Geometric Data Analysis

# Decision Trees

MAT 6480W / STT 6705V

Guy Wolf
guy.wolf@umontreal.ca

Université de Montréal
Fall 2019

# Outline

1. Decision Trees
   - Hunt's algorithm
   - Node splitting
   - Impurity measures
   - Decision boundaries
   - Tree pruning
2. Random forests
   - Ensemble of decision trees
   - Randomization approaches
3. Random projections
   - Johnson-Lindenstrauss lemma
   - Sparse random projections

# Decision trees

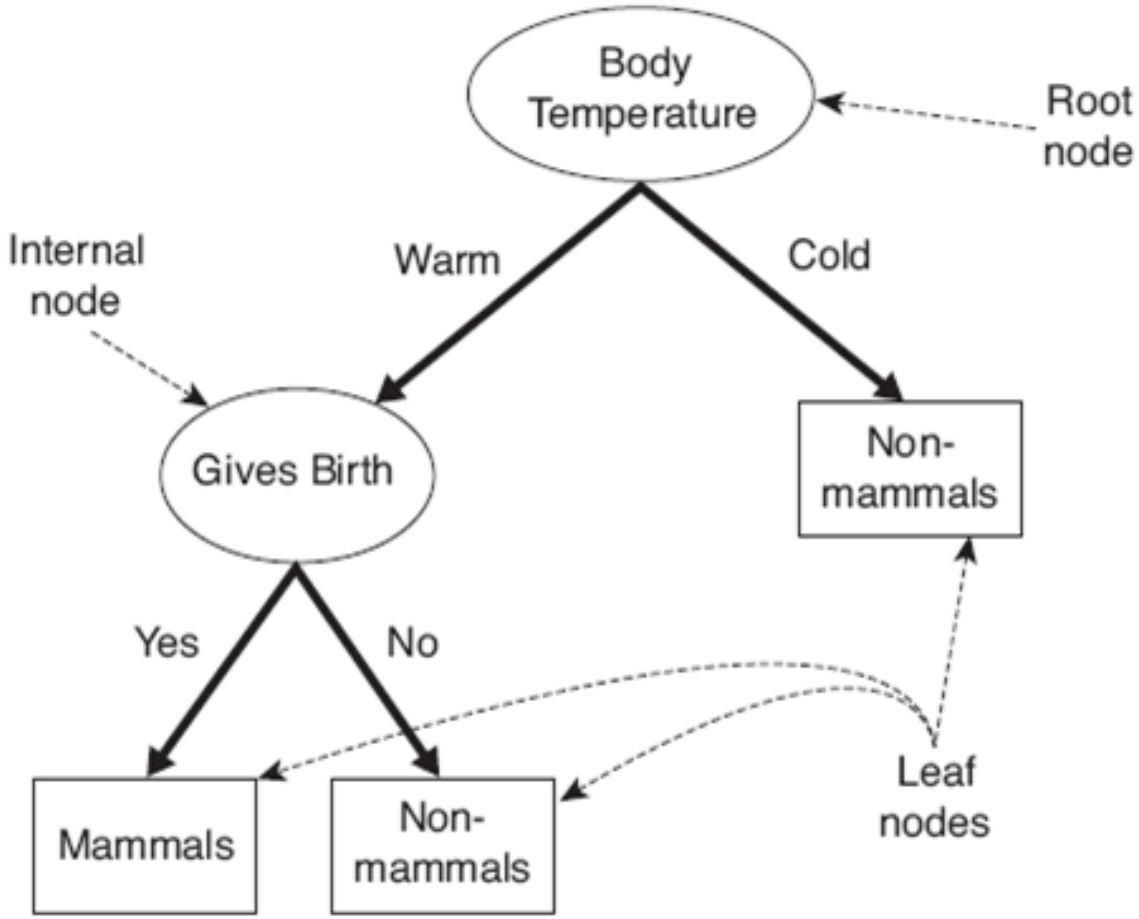
A decision tree is a simple, yet effective model, for classification.

The **tree induction** step essentially builds **a set of IF-THEN rules**, which can be visualized as a tree, for testing class membership of data points.

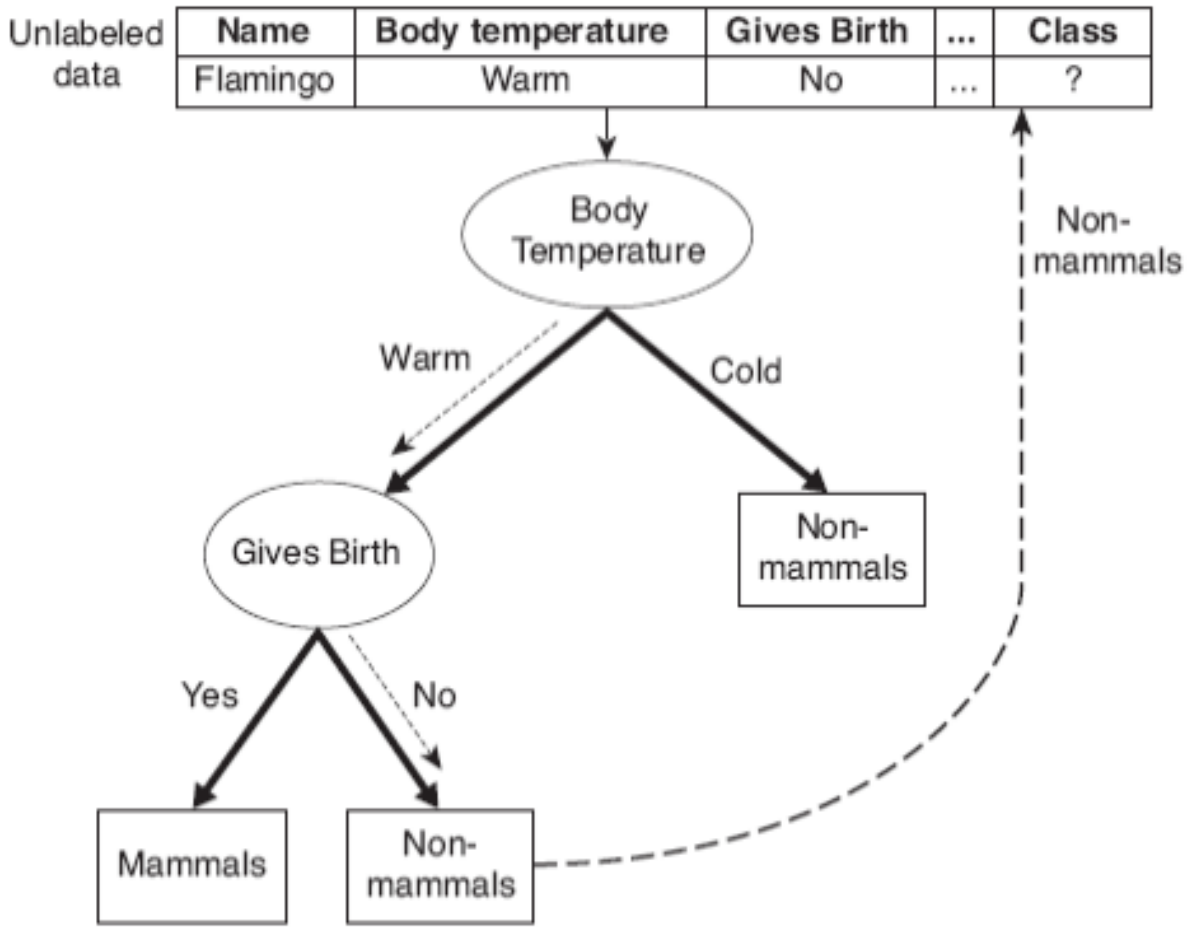
The **deduction** step tests these conditions and **follows the branches of the tree** to establish class membership

Intuitively, this can be thought of as **building an "interview"** for estimating the classification of each data point.

