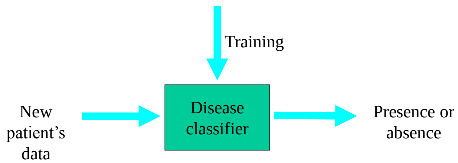
Example: Disease Diagnosis

Database of medical records

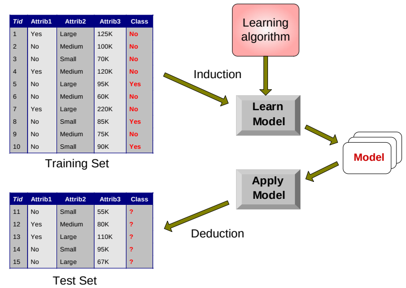
Patient 1's data   Absence

Patient 2's data   Presence

…   …



General Approach for Building Classification Model



Tid	Attrib1	Attrib2	Attrib3	Class
1	Yes	Large	125K	No
2	No	Medium	100K	No
3	No	Small	70K	No
4	Yes	Medium	120K	No
5	No	Large	95K	Yes
6	No	Medium	60K	No
7	Yes	Large	220K	No
8	No	Small	85K	Yes
9	No	Medium	75K	No
10	No	Small	90K	Yes

Training Set

Tid	Attrib1	Attrib2	Attrib3	Class
11	No	Small	55K	?
12	Yes	Medium	80K	?
13	Yes	Large	110K	?
14	No	Small	95K	?
15	No	Large	67K	?

Test Set

k Nearest-Neighbor Classification

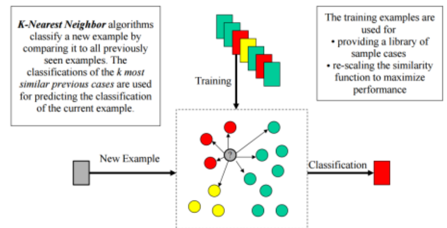
Rote Learning

Day	Temperature	Outlook	Humidity	Windy	Play Golf?
07-05	hot	sunny	high	false	no
07-06	hot	sunny	high	true	no
07-07	hot	overcast	high	false	yes
07-09	cool	rain	normal	false	yes
07-10	cool	overcast	normal	true	yes
07-12	mild	sunny	high	false	no
07-14	cool	sunny	normal	false	yes
07-15	mild	rain	normal	false	yes
07-20	mild	sunny	normal	true	yes
07-21	mild	overcast	high	true	yes
07-22	hot	overcast	normal	false	yes
07-23	mild	rain	high	true	no
07-26	cool	rain	normal	true	no
07-30	mild	rain	high	false	yes
today	cool	sunny	normal	false	yes

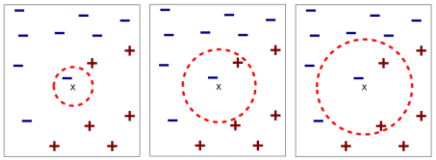
Nearest Neighbor Classification

Day	Temperature	Outlook	Humidity	Windy	Play Golf?
07-05	hot	sunny	high	false	no
07-06	hot	sunny	high	true	no
07-07	hot	overcast	high	false	yes
07-09	cool	rain	normal	false	yes
07-10	cool	overcast	normal	true	yes
07-12	mild	sunny	high	false	no
07-14	cool	sunny	normal	false	yes
07-15	mild	rain	normal	false	yes
07-20	mild	sunny	normal	true	yes
07-21	mild	overcast	high	true	yes
07-22	hot	overcast	normal	false	yes
07-23	mild	rain	high	true	no
07-26	cool	rain	normal	true	no
12-30	mild	rain	high	false	yes
tomorrow	mild	sunny	normal	false	yes

Nearest Neighbor Classifier



Nearest Neighbor Classifier



(a) 1-nearest neighbor (b) 2-nearest neighbor (c) 3-nearest neighbor

k nearest neighbors of an example x are the data points that have the k smallest distances to x

Classification

- The predicted class is determined from the nearest neighbor list
- Take the majority vote of class labels among the k-nearest neighbors

Distance Functions

- Computes the distance between two examples
  - so that we can find the "nearest neighbor" to a given example
- General Idea:
  - reduce the distance $d(x_1, x_2)$ of two examples to the distances $d_A(v_1, v_2)$ between two values for attribute A
- Popular choices
  - Euclidean Distance:
    - straight-line between two points

$$d(x_1, x_2) = \sqrt{\sum_A d_A(v_{1,A}, v_{2,A})^2}$$

  - Manhattan or City-block Distance:
    - sum of axis-parallel line segments

$$d(x_1, x_2) = \sum_A d_A(v_{1,A}, v_{2,A})$$

Distance Functions for Numerical Attributes

- Numerical Attributes:
  - distance between two attribute values

$$d_A(v_1, v_2) = |v_1 - v_2|$$

- Normalization:
  - Different attributes are measured on different scales
    → values need to be normalized in [0,1]:

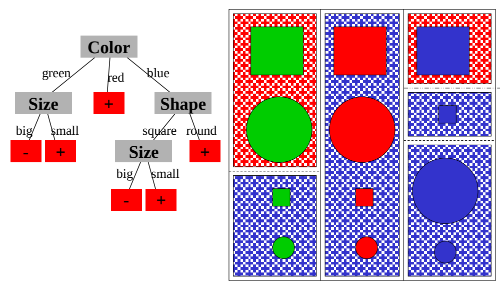
$$\hat{v}_i = \frac{v_i - \min v_j}{\max v_j - \min v_j}$$

  - Note:
    - This normalization assumes a (roughly) uniform distribution of attribute values
    - For other distributions, other normalizations might be preferable
      - e.g.: logarithmic for salaries?

