# Classification

# Classification

- Humans are good at classifying things  into  categories for example tasks such as filtering spam email messages.

- Manual classification suffices for small and simple data sets with only a few attributes, larger and more complex datasets require and automated solution.

# Basic concepts

- The data for a classification task consists of a collection of instances (records).
- Each instance is characterized by a tuple (x,y) where x is the set of attribute values that describe the instance and y is the class label of the instance.
- The attribute set x can contain attributes of any type while the class label y must be categorical.

# Classification Model

- A classification model is an abstract representation of the relationship between the attribute set and the class label. The model can be represented in many ways eg as a tree, a probability table, or simply a vector of real valued parameters.

- Formally this can be expressed mathematically as a target function f that takes as input the attribute set x and produces output corresponding to the predicted class label. The model is said to classify an instance(x,y) correctly if f(x)=y.

# Illustration of a classification task

# Examples of classification tasks

- Spam filtering and tumor classification are **binary classification** where data can be classified in each of two instances.
- If the classes are more than two then this is a **multi-classification** problem.

| Task | Attribute Set | Class label |
|---|---|---|
| Spam filtering | Features extracted from email message header and content | Spam or non-spam |
| Tumor identification | Features extracted from magnetic resonance imaging (MRI) scans | Malignant or benign |
| Galaxy classification | Features extracted from telescope images | Elliptical, spiral or irregular shape |

# Example Vertebrate classification

- The table shows sample data for classifying vertebrates into mammals, reptiles, birds, fish, and amphibians. The attribute set includes characteristics of vertebrate such as its body temperature, skin cover and ability to fly. The data set can also be used for binary classification task such as mammal classification, by grouping the reptiles, birds, fish, and amphibians into a single category called non-mammals.

# Example Vertebrate Classification

| Vertebrate Name | Body temperature | Skin cover | Gives birth | Aquatic creature | Aerial creature | Has legs | Hibernates | Class Label |
|---|---|---|---|---|---|---|---|---|
| human | Warm-blooded | hair | yes | no | no | yes | no | mammal |
| python | Cold-blooded | scales | no | no | no | no | yes | reptile |
| salmon | Cold-blooded | scales | no | yes | no | no | no | fish |
| whale | Warm-blooded | hair | yes | yes | no | no | no | mammal |
| frog | Cold-blooded | none | no | semi | no | yes | yes | amphibian |
| komodo | Cold-blooded | scales | no | no | no | yes | no | reptile |
| bat | Warm-blooded | hair | yes | no | yes | yes | yes | mammal |
| pigeon | Warm-blooded | feathers | no | no | yes | yes | no | bird |
| cat | Warm-blooded | hair | yes | no | no | yes | no | mammal |
| Leopard shark | Cold-blooded | scales | yes | yes | no | no | no | fish |
| turtle | Cold-blooded | scales | no | semi | no | yes | no | reptile |
| penguin | Warm-blooded | feathers | no | semi | no | yes | no | bird |
| porcupine | Warm-blooded | quills | yes | no | no | yes | yes | mammal |
| eel | Cold-blooded | scales | no | yes | no | no | no | fish |
| salamander | Cold-blooded | none | no | semi | no | yes | yes | amphibian |

# Example Loan Borrower Classification

| ID | Home owner | Marital status | Annual Income | Defaulted |
|---|---|---|---|---|
| 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 |

Consider the problem of predicting whether a loan borrower will repay loan or default on the loan payment. The attribute set includes personal information of the borrower such as marital status and annual income, while the class label indicates whether the borrower had defaulted on the payments

# Classification model

- A classification model serves two important roles in data mining.
  - First it is used as a predictive model to classify previously unlabeled instances. A good classification model must provide accurate predictions with a fast response time.
  - Second it serves as a descriptive model to identify characteristics that distinguish instances from different classes. This is particularly useful for applications such as medical diagnosis where it is insufficient to have a model that makes a prediction without justifying how it reaches such a decision.

For example - The classification model induced from the vertebrate data set can be used to predict the class label for

| Vertebrate Name | Body temperature | Skin cover | Gives birth | Aquatic creature | Aerial creature | Has legs | Hibernates | Class Label |
|---|---|---|---|---|---|---|---|---|
| Gila monster | cold-blooded | scales | no | no | no | yes | yes | ? |

It can also be used as descriptive model to help determine the characteristics that define a vertebrate as a mammal, Reptile, a bird, a fish or an amphibian

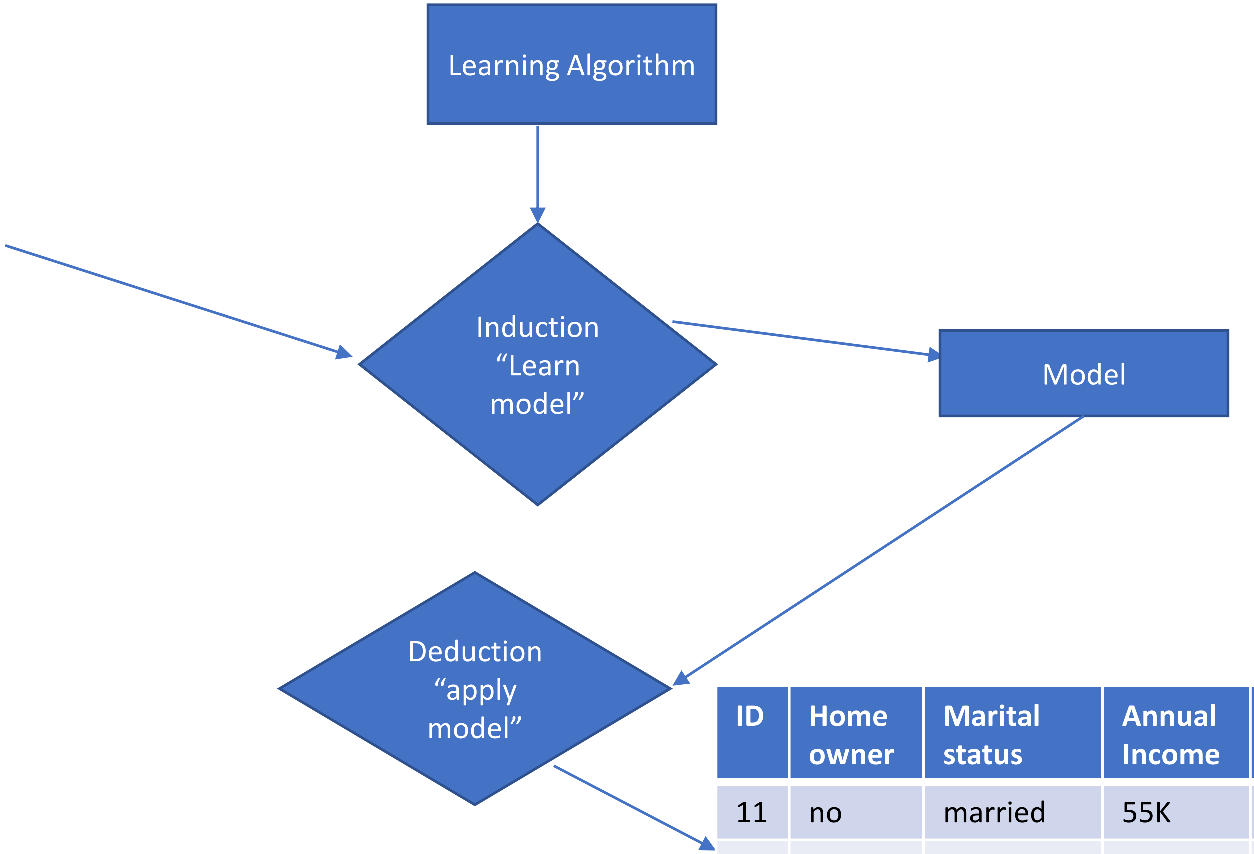
# General Framework for Classification

- Classification is the task of assigning labels to unlabeled data instances, this is done by a classifier. The model is created using a set of instances known as the training set, which contains attribute values as well as class labels for each instance.

- The systematic approach for learning a classification model given a training set is known as a **learning algorithm**. The process of using a learning algorithm to build a classification model from the training data is known as **induction**. The process of applying a classification model on unseen test instances to predict their class labels is known as **deduction**.

# General framework for building a classification model

| ID | Home owner | Marital status | Annual Income | Defaulted borrower |
|---|---|---|---|---|
| 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 |




| ID | Home owner | Marital status | Annual Income | Defaulted borrower |
|---|---|---|---|---|
| 11 | no | married | 55K | ? |
| 12 | yes | divorced | 80K | ? |
| 13 | yes | single | 110K | ? |
| 14 | no | single | 95K | ? |

# Performance of classifier

- The performance of a classifier can be evaluated by comparing the predicted labels against the true labels of the instances.
- The information can be summarized in matrix called **confusion matrix**.
- Each entry $\mathbf{f_{ij}}$ denotes the number of instances from **class i predicted as class j**. For example, $\mathbf{f_{01}}$ is the number of instances from **class 0 incorrectly predicted as class 1**. The number of **correct** predictions made by the model is ($f_{11}$ + $f_{00}$) and the number of incorrect predictions is ($f_{10}$+$f_{01}$)

| | | **Predicted Class** | |
|---|---|---|---|
| | | Class = 1 | Class = 0 |
| Actual Class | Class = 1 | $\mathbf{f_{11}}$ | $f_{10}$ |
| | Class = 0 | $f_{01}$ | $\mathbf{f_{00}}$ |

Confusion Matrix

# Accuracy of classification algorithm

- Although the confusion matrix summarizes the data, combining this to a single number makes it more convenient to compare the relative performance of different models. This can be done using an evaluation metric such as accuracy, computed below:

$$Accuracy = \frac{Number\ of\ correct\ predictions}{Total\ number\ of\ predictions}$$

- For binary classification problems, the accuracy of a model is given by

$$Accuracy = \frac{f_{11} + f_{00}}{f_{11} + f_{10} + f_{01} + f_{00}}$$

# Error Rate

- Error rate is another related metric, which is defined as follows for binary classification problems.

$$Error\ rate = \frac{Number\ of\ incorrect\ predictions}{Total\ number\ of\ predictions} = \frac{f_{10}+f_{01}}{f_{11}+f_{10}+f_{01}+f_{00}}$$

- The learning algorithms of most classification techniques are designed to **learn models that attain** the **highest accuracy**, or equivalently, the **lowest error** when applied to the test set.

# Classification Techniques

- Base Classifiers
  - Decision Tree based Methods
  - Nearest-neighbour
  - Naïve Bayes
  - Support Vector Machines (SVM)

# Decision Tree Classifier

- A simple **classification technique** is known as **Decision Tree classification**.
- To illustrate how a decision tree works, consider the classification problem of distinguishing mammals from non-mammals using the vertebrae data set
- How can we tell  whether an animal is a mammal or not a mammal?
- One approach is to pose a series of questions about the characteristics of the species.
- The first question we may ask is whether the species is cold or warm blooded. If cold blooded then it is definitely not a mammal.

If it is cold blooded then it definitely not a mammal.

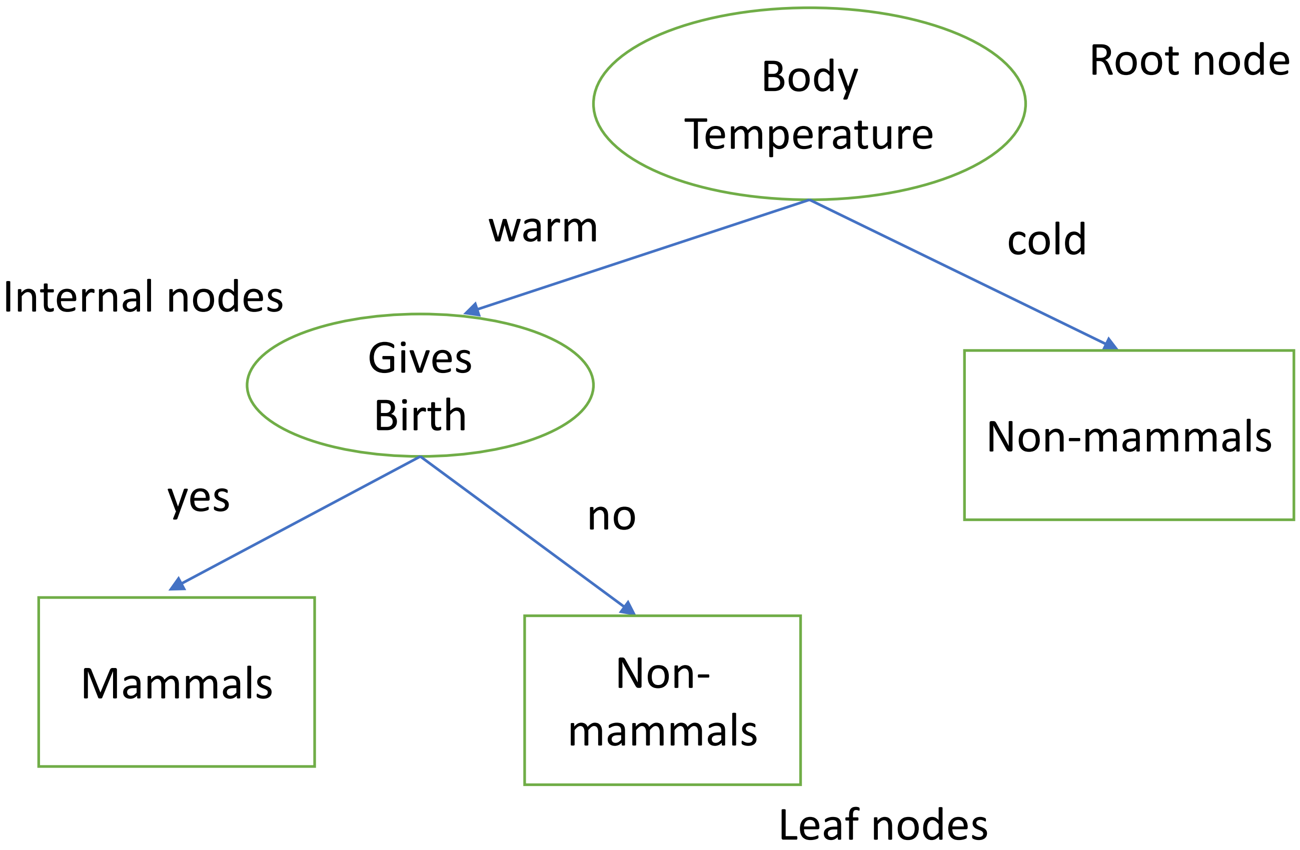
Otherwise it is a bird or a mammal. In the latter case, we need to ask a follow up question .

Do females of these species give birth to their young?

Those that give birth are definitely mammals, while those that do not are definitely not mammals.