# Decision trees



# Decision trees



Over the years many decision tree (induction) algorithms have been proposed.

## Examples (Decision tree induction algorithms)

- CART (Classification And Regression Trees)
- ID3 (Iterative Dichotomiser 3) & C4.5
- SLIQ & SPRINT
- Rainforest & BOAT

Most of them follow a basic top-down paradigm known as Hunt's Algorithm, although some use alternative approaches (e.g., bottom-up constructions) and particular implementation steps to improve performances.

# Decision trees

## Basic approach (Hunt's algorithm)

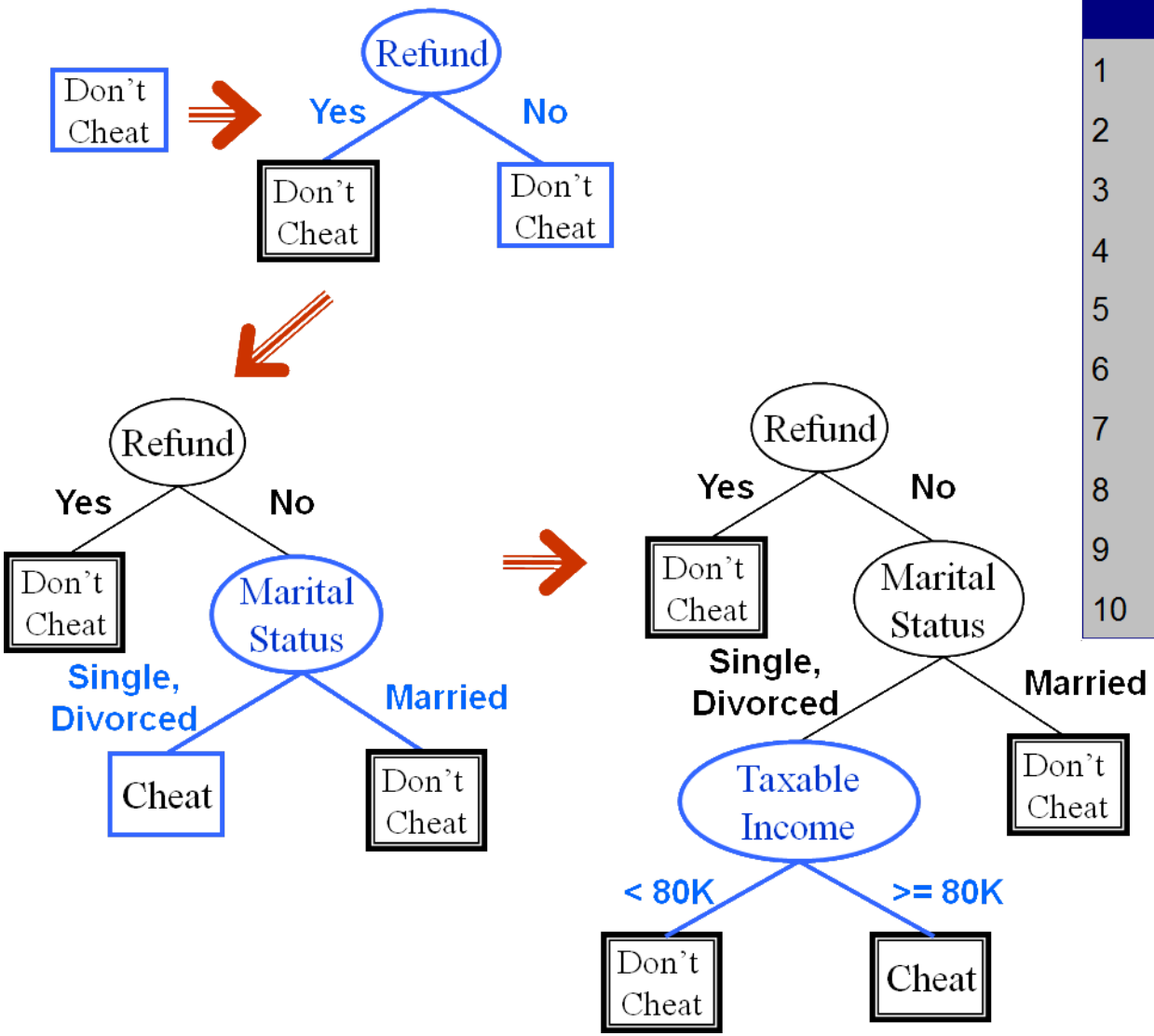
A tree is constructed top-down using a recursive greedy approach:

1. Start with all the training samples at the root
2. Choose the best attribute & split into several data subsets
3. Create a branch & child node for each subset
4. Run the algorithm recursively for each child node and associated subset
5. Stop the recursion when one of the following conditions are met:
   - All the data points in the node have the same class label
   - There are no attributes left to split by
   - The node is empty

If a leaf node contains more than one class label, use majority/plurality voting to set its class.

# Decision trees

## Basic approach (Hunt's algorithm)




| *Tid* | Refund | Marital Status | Taxable Income | Cheat |
|---|---|---|---|---|
| 1 | Yes | Single | 125K | No |
| 2 | No | Married | 100K | No |
| 3 | No | Single | 70K | No |
| 4 | Yes | Married | 120K | No |
| 5 | No | Divorced | 95K | Yes |
| 6 | No | Married | 60K | No |
| 7 | Yes | Divorced | 220K | No |
| 8 | No | Single | 85K | Yes |
| 9 | No | Married | 75K | No |
| 10 | No | Single | 90K | Yes |

# Decision trees

## Node splitting

Each internal node in the tree considers:

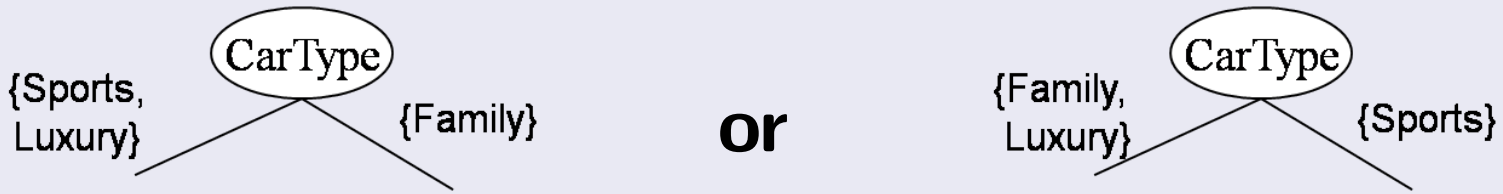
- a subset of the data, based on the path leading to it
- an attribute to test and generate smaller subsets to pass to child nodes

Splitting a node to child nodes depends on the type of the tested attribute and the configuration of the algorithm.

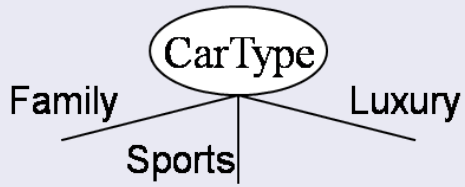
For example, some algorithms force binary splits (e.g.,CART), while others allow multiway splits (e.g, C4.5).

