Decision tree learning

Sunita Sarawagi
IIT Bombay

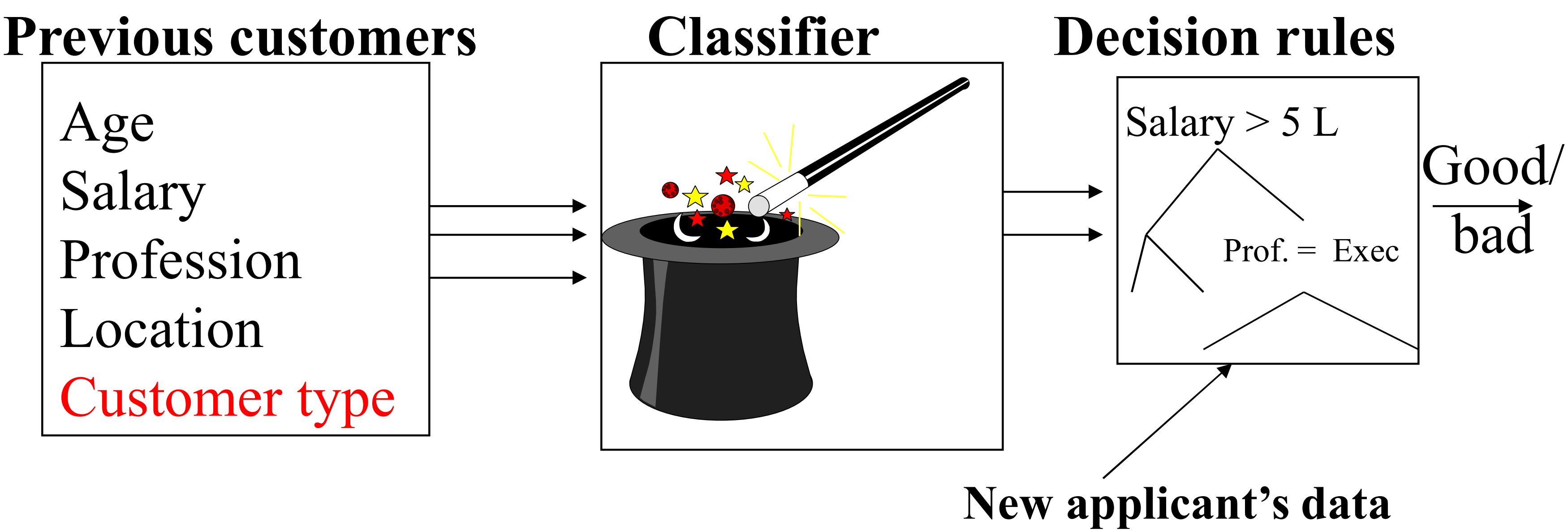
http://www.it.iitb.ac.in/~sunita

Decision tree classifiers

- Widely used learning method
- Easy to interpret: can be re-represented as if-then-else rules
- Approximates function by piece wise constant regions
- Does not require any prior knowledge of data distribution, works well on noisy data.
- Has been applied to:
  - classify medical patients based on the disease,
  - equipment malfunction by cause,
  - loan applicant by likelihood of payment.
  - lots and lots of other applications..

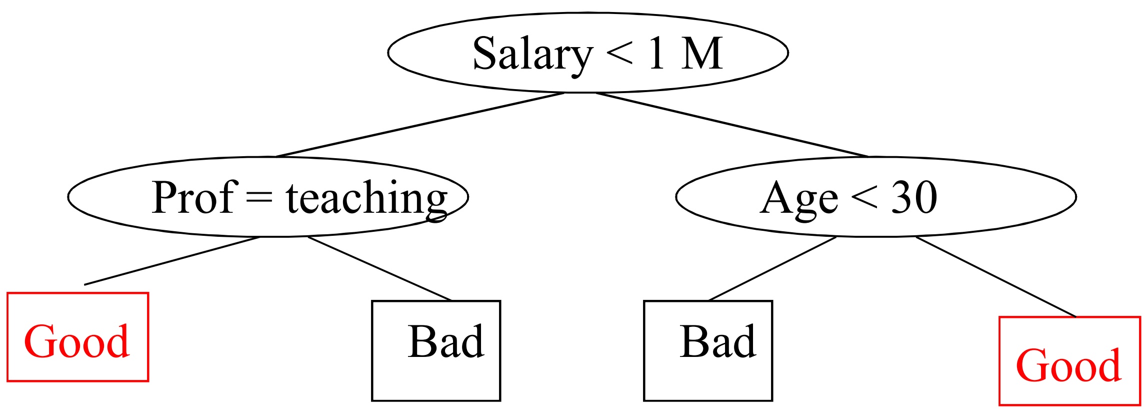
Setting

- Given old data about customers and payments, predict new applicant's loan eligibility.



Decision trees

- Tree where internal nodes are simple decision rules on one or more attributes and leaf nodes are predicted class labels.

