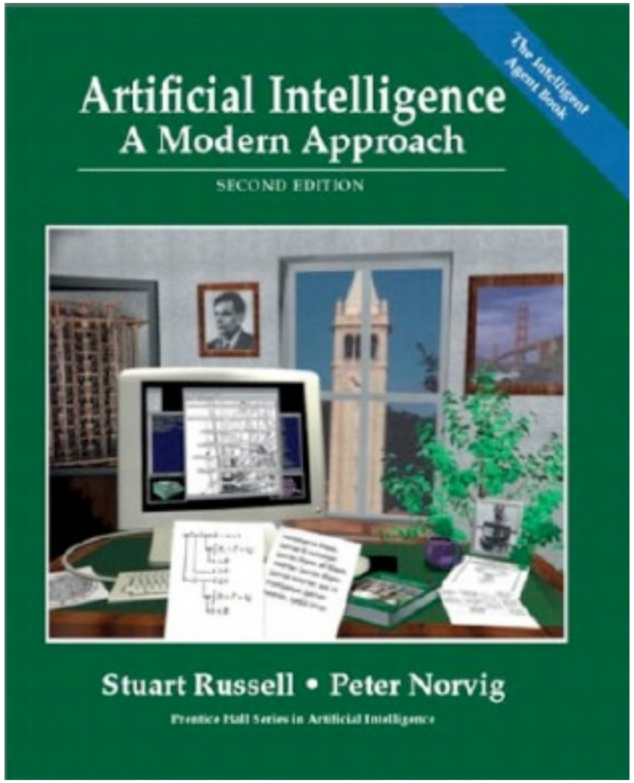
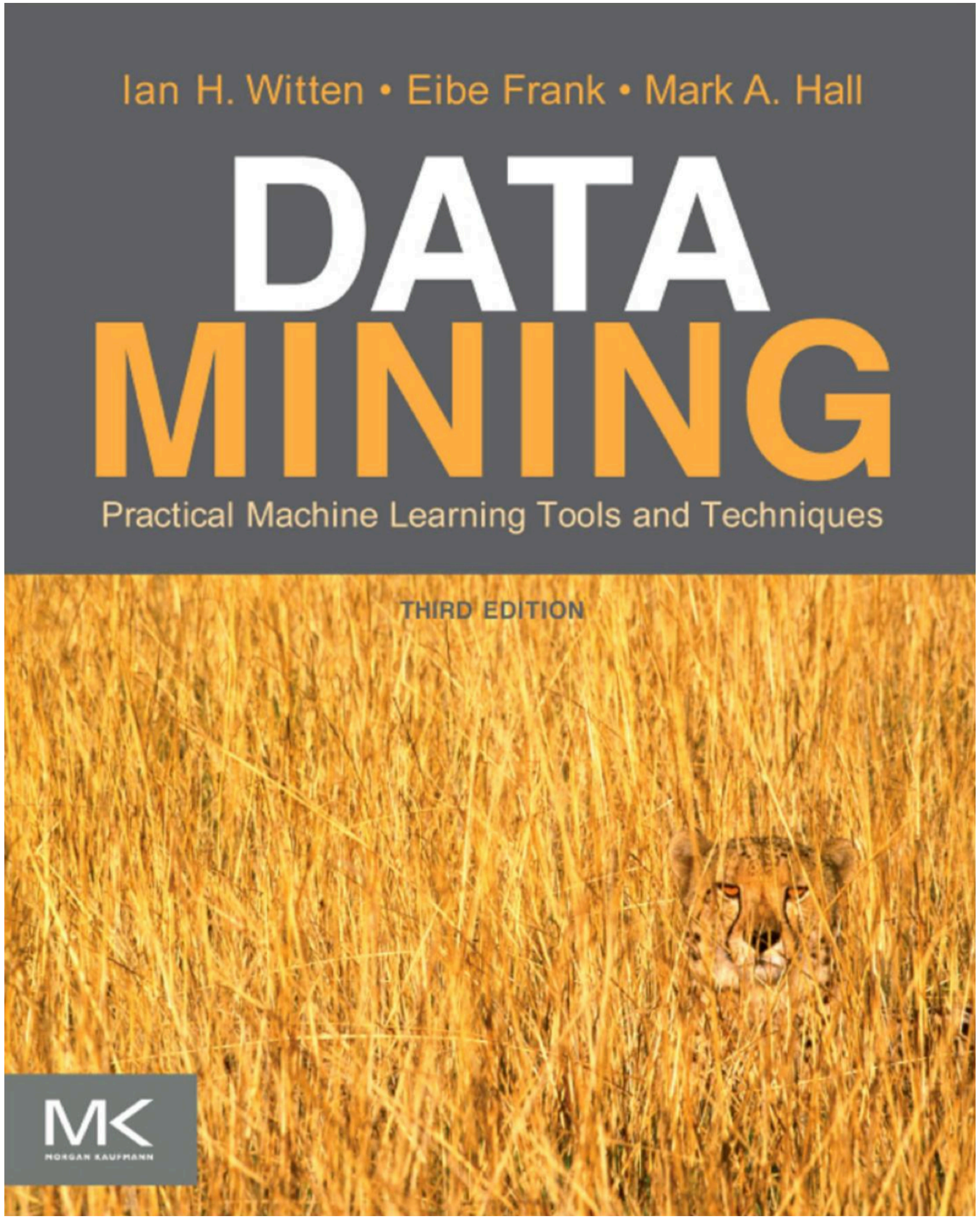
Inductive Learning

Chapter 18



Material adopted from
Yun Peng,
Chuck Dyer,
Gregory Piatetsky-Shapiro & Gary Parker

Chapters 3 and 4



Inductive Learning Framework

Induce a conclusion from the examples

- Raw input data from sensors are preprocessed to obtain a **feature vector**, X, that adequately describes all of the relevant features for classifying examples.
- Each x is a list of (attribute, value) pairs. For example,

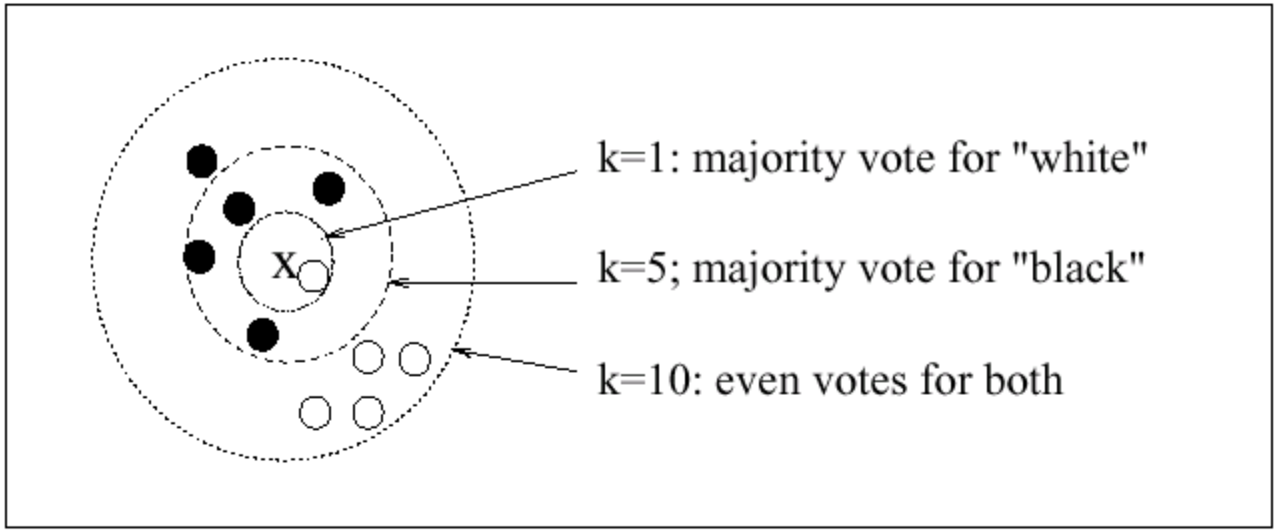
  X = [Person:Sue, EyeColor:Brown, Age:Young, Sex:Female]
- The number of attributes (aka features) is fixed (finite).
- Each attribute has a fixed, finite number of possible values.
- Each example can be interpreted as a point in an n-dimensional **feature space**, where n is the number of attributes.