## Splitting nominal attributes:

Binary splits: use a set of possible values on one branch and its complement on the other:

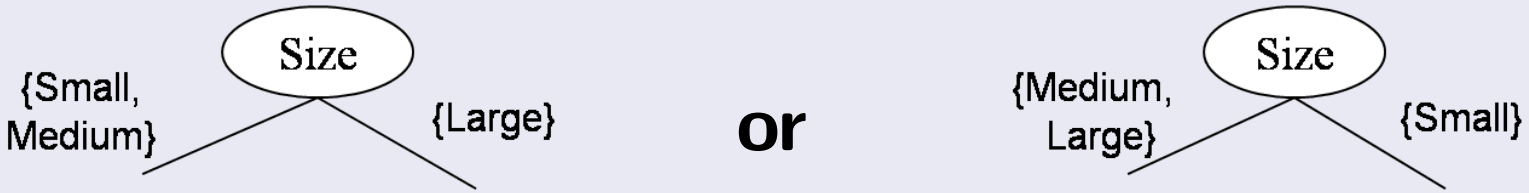



Multiway splits: use a separate branch for each possible value:

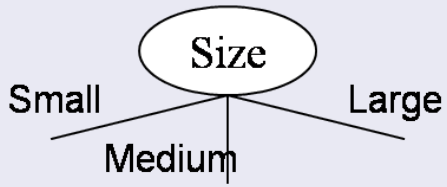


## Splitting ordinal attributes:

Binary splits: find a threshold and partition into values above and below it:

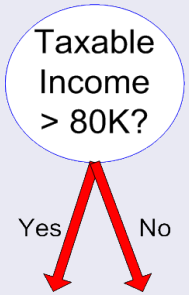



Multiway splits: use a separate branch for each possible value:

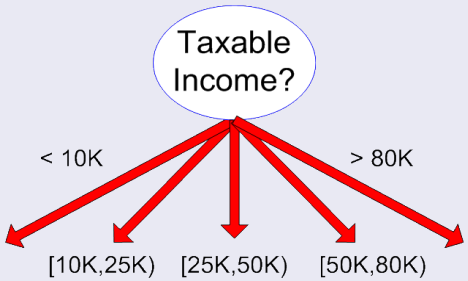


## Splitting numerical attributes:

Binary splits: find a threshold and partition into values above and below it:



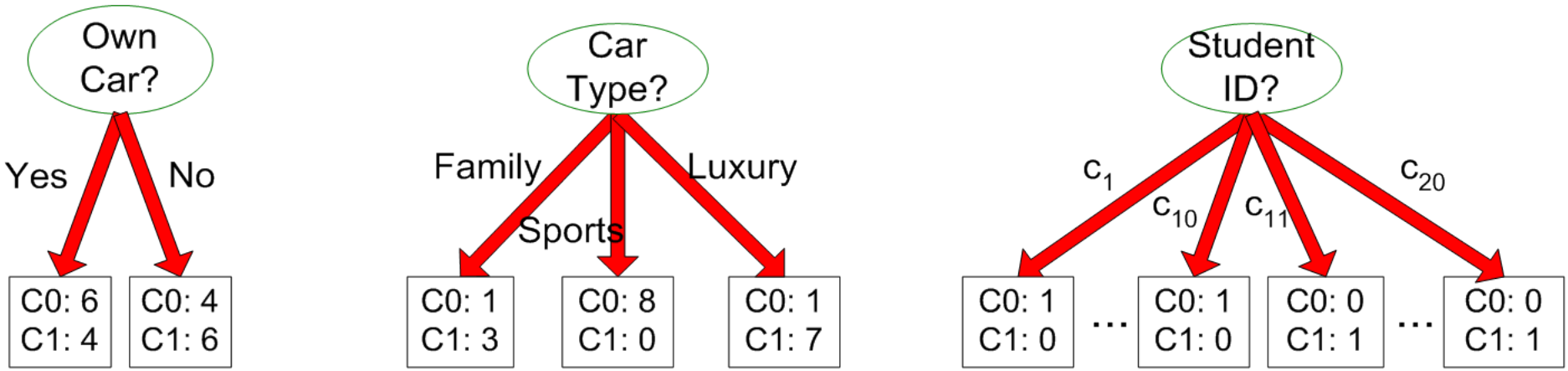

Multiway splits: Discretize the values (statically as preprocessing or dynamically) to form ordinal values:



# Decision trees

## Node splitting

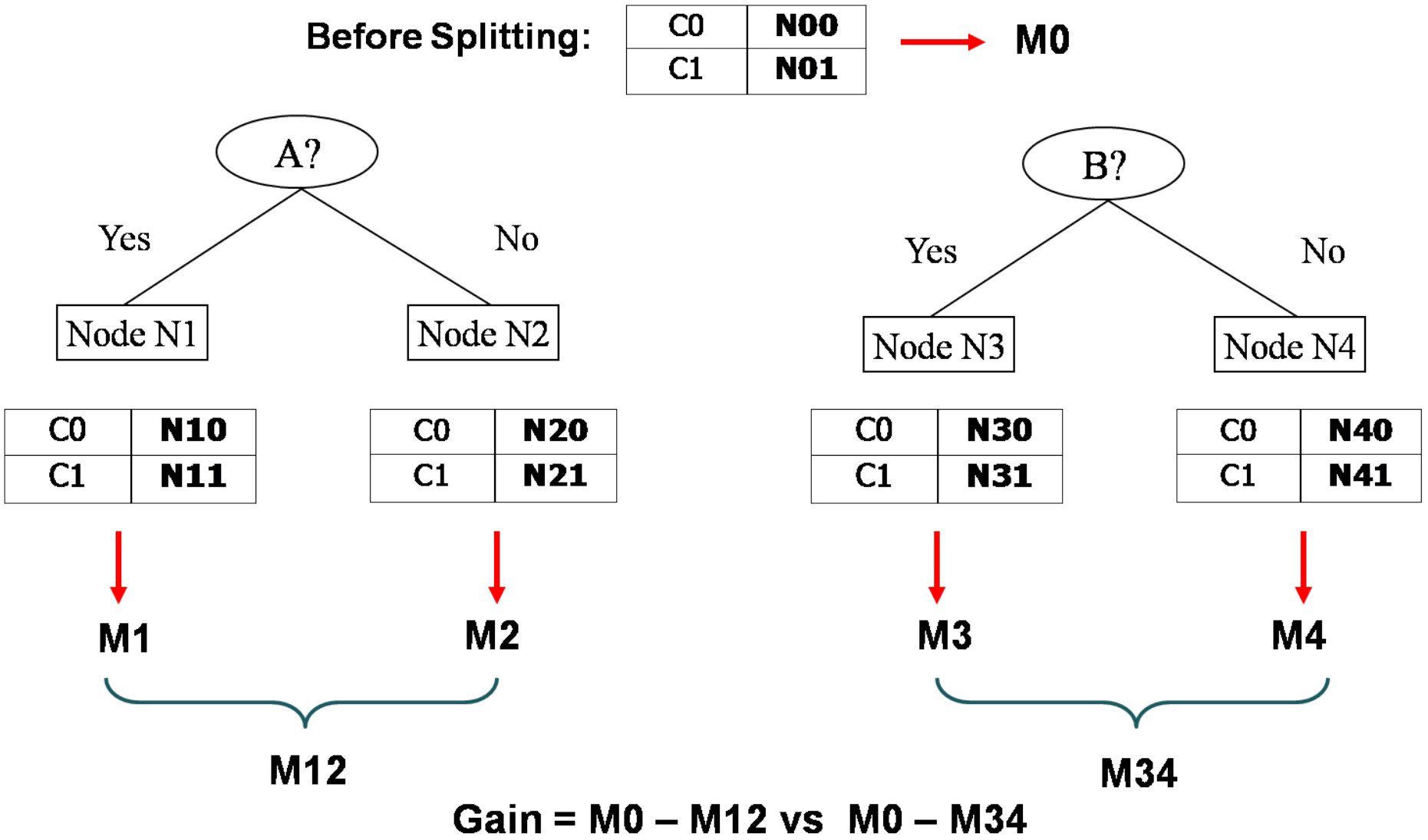
How do we choose the best attribute (and split) to use at each node?




