

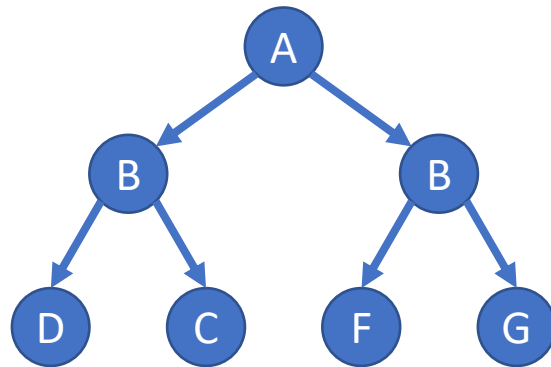
Decision Trees

Dr Mohamed Bennasar

Decision Trees

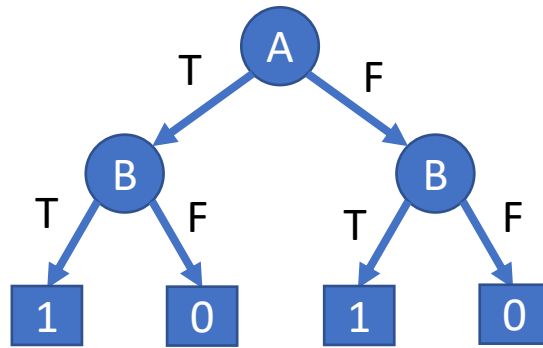
- Patterns in the data can be captured using a set of IF-THEN Rules.
- The set of rules can create **branching** for making a decision, calculations, or prediction.

