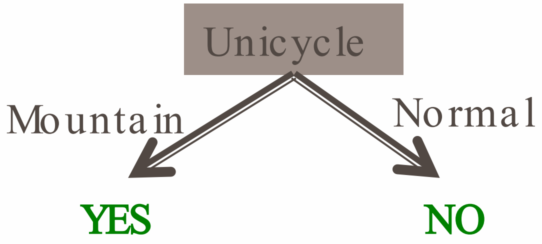


What would the tree look like for…

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

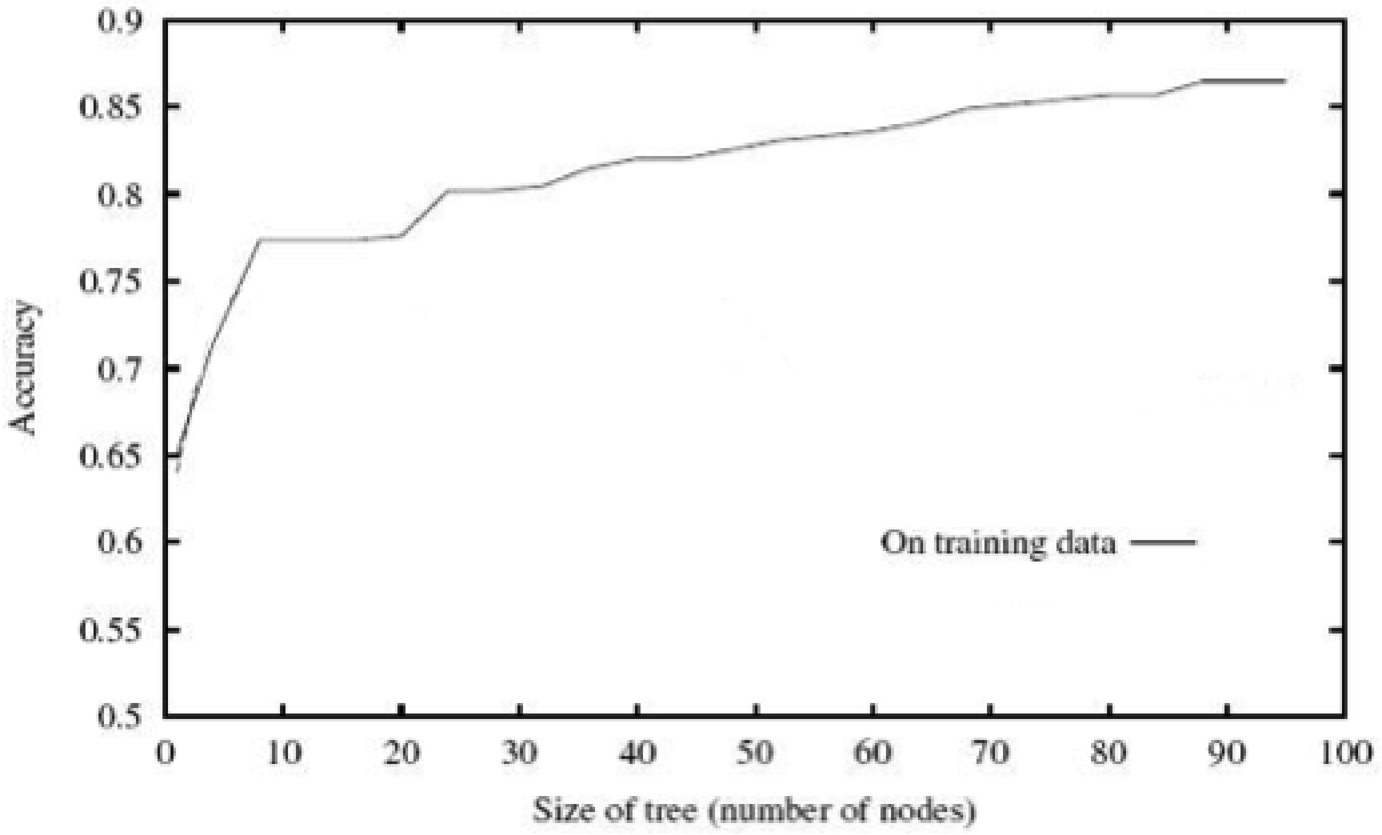
Terrain	Unicycle-type	Weather	Go-For-Ride?
Trail	Mountain	Rainy	YES
Trail	Mountain	Sunny	YES
Road	Mountain	Snowy	YES
Road	Mountain	Sunny	YES
Trail	Normal	Snowy	NO
Trail	Normal	Rainy	NO
Road	Normal	Snowy	YES
Road	Normal	Sunny	NO
Trail	Normal	Sunny	NO