We want to increase the homogeneity and reduce heterogeneity in the resulting subnodes. In other words - we want subsets that are as pure as possible w.r.t. class labels.

# Decision trees

## Impurity measures



Impurity can be quantified in several ways, which vary from one algorithm to another:

## Impurity measures

- Misclassification error
- Entropy (e.g., ID3 and C4.5)
- Gini index (e.g., CART, SLIQ, and SPRINT)

In general, these measures are equivalent in most cases, but there are specific cases when one can be advantageous over others.

Impurity can be quantified in several ways, which vary from one algorithm to another:

## Impurity measures

- Misclassification error
- Entropy (e.g., ID3 and C4.5)
- Gini index (e.g., CART, SLIQ, and SPRINT)

The impurity gain of a split $t \rightarrow t_1, \ldots, t_k$ is the difference

$$\Delta_{\text{Impurity}} = \text{Impurity}(t) - \sum_{i=1} \frac{\#\text{pts}(t_i)}{\#\text{pts}(t)} \text{Impurity}(t_i)$$

between impurity at $t$ and a weighted average of child impurities.

## Misclassification error

The error rate incurred by classifying the entire node by plurality vote:

$$\text{Error}(t) = 1 - \max_c \{p(c|t)\}$$

where $p(c|t)$ is the frequency of class $c$ in node $t$.

- Minimum error is zero - achieved when all data points in the node have the same class
- Maximum error is $1 - \frac{1}{\#\text{classes}}$ - achieved when data points in the node are equally distributed between the classes

## Examples (Misclassification error)

| | |
|---|---|
| C1 | **0** |
| C2 | **6** |

**P(C1) = 0/6 = 0  P(C2) = 6/6 = 1**

**Error = 1 – max (0, 1) = 1 – 1 = 0**

| | |
|---|---|
| C1 | **1** |
| C2 | **5** |

**P(C1) = 1/6  P(C2) = 5/6**

**Error = 1 – max (1/6, 5/6) = 1 – 5/6 = 1/6**

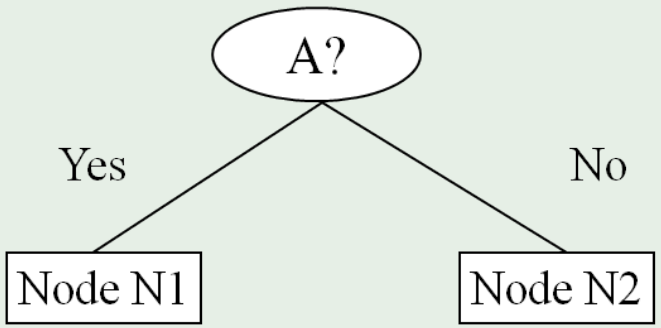
| | |
|---|---|
| C1 | **2** |
| C2 | **4** |

**P(C1) = 2/6  P(C2) = 4/6**

**Error = 1 – max (2/6, 4/6) = 1 – 4/6 = 1/3**

Misclassification error does **not** always detect improvements:

## Example




| | Parent |
|---|---|
| C1 | 7 |
| C2 | 3 |
| Error = 3/10 | |

**Error(N1)**
**= 1 – 3/3 = 0**

**Error(N2)**
**= 1 – 4/7 = 3/7**

| | N1 | N2 |
|---|---|---|
| C1 | 3 | 4 |
| C2 | 0 | 3 |
| Error=3/10 | | |

**Error(Children)**
**= 3/10 • 0 + 7/10 • 3/7**
**= 3/10**

**Error doesn't improve!**

## Entropy

A standard information-theoretic concept that measures the impurity of a node based on the amount "bits" required to represent the class labels in it:

$$\text{Entropy}(t) = -\sum_{c} p(c|t) \log_2 p(c|t)$$

where $p(c|t)$ is the frequency of class $c$ in node $t$.

- Minimum entropy is zero - achieved when all data points in the node have the same class
- Maximum entropy is log(#classes) - achieved when data points in the node are equally distributed between the classes

## Examples (Entropy)

| C1 | 0 |
|---|---|
| C2 | 6 |

P(C1) = 0/6 = 0  P(C2) = 6/6 = 1

Entropy = $-0\log_2 0 - 1\log_2 1 = -0 - 0 = 0$

| C1 | 1 |
|---|---|
| C2 | 5 |

P(C1) = 1/6  P(C2) = 5/6

Entropy = $-(1/6)\log_2(1/6) - (5/6)\log_2(1/6) = 0.65$

| C1 | 2 |
|---|---|
| C2 | 4 |

P(C1) = 2/6  P(C2) = 4/6

Entropy = $-(2/6)\log_2(2/6) - (4/6)\log_2(4/6) = 0.92$

## Information Gain

For a node $t$ split into child nodes $t_1, \ldots, t_k$, the information gain of this split is defined as:

$$\text{Info\_Gain}(t, t1, \ldots, t_k) = \text{Entropy}(t) - \sum_{i=1} \frac{\#\text{pts}(t_i)}{\#\text{pts}(t)} \text{Entropy}(t_i)$$

where $\#\text{pts}(\cdot)$ is the number of data points in a node.

- Measures the reduction in Entropy achieved by the split - an optimal split would maximize this gain.
- Disadvantage: tends to prefer large number of small pure child nodes (e.g., may cause overfitting)

## Gain Ratio

For a node $t$ split into child nodes $t_1, \ldots, t_k$, the gain ratio normalizes the information gain by

$$\text{Split\_Info}(t, t_1, \ldots, t_k) = -\sum_{i=1}^{k} \frac{\#\text{pts}(t_i)}{\#\text{pts}(t)} \log_2 \frac{\#\text{pts}(t_i)}{\#\text{pts}(t)}$$

to get $\text{Gain\_Ratio} = \frac{\text{Info\_Gain}}{\text{Split\_Info}}$.

- Penalizes high-entropy partitions (i.e., with large number of small child nodes)
- Used in C4.5 to overcome the disadvantage of raw information gain.

## Gini index

A "social inequality" (or, more formally, statistical dispersion) index developed by the statistician/sociologist Corrado Gini:

$$\text{Gini}(t) = 1 - \sum_c [p(c|t)]^2$$

where $p(c|t)$ is the frequency of class $c$ in node $t$.