Learning and Decision Trees

- Based on a set of attributes as input, predicted output value, the *decision* is learned
- Boolean or binary classification
  - output values are true or false
  - The simplest case.
- making decisions
  - a sequence of test is performed, testing the value of one of the attributes in each step
  - when a leaf node is reached, its value is returned
  - good correspondence to human decision-making

Decision tree-induced partition – example



Learning decision trees

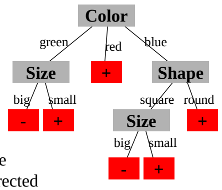
•Goal: Build a **decision tree** to classify examples as positive or negative instances using supervised learning from a training set

•A **decision tree** is a tree where

– each non-leaf node has associated with it an attribute (feature)

–each leaf node has associated with it a classification (+ or -)

–each arc has associated with it one of the possible values of the attribute at the node from which the arc is directed



•Generalization: allow for >2 classes

–e.g., for stocks, classify into {sell, hold, buy}

Types of Variables

- *Numerical*: Domain is ordered and can be represented on the real line (e.g., age, income)
- *Nominal* or *categorical*: Domain is a finite set without any natural ordering (e.g., occupation, marital status, race)
- *Ordinal*: Domain is ordered, but absolute differences between values is unknown (e.g., preference scale, severity of an injury)

Internal Nodes

- Each internal node has an associated *splitting predicate.* Most common are binary predicates.
  Example predicates:
  - Age <= 20
  - Profession in {student, teacher}
  - 5000*Age + 3*Salary – 10000 > 0

Learning decision trees

Problem: decide whether to wait for a table at a restaurant, based on the following attributes:

1. Alternate: is there an alternative restaurant nearby?
2. Bar: is there a comfortable bar area to wait in?
3. Fri/Sat: is today Friday or Saturday?
4. Hungry: are we hungry?
5. Patrons: number of people in the restaurant (None, Some, Full)
6. Price: price range ($, $$, $$$)
7. Raining: is it raining outside?
8. Reservation: have we made a reservation?
9. Type: kind of restaurant (French, Italian, Thai, Burger)
10. WaitEstimate: estimated waiting time (0-10, 10-30, 30-60, >60)

Attribute-based representations

- Examples described by attribute values
- E.g., situations where I will/won't wait for a table:

Example	Attributes										Target
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*Wait*
X_1	T	F	F	T	Some	$$$	F	T	French	0–10	T
X_2	T	F	F	T	Full	$	F	F	Thai	30–60	F
X_3	F	T	F	F	Some	$	F	F	Burger	0–10	T
X_4	T	F	T	T	Full	$	F	F	Thai	10–30	T
X_5	T	F	T	F	Full	$$$	F	T	French	>60	F
X_6	F	T	F	T	Some	$$	T	T	Italian	0–10	T
X_7	F	T	F	F	None	$	T	F	Burger	0–10	F
X_8	F	F	F	T	Some	$$	T	T	Thai	0–10	T
X_9	F	T	T	F	Full	$	T	F	Burger	>60	F
X_{10}	T	T	T	T	Full	$$$	F	T	Italian	10–30	F
X_{11}	F	F	F	F	None	$	F	F	Thai	0–10	F
X_{12}	T	T	T	T	Full	$	F	F	Burger	30–60	T

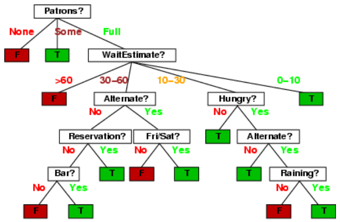
- Classification of examples is positive (T) or negative (F)

Restaurant Sample Set

Example				Attributes						Goal	Exa
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
X1	Yes	No	No	Yes	Some	$$$	No	Yes	French	0-10	Yes	X1
X2	Yes	No	No	Yes	Full	$	No	No	Thai	30-60	No	X2
X3	No	Yes	No	No	Some	$	No	No	Burger	0-10	Yes	X3
X4	Yes	No	Yes	Yes	Full	$	No	No	Thai	10-30	Yes	X4
X5	Yes	No	Yes	No	Full	$$$	No	Yes	French	>60	No	X5
X6	No	Yes	No	Yes	Some	$$	Yes	Yes	Italian	0-10	Yes	X6
X7	No	Yes	No	No	None	$	Yes	No	Burger	0-10	No	X7
X8	No	No	No	Yes	Some	$$	Yes	Yes	Thai	0-10	Yes	X8
X9	No	Yes	Yes	No	Full	$	Yes	No	Burger	>60	No	X9
X10	Yes	Yes	Yes	Yes	Full	$$$	No	Yes	Italian	10-30	No	X10
X11	No	No	No	No	None	$	No	No	Thai	0-10	No	X11
X12	Yes	Yes	Yes	Yes	Full	$	No	No	Burger	30-60	Yes	X12

Decision trees

- One possible representation for hypotheses
- it is a tree for deciding whether to wait:



Learning Decision Trees

- problem: find a decision tree that agrees with the training set
- trivial solution: construct a tree with one branch for each sample of the training set
  - works perfectly for the samples in the training set
  - may not work well for new samples (generalization)
  - results in relatively large trees
- better solution: find a concise tree that still agrees with all samples
  - corresponds to the simplest hypothesis that is consistent with the training set

Restaurant Sample Set

Example					Attributes						Goal	Exar
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*	
X1	Yes	No	No	Yes	Some	$$$	No	Yes	French	0-10	Yes	X1
X2	Yes	No	No	Yes	Full	$	No	No	Thai	30-60	No	X2
X3	No	Yes	No	No	Some	$	No	No	Burger	0-10	Yes	X3
X4	Yes	No	Yes	Yes	Full	$	No	No	Thai	10-30	Yes	X4
X5	Yes	No	Yes	No	Full	$$$	No	Yes	French	>60	No	X5
X6	No	Yes	No	Yes	Some	$$	Yes	Yes	Italian	0-10	Yes	X6
X7	No	Yes	No	No	None	$	Yes	No	Burger	0-10	No	X7
X8	No	No	No	Yes	Some	$$	Yes	Yes	Thai	0-10	Yes	X8
X9	No	Yes	Yes	No	Full	$	Yes	No	Burger	>60	No	X9
X10	Yes	Yes	Yes	Yes	Full	$$$	No	Yes	Italian	10-30	No	X10
X11	No	No	No	No	None	$	No	No	Thai	0-10	No	X11
X12	Yes	Yes	Yes	Yes	Full	$	No	No	Burger	30-60	Yes	X12