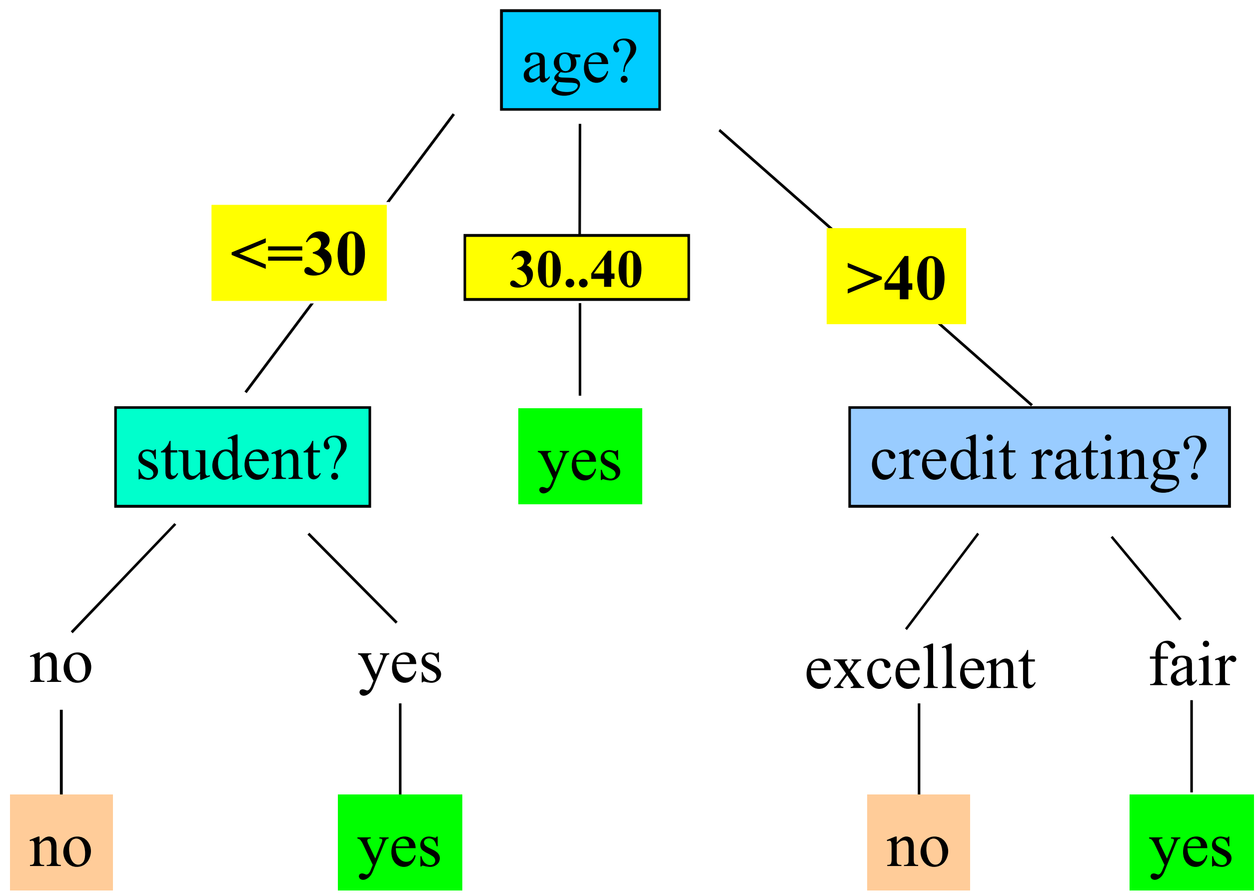


Training Dataset

This follows an example from Quinlan's ID3

age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
30…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

Output: A Decision Tree for "*buys_computer*"

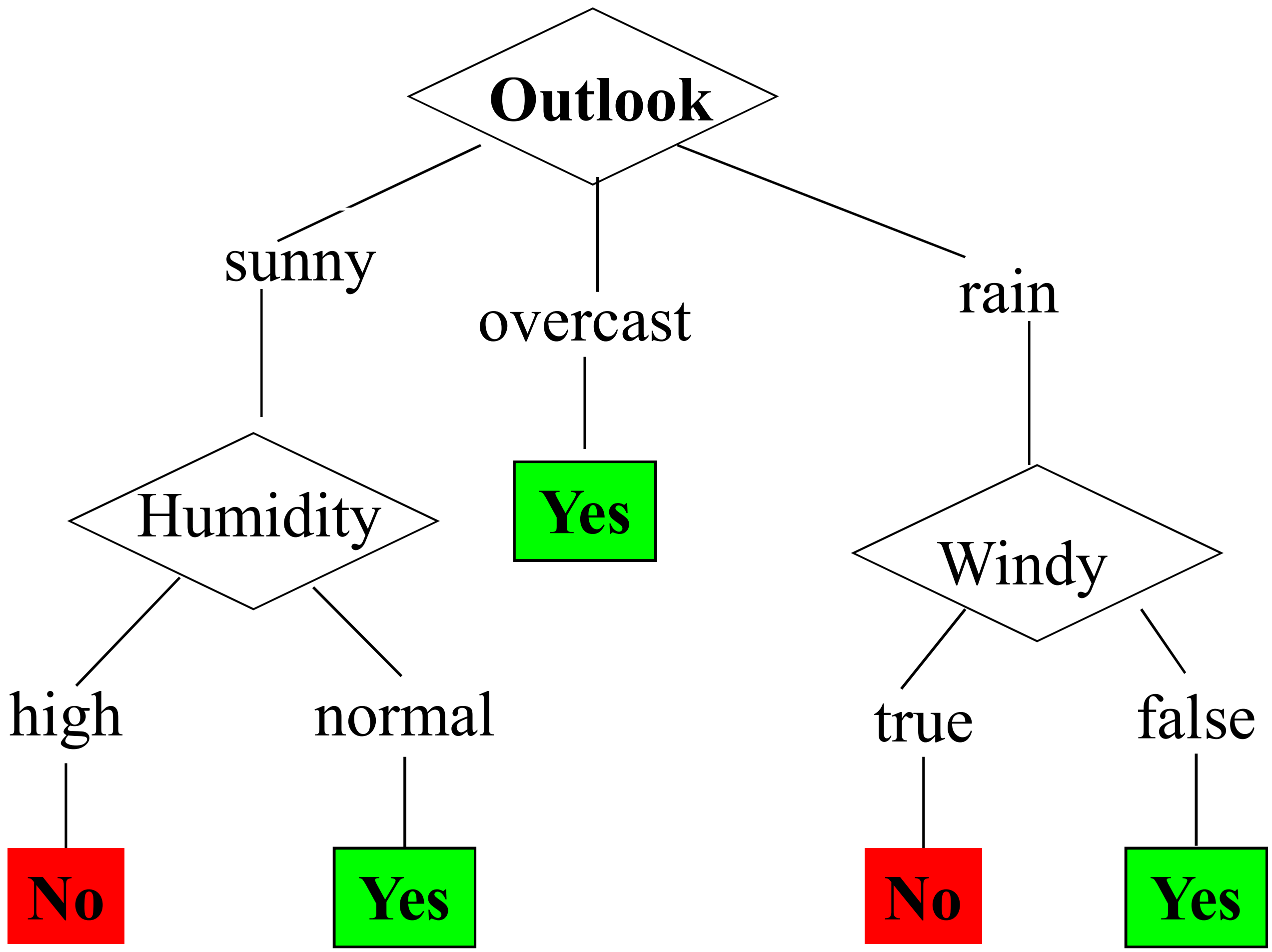


Weather Data: Play or not Play?

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

Note:
Outlook is the Forecast,
no relation to Microsoft email program

Example Tree for "Play?"



