2 Nephi 31

19 And now, my beloved brethren, after ye have gotten into this strait and narrow path, **I would ask if all is done?** Behold, I say unto you, **Nay**; for ye have not come thus far save it were by the word of Christ with unshaken faith in him, relying wholly upon the merits of him who is mighty to save.

20 Wherefore, **ye must press forward with a steadfastness in Christ**, having a perfect brightness of hope, and a love of God and of all men. Wherefore, if ye shall press forward, feasting upon the word of Christ, and endure to the end, behold, thus saith the Father: Ye shall have eternal life.

21 And now, behold, my beloved brethren, **this is the way**; and there is none other way nor name given under heaven whereby man can be saved in the kingdom of God. And now, behold, this is the doctrine of Christ, and the only and true doctrine of the Father, and of the Son, and of the Holy Ghost, which is one God, without end. Amen.

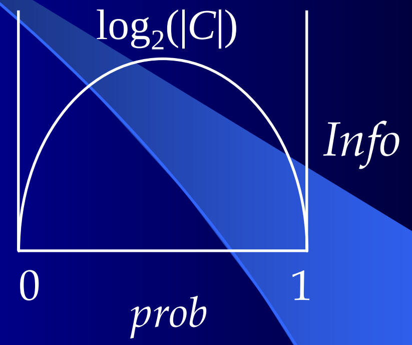
Decision Trees

Information

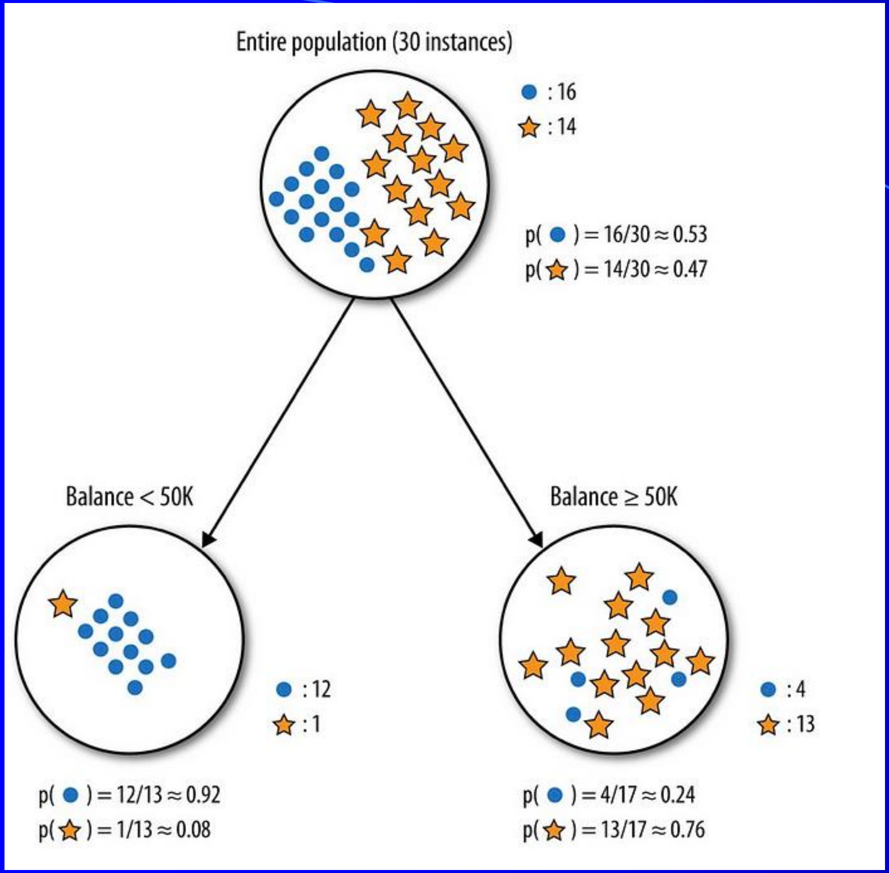
- Information of a message in bits: $I(m) = -\log_2(p_m)$
- If there are 16 equiprobable messages, I for each message is $-\log_2(1/16) = 4$ bits
  - It takes 4 bits to determine which message (Shannon's Information Theory)
- If the messages are not equiprobable then could we represent them with less bits?
  - Highest disorder (randomness) requires maximum information
- If there is a dataset S of c classes, then information for one class is: $I(c) = -\log_2(p_c)$
- Total info of the data set is just the sum of (the info per class times the proportion of that class)
- Info(S) = Entropy(S) = $-\sum_{i=1}^{|C|} p_i \log_2(p_i)$

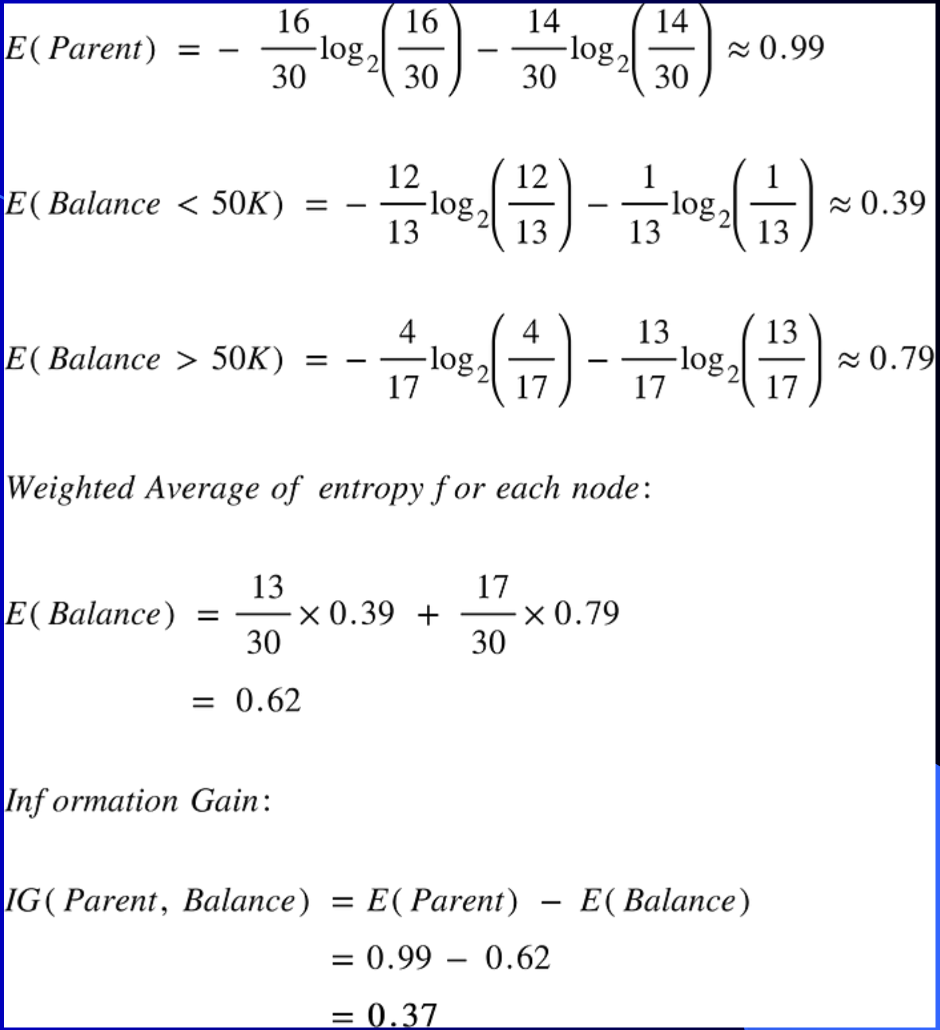
Information Gain Metric

- Info(S) is the average amount of information needed to identify the class of an example in set S
- Info(S) = Entropy(S) = $-\sum_{i=1}^{|C|} p_i \log_2(p_i)$
- $0 \le$ Info(S) $\le \log_2(|C|)$, $|C|$ is # of output classes
- p_i is the probability of each output class
- Expected Information after partitioning using A:
- Info$_A$(S) = $\sum_{i=1}^{|A|} \frac{|S_i|}{|S|} Info(S_i)$ where $|A|$ is # of values for attribute A



- Mostly pure sets have Info(S) = Entropy(S) ≈ 0
- Gain(A) = Info(S) - Info$_A$(S) (i.e. minimize Info$_A$(S))
- Gain/Entropy does not handle the statistical significance issue





$$E(Parent) = -\frac{16}{30}\log_2\left(\frac{16}{30}\right) - \frac{14}{30}\log_2\left(\frac{14}{30}\right) \approx 0.99$$

$$E(Balance < 50K) = -\frac{12}{13}\log_2\left(\frac{12}{13}\right) - \frac{1}{13}\log_2\left(\frac{1}{13}\right) \approx 0.39$$

$$E(Balance > 50K) = -\frac{4}{17}\log_2\left(\frac{4}{17}\right) - \frac{13}{17}\log_2\left(\frac{13}{17}\right) \approx 0.79$$

