

Interpretable Machine Learning

INSA-Toulouse & LAAS-CNRS

June 7, 2024

Motivation

Black-Box Machine Learning

https://www.youtube.com/watch?v=6Kf3I_0yDlI&ab_channel=SouthChinaMorningPost

The COMPAS Tool

**MIT
Technology
Review**

Featured Topics Newsletters Events Podcasts

Sign in Subscribe

TECH POLICY

AI is sending people to jail—and getting it wrong

Using historical data to train risk assessment tools could mean that machines are copying the mistakes of the past.

By Karen Hao January 21, 2019



IAN WALDIE/GETTY IMAGES

COMPASS data and Rule-based Predictions

Sex	Age	Priors	Juvenile Felonies	Juvenile Crimes	Ethnicity
Male	15	1	0	1	Caucasian
Male	15	1	0	1	African-American
Female	33	1	0	1	African-American
Female	27	0	1	0	Caucasian
Male	41	0	1	0	Caucasian
...

The problem is to predict recidivism. That is, the tendency of a convicted criminal to re-offend.

Increasing Number of Real Life and Social AI Applications

Increasing Number of Real Life and Social AI Applications



Increasing Number of Real Life and Social AI Applications



Increasing Number of Real Life and Social AI Applications



Increasing Number of Real Life and Social AI Applications



AI: Increasing Number of Real Life Applications Of Machine Learning

- The diverse applications of AI raised many ethical issues and questions

AI: Increasing Number of Real Life Applications Of Machine Learning

- The diverse applications of AI raised many ethical issues and questions
 - Job applications: AI that parses CVs for software engineers and recommends to hire mostly men

AI: Increasing Number of Real Life Applications Of Machine Learning

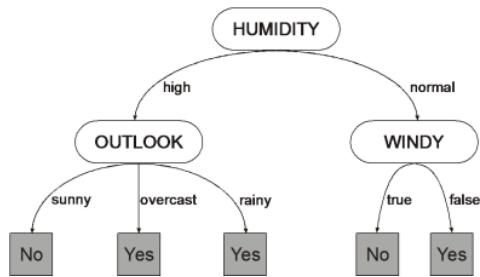
- The diverse applications of AI raised many ethical issues and questions
 - Job applications: AI that parses CVs for software engineers and recommends to hire mostly men
 - Credit scoring: AI that gives a credit score (for bank loans and credit applications) that recommends people from a particular geographical region, specific gender, social class, etc

AI: Increasing Number of Real Life Applications Of Machine Learning

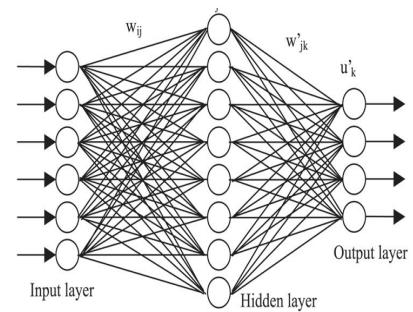
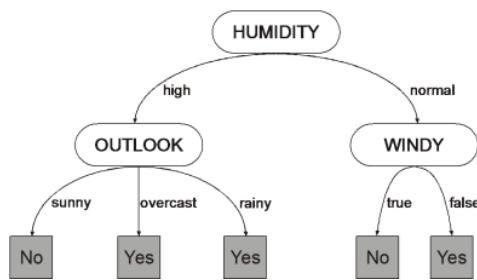
- The diverse applications of AI raised many ethical issues and questions
 - Job applications: AI that parses CVs for software engineers and recommends to hire mostly men
 - Credit scoring: AI that gives a credit score (for bank loans and credit applications) that recommends people from a particular geographical region, specific gender, social class, etc
 - Compass tool: (2016) used by judges in the US to predict which criminals are likely to re-offend is found to be biased by the ethnicity (African-American/Caucasian).

Black-Box vs Interpretable Models

Black-Box vs Interpretable Models



Black-Box vs Interpretable Models



Background

Supervised/Unsupervised/Reinforcement Learning

Supervised/Unsupervised/Reinforcement Learning

- Supervised Learning (Labelled data): Predict a function that associates inputs to outputs based on historical data

Supervised/Unsupervised/Reinforcement Learning

- Supervised Learning (Labelled data): Predict a function that associates inputs to outputs based on historical data
 - Categorical labels (discrete values): Classification

Supervised/Unsupervised/Reinforcement Learning

- Supervised Learning (Labelled data): Predict a function that associates inputs to outputs based on historical data
 - Categorical labels (discrete values): Classification
 - Non-categorical labels (real numbers): Regression

Supervised/Unsupervised/Reinforcement Learning

- Supervised Learning (Labelled data): Predict a function that associates inputs to outputs based on historical data
 - Categorical labels (discrete values): Classification
 - Non-categorical labels (real numbers): Regression
- Unsupervised Learning: The task is to figure out patterns presented in the data (unlabelled data)

Supervised/Unsupervised/Reinforcement Learning

- Supervised Learning (Labelled data): Predict a function that associates inputs to outputs based on historical data
 - Categorical labels (discrete values): Classification
 - Non-categorical labels (real numbers): Regression
- Unsupervised Learning: The task is to figure out patterns presented in the data (unlabelled data)
- Reinforcement learning: learning from a series of rewards /punishments
- But also, depending on the problem, data could be both labelled/non labelled, etc.. (semi-supervised learning)

¹Image from https://en.wikipedia.org/wiki/Global_biodiversity
(Toulouse)



Figure 1: How to teach a child animal recognition? ¹

¹Image from https://en.wikipedia.org/wiki/Global_biodiversity



Figure 1: How to teach a child animal recognition? ¹

Classification task

¹Image from https://en.wikipedia.org/wiki/Global_biodiversity

²Image from <https://en.wikipedia.org/wiki>

(Toulouse)

INSA-Toulouse

June 7, 2024

11 / 93



Figure 2: How to predict a player's performance? ²

²Image from <https://en.wikipedia.org/wiki>



Figure 2: How to predict a player's performance? ²

Regression task

²Image from <https://en.wikipedia.org/wiki>

³Image from <https://en.wikipedia.org/wiki/DNA>

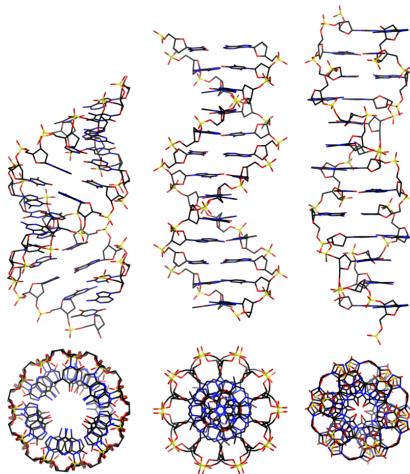


Figure 3: Analysis of evolutionary biology based on DNA patterns ³

³Image from <https://en.wikipedia.org/wiki/DNA>

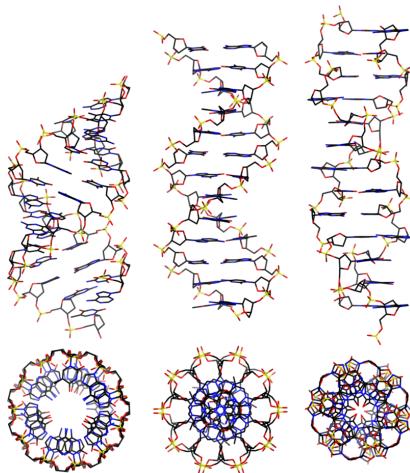


Figure 3: Analysis of evolutionary biology based on DNA patterns ³

Unsupervised learning (clustering) task

³Image from <https://en.wikipedia.org/wiki/DNA>

⁴Image from <https://en.wikipedia.org/wiki/Cycling>



Figure 4: How to cycle? ⁴

⁴Image from <https://en.wikipedia.org/wiki/Cycling>



Figure 4: How to cycle? ⁴

Reinforcement learning

⁴Image from <https://en.wikipedia.org/wiki/Cycling>

Problem Definition

Problem Definition

- Input: data (**training set**) in the form of input-output examples: $\{(x_1, y_1), \dots, (x_n, y_n)\}$ where x_i is an input, y_i is the output of x_i drawn from an unknown distribution

Problem Definition

- Input: data (**training set**) in the form of input-output examples: $\{(x_1, y_1), \dots, (x_n, y_n)\}$ where x_i is an input, y_i is the output of x_i drawn from an unknown distribution
- Find a function f_h (called a hypothesis or model) that approximates the true data distribution

Problem Definition

- Input: data (**training set**) in the form of input-output examples: $\{(x_1, y_1), \dots, (x_n, y_n)\}$ where x_i is an input, y_i is the output of x_i drawn from an unknown distribution
- Find a function f_h (called a hypothesis or model) that approximates the true data distribution
- The approximation criterion can be defined in different ways. We can consider it as a function minimizing some error.

Training Phase

Training Phase

- The function f_h is constructed via a **training algorithm**

Training Phase

- The function f_h is constructed via a **training algorithm**
- The training algorithm depends on the hypothesis space

Training Phase

- The function f_h is constructed via a **training algorithm**
- The training algorithm depends on the hypothesis space
- Examples of hypothesis space (family of functions) include polynomial functions, trigonometry functions, decision trees, decision lists, neural networks, . . .

Testing Phase

Testing Phase

- The evaluation of the constructed hypothesis (or model) is done via a set of unseen examples called **testing set**

Testing Phase

- The evaluation of the constructed hypothesis (or model) is done via a set of unseen examples called **testing set**
- The testing set is usually taken randomly from the initial data. However, it shouldn't be part of the training set

Testing Phase

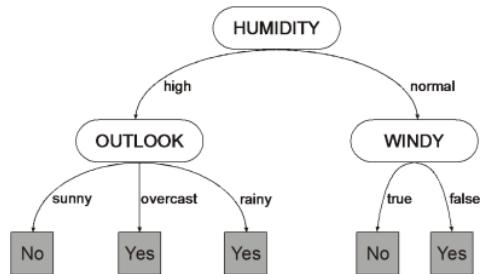
- The evaluation of the constructed hypothesis (or model) is done via a set of unseen examples called **testing set**
- The testing set is usually taken randomly from the initial data. However, it shouldn't be part of the training set
- The common way is to split the initial data into a training and testing sets randomly

Testing Phase

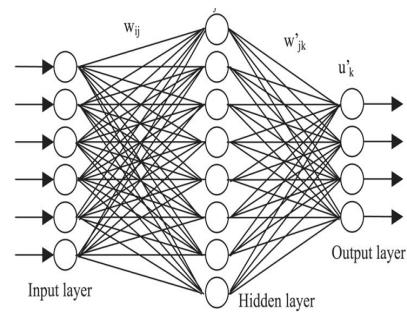
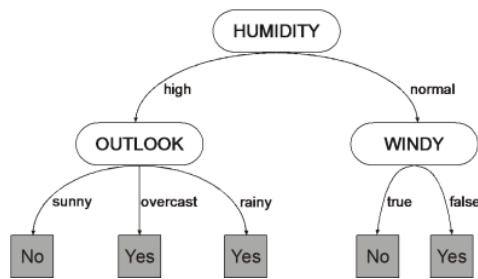
- The evaluation of the constructed hypothesis (or model) is done via a set of unseen examples called **testing set**
- The testing set is usually taken randomly from the initial data. However, it shouldn't be part of the training set
- The common way is to split the initial data into a training and testing sets randomly

Black-Box vs Interpretable Models

Black-Box vs Interpretable Models



Black-Box vs Interpretable Models



Black-Box vs Interpretable Models: Definitions

- **Black-box model** [1]: A formula that is either too complicated for any human to understand, or proprietary, so that one cannot understand its inner workings
- **Interpretable model:** any model whose operational principles are comprehensible to humans

Why Interpretable Models?

Why Interpretable Models?

- Transparent

Why Interpretable Models?

- Transparent
- Trustworthy

Why Interpretable Models?

- Transparent
- Trustworthy
- Inherently Explainable
- Well adapted for troubleshooting and diagnosis

Why Interpretable Models?

- Transparent
- Trustworthy
- Inherently Explainable
- Well adapted for troubleshooting and diagnosis
- **Mandatory criteria in high-stake decision making**

Interpretability vs. Explanability

The Debate & The 1 Million Dollars Reward

nature machine intelligence

Explore content ▾ About the journal ▾ Publish with us ▾ Subscribe

[nature](#) > [nature machine intelligence](#) > [perspectives](#) > [article](#)

Perspective | [Published: 13 May 2019](#)

Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead

[Cynthia Rudin](#) 

[Nature Machine Intelligence](#) 1, 206–215 (2019) | [Cite this article](#)

49k Accesses | 1020 Citations | 402 Altmetric | [Metrics](#)

 A [preprint version](#) of the article is available at arXiv.

<https://www.youtube.com/watch?v=4oXFEDoEcAk>

Explainable Machine Learning

- Explainable Machine Learning = the ability to explain the model predictions
- The notion of explanation is very complex (and philosophical) (see for instance the interview with Richard Feynman on the 'Why' question <https://www.youtube.com/watch?v=36GT2zI8lVA>)

Explainable Machine Learning

- Explainable Machine Learning = the ability to explain the model predictions
- The notion of explanation is very complex (and philosophical) (see for instance the interview with Richard Feynman on the 'Why' question <https://www.youtube.com/watch?v=36GT2zI8lVA>)
- Explain to who ?

Explainable Machine Learning

- Explainable Machine Learning = the ability to explain the model predictions
- The notion of explanation is very complex (and philosophical) (see for instance the interview with Richard Feynman on the 'Why' question <https://www.youtube.com/watch?v=36GT2zI8lVA>)
- Explain to who ?
- To explain predictions one needs a clear context to define explanations

Explainable Machine Learning

- Explainable Machine Learning = the ability to explain the model predictions
- The notion of explanation is very complex (and philosophical) (see for instance the interview with Richard Feynman on the 'Why' question <https://www.youtube.com/watch?v=36GT2zI8lVA>)
- Explain to who ?
- To explain predictions one needs a clear context to define explanations
- In machine learning, we usually use a subset of features that are 'responsible' for the prediction. That is, changing their values would change the prediction
- Explainability can be applied to black box models as a post processing step

Explainability

Explainability

- Explaining black box models is usually done by probing the model few times

Explainability

- Explaining black box models is usually done by probing the model few times
- Sometime it is not even possible to explain black box models

Explainability

- Explaining black box models is usually done by probing the model few times
- Sometime it is not even possible to explain black box models
- No theoretical guarantees

Explainability

- Explaining black box models is usually done by probing the model few times
- Sometime it is not even possible to explain black box models
- No theoretical guarantees
- **It gets worse!** Since different explanations can be used, one might pick a particular explanation to hide model biases (this is observed with many commercial tools!)

Explainability

- Explaining black box models is usually done by probing the model few times
- Sometime it is not even possible to explain black box models
- No theoretical guarantees
- **It gets worse!** Since different explanations can be used, one might pick a particular explanation to hide model biases (this is observed with many commercial tools!)
- Imagine a credit score black box model where a client might have several explanations regarding the refusal. The company might pick an explanation that doesn't show certain bias (such as predictions based on the gender)

Interpretability

- Interpretability guarantees the transparency of the explanations
- No post-processing (in the sense of probing the model) is necessary for explanations. It is enough to look at the model
- However sometimes the explanations are not optimal (in the size of set inclusion). In this case, a user might ask for minimal explanations. This task can be done as a post-processing step
- Unfortunately, interpretable models (so far) are not adapted to all applications (for instance in tumor detection and computer vision). Such applications depend heavily on recent advances of black box models

Think about it..



Geoffrey Hinton
@geoffreyhinton



Suppose you have cancer and you have to choose between a black box AI surgeon that cannot explain how it works but has a 90% cure rate and a human surgeon with an 80% cure rate. Do you want the AI surgeon to be illegal?

8:37 PM · Feb 20, 2020 · [Twitter Web App](#)

Interpretable Models

Case Study: Decision Trees

Case Study: Decision Trees

- Restrictions & notations: Tabular data with binary features and output

Case Study: Decision Trees

- Restrictions & notations: Tabular data with binary features and output
- Let $\mathbb{F} = \{f_1, \dots, f_k\}$ be a set of binary features (or attributes).