Inductive Learning by Nearest-Neighbor Classification

- One simple approach to inductive learning is to save each training example as a point in feature space
- Classify a new example by giving it the same classification (+ or -) as its nearest neighbor in Feature Space.
  - A variation involves computing a weighted sum of class of a set of neighbors where the weights correspond to distances
- The problem with this approach is that it doesn't necessarily generalize well if the examples are not well *"clustered."*

KNN example

K-Nearest Neighbor using a *majority* voting scheme



Simplicity first

- Simple algorithms often work very well!
- There are many kinds of simple structure, eg:
  - One attribute does all the work
  - All attributes contribute equally & independently
  - A weighted linear combination might do
  - Instance-based: use a few prototypes
  - Use simple logical rules
- Success of method depends on the domain

Inferring rudimentary rules

- 1R: learns a 1-level decision tree
  - I.e., rules that all test one particular attribute
- Basic version
  - One branch for each value
  - Each branch assigns most frequent class
  - Error rate: proportion of instances that don't belong to the majority class of their corresponding branch
  - Choose attribute with lowest error rate

(*assumes nominal attributes*)

Evaluating the weather attributes

Classification

Outlook	Temp	Humidity	Windy	Play
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes
Rainy	Mild	High	False	Yes
Rainy	Cool	Normal	False	Yes
Rainy	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Sunny	Mild	High	False	No
Sunny	Cool	Normal	False	Yes
Rainy	Mild	Normal	False	Yes
Sunny	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Rainy	Mild	High	True	No

Attribute	Rules	Errors	Total errors
Outlook	Sunny → No	2/5	4/14
	Overcast → Yes	0/4	
	Rainy → Yes	2/5	
Temp	Hot → No*	2/4	5/14
	Mild → Yes	2/6	
	Cool → Yes	1/4	
Humidity	High → No	3/7	4/14
	Normal → Yes	1/7	
Windy	False → Yes	2/8	5/14
	True → No*	3/6	

* indicates a tie

Dealing with numeric attributes

- Discretize numeric attributes
- Divide each attribute's range into intervals
  - Sort instances according to attribute's values
  - Place breakpoints where the class changes (the majority class)
  - This minimizes the total error
- Example: *temperature* from weather data

```
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
```

The problem of overfitting

- This procedure is very sensitive to noise
  - One instance with an incorrect class label will probably produce a separate interval
- Also: *time stamp* attribute will have zero errors
- Simple solution:
  enforce minimum number of instances in majority class per interval

Discretization example

- Example (with min = 3):

```
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
```