Weighted Average of entropy for each node:

$$E(Balance) = \frac{13}{30}\times 0.39 + \frac{17}{30}\times 0.79 = 0.62$$

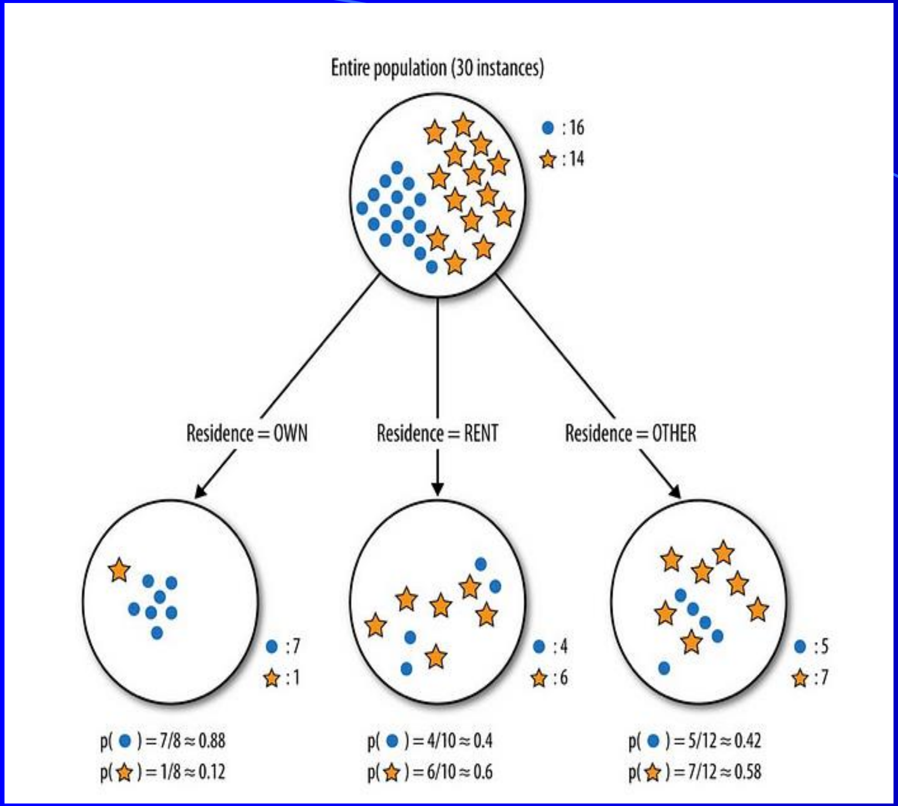
Information Gain:

$$\begin{aligned} IG(Parent, Balance) &= E(Parent) - E(Balance) \\ &= 0.99 - 0.62 \\ &= 0.37 \end{aligned}$$

$$Info(S) = -\sum_{i=1}^{|C|} p_i log_2 p_i$$

$$Info_A(S) = \sum_{j=1}^{|A|} \frac{|S_j|}{|S|} Info(S_j) = \sum_{j=1}^{|A|} \frac{|S_j|}{|S|} \cdot -\sum_{i=1}^{|C|} p_i log_2 p_i$$

Gain(A) = Info(S) - Info$_A$(S)



$$E(Residence = OWN) = -\frac{7}{8}\log_2\left(\frac{7}{8}\right) - \frac{1}{8}\log_2\left(\frac{1}{8}\right) \approx 0.54$$

$$E(Residence = RENT) = -\frac{4}{10}\log_2\left(\frac{4}{10}\right) - \frac{6}{10}\log_2\left(\frac{6}{10}\right) \approx 0.97$$

$$E(Residence = OTHER) = -\frac{5}{12}\log_2\left(\frac{5}{12}\right) - \frac{7}{12}\log_2\left(\frac{7}{12}\right) \approx 0.98$$

Weighted Average of entropies for each node:

$$E(Residence) = \frac{8}{30} \times 0.54 + \frac{10}{30} \times 0.97 + \frac{12}{30} \times 0.98 = 0.86$$

Information Gain:

$$\begin{aligned} IG(Parent, Residence) &= E(Parent) - E(Residence) \\ &= 0.99 - 0.86 \\ &= 0.13 \end{aligned}$$

$$Info(S) = -\sum_{i=1}^{|C|} p_i log_2 p_i$$

$$Info_A(S) = \sum_{j=1}^{|A|} \frac{|S_j|}{|S|} Info(S_j) = \sum_{j=1}^{|A|} \frac{|S_j|}{|S|} \cdot -\sum_{i=1}^{|C|} p_i log_2 p_i$$

Gain(A) = Info(S) - Info$_A$(S)

ID3/C4.5 Learning Algorithm

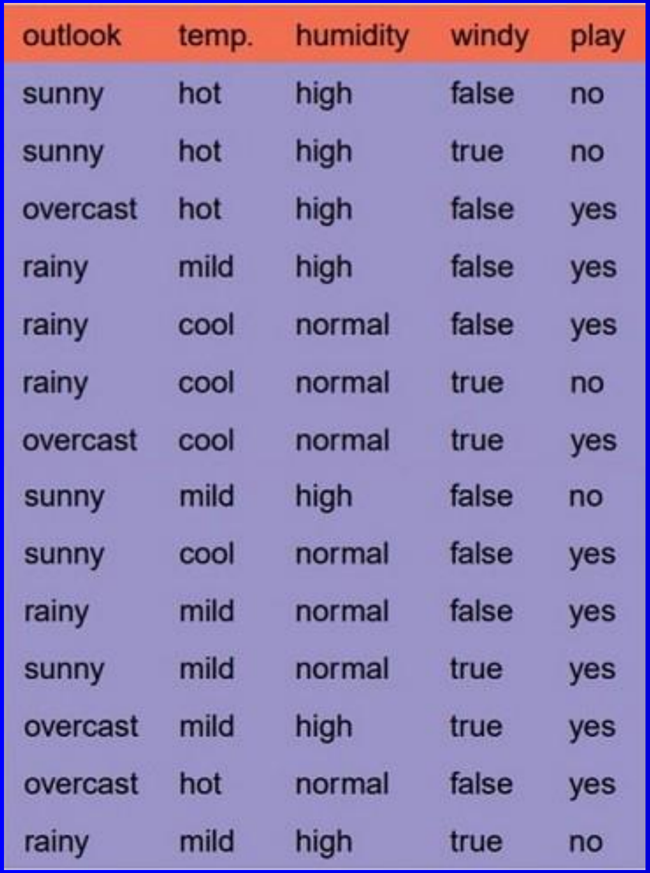
1. S = Training Set
2. Calculate gain for each remaining attribute: Gain(A) = Info(S) - Info$_A$(S)
3. Select attribute with highest gain and create a new node for each partition
4. For each new child node
   - if pure (one class), or if no attributes remain, or if stopping criteria met (e.g. pure enough, too small a number of examples remaining, not enough information gain, max depth reached), then label node with majority class and end
   - else recurse to 2 with remaining attributes and training set

$$Info(S) = -\sum_{i=1}^{|C|} p_i log_2 p_i$$

$$Info_A(S) = \sum_{j=1}^{|A|} \frac{|S_j|}{|S|} Info(S_j) = \sum_{j=1}^{|A|} \frac{|S_j|}{|S|} \cdot -\sum_{i=1}^{|C|} p_i log_2 p_i$$

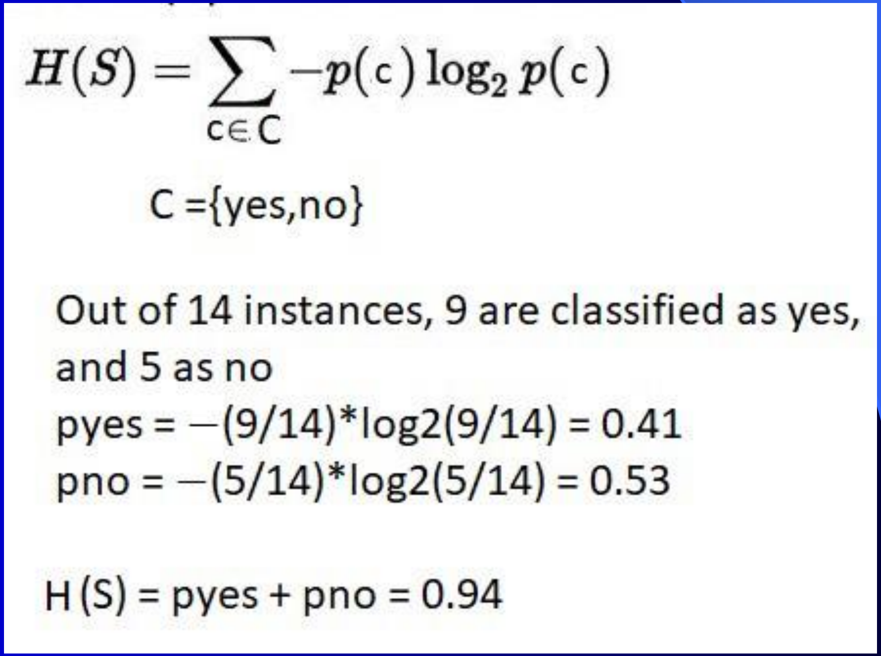
Where S is the remaining training set at the node
$|A|$ is the number of attribute values for the feature
$|C|$ is the number of output classes for the task