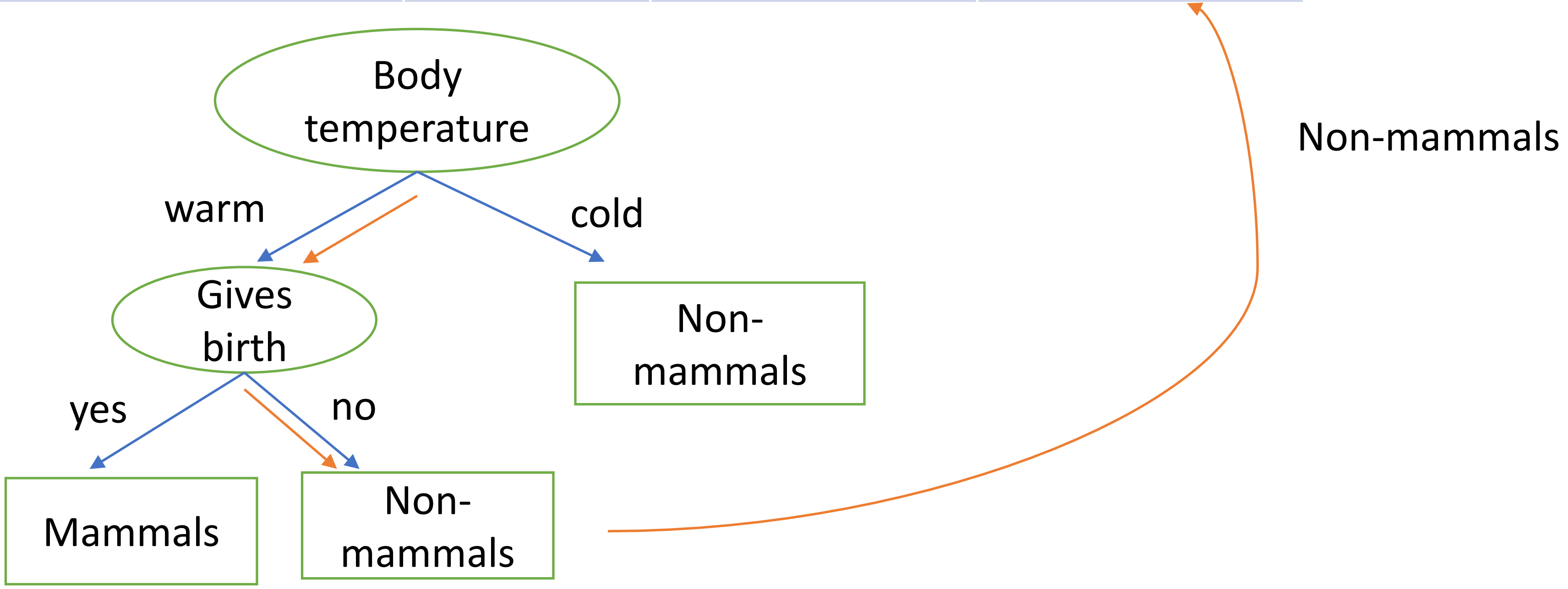
- This example illustrated solving a classification problem by asking a series of questions about the attributes of the test instances.
- Each time an answer is received then a follow up question is asked until the class label is decided. The series of questions and their possible answers can be organized into a hierarchical structure called a decision tree.
- The tree has three types of nodes
  - A root node
  - Internal nodes
  - Leaf nodes
- Every leaf node in the decision tree is associated with a class label.
- The non-leaf nodes, which include the root and internal nodes contain attribute test conditions that are typically defined using a single attribute.

- A decision tree for a mammal classification problem



- Classifying an unlabeled vertebrate.

| Name | Body temperature | Gives birth | …. | Class |
|---|---|---|---|---|
| Flamingo | Warm | No | …. | ? |

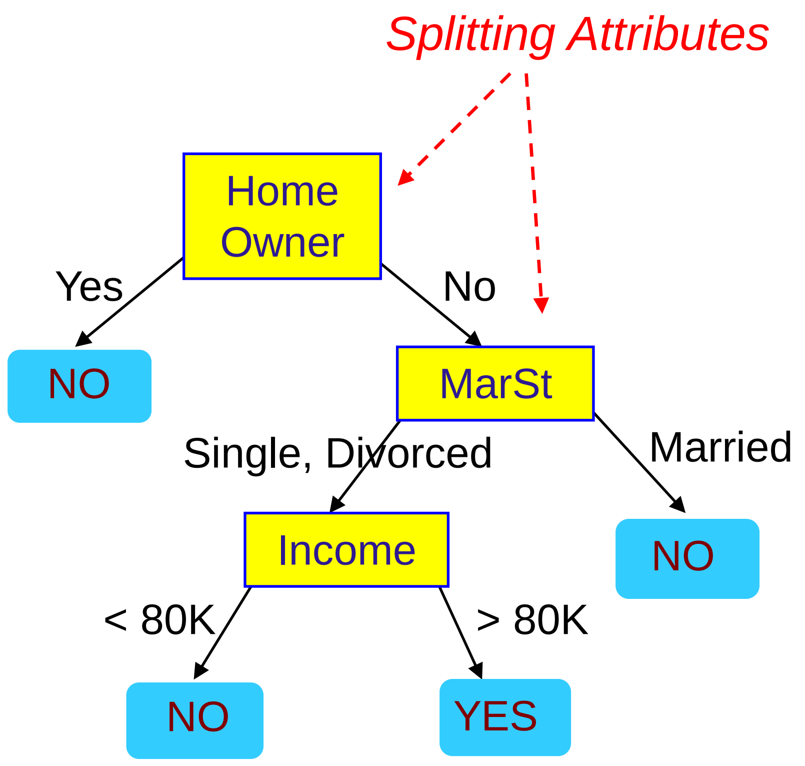


# Example of a Decision Tree

categorical categorical continuous class

| ID | Home Owner | Marital Status | Annual Income | Defaulted |
|---|---|---|---|---|
| 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 |

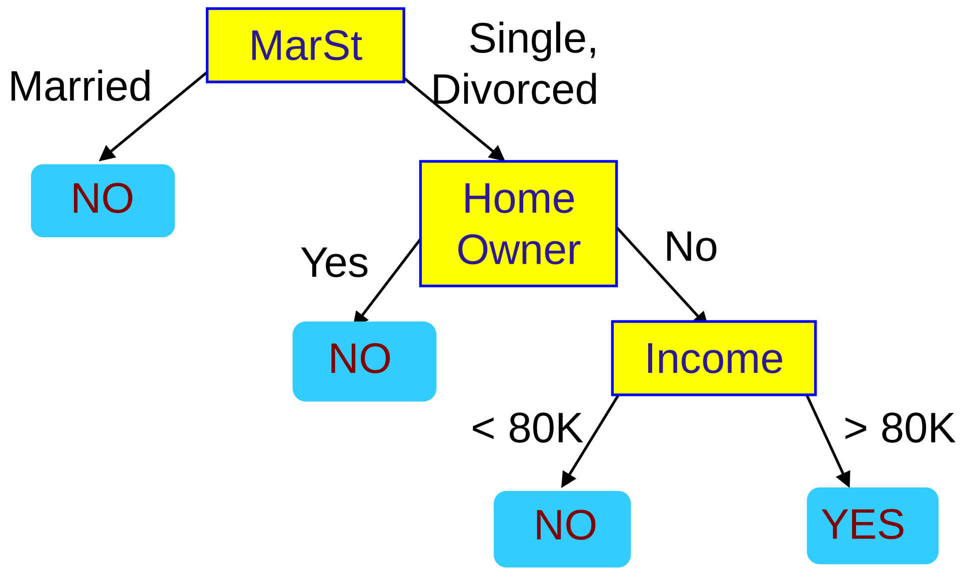
Training Data




Model: Decision Tree

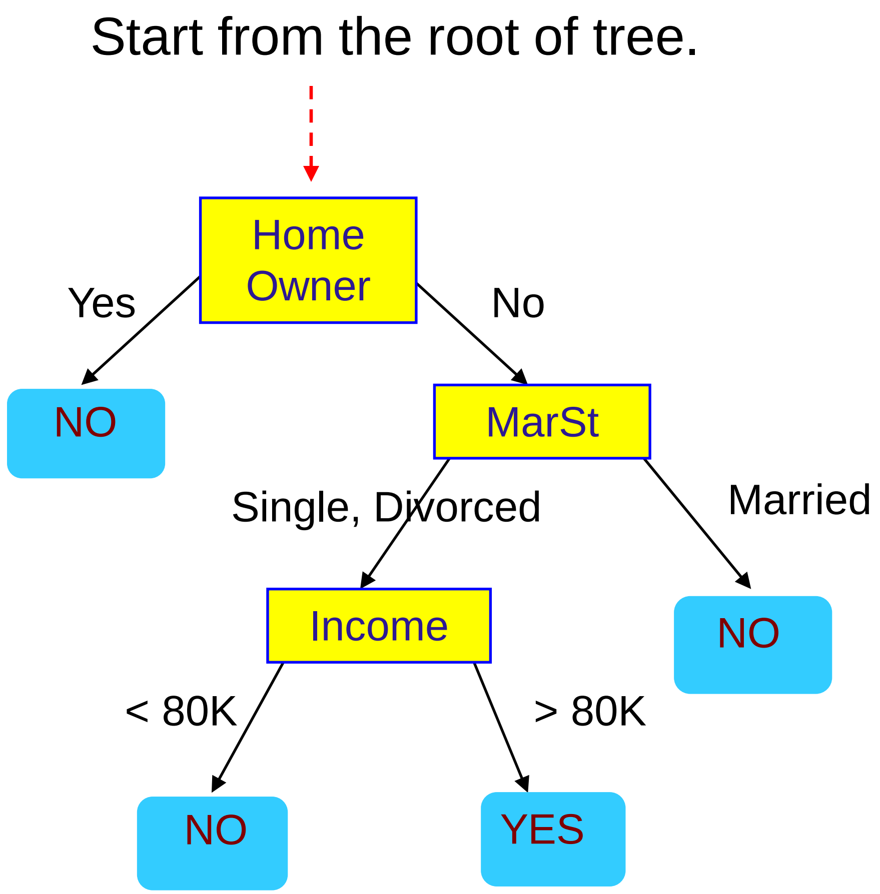
# Another Example of Decision Tree

| ID | Home Owner | Marital Status | Annual Income | Defaulted Borrower |
|---|---|---|---|---|
| | categorical | categorical | continuous | class |
| 1 | Yes | Single | 125K | **No** |
| 2 | No | Married | 100K | **No** |
| 3 | No | Single | 70K | **No** |
| 4 | Yes | Married | 120K | **No** |
| 5 | No | Divorced | 95K | **Yes** |
| 6 | No | Married | 60K | **No** |
| 7 | Yes | Divorced | 220K | **No** |
| 8 | No | Single | 85K | **Yes** |
| 9 | No | Married | 75K | **No** |
| 10 | No | Single | 90K | **Yes** |




There could be more than one tree that fits the same data!

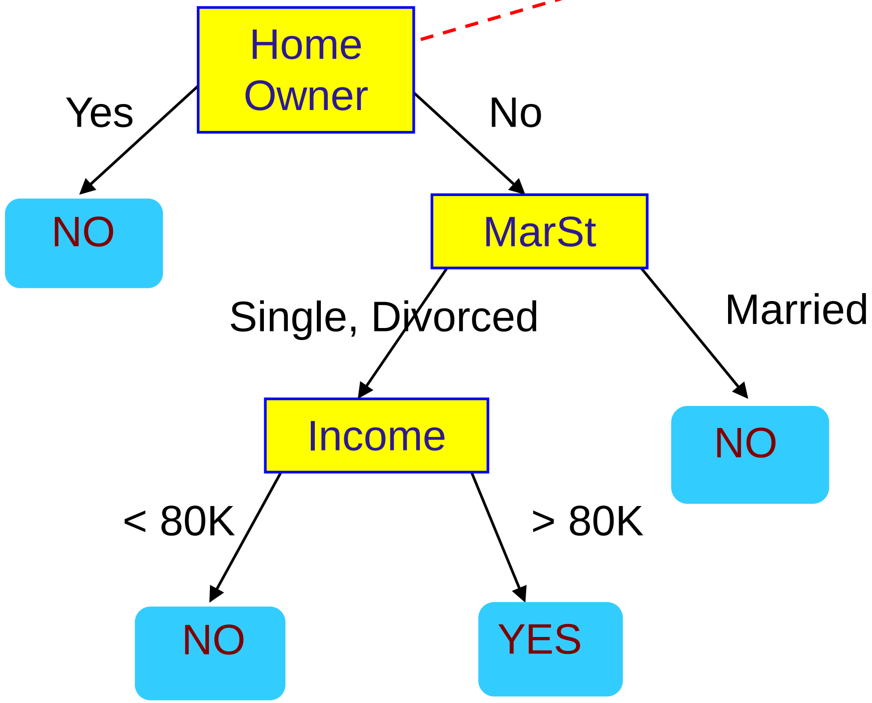
# Apply Model to Test Data




Test Data

| Home Owner | Marital Status | Annual Income | Defaulted Borrower |
|---|---|---|---|
| No | Married | 80K | ? |

# Apply Model to Test Data



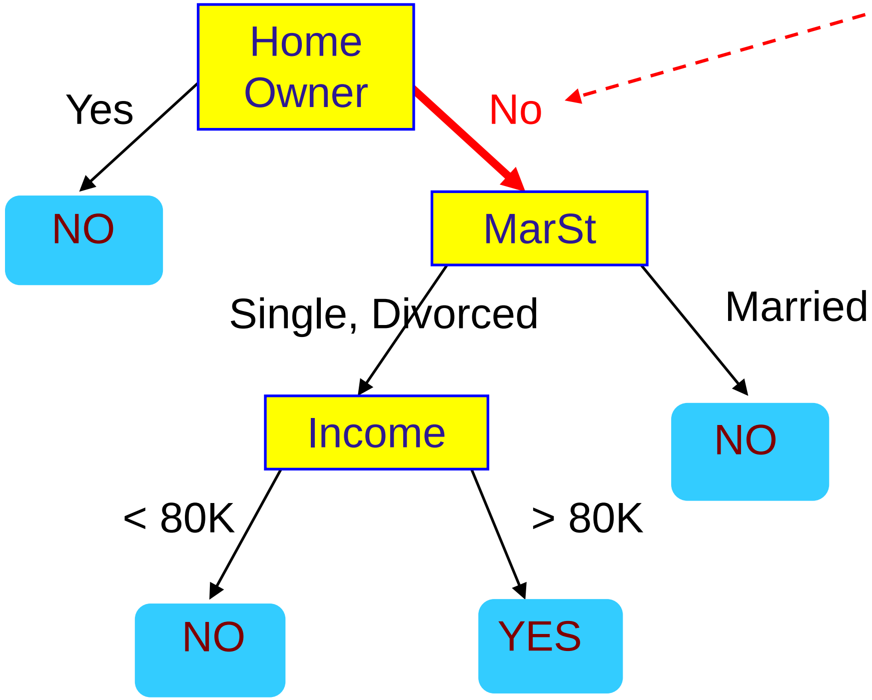

Test Data

| Home Owner | Marital Status | Annual Income | Defaulted Borrower |
|---|---|---|---|
| No | Married | 80K | ? |