- Minimum Gini value is zero - achieved when all data points in the node have the same class
- Maximum Gini index is $1 - \frac{1}{\#\text{classes}}$ - achieved when data points in the node are equally distributed between the classes

## Examples (Gini index)

| | |
|---|---|
| C1 | 0 |
| C2 | 6 |

P(C1) = 0/6 = 0  P(C2) = 6/6 = 1

Gini = 1 – P(C1)² – P(C2)² = 1 – 0 – 1 = 0

| | |
|---|---|
| C1 | 1 |
| C2 | 5 |

P(C1) = 1/6  P(C2) = 5/6

Gini = 1 – (1/6)² – (5/6)² = 0.278

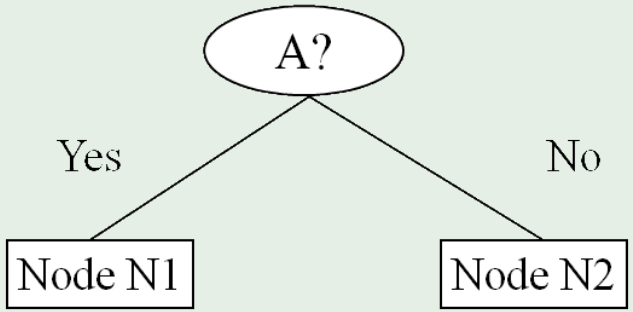
| | |
|---|---|
| C1 | 2 |
| C2 | 4 |

P(C1) = 2/6  P(C2) = 4/6

Gini = 1 – (2/6)² – (4/6)² = 0.444

Gini for a split is computed similarly to misclassification error, but does better:

## Example




| | Parent |
|---|---|
| C1 | 7 |
| C2 | 3 |
| Gini = 0.42 | |

**Gini(N1)**
**= 1 – (3/3)² – (0/3)²**
**= 0**

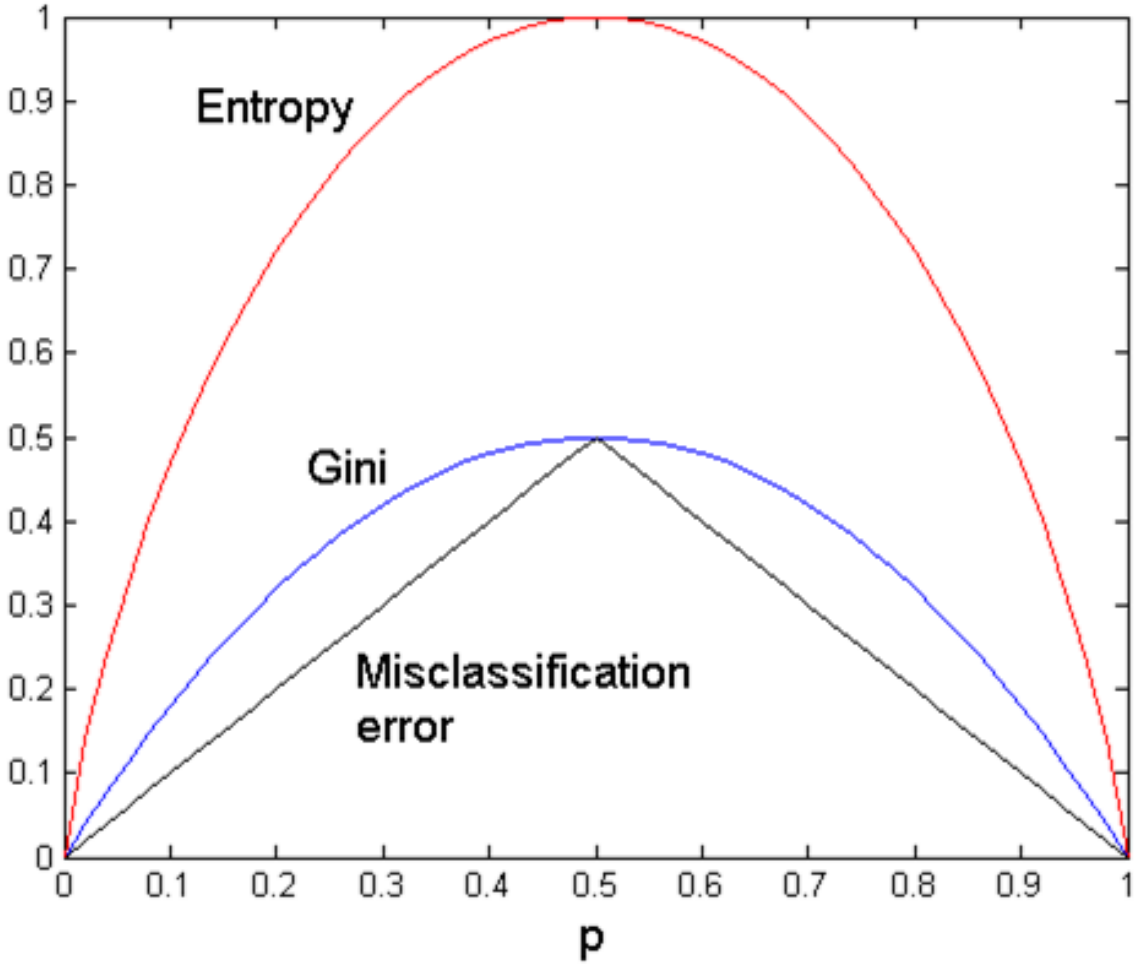
**Gini(N2)**
**= 1 – (4/7)² – (3/7)²**
**= 0.489**

| | N1 | N2 |
|---|---|---|
| C1 | 3 | 4 |
| C2 | 0 | 3 |
| Gini=0.342 | | |

**Gini(Children)**
**= 3/10 * 0**
**+ 7/10 * 0.489**
**= 0.342**

**Gini improves!**

Comparison of the three impurity measures for two classes, where $p$ is the portion of points in the first class (and $1 - p$ in the other class):



# Decision trees

## Impurity measures

Studies have shown the choice of impurity measures has little effect on classification quality. However, each choice has its own bias.