Topics to be covered

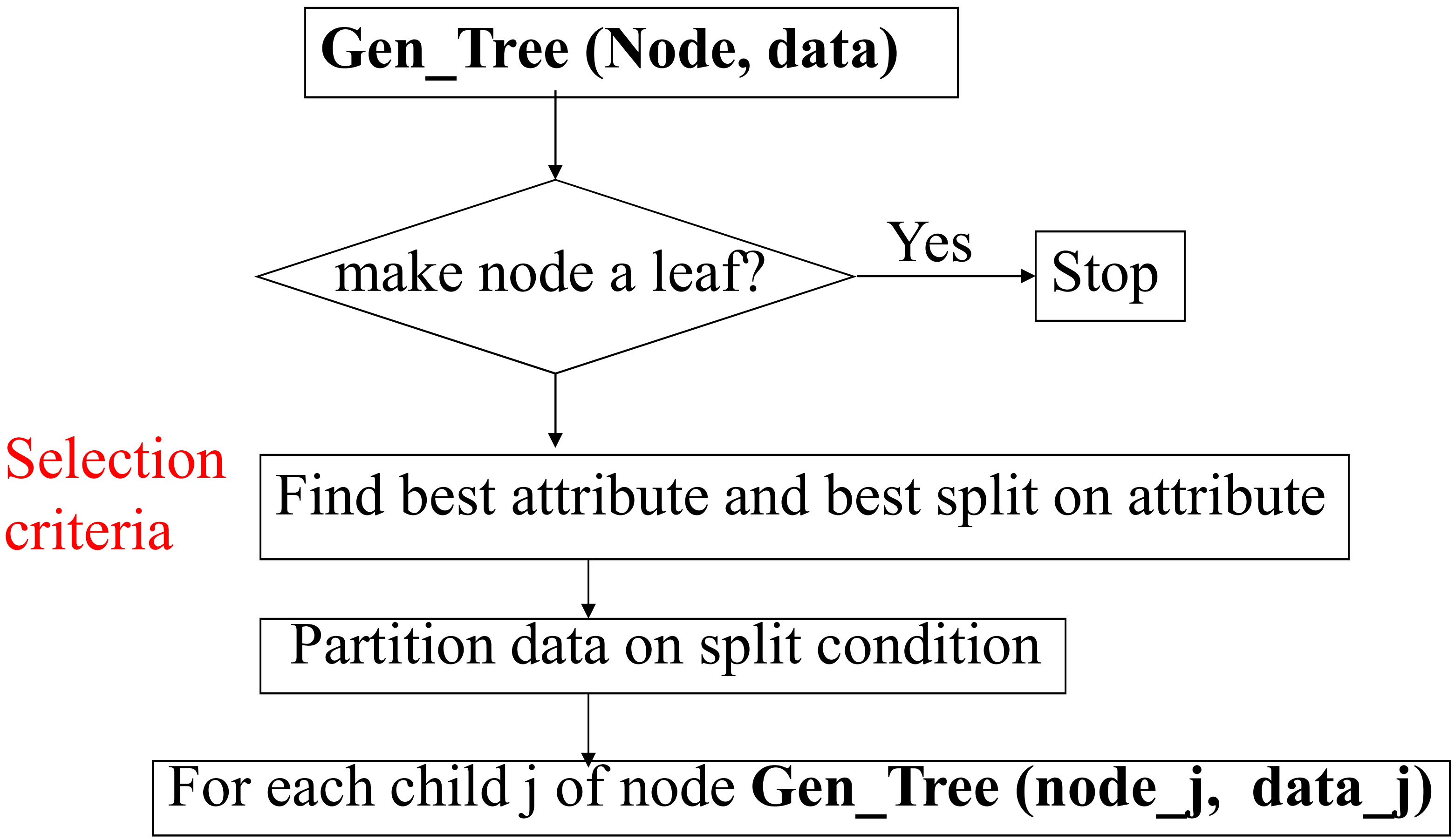
- Tree construction:
  - Basic tree learning algorithm
  - Measures of predictive ability
  - High performance decision tree construction: Sprint
- Tree pruning:
  - Why prune
  - Methods of pruning
- Other issues:
  - Handling missing data
  - Continuous class labels
  - Effect of training size

Tree learning algorithms

- ID3 (Quinlan 1986)
- Successor C4.5 (Quinlan 1993)
- CART
- SLIQ (Mehta et al)
- SPRINT (Shafer et al)

Basic algorithm for tree building

- Greedy top-down construction.



Split criteria

- Select the attribute that is best for classification.
- Intuitively pick one that best separates instances of different classes.
- Quantifying the intuitive: measuring separability:
- First define *impurity* of an arbitrary set S consisting of K classes
- Smallest when consisting of only one class, highest when all classes in equal number.
- Should allow computations in multiple stages.

Measures of impurity

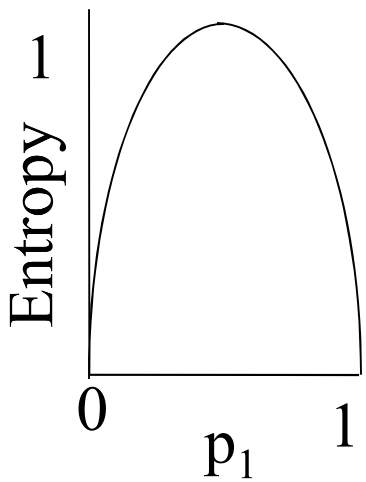
- Entropy

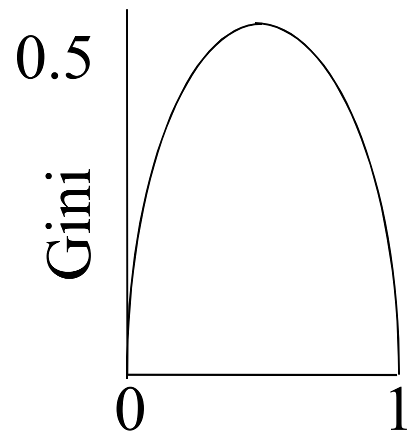
$$Entropy(S) = -\sum_{i=1}^{k} p_i \log p_i$$

- Gini

$$Gini(S) = 1 - \sum_{i=1}^{k} p_i^{\,2}$$

Information gain





- Information gain on partitioning S into r subsets
- Impurity (S) - sum of weighted impurity of each subset

$$Gain(S, S_1..S_r) = Entropy(S) - \sum_{j=1}^{r} \frac{S_j}{S} Entropy(S_j)$$

*Properties of the entropy

- The multistage property:

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

- Simplification of computation:

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

Information gain: example

$K = 2$, $|S| = 100$, $p_1 = 0.6$, $p_2 = 0.4$

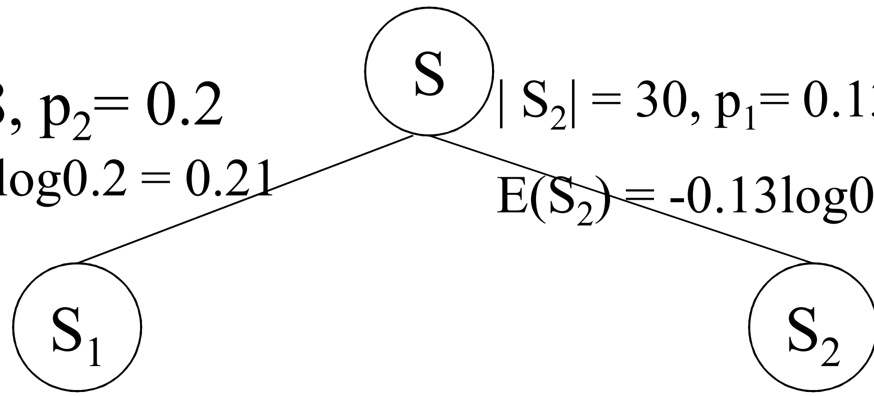
$E(S) = -0.6 \log(0.6) - 0.4 \log(0.4) = 0.29$

