Decision Trees

Decision Trees

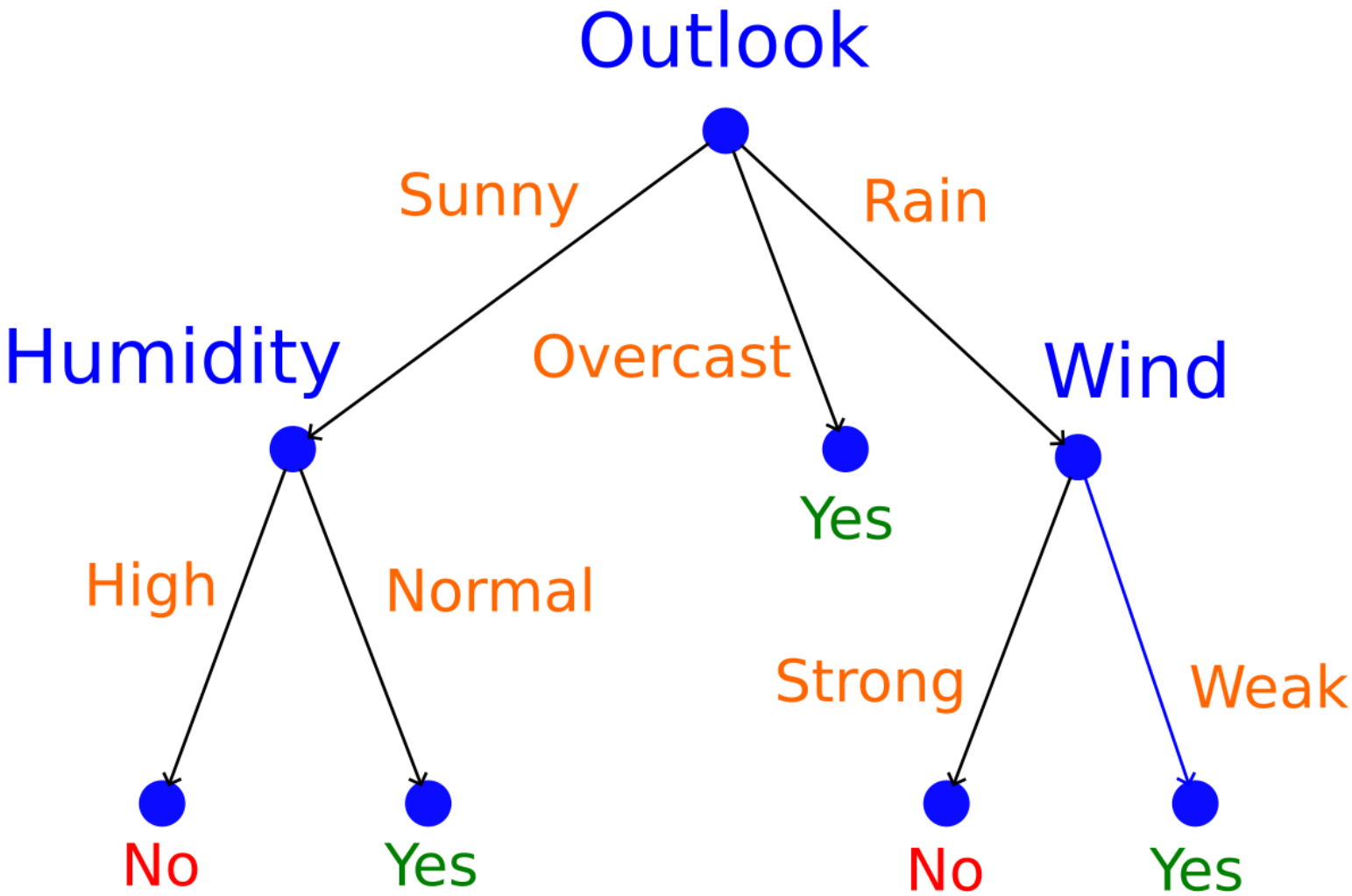
- One of the widely used methods for machine learning.
- Intuitively simple, directly explainable.
- Basis for random forests (a powerful model).

Decision Trees

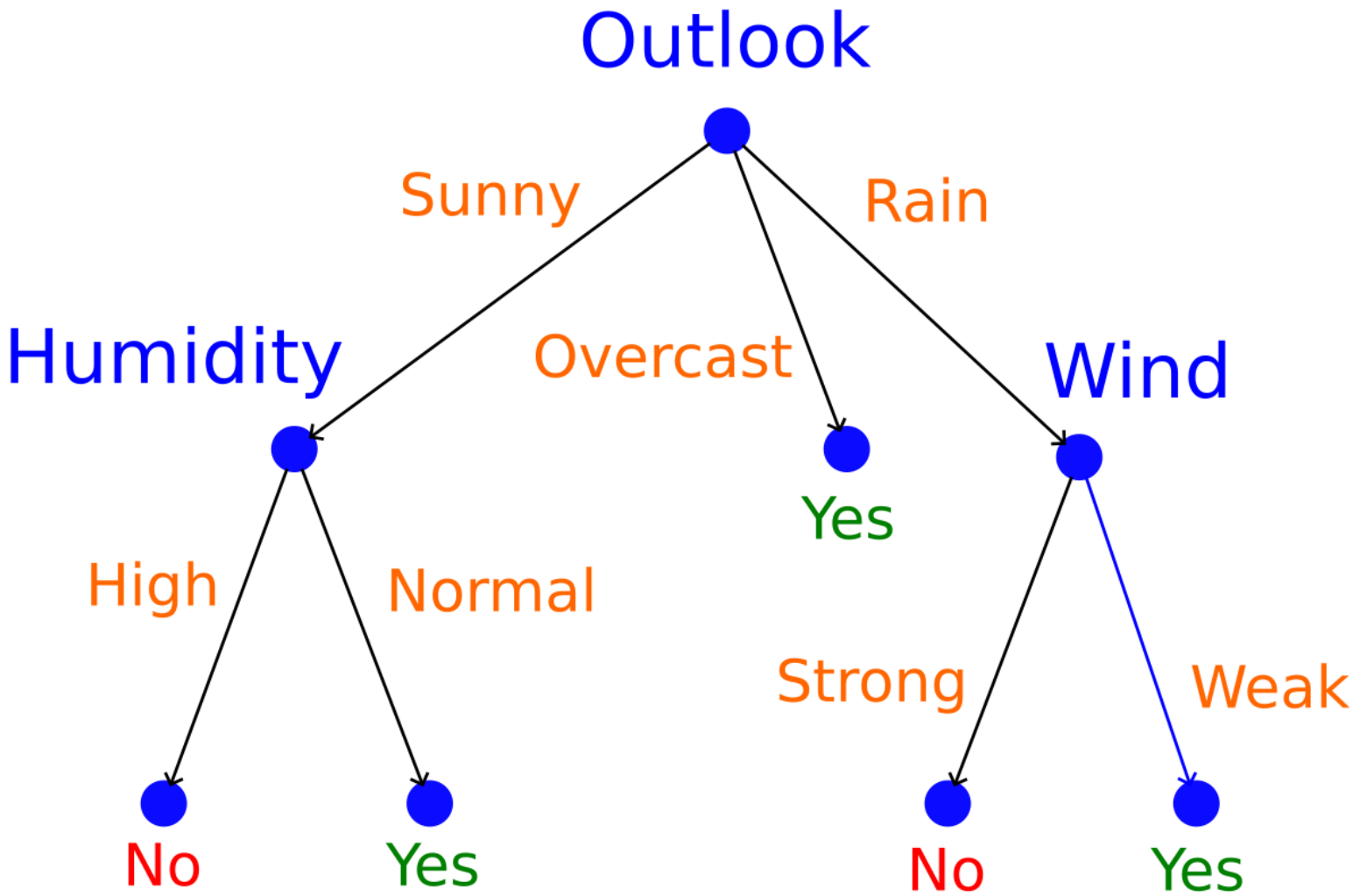
- One of the widely used methods for machine learning.
- Intuitively simple, directly explainable.
- Basis for random forests (a powerful model).

- We will consider the ID3 algorithm.
  Quinlan, 1979
- Various adjustments that appear in C4.5, CART, etc.

Consider the weather forecast for tennis playing. How would you decide whether to play today?



Consider the weather forecast for tennis playing. How would you decide whether to play today?



How do we obtain such a tree based on experience/data?

Learning Decision Trees

Consider data represented as follows:

- A finite set of *attributes* $\mathcal{A} = \{A_1, \ldots, A_n\}$.
- Each attribute $A \in \mathcal{A}$ has its *set of values* $V(A)$.

We start with trees on discrete datasets, that is, assume $V(A)$ finite for all $A \in \mathcal{A}$.

Learning Decision Trees

Consider data represented as follows:

- A finite set of *attributes* $\mathcal{A} = \{A_1, \ldots, A_n\}$.
- Each attribute $A \in \mathcal{A}$ has its *set of values* $V(A)$.

We start with trees on discrete datasets, that is, assume $V(A)$ finite for all $A \in \mathcal{A}$.

Objects to be classified are described by vectors of values of all attributes:

$$\vec{x} = (x_1, \ldots, x_n) \in V(A_1) \times \cdots \times V(A_n)$$

Given $\vec{x}$ and an attribute A_k we denote by $A_k(\vec{x})$ the value x_k of the attribute A_k in $\vec{x}$.

Learning Decision Trees

Consider data represented as follows:

- A finite set of *attributes* $\mathcal{A} = \{A_1, \ldots, A_n\}$.
- Each attribute $A \in \mathcal{A}$ has its *set of values* $V(A)$.

We start with trees on discrete datasets, that is, assume $V(A)$ finite for all $A \in \mathcal{A}$.

Objects to be classified are described by vectors of values of all attributes:

$$\vec{x} = (x_1, \ldots, x_n) \in V(A_1) \times \cdots \times V(A_n)$$

Given $\vec{x}$ and an attribute A_k we denote by $A_k(\vec{x})$ the value x_k of the attribute A_k in $\vec{x}$.

Consider a set C of *classes*.

We consider a multiclass classification in general, i.e., C is an arbitrary finite set.

Example

The tennis problem:

- The attributes are:

$$A_1 = \textit{Outlook}, A_2 = \textit{Temperature}, A_3 = \textit{Humidity}, A_4 = \textit{Wind}$$

Example

The tennis problem:

- The attributes are:

  $A_1 = \textit{Outlook}, A_2 = \textit{Temperature}, A_3 = \textit{Humidity}, A_4 = \textit{Wind}$

- The sets of values of the attributes:
  - $V(A_1) = \{\textit{Sunny}, \textit{Overcast}, \textit{Rain}\}$
  - $V(A_2) = \{\textit{Hot}, \textit{Mild}, \textit{Cool}\}$
  - $V(A_3) = \{\textit{High}, \textit{Normal}\}$
  - $V(A_4) = \{\textit{Strong}, \textit{Weak}\}$

Example

The tennis problem:

- The attributes are:

$$A_1 = \textit{Outlook}, A_2 = \textit{Temperature}, A_3 = \textit{Humidity}, A_4 = \textit{Wind}$$

- The sets of values of the attributes:
  - $V(A_1) = \{\textit{Sunny}, \textit{Overcast}, \textit{Rain}\}$
  - $V(A_2) = \{\textit{Hot}, \textit{Mild}, \textit{Cool}\}$
  - $V(A_3) = \{\textit{High}, \textit{Normal}\}$
  - $V(A_4) = \{\textit{Strong}, \textit{Weak}\}$
- Consider

$$\vec{x} = (\textit{Overcast}, \textit{Hot}, \textit{Normal}, \textit{Weak}) \in V(A_1) \times V(A_2) \times V(A_3) \times V(A_4)$$

Example

The tennis problem:

- The attributes are:

$$A_1 = \textit{Outlook}, A_2 = \textit{Temperature}, A_3 = \textit{Humidity}, A_4 = \textit{Wind}$$

- The sets of values of the attributes:
  - $V(A_1) = \{\textit{Sunny}, \textit{Overcast}, \textit{Rain}\}$
  - $V(A_2) = \{\textit{Hot}, \textit{Mild}, \textit{Cool}\}$
  - $V(A_3) = \{\textit{High}, \textit{Normal}\}$
  - $V(A_4) = \{\textit{Strong}, \textit{Weak}\}$
- Consider

$$\vec{x} = (\textit{Overcast}, \textit{Hot}, \textit{Normal}, \textit{Weak}) \in V(A_1) \times V(A_2) \times V(A_3) \times V(A_4)$$

Then

$$A_3(\vec{x}) = \textit{Humidity}(\vec{x}) = \textit{Normal}$$

$$A_4(\vec{x}) = \textit{Wind}(\vec{x}) = \textit{Weak}$$

Example

The tennis problem:

- The attributes are:

  $A_1 = \mathit{Outlook}, A_2 = \mathit{Temperature}, A_3 = \mathit{Humidity}, A_4 = \mathit{Wind}$

- The sets of values of the attributes:
  - $V(A_1) = \{\mathit{Sunny}, \mathit{Overcast}, \mathit{Rain}\}$
  - $V(A_2) = \{\mathit{Hot}, \mathit{Mild}, \mathit{Cool}\}$
  - $V(A_3) = \{\mathit{High}, \mathit{Normal}\}$
  - $V(A_4) = \{\mathit{Strong}, \mathit{Weak}\}$
- Consider

  $$\vec{x} = (\mathit{Overcast}, \mathit{Hot}, \mathit{Normal}, \mathit{Weak}) \in V(A_1) \times V(A_2) \times V(A_3) \times V(A_4)$$

  Then

  $$A_3(\vec{x}) = \mathit{Humidity}(\vec{x}) = \mathit{Normal}$$

  $$A_4(\vec{x}) = \mathit{Wind}(\vec{x}) = \mathit{Weak}$$

- $C = \{\mathit{Yes}, \mathit{No}\}$

Decision Trees

Consider (directed, rooted) *trees* $\mathcal{T} = (T, E)$ where T is a set of nodes and $E \subseteq T \times T$ is a set of directed edges.

Decision Trees

Consider (directed, rooted) *trees* $\mathcal{T} = (T, E)$ where T is a set of nodes and $E \subseteq T \times T$ is a set of directed edges.

Denote by $T_{leaf} \subseteq T$ the set of all *leaves* of the tree and by T_{int} the set $T \setminus T_{leaf}$ of *internal nodes*.

Decision Trees

Consider (directed, rooted) *trees* $\mathcal{T} = (T, E)$ where T is a set of nodes and $E \subseteq T \times T$ is a set of directed edges.

Denote by $T_{leaf} \subseteq T$ the set of all *leaves* of the tree and by T_{int} the set $T \setminus T_{leaf}$ of *internal nodes*.

A *decision tree* is

- a tree $\mathcal{T} = (T, E)$ where
- each leaf $\tau \in T_{leaf}$ is assigned a class $C(\tau) \in C$,
- each internal node $\tau \in T_{int}$ is assigned an attribute $A(\tau) \in \mathcal{A}$,
- and there is a bijection between edges from τ and values of the attribute $A(\tau)$. Given an edge $(\tau, \tau') \in E$ we write $V(\tau, \tau')$ to denote the value of the attribute $A(\tau)$ assigned to the edge.

Decision Trees

Consider (directed, rooted) *trees* $\mathcal{T} = (T, E)$ where T is a set of nodes and $E \subseteq T \times T$ is a set of directed edges.

Denote by $T_{leaf} \subseteq T$ the set of all *leaves* of the tree and by T_{int} the set $T \setminus T_{leaf}$ of *internal nodes*.

A *decision tree* is

- a tree $\mathcal{T} = (T, E)$ where
- each leaf $\tau \in T_{leaf}$ is assigned a class $C(\tau) \in C$,
- each internal node $\tau \in T_{int}$ is assigned an attribute $A(\tau) \in \mathcal{A}$,
- and there is a bijection between edges from τ and values of the attribute $A(\tau)$. Given an edge $(\tau, \tau') \in E$ we write $V(\tau, \tau')$ to denote the value of the attribute $A(\tau)$ assigned to the edge.

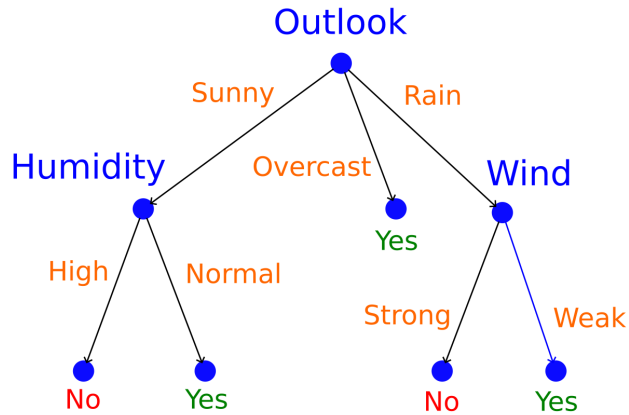
Inference: Given an input $\vec{x}$, we traverse the tree from the root to a leaf, always choosing edges labeled with values of attributes from $\vec{x}$. The output is the class labeling the leaf.

Example

$T = \{O, H, W, z_1, z_2, z_3, z_4, z_5\}$

$T_{leaf} = \{z_1, z_2, z_3, z_4, z_5\}$, $T_{int} = \{O, H, W\}$

$E = \{(O, H), (O, W), (H, z_1), (H, z_2), (O, z_3), (W, z_4), (W, z_5)\}$



$C(z_1) = C(z_3) = \mathit{No}$, $C(z_2) = C(z_4) = \mathit{Yes}$

$A(O) = \mathit{Outlook}$, $A(H) = \mathit{Humidity}$, $A(W) = \mathit{Wind}$

$V(O, H) = \mathit{Sunny}$, $V(O, z_3) = \mathit{Overcast}$, $V(O, W) = \mathit{Rain}$
$V(H, z_1) = \mathit{High}$, $V(H, z_2) = \mathit{Normal}$
$V(W, z_4) = \mathit{Strong}$, $V(W, z_5) = \mathit{Weak}$

Inference: For $(\mathit{Rain}, \mathit{Hot}, \mathit{High}, \mathit{Strong})$ we reach z_4, yielding No.

Training Dataset

Consider a *training dataset*

$$\mathcal{D} = \{(\vec{x}_k, c_k) \mid k = 1, \ldots, p\}$$

Here $\vec{x}_k \in V(A_1) \times \cdots \times V(A_k)$ and $c_k \in C$ for every k.

Technically $\mathcal{D}$ can be a multiset containing several occurrences of the same vector.

Index	Outlook	Temperature	Humidity	Wind	PlayTennis
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Weak	Yes
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
10	Rain	Mild	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No

$$\mathcal{D} = \{((\mathit{Sunny}, \mathit{Hot}, \mathit{High}, \mathit{Weak}), \mathit{No}),$$
$$((\mathit{Sunny}, \mathit{Hot}, \mathit{High}, \mathit{Strong}), \mathit{No})$$
$$\ldots$$
$$((\mathit{Rain}, \mathit{Mild}, \mathit{High}, \mathit{Strong}), \mathit{No})\}$$

Learning Decision Trees

The learning algorithm ID3 works as follows:

- Start with the whole training dataset $\mathcal{D}$.

Learning Decision Trees

The learning algorithm ID3 works as follows:

- Start with the whole training dataset $\mathcal{D}$.
- If there is just a single class in $\mathcal{D}$, create a single node decision tree that returns the class.

Learning Decision Trees

The learning algorithm ID3 works as follows:

- Start with the whole training dataset $\mathcal{D}$.
- If there is just a single class in $\mathcal{D}$, create a single node decision tree that returns the class.
- Otherwise, identify an attribute $A \in \mathcal{A}$ which *best classifies* the examples in $\mathcal{D}$. For every $v \in V(A)$ we obtain

$$\mathcal{D}_v = \{\vec{x} \mid \vec{x} \in \mathcal{D}, A(\vec{x}) = v\}$$

We aim to have each $\mathcal{D}_v$ as pure as possible, that is, ideally, to contain examples of just a single class.

Learning Decision Trees

The learning algorithm ID3 works as follows:

- Start with the whole training dataset $\mathcal{D}$.
- If there is just a single class in $\mathcal{D}$, create a single node decision tree that returns the class.
- Otherwise, identify an attribute $A \in \mathcal{A}$ which *best classifies* the examples in $\mathcal{D}$. For every $v \in V(A)$ we obtain

$$\mathcal{D}_v = \{\vec{x} \mid \vec{x} \in \mathcal{D}, A(\vec{x}) = v\}$$

  We aim to have each $\mathcal{D}_v$ as pure as possible, that is, ideally, to contain examples of just a single class.
- Finally,
  - create a root node τ of a decision tree,

Learning Decision Trees

The learning algorithm ID3 works as follows:

- Start with the whole training dataset $\mathcal{D}$.
- If there is just a single class in $\mathcal{D}$, create a single node decision tree that returns the class.
- Otherwise, identify an attribute $A \in \mathcal{A}$ which *best classifies* the examples in $\mathcal{D}$. For every $v \in V(A)$ we obtain

$$\mathcal{D}_v = \{\vec{x} \mid \vec{x} \in \mathcal{D}, A(\vec{x}) = v\}$$

  We aim to have each $\mathcal{D}_v$ as pure as possible, that is, ideally, to contain examples of just a single class.
- Finally,
  - create a root node τ of a decision tree,
  - assign the attribute A to τ,

Learning Decision Trees

The learning algorithm ID3 works as follows:

- Start with the whole training dataset $\mathcal{D}$.
- If there is just a single class in $\mathcal{D}$, create a single node decision tree that returns the class.
- Otherwise, identify an attribute $A \in \mathcal{A}$ which *best classifies* the examples in $\mathcal{D}$. For every $v \in V(A)$ we obtain

$$\mathcal{D}_v = \{\vec{x} \mid \vec{x} \in \mathcal{D}, A(\vec{x}) = v\}$$

  We aim to have each $\mathcal{D}_v$ as pure as possible, that is, ideally, to contain examples of just a single class.
- Finally,
  - create a root node τ of a decision tree,
  - assign the attribute A to τ,
  - for every $v \in V(A)$, recursively construct a decision tree with a root τ_v using $\mathcal{D}_v$,

Learning Decision Trees

The learning algorithm ID3 works as follows:

- Start with the whole training dataset $\mathcal{D}$.
- If there is just a single class in $\mathcal{D}$, create a single node decision tree that returns the class.
- Otherwise, identify an attribute $A \in \mathcal{A}$ which *best classifies* the examples in $\mathcal{D}$. For every $v \in V(A)$ we obtain

$$\mathcal{D}_v = \{\vec{x} \mid \vec{x} \in \mathcal{D}, A(\vec{x}) = v\}$$

  We aim to have each $\mathcal{D}_v$ as pure as possible, that is, ideally, to contain examples of just a single class.
- Finally,
  - create a root node τ of a decision tree,
  - assign the attribute A to τ,
  - for every $v \in V(A)$, recursively construct a decision tree with a root τ_v using $\mathcal{D}_v$,
  - for every $v \in V(A)$ introduce an edge (τ, τ_v) assigned v.

1: **function** ID3(dataset $\mathcal{D}$, attribute set $\mathcal{A}$)
2: Create a root node τ for the tree
3: **if** $\mathcal{D} = \emptyset$ **then**
4: Return the single node τ assigned with a default class.
5: **else if** all examples in $\mathcal{D}$ are of the same class c **then**
6: Return the single-node tree, where τ is assigned c
7: **else if** set of attributes $\mathcal{A}$ is empty **then**
8: Return the single-node tree where τ is assigned
the most common class in $\mathcal{D}$
9: **else**
10: Choose attribute $A \in \mathcal{A}$ *best classifying* examples in $\mathcal{D}$.
11: Set the decision attribute for τ to A
12: **for** each value $v \in D(A)$ **do**
13: Compute a decision tree ID3($\mathcal{D}_v, \mathcal{A} \setminus \{A\}$) with root τ_v,
14: add a new edge (τ, τ_v) assigned v.
15: **end for**
16: **end if**
17: **return** τ
18: **end function**

Best Classifying Attribute

We aim to choose an attribute that best informs us about the class.

As a result, we would possibly use as few attributes as possible and obtain a small tree containing only class-relevant decisions.

How to choose an attribute that best classifies examples in $\mathcal{D}$?

There are several measures used in practice.

The most common are

- *information gain*
- *Gini impurity decrease*

Information Gain

The information gain is based on the notion of entropy.

Information Gain

The information gain is based on the notion of entropy.

We need some notation:

- Given a training dataset $\mathcal{D}$ and a class $c \in C$ we denote by p_c the proportion of examples with class c in $\mathcal{D}$.

Information Gain

The information gain is based on the notion of entropy.

We need some notation:

- Given a training dataset $\mathcal{D}$ and a class $c \in C$ we denote by p_c the proportion of examples with class c in $\mathcal{D}$.
- We define the *entropy* of $\mathcal{D}$ by

$$Entropy(\mathcal{D}) = \sum_{c \in C} -p_c \log_2 p_c$$

Information Gain

The information gain is based on the notion of entropy.

We need some notation:

- Given a training dataset $\mathcal{D}$ and a class $c \in C$ we denote by p_c the proportion of examples with class c in $\mathcal{D}$.
- We define the *entropy* of $\mathcal{D}$ by

$$Entropy(\mathcal{D}) = \sum_{c \in C} -p_c \log_2 p_c$$

- The *information gain* of an attribute A is then defined by

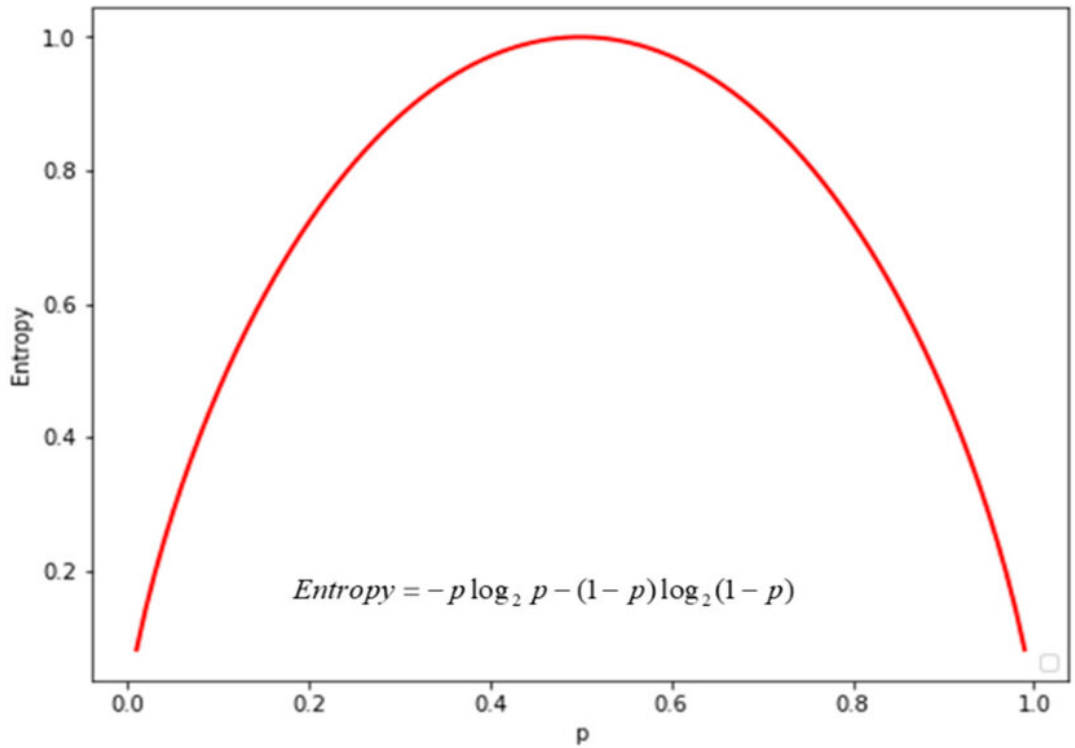
$$Gain(\mathcal{D}, A) = Entropy(\mathcal{D}) - \sum_{v \in V(A)} \frac{|\mathcal{D}_v|}{|\mathcal{D}|} Entropy(\mathcal{D}_v)$$

In every step of the ID3 algorithm, we choose an attribute *maximizing* the information gain for the current dataset $\mathcal{D}$.

Information Gain

The intuition behind information gain:

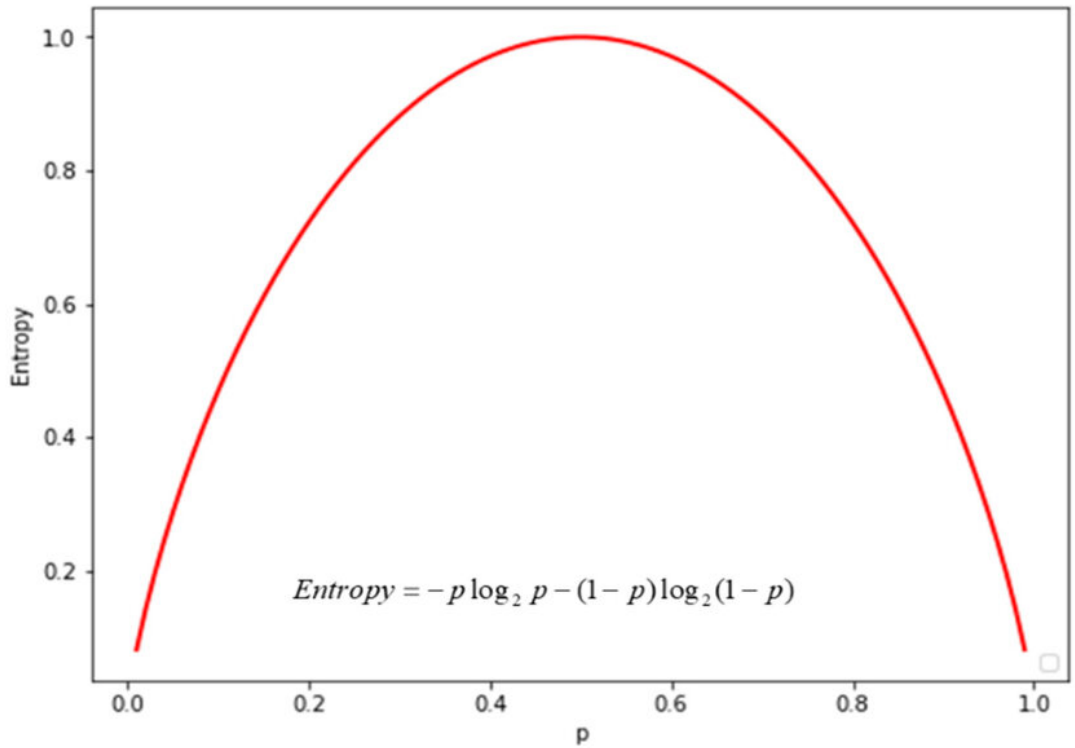
- Consider $C = \{0, 1\}$ and p the proportion of examples of class 1. p measures the "uncertainty" of the class:



Information Gain

The intuition behind information gain:

- Consider $C = \{0, 1\}$ and p the proportion of examples of class 1. p measures the "uncertainty" of the class:

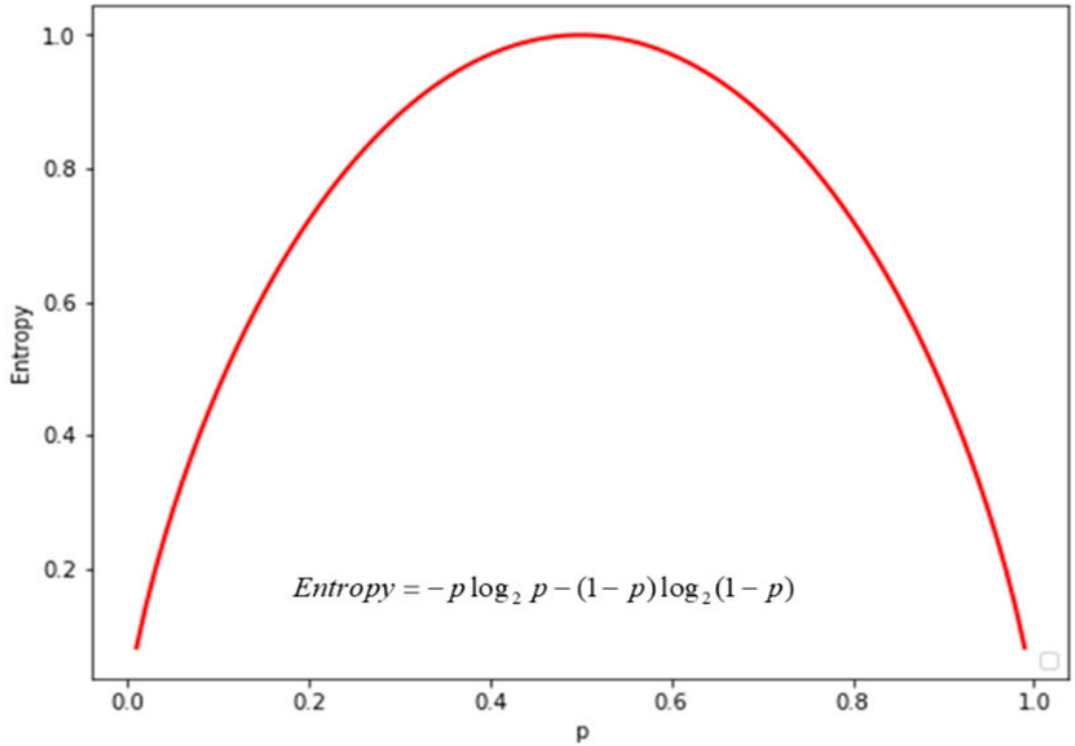


- $\sum_{v \in V(A)} \frac{|\mathcal{D}_v|}{|\mathcal{D}|} Entropy(\mathcal{D}_v)$ is weighted uncertainty of classes in each $\mathcal{D}_v$ (weighted by the relative size of $\mathcal{D}_v$).

Information Gain

The intuition behind information gain:

- Consider $C = \{0, 1\}$ and p the proportion of examples of class 1. p measures the "uncertainty" of the class:



- $\sum_{v \in V(A)} \frac{|\mathcal{D}_v|}{|\mathcal{D}|} Entropy(\mathcal{D}_v)$ is weighted uncertainty of classes in each $\mathcal{D}_v$ (weighted by the relative size of $\mathcal{D}_v$).
- $Gain(\mathcal{D}, A)$ measures reduction in uncertainty of classes by splitting $\mathcal{D}$ according to A.

Gini Impurity

- We define *Gini impurity* of $\mathcal{D}$ by

$$Gini(\mathcal{D}) = 1 - \sum_{c \in C} p_c^2$$

Gini Impurity

- We define *Gini impurity* of $\mathcal{D}$ by

$$Gini(\mathcal{D}) = 1 - \sum_{c \in C} p_c^2$$

- The *impurity decrease* of an attribute A is then defined similarly to the gain in the entropy case

$$ImpDec(\mathcal{D}, A) = Gini(\mathcal{D}) - \sum_{v \in V(A)} \frac{|\mathcal{D}_v|}{|\mathcal{D}|} Gini(\mathcal{D}_v)$$

In every step of the ID3 algorithm, we choose an attribute *maximizing* the impurity decrease for the current dataset $\mathcal{D}$.

Gini Impurity

What is the intuition behind $Gini(\mathcal{D})$?

Gini Impurity

What is the intuition behind $Gini(\mathcal{D})$?

Assume we randomly independently choose objects from $\mathcal{D}$.