Representing the tree as an array



A	B	C	D	E	F	G
X	A	B	B	A	E	E

Logic Proposition



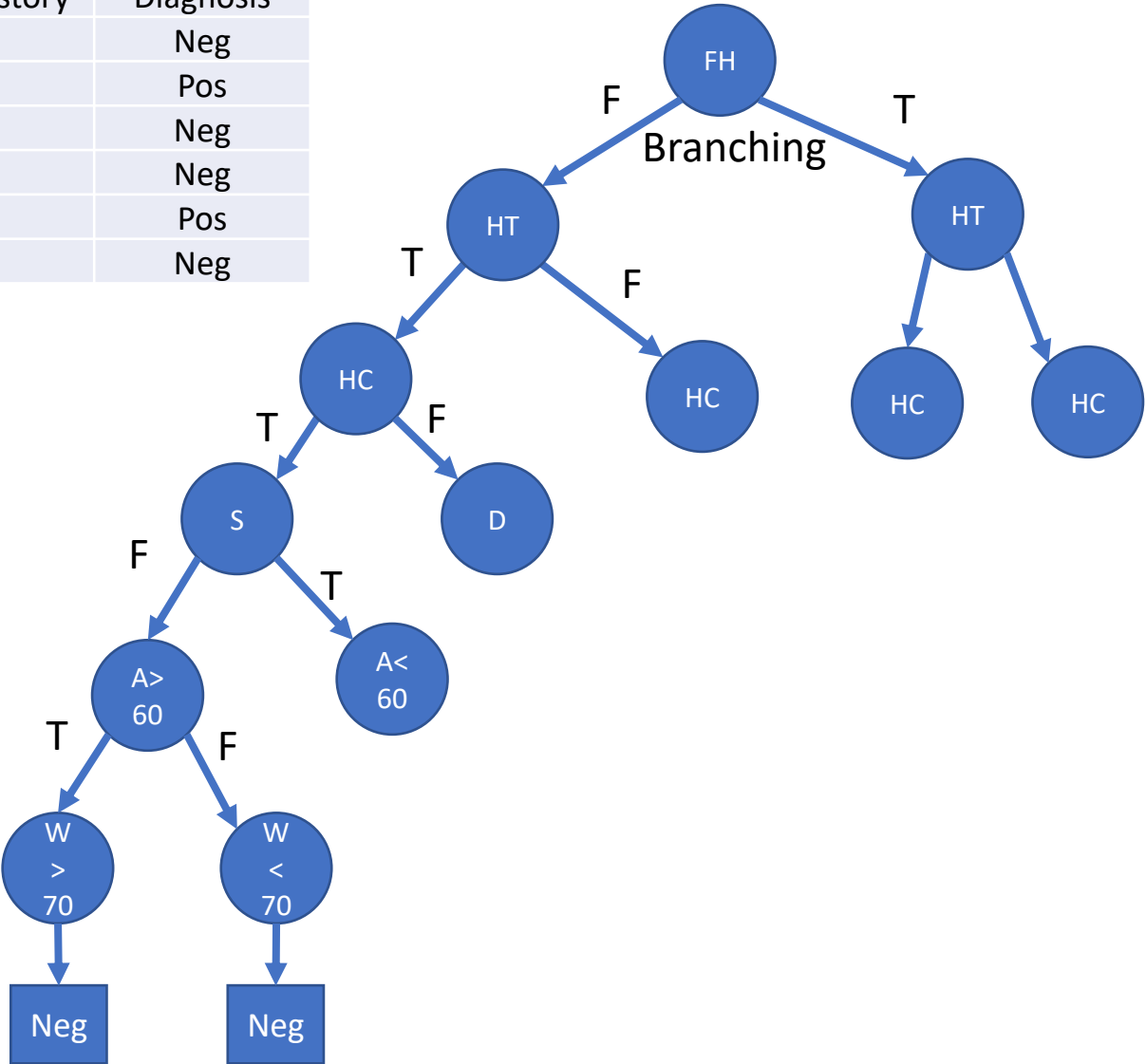
=

IF (A&B) OR (not(A) & not(B)) then 1 ELSE 0

Heart Disease Diagnosis

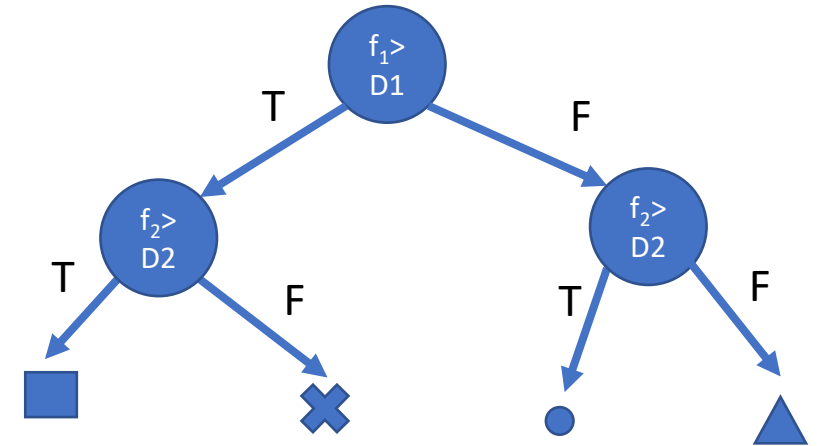
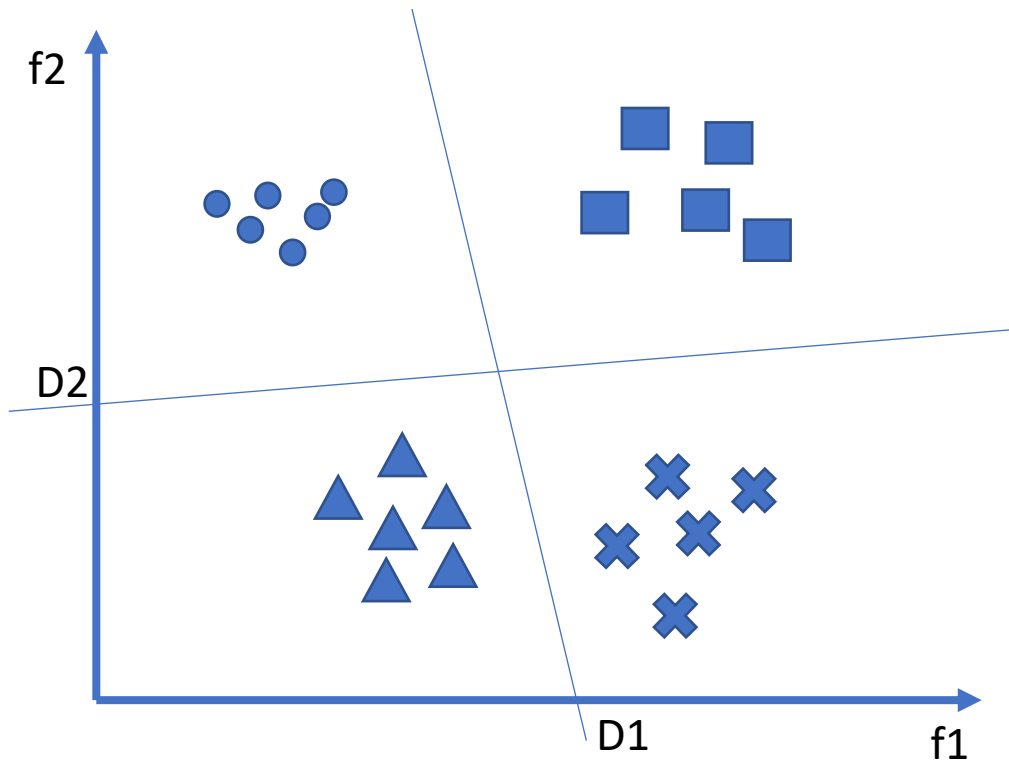
Age	Weight	Hypertensive	Hypercholesterolemia	Smoking	Family History	Diagnosis
77.00	73.00	1.00	1.00	0.00	0.00	Neg
85.00	70.00	1.00	1.00	1.00	1.00	Pos
67.00	63.00	1.00	1.00	0.00	0.00	Neg
51.00	89.00	1.00	1.00	0.00	1.00	Neg
72.00	56.00	1.00	1.00	1.00	1.00	Pos
67.00	76.00	0.00	1.00	0.00	0.00	Neg

Age= A
Weight= W
Hypertensive=HT
Hypercholesterolemia=HC
Smoking= S
Family History= FH



Decision

How to construct the trees



Decision tree consists of :

- 1- Nodes to represent each feature, at each node the value of the feature is tested using logic relations.
- 2- Branches that connect the nodes. The branching has to be binary. Therefore, the two branches can be True/ False, Yes/ No, or 1/0.
- 3- The last nodes at the bottom of the tree are called leaves that classify the data points.

When to use decision tree.

- When the class label is discrete.
- When we do not have large dataset.
- When data is noisy.
- Class are well separated.

Why Decision Tree?

- Relatively fast compared to other classification models
- Obtain similar and sometimes better accuracy compared to other models
- Simple and easy to understand
- Can be converted into simple and easy to understand classification rules

How is the decision tree built?

How to order the features to use them to generate the most compact branching?
In another word, what is the most important feature to use at the root node?

- Let us assume that we have only binary data?
- Let us assume that we divided the data into training (S) and testing samples (T).
- S includes positive data points S^+ , and negative data points S^- .

Entropy

- Entropy measures the amount of uncertainty about a variable. In other words, it is the amount of information that we need to predict the value of the variable.

$$H(F) = - \sum_{i=1}^N p(S_i) \log_2(p(S_i))$$

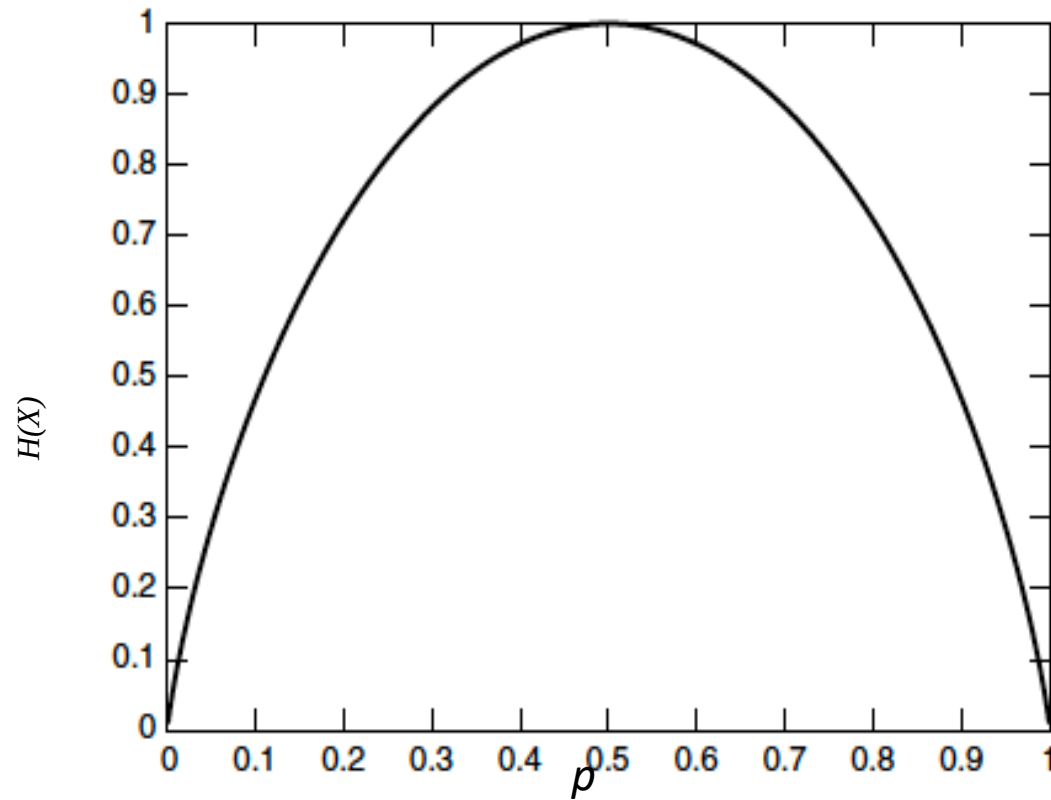
$p(S_i)$ is the probability mass function.