Case Study: Decision Trees

- Restrictions & notations: Tabular data with binary features and output
- Let $\mathbb{F} = \{f_1, \dots, f_k\}$ be a set of binary features (or attributes).
- An example e_i is represented as (x_1, \dots, x_k, y_i) where x_i are the values associated to the different features and $y_i \in \{0, 1\}$ is the class of e_i

Case Study: Decision Trees

- Restrictions & notations: Tabular data with binary features and output
- Let $\mathbb{F} = \{f_1, \dots, f_k\}$ be a set of binary features (or attributes).
- An example e_i is represented as (x_1, \dots, x_k, y_i) where x_i are the values associated to the different features and $y_i \in \{0, 1\}$ is the class of e_i
- Without loss of generality, we use the term 'positive' for the class 1 and 'negative' for the class 0
- The data is a collection of examples $\{e_1, \dots, e_n\}$

Toy Example: The Likelihood of Animal Extinction

Toy Example: The Likelihood of Animal Extinction

Big Size	Carnivore	Seasonal Reproduction	Solitary	Extinct
0	1	0	1	yes
1	0	0	1	yes
0	0	0	1	no
1	1	1	0	no
0	0	1	0	yes
0	1	1	0	yes
1	1	1	0	no

Toy Example: The Likelihood of Animal Extinction

Big Size	Carnivore	Seasonal Reproduction	Solitary	Extinct
0	1	0	1	yes
1	0	0	1	yes
0	0	0	1	no
1	1	1	0	no
0	0	1	0	yes
0	1	1	0	yes
1	1	1	0	no

- $k = 4$ binary features, $n = 7$ examples

Toy Example: The Likelihood of Animal Extinction

Big Size	Carnivore	Seasonal Reproduction	Solitary	Extinct
0	1	0	1	yes
1	0	0	1	yes
0	0	0	1	no
1	1	1	0	no
0	0	1	0	yes
0	1	1	0	yes
1	1	1	0	no

- $k = 4$ binary features, $n = 7$ examples
- Features $\mathbb{F} = \{ \text{Big size, Carnivore, Seasonal Reproduction, Solitary} \}$

Toy Example: The Likelihood of Animal Extinction

Big Size	Carnivore	Seasonal Reproduction	Solitary	Extinct
0	1	0	1	yes
1	0	0	1	yes
0	0	0	1	no
1	1	1	0	no
0	0	1	0	yes
0	1	1	0	yes
1	1	1	0	no

- $k = 4$ binary features, $n = 7$ examples
- Features $\mathbb{F} = \{ \text{Big size, Carnivore, Seasonal Reproduction, Solitary} \}$
- Binary output: Extinct (the animal is extinct)

Toy Example: The Likelihood of Animal Extinction

Big Size	Carnivore	Seasonal Reproduction	Solitary	Extinct
0	1	0	1	yes
1	0	0	1	yes
0	0	0	1	no
1	1	1	0	no
0	0	1	0	yes
0	1	1	0	yes
1	1	1	0	no

- $k = 4$ binary features, $n = 7$ examples
- Features $\mathbb{F} = \{ \text{Big size, Carnivore, Seasonal Reproduction, Solitary} \}$
- Binary output: Extinct (the animal is extinct)
- Example $e_6 = (0, 1, 1, 0, 1)$

Toy Example: The Likelihood of Animal Extinction

Big Size	Carnivore	Seasonal Reproduction	Solitary	Extinct
0	1	0	1	yes
1	0	0	1	yes
0	0	0	1	no
1	1	1	0	no
0	0	1	0	yes
0	1	1	0	yes
1	1	1	0	no

- $k = 4$ binary features, $n = 7$ examples
- Features $\mathbb{F} = \{ \text{Big size, Carnivore, Seasonal Reproduction, Solitary} \}$
- Binary output: Extinct (the animal is extinct)
- Example $e_6 = (0, 1, 1, 0, 1)$
- Data= $\{e_1, e_2, \dots e_7\}$

Definition of Decision Tree (in the case of binary classification)

- A decision tree is a binary tree where each leaf node corresponds to a binary value (positive/negative class) and each internal node j is associated to a feature $\text{feature}(j) \in \mathbb{F}$
- Let DT be a decision tree. Denote by $\text{feature}(j)$ the feature associated to node j in DT. We name the children of an internal node j as right and left. We also use $(j, r(j))$ ($(j, l(j))$ respectively) to denote the arc from a node j to its right (respectively left) child .

Definition of Decision Tree (in the case of binary classification)

- A decision tree is a binary tree where each leaf node corresponds to a binary value (positive/negative class) and each internal node j is associated to a feature $feature(j) \in \mathbb{F}$
- Let DT be a decision tree. Denote by $feature(j)$ the feature associated to node j in DT. We name the children of an internal node j as right and left. We also use $(j, r(j))$ ($(j, l(j))$ respectively) to denote the arc from a node j to its right (respectively left) child .
- Classifying an example e_i by a decision DT is done by following the path $P(e_i)$ from the root to a leaf node where $(j, r(j)) \in P(e_i)$ if $x(feature(j)) = 1$, otherwise $(j, l(j)) \in P(e_i)$. The leaf node of $P(e_i)$ is the class of e_i decided by DT

Definition of Decision Tree (in the case of binary classification)

- A decision tree is a binary tree where each leaf node corresponds to a binary value (positive/negative class) and each internal node j is associated to a feature $\text{feature}(j) \in \mathbb{F}$
- Let DT be a decision tree. Denote by $\text{feature}(j)$ the feature associated to node j in DT. We name the children of an internal node j as right and left. We also use $(j, r(j))$ ($(j, l(j))$ respectively) to denote the arc from a node j to its right (respectively left) child .
- Classifying an example e_i by a decision DT is done by following the path $P(e_i)$ from the root to a leaf node where $(j, r(j)) \in P(e_i)$ if $x(\text{feature}(j)) = 1$, otherwise $(j, l(j)) \in P(e_i)$. The leaf node of $P(e_i)$ is the class of e_i decided by DT
- This definition can be extended to multiple classification and regression (by adapting the leaf values)

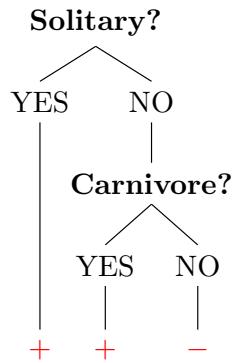
Toy Example: DTs to Predict The Likelihood of Animal Extinction

Toy Example: DTs to Predict The Likelihood of Animal Extinction

Big Size	Carnivore	Seasonal Reproduction	Solitary	Extinct
0	1	0	1	yes
1	0	0	1	yes
0	0	0	1	no
1	1	1	0	no
0	0	1	0	yes

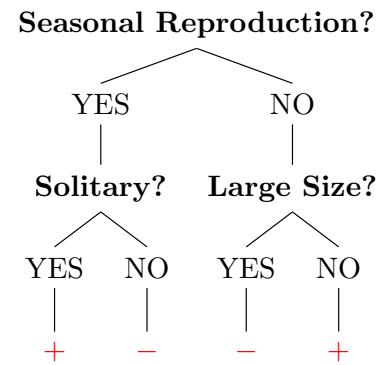
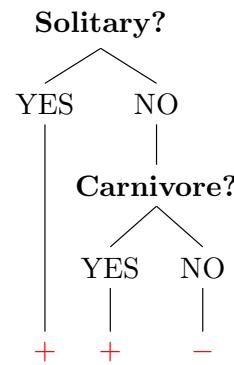
Toy Example: DTs to Predict The Likelihood of Animal Extinction

Big Size	Carnivore	Seasonal Reproduction	Solitary	Extinct
0	1	0	1	yes
1	0	0	1	yes
0	0	0	1	no
1	1	1	0	no
0	0	1	0	yes



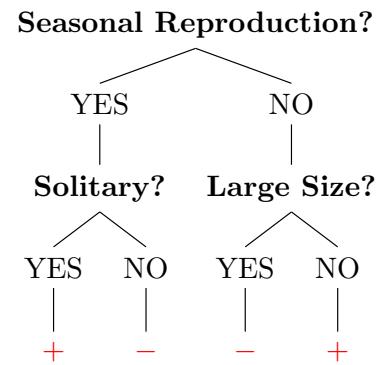
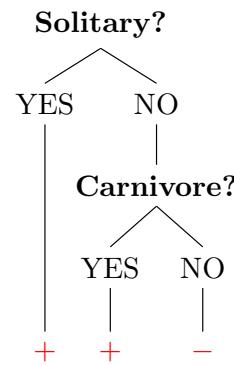
Toy Example: DTs to Predict The Likelihood of Animal Extinction

Big Size	Carnivore	Seasonal Reproduction	Solitary	Extinct
0	1	0	1	yes
1	0	0	1	yes
0	0	0	1	no
1	1	1	0	no
0	0	1	0	yes



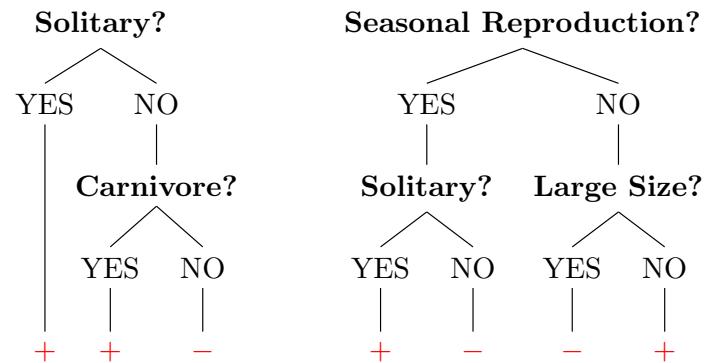
Toy Example: DTs to Predict The Likelihood of Animal Extinction

Big Size	Carnivore	Seasonal Reproduction	Solitary	Extinct
0	1	0	1	yes
1	0	0	1	yes
0	0	0	1	no
1	1	1	0	no
0	0	1	0	yes



Toy Example: DTs to Predict The Likelihood of Animal Extinction

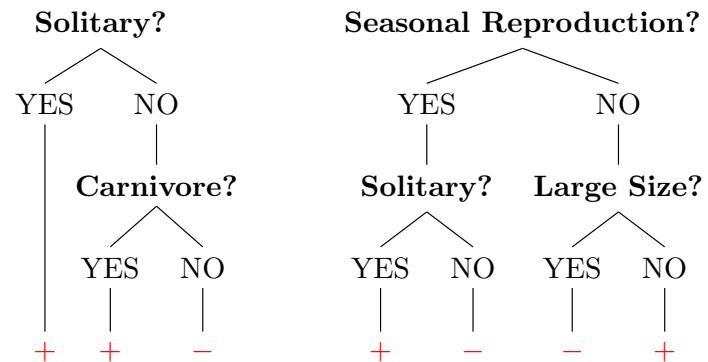
Big Size	Carnivore	Seasonal Reproduction	Solitary	Extinct
0	1	0	1	yes
1	0	0	1	yes
0	0	0	1	no
1	1	1	0	no
0	0	1	0	yes



- Tabular data

Toy Example: DTs to Predict The Likelihood of Animal Extinction

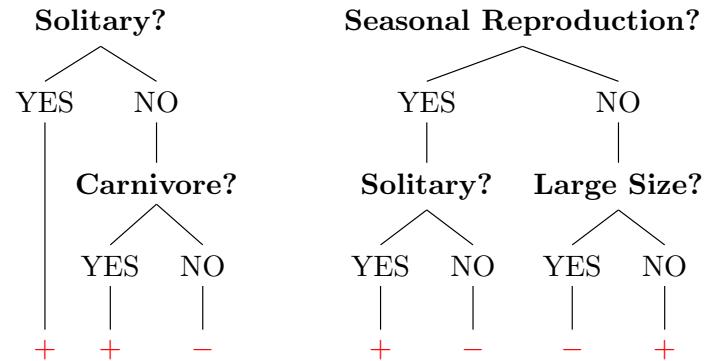
Big Size	Carnivore	Seasonal Reproduction	Solitary	Extinct
0	1	0	1	yes
1	0	0	1	yes
0	0	0	1	no
1	1	1	0	no
0	0	1	0	yes



- Tabular data
- Hypothesis space: Decision trees

Toy Example: DTs to Predict The Likelihood of Animal Extinction

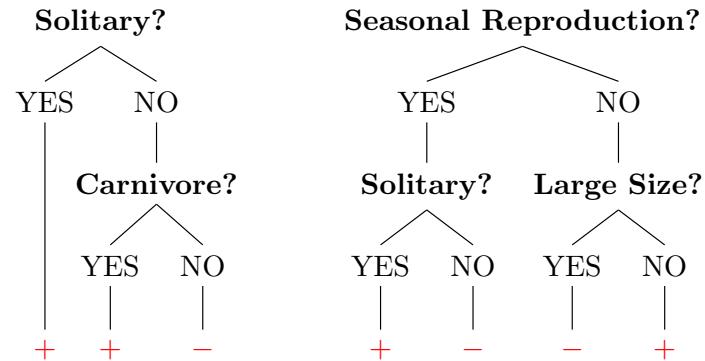
Big Size	Carnivore	Seasonal Reproduction	Solitary	Extinct
0	1	0	1	yes
1	0	0	1	yes
0	0	0	1	no
1	1	1	0	no
0	0	1	0	yes



- Tabular data
- Hypothesis space: Decision trees
- Left tree: accuracy:

Toy Example: DTs to Predict The Likelihood of Animal Extinction

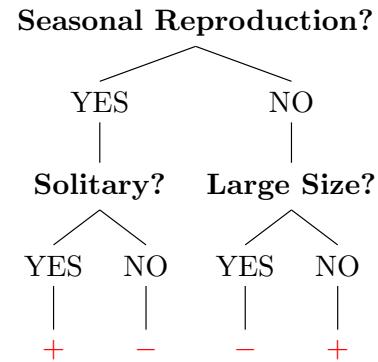
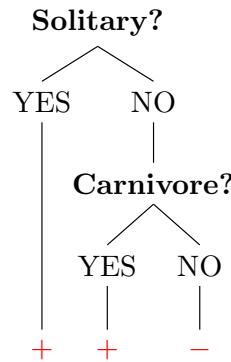
Big Size	Carnivore	Seasonal Reproduction	Solitary	Extinct
0	1	0	1	yes
1	0	0	1	yes
0	0	0	1	no
1	1	1	0	no
0	0	1	0	yes



- Tabular data
- Hypothesis space: Decision trees
- Left tree: accuracy: $2/5 = 40\%$

Toy Example: DTs to Predict The Likelihood of Animal Extinction

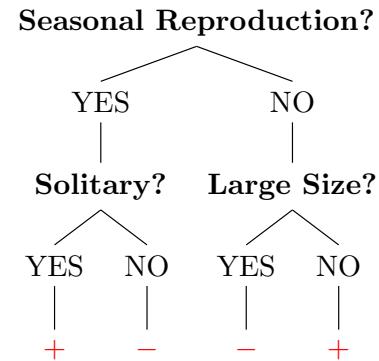
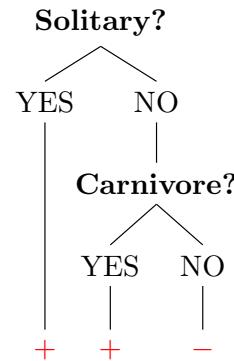
Big Size	Carnivore	Seasonal Reproduction	Solitary	Extinct
0	1	0	1	yes
1	0	0	1	yes
0	0	0	1	no
1	1	1	0	no
0	0	1	0	yes



- Tabular data
- Hypothesis space: Decision trees
- Left tree: accuracy: $2/5 = 40\%$
- Right tree: accuracy:

Toy Example: DTs to Predict The Likelihood of Animal Extinction

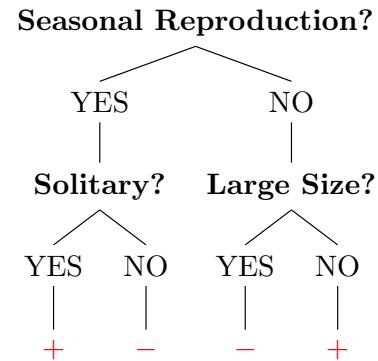
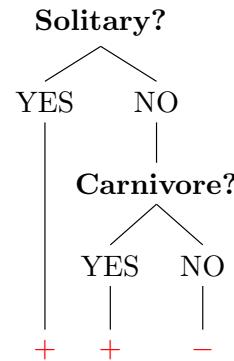
Big Size	Carnivore	Seasonal Reproduction	Solitary	Extinct
0	1	0	1	yes
1	0	0	1	yes
0	0	0	1	no
1	1	1	0	no
0	0	1	0	yes