**Information gain:**

- biased towards multivalued attributes

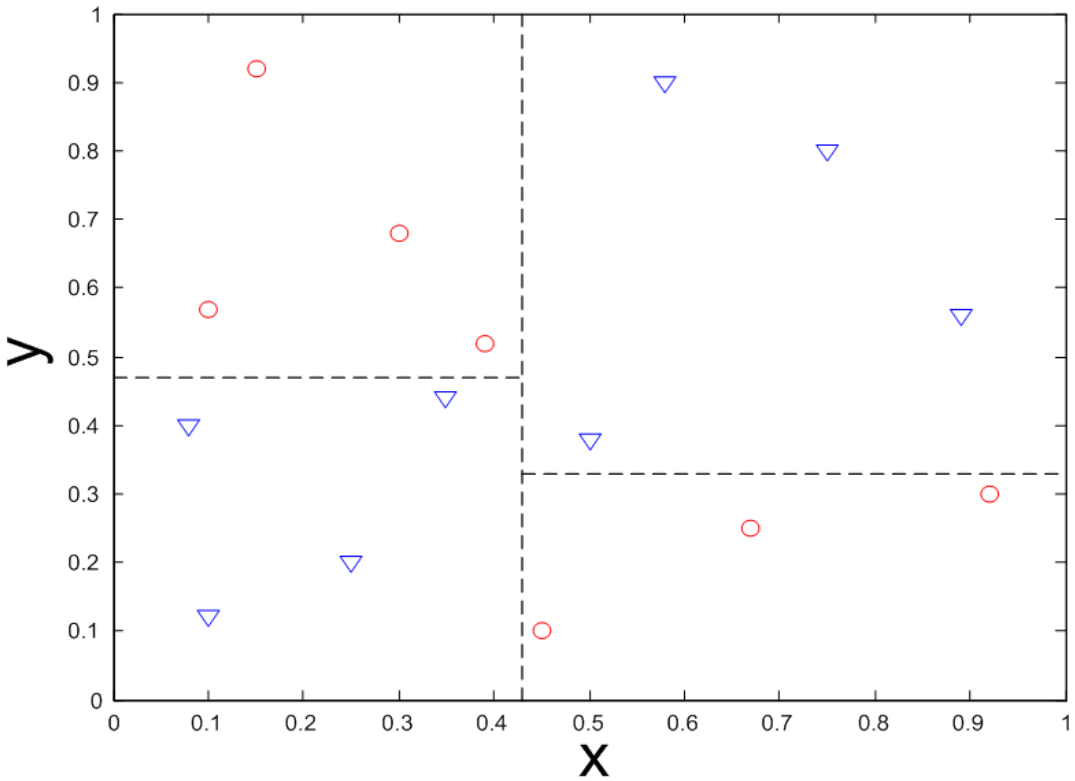
**Gain ratio:**

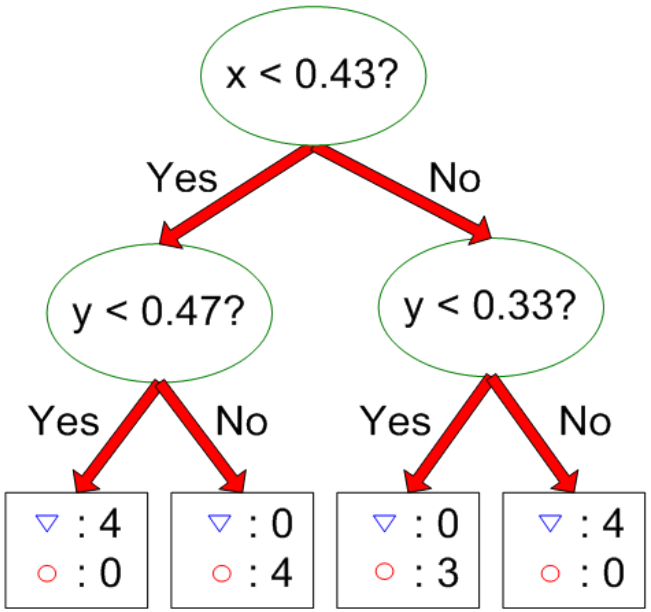
- tends to prefer unbalanced splits where one partition is significantly smaller than others

**Gini index:**

- biased toward multivalued attributes
- has difficulty when #classes is large
- tends to favor equal-sized partitions with equal purity

Decision boundaries show which regions correspond to which class:

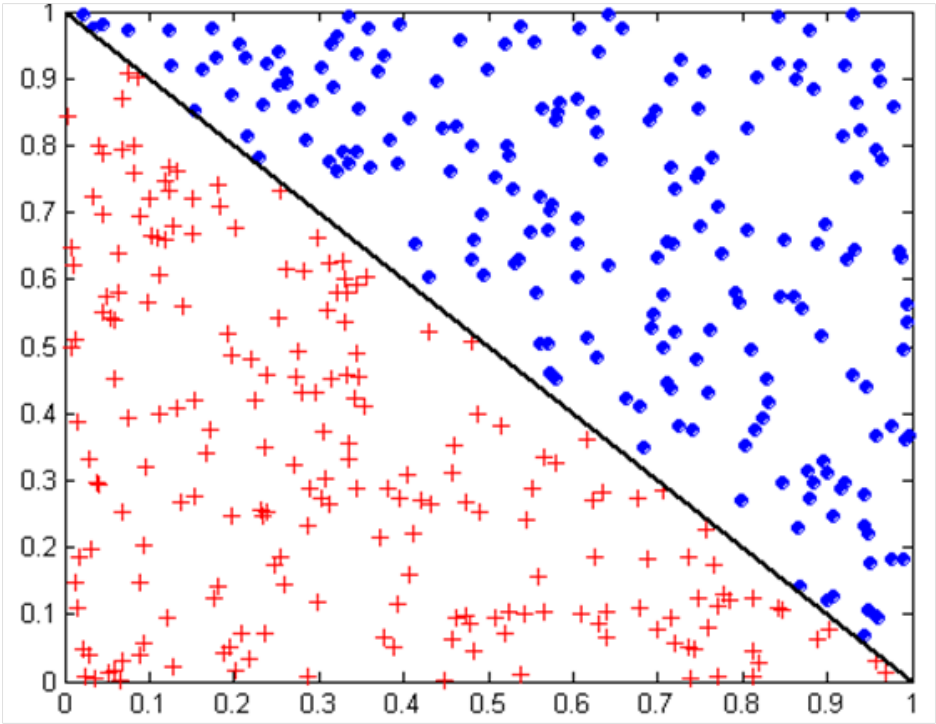


# Decision trees

## Decision boundaries

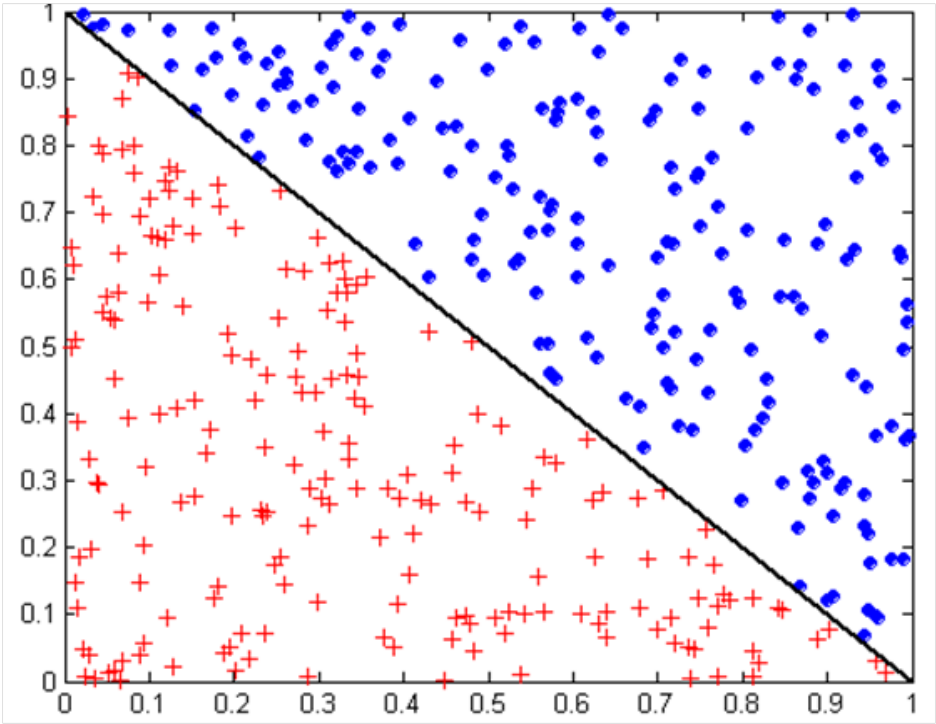
What if we want to consider non-rectangle regions?

