# Explainable Artificial Intelligence

## 02 – Interpretability by design

MSc in Artificial Intelligence

MSc in Computer Engineering

**Marco Lippi**

marco.lippi@unifi.it


UNIVERSITÀ
DEGLI STUDI
FIRENZE

UNIVERSITÀ DEGLI STUDI FIRENZE

# Disclaimer

o

These slides represent just a **summary of the topics** presented during classes and their content **should not** be used in place of handouts, notes, papers, books and software tools that have been used for each lecture and topic

# Table of Contents

# Decision trees

## 1 Rule-based systems

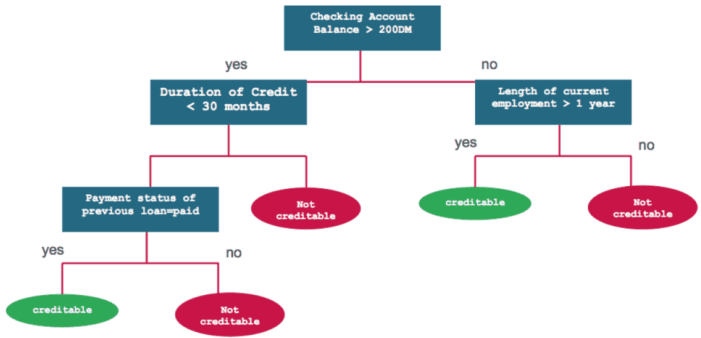
Typically used for **tabular data**, they show a strong interpretability
Classic solutions (e.g., CART, C4.5, etc.) are based on **greedy** algorithms

| | Credit History | Income | Credit Duration | ... | Loan? |
|---|---|---|---|---|---|
| User1 | Good | Low | 24 | ... | Yes |
| User2 | Bad | Low | 48 | ... | No |
| User3 | Bad | High | 96 | ... | Yes |
| User4 | Good | High | 96 | ... | Yes |
| ... | | | | | |

UNIVERSITÀ DEGLI STUDI FIRENZE

# Decision trees

## 1 Rule-based systems



# Decision trees



- Classic (**greedy**) approaches to learn decision trees: CART, C4.5, etc.
- Typically **fail** to deal with **linear** relationships (approximated by splits)
- Lack of **smoothness**: small changes in input produce very different outcomes
- **Unstable** also with respect to changes in the training set
- Highly complex with increasing **depth** (many terminal nodes)

# Rule-based systems



Classic approaches to rule induction produce **sets** and **lists** of rules

- Lists are **ordered**
- Sets are **unordered**

In both cases, two issues must be addressed

- How to handle **overlap** between rules
- What decision to take if **no rule applies**

# Rule-based systems



- **Sequential covering**: incrementally learn rules that cover the entire dataset
- **Bayesian Rule Lists**: recent approach combining rules and Bayesian analysis

Pro: easy to interpret, as expressive as decision trees but often **more compact**
Con: mostly focus on **classification**, and on **categorical** features

# Bayesian Rule Lists



1. Extract **frequent patterns** (e.g., with apriori or fp-growth)
2. Learn a **decision list** from pre-mined patterns, using priors (assumptions)
   - Randomly generate initial list from **prior distribution**
   - **Modify** list by adding/removing/updating rules of the list
   - Choose best list according to **posterior distribution**

The posterior probability is proportional to the product between likelihood and prior

$$P(\boldsymbol{d}|\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{A}, \alpha, \lambda, \eta) \propto P(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{d}, \alpha)P(\boldsymbol{d}|\boldsymbol{A}, \lambda, \eta)$$

# RuleFit algorithm



Combine rule lists with linear models

- Learn some decision tree
- Extract classification rules from tree, to be used as **features**
- Learn a linear model (LASSO) from the set of **features and rules**

LASSO is used to regularize the model in order to obtain a sparse solution (i.e., few rules extracted from the trees are actually used for the classification)

UNIVERSITÀ DEGLI STUDI FIRENZE

# Table of Contents

2 Optimal decision trees

# Towards Optimal Decision Trees



Existing approaches to decision tree learning are **greedy** as they consider at each split which is the most appropriate attribute (e.g., via Gini index or Information Gain Ratio)

Unfortunately, these approaches fail in optimizing the **overall cost function** of the whole decision tree, and are not designed to optimize any particular performance metric

