Practical AI

Emanuele Fabbiani

From the Last Lecture…

1. Business Understanding

- Workshop
- Evaluation metrics
- Literature review

2. Data Gathering

- Source discovery
- Data preparation
- Quality assessment

3. Data Exploration

- Explorative analysis
- Data cleaning

4. Modelling

- Feature engineering
- Model design
- Model implementation

5. Evaluation

- Performance evaluation
- Time evaluation

6. Deployment

- Execution scheduling
- Performance tuning
- Integration

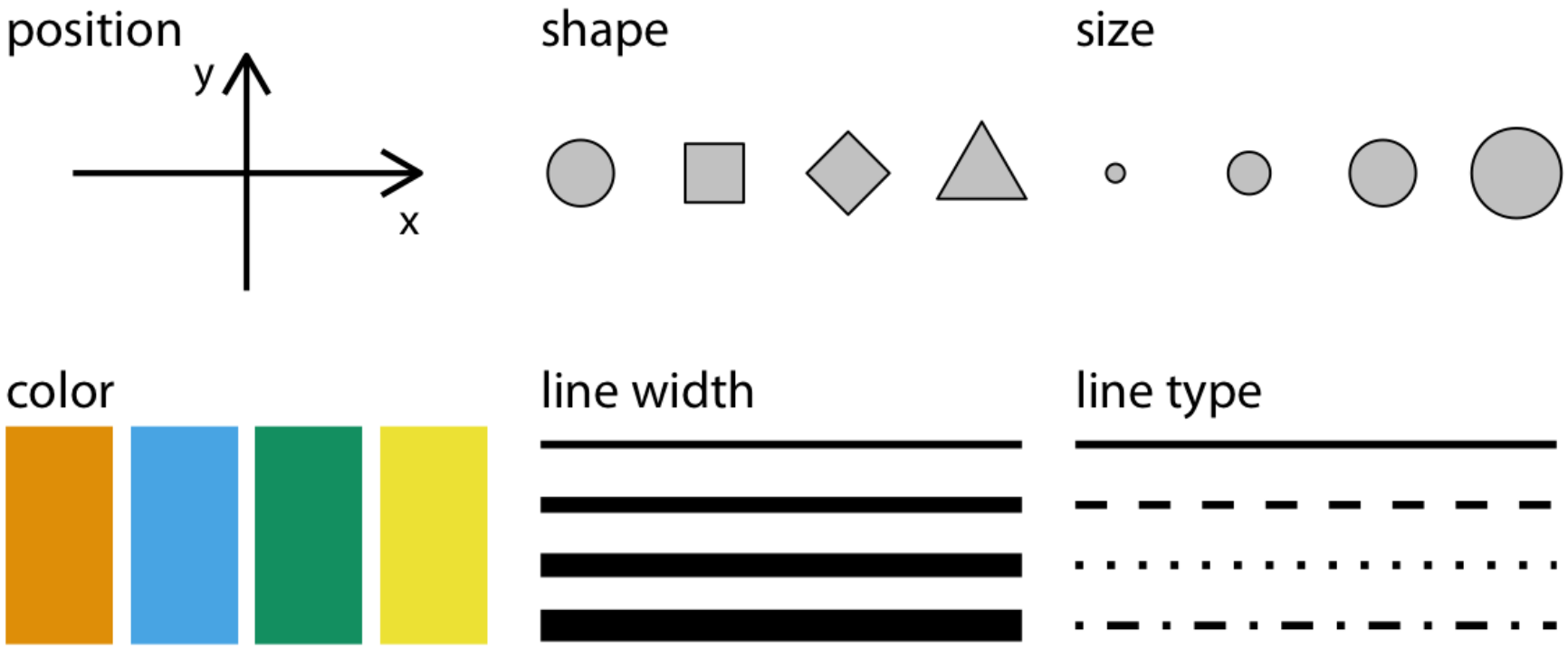
You Have a Dataset. Now What?

1. **Look** at the data.
2. **Understand** what each variable means.
3. Look for **missing data** / unreasonable values.
4. Perform **univariate** analysis – look at each variable alone with statistics and charts.
5. Perform **multivariate** analysis – look at the interaction between variables with statistics and charts.
6. Understand how to **handle missing data** and outliers.
7. Document and **report** your findings – or you will forget and regret it.


Summary Stats
Plots
imgflip.com

What is Data Visualization?

