Department of Mathematics
ETH Zurich

CAS ETH Machine Learning in Finance and Insurance

BLOCK I. Introduction to Machine Learning. Lecture 6.

Dr. A. Ferrario, ETH Zurich and UZH

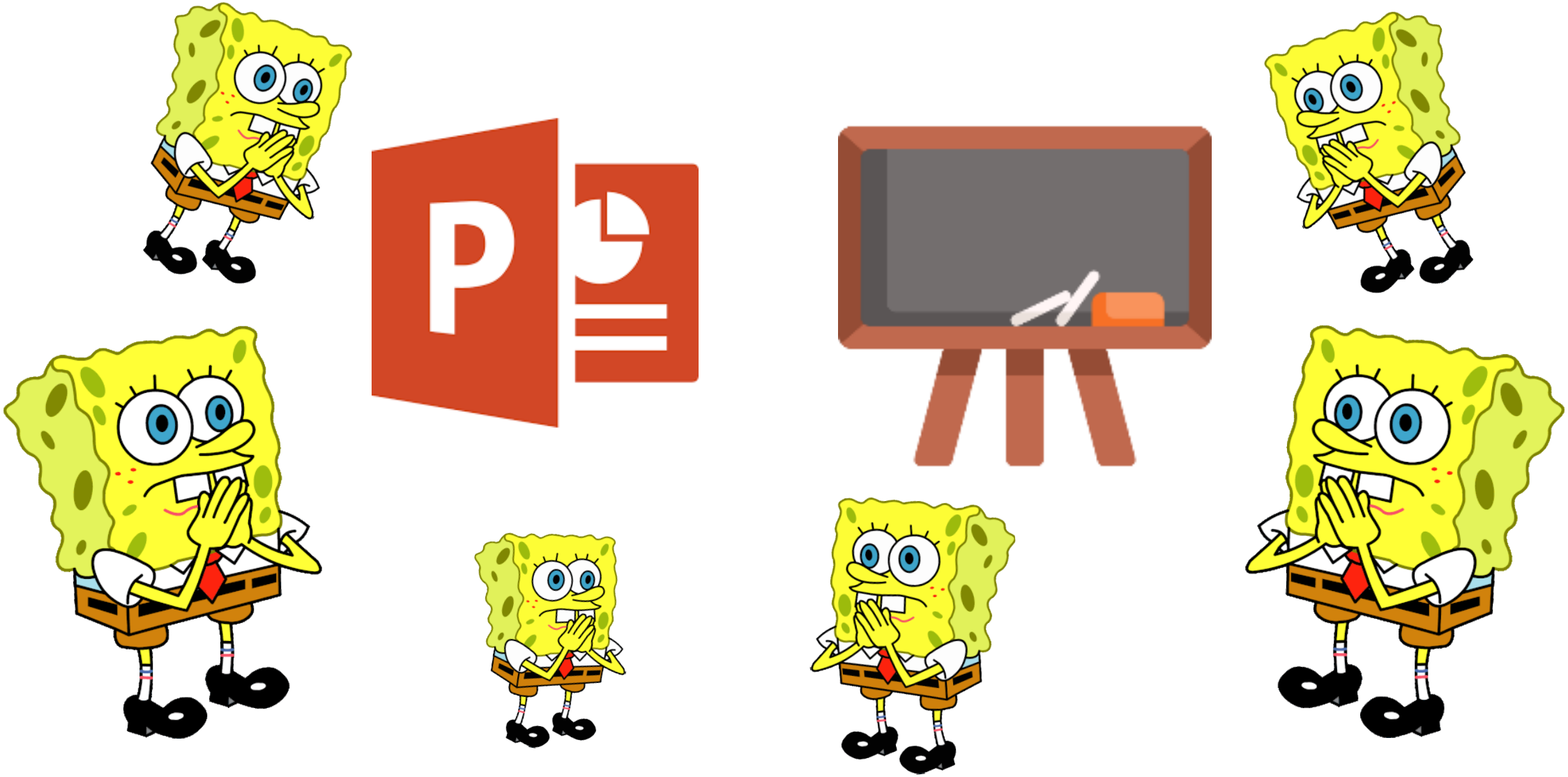
On April 19th…

1	We concluded our investigation of feedforward neural networks (FNNs) by focusing on selected design elements (e.g., activation functions, depth and width)
2	**You successfully tackled our interactive exercise**, introducing a machine learning task, designing an FNN, discussing its parameters, and writing the function representing it – good job!
3	Finally, we discussed key elements of training FNNs, including weights initialization, variations of stochastic gradient descent, regularization and cross-validation

Machine Learning Methods (Part 6)

- Decision Trees: : Overview
- Training Decision Trees

We will use slides to introduce our topics and the blackboard to deep-dive into selected items



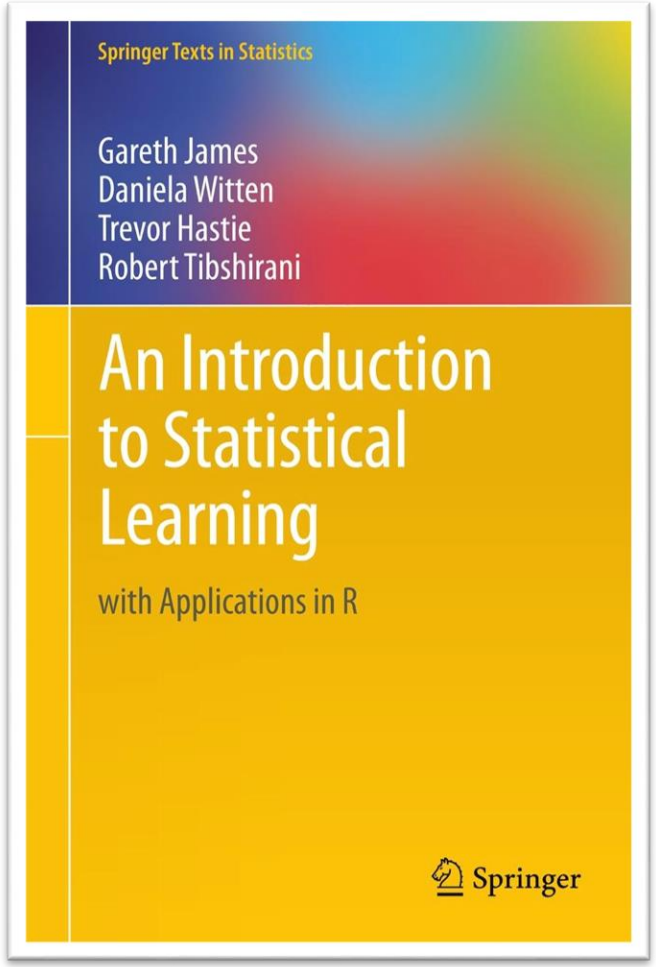
Decision Trees: Overview

Some trees making decisions in a court room by ChatGPT 4.0

After all that time and work on deep learning models, why (decision) trees?
Do we really need to talk about them?

"Decision trees for regression and classification have a number of advantages over [linear and logistic regression models]:

- Trees are very easy to explain to people. In fact, they are even easier to explain than linear regression!
- Some people believe that decision trees more closely mirror human decision-making than do the regression and classification approaches […]
- Trees can be displayed graphically, and are easily interpreted even by a non-expert (especially if they are small)
- Trees can easily handle qualitative predictors without the need to create dummy variables" (pag. 315, emphasis in original)

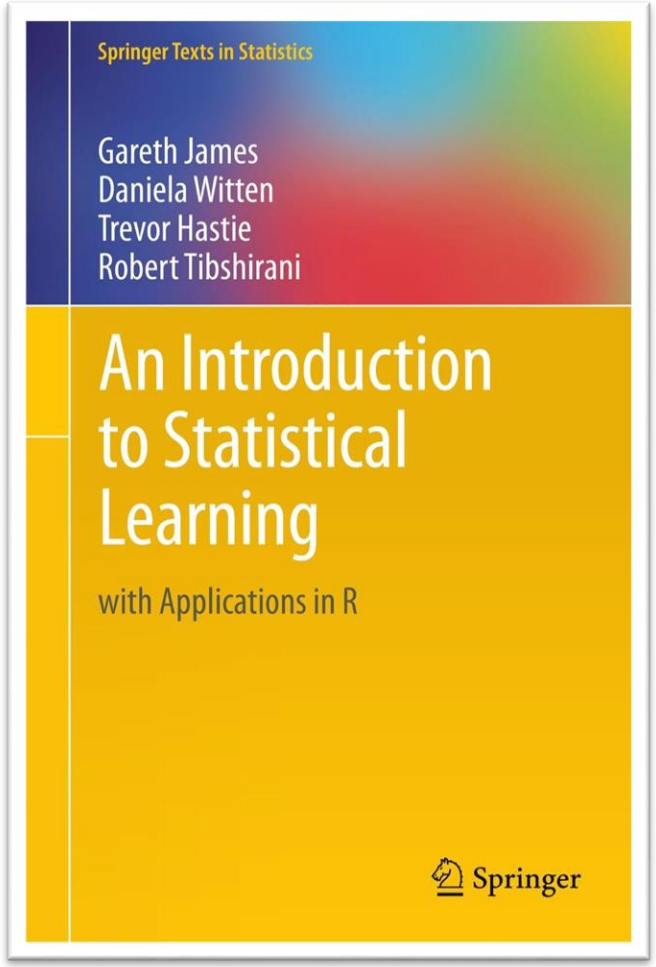


James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). An introduction to statistical learning. New York: Springer

After all that time and work on deep learning models, why (decision) trees?
Do we really need to talk about them?

More advanced methods (e.g., deep learning) can beat a decision tree. However, **aggregating many trees** together can give rise to a more powerful machine learning model

Random forests, bagging, and boosting methods…



James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). An introduction to statistical learning. New York: Springer

Decision trees
Giving the idea before a definition

We introduce the idea behind **binary** decision trees before commenting on what is needed to learn them

Decision trees
A binary classification use case

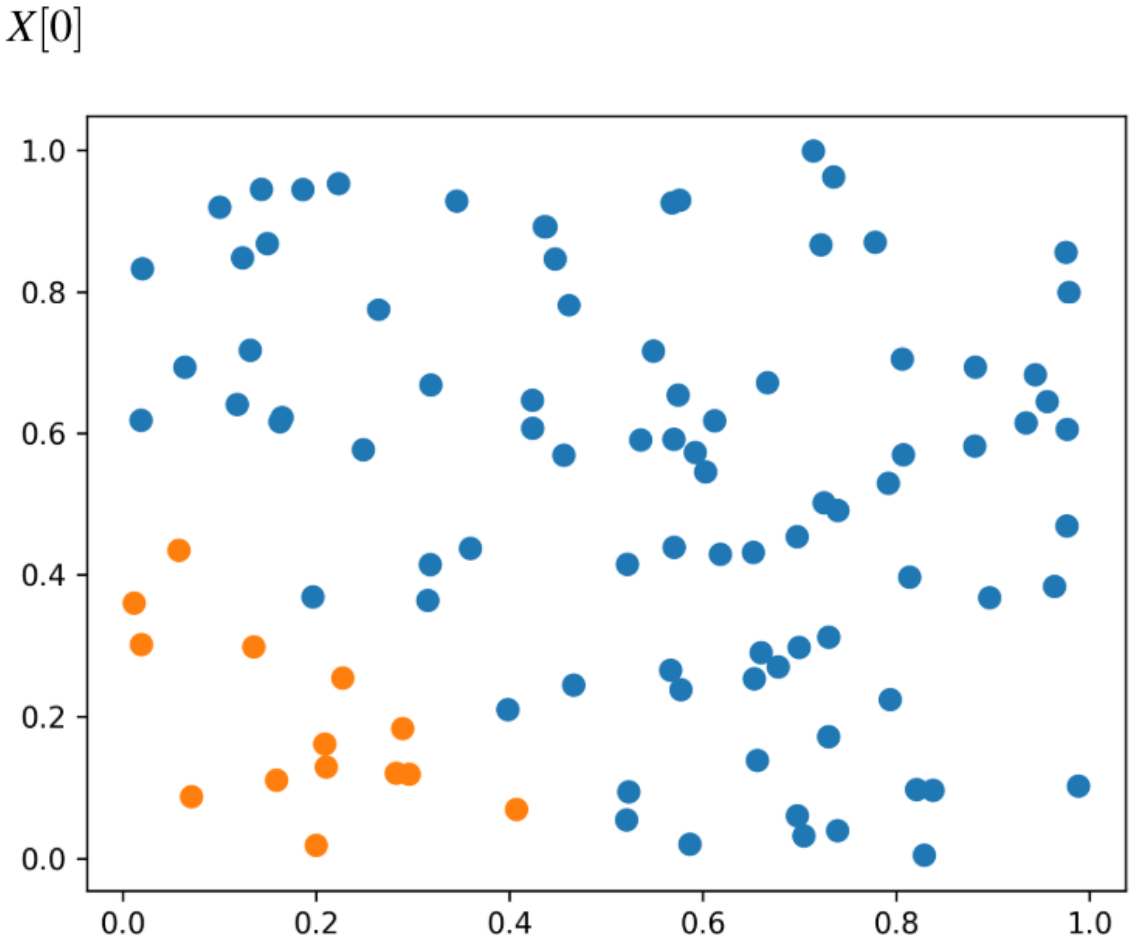
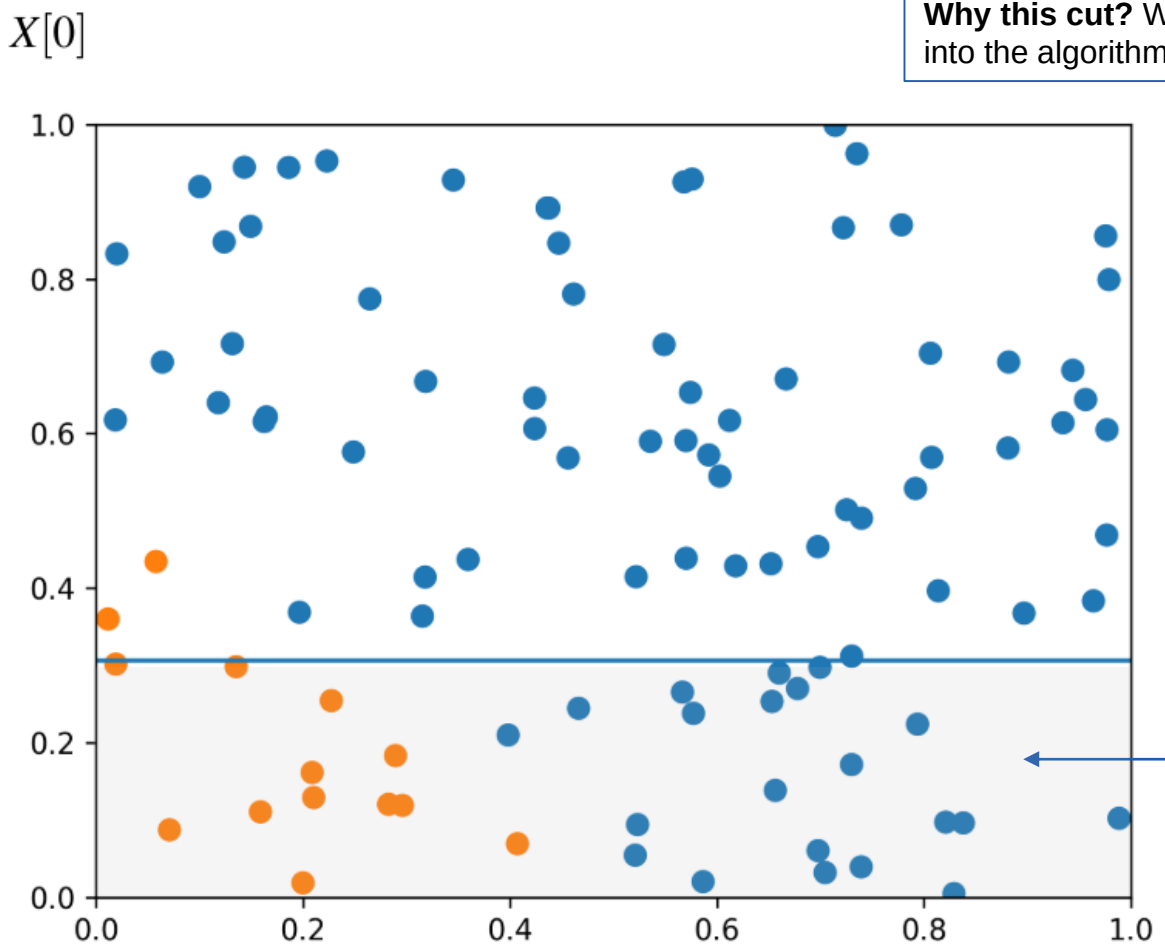


Figure: Points $x^1, x^2, \ldots, x^m \in \mathbb{R}^2$ with corresponding labels $y^1, y^2, \ldots, y^m \in \{-1, 1\}$ (blue or orange) with $m = 100$.

The **decision tree algorithm** will try learning to classify the *m* points by splitting them into “regions” iteratively.

Decision trees
A binary classification use case



Why this cut? We would need to deep-dive into the algorithm to answer this question.

?

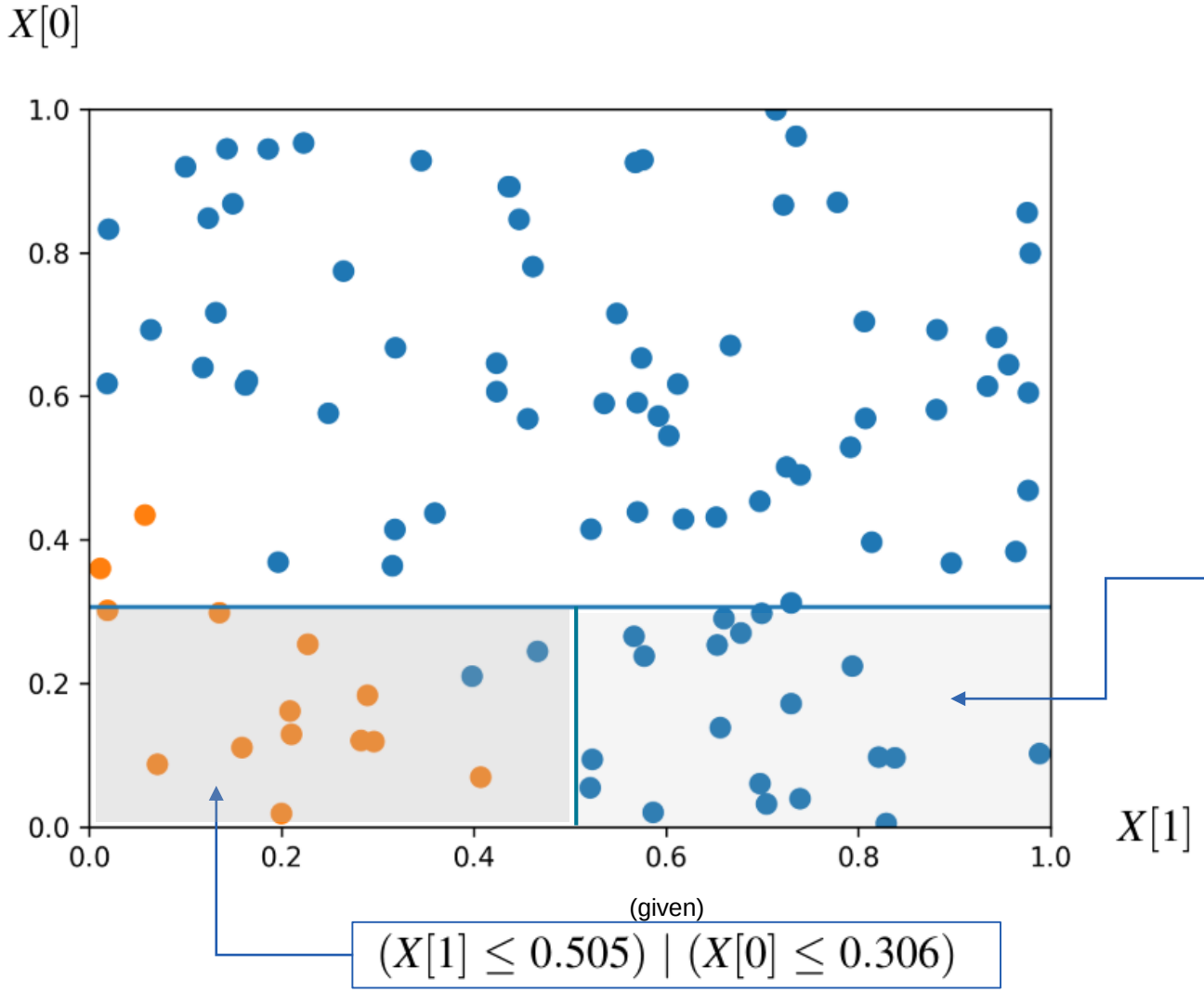
$X[0] \leq 0.306$

?

We create **rectangular** regions learning inequalities

First step: the algorithm cuts the data points into two regions along $X[0]$.