$| S_1 | = 70$, $p_1 = 0.8$, $p_2 = 0.2$

$E(S_1) = -0.8\log 0.8 - 0.2\log 0.2 = 0.21$

$| S_2| = 30$, $p_1 = 0.13$, $p_2 = 0.87$

$E(S_2) = -0.13\log 0.13 - 0.87 \log 0.87 = .16$

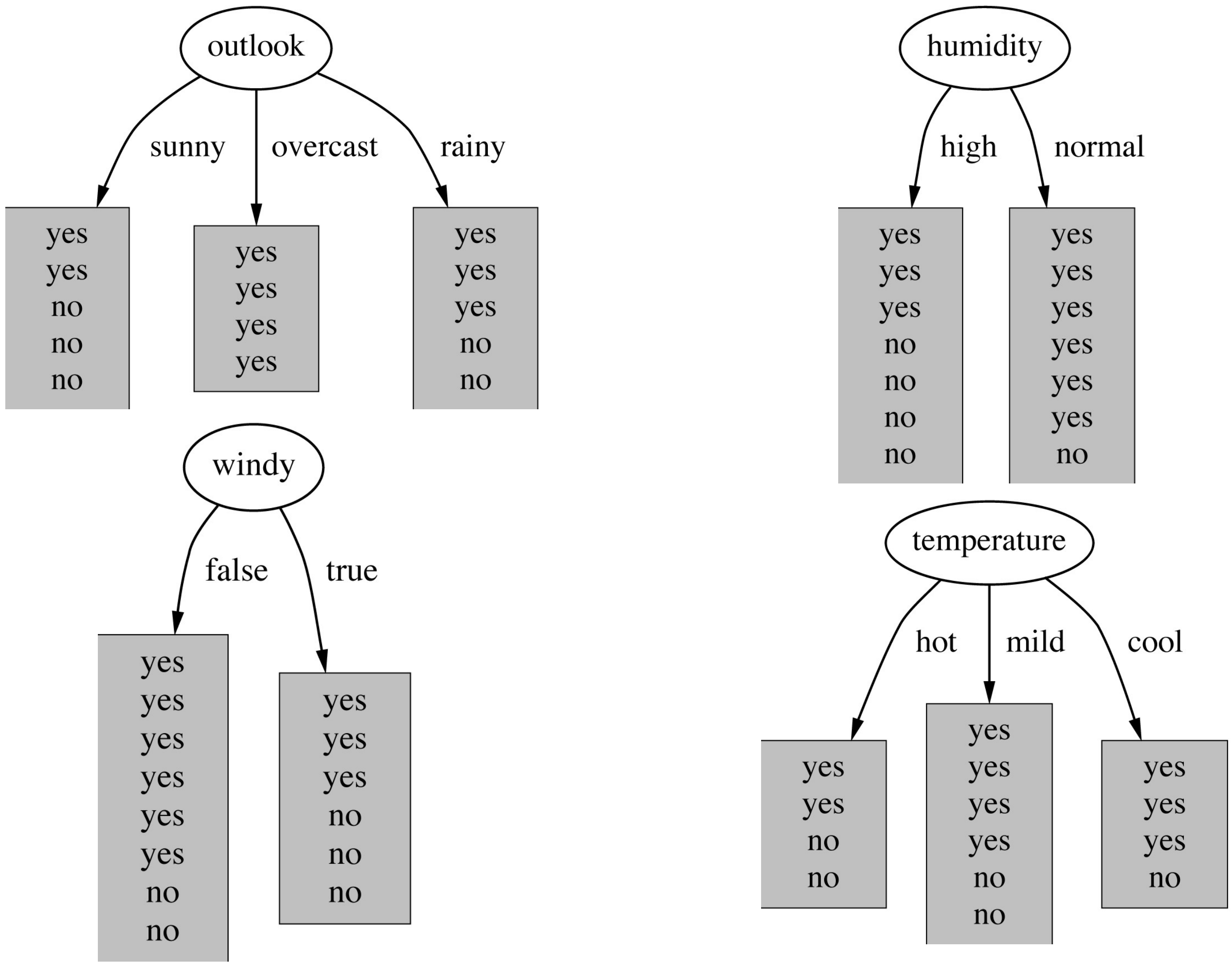


Information gain: $E(S) - (0.7\, E(S_1) + 0.3\, E(S_2)) = 0.1$

Weather Data: Play or not 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

Which attribute to select?



Example: attribute “Outlook”

- “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:

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

Computing the information gain

- Information gain:

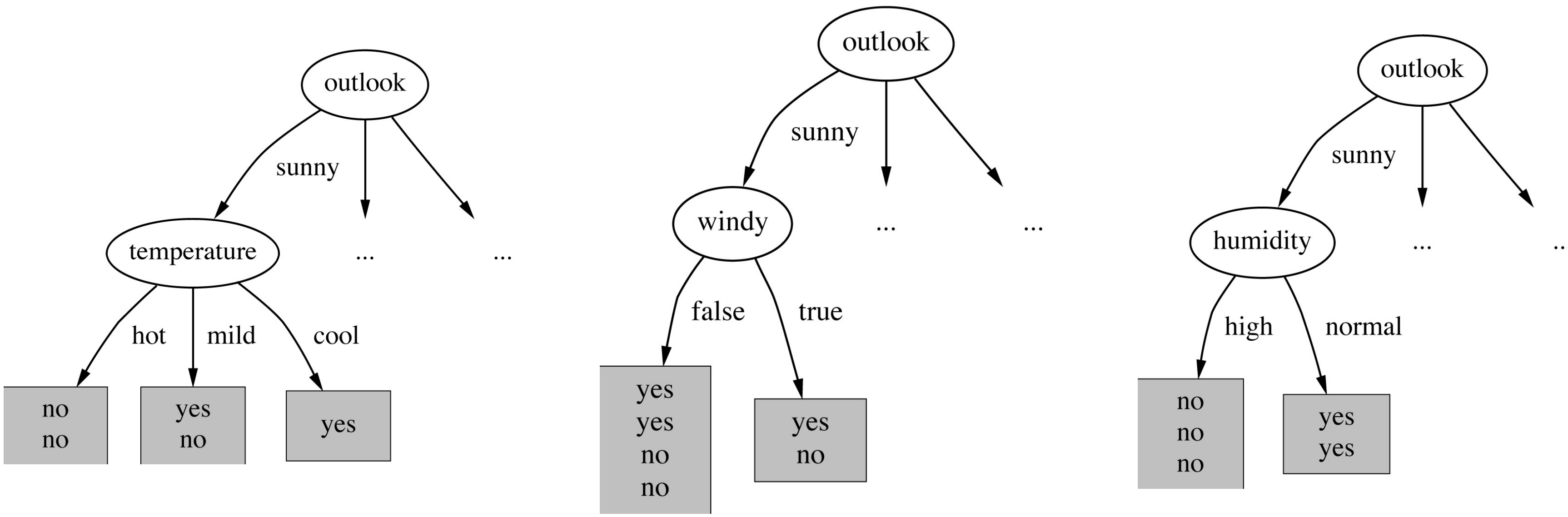
(information before split) – (information after split)

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

- 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



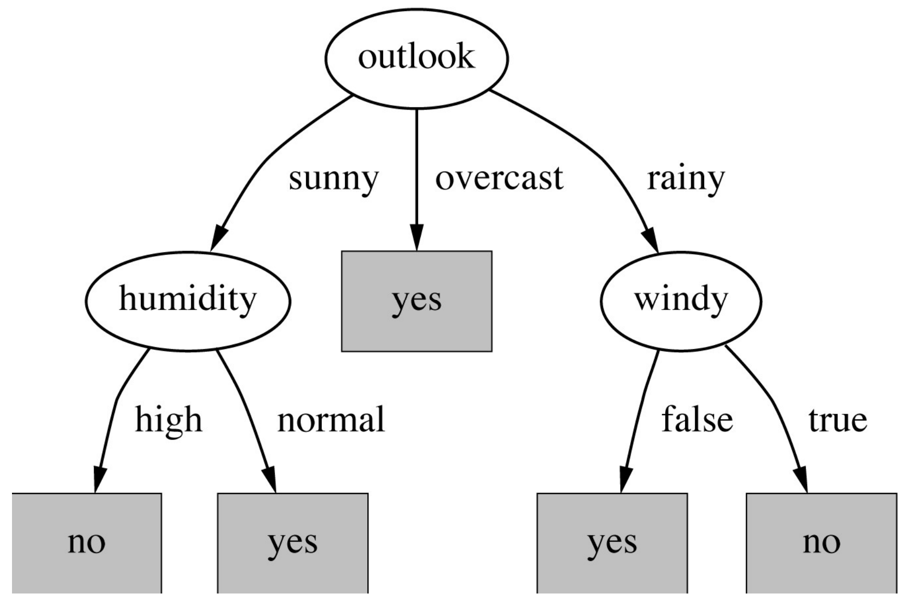
$$\text{gain("Humidity")} = 0.971 \text{ bits}$$