All data visualizations map data values into quantifiable features of the resulting graphic. We refer to these features as **aesthetics**.



https://clauswilke.com/dataviz/aesthetic-mapping.html

Rescheduling

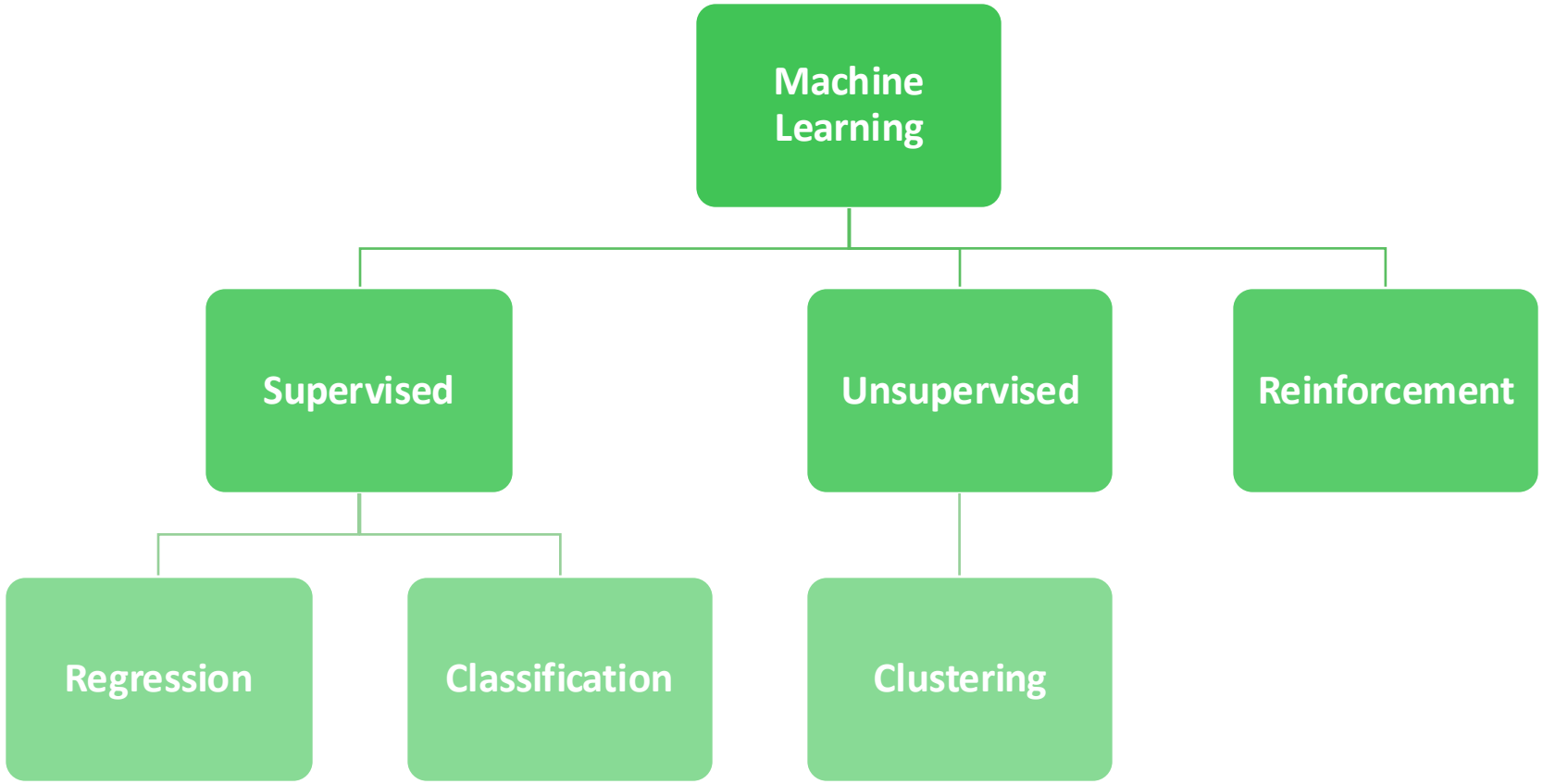
We need to reschedule 1 lecture and 2 exercise sessions, each 4 hours long.

- Wednesday **26th February**.
- Wednesday **5th March**.