Example



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

1. compute the entropy for data-set
2. for every attribute/feature:
    1. calculate entropy for all categorical values
    2. take average information entropy for the current attrib
    3. calculate gain for the current attribute
3. pick the highest gain attribute.
4. Repeat until we get the tree we desired.



$$H(S) = \sum_{c \in C} -p(c) \log_2 p(c)$$

C ={yes,no}

Out of 14 instances, 9 are classified as yes, and 5 as no

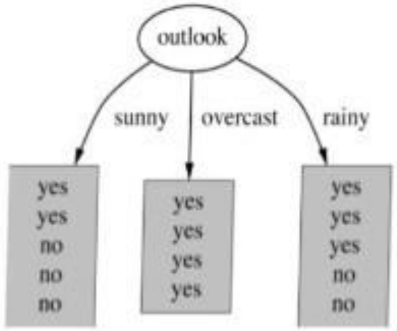
pyes = $-(9/14)*\log2(9/14) = 0.41$

pno = $-(5/14)*\log2(5/14) = 0.53$

H (S) = pyes + pno = 0.94

Example

For every feature, calculate the entropy and the information gain



$$E\,(\text{Outlook=sunny}) = -\tfrac{2}{5}\log\left(\tfrac{2}{5}\right) - \tfrac{3}{5}\log\left(\tfrac{3}{5}\right) = 0.971$$

$$E\,(\text{Outlook=overcast}) = -1\log(1) - 0\log(0) = 0$$

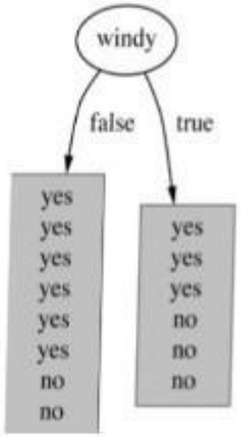
$$E\,(\text{Outlook=rainy}) = -\tfrac{3}{5}\log\left(\tfrac{3}{5}\right) - \tfrac{2}{5}\log\left(\tfrac{2}{5}\right) = 0.971$$

$$\Big\} \; H(S,\text{Outlook})$$

Average Entropy information for Outlook

$$I\,(\text{Outlook}) = \tfrac{5}{14} * 0.971 + \tfrac{4}{14} * 0 + \tfrac{5}{14} * 0.971 = 0.693 \;\Big\}\; \sum_{t\in T} p(t)H(t)$$

$$\text{Gain (Outlook)} = E(S) - I\,(\text{outlook}) = 0.94\text{-}.693 = 0.247 \;\Longrightarrow\; IG(A,S) = H(S) - \sum_{t\in T} p(t)H(t)$$



$$E\,(\text{Windy=false}) = -\tfrac{6}{8}\log\left(\tfrac{6}{8}\right) - \tfrac{2}{8}\log\left(\tfrac{2}{8}\right) = 0.811$$

$$E\,(\text{Windy=true}) = -\tfrac{3}{6}\log\left(\tfrac{3}{6}\right) - \tfrac{3}{6}\log\left(\tfrac{3}{6}\right) = 1$$

Average entropy information for Windy

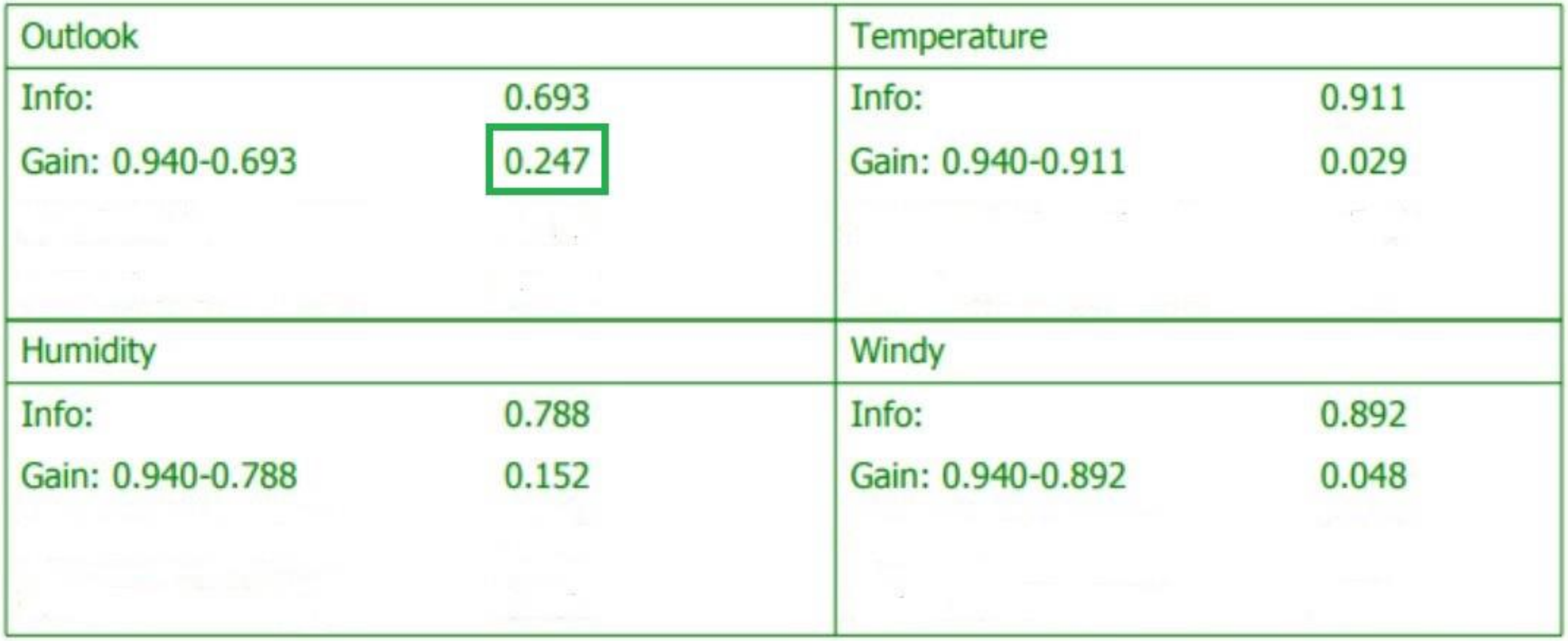
$$I\,(\text{Windy}) = \tfrac{8}{14} * 0.811 + \tfrac{6}{14} * 1 = 0.892$$

$$\text{Gain (Windy)} = E(S) - I\,(\text{Windy}) = 0.94\text{-}0.892\text{=}0.048$$

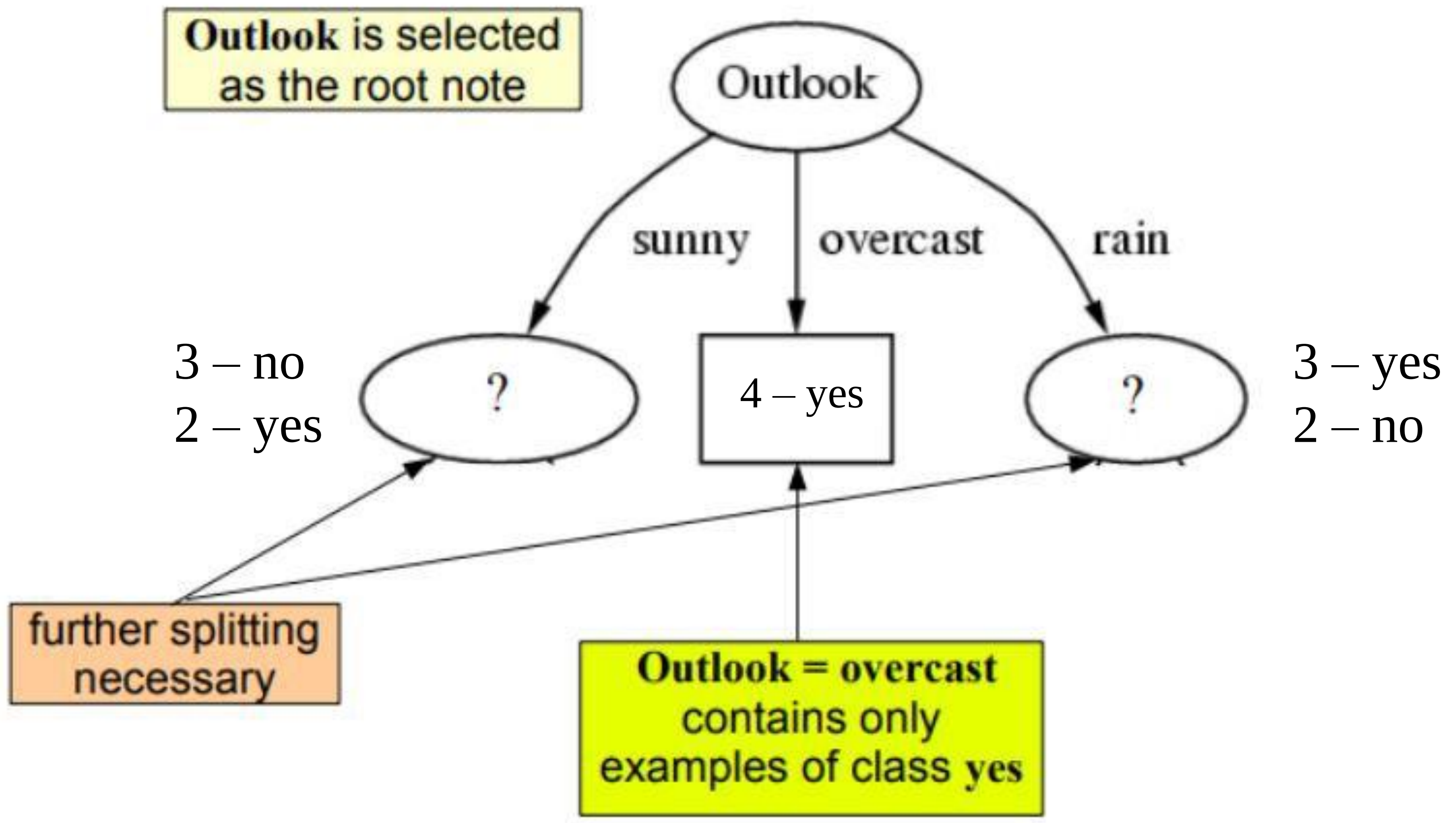
Similarly, calculate the entropy and the information gain for Humidity and Temperature