# Apply Model to Test Data

Test Data

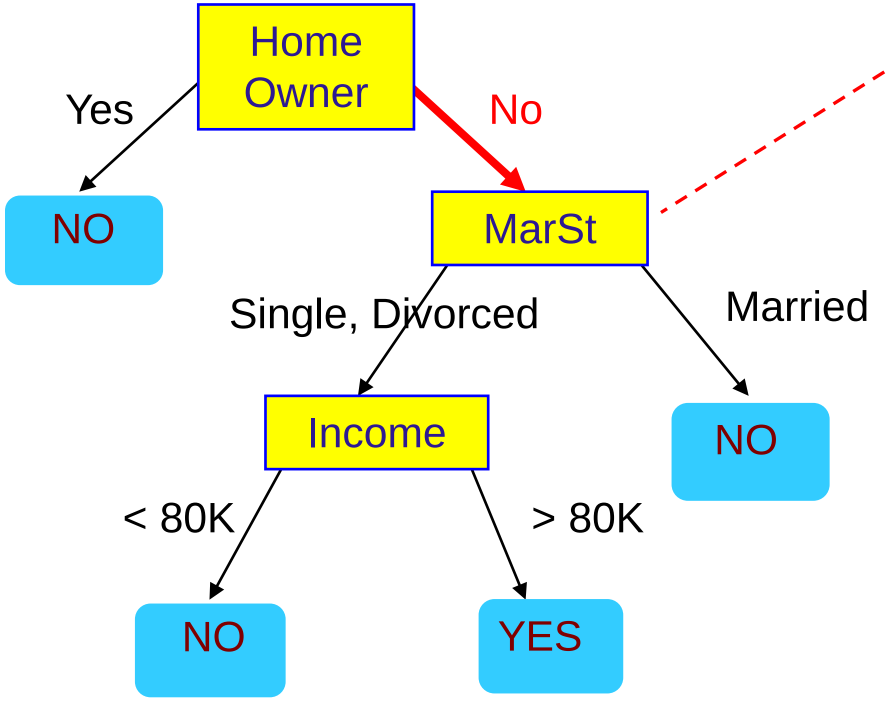
| Home Owner | Marital Status | Annual Income | Defaulted Borrower |
|---|---|---|---|
| No | Married | 80K | ? |



# Apply Model to Test Data

Test Data

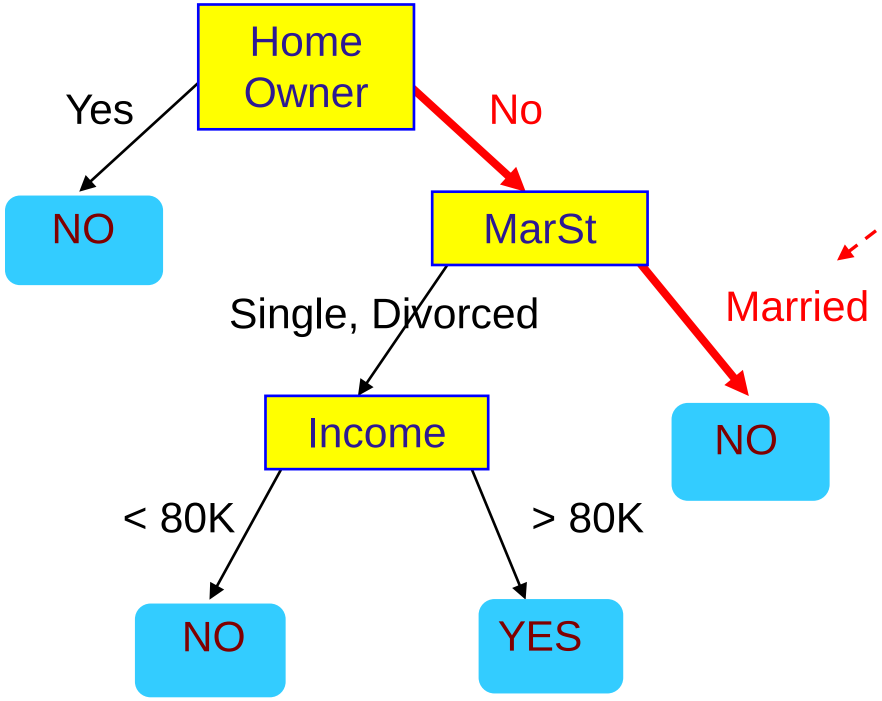
| Home Owner | Marital Status | Annual Income | Defaulted Borrower |
|---|---|---|---|
| No | Married | 80K | ? |



# Apply Model to Test Data

Test Data

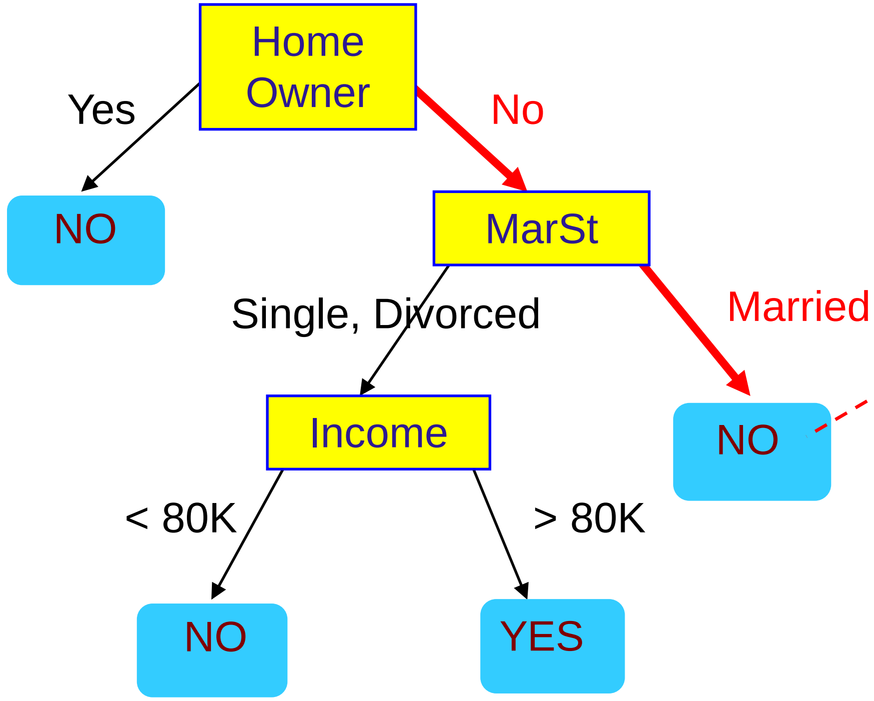
| Home Owner | Marital Status | Annual Income | Defaulted Borrower |
|---|---|---|---|
| No | Married | 80K | ? |



# Apply Model to Test Data

Test Data

| Home Owner | Marital Status | Annual Income | Defaulted Borrower |
|---|---|---|---|
| No | Married | 80K | ? |




Assign Defaulted to "No"

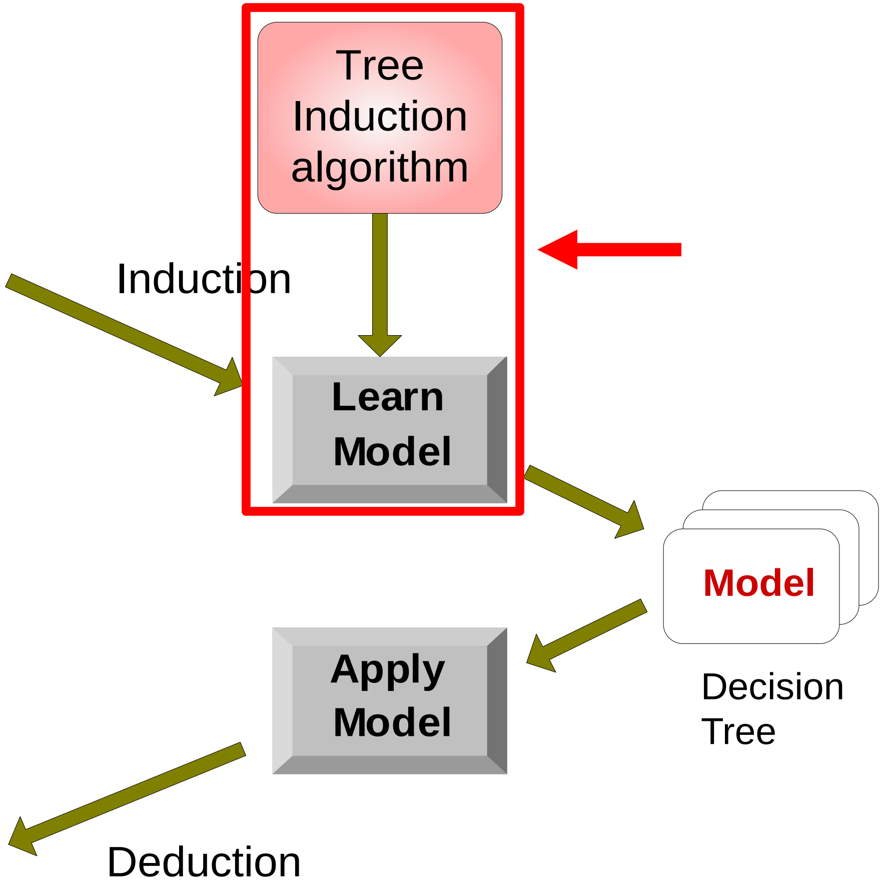
# Decision Tree Classification Task

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

Training Set

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

Test Set



# A basic algorithm to build a decision tree

- One of the earliest algorithms is the Hunt’s Algorithm, which is the basis for many current implementation of decision tree algorithms ID3, C4.5 and CART.

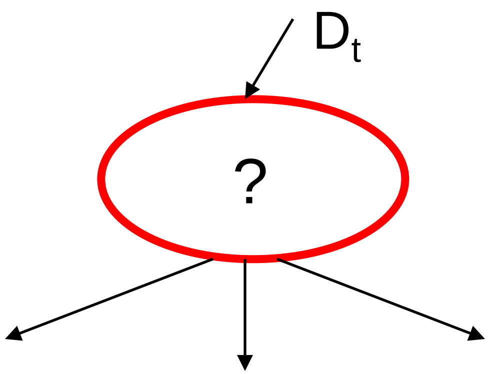
- In Hunt’s algorithm a decision tree is grown in a recursive fashion, the tree initially contains a single root node that is associated with all the training instances.

# General Structure of Hunt's Algorithm

- Let $D_t$ be the set of training records that reach a node t

- General Procedure:
  - If $D_t$ contains records that belong the same class $y_t$, then t is a leaf node labeled as $y_t$
  - If $D_t$ contains records that belong to more than one class, use an attribute test to split the data into smaller subsets. Recursively apply the procedure to each subset.

| ID | Home Owner | Marital Status | Annual Income | Defaulted Borrower |
|---|---|---|---|---|
| 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 |



# Hunt's Algorithm

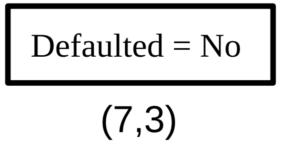
$$\boxed{\text{Defaulted = No}}$$

(7,3)

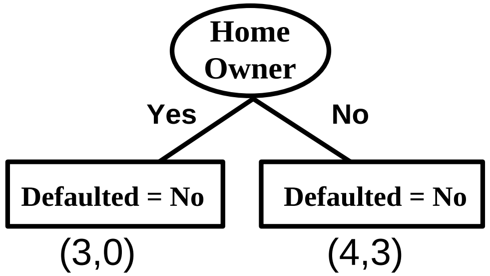
(a)

| ID | Home Owner | Marital Status | Annual Income | Defaulted Borrower |
|---|---|---|---|---|
| 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 |

# Hunt's Algorithm



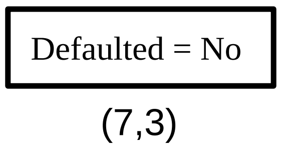

(a)



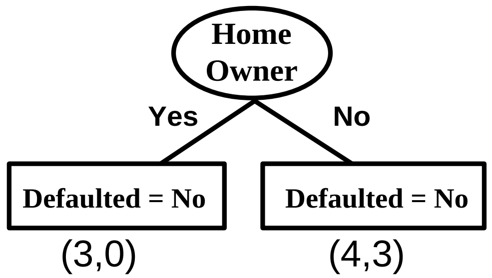

(b)

| ID | Home Owner | Marital Status | Annual Income | Defaulted Borrower |
|---|---|---|---|---|
| 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 |

# Hunt's Algorithm

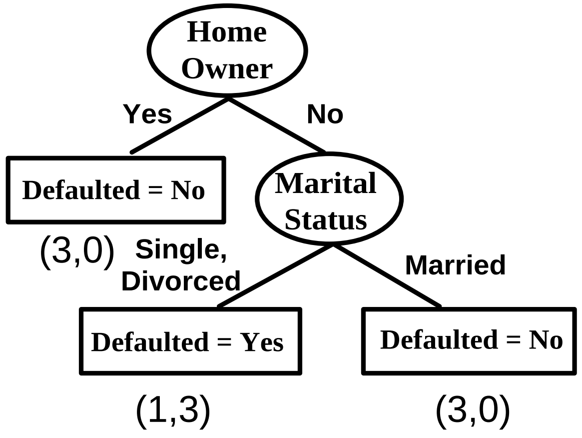



(a)



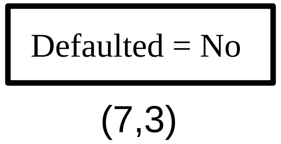

(b)

| ID | Home Owner | Marital Status | Annual Income | Defaulted Borrower |
|---|---|---|---|---|
| 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 |

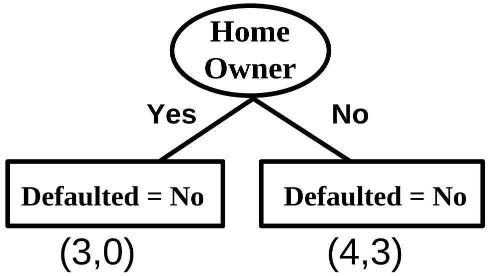



(c)

# Hunt’s Algorithm

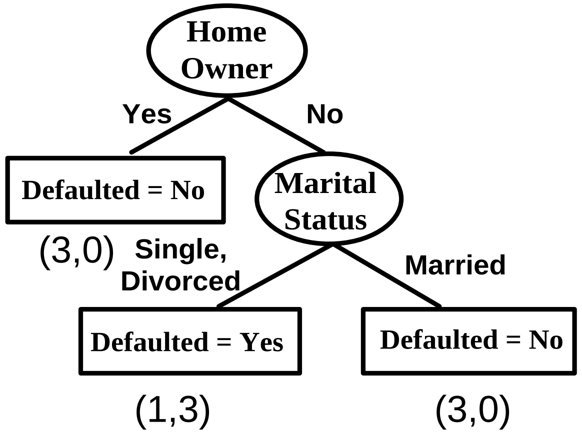



(a)