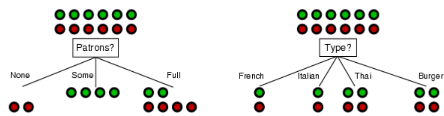
Restaurant Sample Set

Example					Attributes						Goal	Exar
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*	
X1	Yes	No	No	Yes	Some	$$$	No	Yes	French	0-10	Yes	X1
X2	Yes	No	No	Yes	Full	$	No	No	Thai	30-60	No	X2
X3	No	Yes	No	No	Some	$	No	No	Burger	0-10	Yes	X3
X4	Yes	No	Yes	Yes	Full	$	No	No	Thai	10-30	Yes	X4
X5	Yes	No	Yes	No	Full	$$$	No	Yes	French	>60	No	X5
X6	No	Yes	No	Yes	Some	$$	Yes	Yes	Italian	0-10	Yes	X6
X7	No	Yes	No	No	None	$	Yes	No	Burger	0-10	No	X7
X8	No	No	No	Yes	Some	$$	Yes	Yes	Thai	0-10	Yes	X8
X9	No	Yes	Yes	No	Full	$	Yes	No	Burger	>60	No	X9
X10	Yes	Yes	Yes	Yes	Full	$$$	No	Yes	Italian	10-30	No	X10
X11	No	No	No	No	None	$	No	No	Thai	0-10	No	X11
X12	Yes	Yes	Yes	Yes	Full	$	No	No	Burger	30-60	Yes	X12

- select best attribute
  - candidate 1: *Pat* — *Some* and *None* in agreement with goal
  - candidate 2: *Type* — No values in agreement with goal

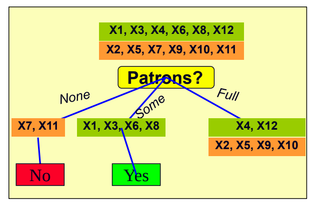
Choosing an attribute

- Idea: a good attribute splits the examples into subsets that are (ideally) "all positive" or "all negative"



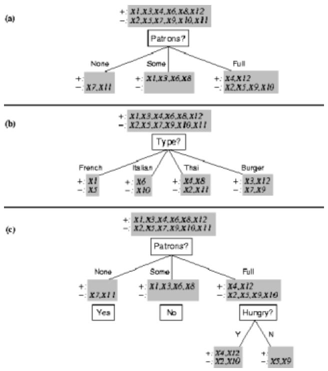
- *Patrons?* is a better choice

Partial Decision Tree



- Patrons needs further discrimination only for the Full value
- None and Some agree with the Will Wait goal predicate
- the next step will be performed on the remaining samples for the Full value of Patrons

Splitting examples by testing attributes

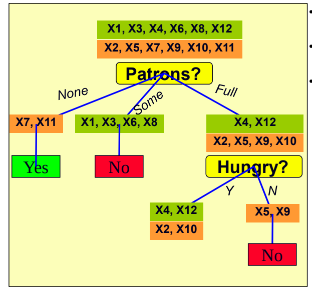


Restaurant Sample Set

Example					Attributes						Goal	Exar
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*	
X2	Yes	No	No	Yes	Full	$	No	No	Thai	30-60	No	X2
X4	Yes	No	Yes	Yes	Full	$	No	No	Thai	10-30	Yes	X4
X5	Yes	No	Yes	No	Full	$$$	No	Yes	French	>60	No	X5
X9	No	Yes	Yes	No	Full	$	Yes	No	Burger	>60	No	X9
X10	Yes	Yes	Yes	Yes	Full	$$$	No	Yes	Italian	10-30	No	X10
X12	Yes	Yes	Yes	Yes	Full	$	No	No	Burger	30-60	Yes	X12

- select next best attribute
  - candidate 1: *Hungry* — *No* in agreement with goal
  - candidate 2: *Type* — No values in agreement with goal

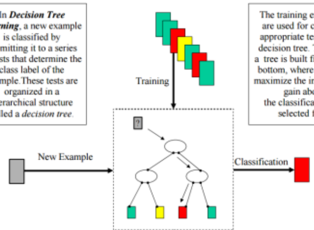
Partial Decision Tree



- *Hungry* needs further discrimination only for the *Yes* value
- *No* agrees with the *WillWait* goal predicate
- the next step will be performed on the remaining samples for the *Yes* value of *Hungry*

Decision Tree Learning

In ***Decision Tree Learning***, a new example is classified by submitting it to a series of tests that determine the class label of the example. These tests are organized in a hierarchical structure called a *decision tree*.



The training examples are used for choosing appropriate tests in the decision tree. Typically, a tree is built from top to bottom, where tests that maximize the information gain about the classification are selected first.

Finding ‘compact’ decision trees

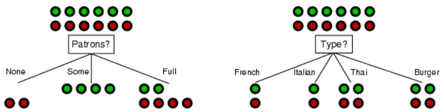
- Finding the smallest decision tree is difficult.
- There are heuristics that find reasonable decision trees in most practical cases.
- Idea: test the most important attribute first
  - attribute that makes the most difference for the classification of an example
  - hopefully will yield the correct classification with few tests

Choosing the best attribute

- Key problem: choosing which attribute to split a given set of examples
- Some possibilities are:
  - **Random:** Select any attribute at random
  - **Least-Values:** Choose the attribute with the smallest number of possible values (e.g. Hungry)
  - **Most-Values:** Choose the attribute with the largest number of possible values (e.g. Type)
  - **Max-Gain:** Choose the attribute that has the largest expected *information gain*–i.e., attribute that results in smallest expected size of sub-trees rooted at its children
- The ID3 algorithm uses the Max-Gain method of selecting the best attribute

Choosing an attribute

Idea: a good attribute splits the examples into subsets that are (ideally) “all positive” or “all negative”



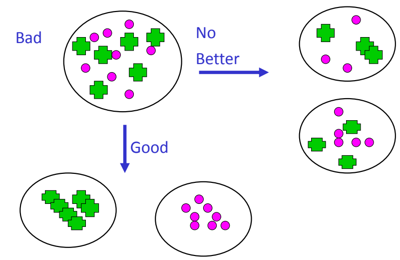
Which is better: *Patrons?* or *Type?*

Decision tree learning

- Aim: find a small tree consistent with the training examples
- Idea: (recursively) choose "most significant" attribute as root of (sub)tree

```
function DTL(examples, attributes, default) returns a decision tree
   if examples is empty then return default
   else if all examples have the same classification then return the classification
   else if attributes is empty then return MODE(examples)
   else
      best ← CHOOSE-ATTRIBUTE(attributes, examples)
      tree ← a new decision tree with root test best
      for each value v_i of best do
         examples_i ← {elements of examples with best = v_i}
         subtree ← DTL(examples_i, attributes − best, MODE(examples))
         add a branch to tree with label v_i and subtree subtree
      return tree
```

Disorder is bad
Homogeneity is good



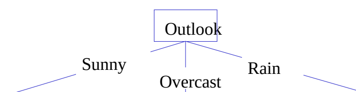
Disorder is bad
Homogeneity is good

- We want a measure that prefers attributes that have a high degree of “order”:
  - Maximum order: All examples are of the same class
  - Minimum order: All classes are equally likely
- Entropy is a measure for (un-)orderedness
  - all examples of the same class → no information
- ID3 splits attributes based on their *entropy*.