- Final result for 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
```

With overfitting avoidance

- Resulting rule set:

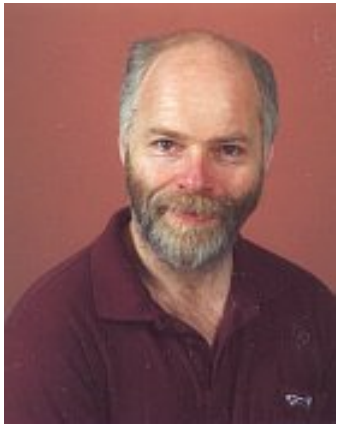
Attribute	Rules	Errors	Total errors
Outlook	Sunny → No	2/5	4/14
	Overcast → Yes	0/4	
	Rainy → Yes	2/5	
Temperature	≤ 77.5 → Yes	3/10	5/14
	> 77.5 → No*	2/4	
Humidity	≤ 82.5 → Yes	1/7	3/14
	> 82.5 and ≤ 95.5 → No	2/6	
	> 95.5 → Yes	0/1	
Windy	False → Yes	2/8	5/14
	True → No*	3/6	

Discussion of 1R

- 1R was described in a paper by Holte (1993)
  - Contains an experimental evaluation on 16 datasets (using *cross-validation* so that results were representative of performance on future data)
  - Minimum number of instances was set to 6 after some experimentation
  - 1R's simple rules performed not much worse than much more complex decision trees
- Simplicity first pays off!

Very Simple Classification Rules Perform Well on Most Commonly Used Datasets

Robert C. Holte, Computer Science Department, University of Ottawa



Classification:

Decision Trees

Outline

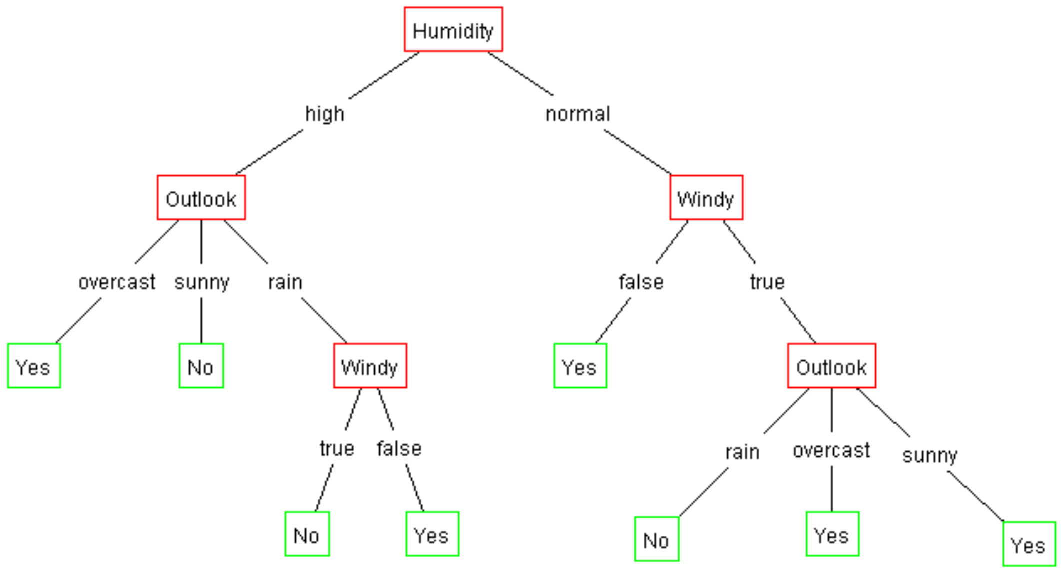
- Top-Down Decision Tree Construction
- Choosing the Splitting Attribute ????

DECISION TREE

- An internal node is a test on an attribute.
- A branch represents an outcome of the test, e.g., Color=red.
- A leaf node represents a class label or class label distribution.
- At each node, one attribute is chosen to split training examples into distinct classes as much as possible
- A new case is classified by following a matching path to a leaf node.

Example Tree for "Play?"

Outlook	Temperature	Humidity	Windy	Play?
sunny	hot	high	false	No
sunny	hot	high	true	No
overcast	hot	high	false	Yes
rain	mild	high	false	Yes
rain	cool	normal	false	Yes
rain	cool	normal	true	No
overcast	cool	normal	true	Yes
sunny	mild	high	false	No
sunny	cool	normal	false	Yes
rain	mild	normal	false	Yes
sunny	mild	normal	true	Yes
overcast	mild	high	true	Yes
overcast	hot	normal	false	Yes
rain	mild	high	true	No



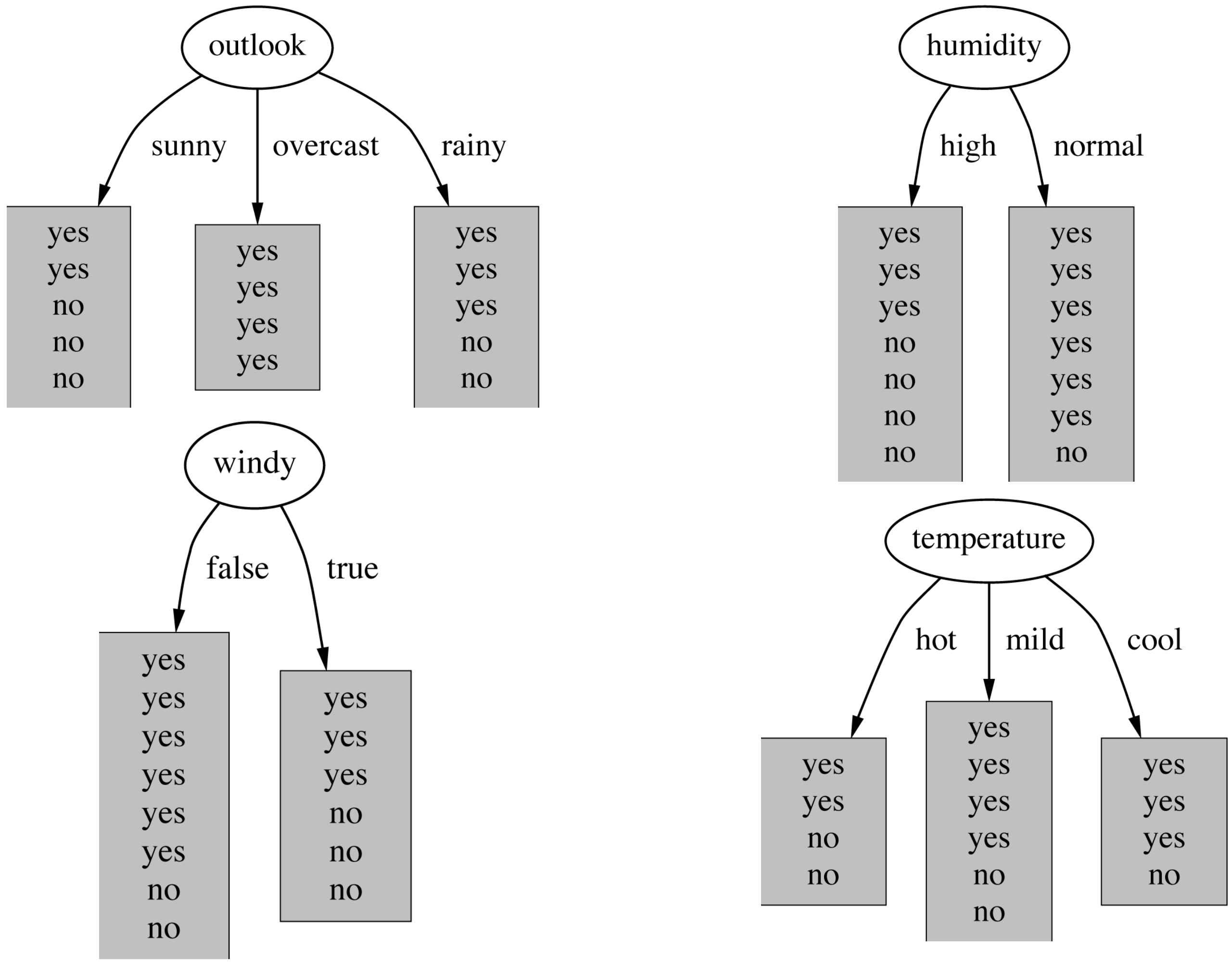
Building Decision Tree [Q93]

- Top-down tree construction
  - At start, all training examples are at the root.
  - Partition the examples recursively by choosing one attribute each time.
- Bottom-up tree pruning
  - Remove subtrees or branches, in a bottom-up manner, to improve the estimated accuracy on new cases.

Choosing the Best Attribute

- The key problem is 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
  - **Most-Values:** Choose the attribute with the largest number of possible values
  - **Information gain:** Choose the attribute that has the largest expected information gain, i.e. select attribute that will result in the smallest expected size of the subtrees rooted at its children.

Which attribute to select?



A criterion for attribute selection

- Which is the best attribute?
  - The one which will result in the smallest tree
  - Heuristic: choose the attribute that produces the "purest" nodes
- Popular *impurity criterion*: *information gain*
  - Information gain increases with the average purity of the subsets that an attribute produces
- Strategy: choose attribute that results in greatest information gain

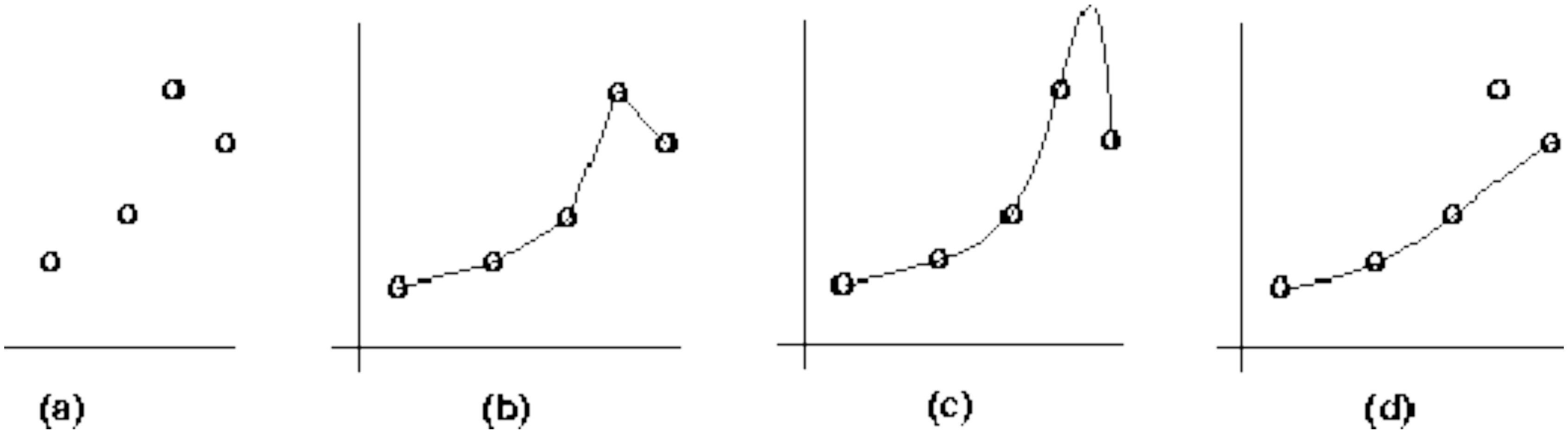
Choosing the Splitting Attribute

- At each node, available attributes are evaluated on the basis of separating the classes of the training examples. A goodness function is used for this purpose.
- Typical goodness functions used for DTrees:
  - information gain (ID3/C4.5)
  - information gain ratio
  - gini index (CART)

Preference Bias: Ockham's Razor

- Aka Occam's Razor, Law of Economy, or Law of Parsimony