$1 - \sum_{c \in C} p_c^2$ is the probability of choosing two objects of different classes in two consecutive independent trials.

Indeed, p_c is the probability of choosing an object of class c, p_c^2 the probability of choosing objects of the class c twice, and $\sum_{c \in C} p_c^2$ the probability of choosing two objects of the same class.

In what follows (and at the exam), we will work only with the Gini impurity as it is easier to compute.

Example

Consider our tennis example (see the table).

- Consider the whole dataset $\mathcal{D}$.
  - $p_{Yes} = 9/14$
  - $p_{No} = 5/14$
  - $Gini(\mathcal{D}) = 1 - (9/14)^2 - (5/14)^2 = 0.45918$

Example

Consider our tennis example (see the table).

- Consider the whole dataset $\mathcal{D}$.
  - $p_{Yes} = 9/14$
  - $p_{No} = 5/14$
  - $Gini(\mathcal{D}) = 1 - (9/14)^2 - (5/14)^2 = 0.45918$
- For $A = Outlook$ we get
  - $Gini(\mathcal{D}_{Sunny}) = 1 - (2/5)^2 - (3/5)^2 = 0.48$
  - $Gini(\mathcal{D}_{Overcast}) = 1 - 1^2 - 0^2 = 0$
  - $Gini(\mathcal{D}_{Rain}) = 1 - (3/5)^2 - (2/5)^2 = 0.48$

Example

Consider our tennis example (see the table).

- Consider the whole dataset $\mathcal{D}$.
  - $p_{Yes} = 9/14$
  - $p_{No} = 5/14$
  - $Gini(\mathcal{D}) = 1 - (9/14)^2 - (5/14)^2 = 0.45918$
- For $A = Outlook$ we get
  - $Gini(\mathcal{D}_{Sunny}) = 1 - (2/5)^2 - (3/5)^2 = 0.48$
  - $Gini(\mathcal{D}_{Overcast}) = 1 - 1^2 - 0^2 = 0$
  - $Gini(\mathcal{D}_{Rain}) = 1 - (3/5)^2 - (2/5)^2 = 0.48$

  Thus

$$ImpDec(\mathcal{D}, Outlook) = 0.459 - (5/14) \cdot 0.48 - (4/14) \cdot 0 - (5/14) \cdot 0.48 = 0.117$$

- $ImpDec(\mathcal{D}, Temperature) = 0.018$
- $ImpDec(\mathcal{D}, Humidity) = 0.091$
- $ImpDec(\mathcal{D}, Wind) = 0.030$

So the largest information gain is given by the *Outlook*.

Example

Going further on, consider $\mathcal{D} = \mathcal{D}_{Sunny}$. We get

- $ImpDec(\mathcal{D}, Temperature) = 0.279$
- $ImpDec(\mathcal{D}, Humidity) = 0.48$
- $ImpDec(\mathcal{D}, Wind) = 0.013$

The best choice attribude after *Sunny* in *Outlook* is *Humidity*.

Example

Going further on, consider $\mathcal{D} = \mathcal{D}_{Sunny}$. We get

- $ImpDec(\mathcal{D}, Temperature) = 0.279$
- $ImpDec(\mathcal{D}, Humidity) = 0.48$
- $ImpDec(\mathcal{D}, Wind) = 0.013$

The best choice attribude after *Sunny* in *Outlook* is *Humidity*.

Now consider $\mathcal{D} = \mathcal{D}_{Rain}$.

- $ImpDec(\mathcal{D}, Temperature) = 0.013$
- $ImpDec(\mathcal{D}, Humidity) = 0.013$
- $ImpDec(\mathcal{D}, Wind) = 0.48$

The best choice attribude after *Rain* in *Outlook* is *Wind*.

Continuous-Valued Attributes

What if values of an attribute A come from a continuous variable?

A is a numerical attribute that can take any value in an interval, such as temperature, size, time, etc.

Continuous-Valued Attributes

What if values of an attribute A come from a continuous variable?

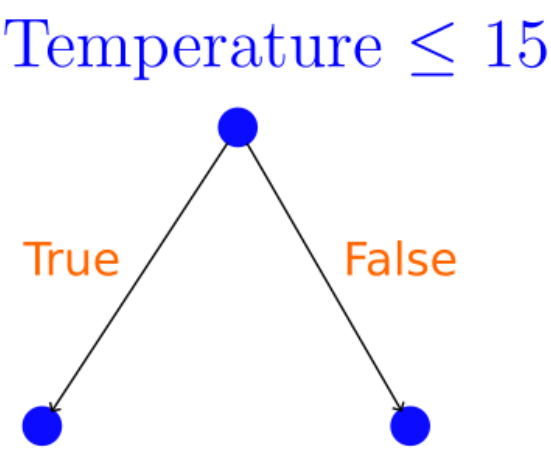
A is a numerical attribute that can take any value in an interval, such as temperature, size, time, etc.

Consider an internal node $\tau \in T_{int}$ assigned such a continuous attribute A. Then

- τ is assigned a threshold value called a *cut point* $H \in \mathbb{R}$,
- there are two edges $e_{\text{true}}, e_{\text{false}}$ from τ,
- e_{true} labeled with True and e_{false} labeled with False.

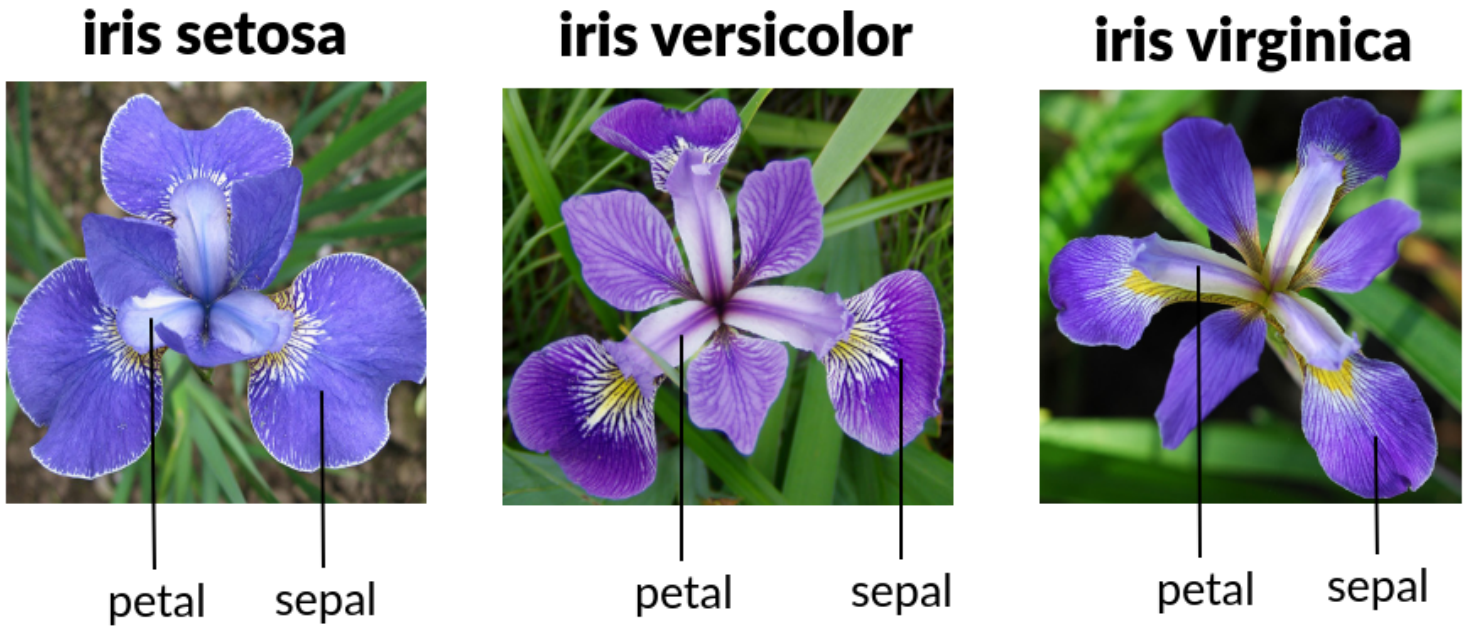
During inference, when considering an example $\vec{x}$ in the node τ,

- evaluate $A(\vec{x}) \leq H$,
- if $A(\vec{x}) \leq H$, then follow e_{true},
- else follow e_{false}.



In training, the cut point is chosen from the attribute values in the training set using information gain/impurity decrease similar to discrete attributes.

Iris Example



Attributes

Sepal.Length, Sepal.Width, Petal.Length, Petal.Width

Classes (Variety)

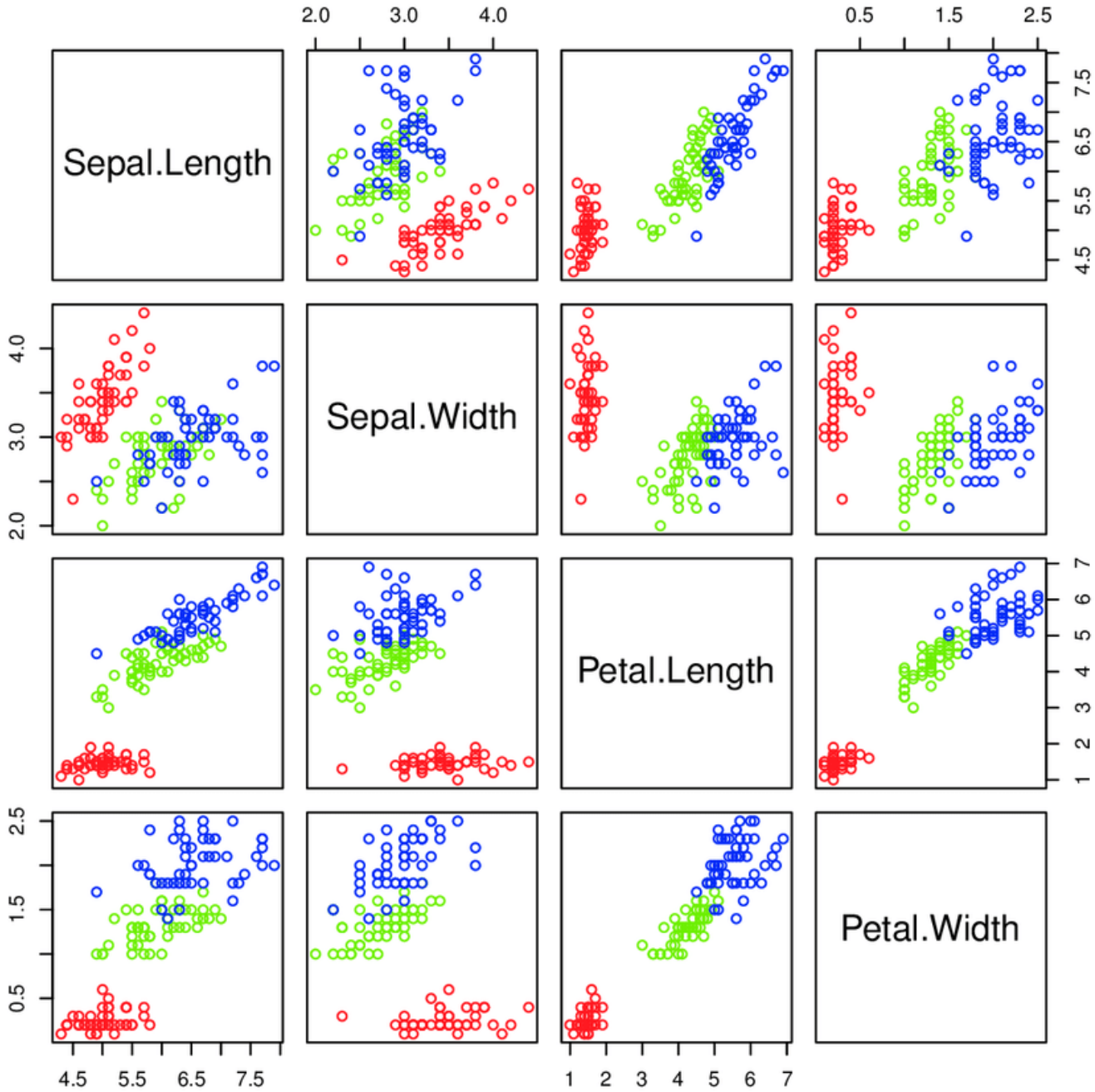
Setosa, Versicolor, Virginica

Iris Example

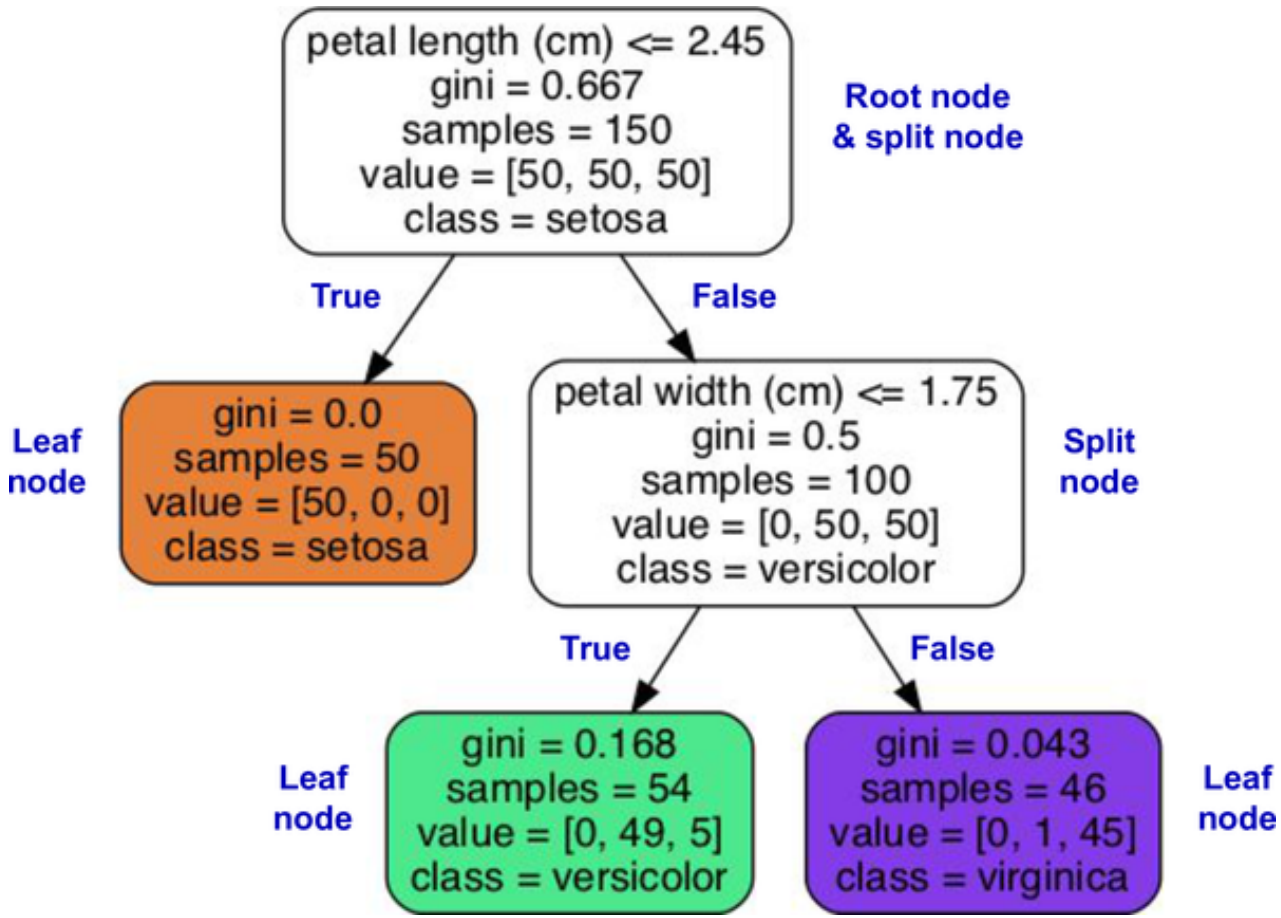
The dataset (150 examples):

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Variety
5.5	3.5	1.3	0.2	Setosa
6.8	2.8	4.8	1.4	Versicolor
6.7	3.1	4.7	1.5	Versicolor
6.9	3.1	5.1	2.3	Virginica
7.3	2.9	6.3	1.8	Virginica
5.4	3.7	1.5	0.2	Setosa
4.6	3.4	1.4	0.3	Setosa
6.2	2.8	4.8	1.8	Virginica
5.4	3.0	4.5	1.5	Versicolor
4.7	3.2	1.6	0.2	Setosa
6.7	3.3	5.7	2.1	Virginica
5.0	3.4	1.5	0.2	Setosa
5.0	3.0	1.6	0.2	Setosa
4.4	2.9	1.4	0.2	Setosa
6.0	3.4	4.5	1.6	Versicolor
5.1	3.5	1.4	0.2	Setosa
6.6	3.0	4.4	1.4	Versicolor
5.9	3.2	4.8	1.8	Versicolor
5.6	2.8	4.9	2.0	Virginica
...				

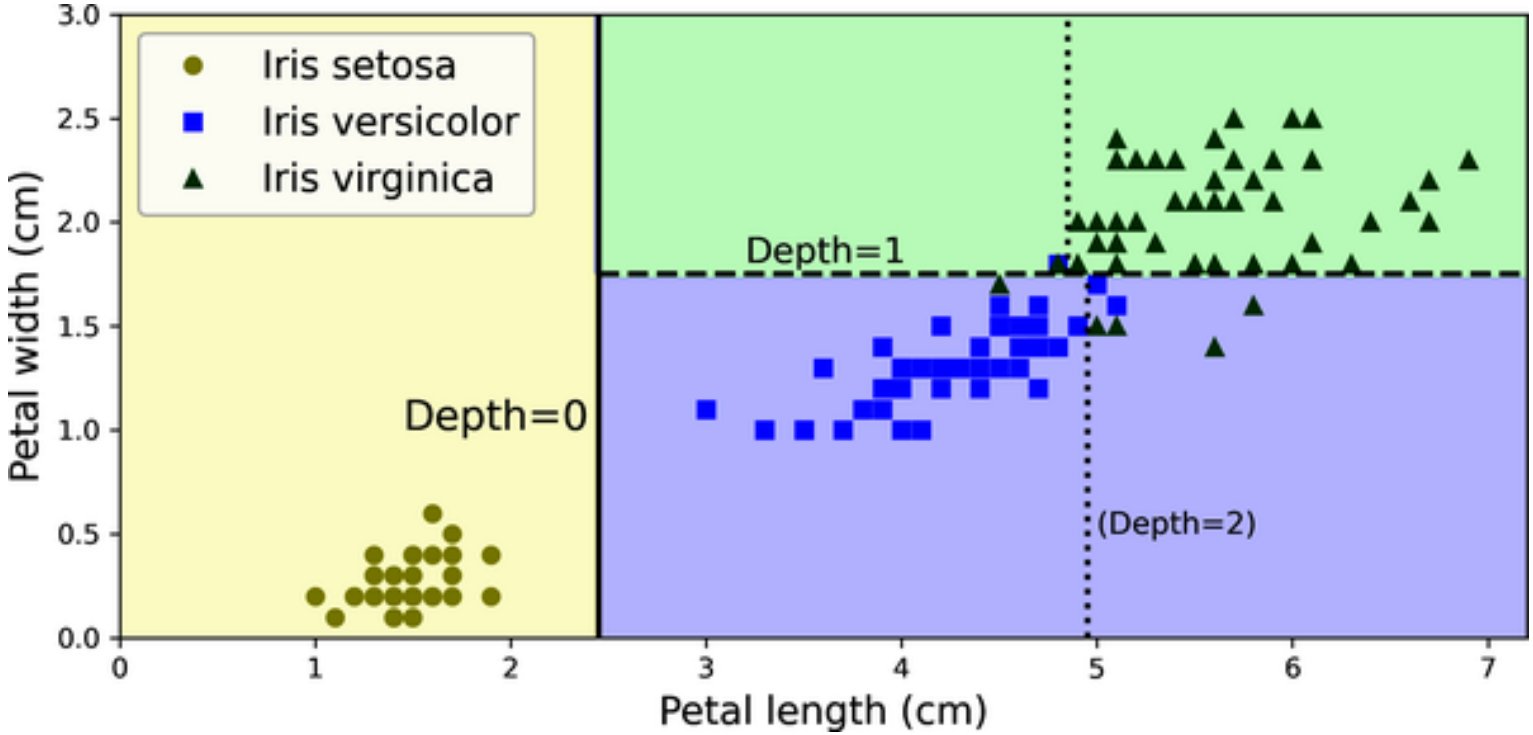
Iris Example



Iris Example - Decision Tree



Iris Example - Decision Tree Boudaries



If the leaves are split further, the Depth $= 2$ boundary would be added.

Attribute Importance Computation

How important are attributes for the trained tree $\mathcal{T}$? Depends on

- how close they are to the root of $\mathcal{T}$,
- how large information gain/decrease in impurity they give.

There are several formulae for computing the importance.

Attribute Importance Computation

How important are attributes for the trained tree $\mathcal{T}$? Depends on

- how close they are to the root of $\mathcal{T}$,
- how large information gain/decrease in impurity they give.

There are several formulae for computing the importance.

One example is *mean decrease impurity* defined as follows:

Consider a decision tree T trained on a dataset $\mathcal{D}$ using the ID3.

Attribute Importance Computation

How important are attributes for the trained tree $\mathcal{T}$? Depends on

- how close they are to the root of $\mathcal{T}$,
- how large information gain/decrease in impurity they give.

There are several formulae for computing the importance.

One example is *mean decrease impurity* defined as follows:

Consider a decision tree T trained on a dataset $\mathcal{D}$ using the ID3.

For every node τ of $\mathcal{T}$, denote by $\mathcal{D}[\tau]$ the subset of $\mathcal{D}$ which was used in the ID3 procedure when the node τ was created (line 2).

Attribute Importance Computation

How important are attributes for the trained tree $\mathcal{T}$? Depends on

- how close they are to the root of $\mathcal{T}$,
- how large information gain/decrease in impurity they give.

There are several formulae for computing the importance.

One example is *mean decrease impurity* defined as follows:

Consider a decision tree T trained on a dataset $\mathcal{D}$ using the ID3.

For every node τ of $\mathcal{T}$, denote by $\mathcal{D}[\tau]$ the subset of $\mathcal{D}$ which was used in the ID3 procedure when the node τ was created (line 2).

Consider an attribute A and denote by $T[A] \subseteq T_{int}$ the set of all nodes of $\mathcal{T}$ assigned the attribute A by ID3 (line 11).

Attribute Importance Computation

How important are attributes for the trained tree $\mathcal{T}$? Depends on

- how close they are to the root of $\mathcal{T}$,
- how large information gain/decrease in impurity they give.

There are several formulae for computing the importance.

One example is *mean decrease impurity* defined as follows:

Consider a decision tree T trained on a dataset $\mathcal{D}$ using the ID3.

For every node τ of $\mathcal{T}$, denote by $\mathcal{D}[\tau]$ the subset of $\mathcal{D}$ which was used in the ID3 procedure when the node τ was created (line 2).

Consider an attribute A and denote by $T[A] \subseteq T_{int}$ the set of all nodes of $\mathcal{T}$ assigned the attribute A by ID3 (line 11).

Then define the importance as the average decrease in Gini impurity (i.e., average *ImpDec*) in the nodes of $T[A]$:

$$GiniImportance(A) = \sum_{\tau \in T[A]} \frac{|\mathcal{D}[\tau]|}{|\mathcal{D}|} ImpDec(\mathcal{D}[\tau], A)$$

Decision Trees

Practical Issues

Practical Issues

- Data preprocessing
- Model tunning (overfitting and underfitting)
- Sensitivity to changes in data/hyperparameters
- Learning representation problems (the XOR)

Data Preprocessing

Little preprocessing is needed for decision trees.

Data Preprocessing

Little preprocessing is needed for decision trees.

Of course, ideally, clean up the data, but

- Missing values are not such a big issue
  (considering a concrete example, exclude the attributes with missing values)

Data Preprocessing

Little preprocessing is needed for decision trees.

Of course, ideally, clean up the data, but

- Missing values are not such a big issue
  (considering a concrete example, exclude the attributes with missing values)
- Outliers are not such a big issue either; the division of nodes is done based on relative counts, not concrete values.

Data Preprocessing

Little preprocessing is needed for decision trees.

Of course, ideally, clean up the data, but

- Missing values are not such a big issue
  (considering a concrete example, exclude the attributes with missing values)
- Outliers are not such a big issue either; the division of nodes is done based on relative counts, not concrete values.
- Decision trees can cope with continuous and categorical values directly (i.e., no encoding necessary)

Data Preprocessing

Little preprocessing is needed for decision trees.

Of course, ideally, clean up the data, but

- Missing values are not such a big issue
  (considering a concrete example, exclude the attributes with missing values)
- Outliers are not such a big issue either; the division of nodes is done based on relative counts, not concrete values.
- Decision trees can cope with continuous and categorical values directly (i.e., no encoding necessary)

Imbalanced classes might cause problems because of small information gain/impurity decrease in splitting.

Imbalanced Classes

Consider a dataset $\mathcal{D}$ where

- there are two classes, $C = \{0, 1\}$,
- 10^6 examples have the class 1,
- 100 examples have the class 0.

Imbalanced Classes

Consider a dataset $\mathcal{D}$ where

- there are two classes, $C = \{0, 1\}$,
- 10^6 examples have the class 1,
- 100 examples have the class 0.

Then

- $p_1 = 10^6/(10^6 + 100) \approx 1$ and $p_0 = 100/10^6 \approx 0$,
- thus the Gini impurity $1 - p_1^2 - p_0^2 \approx 0$.

Imbalanced Classes

Consider a dataset $\mathcal{D}$ where

- there are two classes, $C = \{0, 1\}$,
- 10^6 examples have the class 1,
- 100 examples have the class 0.

Then

- $p_1 = 10^6/(10^6 + 100) \approx 1$ and $p_0 = 100/10^6 \approx 0$,
- thus the Gini impurity $1 - p_1^2 - p_0^2 \approx 0$.

Consider an attribute A with $V(A) = \{a, b\}$.

Splitting $\mathcal{D}$ according to A gives to sets $\mathcal{D}_a$ and $\mathcal{D}_b$.

Imbalanced Classes

Consider a dataset $\mathcal{D}$ where

- there are two classes, $C = \{0, 1\}$,
- 10^6 examples have the class 1,
- 100 examples have the class 0.

Then

- $p_1 = 10^6/(10^6 + 100) \approx 1$ and $p_0 = 100/10^6 \approx 0$,
- thus the Gini impurity $1 - p_1^2 - p_0^2 \approx 0$.

Consider an attribute A with $V(A) = \{a, b\}$.

Splitting $\mathcal{D}$ according to A gives to sets $\mathcal{D}_a$ and $\mathcal{D}_b$.

What is the impurity decrease caused by the attribute?

$$ImpDec(\mathcal{D}, A) = Gini(\mathcal{D}) - \frac{|\mathcal{D}_a|}{|\mathcal{D}|} Gini(\mathcal{D}_a) - \frac{|\mathcal{D}_b|}{|\mathcal{D}|} Gini(\mathcal{D}_b)$$

Imbalanced Classes

Consider a dataset $\mathcal{D}$ where

- there are two classes, $C = \{0, 1\}$,
- 10^6 examples have the class 1,
- 100 examples have the class 0.

Then

- $p_1 = 10^6/(10^6 + 100) \approx 1$ and $p_0 = 100/10^6 \approx 0$,
- thus the Gini impurity $1 - p_1^2 - p_0^2 \approx 0$.

Consider an attribute A with $V(A) = \{a, b\}$.

Splitting $\mathcal{D}$ according to A gives to sets $\mathcal{D}_a$ and $\mathcal{D}_b$.

What is the impurity decrease caused by the attribute?

$$ImpDec(\mathcal{D}, A) = Gini(\mathcal{D}) - \frac{|\mathcal{D}_a|}{|\mathcal{D}|} Gini(\mathcal{D}_a) - \frac{|\mathcal{D}_b|}{|\mathcal{D}|} Gini(\mathcal{D}_b)$$

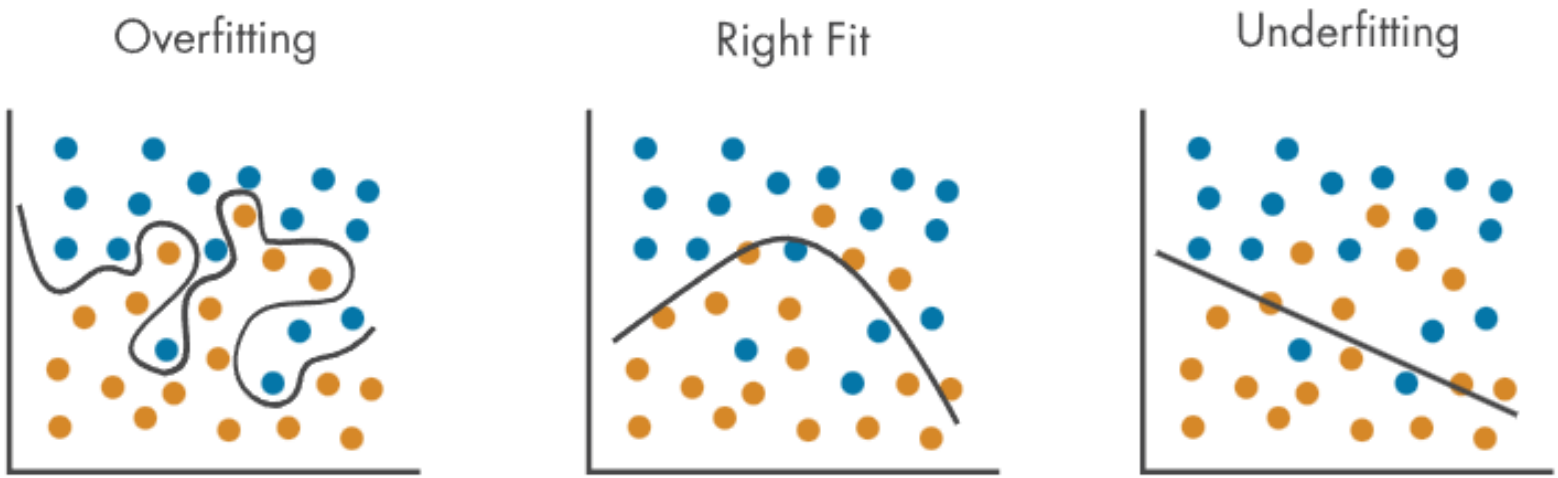
For small $|\mathcal{D}_a|$ (say ≤ 1000) we have small $|\mathcal{D}_a|/|\mathcal{D}|$

For not so small $\mathcal{D}_a$ we have $Gini(\mathcal{D}_a) \approx 0$.

In both cases, the impurity decrease is very small.

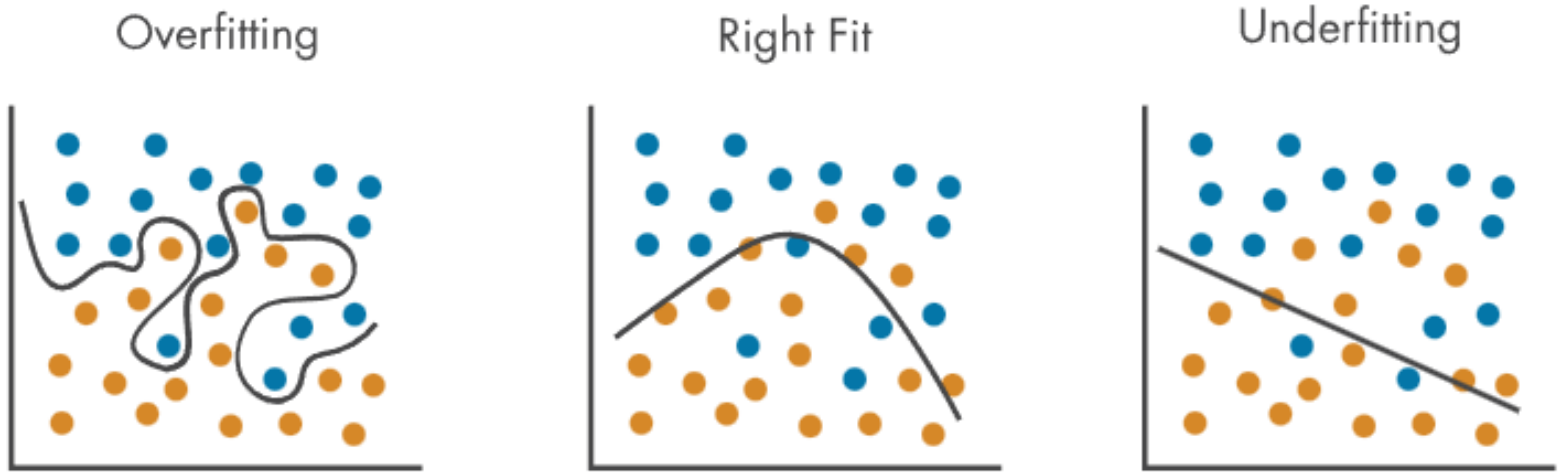
Model Tuning - Over/Under Fitting

The behavior of the model on the training set:



Model Tuning - Over/Under Fitting

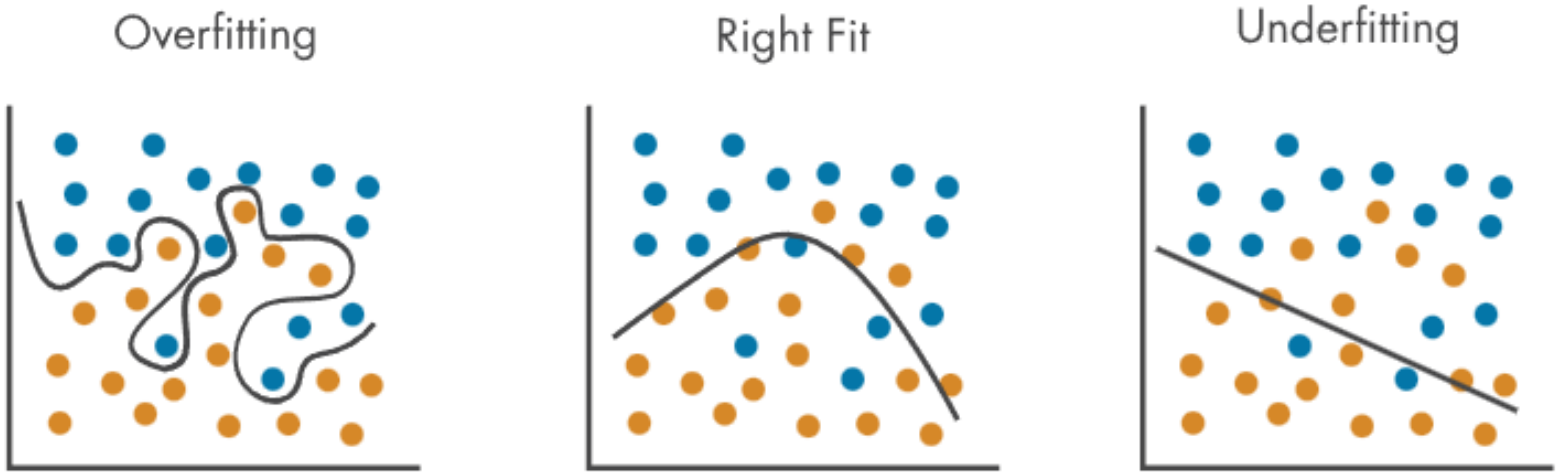
The behavior of the model on the training set:



- The left one is strongly overfitting. It would possibly not work well on new data.