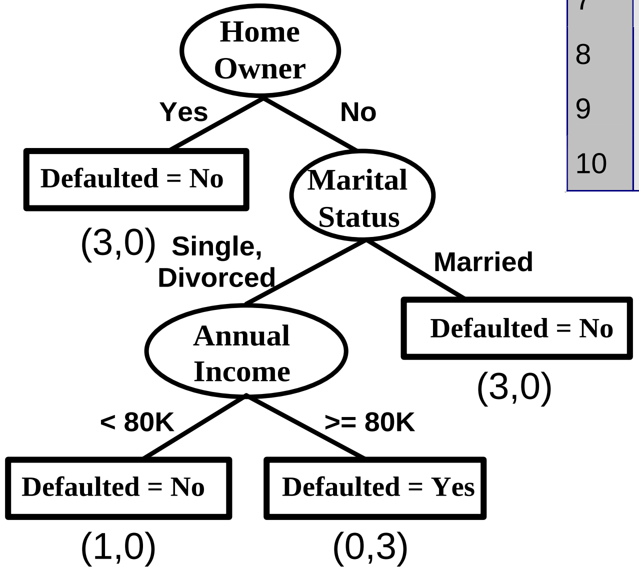



(b)

| ID | Home Owner | Marital Status | Annual Income | Defaulted Borrower |
|---|---|---|---|---|
| 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 |




(c)




(d)

# Design Issues of Decision Tree Induction

- How should training records be split?
  - Method for specifying test condition
    - depending on attribute types
  - Measure for evaluating the goodness of a test condition

- How should the splitting procedure stop?
  – Stop splitting if all the records belong to the same class or have identical attribute values

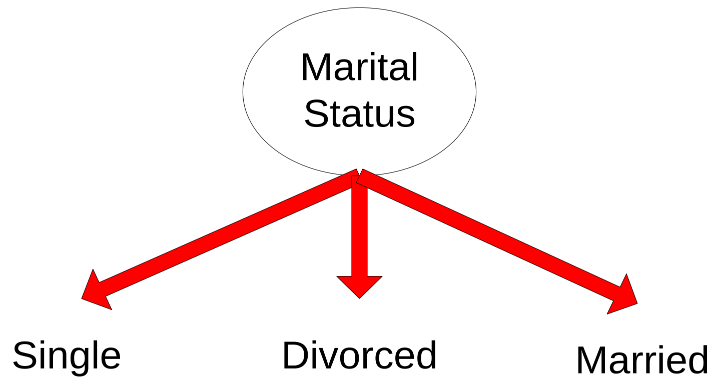
# Methods for Expressing Attribute Test Conditions

- Depends on attribute types
  - Binary - The test condition for a binary attribute generates two potential outcomes.
  - Nominal - A nominal attributes can have many values its attribute test condition can be expressed  in two ways either as a multiway split or a binary split
  - Ordinal – this can be expressed as a multiway split or a binary split
  - Continuous – this can be expressed as a multiway split or a binary split

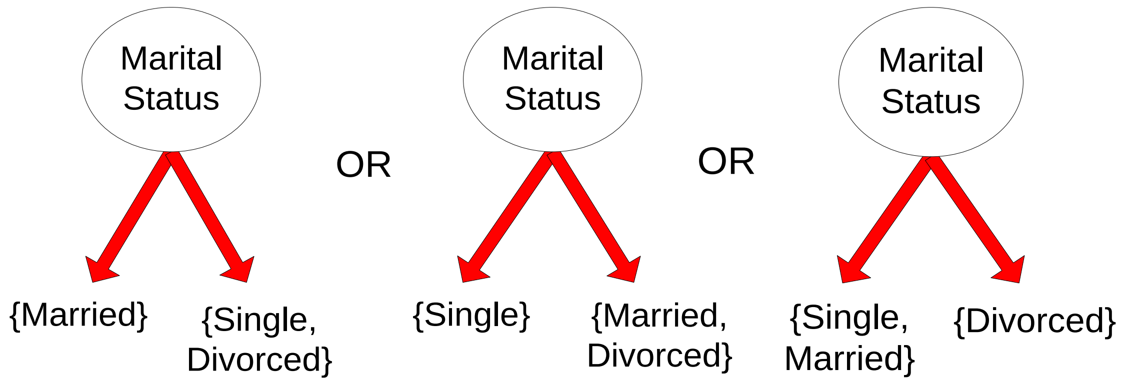
- Depends on number of ways to split
  - 2-way split
  - Multi-way split

# Test Condition for Nominal Attributes

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

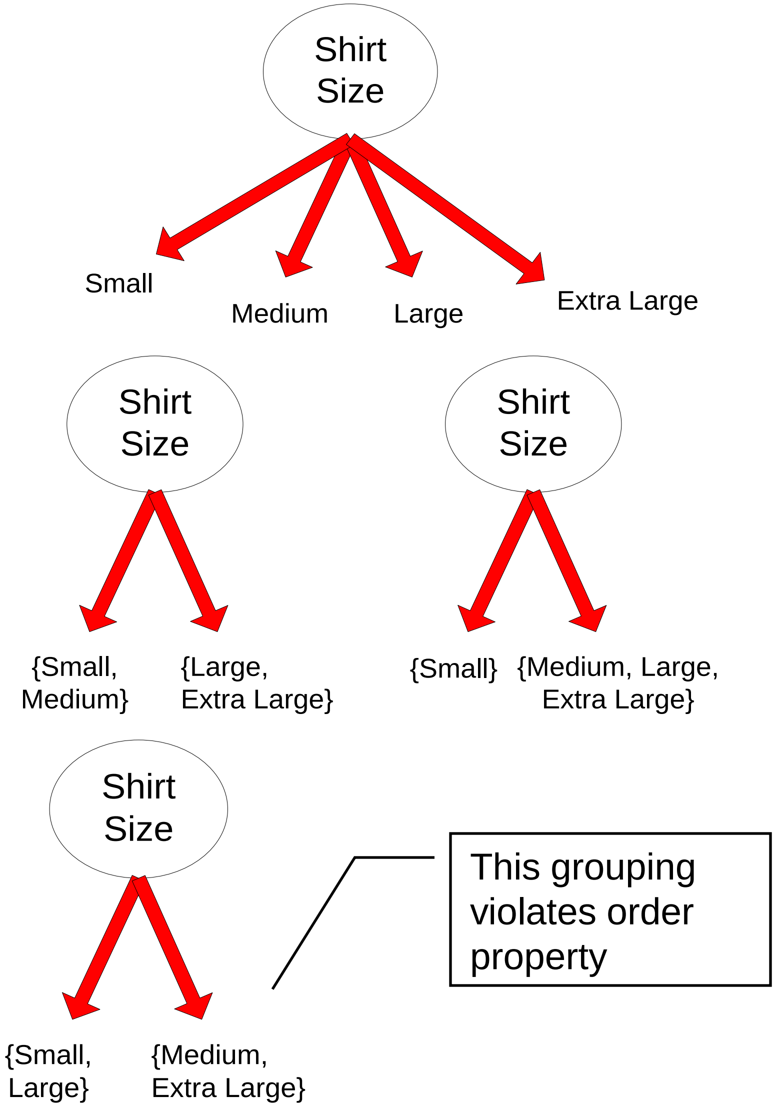



- Binary split:
  - Divides values into two subsets

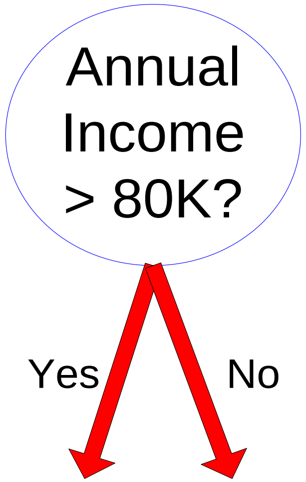


# Test Condition for Ordinal Attributes

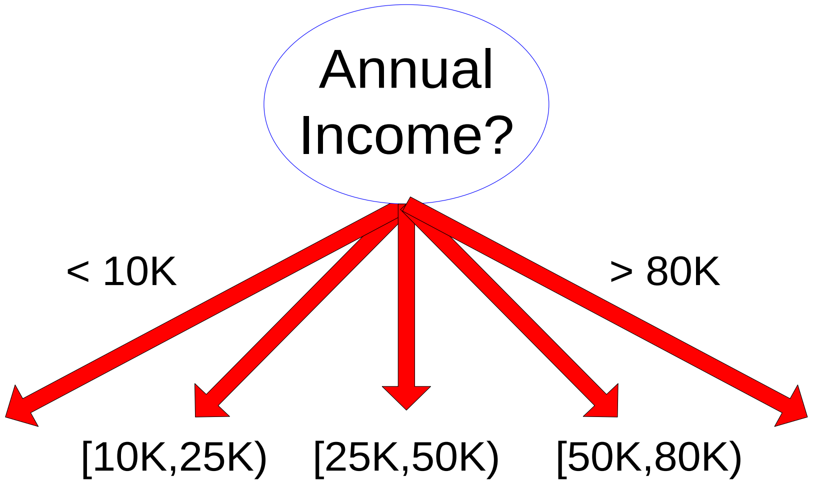
- Multi-way split:
  - Use as many partitions as distinct values
- Binary split:
  - Divides values into two subsets
  - Preserve order property among attribute values



# Test Condition for Continuous Attributes




(i) Binary split




(ii) Multi-way split

# Splitting Based on Continuous Attributes

- Different ways of handling continuous attributes
  - Discretization to form an ordinal categorical attribute

    Ranges can be found by equal interval bucketing, equal frequency bucketing (percentiles), or clustering.

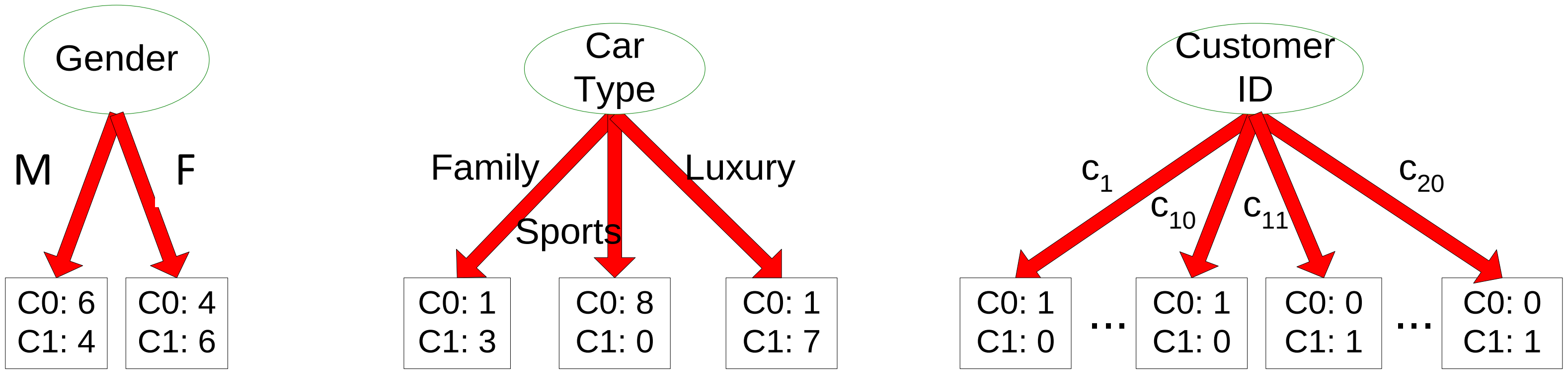
    - Static – discretize once at the beginning
    - Dynamic – repeat at each node

  - Binary Decision: $(A < v)$ or $(A \geq v)$
    - consider all possible splits and finds the best cut
    - can be more compute intensive

# How to determine the Best Split

| Customer Id | Gender | Car Type | Shirt Size | Class |
|---|---|---|---|---|
| 1 | M | Family | Small | C0 |
| 2 | M | Sports | Medium | C0 |
| 3 | M | Sports | Medium | C0 |
| 4 | M | Sports | Large | C0 |
| 5 | M | Sports | Extra Large | C0 |
| 6 | M | Sports | Extra Large | C0 |
| 7 | F | Sports | Small | C0 |
| 8 | F | Sports | Small | C0 |
| 9 | F | Sports | Medium | C0 |
| 10 | F | Luxury | Large | C0 |
| 11 | M | Family | Large | C1 |
| 12 | M | Family | Extra Large | C1 |
| 13 | M | Family | Medium | C1 |
| 14 | M | Luxury | Extra Large | C1 |
| 15 | F | Luxury | Small | C1 |
| 16 | F | Luxury | Small | C1 |
| 17 | F | Luxury | Medium | C1 |
| 18 | F | Luxury | Medium | C1 |
| 19 | F | Luxury | Medium | C1 |
| 20 | F | Luxury | Large | C1 |

Before Splitting: 10 records of class 0,
10 records of class 1




Which test condition is the best?

# How to determine the Best Split

- Greedy approach:
  - Nodes with purer class distribution are preferred

- Need a measure of node impurity:

| C0: 5 |
| --- |
| C1: 5 |

High degree of impurity

| C0: 9 |
| --- |
| C1: 1 |

Low degree of impurity

# Measures for selecting an attribute test condition

- There are many measures that can be used to determine the goodness of an attribute test condition.

- These measures try to give preference to attribute test conditions that partition the training instances into purer subsets in the child nodes, which mostly have the same class labels.

- Having purer nodes is useful since a node that has all of its training instances from the same class labels is a leaf node.

# Impurity Measures for a single node

- The impurity of a node measures how dissimilar the class labels are for the  instances belonging to a common node.
- Following are examples of measures that can used to evaluate the impurity of node t:
  - $Entropy = -\sum_{i=1}^{c} p_i(t) log_2 p_i(t)$
  - $Gini\ index = 1 - \sum_{i=1}^{c} p_i(t)^2$
  - $Classification\ error = 1 - \max[pi(t)]$
- Where $p_i(t)$ is the relative frequency of training instances that belong to class i at node t, c is the total number of classes.