Full decision tree optimization is known to be an **NP-complete** problem

# Generalized Optimal Sparse Decision Trees (GOSDT)

## 2 Optimal decision trees

GOSDT optimizes the following cost function:

$$\min_{f \in \text{Trees}} \frac{1}{n} \sum_{i=1}^{n} \text{Loss}(f, z_i) + C \cdot \text{NumLeaves}(f)$$

$$s.t. \qquad depth(f) \leq D$$

Hyper-parameters $\lambda$ and $D$ control **sparsity** and **computational cost**

Prevent exploration of the whole search space with **dynamic programming with bounds**

# Generalized Optimal Sparse Decision Trees (GOSDT)

## 2 Optimal decision trees

The GOSDT algorithm is based on the following key ideas

- Binary representation of data
- Binary representation of problems
- Priority queue to handle priority of candidate expansion
- Dependency graph to store sub-problems to be solved
- Pruning techniques to evaluate lower and upper bound of problems

# Table of Contents

3 Explainable Boosting Machines

# Generalized additive models

## 3 Explainable Boosting Machines

A generalized additive model (GAM) computes its predictions by exploiting and learning one different function $f_1, \ldots, f_k$ for each single feature $x_1, \ldots, x_k$

$$g(E[y]) = f_1(x_1) + f_2(x_2) + \ldots + f_k(x_k)$$

The approach can be extended in order to account for pairwise interactions, leading to a model that is called GA$^2$M:

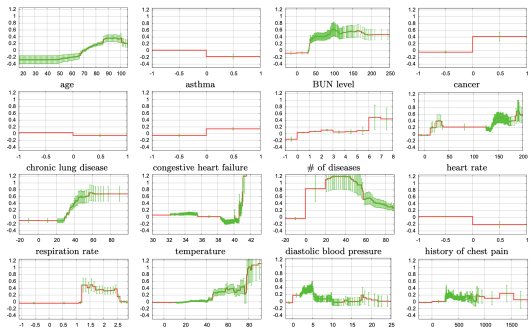
$$g(E[y]) = \sum_{i=1}^{k} f_i(x_i) + \sum_{i=1}^{k} \sum_{j=1}^{k} f_{ij}(x_i, x_j)$$

Note: Explainable Boosting Machines are GA$^2$Ms implemented with boosting trees

# Generalized additive models

## 3 Explainable Boosting Machines

Learned functions can be easily represented via a function chart

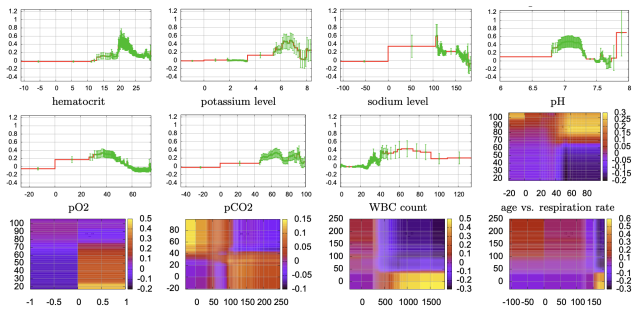



[Source: Caruana et al., 2015]

# Generalized additive models

## 3 Explainable Boosting Machines

In case of GA2Ms, heatmaps are necessary to show dependencies

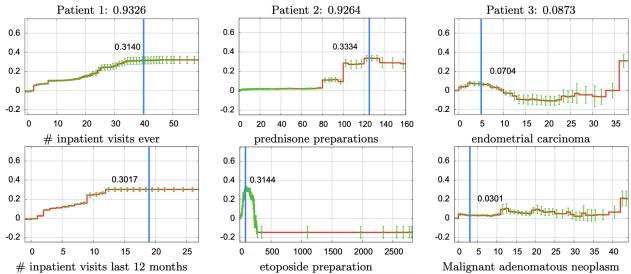



[Source: Caruana et al., 2015]

# Generalized additive models

## 3 Explainable Boosting Machines

The model can provide **local** explanations as well

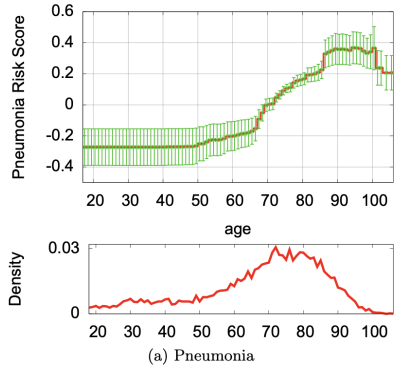



[Source: Caruana et al., 2015]