Model Tuning - Over/Under Fitting

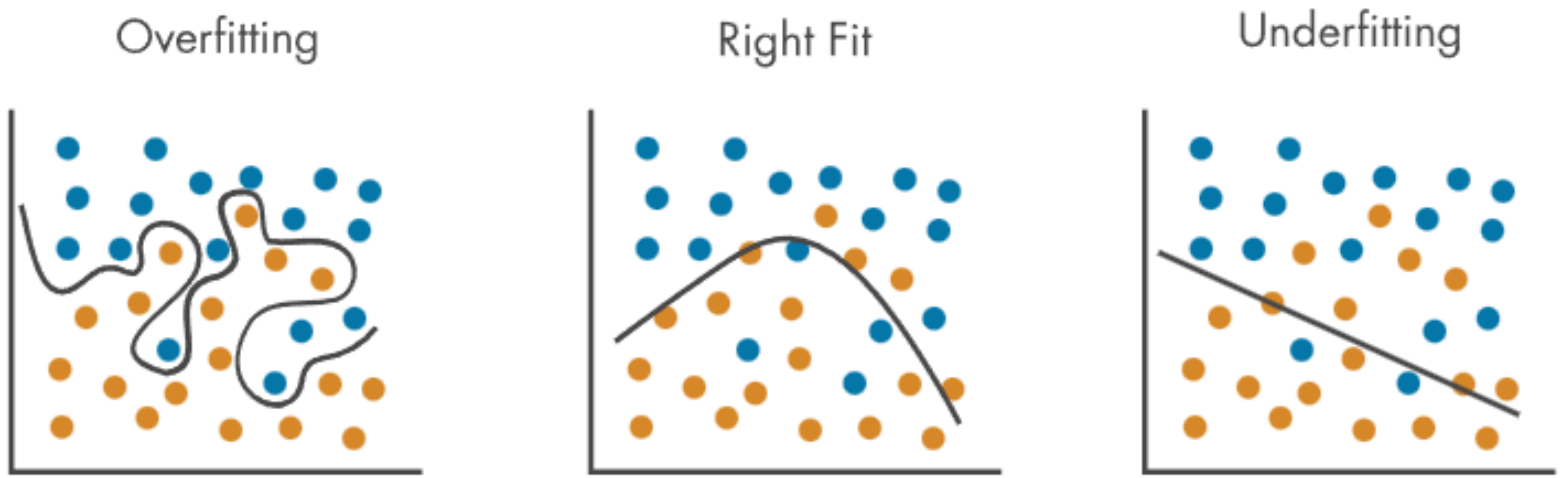
The behavior of the model on the training set:



- The left one is strongly overfitting. It would possibly not work well on new data.
- The right one is strongly underfitting. It would probably give poor classification results.

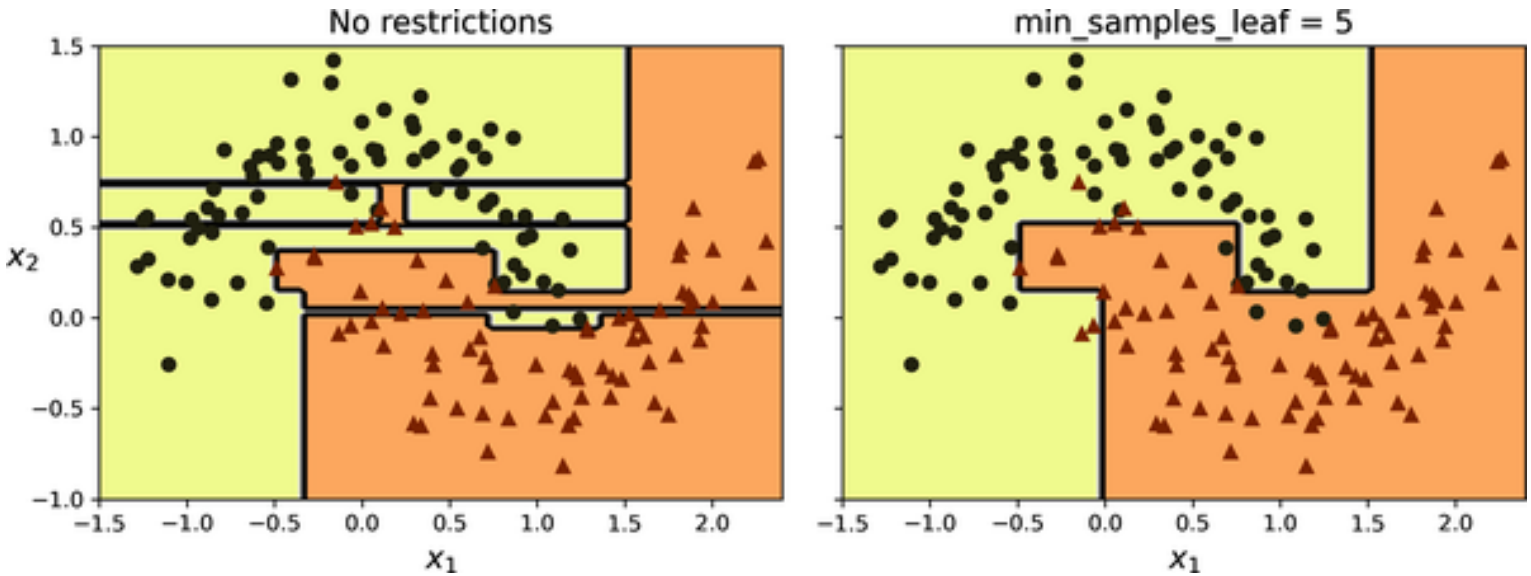
Model Tuning - Over/Under Fitting

The behavior of the model on the training set:



- The left one is strongly overfitting. It would possibly not work well on new data.
- The right one is strongly underfitting. It would probably give poor classification results.
- The middle one seems good (but still needs to be tested on fresh data).

Model Tuning - Overfitting in Decision Trees

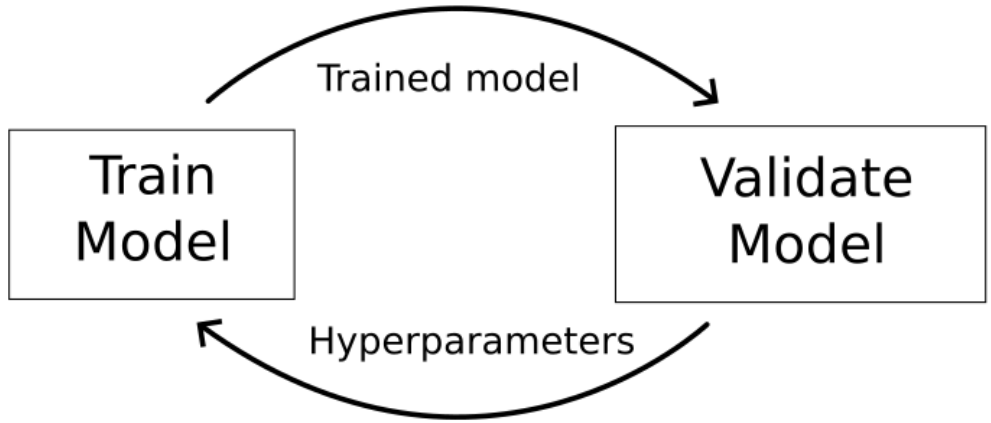


See the overfitting on the left and the "nice" model on the right.

Both overfitting and underfitting are best avoided. But how do we find out?

Model Tuning (In General)

Recall from the first lecture:

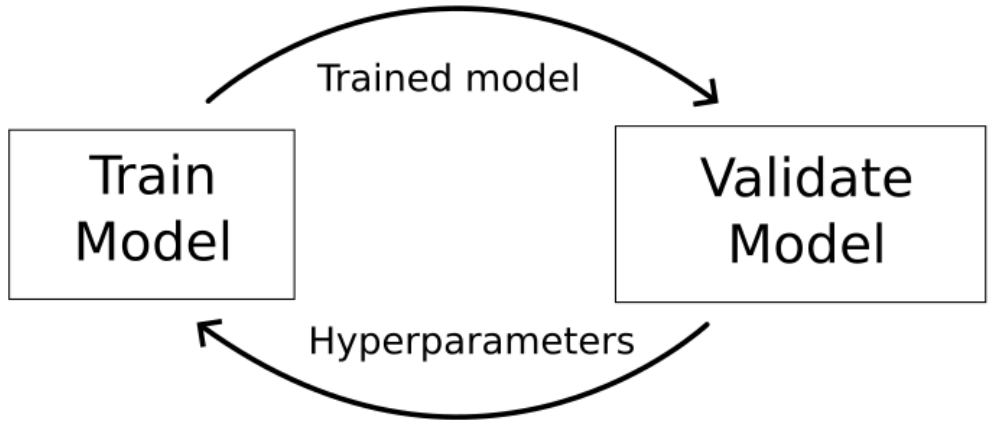


The validation should be done on a **validation set** separated from the training set.

We will discuss more sophisticated techniques later.

Model Tuning (In General)

Recall from the first lecture:



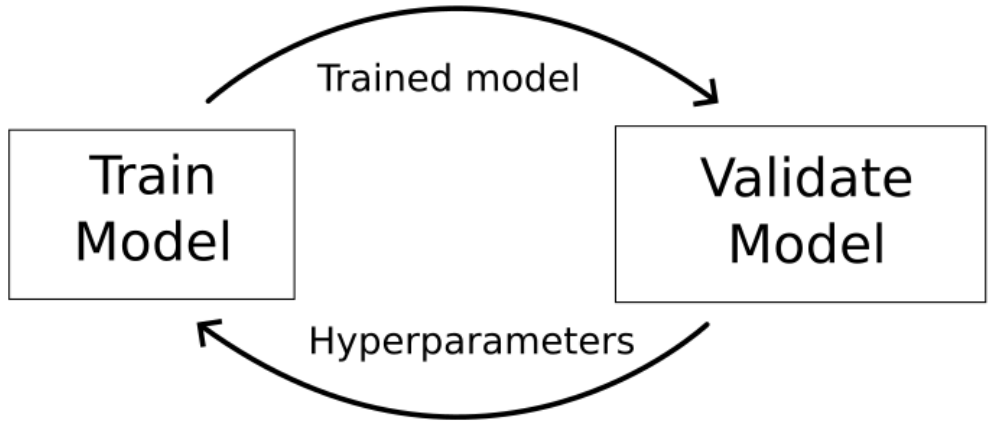
The validation should be done on a **validation set** separated from the training set.

We will discuss more sophisticated techniques later.

What hyperparameters to set? (see the next slide)

Model Tuning (In General)

Recall from the first lecture:



The validation should be done on a **validation set** separated from the training set.

We will discuss more sophisticated techniques later.

What hyperparameters to set? (see the next slide)

What to observe? In the case of decision trees, one should observe the difference between performance measures (e.g., classification accuracy) on the training and validation sets.

The too-large difference implies an improperly fitting model.

How to Fit Decision Trees?

There are several approaches available for decision trees.

Generally, the overfitting can be either prevented or resolved.

How to Fit Decision Trees?

There are several approaches available for decision trees.

Generally, the overfitting can be either prevented or resolved.

- Pre-pruning: Build the tree so it does not overfit by restricting its size.

How to Fit Decision Trees?

There are several approaches available for decision trees.

Generally, the overfitting can be either prevented or resolved.

- Pre-pruning: Build the tree so it does not overfit by restricting its size.
- Post-pruning: Overfit with a large tree and remove subtrees to make it smaller.

How to Fit Decision Trees?

There are several approaches available for decision trees.

Generally, the overfitting can be either prevented or resolved.

- Pre-pruning: Build the tree so it does not overfit by restricting its size.
- Post-pruning: Overfit with a large tree and remove subtrees to make it smaller.
- Ensemble methods: Fit several different trees and let them classify together (e.g., using majority voting).

How to Fit Decision Trees?

There are several approaches available for decision trees.

Generally, the overfitting can be either prevented or resolved.

- Pre-pruning: Build the tree so it does not overfit by restricting its size.
- Post-pruning: Overfit with a large tree and remove subtrees to make it smaller.
- Ensemble methods: Fit several different trees and let them classify together (e.g., using majority voting).

The post-pruning approach has been more successful in practice than the pre-pruning because it is usually hard to say when to stop growing the tree.

We shall meet this controversy also in deep learning, where recent history shows a similar phenomenon.

The ensemble methods will be covered later when we discuss random forests.

Pre-Prunning - Hyperparamaters

Hyperparameters controlling the size of the tree:

- Maximum depth - do not grow the tree beyond the max depth

  The deeper the tree, the more complex models you can create $\Rightarrow$ overfitting. Low depth may restrict expressivity.

Pre-Prunning - Hyperparamaters

Hyperparameters controlling the size of the tree:

- Maximum depth - do not grow the tree beyond the max depth

  The deeper the tree, the more complex models you can create $\Rightarrow$ overfitting. Low depth may restrict expressivity.

- Minimum number of examples to split a node - if $\mathcal{D}[\tau]$ is small, τ becomes a leaf (labeled with the majority class)

  Higher values prevent a model from learning relations specific only for a few examples. Too high values may result in underfitting.

Pre-Prunning - Hyperparamaters

Hyperparameters controlling the size of the tree:

- Maximum depth - do not grow the tree beyond the max depth

  The deeper the tree, the more complex models you can create $\Rightarrow$ overfitting. Low depth may restrict expressivity.

- Minimum number of examples to split a node - if $\mathcal{D}[\tau]$ is small, τ becomes a leaf (labeled with the majority class)

  Higher values prevent a model from learning relations specific only for a few examples. Too high values may result in underfitting.

- Minimum number of examples required to be in a leaf

  Similar to the previous one. A higher number means we cannot have very specific branches concerned with particular combinations of values.

Pre-Prunning - Hyperparamaters

Hyperparameters controlling the size of the tree:

- Maximum depth - do not grow the tree beyond the max depth

  The deeper the tree, the more complex models you can create $\Rightarrow$ overfitting. Low depth may restrict expressivity.

- Minimum number of examples to split a node - if $\mathcal{D}[\tau]$ is small, τ becomes a leaf (labeled with the majority class)

  Higher values prevent a model from learning relations specific only for a few examples. Too high values may result in underfitting.

- Minimum number of examples required to be in a leaf

  Similar to the previous one. A higher number means we cannot have very specific branches concerned with particular combinations of values.

- Minimum information gain/impurity decrease

  A small impurity decrease means that the split does not contribute too much to the classification (their proportions after a split are similar to proportions before a split). However, keep in mind that it is *weighted average impurity* after the split.

Post-Pruning - Reduced Error Pruning

Train a large tree and then remove nodes that make classification worse on the validation set.

Post-Pruning - Reduced Error Pruning

Train a large tree and then remove nodes that make classification worse on the validation set.

Given a decision tree $\mathcal{T}$ and its internal node $\tau \in \mathcal{T}_{int}$, we denote by $\mathcal{T}_{-\tau}$ the tree obtained from $\mathcal{T}$ by removing the subtree rooted in τ, i.e., τ is a leaf of $\mathcal{T}_{-\tau}$.

Post-Pruning - Reduced Error Pruning

Train a large tree and then remove nodes that make classification worse on the validation set.

Given a decision tree $\mathcal{T}$ and its internal node $\tau \in \mathcal{T}_{int}$, we denote by $\mathcal{T}_{-\tau}$ the tree obtained from $\mathcal{T}$ by removing the subtree rooted in τ, i.e., τ is a leaf of $\mathcal{T}_{-\tau}$.

1: Train $\mathcal{T}$ to maximum fit on the *training dataset.*
2: **while** true **do**
3: $\quad Err[\mathcal{T}] \leftarrow$ the error of $\mathcal{T}$ on the validation set.
4: $\quad$ **for** $\tau \in \mathcal{T}_{int}$ **do**
5: $\quad\quad Err[\mathcal{T}_{-\tau}] \leftarrow$ the error of $\mathcal{T}_{-\tau}$ on the validation set.
6: $\quad$ **end for**
7: $\quad$ **if** $Err[\mathcal{T}] \leq \min\{Err[\mathcal{T}_{-\tau}] \mid \tau \in \mathcal{T}_{int}\}$ **then return** $\mathcal{T}$
8: $\quad$ **else**
9: $\quad\quad \mathcal{T} \leftarrow argmin\{Err[\mathcal{T}_{-\tau}] \mid \tau \in \mathcal{T}_{int}\}$
10: $\quad$ **end if**
11: **end while**

The error $Err[\mathcal{T}]$ can be any measure of the "badness" of the decision tree $\mathcal{T}$. For example, $1 - Accuracy$.

Other Pruning Methods

There are more pruning methods.

- Rule Post-Pruning:
  - Transform the tree into a set of rules.
    Rules correspond to paths in the tree; they have a form of implication: Specific values of attributes imply a class.
  - Remove the attribute conditions from the premises of the implications.

This gives a more refined pruning: Instead of removing the whole subtree, each path of the subtree can be pruned individually.

Other Pruning Methods

There are more pruning methods.

- Rule Post-Pruning:
  - Transform the tree into a set of rules.
    Rules correspond to paths in the tree; they have a form of implication: Specific values of attributes imply a class.
  - Remove the attribute conditions from the premises of the implications.

  This gives a more refined pruning: Instead of removing the whole subtree, each path of the subtree can be pruned individually.
- Using cost complexity measure: Evaluate trees not only based on the classification error but also based on their size.

Other Pruning Methods

There are more pruning methods.

- Rule Post-Pruning:
  - Transform the tree into a set of rules.
    Rules correspond to paths in the tree; they have a form of implication: Specific values of attributes imply a class.
  - Remove the attribute conditions from the premises of the implications.

  This gives a more refined pruning: Instead of removing the whole subtree, each path of the subtree can be pruned individually.
- Using cost complexity measure: Evaluate trees not only based on the classification error but also based on their size.

  Typically introduce regularization into the error functions: Given a decision tree $\mathcal{T}$

  $$Err_\alpha(\mathcal{T}) = Err(\mathcal{T}) + \alpha|\mathcal{T}|$$

  The original paper by Breiman et al. (1984) defined $Err(\mathcal{T})$ to be the misclassification rate on the training dataset, and $|\mathcal{T}|$ is the number of nodes of the tree $\mathcal{T}$.

Sensitivity to Small Changes and Randomness

- Decision trees are sensitive to small changes in data and hyperparameters.

  Several attributes may provide (almost) identical information gain but divide the training dataset very differently.
- Some implementations choose attributes partially in random (sci-kit-learn). You may get completely different trees.

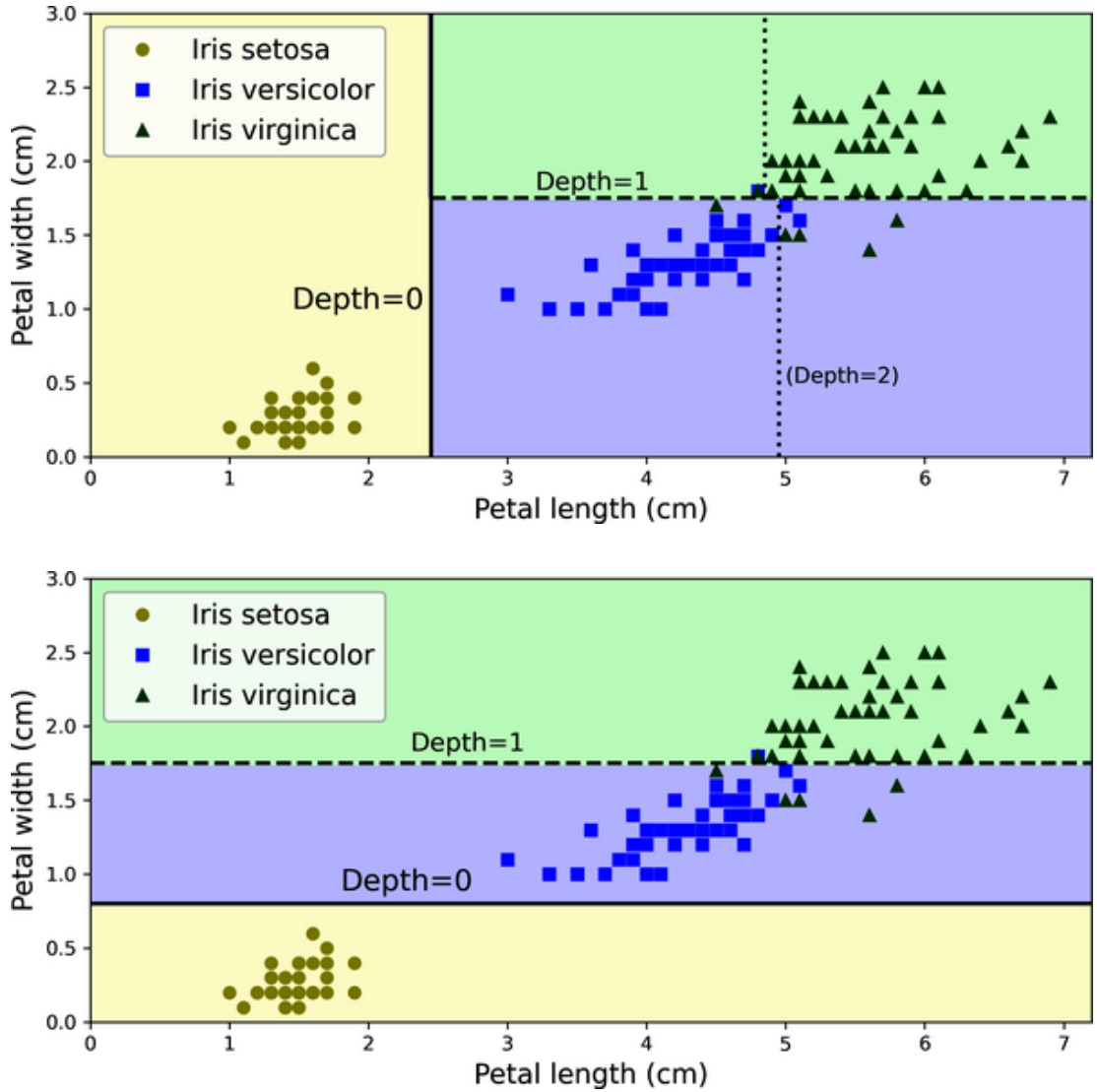
Sensitivity to Small Changes and Randomness

- Decision trees are sensitive to small changes in data and hyperparameters.
  Several attributes may provide (almost) identical information gain but divide the training dataset very differently.
- Some implementations choose attributes partially in random (sci-kit-learn). You may get completely different trees.

A solution is to train an ensemble of many decision trees and then use majority voting for classification.

This is the fundamental idea behind random forests (see later lectures).

Iris - Illustration

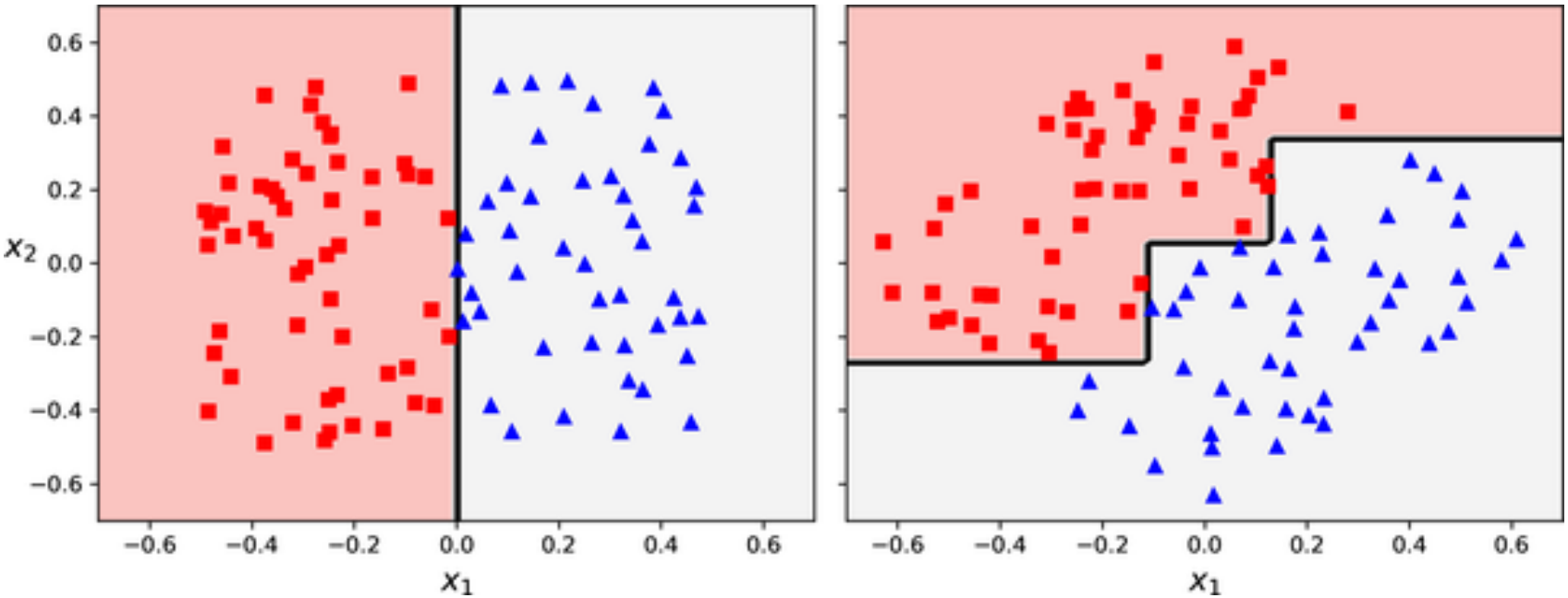


Decision trees trained on the Iris dataset.

Iris Setosa is perfectly separated by many choices for the first split.

Axis Sensitivity

The decision makes divisions along particular axes:

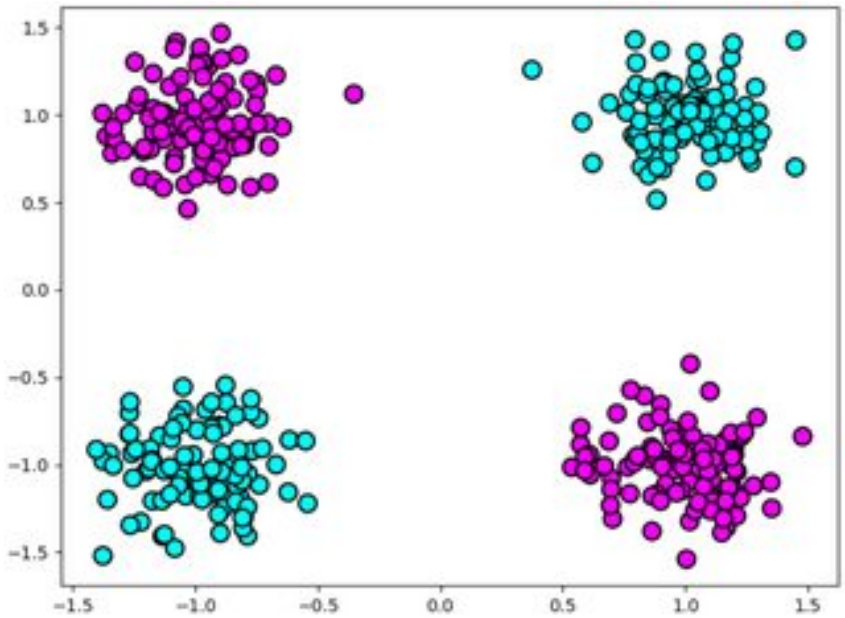


That is, rotated data may result in a completely different model.

That is why decision trees are often preceded by the *principal component analysis (PCA)* transformation, which aligns data along the axes of maximum data variance.

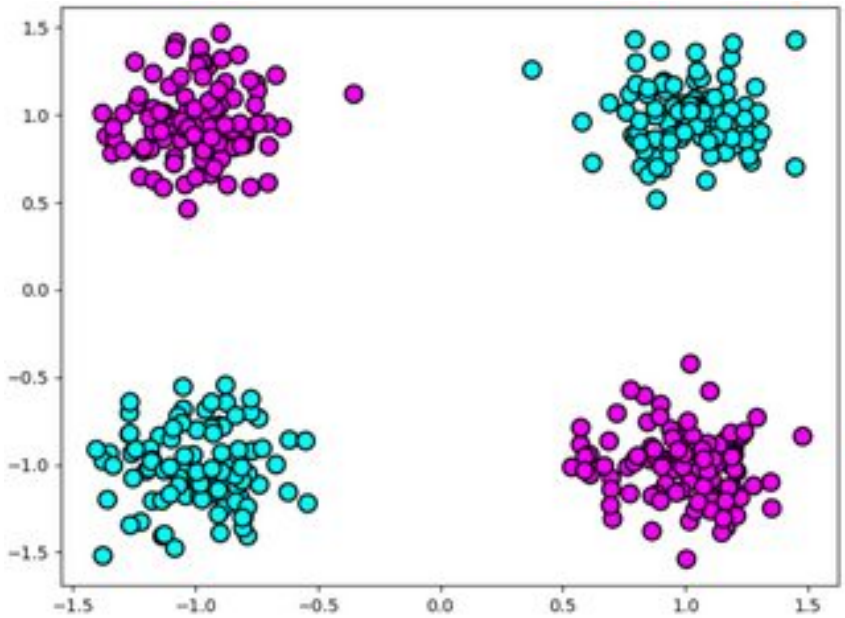
XOR Training Problem

Consider the following training dataset:

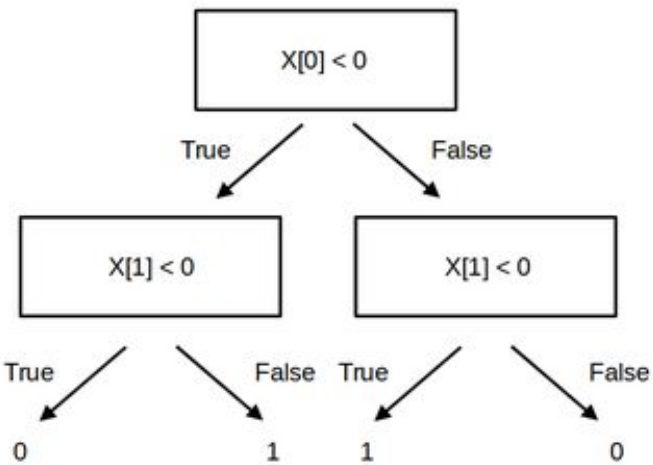


XOR Training Problem

Consider the following training dataset:

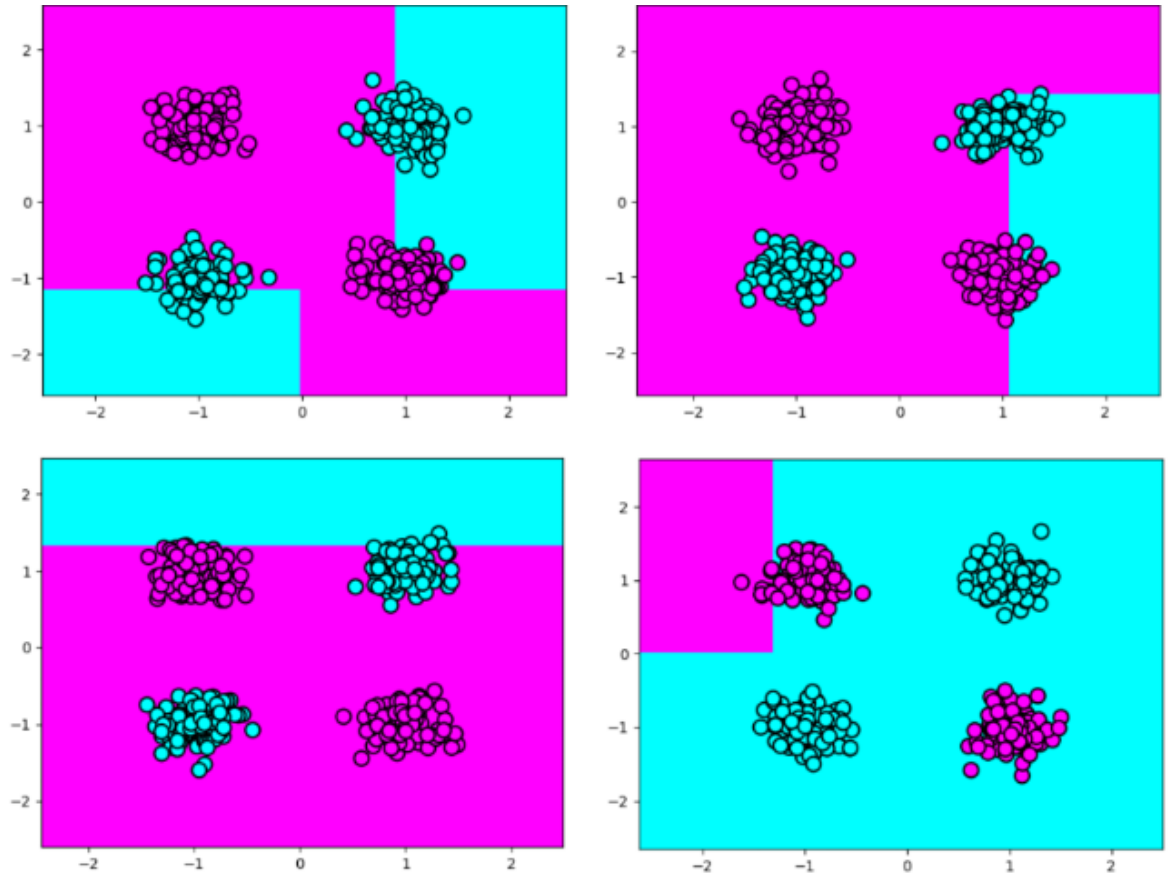


An ideal decision tree would look like this:



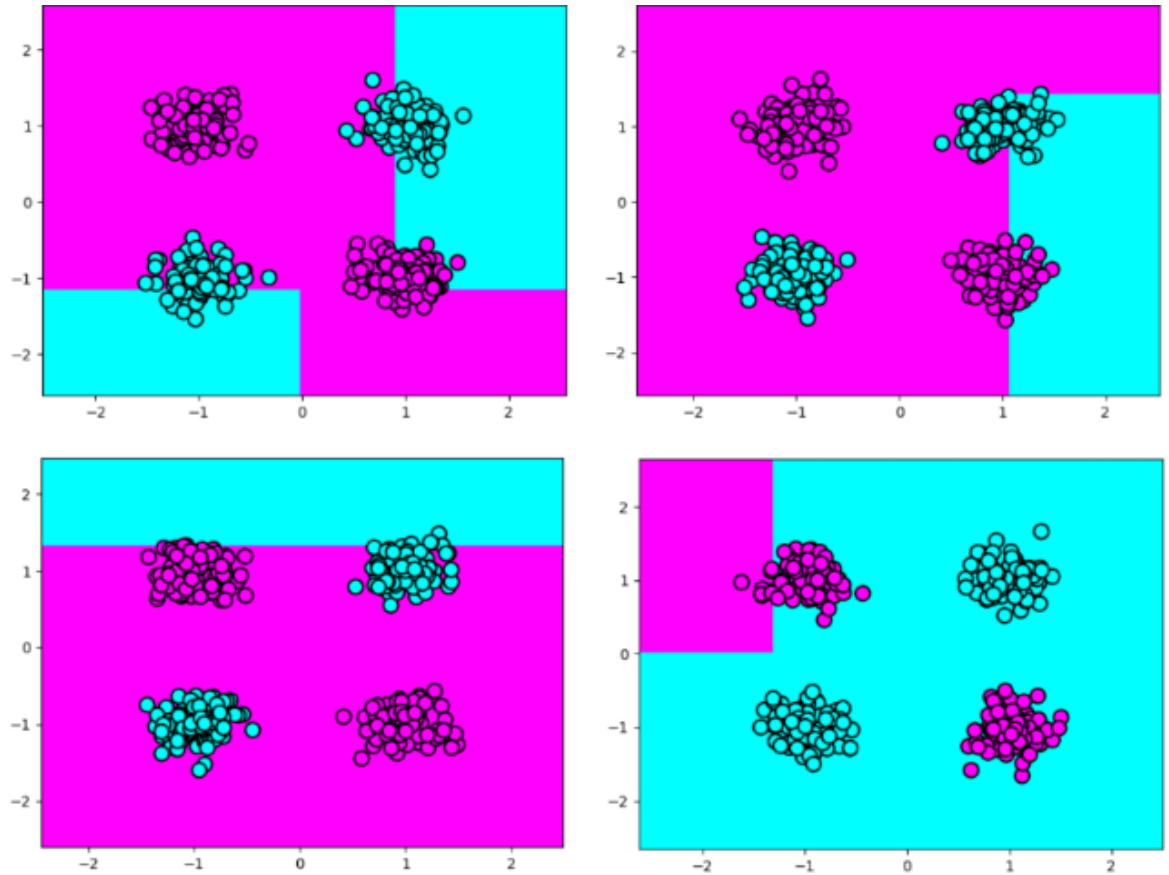
Attempts at Training on XOR

Max depth $= 2$:



Attempts at Training on XOR

Max depth = 2:

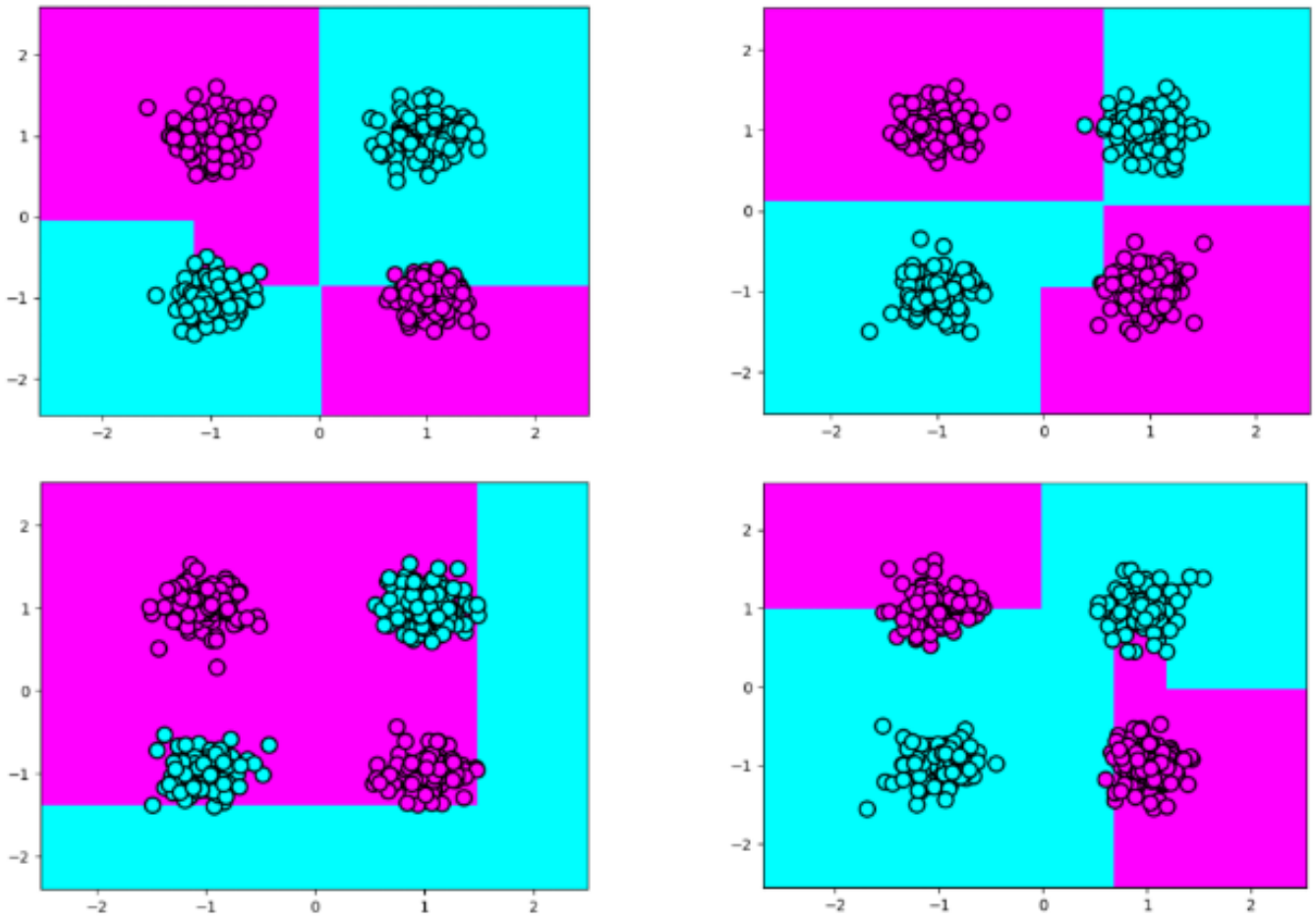


The problem: Both information gain and decrease in impurity consider only the relationship of a *single* attribute and the class.

However, there is no relationship between a single attribute and the class; both attributes need to be considered together!

More Attempts at Training on XOR

Max depth = 3:



It's better but still fails occasionally.

Advantages of Decision Trees

- Simple to understand and interpret; trees can be visualized.
- Uses a white box model, where conditions are easily explained by boolean logic.

Advantages of Decision Trees

- Simple to understand and interpret; trees can be visualized.
- Uses a white box model, where conditions are easily explained by boolean logic.
- Can approximate an arbitrary (reasonable) boundary and capture complex non-linear relationships between attributes.
- Capable of handling multi-class problems.

Advantages of Decision Trees

- Simple to understand and interpret; trees can be visualized.
- Uses a white box model, where conditions are easily explained by boolean logic.
- Can approximate an arbitrary (reasonable) boundary and capture complex non-linear relationships between attributes.
- Capable of handling multi-class problems.
- Little data preparation, unlike other techniques requiring normalization, dummy variables, or missing value removal.
- Handles numerical and categorical data.
- Not sensitive to outliers since the splitting is based on the proportion of examples within the split ranges and not on absolute values.

Advantages of Decision Trees

- Simple to understand and interpret; trees can be visualized.
- Uses a white box model, where conditions are easily explained by boolean logic.
- Can approximate an arbitrary (reasonable) boundary and capture complex non-linear relationships between attributes.
- Capable of handling multi-class problems.
- Little data preparation, unlike other techniques requiring normalization, dummy variables, or missing value removal.
- Handles numerical and categorical data.
- Not sensitive to outliers since the splitting is based on the proportion of examples within the split ranges and not on absolute values.
- The cost of using a well-balanced tree is logarithmic in the number of data points used to train it.

Disadvantages of Decision Trees

- Overfitting: Trees can be over-complex and not generalize well, needing pruning or limits on tree depth.
- Instability: Small data variations can result in very different trees. This is mitigated in ensemble methods.
- Non-smooth predictions: Decision trees make piecewise constant approximations, which are not suitable for extrapolation.

Disadvantages of Decision Trees

- Overfitting: Trees can be over-complex and not generalize well, needing pruning or limits on tree depth.
- Instability: Small data variations can result in very different trees. This is mitigated in ensemble methods.
- Non-smooth predictions: Decision trees make piecewise constant approximations, which are not suitable for extrapolation.
- Difficulty expressing certain concepts, such as XOR, parity, or multiplexer problems (see the next slide).
- Bias in trees: Decision trees can create biased trees if some classes dominate. Balancing the dataset is recommended.

Disadvantages of Decision Trees

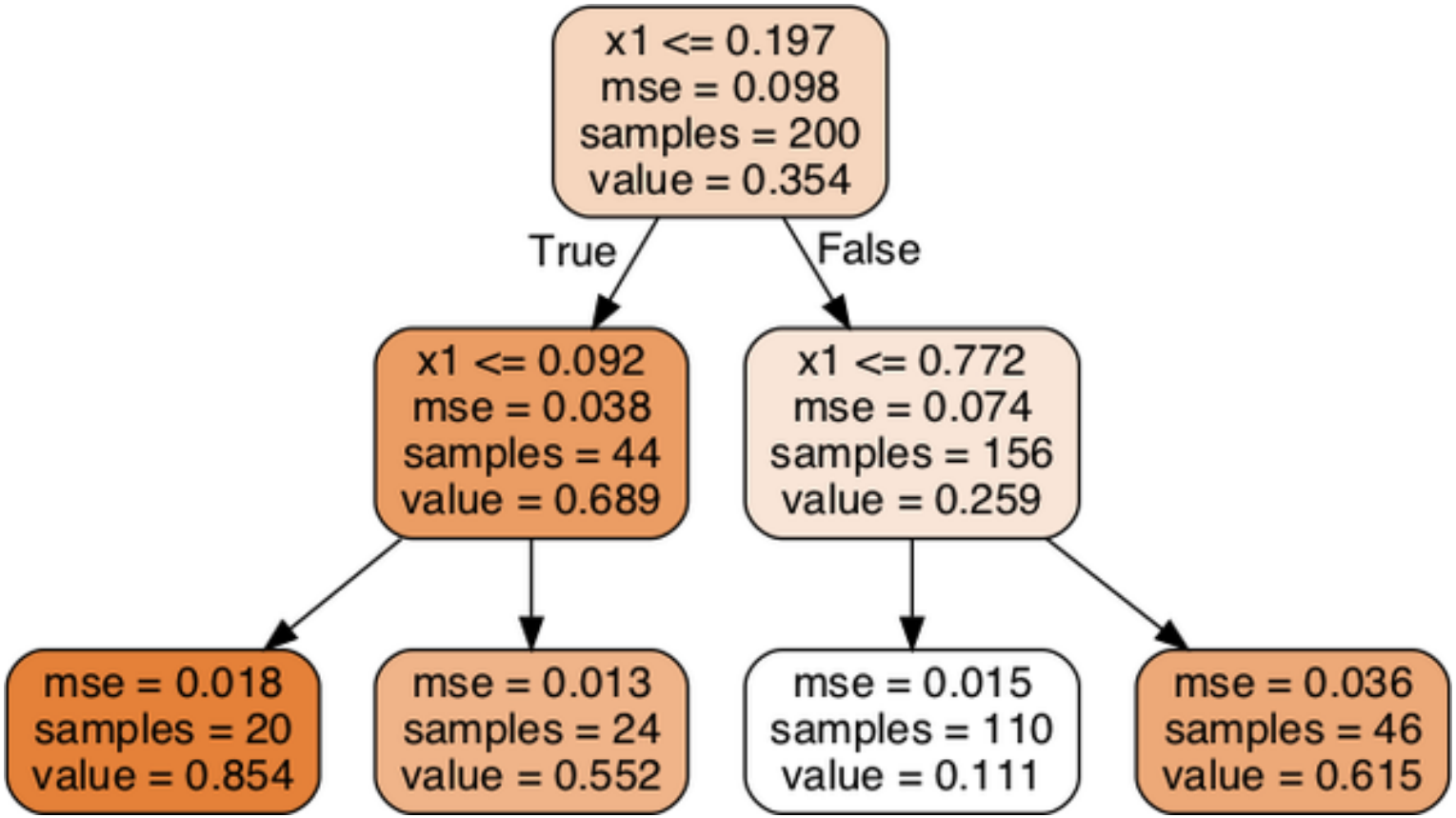
- Overfitting: Trees can be over-complex and not generalize well, needing pruning or limits on tree depth.
- Instability: Small data variations can result in very different trees. This is mitigated in ensemble methods.
- Non-smooth predictions: Decision trees make piecewise constant approximations, which are not suitable for extrapolation.
- Difficulty expressing certain concepts, such as XOR, parity, or multiplexer problems (see the next slide).
- Bias in trees: Decision trees can create biased trees if some classes dominate. Balancing the dataset is recommended.
- Learning optimal trees is NP-complete: Heuristic algorithms like greedy algorithms are used, which do not guarantee globally optimal trees. Ensemble methods can help.

History of Decision Trees

- Hunt and colleagues use exhaustive search decision-tree methods (CLS) to model human concept learning in the 1960's.
- In the late 70's, Quinlan developed ID3 with the information gain heuristic to learn expert systems from examples.
- Simultaneously, Breiman, Friedman, and colleagues develop CART (Classification and Regression Trees), similar to ID3.
- In the 1980s, various improvements were introduced to handle noise, continuous features, missing features, and improved splitting criteria. Various expert-system development tools results.
- Quinlan's updated decision-tree package (C4.5) released in 1993.

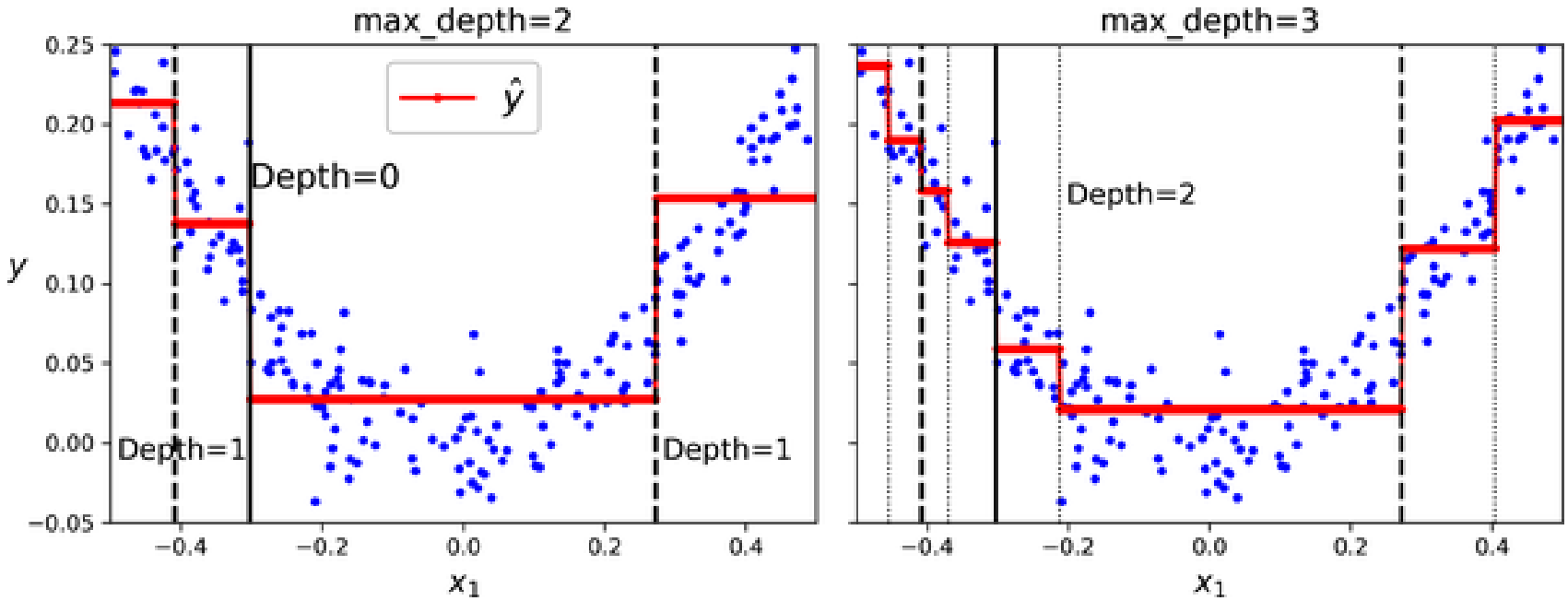
Comment on Regression Trees

Decision trees can also be used to approximate functions. Assign a function value to the leaves instead of classes.



Here, "mse" is the mean-squared-error.

Comment on Regression Trees



Intuitively, for every subinterval of x_1, the value (the red line) is at the average y over the subinterval.

How are the subintervals being set?

Regression Trees

A *regression tree* is a decision tree whose leaves are labeled by values from $\mathbb{R}$.

We follow the same procedure as in decision trees during inference on an input $\vec{x}$.

Regression Trees

A *regression tree* is a decision tree whose leaves are labeled by values from $\mathbb{R}$.

We follow the same procedure as in decision trees during inference on an input $\vec{x}$.

How exactly are such trees trained?

We aim to minimize the mean squared error.

Regression Trees

A *regression tree* is a decision tree whose leaves are labeled by values from $\mathbb{R}$.

We follow the same procedure as in decision trees during inference on an input $\vec{x}$.

How exactly are such trees trained?

We aim to minimize the mean squared error.

Assume, for the moment, that we want to train a single-node tree. What will be the best labeling value for the single node?

Regression Trees

A *regression tree* is a decision tree whose leaves are labeled by values from $\mathbb{R}$.

We follow the same procedure as in decision trees during inference on an input $\vec{x}$.

How exactly are such trees trained?

We aim to minimize the mean squared error.

Assume, for the moment, that we want to train a single-node tree. What will be the best labeling value for the single node?

Now, consider the training for arbitrary trees.

Assume ordinal attributes.

The algorithm also works for discrete attributes; ordinal attributes, however, allow us to make binary splits.

Regression Trees

A *regression tree* is a decision tree whose leaves are labeled by values from $\mathbb{R}$.

We follow the same procedure as in decision trees during inference on an input $\vec{x}$.

How exactly are such trees trained?

We aim to minimize the mean squared error.

Assume, for the moment, that we want to train a single-node tree. What will be the best labeling value for the single node?

Now, consider the training for arbitrary trees.

Assume ordinal attributes.

The algorithm also works for discrete attributes; ordinal attributes, however, allow us to make binary splits.

The training procedure is the same as for the decision trees, except that the splits and cut points are selected differently.

Regression Trees

Given a dataset $\mathcal{D} = \{(\vec{x}_1, d_1), \ldots, (\vec{x}_p, d_p)\}$, we denote by $\bar{\mathcal{D}}$ the average *desired* value in $\mathcal{D}$, that is $\bar{\mathcal{D}} = \frac{1}{p} \sum_{k=1}^{p} d_k$.

Regression Trees

Given a dataset $\mathcal{D} = \{(\vec{x}_1, d_1), \ldots, (\vec{x}_p, d_p)\}$, we denote by $\bar{\mathcal{D}}$ the average *desired* value in $\mathcal{D}$, that is $\bar{\mathcal{D}} = \frac{1}{p} \sum_{k=1}^{p} d_k$.

Consider a dataset $\mathcal{D}$ and let us find a cut point for A in $\mathcal{D}$.

Regression Trees

Given a dataset $\mathcal{D} = \{(\vec{x}_1, d_1), \ldots, (\vec{x}_p, d_p)\}$, we denote by $\bar{\mathcal{D}}$ the average *desired* value in $\mathcal{D}$, that is $\bar{\mathcal{D}} = \frac{1}{p}\sum_{k=1}^{p} d_k$.

Consider a dataset $\mathcal{D}$ and let us find a cut point for A in $\mathcal{D}$.

We are looking for a value H of the attribute A such that the split:

$$\mathcal{D}_{\leq H} = \{(\vec{x}, d) \in \mathcal{D} \mid A(\vec{x}) \leq H\} \quad \mathcal{D}_{>H} = \{(\vec{x}, d) \in \mathcal{D} \mid A(\vec{x}) > H\}$$

Minimizes the following *split error*:

$$\frac{1}{|\mathcal{D}_{\leq H}|} \sum_{(\vec{x},d)\in\mathcal{D}_{\leq H}} \left(d - \bar{\mathcal{D}}_{\leq H}\right)^2 + \frac{1}{|\mathcal{D}_{>H}|} \sum_{(\vec{x},d)\in\mathcal{D}_{>H}} \left(d - \bar{\mathcal{D}}_{>H}\right)^2$$

Regression Trees

Given a dataset $\mathcal{D} = \{(\vec{x}_1, d_1), \ldots, (\vec{x}_p, d_p)\}$, we denote by $\bar{\mathcal{D}}$ the average *desired* value in $\mathcal{D}$, that is $\bar{\mathcal{D}} = \frac{1}{p}\sum_{k=1}^{p} d_k$.

Consider a dataset $\mathcal{D}$ and let us find a cut point for A in $\mathcal{D}$.

We are looking for a value H of the attribute A such that the split:

$$\mathcal{D}_{\leq H} = \{(\vec{x}, d) \in \mathcal{D} \mid A(\vec{x}) \leq H\} \quad \mathcal{D}_{>H} = \{(\vec{x}, d) \in \mathcal{D} \mid A(\vec{x}) > H\}$$

Minimizes the following *split error*:

$$\frac{1}{|\mathcal{D}_{\leq H}|} \sum_{(\vec{x},d)\in\mathcal{D}_{\leq H}} \left(d - \bar{\mathcal{D}}_{\leq H}\right)^2 + \frac{1}{|\mathcal{D}_{>H}|} \sum_{(\vec{x},d)\in\mathcal{D}_{>H}} \left(d - \bar{\mathcal{D}}_{>H}\right)^2$$

Denote by $\mathrm{MinE}(\mathcal{D})$ the minimum of the split error over all attributes A and all cut points H.

Regression Trees

Given a dataset $\mathcal{D} = \{(\vec{x}_1, d_1), \ldots, (\vec{x}_p, d_p)\}$, we denote by $\bar{\mathcal{D}}$ the average *desired* value in $\mathcal{D}$, that is $\bar{\mathcal{D}} = \frac{1}{p}\sum_{k=1}^{p} d_k$.

Consider a dataset $\mathcal{D}$ and let us find a cut point for A in $\mathcal{D}$.

We are looking for a value H of the attribute A such that the split:

$$\mathcal{D}_{\leq H} = \{(\vec{x}, d) \in \mathcal{D} \mid A(\vec{x}) \leq H\} \quad \mathcal{D}_{>H} = \{(\vec{x}, d) \in \mathcal{D} \mid A(\vec{x}) > H\}$$

Minimizes the following *split error*:

$$\frac{1}{|\mathcal{D}_{\leq H}|} \sum_{(\vec{x},d)\in\mathcal{D}_{\leq H}} \left(d - \bar{\mathcal{D}}_{\leq H}\right)^2 + \frac{1}{|\mathcal{D}_{>H}|} \sum_{(\vec{x},d)\in\mathcal{D}_{>H}} \left(d - \bar{\mathcal{D}}_{>H}\right)^2$$

Denote by MinE($\mathcal{D}$) the minimum of the split error over all attributes A and all cut points H.Compute

$$\Delta = \left(\frac{1}{|\mathcal{D}|} \sum_{(\vec{x},d)\in\mathcal{D}} \left(d - \bar{\mathcal{D}}\right)^2\right) - \text{MinE}$$

Regression Trees

Given a dataset $\mathcal{D} = \{(\vec{x}_1, d_1), \ldots, (\vec{x}_p, d_p)\}$, we denote by $\bar{\mathcal{D}}$ the average *desired* value in $\mathcal{D}$, that is $\bar{\mathcal{D}} = \frac{1}{p}\sum_{k=1}^{p} d_k$.

Consider a dataset $\mathcal{D}$ and let us find a cut point for A in $\mathcal{D}$.

We are looking for a value H of the attribute A such that the split:

$$\mathcal{D}_{\leq H} = \{(\vec{x}, d) \in \mathcal{D} \mid A(\vec{x}) \leq H\} \quad \mathcal{D}_{>H} = \{(\vec{x}, d) \in \mathcal{D} \mid A(\vec{x}) > H\}$$

Minimizes the following *split error*:

$$\frac{1}{|\mathcal{D}_{\leq H}|} \sum_{(\vec{x},d)\in\mathcal{D}_{\leq H}} \left(d - \bar{\mathcal{D}}_{\leq H}\right)^2 + \frac{1}{|\mathcal{D}_{>H}|} \sum_{(\vec{x},d)\in\mathcal{D}_{>H}} \left(d - \bar{\mathcal{D}}_{>H}\right)^2$$

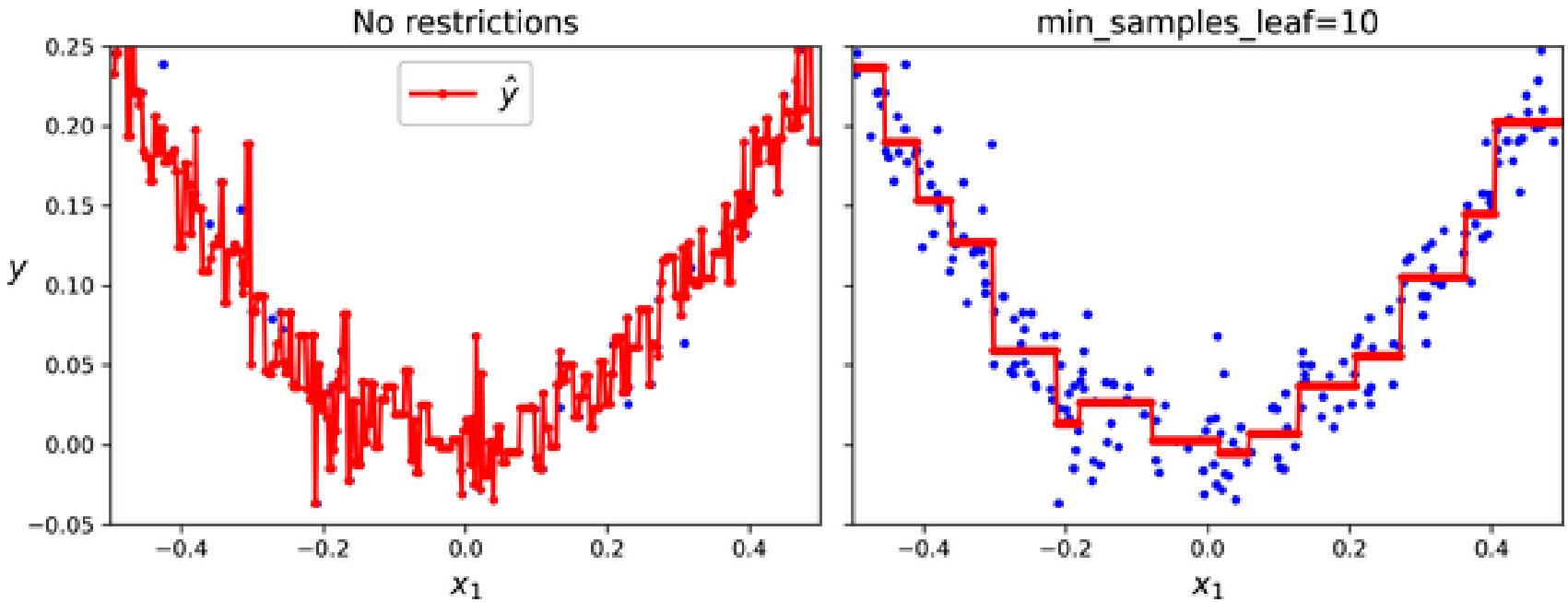
Denote by MinE($\mathcal{D}$) the minimum of the split error over all attributes A and all cut points H.Compute

$$\Delta = \left(\frac{1}{|\mathcal{D}|} \sum_{(\vec{x},d)\in\mathcal{D}} \left(d - \bar{\mathcal{D}}\right)^2\right) - \text{MinE}$$

If Δ is large enough, split on A and H that minimize the split error. Otherwise, stop splitting and label the leaf with $\bar{D}$.

Regression Tress

Without any lower bound on the number of examples in the leaves, the algorithm will eventually overfit by splitting into (possibly) singleton leaves.



Anomaly Detection

What is an Anomaly?

Anomalies are hard to define in general.

What is an Anomaly?

Anomalies are hard to define in general.

Attempts at a generic definition include:

- Observations that seem to be generated by a different mechanism.

What is an Anomaly?

Anomalies are hard to define in general.

Attempts at a generic definition include:

- Observations that seem to be generated by a different mechanism.
- Data instances that occur rarely and whose features differ significantly from most data.

What is an Anomaly?

Anomalies are hard to define in general.

Attempts at a generic definition include:

- Observations that seem to be generated by a different mechanism.
- Data instances that occur rarely and whose features differ significantly from most data.
- Outliers - observations that appear to be inconsistent (far away) with the remainder of the data

  However, inliers are not covered, that is, anomalous data that lie within the normal data distribution (say 0 in measurement results that are very likely non-zero but distributed around 0).

What is an Anomaly?

Anomalies are hard to define in general.

Attempts at a generic definition include:

- Observations that seem to be generated by a different mechanism.
- Data instances that occur rarely and whose features differ significantly from most data.
- Outliers - observations that appear to be inconsistent (far away) with the remainder of the data

  However, inliers are not covered, that is, anomalous data that lie within the normal data distribution (say 0 in measurement results that are very likely non-zero but distributed around 0).
- Patterns in data that do not conform to a well-defined notion of normal behavior(??)

What is an Anomaly?

Anomalies are hard to define in general.

Attempts at a generic definition include:

- Observations that seem to be generated by a different mechanism.
- Data instances that occur rarely and whose features differ significantly from most data.
- Outliers - observations that appear to be inconsistent (far away) with the remainder of the data

  However, inliers are not covered, that is, anomalous data that lie within the normal data distribution (say 0 in measurement results that are very likely non-zero but distributed around 0).
- Patterns in data that do not conform to a well-defined notion of normal behavior(??)

What is an Anomaly?

Anomalies are hard to define in general.

Attempts at a generic definition include:

- Observations that seem to be generated by a different mechanism.
- Data instances that occur rarely and whose features differ significantly from most data.
- Outliers - observations that appear to be inconsistent (far away) with the remainder of the data

  However, inliers are not covered, that is, anomalous data that lie within the normal data distribution (say 0 in measurement results that are very likely non-zero but distributed around 0).
- Patterns in data that do not conform to a well-defined notion of normal behavior(??)

This lecture presents algorithms detecting specific data instances that may be anomalous w.r.t. the used algorithm.

It is up to the context-knowing user to decide whether such data are anomalous.

What is the Anomaly Detection Good For?

There are two main sources of motivation for anomaly detection in machine learning:

- *Modeling perspective*: Many models are sensitive to anomalous data.

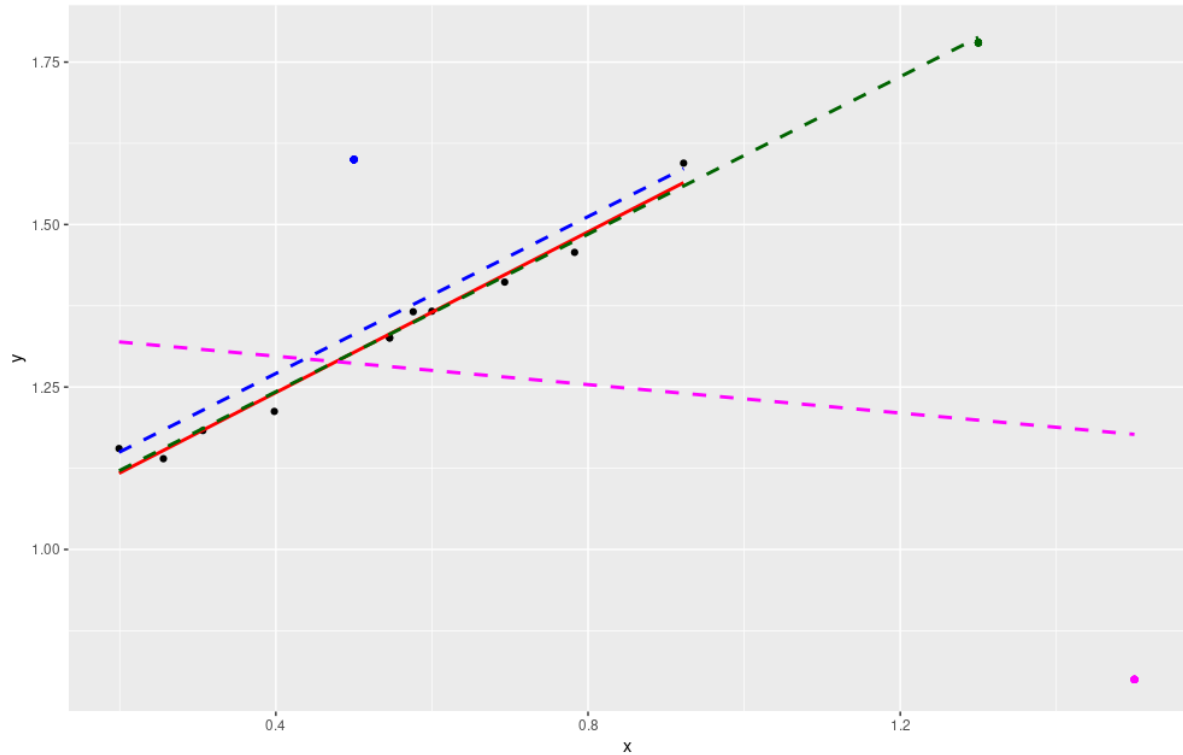
What is the Anomaly Detection Good For?

There are two main sources of motivation for anomaly detection in machine learning:

- *Modeling perspective*: Many models are sensitive to anomalous data.
- *Application perspective*: Many tasks are based on searching for anomalies in data.

Modeling Perspective

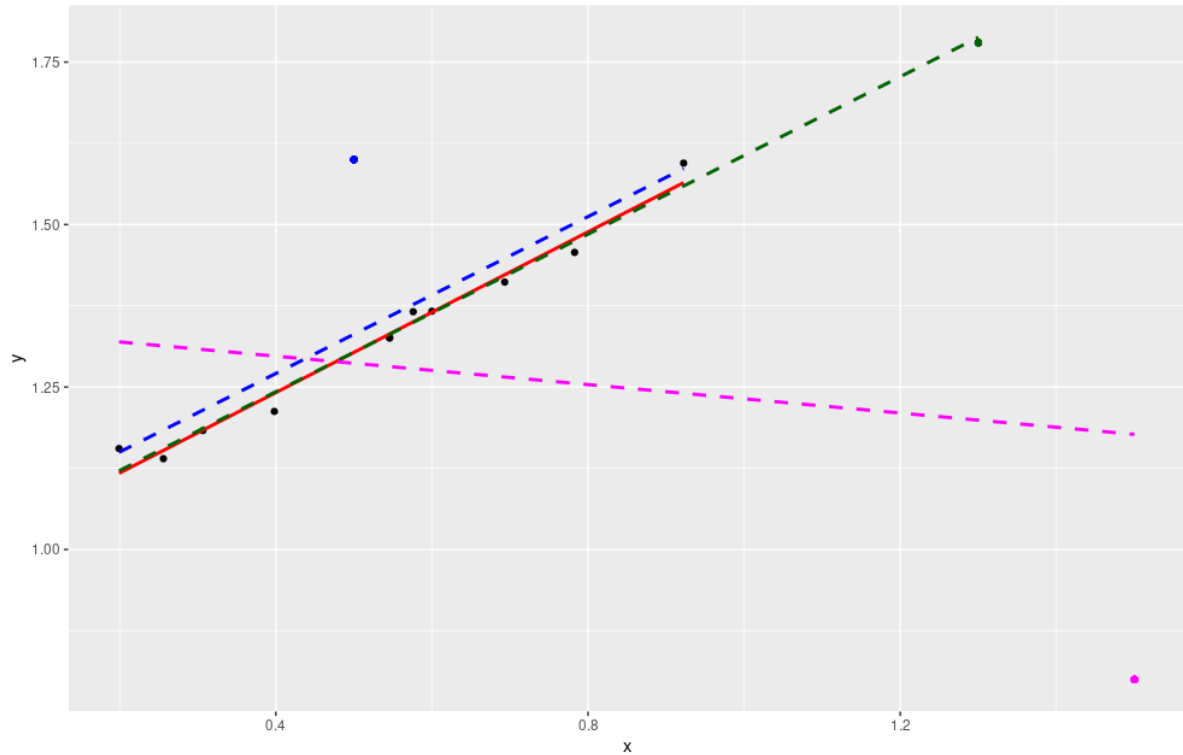
Recall the behavior of the linear model.