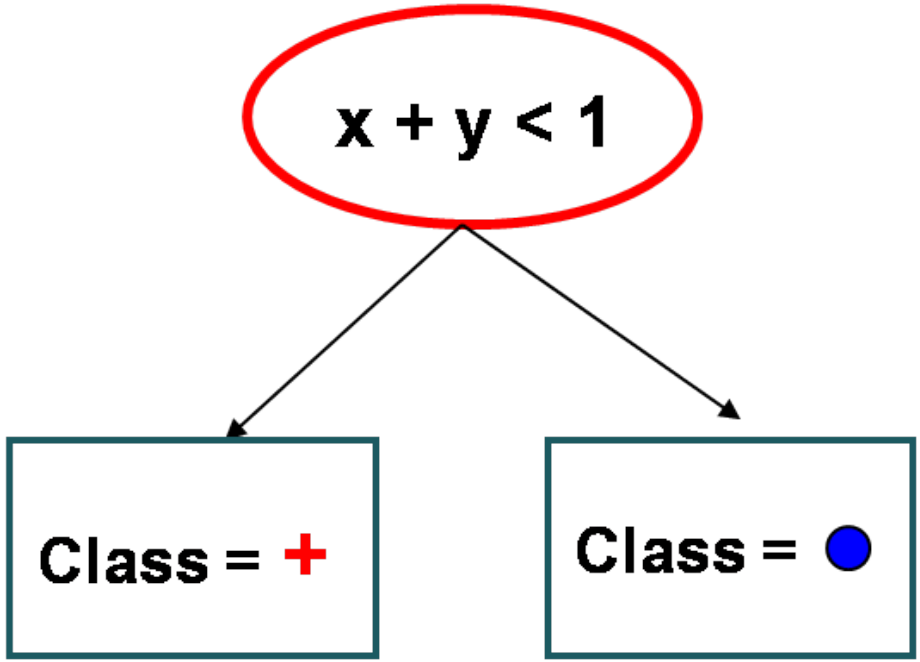


# Decision trees

## Decision boundaries

What if we want to consider non-rectangle regions?




Oblique decision trees consider linear combinations of attributes

# Decision trees

## Tree pruning

Overgrown decision trees can easily overfit the training data and not generalize well.

Model complexity in this case can be quantified by the number of nodes in the tree. To avoid overfitting the training set, the size of the induced decision tree needs to be limited.

Alternatively, we can use the information-theoretic principle of Minimum Description Length (MDL):

### MDL

Minimize $\text{Cost}(\text{data}, \text{model}) = \text{Cost}(\textit{model}) + \text{Cost}(\text{data}|\text{model})$, where the latter cost only considers class labels for misclassified data points.

# Decision trees

## Tree pruning

Decision tree size is reduced by pruning:

### Prepruning

Stop the tree learning algorithm before the tree is fully grown using restrictive stopping conditions, such as:

- gain in impurity is smaller than a given threshold
- size of considered subset is smaller than a threshold

### Postpruning

First learn a full tree, and then trim it using the following operations:

- Subtree replacement - trim a subtree and replace it with a leaf
- Subtree raising - use the most traversed branch at a node, raise its subtree by one level, and eliminate the sibling subtrees.

# Random Forests

## Ensemble of decision trees

While decision trees are simple and effective, they are also sensitive to training variations and overfitting. To reduce variance and increase stability, several decision trees can be combined together to form a forest:

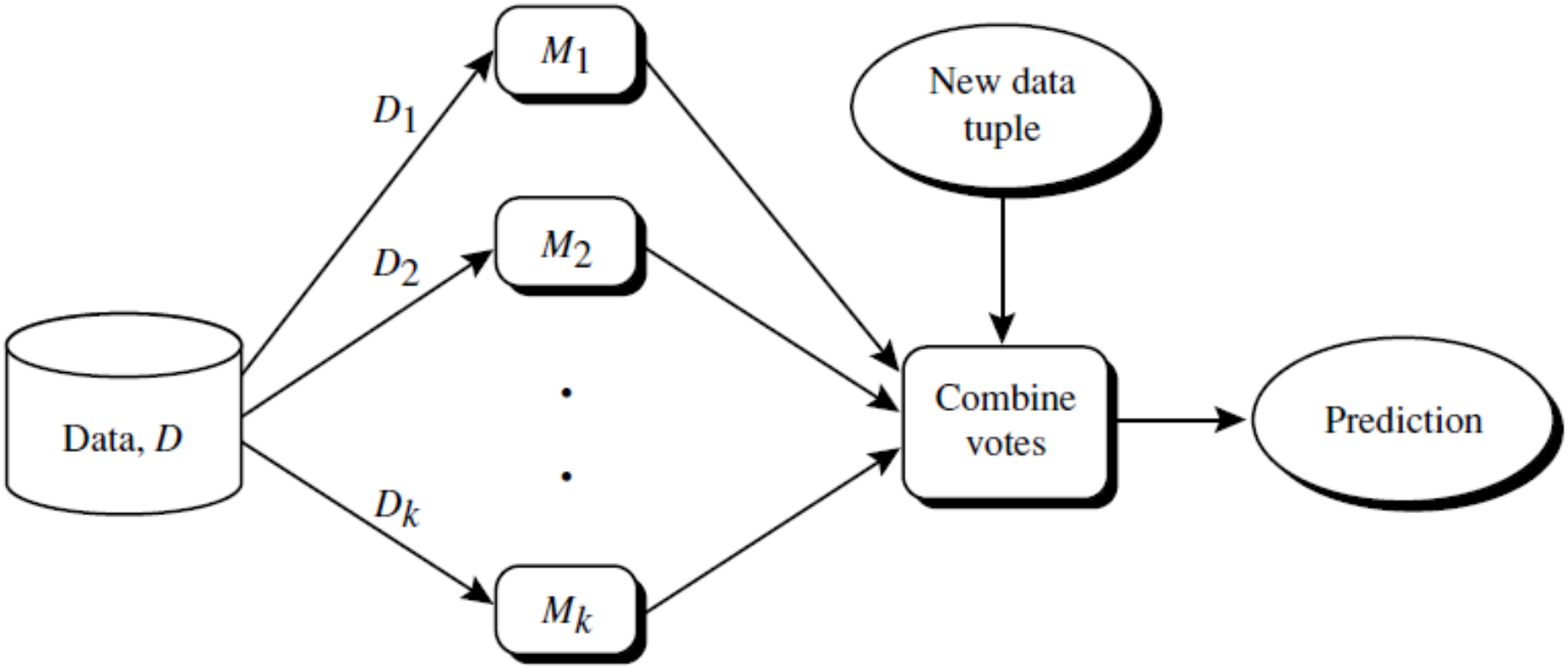
### Random Forest

An ensemble method that combines together several decision trees and aggregates their results. To build the trees, several random vectors are sampled i.i.d. from the same distribution and each individual decision tree construction depends on the data and on one of these vectors.

Random forests are more computationally efficient than other ensemble methods, while having comparable accuracy.