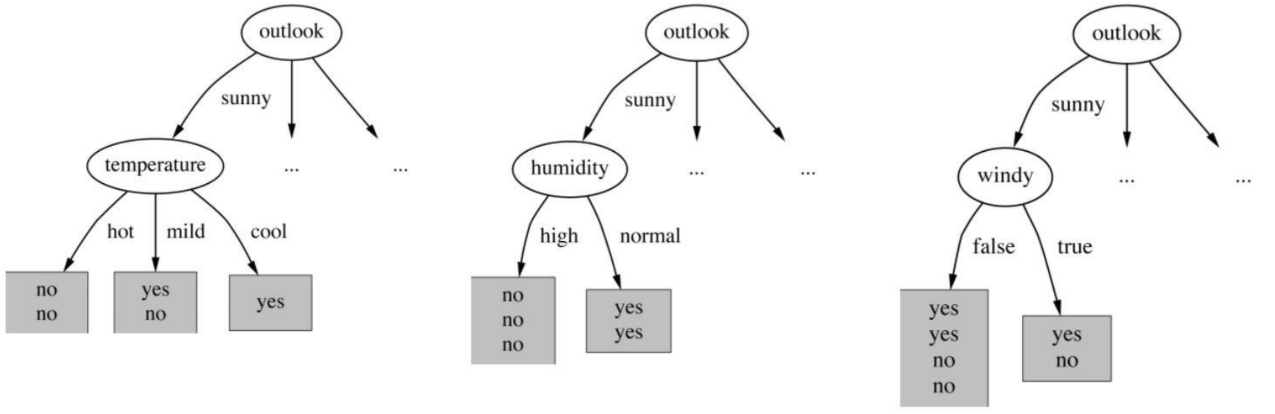
Example

Pick the highest Gain attribute



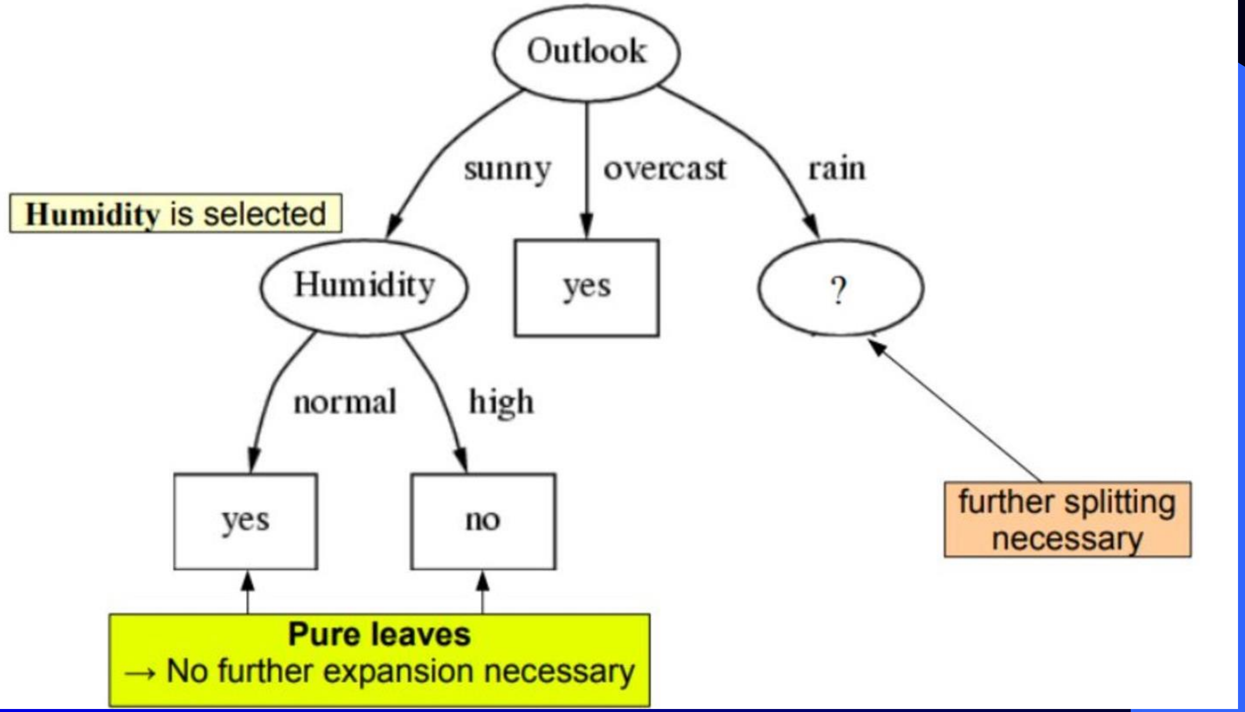
Outlook		Temperature	
Info:	0.693	Info:	0.911
Gain: 0.940-0.693	$\boxed{0.247}$	Gain: 0.940-0.911	0.029
Humidity		Windy	
Info:	0.788	Info:	0.892
Gain: 0.940-0.788	0.152	Gain: 0.940-0.892	0.048


Outlook is selected as the root note
Outlook
sunny
overcast
rain
3 – no
2 – yes
?
4 – yes
?
3 – yes
2 – no
further splitting necessary
Outlook = overcast contains only examples of class yes

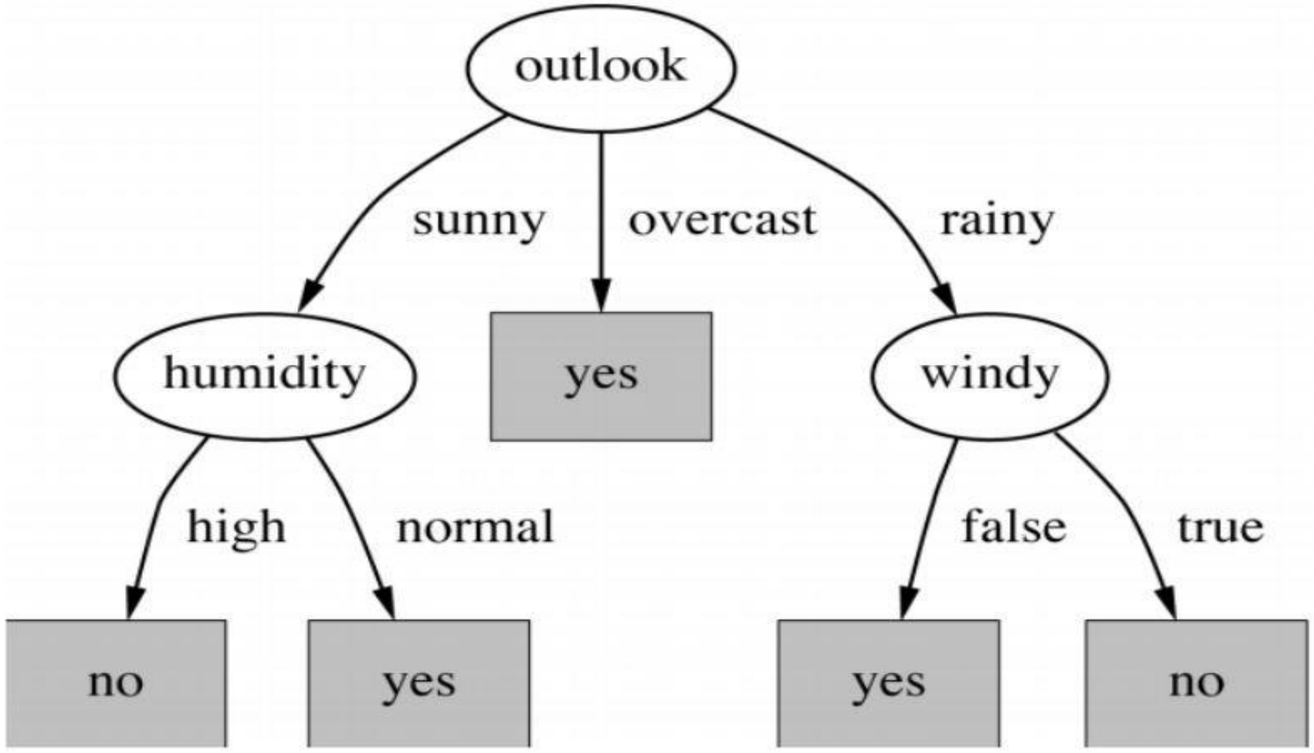


$$\left.\begin{aligned}\text{Gain}(\textit{Temperature}) &= 0.571 \text{ bits}\\ \text{Gain}(\textit{Humidity}) &= 0.971 \text{ bits}\\ \text{Gain}(\textit{Windy}) &= 0.020 \text{ bits}\end{aligned}\right\}$$