$$p(S_i) = \frac{\text{number of data points with value } S_i}{\text{the total number of data points (Cardinality of } (S))}$$

The logarithms are to the base 2.

Let us compare logarithms to the base 10, and logarithms to the base 2 :

$10^0 = 1$	$\log_{10} 1 = 0$	$2^0 = 1$	$\log_2 1 = 0$
$10^1 = 10$	$\log_{10} 10 = 1$	$2^1 = 2$	$\log_2 2 = 1$
$10^2 = 100$	$\log_{10} 100 = 2$	$2^2 = 4$	$\log_2 4 = 2$



- The value of $H(X)$ is 0 if the probability is 1 or 0, in such case there is no any uncertainty about the variable ($H(X)= 0$).
- The maximum value is 1 when the probability is 0.5 which the highest uncertainty we can have about predicting the value of the variable.

Example

- Assume that we have a small dataset about customer review of a service. Let us estimate the entropy of the variables in the following data:

X (age group)	Y (review)
18–30	Like
>50	Dislike
31–50	Like
18–30	Dislike
18–30	Dislike
>50	Like
18–30	Like
31–50	Dislike

$$(18 - 30) = \frac{4}{8} = \frac{1}{2}$$

$$P(31 - 50) = \frac{2}{8} = \frac{1}{4}$$

$$P(> 50) = \frac{2}{8} = \frac{1}{4}$$

The entropy of X is $H(X) = -\frac{1}{2}\log\left(\frac{1}{2}\right) - \frac{1}{4}\log\left(\frac{1}{4}\right) - \frac{1}{4}\log\left(\frac{1}{4}\right) = 1.5$

$$P(\text{Like}) = \frac{4}{8} = \frac{1}{2}$$

$$P(\text{Dislike}) = \frac{4}{8} = \frac{1}{2}$$

The entropy of Y is $H(Y) = -\frac{1}{2}\log\left(\frac{1}{2}\right) - \frac{1}{2}\log\left(\frac{1}{2}\right) = 1$

Another Example

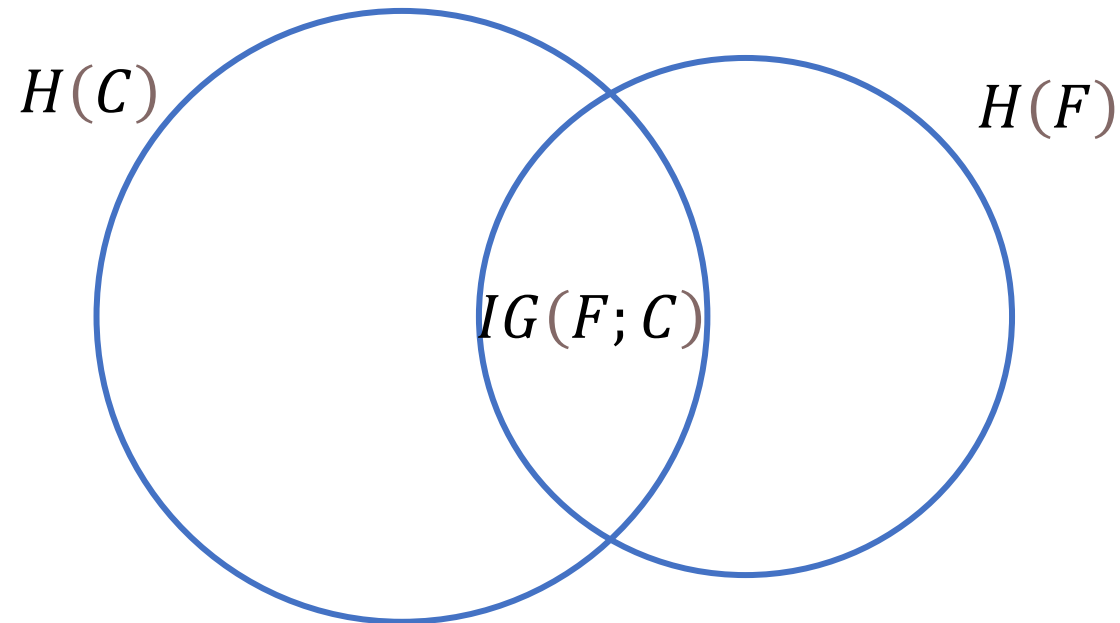
- To explain more what entropy means, assume there are 8 horses competing in a horse race. The probabilities of winning of each one are $\left(\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}, \frac{1}{64}, \frac{1}{64}, \frac{1}{64}, \frac{1}{64}\right)$.
- According to the definition of the entropy, the amount of information that we need to know which horse may win is $H(x)$.
- $H(X) = -\frac{1}{2} \log \frac{1}{2} - \frac{1}{4} \log \frac{1}{4} - \frac{1}{8} \log \frac{1}{8} - \frac{1}{16} \log \frac{1}{16} - 4 \frac{1}{64} \log \frac{1}{64} =$
2 bits

Constructing the tree

- How much information that can each feature tell us about the class label(about the prediction)? In another work, how important each feature for predicting the class label.
- We need to quantify the significance of each feature.
- Information gain in the amount of reduction in the entropy of the class label if we know the feature F .

Information Gain (IG)

The aim is to reduce the amount of entropy of the class $H(C)$



$$IG(F; C) = H(C) + H(F) - H(C, F)$$

$$IG(F; C) = H(C) - H(C|F)$$

Example

Let us assume that we have a small dataset that represents the weather conditions that determine whether a tennis game is played or not.

The data consists of 4 features that are categorical variables.