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



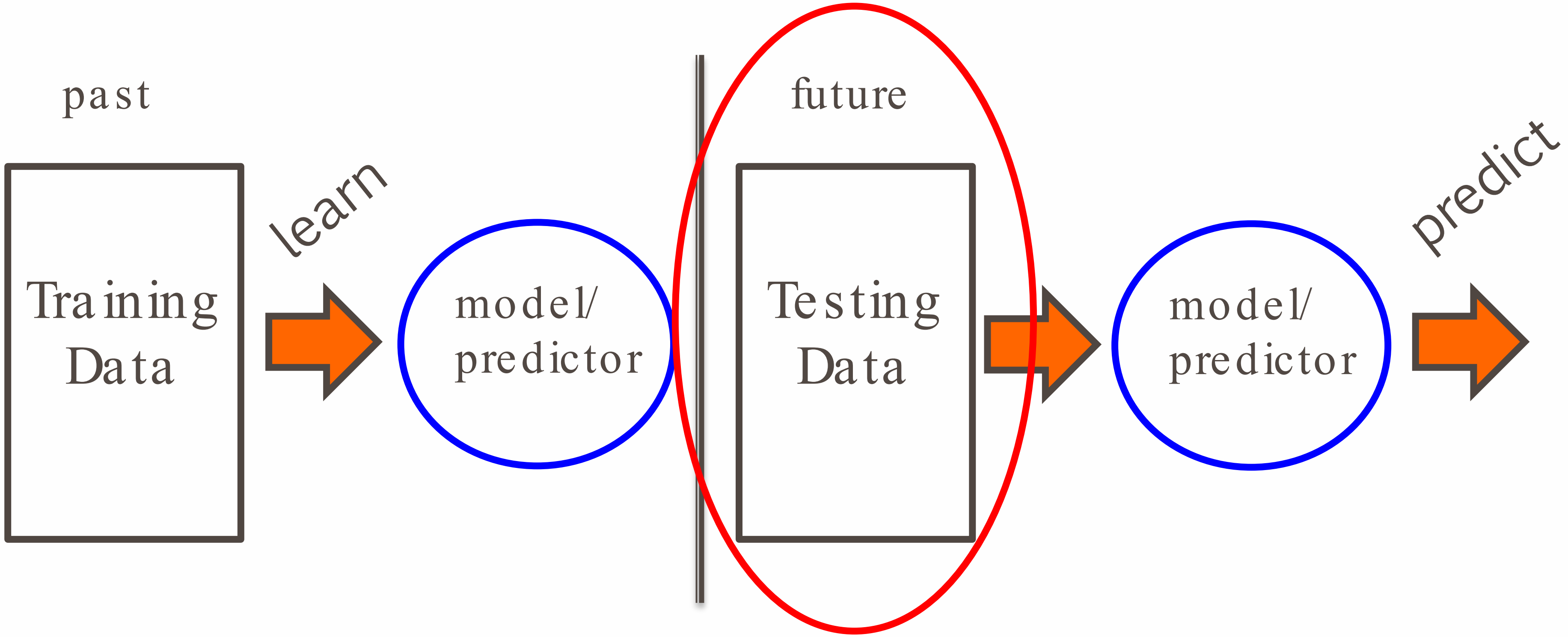
Our decision tree learning procedure always decreases training error

Is that what we want?