Humidity is selected



Final decision tree



C4.5 Learning Algorithm

1. S = Training Set
2. Calculate gain for each remaining attribute: Gain(A) = Info(S) - Info$_A$(S)
3. Select attribute with highest gain and create a new node for each partition
4. For each new child node
   - if pure (one class), or if no attributes remain, or if stopping criteria met (e.g. pure enough, too small a number of examples remaining, not enough information gain, max depth reached), then label node with majority class and end
   - else recurse to 2 with remaining attributes and training set

$$Info(S) = -\sum_{i=1}^{|C|} p_i log_2 p_i$$

$$Info_A(S) = \sum_{j=1}^{|A|} \frac{|S_j|}{|S|} Info(S_j) = \sum_{j=1}^{|A|} \frac{|S_j|}{|S|} \cdot -\sum_{i=1}^{|C|} p_i log_2 p_i$$

Where S is the remaining training set at the node
|A| is the number of attribute values for the feature
|C| is the number of output classes for the task

Meat N,Y	Crust D,S,T	Veg N,Y	Quality B,G,Gr
Y	Thin	N	Great
N	Deep	N	Bad
N	Stuffed	Y	Good
Y	Stuffed	Y	Great
Y	Deep	N	Good
Y	Deep	Y	Great
N	Thin	Y	Good
Y	Deep	N	Good
N	Thin	N	Bad

Meat N,Y	Crust D,S,T	Veg N,Y	Quality B,G,Gr
Y	Thin	N	Great
N	Deep	N	Bad
N	Stuffed	Y	Good
Y	Stuffed	Y	Great
Y	Deep	N	Good
Y	Deep	Y	Great
N	Thin	Y	Good
Y	Deep	N	Good
N	Thin	N	Bad

Example

$$Info(S) = -\sum_{i=1}^{|C|} p_i log_2 p_i$$

Where *S* is the remaining training set at the node
|*A*| is the number of attribute values for the feature
|*C*| is the number of output classes for the task

$$Info_A(S) = \sum_{j=1}^{|A|} \frac{|S_j|}{|S|} Info(S_j) = \sum_{j=1}^{|A|} \frac{|S_j|}{|S|} \cdot -\sum_{i=1}^{|C|} p_i log_2 p_i$$

- Info(*S*) = - 2/9·log$_2$2/9 - 4/9·log$_2$4/9 -3/9·log$_2$3/9 = 1.53
  - Not necessary, but gain can be used as a stopping criteria
- Starting with all instances, calculate gain for each attribute
- Let's do Meat:
- Info$_{Meat}$(*S*) = ?
  - Information Gain is ?

Meat N,Y	Crust D,S,T	Veg N,Y	Quality B,G,Gr
Y	Thin	N	Great
N	Deep	N	Bad
N	Stuffed	Y	Good
Y	Stuffed	Y	Great
Y	Deep	N	Good
Y	Deep	Y	Great
N	Thin	Y	Good
Y	Deep	N	Good
N	Thin	N	Bad

Example

$$Info(S) = -\sum_{i=1}^{|C|} p_i log_2 p_i$$

Where S is the remaining training set at the node
$|A|$ is the number of attribute values for the feature
$|C|$ is the number of output classes for the task

$$Info_A(S) = \sum_{j=1}^{|A|} \frac{|S_j|}{|S|} Info(S_j) = \sum_{j=1}^{|A|} \frac{|S_j|}{|S|} \cdot -\sum_{i=1}^{|C|} p_i log_2 p_i$$

- Info(S) = - 2/9·log$_2$2/9 - 4/9·log$_2$4/9 -3/9·log$_2$3/9 = 1.53
  - Not necessary, but gain can be used as a stopping criteria
- Starting with all instances, calculate gain for each attribute
- Let’s do Meat:
- Info$_{Meat}$(S) = 4/9·(-2/4log$_2$2/4 - 2/4·log$_2$2/4 - 0·log$_2$0/4) + 5/9·(-0/5·log$_2$0/5 - 2/5·log$_2$2/5 - 3/5·log$_2$3/5) = .98
  - Information Gain is 1.53 - .98 = .55

Meat N,Y	Crust D,S,T	Veg N,Y	Quality B,G,Gr
Y	Thin	N	Great
N	Deep	N	Bad
N	Stuffed	Y	Good
Y	Stuffed	Y	Great
Y	Deep	N	Good
Y	Deep	Y	Great
N	Thin	Y	Good
Y	Deep	N	Good
N	Thin	N	Bad

Challenge Question

$$Info(S) = -\sum_{i=1}^{|C|} p_i log_2 p_i$$

Where S is the remaining training set at the node
$|A|$ is the number of attribute values for the feature
$|C|$ is the number of output classes for the task

$$Info_A(S) = \sum_{j=1}^{|A|} \frac{|S_j|}{|S|} Info(S_j) = \sum_{j=1}^{|A|} \frac{|S_j|}{|S|} \cdot -\sum_{i=1}^{|C|} p_i log_2 p_i$$

- What is the information for crust $\text{Info}_{\text{Crust}}(S)$:
  - A. .98
  - B. 1.35
  - C. .12
  - D. 1.41
  - E. None of the Above
- Is it a better attribute to split on than Meat?

Meat N,Y	Crust D,S,T	Veg N,Y	Quality B,G,Gr
Y	Thin	N	Great
N	Deep	N	Bad
N	Stuffed	Y	Good
Y	Stuffed	Y	Great
Y	Deep	N	Good
Y	Deep	Y	Great
N	Thin	Y	Good
Y	Deep	N	Good
N	Thin	N	Bad

Decision Tree Example

$$Info(S) = -\sum_{i=1}^{|C|} p_i log_2 p_i$$

$$Info_A(S) = \sum_{j=1}^{|A|} \frac{|S_j|}{|S|} Info(S_j) = \sum_{j=1}^{|A|} \frac{|S_j|}{|S|} \cdot -\sum_{i=1}^{|C|} p_i log_2 p_i$$

- $\text{Info}_{\text{Meat}}(S) = 4/9\cdot(-2/4\log_2 2/4 - 2/4\cdot\log_2 2/4 - 0\cdot\log_2 0/4) +$
  $5/9\cdot(-0/5\cdot\log_2 0/5 - 2/5\cdot\log_2 2/5 - 3/5\cdot\log_2 3/5) = .98$
- $\text{Info}_{\text{Crust}}(S) = 4/9\cdot(-1/4\log_2 1/4 - 2/4\cdot\log_2 2/4 - 1/4\cdot\log_2 1/4) +$
  $2/9\cdot(-0/2\cdot\log_2 0/2 - 1/2\cdot\log_2 1/2 - 1/2\cdot\log_2 1/2) +$
  $3/9\cdot(-1/3\cdot\log_2 1/3 - 1/3\cdot\log_2 1/3 - 1/3\cdot\log_2 1/3) = 1.41$
- Meat leaves less info (higher gain) and thus is the better of these two

Meat N,Y	Crust D,S,T	Veg N,Y	Quality B,G,Gr
Y	Thin	N	Great
N	Deep	N	Bad
N	Stuffed	Y	Good
Y	Stuffed	Y	Great
Y	Deep	N	Good
Y	Deep	Y	Great
N	Thin	Y	Good
Y	Deep	N	Good
N	Thin	N	Bad

Decision Tree Homework

$$Info(S) = -\sum_{i=1}^{|C|} p_i log_2 p_i$$

$$Info_A(S) = \sum_{j=1}^{|A|} \frac{|S_j|}{|S|} Info(S_j) = \sum_{j=1}^{|A|} \frac{|S_j|}{|S|} \cdot -\sum_{i=1}^{|C|} p_i log_2 p_i$$

- Finish the first level, find the best attribute and split
- Then find the best attribute for the left most node at the second level and split the node accordingly
  - Assume sub-nodes are sorted alphabetically left to right by attribute
  - Label any leaf nodes with their majority class
  - You could/should continue with the other nodes to get more practice