Variable Quality Measures

- Let S be a sample of training instances and p_j be the proportions of instances of class j (j=1,…,J) in S.
- Define an **impurity** measure $I(S)$ that satisfies:
  - $I(S)$ is minimum only when p_i=1 and p_j=0 for $j \neq i$ (all objects are of the same class);
  - $I(S)$ is maximum only when p_j =1/J (there is exactly the same number of objects of all classes);
  - $I(S)$ is symmetric with respect to $p_1,\ldots,p_J$;

Variable Quality Measures



Outlook	Temp	Hum	Wind	Play
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Sunny	Mild	High	Weak	No
Sunny	Cool	Normal	Weak	Yes
Sunny	Mild	Normal	Strong	Yes

Outlook	Temp	Hum	Wind	Play
Overcast	Hot	High	Weak	Yes
Overcast	Cool	Normal	Strong	Yes

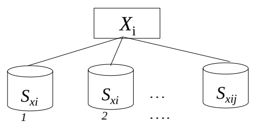
Outlook	Temp	Hum	Wind	Play
Rain	Mild	High	Weak	Yes
Rain	Cool	Normal	Weak	Yes
Rain	Cool	Normal	Strong	No
Rain	Mild	Normal	Weak	Yes
Rain	Mild	High	Strong	No

Reduction of Impurity: Discrete Variables

- The “best” variable is the variable X_i that determines a split maximizing the expected reduction of impurity:

$$\Delta I(S, X_i) = I(S) - \sum_j \frac{|S_{x_{ij}}|}{|S|} I(S_{xij})$$

where S_{xij} is the subset of instances from S s.t. $X_i = x_{ij}$.



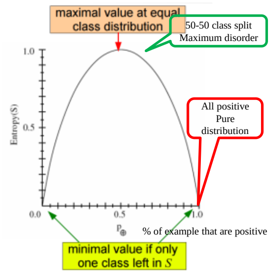
Entropy (for two classes)

- S is a set of examples
- $p_\oplus$ is the proportion of examples in class $\oplus$
- $p_\ominus = 1 - p_\oplus$ is the proportion of examples in class $\ominus$

Entropy:

$$E(S) = -p_\oplus \cdot \log_2 p_\oplus - p_\ominus \cdot \log_2 p_\ominus$$

- Interpretation:
  - amount of unorderedness in the class distribution of S



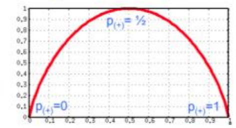
Entropy

- Entropy: $H(S) = -p_{(+)} \log_2 p_{(+)} - p_{(-)} \log_2 p_{(-)}$ bits
  - S … subset of training examples
  - $p_{(+)}$ / $p_{(-)}$ … % of positive / negative examples in S
- Interpretation: assume item X belongs to S
  - how many bits need to tell if X positive or negative
- impure (3 yes / 3 no):

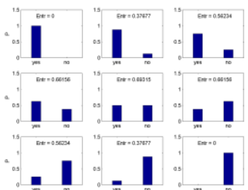
$$H(S) = -\frac{3}{6}\log_2\frac{3}{6} - \frac{3}{6}\log_2\frac{3}{6} = 1 \text{ bits}$$

- pure set (4 yes / 0 no):

$$H(S) = -\frac{4}{4}\log_2\frac{4}{4} - \frac{0}{4}\log_2\frac{0}{4} = 0 \text{ bits}$$



@ Victor Lavrenko



The entropy is maximal when all possibilities are equally likely.

The goal of the decision tree is to decrease the entropy in each node.

Entropy is zero in a pure ”yes” node (or pure ”no” node).

Attribution selection criteria:

- Create pure nodes whenever possible
- If pure nodes are not possible, choose the split that leads to the largest decrease in entropy.

$$\boxed{\text{Entropy} = -P(yes)\ln[P(yes)] - P(no)\ln[P(no)]}$$

General form: $\text{Entropy} = -\sum_i P(v_i)\ln[P(v_i)]$

Information Gain

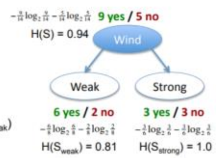
- Want many items in pure sets
- Expected drop in entropy after split:

$$Gain(S,A) = H(S) - \sum_{V \in Values(A)} \frac{|S_V|}{|S|} H(S_V)$$

V … possible values of A
S … set of examples {X}
S_v … subset where $X_A = V$

- Mutual Information
  - between attribute A and class labels of S

Gain (S, Wind)
= H(S) – $^8/_{14}$ H(S_{weak}) – $^6/_{14}$ H(S_{weak})
= 0.94 – $^8/_{14}$ * 0.81 – $^6/_{14}$ * 1.0
= 0.049



@ Victor Lavrenko

Decision tree learning example

10 attributes:

1. **Alternate:** Is there a suitable alternative restaurant nearby? {yes,no}
2. **Bar:** Is there a bar to wait in? {yes,no}
3. **Fri/Sat:** Is it Friday or Saturday? {yes,no}
4. **Hungry:** Are you hungry? {yes,no}
5. **Patrons:** How many are seated in the restaurant? {none, some, full}
6. **Price:** Price level {$,$$,$$$}
7. **Raining:** Is it raining? {yes,no}
8. **Reservation:** Did you make a reservation? {yes,no}
9. **Type:** Type of food {French,Italian,Thai,Burger}
10. **Wait:** {0-10 min, 10-30 min, 30-60 min, >60 min}

Decision tree learning example

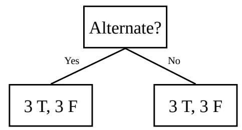
Example	Attributes										Target
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
X_1	T	F	F	T	Some	$$$	F	T	French	0–10	T
X_2	T	F	F	T	Full	$	F	F	Thai	30–60	F
X_3	F	T	F	F	Some	$	F	F	Burger	0–10	T
X_4	T	F	T	T	Full	$	F	F	Thai	10–30	T
X_5	T	F	T	F	Full	$$$	F	T	French	>60	F
X_6	F	T	F	T	Some	$$	T	T	Italian	0–10	T
X_7	F	T	F	F	None	$	T	F	Burger	0–10	F
X_8	F	F	F	T	Some	$$	T	T	Thai	0–10	T
X_9	F	T	T	F	Full	$	T	F	Burger	>60	F
X_{10}	T	T	T	T	Full	$$$	F	T	Italian	10–30	F
X_{11}	F	F	F	F	None	$	F	F	Thai	0–10	F
X_{12}	T	T	T	T	Full	$	F	F	Burger	30–60	T