To train such a model properly, the outliers should be removed.

Modeling Perspective

Recall the behavior of the linear model.



To train such a model properly, the outliers should be removed.

On the other hand, we will see that the sensitivity of some models can be used to *detect* the anomalies.

Applications Perspective

- **Fraud Detection**
  - Analysis of purchasing behavior to detect credit card theft.
  - Identification of theft through uncharacteristic buying patterns and behavioral changes.

Applications Perspective

- **Fraud Detection**
  - Analysis of purchasing behavior to detect credit card theft.
  - Identification of theft through uncharacteristic buying patterns and behavioral changes.
- **Intrusion Detection**
  - Monitoring for attacks on computer systems and networks.
  - Many attacks exhibit themselves as anomalous behavior.

Applications Perspective

- **Fraud Detection**
  - Analysis of purchasing behavior to detect credit card theft.
  - Identification of theft through uncharacteristic buying patterns and behavioral changes.
- **Intrusion Detection**
  - Monitoring for attacks on computer systems and networks.
  - Many attacks exhibit themselves as anomalous behavior.
- **Ecosystem Disturbances**
  - Detect atypical natural world events.
  - Prediction and understanding of hurricanes, floods, droughts, heat waves, and fires.

Applications Perspective

- **Fraud Detection**
  - Analysis of purchasing behavior to detect credit card theft.
  - Identification of theft through uncharacteristic buying patterns and behavioral changes.
- **Intrusion Detection**
  - Monitoring for attacks on computer systems and networks.
  - Many attacks exhibit themselves as anomalous behavior.
- **Ecosystem Disturbances**
  - Detect atypical natural world events.
  - Prediction and understanding of hurricanes, floods, droughts, heat waves, and fires.
- **Medicine**
  - Unusual patient symptoms or test results may indicate health problems.
  - Balancing the need for further tests with the potential costs and risks.
- ...

Causes of Anomalies in Data Preprocessing

Anomaly detection is a standard step in data preprocessing.

Causes of Anomalies in Data Preprocessing

Anomaly detection is a standard step in data preprocessing.

We typically search for anomalies from the following sources:

- Objects from different classes: Pear among apples.

Causes of Anomalies in Data Preprocessing

Anomaly detection is a standard step in data preprocessing.

We typically search for anomalies from the following sources:

- Objects from different classes: Pear among apples.
- Natural variation: Abnormally tall person - not from a different class (humans) but in the sense of an extreme characteristic.

Causes of Anomalies in Data Preprocessing

Anomaly detection is a standard step in data preprocessing.

We typically search for anomalies from the following sources:

- Objects from different classes: Pear among apples.
- Natural variation: Abnormally tall person - not from a different class (humans) but in the sense of an extreme characteristic.
- Data measurement and collection error:
  - Thermometer positioned on the direct sun (possibly a measurement error)
  - 40 kg infant is not usual among humans (possibly a data collection error)

Causes of Anomalies in Data Preprocessing

Anomaly detection is a standard step in data preprocessing.

We typically search for anomalies from the following sources:

- Objects from different classes: Pear among apples.
- Natural variation: Abnormally tall person - not from a different class (humans) but in the sense of an extreme characteristic.
- Data measurement and collection error:
  - Thermometer positioned on the direct sun (possibly a measurement error)
  - 40 kg infant is not usual among humans (possibly a data collection error)

The anomalies mentioned above should be inspected by domain experts and possibly corrected/removed.

Approaches to Anomaly Detection

- Model-based techniques:
  - Build a model of data.
  - Anomalies should not fit the model very well.
    For example, using a clustering model, an anomaly may lie far away from larger clusters.
  - Alternatively, removing anomalies should have the strongest impact on the model parameters (more on this later).

Approaches to Anomaly Detection

- Model-based techniques:
  - Build a model of data.
  - Anomalies should not fit the model very well.
    For example, using a clustering model, an anomaly may lie far away from larger clusters.
  - Alternatively, removing anomalies should have the strongest impact on the model parameters (more on this later).
- Proximity-based techniques: Given distance between objects, anomalies are objects distant from others.

Approaches to Anomaly Detection

- Model-based techniques:
  - Build a model of data.
  - Anomalies should not fit the model very well.
    For example, using a clustering model, an anomaly may lie far away from larger clusters.
  - Alternatively, removing anomalies should have the strongest impact on the model parameters (more on this later).
- Proximity-based techniques: Given distance between objects, anomalies are objects distant from others.
- Density-based techniques: Estimate the density distribution of the objects. Anomalies would probably be outside of dense regions.

I would count the most recent deep learning-based anomaly detection among the model-based techniques even though a combination of the above approaches is usually used.

We will make the above ideas more precise in the rest of this lecture.

Anomaly Detecion Learning

There are three approaches to anomaly detection based on available information about data:

- Supervised: Given a training set distinguishing normal and anomalous data. We may train a supervised learning classifier.

  As anomalies are rare, approaches based on supervised learning are rare.

Anomaly Detecion Learning

There are three approaches to anomaly detection based on available information about data:

- Supervised: Given a training set distinguishing normal and anomalous data. We may train a supervised learning classifier.

  As anomalies are rare, approaches based on supervised learning are rare.
- Semi-supervised:
  - We may know what instances are normal.

    A tumor detection system trained on healthy people detects tumors as anomalies.

    In this case, we detect anomalies that are dissimilar to normal instances. We may train a model representing the normal instances and detect instances that do not fit the model.
  - We may have information about (some) normal and some anomalous instances. Here, we can use methods for semi-supervised classification.

Anomaly Detecion Learning

There are three approaches to anomaly detection based on available information about data:

- Supervised: Given a training set distinguishing normal and anomalous data. We may train a supervised learning classifier.
  As anomalies are rare, approaches based on supervised learning are rare.
- Semi-supervised:
  - We may know what instances are normal.
    A tumor detection system trained on healthy people detects tumors as anomalies.
    In this case, we detect anomalies that are dissimilar to normal instances. We may train a model representing the normal instances and detect instances that do not fit the model.
  - We may have information about (some) normal and some anomalous instances. Here, we can use methods for semi-supervised classification.
- Unsupervised: Create a model of all instances and hope that anomalous instances will still be dissimilar to instances typical for the model.

Issues

Task-specific issues comprise:

- Single vs. multi-attribute anomaly: The question is whether an object is anomalous due to a single attribute (200 kg person) or a combination of attributes (100 kg person of 1 meter height).

  This is especially important when data is high-dimensional. For example, in genetic data, every sample is an anomaly(?)

Issues

Task-specific issues comprise:

- Single vs. multi-attribute anomaly: The question is whether an object is anomalous due to a single attribute (200 kg person) or a combination of attributes (100 kg person of 1 meter height).

  This is especially important when data is high-dimensional. For example, in genetic data, every sample is an anomaly(?)

- Global vs. local perspective: An object may be anomalous w.r.t. all objects but not w.r.t. objects in its local neighborhood (210 cm high basketball player).

Issues

Task-specific issues comprise:

- Single vs. multi-attribute anomaly: The question is whether an object is anomalous due to a single attribute (200 kg person) or a combination of attributes (100 kg person of 1 meter height).

  This is especially important when data is high-dimensional. For example, in genetic data, every sample is an anomaly(?)

- Global vs. local perspective: An object may be anomalous w.r.t. all objects but not w.r.t. objects in its local neighborhood (210 cm high basketball player).
- Degree of anomaly: Usually, anomalies are reported in a binary fashion. However, we often want to measure how anomalous an object is, similar to normal objects.

Issues

Task-specific issues comprise:

- Single vs. multi-attribute anomaly: The question is whether an object is anomalous due to a single attribute (200 kg person) or a combination of attributes (100 kg person of 1 meter height).

  This is especially important when data is high-dimensional. For example, in genetic data, every sample is an anomaly(?)

- Global vs. local perspective: An object may be anomalous w.r.t. all objects but not w.r.t. objects in its local neighborhood (210 cm high basketball player).
- Degree of anomaly: Usually, anomalies are reported in a binary fashion. However, we often want to measure how anomalous an object is, similar to normal objects.

Another issue is evaluation: How can we determine how good an anomaly detector is?

In the supervised case, we have a ground truth, but usually, we just observe detected anomalies.

Various Approaches to Anomaly Detection

We shall have a look at five approaches to the unsupervised anomaly detection:

- Statistical
- Proximity-based
- Density-based
- Cluster-based
- Autoencoders

The above approaches are also used in the semi-supervised setting where we have a dataset of normal instances. However, some issues discussed further will not appear in this case.

Statistical

Definition 1

An outlier is an object with a low probability in the probability distribution of the data.

Statistical

Definition 1

An outlier is an object with a low probability in the probability distribution of the data.

Suppose we knew the "true" probability distribution P on objects. In that case, we may set up a threshold t (using an appropriate training set) and say that every object A with $P(A) < t$ is anomalous.

Statistical

Definition 1

An outlier is an object with a low probability in the probability distribution of the data.

Suppose we knew the "true" probability distribution P on objects. In that case, we may set up a threshold t (using an appropriate training set) and say that every object A with $P(A) < t$ is anomalous.

However, in most cases, we do not know P and have to resort to a model of P created using a dataset of feature vectors.

Statistical

Definition 1

An outlier is an object with a low probability in the probability distribution of the data.

Suppose we knew the "true" probability distribution P on objects. In that case, we may set up a threshold t (using an appropriate training set) and say that every object A with $P(A) < t$ is anomalous.

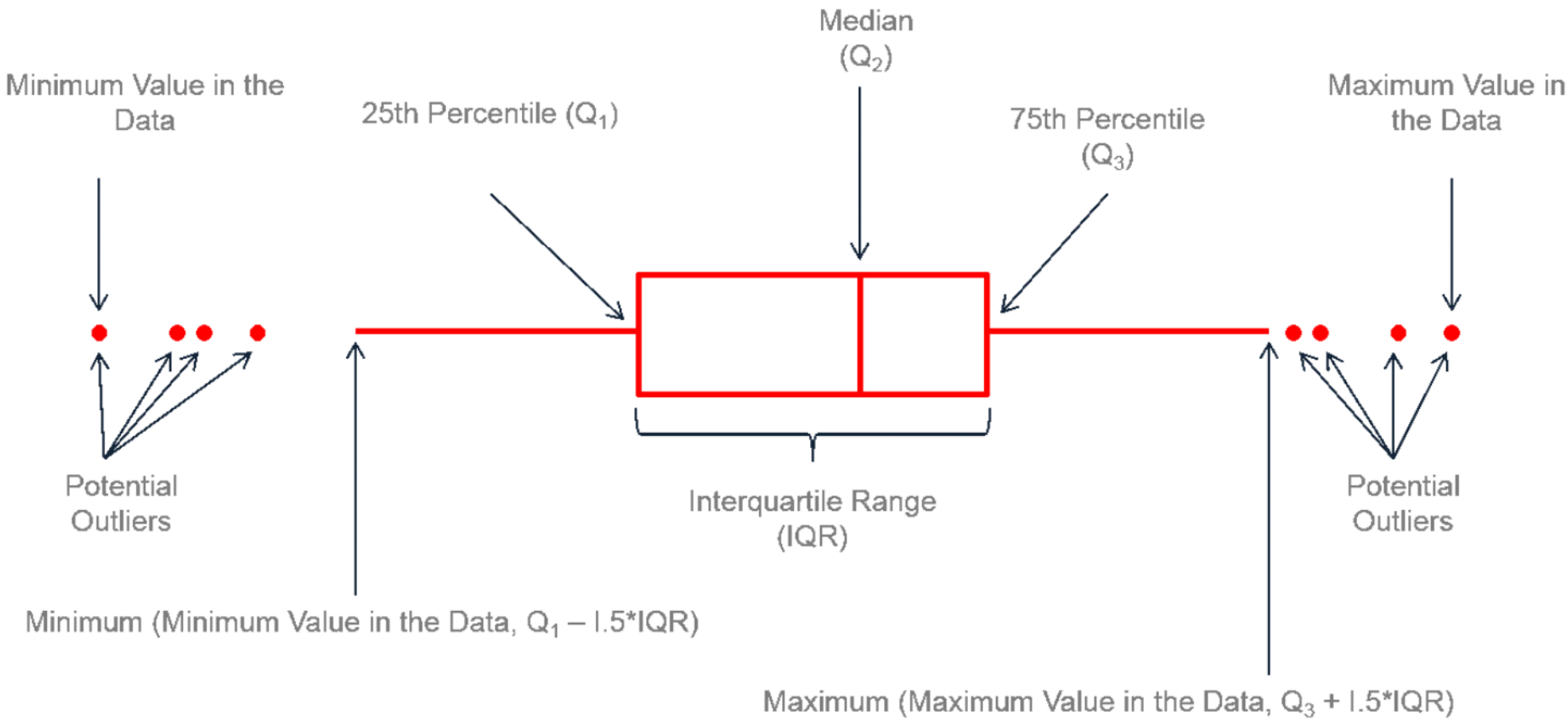
However, in most cases, we do not know P and have to resort to a model of P created using a dataset of feature vectors.

There are many more or less sophisticated tests based on models of P in the literature.

Some use rather advanced statistical methods.

Let us have a look at a few simple examples.

Box Plot



A simple method for outlier detection in the *univariate* case, that is, for values of a single attribute.

Univariate Gaussian Model

Consider a numeric attribute whose values x are normally distributed with mean μ and standard deviation σ.

We write $N(\mu, \sigma^2)$.

Univariate Gaussian Model

Consider a numeric attribute whose values x are normally distributed with mean μ and standard deviation σ.

We write $N(\mu, \sigma^2)$.

Given an attribute value x, we compute the z-score:

$$z = (x - \mu)/\sigma$$

Then z has the distribution $N(0, 1)$.

Univariate Gaussian Model

Consider a numeric attribute whose values x are normally distributed with mean μ and standard deviation σ.

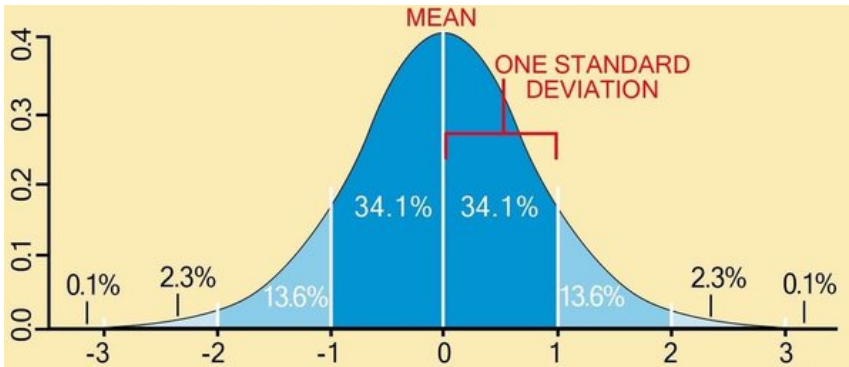
We write $N(\mu, \sigma^2)$.

Given an attribute value x, we compute the z-score:

$$z = (x - \mu)/\sigma$$

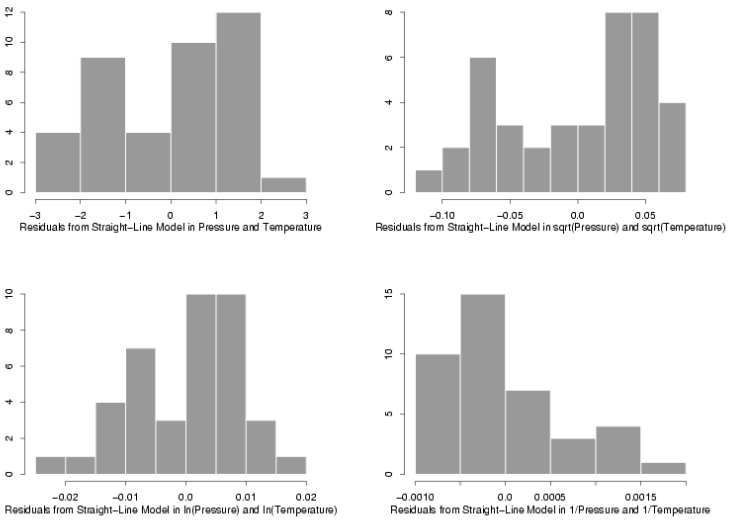
Then z has the distribution $N(0, 1)$.

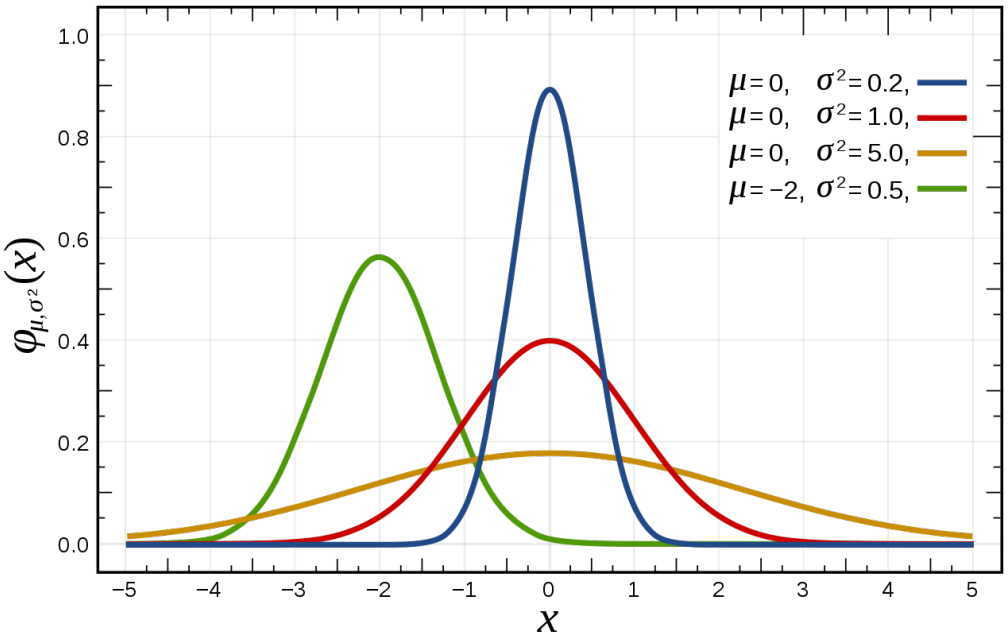
Now, choose c so the probability $P(|z| \geq c)$ is small enough for z to be an outlier w.r.t. $N(0, 1)$.



Given an attribute value x, we may decide whether x is an outlier by deciding whether $|z| = |(x - \mu)/\sigma| \geq c$.

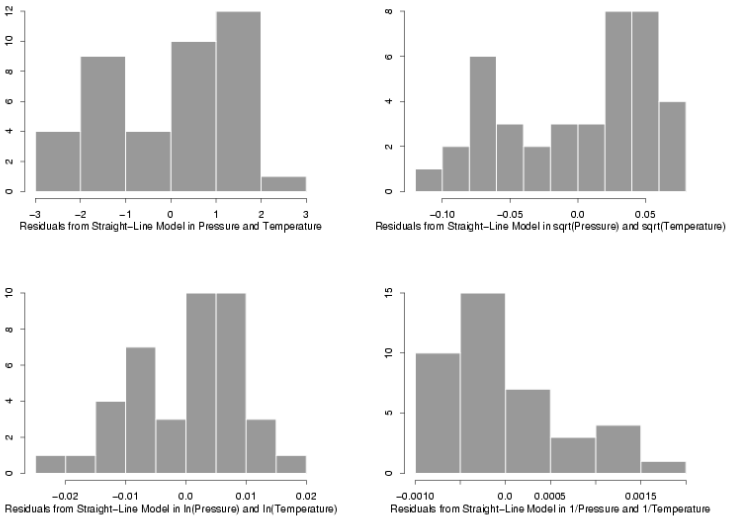
Univariate Gaussian

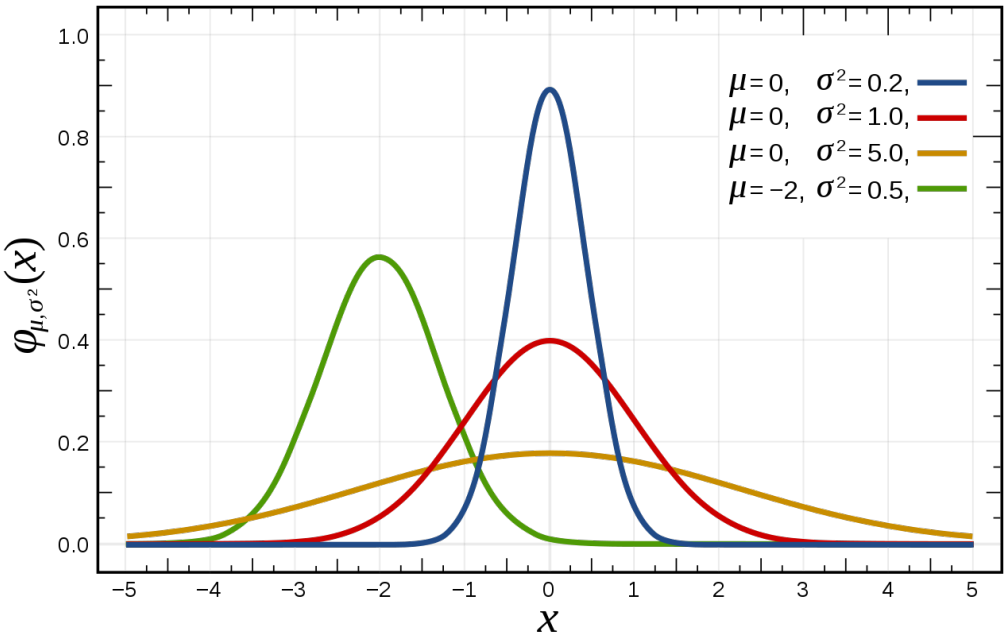




Problem 1: The attribute does not have a normal distribution.

Univariate Gaussian





Problem 1: The attribute does not have a normal distribution.

Before starting, use a normality test (observe the histogram, use a specialized test such as Shapiro-Wilk).

Some transformations may sometimes succeed in normalizing an attribute (Box-Cox transformation, etc.)

Different distributions can be used to model the attribute (Log Normal, Weibull, etc.), but be aware of the assumptions of anomaly detection tests!

Univariate Gaussian

Problem 2: We typically do not know the population mean μ and the population variance σ^2.

Univariate Gaussian

Problem 2: We typically do not know the population mean μ and the population variance σ^2.

They can be estimated from a dataset by the sample mean and the sample variance:

$$\bar{\mu} = \frac{1}{p}\sum_{i=1}^{p} x_i \qquad s^2 = \frac{1}{p-1}\sum_{i=1}^{p}(x_i - \bar{\mu})^2$$

However, then $z = (x - \bar{\mu})/s$ does no longer have the distribution $N(0, 1)$.

The distribution of z is normal if x is sampled independently of the data yielding $\bar{\mu}$ and s^2.

Univariate Gaussian

Problem 2: We typically do not know the population mean μ and the population variance σ^2.

They can be estimated from a dataset by the sample mean and the sample variance:

$$\bar{\mu} = \frac{1}{p}\sum_{i=1}^{p} x_i \qquad s^2 = \frac{1}{p-1}\sum_{i=1}^{p}(x_i - \bar{\mu})^2$$

However, then $z = (x - \bar{\mu})/s$ does no longer have the distribution $N(0, 1)$.

The distribution of z is normal if x is sampled independently of the data yielding $\bar{\mu}$ and s^2.

Note that $\bar{\mu}$ and s^2 are unbiased estimates of μ and σ^2. Also, $\bar{\mu}$ converges to μ and s^2 converges to σ^2 with growing sample size.

Univariate Gaussian Model

Problem 3: The outliers distort the estimates $\bar{\mu}$ and s^2 of the mean and the standard deviation.

For example, a millionaire would not look like an outlier in a group containing a billionaire.

There is a circularity here: To get outliers using the normal distribution model, we need to remove the outliers to have a good model.

Univariate Gaussian Model

Problem 3: The outliers distort the estimates $\bar{\mu}$ and s^2 of the mean and the standard deviation.

For example, a millionaire would not look like an outlier in a group containing a billionaire.

There is a circularity here: To get outliers using the normal distribution model, we need to remove the outliers to have a good model.

One possible heuristic is to remove the outliers one by one, starting from the most extreme, hoping the most extreme outlier will be detected in every step.

One such heuristic is the Grubb's test.

Grubb's Test

Consider a dataset $D = \{x_1, \ldots, x_p\}$ of values of a normally distributed attribute and choose $\alpha > 0$.

Grubb's Test

Consider a dataset $D = \{x_1, \ldots, x_p\}$ of values of a normally distributed attribute and choose $\alpha > 0$.

The *Grubb's statistic* is defined by

$$G = \frac{\max_{i=1,\ldots,p} |x_i - \bar{x}|}{s}$$

Here $\bar{x}$ is the sample mean and s sample standard deviation of D.

Grubb's Test

Consider a dataset $D = \{x_1, \ldots, x_p\}$ of values of a normally distributed attribute and choose $\alpha > 0$.

The *Grubb's statistic* is defined by

$$G = \frac{\max_{i=1,\ldots,p} |x_i - \bar{x}|}{s}$$

Here $\bar{x}$ is the sample mean and s sample standard deviation of D.

Now $P(G \geq c_{\alpha,p}) = \alpha$ if

$$c_{\alpha,p} = \frac{p-1}{\sqrt{p}} \sqrt{\frac{t^2}{p-2+t^2}}$$

Here t is such a value that makes $P(T \geq t) = \alpha/(2p)$ for T with the t-distribution with $p-2$ degrees of freedom.

For finding t we used to use tables.

Grubb's Test

Consider a dataset $D = \{x_1, \ldots, x_p\}$ of values of a normally distributed attribute and choose $\alpha > 0$.

The *Grubb's statistic* is defined by

$$G = \frac{\max_{i=1,\ldots,p} |x_i - \bar{x}|}{s}$$

Here $\bar{x}$ is the sample mean and s sample standard deviation of D.

Now $P(G \geq c_{\alpha,p}) = \alpha$ if

$$c_{\alpha,p} = \frac{p-1}{\sqrt{p}} \sqrt{\frac{t^2}{p-2+t^2}}$$

Here t is such a value that makes $P(T \geq t) = \alpha/(2p)$ for T with the t-distribution with $p-2$ degrees of freedom.

For finding t we used to use tables.

Grubb's test simply iteratively tests whether $G \geq c_{\alpha,p}$ and, if yes, removes an x_i maximizing $|x_i - \bar{x}|$ from D.

Multivariate Gaussian Model

Univariate tests cannot find all anomalies if the anomaly depends on combinations of particular attribute values.

Multivariate Gaussian Model

Univariate tests cannot find all anomalies if the anomaly depends on combinations of particular attribute values.

The univariate Gaussian approach can be generalized to the multivariate Gaussian by considering the multivariate Gaussian distribution.

Multivariate Gaussian Model

Univariate tests cannot find all anomalies if the anomaly depends on combinations of particular attribute values.

The univariate Gaussian approach can be generalized to the multivariate Gaussian by considering the multivariate Gaussian distribution.

Mahalanobis distance:

$$\text{mahalanobis}(\vec{x}, \vec{z}) = (\vec{x} - \vec{z})\Sigma^{-1}(\vec{x} - \vec{z})^{\top}$$

Here Σ is the covariance matrix of the data.

Multivariate Gaussian Model

Univariate tests cannot find all anomalies if the anomaly depends on combinations of particular attribute values.

The univariate Gaussian approach can be generalized to the multivariate Gaussian by considering the multivariate Gaussian distribution.

Mahalanobis distance:

$$\text{mahalanobis}(\vec{x}, \vec{z}) = (\vec{x} - \vec{z})\Sigma^{-1}(\vec{x} - \vec{z})^{\top}$$

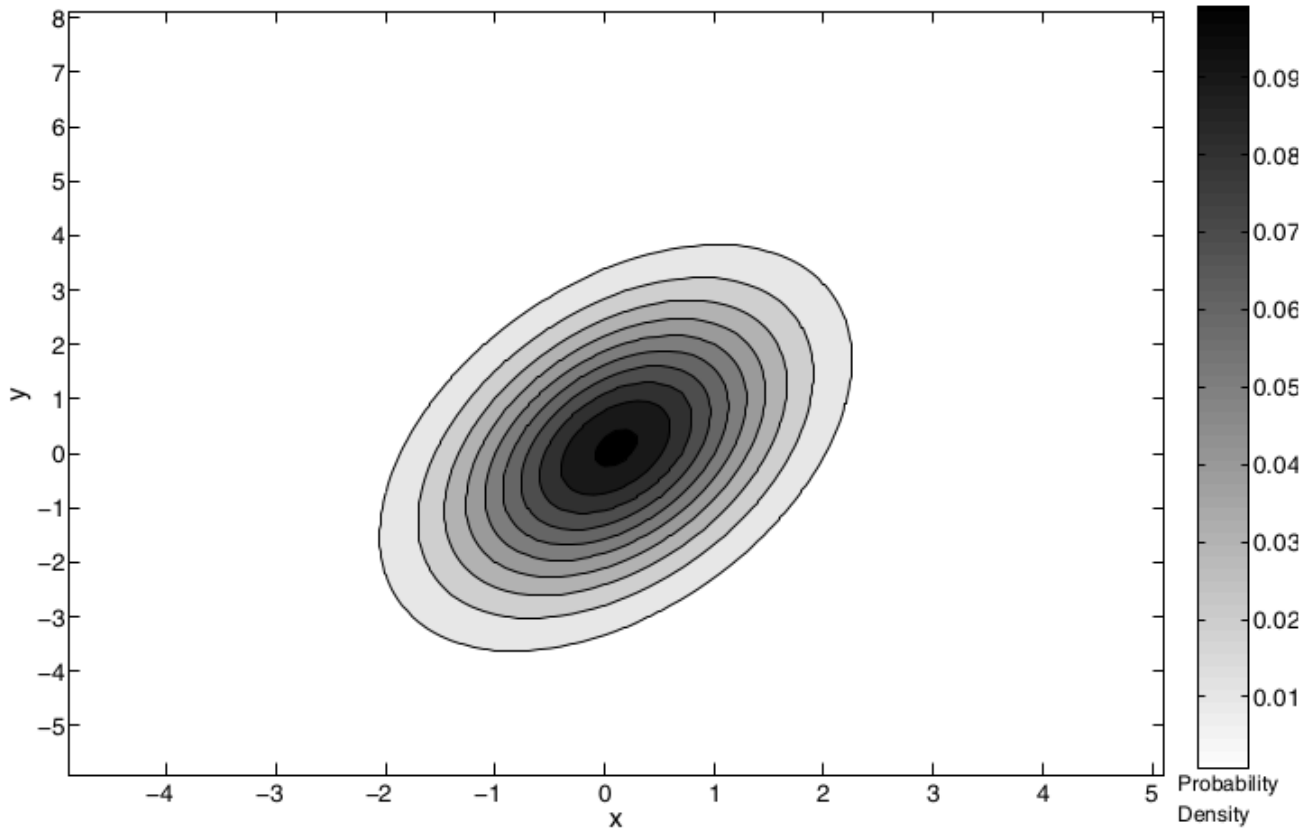
Here Σ is the covariance matrix of the data.

Intuitively, the Mahalanobis distance generalizes the squared Euclidean distance:

$$(\vec{x} - \vec{z})(\vec{x} - \vec{z})^{\top}$$

By taking into account the "spread" of the data.

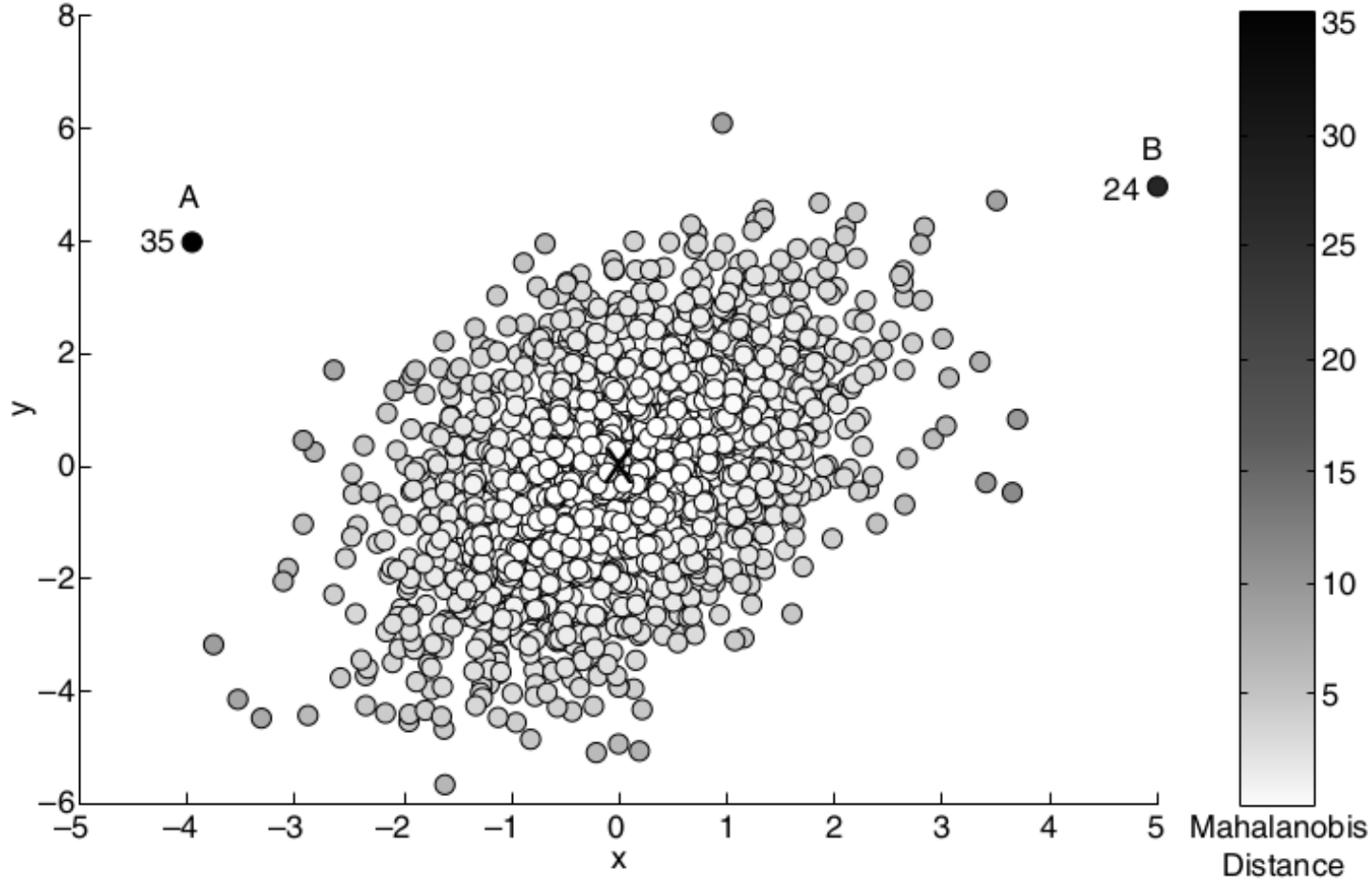
Mahalanobis Distance



Here, we assume multivariate normal data distribution. The covariance matrix is

$$\mathbf{\Sigma} = \begin{pmatrix} 1.00 & 0.75 \\ 0.75 & 3.00 \end{pmatrix}$$

Mahalanobis Distance



Notice that A has a larger Mahalanobis distance from the cluster's center than B even though it is closer in the Euclidean distance.