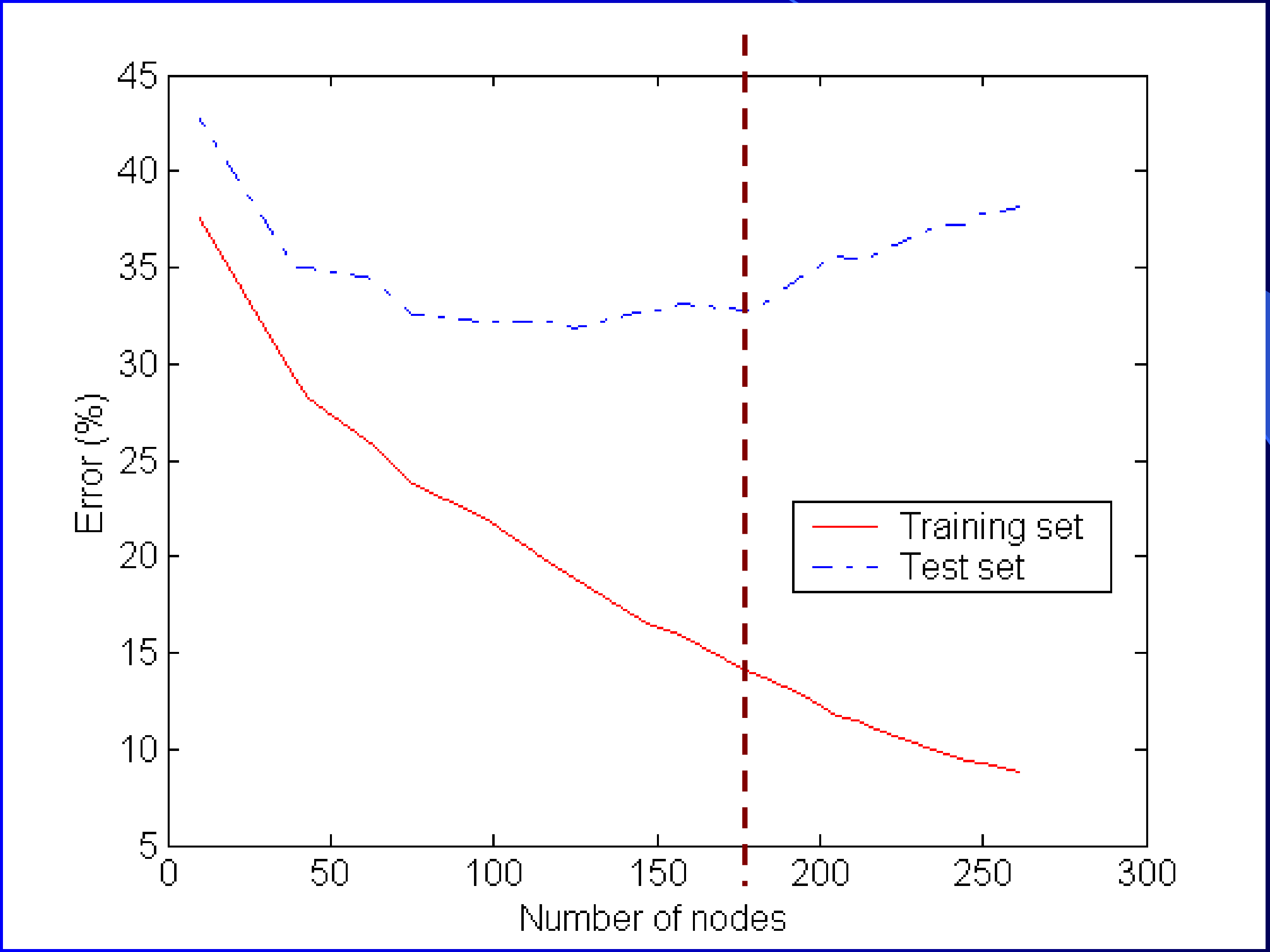
Notes

- Attributes which best discriminate between classes are chosen
- If the same ratios are found in a partitioned set, then gain is 0
- Complexity:
  - At each tree node with a set of instances the work is
    - O(|*Instances*| * |*remaining attributes*|), which is Polynomial
  - Total complexity is empirically polynomial
    - O(|*TrainingSet*| * |*attributes*| * |*nodes in the tree*|)
    - where the number of nodes is bound by the number of attributes and can be kept smaller through stopping criteria, etc.
- Keep in mind the number of instances
  - It may be O(|*Instances*| * |*attributes*|) but … Big Data

Overfitting and Underfitting

- Overfitting:
  - Given a model space *H*, a specific model $h \in H$ is said to overfit the training data if there exists some alternative model $h' \in H$, such that *h* has smaller error than *h'* over the training examples, but *h'* has smaller error than *h* over the entire distribution of instances
- Underfitting:
  - The model is too simple, so that both training and test errors are large

Detecting Overfitting



Overfitting in Decision Tree Learning

- Overfitting results in decision trees that are more complex than necessary
  - Tree growth went too far
  - Number of instances gets smaller as we build the tree (e.g., several leaves match a single example)

- Training error no longer provides a good estimate of how well the tree will perform on previously unseen records

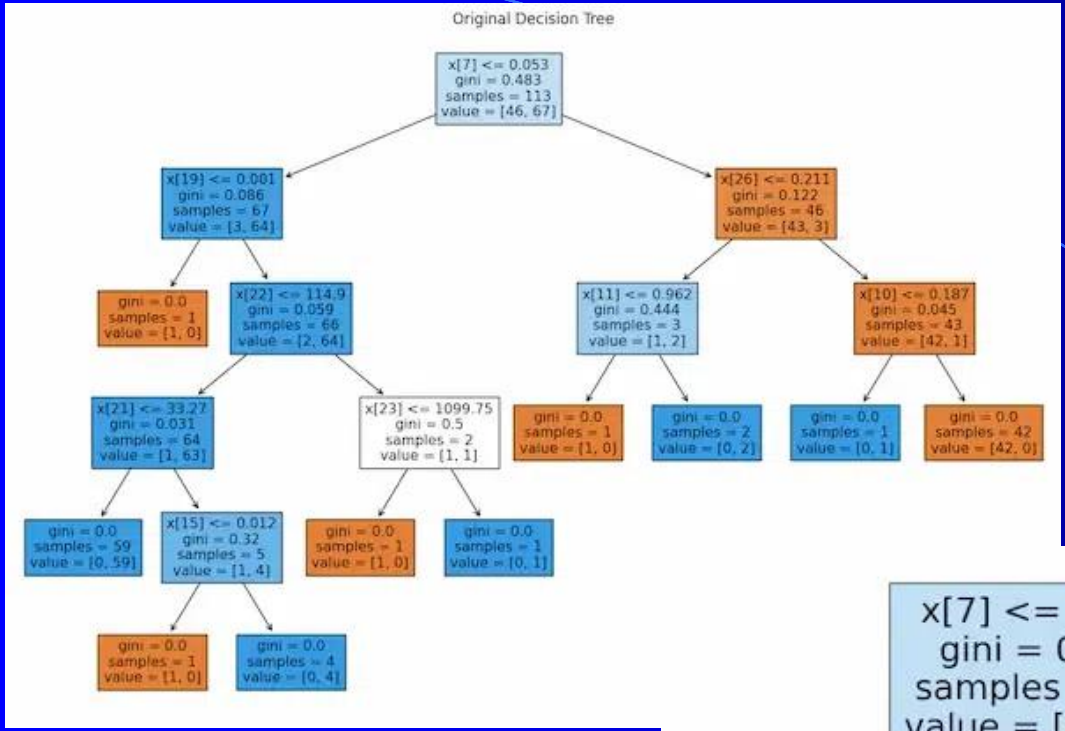
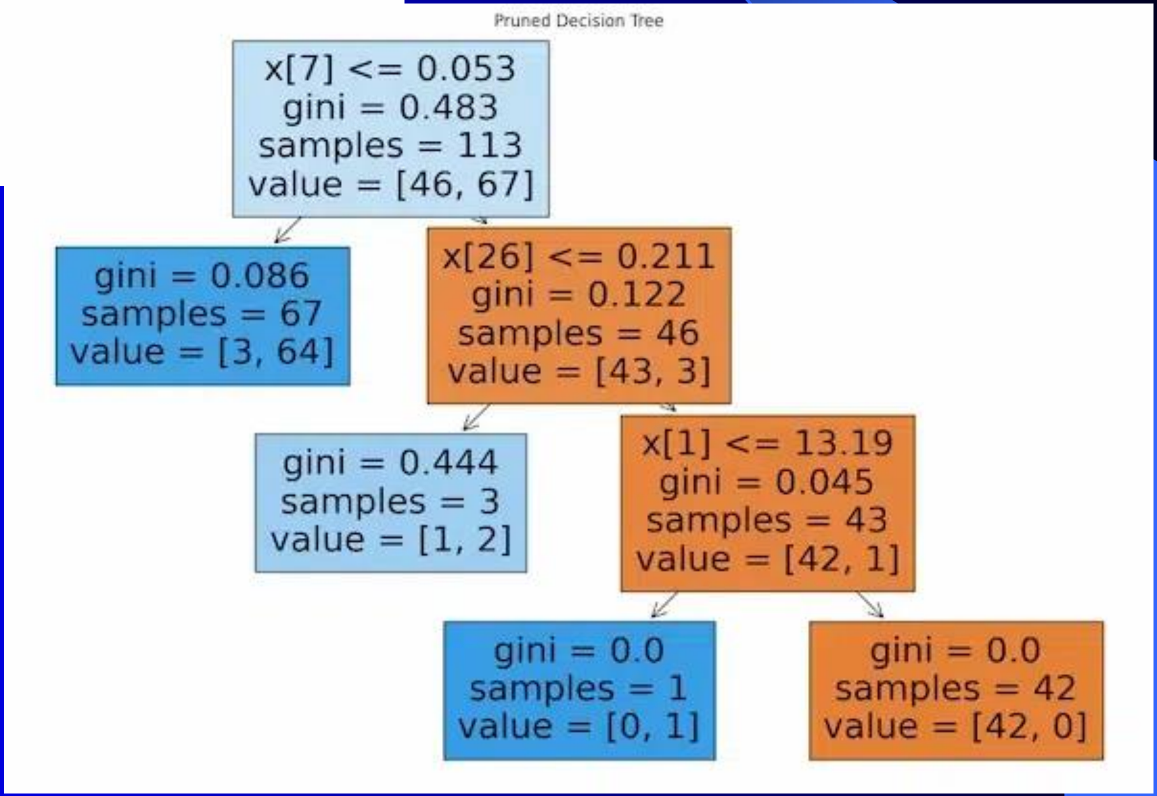
Avoiding Tree Overfitting – Pre-Pruning