- Tabular data
- Hypothesis space: Decision trees
- Left tree: accuracy: $2/5 = 40\%$
- Right tree: accuracy: $2/5 = 40\%$

Toy Example: DTs to Predict The Likelihood of Animal Extinction

Big Size	Carnivore	Seasonal Reproduction	Solitary	Extinct
0	1	0	1	yes
1	0	0	1	yes
0	0	0	1	no
1	1	1	0	no
0	0	1	0	yes



- Tabular data
- Hypothesis space: Decision trees
- Left tree: accuracy: $2/5 = 40\%$
- Right tree: accuracy: $2/5 = 40\%$

Dealing with Intractability

Dealing with Intractability

Dealing with Intractability

- Enormous search space

Dealing with Intractability

- Enormous search space
- Building short trees under specific constraints is usually intractable

Dealing with Intractability

- Enormous search space
- Building short trees under specific constraints is usually intractable
- Exact algorithms hardly scale up

Dealing with Intractability

- Enormous search space
- Building short trees under specific constraints is usually intractable
- Exact algorithms hardly scale up
- Most of the approaches are greedy (heuristic) approaches

Dealing with Intractability

- Enormous search space
- Building short trees under specific constraints is usually intractable
- Exact algorithms hardly scale up
- Most of the approaches are greedy (heuristic) approaches
- Greedy algorithms follow a top-down approach: at each step, choose the best feature (to split the data) then recursively apply the same for the children until a certain stopping criterion

Building a decision Tree

- Decision trees can be represented as follows $(f, right, left)$ where f is a feature and $right$ (respectively $left$) are either decision trees or binary values (an outcome)
- We use the following oracles (functions):
 - $SelectBestFeature(data)$: select the best splitting feature according to some criterion
 - $UpdateInformation(Tree, Node)$: update information related to a given stopping requirement
 - $SelectClass(\mathbb{E})$: returns a class according to a selection criterion
 - $Explore(\mathbb{E}, info)$: a Boolean that indicates if the algorithm should develop more the tree
- The following is a high level greedy algorithm:

Building a Decision Tree: A Greedy Algorithm

Algorithm 1 GREEDY

Require: $\mathbb{F} = \{f_1, \dots, f_k\}$, $\mathbb{E} = \{e_1, \dots, e_n\}$, a parent node *parent*, and an information *info* regarding the stopping conditions

Result: A decision tree

if *Explore*(\mathbb{E}, info) **then**

Building a Decision Tree: A Greedy Algorithm

Algorithm 2 GREEDY

Require: $\mathbb{F} = \{f_1, \dots, f_k\}$, $\mathbb{E} = \{e_1, \dots, e_n\}$, a parent node *parent*, and an information *info* regarding the stopping conditions

Result: A decision tree

if *Explore*(\mathbb{E}, info) **then**

$f_j \leftarrow \text{SelectBestFeature}(data)$

Building a Decision Tree: A Greedy Algorithm

Algorithm 3 GREEDY

Require: $\mathbb{F} = \{f_1, \dots, f_k\}$, $\mathbb{E} = \{e_1, \dots, e_n\}$, a parent node *parent*, and an information *info* regarding the stopping conditions

Result: A decision tree

if *Explore*(\mathbb{E}, info) **then**

$f_j \leftarrow \text{SelectBestFeature}(data)$

$L \leftarrow \{x \in E | f_j = 0\}; R \leftarrow \{x \in E | f_j = 1\};$

Building a Decision Tree: A Greedy Algorithm

Algorithm 4 GREEDY

Require: $\mathbb{F} = \{f_1, \dots, f_k\}$, $\mathbb{E} = \{e_1, \dots, e_n\}$, a parent node *parent*, and an information *info* regarding the stopping conditions

Result: A decision tree

if *Explore*(\mathbb{E}, info) **then**

$f_j \leftarrow \text{SelectBestFeature}(data)$

$L \leftarrow \{x \in E | f_j = 0\}; R \leftarrow \{x \in E | f_j = 1\};$

$\text{LeftInfo} \leftarrow \text{UpdateInformation}(L, \text{parent})$;

Building a Decision Tree: A Greedy Algorithm

Algorithm 5 GREEDY

Require: $\mathbb{F} = \{f_1, \dots, f_k\}$, $\mathbb{E} = \{e_1, \dots, e_n\}$, a parent node *parent*, and an information *info* regarding the stopping conditions

Result: A decision tree

if *Explore*(\mathbb{E}, info) **then**

$f_j \leftarrow \text{SelectBestFeature}(\text{data})$

$L \leftarrow \{x \in \mathbb{E} | f_j = 0\}; R \leftarrow \{x \in \mathbb{E} | f_j = 1\};$

$\text{LeftInfo} \leftarrow \text{UpdateInformation}(L, \text{parent});$

$\text{RightInfo} \leftarrow \text{UpdateInformation}(R, \text{parent});$

Building a Decision Tree: A Greedy Algorithm

Algorithm 6 GREEDY

Require: $\mathbb{F} = \{f_1, \dots, f_k\}$, $\mathbb{E} = \{e_1, \dots, e_n\}$, a parent node *parent*, and an information *info* regarding the stopping conditions

Result: A decision tree

if *Explore*(\mathbb{E}, info) **then**

$f_j \leftarrow \text{SelectBestFeature}(\text{data})$

$L \leftarrow \{x \in \mathbb{E} | f_j = 0\}; R \leftarrow \{x \in \mathbb{E} | f_j = 1\};$

$\text{LeftInfo} \leftarrow \text{UpdateInformation}(L, \text{parent});$

$\text{RightInfo} \leftarrow \text{UpdateInformation}(R, \text{parent});$

$\text{LeftTree} \leftarrow \text{GREEDY}(\mathbb{F} \setminus f_j, L, f_j, \text{LeftInformation});$

Building a Decision Tree: A Greedy Algorithm

Algorithm 7 GREEDY

Require: $\mathbb{F} = \{f_1, \dots, f_k\}$, $\mathbb{E} = \{e_1, \dots, e_n\}$, a parent node *parent*, and an information *info* regarding the stopping conditions

Result: A decision tree

if *Explore*(\mathbb{E}, info) **then**

$f_j \leftarrow \text{SelectBestFeature}(\text{data})$

$L \leftarrow \{x \in \mathbb{E} | f_j = 0\}; R \leftarrow \{x \in \mathbb{E} | f_j = 1\};$

$\text{LeftInfo} \leftarrow \text{UpdateInformation}(L, \text{parent});$

$\text{RightInfo} \leftarrow \text{UpdateInformation}(R, \text{parent});$

$\text{LeftTree} \leftarrow \text{GREEDY}(\mathbb{F} \setminus f_j, L, f_j, \text{LeftInformation});$

$\text{RightTree} \leftarrow \text{GREEDY}(\mathbb{F} \setminus f_j, R, f_j, \text{RightInformation});$

Building a Decision Tree: A Greedy Algorithm

Algorithm 8 GREEDY

Require: $\mathbb{F} = \{f_1, \dots, f_k\}$, $\mathbb{E} = \{e_1, \dots, e_n\}$, a parent node *parent*, and an information *info* regarding the stopping conditions

Result: A decision tree

if *Explore*(\mathbb{E}, info) **then**

$f_j \leftarrow \text{SelectBestFeature}(\text{data})$

$L \leftarrow \{x \in \mathbb{E} | f_j = 0\}; R \leftarrow \{x \in \mathbb{E} | f_j = 1\};$

LeftInfo $\leftarrow \text{UpdateInformation}(L, \text{parent})$;

RightInfo $\leftarrow \text{UpdateInformation}(R, \text{parent})$;

LeftTree $\leftarrow \text{GREEDY}(\mathbb{F} \setminus f_j, L, f_j, \text{LeftInformation})$;

RightTree $\leftarrow \text{GREEDY}(\mathbb{F} \setminus f_j, R, f_j, \text{RightInformation})$;

return ($f_j, \text{LeftTree}, \text{RightTree}$)

else

return *SelectClass*(\mathbb{E})

end if;

Decision Trees for Regression? Yes!

Decision Trees for Regression? Yes!

Algorithm 10 GREEDY

Require: $\mathbb{F} = \{f_1, \dots, f_k\}$, $\mathbb{E} = \{e_1, \dots, e_n\}$, a parent node *parent*, and an information *info* regarding the stopping conditions

Result: A decision tree

if *Explore*($\mathbb{E}, \textit{info}$) **then**

Decision Trees for Regression? Yes!

Algorithm 11 GREEDY

Require: $\mathbb{F} = \{f_1, \dots, f_k\}$, $\mathbb{E} = \{e_1, \dots, e_n\}$, a parent node *parent*, and an information *info* regarding the stopping conditions

Result: A decision tree

if *Explore*(\mathbb{E}, info) **then**

$f_j \leftarrow \text{SelectBestFeature}(data)$

Decision Trees for Regression? Yes!

Algorithm 12 GREEDY

Require: $\mathbb{F} = \{f_1, \dots, f_k\}$, $\mathbb{E} = \{e_1, \dots, e_n\}$, a parent node *parent*, and an information *info* regarding the stopping conditions

Result: A decision tree

if *Explore*(\mathbb{E}, info) **then**

$f_j \leftarrow \text{SelectBestFeature}(data)$

$L \leftarrow \{x \in E | f_j = 0\}; R \leftarrow \{x \in E | f_j = 1\};$

Decision Trees for Regression? Yes!

Algorithm 13 GREEDY

Require: $\mathbb{F} = \{f_1, \dots, f_k\}$, $\mathbb{E} = \{e_1, \dots, e_n\}$, a parent node *parent*, and an information *info* regarding the stopping conditions

Result: A decision tree

if *Explore*(\mathbb{E}, info) **then**

$f_j \leftarrow \text{SelectBestFeature}(data)$

$L \leftarrow \{x \in E | f_j = 0\}; R \leftarrow \{x \in E | f_j = 1\};$

$\text{LeftInfo} \leftarrow \text{UpdateInformation}(L, \text{parent})$;

Decision Trees for Regression? Yes!

Algorithm 14 GREEDY

Require: $\mathbb{F} = \{f_1, \dots, f_k\}$, $\mathbb{E} = \{e_1, \dots, e_n\}$, a parent node *parent*, and an information *info* regarding the stopping conditions

Result: A decision tree

if *Explore*(\mathbb{E}, info) **then**

$f_j \leftarrow \text{SelectBestFeature}(\text{data})$

$L \leftarrow \{x \in E | f_j = 0\}; R \leftarrow \{x \in E | f_j = 1\};$

$\text{LeftInfo} \leftarrow \text{UpdateInformation}(L, \text{parent});$

$\text{RightInfo} \leftarrow \text{UpdateInformation}(R, \text{parent});$

Decision Trees for Regression? Yes!

Algorithm 15 GREEDY

Require: $\mathbb{F} = \{f_1, \dots, f_k\}$, $\mathbb{E} = \{e_1, \dots, e_n\}$, a parent node *parent*, and an information *info* regarding the stopping conditions

Result: A decision tree

if *Explore*(\mathbb{E}, info) **then**

$f_j \leftarrow \text{SelectBestFeature}(\text{data})$

$L \leftarrow \{x \in E | f_j = 0\}; R \leftarrow \{x \in E | f_j = 1\};$

$\text{LeftInfo} \leftarrow \text{UpdateInformation}(L, \text{parent});$

$\text{RightInfo} \leftarrow \text{UpdateInformation}(R, \text{parent});$

$\text{LeftTree} \leftarrow \text{GREEDY}(\mathbb{F} \setminus f_j, L, f_j, \text{LeftInformation});$

Decision Trees for Regression? Yes!

Algorithm 16 GREEDY

Require: $\mathbb{F} = \{f_1, \dots, f_k\}$, $\mathbb{E} = \{e_1, \dots, e_n\}$, a parent node *parent*, and an information *info* regarding the stopping conditions

Result: A decision tree

if *Explore*(\mathbb{E}, info) **then**

$f_j \leftarrow \text{SelectBestFeature}(\text{data})$

$L \leftarrow \{x \in \mathbb{E} | f_j = 0\}; R \leftarrow \{x \in \mathbb{E} | f_j = 1\};$

$\text{LeftInfo} \leftarrow \text{UpdateInformation}(L, \text{parent})$;

$\text{RightInfo} \leftarrow \text{UpdateInformation}(R, \text{parent})$;

$\text{LeftTree} \leftarrow \text{GREEDY}(\mathbb{F} \setminus f_j, L, f_j, \text{LeftInformation})$;

$\text{RightTree} \leftarrow \text{GREEDY}(\mathbb{F} \setminus f_j, R, f_j, \text{RightInformation})$;

Decision Trees for Regression? Yes!

Algorithm 17 GREEDY

Require: $\mathbb{F} = \{f_1, \dots, f_k\}$, $\mathbb{E} = \{e_1, \dots, e_n\}$, a parent node *parent*, and an information *info* regarding the stopping conditions

Result: A decision tree

if *Explore*(\mathbb{E}, info) **then**

$f_j \leftarrow \text{SelectBestFeature}(\text{data})$

$L \leftarrow \{x \in \mathbb{E} | f_j = 0\}; R \leftarrow \{x \in \mathbb{E} | f_j = 1\};$

LeftInfo $\leftarrow \text{UpdateInformation}(L, \text{parent})$;

RightInfo $\leftarrow \text{UpdateInformation}(R, \text{parent})$;

LeftTree $\leftarrow \text{GREEDY}(\mathbb{F} \setminus f_j, L, f_j, \text{LeftInformation})$;

RightTree $\leftarrow \text{GREEDY}(\mathbb{F} \setminus f_j, R, f_j, \text{RightInformation})$;

return ($f_j, \text{LeftTree}, \text{RightTree}$)

else

return *SelectValue*(\mathbb{E})

end if;

Information Gain

- There are several ways to choose a 'good' feature
- The Information Gain is one of the most used criterion
- It uses the notion of Entropy that evaluates data uncertainty
(initially proposed in the context of information theory by Shanon and Weaver)

Entropy

Entropy

- Entropy is a statistical measure (proposed by Claude Shannon and Weaver) as the number of bits needed to represent uncertainty

Entropy

- Entropy is a statistical measure (proposed by Claude Shannon and Weaver) as the number of bits needed to represent uncertainty
- Imagine you toss a normal coin. Both heads and tails have a 50% chance to occur

Entropy

- Entropy is a statistical measure (proposed by Claude Shannon and Weaver) as the number of bits needed to represent uncertainty
- Imagine you toss a normal coin. Both heads and tails have a 50% chance to occur
- Guessing the outcome of the toss is highly uncertain because of the equal chances

Entropy

- Entropy is a statistical measure (proposed by Claude Shannon and Weaver) as the number of bits needed to represent uncertainty
- Imagine you toss a normal coin. Both heads and tails have a 50% chance to occur
- Guessing the outcome of the toss is highly uncertain because of the equal chances
- In this case, $Entropy = 1$

Entropy

- Entropy is a statistical measure (proposed by Claude Shannon and Weaver) as the number of bits needed to represent uncertainty
- Imagine you toss a normal coin. Both heads and tails have a 50% chance to occur
- Guessing the outcome of the toss is highly uncertain because of the equal chances
- In this case, $Entropy = 1$
- In the other extreme, a coin with heads on both sides has no uncertainty because of the constant outcome (always heads)
- In this case, $Entropy = 0$

Entropy

Let Y be a discrete random variable taking values y_j , the entropy of Y is defined as follows:

$$H(Y) = \sum_j P(y_j) \times \log_2(1/P(y_j))$$

where $P(y_j)$ is the probability of the value y_j

Example: For a fair coin:

$$H(Y) = 0.5 \times \log_2(2) + 0.5 \times \log_2(2) = 1$$

For a coin with 90% with heads chance:

$$H(V) = 0.9 \times \log_2(10/9) + 0.1 \times \log_2(10) = 0.46$$

Entropy in the case of binary decision trees

Entropy in the case of binary decision trees

- Back to binary classification with a set E of n examples containing a positive examples and b negative examples. Consider the classification outcome as a random variable. We denote the entropy of this Boolean random variable as $H(data)$
- For a feature f_j , we define $n_1 = |E_1|$ where $E_1 = E \setminus \{x|x_j = 1\}$ and $n_0 = |E_0|$ where $E_0 = E \setminus \{x|x_j = 0\}$. We also denote by a_1 (respectively a_0) the number of positive examples in E_1 (respectively E_0) and by b_1 (respectively b_0) the number of negative examples in E_1 (respectively E_0)

Entropy in the case of binary decision trees

The expected entropy after splitting the data with the f_j is

$$\text{Remaining}(f_j) = n_1/n \times H(E_1) + n_0/n \times H(E_0)$$

We are looking for a feature that has a low level of uncertainty when splitting the data. A good splitter f_j is a feature with a minimum value of $\text{Remaining}(f_j)$ (this measures how much uncertainty is removed from the data).