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

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

witten&eibe

The final decision tree



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

  ⇒ Splitting stops when data can't be split any further

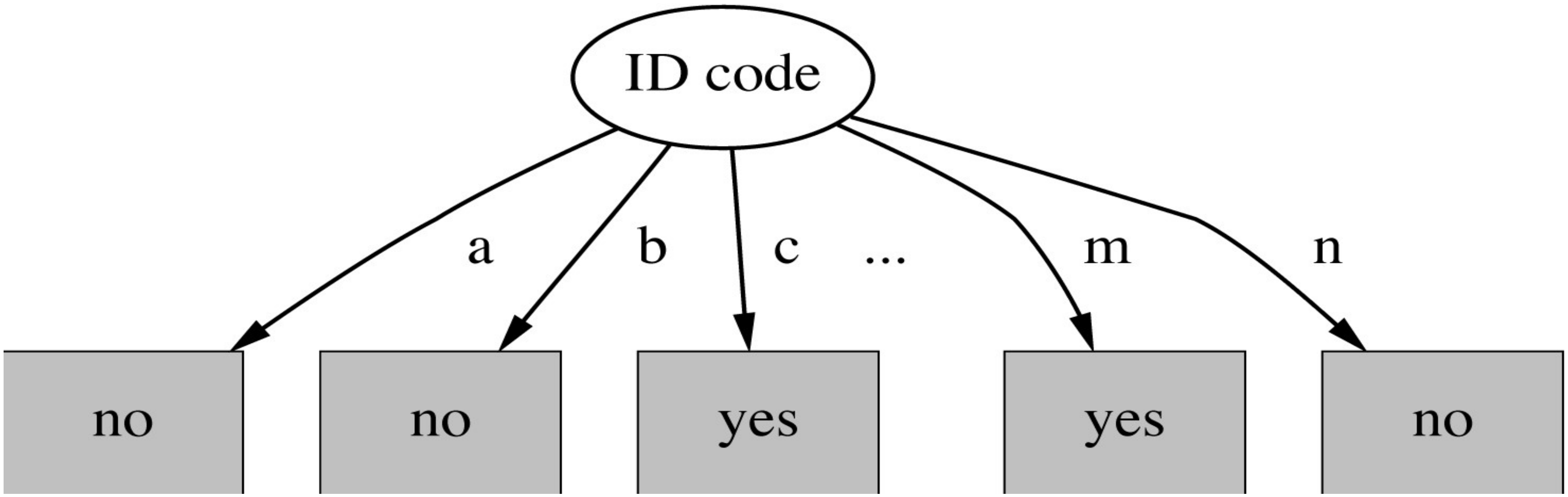
Highly-branching attributes

- 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

witten&eibe

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.

- Intrinsic information: entropy of distribution of instances into branches

$$IntrinsicInfo(S,A) \equiv -\sum \frac{|S_i|}{|S|} \log_2 \frac{|S_i|}{|S|}.$$

- *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:

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

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

More on the gain ratio

- "Outlook" still comes out top
- However: "ID code" has greater 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
  - Standard fix:
    - First, only consider attributes with greater than average information gain
    - Then, compare them on gain ratio

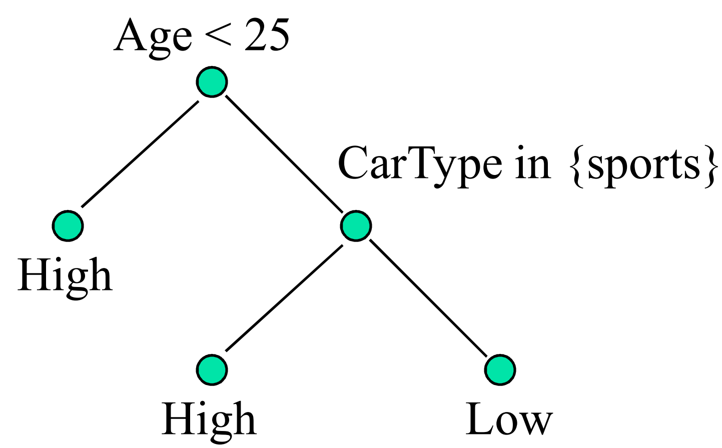
SPRINT (Serial PaRallelizable INduction of decision Trees)

- Decision-tree classifier for data mining
- Design goals:
  - Able to handle large disk-resident training sets
  - No restrictions on training-set size
  - Easily parallelizable

Example

- Example Data

Age	**Car Type**
42	family
17	truck
57	sports
21	sports
28	family
68	truck



Risk
Low
High
High
High
Low
Low

Building tree

```
GrowTree(TrainingData D)
    Partition(D);

Partition(Data D)
    if (all points in D belong to the same class) then
            return;
    for each attribute A do
            evaluate splits on attribute A;
    use best split found to partition D into D1 and D2;
    Partition(D1);
    Partition(D2);
```

Evaluating Split Points

- Gini Index
  - if data D contains examples from *c* classes

$$Gini(D) = 1 - \sum p_j^2$$

where p_j is the relative frequency of class j in D

- If D split into D_1 & D_2 with n_1 & n_2 tuples each

$$Gini_{split}(D) = \frac{n_1}{n} * gini(D_1) + \frac{n_2}{n} * gini(D_2)$$

- Note: Only class frequencies are needed to compute index

Finding Split Points

- For each attribute A do
  - evaluate splits on attribute A using attribute list

- Keep split with lowest GINI index

Split Points: Continuous Attrib.

- Consider splits of form: *value(A) < x*
  - Example: Age < 17
- Evaluate this split-form for every value in an attribute list
- To evaluate splits on attribute *A* for a given tree-node:

Initialize class-histogram of left child to zeroes;
Initialize class-histogram of right child to same as its parent;

for each record in the attribute list **do**
    evaluate splitting index for *value(A)* < *record.value*;
    using class label of the record, update class histograms;

Data Setup: Attribute Lists

- One list for each attribute
- Entries in an Attribute List consist of:
  - attribute value
  - class value
  - record id

Example list:

Age	Risk	RID
17	High	1
20	High	5
23	High	0
32	Low	4
43	High	2
68	Low	3

- Lists for continuous attributes are in sorted order
- Lists may be disk-resident

- Each leaf-node has its own set of attribute lists representing the training examples belonging to that leaf

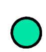
Attribute Lists: Example

Age	Car Type	Risk
23	family	High
17	sports	High
43	sports	High
68	family	Low
32	truck	Low
20	family	High

Age	Risk	RID
23	High	0
17	High	1
43	High	2
68	Low	3
32	Low	4
20	High	5

Car Type	Risk	RID
family	High	0
sports	High	1
sports	High	2
family	Low	3
truck	Low	4
family	High	5

Initial Attribute Lists for the root node:



Age	Risk	RID
17	High	1
20	High	5
23	High	0
32	Low	4
43	High	2
68	Low	3

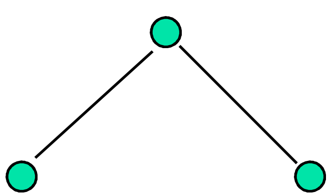
Car Type	Risk	RID
family	High	0
family	High	5
family	Low	3
sports	High	2
sports	High	1
truck	Low	4

Split Points: Continuous Attrib.