- 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 GAIN)
  - Use a validation set and only add a new node if improvement (or no decrease) in accuracy on the validation set – checked independently at each branch of the tree using data set from parent
    - But shrinking data problem with decision trees

Reduced Error Pruning

- Pruning a full tree (one where all possible nodes have been added)
  - Prune any nodes which would not hurt accuracy
  - Could allow some higher order combinations that would have been missed with early stopping
  - Can simultaneously consider all nodes for pruning rather than just the current frontier

1. Train tree out fully (empty or consistent partitions or no more attributes)
2. For EACH non-leaf node, test accuracy on a validation set for a modified tree where the sub-trees of the node are removed and the node is assigned the majority class based on the instances it represents from the training set
3. Keep pruned tree which does best on the validation set and does at least as well as the current tree on the validation set
4. Repeat until no pruned tree does as well as the current tree


Original Decision Tree
x[7] <= 0.053 gini = 0.483 samples = 113 value = [46, 67]
x[19] <= 0.001 gini = 0.086 samples = 67 value = [3, 64]
x[26] <= 0.211 gini = 0.122 samples = 46 value = [43, 3]
gini = 0.0 samples = 1 value = [1, 0]
x[22] <= 114.9 gini = 0.059 samples = 66 value = [2, 64]
x[11] <= 0.962 gini = 0.444 samples = 3 value = [1, 2]
x[10] <= 0.187 gini = 0.045 samples = 43 value = [42, 1]
x[21] <= 33.27 gini = 0.031 samples = 64 value = [1, 63]
x[23] <= 1099.75 gini = 0.5 samples = 2 value = [1, 1]
gini = 0.0 samples = 1 value = [1, 0]
gini = 0.0 samples = 2 value = [0, 2]
gini = 0.0 samples = 1 value = [0, 1]
gini = 0.0 samples = 42 value = [42, 0]
gini = 0.0 samples = 59 value = [0, 59]
x[15] <= 0.012 gini = 0.32 samples = 5 value = [1, 4]
gini = 0.0 samples = 1 value = [1, 0]
gini = 0.0 samples = 1 value = [0, 1]
gini = 0.0 samples = 1 value = [1, 0]
gini = 0.0 samples = 4 value = [0, 4]

Pruned Decision Tree
x[7] <= 0.053 gini = 0.483 samples = 113 value = [46, 67]
gini = 0.086 samples = 67 value = [3, 64]
x[26] <= 0.211 gini = 0.122 samples = 46 value = [43, 3]
gini = 0.444 samples = 3 value = [1, 2]
x[1] <= 13.19 gini = 0.045 samples = 43 value = [42, 1]
gini = 0.0 samples = 1 value = [0, 1]
gini = 0.0 samples = 42 value = [42, 0]

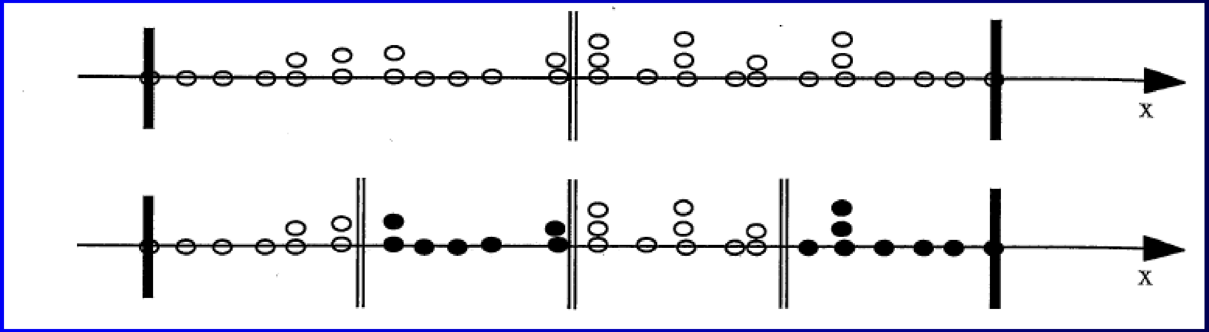
Missing Values

- Can use any of the methods we discussed previously –
  - new attribute value very natural and effective with typical nominal data

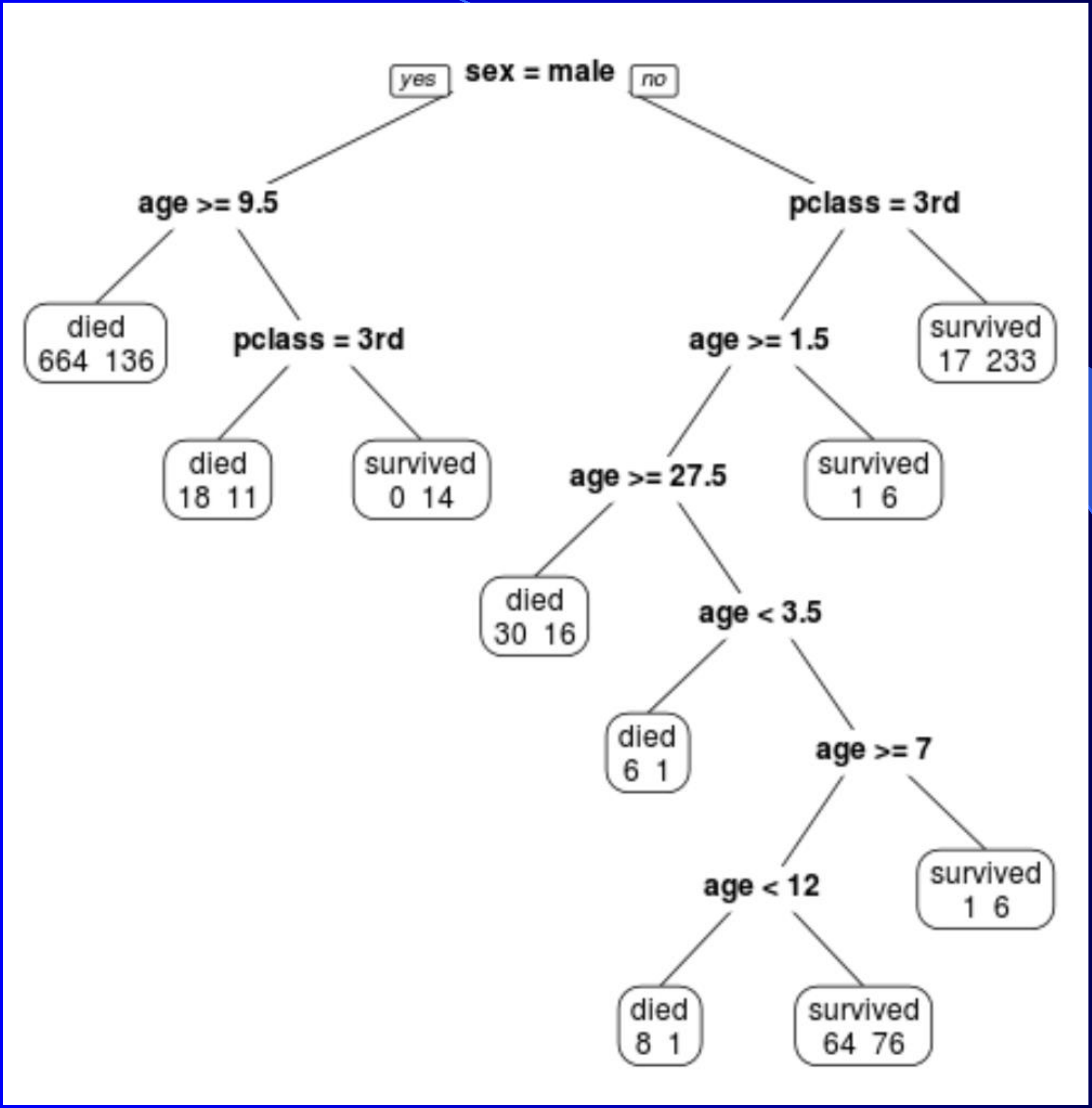
- Another approach, particular to decision trees:
  - When arriving at an attribute test for which the attribute is missing do the following:
  - Each branch has a probability of being taken based on what percentage of examples at that parent node have the branch's value for the missing attribute
  - Take <u>all</u> branches, but carry a weight representing that probability. These weights could be further modified (multiplied) by other missing attributes in the current example as they continue down the tree.
  - Thus, a single instance gets broken up and appropriately distributed down the tree but its total weight throughout the tree will always sum to 1
- Results in multiple active leaf nodes. For execution, set output as leaf with highest weight, or sum weights for each output class, and output the class with the largest sum, (or output the class confidence).
- During learning, scale instance contribution by instance weights.
- This approach could also be used for labeled probabilistic inputs with subsequent probabilities tied to outputs