The reason is that the density function falls more rapidly in the direction of A than in the direction of B.

Likelihood-Based Detection

Assume that the dataset D contains objects from a mixture of two probability distributions:

- M, the distribution of the majority of normal objects,
- A, the distribution of anomalous objects.

Likelihood-Based Detection

Assume that the dataset D contains objects from a mixture of two probability distributions:

- M, the distribution of the majority of normal objects,
- A, the distribution of anomalous objects.

Let us choose $\alpha > 0$, indicating how rare the anomalies should be.

Likelihood-Based Detection

Assume that the dataset D contains objects from a mixture of two probability distributions:

- M, the distribution of the majority of normal objects,
- A, the distribution of anomalous objects.

Let us choose $\alpha > 0$, indicating how rare the anomalies should be.

Let us assume that we have M and A models to compute $P_M(x)$ and $P_A(x)$ the likelihoods of generating x.

A is often considered uniform, and M is learned from the data (yes, distorted by the outliers).

Likelihood-Based Detection

Assume that the dataset D contains objects from a mixture of two probability distributions:

- M, the distribution of the majority of normal objects,
- A, the distribution of anomalous objects.

Let us choose $\alpha > 0$, indicating how rare the anomalies should be.

Let us assume that we have M and A models to compute $P_M(x)$ and $P_A(x)$ the likelihoods of generating x.

A is often considered uniform, and M is learned from the data (yes, distorted by the outliers).

Now, we compute the total likelihood of the dataset before and after removing each element. If the likelihood changes significantly, we have probably eliminated an anomaly.

Given a partition U, V of D we define

$$L(U, V) = \left((1 - \alpha)^{|U|} \prod_{x_i \in U} P_M(x_i) \right) \left(\alpha^{|V|} \prod_{x_i \in V} P_A(x_i) \right)$$

Given a partition U, V of D we define

$$L(U, V) = \left((1-\alpha)^{|U|} \prod_{x_i \in U} P_M(x_i) \right) \left(\alpha^{|V|} \prod_{x_i \in V} P_A(x_i) \right)$$

This is the likelihood of generating D using the following random process:

Given a partition U, V of D we define

$$L(U,V) = \left((1-\alpha)^{|U|} \prod_{x_i \in U} P_M(x_i)\right) \left(\alpha^{|V|} \prod_{x_i \in V} P_A(x_i)\right)$$

This is the likelihood of generating D using the following random process: Generate $x_1, x_2, \ldots, x_p$ sequentially and independently where each x_i is generated as follows:

Given a partition U, V of D we define

$$L(U,V) = \left((1-\alpha)^{|U|} \prod_{x_i \in U} P_M(x_i)\right) \left(\alpha^{|V|} \prod_{x_i \in V} P_A(x_i)\right)$$

This is the likelihood of generating D using the following random process: Generate $x_1, x_2, \ldots, x_p$ sequentially and independently where each x_i is generated as follows:

- Randomly choose between normal and anomaly, where the probability of choosing anomaly is α.

Given a partition U, V of D we define

$$L(U,V) = \left((1-\alpha)^{|U|} \prod_{x_i \in U} P_M(x_i)\right) \left(\alpha^{|V|} \prod_{x_i \in V} P_A(x_i)\right)$$

This is the likelihood of generating D using the following random process: Generate $x_1, x_2, \ldots, x_p$ sequentially and independently where each x_i is generated as follows:

- Randomly choose between normal and anomaly, where the probability of choosing anomaly is α.
- If normal has been chosen, choose x_i from the distribution M.
- If anomaly has been chosen, choose x_i from the distribution A.

Given a partition U, V of D we define

$$L(U,V) = \left((1-\alpha)^{|U|} \prod_{x_i \in U} P_M(x_i)\right)\left(\alpha^{|V|} \prod_{x_i \in V} P_A(x_i)\right)$$

This is the likelihood of generating D using the following random process: Generate $x_1, x_2, \ldots, x_p$ sequentially and independently where each x_i is generated as follows:

- Randomly choose between normal and anomaly, where the probability of choosing anomaly is α.
- If normal has been chosen, choose x_i from the distribution M.
- If anomaly has been chosen, choose x_i from the distribution A.

To eliminate the "large" product, we consider the log-likelihood:

$$\begin{aligned} LL(U,V) &= \log(L(U,V)) \\ &= |U| \log(1-\alpha) + \sum_{x_i \in U} \log P_M(x_i) \\ &\quad + |V| \log \alpha + \sum_{x_i \in V} \log P_A(x_i) \end{aligned}$$

Likelihood-Based Anomaly Detection

Start with sets $M_0 = D$ and $A_0 = \emptyset$.

Likelihood-Based Anomaly Detection

Start with sets $M_0 = D$ and $A_0 = \emptyset$.

The algorithm computes $M_1, M_2, \ldots$ and $A_1, A_2, \ldots$ by moving elements from M_k to A_k as follows:

Likelihood-Based Anomaly Detection

Start with sets $M_0 = D$ and $A_0 = \emptyset$.

The algorithm computes $M_1, M_2, \ldots$ and $A_1, A_2, \ldots$ by moving elements from M_k to A_k as follows:

- For every $i = 1, \ldots, p$ such that $x_i \in M_k$ compute

$$\Delta_i = |LL(M_k, A_k) - LL(M_k \setminus \{x_i\}, A_k \cup \{x_i\})|$$

Likelihood-Based Anomaly Detection

Start with sets $M_0 = D$ and $A_0 = \emptyset$.

The algorithm computes $M_1, M_2, \ldots$ and $A_1, A_2, \ldots$ by moving elements from M_k to A_k as follows:

- For every $i = 1, \ldots, p$ such that $x_i \in M_k$ compute

$$\Delta_i = |LL(M_k, A_k) - LL(M_k \setminus \{x_i\}, A_k \cup \{x_i\})|$$

- Consider i maximizing Δ_i. If $\Delta_i \geq c$, then

$$M_{k+1} = M_k \setminus \{x_i\} \qquad A_{k+1} = A_k \cup \{x_i\}$$

Else, stop.

Here, c is a threshold we must set up.

Likelihood-Based Anomaly Detection

Start with sets $M_0 = D$ and $A_0 = \emptyset$.

The algorithm computes $M_1, M_2, \ldots$ and $A_1, A_2, \ldots$ by moving elements from M_k to A_k as follows:

- For every $i = 1, \ldots, p$ such that $x_i \in M_k$ compute

$$\Delta_i = |LL(M_k, A_k) - LL(M_k \setminus \{x_i\}, A_k \cup \{x_i\})|$$

- Consider i maximizing Δ_i. If $\Delta_i \geq c$, then

$$M_{k+1} = M_k \setminus \{x_i\} \qquad A_{k+1} = A_k \cup \{x_i\}$$

  Else, stop.

  Here, c is a threshold we must set up.

Ultimately, the A_k will contain the anomalies the algorithm detects.

Note that we may also move all anomalies detected in a single iteration from M_k to A_k. This would result in a different method (also valid).

Summary of Statistical Anomaly Detection

- Build on strong statistical foundations.
- Tests are very effective if the dataset is sufficiently large (informative).
- Lots of tests for univariate data, the area has developed for a long time.
- Fewer options for multivariate data and problematic for highly dimensional data (curse of dimensionality).

The likelihood-based approach does not assume a particular distribution shape: It can be used with arbitrary models of P_M and P_A, including deep learning ones.

Proximity-Based Methods

Proximity-Based Outlier Detection

Assume a distance measure d. That is, given two feature vectors $\vec{x}, \vec{z}$ their distance is $d(\vec{x}, \vec{z})$.

We consider the Euclidean distance for simplicity.

Definition 2

The outlier score of an object is given by the distance to its k-nearest neighbor.

Proximity-Based Outlier Detection

Assume a distance measure d. That is, given two feature vectors $\vec{x}, \vec{z}$ their distance is $d(\vec{x}, \vec{z})$.

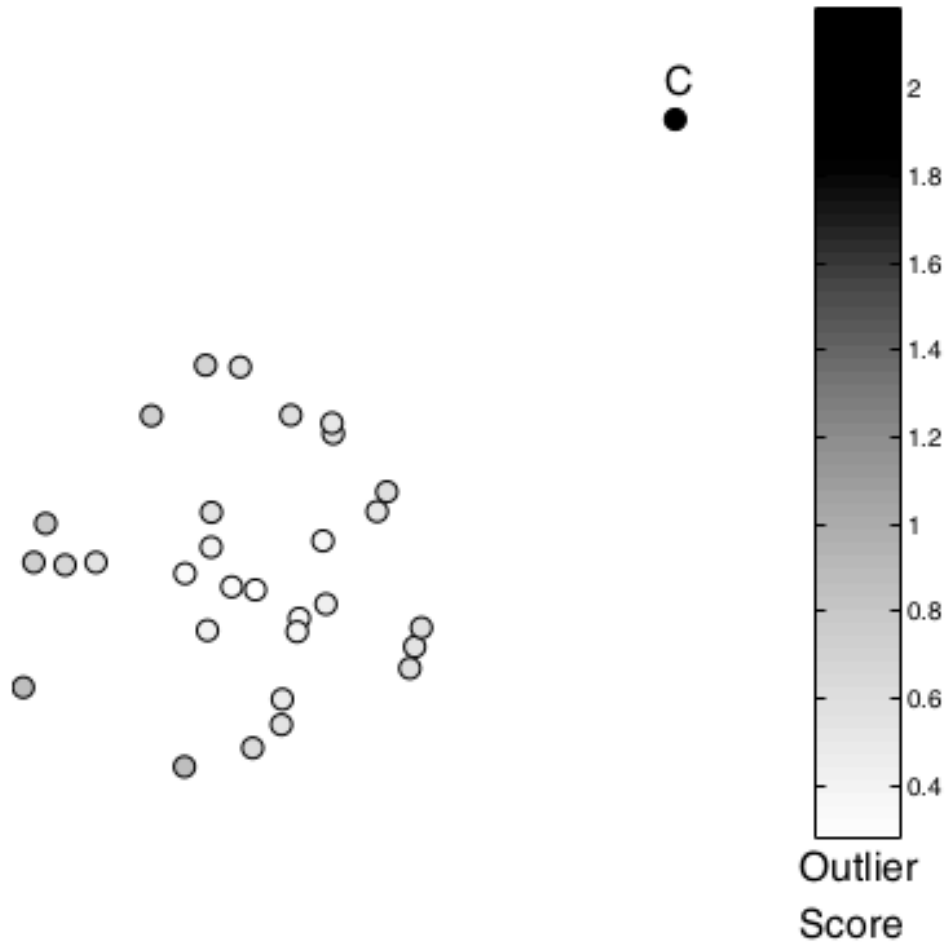
We consider the Euclidean distance for simplicity.

Definition 2

The outlier score of an object is given by the distance to its k-nearest neighbor.

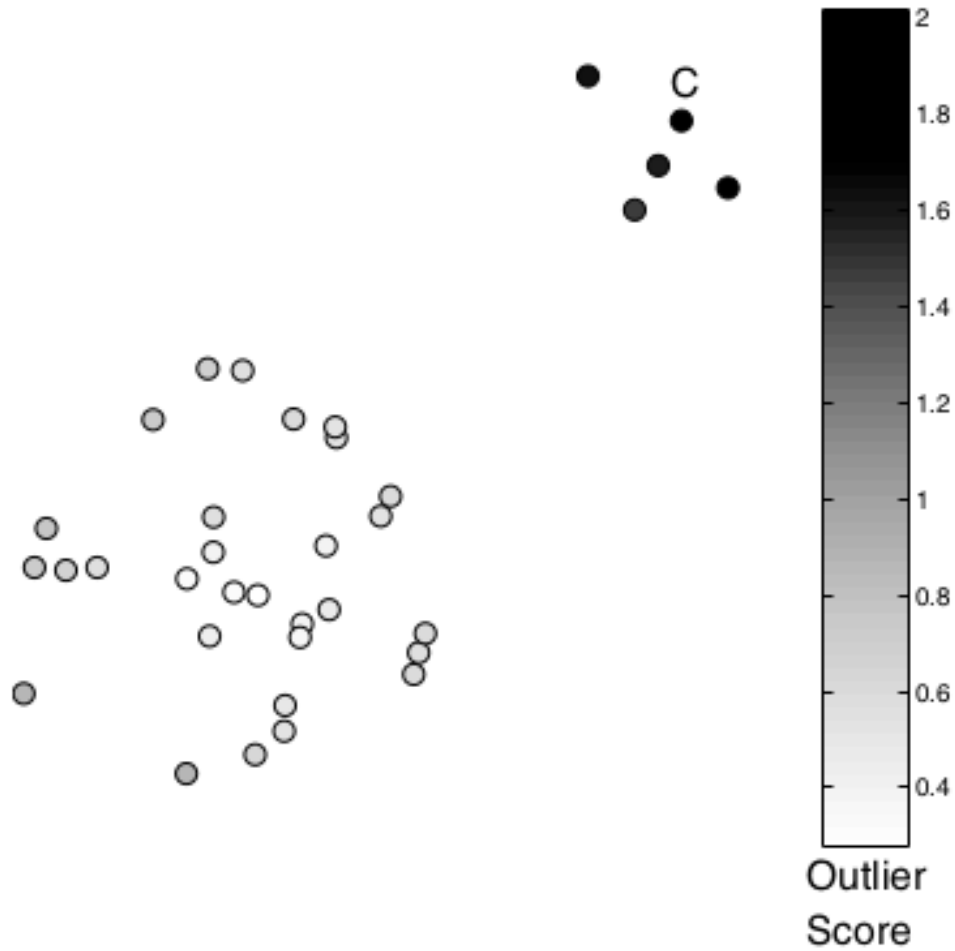
A threshold on the minimum distance of an outlier can be set on a training set.

Outlier Score



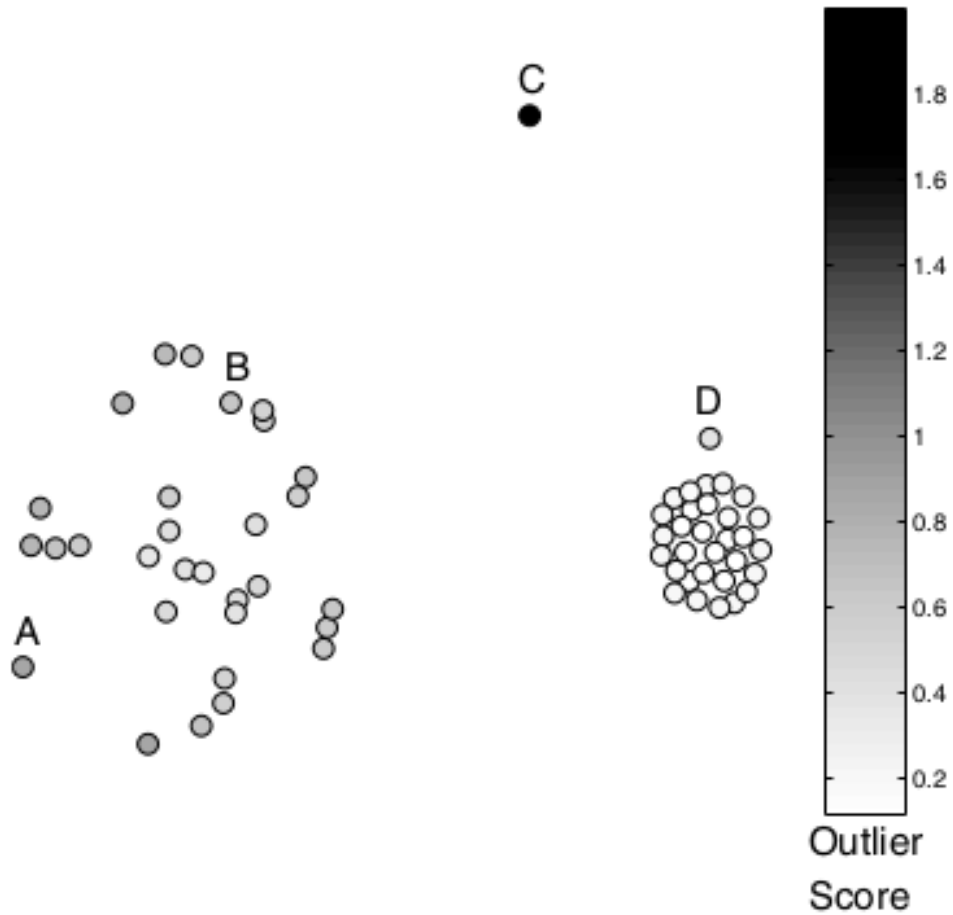
The outlier score is based on the distance to the fifth nearest neighbor.

Outlier Score



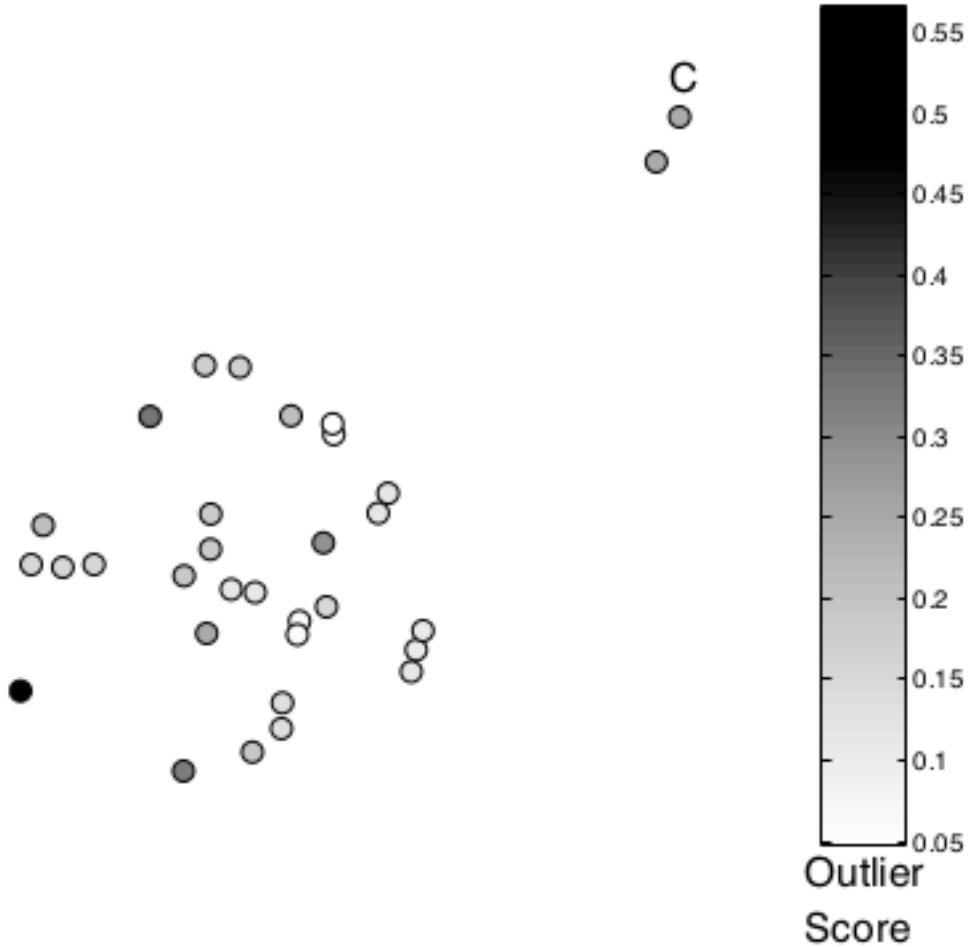
The outlier score is based on the distance to the fifth nearest neighbor.

Outlier Score



The outlier score is based on the distance to the fifth nearest neighbor.

Outlier Score



The outlier score is based on the distance to the first nearest neighbor.

Proximity-Based Approaches

- Conceptually simple and easy to implement.
- Time complexity typically $\mathcal{O}(p^2)$ where p is the number of feature vectors in the dataset.
- Sensitivity to the choice of parameters (the number of neighbors).
- Density/compactness not taken explicitly into account.

Density-Based Methods

Density-Based Approaches

Definition 3

The outlier score of an object is the inverse of the density around the object.

Density-Based Approaches

Definition 3

The outlier score of an object is the inverse of the density around the object.

We consider the *Local Outlier Factor (LOF)* method:

A *local density* of an object corresponds to the inverse of the average distance to its k-nearest neighbors.

Density-Based Approaches

Definition 3

The outlier score of an object is the inverse of the density around the object.

We consider the *Local Outlier Factor (LOF)* method:

A *local density* of an object corresponds to the inverse of the average distance to its k-nearest neighbors.

Compute the LOF score for an object A by

- computing the local density D_A of A,

Density-Based Approaches

Definition 3

The outlier score of an object is the inverse of the density around the object.

We consider the *Local Outlier Factor (LOF)* method:

A *local density* of an object corresponds to the inverse of the average distance to its k-nearest neighbors.

Compute the LOF score for an object A by

- computing the local density D_A of A,
- computing the average $\bar{D}$ of the local densities of k-nearest neighbors of A,

Density-Based Approaches

Definition 3

The outlier score of an object is the inverse of the density around the object.

We consider the *Local Outlier Factor (LOF)* method:

A *local density* of an object corresponds to the inverse of the average distance to its k-nearest neighbors.

Compute the LOF score for an object A by

- computing the local density D_A of A,
- computing the average $\bar{D}$ of the local densities of k-nearest neighbors of A,
- dividing the average local density with the local density of A, that is, compute $\bar{D}/D_A$.

Density-Based Approaches

Definition 3

The outlier score of an object is the inverse of the density around the object.

We consider the *Local Outlier Factor (LOF)* method:

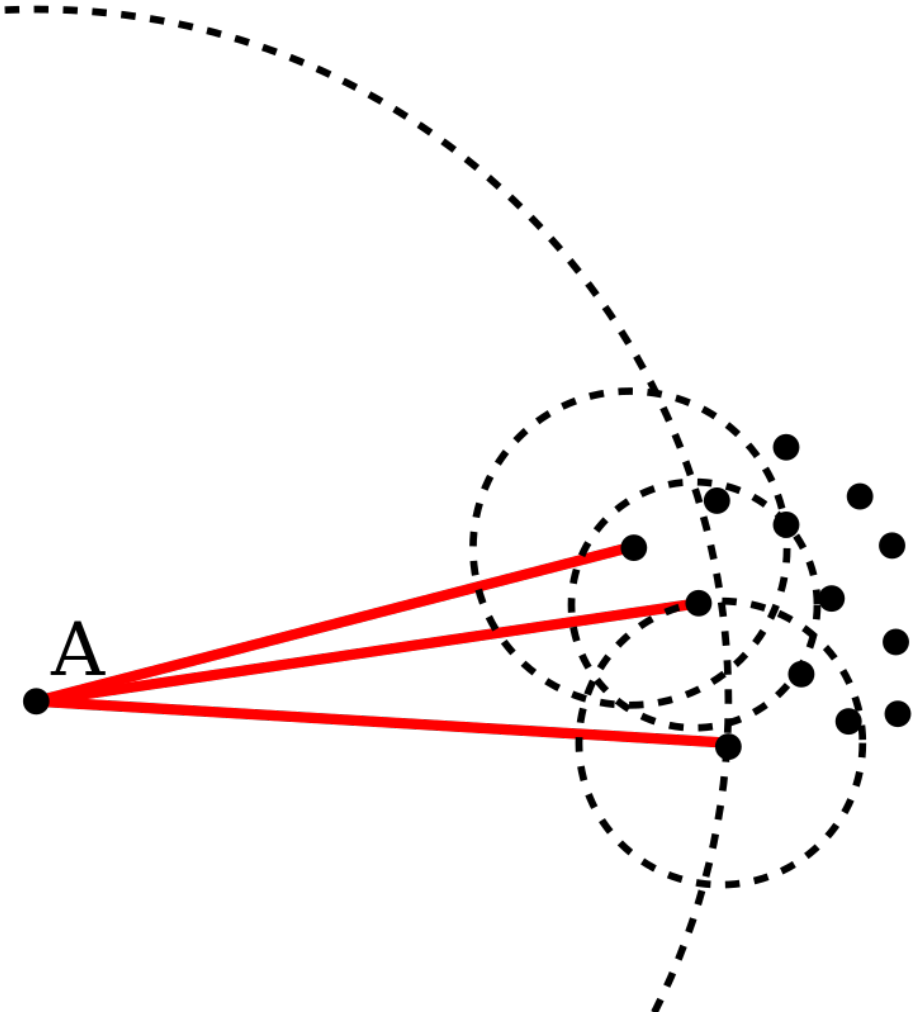
A *local density* of an object corresponds to the inverse of the average distance to its k-nearest neighbors.

Compute the LOF score for an object A by

- computing the local density D_A of A,
- computing the average $\bar{D}$ of the local densities of k-nearest neighbors of A,
- dividing the average local density with the local density of A, that is, compute $\bar{D}/D_A$.

The question is, how exactly can the local density be defined?

LOF - Illustration



Here, the density around A is smaller than the densities around the other points.

LOF - Formally

Let us fix $k \in \mathbb{N}$.

Define k-distance(A) as the distance of the k-th nearest neighbor.

LOF - Formally

Let us fix $k \in \mathbb{N}$.

Define k-distance(A) as the distance of the k-th nearest neighbor.

Denote by $N_k(A)$ the set of all objects in the distance from A up to k-distance(A).

Note that if there are objects in the same distance from A, we may have more than k elements in $N_k(A)$.

LOF - Formally

Let us fix $k \in \mathbb{N}$.

Define k-distance(A) as the distance of the k-th nearest neighbor.

Denote by $N_k(A)$ the set of all objects in the distance from A up to k-distance(A).

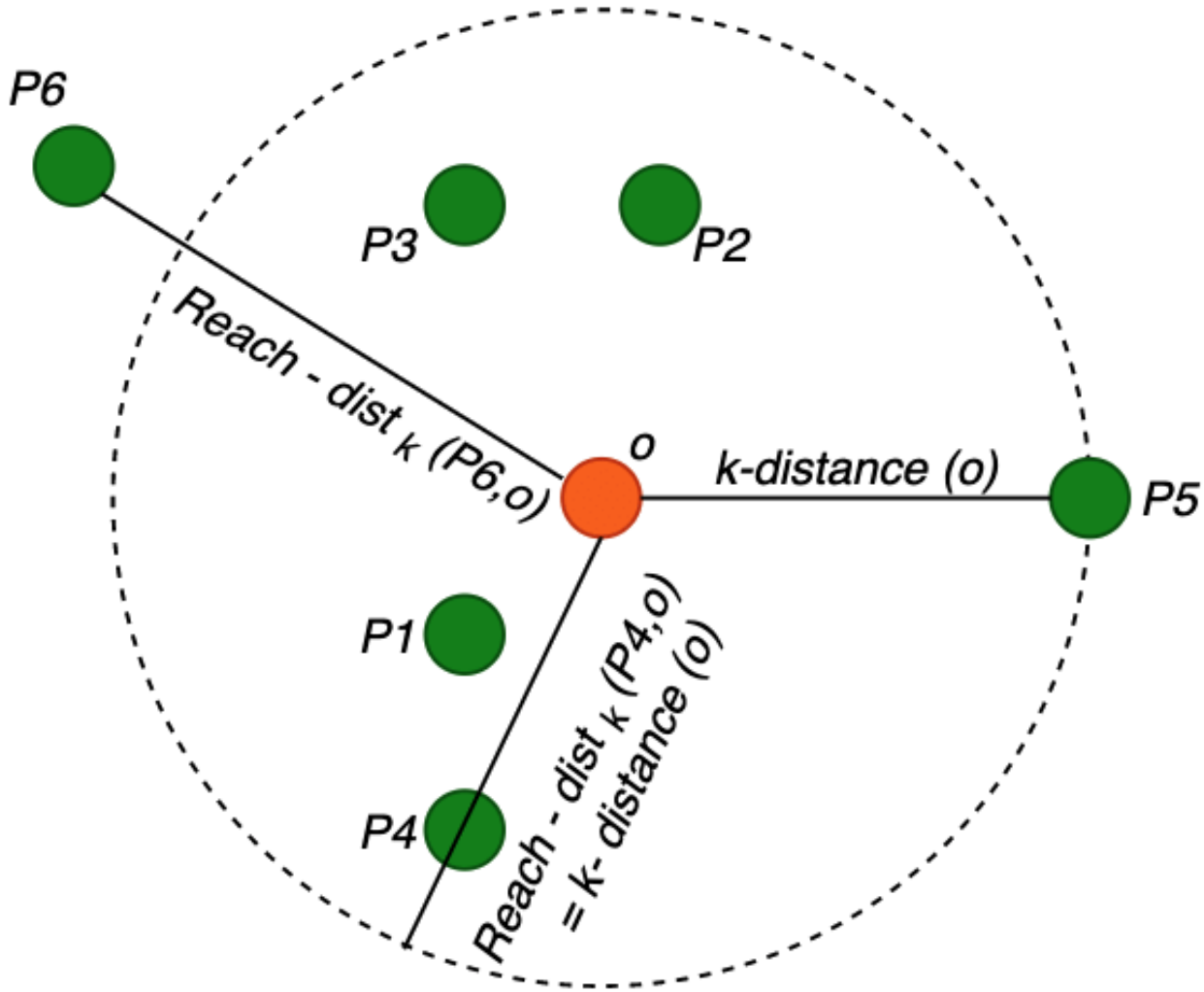
Note that if there are objects in the same distance from A, we may have more than k elements in $N_k(A)$.

Define

$$\text{reachability-distance}_k(A, B) = \max\{k\text{-distance}(B), d(A, B)\}$$

The reachability distance of A from B is the distance between A and B but at least the distance from B to the k-nearest neighbor.

Reachability Distance



LOF

Define *local reachability density*

$$\mathrm{lrd}_k(A) = \left(\frac{\sum_{B \in N_k(A)} \text{reachability-distance}_k(A, B)}{|N_k(A)|} \right)^{-1}$$

That is the reciprocal of the average reachability distance of A from its k-nearest neighbors.

LOF

Define *local reachability density*

$$\mathrm{lrd}_k(A) = \left(\frac{\sum_{B \in N_k(A)} \text{reachability-distance}_k(A, B)}{|N_k(A)|} \right)^{-1}$$

That is the reciprocal of the average reachability distance of A from its k-nearest neighbors.

We define *local outlier factor* of an object A by

$$\mathrm{LOF}_k(A) = \left(\frac{\sum_{B \in N_k(A)} \mathrm{lrd}_k(B)}{|N_k(A)|} \right) / \mathrm{lrd}_k(A)$$

LOF

Define *local reachability density*

$$\mathrm{lrd}_k(A) = \left(\frac{\sum_{B \in N_k(A)} \text{reachability-distance}_k(A, B)}{|N_k(A)|} \right)^{-1}$$

That is the reciprocal of the average reachability distance of A from its k-nearest neighbors.

We define *local outlier factor* of an object A by

$$\mathrm{LOF}_k(A) = \left(\frac{\sum_{B \in N_k(A)} \mathrm{lrd}_k(B)}{|N_k(A)|} \right) / \mathrm{lrd}_k(A)$$

Now

- $\mathrm{LOF}_k(A) \approx 1$ - similar density as the neighbors have
- $\mathrm{LOF}_k(A) < 1$ - higher density of neighbors than the neighbors have (inlier?)
- $\mathrm{LOF}_k(A) > 1$ - smaller density of neighbors than the neighbors have (outlier?)

Example

Consider a dataset

$$D = \{A, B, C, D\} = \{5, 2, 1, 0\}$$

Consider $k = 2$ and the distance $d(U, V) = |U - V|$.