- Principle stated by William of Ockham (1285-1347/49), an English philosopher, that
  - "*non sunt multiplicanda entia praeter necessitatem*"
  - or, entities are not to be multiplied beyond necessity.
- The simplest explanation that is consistent with all observations is the best.
- Therefore, the smallest decision tree that correctly classifies all of the training examples is the best.
- Finding the provably smallest decision tree is intractable (NP-hard), so instead of constructing the absolute smallest tree consistent with the training examples, construct one that is pretty small.

Inductive Learning and Bias



- Suppose that we want to learn a function f(x) = y and we are given some sample (x,y) pairs, as points in figure (a).
- There are several hypotheses we could make about this function, e.g.: (b), (c) and (d).
- A preference for one over the others reveals the **bias** of our learning technique, e.g.:
  - prefer piece-wise functions
  - prefer a smooth function
  - prefer a simple function and treat outliers as noise

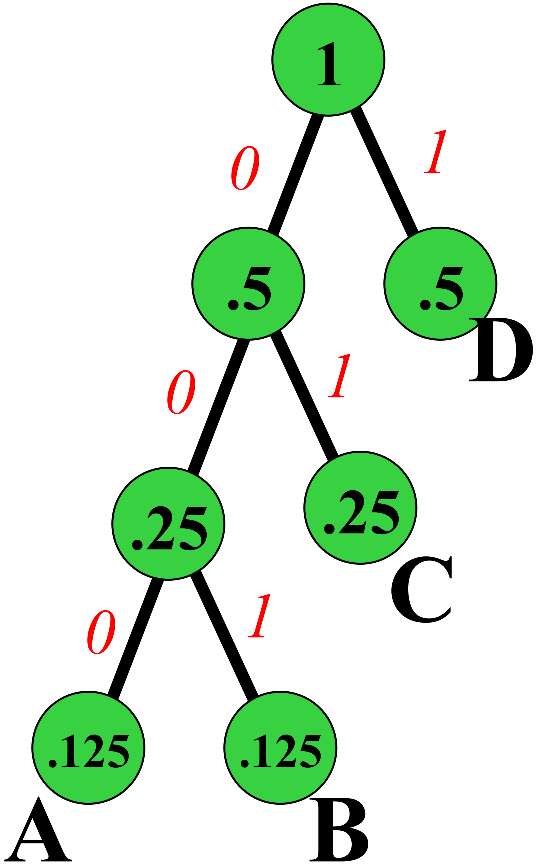
Information Theory

- Assume you can bet 1$ for a coin flip (10000 bets), if your bet is right, you get back 2$ otherwise you get nothing
- You know that the coin used is rigged and comes up heads with probability 0.99, so you bet heads - obviously (but find somebody arranging this bet :)
- The expected value for the bet is 1.98$
- How much will you be willing to pay for the advance information about the actual outcome of the flip? What the value of the advance information?
- Less than 0.02$!
- If the coin were fair, your expected value would 1$ and you would be willing to pay up to 1$
- The less you know, the more valuable the information
- Information theory does not measure the value of information in $ but the information content of a message in bits.

Example: Huffman code

- In 1952 MIT student David Huffman devised, in the course of doing a homework assignment, an elegant coding scheme which is optimal in the case where all symbols' probabilities are integral powers of 1/2.
- A Huffman code can be built in the following manner:
  - Rank all symbols in order of probability of occurrence.
  - Successively combine the two symbols of the lowest probability to form a new composite symbol; eventually we will build a binary tree where each node is the probability of all nodes beneath it.
  - Trace a path to each leaf, noticing the direction at each node.

Huffman code example



M	code	length	prob	
A	000	3	0,125	0,375
B	001	3	0,125	0,375
C	01	2	0,250	0,500
D	1	1	0,500	0,500
average message length				1,750

If we need to send many messages (A,B,C or D) and they have this probability distribution and we use this code, then over time, the average bits/message should approach 1.75

Information Theory Background

- If there are n equally probable possible messages, then the probability p of each is 1/n
- Information conveyed by a message is -log(p) = log(n)
- Eg, if there are 16 messages, then log(16) = 4 and we need 4 bits to identify/send each message.
- In general, if we are given a probability distribution

  $P = (p1, p2, .., pn)$

- the information conveyed by distribution (aka entropy of P) is:

  $I(P) = -(p1*\log(p1) + p2*\log(p2) + .. + pn*\log(pn))$

  $= - \sum_i pi*\log(pi)$

Information Theory Background

- Information conveyed by distribution (aka Entropy of P) is:

  I(P) = -(p1*log(p1) + p2*log(p2) + .. + pn*log(pn))

- Examples:
  - if P is (0.5, 0.5) then I(P) is 1
  - if P is (0.67, 0.33) then I(P) is 0.92,
  - if P is (1, 0) or (0,1) then I(P) is 0.
- The more uniform is the probability distribution, the greater is its information.