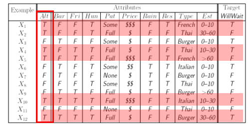
T = True, F = False

6 True,
6 False

$$\text{Entropy} = -\left(\tfrac{6}{12}\right)\ln\left(\tfrac{6}{12}\right) - \left(\tfrac{6}{12}\right)\ln\left(\tfrac{6}{12}\right) = 0.30$$

Decision tree learning example

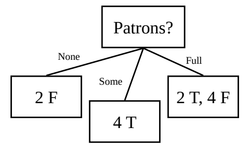


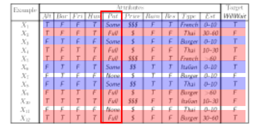


$$\text{Entropy} = \frac{6}{12}\left[-\left(\tfrac{3}{6}\right)\ln\left(\tfrac{3}{6}\right) - \left(\tfrac{3}{6}\right)\ln\left(\tfrac{3}{6}\right)\right] + \frac{6}{12}\left[-\left(\tfrac{3}{6}\right)\ln\left(\tfrac{3}{6}\right) - \left(\tfrac{3}{6}\right)\ln\left(\tfrac{3}{6}\right)\right] = 0.30$$

Entropy decrease = 0.30 – 0.30 = 0

Decision tree learning example

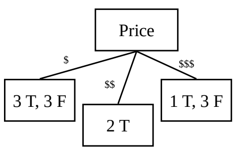


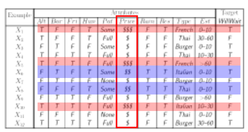


$$\text{Entropy} = \frac{2}{12}\left[-\left(\tfrac{0}{2}\right)\ln\left(\tfrac{0}{2}\right) - \left(\tfrac{2}{2}\right)\ln\left(\tfrac{2}{2}\right)\right] + \frac{4}{12}\left[-\left(\tfrac{4}{4}\right)\ln\left(\tfrac{4}{4}\right) - \left(\tfrac{0}{4}\right)\ln\left(\tfrac{0}{4}\right)\right]$$
$$+ \frac{6}{12}\left[-\left(\tfrac{2}{6}\right)\ln\left(\tfrac{2}{6}\right) - \left(\tfrac{4}{6}\right)\ln\left(\tfrac{4}{6}\right)\right] = 0.14$$

Entropy decrease = 0.30 – 0.14 = 0.16

Decision tree learning example





$$\text{Entropy} = \frac{6}{12}\left[-\left(\tfrac{3}{6}\right)\ln\left(\tfrac{3}{6}\right) - \left(\tfrac{3}{6}\right)\ln\left(\tfrac{3}{6}\right)\right] + \frac{2}{12}\left[-\left(\tfrac{2}{2}\right)\ln\left(\tfrac{2}{2}\right) - \left(\tfrac{0}{2}\right)\ln\left(\tfrac{0}{2}\right)\right]$$
$$+ \frac{4}{12}\left[-\left(\tfrac{1}{4}\right)\ln\left(\tfrac{1}{4}\right) - \left(\tfrac{3}{4}\right)\ln\left(\tfrac{3}{4}\right)\right] = 0.23$$

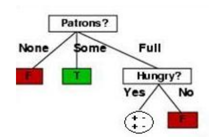
Entropy decrease = 0.30 – 0.23 = 0.07

ID3 Algorithm

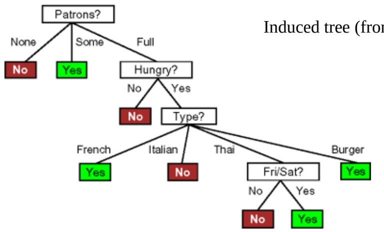
- A greedy algorithm for decision tree construction developed by Ross Quinlan circa 1987
- Top-down construction of decision tree by recursively selecting "best attribute" to use at the current node in tree
  - Once attribute is selected for current node, generate child nodes, one for each possible value of selected attribute
  - Partition examples using the possible values of this attribute, and assign these subsets of the examples to the appropriate child node
  - Repeat for each child node until all examples associated with a node are either all positive or all negative

Next step

Given *Patrons* as root node, the next attribute chosen is *Hungry?*



Decision tree learning example



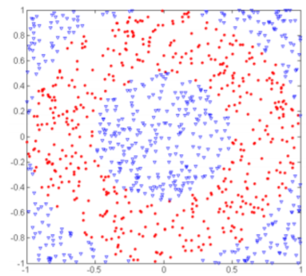
Induced tree (from examples)

ID3-Disadvantages

Error Propagation: Since decision trees work by a series of local decisions, what happens when one of these local decisions is wrong?

- Every decision from that point on may be wrong
- We may never return to the correct path of the tree

Underfitting and Overfitting (Example)



500 circular and 500 triangular data points.

Circular points:

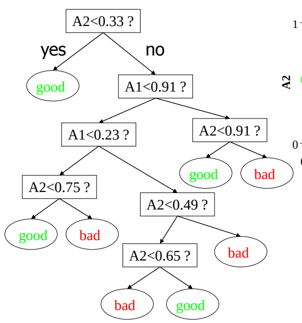
$0.5 \leq \mathrm{sqrt}(x_1^2+x_2^2) \leq 1$

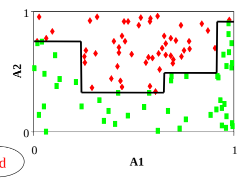
Triangular points:

$\mathrm{sqrt}(x_1^2+x_2^2) > 0.5$ or

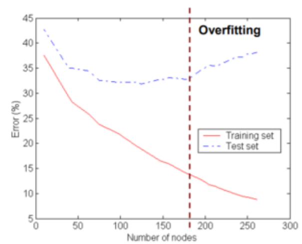
$\mathrm{sqrt}(x_1^2+x_2^2) < 1$

Decision Trees are Non-linear Classifiers





Underfitting and Overfitting



Underfitting: when model is too simple, both training and test errors are large

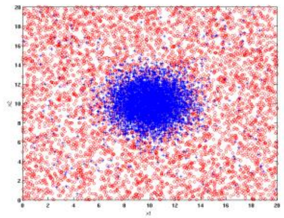
Overfitting

- Definition: If your machine learning algorithm fits noise (i.e. pays attention to parts of the data that are irrelevant) it is overfitting.
- Fact (theoretical and empirical): If your machine learning algorithm is overfitting then it may perform less well on test set data.