Real Valued Features

- C4.5: Continuous data is handled by testing all *n*-1 possible binary thresholds for each continuous feature to see which gives best information gain. The split point with highest gain gives the score for that feature which then competes with all other features.
  - More efficient to just test thresholds where there is a change of classification.
  - Is binary split sufficient? Attribute may need to be split again lower in the tree, no longer have a strict depth bound



Titanic Survival Dataset



DT Interpretability

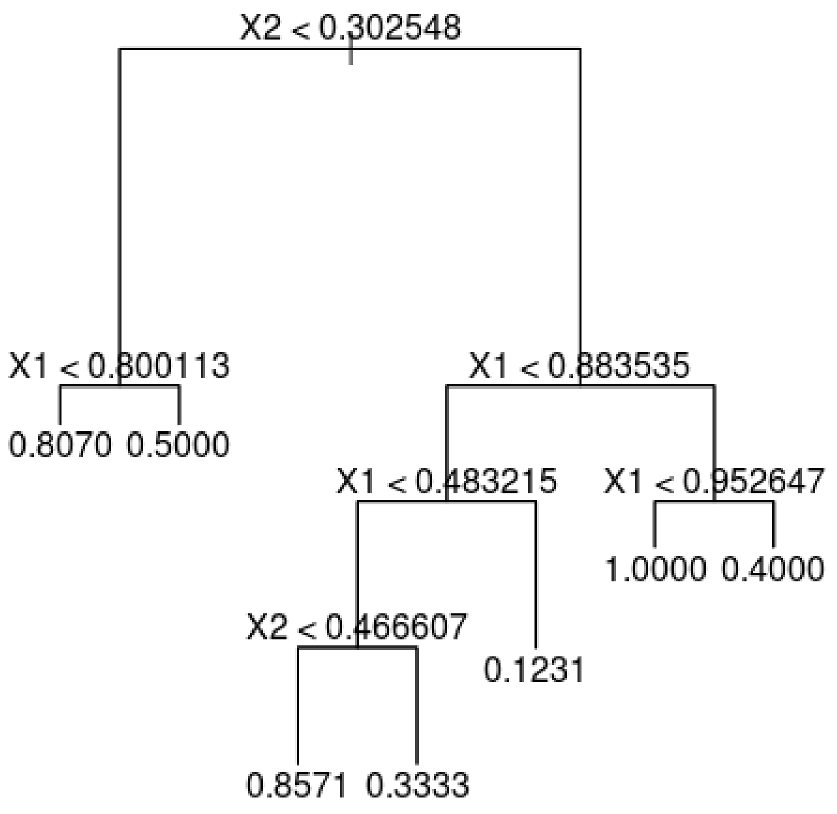
- Intelligibility of DT – When trees get large, intelligibility drops off
- C4.5 rules - transforms tree into prioritized rule list with default (most common output for examples not covered by rules).
  - It does simplification of superfluous attributes by greedy elimination strategy (based on statistical error confidence as in error pruning). Prunes less productive rules within rule classes
- How critical is intelligibility in general?
  - Will truly hard problems have a simple explanation?

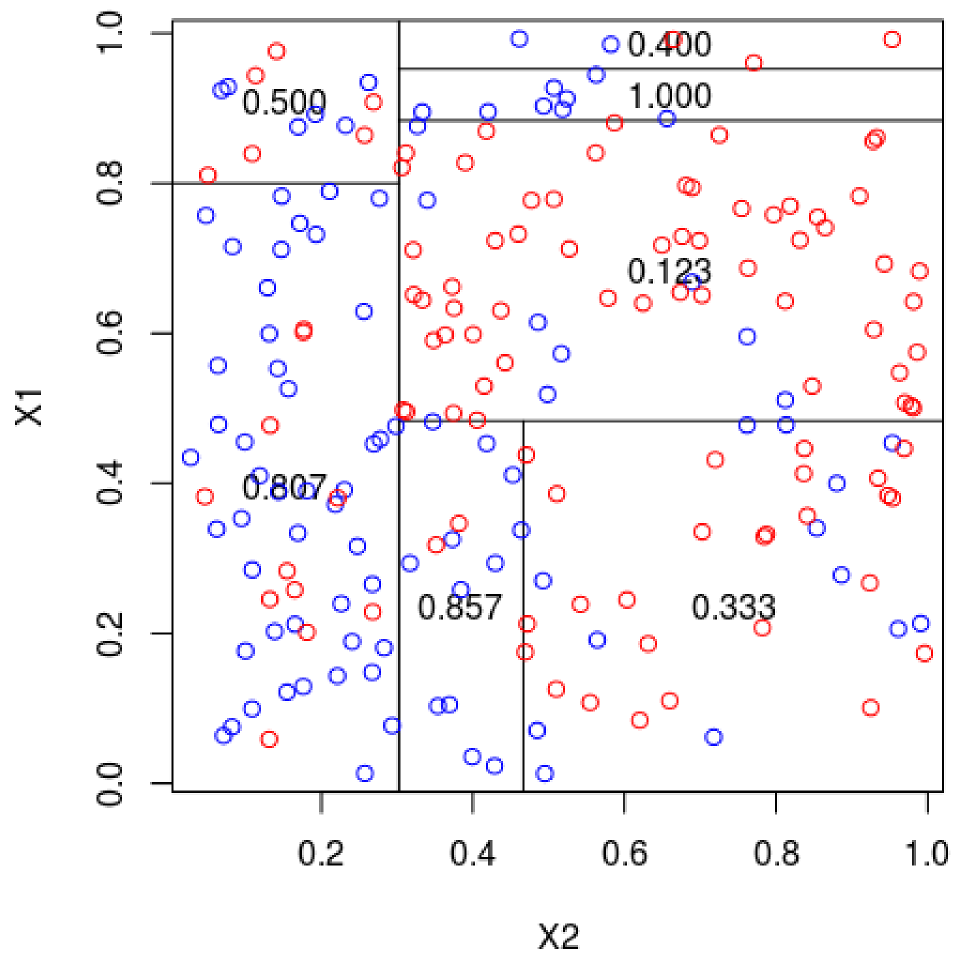
Information gain favors attributes with many attribute values

- If *A* has random values (SS#), but ends up with only 1 example in each partition, it would have maximum information gain, though a terrible choice.
- Occam's razor would suggest seeking trees with less overall nodes. Thus, attributes with less possible values might be given some kind of preference.
- Binary attributes (CART) are one solution, but lead to deeper trees, and somewhat higher complexity in possible ways of splitting attributes
- Can use a penalty for attributes with many values such as Laplacian: $(n_c+1)/(n+|C|)$), though real issue is splits with little data
- Gain Ratio is the approach used in original ID3/C4.5

Regression

- DecisionTreeRegressor
- Regression Tree – The output value for a leaf is just the average of the outputs of the instances represented by that leaf
  - Could adjust for outliers, etc.
- For regression training the score for a node is not GINI/Info impurity, but is the SSE of instances represented by the node
- The feature score for a potential split is the weighted sum of the two child nodes scores (SSE)
- Then, just like with classification, we choose the lowest score amongst all possible feature splits
- As long as there is significant variance in node examples, splitting will continue


X2 < 0.302548
X1 < 0.800113
X1 < 0.883535
0.8070 0.5000
X1 < 0.483215
X1 < 0.952647
1.0000 0.4000
X2 < 0.466607
0.1231
0.8571 0.3333


X1
X2
0.0
0.2
0.4
0.6
0.8
1.0
0.2
0.4
0.6
0.8
1.0
0.500
0.400
1.000
0.123
0.807
0.857
0.333

Decision Trees - Conclusion

- Good Empirical Results
- Comparable application robustness and accuracy with MLPs
- Fast learning since no iterations
- MLPs can be more natural with continuous inputs, while DT natural with nominal inputs
- One of the most used and well known of current symbolic systems
- Can be used as a feature filter for other algorithms – Attributes higher in the tree are best, those rarely used can be dropped

Decision Tree Lab

- Nominals – SK CART only accepts numeric features - Fits CART fine since that is how CART thinks of nominal features anyways, breaking them into separate one-hot features for each possible feature value
- So a nominal attribute Color with 3 attribute values (Red, Green, Blue), would be represented as 3 one-hot features
  - Is-Red, Is-Green, Is-Blue
  - Binary features can just be represented as 0/1
- Note that Color could appear multiple times in a branch unlike in C4/5. A not Is-Red branch could later consider Is-Blue or Is-Green.