- The entropy is the average number of bits/message needed to represent a stream of messages.

Example: attribute “Outlook”, 1

Outlook	Temperature	Humidity	Windy	Play?
sunny	hot	high	false	No
sunny	hot	high	true	No
overcast	hot	high	false	Yes
rain	mild	high	false	Yes
rain	cool	normal	false	Yes
rain	cool	normal	true	No
overcast	cool	normal	true	Yes
sunny	mild	high	false	No
sunny	cool	normal	false	Yes
rain	mild	normal	false	Yes
sunny	mild	normal	true	Yes
overcast	mild	high	true	Yes
overcast	hot	normal	false	Yes
rain	mild	high	true	No

Example: attribute “Outlook”, 2

- “Outlook” = “Sunny”:

$$\text{info}([2,3]) = \text{entropy}(2/5,3/5) = -2/5\log(2/5) - 3/5\log(3/5) = 0.971 \text{ bits}$$

- “Outlook” = “Overcast”:

$$\text{info}([4,0]) = \text{entropy}(1,0) = -1\log(1) - 0\log(0) = 0 \text{ bits}$$

Note: log(0) is not defined, but we evaluate 0*log(0) as zero

- “Outlook” = “Rainy”:

$$\text{info}([3,2]) = \text{entropy}(3/5,2/5) = -3/5\log(3/5) - 2/5\log(2/5) = 0.971 \text{ bits}$$

- Expected information for attribute:

$$\text{info}([3,2],[4,0],[3,2]) = (5/14)\times 0.971 + (4/14)\times 0 + (5/14)\times 0.971$$
$$= 0.693 \text{ bits}$$

Computing the information gain

- Information gain:

(information before split) – (information after split)

$$\text{gain}(\text{"Outlook"}) = \text{info}([9,5]) - \text{info}([2,3],[4,0],[3,2]) = 0.940 - 0.693$$
$$= 0.247 \text{ bits}$$

Computing the information gain

- Information gain:

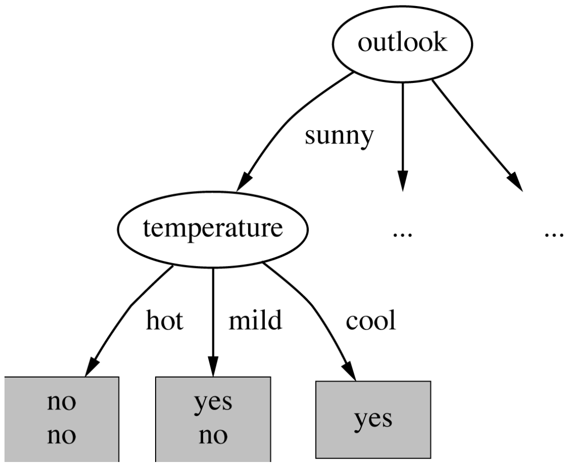
(information before split) – (information after split)

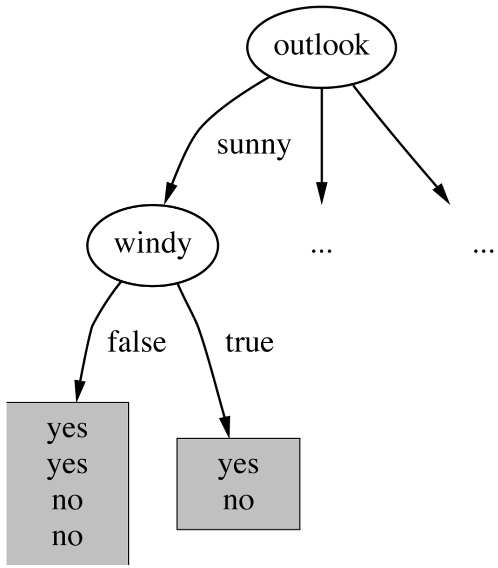
$$\text{gain}(\text{"Outlook"}) = \text{info}([9,5]) - \text{info}([2,3],[4,0],[3,2]) = 0.940 - 0.693$$
$$= 0.247 \text{ bits}$$

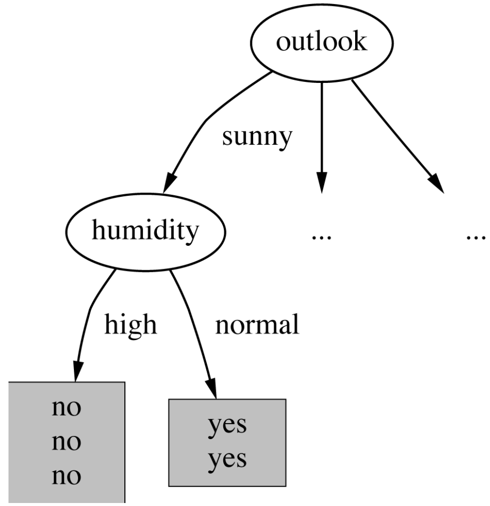
- Information gain for attributes from weather data:

$$\text{gain}(\text{"Outlook"}) = 0.247 \text{ bits}$$
$$\text{gain}(\text{"Temperature"}) = 0.029 \text{ bits}$$
$$\text{gain}(\text{"Humidity"}) = 0.152 \text{ bits}$$
$$\text{gain}(\text{"Windy"}) = 0.048 \text{ bits}$$

Continuing to split





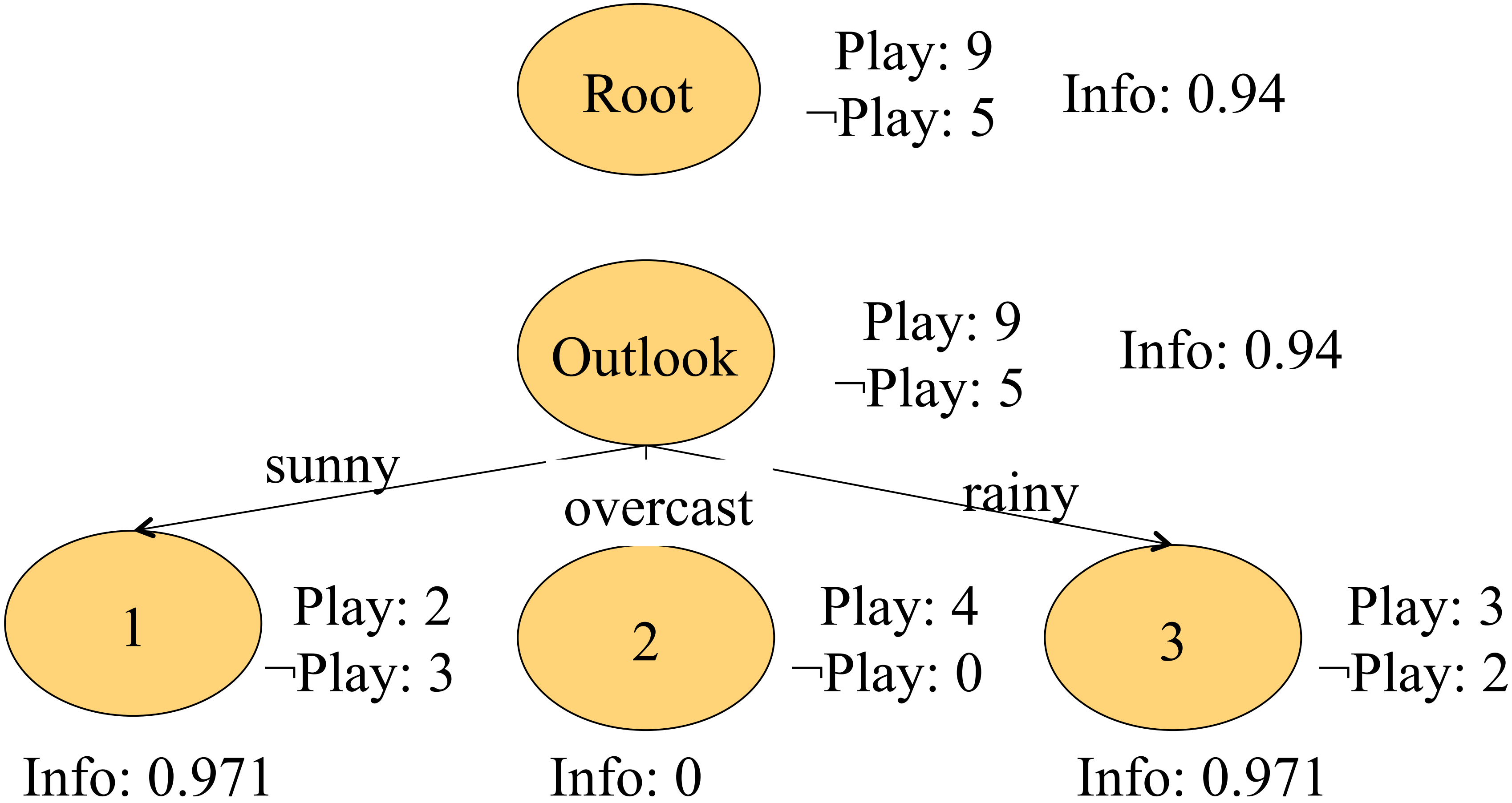


$$\text{gain}(\text{"Humidity"}) = 0.971 \text{ bits}$$

$$\text{gain}(\text{"Temperature"}) = 0.571 \text{ bits}$$

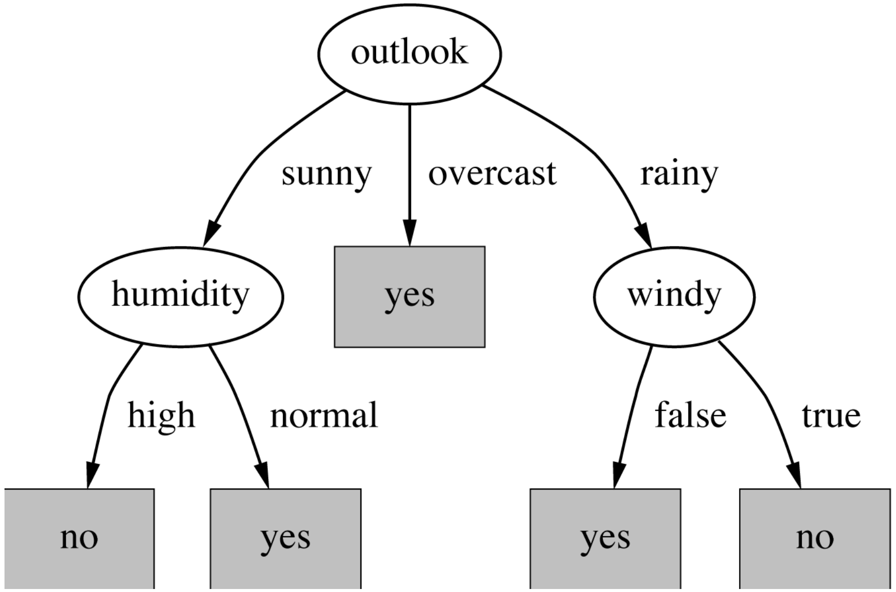
$$\text{gain}(\text{"Windy"}) = 0.020 \text{ bits}$$

Schematicly



$$Gain = 0.94 - (5/14)\times 0.971 + (4/14)\times 0 + (5/14)\times 0.971 = 0.693$$

The final decision tree



- Note: not all leaves need to be pure; sometimes identical instances have different classes

  $\Rightarrow$ Splitting stops when data can't be split any further

*Wish list for a purity measure

- Properties we require from a purity measure:
  - When node is pure, measure should be zero
  - When impurity is maximal (i.e. all classes equally likely), measure should be maximal
  - Measure should obey *multistage property* (i.e. decisions can be made in several stages):

$$\text{measure}([2,3,4]) = \text{measure}([2,7]) + (7/9) \times \text{measure}([3,4])$$

- Entropy is a function that satisfies all three properties!

*Properties of the entropy

- The multistage property:

$$\text{entropy}(p,q,r) = \text{entropy}(p,q+r) + (q+r) \times \text{entropy}(\frac{q}{q+r}, \frac{r}{q+r})$$

- Simplification of computation:

$$\text{info}([2,3,4]) = -2/9 \times \log(2/9) - 3/9 \times \log(3/9) - 4/9 \times \log(4/9)$$
$$= [-2\log 2 - 3\log 3 - 4\log 4 + 9\log 9]/9$$

- Note: instead of maximizing info gain we could just minimize information

The ID3 algorithm is used to build a decision tree, given a set of non-categorical attributes C1, C2, .., Cn, the class attribute C, and a training set T of records.