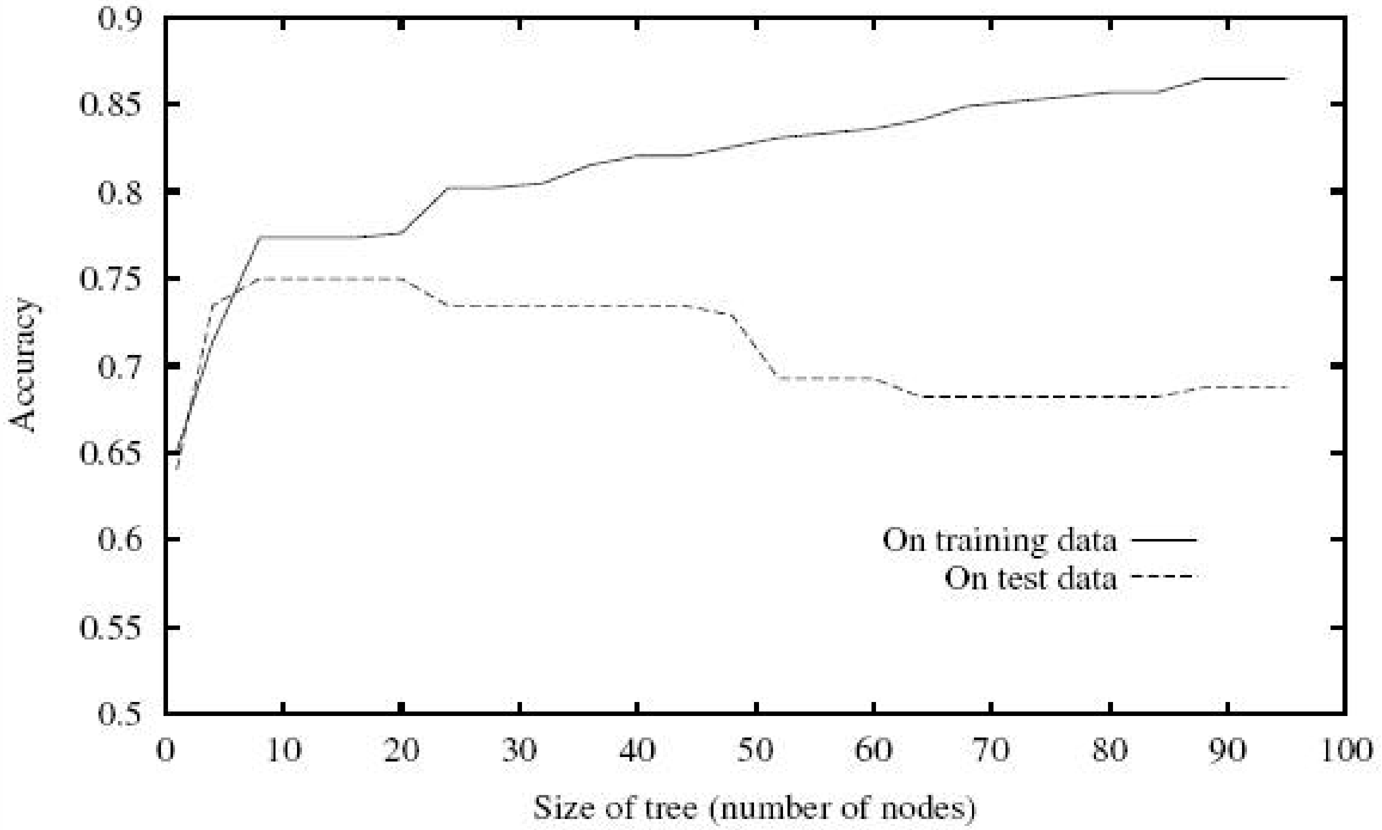
Test set error!

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

-- Hal Daume III



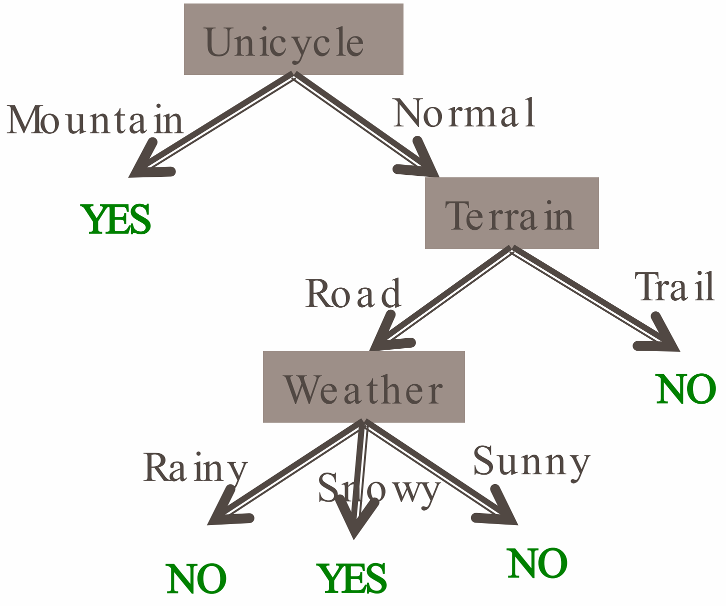
Overfitting



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



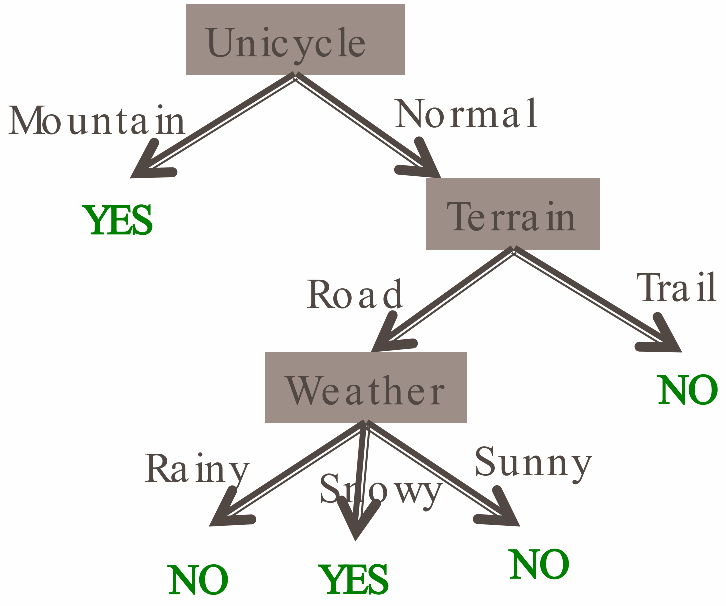
How do we prevent overfitting?