Decision trees
A binary classification use case

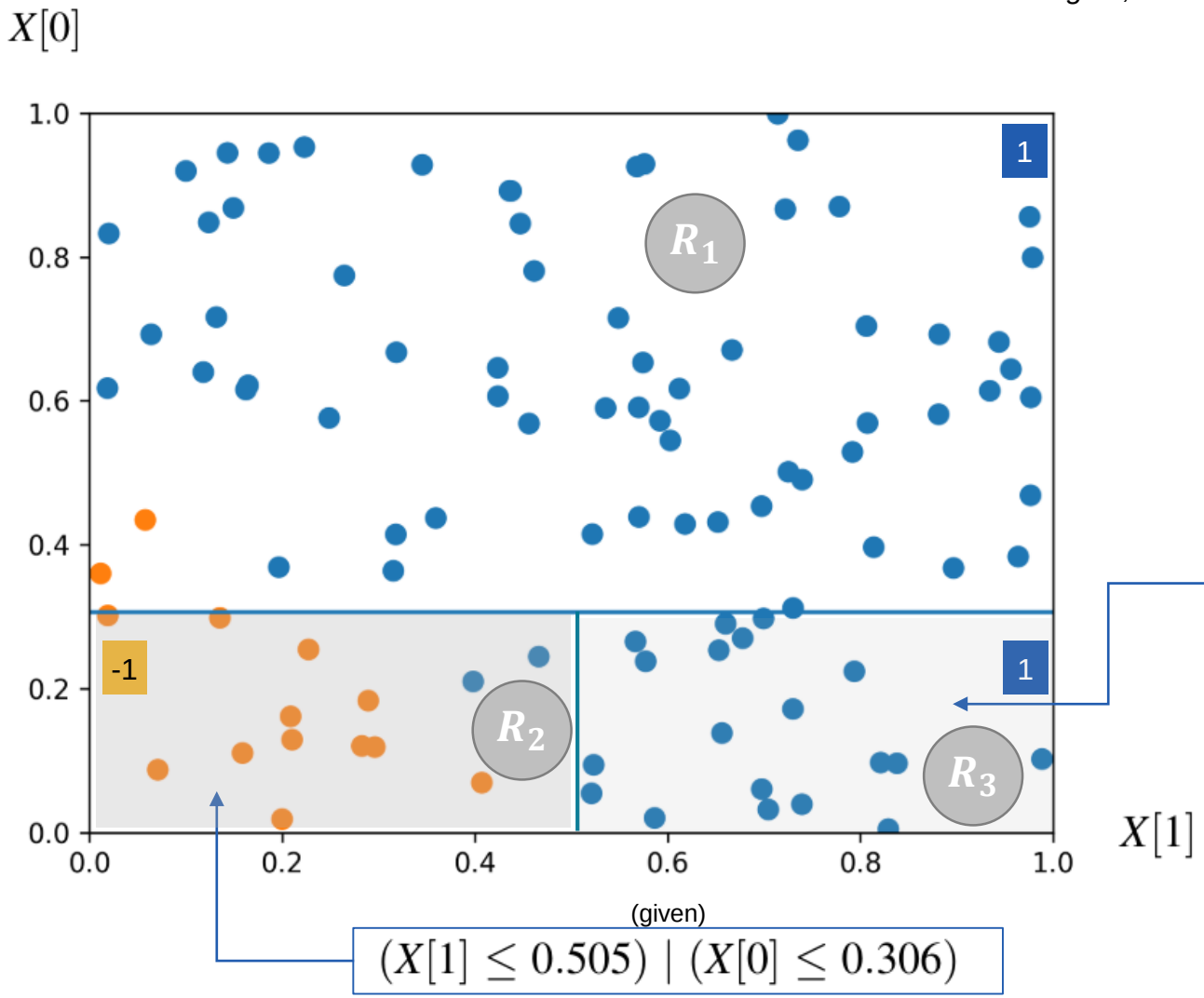


Second step: the algorithm adds another cut, this time, along $X[1]$ and **<u>it stops</u>**.

Why does it stop now? We need to deep-dive into the algorithm to explain this.

Decision trees
A binary classification use case

"*region, terminal node, leaf*" R_*



$X[0] \leq 0.306$

(given)

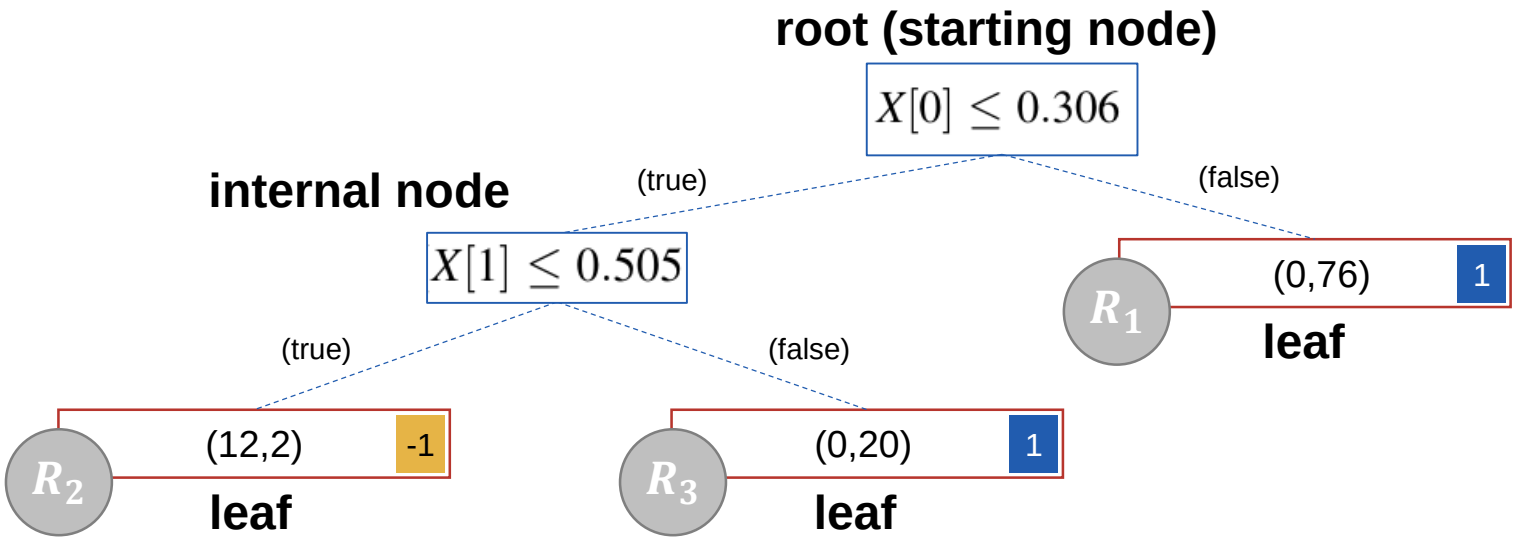
$(X[1] \leq 0.505) \mid (X[0] \leq 0.306)$

Final step: the algorithm assigns a label to all data points in a region, for all regions, using the **majority rule**.

Why this rule? Customary choice in classification problems. It is equivalent to computing the mode of the distribution of the labels in each region.

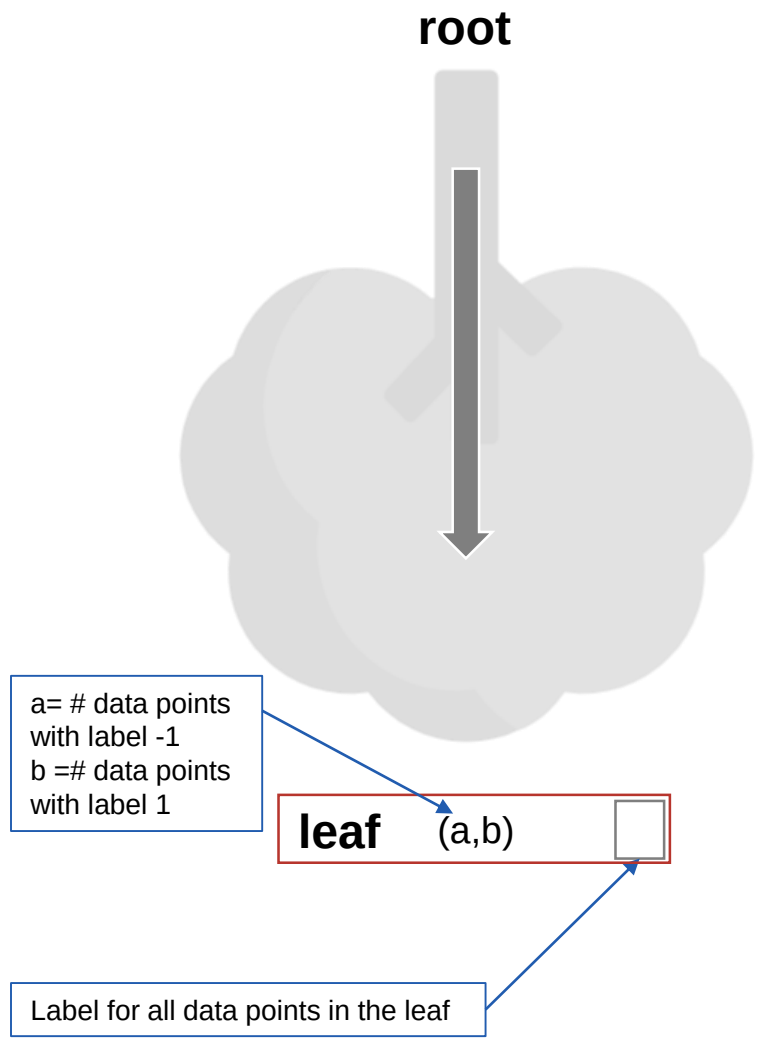
Decision trees

This is how our decision tree for the classification use case would look like in Python



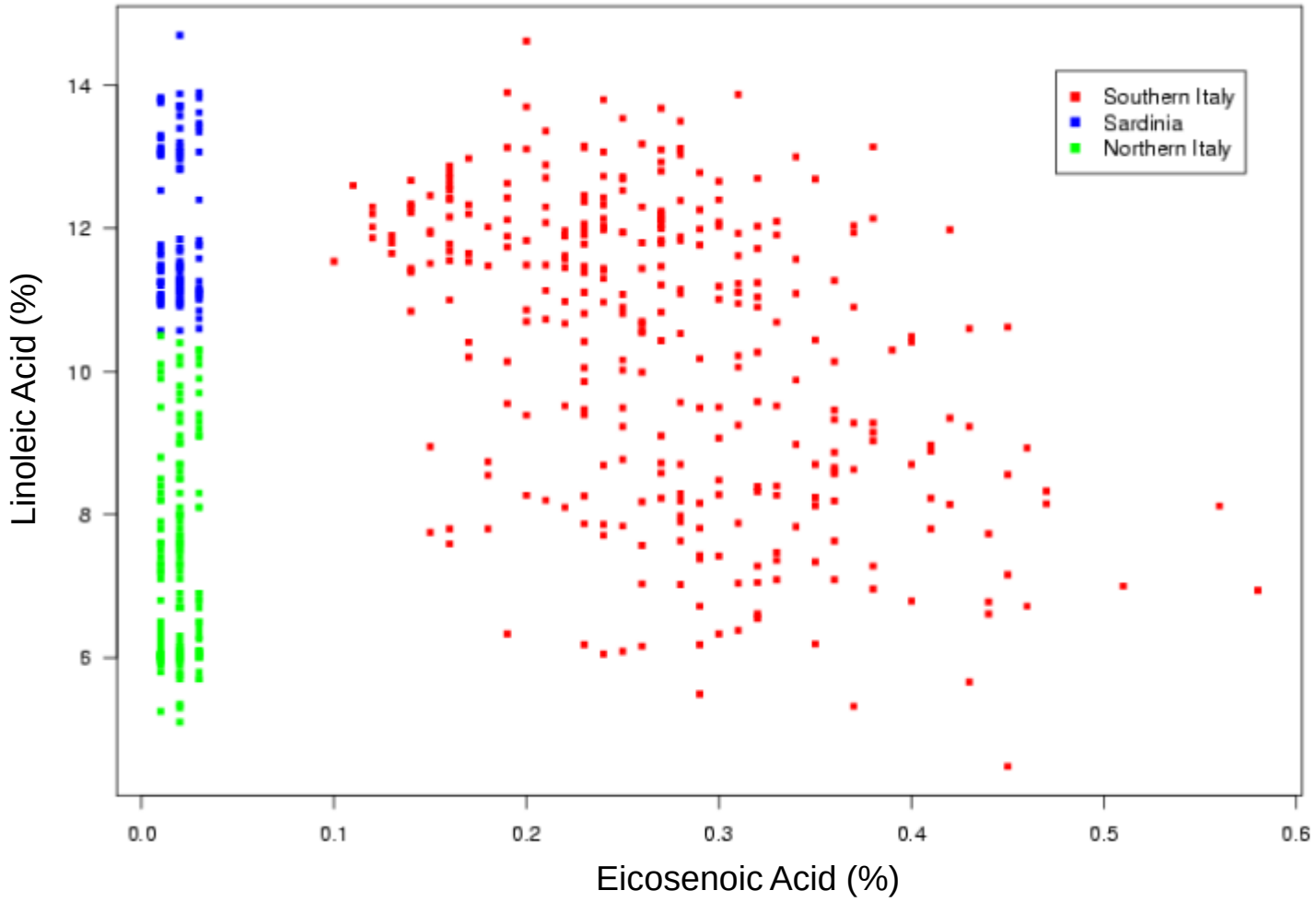
(leaves are terminal nodes)

Our simple decision tree classifies correctly 98/100 data points…is this good or not?



Decision trees

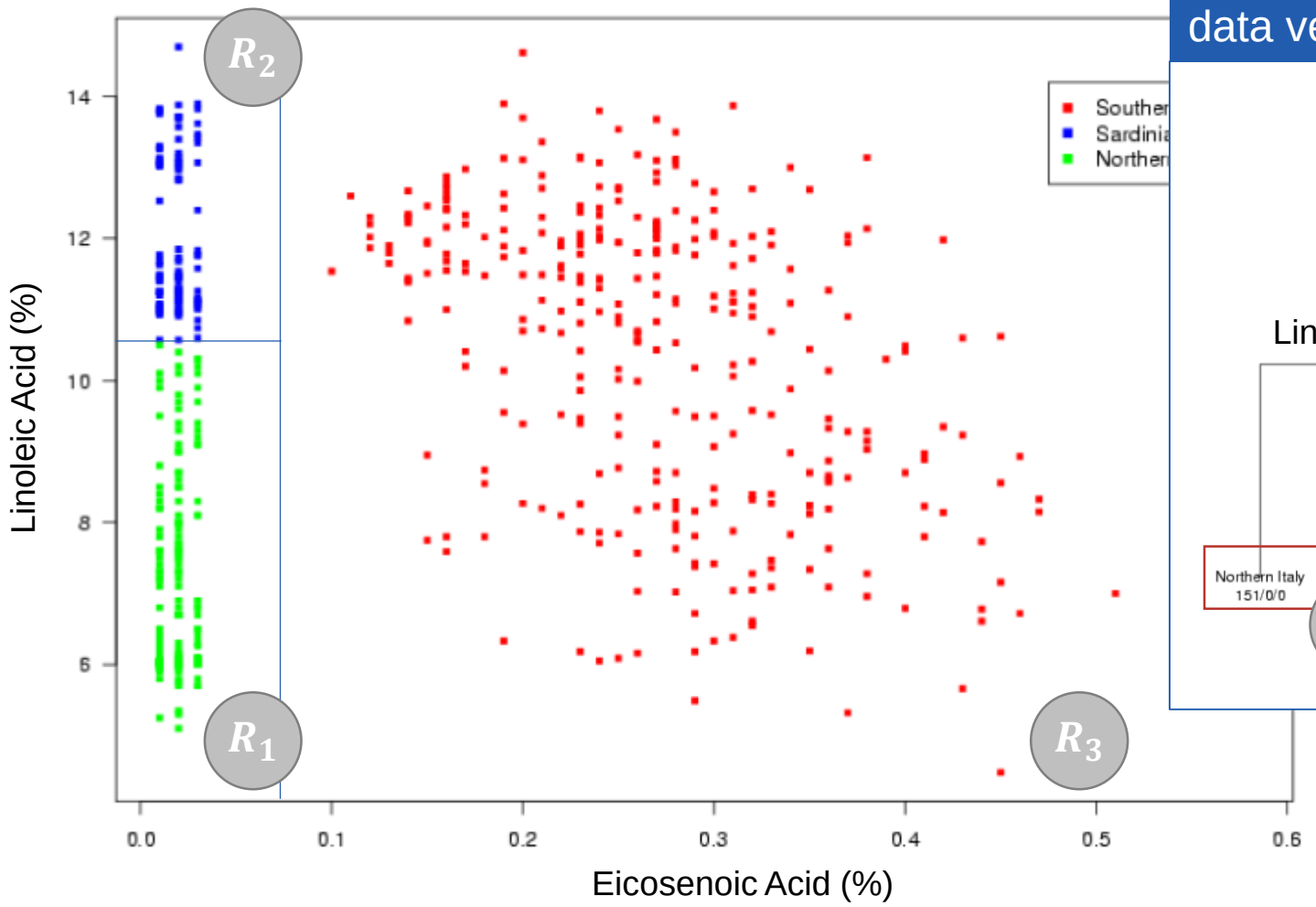
Sometimes data just need a simple decision tree: A real-world classification use case from food research



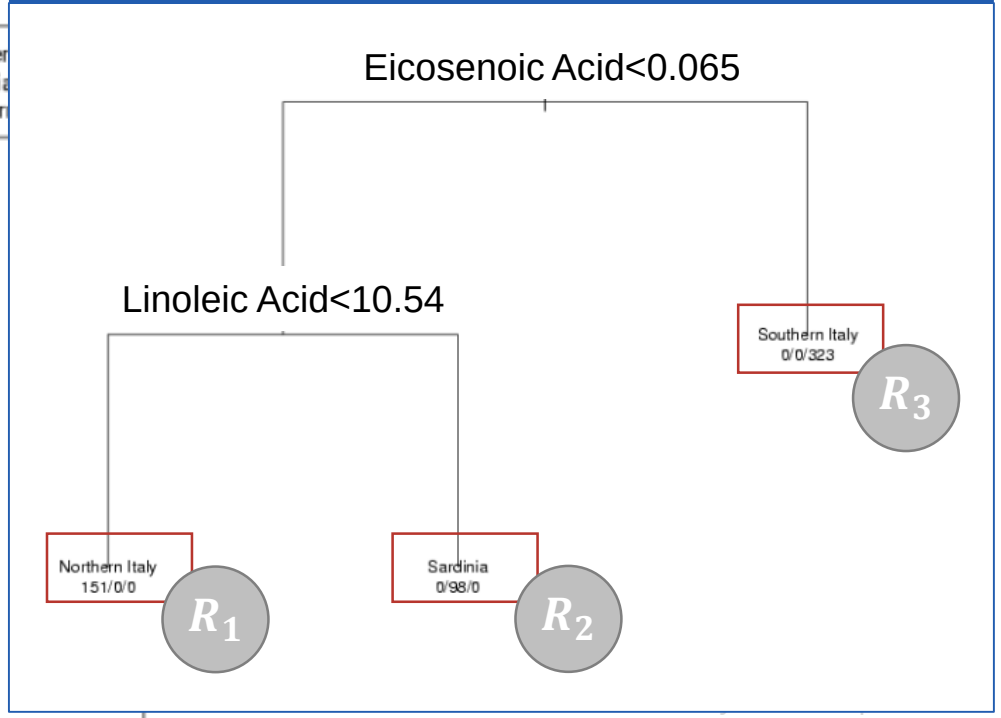
- 572 olive oil samples from three Italian regions: Northern, Southern and Sardinia.
- Eight fatty acids, including linoleic and eicosenoic.

Forina M, Armanino C, Lanteri S, and Tiscornia E (1983). Classification of olive oils from their fatty acid composition. In Martens H and Russwurm Jr H, editors, Food Research and Data Analysis, pp 189-214. Applied Science Publishers, London.