# Generalized additive models

## 3 Explainable Boosting Machines

Expert knowledge can be used to interpret charts



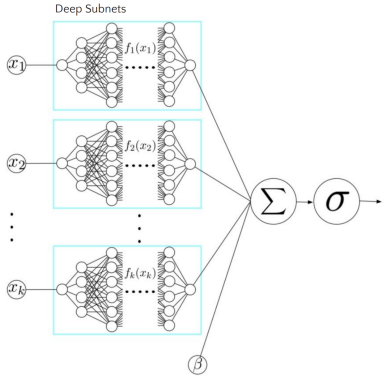

(a) Pneumonia

[Source: Caruana et al., 2015]

# Neural Additive Models

## 3 Explainable Boosting Machines

- Straightforward extension of GAMs
- Separate subnet for each feature
- Contributions added at output layer
- Subnets learned in parallel on GPU




[Source: repository on NAMs]

# Table of Contents

4 Prototypes

# Idea: K-Nearest Neighbors

## 4 Prototypes

Interpretability in KNN

- Explain by referring to most similar examples in training data
- Intuitive for some data types (e.g., images)
- Inherently local and not global
- Hard to interpret with many features (curse of dimensionality)
- Similar to how humans reason sometimes

# Prototypes

## 4 Prototypes

This looks like that... Prototypical Part Networks (ProtoPNet)

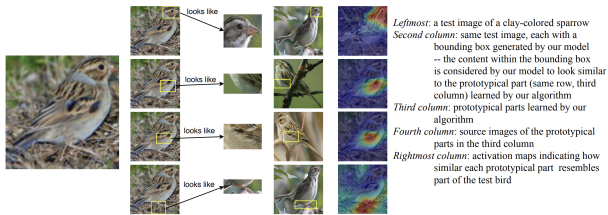



Figure 1: Image of a clay colored sparrow and how parts of it look like some learned prototypical parts of a clay colored sparrow used to classify the bird's species.

[Source: Chen et al., 2019]

# ProtoPNet

## 4 Prototypes

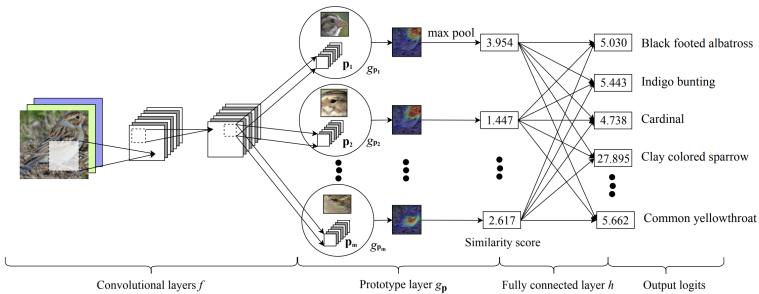



Figure 2: ProtoPNet architecture.

[Source: Chen et al., 2019]

# ProtoPNet

## 4 Prototypes

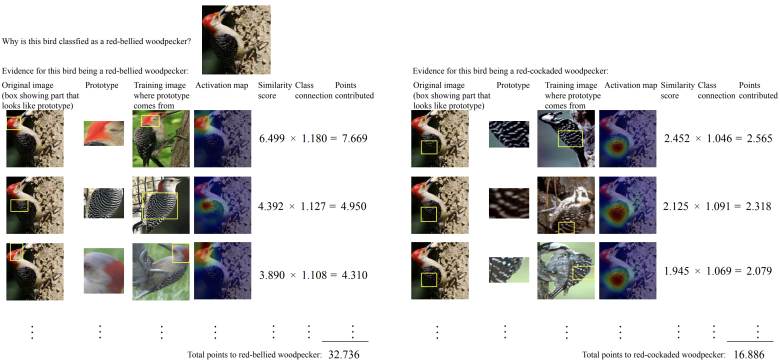



Figure 3: The reasoning process of our network in deciding the species of a bird (top).

[Source: Chen et al., 2019]

# ProtoPNet

## 4 Prototypes

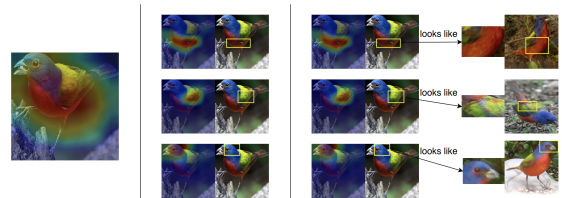



Figure 4: Visual comparison of different types of model interpretability: (a) object-level attention map (e.g., class activation map [56]); (b) part attention (provided by attention-based interpretable models); and (c) part attention with similar prototypical parts (provided by our model).