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

One idea: stop building the tree early

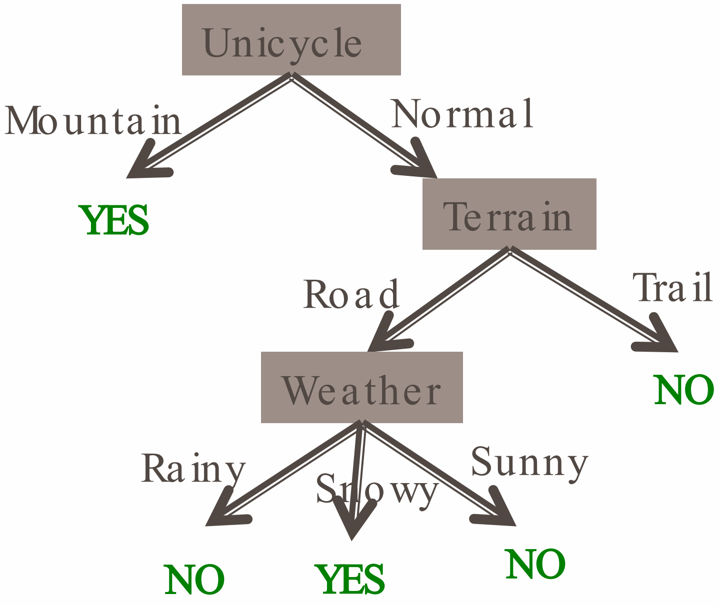
Preventing overfitting: pruning



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

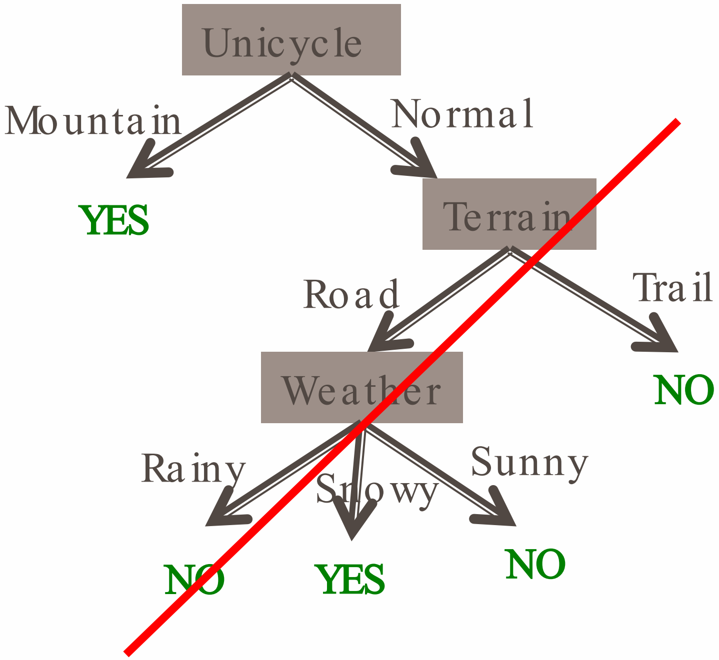
Similar to stopping early, but done after the entire tree is built