# Example

- Below are the examples of calculating these

$$Entropy = -\sum_{i=1}^{c} p_i(t) log_2 p_i(t)$$

$$Gini\ index = 1 - \sum_{i=1}^{c} p_i(t)^2$$

$$Classification\ error = 1 - \max[pi(t)]$$

| Node $N_1$ | Count |
|---|---|
| Class = 0 | 0 |
| Class =1 | 6 |

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

| Node $N_2$ | Count |
|---|---|
| Class = 0 | 1 |
| Class =1 | 5 |

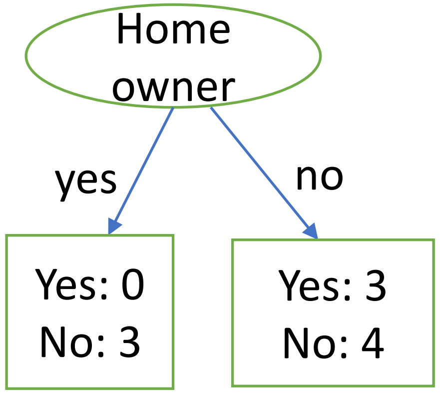
Gini = 1 – $(1/6)^2$ - $(5/6)^2$ = 0.278
Entropy = -(1/6)$\log_2$(1/6)-(5/6)$\log_2$(5/6)=0.650
Error = 1- max[1/6,5/6]=0.167

| Node $N_3$ | Count |
|---|---|
| Class = 0 | 3 |
| Class =1 | 3 |

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

# Example [Entropy]

- Consider the candidate attribute test condition shown in the figure for the loan borrower classification problem. Splitting on the Home Owner attribute will generate two child nodes whose weighted entropy can be calculated as:



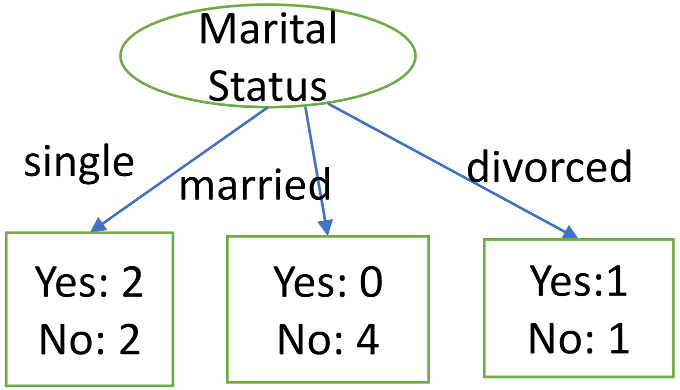

Entropy( home owner, yes)

$=- 0/3 \log_2 0/3 – 3/3 \log_2 3/3 = 0$

Entropy( home owner, no)

$=- 3/7 \log_2 3/7 – 4/7 \log_2 4/7 = 0.985$

- Consider the candidate attribute test condition shown in the figure for the Marital Status classification problem. Splitting on the Martial Status attribute will generate two child nodes whose weighted entropy can be calculated as:




Entropy( MartSt, single)

$=- 2/4 \log_2 2/4 – 2/4 \log_2 2/4 = 1$

Entropy( MartSt, married)

$=- 0/4 \log_2 0/7 – 4/4 \log_2 4/4 = 0$

Entropy( MartSt, divorced)

$=- 1/2 \log_2 1/2 – 1/2 \log_2 1/2= 1$

# Identifying the best attribute test condition

- To determine the **goodness of an attribute test condition** we need to compare the **degree of impurity of the parent node (before splitting)** with the **weighted degree of the impurity child nodes (after splitting).** The l**arger** their difference the **better** the test condition. This difference is also termed as the gain in purity of an attribute test condition

$$\Delta = I(parent) - I(children)$$

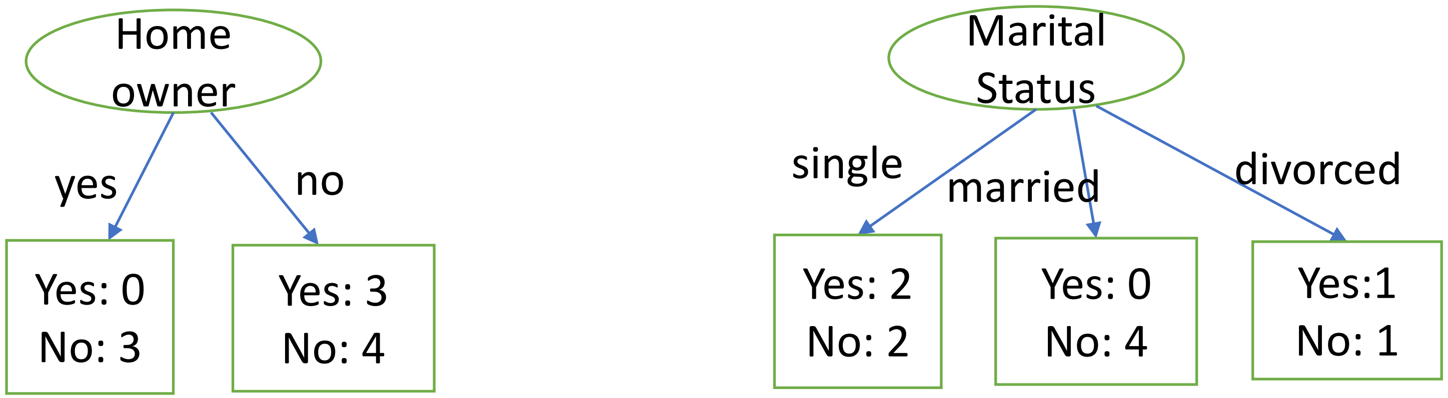
Where *I(parent)* is the impurity of the node before splitting and *I(children)* is the weighted impurity of the node after splitting

This is commonly known as **information gain**

# Splitting of Qualitative Attributes

Consider the first two candidate splits shown in the figure involving qualitative attributes Home Owner and Marital Status. The initial class distribution at the parent node is that there are 3 instances of class Yes and 7 instances of class No in the training data. Thus:

$$I(parent) = -\frac{3}{10}\log_2\frac{3}{10} - \frac{7}{10}\log_2\frac{7}{10} = 0.881$$

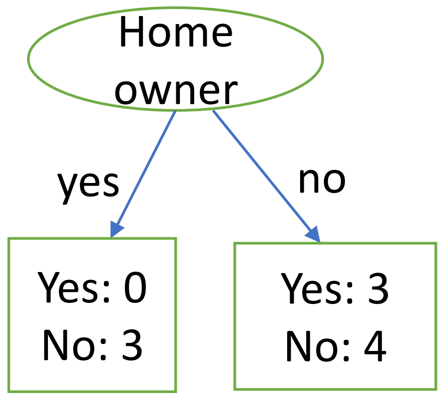


- Splitting on the home owner

$$I(Home\ owner = yes) = -\frac{0}{3}\log_2\frac{0}{3} - \frac{3}{3}\log_2\frac{3}{3} = 0$$

$$I(Home\ owner = no) = -\frac{3}{7}\log_2\frac{3}{7} - \frac{4}{7}\log_2\frac{4}{7} = 0.985$$

I(Home owner)= $\frac{3}{10}$ x 0 + $\frac{7}{10}$ x 0.985 = 0.690



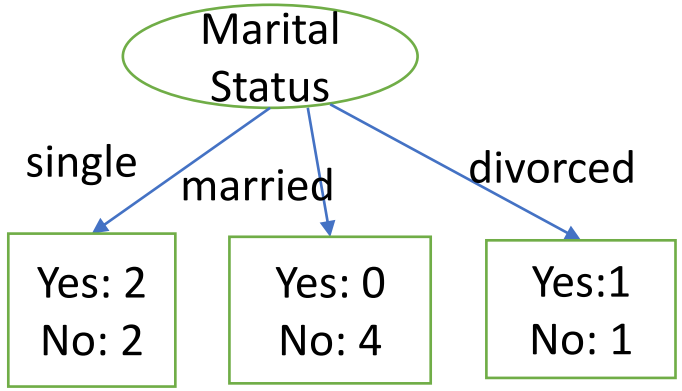

- Splitting on the martial status

$$I(Martial\ Status = Single) = -\frac{2}{4}\log_2\frac{2}{4} - \frac{2}{4}\log_2\frac{2}{4} = 1$$

$$I(Martial\ Status = Married) = -\frac{0}{4}\log_2\frac{0}{4} - \frac{4}{4}\log_2\frac{4}{4} = 0$$

$$I(Martial\ Status = Divorced) - \frac{1}{2}\log_2\frac{1}{2} - \frac{1}{2}\log_2\frac{1}{2} = 1$$

I(Martial status)= $\frac{4}{10}$ x 1 + $\frac{4}{10}$ x 0+ $\frac{2}{10}$ x1 = 0.6



$$\Delta = I(parent) - I(children)$$

$$\Delta(Home\ Owner) = 0.881\ - 0.690 = 0.191$$
$$\Delta(Marital\ Status) = 0.881\ - 0.6 = 0.281$$

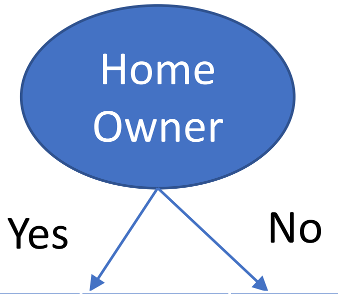
Martial status has the larger difference

# Binary Splitting of Qualitative Attributes

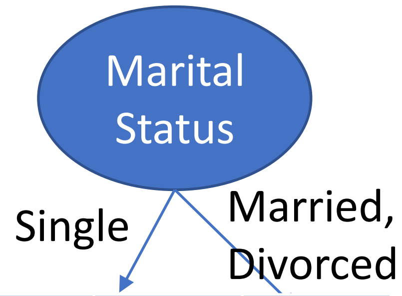
- Consider building a decision tree using only binary splits and the Gini index as the impurity measure. Consider the diagram below which shows four candidate splitting for the Home Owner and Marital Status Attributes.

| | Parent |
|---|---|
| No | 7 |
| Yes | 3 |
| Gini = 0.420 | |

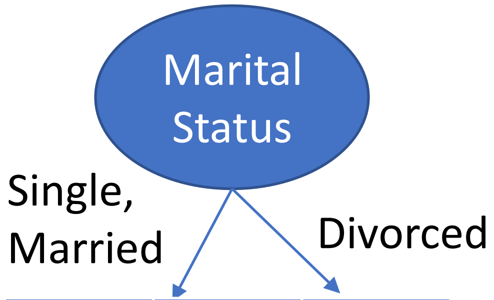
$$Gini\ index = 1 - \sum_{i=1}^{c} p_i(t)^2$$



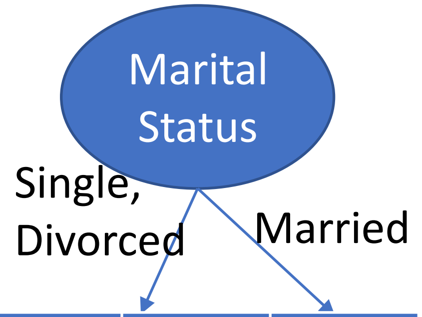

| | N1 | N2 |
|---|---|---|
| No | 3 | 4 |
| Yes | 0 | 3 |
| Gini | 0 | 0.490 |
| Weighted Gini = 0.343 | | |




| | N1 | N2 |
|---|---|---|
| No | 2 | 5 |
| Yes | 2 | 1 |
| Gini | 0.5 | 0.278 |
| Weighted Gini = 0.367 | | |




| | N1 | N2 |
|---|---|---|
| No | 6 | 1 |
| Yes | 2 | 1 |
| Gini | 0.375 | 0.5 |
| Weighted Gini = 0.400 | | |




| | N 1 | N2 |
|---|---|---|
| No | 3 | 4 |
| Yes | 3 | 0 |
| Gini | 0.5 | 0 |
| Weighted Gini = 0.3 | | |

- The Gini index of the parent node before splitting is

$$1 - \left(\frac{3}{10}\right)^2 - \left(\frac{7}{10}\right)^2 = 0.420$$

If Home Owner is chosen as the splitting attribute, the Gini index for the child nodes $N_1$ and $N_2$ are 0 and 0.490, respectively. The weighted index for the children is

$$\frac{3}{10} \times 0 + \frac{7}{10} \times 0.490 = 0.343$$

The gain in Home Owner as the splitting attribute is 0.420 – 0.343 = 0.077.

# Binary Splitting of Quantitative Attributes

- Consider the problem of identifying the best binary split for Annual Income. The training instances are first sorted based on their Annual Income. The candidate split is given by  a midpoint split between every adjacent sorted values  £55,000, £65,0000, £72,500 and so on.

| Class | No | No | No | Yes | Yes | Yes | No | No | No | No |
|---|---|---|---|---|---|---|---|---|---|---|
| Annual Income | 60 | 70 | 75 | 85 | 90 | 95 | 100 | 120 | 125 | 220 |

| | 55 | | 65 | | 72 | | 80 | | 87 | | 92 | | 97 | | 110 | | 122 | | 172 | | 230 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | <= | > | <= | > | <= | > | <= | > | <= | > | <= | > | <= | > | <= | > | <= | > | <= | > | <= | > |
| Yes | 0 | 3 | 0 | 3 | 0 | 3 | 0 | 3 | 1 | 2 | 2 | 1 | 3 | 0 | 3 | 0 | 3 | 0 | 3 | 0 | 3 | 0 |
| No | 0 | 7 | 1 | 6 | 2 | 5 | 3 | 4 | 3 | 4 | 3 | 4 | 3 | 4 | 4 | 3 | 5 | 2 | 6 | 1 | 7 | 0 |
| Gini | 0.420 | | 0.400 | | 0.375 | | 0.343 | | 0.417 | | 0.400 | | ***0.300*** | | 0.343 | | 0.375 | | 0.400 | | 0.420 | |

| ID | Home Owner | Marital Status | Annual Income | Defaulted |
|---|---|---|---|---|
| 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 |

# Continuous Attributes: Computing Gini Index...

- For efficient computation: for each attribute,
  - Sort the attribute on values
  - Linearly scan these values, each time updating the count matrix and computing gini index
  - Choose the split position that has the least gini index

| | Class | No | No | No | Yes | Yes | Yes | No | No | No | No |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Annual Income | | | | | | | | | |
| Sorted Values → | | 60 | 70 | 75 | 85 | 90 | 95 | 100 | 120 | 125 | 220 |

# Continuous Attributes: Computing Gini Index...

- For efficient computation: for each attribute,
  - Sort the attribute on values
  - Linearly scan these values, each time updating the count matrix and computing gini index
  - Choose the split position that has the least gini index

| Class | No | No | No | Yes | Yes | Yes | No | No | No | No |
|---|---|---|---|---|---|---|---|---|---|---|
| | Annual Income | | | | | | | | | |
| Sorted Values | 60 | 70 | 75 | 85 | 90 | 95 | 100 | 120 | 125 | 220 |

| Split Positions | 55 | | 65 | | 72 | | 80 | | 87 | | 92 | | 97 | | 110 | | 122 | | 172 | | 230 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | <= | > | <= | > | <= | > | <= | > | <= | > | <= | > | <= | > | <= | > | <= | > | <= | > | <= | > |