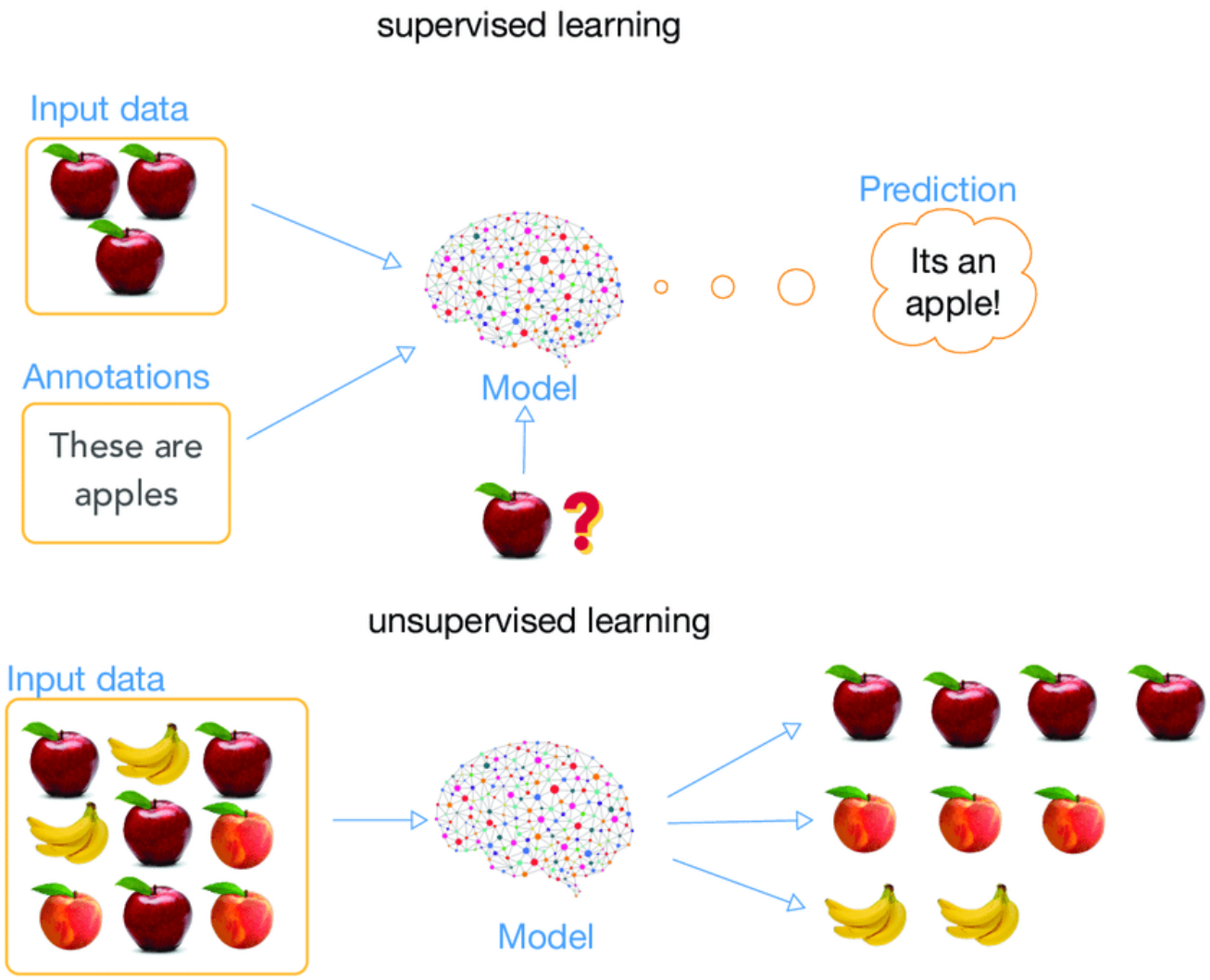
We still miss one, and I have no Wednesdays available.