Outlook	Temperature	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

- There are 14 data points.
- Outlook can take one of the three possible value:
 - Sunny, overcast or rainy.
- Temperature can take one of the three possible value:
 - Hot, Mild or Cool.
- Humidity can take one of the two possible values:
 - High or Normal
- Windy is a binary feature i.e. it can take one of the two possible values:
 - True when it is windy or false when it is not
- There are 36 possible combinations, less that the have are in the data.

Construct a Decision Tree Classifier

The aim: it to construct a decision tree trained model that is able to predict whether a game is played or not:

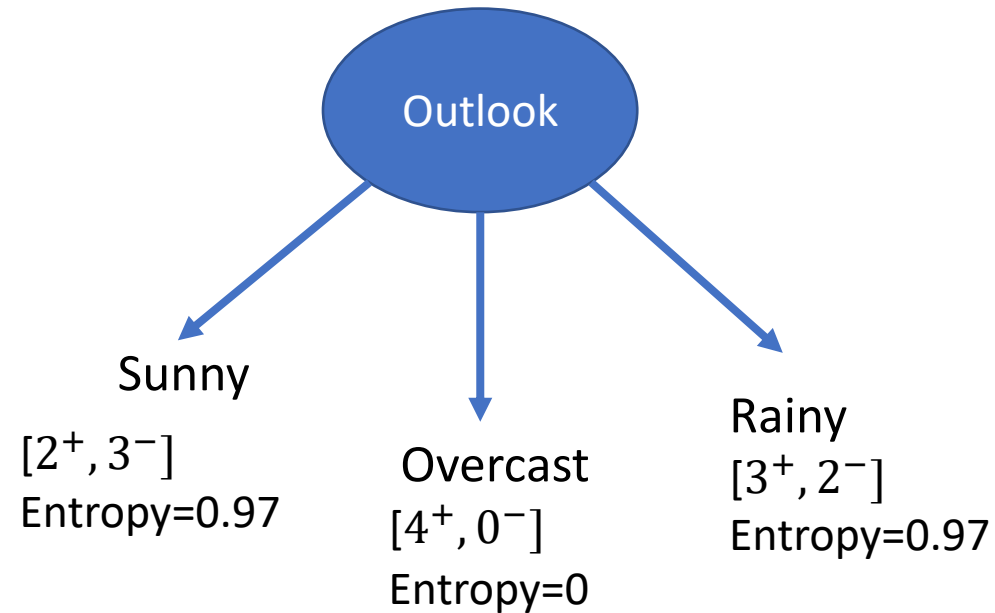
- Define the nodes, and the branches.
- Identify the root node, and the order of the child nodes .
- If an attribute has more than two possible values, we still take binary decision.

Identifying the Root Node

- Calculating the entropy of the class label: The number of scenarios where the decision is yes is 9 and the number of times the decision is no is 5.
- The probability mass function of Yes= $p_{\text{yes}} = 9/14 = 0.643$.
- The probability mass function of No= $p_{\text{no}} = 5/14 = 0.357$.
- $p_{\text{yes}} + p_{\text{no}} = 1$
- $H(c) = -(0.643 \log(0.643)) - (0.357 \log(0.357)) = 0.41 + 0.53 = 0.94$ bit

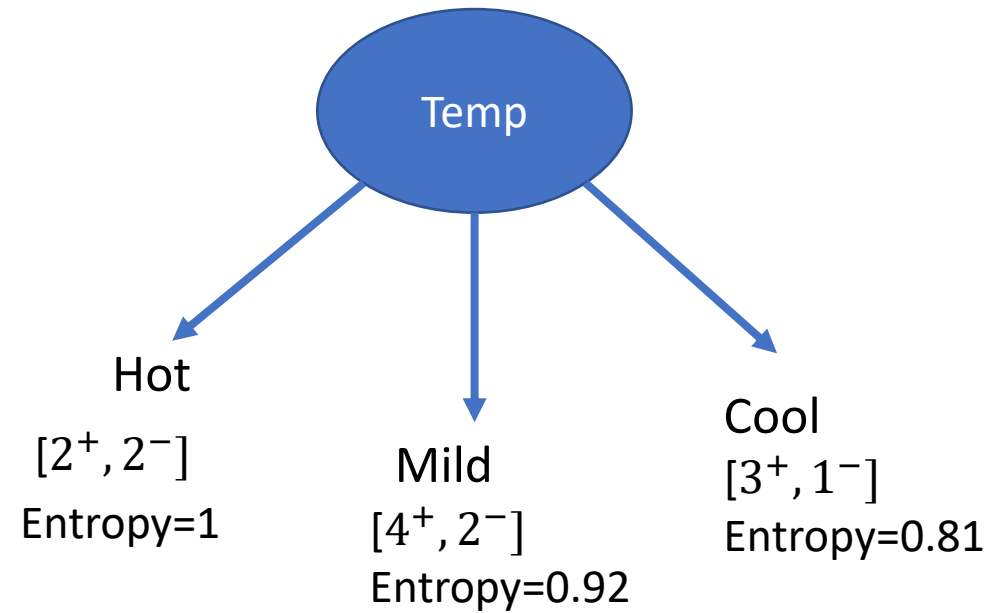
Identifying the Root Node

- Calculating the IG between Decision and Outlook feature:



$$IG(F; C) = H(C) - H(C|F)$$

$$IG = 0.94 - ((5/14) 0.97 + (4/14) 0 + (5/14) 0.97) = 0.246$$

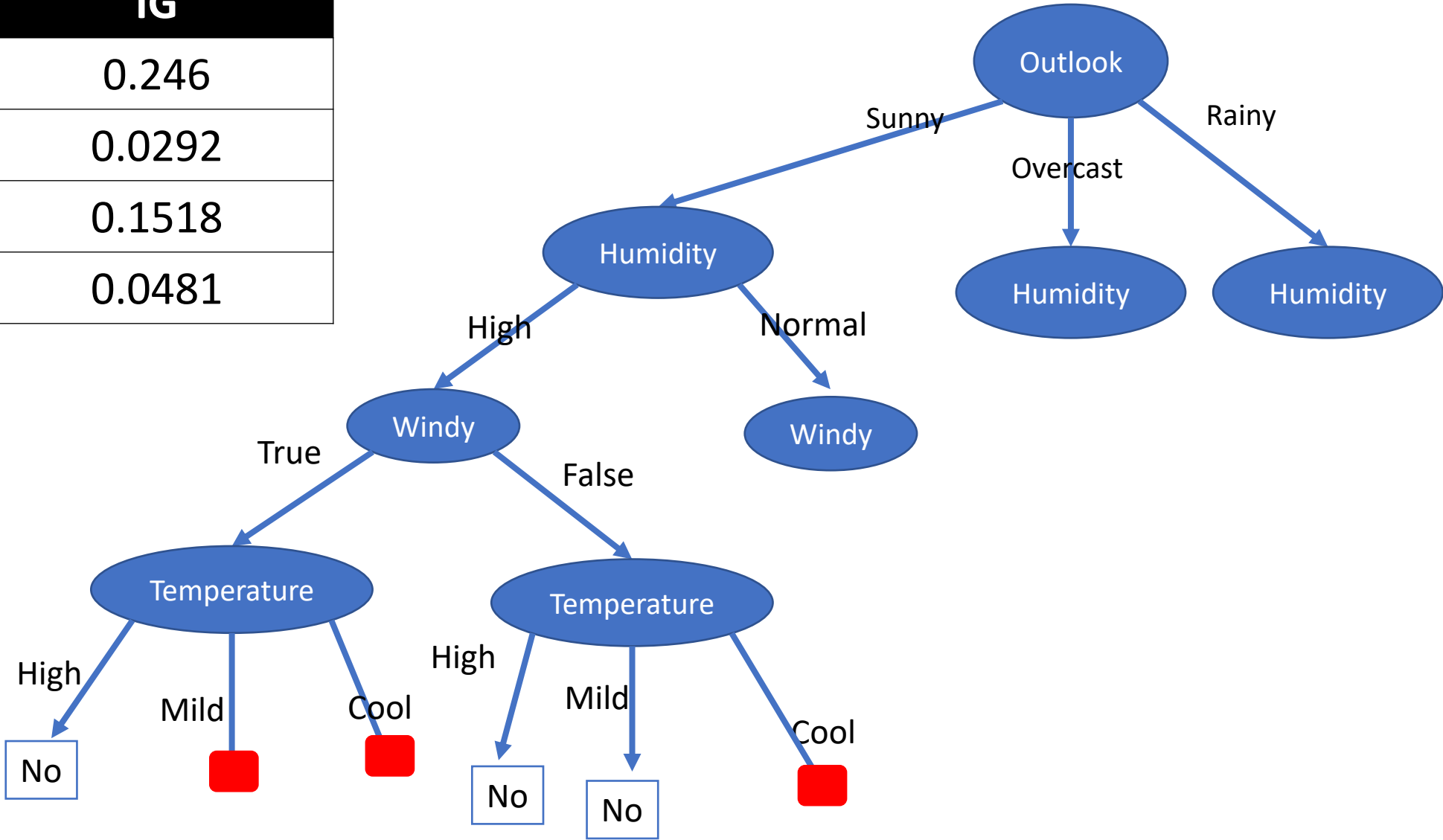


$$IG = 0.94 - \left(\left(\frac{4}{14} \right) 1 + \left(\frac{6}{14} \right) 0.92 + \left(\frac{4}{14} \right) 0.81 \right) = 0.029$$

$IG(\text{Outlook}) > IG(\text{Temperature})$

Which feature is better to start with?

Feature	IG
Outlook	0.246
Temperature	0.0292
Humidity	0.1518
Windy	0.0481



Outlook	Temperature	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

Decision Trees Algorithms

- ID3 (Iterative Dichotomise V3)
- C4.5 and C5.0

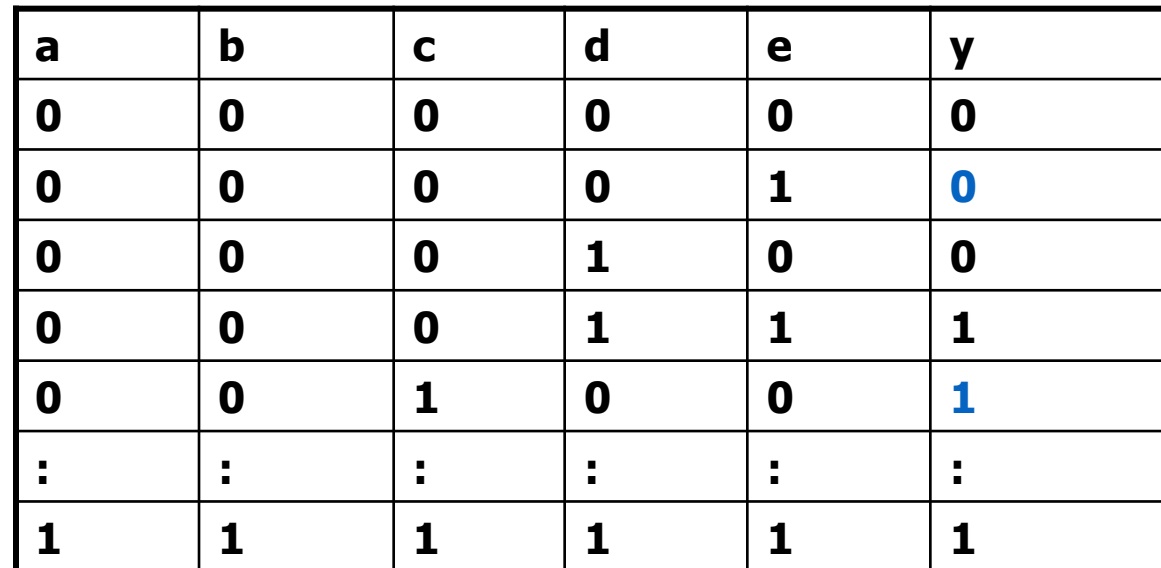
Another Example

- We'll create a training dataset

Five features, all the possible 32 combinations are generated

Class label (y) = copy of feature e, except a random 25% of the records in y are set to the opposite of e