What’s Overfitting?

- **Overfitting =** Given a hypothesis space H, a hypothesis h∈H is said to overfit the training data if there exists some alternative hypothesis h’∈H, such that
  1. **h has smaller error than h’ over the training examples, but**
  2. **h’ has a smaller error than h over the entire distribution of instances.**

Example Data Set



Two class problem:

+ : 5200 instances

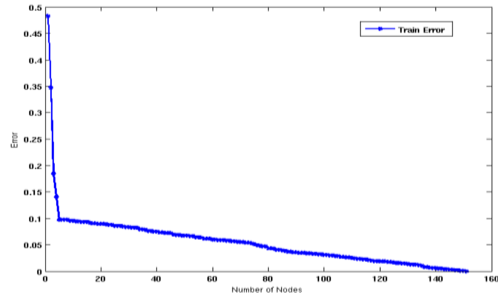
- 5000 instances generated from a Gaussian centered at (10,10)
- 200 noisy instances added

o : 5200 instances

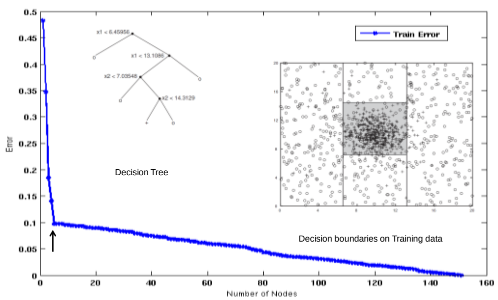
- Generated from a uniform distribution

10 % of the data used for training and 90% of the data used for testing

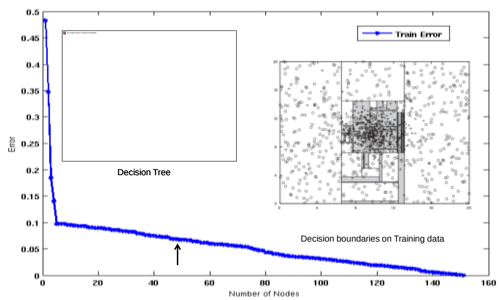
Increasing number of nodes in Decision Trees



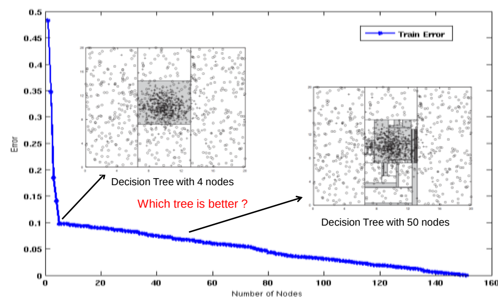
Decision Tree with 4 nodes



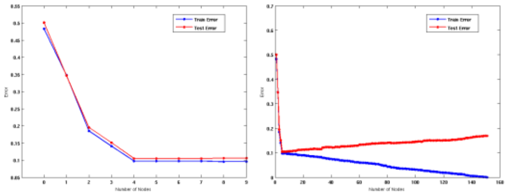
Decision Tree with 50 nodes



Which tree is better?



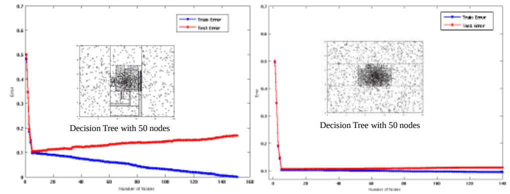
Model Overfitting



Underfitting: when model is too simple, both training and test errors are large

Overfitting: when model is too complex, training error is small but test error is large

Model Overfitting

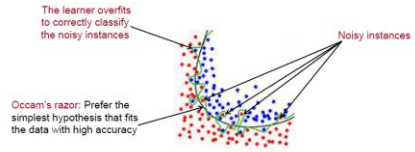


Using twice the number of data instances

- If training data is under-representative, testing errors increase and training errors decrease on increasing number of nodes
- Increasing the size of training data reduces the difference between training and testing errors at a given number of nodes

Why Does my Method Overfit ?

- **In domains with noise or uncertainty** the system may try to decrease the training error by completely fitting all the training examples



Model Selection: Using Validation Set

- Divide training data into two parts:
  - Training set:
    - use for model building
  - Validation set:
    - use for estimating generalization error
    - Note: validation set is not the same as test set
- Drawback:
  - Less data available for training

Avoiding overfitting

- Usually we do not know in advance which are the irrelevant variables
- …and it may depend on the context

  For example, if y = a AND b then b is an irrelevant variable only in the portion of the tree in which a=0

But we can use simple statistics to warn us that we might be overfitting.

Pruning

- Goal: Prevent overfitting to noise in the data
- Two strategies for “pruning” the decision tree:
  - *Postpruning* - take a fully-grown decision tree and discard unreliable parts
  - *Prepruning* - stop growing a branch when information becomes unreliable
- Postpruning preferred in practice—prepruning can “stop too early”

Do not include branches that fit data too specifically

Model Selection:

Incorporating Model Complexity

- Rationale: Occam's Razor
  - Given two models of similar generalization errors, one should prefer the simpler model over the more complex model
  - A complex model has a greater chance of being fitted accidentally by errors in data
  - Therefore, one should include model complexity when evaluating a model

Gen. Error(Model) = Train. Error(Model, Train. Data) + α x Complexity(Model)

How to Address Overfitting

- Pre-Pruning (Early Stopping Rule)
  - Stop the algorithm before it becomes a fully-grown tree
  - Typical stopping conditions for a node:
    - Stop if all instances belong to the same class
    - Stop if all the attribute values are the same
  - More restrictive conditions:
    - Stop if number of instances is less than some user-specified threshold
    - Stop if class distribution of instances are independent of the available features (e.g., using χ^2 test)
    - Stop if expanding the current node does not improve impurity measures (e.g., Gini or information gain).