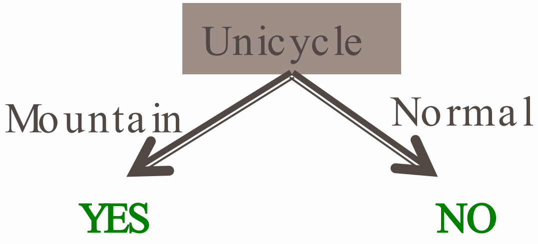
Preventing overfitting: pruning



Build the full tree

Preventing overfitting: pruning

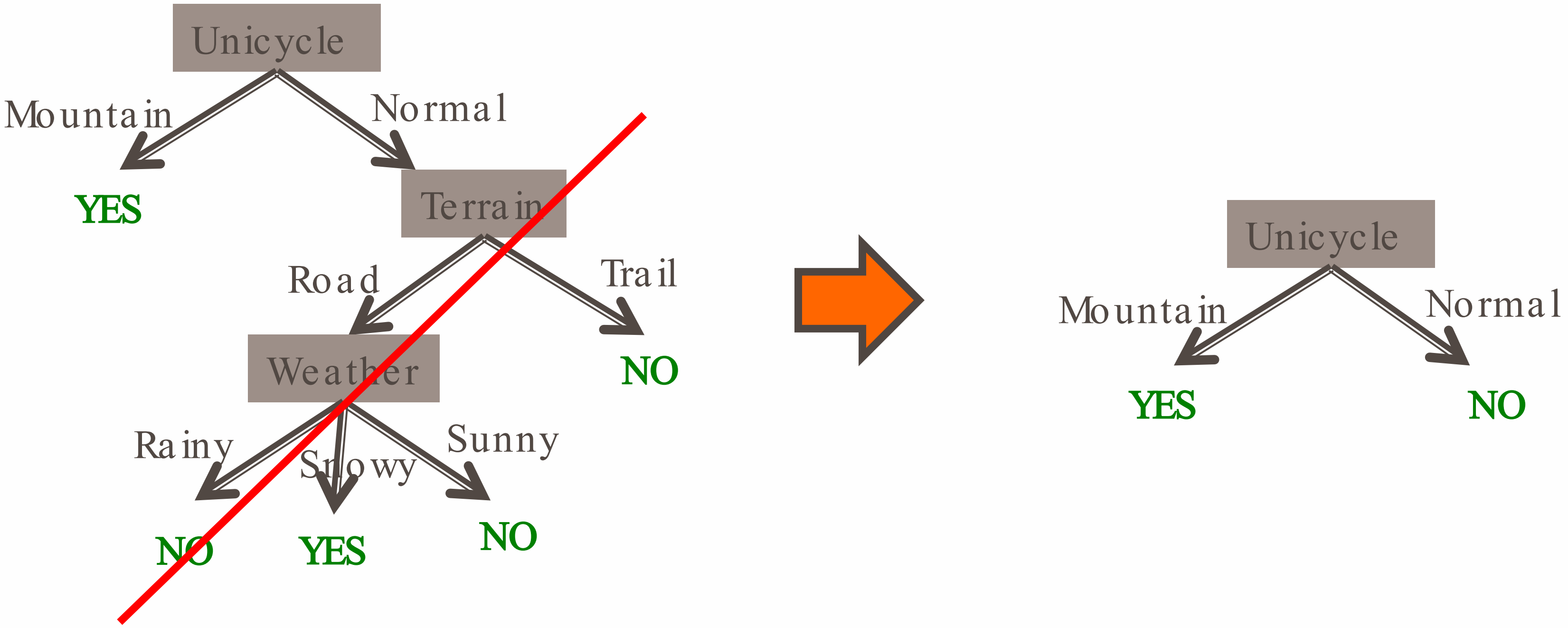




Build the full tree

Prune back leaves that are too specific

Preventing overfitting: pruning



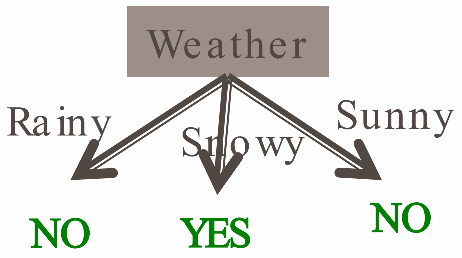
Pruning criterion?

Handling non-binary attributes

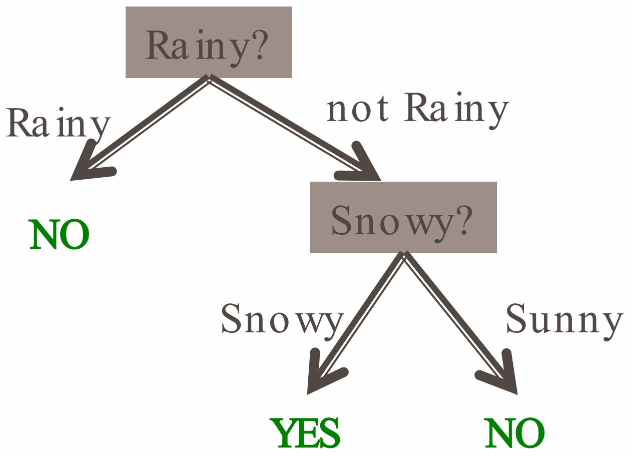
PassengerId	Pclass	Sex	Age	SibSp	Parch	Ticket	Fare	Embarked	Survived
804	3	0	0.42	0	1	2625	8.5167	0	1
756	2	0	0.67	1	1	250649	14.5	2	1
470	3	1	0.75	2	1	2666	19.2583	0	1
645	3	1	0.75	2	1	2666	19.2583	0	1
79	2	0	0.83	0	2	248738	29	2	1
832	2	0	0.83	1	1	29106	18.75	2	1
306	1	0	0.92	1	2	113781	151.55	2	1
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



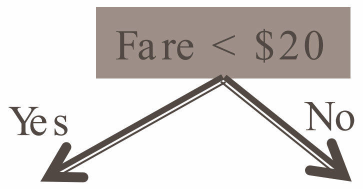
Treat as an n-ary split

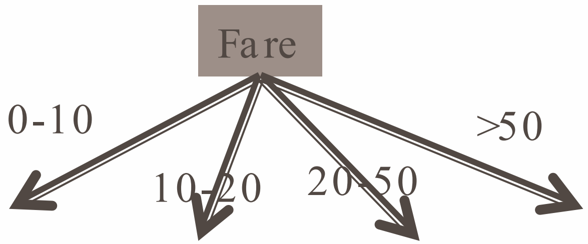


Treat as multiple binary splits

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?