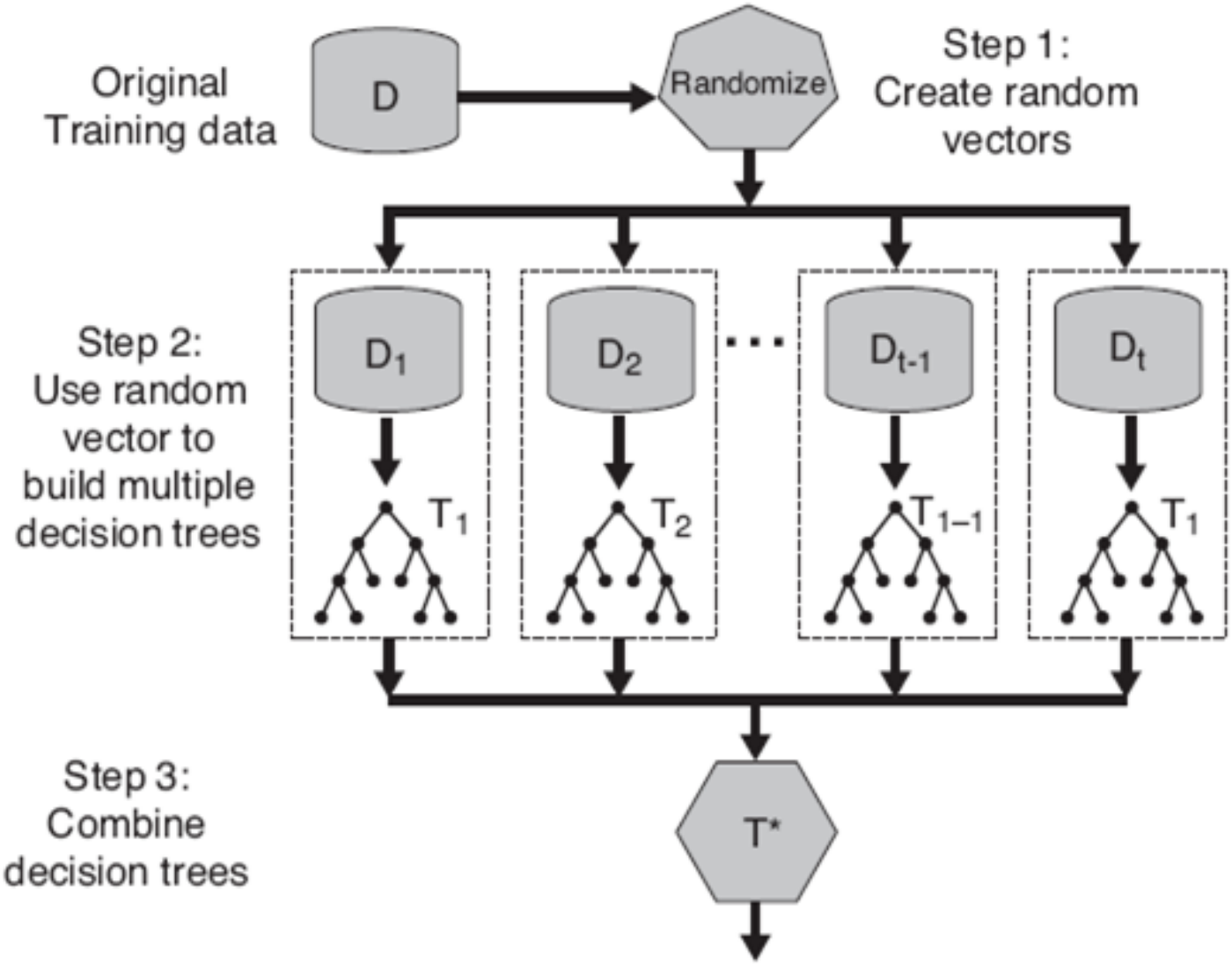
# Random Forests

## Ensemble of decision trees



# Random Forests

## Ensemble of decision trees



# Random Forests

## Tree randomization approaches

Decision tree inputs can be randomized in two ways:

**Random input selection (Forest-RI):**

Randomly select a subset of attributes as candidates for splitting nodes instead of using all the attributes in the data. This random selection is typically done separately at each node.

**Random linear combinations (Forest-RC):**

Creates new attributes that are (random) linear combination of existing attributes. This reduces the correlation between individual classifiers, but also allows nonrectangular decision boundaries.

The latter can be related to random projections, which are also used in other tasks (e.g., $k$NN search and dimensionality reduction).

# Random Projections

Random projections (RP) are a popular an efficient way to project (numerical) data into a space of arbitrary dimensions, without having to learn the embedding function from the data.

For a dataset of $n$ data points in $d$ dimensions, a typical construction uses the following steps:

- Choose target dimension $k > 0$, draw $k \cdot d$ samples i.i.d. from a given distribution, and organize them in a matrix $R \quad \mathbb{R}^{d \times k}$
- Scale $R$ by some factor to get matrix $A$ (e.g., $A = k^{-1/2}R$)
- Organize the data points as rows of a matrix $X \quad \mathbb{R}^{n \times d}$ and embed via matrix multiplication $XA \quad \mathbb{R}^{n \times k}$

Random forests (FC) can be regarded as building decision trees on multiple instantiations of RP.

# Random Projections

## Johnson-Lindenstrauss lemma

The motivation and reasoning behind RP come from the famous JL lemma and its proof (omitted as out of scope for this class):

### Lemma (Johnson-Lindenstrauss, 1984)

*Let* $X \quad \mathbb{R}^d$ *be a finite dataset of size* $|X| = n$. *Given any arbitrary* $0 < \varepsilon < 1$ *and* $k > 8\frac{\ln(n)}{\varepsilon^2}$, *there exist a linear embedding of* $X$ *into* $\mathbb{R}^k$ *such that* $(1 - \varepsilon) \leq \frac{f(x)-f(y)\ ^2}{x-y\ ^2} \leq (1 + \varepsilon)$.

### Lemma (Alternative version - distributional JL lemma)

*Given arbitrary* $0 < \varepsilon, \delta < \frac{1}{2}$ *and* $k > \frac{C}{\varepsilon^2}\ln(\frac{1}{\delta})$, *where* $C$ *is a constant, let* $R$ *be a* $k \times d$ *matrix s.t.* $R_{ij} \overset{i.i.d.}{\sim} N(0,1)$, *Then, for* $A = \frac{1}{\sqrt{k}}R$ *and for any* $x \quad \mathbb{R}^d$ *with unit norm,* $\Pr[|\ Ax\ ^2 - 1| \leq \varepsilon] \geq (1 - \delta)$.

# Random Projections

## Sparse random projections

There are several ways to construct a random projection matrix, e.g.:

### Example (Achlioptas 2001)

A simple yet effective scheme to the projection matrix is by randomly (i.i.d.) drawing the matrix $R$ (recall $A = \sqrt{k}^{-1} R$) from:

$$R_{ij} = \begin{cases} +\sqrt{s} & \text{with probability } \frac{1}{2s} \\ 0 & \text{with probability } (1 - \frac{1}{s}) \\ -\sqrt{s} & \text{with probability } \frac{1}{2s} \end{cases}$$

where $s > 0$ controls the sparsity of the constructed matrix.

Other construction allow sparser matrices, or alternatively, enforce orthogonality for proper projection.

# Summary

Decision trees are a family of popular simple classifiers

- Induction algorithm design a hierarchical partitioning of the data to homogeneous subsets that mostly consist of a single class
- Classification of new data points is done by following tree branches and majority vote in leaf nodes

Overfitting and high variance are common with decision trees, but can be alleviated by pruning and randomization

Random forests are a popular example of ensemble classification

- They leverage multiple trees to obtain robust and flexible decision boundaries
- Randomization is achieved by random feature selection or projection prior to each decission tree construction