```
function ID3(R: a set of non-categorical attributes,/input attribute e.g. outlook
             C: the categorical attribute,  // output attribute e.g. play
             S: a training set) returns a decision tree;
   begin
     If S is empty, return a single node with value Failure;
     If every example in S has the same value for C, return
       single node with that value;
     If R is empty, then return a single node with most
       frequent of the values of C found in examples S;
       [note: there will be errors, i.e., improperly classified
       records];
     Let D be attribute with largest Gain(D,S) among attributes in R;
     Let {dj| j=1,2, .., m} be the values of attribute D;
     Let {Sj| j=1,2, .., m} be the subsets of S consisting
        respectively of records with value dj for attribute D;
     Return a tree with root labeled D and arcs labeled
        d1, d2, .., dm going respectively to the trees
        ID3(R-{D},C,S1), ID3(R-{D},C,S2) ,.., ID3(R-{D},C,Sm);
   end ID3;
```

How well does it work?

Many case studies have shown that decision trees are at least as accurate as human experts.

- A study for diagnosing breast cancer had humans correctly classifying the examples 65% of the time, and the decision tree classified 72% correct.
- British Petroleum designed a decision tree for gas-oil separation for offshore oil platforms that replaced an earlier rule-based expert system.
- Cessna designed an airplane flight controller using 90,000 examples and 20 attributes per example.

Highly-branching attributes

- Problematic: attributes with a large number of values (extreme case: ID code)
- Subsets are more likely to be pure if there is a large number of values

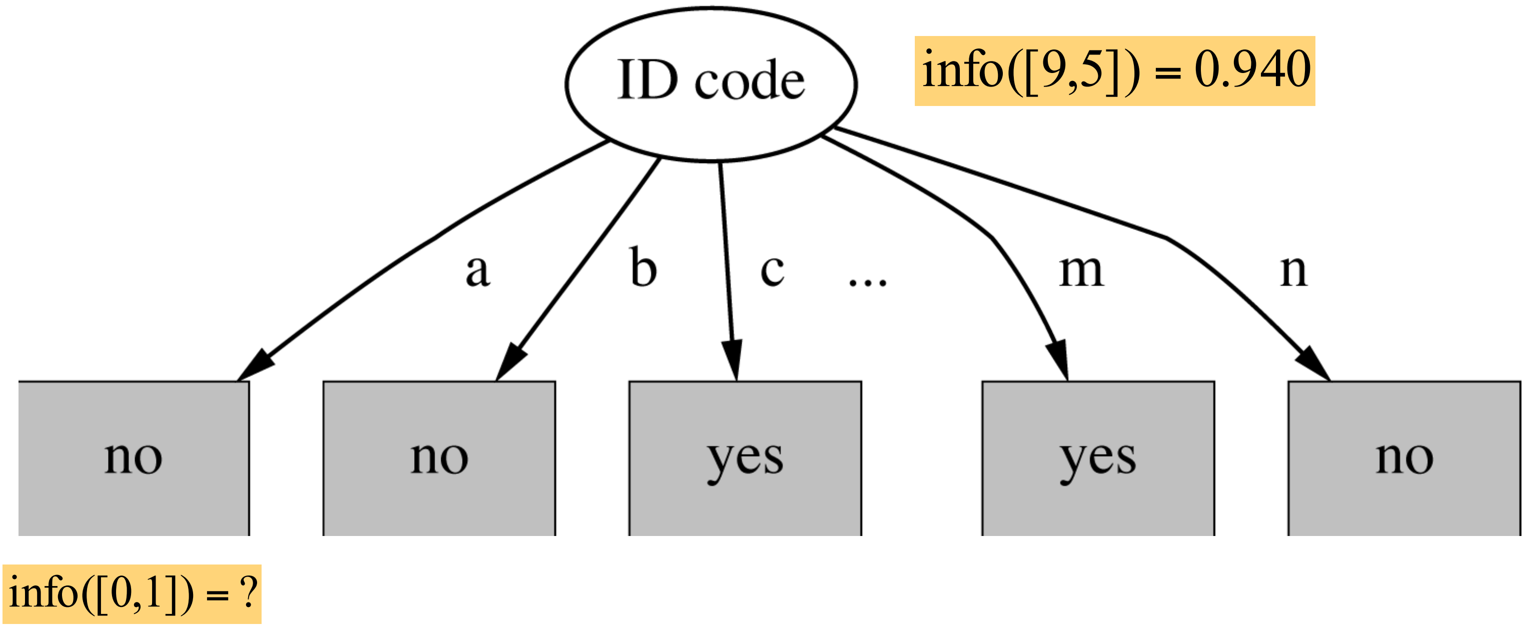
  ⇒Information gain is biased towards choosing attributes with a large number of values

  ⇒This may result in *overfitting* (selection of an attribute that is non-optimal for prediction)

Weather Data with ID code

ID	Outlook	Temperature	Humidity	Windy	Play?
A	sunny	hot	high	false	No
B	sunny	hot	high	true	No
C	overcast	hot	high	false	Yes
D	rain	mild	high	false	Yes
E	rain	cool	normal	false	Yes
F	rain	cool	normal	true	No
G	overcast	cool	normal	true	Yes
H	sunny	mild	high	false	No
I	sunny	cool	normal	false	Yes
J	rain	mild	normal	false	Yes
K	sunny	mild	normal	true	Yes
L	overcast	mild	high	true	Yes
M	overcast	hot	normal	false	Yes
N	rain	mild	high	true	No

Split for ID Code Attribute

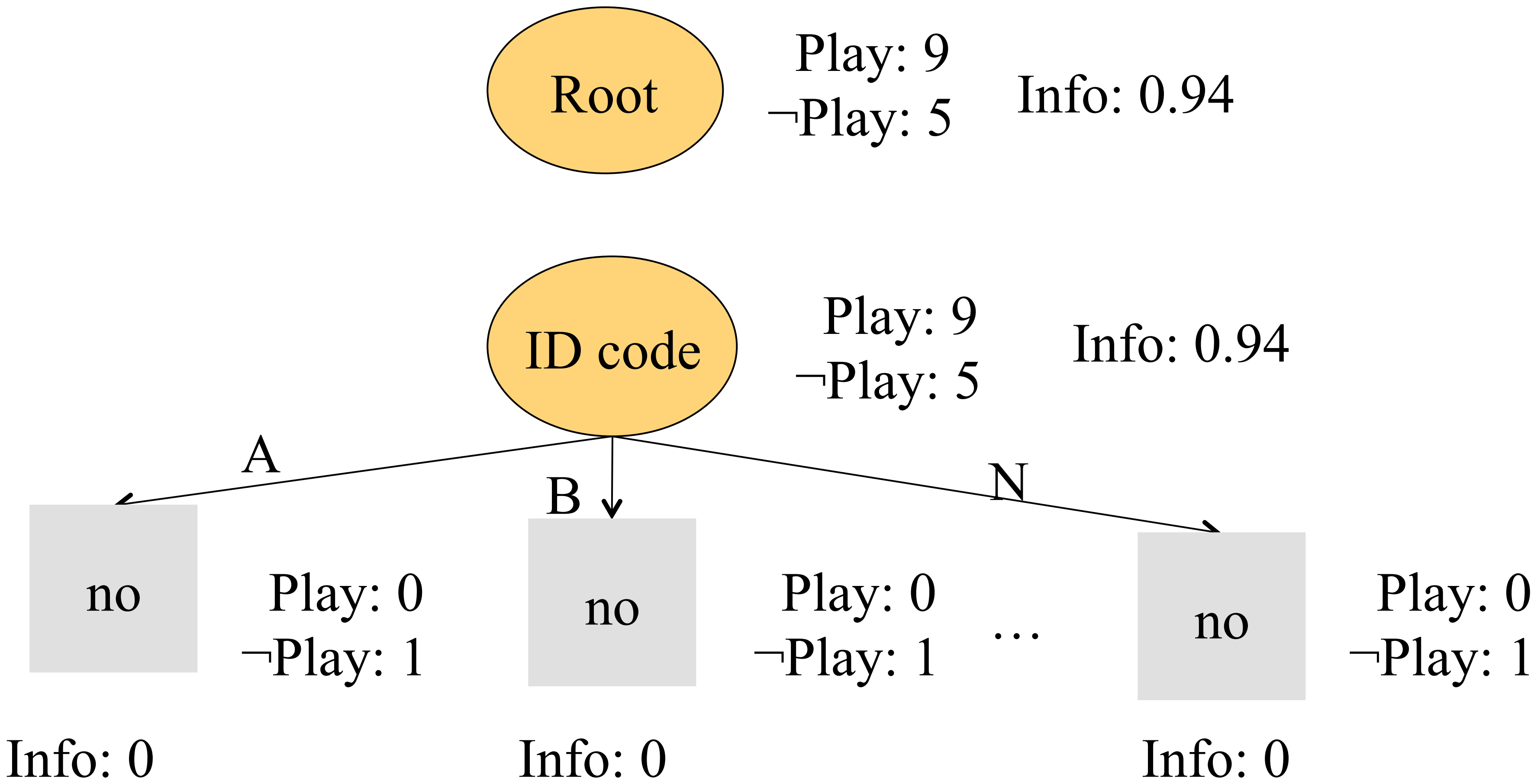


Entropy of split = 0 (since each leaf node is "pure", having only one case.

Information gain is maximal for ID code.

Customers are not different because of different credit car number.

Schematicly



$$Gain = 0.94 - (1/14)\times 0 + \ldots + (1/14)\times 0 = 0.94$$

Gain ratio

- *Gain ratio*: a modification of the information gain that reduces its bias on high-branch attributes
- Gain ratio should be
  - Large when data is evenly spread
  - Small when all data belong to one branch
- Gain ratio takes number and size of branches into account when choosing an attribute
  - It corrects the information gain by taking the *intrinsic information* of a split into account (i.e., how much info do we need to tell which branch an instance belongs to)

Gain Ratio and Intrinsic Info./Split Info

- Intrinsic information: entropy of distribution of instances into branches

$$IntrinsicInfo(S,A) \equiv Info(|S_1|/|S|, |S_2|/|S|, \ldots, |S_n|/|S|$$

$$= -\sum \frac{|S_i|}{|S|} \log_2 \frac{|S_i|}{|S|}.$$

For Information Gain we summed over the info of each resulting node not the info of the split