This is equivalent to maximizing the *information gain* (IG):

$$IG(f_j) = 1 - \text{Remaining}(f_j)$$

Back to Greedy Algorithms

Back to Greedy Algorithms

- Different algorithms use different criteria

Back to Greedy Algorithms

- Different algorithms use different criteria
- For instance, Information gain is used in C4.5 [2] and Gini Index is used in in CART (Classification and regression trees [3])

Back to Greedy Algorithms

- Different algorithms use different criteria
- For instance, Information gain is used in C4.5 [2] and Gini Index is used in in CART (Classification and regression trees [3])
- Information gain reflects uncertainty and the purpose is to favor the feature that minimize uncertainty

Back to Greedy Algorithms

- Different algorithms use different criteria
- For instance, Information gain is used in C4.5 [2] and Gini Index is used in in CART (Classification and regression trees [3])
- Information gain reflects uncertainty and the purpose is to favor the feature that minimize uncertainty
- *Gini Index (GI)* reflects the purity of the data.

Back to Greedy Algorithms

- Different algorithms use different criteria
- For instance, Information gain is used in C4.5 [2] and Gini Index is used in in CART (Classification and regression trees [3])
- Information gain reflects uncertainty and the purpose is to favor the feature that minimize uncertainty
- *Gini Index (GI)* reflects the purity of the data. The values range from 0 to 1 where 0 represents a pure data (with one class), 1 represents a random distribution, and 0.5 represents a completely equal distribution.

Back to Greedy Algorithms

- Different algorithms use different criteria
- For instance, Information gain is used in C4.5 [2] and Gini Index is used in in CART (Classification and regression trees [3])
- Information gain reflects uncertainty and the purpose is to favor the feature that minimize uncertainty
- *Gini Index (GI)* reflects the purity of the data. The values range from 0 to 1 where 0 represents a pure data (with one class), 1 represents a random distribution, and 0.5 represents a completely equal distribution. The chosen split is to the one that minimizes the GI

Back to Greedy Algorithms

- Different algorithms use different criteria
- For instance, Information gain is used in C4.5 [2] and Gini Index is used in in CART (Classification and regression trees [3])
- Information gain reflects uncertainty and the purpose is to favor the feature that minimize uncertainty
- *Gini Index (GI)* reflects the purity of the data. The values range from 0 to 1 where 0 represents a pure data (with one class), 1 represents a random distribution, and 0.5 represents a completely equal distribution. The chosen split is to the one that minimizes the GI
- Both algorithms are efficient in practice, however without guarantee of optimality
- A trend is observed recently to build optimal DTs (for instance [4, 5])

Exercise: The Likelihood of Animal Extinction

Exercise: The Likelihood of Animal Extinction

Big Size	Carnivore	Seasonal Reproduction	Solitary	Extinct
0	1	0	1	yes
1	0	0	1	yes
0	0	0	1	no
1	1	1	0	no
0	0	1	0	yes
0	1	1	0	yes
1	1	1	0	no

Exercise: The Likelihood of Animal Extinction

Big Size	Carnivore	Seasonal Reproduction	Solitary	Extinct
0	1	0	1	yes
1	0	0	1	yes
0	0	0	1	no
1	1	1	0	no
0	0	1	0	yes
0	1	1	0	yes
1	1	1	0	no

- What is the value of information gain of the original dataset ?
- Which feature is better according to the information gain value (between Big Size and Solitary)?
- Build a decision tree with the previous approach where the height is at most 3, and the classification follows a majority rule

Pruning as a post processing

Pruning as a post processing

- Any training algorithm might build dense and long trees. This might induce overfitting

Pruning as a post processing

- Any training algorithm might build dense and long trees. This might induce overfitting
- A simple way to overcome this issue is to ‘trim’ the tree as a post-processing step by removing useless branches or nodes (the ones causing overfitting)

Pruning as a post processing

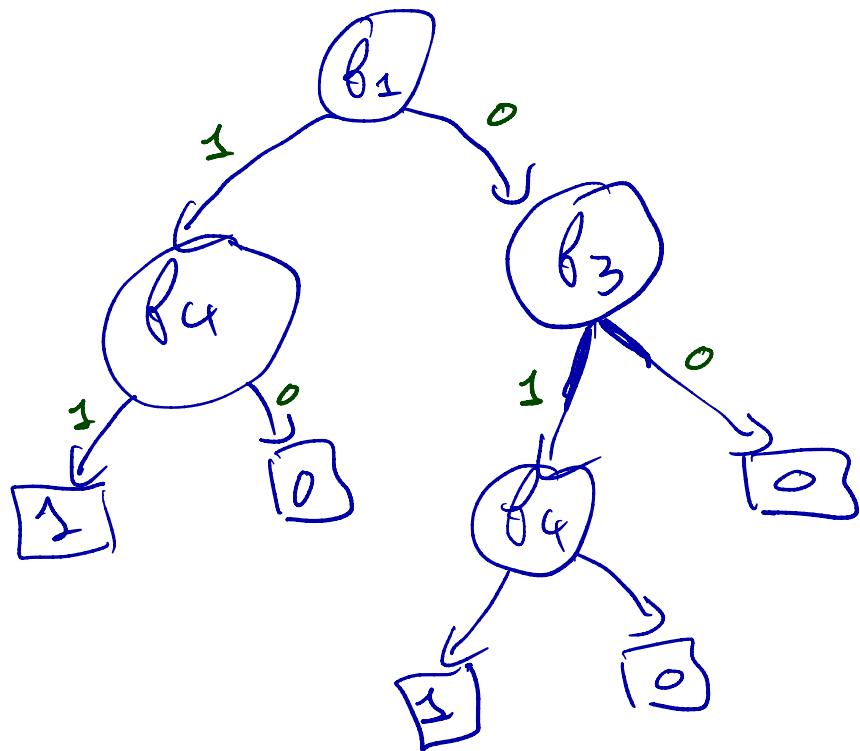
- Any training algorithm might build dense and long trees. This might induce overfitting
- A simple way to overcome this issue is to ‘trim’ the tree as a post-processing step by removing useless branches or nodes (the ones causing overfitting)
- Useless branches are typically long and are used to classify a limited number of examples

Pruning as a post processing

- Any training algorithm might build dense and long trees. This might induce overfitting
- A simple way to overcome this issue is to ‘trim’ the tree as a post-processing step by removing useless branches or nodes (the ones causing overfitting)
- Useless branches are typically long and are used to classify a limited number of examples
- The post processing might include other operations such as removing redundant sub-trees and useless splits

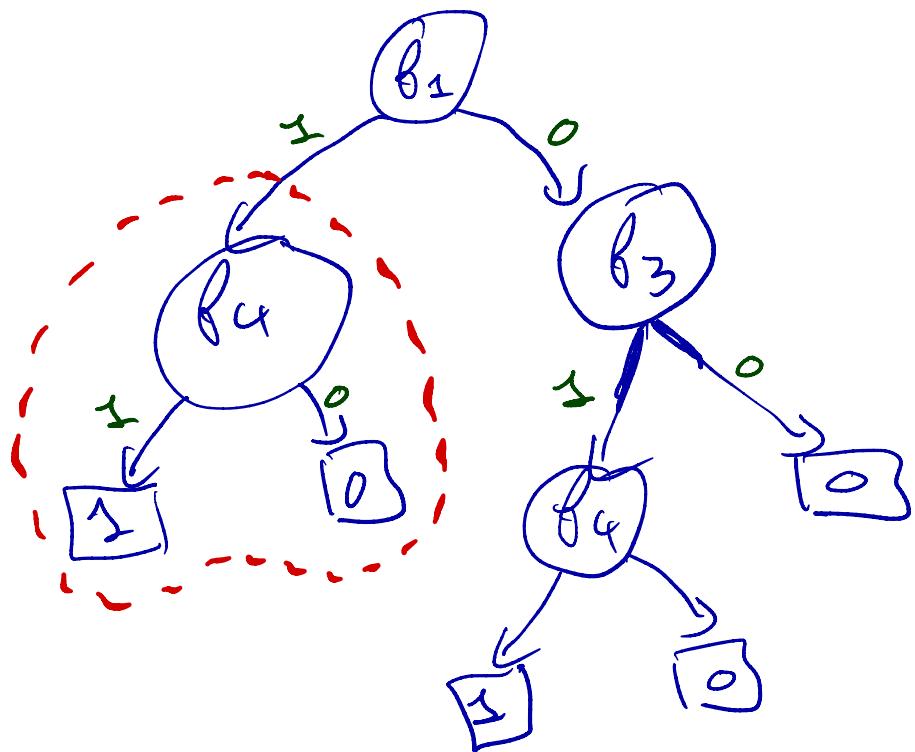
Something can be improved, can you find it ?

Something can be improved, can you find it ?



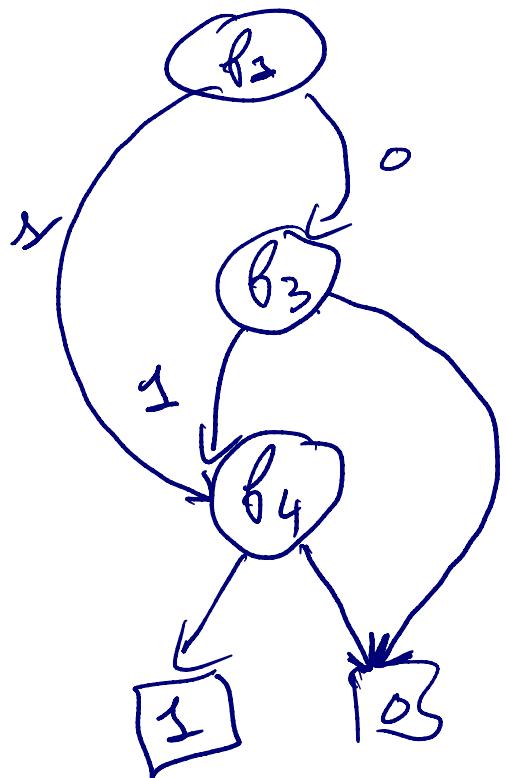
Something can be improved, can you guess it ?

Something can be improved, can you guess it ?



Binary Decision Diagram as an Alternative Model

Binary Decision Diagram as an Alternative Model



Optimal Decision Trees

Optimal Decision Trees

- Let $\mathbb{F} = \{f_1, \dots, f_k\}$ is a set of binary features
- $\mathbb{E} = \{e_1, \dots, e_n\}$ is the training dataset
- Propose an algorithm to find an optimal decision tree given a certain height with the minimum misclassification error

Optimal Decision Trees

- Let $\mathbb{F} = \{f_1, \dots, f_k\}$ is a set of binary features
- $\mathbb{E} = \{e_1, \dots, e_n\}$ is the training dataset
- Propose an algorithm to find an optimal decision tree given a certain height with the minimum misclassification error
- You can use the following oracles:
 - $SelectClass(\mathbb{E})$: returns the most probable class in the dataset \mathbb{E}
 - $Explore(\mathbb{E}, info)$: a Boolean that indicates if the algorithm should develop more the tree

A Recursive Exact Algorithm To Find an Optimal Tree

Algorithm 18 OptimalTree

Require: $\mathbb{F} = \{f_1, \dots, f_k\}$, $\mathbb{E} = \{e_1, \dots, e_n\}$, a parent node *parent*, and an information *info* regarding the stopping conditions

Result: $(\text{Decisiontree}, \text{Error})$

```

if Explore( $\mathbb{E}$ , info) then
    for  $j \in [1, K]$  do
         $Left_j \leftarrow \{x \in E | f_j = 0\}$ 
         $Right_j \leftarrow \{x \in E | f_j = 1\}$ 
         $Info_j \leftarrow$  Stopping information as if  $f_j$  is selected
         $(RightTree_j, RightError_j) \leftarrow OptimalTree(F \setminus \{f_j\}, Right_j, f_j, Info_j)$ 
         $(LeftTree_j, LeftError_j) \leftarrow OptimalTree(F \setminus \{f_j\}, Left_j, f_j, Info_j)$ 
         $Error_j \leftarrow RightError_j + LeftError_j$ 
    end for
     $j^* \leftarrow ArgMin\{Error_j\}$ 
    return  $(f_{j^*}, LeftTree_{j^*}, RightTree_{j^*})$ 
else
     $C \leftarrow SelectClass(\mathbb{E})$ 
     $FixedError \leftarrow$  Error when choosing C
    return  $(SelectClass(\mathbb{E}), FixedError)$ 
end if;

```

Search Symmetry

Search Symmetry

- Any two branches that share the exact same features (in different order) have the exact same optimal solutions

Search Symmetry

- Any two branches that share the exact same features (in different order) have the exact same optimal solutions
- In the recursive algorithm, one can avoid this redundant calls by caching the results of each branch
- The caching contains strictly different branches

A Recursive Exact Algorithm To Find an Optimal Tree

Algorithm 19 OptimalTree

Require: $F = \{f_1, \dots, f_k\}$, $\mathbb{E} = \{e_1, \dots, e_n\}$, a parent node *parent*, and an information *info* regarding the stopping conditions, $B = \text{current branch}$

Result: $(\text{Decisiontree}, \text{Error})$

```

if Explore( $\mathbb{E}$ , info) then
    for  $j \in [1, K]$  do
        if Saved( $B + j$ )  $\neq \emptyset$  then  $(\text{Error}_j, \text{RightTree}_j, \text{LeftTree}_j) = \text{Saved}(B + j)$ 
        else
             $\text{Left}_j \leftarrow \{x \in E | f_j = 0\}$ 
             $\text{Right}_j \leftarrow \{x \in E | f_j = 1\}$ 
             $\text{Info}_j \leftarrow \text{Stopping information as if } f_j \text{ is selected}$ 
             $(\text{RightTree}_j, \text{RightError}_j) \leftarrow \text{OptimalTree}(F \setminus \{f_j\}, \text{Right}_j, f_j, \text{Info}_j)$ 
             $(\text{LeftTree}_j, \text{LeftError}_j) \leftarrow \text{OptimalTree}(F \setminus \{f_j\}, \text{Left}_j, f_j, \text{Info}_j)$ 
             $\text{Error}_j \leftarrow \text{RightError}_j + \text{LeftError}_j$ 
             $\text{Saved}(B + j) = (\text{Error}_j, \text{RightTree}_j, \text{LeftTree}_j)$ 
        end if;
    end for
     $j^* \leftarrow \text{ArgMin}\{\text{Error}_j\}$ 
    return  $(f_{j^*}, \text{LeftTree}_{j^*}, \text{RightTree}_{j^*})$ 
else
     $C \leftarrow \text{SelectClass}(\mathbb{E})$ 
     $\text{FixedError} \leftarrow \text{Error when choosing } C$ 
    return  $(\text{SelectClass}(\mathbb{E}), \text{FixedError})$ 
end if;

```

Random Forest, and Boosted Trees

- Ensemble Learning is a learning methodology that relies on training a number of prediction models. The idea is to improve a constructed model by using instead a set of models

Random Forest, and Boosted Trees

- Ensemble Learning is a learning methodology that relies on training a number of prediction models. The idea is to improve a constructed model by using instead a set of models
- There are two types of ensemble learning: Boosting and Bagging