32 data points



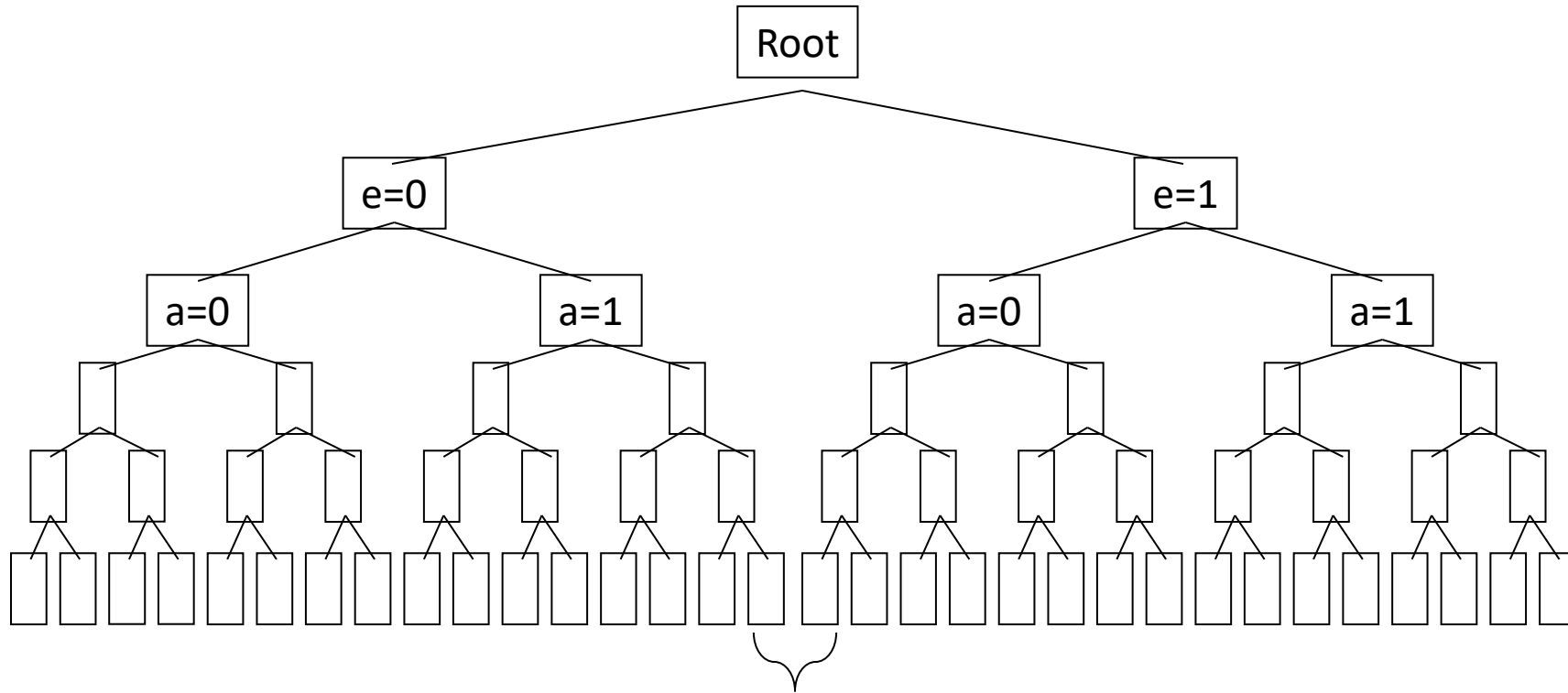
a	b	c	d	e	y
0	0	0	0	0	0
0	0	0	0	1	0
0	0	0	1	0	0
0	0	0	1	1	1
0	0	1	0	0	1
:	:	:	:	:	:
1	1	1	1	1	1

In this example

- Let us assume that some of the values in the class label (y) are corrupted (noise or mislabeled)
- Suppose that we have also a test set are generated according to the same method.
- The test set is identical, except that some of values in class label (y) will be different.
- Assume that the corrupted values in training set class label (y) will be corrected in the testing set.
- Assume also that some of uncorrupted values in training set class label (y) values will be corrupted in the testing set.

Building a Decision Tree using the Training Data

- Suppose we build a full tree



25% of these leaf node labels will be wrong

Training Error

Every leaf node contains exactly one record and so...

- **We would have a training set error of zero**

Testing Error

	1/4 of the tree nodes are corrupted	3/4 are fine
1/4 of the test set records are corrupted	1/16 of the test set will be correctly predicted for the wrong reasons	3/16 of the test set will be wrongly predicted because the test record is corrupted
3/4 are fine	3/16 of the test predictions will be wrong because the tree node is corrupted	9/16 of the test predictions will be fine

The testing error is expected to be about $3/8$

What has this example shown us?

- It shows that the constructed tree is not performing well on the the unseen data (testing set). However, the The performance was perfect on the training data.