- *Gain ratio* (Quinlan'86) normalizes info gain by:

$$GainRatio(S,A) = \frac{Gain(S,A)}{IntrinsicInfo(S,A)}.$$

Computing the gain ratio

- Example: intrinsic information for ID code

$$\text{info}([1,1,\ldots,1]) = 14 \times (-1/14 \times \log 1/14) = 3.807 \text{ bits}$$

- **Importance of attribute decreases as intrinsic information gets larger**
- Example of gain ratio:

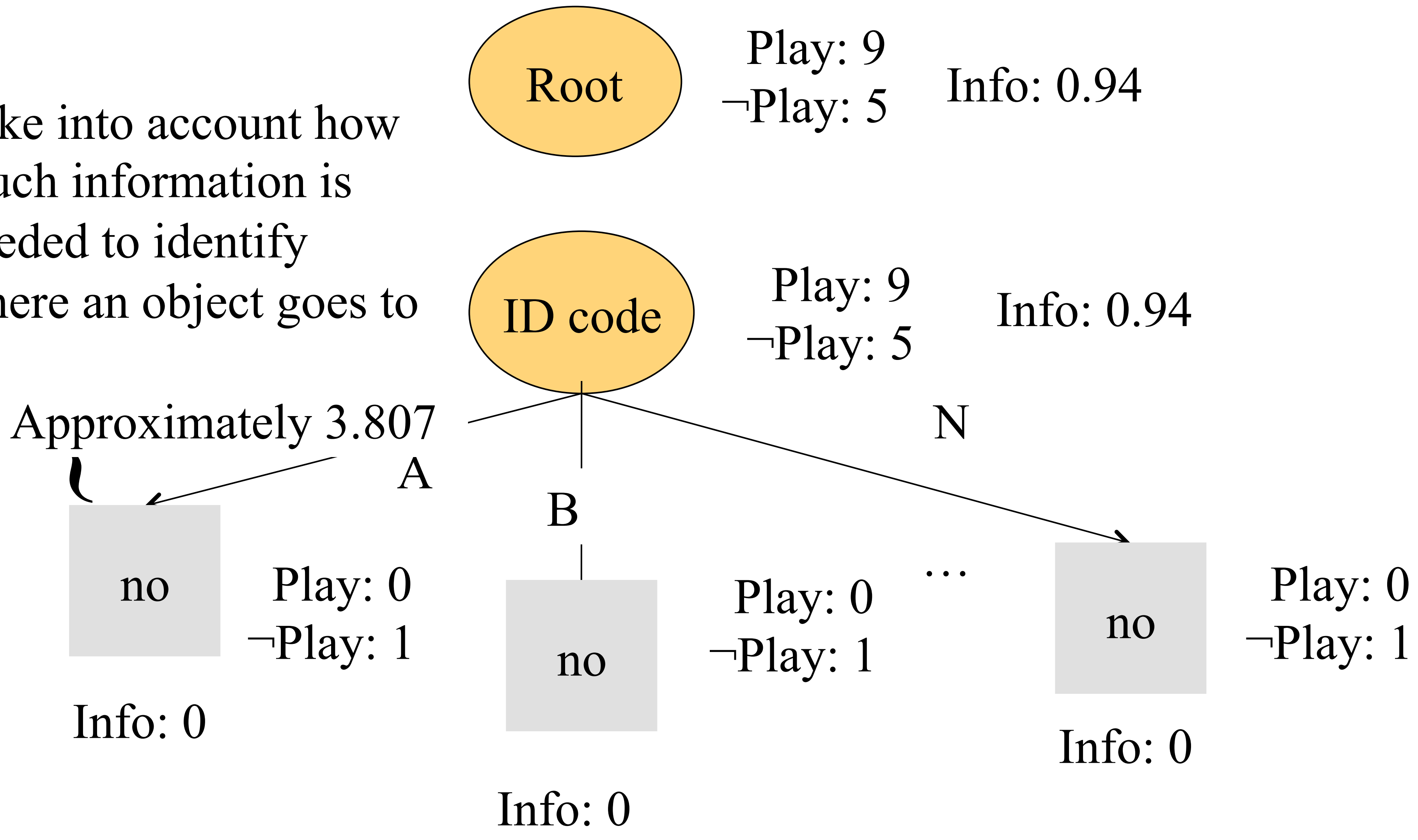
$$\text{gain\_ratio}(\text{"Attribute"}) = \frac{\text{gain}(\text{"Attribute"})}{\text{intrinsic\_info}(\text{"Attribute"})}$$

- Example:

$$\text{gain\_ratio}(\text{"ID\_code"}) = \frac{0.940 \text{ bits}}{3.807 \text{ bits}} = 0.246$$

Schematicly

Take into account how much information is needed to identify where an object goes to



Gain ratios for weather data

Outlook		**Temperature**	
Info:	0.693	Info:	0.911
Gain: 0.940-0.693	0.247	Gain: 0.940-0.911	0.029
Split info: info([5,4,5])	1.577	Split info: info([4,6,4])	1.362
Gain ratio: 0.247/1.577	0.156	Gain ratio: 0.029/1.362	0.021

Humidity		**Windy**	
Info:	0.788	Info:	0.892
Gain: 0.940-0.788	0.152	Gain: 0.940-0.892	0.048
Split info: info([7,7])	1.000	Split info: info([8,6])	0.985
Gain ratio: 0.152/1	0.152	Gain ratio: 0.048/0.985	0.049

$$\text{gain\_ratio("ID\_code")} = \frac{0.940 \text{ bits}}{3.807 \text{ bits}} = 0.246$$

More on the gain ratio

- "Outlook" still comes out top
- However: "ID code" has high gain ratio
  - Standard fix: *ad hoc* test to prevent splitting on that type of attribute
- Problem with gain ratio: it may overcompensate
  - May choose an attribute just because its intrinsic information is very low. Note how close humidity and outlook became. Maybe that's not such a good thing?
  - Standard fix:
    - First, only consider attributes with greater than average information gain
    - Then, compare them on gain ratio

Gini Index: Measure of Diversity

- Suppose we randomly draw an object from node t and give it the class i. What is the error for picking an object of class i if the intended object is of class j?

  p(i|t) p(j|t)

- The more homogeneous a set the smaller the error

- We get the expected error (for all classes j) if we compute this for all possible misclassifications:

$$\sum_{i\neq j} p(i\,|\,t)p(j\,|\,t)$$

For 3 classes:

(p1p2+p1p3+p1p1)+(p2p1+p2p3+p2p2)+(p3p1+p3p2+p3p3) =

$p1^2+p2^2+p3^2$+ 2p1p2+2p1p3+2p2p3 = $(p1+p2+p3)^2$

$$\sum_{i\neq j} p(i\,|\,t)p(j\,|\,t) = \underbrace{\left(\sum_i p_i\right)^2}_{=1} - \sum_i p_i^2 = 1 - \sum_i p_i^2$$

Gini Index: Used in the CART Learner
CART = Classification and Regression Tree

After splitting T into two subsets T1 and T2 with sizes N1 and N2, the gini index of the split data is defined as

$$gini_{split}(T) = \frac{N_1}{N} gini(T_1) + \frac{N_2}{N} gini(T_2)$$

- The attribute providing smallest $gini_{split}(T)$ is chosen to split the node.

Properties of the goodness function:
F(0.5,0.5) = max
F(0,1) = F(1,0) = 0
Increasing for [0;0.5] decreasing for [0.5;1]

Gini, Entropy, Error Examples for a Single Node

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

Node N_1	Count
Class=0	0
Class=1	6

$\text{Gini} = 1 - (0/6)^2 - (6/6)^2 = 0$
$\text{Entropy} = -(0/6)\log_2(0/6) - (6/6)\log_2(6/6) = 0$
$\text{Error} = 1 - \max[0/6, 6/6] = 0$

Node N_2	Count
Class=0	1
Class=1	5

$\text{Gini} = 1 - (1/6)^2 - (5/6)^2 = 0.278$
$\text{Entropy} = -(1/6)\log_2(1/6) - (5/6)\log_2(5/6) = 0.650$
$\text{Error} = 1 - \max[1/6, 5/6] = 0.167$

Node N_3	Count
Class=0	3
Class=1	3

$\text{Gini} = 1 - (3/6)^2 - (3/6)^2 = 0.5$