# Continuous Attributes: Computing Gini Index...

- For efficient computation: for each attribute,
  - Sort the attribute on values
  - Linearly scan these values, each time updating the count matrix and computing gini index
  - Choose the split position that has the least gini index

| Class | No | No | No | Yes | Yes | Yes | No | No | No | No |
|---|---|---|---|---|---|---|---|---|---|---|
| | Annual Income | | | | | | | | | |
| Sorted Values | 60 | 70 | 75 | 85 | 90 | 95 | 100 | 120 | 125 | 220 |

| Split Positions | 55 | 65 | 72 | 80 | 87 | 92 | 97 | 110 | 122 | 172 | 230 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | <= / > | <= / > | <= / > | <= / > | <= / > | <= / > | <= / > | <= / > | <= / > | <= / > | <= / > |
| Yes | | | | 0 / 3 | | | | | | | |
| No | | | | 3 / 4 | | | | | | | |
| Gini | | | | 0.343 | | | | | | | |

# Continuous Attributes: Computing Gini Index...

- For efficient computation: for each attribute,
  - Sort the attribute on values
  - Linearly scan these values, each time updating the count matrix and computing gini index
  - Choose the split position that has the least gini index

| Class | No | No | No | Yes | Yes | Yes | No | No | No | No |
|---|---|---|---|---|---|---|---|---|---|---|
| | Annual Income | | | | | | | | | |
| Sorted Values | 60 | 70 | 75 | 85 | 90 | 95 | 100 | 120 | 125 | 220 |

| Split Positions | 55 | | 65 | | 72 | | 80 | | 87 | | 92 | | 97 | | 110 | | 122 | | 172 | | 230 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | <= | > | <= | > | <= | > | <= | > | <= | > | <= | > | <= | > | <= | > | <= | > | <= | > | <= | > |
| Yes | | | | | | | 0 | 3 | 1 | 2 | | | | | | | | | | | | |
| No | | | | | | | 3 | 4 | 3 | 4 | | | | | | | | | | | | |
| Gini | | | | | | | 0.343 | | 0.417 | | | | | | | | | | | | | |

# Continuous Attributes: Computing Gini Index...

- For efficient computation: for each attribute,
  - Sort the attribute on values
  - Linearly scan these values, each time updating the count matrix and computing Gini index
  - Choose the split position that has the least Gini index

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Class | No | No | No | Yes | Yes | Yes | No | No | No | No |
| | Annual Income | | | | | | | | | |
| Sorted Values → | 60 | 70 | 75 | 85 | 90 | 95 | 100 | 120 | 125 | 220 |

| Split Positions → | 55 | | 65 | | 72 | | 80 | | 87 | | 92 | | 97 | | 110 | | 122 | | 172 | | 230 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | <= | > | <= | > | <= | > | <= | > | <= | > | <= | > | <= | > | <= | > | <= | > | <= | > | <= | > |
| Yes | 0 | 3 | 0 | 3 | 0 | 3 | 0 | 3 | 1 | 2 | 2 | 1 | 3 | 0 | 3 | 0 | 3 | 0 | 3 | 0 | 3 | 0 |
| No | 0 | 7 | 1 | 6 | 2 | 5 | 3 | 4 | 3 | 4 | 3 | 4 | 3 | 4 | 4 | 3 | 5 | 2 | 6 | 1 | 7 | 0 |
| Gini | 0.420 | | 0.400 | | 0.375 | | 0.343 | | 0.417 | | 0.400 | | *0.300* | | 0.343 | | 0.375 | | 0.400 | | 0.420 | |

# Building a decision tree

Using the loan borrower data, let's build a decision tree using entropy to measure impurity.

From the root node, there are three possible attributes to split on:

- Home owner: yes, no
- Marital status: 3-way split (single, married, divorced) or a binary split
- Annual income: ≤ $n$, > $n$ (for n to be found)

| ID | Home Owner | Marital Status | Annual Income | Defaulted Borrower |
|---|---|---|---|---|
| 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 |

# Building a decision tree

Root node: No – 7; Yes – 3.

Entropy $= -\frac{7}{10}\log_2\frac{7}{10} - \frac{3}{10}\log_2\frac{3}{10} = 0.881$

| ID | Home Owner | Marital Status | Annual Income | Defaulted Borrower |
|---|---|---|---|---|
| 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** |

# Building a decision tree

Splitting on **home owner**:

Yes: no – 3; yes – 0

entropy = $-\frac{3}{3}\log_2\frac{3}{3}-\frac{0}{3}\log_2\frac{0}{3}=0$

No: no – 4; yes – 3

entropy = $-\frac{4}{7}\log_2\frac{4}{7}-\frac{3}{7}\log_2\frac{3}{7}=0.985$

Weighted entropy = $\frac{3}{10}\times 0+\frac{7}{10}\times 0.985=0.690$

Information gain = $0.881-0.690=0.192$

| ID | Home Owner | Marital Status | Annual Income | Defaulted Borrower |
|---|---|---|---|---|
| 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 |

# Building a decision tree

Splitting on **marital status**: **4 options**

3-way split: **single, married, divorced**

Single: no – 2; yes – 2: entropy = 1

Married: no – 4; yes – 0: entropy = 0

Divorced: no – 1; yes = 1: entropy = 1

Weighted entropy = $\frac{4}{10} \times 1 + \frac{4}{10} \times 0 + \frac{2}{10} \times 1 = 0.6$

Information gain = $0.881 - 0.6 = 0.281$

| ID | Home Owner | Marital Status | Annual Income | Defaulted Borrower |
|---|---|---|---|---|
| 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 |

# Building a decision tree

Splitting on marital status: 4 options

binary split: **single, married/divorced**

Single: no – 2; yes – 2: entropy = 1

Married/divorced: no – 5; yes – 1:

entropy = $-\frac{5}{6}\log_2\frac{5}{6} - \frac{1}{6}\log_2\frac{1}{6} = 0.650$

Weighted entropy = $\frac{4}{10} \times 1 + \frac{6}{10} \times 0.650 = 0.790$

Information gain = $0.881 - 0.790 = 0.091$

| ID | Home Owner | Marital Status | Annual Income | Defaulted Borrower |
|---|---|---|---|---|
| 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 |

# Building a decision tree

Splitting on marital status: 4 options

binary split: **single/married, divorced**

Single/married: no – 6; yes – 2:

entropy = $-\frac{6}{8}\log_2\frac{6}{8} - \frac{2}{8}\log_2\frac{2}{8} = 0.811$

Divorced: no – 1; yes – 1: entropy = 1

Weighted entropy = $\frac{8}{10} \times 0.811 + \frac{2}{10} \times 1 = 0.849$

Information gain = $0.881 - 0.849 = 0.032$

| ID | Home Owner | Marital Status | Annual Income | Defaulted Borrower |
|---|---|---|---|---|
| 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 |

# Building a decision tree

Splitting on marital status: 4 options

binary split: **single/divorced**, **married**

Single/divorced: no – 3; yes – 3: entropy = 1

Married: no – 4; yes – 0: entropy = 0

Weighted entropy = $\frac{6}{10} \times 1 + \frac{4}{10} \times 0 = 0.6$

Information gain = $0.881 - 0.6 = 0.281$

Note: **3-way** and **single/divorced, married** both have gain = 0.281

| ID | Home Owner | Marital Status | Annual Income | Defaulted Borrower |
|---|---|---|---|---|
| 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 |

# Building a decision tree

## Splitting on annual income

| Split | <55 | >55 | <65 | >65 | <72 | >72 | <80 | >80 | <87 | >87 | <92 | >92 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| No | 0 | 7 | 1 | 6 | 2 | 5 | 3 | 4 | 3 | 4 | 3 | 4 |
| Yes | 0 | 3 | 0 | 3 | 0 | 3 | 0 | 3 | 1 | 2 | 2 | 1 |
| Entropy | - | 0.881 | 0 | 0.918 | 0 | 0.954 | 0 | 0.985 | 0.811 | 0.981 | 0.971 | 0.722 |
| **Weighted** | **0.881** | | **0.826** | | **0.764** | | **0.690** | | **0.875** | | **0.846** | |

| Split | <97 | >97 | <110 | >110 | <122 | >122 | <172 | >172 | <230 | >230 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| No | 3 | 4 | 4 | 3 | 5 | 2 | 6 | 1 | 7 | 0 | | |
| Yes | 3 | 0 | 3 | 0 | 3 | 0 | 3 | 0 | 3 | 0 | | |
| Entropy | 1 | 0 | 0.985 | 0 | 0.954 | 0 | 0.918 | 0 | 0.881 | - | | |
| **Weighted** | **0.6** | | **0.690** | | **0.764** | | **0.826** | | **0.881** | | | |

**Information gain 0.281**

# Building a decision tree

Summary

Split marital status: single, married, divorced

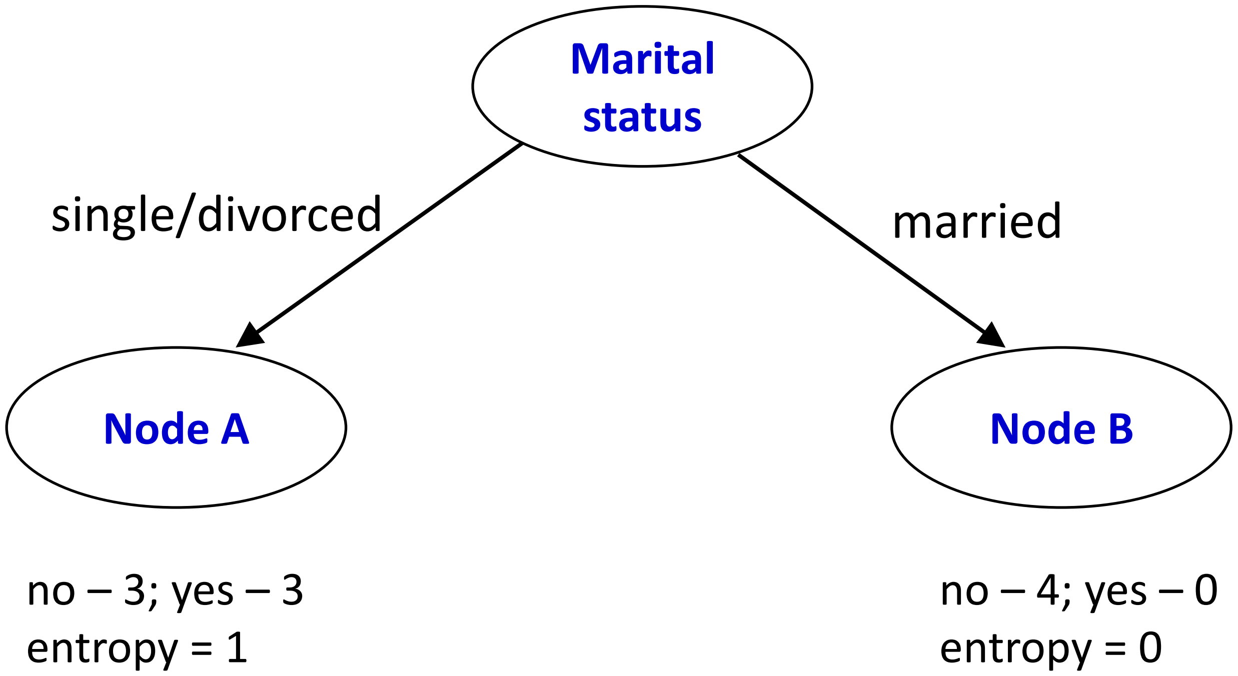
Split marital status: single/divorced, married

Split annual income: $\leq 97$, $> 97$

Each has information gain = 0.281.

Choose: marital status: single/divorced, married

# Building a decision tree: level 1




Node A: split on home owner or income    Node B: leaf node: No

# Node A: single/divorced

| ID | Home Owner | Marital Status | Annual Income | Defaulted Borrower |
|---|---|---|---|---|
| 1 | Yes | Single | 125K | No |
| 3 | No | Single | 70K | No |
| 5 | No | Divorced | 95K | Yes |
| 7 | Yes | Divorced | 220K | No |
| 8 | No | Single | 85K | Yes |
| 10 | No | Single | 90K | Yes |

## Splitting on **home owner**

Yes: no – 2; yes – 0: entropy = 0

No: no – 1; yes – 3: entropy = 0.811

Weighted entropy = $\frac{2}{6} \times 0 + \frac{4}{6} \times 0.811 = 0.541$

Information gain = $1.0 - 0.541 = 0.459$

# Node A: single/divorced

| ID | Home Owner | Marital Status | Annual Income | Defaulted Borrower |
|---|---|---|---|---|
| 1 | Yes | Single | 125K | No |
| 3 | No | Single | 70K | No |
| 5 | No | Divorced | 95K | Yes |
| 7 | Yes | Divorced | 220K | No |
| 8 | No | Single | 85K | Yes |
| 10 | No | Single | 90K | Yes |

Splitting on annual income

Best split is: income ≤ 110, income > 110

≤ 110: no – 1; yes – 3: entropy = 0.811

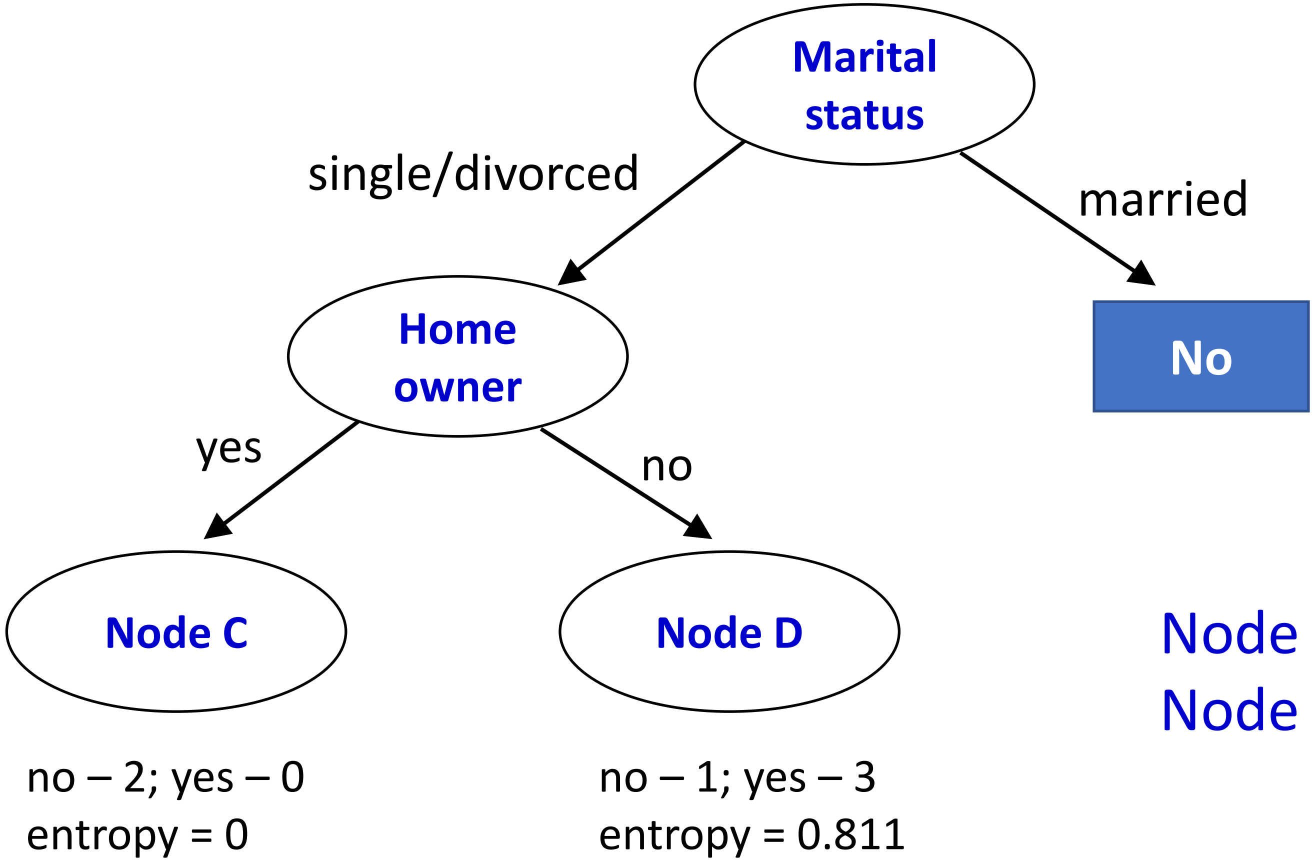
> 110: no – 2; yes – 0: entropy = 0

Weighted entropy = $\frac{4}{6} \times 0.811 + \frac{2}{6} \times 0 = 0.541$

Information gain = $1.0 - 0.541 = 0.459$

Information gain is same splitting on home owner or income; choose home owner

# Building a decision tree: level 2




Node C: leaf node: No
Node D: split on Income

# Node D: split on income

| ID | Home Owner | Marital Status | Annual Income | Defaulted Borrower |
|---|---|---|---|---|
| 3 | No | Single | 70K | **No** |
| 5 | No | Divorced | 95K | **Yes** |
| 8 | No | Single | 85K | **Yes** |
| 10 | No | Single | 90K | **Yes** |