Example

Consider a dataset

$$D = \{A, B, C, D\} = \{5, 2, 1, 0\}$$

Consider $k = 2$ and the distance $d(U, V) = |U - V|$.

$$N_k(A) = \{B, C\}$$

	k-distance(.)	$d(A, .)$	reach-dist$_k(A, .)$
B	2	3	3
C	1	4	4

$$\text{lrd}_k(A) = \left(\frac{1}{2} \left(\text{reach-dist}_k(A, B) + \text{reach-dist}_k(A, C) \right) \right)^{-1} = 2/7$$

Example

Consider a dataset

$$D = \{A, B, C, D\} = \{5, 2, 1, 0\}$$

Consider $k = 2$ and the distance $d(U, V) = |U - V|$.

$$N_k(B) = \{C, D\}$$

	k-distance(.)	$d(B, .)$	reach-dist$_k(B, .)$
C	1	1	1
D	2	2	2

$$\text{lrd}_k(B) = \left(\frac{1}{2} \left(\text{reach-dist}_k(B, C) + \text{reach-dist}_k(B, D) \right) \right)^{-1} = 2/3$$

Example

Consider a dataset

$$D = \{A, B, C, D\} = \{5, 2, 1, 0\}$$

Consider $k = 2$ and the distance $d(U, V) = |U - V|$.

$$N_k(C) = \{B, D\}$$

	k-distance(.)	$d(C, .)$	reach-dist$_k(C, .)$
B	2	1	2
D	2	1	2

$$\text{lrd}_k(C) = \left(\frac{1}{2}\left(\text{reach-dist}_k(C, B) + \text{reach-dist}_k(C, D)\right)\right)^{-1} = 1/2$$

Example

Consider a dataset

$$D = \{A, B, C, D\} = \{5, 2, 1, 0\}$$

Consider $k = 2$ and the distance $d(U, V) = |U - V|$.

$$N_k(D) = \{B, C\}$$

	k-distance(.)	$d(D, .)$	reach-dist$_k(D, .)$
B	2	2	2
C	1	1	1

$$\text{lrd}_k(D) = \left(\frac{1}{2}\left(\text{reach-dist}_k(D, B) + \text{reach-dist}_k(D, C)\right)\right)^{-1} = 2/3$$

Example

$$\mathrm{lrd}_k(A) = 2/7$$
$$\mathrm{lrd}_k(B) = 2/3$$
$$\mathrm{lrd}_k(C) = 1/2$$
$$\mathrm{lrd}_k(D) = 2/3$$

Example

$$\text{lrd}_k(A) = 2/7$$
$$\text{lrd}_k(B) = 2/3$$
$$\text{lrd}_k(C) = 1/2$$
$$\text{lrd}_k(D) = 2/3$$

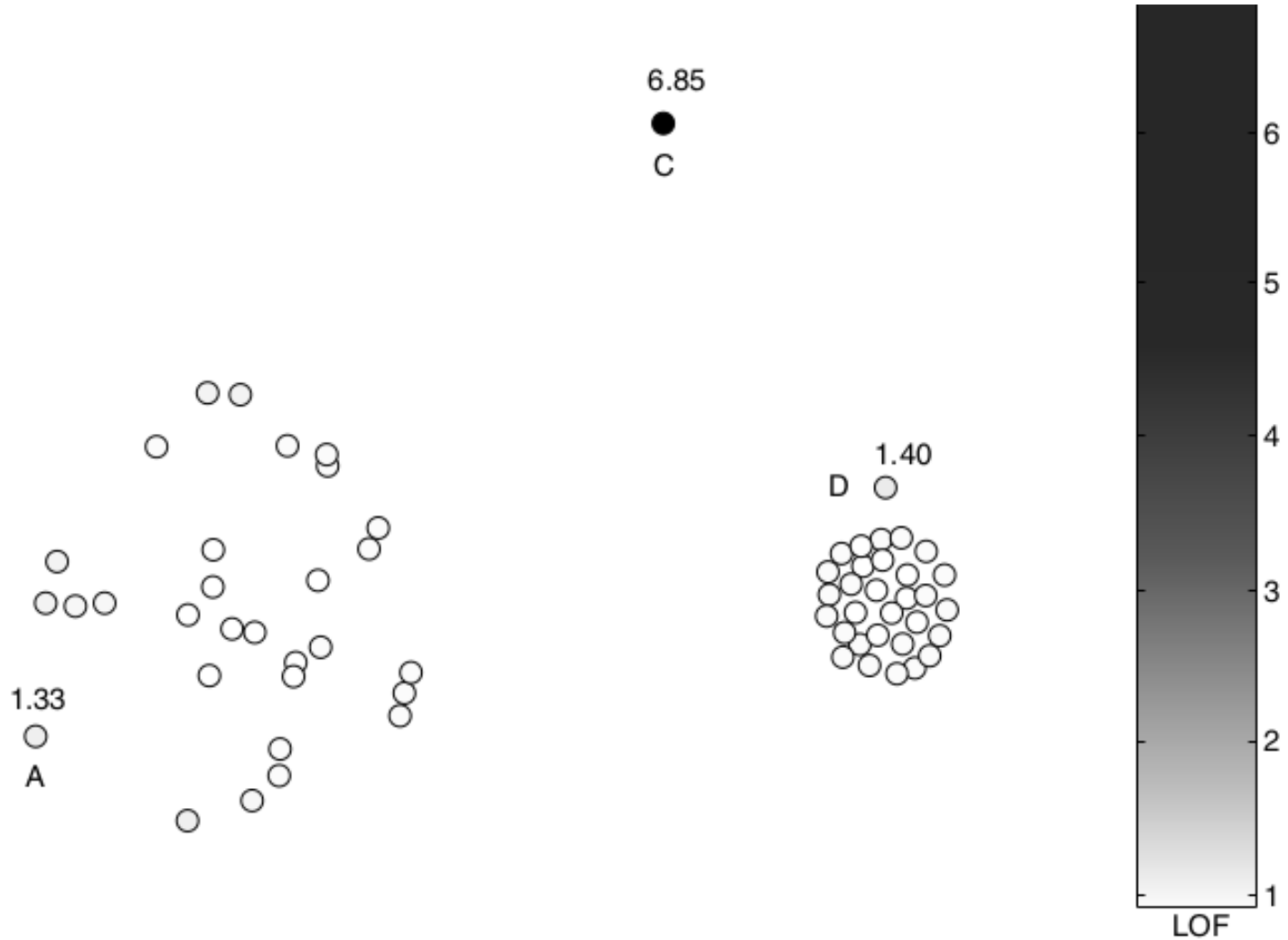
$$\begin{aligned}\text{LOF}_k(A) &= \frac{1}{2}(\text{lrd}_k(B) + \text{lrd}_k(C))/\text{lrd}_k(A) \\ &= (1/2)(2/3 + 1/2)/(2/7) = 2.041\end{aligned}$$

$$\begin{aligned}\text{LOF}_k(B) &= \frac{1}{2}(\text{lrd}_k(C) + \text{lrd}_k(D))/\text{lrd}_k(B) \\ &= (1/2)(1/2 + 2/3)/(2/3) = 0.875\end{aligned}$$

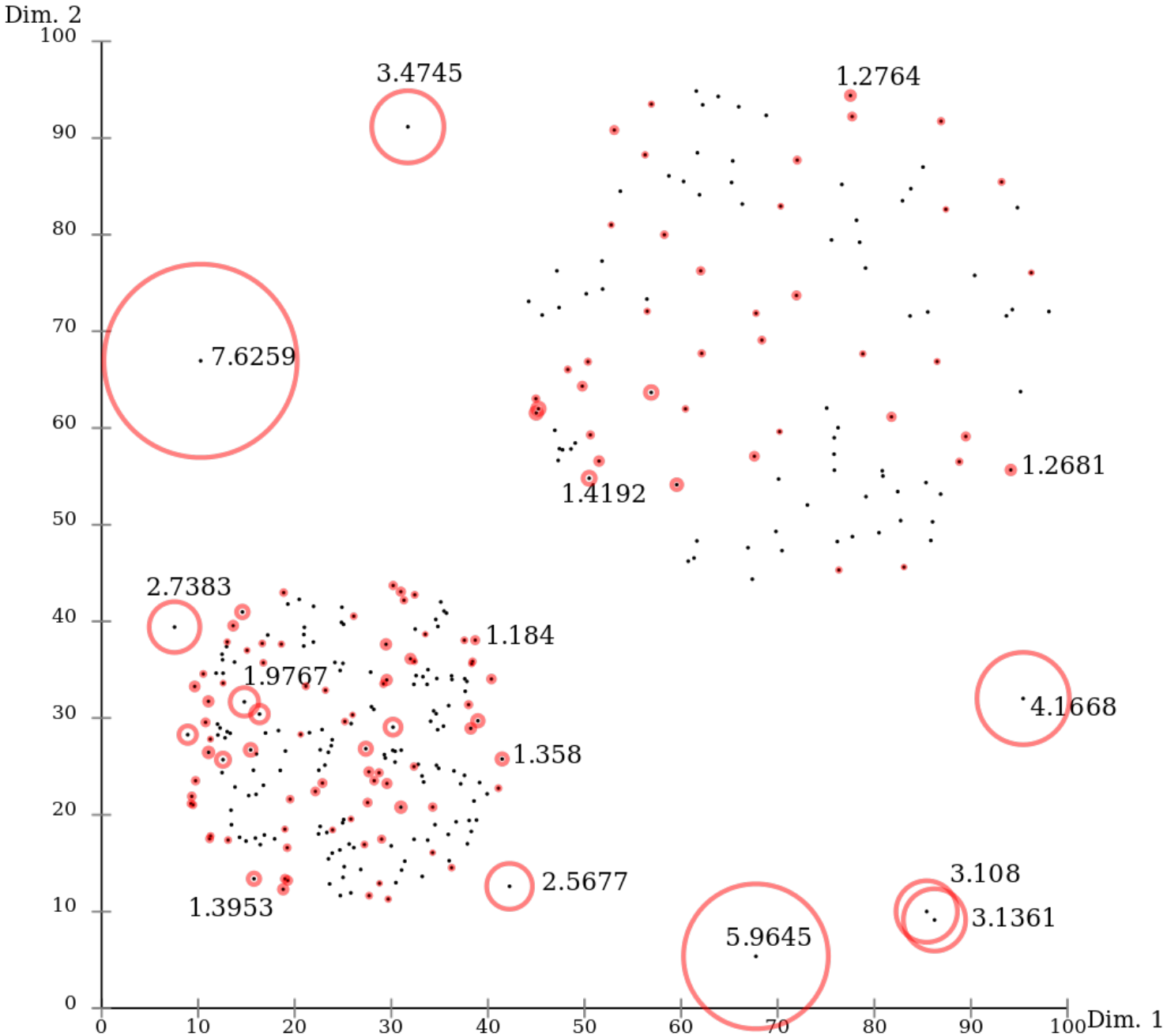
$$\begin{aligned}\text{LOF}_k(C) &= \frac{1}{2}(\text{lrd}_k(B) + \text{lrd}_k(D))/\text{lrd}_k(B) \\ &= (1/2)(2/3 + 2/3)/(2/3) = 1\end{aligned}$$

$$\text{LOF}_k(D) = 0.875$$

Another Example



Yet Another Example



Comments on Density Based Methods

- As opposed to the distance-based methods, quantifies the distance of the neighborhood of the (potential) outliers.
- Time complexity still $\mathcal{O}(p^2)$ but may be reduced for low dimensional data (to $\mathcal{O}(p \log p)$) using special data structures.
- Still need to determine the parameter k.

Clustering-Based Methods

Clustering Approach

Clustering is supposed to group similar objects.

Clustering Approach

Clustering is supposed to group similar objects.

Anomaly detection detects objects dissimilar to others.

Clustering Approach

Clustering is supposed to group similar objects.

Anomaly detection detects objects dissimilar to others.

So, clustering inherently solves similar problems.

Clustering Approach

Clustering is supposed to group similar objects.

Anomaly detection detects objects dissimilar to others.

So, clustering inherently solves similar problems.

But how do we detect anomalies based on clustering?

Clustering Approach

Clustering is supposed to group similar objects.

Anomaly detection detects objects dissimilar to others.

So, clustering inherently solves similar problems.

But how do we detect anomalies based on clustering?

We may detect anomalies as elements of small clusters.

This approach is problematic as it strongly depends on the size/number of clusters.

Clustering Approach

Clustering is supposed to group similar objects.

Anomaly detection detects objects dissimilar to others.

So, clustering inherently solves similar problems.

But how do we detect anomalies based on clustering?

We may detect anomalies as elements of small clusters.

This approach is problematic as it strongly depends on the size/number of clusters.

A better approach would be to asses how strongly objects belong to clusters.

Definition 4

An object is a cluster-based outlier if the object does not strongly belong to any cluster.

Clustering Approach

Clustering is supposed to group similar objects.

Anomaly detection detects objects dissimilar to others.

So, clustering inherently solves similar problems.

But how do we detect anomalies based on clustering?

We may detect anomalies as elements of small clusters.

This approach is problematic as it strongly depends on the size/number of clusters.

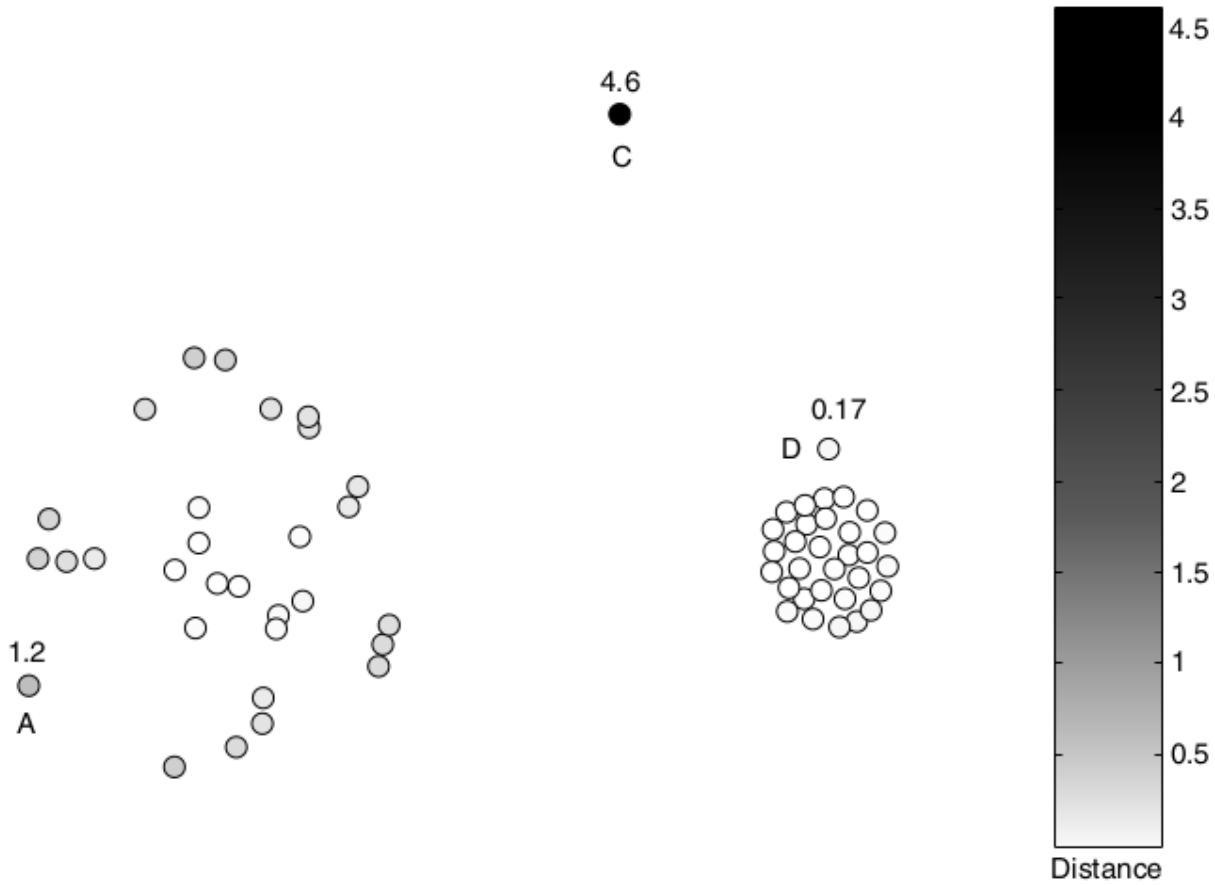
A better approach would be to asses how strongly objects belong to clusters.

Definition 4

An object is a cluster-based outlier if the object does not strongly belong to any cluster.

Depending on the particular clustering algorithm, we have (at least some) information about the cohesion of objects.

Anomaly Detection Using k-Means Clustering

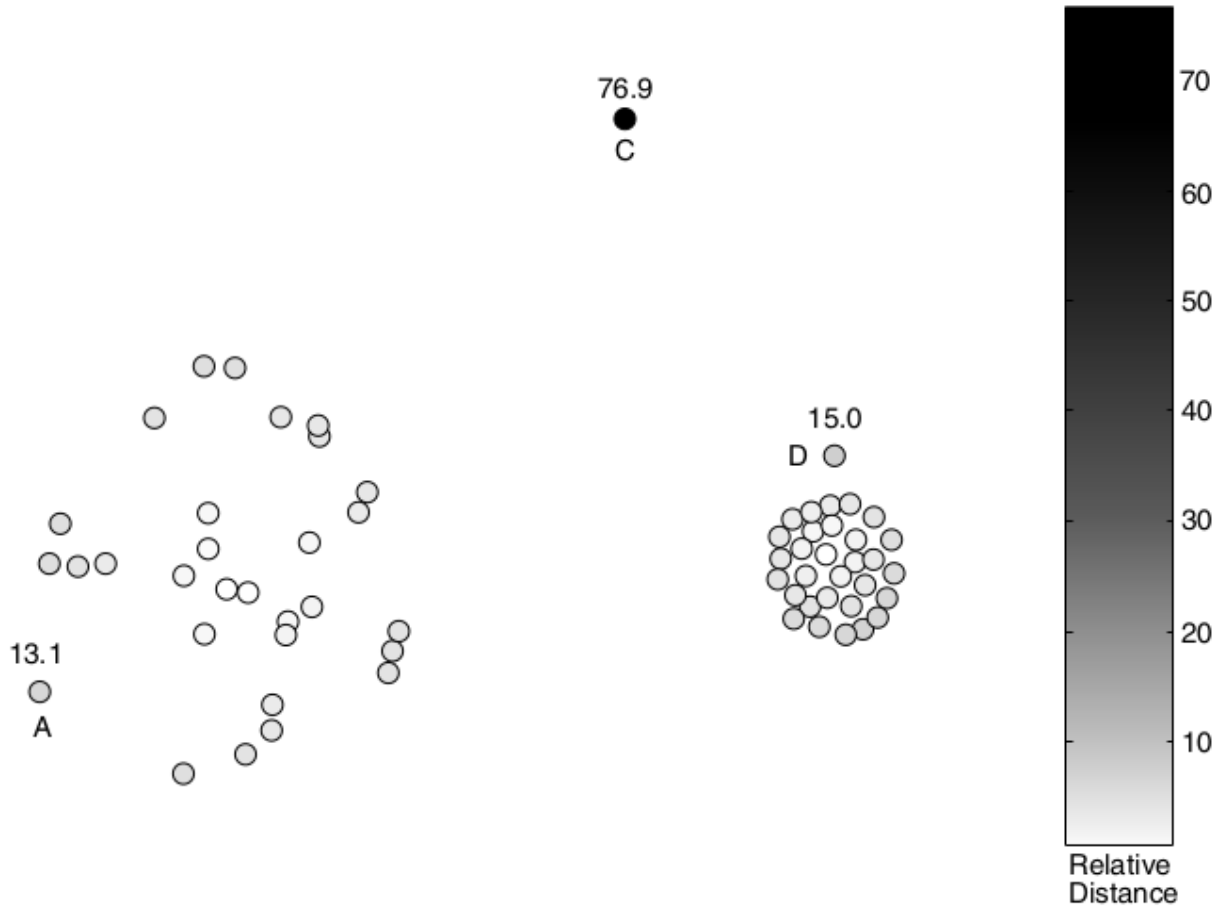


In k-means, the clusters are represented by centroids.

We may measure the anomaly of a given object using the distance to its cluster centroid.

This approach does not take into account the density of clusters.

Anomaly Detection Using k-Means Clustering



Here, we compute the relative distance to the cluster center, that is, the ratio of the object's distance to the centroid and the median distance of all objects of the same cluster to the centroid.

Clustering Based Anomaly Detection

Other algorithms may provide different information (probability density of the clusters, etc.).

Clustering Based Anomaly Detection

Other algorithms may provide different information (probability density of the clusters, etc.).

The usual problem: Outliers have an impact on the clustering itself.

Clustering Based Anomaly Detection

Other algorithms may provide different information (probability density of the clusters, etc.).

The usual problem: Outliers have an impact on the clustering itself.

Some algorithms can be modified to treat potential outliers specially.

For example, during the k-means clustering, put objects very far away from the current cluster centroids into a special category not used to move the centroids. After every step, test whether some of these objects became close enough to at least one centroid (in which case these objects will be removed from the special category).

Clustering Based Anomaly Detection

Other algorithms may provide different information (probability density of the clusters, etc.).

The usual problem: Outliers have an impact on the clustering itself.

Some algorithms can be modified to treat potential outliers specially.

For example, during the k-means clustering, put objects very far away from the current cluster centroids into a special category not used to move the centroids. After every step, test whether some of these objects became close enough to at least one centroid (in which case these objects will be removed from the special category).

Setting the proper number of clusters is an issue as well.

Clustering Based Anomaly Detection

Other algorithms may provide different information (probability density of the clusters, etc.).

The usual problem: Outliers have an impact on the clustering itself.

Some algorithms can be modified to treat potential outliers specially.

For example, during the k-means clustering, put objects very far away from the current cluster centroids into a special category not used to move the centroids. After every step, test whether some of these objects became close enough to at least one centroid (in which case these objects will be removed from the special category).

Setting the proper number of clusters is an issue as well.

For anomaly detection, a larger number of smaller clusters may be beneficial as they may be more cohesive. If an object seems to be an outlier with small clusters, it is probably an outlier.

Comments on Clustering Based Methods

Some clustering methods have a sub-quadratic complexity.

Conceptually, clustering complements anomaly detection, so it is natural to compute both together.

On the other hand, the results strongly depend on the number of clusters, and anomalies may distort the clustering.

Autoencoders as Anomaly Detectors

... just a short comment

Neural Networks in Anomaly Detection

The previous approaches work well with (relatively) low-dimensional data.

Neural Networks in Anomaly Detection

The previous approaches work well with (relatively) low-dimensional data.

However, what should we do with data with high or even variable dimensions?

- Images
- Text
- Video
- Semantic graphs
- ...

One way is to transform them into lower-dimensional data.

Neural Networks in Anomaly Detection

The previous approaches work well with (relatively) low-dimensional data.

However, what should we do with data with high or even variable dimensions?

- Images
- Text
- Video
- Semantic graphs
- ...

One way is to transform them into lower-dimensional data.

Train a neural network that takes the large input and returns a smaller representation, which (hopefully) preserves crucial features of the input.

Autoencoders

An autoencoder consists of two parts:

- $\phi : \mathbb{R}^n \to \mathbb{R}^m$ the encoder
- $\psi : \mathbb{R}^m \to \mathbb{R}^n$ the decoder

The goal is to find ϕ, ψ so that $\psi \circ \phi$ is (almost) identity.

Autoencoders

An autoencoder consists of two parts:

- $\phi : \mathbb{R}^n \to \mathbb{R}^m$ the encoder
- $\psi : \mathbb{R}^m \to \mathbb{R}^n$ the decoder

The goal is to find ϕ, ψ so that $\psi \circ \phi$ is (almost) identity.

The value $\vec{h} = \phi(\vec{x})$ is called the *latent representation* of $\vec{x}$.

Autoencoders

An autoencoder consists of two parts:

- $\phi : \mathbb{R}^n \to \mathbb{R}^m$ the encoder
- $\psi : \mathbb{R}^m \to \mathbb{R}^n$ the decoder

The goal is to find ϕ, ψ so that $\psi \circ \phi$ is (almost) identity.

The value $\vec{h} = \phi(\vec{x})$ is called the *latent representation* of $\vec{x}$.

Training: Assume

$$\mathcal{T} = \{\vec{x}_1, \ldots, \vec{x}_p\}$$

where $\vec{x}_i \in \mathbb{R}^n$ for all $i \in \{1, \ldots, n\}$.

Autoencoders

An autoencoder consists of two parts:

- $\phi : \mathbb{R}^n \to \mathbb{R}^m$ the encoder
- $\psi : \mathbb{R}^m \to \mathbb{R}^n$ the decoder

The goal is to find ϕ, ψ so that $\psi \circ \phi$ is (almost) identity.

The value $\vec{h} = \phi(\vec{x})$ is called the *latent representation* of $\vec{x}$.

Training: Assume

$$\mathcal{T} = \{\vec{x}_1, \ldots, \vec{x}_p\}$$

where $\vec{x}_i \in \mathbb{R}^n$ for all $i \in \{1, \ldots, n\}$.

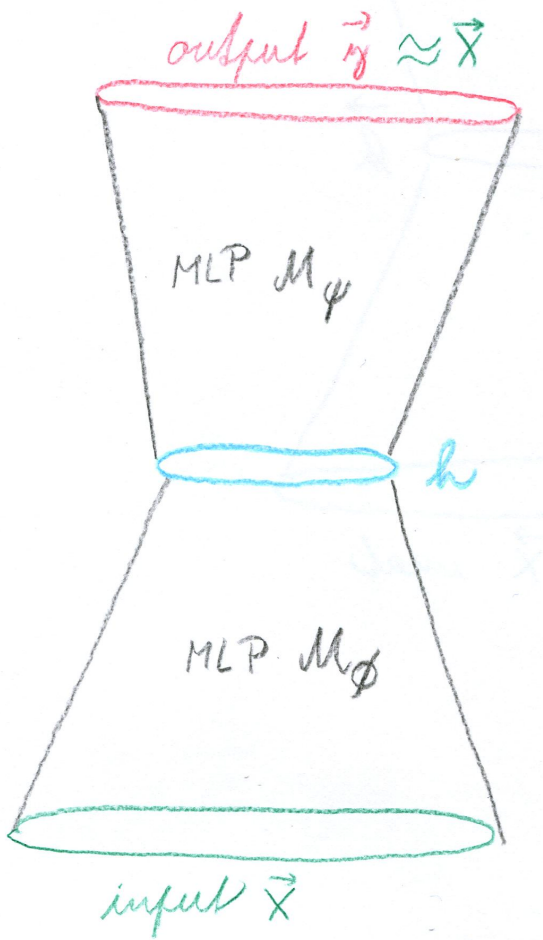
Minimize the **reconstruction** error

$$E = \sum_{i=1}^{p} (\vec{x}_i - \psi(\phi(\vec{x}_i)))^2$$

Autoencoders – neural networks

Both ϕ and ψ can be represented using MLP $\mathcal{M}_\phi$ and $\mathcal{M}_\psi$, respectively.

$\mathcal{M}_\phi$ and $\mathcal{M}_\psi$ can be connected into a single network.



Autoencoders – Usage

- Compression/feature extraction – from $\vec{x}$ to $\vec{h}$.

Autoencoders – Usage

- Compression/feature extraction – from $\vec{x}$ to $\vec{h}$.
- Dimensionality reduction – the latent representation $\vec{h}$ has a smaller dimension.

Autoencoders – Usage

- Compression/feature extraction – from $\vec{x}$ to $\vec{h}$.
- Dimensionality reduction – the latent representation $\vec{h}$ has a smaller dimension.
- Generative versions – (roughly) generate $\vec{h}$ from a known distribution, let $\mathcal{M}_\psi$ generate realistic inputs/outputs $\vec{x}$

Autoencoders – Usage

- Compression/feature extraction – from $\vec{x}$ to $\vec{h}$.
- Dimensionality reduction – the latent representation $\vec{h}$ has a smaller dimension.
- Generative versions – (roughly) generate $\vec{h}$ from a known distribution, let $\mathcal{M}_\psi$ generate realistic inputs/outputs $\vec{x}$
- **Anomaly detection** (see the next slide)

Anomaly Detection with Autoencoders

Straightforward approach:

- Train the autoencoder on the normal data.
- Detect an anomaly using the large reconstruction error.

  The idea is that an anomaly will not be properly reconstructed.

Anomaly Detection with Autoencoders

Straightforward approach:

- Train the autoencoder on the normal data.
- Detect an anomaly using the large reconstruction error.

  The idea is that an anomaly will not be properly reconstructed.

More general approach:

- Train the autoencoder and transform the normal data to their low dimensional latent representations.
- Use one of the previous approaches to anomaly detection to detect anomalies in the latent representations.

  The assumption is that the latent representation of an anomaly will substantially differ from the latent representations of normal instances.

Summary of Anomaly Detection

- It is hard even to define an anomaly - context knowledge is usually needed.

Summary of Anomaly Detection

- It is hard even to define an anomaly - context knowledge is usually needed.
- Supervised anomaly detection reduces to classification.

Summary of Anomaly Detection

- It is hard even to define an anomaly - context knowledge is usually needed.
- Supervised anomaly detection reduces to classification.
- Unsupervised anomaly detection can be done in various ways; we have seen the following:
  - Statistical
  - Proximity-based
  - Density-based
  - Cluster-based
  - Autoencoders

Summary of Anomaly Detection

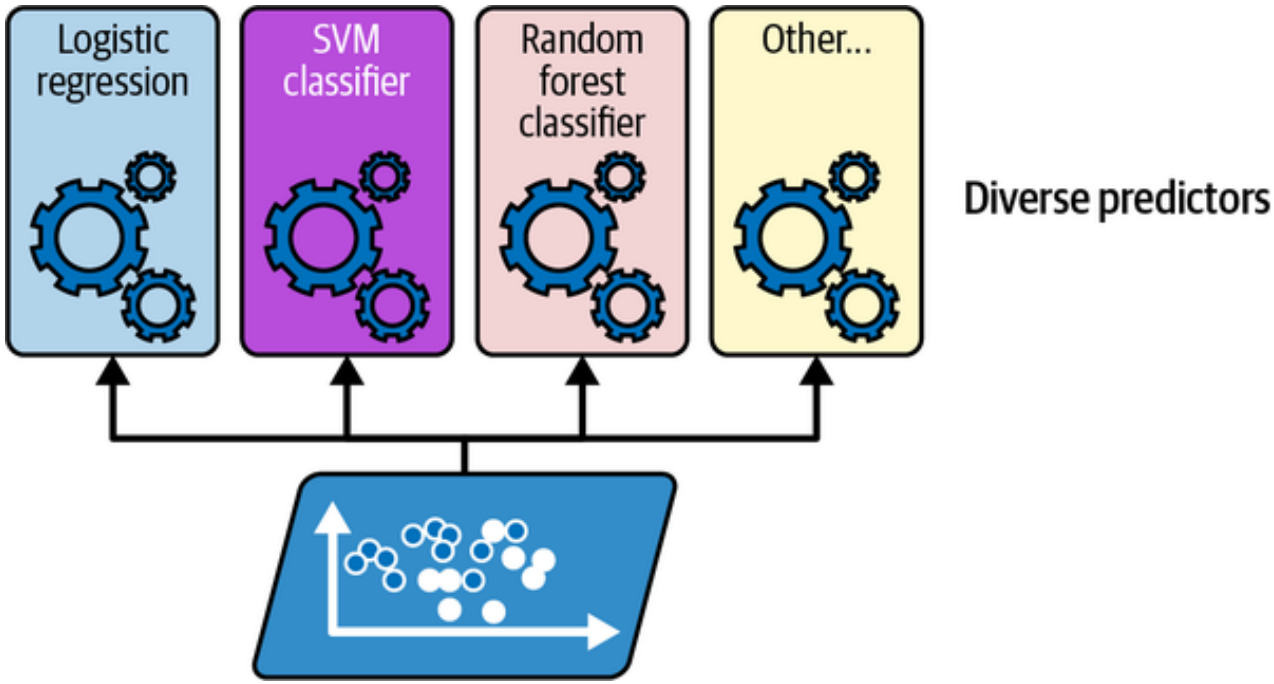
- It is hard even to define an anomaly - context knowledge is usually needed.
- Supervised anomaly detection reduces to classification.
- Unsupervised anomaly detection can be done in various ways; we have seen the following:
  - Statistical
  - Proximity-based
  - Density-based
  - Cluster-based
  - Autoencoders
- Semi-supervised anomaly detection is usually concerned with data where the normal class is known.

Summary of Anomaly Detection

- It is hard even to define an anomaly - context knowledge is usually needed.
- Supervised anomaly detection reduces to classification.
- Unsupervised anomaly detection can be done in various ways; we have seen the following:
  - Statistical
  - Proximity-based
  - Density-based
  - Cluster-based
  - Autoencoders
- Semi-supervised anomaly detection is usually concerned with data where the normal class is known.
- There are many more methods: One class SVM, isolation forests, etc.

Ensemble Methods

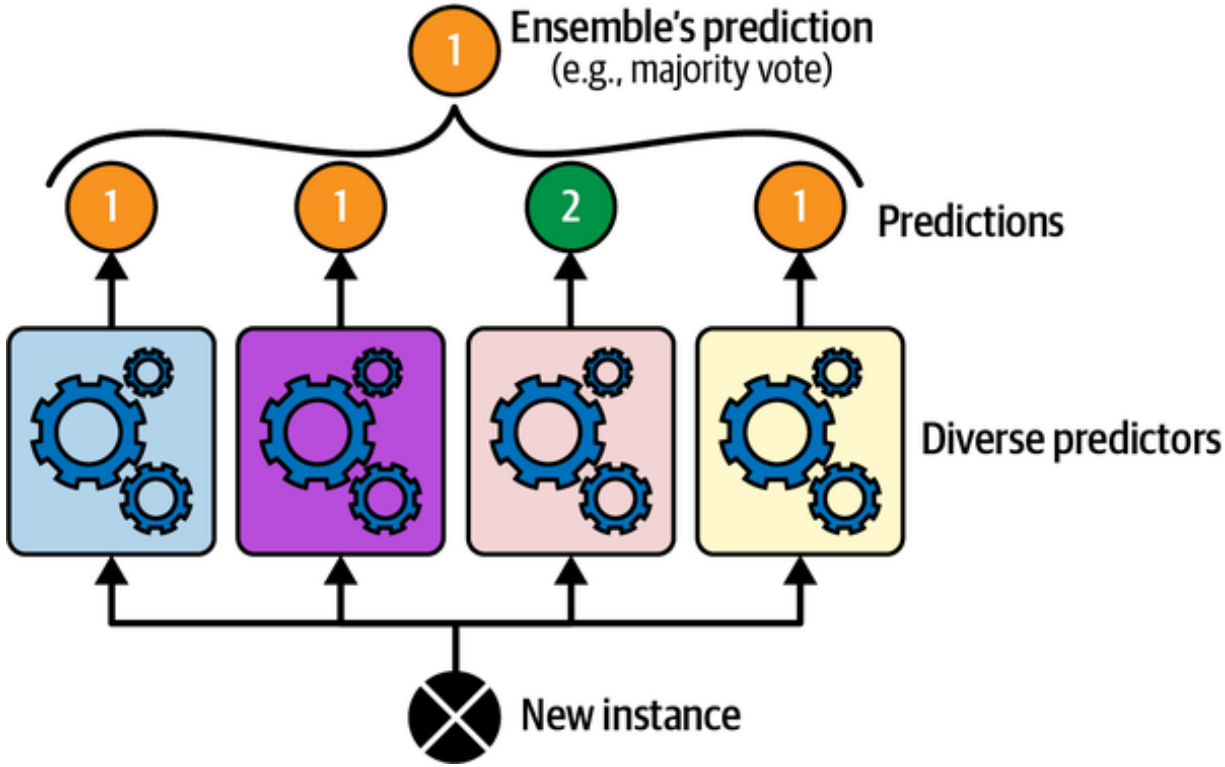
Voting Classifiers



Train several models. They may differ in

- Structure (completely different models)
- Training data (subsample the training set)

Voting Classifiers



During the inference, ensemble the predictions. For binary classifiers, you may do the following:

- Take the majority vote.
- Summarize the output probabilities (e.g., by averaging)
- If logistic regression or neural networks with logistic output activation are used, summarize the outputs before the application of the last logistic sigmoid.

The Ensemble Idea

An ensemble of *weak* classifiers may be a *strong classifier*.

The Ensemble Idea

An ensemble of *weak* classifiers may be a *strong classifier*.

Analogy: Assume a coin toss that has a 51 percent chance of heads and 49 percent tails.

The Ensemble Idea

An ensemble of *weak* classifiers may be a *strong classifier*.

Analogy: Assume a coin toss that has a 51 percent chance of heads and 49 percent tails.

Try 1,000 tosses in a row. Do this repeatedly. In 75 percent of cases, you get more heads than tails. With 10,000 tosses, it would be 97 percent.

The Ensemble Idea

An ensemble of *weak* classifiers may be a *strong classifier*.

Analogy: Assume a coin toss that has a 51 percent chance of heads and 49 percent tails.

Try 1,000 tosses in a row. Do this repeatedly. In 75 percent of cases, you get more heads than tails. With 10,000 tosses, it would be 97 percent.

Intuitively, we might suspect that a voting ensemble of 1,000 sufficiently independent classifiers, each with an accuracy of around 51 percent, would have an accuracy of around 70 percent.

The Ensemble Idea

An ensemble of *weak* classifiers may be a *strong classifier*.

Analogy: Assume a coin toss that has a 51 percent chance of heads and 49 percent tails.

Try 1,000 tosses in a row. Do this repeatedly. In 75 percent of cases, you get more heads than tails. With 10,000 tosses, it would be 97 percent.

Intuitively, we might suspect that a voting ensemble of 1,000 sufficiently independent classifiers, each with an accuracy of around 51 percent, would have an accuracy of around 70 percent.

The ensemble methods work best when the models are as independent as possible.

Use different learning methods or independently chosen training sets (see later slides).

Let us try to formalize the above intuition.

Bias/Variance Decomposition

Consider a random function $g(x) + \varepsilon$ where $g : \mathbb{R} \to \mathbb{R}$ and ε is random noise with mean 0 and variance σ^2.