INFORMATION GAIN

GINI, Entropy, Information

Information

- Imagine:
- 1. Someone is about to tell you your own name
- 2. You are about to observe the outcome of a dice roll
- 2. You are about to observe the outcome of a coin flip
- 3. You are about to observe the outcome of a biased coin flip

- Each situation have a different amount of uncertainty
- as to what outcome you will observe.

Information

- Information:
- reduction in uncertainty (amount of surprise in the outcome)

$$I(E) = \log_2 \frac{1}{p(x)} = -\log_2 p(x)$$

If the probability of this event happening is small and it happens the information is large.

1. Observing the outcome of a coin flip is head $\longrightarrow$ $I = -\log_2 1/2 = 1$

2. Observe the outcome of a dice is 6 $\longrightarrow$ $I = -\log_2 1/6 = 2.58$

Entropy

- The *expected amount of information* when observing the output of a random variable X

$$H(X) = E(I(X)) = \sum_i p(x_i) I(x_i) = -\sum_i p(x_i) \log_2 p(x_i)$$

If there X can have 8 outcomes and all are equally likely

$$H(X) == -\sum_i 1/8 \log_2 1/8 = 3 \quad \text{bits}$$

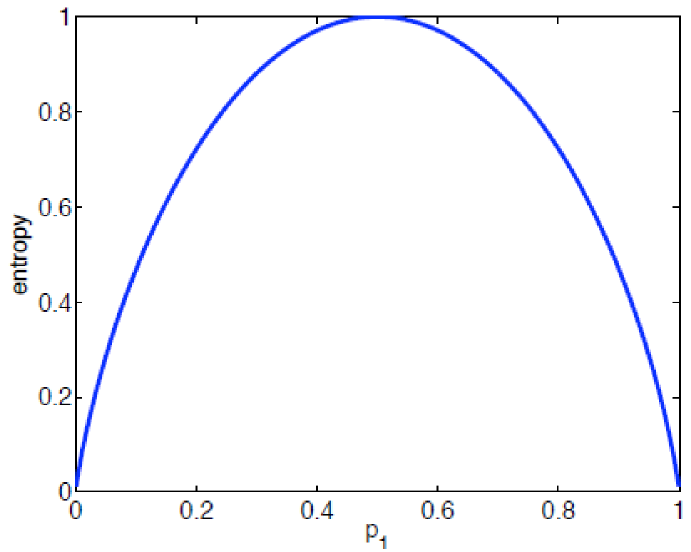
Entropy

- If there are *k* possible outcomes

$$H(X) \leq \log_2 k$$

- Equality holds when all outcomes are equally likely

- The more the probability distribution the deviates from uniformity the lower the entropy



Entropy, purity

- Entropy measures the purity

4 +
4 -

8 +
0 -

The distribution is less uniform
Entropy is lower
The node is purer

Other metric: Information Gain

- Split node by selecting the maximal information gain
- Assumed binary class
  - where we have *p* positive samples and *n* negative samples in dataset **S**.
- Information value for the whole data can be calculated as

$$I(p,n) = -\frac{p}{p+n}\log_2\frac{p}{p+n} - \frac{n}{p+n}\log_2\frac{n}{p+n}$$

  - When the p=n, we have the maximal information value

Other metric: Information Gain

- Dataset S={S_1，S_2，…，S_v}
  - where S_i includes p_i positive samples and n_i negative samples
- We calculate all information values for each nodes

$$E(A) = \sum_{i=1}^{v} \frac{p_i + n_i}{p + n} I(p_i, n_i)$$

  - The best case (ideal case) will be at classification error=0
    - In which all prediction results are correct