A taxonomy for Machine Learning

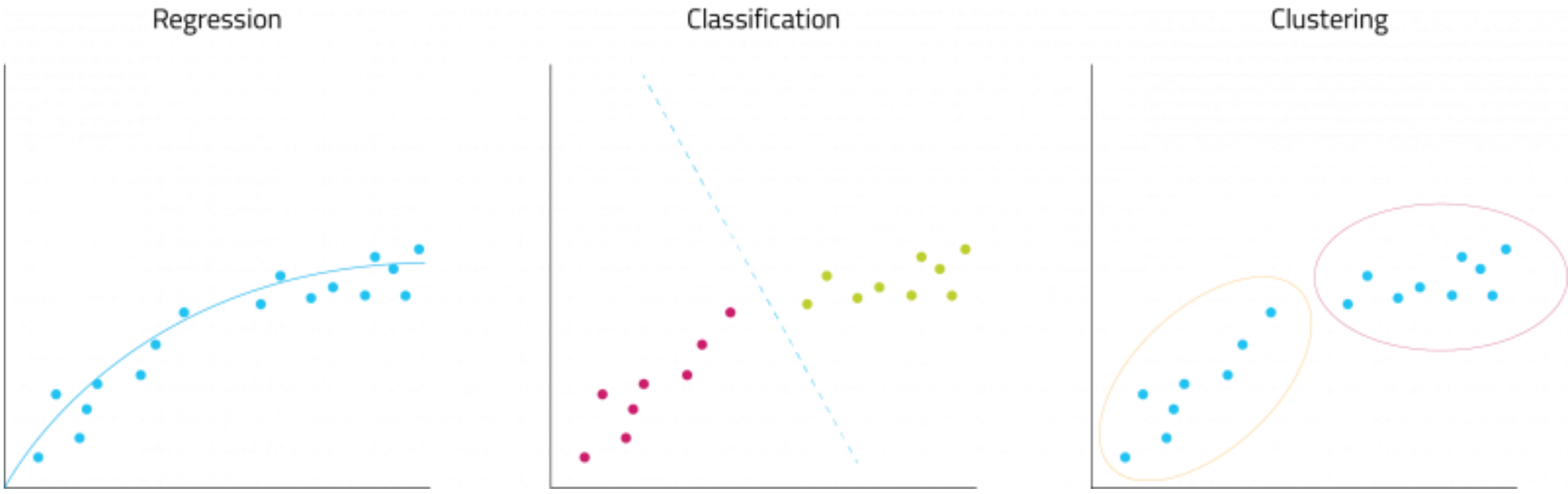
Machine Learning tasks



Supervised vs unsupervised



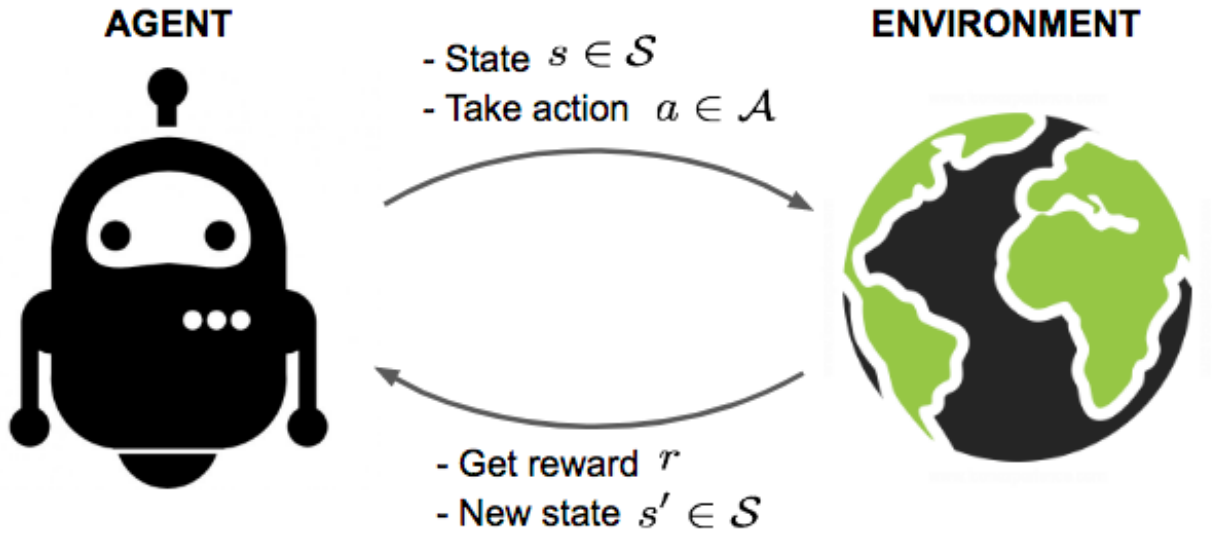
Regression, classification, clustering

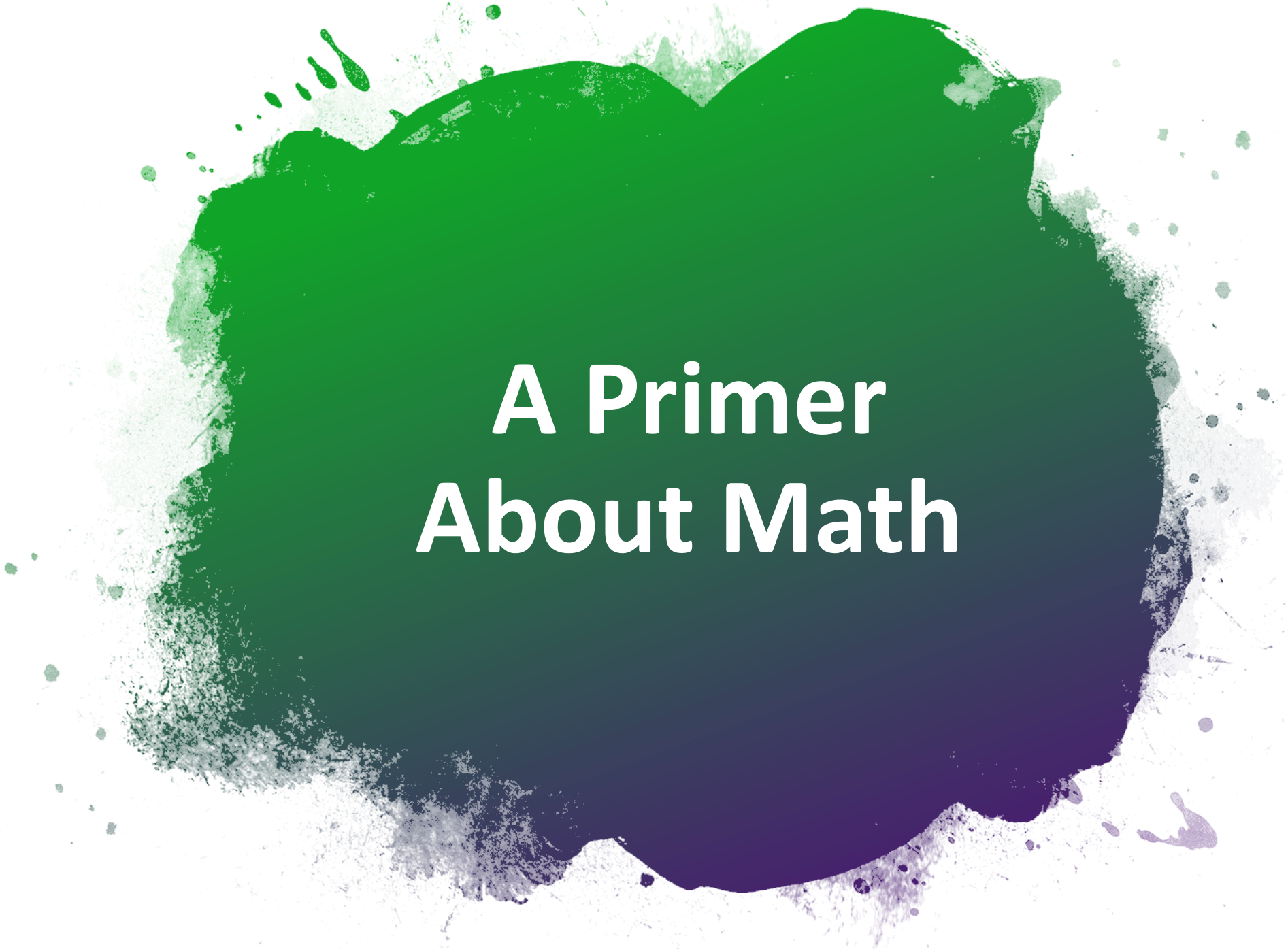


Reinforcement Learning

An agent:

- learns by interacting with its **environment**
- receives **rewards** if it performing correctly and **penalties** if it does not
- learns without intervention from a human by **maximizing its reward** and minimizing its penalty