Splitting on annual income

Best split is: income ≤ 77, income > 77

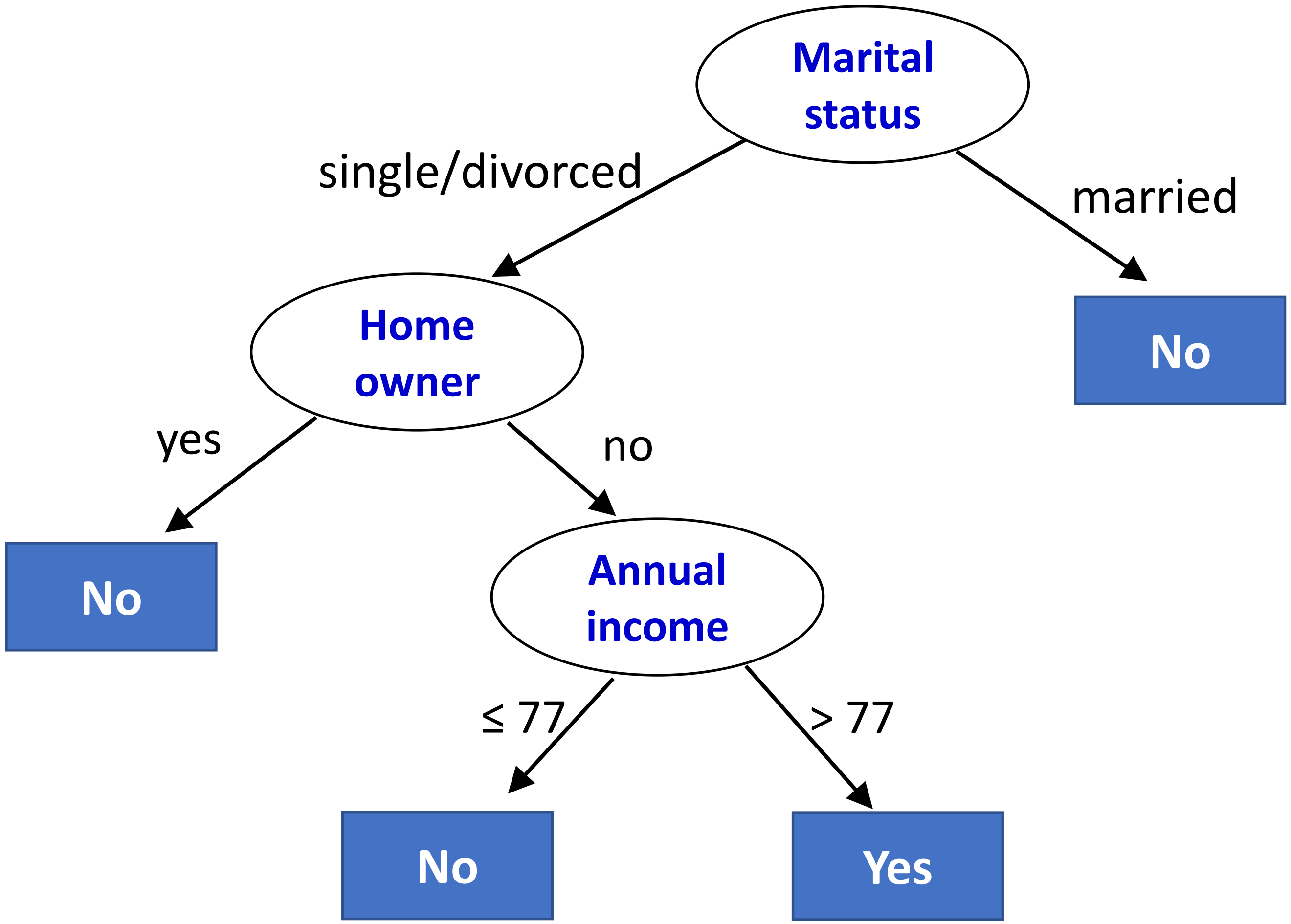
≤ 77: no – 1; yes – 0: entropy = 0

> 77: no – 0; yes – 3: entropy = 0

Weighted entropy = 0; information gain = $0.811 - 0 = 0.811$

This gives the final tree.

# Building a decision tree: the final tree



# Evaluation of Classification Algorithms -Training Set and Test Set

- Machine learning algorithms are evaluated to determine their performance accuracy. Machine learning algorithms performance accuracy depends on how well the are able to predict new unseen data. If the algorithm doesn’t generalize then it will not perform well on new unseen data.

- Two problems are associated with algorithms that do not generalize: Overfitting and Underfitting.

# Overfitting and Underfitting

- Overfitting – the concept in machine learning when the machine learning model fits exactly to its training data. When this happens the model does not perform well on new unseen data.

- Underfitting – the model has not been trained for enough time or the input variables are not significant enough to draw any conclusions between the input and the output values.

# Training set and test set

- To overcome this problem some of the data is used to train the data and some of the  data is used to test the data. This is usually  a split of 80% training data and 20% test data.

- Overfitting, when the data trained does badly on the test set. Accuracy is lower when tested on the test data.

- Underfitting, when the algorithm has poor performance on both the training data and test data.

# K-Fold Cross Validation

K-Fold cross validation is a resampling technique.  The data is split into k sets of training and test folds. The performance is averaged over all sets.


$$= \frac{1}{5}\sum_{i=1}^{5} perfromance_i$$

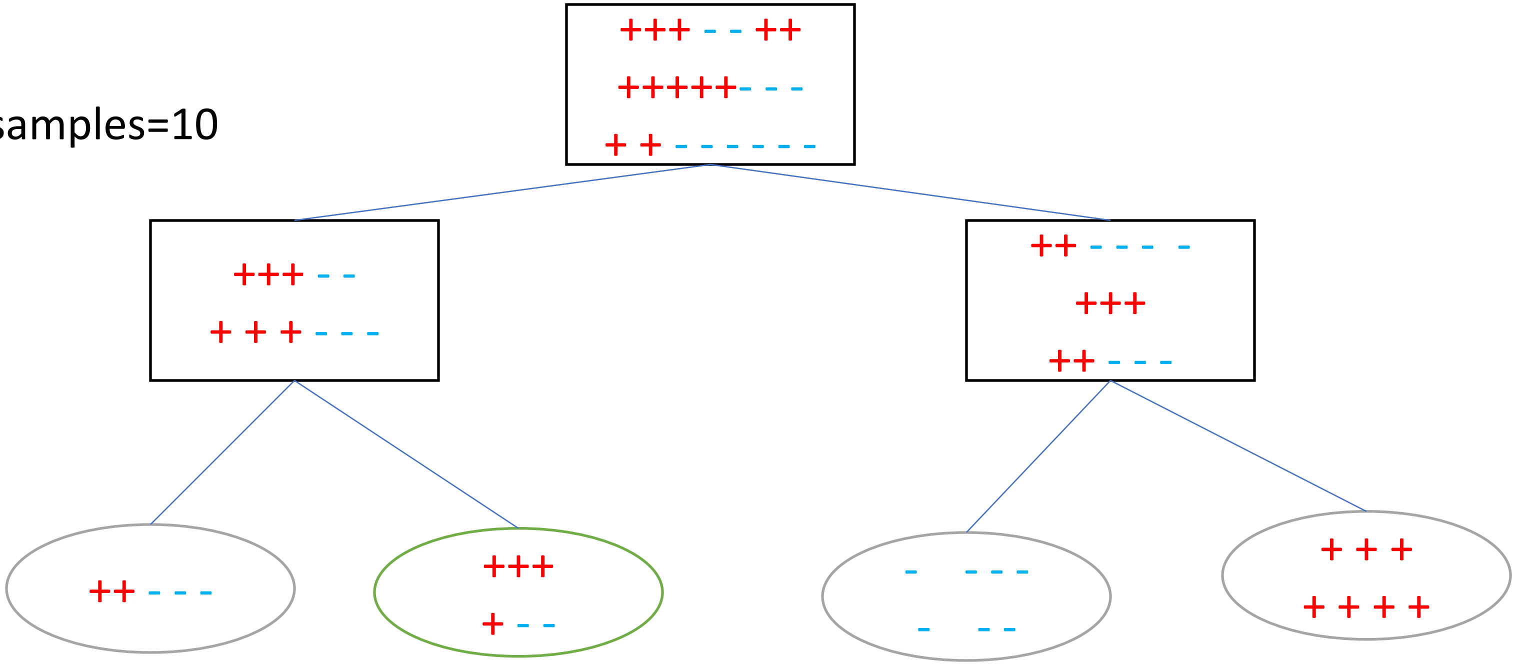
# Optimizing (Pruning) the Decision Tree

- Decision tree optimization  or pruning techniques are used to avoid overfitting and underfitting.

1. Minimum number of samples for a node split
2. Minimum samples for a leaf node
3. Maximum depth of the tree

# Minimum Samples for a Node split

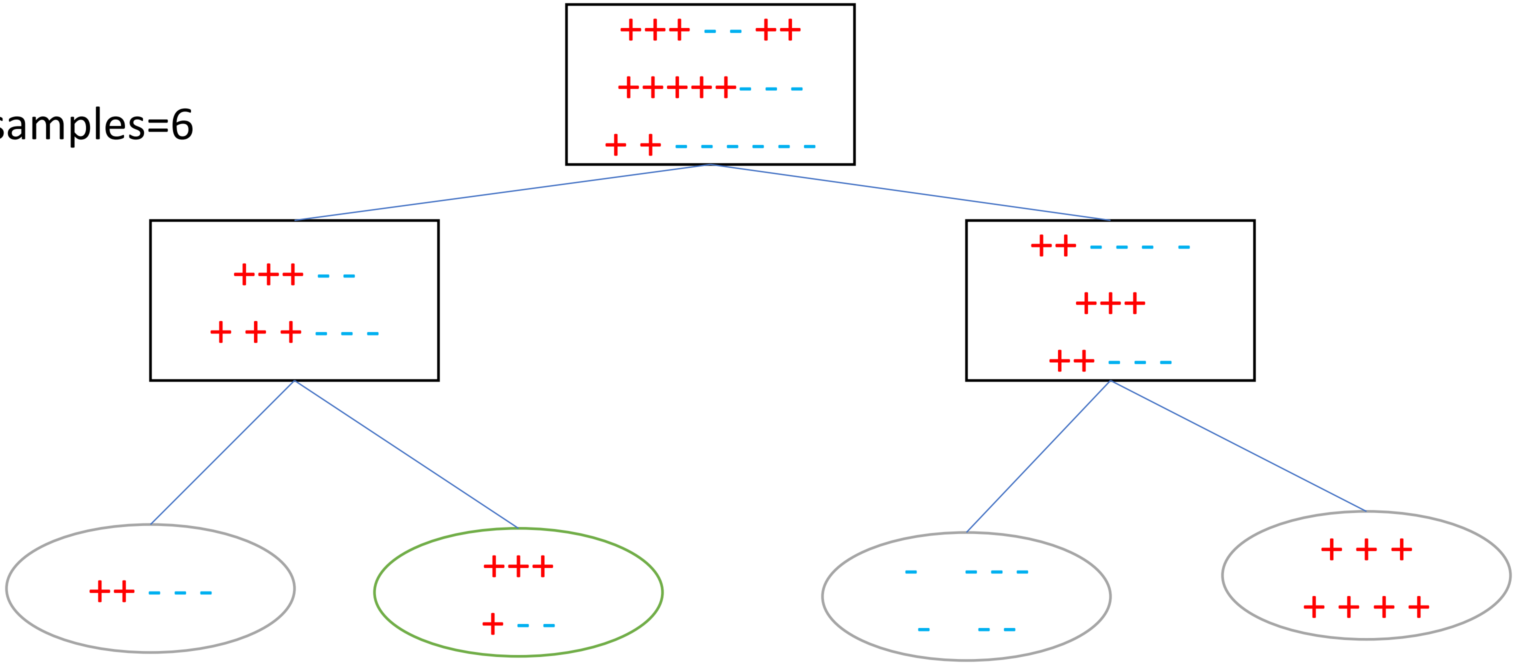
Minimum samples=10




In python the attribute is used with a decision tree classifier
**min_samples_split**