Other metric: Information Gain

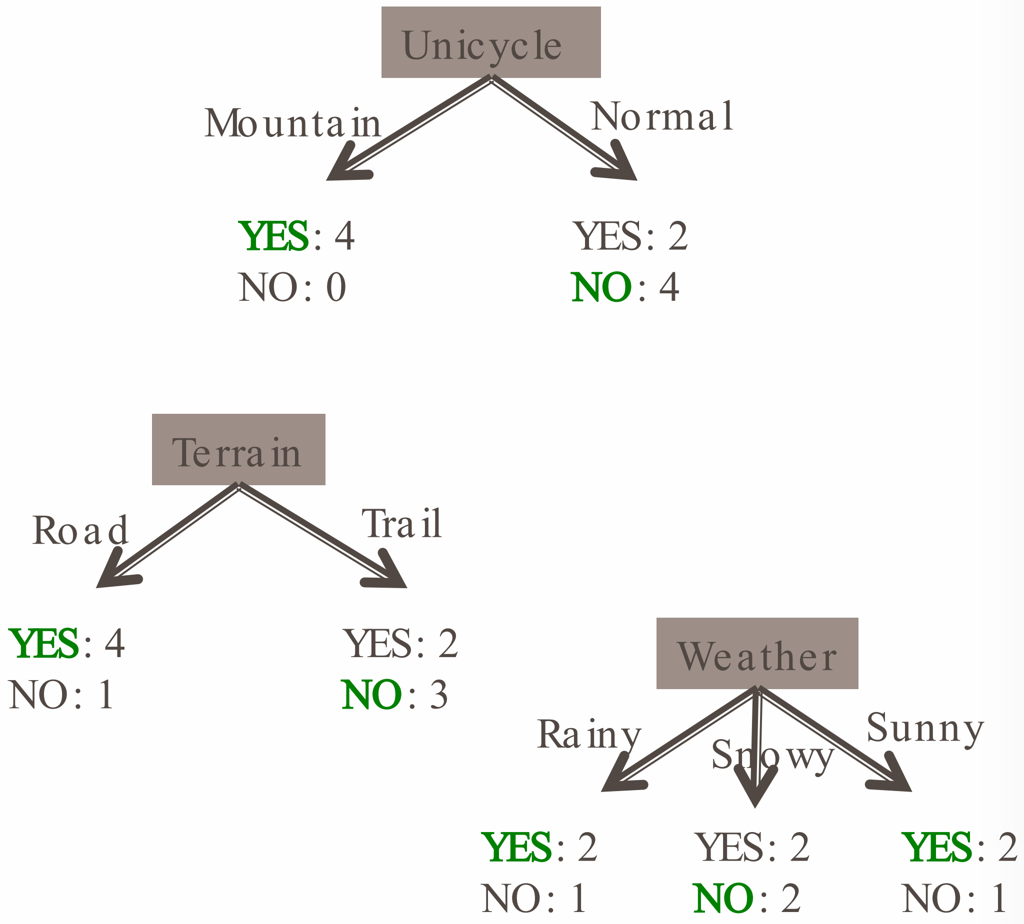
- Now, we have information gain, which is defined as

$$Gain(A) = I(p,n) - E(A)$$

- As we may know, I(p,n) will have the maximal value when the answers got all wrong
  - We also know that E is the subtotal of information values for each nodes.
- If Gain(A) is maximal, then either E(A) is smaller or I(p,n) is larger

Other metric: Information Gain

- Total: 10 samples
  - Yes: 6 samples, No : 4 samples
- I(p,n)=I(6,4)=0.970
- E(A) is the subtotal of each node
  - I_Unicycle(p, n) = I(4,0) + I(2,4)=0.55
  - I_terrain(p,n)=I(4, 1)+I(2,3)=0.846
  - I_Weather(p,n)=I(2,1)+I(2,2)+I(2,1)=0.95
- Apparently, Unicycle is a good one