Random Forest, and Boosted Trees

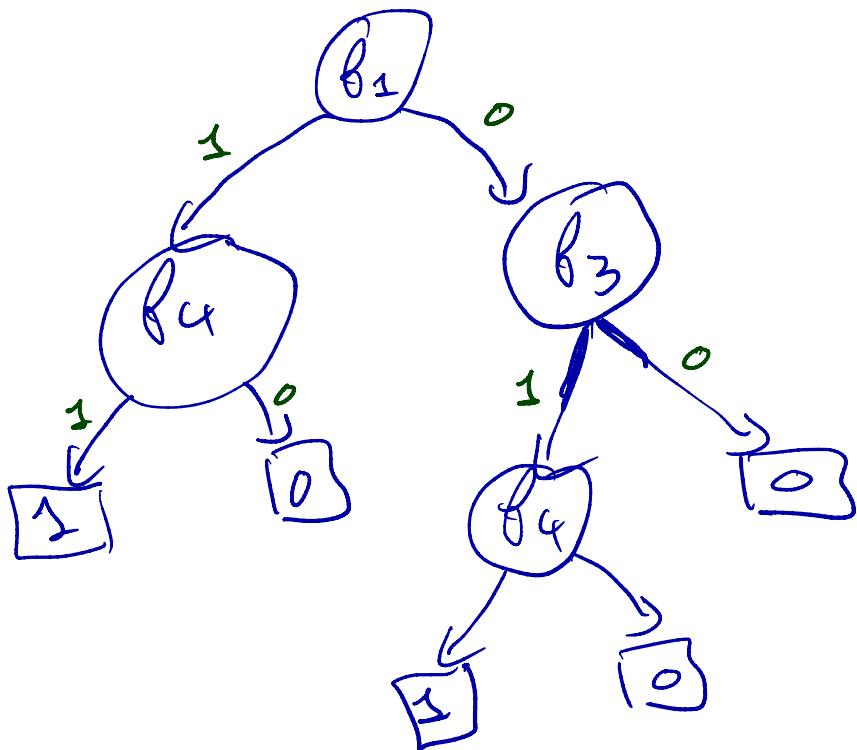
- Ensemble Learning is a learning methodology that relies on training a number of prediction models. The idea is to improve a constructed model by using instead a set of models
- There are two types of ensemble learning: Boosting and Bagging
- Bagging is a technique that learns several models by randomly selecting a subset of the data for each model. The predictions are made based on majority vote (in case of binary classification). In the case of bagging with decision trees, the model is called *random forest*

Random Forest, and Boosted Trees

- Ensemble Learning is a learning methodology that relies on training a number of prediction models. The idea is to improve a constructed model by using instead a set of models
- There are two types of ensemble learning: Boosting and Bagging
- Bagging is a technique that learns several models by randomly selecting a subset of the data for each model. The predictions are made based on majority vote (in case of binary classification). In the case of bagging with decision trees, the model is called *random forest*
- Boosting is a technique that learns several models in a sequence where each model relies on the mistakes of the previous ones to improve the quality of the learning. Usually, when boosting a model, each example is weighted by how many times it is badly classified in order to give it an advantage

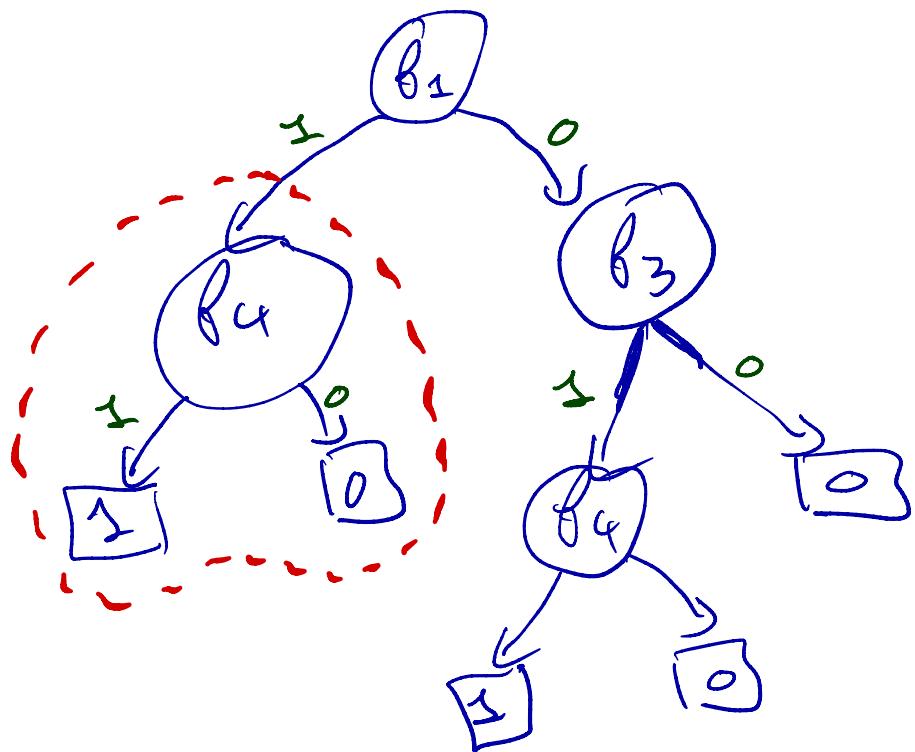
Binary Decision Diagrams

Binary Decision Diagrams



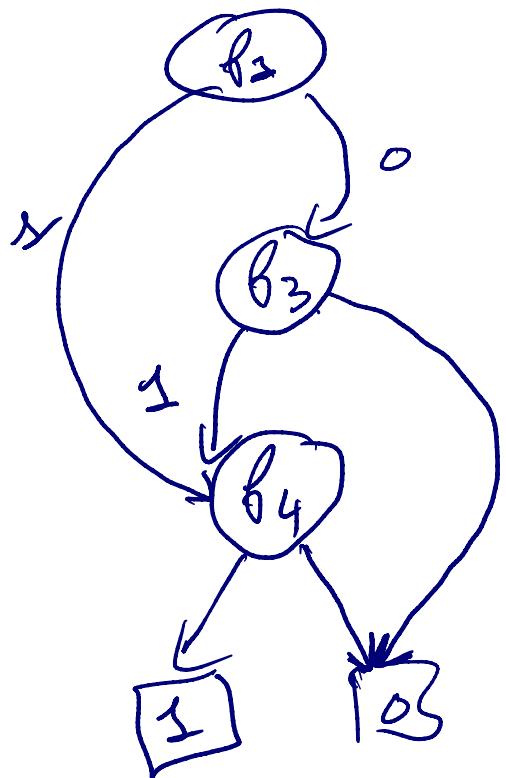
Something can be improved, can you guess it ?

Something can be improved, can you guess it ?



Binary Decision Diagram as an Alternative Model

Binary Decision Diagram as an Alternative Model



Binary Decision Diagrams (BDD)

- A BDD is a rooted, directed, acyclic graph that has two types of vertices (terminal and non-terminal).

Binary Decision Diagrams (BDD)

- A BDD is a rooted, directed, acyclic graph that has two types of vertices (terminal and non-terminal).
- A terminal vertex is associated to a binary value 0 or 1

Binary Decision Diagrams (BDD)

- A BDD is a rooted, directed, acyclic graph that has two types of vertices (terminal and non-terminal).
- A terminal vertex is associated to a binary value 0 or 1
- A non-terminal vertex is associated to a binary feature and has two children according to the value of the feature

Binary Decision Diagrams (BDD)

- A BDD is a rooted, directed, acyclic graph that has two types of vertices (terminal and non-terminal).
- A terminal vertex is associated to a binary value 0 or 1
- A non-terminal vertex is associated to a binary feature and has two children according to the value of the feature
- BDDs are often assumed to be ordered and reduced:
 - Ordered: the graph can be structured into levels where only one feature occurs on each level
 - Reduced: Children are different

Binary Decision Diagrams (BDD)

- A BDD is a rooted, directed, acyclic graph that has two types of vertices (terminal and non-terminal).
- A terminal vertex is associated to a binary value 0 or 1
- A non-terminal vertex is associated to a binary feature and has two children according to the value of the feature
- BDDs are often assumed to be ordered and reduced:
 - Ordered: the graph can be structured into levels where only one feature occurs on each level
 - Reduced: Children are different
- Advantage:

Binary Decision Diagrams (BDD)

- A BDD is a rooted, directed, acyclic graph that has two types of vertices (terminal and non-terminal).
- A terminal vertex is associated to a binary value 0 or 1
- A non-terminal vertex is associated to a binary feature and has two children according to the value of the feature
- BDDs are often assumed to be ordered and reduced:
 - Ordered: the graph can be structured into levels where only one feature occurs on each level
 - Reduced: Children are different
- Advantage: It avoids redundancy (of subtrees)

Binary Decision Diagrams (BDD)

- A BDD is a rooted, directed, acyclic graph that has two types of vertices (terminal and non-terminal).
- A terminal vertex is associated to a binary value 0 or 1
- A non-terminal vertex is associated to a binary feature and has two children according to the value of the feature
- BDDs are often assumed to be ordered and reduced:
 - Ordered: the graph can be structured into levels where only one feature occurs on each level
 - Reduced: Children are different
- Advantage: It avoids redundancy (of subtrees)
- Shortcoming:

Binary Decision Diagrams (BDD)

- A BDD is a rooted, directed, acyclic graph that has two types of vertices (terminal and non-terminal).
- A terminal vertex is associated to a binary value 0 or 1
- A non-terminal vertex is associated to a binary feature and has two children according to the value of the feature
- BDDs are often assumed to be ordered and reduced:
 - Ordered: the graph can be structured into levels where only one feature occurs on each level
 - Reduced: Children are different
- Advantage: It avoids redundancy (of subtrees)
- Shortcoming: it uses a limited number of features. With a height h , a BDD uses exactly h features whereas a DT can use up to 2^h features

A Greedy Approach To Learn Binary Decision Diagrams

A Greedy Approach To Learn Binary Decision Diagrams

- A conversion of a decision tree does not work in general ? Why ?

A Greedy Approach To Learn Binary Decision Diagrams

- A conversion of a decision tree does not work in general ? Why ? because one might encounter cycles

A Greedy Approach To Learn Binary Decision Diagrams

- A conversion of a decision tree does not work in general ? Why ? because one might encounter cycles
- Any solution ?

A Greedy Approach To Learn Binary Decision Diagrams

- A conversion of a decision tree does not work in general ? Why ? because one might encounter cycles
- Any solution ?
- The idea is to build an “*oblivious decision tree (ODT)*” then convert it into a BDD by merging equivalent subtrees

A Greedy Approach To Learn Binary Decision Diagrams

- A conversion of a decision tree does not work in general ? Why ? because one might encounter cycles
- Any solution ?
- The idea is to build an “*oblivious decision tree (ODT)*” then convert it into a BDD by merging equivalent subtrees
- An ODT is a tree where only one feature occurs on each level

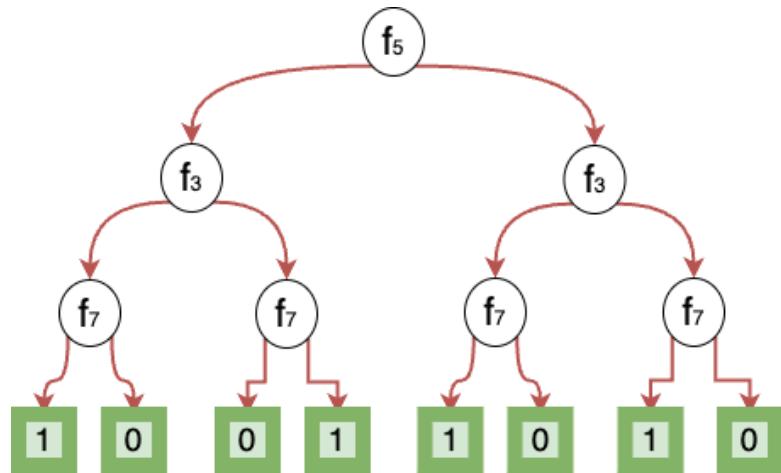
A Greedy Approach To Learn Binary Decision Diagrams

- A conversion of a decision tree does not work in general ? Why ? because one might encounter cycles
- Any solution ?
- The idea is to build an “*oblivious decision tree (ODT)*” then convert it into a BDD by merging equivalent subtrees
- An ODT is a tree where only one feature occurs on each level
- It is a very restricted case of decision trees

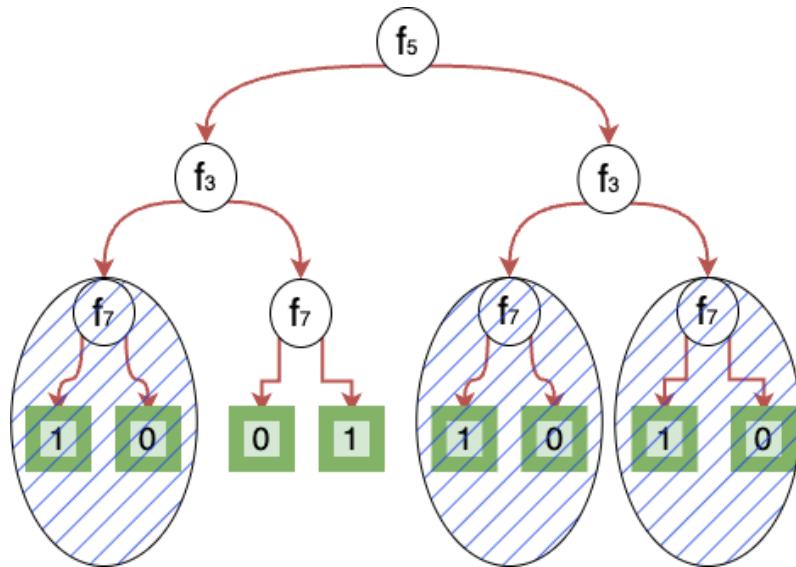
A Greedy Approach To Learn Binary Decision Diagrams

- A conversion of a decision tree does not work in general ? Why ? because one might encounter cycles
- Any solution ?
- The idea is to build an “*oblivious decision tree (ODT)*” then convert it into a BDD by merging equivalent subtrees
- An ODT is a tree where only one feature occurs on each level
- It is a very restricted case of decision trees
- Given an ODT, the correspondent BDD can be constructed by merging equivalent subtrees

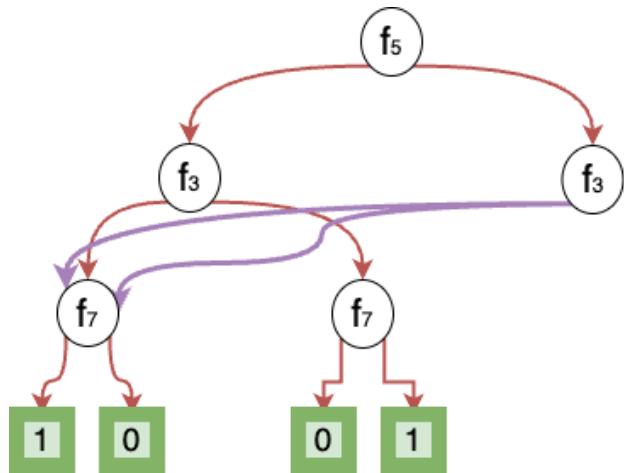
A Greedy Approach To Learn Binary Decision Diagrams



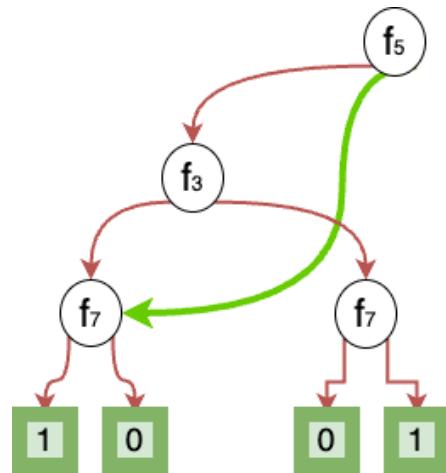
A Greedy Approach To Learn Binary Decision Diagrams



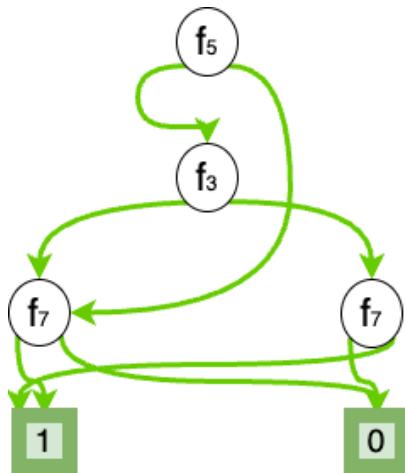
A Greedy Approach To Learn Binary Decision Diagrams



A Greedy Approach To Learn Binary Decision Diagrams



A Greedy Approach To Learn Binary Decision Diagrams



A Greedy Approach To Learn Binary Decision Diagrams

A Greedy Approach To Learn Binary Decision Diagrams

- How to build an ODT ?