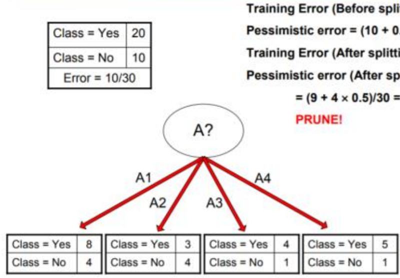
Prepruning

- Based on statistical significance test
  - Stop growing the tree when there is no *statistically significant* association between any attribute and the class at a particular node
- Most popular test: *chi-squared test*
- ID3 used chi-squared test in addition to information gain
  - Only statistically significant attributes were allowed to be selected by information gain procedure

Estimating Generalization Errors

- Re-substitution errors: error on training (Σ e(t))
- Generalization errors: error on testing (Σ e'(t))
- Methods for estimating generalization errors:
  - Optimistic approach: e'(t) = e(t)
  - Pessimistic approach:
    - For each leaf node: e'(t) = (e(t)+0.5)
    - Total errors: e'(T) = e(T) + N $\times$ 0.5 (N: number of leaf nodes)
    - For a tree with 30 leaf nodes and 10 errors on training (out of 1000 instances):
      Training error = 10/1000 = 1%
      Generalization error = (10 + 30$\times$0.5)/1000 = 2.5%
  - Reduced error pruning (REP):
    - uses validation data set to estimate generalization error

Example of Post-Pruning



Training Error (Before splitting) = 10/30

Pessimistic error = (10 + 0.5)/30 = 10.5/30

Training Error (After splitting) = 9/30

Pessimistic error (After splitting)

= (9 + 4 $\times$ 0.5)/30 = 11/30

PRUNE!

Missing as a separate value

- Missing value denoted "?" in C4.X
- Simple idea: treat missing as a separate value
- Q: When this is not appropriate?
- A: When values are missing due to different reasons
  - Example: field **IsPregnant**=missing for a male patient should be treated differently (no) than for a female patient of age 25 (unknown)

72

Handling Missing Attribute Values

- Missing values affect decision tree construction in three different ways:
  - Affects how impurity measures are computed
  - Affects how to distribute instance with missing value to child nodes
  - Affects how a test instance with missing value is classified

- Many possible approaches

 -Treat them as different values

 - Propogate the cases containing such values down the tree without considering them in the "Information Gain" calculation

Missing values - advanced

Split instances with missing values into pieces

- A piece going down a branch receives a weight proportional to the popularity of the branch
- weights sum to 1

- Info gain works with fractional instances
  - use sums of weights instead of counts
- During classification, split the instance into pieces in the same way
  - Merge probability distribution using weights

Computing Impurity Measure

Tid	Refund	Marital Status	Taxable Income	Class
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	?	Single	90K	Yes

Missing value

Before Splitting:

Entropy(Parent)
= -0.3 log(0.3)-(0.7)log(0.7) = 0.8813

	Class = Yes	Class = No
Refund=Yes	0	3
Refund=No	2	4
Refund=?	1	0

Split on Refund:

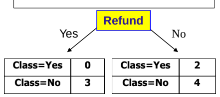
Entropy(Refund=Yes) = 0

Entropy(Refund=No)
= -(2/6)log(2/6) – (4/6)log(4/6) = 0.9183

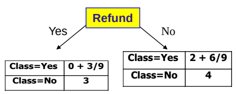
Entropy(Children)
= 0.3 (0) + 0.6 (0.9183) = 0.551

Gain = 0.9 × (0.8813 – 0.551) = 0.3303

Distribute Instances



Tid	Refund	Marital Status	Taxable Income	Class
10	?	Single	90K	Yes



Probability that Refund=Yes is 3/9

Probability that Refund=No is 6/9

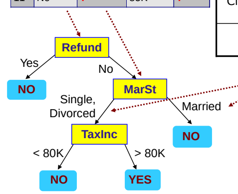
Assign record to the left child with weight = 3/9 and to the right child with weight = 6/9

Classify Instances

New record:

Tid	Refund	Marital Status	Taxable Income	Class
11	No	?	85K	?

	Married	Single	Divorced	Total
Class=No	3	1	0	4
Class=Yes	6/9	1	1	2.67
Total	3.67	2	1	6.67



Probability that Marital Status = Married is 3.67/6.67

Probability that Marital Status ={Single,Divorced} is 3/6.67

Numeric attributes

- Standard method: binary splits
  - E.g. temp < 45
- Unlike nominal attributes, every attribute has many possible split points
- Solution is straightforward extension:
  - Evaluate info gain (or other measure) for every possible split point of attribute
  - Choose "best" split point
  - Info gain for best split point is info gain for attribute
- Computationally more demanding

How to Specify Test Condition?

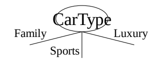
- Depends on attribute types
  - Nominal
  - Ordinal
  - Continuous
- Depends on number of ways to split
  - 2-way split
  - Multi-way split

Binary *vs.* multi-way splits

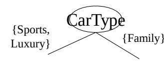
- Splitting (multi-way) on a nominal attribute exhausts all information in that attribute
  - Nominal attribute is tested (at most) once on any path in the tree
- Not so for binary splits on numeric attributes!
  - Numeric attribute may be tested several times along a path in the tree
- Disadvantage: tree is hard to read
- Remedy:
  - pre-discretize numeric attributes, *or*
  - use multi-way splits instead of binary ones

Splitting Based on Nominal Attributes

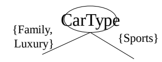
- Multi-way split: Use as many partitions as distinct values.



- Binary split: Divides values into two subsets.
  Need to find optimal partitioning.

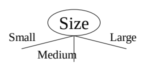


OR

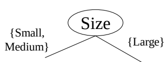


Splitting Based on Ordinal Attributes

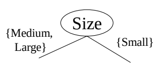
- Multi-way split: Use as many partitions as distinct values.



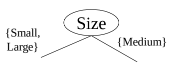
- Binary split: Divides values into two subsets.
  Need to find optimal partitioning.



OR



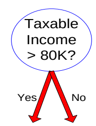
- What about this split?