[Source: Chen et al., 2019]

# ProtoPNet

## 4 Prototypes

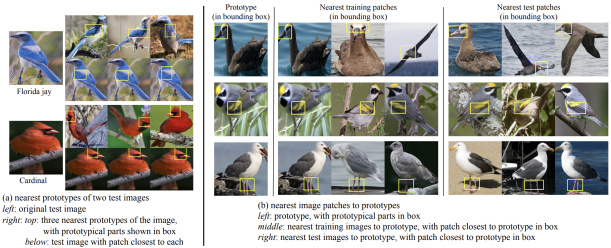



Figure 5: Nearest prototypes to images and nearest images to prototypes. The prototypes are learned from the training set.

[Source: Chen et al., 2019]

# ProtoPNet

## 4 Prototypes

Learning algorithm consists in three steps (to be iterated)

- **Stochastic gradient descent** for all layers except last one (to jointly optimize convolutional layers and prototypes)
- **Prototype projection** to project back the learned prototypes onto the closest patches of the training set (to enhance interpretability)
- **Convex optimization** of last fully-connected layer that performs classification on the basis of the learned (and projected) prototypes

UNIVERSITÀ DEGLI STUDI FIRENZE

# Table of Contents

# Logic programming

## 5 Inductive logic programming

Logic programming is a programming and representation paradigm grounded on **logic**

**Knowledge** is represented in the form of variables, constants, predicates, terms, atoms, literals, and clauses (e.g., `father(X,alice)` where `father` is a predicate, `X` is a variable and `alice` a constant

One of the most popular **programming languages** for logic programming is Prolog

# Inductive logic programming



The goal of inductive logic programming (ILP) is to **learn a logic theory** (i.e., a collection of rules) from a given set of facts

The **semantics** is defined by the concepts of:

- Herbrand universe: the set of all possible ground terms that can be formed by using constants and functions (e.g., `alice`)
- Herbrand base: set of all possible ground atoms that can be formed bypredicates and functions, together with the Herbrand universe (e.g., `happy(alice)`)
- Herbrand interpretation: a truth assignment to every element in the Herbrand base (e.g., `happy(alice) is FALSE`)

# Inductive logic programming



Learning in ILP systems can happen with the paradigm of

- Learning from Entailment (LFE)
- Learning from Interpretations (LFI)
- Other settings are possible, but much less used

# Learning from Entailment

## 5 Inductive logic programming

Learning from Entailment is based on a knowledge base $B$, positive examples $E^+$ and negative examples $E^-$ of the concept we aim to learn: the goal is to learn a logic theory $H$ from a hypothesis space $\mathcal{H}$ that explains the target concept

$$\forall e \in E^+ \qquad H \cup \mathcal{H} \models e$$

$$\forall e \in E^- \qquad H \cup \mathcal{H} \not\models e$$

# Learning from Interpretations



Learning from Entailment is based on a knowledge base $B$ and on sets of positive examples $E^+$ and negative examples $E^-$ that come from different Herbrand interpretations: the goal is to learn a logic theory $H$ from a hypothesis space $\mathcal{H}$ that models positive examples

$$\forall e \in E^+ \qquad e \quad \text{is a model for} \quad H \cup B$$

$$\forall e \in E^- \qquad e \quad \text{not a model for} \quad H \cup B$$

# How to define an ILP system



Which elements to define when developing a novel ILP system?

1. Learning setting to adopt (i.e., LFE vs. LFI)
2. Representation language (limitations on hypotheses and background knowledge)
3. Language bias (how to define the hypothesis space)
4. Search method (how to search the hypothesis space)

# ALEPH

## 5 Inductive logic programming

A classic ILP system developed in 2001, grounded in the LFE setting

1. Select a positive example to generalize
2. Construct the **most specific clause** that is **consistent** with the language bias and **entails** the example $\longrightarrow$ this is called the **bottom clause**
3. Search for a clause that is more **general** than the bottom clause and that has some best score ($P - N$ being $P$ the positive and negative examples covered by the clause)
4. Add clause to $H$, remove **positive** examples covered by clause, and go back to step 1