A Greedy Approach To Learn Binary Decision Diagrams

- How to build an ODT ?
- Look for a “good” sequence of features (according to a heuristic) then fill the tree (i.e., the leaf nodes)

A Greedy Approach To Learn Binary Decision Diagrams

- How to build an ODT ?
- Look for a “good” sequence of features (according to a heuristic) then fill the tree (i.e., the leaf nodes)
- Information gain (IG) is not well suited

A Greedy Approach To Learn Binary Decision Diagrams

- How to build an ODT ?
- Look for a “good” sequence of features (according to a heuristic) then fill the tree (i.e., the leaf nodes)
- Information gain (IG) is not well suited
- **Mutual Information (MI)** is often used

A Greedy Approach To Learn Binary Decision Diagrams

- How to build an ODT ?
- Look for a “good” sequence of features (according to a heuristic) then fill the tree (i.e., the leaf nodes)
- Information gain (IG) is not well suited
- **Mutual Information (MI)** is often used
- Similarly to IG, MI is defined in the context of information theory

A Greedy Approach To Learn Binary Decision Diagrams

- How to build an ODT ?
- Look for a “good” sequence of features (according to a heuristic) then fill the tree (i.e., the leaf nodes)
- Information gain (IG) is not well suited
- **Mutual Information (MI)** is often used
- Similarly to IG, MI is defined in the context of information theory
- The MI between two random variables X and Y captures how much information is shared between X and Y

$$I(X; Y) = \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p(x, y) \log \left(\frac{p(x, y)}{p(x)p(y)} \right)$$

- $I(X; Y) = 0$ iff X and Y are fully independent

A Greedy Approach To Learn Binary Decision Diagrams

- ① Compute MI between each feature and the prediction output, then pick the k best features (k is the height)
- ② Construct the correspondent ODT
- ③ Remove useless splits
- ④ Merge redundant subtrees
- ⑤ Build the BDD

From ODT to BDD

From ODT to BDD

- How to fill an ODT ?

From ODT to BDD

- How to fill an ODT ?
- Simple! for each path, look at the correspondent dataset, then apply a majority vote

From ODT to BDD

- How to fill an ODT ?
- Simple! for each path, look at the correspondent dataset, then apply a majority vote
- There is an issue!

From ODT to BDD

- How to fill an ODT ?
- Simple! for each path, look at the correspondent dataset, then apply a majority vote
- There is an issue! Some paths lead to empty datasets! How to cope with that ?

From ODT to BDD

- How to fill an ODT ?
- Simple! for each path, look at the correspondent dataset, then apply a majority vote
- There is an issue! Some paths lead to empty datasets! How to cope with that ?
- We can use the parent's majority class

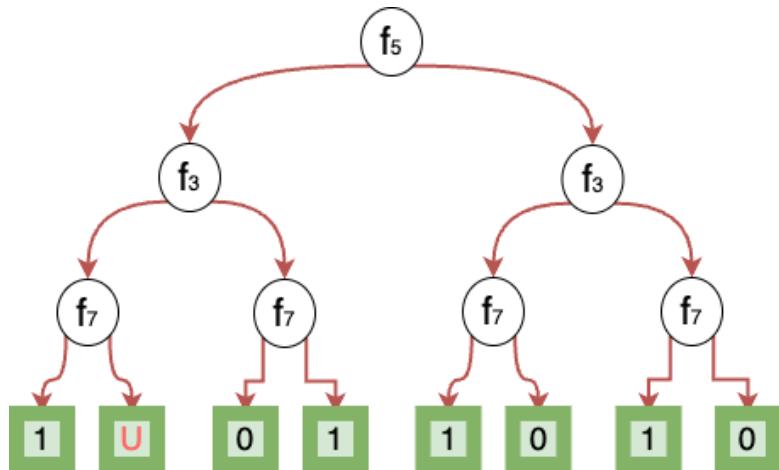
From ODT to BDD

- How to fill an ODT ?
- Simple! for each path, look at the correspondent dataset, then apply a majority vote
- There is an issue! Some paths lead to empty datasets! How to cope with that ?
- We can use the parent's majority class
- We can fill the unknowns in a way that improves the BDD's compactness

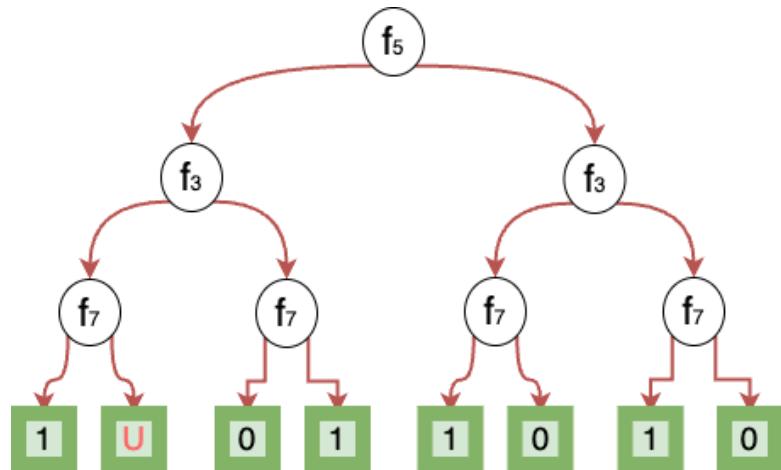
From ODT to BDD

- How to fill an ODT ?
- Simple! for each path, look at the correspondent dataset, then apply a majority vote
- There is an issue! Some paths lead to empty datasets! How to cope with that ?
- We can use the parent's majority class
- We can fill the unknowns in a way that improves the BDD's compactness
- We can fill the unknowns in a way that favour useless splits
- Once the ODT is constructed, build the correspondent BDD by merging subtrees in a bottom up way

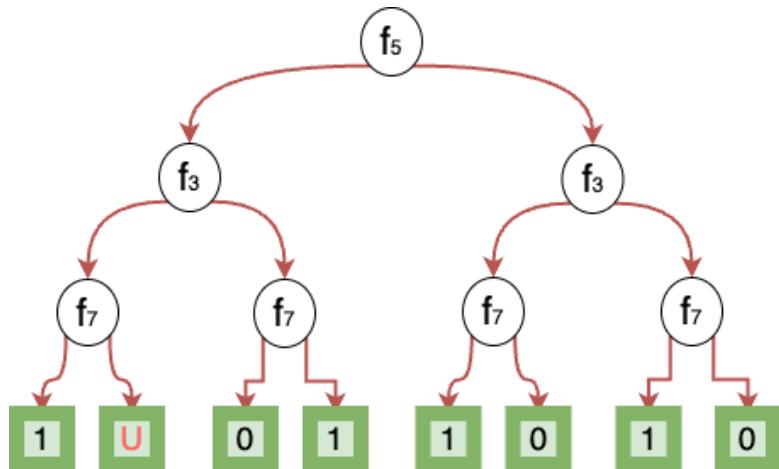
ODT with unknown leaves



ODT with unknown leaves



ODT with unknown leaves



- Different Strategies!
 - One can use the parent's majority class
 - We can fill the unknowns in a way that favour useless splits
 - We can fill the unknowns in a way that favour redundant sub-trees

Decision Rules & Decision Sets

Decision Rules & Decision Sets

- Prediction models based on If-Condition-Then-Prediction rules

Decision Rules & Decision Sets

- Prediction models based on If-Condition-Then-Prediction rules
- **Decision sets:** no specific order is given between the rules. Ties are broken by majority votes

Decision Rules & Decision Sets

- Prediction models based on If-Condition-Then-Prediction rules
- **Decision sets:** no specific order is given between the rules. Ties are broken by majority votes
- **Decision rules:** rules are ordered by priority

The COMPASS Example

Sex	Age	Priors	Juvenile Felonies	Juvenile Crimes	Ethnicity
Male	15	1	0	1	Caucasian
Male	15	1	0	1	African-American
Female	33	1	0	1	African-American
Female	27	0	1	0	Caucasian
Male	41	0	1	0	Caucasian
...

The problem is to predict recidivism. That is, the tendency of a convicted criminal to re-offend.

The COMPASS Example

Sex	Age	Priors	Juvenile Felonies	Juvenile Crimes	Ethnicity
Male	15	1	0	1	Caucasian
Male	15	1	0	1	African-American
Female	33	1	0	1	African-American
Female	27	0	1	0	Caucasian
Male	41	0	1	0	Caucasian
...

The problem is to predict recidivism. That is, the tendency of a convicted criminal to re-offend.

- Data : <https://www.kaggle.com/danofer/compass>
- FairCORELS: An open source tool to learn fair rule lists
<https://github.com/ferryjul/fairCORELS>

Example

Rule List found by FairCORELS on the COMPASS Sataset

Example

Rule List found by FairCORELS on the COMPASS Sataset

```
if [priors:>3] then [recidivism]
else if [age:21-22 && gender:Male] then [recidivism]
else if [age:18-20] then [recidivism]
else if [age:23-25 && priors:2-3] then [recidivism]
else [no recidivism]
```

Rule list 5. Example of an unconstrained rule list found by FairCORELS on COMPAS dataset, with Accuracy = 0.681, UNF_{EODds} = 0.217 and UNF_{CUAE} = 0.046

Optimal Rule Lists

Optimal Rule Lists

- A prediction rule is defined as an If-Condition-Then-Prediction

Optimal Rule Lists

- A prediction rule is defined as an If-Condition-Then-Prediction
- The "condition" is called antecedent
- A **rule list** model is an ordered list of m rules. Rule i is checked only when all previous rules do not apply
- If all rules do not apply, a majority vote prediction is used
- We want to find an algorithm that builds an optimal rule list for binary classification using a given set of pre-mined antecedents

Optimal Rule Lists

- A prediction rule is defined as an If-Condition-Then-Prediction
- The "condition" is called antecedent
- A **rule list** model is an ordered list of m rules. Rule i is checked only when all previous rules do not apply
- If all rules do not apply, a majority vote prediction is used
- We want to find an algorithm that builds an optimal rule list for binary classification using a given set of pre-mined antecedents
- The set of pre-mined antecedents is denoted by $A = \{a_1, \dots, a_l\}$

Optimal Rule Lists

- A prediction rule is defined as an If-Condition-Then-Prediction
- The "condition" is called antecedent
- A **rule list** model is an ordered list of m rules. Rule i is checked only when all previous rules do not apply
- If all rules do not apply, a majority vote prediction is used
- We want to find an algorithm that builds an optimal rule list for binary classification using a given set of pre-mined antecedents
- The set of pre-mined antecedents is denoted by $A = \{a_1, \dots, a_l\}$

Example

```
if [priors:>3] then [recidivism]
else if [age:21-22 && gender:Male] then [recidivism]
else if [age:18-20] then [recidivism]
else if [age:23-25 && priors:2-3] then [recidivism]
else [no recidivism]
```

Rule list 5. Example of an unconstrained rule list found by FairCORELS on COMPAS dataset, with Accuracy = 0.681, $\text{JINF}_{\text{Odds}} = 0.217$ and
(Toulouse)

A Greedy Algorithm to learn Rule Lists

- Find a greedy algorithm to build a rule list with height H given a set of antecedents

A Greedy Algorithm to learn Rule Lists

- Find a greedy algorithm to build a rule list with height H given a set of antecedents
 - At each iteration i , evaluate the error as if antecedent a_j is used on the current dataset (for each potential antecedent a_j)
 - Pick the antecedent with the minimum error
 - Call the algorithm on the new dataset with height $H - 1$ without the chosen antecedent
 - Once the limit is reached, use a majority vote

A Branch and Bound Algorithm for Optimal Rule Lists

A Branch and Bound Algorithm for Optimal Rule Lists

Algorithm 21 OptimalRuleList

Require: $E = \{e_1, \dots, e_n\}$

Allowed length H

Current rule list R

A set of antecedents A

Result: $(RuleList, Error)$

A Branch and Bound Algorithm for Optimal Rule Lists

Algorithm 22 OptimalRuleList

Require: $\mathbb{E} = \{e_1, \dots, e_n\}$

Allowed length H

Current rule list R

A set of antecedents A

Result: $(RuleList, Error)$

$BestError \leftarrow |\mathbb{E}|$

$Best \leftarrow \emptyset$

A Branch and Bound Algorithm for Optimal Rule Lists

Algorithm 23 OptimalRuleList

Require: $\mathbb{E} = \{e_1, \dots, e_n\}$

Allowed length H

Current rule list R

A set of antecedents A

Result: $(RuleList, Error)$

$BestError \leftarrow |\mathbb{E}|$

$Best \leftarrow \emptyset$

if $Length < H$ **then**

A Branch and Bound Algorithm for Optimal Rule Lists

Algorithm 24 OptimalRuleList

Require: $\mathbb{E} = \{e_1, \dots, e_n\}$

Allowed length H

Current rule list R

A set of antecedents A

Result: $(RuleList, Error)$

$BestError \leftarrow |\mathbb{E}|$

$Best \leftarrow \emptyset$

if $Length < H$ **then**

for $a \in A$ **do**

A Branch and Bound Algorithm for Optimal Rule Lists

Algorithm 25 OptimalRuleList

Require: $\mathbb{E} = \{e_1, \dots, e_n\}$

Allowed length H

Current rule list R

A set of antecedents A

Result: $(RuleList, Error)$

$BestError \leftarrow |\mathbb{E}|$

$Best \leftarrow \emptyset$

if $Length < H$ **then**

for $a \in A$ **do**

$Prediction \leftarrow$ Majority class if a is applied

$ErrorP \leftarrow Error(a \implies Prediction)$ on \mathbb{E}

$E_a \leftarrow$ The dataset after applying a

A Branch and Bound Algorithm for Optimal Rule Lists

Algorithm 26 OptimalRuleList

```

Require:  $\mathbb{E} = \{e_1, \dots, e_n\}$ 
    Allowed length  $H$ 
    Current rule list  $R$ 
    A set of antecedents  $A$ 
Result:  $(RuleList, Error)$ 
 $BestError \leftarrow |\mathbb{E}|$ 
 $Best \leftarrow \emptyset$ 
if  $Length < H$  then
    for  $a \in A$  do
         $Prediction \leftarrow$  Majority class if  $a$  is applied
         $ErrorP \leftarrow Error(a \implies Prediction)$  on  $\mathbb{E}$ 
         $E_a \leftarrow$  The dataset after applying  $a$ 
        if  $ErrorP < BestError$  then
             $(RL, error) \leftarrow OptimalRuleList(E_a, A \setminus \{a\}, H - 1)$ 

```

A Branch and Bound Algorithm for Optimal Rule Lists

Algorithm 27 OptimalRuleList

```

Require:  $\mathbb{E} = \{e_1, \dots, e_n\}$ 
    Allowed length  $H$ 
    Current rule list  $R$ 
    A set of antecedents  $A$ 
Result:  $(RuleList, Error)$ 
 $BestError \leftarrow |\mathbb{E}|$ 
 $Best \leftarrow \emptyset$ 
if  $Length < H$  then
    for  $a \in A$  do
         $Prediction \leftarrow$  Majority class if  $a$  is applied
         $ErrorP \leftarrow Error(a \implies Prediction)$  on  $\mathbb{E}$ 
         $E_a \leftarrow$  The dataset after applying  $a$ 
        if  $ErrorP < BestError$  then
             $(RL, error) \leftarrow OptimalRuleList(E_a, A \setminus \{a\}, H - 1)$ 
            if  $error + ErrorP < BestError$  then
                 $BestError \leftarrow error + ErrorP$ 
                 $Best \leftarrow (R + RL)$ 
            end if
        end if

```

A Branch and Bound Algorithm for Optimal Rule Lists

Algorithm 28 OptimalRuleList

```

Require:  $\mathbb{E} = \{e_1, \dots, e_n\}$ 
    Allowed length  $H$ 
    Current rule list  $R$ 
    A set of antecedents  $A$ 
Result:  $(RuleList, Error)$ 
 $BestError \leftarrow |\mathbb{E}|$ 
 $Best \leftarrow \emptyset$ 
if  $Length < H$  then
    for  $a \in A$  do
         $Prediction \leftarrow$  Majority class if  $a$  is applied
         $ErrorP \leftarrow Error(a \implies Prediction)$  on  $\mathbb{E}$ 
         $E_a \leftarrow$  The dataset after applying  $a$ 
        if  $ErrorP < BestError$  then
             $(RL, error) \leftarrow OptimalRuleList(E_a, A \setminus \{a\}, H - 1)$ 
            if  $error + ErrorP < BestError$  then
                 $BestError \leftarrow error + ErrorP$ 
                 $Best \leftarrow (R + RL)$ 
            end if
        end if
    end for
else