Attribute List

Age	Risk	RID
17	High	1
20	High	5
23	High	0
32	Low	4
43	High	2
68	Low	3

High	Low
4	2



Position of cursor in scan; State of Class Histograms: Left Child, Right Child; GINI Index:

Position of cursor in scan	Left Child: High	Left Child: Low	Right Child: High	Right Child: Low	GINI Index:
0: Age < 17	0	0	4	2	GINI = undef
1: Age < 20	1	0	3	2	GINI = 0.4
3: Age < 32	3	0	1	2	GINI = 0.222
6	4	2	0	0	GINI = undef

Split Points: Categorical Attrib.

- Consider splits of the form: *value(A)* $\in$ *{x1, x2, ..., xn}*
  - Example: CarType $\in$ {family, sports}
- Evaluate this split-form for subsets of *domain(A)*
- To evaluate splits on attribute *A* for a given tree node:

```
initialize class/value matrix of node to zeroes;

for each record in the attribute list do
        increment appropriate count in matrix;

evaluate splitting index for various subsets using the constructed matrix;
```

Finding Split Points: Categorical Attrib.

Attribute List

Car Type	Risk	RID
family	High	0
family	High	5
family	Low	3
sports	High	2
sports	High	1
truck	Low	4

class/value matrix

	High	Low
family	2	1
sports	2	0
truck	0	1

	Left Child		Right Child		GINI Index:
	High	**Low**	**High**	**Low**	
CarType in {family}	2	1	2	1	GINI = 0.444
CarType in {sports}	2	0	2	2	GINI = 0.333
CarType in {truck}	0	1	4	1	GINI = 0.267

Performing the Splits

- The attribute lists of every node must be divided among the two children
- To split the attribute lists of a give node:

for the list of the attribute used to split this node **do**
    use the split test to divide the records;
    collect the record ids;

build a hashtable from the collected ids;

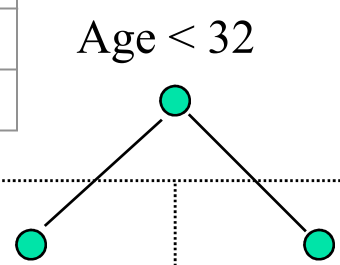
for the remaining attribute lists **do**
    use the hashtable to divide each list;

build class-histograms for each new leaf;

Performing the Splits: Example

Age	Risk	RID
17	High	1
20	High	5
23	High	0
32	Low	4
43	High	2
68	Low	3

Car Type	Risk	RID
family	High	0
family	High	5
family	Low	3
sports	High	2
sports	High	1
truck	Low	4



Age	Risk	RID
17	High	1
20	High	5
23	High	0

Age	Risk	RID
32	Low	4
43	High	2
68	Low	3

Hash Table

0 → Left
1 → Left
2 → Right
3 → Right
4 → Right
5 → Left

Car Type	Risk	RID
family	High	0
family	High	5
sports	High	1

Car Type	Risk	RID
family	Low	3
sports	High	2
truck	Low	4

Sprint: summary

- Each node of the decision tree classifier, requires examining possible splits on each value of each attribute.
- After choosing a split attribute, need to partition *all* data into its subset.
- Need to make this search efficient.
- Evaluating splits on numeric attributes:
  - Sort on attribute value, incrementally evaluate gini
- Splits on categorical attributes
  - For each subset, find gini and choose the best
  - For large sets, use greedy method

Preventing overfitting

- A tree T overfits if there is another tree T′ that gives higher error on the training data yet gives lower error on unseen data.
- An overfitted tree does not generalize to unseen instances.
- Happens when data contains noise or irrelevant attributes and training size is small.
- Overfitting can reduce accuracy drastically:
  - 10-25% as reported in Minger's 1989 Machine learning
- Example of over-fitting with binary data.

Training Data Vs. Test Data Error Rates

- Compare error rates measured by
  - learn data
  - large test set
- Learn R(T) always decreases as tree grows (Q: Why?)
- Test R(T) first declines then increases (Q: Why?)
- Overfitting is the result tree of too much reliance on learn R(T)
- Can lead to disasters when applied to new data

No. Terminal Nodes	R(T)	R^{ts}(T)
71	.00	.42
63	.00	.40
58	.03	.39
40	.10	.32
34	.12	.32
19	.20	.31
**10	.29	.30
9	.32	.34
7	.41	.47
6	.46	.54
5	.53	.61
2	.75	.82
1	.86	.91

Digit recognition dataset: CART book

Overfitting example

- Consider the case where a single attribute xj is adequate for classification but with an error of 20%
- Consider lots of other noise attributes that enable zero error during training
- This detailed tree during testing will have an expected error of $(0.8*0.2 + 0.2*0.8) = 32\%$ whereas the pruned tree with only a single split on xj will have an error of only 20%.

Occam’s Razor

- *prefer the simplest hypothesis that fits the data*
- Two interpretations:
  - Of two models with the same generalization error, prefer the simpler because simplicity is a goal in itself
  - Of two models with the same error on training data, the simpler model will have smaller generalization error
- Second interpretation questionable

Approaches to prevent overfitting

- Two Approaches:
  - Stop growing the tree beyond a certain point
    - Tricky, since even when information gain is zero an attribute might be useful (XOR example)
  - First over-fit, then post prune. (More widely used)
    - Tree building divided into phases:
      - Growth phase
      - Prune phase

Criteria for finding correct final tree size:

- Three criteria:
  - Cross validation with separate test data
  - Statistical bounds: use all data for training but apply statistical test to decide right size. (cross-validation dataset may be used to threshold)
  - Use some criteria function to choose best size
    - Example: Minimum description length (MDL) criteria

Cross validation

- Partition the dataset into two disjoint parts:
    - 1. Training set used for building the tree.
    - 2. Validation set used for pruning the tree:
  - Rule of thumb: 2/3rds training, 1/3rd validation
- Evaluate the tree on the validation set and at each leaf and internal node keep count of correctly labeled data.
  - Starting bottom-up, prune nodes with error less than its children.
- What if training data set size is limited?
  - n-fold cross validation: partition training data into n parts D1, D2…Dn.
  - Train n classifiers with D-Di as training and Di as test instance.
  - Pick average. (how?)