Decision trees
A real-world classification use case from food research



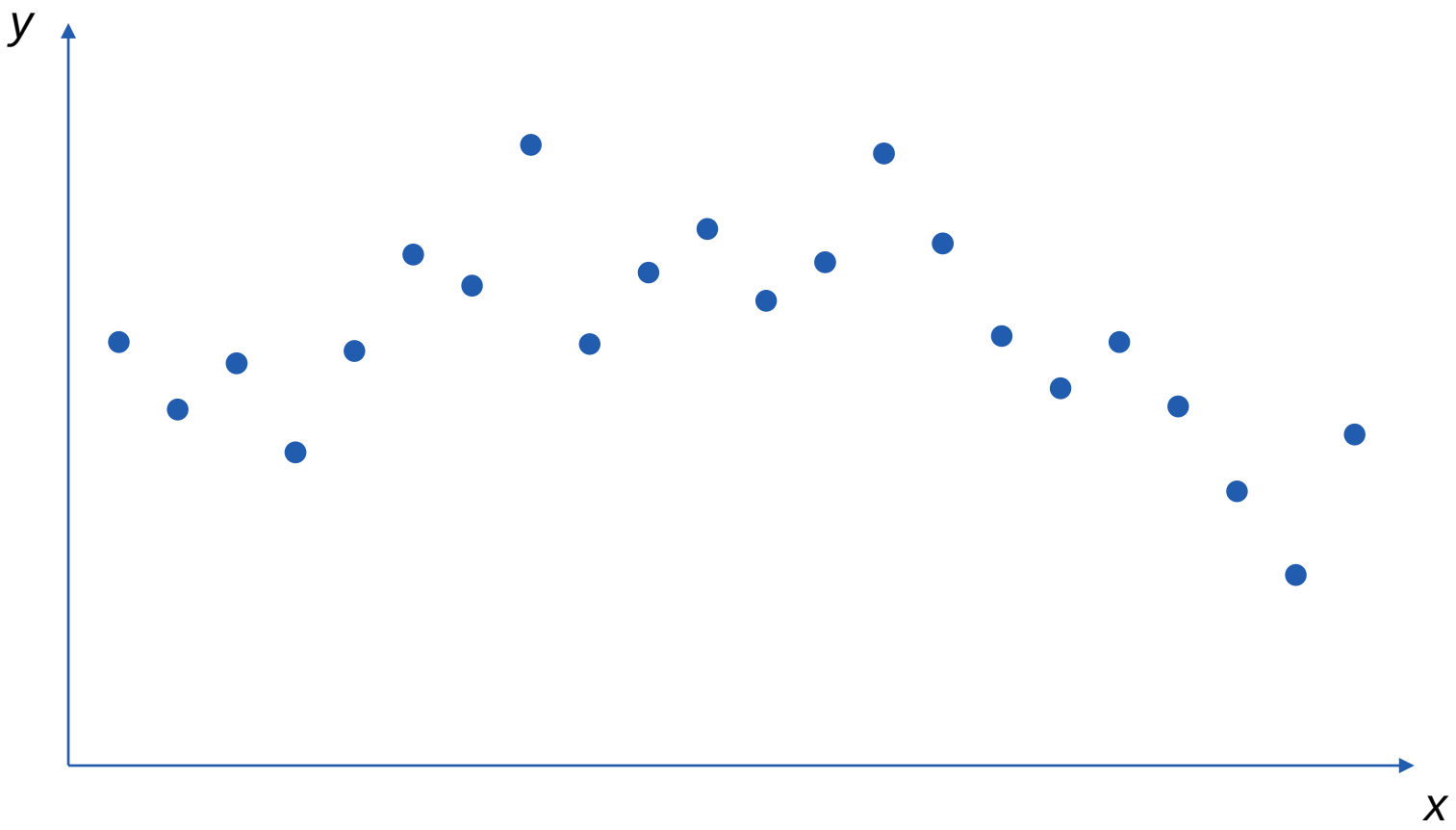
A simple decision tree can fit the structure of data very accurately



In Martens H and Russwurm Jr H, editors, Food Research and Data Analysis, pp 189-214. Applied Science Publishers, London.

Decision trees
A regression use case

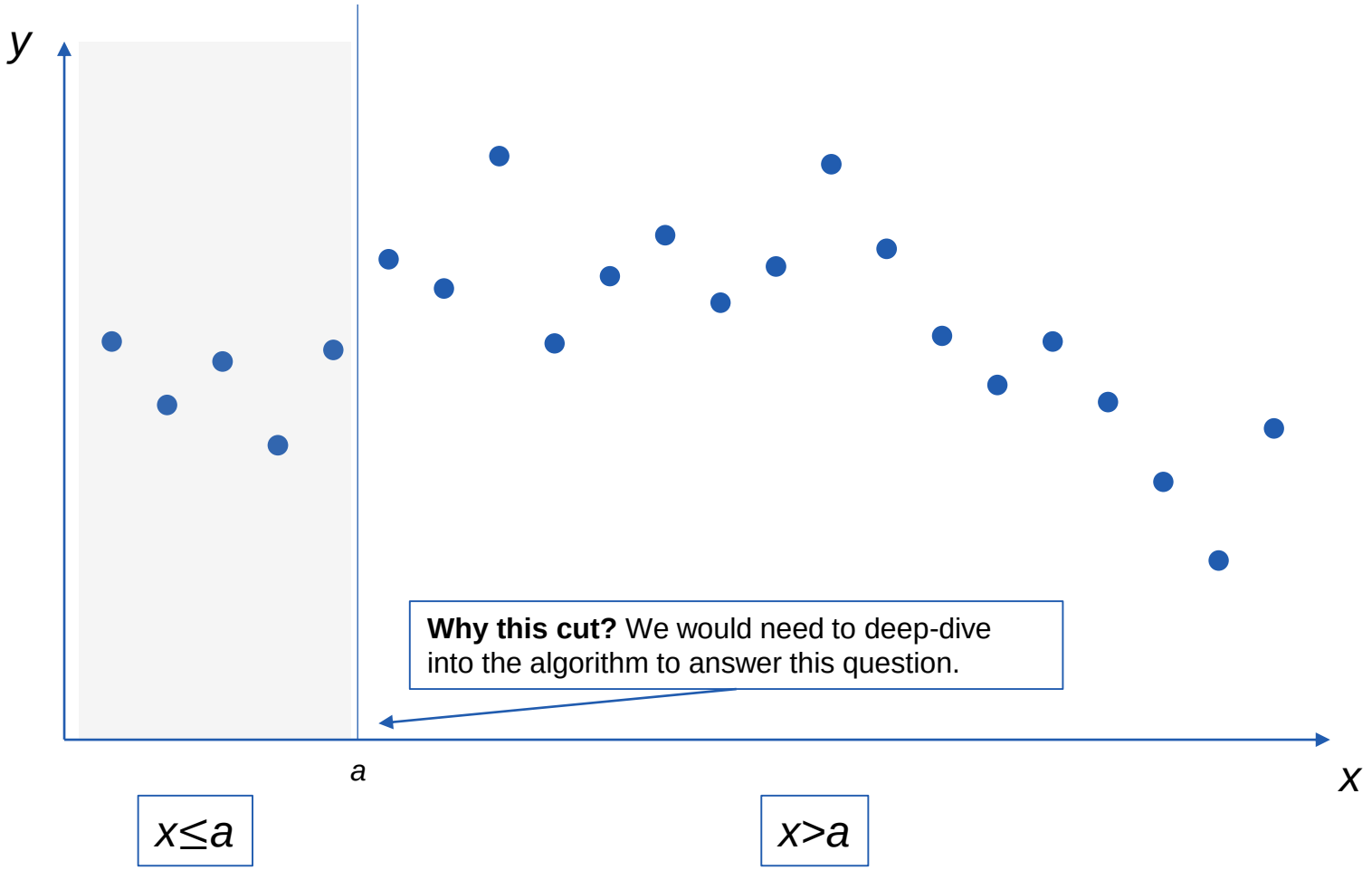
II



The **decision tree algorithm** will try learning to predict the *y*-value of the data points by splitting them into “regions” iteratively.

Decision trees
A regression use case

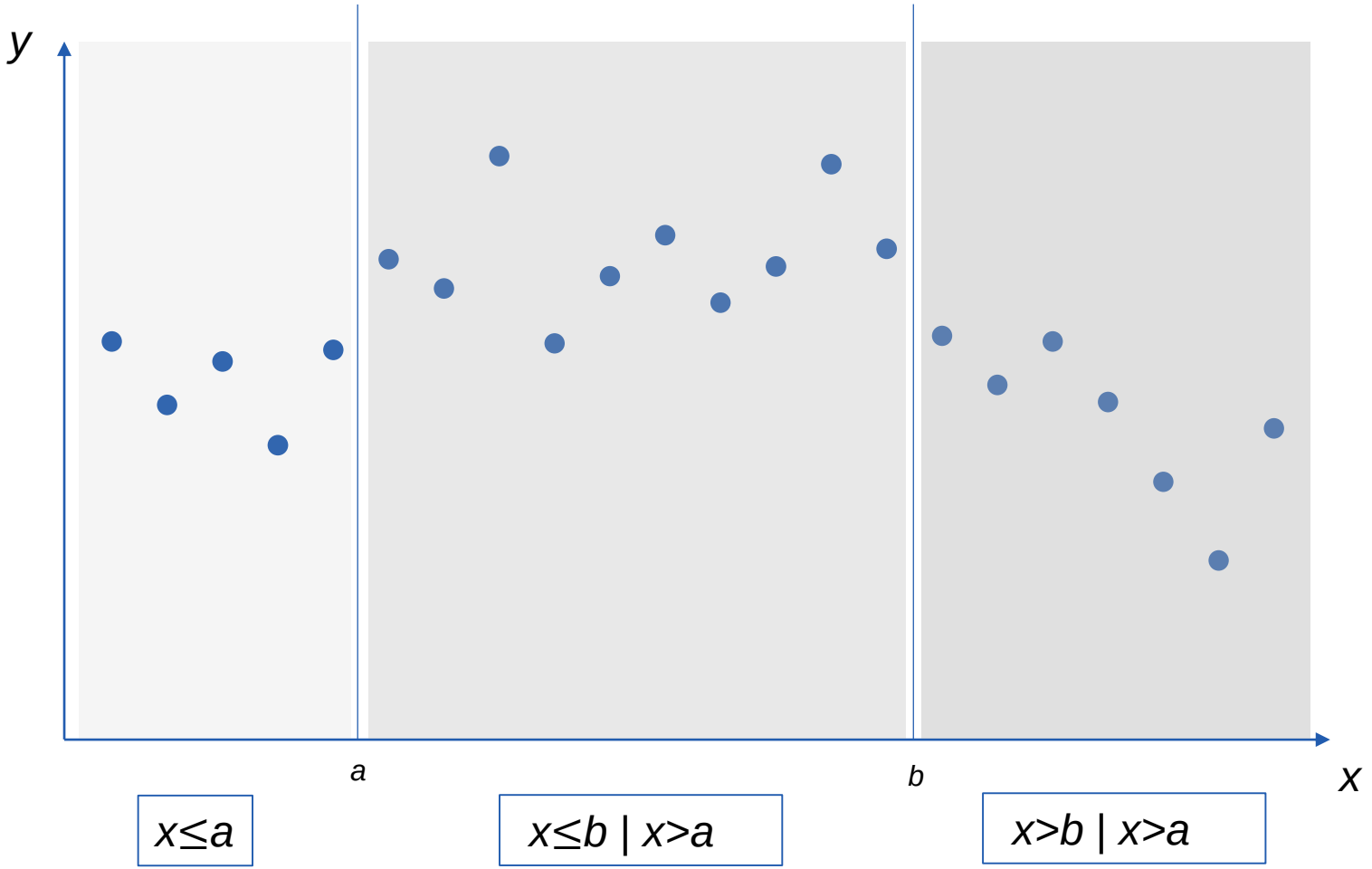
II



First step. The decision tree algorithm cuts the predictor *x* at *x=a*.

Decision trees
A regression use case

II

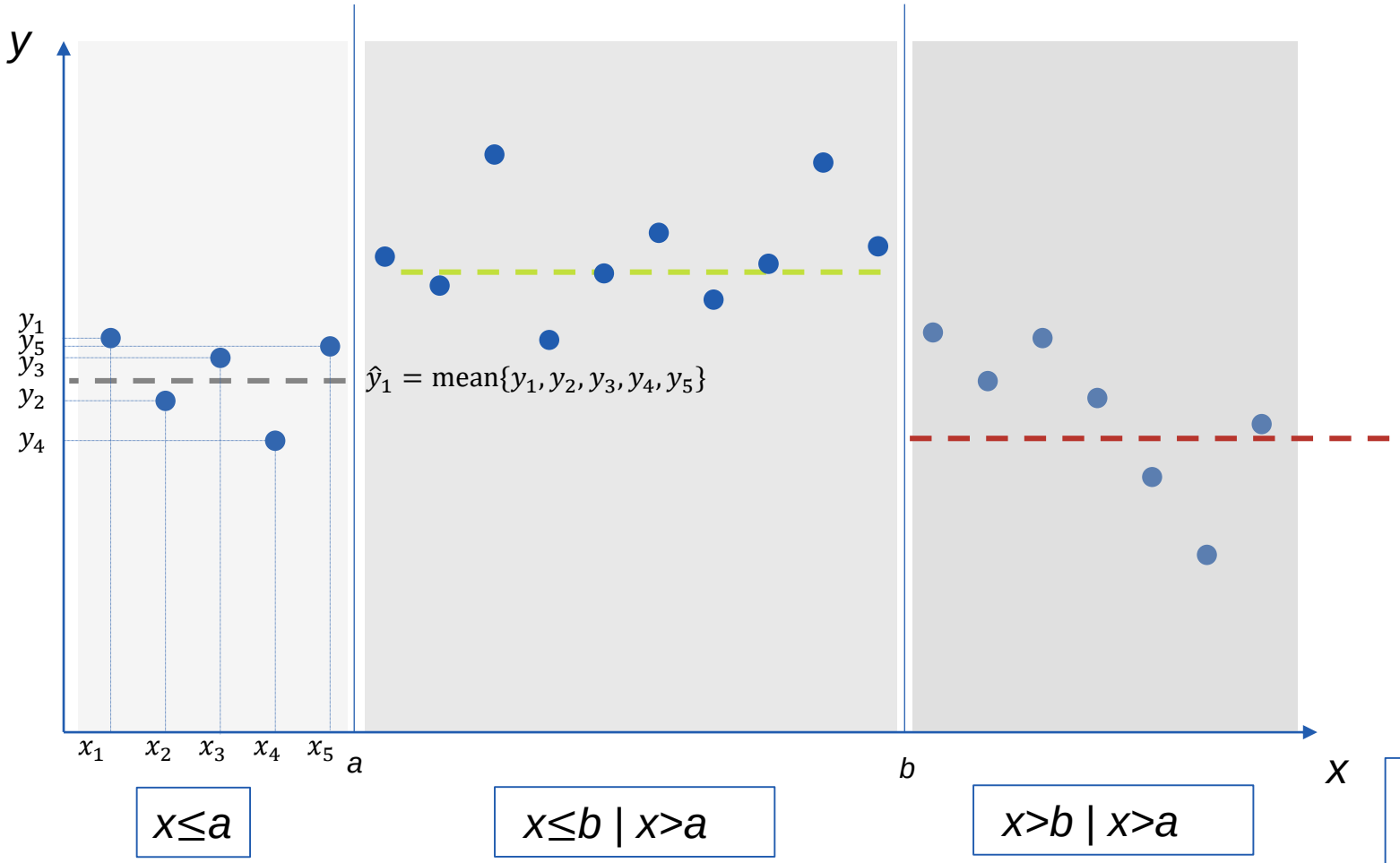


Second step. the algorithm adds another cut at $x=b$ and **it stops**.

Again, why does it stop now? We need to deep-dive into the algorithm to explain this.

Decision trees
A regression use case

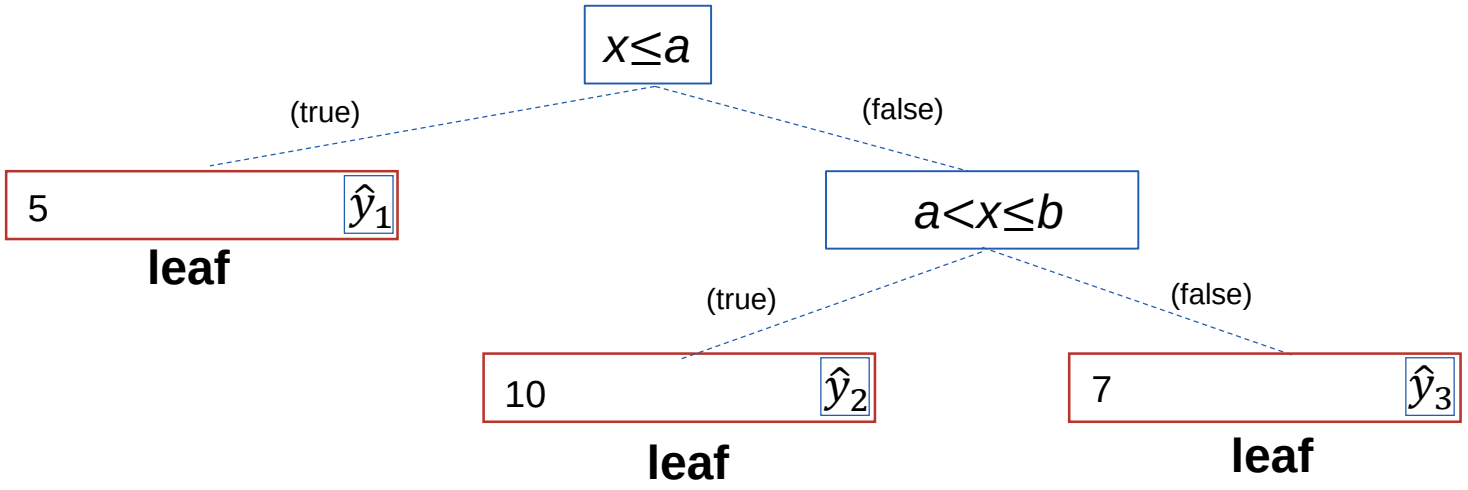
II

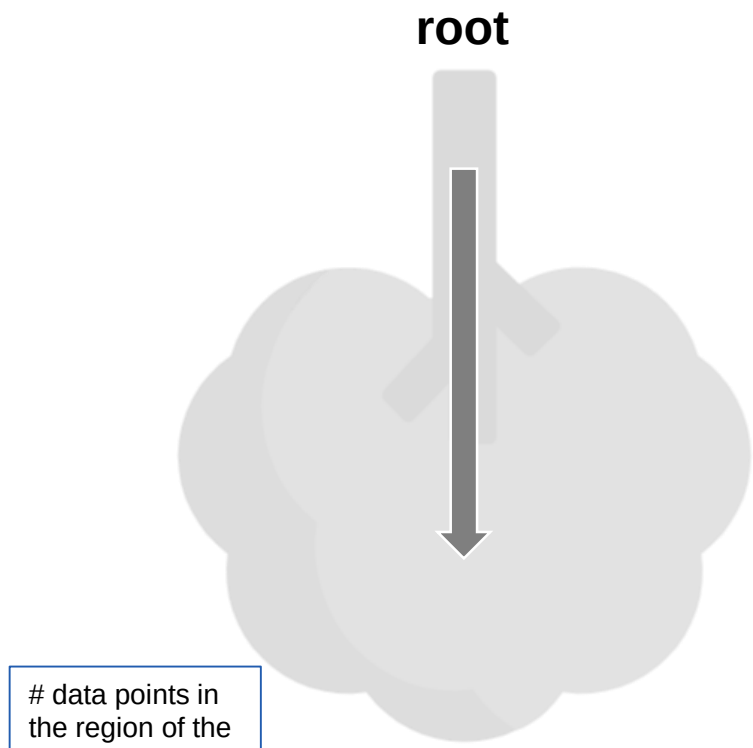


Last step. the algorithm computes the predicted *y*-value for all data points in a region using the **mean rule**, for all regions.

Why this rule? Customary choice in regression problems. It is equivalent to computing the average of the distribution of the *y*-values in each region.

Decision trees
A regression use case





Summary

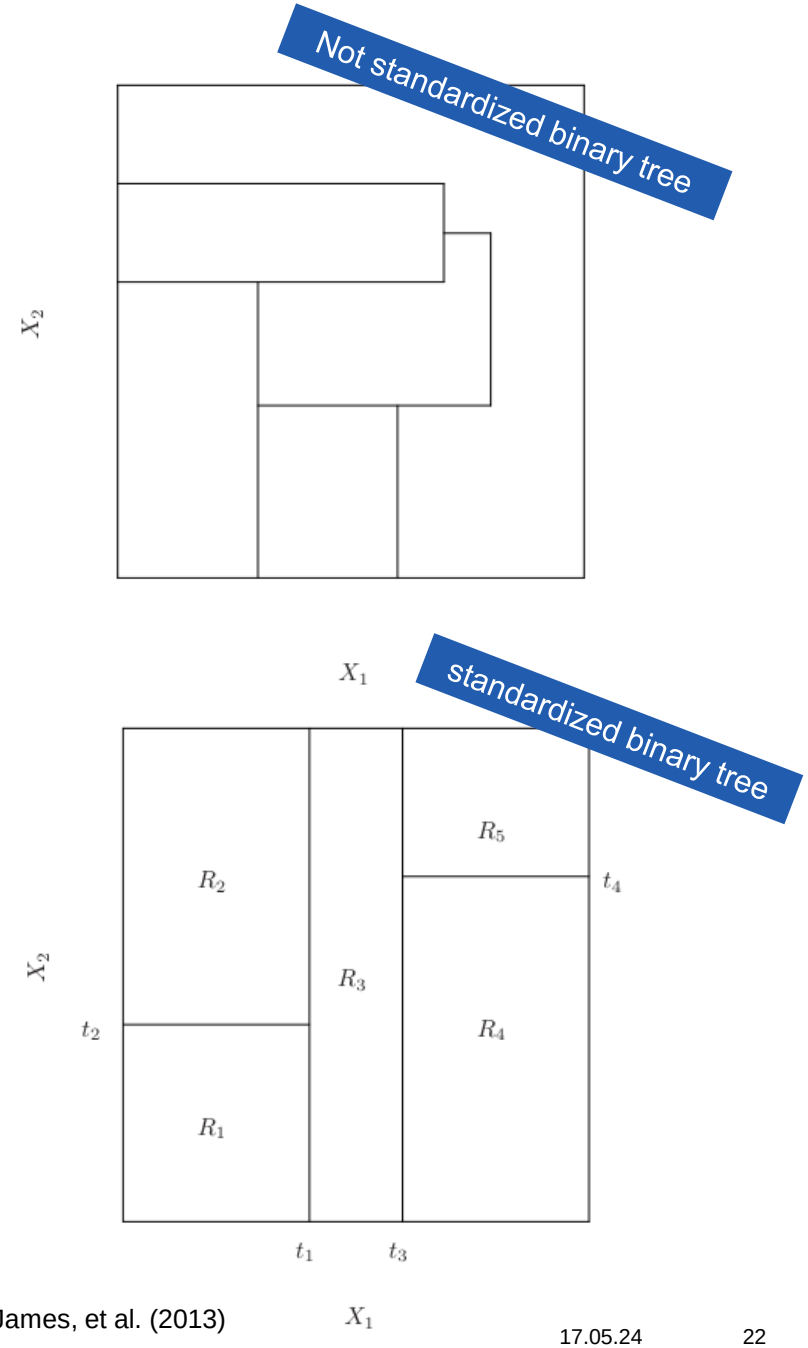
We consider **standardized binary trees**

I

Binary – each “decision”, or “split”, is binary (true/false, yes/no etc.)