Bias/Variance Decomposition

Consider a random function $g(x) + \varepsilon$ where $g : \mathbb{R} \to \mathbb{R}$ and ε is random noise with mean 0 and variance σ^2.

Consider a dataset for regression:

$$D = \{(x_1, d_1), \ldots, (x_p, d_p)\}$$

Here $x_1, \ldots, x_p$ are fixed inputs, and each $d_k = g(x_k) + \varepsilon_k$ where $\varepsilon_1, \ldots, \varepsilon_p$ are independent samples from the noise ε.

We get different models for different samples of the training set D.

Bias/Variance Decomposition

Consider a random function $g(x) + \varepsilon$ where $g : \mathbb{R} \to \mathbb{R}$ and ε is random noise with mean 0 and variance σ^2.

Consider a dataset for regression:

$$D = \{(x_1, d_1), \ldots, (x_p, d_p)\}$$

Here $x_1, \ldots, x_p$ are fixed inputs, and each $d_k = g(x_k) + \varepsilon_k$ where $\varepsilon_1, \ldots, \varepsilon_p$ are independent samples from the noise ε.

We get different models for different samples of the training set D.

We may ask:

- How much do the models trained on different samples differ?
- How much do they differ (on average) from g?

Bias/Variance Decomposition

Consider a random function $g(x) + \varepsilon$ where $g : \mathbb{R} \to \mathbb{R}$ and ε is random noise with mean 0 and variance σ^2.

Consider a dataset for regression:

$$D = \{(x_1, d_1), \ldots, (x_p, d_p)\}$$

Here $x_1, \ldots, x_p$ are fixed inputs, and each $d_k = g(x_k) + \varepsilon_k$ where $\varepsilon_1, \ldots, \varepsilon_p$ are independent samples from the noise ε.

We get different models for different samples of the training set D.

We may ask:

- How much do the models trained on different samples differ?
- How much do they differ (on average) from g?

Let us denote by $h_D : \mathbb{R} \to \mathbb{R}$ a model trained on the dataset D by minimizing the squared error $\sum_{k=1}^{p} (h_D(x_k) - d_k)^2$.

Bias/Variance Decomposition

Let us consider the expected value of the squared error:

$$\mathbb{E}\left(\sum_{k=1}^{p}(h_D(x_k)-d_k)^2\right)=\sum_{k=1}^{p}\mathbb{E}\,(h_D(x_k)-d_k)^2$$

The expectation is computed over the distribution of the datasets D.

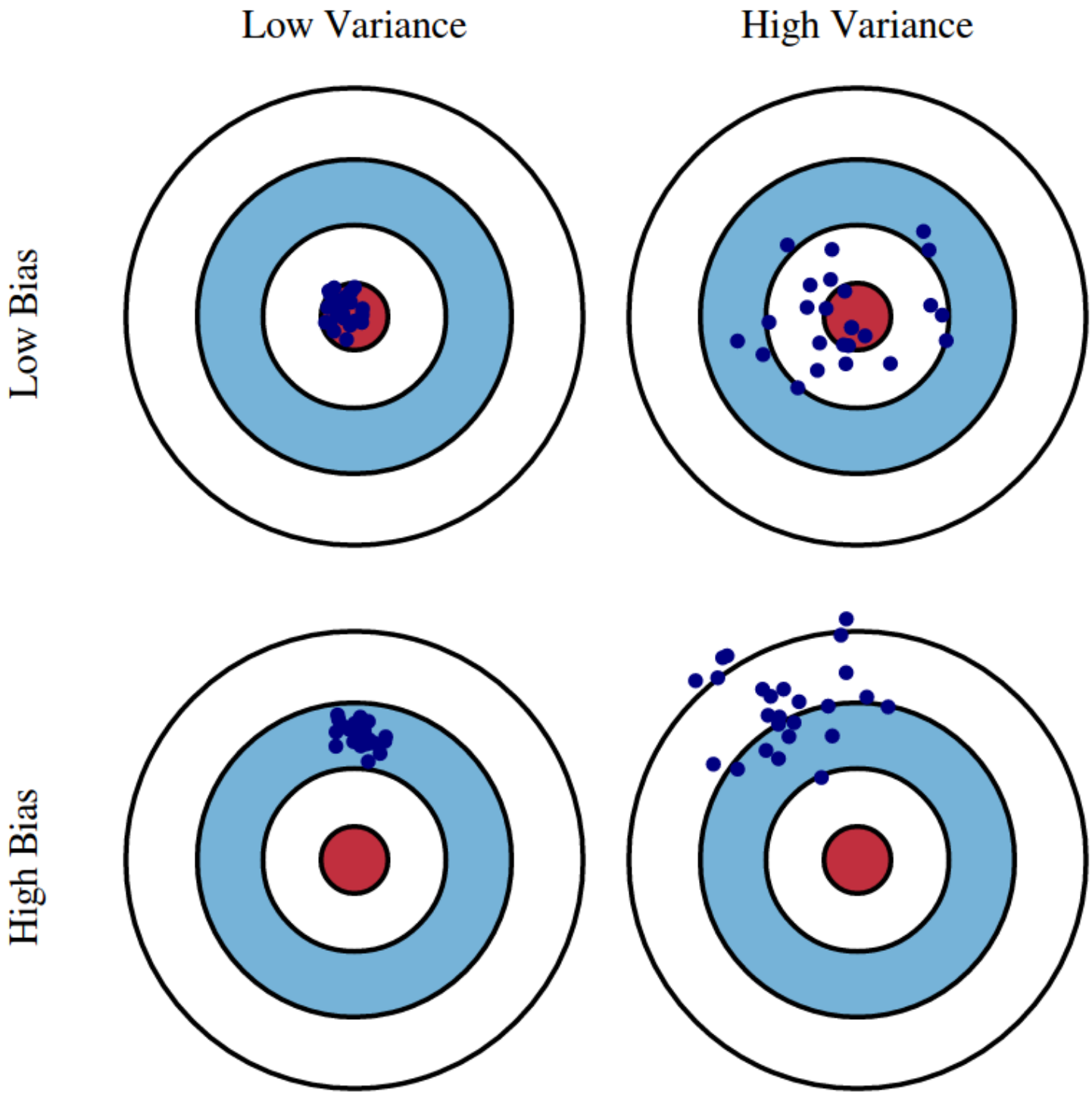
One can prove that

$$\sum_{k=1}^{p}\mathbb{E}\,(h_D(x_k)-d_k)^2=$$

$$\sum_{k=1}^{p}\mathbb{E}\,(h_D(x_k)-\mathbb{E}[h_D(x_k)])^2+(\mathbb{E}[h_D(x_k)]-g(x_k))^2+\sigma^2$$

Here

- $\mathbb{E}\,(h_D(x_k)-\mathbb{E}[h_D(x_k)])^2$ is the *Variance* of the model on x_k
  how much the model's value jumps around its average on x_k.
- $\mathbb{E}[h_D(x_k)]-g(x_k)$ is the *Bias* of the model.
  how much the average model's value differs from the true values.
- σ^2 is the noise variance.


Low Variance
High Variance
Low Bias
High Bias

B/V Decomposition Proof (Optional)

Let us study $\mathbb{E}\left(h_D(x_k) - d_k\right)^2$. To simplify notation we drop the index k and write $\mathbb{E}_D\left(h_D(x) - d\right)^2$.

$$\begin{aligned}
\left(h_D(x) - d\right)^2 &= \left(h_D(x) - \mathbb{E}[h_D(x)] + \mathbb{E}[h_D(x)] - d\right)^2 \\
&= \left(h_D(x) - \mathbb{E}[h_D(x)]\right)^2 \\
&\quad + \left(\mathbb{E}[h_D(x)] - d\right)^2 \\
&\quad + 2\left(h_D(x) - \mathbb{E}[h_D(x)]\right)\left(\mathbb{E}[h_D(x)] - d\right)
\end{aligned}$$

Now, just apply $\mathbb{E}$ to both sides. As

$$\mathbb{E}(d) = \mathbb{E}(g(x) + \varepsilon) = g(x)$$

$$\mathbb{E}(d^2) = \mathbb{E}(g(x) + \varepsilon)^2 = \mathbb{E}(g(x)^2 + 2\varepsilon g(x) + \varepsilon^2) = g(x)^2 + \sigma^2$$

We obtain ...

B/V Decomposition Proof (Optional)

... this

$$
\begin{aligned}
&\mathbb{E}\left(\mathbb{E}[h_D(x)] - d\right)^2 \\
&\quad = \mathbb{E}\left(\mathbb{E}[h_D(x)]^2 - 2d\mathbb{E}[h_D(x)] + d^2\right) \\
&\quad = \mathbb{E}[h_D(x)]^2 - 2\mathbb{E}(d)\mathbb{E}[h_D(x)] + \mathbb{E}d^2 \\
&\quad = \mathbb{E}[h_D(x)]^2 - 2g(x)\mathbb{E}[h_D(x)] + g(x)^2 + \sigma^2 \\
&\quad = \mathbb{E}\left(\mathbb{E}[h_D(x)] - g(x)\right)^2 + \sigma^2
\end{aligned}
$$

and this

$$
\begin{aligned}
&\mathbb{E}\left[2\left(h_D(x) - \mathbb{E}[h_D(x)]\right)\left(\mathbb{E}[h_D(x)] - d\right)\right] \\
&\quad = \mathbb{E}\left[2\left(h_D(x) - \mathbb{E}[h_D(x)]\right)\left(\mathbb{E}[h_D(x)] - g(x) - \varepsilon\right)\right] \\
&\quad = \mathbb{E}\left[2\left(h_D(x) - \mathbb{E}[h_D(x)]\right)\left(\mathbb{E}[h_D(x)] - g(x)\right) - 2\left(h_D(x) - \mathbb{E}[h_D(x)]\right)\varepsilon\right] \\
&\quad = \mathbb{E}\left[2\left(h_D(x) - \mathbb{E}[h_D(x)]\right)\left(\mathbb{E}[h_D(x)] - g(x)\right)\right] \\
&\quad = \left(\mathbb{E}[h_D(x)] - g(x)\right)\mathbb{E}\left[2\left(h_D(x) - \mathbb{E}[h_D(x)]\right)\right] \\
&\quad = 0
\end{aligned}
$$

Here, the third equality follows from the zero mean of ε.

Ensemble Methods vs Bias/Variance

Suppose that we could somehow sample m independent datasets: $D_1, \ldots, D_m$.

Ensemble Methods vs Bias/Variance

Suppose that we could somehow sample m independent datasets: $D_1, \ldots, D_m$.

We train m models $h_{D_1}, \ldots, h_{D_m}$ and consider the ensemble model $h_{ens}(x) = \frac{1}{m} \sum_{\ell=1}^{m} h_{D_\ell}(x)$.

Ensemble Methods vs Bias/Variance

Suppose that we could somehow sample m independent datasets: $D_1, \ldots, D_m$.

We train m models $h_{D_1}, \ldots, h_{D_m}$ and consider the ensemble model $h_{ens}(x) = \frac{1}{m} \sum_{\ell=1}^{m} h_{D_\ell}(x)$.

Does the ensemble h_{ens} have a different bias-variance decomp.?

Ensemble Methods vs Bias/Variance

Suppose that we could somehow sample m independent datasets: $D_1, \ldots, D_m$.

We train m models $h_{D_1}, \ldots, h_{D_m}$ and consider the ensemble model $h_{ens}(x) = \frac{1}{m} \sum_{\ell=1}^{m} h_{D_\ell}(x)$.

Does the ensemble h_{ens} have a different bias-variance decomp.?

- The noise variance σ^2 does not change.

Ensemble Methods vs Bias/Variance

Suppose that we could somehow sample m independent datasets: $D_1, \ldots, D_m$.

We train m models $h_{D_1}, \ldots, h_{D_m}$ and consider the ensemble model $h_{ens}(x) = \frac{1}{m} \sum_{\ell=1}^{m} h_{D_\ell}(x)$.

Does the ensemble h_{ens} have a different bias-variance decomp.?

- The noise variance σ^2 does not change.
- The Bias does not change ($\mathbb{E}[h_{D_\ell}(x_k)]$ is independent of ℓ):

$$\mathbb{E}\left[\frac{1}{m}\sum_{\ell=1}^{m} h_{D_\ell}(x_k)\right] - g(x_k) = \mathbb{E}[h_{D_\ell}(x_k)] - g(x_k)$$

Ensemble Methods vs Bias/Variance

Suppose that we could somehow sample m independent datasets: $D_1, \ldots, D_m$.

We train m models $h_{D_1}, \ldots, h_{D_m}$ and consider the ensemble model $h_{ens}(x) = \frac{1}{m} \sum_{\ell=1}^{m} h_{D_\ell}(x)$.

Does the ensemble h_{ens} have a different bias-variance decomp.?

- The noise variance σ^2 does not change.
- The Bias does not change ($\mathbb{E}[h_{D_\ell}(x_k)]$ is independent of ℓ):

$$\mathbb{E}\left[\frac{1}{m}\sum_{\ell=1}^{m} h_{D_\ell}(x_k)\right] - g(x_k) = \mathbb{E}[h_{D_\ell}(x_k)] - g(x_k)$$

- Only the Variance changes, it is *reduced*:

$$\mathbb{E}\left(h_{ens}(x_k) - \mathbb{E}[h_{ens}(x_k)]\right)^2 = \frac{1}{m}\mathbb{E}(h_{D_\ell}(x_k) - \mathbb{E}[h_{D_\ell}(x_k)])^2$$

  The equality follows from basic facts about the variance of sums of independent variables.

Now, having m independently sampled datasets is a real luxury!

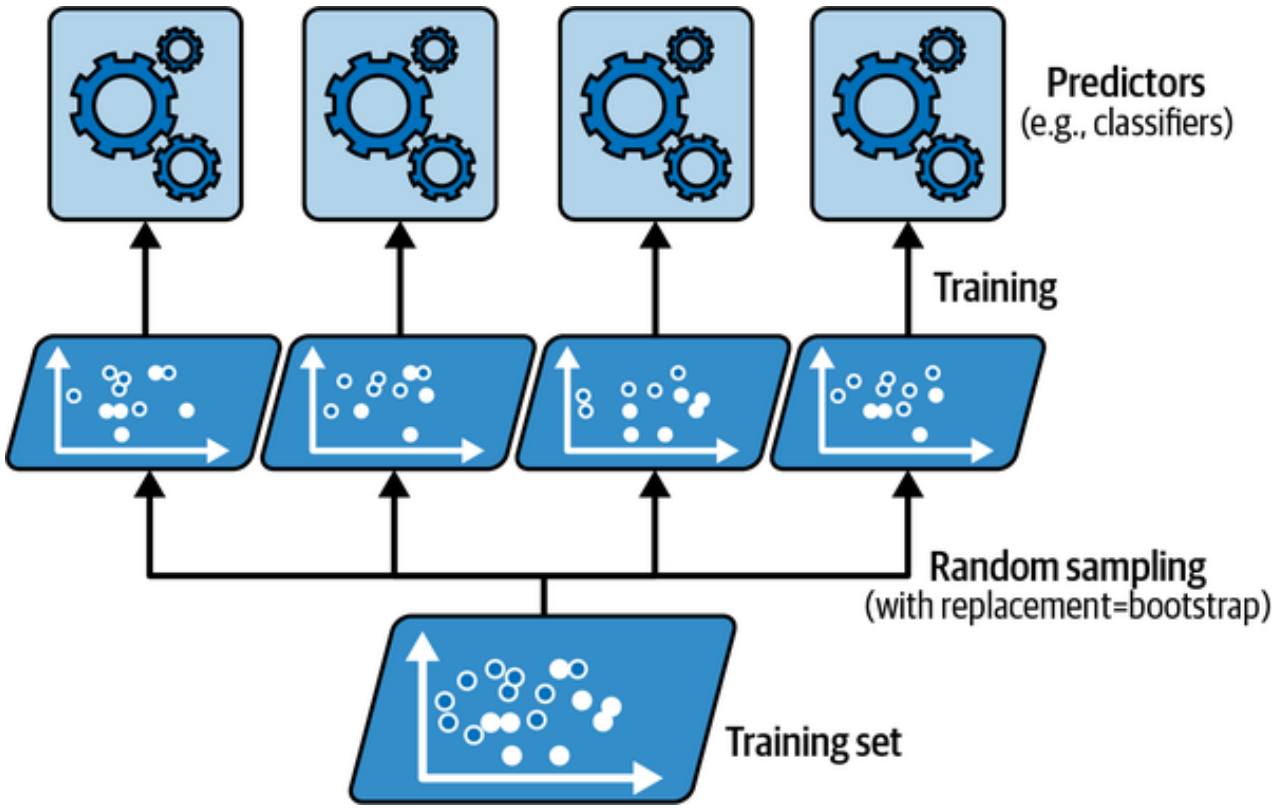
Bagging

In practice, we use sampled subsets of a given dataset.

Bagging

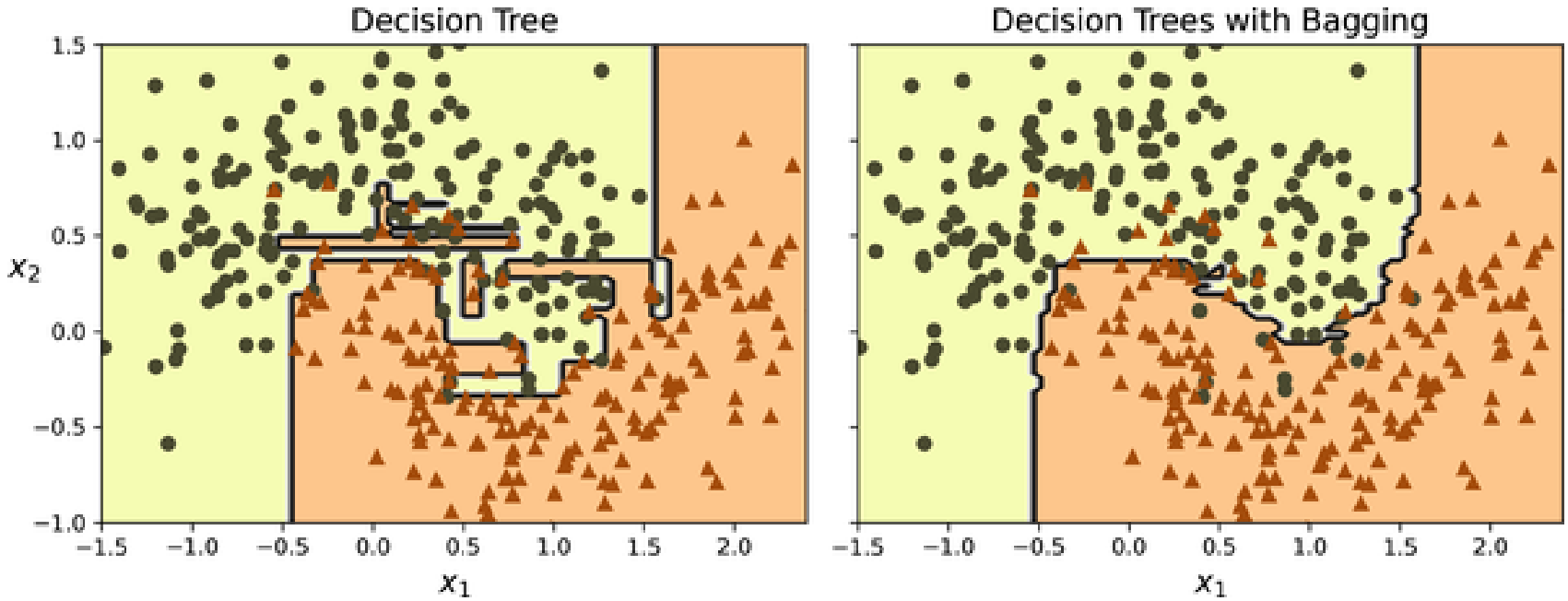
In practice, we use sampled subsets of a given dataset.

One example of using different subsets of the training set is *bagging (Bootstrap Aggregating)*.



- Bootstrap sampling = sampling data subsets with replacement
- Aggregating = majority voting (classification) or averaging (regression)

Bagging Decision Trees



Left: A single decision tree (overfit)

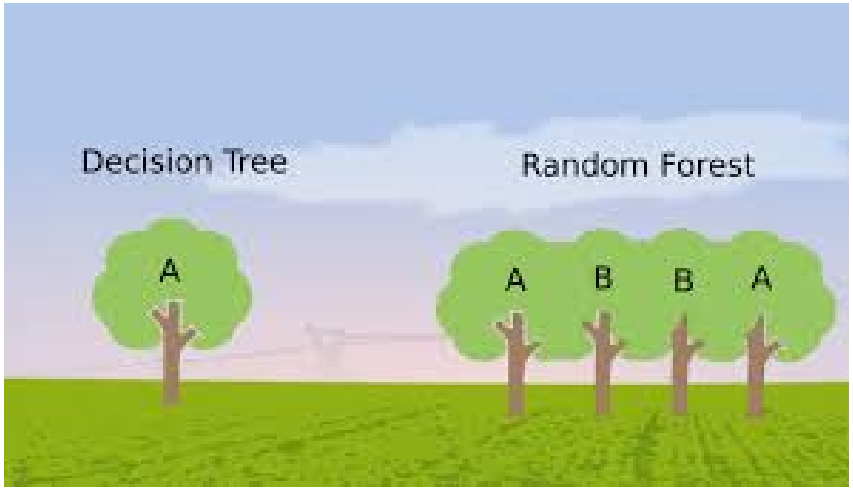
Right: A bagging ensemble of 500 trees

Summary of Bagging

- Reduces overfitting (variance)
- Can work with any type of classifier
- Easy to parallelize

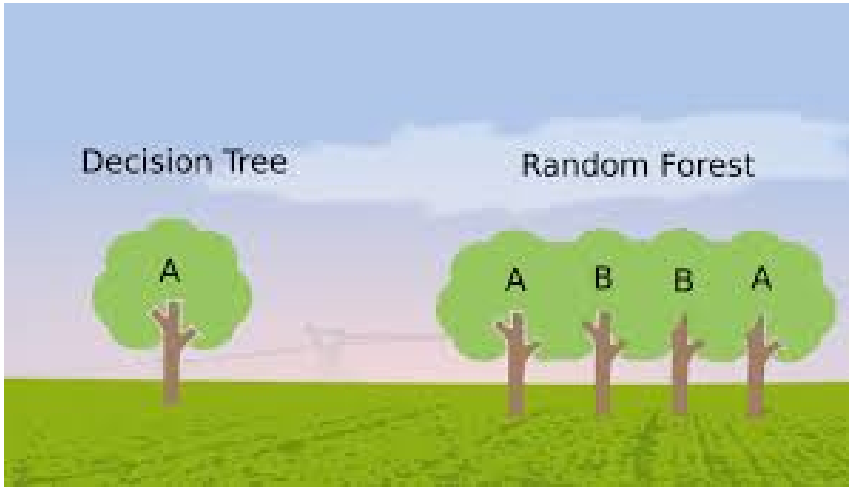
  Just train each model independently, inference can also be parallelized.
- Loses (some) interpretability even for interpretable models (how could you read 500 decision trees?)

Random Forest



Random forest = bagging ensemble of decision trees.

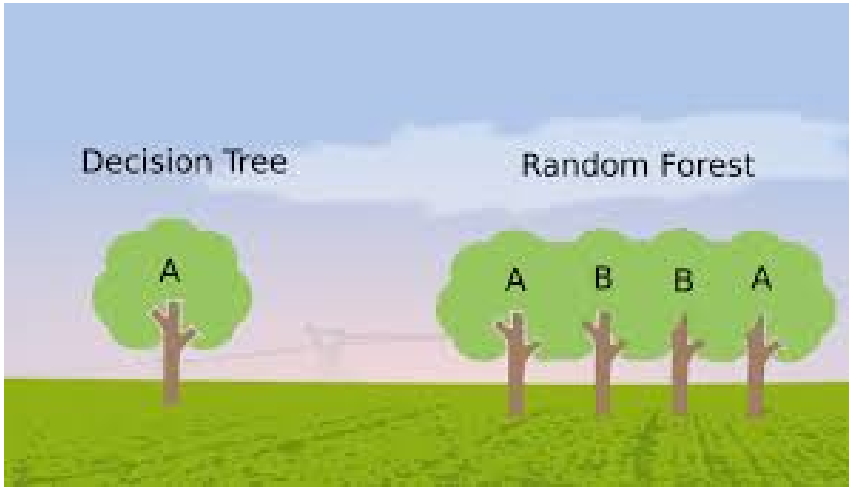
Random Forest



Random forest = bagging ensemble of decision trees.

Has all hyperparameters of decision trees + the number of trees

Random Forest

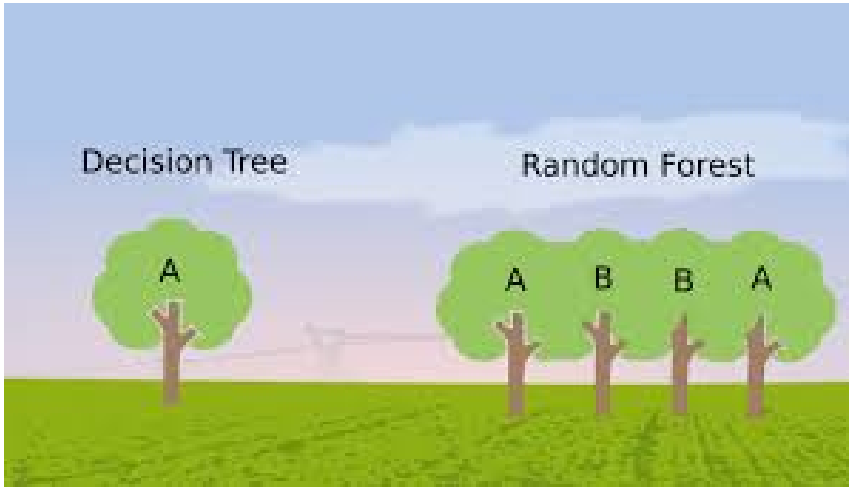


Random forest = bagging ensemble of decision trees.

Has all hyperparameters of decision trees + the number of trees

The random forest algorithm of scikit-learn introduces the following randomness:
When searching for a split attribute, look only into a randomly sampled subset of attributes (by default $\sqrt{n}$ of n attributes).

Random Forest



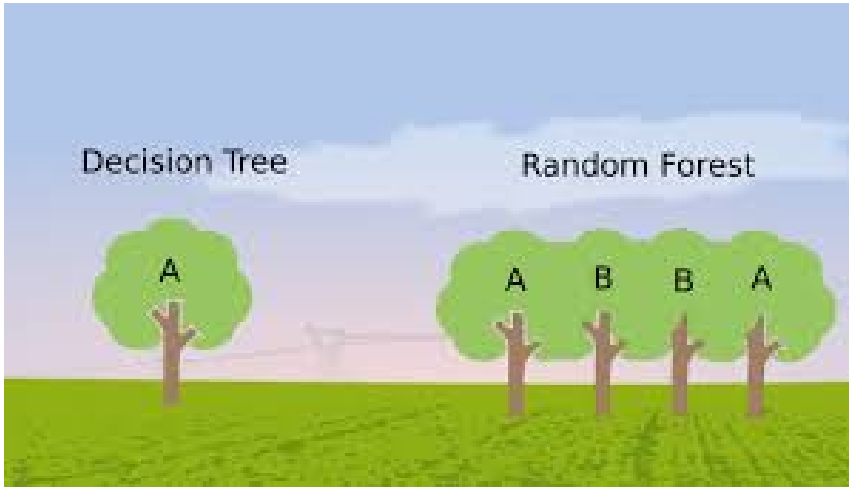
Random forest = bagging ensemble of decision trees.

Has all hyperparameters of decision trees + the number of trees

The random forest algorithm of scikit-learn introduces the following randomness:
When searching for a split attribute, look only into a randomly sampled subset of attributes (by default $\sqrt{n}$ of n attributes).

This gives a greater diversity of the forests.

Random Forest



Random forest = bagging ensemble of decision trees.

Has all hyperparameters of decision trees + the number of trees

The random forest algorithm of scikit-learn introduces the following randomness:
When searching for a split attribute, look only into a randomly sampled subset of attributes (by default $\sqrt{n}$ of n attributes).

This gives a greater diversity of the forests.

The notion of the feature importance can be easily generalized to random forests (computed over all trees).

Boosting

An ensemble method in which weak learners are trained sequentially.

Each new learner tries to correct the error of the current ensemble.

Boosting

An ensemble method in which weak learners are trained sequentially.

Each new learner tries to correct the error of the current ensemble.

There are several variants of boosting. We just have a look at the basic idea behind *AdaBoost* (adaptive boosting).

Boosting

An ensemble method in which weak learners are trained sequentially.

Each new learner tries to correct the error of the current ensemble.

There are several variants of boosting. We just have a look at the basic idea behind *AdaBoost* (adaptive boosting).

To implement this kind of learning algorithm, we need to be able to train models on weighted datasets.

Weighted Training

We are going to use weights on samples.

Note that these are different from the weights used in neural-like models!

Weighted Training

We are going to use weights on samples.

Note that these are different from the weights used in neural-like models!

- Linear regression: Assume a given dataset $D = \{(\vec{x}_1, f_1), \ldots, (\vec{x}_p, f_p)\}$ and weights $a_1, \ldots, a_p$ associated to examples from D.

Weighted Training

We are going to use weights on samples.

Note that these are different from the weights used in neural-like models!

- Linear regression: Assume a given dataset $D = \{(\vec{x}_1, f_1), \ldots, (\vec{x}_p, f_p)\}$ and weights $a_1, \ldots, a_p$ associated to examples from D.

  Let $\vec{w}$ be a vector of model weights. We minimize the *weighted mean squared error*

$$E(\vec{w}) = \frac{1}{p} \sum_{i=1}^{p} a_i (\vec{w} \cdot x_i - f_i)^2$$

Weighted Training

We are going to use weights on samples.

Note that these are different from the weights used in neural-like models!

- Linear regression: Assume a given dataset $D = \{(\vec{x}_1, f_1), \ldots, (\vec{x}_p, f_p)\}$ and weights $a_1, \ldots, a_p$ associated to examples from D.

  Let $\vec{w}$ be a vector of model weights. We minimize the *weighted mean squared error*

$$E(\vec{w}) = \frac{1}{p} \sum_{i=1}^{p} a_i (\vec{w} \cdot x_i - f_i)^2$$

- Decision trees: Given dataset D with p samples, we have weights $a_1, \ldots, a_p \in \mathbb{R}$ (one weight for each example in D).

Weighted Training

We are going to use weights on samples.

Note that these are different from the weights used in neural-like models!

- Linear regression: Assume a given dataset $D = \{(\vec{x}_1, f_1), \ldots, (\vec{x}_p, f_p)\}$ and weights $a_1, \ldots, a_p$ associated to examples from D.

  Let $\vec{w}$ be a vector of model weights. We minimize the *weighted mean squared error*

$$E(\vec{w}) = \frac{1}{p} \sum_{i=1}^{p} a_i (\vec{w} \cdot x_i - f_i)^2$$

- Decision trees: Given dataset D with p samples, we have weights $a_1, \ldots, a_p \in \mathbb{R}$ (one weight for each example in D).

  Given a class $c \in C$, we compute the class probability p_c as

$$p_c = \sum \{a_i \mid i\text{-th sample belongs to } c\} \,/\, \sum_{i=1}^{p} a_i$$

  Everything else (Gini impurity, etc.) is the same.

AdaBoost Idea

- Train classifiers using weighted samples in the training dataset.

AdaBoost Idea

- Train classifiers using weighted samples in the training dataset.
- Start with uniform weights, that is, each sample in the dataset has the same weight.

AdaBoost Idea

- Train classifiers using weighted samples in the training dataset.
- Start with uniform weights, that is, each sample in the dataset has the same weight.
- In k-th iteration:
  - train a new classifier h_k on weighted samples,

AdaBoost Idea

- Train classifiers using weighted samples in the training dataset.
- Start with uniform weights, that is, each sample in the dataset has the same weight.
- In k-th iteration:
  - train a new classifier h_k on weighted samples,
  - obtain a coefficient $\alpha_k > 0$ of the classifier h_k,

AdaBoost Idea

- Train classifiers using weighted samples in the training dataset.
- Start with uniform weights, that is, each sample in the dataset has the same weight.
- In k-th iteration:
  - train a new classifier h_k on weighted samples,
  - obtain a coefficient $\alpha_k > 0$ of the classifier h_k,
  - Consider the current ensemble classifier

$$h^k_{ens}(x) = \text{sign}\left(\sum_{\ell=1}^{k} \alpha_\ell h_\ell(x)\right)$$

Shift the weights so that the "most wrong" training instances for the current ensemble have the largest weights.

Consider a training example (x, c) with $c \in \{-1, 1\}$. Then h^k_{ens} misclassifies x iff $c \cdot \sum_{\ell=1}^{k} \alpha_\ell h_\ell(x) < 0$. The more negative this number is, the more wrong the ensemble classifier is.

AdaBoost Idea

- Train classifiers using weighted samples in the training dataset.
- Start with uniform weights, that is, each sample in the dataset has the same weight.
- In k-th iteration:
  - train a new classifier h_k on weighted samples,
  - obtain a coefficient $\alpha_k > 0$ of the classifier h_k,
  - Consider the current ensemble classifier

$$h^k_{ens}(x) = \text{sign}\left(\sum_{\ell=1}^{k} \alpha_\ell h_\ell(x)\right)$$

Shift the weights so that the "most wrong" training instances for the current ensemble have the largest weights.
Consider a training example (x, c) with $c \in \{-1, 1\}$. Then h^k_{ens} misclassifies x iff $c \cdot \sum_{\ell=1}^{k} \alpha_\ell h_\ell(x) < 0$. The more negative this number is, the more wrong the ensemble classifier is.

After K iterations, the ensemble classifier h^K_{ens} is the output.

AdaBoost Idea

- Train classifiers using weighted samples in the training dataset.
- Start with uniform weights, that is, each sample in the dataset has the same weight.
- In k-th iteration:
  - train a new classifier h_k on weighted samples,
  - obtain a coefficient $\alpha_k > 0$ of the classifier h_k,
  - Consider the current ensemble classifier

$$h^k_{ens}(x) = \text{sign}\left(\sum_{\ell=1}^{k} \alpha_\ell h_\ell(x)\right)$$

Shift the weights so that the "most wrong" training instances for the current ensemble have the largest weights.
Consider a training example (x, c) with $c \in \{-1, 1\}$. Then h^k_{ens} misclassifies x iff $c \cdot \sum_{\ell=1}^{k} \alpha_\ell h_\ell(x) < 0$. The more negative this number is, the more wrong the ensemble classifier is.

After K iterations, the ensemble classifier h^K_{ens} is the output.

For details, see more advanced courses in Machine Learning.

Summary of Ensemble Methods

- Ensemble methods may improve "independently" trained models by grouping them and letting them decide collectively.

Summary of Ensemble Methods

- Ensemble methods may improve "independently" trained models by grouping them and letting them decide collectively.
- Technically, the ensembling decreases the model variance.

Summary of Ensemble Methods

- Ensemble methods may improve "independently" trained models by grouping them and letting them decide collectively.
- Technically, the ensembling decreases the model variance.
- There are several approaches to ensembling:
  - Bagging - training of several models on bootstrap subsamples of the training dataset (random forest is an example)
  - Boosting - ensemble is built sequentially, the newly added model possibly solves the hardest instances
  - There are many more algorithms (gradient boosting, etc.)

THE END