$\text{Entropy} = -(3/6)\log_2(3/6) - (3/6)\log_2(3/6) = 1$
$\text{Error} = 1 - \max[3/6, 3/6] = 0.5$

Goodness functions

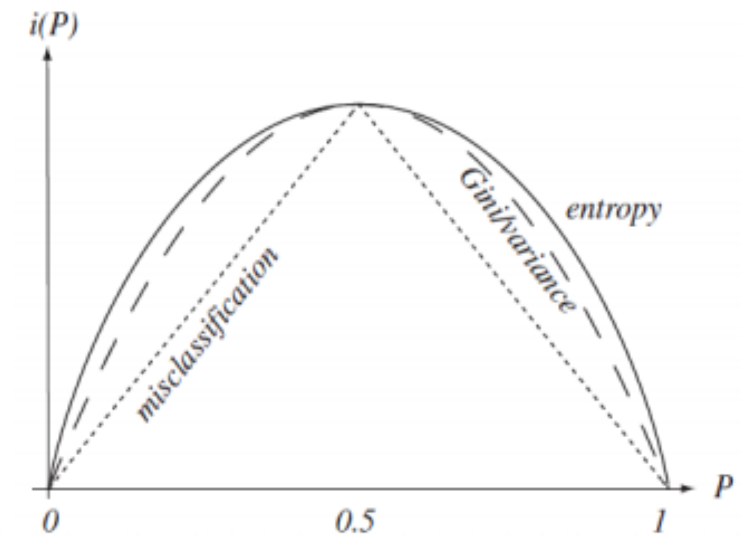
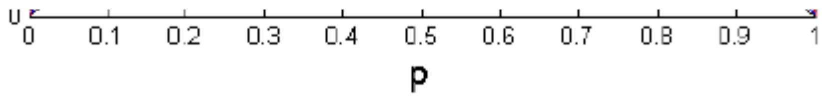
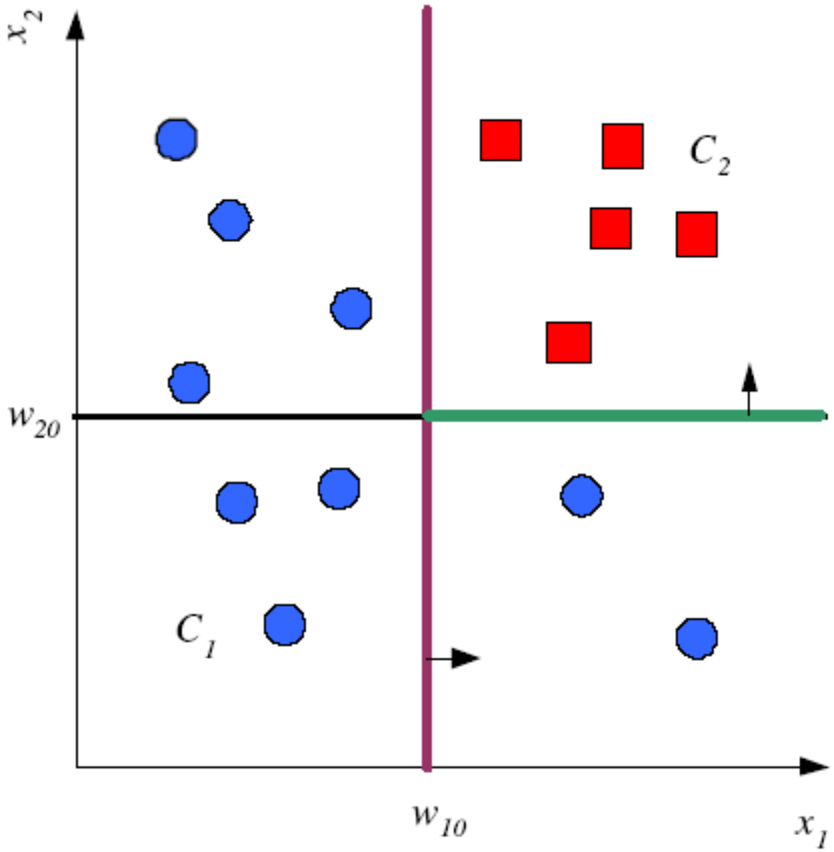
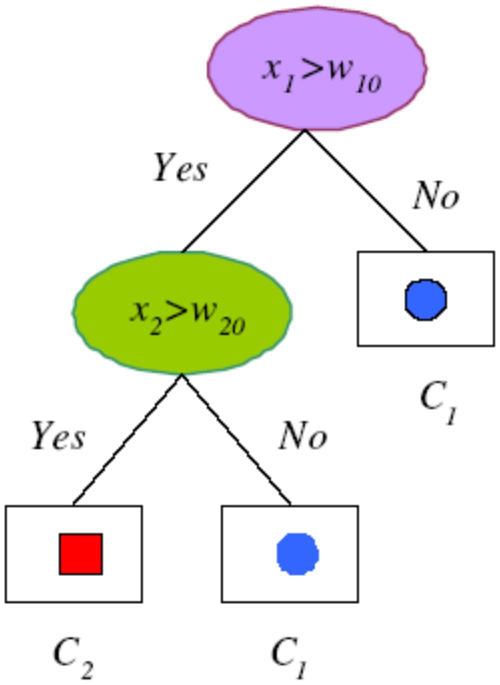


Figure 8.4: For the two-category case, the impurity functions peak at equal class frequencies and the variance and the Gini impurity functions are identical. To facilitate comparisons, the entropy, variance, Gini and misclassification impurities (given by Eqs. 1 – 4, respectively) have been adjusted in scale and offset to facilitate comparison; such scale and offset does not directly affect learning or classification.

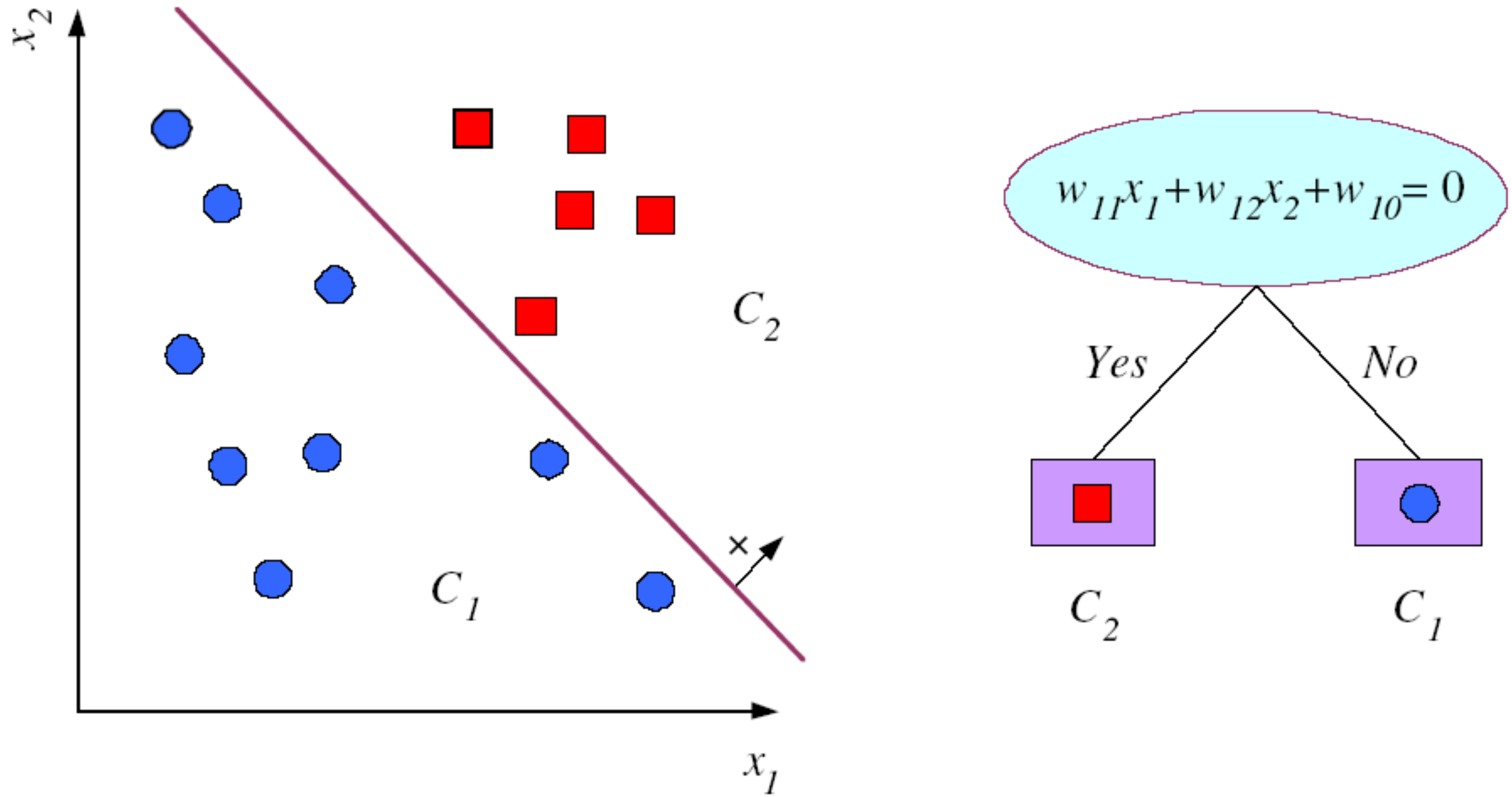


Univariate Splits





Multivariate Splits



Noisy data and Overfitting

- Many kinds of "noise" that could occur in the examples:
  - Two examples have same attribute/value pairs, but different classifications
  - Some values of attributes are incorrect because of errors in the data acquisition process or the preprocessing phase
  - The classification is wrong (e.g., + instead of -) because of some error
  - Some attributes are irrelevant to the decision-making process, e.g., color of a die is irrelevant to its outcome.

Noisy data and Overfitting

- The last problem, irrelevant attributes, can result in overfitting
  - if hypothesis space has many dimensions because of a large number of attributes, we may find **meaningless regularity** in the data that is irrelevant to the true, important, distinguishing features.
  - Fix by pruning lower nodes in the decision tree (see C4.5 next section)
  - For example, if Gain of the best attribute at a node is below a threshold, stop and make this node a leaf rather than generating children nodes.

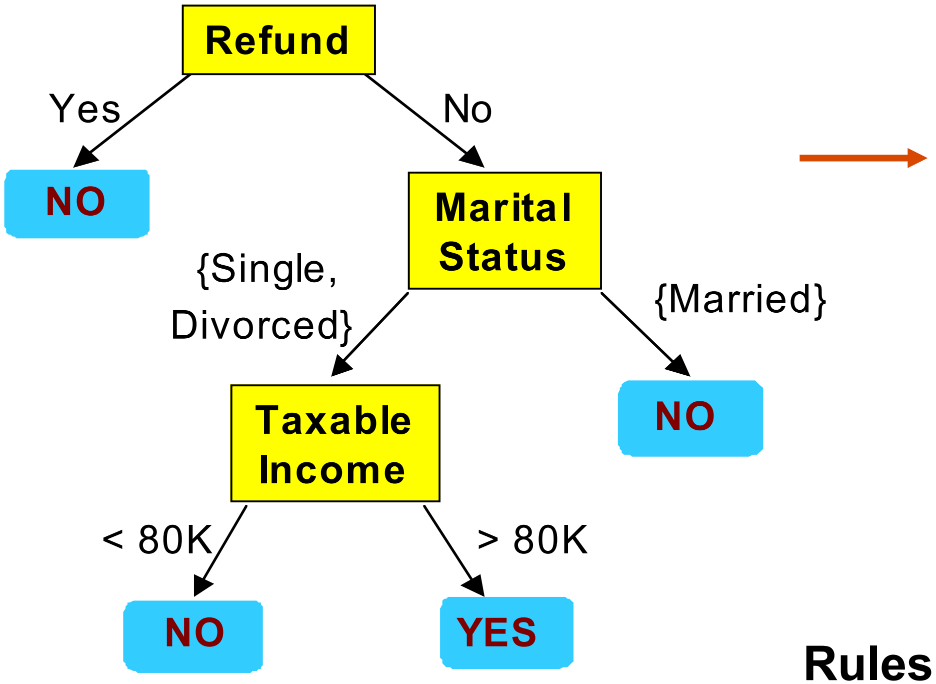
Summary introduction

- Top-Down Decision Tree Construction
- Choosing the Splitting Attribute
  - Information Gain biased towards attributes with a large number of values
  - Gain Ratio takes number and size of branches into account when choosing an attribute
  - Gini Index measure the misclassification ratio
- Many other impurity measures are available but no general better solution

Decision Trees to Rules

- It is easy to derive a rule set from a decision tree: write a rule for each path in the decision tree from the root to a leaf.
- In that rule the left-hand side is easily built from the label of the nodes and the labels of the arcs.
- The resulting rules set can be simplified:
  - Let LHS be the left hand side of a rule.
  - Let LHS' be obtained from LHS by eliminating some conditions.
  - We can certainly replace LHS by LHS' in this rule if the subsets of the training set that satisfy respectively LHS and LHS' are equal.
  - A rule may be eliminated by using meta-conditions such as "if no other rule applies".

From Decision Trees To Rules



Classification Rules

(Refund=Yes) ==> No

(Refund=No, Marital Status={Single,Divorced}, Taxable Income<80K) ==> No

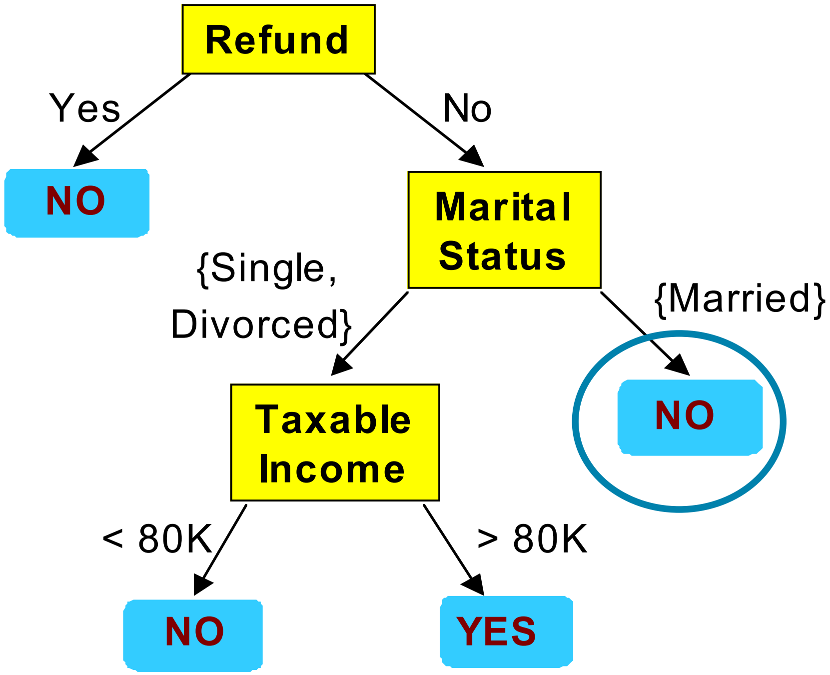
(Refund=No, Marital Status={Single,Divorced}, Taxable Income>80K) ==> Yes

(Refund=No, Marital Status={Married}) ==> No

Rules are mutually exclusive and exhaustive

Rule set contains as much information as the tree

Rules Can Be Simplified



Tid	Refund	Marital Status	Taxable Income	Cheat
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	No	Single	90K	Yes

Initial Rule: (Refund=No) $\wedge$ (Status=Married) $\rightarrow$ No

Simplified Rule: (Status=Married) $\rightarrow$ No