II

Standardized – each decision, or split, is based on a single feature only.



Source: James, et al. (2013)

Summary (training)

The algorithms used to train decision trees need three key specifications

I **Splitting criterion** – "*how to grow the tree?*"

II **Stopping criterion** – "*when is the tree grown enough?*"

III **Rule to assigning values to all leaves** – "*how to compute predictions?*"

1 Also known as a **partition** of the feature space

As a result, the decision tree algorithm

1. "divide[s] the predictor space—that is, the set of possible values for $X_1, X_2, \ldots, X_p$ –into J distinct and non-overlapping regions, $R_1, R_2, \ldots, R_J$."

2. "[f]or ever observation that falls in the region R_J, [it] make[s] the same prediction"

(pag. 306, James et al. 2013)

2 Decision trees implement **piecewise constant functions**

Summary (predicting)
Predicting with decision trees

Data point	$x = (x_1, \ldots, x_p) \in \mathcal{X}$
"Regions" learned by the tree	$R_1, \ldots, R_J$

$$f : \mathcal{X} \to \mathcal{Y}, \quad f(x) = \sum_{j=1}^{J} y_j \ \mathbf{1}_{x \in R_j}$$

(Piecewise constant) function represented by the tree

Summary (predicting)
Predicting with decision trees

Data point

$$x = (x_1, \ldots, x_p) \in \mathcal{X}$$

"Regions" learned by the tree

$$R_1, \ldots, R_J$$

1. Usual notation: each data point is a vector with *p* components ("features" or "predictors").

2. Here, we assume that a decision tree algorithm has "learned" *J* distinct, non-overlapping regions of $\mathcal{X}$.

3. Predicted y-value **for all data points** in the region/leaf R_j

$$f : \mathcal{X} \to \mathcal{Y}, \quad f(x) = \sum_{j=1}^{J} y_j \mathbf{1}_{x \in R_j}$$

(Piecewise constant) function represented by the tree

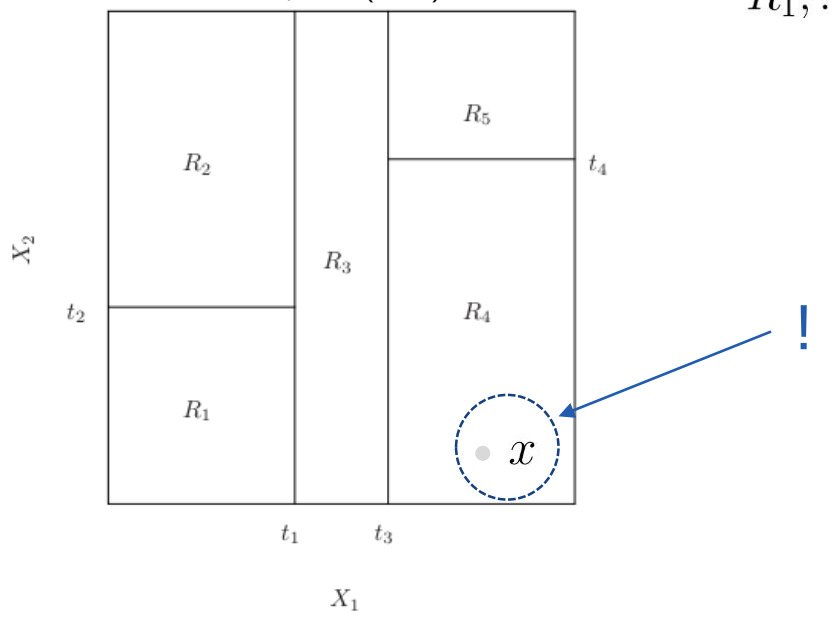
4. Indicator function: it returns **1** for the (unique) *j* such that $x \in R_j$, **0** otherwise

Summary (predicting)
Predicting with decision trees

$$x = (x_1, x_2) \in \mathcal{X}$$

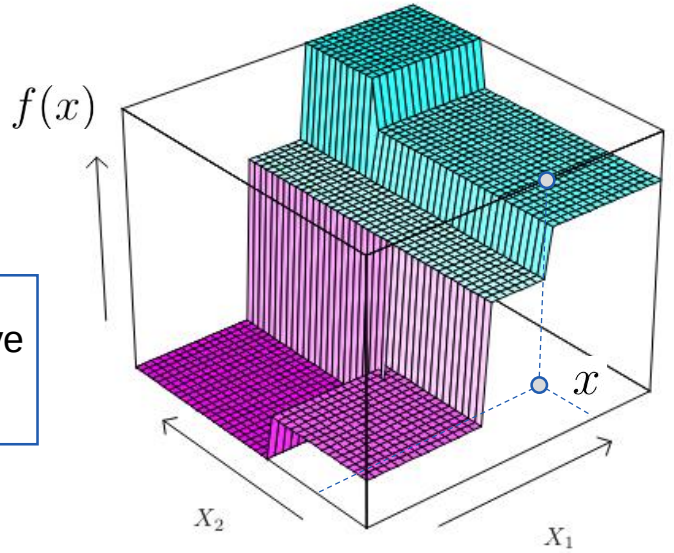
$$R_1, \ldots, R_5$$

Source: James, et al. (2013)



Data point	$x = (x_1, \ldots, x_p) \in \mathcal{X}$
"Regions" learned by the tree	$R_1, \ldots, R_J$

Piecewise constant function: on each of the five regions, the function f is constant.



$$(x_1, x_2, f(x))$$

$$f(x) = y_4$$

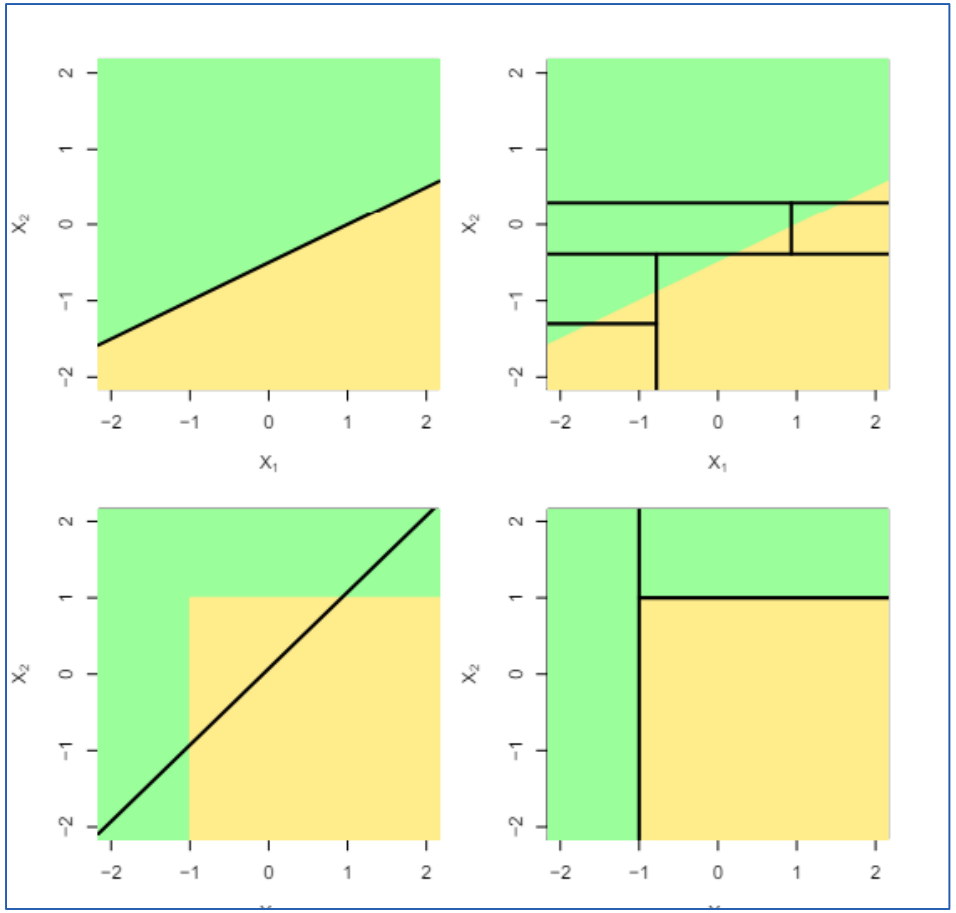
Source: James, et al. (2013)

Before moving to training: decision trees vs. linear regression models

Which model is better?

"Which model is better? It depends on the problem at hand. If the relationship between the features and the response is well approximated by a linear model […], then an approach such as linear regression will likely work well […] If instead there is a highly non-linear and complex relationship between the features ad the response […] decision trees may outperform classical approaches." (pag. 314 James et al. 2013)

"Of course, other considerations beyond simply test error may come into play in selecting a statistical learning method; for instance, in certain settings, prediction using a tree may be preferred for the sake of interpretability and visualization" (pag. 314, James et al. 2013)

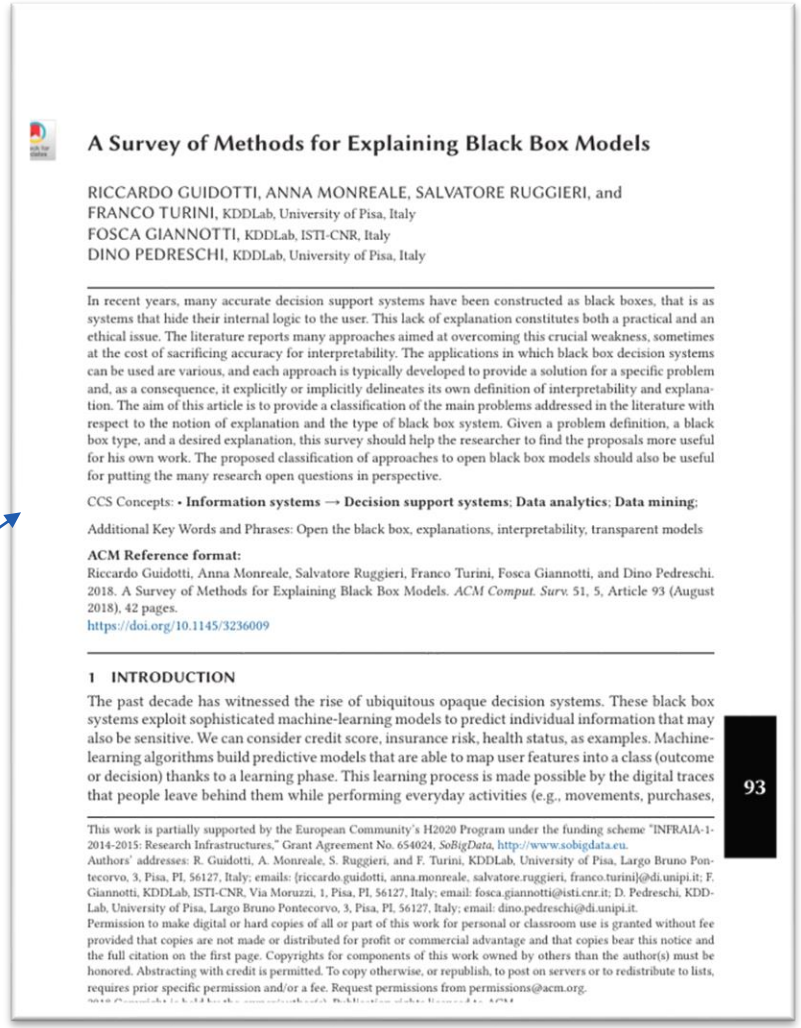


Source: James et al. (2013)

An important word what will return later in our CAS: “interpretability”

“Which model is better? It depends on the problem at hand. If the relationship between the features and the response is well approximated by a linear model […], then an approach such as linear regression will likely work well […] If instead there is a highly non-linear and complex relationship between the features ad the response […] decision trees may outperform classical approaches.” (pag. 314 James et al. 2013)

“Of course, other considerations beyond simply test error may come into play in selecting a statistical learning method; for instance, in certain settings, prediction using a tree may be preferred for the sake of ***interpretability*** and visualization” (pag. 314, James et al. 2013)



A Survey of Methods for Explaining Black Box Models

RICCARDO GUIDOTTI, ANNA MONREALE, SALVATORE RUGGIERI, and
FRANCO TURINI, KDDLab, University of Pisa, Italy
FOSCA GIANNOTTI, KDDLab, ISTI-CNR, Italy
DINO PEDRESCHI, KDDLab, University of Pisa, Italy

In recent years, many accurate decision support systems have been constructed as black boxes, that is as systems that hide their internal logic to the user. This lack of explanation constitutes both a practical and an ethical issue. The literature reports many approaches aimed at overcoming this crucial weakness, sometimes at the cost of sacrificing accuracy for interpretability. The applications in which black box decision systems can be used are various, and each approach is typically developed to provide a solution for a specific problem and, as a consequence, it explicitly or implicitly delineates its own definition of interpretability and explanation. The aim of this article is to provide a classification of the main problems addressed in the literature with respect to the notion of explanation and the type of black box system. Given a problem definition, a black box type, and a desired explanation, this survey should help the researcher to find the proposals more useful for his own work. The proposed classification of approaches to open black box models should also be useful for putting the many research open questions in perspective.

CCS Concepts: • **Information systems** → **Decision support systems**; **Data analytics**; **Data mining**;

Additional Key Words and Phrases: Open the black box, explanations, interpretability, transparent models

ACM Reference format:
Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Franco Turini, Fosca Giannotti, and Dino Pedreschi. 2018. A Survey of Methods for Explaining Black Box Models. *ACM Comput. Surv.* 51, 5, Article 93 (August 2018), 42 pages.
https://doi.org/10.1145/3236009

1 INTRODUCTION

The past decade has witnessed the rise of ubiquitous opaque decision systems. These black box systems exploit sophisticated machine-learning models to predict individual information that may also be sensitive. We can consider credit score, insurance risk, health status, as examples. Machine-learning algorithms build predictive models that are able to map user features into a class (outcome or decision) thanks to a learning phase. This learning process is made possible by the digital traces that people leave behind them while performing everyday activities (e.g., movements, purchases,

93

This work is partially supported by the European Community's H2020 Program under the funding scheme "INFRAIA-1-2014-2015: Research Infrastructures," Grant Agreement No. 654024, *SoBigData*, http://www.sobigdata.eu.
Authors' addresses: R. Guidotti, A. Monreale, S. Ruggieri, and F. Turini, KDDLab, University of Pisa, Largo Bruno Pontecorvo, 3, Pisa, PI, 56127, Italy; emails: {riccardo.guidotti, anna.monreale, salvatore.ruggieri, franco.turini}@di.unipi.it; F. Giannotti, KDDLab, ISTI-CNR, Via Moruzzi, 1, Pisa, PI, 56127, Italy; email: fosca.giannotti@isti.cnr.it; D. Pedreschi, KDDLab, University of Pisa, Largo Bruno Pontecorvo, 3, Pisa, PI, 56127, Italy; email: dino.pedreschi@di.unipi.it.

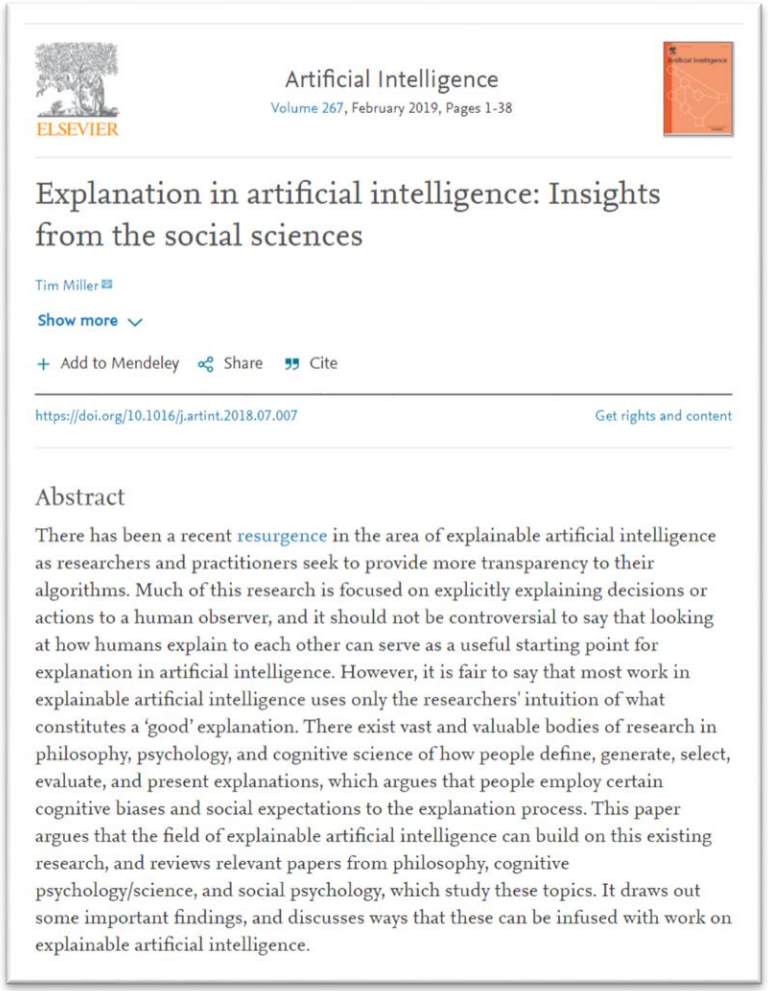
Guidotti, R., Monreale, A., Ruggieri, S., Turini, F., Giannotti, F., & Pedreschi, D. (2018). A survey of methods for explaining black box models. *ACM computing surveys (CSUR)*, *51*(5), 1-42.

An important word what will return later in our CAS: “interpretability”

The research domain called “Explainable Artificial Intelligence” (XAI) is devoted to the investigation of concepts such as “interpretability”

“Which model is better? It depends on the problem at hand. If the relationship between the features and the response is well approximated by a linear model […], then an approach such as linear regression will likely work well […] If instead there is a highly non-linear and complex relationship between the features ad the response […] decision trees may outperform classical approaches.” (pag. 314 James et al. 2013)

“Our course, other considerations beyond simply test error may come into play in selecting a statistical learning method; for instance, in certain settings, prediction using a tree may be preferred for the sake of ***interpretability*** and visualization” (pag. 314, James et al. 2013)



ELSEVIER

Artificial Intelligence

Volume 267, February 2019, Pages 1-38

Explanation in artificial intelligence: Insights from the social sciences

Tim Miller

Show more

+ Add to Mendeley Share Cite

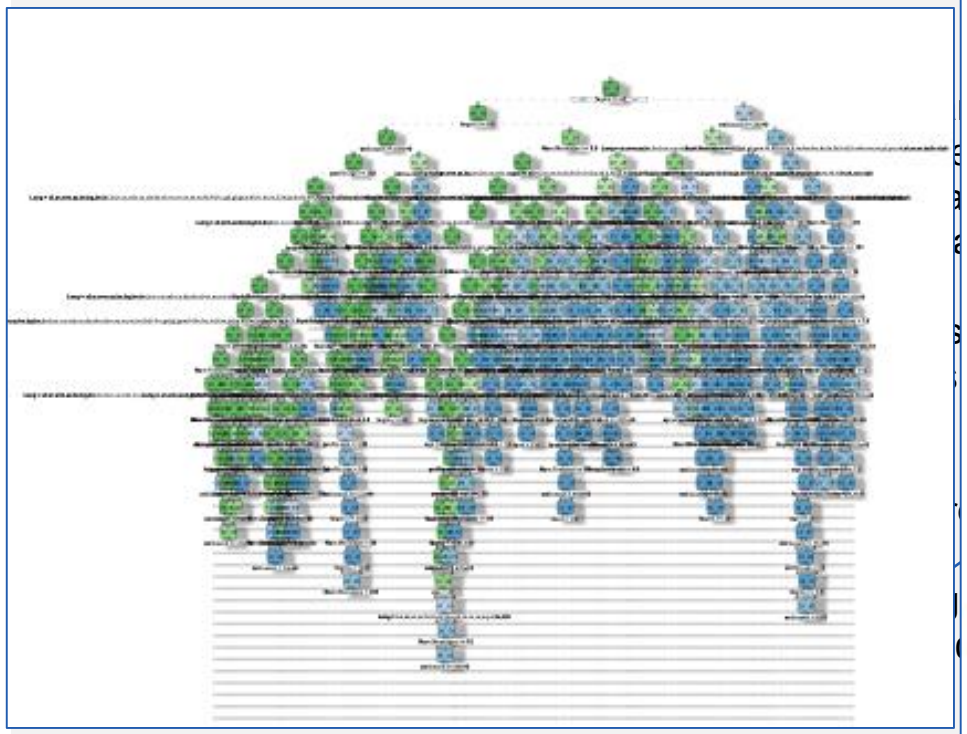
https://doi.org/10.1016/j.artint.2018.07.007 Get rights and content