```

A Branch and Bound Algorithm for Optimal Rule Lists

Algorithm 29 OptimalRuleList

```

Require:  $\mathbb{E} = \{e_1, \dots, e_n\}$ 
    Allowed length  $H$ 
    Current rule list  $R$ 
    A set of antecedents  $A$ 
Result:  $(RuleList, Error)$ 
 $BestError \leftarrow |\mathbb{E}|$ 
 $Best \leftarrow \emptyset$ 
if  $Length < H$  then
    for  $a \in A$  do
         $Prediction \leftarrow$  Majority class if  $a$  is applied
         $ErrorP \leftarrow Error(a \implies Prediction)$  on  $\mathbb{E}$ 
         $E_a \leftarrow$  The dataset after applying  $a$ 
        if  $ErrorP < BestError$  then
             $(RL, error) \leftarrow OptimalRuleList(E_a, A \setminus \{a\}, H - 1)$ 
            if  $error + ErrorP < BestError$  then
                 $BestError \leftarrow error + ErrorP$ 
                 $Best \leftarrow (R + RL)$ 
            end if
        end if
    end for
else
     $Best \leftarrow (R, Else)$ 
     $BestError \leftarrow Error(R) + Error$  of majority prediction on  $\mathbb{E}$  ;
end if
return  $(Best, BestError)$ 

```

- How to Find Lower Bounds of the objective function at each node ?
 - By considering a best hypothetical scenario at each node, one can compute its objective function and use its value (that we denote by *bound*) as a bound for the rest of the search tree
 - If *bound* is worse than the best objective function found, one can safely prune the current note and not exploring its children

A Branch and Bound Algorithm for Optimal Rule Lists

Algorithm 30 OptimalRuleList

Require: $E = \{e_1, \dots, e_n\}$

Allowed length H

Current rule list R

A set of antecedents A

Result: $(RuleList, Error)$

A Branch and Bound Algorithm for Optimal Rule Lists

Algorithm 31 OptimalRuleList

Require: $\mathbb{E} = \{e_1, \dots, e_n\}$

Allowed length H

Current rule list R

A set of antecedents A

Result: $(RuleList, Error)$

$BestError \leftarrow |\mathbb{E}|$

$Best \leftarrow \emptyset$

A Branch and Bound Algorithm for Optimal Rule Lists

Algorithm 32 OptimalRuleList

Require: $\mathbb{E} = \{e_1, \dots, e_n\}$

Allowed length H

Current rule list R

A set of antecedents A

Result: $(RuleList, Error)$

$BestError \leftarrow |\mathbb{E}|$

$Best \leftarrow \emptyset$

if $Length < H$ **then**

A Branch and Bound Algorithm for Optimal Rule Lists

Algorithm 33 OptimalRuleList

Require: $\mathbb{E} = \{e_1, \dots, e_n\}$

Allowed length H

Current rule list R

A set of antecedents A

Result: $(RuleList, Error)$

$BestError \leftarrow |\mathbb{E}|$

$Best \leftarrow \emptyset$

if $Length < H$ **then**

for $a \in A$ **do**

A Branch and Bound Algorithm for Optimal Rule Lists

Algorithm 34 OptimalRuleList

Require: $\mathbb{E} = \{e_1, \dots, e_n\}$

Allowed length H

Current rule list R

A set of antecedents A

Result: $(RuleList, Error)$

$BestError \leftarrow |\mathbb{E}|$

$Best \leftarrow \emptyset$

if $Length < H$ **then**

for $a \in A$ **do**

$Prediction \leftarrow$ Majority class if a is applied

$ErrorP \leftarrow Error(a \implies Prediction)$ on \mathbb{E}

$E_a \leftarrow$ The dataset after applying a

A Branch and Bound Algorithm for Optimal Rule Lists

Algorithm 35 OptimalRuleList

Require: $\mathbb{E} = \{e_1, \dots, e_n\}$

Allowed length H

Current rule list R

A set of antecedents A

Result: $(RuleList, Error)$

$BestError \leftarrow |\mathbb{E}|$

$Best \leftarrow \emptyset$

if $Length < H$ **then**

for $a \in A$ **do**

$Prediction \leftarrow$ Majority class if a is applied

$ErrorP \leftarrow Error(a \implies Prediction)$ on \mathbb{E}

$E_a \leftarrow$ The dataset after applying a

if $ErrorP + ErrorLowerBound(E_a, A \setminus \{a\}) < BestError$ **then**

$(RL, error) \leftarrow OptimalRuleList(E_a, A \setminus \{a\}, H - 1)$

A Branch and Bound Algorithm for Optimal Rule Lists

Algorithm 36 OptimalRuleList

Require: $\mathbb{E} = \{e_1, \dots, e_n\}$

Allowed length H

Current rule list R

A set of antecedents A

Result: $(RuleList, Error)$

$BestError \leftarrow |\mathbb{E}|$

$Best \leftarrow \emptyset$

if $Length < H$ **then**

for $a \in A$ **do**

$Prediction \leftarrow$ Majority class if a is applied

$ErrorP \leftarrow Error(a \implies Prediction)$ on \mathbb{E}

$E_a \leftarrow$ The dataset after applying a

if $ErrorP + ErrorLowerBound(E_a, A \setminus \{a\}) < BestError$ **then**

$(RL, error) \leftarrow OptimalRuleList(E_a, A \setminus \{a\}, H - 1)$

if $error + ErrorP < BestError$ **then**

$BestError \leftarrow error + ErrorP$

$Best \leftarrow (R + RL)$

end if

A Branch and Bound Algorithm for Optimal Rule Lists

Algorithm 37 OptimalRuleList

Require: $\mathbb{E} = \{e_1, \dots, e_n\}$

Allowed length H

Current rule list R

A set of antecedents A

Result: $(RuleList, Error)$

$BestError \leftarrow |\mathbb{E}|$

$Best \leftarrow \emptyset$

if $Length < H$ **then**

for $a \in A$ **do**

$Prediction \leftarrow$ Majority class if a is applied

$ErrorP \leftarrow Error(a \implies Prediction)$ on \mathbb{E}

$E_a \leftarrow$ The dataset after applying a

if $ErrorP + ErrorLowerBound(E_a, A \setminus \{a\}) < BestError$ **then**

$(RL, error) \leftarrow OptimalRuleList(E_a, A \setminus \{a\}, H - 1)$

if $error + ErrorP < BestError$ **then**

$BestError \leftarrow error + ErrorP$

$Best \leftarrow (R + RL)$

end if

end if

end for

else

A Branch and Bound Algorithm for Optimal Rule Lists

Algorithm 38 OptimalRuleList

```

Require:  $\mathbb{E} = \{e_1, \dots, e_n\}$ 
    Allowed length  $H$ 
    Current rule list  $R$ 
    A set of antecedents  $A$ 
Result:  $(RuleList, Error)$ 
 $BestError \leftarrow |\mathbb{E}|$ 
 $Best \leftarrow \emptyset$ 
if  $Length < H$  then
    for  $a \in A$  do
         $Prediction \leftarrow$  Majority class if  $a$  is applied
         $ErrorP \leftarrow Error(a \implies Prediction)$  on  $\mathbb{E}$ 
         $E_a \leftarrow$  The dataset after applying  $a$ 
        if  $ErrorP + ErrorLowerBound(E_a, A \setminus \{a\}) < BestError$  then
             $(RL, error) \leftarrow OptimalRuleList(E_a, A \setminus \{a\}, H - 1)$ 
            if  $error + ErrorP < BestError$  then
                 $BestError \leftarrow error + ErrorP$ 
                 $Best \leftarrow (R + RL)$ 
            end if
        end if
    end for
else
     $Best \leftarrow (R, Else)$ 
     $BestError \leftarrow Error(R) + Error$  of majority prediction on  $\mathbb{E}$  ;
end if
return  $(Best, BestError)$ 

```

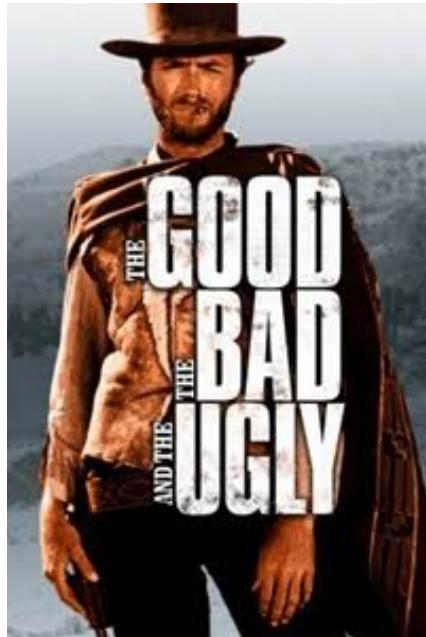
• Symmetry Breaking ?

- If two nodes share the same set of antecedents then necessarily they share the best error value. Therefore, one can use caching to save the objective value for each set of antecedents that are already explored. By doing so, each time a sequence of antecedents is about to be explored, one can check whether its correspondent set is already explored. If it is the case, then the search space can be pruned.

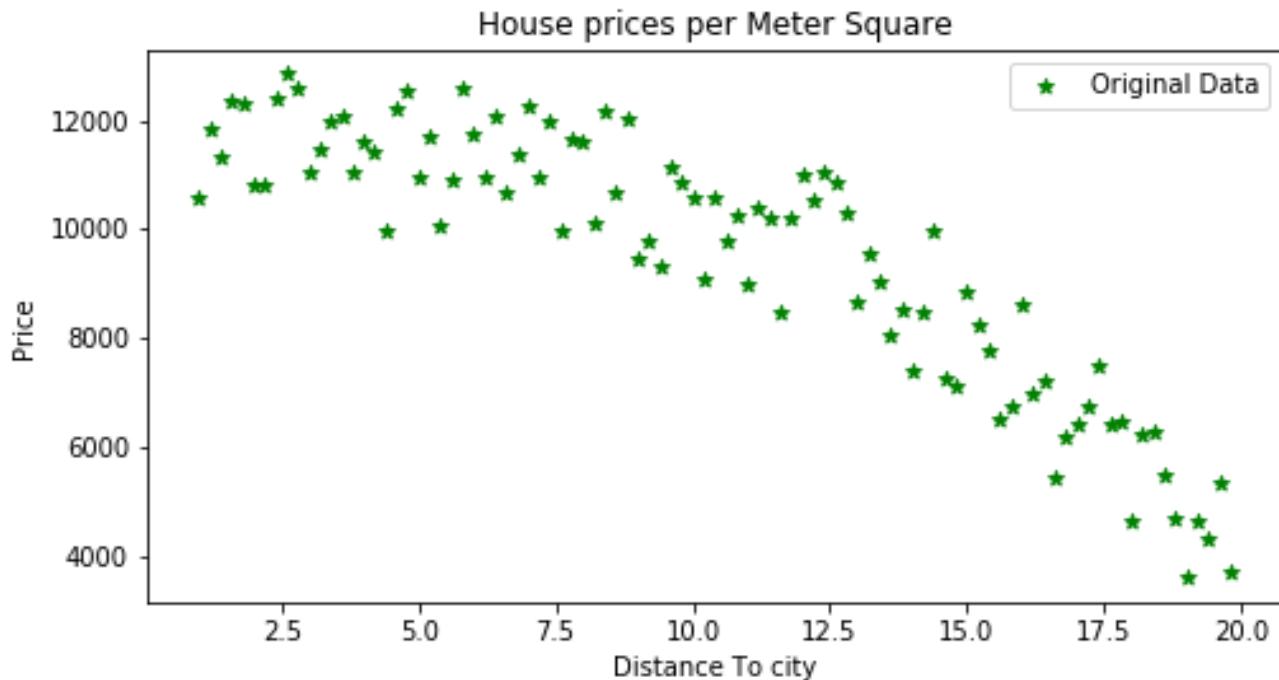
Overfitting, Underfitting, and Goodfitting

Model Evaluation

Overfitting, Underfitting, and Goodfitting

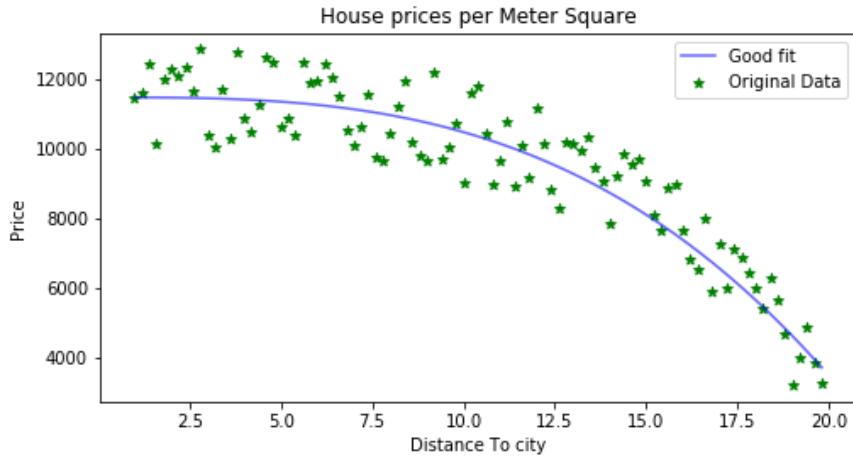


The Housing Prices Example



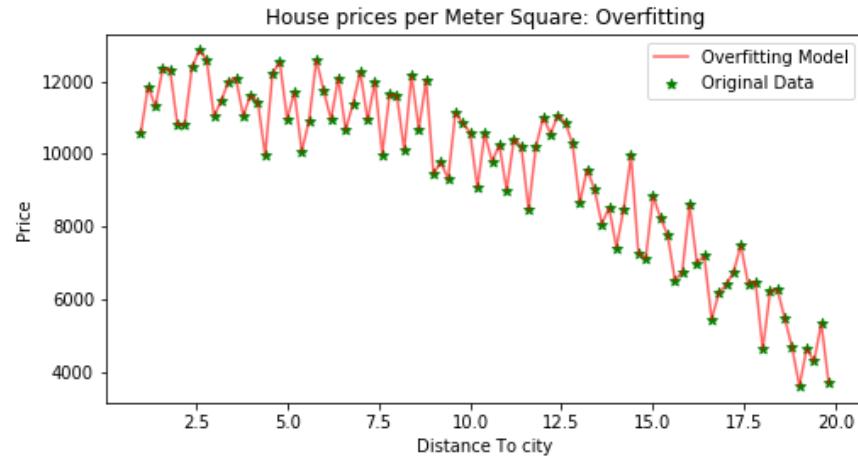
This data includes some **noise**. That is, points that are not correctly collected (which is often the case in real applications)

The Housing Prices Example: The Good

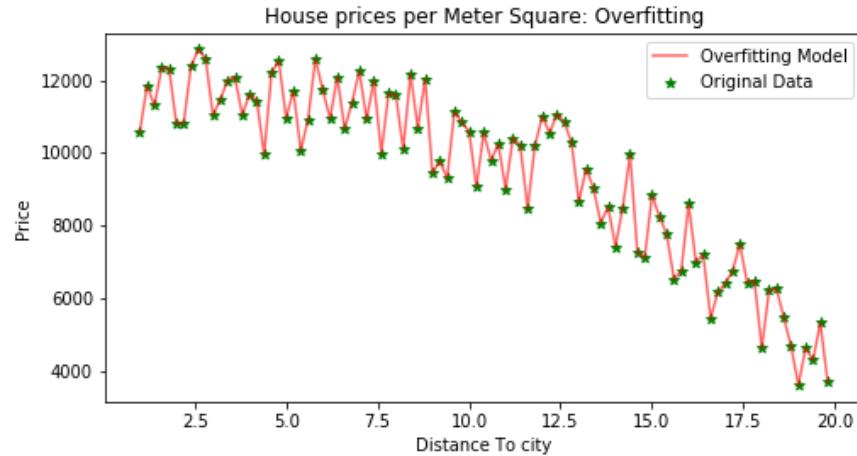


We can make an analogy to a smart student who has a good understanding of a lecture

The Housing Prices Example: The Bad (Overfitting)

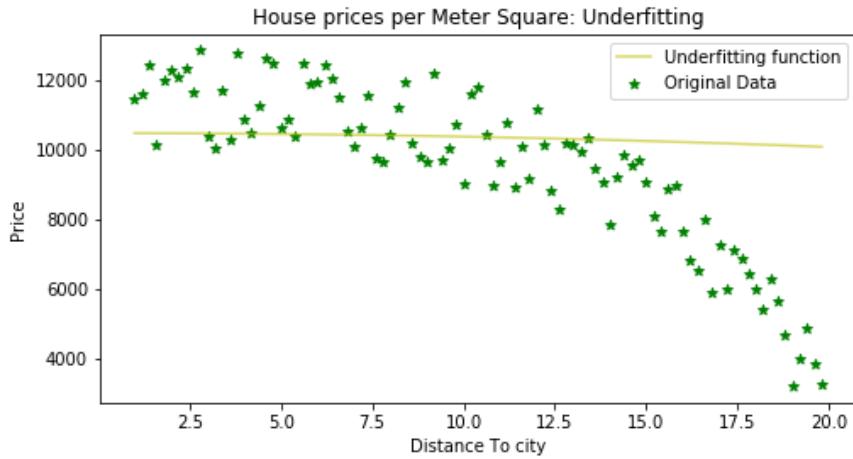


The Housing Prices Example: The Bad (Overfitting)



We can make an analogy to the student who "learns" the lecture mechanically without a real understanding.