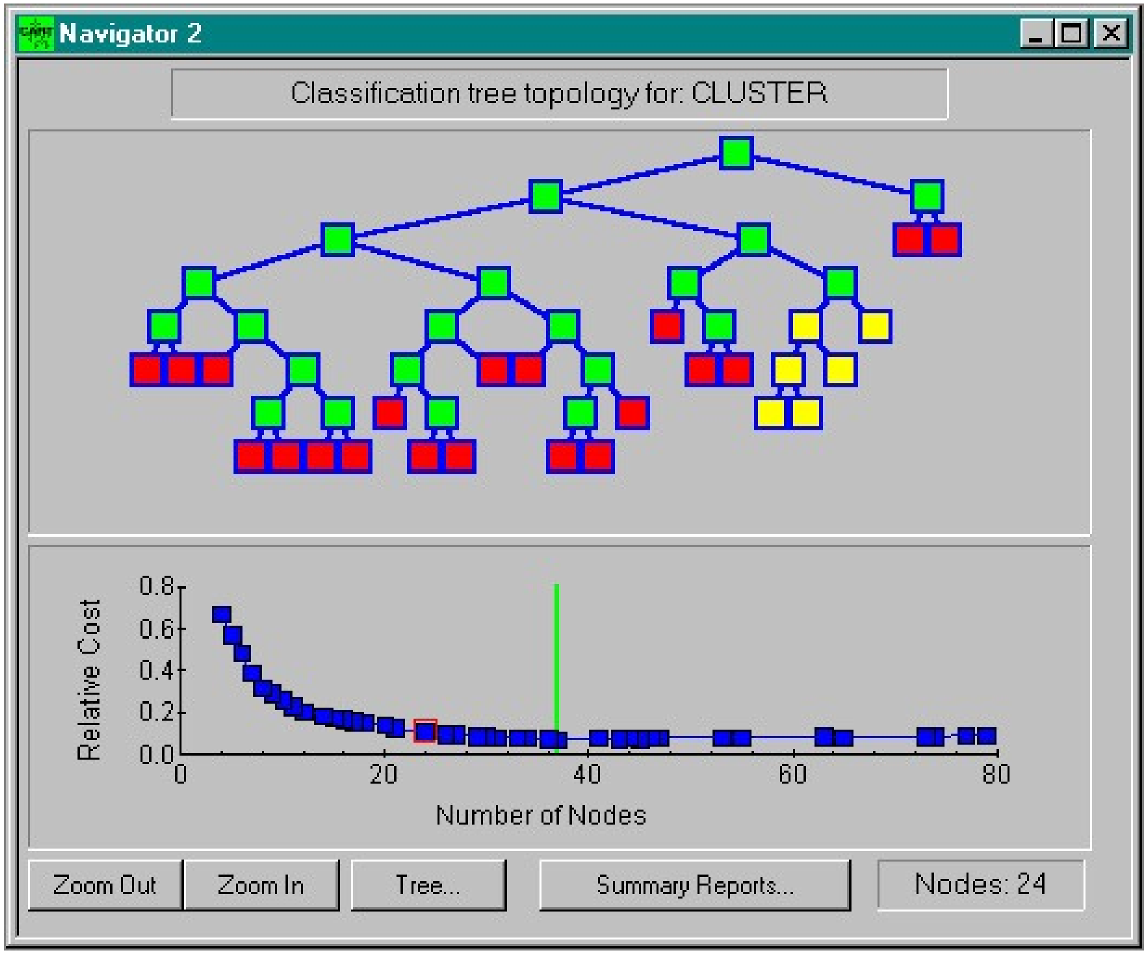
That was a simplistic view..

- A tree with minimum error on a single test set may not be stable.
- In what order do you prune?

Minimum Cost complexity pruning in CART

- For each cross-validation run
  - Construct the full tree Tmax
  - Use some error estimates to prune Tmax
    - Delete subtrees in decreasing order of strength..
    - All subtrees of the same strength go together.
  - This gives several trees of various sizes
  - Use validation partition to note error against various tree size.
- Choose tree size with smallest error over all CV partitions
- Run a complicated search involving growing and shrinking phases to find the best tree of the chosen size using all data.