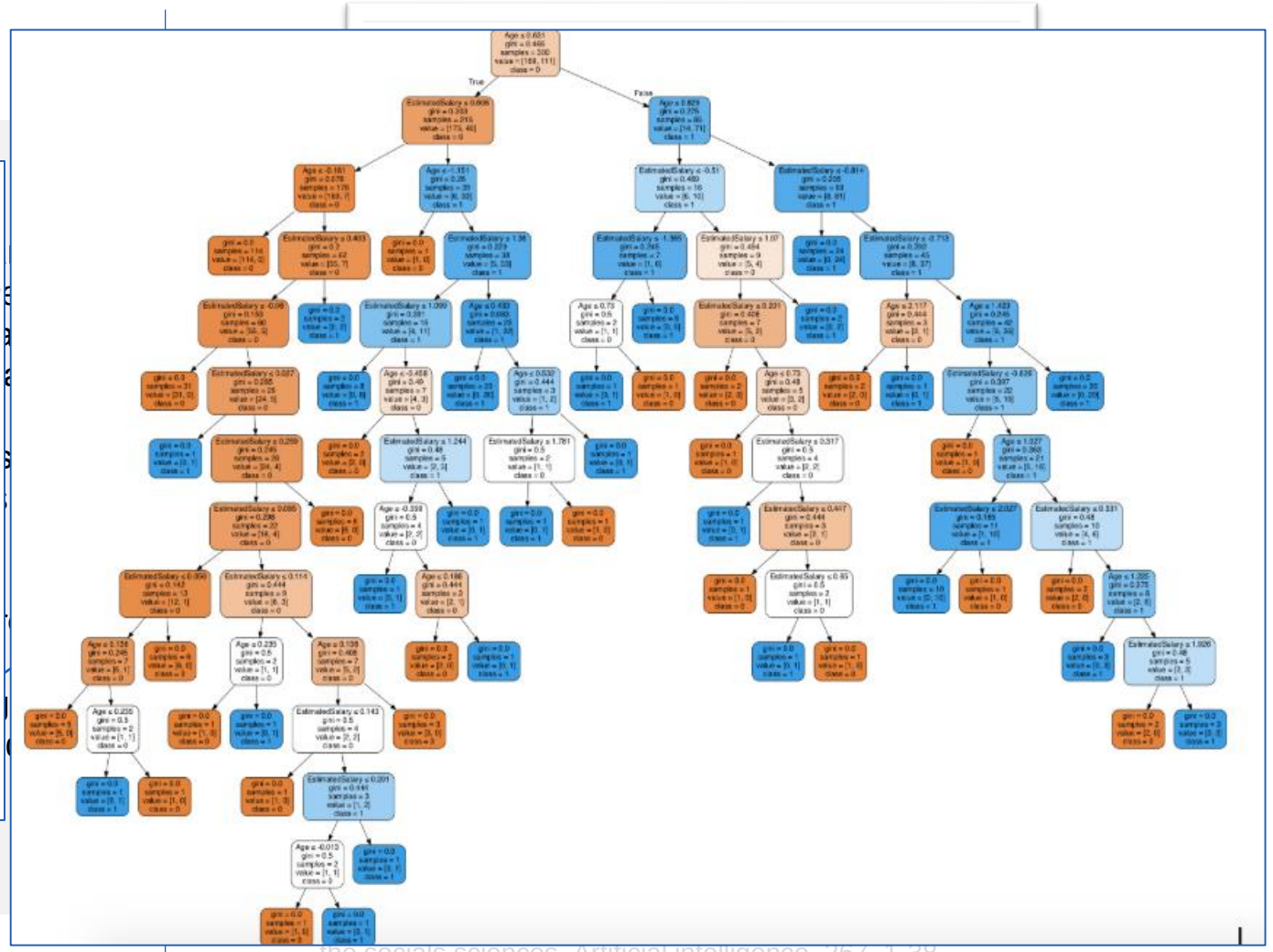
Abstract

There has been a recent resurgence in the area of explainable artificial intelligence as researchers and practitioners seek to provide more transparency to their algorithms. Much of this research is focused on explicitly explaining decisions or actions to a human observer, and it should not be controversial to say that looking at how humans explain to each other can serve as a useful starting point for explanation in artificial intelligence. However, it is fair to say that most work in explainable artificial intelligence uses only the researchers' intuition of what constitutes a 'good' explanation. There exist vast and valuable bodies of research in philosophy, psychology, and cognitive science of how people define, generate, select, evaluate, and present explanations, which argues that people employ certain cognitive biases and social expectations to the explanation process. This paper argues that the field of explainable artificial intelligence can build on this existing research, and reviews relevant papers from philosophy, cognitive psychology/science, and social psychology, which study these topics. It draws out some important findings, and discusses ways that these can be infused with work on explainable artificial intelligence.

Miller, T. (2019). Explanation in artificial intelligence: Insights from the socials sciences. Artificial intelligence, 267, 1-38.

Are decision trees really interpretable?





the socials sciences. Artificial intelligence, 267, 1-38.

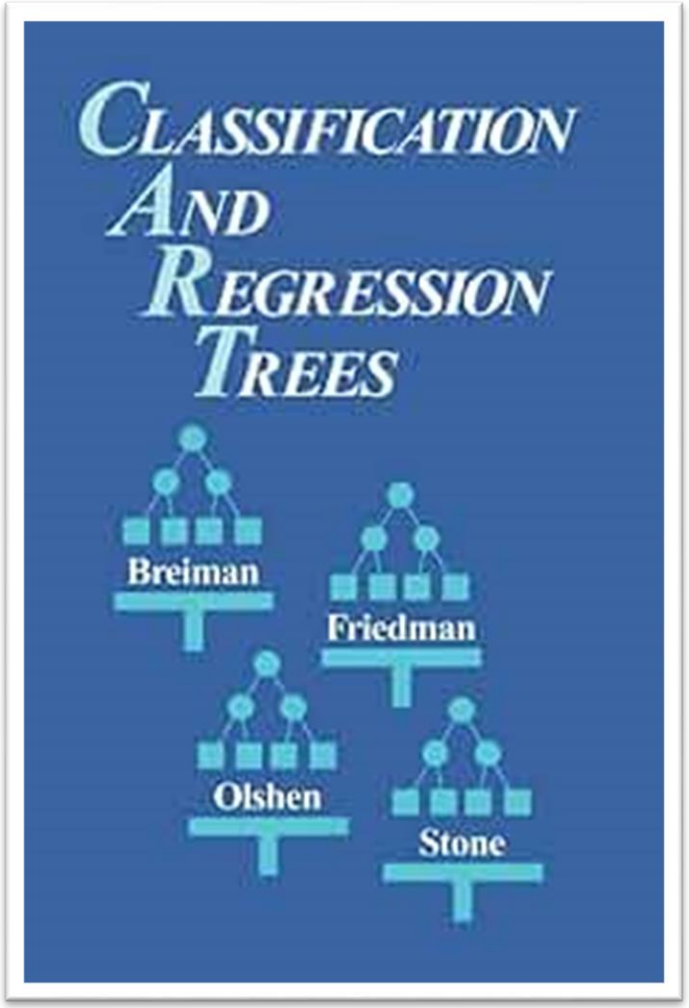
Training Decision Trees

Breiman et al.’s Classification And Regression Tree (CART) algorithm

Where it all began – the first example of classification tree from Breiman et al.’s (1984) book

At the University of California, San Diego Medical Center, when a heart attack patient is admitted, 19 variables are measured during the first 24 hours. These include blood pressure, age, and 17 other ordered and binary variables summarizing the medical symptoms considered as important indicators of the patient's condition.

The goal of a recent medical study (see Chapter 6) was the development of a method to identify high risk patients (those who will not survive at least 30 days) on the basis of the initial 24-hour data.



Breiman, L., Friedman, J.H., Olshen, R.A., Stone, C.J. (1984). Classification and Regression Trees. Wadsworth Statistics/Probability Series.

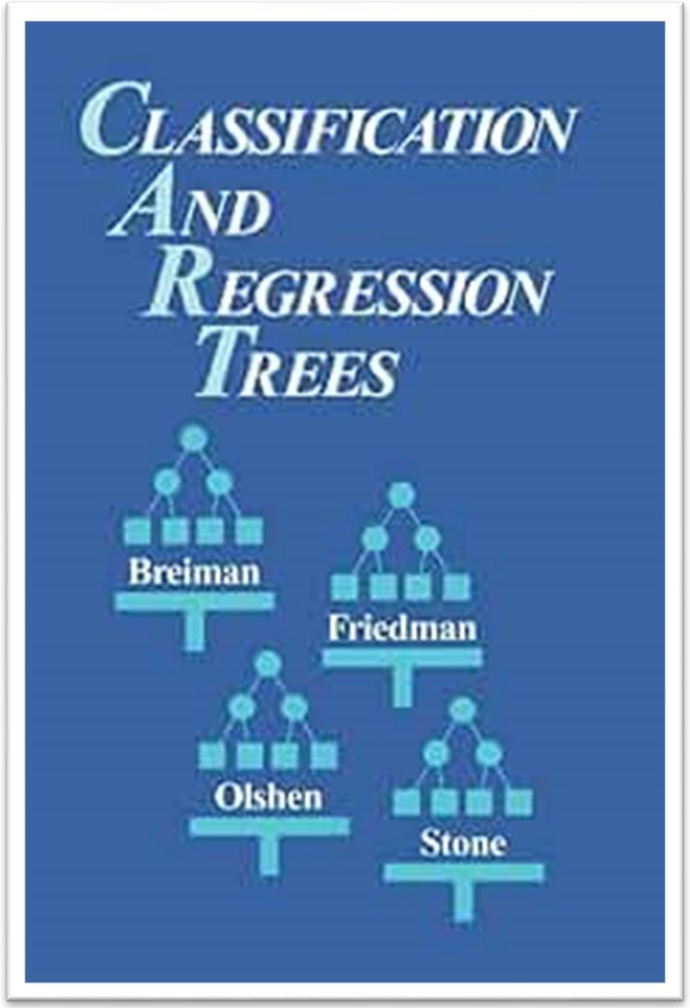
Breiman et al.'s Classification And Regression Tree (CART) algorithm

Where it all began – the first example of classification tree from Breiman et al.'s (1984) book

At the University of California, San Diego Medical Center, when a heart attack patient is admitted, 19 variables are measured during the first 24 hours. These include blood pressure, age, and 17 other ordered and binary variables summarizing the medical symptoms considered as important indicators of the patient's condition.

The goal of a recent medical study (see Chapter 6) was the development of a method to identify high risk patients (those who will not survive at least 30 days) on the basis of the initial 24-hour data.

This rule classifies incoming patients as *F* or *G* depending on the yes-no answers to at most three questions.



Breiman, L., Friedman, J.H., Olshen, R.A., Stone, C.J. (1984). Classification and Regression Trees. Wadsworth Statistics/Probability Series.

Breiman et al.'s Classification And Regression Tree (CART) algorithm

Where it all began – the first example of classification tree from Breiman et al.'s (1984) book

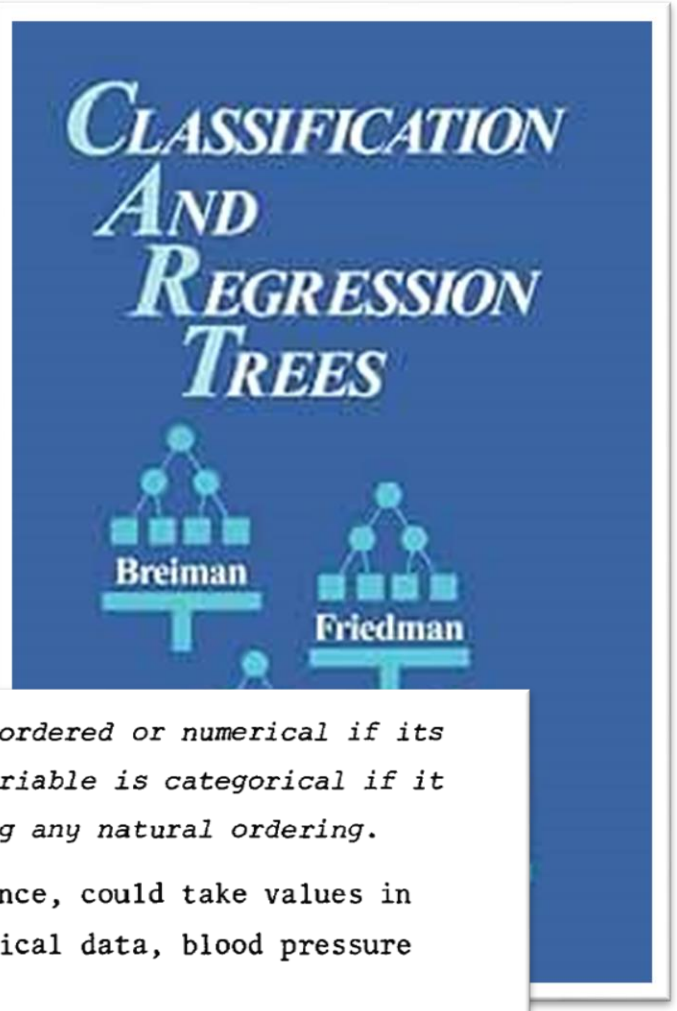
At the University of California, San Diego Medical Center, when a heart attack patient is admitted, 19 variables are measured during the first 24 hours. These include blood pressure, age, and 17 other ordered and binary variables summarizing the medical symptoms considered as important indicators of the patient's condition.

The goal of a recent medical study (see Chapter 6) was the development of a method to identify high risk patients (those who will not survive at least 30 days) on the basis of the initial 24-hour data.

This rule classifies incomi
the yes-no answers to at most

DEFINITION 1.4. *A variable is called ordered or numerical if its measured values are real numbers. A variable is categorical if it takes values in a finite set not having any natural ordering.*

A categorical variable, for instance, could take values in the set {red, blue, green}. In the medical data, blood pressure and age are ordered variables.



Breiman, L., Friedman, J.H., Olshen, R.A., Stone, C.J. (1984). Classification and Regression Trees. Wadsworth Statistics/Probability Series.

Breiman et al.'s Classification And Regression Tree (CART) algorithm

Where it all began – the first example of classification tree from Breiman et al.'s (1984) book

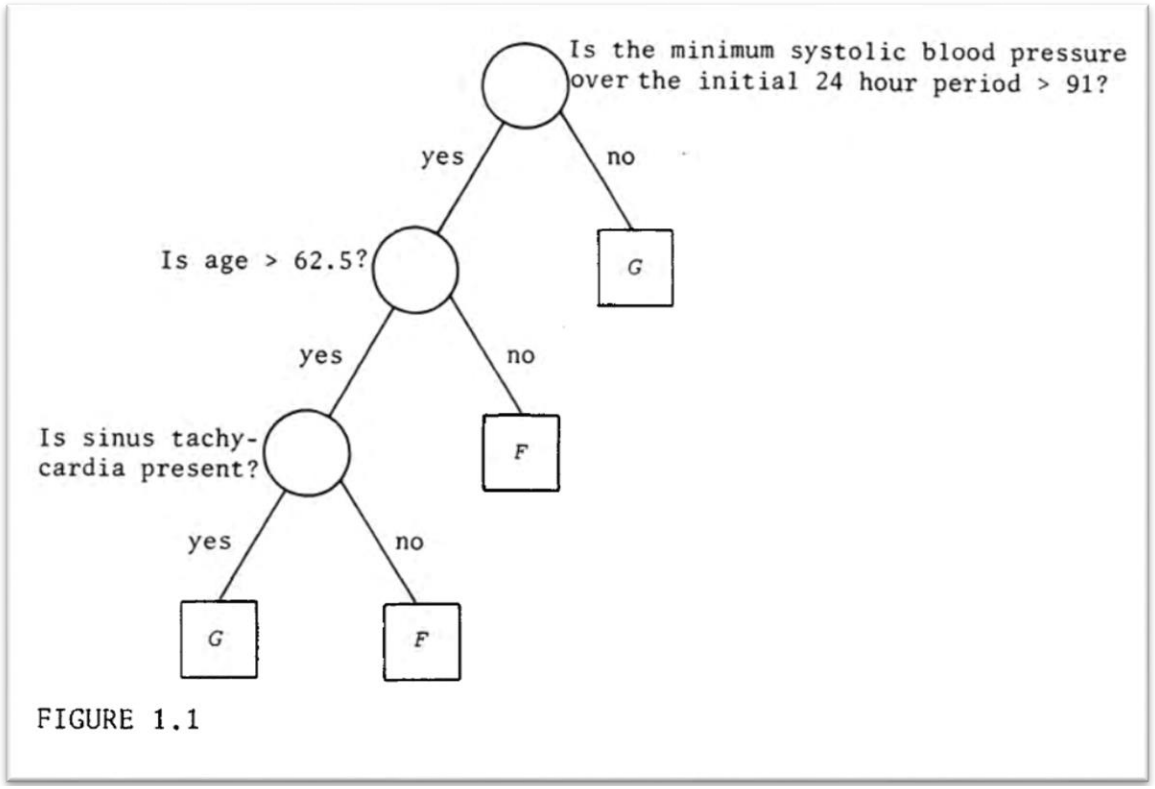
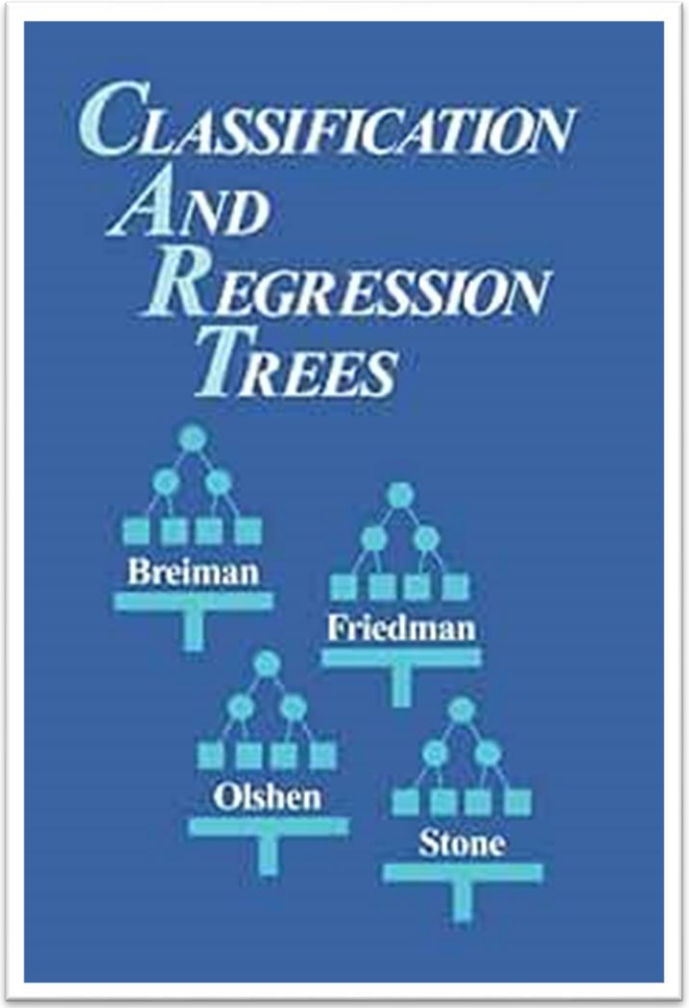


FIGURE 1.1

Figure 1.1 is a picture of the tree structured classification rule that was produced in the study. The letter *F* means not high risk; *G* means high risk.



Breiman, L., Friedman, J.H., Olshen, R.A., Stone, C.J. (1984). Classification and Regression Trees. Wadsworth Statistics/Probability Series.

As previously mentioned, to describe the training of decision trees we need to discuss three key specifications

I **Splitting criterion** – "*how to grow the tree?*"

II **Stopping criterion** – "*when is the tree grown enough?*"

III **Rule to assigning values to all leaves** – "*how to compute predictions?*"

Splitting criterion – "*how to grow the tree?*"
General idea

Data

$$\{(x_i, y_i)\}_{i=1,\dots,N}$$

$$x_i = (x_{i1}, \dots, x_{ip}) \in \mathcal{X}$$

$$y_i \in \mathcal{Y}, \text{ e.g., } \mathcal{Y} = \{0, 1\} \text{ or } \mathcal{Y} = \mathbb{R}$$

Which feature to split? (Remember: we use only one feature per split)

How to split a feature? How many splits to consider?