Other Metric: impurity

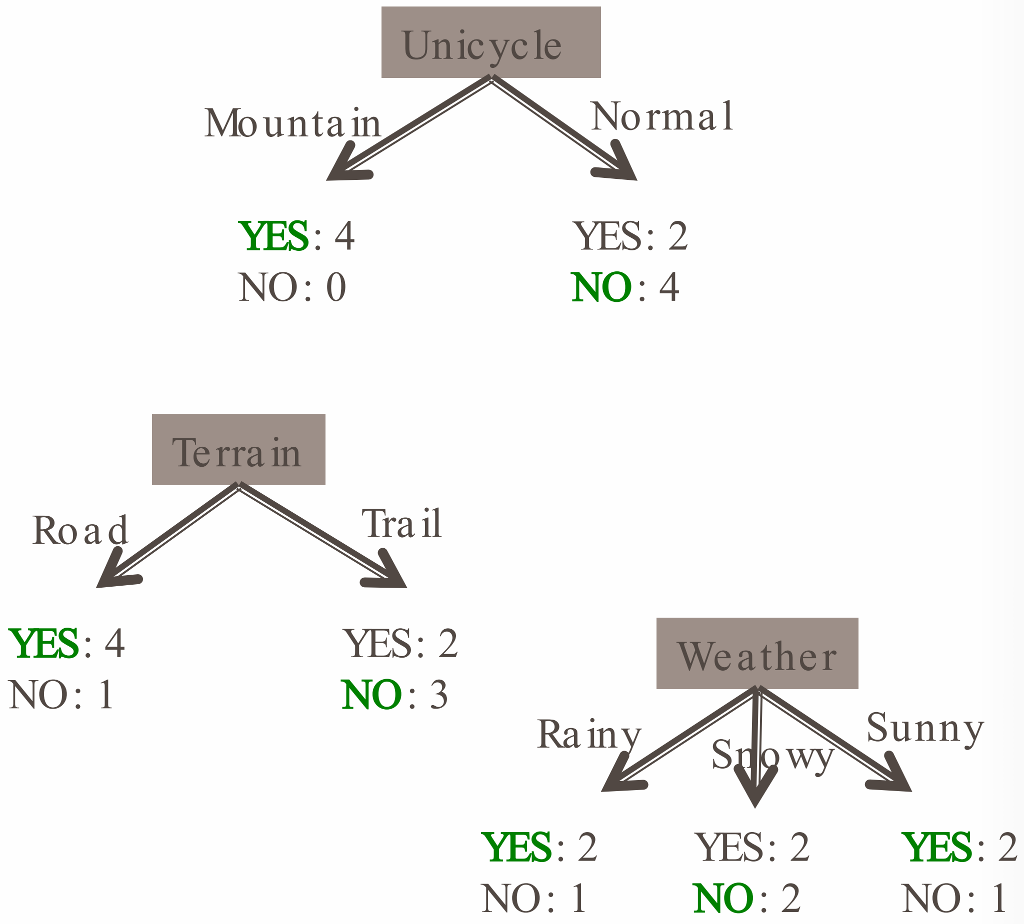
- GINI Impurity

$$I_G(t) = \sum_{i=1}^{c} p(i|t)(1 - p(i|t)) = 1 - \sum_{i=1}^{c} p(i|t)^2$$

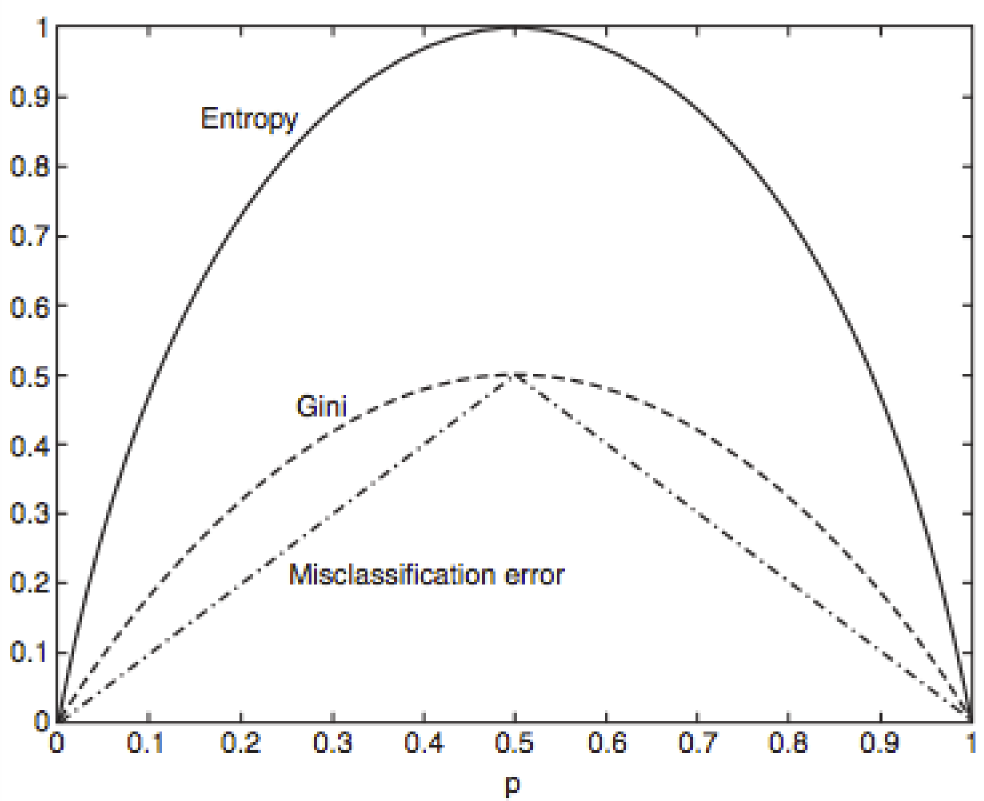
- Which p(i|t) means that given a node in which the #total samples is *t* and #i-th class samples is i
  - It actually can be calculated by normalization * (p/t) + (n/t)

Other metric: Impurity

- Total: 10 samples
  - Yes: 6 samples, No : 4 samples
- g(p,n)=I(6,4)=0.48
- E(A) is the subtotal of each node
  - I_Unicycle(p, n) = I(4,0) + I(2,4)=0.26
  - I_terrain(p,n)=I(4, 1)+I(2,3)=0.399
  - I_Weather(p,n)=g (2,1)+g (2,2)+g(2,1)=0.466
- Apparently, Unicycle is also a good one



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