Suppose we Trim the irrelevant features in the data

- Let's not look at the irrelevant bits

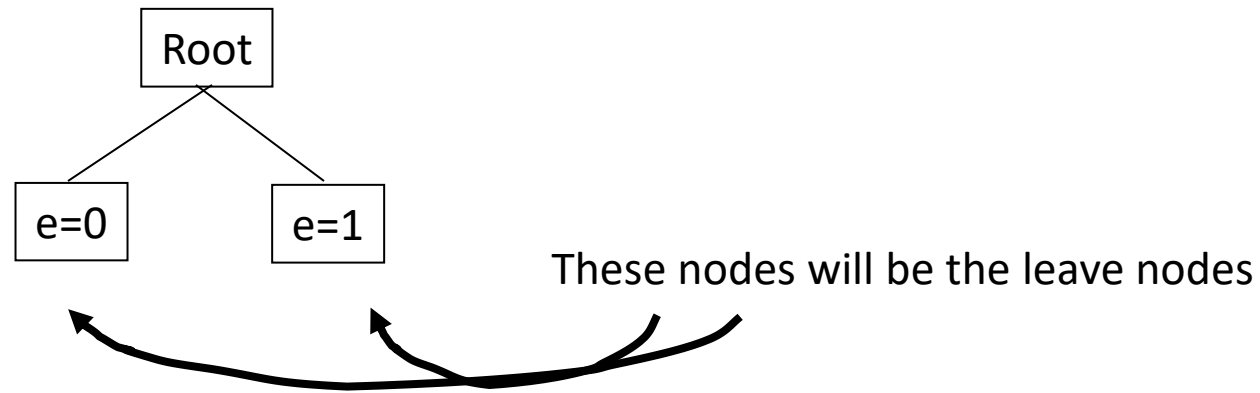
These features are included

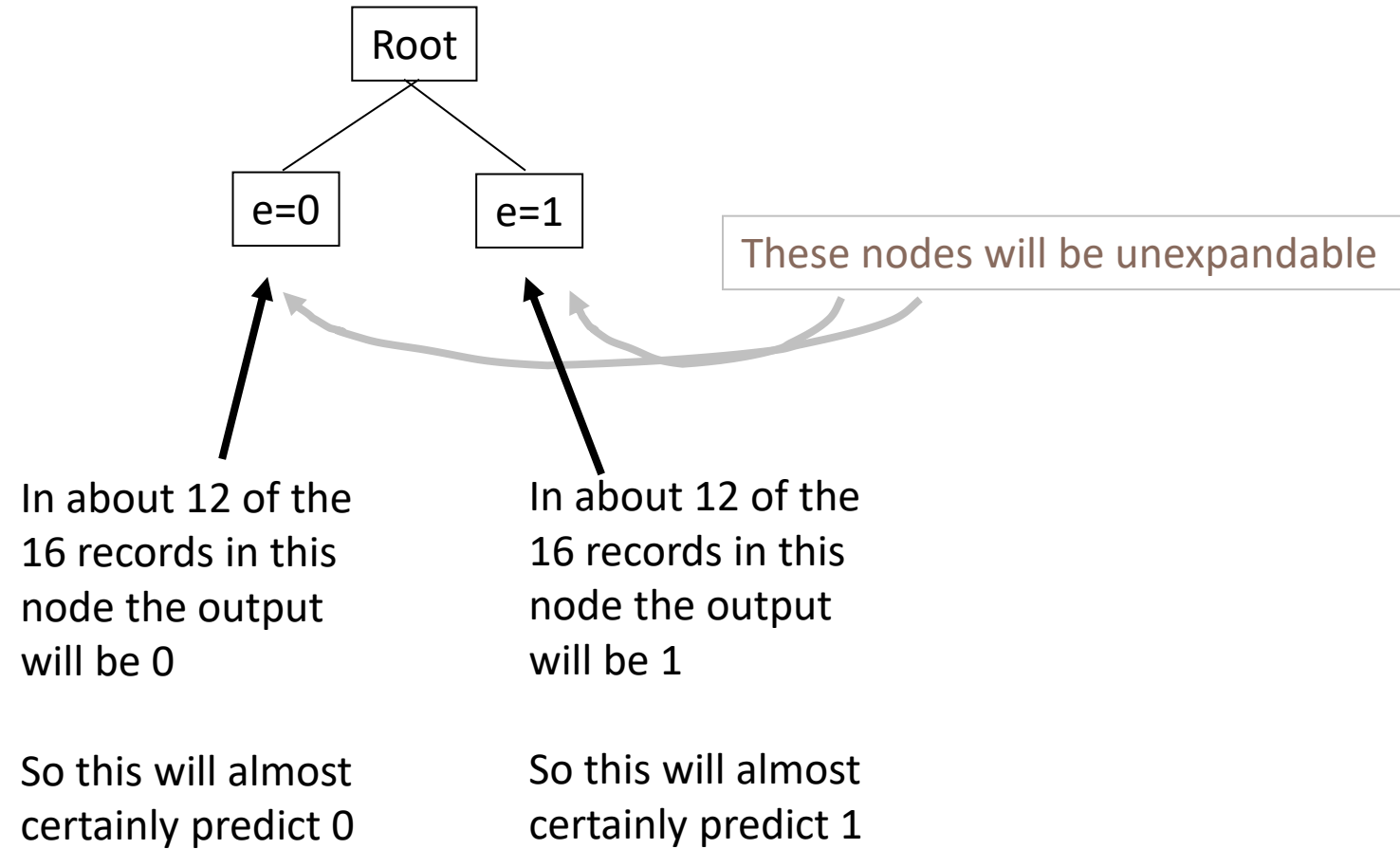
Output y is a copy of e, except a random 25% of the values of y which are set to the opposite of e

32 records	a	b	c	d	e	y
	0	0	0	0	0	0
	0	0	0	0	1	0
	0	0	0	1	0	0
	0	0	0	1	1	1
	0	0	1	0	0	1
	:	:	:	:	:	:
	1	1	1	1	1	1

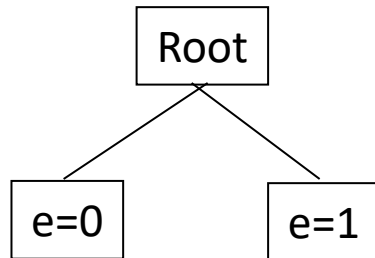
What decision tree would we learn now?

Without including the irrelevant bits, we will have this tree:





The performance of the new tree

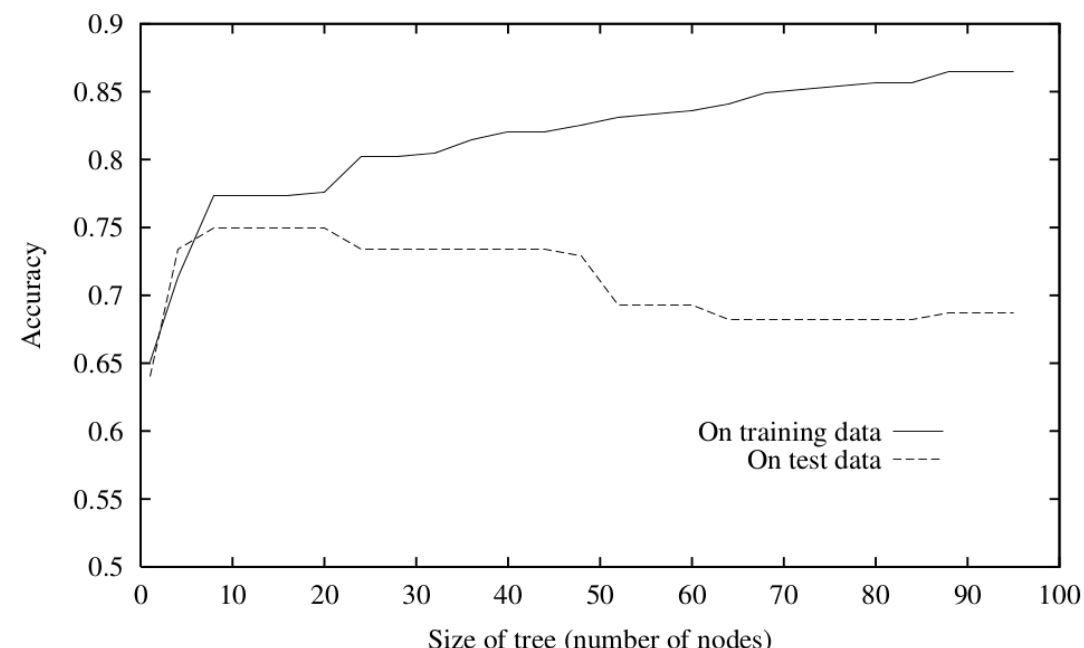
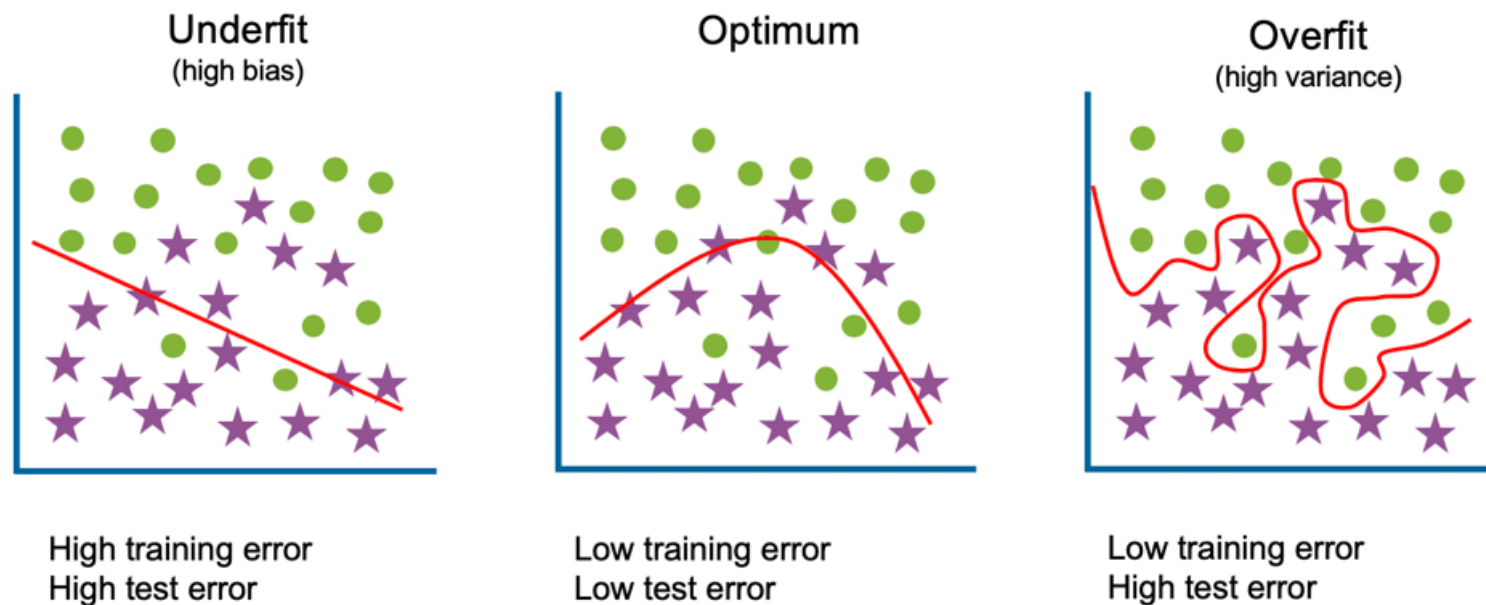