To grow the tree, we need to introduce a logic to select the "best" split among all **possible splits (or "cut-off" values)** of all **features/variables/predictors**, at each step

Splitting criterion – "*how to grow the tree?*"

REGRESSION

Regression tree – we split minimizing the sum of the Residual Sum of Squares (RSS)

Data

$$\{(x_i, y_i)\}_{i=1,\dots,N}$$

$$x_i = (x_{i1}, \dots, x_{ip}) \in \mathcal{X}$$

$$y_i \in \mathbb{R}$$

Goal

Find $\begin{cases} j \in \{1, \dots, p\} \\ s \text{ split of the } j\text{-th feature} \end{cases}$ minimizing the sum:

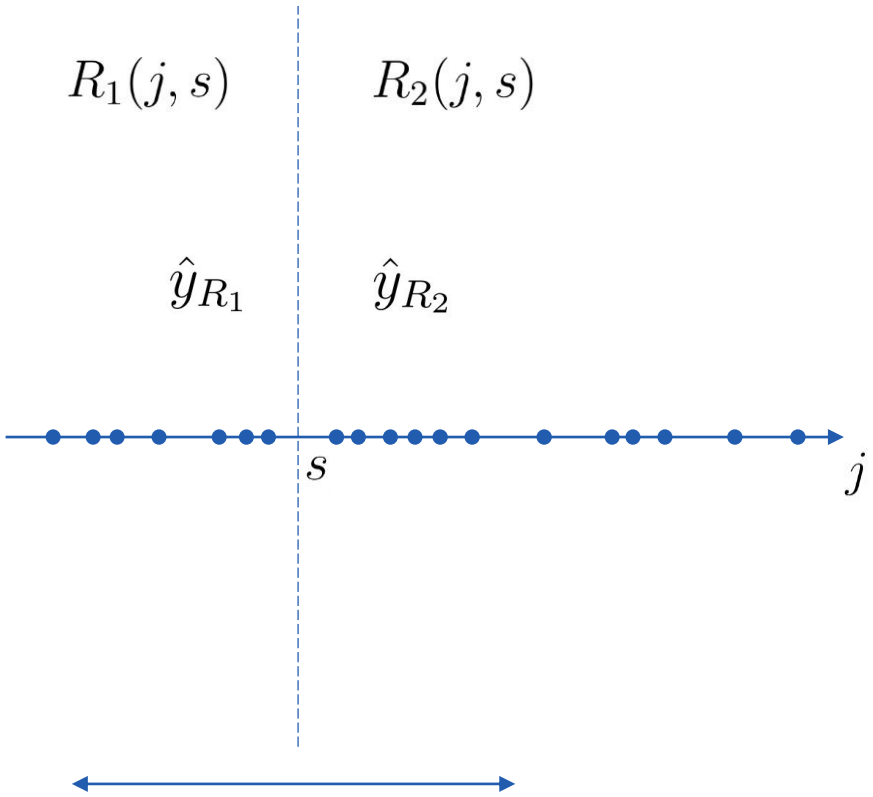
$$\sum_{i:x_i \in R_1(j,s)} (y_i - \hat{y}_{R_1})^2 + \sum_{i:x_i \in R_2(j,s)} (y_i - \hat{y}_{R_2})^2$$

RSS $R_1(j, s)$ RSS $R_2(j, s)$

Splitting criterion – "*how to grow the tree?*"

Regression tree

REGRESSION



"scan" all possible splits s of the feature j

3 Average y-value of the data points in $R_2(j,s)$

4 Average y-value of the data points in $R_1(j,s)$

Find $\begin{cases} j \in \{1,\dots,p\} \\ s \text{ split of the } j\text{-th feature} \end{cases}$ minimizing the sum:

$$\sum_{i:x_i \in R_1(j,s)} (y_i - \hat{y}_{R_1})^2 + \sum_{i:x_i \in R_2(j,s)} (y_i - \hat{y}_{R_2})^2$$

RSS $R_1(j,s)$

RSS $R_2(j,s)$

2 $R_2(j,s) = \{x_i \in \mathcal{X} | x_{ij} \geq s\}$

right half-plane

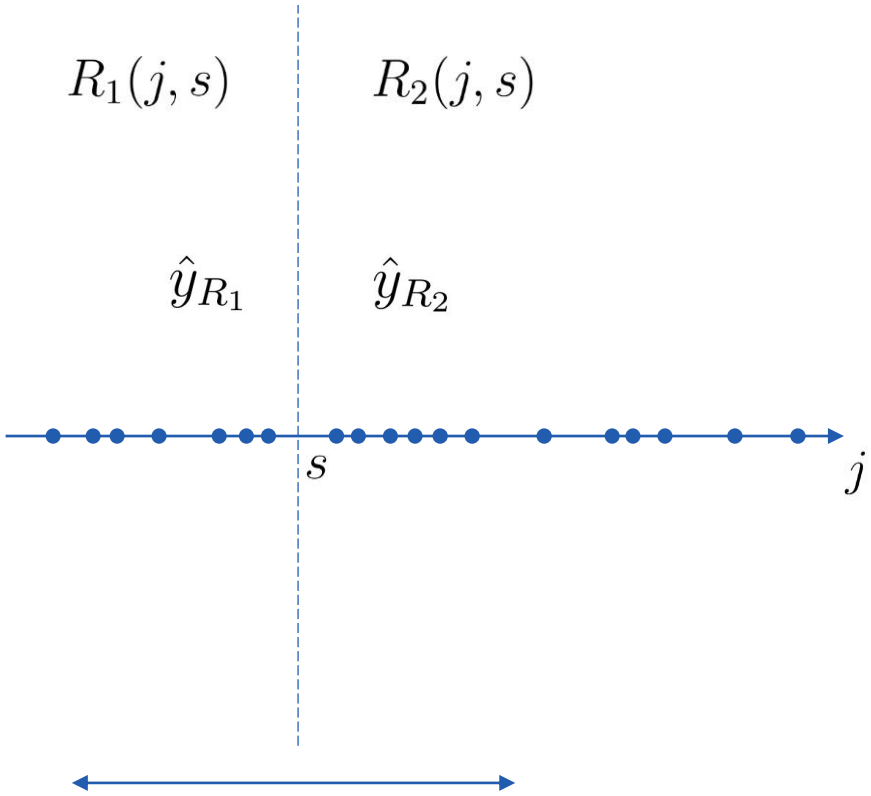
1 $R_1(j,s) = \{x_i \in \mathcal{X} | x_{ij} < s\}$

left half-plane

Splitting criterion – "*how to grow the tree?*"

Regression tree – how to determine the splits?

REGRESSION



"scan" all possible splits s of the feature j

s ?

Ordered feature: consider all mid-points of consecutive values
E.g., *j*-th feature=…,3,4… then s = 3.5 etc.

Categorical feature: use a "*supervised encoding*" called **target encoding**. With target encording, we replace each category with the average value of the target variable for that category.

decision: $X_{\bar{j}} < \bar{s}$ *(and more forward)*

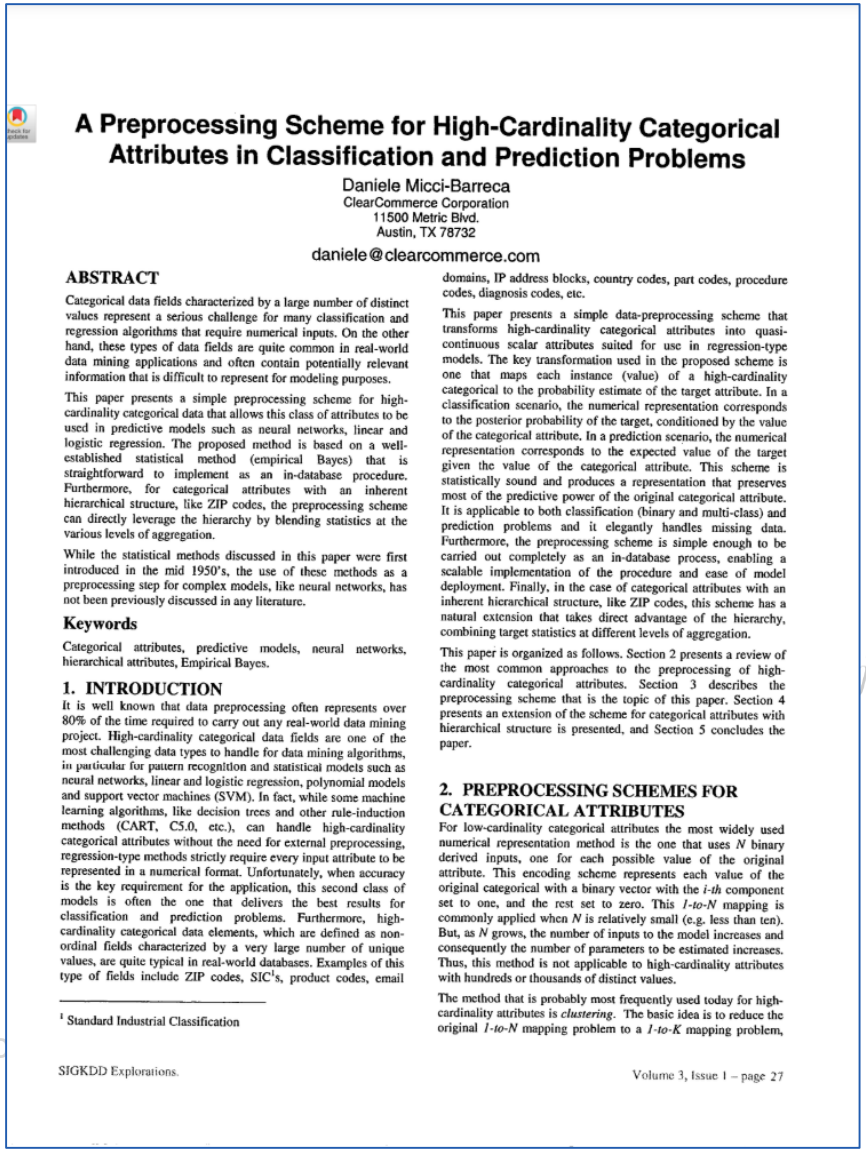
(best split)

(selected feature)

Splitting criterion – "*how to grow the tree?*"

Regression tree – how to determine the splits?

REGRESSION



A Preprocessing Scheme for High-Cardinality Categorical Attributes in Classification and Prediction Problems

Daniele Micci-Barreca
ClearCommerce Corporation
11500 Metric Blvd.
Austin, TX 78732
daniele@clearcommerce.com

ABSTRACT

Categorical data fields characterized by a large number of distinct values represent a serious challenge for many classification and regression algorithms that require numerical inputs. On the other hand, these types of data fields are quite common in real-world data mining applications and often contain potentially relevant information that is difficult to represent for modeling purposes.

This paper presents a simple preprocessing scheme for high-cardinality categorical data that allows this class of attributes to be used in predictive models such as neural networks, linear and logistic regression. The proposed method is based on a well-established statistical method (empirical Bayes) that is straightforward to implement as an in-database procedure. Furthermore, for categorical attributes with an inherent hierarchical structure, like ZIP codes, the preprocessing scheme can directly leverage the hierarchy by blending statistics at the various levels of aggregation.

While the statistical methods discussed in this paper were first introduced in the mid 1950's, the use of these methods as a preprocessing step for complex models, like neural networks, has not been previously discussed in any literature.

Keywords

Categorical attributes, predictive models, neural networks, hierarchical attributes, Empirical Bayes.

1. INTRODUCTION

It is well known that data preprocessing often represents over 80% of the time required to carry out any real-world data mining project. High-cardinality categorical data fields are one of the most challenging data types to handle for data mining algorithms, in particular for pattern recognition and statistical models such as neural networks, linear and logistic regression, polynomial models and support vector machines (SVM). In fact, while some machine learning algorithms, like decision trees and other rule-induction methods (CART, C5.0, etc.), can handle high-cardinality categorical attributes without the need for external preprocessing, regression-type methods strictly require every input attribute to be represented in a numerical format. Unfortunately, when accuracy is the key requirement for the application, this second class of models is often the one that delivers the best results for classification and prediction problems. Furthermore, high-cardinality categorical data elements, which are defined as non-ordinal fields characterized by a very large number of unique values, are quite typical in real-world databases. Examples of this type of fields include ZIP codes, SIC[1]s, product codes, email domains, IP address blocks, country codes, part codes, procedure codes, diagnosis codes, etc.

This paper presents a simple data-preprocessing scheme that transforms high-cardinality categorical attributes into quasi-continuous scalar attributes suited for use in regression-type models. The key transformation used in the proposed scheme is one that maps each instance (value) of a high-cardinality categorical to the probability estimate of the target attribute. In a classification scenario, the numerical representation corresponds to the posterior probability of the target, conditioned by the value of the categorical attribute. In a prediction scenario, the numerical representation corresponds to the expected value of the target given the value of the categorical attribute. This scheme is statistically sound and produces a representation that preserves most of the predictive power of the original categorical attribute. It is applicable to both classification (binary and multi-class) and prediction problems and it elegantly handles missing data. Furthermore, the preprocessing scheme is simple enough to be carried out completely as an in-database process, enabling a scalable implementation of the procedure and ease of model deployment. Finally, in the case of categorical attributes with an inherent hierarchical structure, like ZIP codes, this scheme has a natural extension that takes direct advantage of the hierarchy, combining target statistics at different levels of aggregation.

This paper is organized as follows. Section 2 presents a review of the most common approaches to the preprocessing of high-cardinality categorical attributes. Section 3 describes the preprocessing scheme that is the topic of this paper. Section 4 presents an extension of the scheme for categorical attributes with hierarchical structure is presented, and Section 5 concludes the paper.

2. PREPROCESSING SCHEMES FOR CATEGORICAL ATTRIBUTES

For low-cardinality categorical attributes the most widely used numerical representation method is the one that uses N binary derived inputs, one for each possible value of the original attribute. This encoding scheme represents each value of the original categorical with a binary vector with the *i-th* component set to one, and the rest set to zero. This *1-to-N* mapping is commonly applied when N is relatively small (e.g. less than ten). But, as N grows, the number of inputs to the model increases and consequently the number of parameters to be estimated increases. Thus, this method is not applicable to high-cardinality attributes with hundreds or thousands of distinct values.

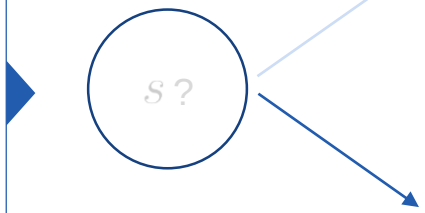
The method that is probably most frequently used today for high-cardinality attributes is *clustering*. The basic idea is to reduce the original *1-to-N* mapping problem to a *1-to-K* mapping problem,

[1] Standard Industrial Classification

sklearn.preprocessing.TargetEncoder

class `sklearn.preprocessing.TargetEncoder`(*categories='auto', target_type='auto', smooth='auto', cv=5, shuffle=True, random_state=None*) [source]

(It can be used independently of decision trees, of course. Target encoding is much more parsimonious than one-hot encoding and preferred in case of high cardinality categorical variables)



Ordered feature: consider all mid-points of consecutive values
E.g., *j*-th feature=…,3,4… then s = 3.5 etc.

Categorical feature: use a "*supervised encoding*" called **target encoding**. With target encording, we replace each category with the average value of the target variable for that category.

Micci-Barreca, D. (2001). A preprocessing scheme for high-cardinality categorical attributes in classification and prediction problems. *ACM SIGKDD explorations newsletter*, *3*(1), 27-32.

Splitting criterion – "*how to grow the tree?*"

REGRESSION

Regr

Amazon Employee Access Challenge

Data Card | Code (5) | Discussion (1) | Suggestions (0)

About Dataset

Context

When an employee at any company starts work, they first need to obtain the computer access necessary to fulfill their role. This access may allow an employee to read/manipulate resources through various applications or web portals. It is assumed that employees fulfilling the functions of a given role will access the same or similar resources. It is often the case that employees figure out the access they need as they encounter roadblocks during their daily work (e.g. not able to log into a reporting portal). A knowledgeable supervisor then takes time to manually grant the needed access in order to overcome access obstacles. As employees move throughout a company, this access discovery/recovery cycle wastes a nontrivial amount of time and money.

There is a considerable amount of data regarding an employee's role within an organization and the resources to which they have access. Given the data related to current employees and their provisioned access, models can be built that automatically determine access privileges as employees enter and leave roles within a company. These auto-access models seek to minimize the human involvement required to grant or revoke employee access.

Part of the competition "Amazon.com - Employee Access Challenge" (https://www.kaggle.com/c/amazon-employee-access-challenge), the data consists of real historical data collected from 2010 & 2011. Employees are manually allowed or denied access to resources over time. Your task is to create an algorithm capable of learning from this historical data to predict approval/denial for an unseen set of employees.

Usability ⓘ
7.65

License
CC0: Public Domain

Expected update frequency
Never

Tags

Tabular

!

https://www.kaggle.com/datasets/lucamassaron/amazon-employee-access-challenge

SIGKDD explorations newsletter, *3*(1), 27-32.

Splitting criterion – "*how to grow the tree?*"

Regr

REGRESSION

Amazon Employee Acces

Partners

This competition is hosted in collaboration with the **IEEE International Workshop on Machine Learning for Signal Processing (MLSP 2013)**

IEEE Signal Processing Society

IEEE

Data Card | Code (5) | Discussion (1) | Suggestions (

About Dataset

Context

When an employee at any company starts work, they first need to ... ry to fulfill their role. This access may allow an employee to read/manipulate resources through various a... med that employees fulfilling the functions of a given role will access the same or similar resources... gure out the access they need as they encounter roadblocks during their daily work (e.g. not able to log i... able supervisor then takes time to manually grant the needed access in order to overcome access ob... out a company, this access discovery/recovery cycle wastes a nontrivial amount of time and n...

There is a considerable amount of data regarding an employee's ro... ources to which they have access. Given the data related to current employees and their provisioned access, models can be built that automatically determine access privileges as employees enter and leave roles within a company. These auto-access models seek to minimize the human involvement required to grant or revoke employee access.

Part of the competition "Amazon.com - Employee Access Challenge" (https://www.kaggle.com/c/amazon-employee-access-challenge), the data consists of real historical data collected from 2010 & 2011. Employees are manually allowed or denied access to resources over time. Your task is to create an algorithm capable of learning from this historical data to predict approval/denial for an unseen set of employees.

Prizes & Awards

$5,000

Awards Points & Medals

Participation

2,073 Entrants

1,839 Participants

1,686 Teams

16,870 Submissions

Usability ⓘ

7.65

License

CC0: Public Domain

Expected update frequency

Never

Tags

Tabular

https://www.kaggle.com/datasets/lucamassaron/amazon-employee-access-challenge

SIGKDD explorations newsletter, *3*(1), 27-32.

Splitting criterion – "*how to grow the tree?*"

Regr

REGRESSION

Amazon Employee Access Challenge

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32769 entries, 0 to 32768
Data columns (total 10 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   ACTION            32769 non-null  object
 1   RESOURCE          32769 non-null  object
 2   MGR_ID            32769 non-null  object
 3   ROLE_ROLLUP_1     32769 non-null  object
 4   ROLE_ROLLUP_2     32769 non-null  object
 5   ROLE_DEPTNAME     32769 non-null  object
 6   ROLE_TITLE        32769 non-null  object
 7   ROLE_FAMILY_DESC  32769 non-null  object
 8   ROLE_FAMILY       32769 non-null  object
 9   ROLE_CODE         32769 non-null  object
```

Column Name	Description
ACTION	ACTION is 1 if the resource was approved, 0 if the resource was not
RESOURCE	An ID for each resource
MGR_ID	The EMPLOYEE ID of the manager of the current EMPLOYEE ID record; an employee may have only one manager at a time
ROLE_ROLLUP_1	Company role grouping category id 1 (e.g. US Engineering)
ROLE_ROLLUP_2	Company role grouping category id 2 (e.g. US Retail)
ROLE_DEPTNAME	Company role department description (e.g. Retail)
ROLE_TITLE	Company role business title description (e.g. Senior Engineering Retail Manager)
ROLE_FAMILY_DESC	Company role family extended description (e.g. Retail Manager, Software Engineering)
ROLE_FAMILY	Company role family description (e.g. Retail Manager)
ROLE_CODE	Company role code; this code is unique to each role (e.g. Manager)

https://www.kaggle.com/datasets/lucamassaron/amazon-employee-access-challenge

SIGKDD explorations newsletter, *3*(1), 27-32.

Splitting criterion – "*how to grow the tree?*"

Regr

REGRESSION

Amazon Employee Access Challenge

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32769 entries, 0 to 32768
Data columns (total 10 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   ACTION            32769 non-null  object
 1   RESOURCE          32769 non-null  object
 2   MGR_ID            32769 non-null  object
 3   ROLE_ROLLUP_1     32769 non-null  object
 4   ROLE_ROLLUP_2     32769 non-null  object
 5   ROLE_DEPTNAME     32769 non-null  object
 6   ROLE_TITLE        32769 non-null  object
 7   ROLE_FAMILY_DESC  32769 non-null  object
 8   ROLE_FAMILY       32769 non-null  object
 9   ROLE_CODE         32769 non-null  object
```

```
Number of unique values in each column:
ACTION              2
RESOURCE         7518
MGR_ID           4243
ROLE_ROLLUP_1     128
ROLE_ROLLUP_2     177
ROLE_DEPTNAME     449
ROLE_TITLE        343
ROLE_FAMILY_DESC 2358
ROLE_FAMILY        67
ROLE_CODE         343
dtype: int64
```

There is a considerable amount of data regarding an employee's role within an organization and the resources to which they have access. Given the data related to current employees and their provisioned access, models can be built that automatically determine access privileges as employees enter and leave roles within a company. These auto-access models seek to minimize the human involvement required to grant or revoke employee access.

Part of the competition "Amazon.com - Employee Access Challenge" (https://www.kaggle.com/c/amazon-employee-access-challenge), the data consists of real historical data collected from 2010 & 2011. Employees are manually allowed or denied access to resources over time. Your task is to create an algorithm capable of learning from this historical data to predict approval/denial for an unseen set of employees.

https://www.kaggle.com/datasets/lucamassaron/amazon-employee-access-challenge

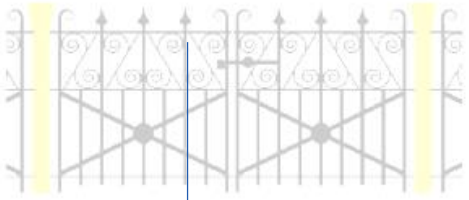
SIGKDD explorations newsletter, *3*(1), 27-32.