The Housing Prices Example: The Ugly (Underfitting)

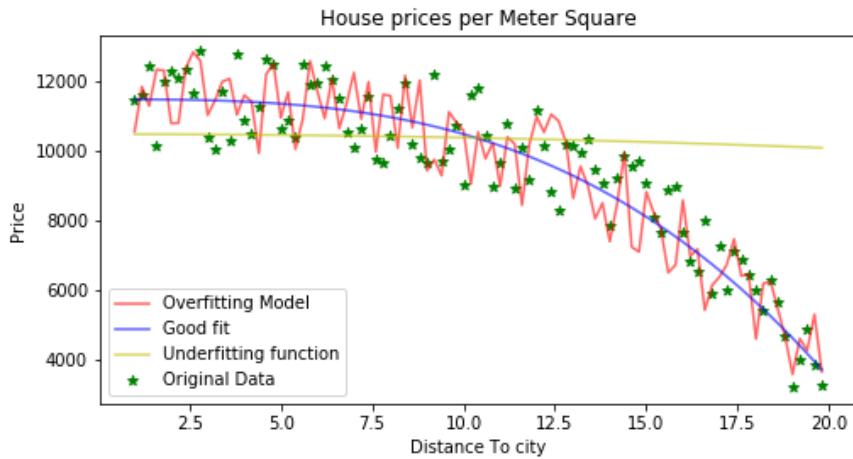


The Housing Prices Example: The Ugly (Underfitting)



We can make an analogy to a lazy student who barely remember the lecture without any understanding

The Housing Prices Example: All Together



Overfitting, Underfitting, and a Good Fit

Overfitting, Underfitting, and a Good Fit

- Overfitting happens when the model tries to squeeze everything in including noise without an "intuitive understanding of the data"

Overfitting, Underfitting, and a Good Fit

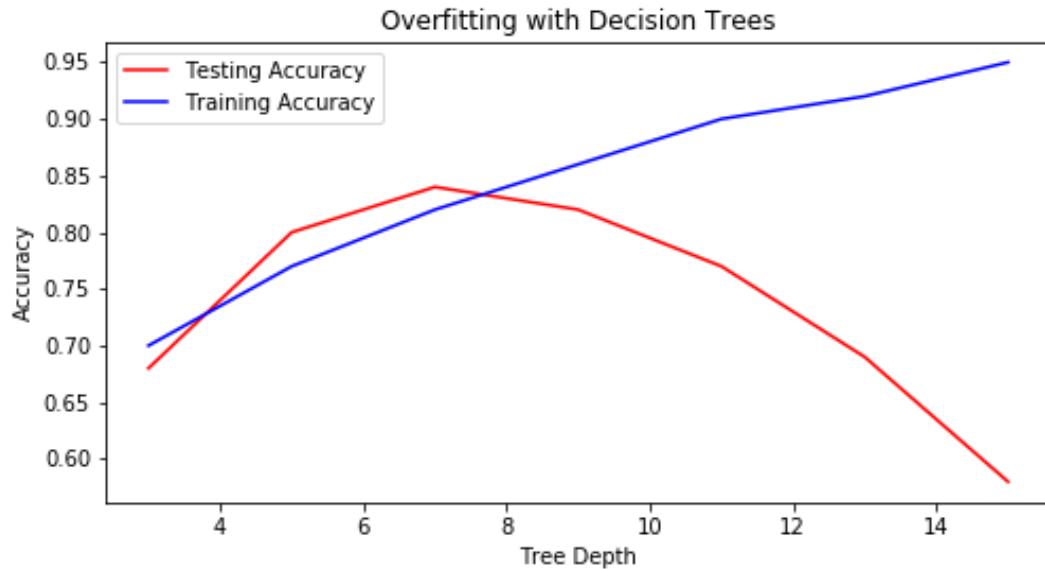
- Overfitting happens when the model tries to squeeze everything in including noise without an "intuitive understanding of the data"
- Underfitting happens when the model performs badly on the training and testing data (no real learning).

Overfitting, Underfitting, and a Good Fit

- Overfitting happens when the model tries to squeeze everything in including noise without an "intuitive understanding of the data"
- Underfitting happens when the model performs badly on the training and testing data (no real learning).
- A good fit happens when the model approximates well the true distribution without being disturbed by noise (good generalisation)

Overfitting with Decision Trees as an Example

Overfitting with Decision Trees as an Example



Overfitting with Decision Trees as an Example

- The longer the tree, the better the training accuracy gets, however, this is not necessarily the case for the testing accuracy

Overfitting with Decision Trees as an Example

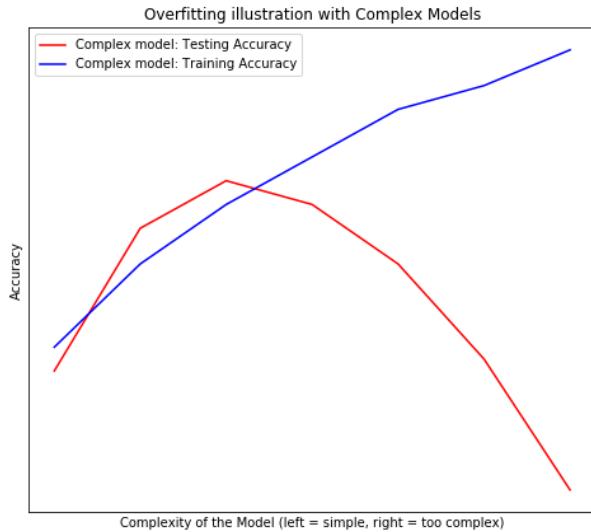
- The longer the tree, the better the training accuracy gets, however, this is not necessarily the case for the testing accuracy
- Testing accuracy increases at the beginning until a certain value (depth = 7), then it decreases afterwards

Overfitting with Decision Trees as an Example

- The longer the tree, the better the training accuracy gets, however, this is not necessarily the case for the testing accuracy
- Testing accuracy increases at the beginning until a certain value (depth = 7), then it decreases afterwards
- This happens because with longer trees, the model can classify correctly more examples in the training set, however, this includes noise.

Overfitting Based on the Complexity of the Model

Overfitting Based on the Complexity of the Model



- When the model is too simple, there is a risque of underfitting
- When the model is too complex, there is a risque of overfitting
- We need a Model that is somehow in between
- ML libraries offer parameters for regulation to avoid overfitting/underfitting

Complexity/Quality Tradeoff

Complexity/Quality Tradeoff

- Training algorithms have resources costs: memory and runtime

Complexity/Quality Tradeoff

- Training algorithms have resources costs: memory and runtime
- For instance, training quadratic functions is much harder (computationally) than training linear functions

Complexity/Quality Tradeoff

- Training algorithms have resources costs: memory and runtime
- For instance, training quadratic functions is much harder (computationally) than training linear functions
- However, may be a quadratic function is a better fit for the data at hand

Complexity/Quality Tradeoff

- Training algorithms have resources costs: memory and runtime
- For instance, training quadratic functions is much harder (computationally) than training linear functions
- However, may be a quadratic function is a better fit for the data at hand
- There is a trade-off between the quality of predictions and the model complexity

Complexity/Quality Tradeoff

- Training algorithms have resources costs: memory and runtime
- For instance, training quadratic functions is much harder (computationally) than training linear functions
- However, may be a quadratic function is a better fit for the data at hand
- There is a trade-off between the quality of predictions and the model complexity
- For example training a tree with depth 5 is much faster than training a tree of depth 9, but in terms of training quality, trees of depth 9 are better. However, trees with depth 9 might overfit

Complexity/Quality Tradeoff

- Training algorithms have resources costs: memory and runtime
- For instance, training quadratic functions is much harder (computationally) than training linear functions
- However, may be a quadratic function is a better fit for the data at hand
- There is a trade-off between the quality of predictions and the model complexity
- For example training a tree with depth 5 is much faster than training a tree of depth 9, but in terms of training quality, trees of depth 9 are better. However, trees with depth 9 might overfit
- Most ML libraries offer the possibility to control the complexity with a regularization parameter

Ockham's Razor Principle

⁵Philosopher https://en.wikipedia.org/wiki/William_of_Ockham

Ockham's Razor Principle

- In the animal extinction example, we have two different trees with the same accuracy

⁵Philosopher https://en.wikipedia.org/wiki/William_of_Ockham

Ockham's Razor Principle

- In the animal extinction example, we have two different trees with the same accuracy
- Which tree to choose?

⁵Philosopher https://en.wikipedia.org/wiki/William_of_Ockham

Ockham's Razor Principle

- In the animal extinction example, we have two different trees with the same accuracy
- Which tree to choose?
- Hard to answer without specific requirements

⁵Philosopher https://en.wikipedia.org/wiki/William_of_Ockham

Ockham's Razor Principle

- In the animal extinction example, we have two different trees with the same accuracy
- Which tree to choose?
- Hard to answer without specific requirements
- Ockham's Razor Principle⁵: pick the simplest!

⁵Philosopher https://en.wikipedia.org/wiki/William_of_Ockham

Ockham's Razor Principle

- In the animal extinction example, we have two different trees with the same accuracy
- Which tree to choose?
- Hard to answer without specific requirements
- Ockham's Razor Principle⁵: pick the simplest!
- Simplicity is also hard to define

⁵Philosopher https://en.wikipedia.org/wiki/William_of_Ockham

Ockham's Razor Principle

- In the animal extinction example, we have two different trees with the same accuracy
- Which tree to choose?
- Hard to answer without specific requirements
- Ockham's Razor Principle⁵: pick the simplest!
- Simplicity is also hard to define
- In decision trees, simplicity could be the depth, the number of features, a combination of both, etc

⁵Philosopher https://en.wikipedia.org/wiki/William_of_Ockham

Ockham's Razor Principle

- In the animal extinction example, we have two different trees with the same accuracy
- Which tree to choose?
- Hard to answer without specific requirements
- Ockham's Razor Principle⁵: pick the simplest!
- Simplicity is also hard to define
- In decision trees, simplicity could be the depth, the number of features, a combination of both, etc
- When using polynomials (as a hypothesis space), lower degrees seem to be simpler

⁵Philosopher https://en.wikipedia.org/wiki/William_of_Ockham

Ockham's Razor Principle

- In the animal extinction example, we have two different trees with the same accuracy
- Which tree to choose?
- Hard to answer without specific requirements
- Ockham's Razor Principle⁵: pick the simplest!
- Simplicity is also hard to define
- In decision trees, simplicity could be the depth, the number of features, a combination of both, etc
- When using polynomials (as a hypothesis space), lower degrees seem to be simpler
- In other cases it is very hard to define simplicity

⁵Philosopher https://en.wikipedia.org/wiki/William_of_Ockham

Complexity/Quality/Overfitting Tradeoff

The bottom line

Complexity/Quality/Overfitting Tradeoff

The bottom line

- There are fine lines between:

Complexity/Quality/Overfitting Tradeoff

The bottom line

- There are fine lines between:
 - overfitting/underfitting

Complexity/Quality/Overfitting Tradeoff

The bottom line

- There are fine lines between:
 - overfitting/underfitting
 - hard/easy training algorithms

Complexity/Quality/Overfitting Tradeoff

The bottom line

- There are fine lines between:
 - overfitting/underfitting
 - hard/easy training algorithms
 - complex/simple models
- Complex models can be computationally hard, however have better flexibility (some parameters can be turned off) and might have better quality

Complexity/Quality/Overfitting Tradeoff

The bottom line

- There are fine lines between:
 - overfitting/underfitting
 - hard/easy training algorithms
 - complex/simple models
- Complex models can be computationally hard, however have better flexibility (some parameters can be turned off) and might have better quality
- Complex models might overfit

Complexity/Quality/Overfitting Tradeoff

The bottom line

- There are fine lines between:
 - overfitting/underfitting
 - hard/easy training algorithms
 - complex/simple models
- Complex models can be computationally hard, however have better flexibility (some parameters can be turned off) and might have better quality
- Complex models might overfit
- Simple models might underfit

Complexity/Quality/Overfitting Tradeoff

The bottom line

- There are fine lines between:
 - overfitting/underfitting
 - hard/easy training algorithms
 - complex/simple models
- Complex models can be computationally hard, however have better flexibility (some parameters can be turned off) and might have better quality
- Complex models might overfit
- Simple models might underfit
- Ideally, we look for a hypothesis that is ‘easy’ to compute and simple enough to be a good fit

Trustworthy AI

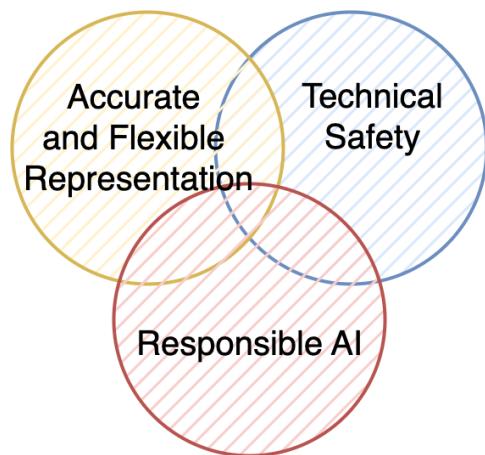


Figure 5: Holistic Framework for Trustworthy AI

- **Technical safety.** An agent has to accomplish precisely what they are expected to do with the backing of formal guarantees.
- **Accurate and flexible representation.**
An agent has to be carefully designed to incorporate rigorous modeling while providing an appropriate margin of flexibility for potential developments.
- **Responsible AI**
Whenever relevant, ethical considerations, societal and environmental concerns have to be thoughtfully evaluated.

- **Technical safety.**

An agent has to accomplish precisely what they are expected to do with the backing of formal guarantees.

- **Technical safety.**

An agent has to accomplish precisely what they are expected to do with the backing of formal guarantees.

- **Accurate and flexible representation.**

An agent has to be carefully designed to incorporate rigorous modeling while providing an appropriate margin of flexibility for potential developments.

- **Technical safety.**

An agent has to accomplish precisely what they are expected to do with the backing of formal guarantees.

- **Accurate and flexible representation.**

An agent has to be carefully designed to incorporate rigorous modeling while providing an appropriate margin of flexibility for potential developments.

- **Responsible AI**

Whenever relevant, ethical considerations, societal and environmental concerns have to be thoughtfully evaluated

Eleven Trustworthy AI Guidelines

- Transparency
- Justice & fairness
- Non-maleficence
- Responsibility
- Privacy
- Beneficence
- Freedom & autonomy
- Trust
- Sustainability
- Dignity
- Solidarity

References



C. Rudin, C. Chen, Z. Chen, H. Huang, L. Semenova, and C. Zhong, “Interpretable machine learning: Fundamental principles and 10 grand challenges,” *CoRR*, vol. abs/2103.11251, 2021.



J. R. Quinlan, *C4.5: Programs for Machine Learning*.
Morgan Kaufmann, 1993.



L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees*.
Wadsworth, 1984.



H. Hu, M. Siala, E. Hebrard, and M. Huguet, “Learning optimal decision trees with maxsat and its integration in adaboost,” in *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020* (C. Bessiere, ed.), pp. 1170–1176, ijcai.org, 2020.



E. Demirovic, A. Lukina, E. Hebrard, J. Chan, J. Bailey, C. Leckie, K. Ramamohanarao, and P. J. Stuckey, “Murtree: Optimal decision trees via dynamic programming and search,” *J. Mach. Learn. Res.*, vol. 23, pp. 26:1–26:47, 2022.