	almost certainly none of the tree nodes are corrupted	almost certainly all are fine
1/4 of the test set records are corrupted	n/a	1/4 of the test set will be wrongly predicted because the test record is corrupted
3/4 are fine	n/a	3/4 of the test predictions will be fine

The error rate of this tree is expected to be only 1/4 of the test set predictions

What is overfitting?

- Overfitting is a concept in data science, which occurs when a statistical model fits exactly against its training data.
- When this happens, the algorithm unfortunately cannot perform accurately against unseen data, defeating its purpose.
- Generalisation of a model to new data is ultimately what allows us to use machine learning algorithms every day to make predictions and classify data.



Sources of Overfitting

- Noise
- Small number of examples are associated with each leaf node.

Dealing with overfitting

- Stop growing the tree
- Post prune the tree after overfitting
- How do we determine correct final tree (size)
 - Validation set (1/3rd)
 - Statistical (chi-square) test to determine whether to grow the tree
 - Minimize MDL (measure of complexity)
 - $\text{size}(\text{tree}) + \text{size}(\text{misclassifications})$

Avoiding overfitting

- Two approaches:
 - Stop growing the tree when data split is not statistically significant.
 - Grew tree fully, then perform post-prune.
- What is the correct tree size?
 - Balanced tree are preferred.
 - Trees that grow on one side usually tend to be shallow.
 - Too many splits just to reach very few datapoints is preferred.
- How do we determine correct final tree (size)
 - Validation set (1/3rd)
 - Statistical (chi-square) test to determine whether to grow the tree
 - Minimize MDL (measure of complexity)
 - $\text{size}(\text{tree}) + \text{size}(\text{misclassifications})$

Using Chi-squared to avoid overfitting

- Build the full decision tree as before.
- But when you can grow it no more, start to prune:
 - Beginning at the bottom of the tree, delete splits in which $p_{chance} > \text{MaxPchance}$.
 - Continue working your way up until there are no more prunable nodes.

MaxPchance is a magic parameter you must specify to the decision tree, indicating your willingness to risk fitting noise.

MaxPchance

- **Good news:** The decision tree can automatically adjust its pruning decisions according to the amount of apparent noise and data.
- **Bad news:** The user must come up with a good value of MaxPchance. (we may use 0.05, which is his favorite value for any magic parameter).
- **Good news:** But with extra work, the best MaxPchance value can be estimated automatically by cross-validation.

Reduced error pruning

- Do until further pruning is harmful:
 - Remove subtree at node and replace with leaf.
 - Assign most common class at node to leaf.
 - Remove the node that most improves the validation accuracy.
 - Only select node for removal if error $\leq$ error of original tree on validation set

Rule post pruning

- Convert tree to equivalent set of rules
- Prune each rule independently of others
 - Remove precondition and test the validation accuracy.
- Sort final rules into sequence by estimated accuracy and consider them in this sequence.
- Classify new instances using the sorted sequence of rules.

Why rules then pruning?

- Each path through a node produces a different rule, so you have many rules per node that can be pruned versus removing one node (and subtree)
- In rules, tests near the root do not mean more than tests near leaves
- Rules are often easier to read and understand.

Conclusions

- Decision trees are the one of the most popular Supervised ML algorithms:
 - Easy to understand
 - Easy to implement
 - Easy to use
 - Computationally cheap
 - Easy to interpret its decisions.
- It's possible to get in trouble with overfitting
- They do classification: predict a categorical output from categorical and/or real inputs.

What you should know

- What's a contingency table?
- What's information gain, and why we use it
- The recursive algorithm for building an unpruned decision tree
- What are training and test set errors
- Why test set errors can be bigger than training set
- Why pruning can reduce test set error
- How to exploit real-valued inputs

What we haven't discussed

- It's easy to have real-valued outputs too---these are called Regression Trees
- Bayesian Decision Trees can take a different approach to preventing overfitting
- Computational complexity.
- Alternatives to Information Gain for splitting nodes
- How to choose MaxPchance automatically.
- The details of Chi-Squared testing.
- Handling missing values.

For more information

- **Elements of information theory**, John Wiley & Sons, New York (2006)
- Witten, I.H., Frank, E., Hall, M.A., Pal, C.J. and DATA, M., 2005.
Practical machine learning tools and techniques.