Splitting criterion – "*how to grow the tree?*"

Regr

REGRESSION

Amazon Employee Access Challenge



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32769 entries, 0 to 32768
Data columns (total 10 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   ACTION            32769 non-null  object
 1   RESOURCE          32769 non-null  object
 2   MGR_ID            32769 non-null  object
 3   ROLE_ROLLUP_1     32769 non-null  object
 4   ROLE_ROLLUP_2     32769 non-null  object
 5   ROLE_DEPTNAME     32769 non-null  object
 6   ROLE_TITLE        32769 non-null  object
 7   ROLE_FAMILY_DESC  32769 non-null  object
 8   ROLE_FAMILY       32769 non-null  object
 9   ROLE_CODE         32769 non-null  object
```

```
Number of unique values in each column:
ACTION              2
RESOURCE         7518
MGR_ID           4243
ROLE_ROLLUP_1     128
ROLE_ROLLUP_2     177
ROLE_DEPTNAME     449
ROLE_TITLE        343
ROLE_FAMILY_DESC 2358
ROLE_FAMILY        67
ROLE_CODE         343
dtype: int64
```

one-hot encoding

```
(32769, 15618)
```

https://www.kaggle.com/datasets/lucamassaron/amazon-employee-access-challenge

SIGKDD explorations newsletter, *3*(1), 27-32.

Splitting criterion – "*how to grow the tree?*"

Regr

REGRESSION

#	△	Team	Members	Score	Entries	Last	Solution
1	▲ 2	Paul Duan & BS Man		0.92360	122	11y	

PAUL DUAN · 1ST IN THIS COMPETITION · POSTED 11 YEARS AGO

Winning solution code and methodology

Hi everyone,

I just released the code for our solution on Github. It is published under a MIT License:

https://github.com/pyduan/amazonaccess

I did some refactoring but more may be needed. In particular, I have pretty much left Ben's code as is (in BSMan/) and only made minor modifications to files that I have integrated into my own code but haven't written myself (in external/). I'll do some additional cleaning if I have time.

As of now, the code that is being ran when launching classifier.py is my model (which include some of Ben's features) -- to obtain the final blend, one must run Ben's code independently then merge the predictions (which can be done by slightly modifying combine.py, which was written ad-hoc and which I haven't generalized yet). As you can see, the skeleton is basically the same as the one I used in the starter code I published.

Also, special thanks to Miroslaw, whose code I shamelessly copied to create some of my datasets.

Benjamin Solecki

Ben's approach is itself a blend of a logistic model and a mixture of tree-based models. He explained his approach for the logistic model in more detail here. As for the tree-based models, it is a combination of Random Forests, GBMs, and Extremely Randomized Trees that are grown using features based on counts and frequencies (e.g. number of resources managed by a given manager, etc.). I'll let him explain his approach further if he wishes to -- we merged comparatively late in the game (2 weeks before the deadline) so I would risk misrepresenting his train of thoughts.

https://www.kaggle.com/datasets/lucamassaron/amazon-employee-access-challenge

SIGKDD explorations newsletter, *3*(1), 27-32.

Splitting criterion – "*how to grow the tree?*"

Regression tree – how to determine the splits?

REGRESSION

In Python, the `sklearn` functions `DecisionTreeRegression()` and `DecisionTreeClassifier()` cannot handle categorical features (yet).

Not a big problem: you just need to encode them before using the algorithm (e.g., with target encoding)

sklearn.tree.DecisionTreeRegressor

class sklearn.tree.**DecisionTreeRegressor**(*, criterion='squared_error', splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0, ccp_alpha=0.0, monotonic_cst=None) [source]

sklearn.tree.DecisionTreeClassifier

class sklearn.tree.**DecisionTreeClassifier**(*, criterion='gini', splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0, class_weight=None, ccp_alpha=0.0, monotonic_cst=None) [source]

scikit-learn uses an optimized version of the CART algorithm; however, the scikit-learn implementation does not support categorical variables for now.

"scan" all possible split s of the feature j

Splitting criterion – "*how to grow the tree?*"

Classification tree

Data

$$\{(x_i, y_i)\}_{i=1,\dots,N}$$

$$x_i = (x_{i1}, \dots, x_{ip}) \in \mathcal{X}$$

$$y_i \in \{0, 1\}$$

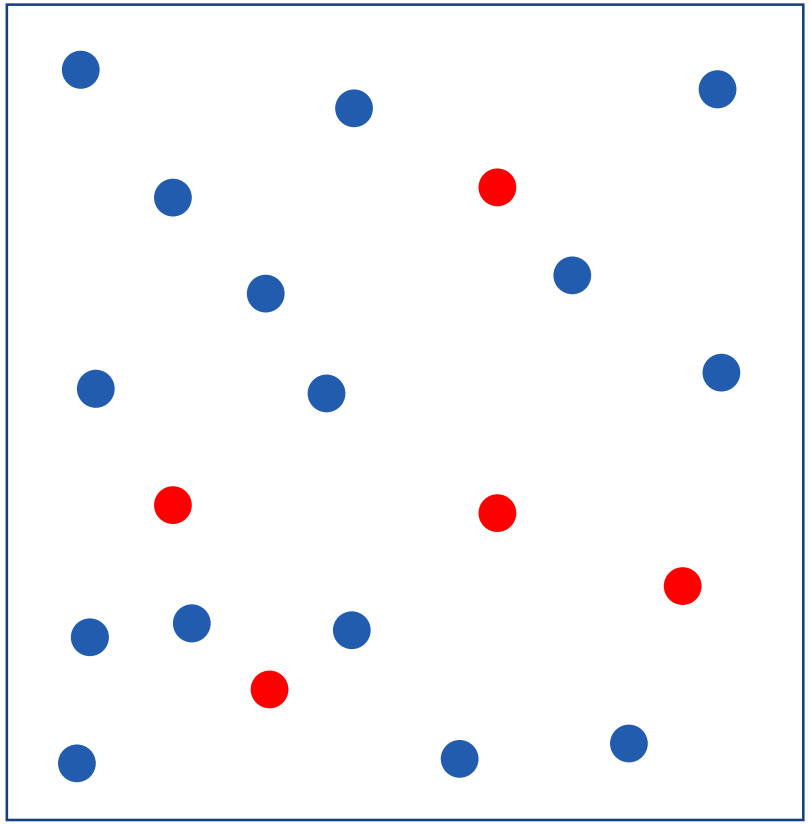
To describe how to grow a classification tree, we need to discuss the concept of **(im)purity**

Splitting criterion – "*how to grow the tree?*"

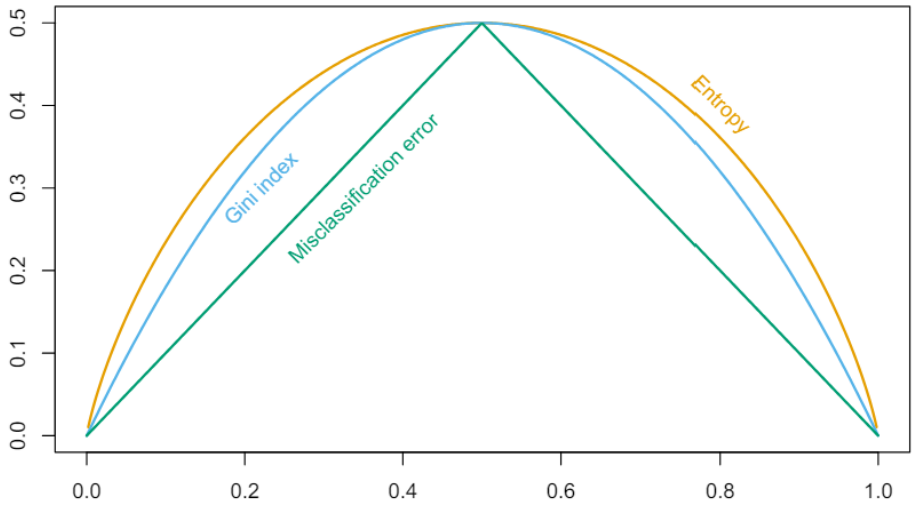
CLASSIFICATION

Classification tree – let us discuss the concept of (im)purity in a binary case

Internal node N (20 data points)



Impurity functions



(no data point is blue)

(all data points are blue)

p=p(●, N)

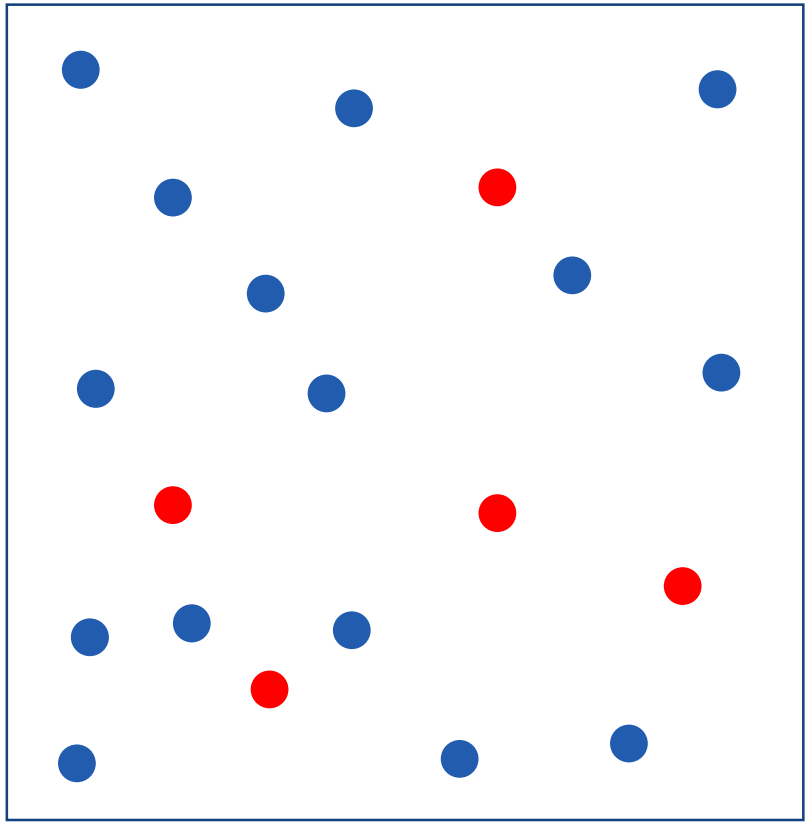
$$\text{Gini Index} = 2p(1-p)$$

$$\text{Cross-entropy or deviance} = -p\log p - (1-p)\log(1-p)$$

Splitting criterion – "*how to grow the tree?*"

Classification tree – let us discuss the concept of (im)purity in a binary case

Internal node N (20 data points)



Parent node

Children nodes

Goal

To split the node N into two nodes N_1 and N_2 that maximize the **impurity reduction** (=**purity gain**):

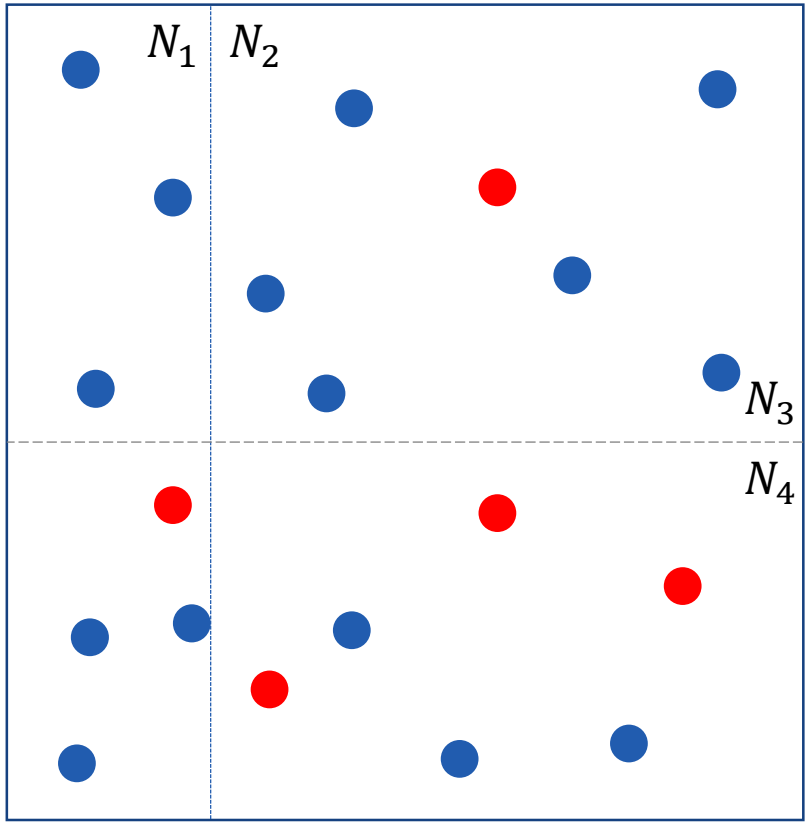
$$\Delta(N, N_1, N_2) = \text{impurity}(N) - n_1 \text{ impurity } (N_1) - n_2 \text{ impurity}(N_2)$$

where $n_* = \frac{\#\text{data in node } N_*}{\#\text{data in node } N}$ and impurity(·) can be computed using the Gini index or cross-entropy, for instance.

Splitting criterion – "*how to grow the tree?*"

Classification tree – let us discuss the concept of (im)purity in a binary case

Internal node N (20 data points)



Exercise together

Should we split N into the pair (N_1, N_2) or (N_3, N_4)?