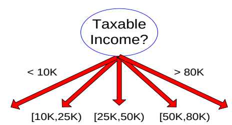
Splitting Based on Continuous Attributes

- Different ways of handling
  - Discretization to form an ordinal categorical attribute
    - Static – discretize once at the beginning
    - Dynamic – ranges can be found by equal interval bucketing, equal frequency bucketing (percentiles), or clustering.
  - Binary Decision: $(A < v)$ or $(A \geq v)$
    - consider all possible splits and finds the best cut
    - can be more compute intensive

Splitting Based on Continuous Attributes



(i) Binary split



(ii) Multi-way split

Deal with continuous data

- **When dealing with nominal data, W**e evaluated the gain for each possible value
- **In continuous data, we have infinite values. What should we do?**
  - Continuous-valued attributes may take infinite values, but we have a limited number of values in our instances (at most N if we have N instances)
- **Therefore, simulate that you have N nominal values**
  - Evaluate information gain for every possible split point of the Attribute Choose the best split point
  - The information gain of the attribute is the information gain of the best split

Example

Example

Outlook	Temperature	Humidity	Windy	Play
Sunny	85	85	False	No
Sunny	80	90	True	No
Overcast	83	86	False	Yes
Rainy	75	80	False	Yes
...	...	...	...	...

Continuous attributes

Split in continuous data

- Split on temperature attribute

64	65	68	69	70	71	72	72	75	75	80	81	83	85
Yes	No	Yes	Yes	Yes	No	No	Yes	Yes	Yes	No	Yes	Yes	No

- For example, in the above array of values the split is occurring between 71 and 72(N distinct values meaning at most N-1 splits)
- Of all such splits , the one with the best Information Gain is chosen for the node

Splitting Based on GINI

- Used in CART, SLIQ, SPRINT.
- When a node p is split into k partitions (children), the quality of split is computed as,

$$GINI_{split} = \sum_{i=1}^{k} \frac{n_i}{n} GINI(i)$$

where, n_i = number of records at child i,

n = number of records at node p.

Measure of Impurity: GINI

- Gini Index for a given node t :

$$GINI(t) = 1 - \sum_j [p(j \mid t)]^2$$

(NOTE: $p(j \mid t)$ is the relative frequency of class j at node t).

- Maximum $(1 - 1/n_c)$ when records are equally distributed among all classes, implying least interesting information
- Minimum (0.0) when all records belong to one class, implying most interesting information

C1	0
C2	6
Gini=0.000	

C1	1
C2	5
Gini=0.278	

C1	2
C2	4
Gini=0.444	

C1	3
C2	3
Gini=0.500	

Examples for computing GINI

$$GINI(t) = 1 - \sum_j [p(j \mid t)]^2$$

C1	0
C2	6

P(C1) = 0/6 = 0 P(C2) = 6/6 = 1

Gini = 1 – $P(C1)^2$ – $P(C2)^2$ = 1 – 0 – 1 = 0

C1	1
C2	5

P(C1) = 1/6 P(C2) = 5/6

Gini = 1 – $(1/6)^2$ – $(5/6)^2$ = 0.278

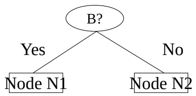
C1	2
C2	4

P(C1) = 2/6 P(C2) = 4/6

Gini = 1 – $(2/6)^2$ – $(4/6)^2$ = 0.444

Binary Attributes: Computing GINI Index

- Splits into two partitions
- Effect of Weighing partitions:
  - Larger and Purer Partitions are sought for.



	Parent
C1	**6**
C2	**6**
Gini = 0.500	

Gini(N1)
= 1 – $(5/6)^2$ – $(2/6)^2$
= 0.194

Gini(N2)
= 1 – $(1/6)^2$ – $(4/6)^2$
= 0.528

	N1	N2
C1	**5**	**1**
C2	**2**	**4**
Gini=0.333		

Gini(Children)
= 7/12 * 0.194 +
5/12 * 0.528
= 0.333

Categorical Attributes: Computing Gini Index

- For each distinct value, gather counts for each class in the dataset
- Use the count matrix to make decisions

Multi-way split

	CarType		
	Family	Sports	Luxury
C1	1	2	1
C2	4	1	1
Gini	0.393		

Two-way split
(find best partition of values)

	CarType	
	{Sports, Luxury}	{Family}
C1	3	1
C2	2	4
Gini	0.400	

	CarType	
	{Sports}	{Family, Luxury}
C1	2	2
C2	1	5
Gini	0.419	

Splitting Criteria based on Classification Error

- Classification error at a node t :

$$Error(t) = 1 - \max_i P(i \mid t)$$

- Measures misclassification error made by a node.
  - Maximum ($1 - 1/n_c$) when records are equally distributed among all classes, implying least interesting information
  - Minimum (0.0) when all records belong to one class, implying most interesting information

Examples for Computing Error

$$Error(t) = 1 - \max_i P(i \mid t)$$

C1	**0**
C2	**6**

P(C1) = 0/6 = 0 P(C2) = 6/6 = 1
Error = 1 – max (0, 1) = 1 – 1 = 0

C1	**1**
C2	**5**

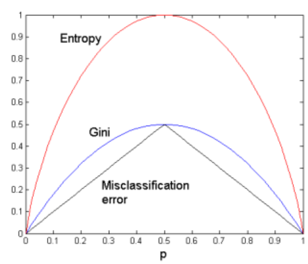
P(C1) = 1/6 P(C2) = 5/6
Error = 1 – max (1/6, 5/6) = 1 – 5/6 = 1/6

C1	**2**
C2	**4**

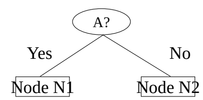
P(C1) = 2/6 P(C2) = 4/6
Error = 1 – max (2/6, 4/6) = 1 – 4/6 = 1/3

Comparison among Splitting Criteria

For a 2-class problem:



Misclassification Error vs Gini



	Parent
C1	**7**
C2	**3**
Gini = 0.42	

Gini(N1)
= 1 – $(3/3)^2$ – $(0/3)^2$
= 0

Gini(N2)
= 1 – $(4/7)^2$ – $(3/7)^2$
= 0.489

	N1	N2
C1	**3**	**4**
C2	**0**	**3**

Gini(Children)
= 3/10 * 0
+ 7/10 * 0.489
= 0.342

Model Evaluation

- Purpose:
  – To estimate performance of classifier on previously unseen data (test set)
- Holdout
  – Reserve k% for training and (100-k)% for testing
  – Random subsampling: repeated holdout
- Cross validation
  – Partition data into k disjoint subsets
  – k-fold: train on k-1 partitions, test on the remaining one
  – Leave-one-out: k=n

Cross-validation Example

- 3-fold cross-validation