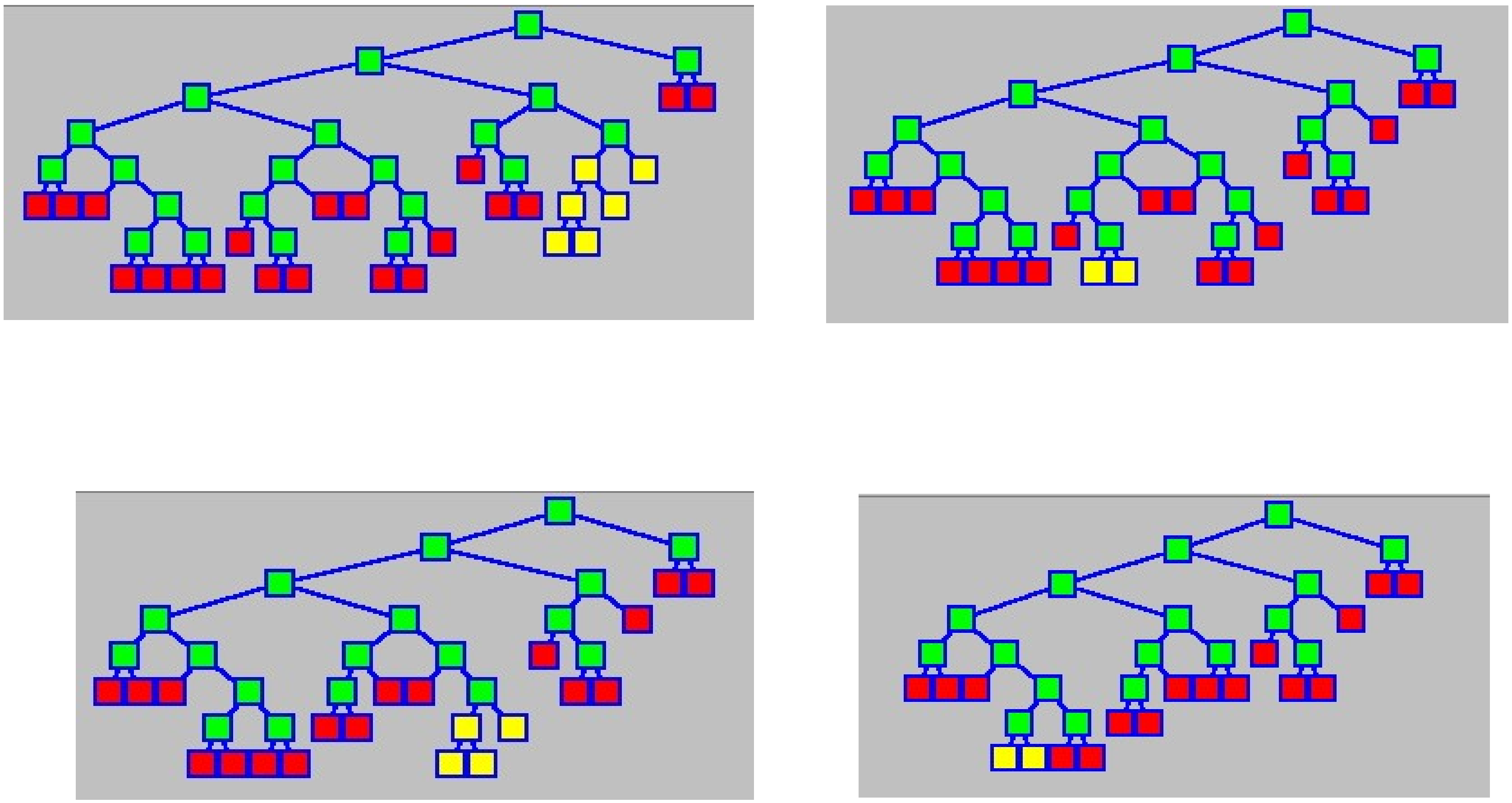
Pruning: Which nodes come off



Order of Pruning: Weakest Link Goes First

- Prune away *"weakest link"* — the nodes that add least to overall accuracy of the tree
    - contribution to overall tree a function of both increase in accuracy and size of node
    - accuracy gain is weighted by share of sample
    - small nodes tend to get removed before large ones
- If several nodes have same contribution they all prune away simultaneously
    - Hence more than two terminal nodes could be cut off in one pruning
- Sequence determined all the way back to root node
    - need to allow for possibility that entire tree is bad
    - if target variable is unpredictable we will want to prune back to root . . . the no model solution

Pruning Sequence Example



Now we test every tree in the pruning sequence

- Take a test data set and drop it down the largest tree in the sequence and measure its predictive accuracy
  - how many cases right and how many wrong
  - measure accuracy overall and by class
- Do same for 2nd largest tree, 3rd largest tree, etc
- Performance of every tree in sequence is measured
- Results reported in table and graph formats
- Note that this critical stage is impossible to complete without test data
- CART procedure requires test data to guide tree evaluation

Pruning via significance tests

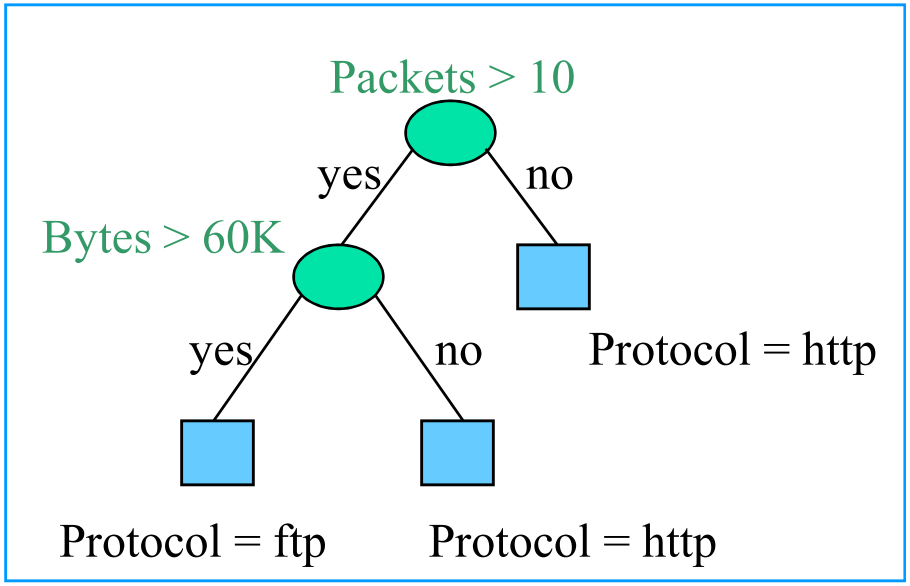
- For each node test on the training data if the class label is independent of the splits of the attribute of this node. Prune if independent.
  - A common statistical test for independence is the Chi-squared test
    - Chi-squared test of independence (board)
  - A second test of independence is mutual information
    - \sum_{x,y} p(x,y) log(p(x,y)/p(x)p(y)

The minimum description length principle (MDL)

- MDL: paradigm for statistical estimation particularly model selection
- Given data D and a class of models M, our choose is to choose a model m in M such that data and model can be encoded using the smallest total length
  - $L(D) = L(D|m) + L(m)$
- How to find encoding length?
  - Answer in Information Theory
  - Consider the problem of transmitting n messages where pi is probability of seeing message i
  - Shannon's theorem: minimum expected length when
    - -log pi bits to message i

MDL Example: Compression with Classification Trees

bytes	packets	protocol
20K	3	http
24K	5	http
20K	8	http
40K	11	ftp
58K	18	http
100K	24	ftp
300K	35	ftp
80K	15	http



Outlier: Row 4, protocol=ftp,
Row 8, protocol =http

From: Anthony Tung, NUS

Encoding data

- Assume t records of training data D
- First send tree m using L(m|M) bits
- Assume all but the class labels of training data known.
- Goal: transmit class labels using L(D|m)
- If tree correctly predicts an instance, 0 bits
- Otherwise, log k bits where k is number of classes.
- Thus, if e errors on training data: total cost
- e log k + L(m|M) bits.
- Complex tree will have higher L(m|M) but lower e.
- Question: how to encode the tree?

Extracting Classification Rules from Trees

- Represent the knowledge in the form of IF-THEN rules
- One rule is created for each path from the root to a leaf
- Each attribute-value pair along a path forms a conjunction
- The leaf node holds the class prediction
- Rules are easier for humans to understand
- Example

  IF *age* = "<=30" AND *student* = "*no*" THEN *buys_computer* = "*no*"

  IF *age* = "<=30" AND *student* = "*yes*" THEN *buys_computer* = "*yes*"

  IF *age* = "31…40" THEN *buys_computer* = "*yes*"

  IF *age* = ">40" AND *credit_rating* = "*excellent*" THEN *buys_computer* = "*yes*"

  IF *age* = "<=30" AND *credit_rating* = "*fair*" THEN *buys_computer* = "*no*"

Rule-based pruning

- Tree-based pruning limits the kind of pruning. If a node is pruned all subtrees under it has to be pruned.
- Rule-based: For each leaf of the tree, extract a rule using a conjuction of all tests upto the root.
- On the validation set, independently prune tests from each rule to get the highest accuracy for that rule.
- Sort rule by decreasing accuracy..

Regression trees

- Decision tree with continuous class labels:
- Regression trees approximates the function with piece-wise constant regions.
- Split criteria for regression trees:
  - Predicted value for a set S = average of all values in S
  - Error: sum of the square of error of each member of S from the predicted average.
  - Pick smallest average error.
- Splits on categorical attributes:
  - Can it be better than for discrete class labels?
  - Homework.

Other types of trees

- Multi-way trees on low-cardinality categorical data
- Multiple splits on continuous attributes [Fayyad 93, Multi-interval discretization of continuous attributes]
- Multi attribute tests on nodes to handle correlated attributes
  - multivariate linear splits [Oblique trees, Murthy 94]

Issues

- Methods of handling missing values
  - assume majority value
  - take most probable path
- Allowing varying costs for different attributes

Pros and Cons of decision trees

- Pros
  - + Reasonable training time
  - + Fast application
  - + Easy to interpret
  - + Easy to implement
  - + Intuitive

- Cons
  - – Not effective for very high dimensional data where information about the class is spread in small ways over many correlated features
    - –Example: words in text classification
  - –Not robust to dropping of important features even when correlated substitutes exist in data