…let us compute $\Delta(N, N_1, N_2)$ and $\Delta(N, N_3, N_4)$ using the Gini index and the cross-entropy functions.

Stopping criterion – "*when is the tree grown enough?*"

A classical example – `Hitters/Baseball` dataset

Data frame: 322 observations of major league players with 20 variables

Variables include number of times at bat in 1986, number of hits in 1986, number of years in the major league, salary in 1987 etc.

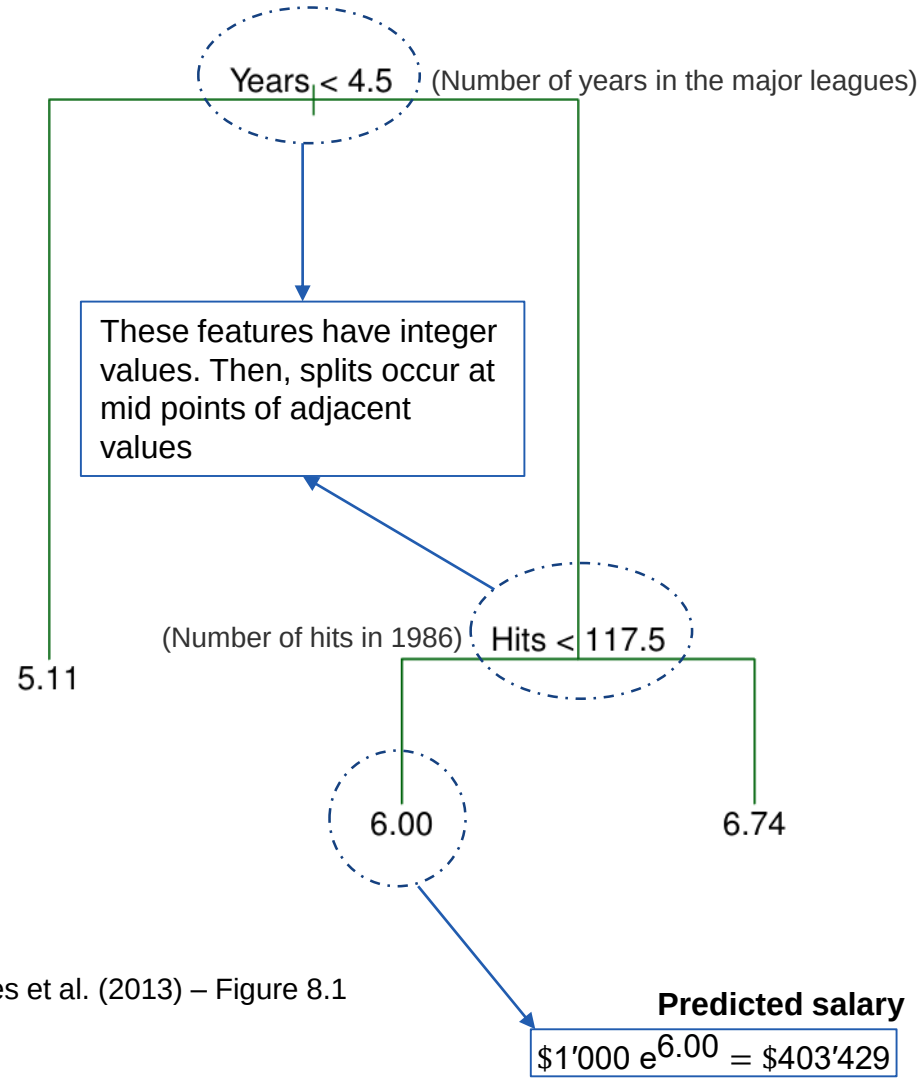
Machine learning task: to predict the 1987 salary of each player, based on their performance up to 1986 included.

Available at:
https://islp.readthedocs.io/en/latest/datasets/Hitters.html

II

Stopping criterion – "*when is the tree grown enough?*"

A classical example – `Hitters/Baseball` dataset

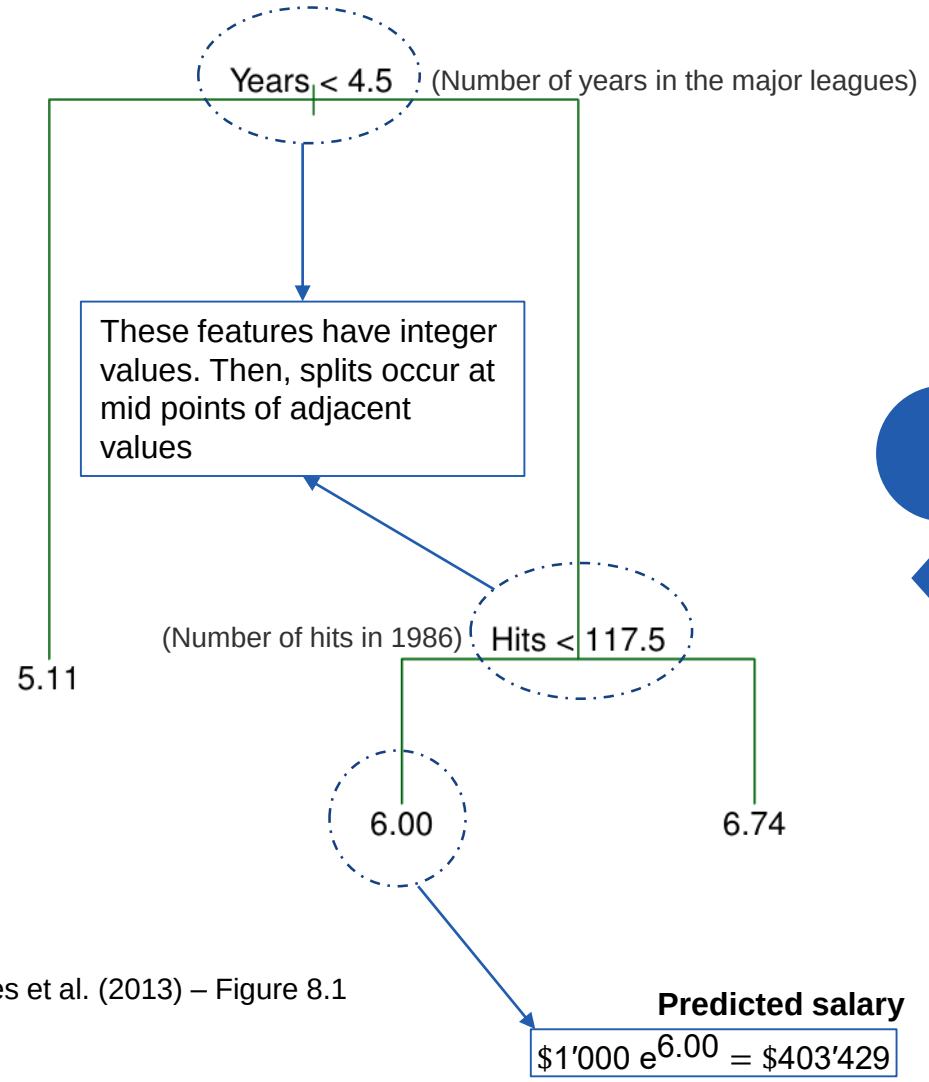


Source: James et al. (2013) – Figure 8.1

A tree of size equal to three = three terminal nodes/leaves

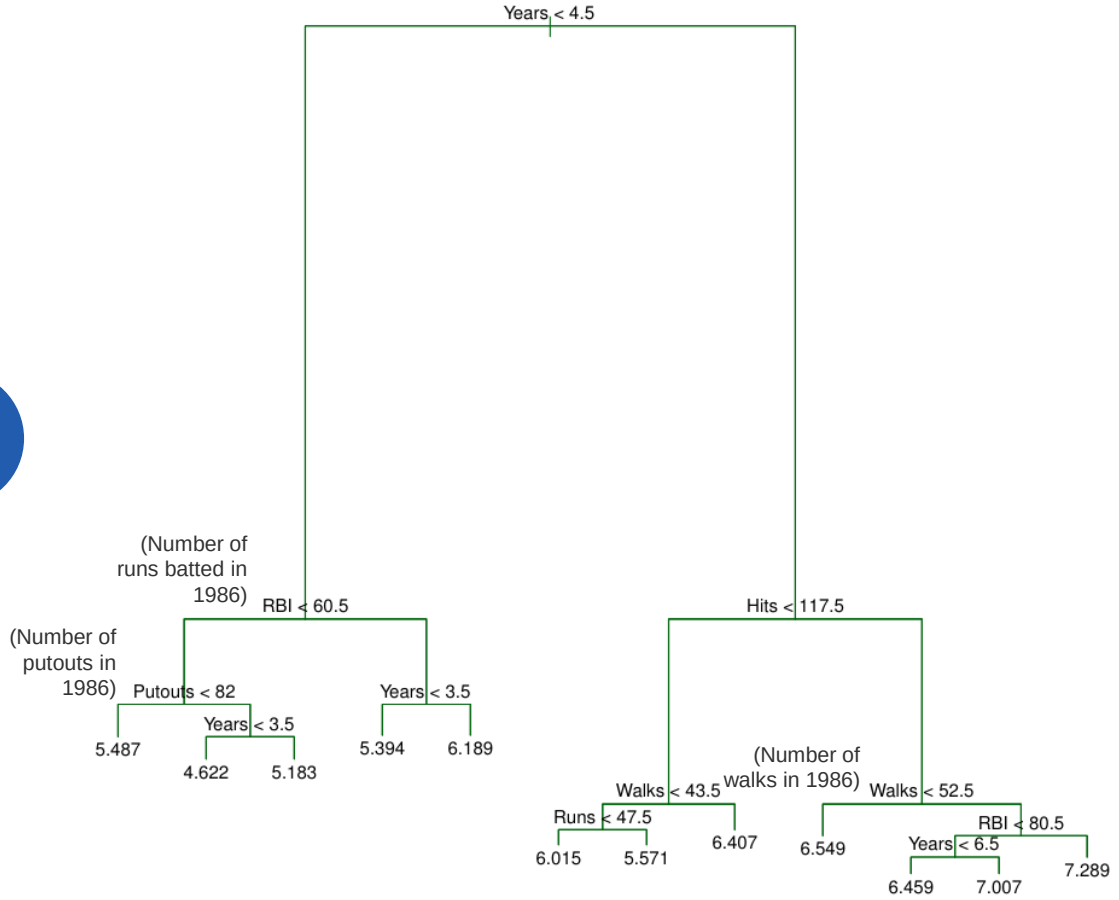
II Stopping criterion – "*when is the tree grown enough?*"

A classical example – `Hitters/Baseball` dataset



Source: James et al. (2013) – Figure 8.1

?



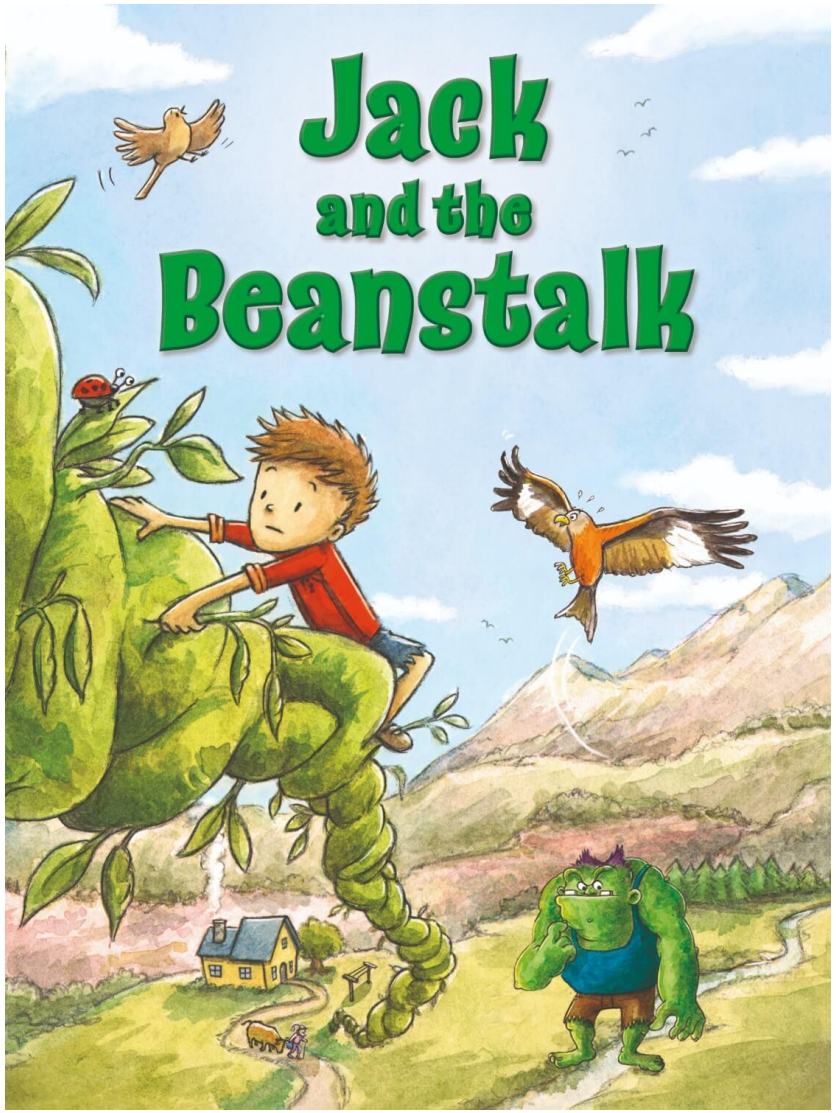
Source: James et al. (2013) – Figure 8.4

II

Stopping criterion – "*when is the tree grown enough?*"

A classical example – `Hitters/Baseball` dataset

When to stop growing a decision tree?



II

Stopping criterion – "*when is the tree grown enough?*"

A problem and two possible solutions

I

If we just keep letting the tree grow, **we are going to overfit the training data** (we may even end up with each leaf containing one observation)*

II

Strategy one: grow a very large tree, then prune it back to obtain a subtree (**pruning**)

II

Strategy two: **train ensembles of trees** and use the wisdom of the crowd to compute predictions

*standard methods to control the growth of the tree include specifying (1) max tree depth, (2) minimum number of samples to split a node/to arrive at a leaf.

II

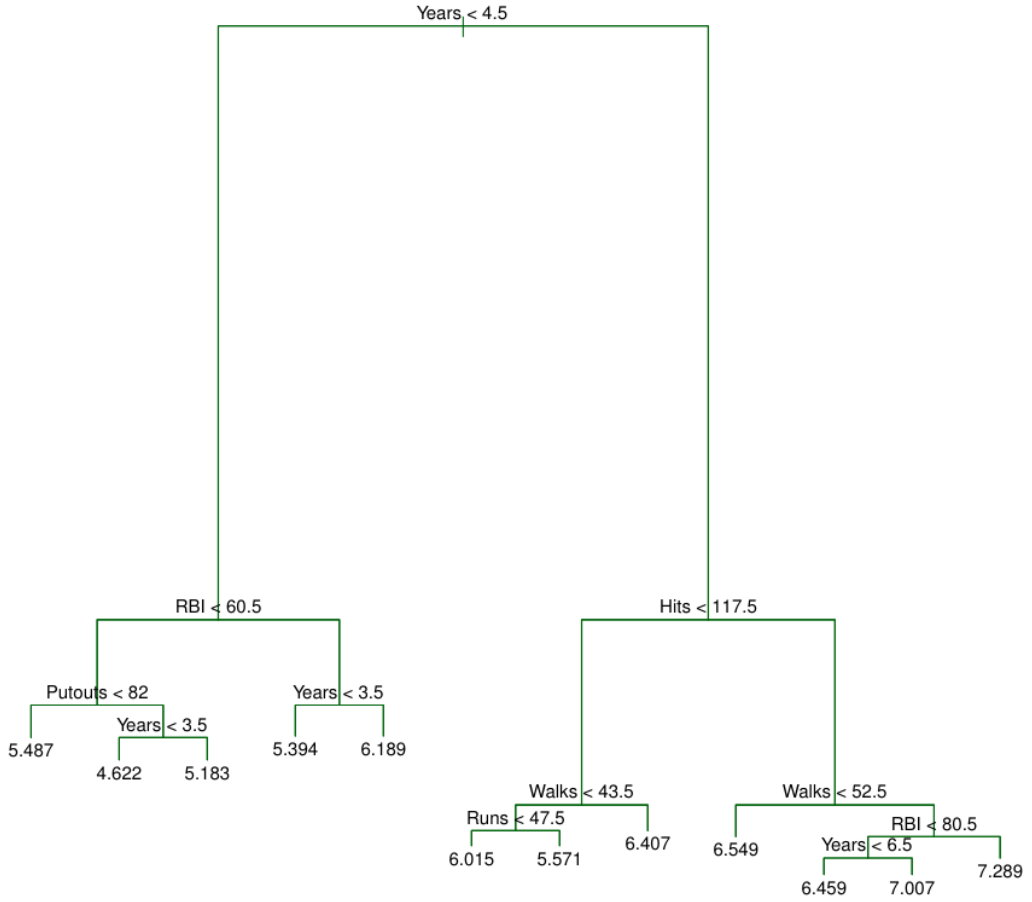
Stopping criterion – "*when is the tree grown enough?*"

Pruning decision trees: **Cost Complexity Pruning**

Our deep decision tree, again

Idea

Given a large tree, let us prune it back to **the subtree that leads to the lowest test error**



Source: James et al. (2013) – Figure 8.4.

II

Stopping criterion – "*when is the tree grown enough?*"

Pruning decision trees: **Cost Complexity Pruning**

Idea

Given a large tree, let us prune it back to **the subtree that leads to the lowest test error**

1. Consider a sequence of subtrees T_α, each parametrized by a nonnegative tuning parameter α. The original, large tree corresponds to α=0.

2. **For each** α, grow the subtree that minimizes

penalization term: $\alpha \cdot$ # leaves

$$\underbrace{\sum_{m=1}^{|T|} \sum_{i:\, x_i \in R_m} (y_i - \hat{y}_{R_m})^2}_{\text{RSS (for regression dec. trees)}} + \alpha|T|$$

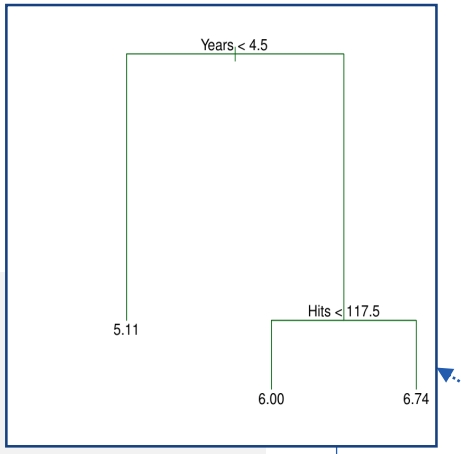
3. We select the subtree in the sequence that corresponds to the value of α that returns the lowest test error (e.g., by cross-validation). **That is our "pruned" subtree**.

II

Stopping criterion – "*when is the tree grown enough?*"

Pruning decision trees: **Cost Complexity Pruning**

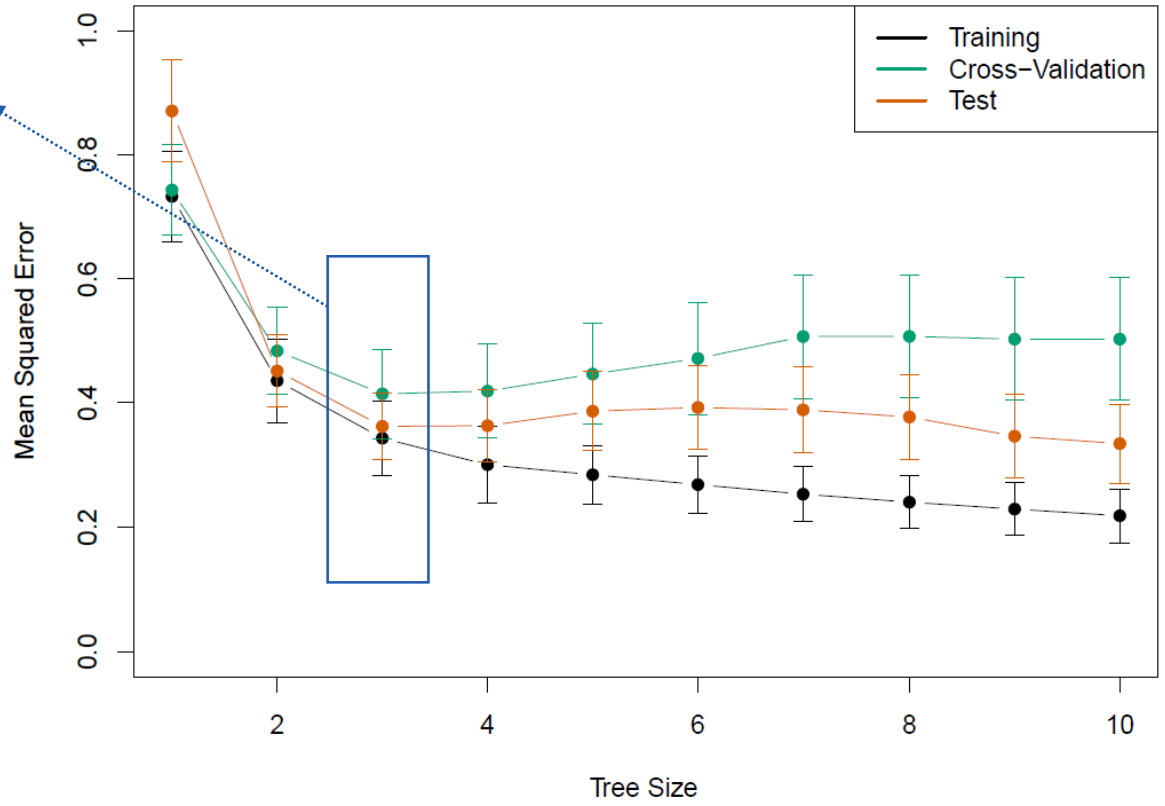
Source: James et al. (2013) – Figure 8.1



Idea

Given a large tree, let us prune it back to **the subtree that leads to the lowest test error**

(Again the `Hitters` data)



Source: James et al. (2013)

II

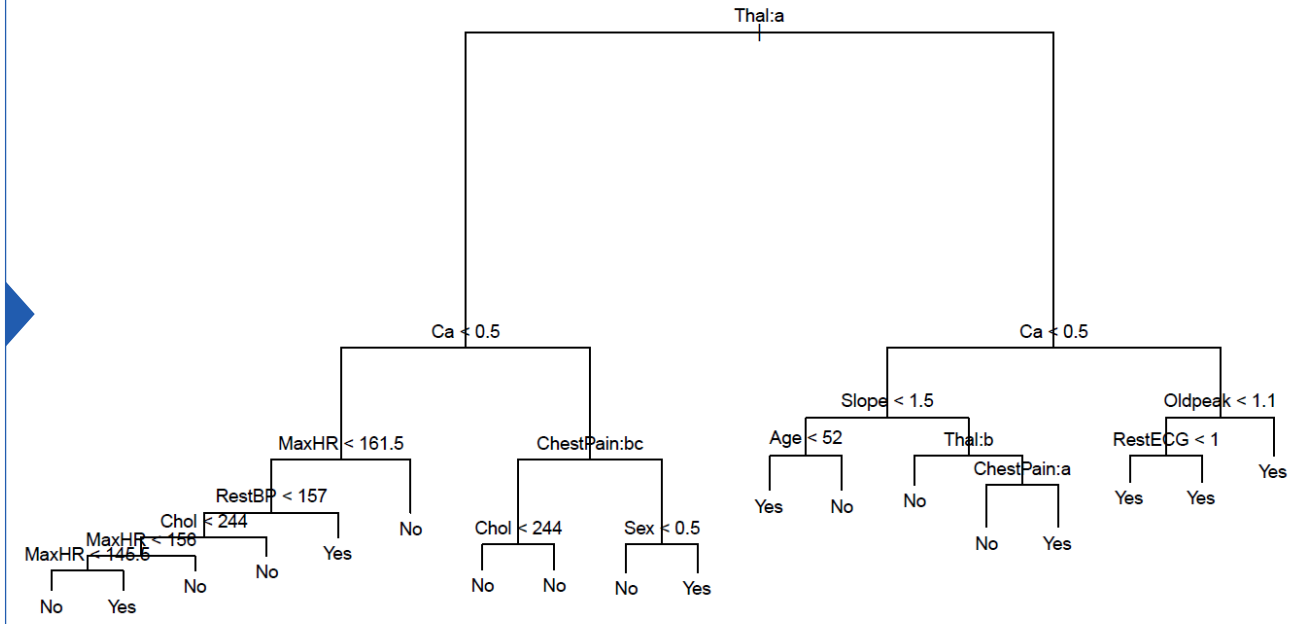
Stopping criterion – "*when is the tree grown enough?*"

Pruning decision trees: **Cost Complexity Pruning**

Idea

Given a large tree, let us prune it back to **the subtree that leads to the lowest test error**

`Heart` data – 303 patients with chest pain. Response: yes/no (heart disease). Unpruned tree.



Source: James et al. (2013)

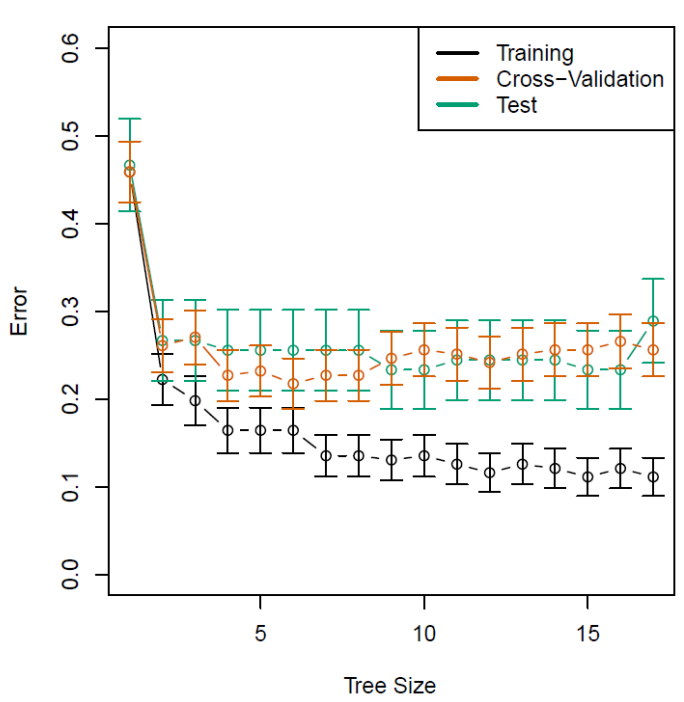
Stopping criterion – "*when is the tree grown enough?*"

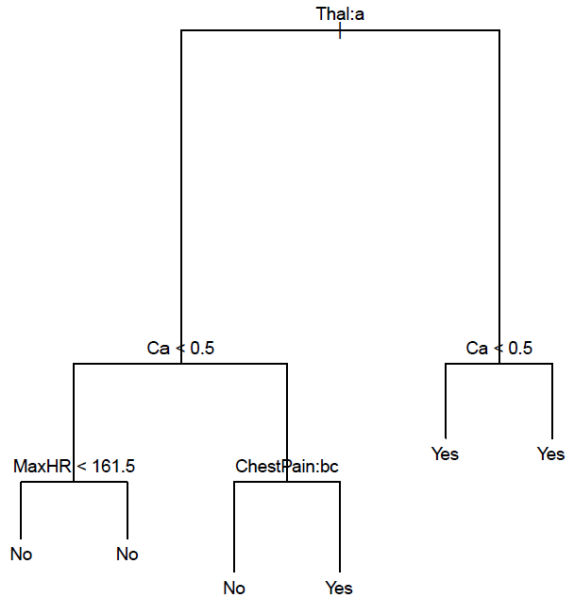
Pruning decision trees: **Cost Complexity Pruning**

Idea

Given a large tree, let us prune it back to **the subtree that leads to the lowest test error**

`Heart` data – 303 patients with chest pain. Response: yes/no (heart disease).
Pruned tree – minimal cross-validation error (Tree Size=6)





Source: James et al. (2013)

II

Stopping criterion – "*when is the tree grown enough?*"

Pruning is not enough

A word of warning: despite pruning, decision trees are sensitive to changes in the training dataset ("high variance") – we need to introduce *other methods* to tackle this problem.

III Rule to assigning values to all leaves – "*how to compute predictions?*"

This should be clear by now: let us test it together…

Lastly, a Google Colab notebook to test what we learned today

```
decision_trees.ipynb
```

Decision trees: Key takeaways

I	**“Interpretability”:** Decision trees are usually simple to interpret and easy to understand, even for non-technical stakeholders. They visually represent decision-making processes, showing clear paths from features to predictions.
II	**Flexibility:** Decision trees do not assume any particular relationship between the response and the input features (compare with linear regression models).
III	**Efficient handling of features**: Decision trees do not require normalization or scaling of data. They can handle varied data scales and types (numerical and categorical) directly.
IV	**Prone to Overfitting:** Decision trees can easily overfit, especially with complex datasets and without proper pruning.
V	**Lack of robustness:** Decision trees can be quite sensitive to small changes in the data (high variance). A slight perturbation in the dataset might result in a completely different tree structure.

Feedback! See you on May 24th