# Minimum Samples for a leaf node

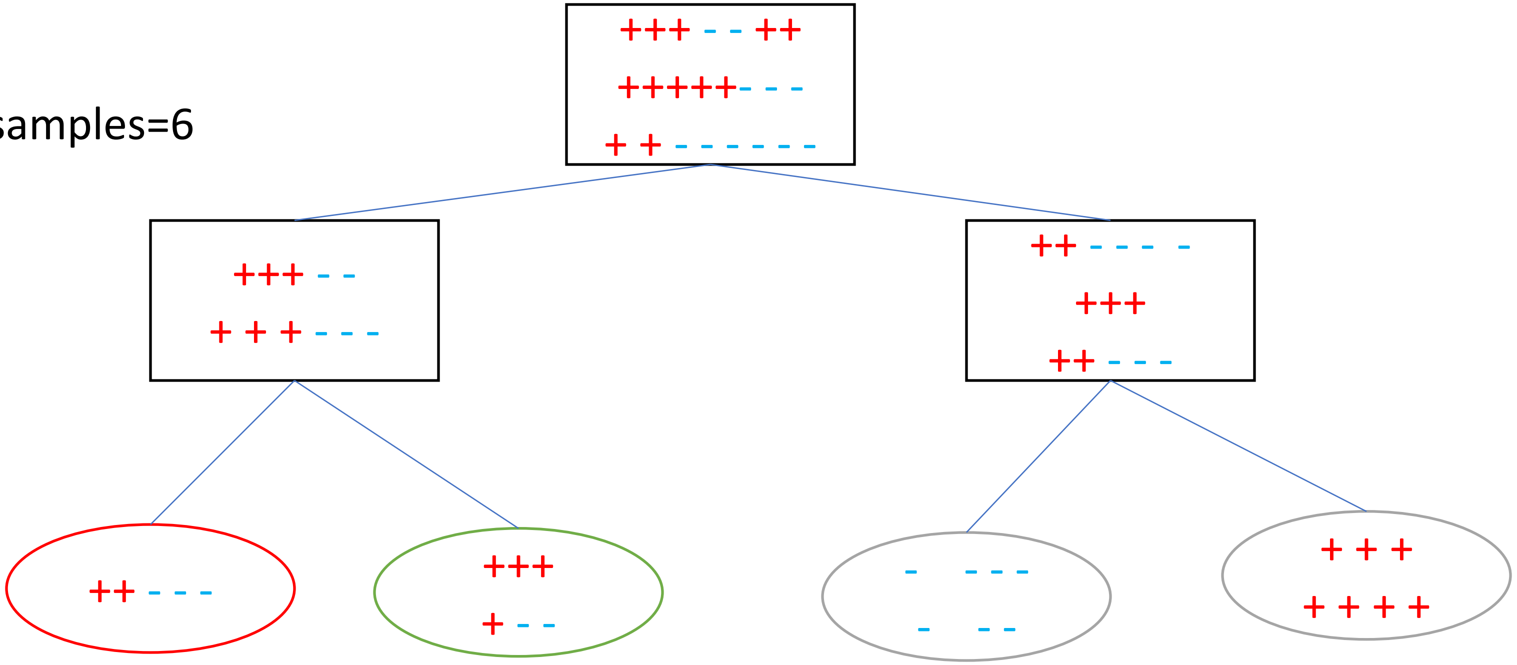
Minimum samples=6




In python the attribute is used with a decision tree classifier
**min_samples_leaf**

# Minimum Samples for a leaf node

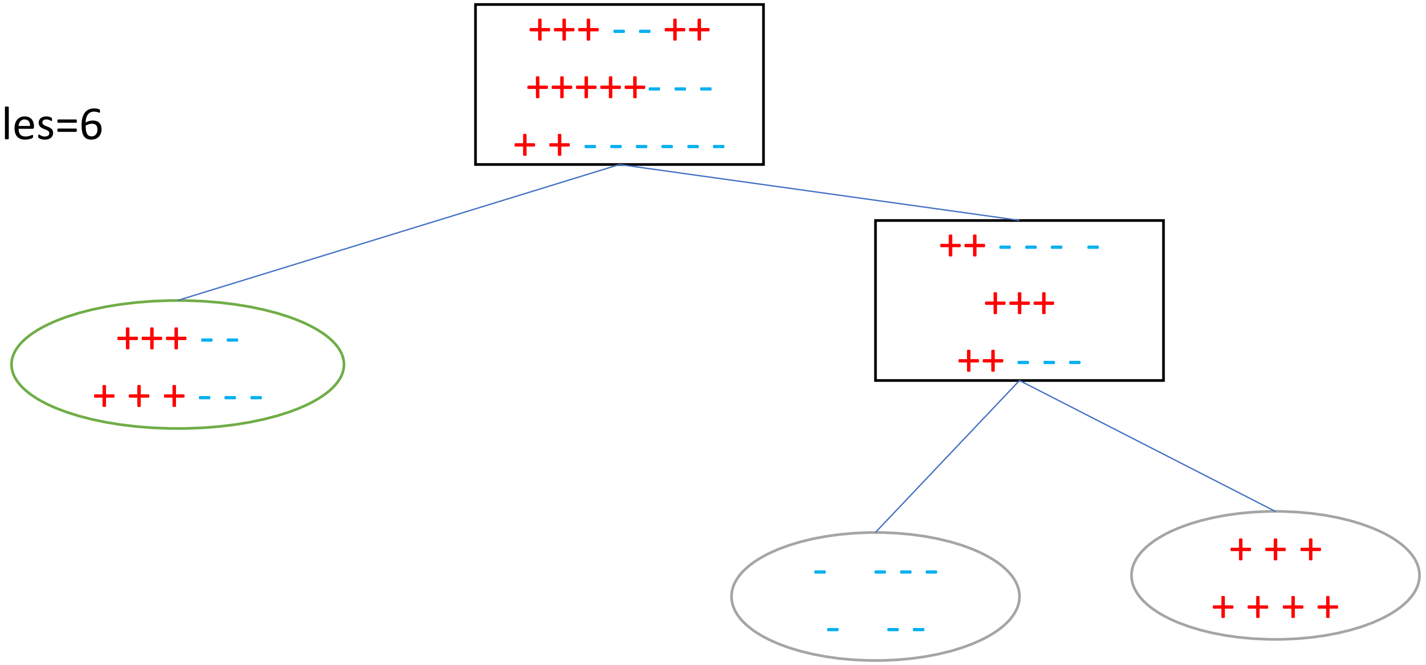
Minimum samples=6




In python the attribute is used with a decision tree classifier
**min_samples_leaf**

# Minimum Samples for a leaf node

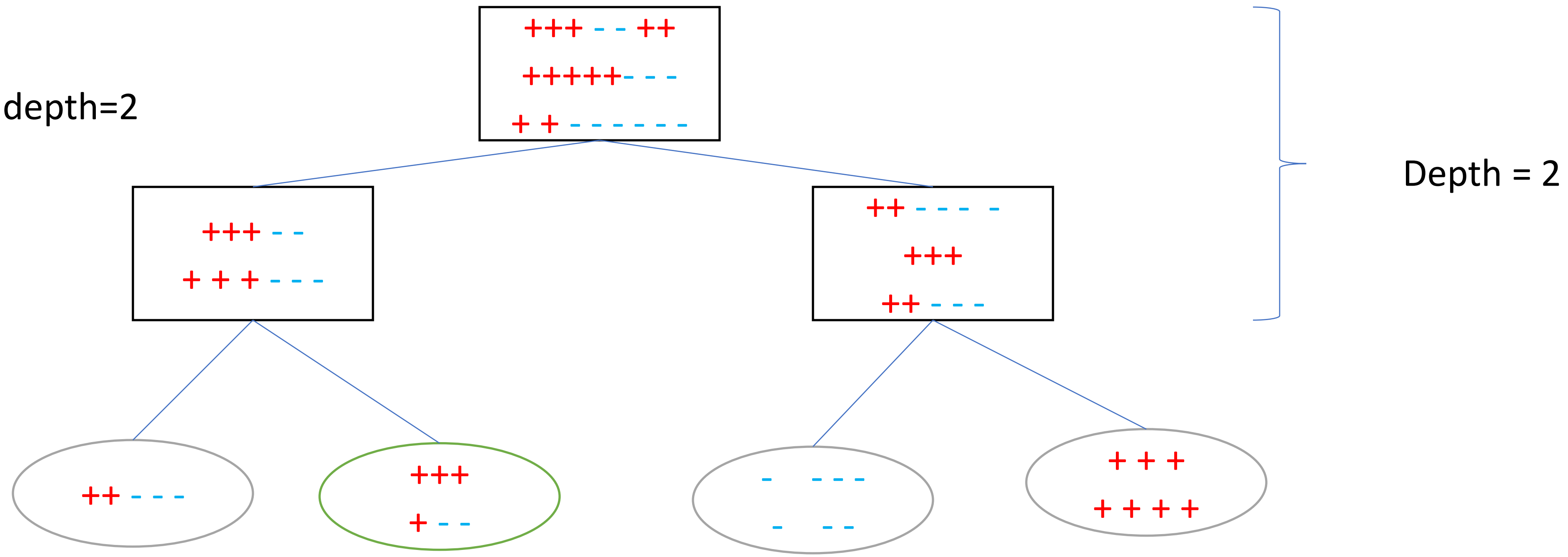
Minimum samples=6




In python the attribute is used with a decision tree classifier
**min_samples_leaf**

# Maximum Depth of the Tree

Maximum depth=2




In python the attribute is used with a decision tree classifier
**max_depth**

# Implementing Decision Trees

Import relevant Python packages used to create a decision tree

- `import pandas as pd`
- `import numpy as np`
- `import matplotlib.pyplot as plt`
- `from sklearn import tree`
- `from sklearn.model_selection import train_test_split`
- `from sklearn.tree import DecisionTreeClassifier`
- `from sklearn import metrics`
- `from sklearn.model_selection import cross_val_score,KFold`

# Pandas

**Reading the data file into pandas using the read_csv() function**

```
data= pd.read_csv('data_file.csv')
```

**Printing first four records**

```
data.head()
```

Assigning the attributes and values of the dataset to variables x and y

y = data[‘class_label’]

X = data.drop([‘class_label’], axis=1)

```
#convert the string values to number
d={'Iris-setosa':0,'Iris-versicolor':1,'Iris-virginica':2}
data['species']=data['species'].map(d)
```

Create a training set with 80%  and test set with 20%  of the samples

- x_train,x_test,y_train,y_test =train_test_split(X,y,test_size=0.20)

Creating an instance of the decision tree classifier

```
dt=DecisionTreeClassifier()
```

# Decision tree classifier parameters

- **criterion***{“gini”, “entropy”, “log_loss”}, default=”gini”*

- **max_depth {***int, default=None}* The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure

- **min_samples_split {***int or float, default=2}* The minimum number of samples required to split an internal node.

- **min_samples_leaf {***int or float, default=1}* The minimum number of samples required to be at a leaf node.

# Decision tree classifier with example parameters

- dt=DecisionTreeClassifier(criterion=‘entropy’,max_depth=3, min_samples_split=6, min_samples_leaf=6)

# Fitting the training data

```
dt.fit(x_train,y_train)
```

# Prediction accuracy

- `y_pred = dt.predict(x_test)`
- `print("Accuracy:",metrics.accuracy_score(y_test, y_pred))`

# K-Fold Accuracy prediction

- kf=**KFold**(n_splits=5)
- score=cross_val_score(dt,X,y,cv=kf)
- print(score.mean())

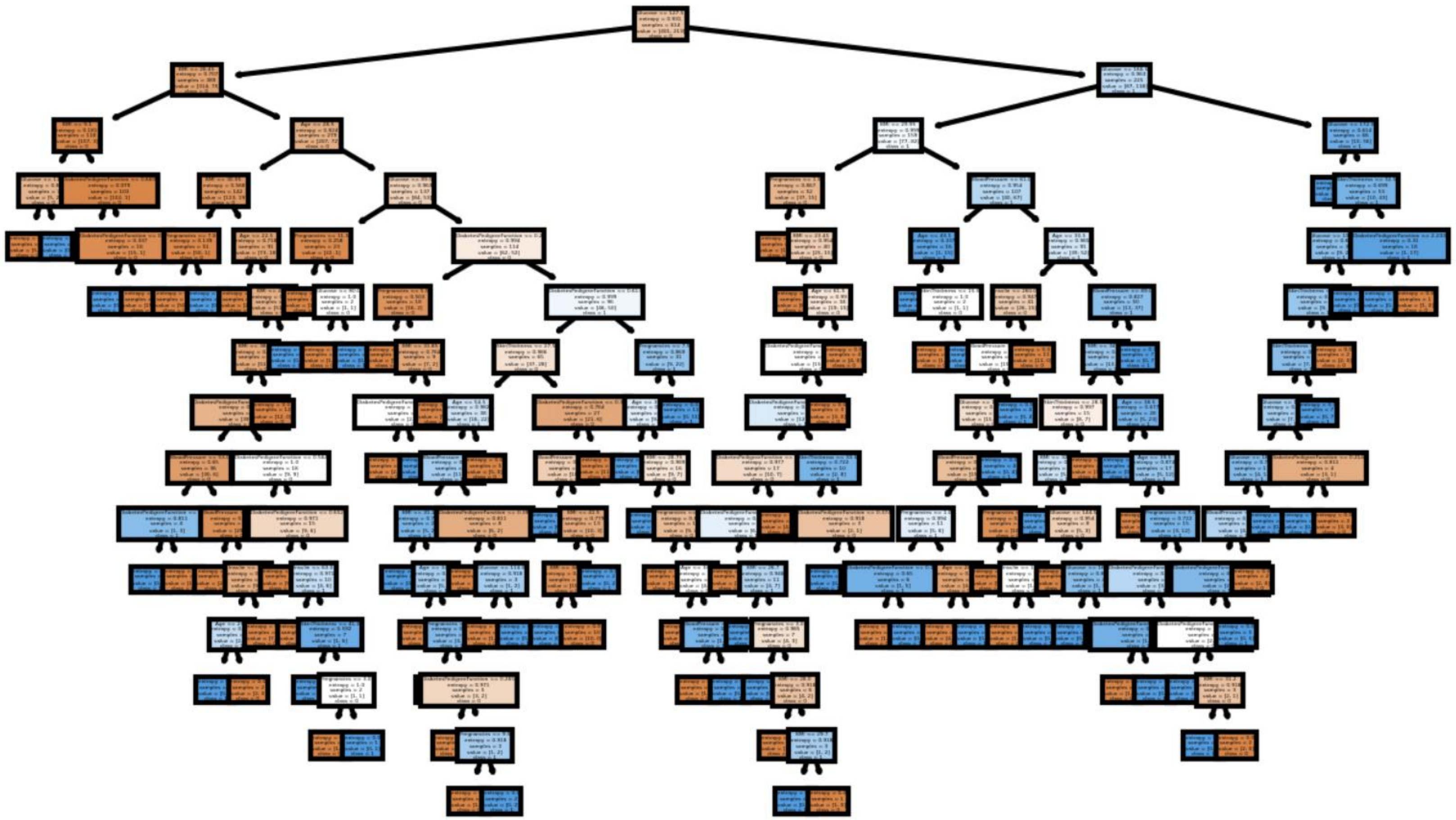
# Displaying the decision tree

```
fig, axes = plt.subplots(nrows = 1,ncols = 1,figsize =
(5,3), dpi=300)

tree.plot_tree(dt,feature_names = [‘A',’B’,’C'],
               class_names=['0','1’,’2’],
               filled = True);

fig.savefig('imagename.png')
```

# Decision Tree before Optimization (pruning)



# Decision Tree after pruning