A Primer
About Math

Vectors

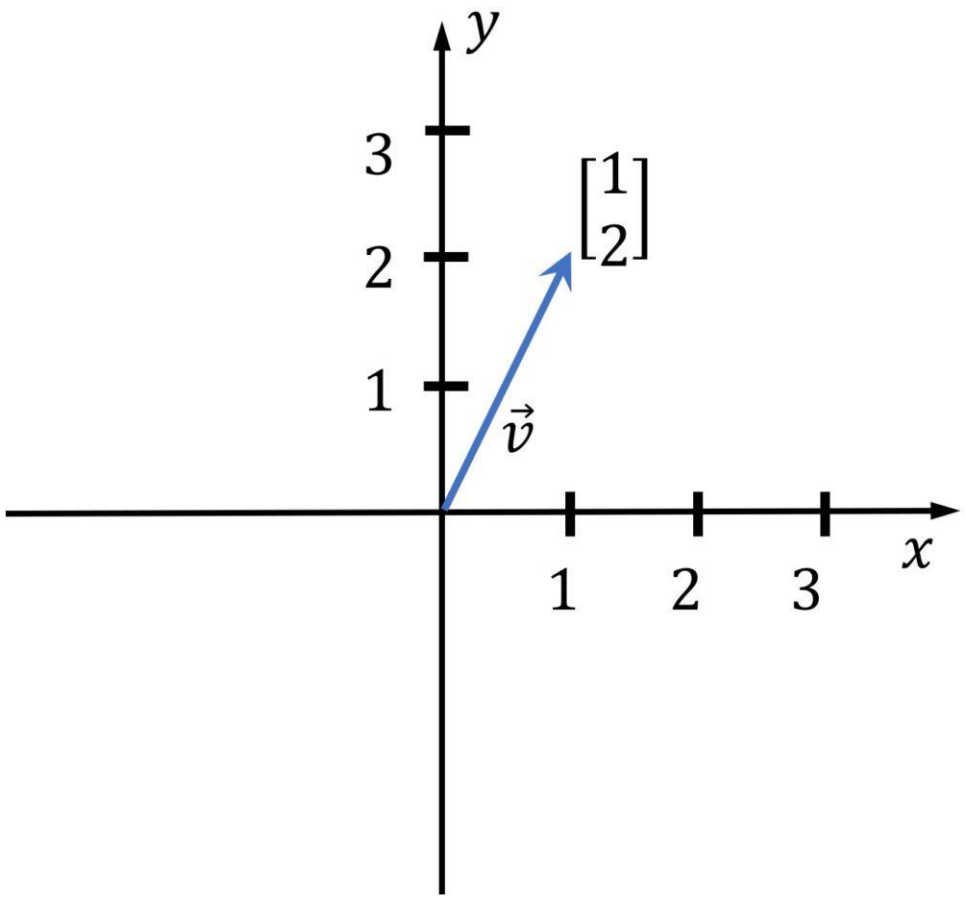
A **scalar** is a single number.

A **vector** is a list of numbers, with each number representing an element of the vector.

The **dimension** of a vector is the count of elements.

A vector can be interpreted in two ways.

- As a **point** in the space.
- As a **direction** and **magnitude**.



Vector Operations

Element-wise addition (results in a vector)

$$\begin{pmatrix}1\\2\end{pmatrix}+\begin{pmatrix}3\\4\end{pmatrix}=\begin{pmatrix}4\\6\end{pmatrix}$$

Scalar multiplication (results in a vector)

$$3\cdot\begin{pmatrix}1\\2\end{pmatrix}=\begin{pmatrix}3\\6\end{pmatrix}$$

Dot product (results in a scalar)

$$\begin{pmatrix}1\\2\end{pmatrix}\cdot\begin{pmatrix}3\\4\end{pmatrix}=1\cdot 3+2\cdot 4=3+8=11$$

Matrices

A matrix is a table of numbers.

A matrix represents a **linear function** applied to a vector. A linear function is a functions f such that:

$$f(x+y) = f(x) + f(y)$$
$$f(ax) = af(x)$$

Where a is a scalar and x, y are any two elements in the domain of f.

Geometrically, a matrix represents a transformation that stretches and rotates a vector.

For instance, a matrix A with 3 rows and 2 columns can me defined as:

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}$$

The dimension of A is 3×2.

Matrix and Vector Notation

Let x be a vector and X a matrix.

Then:

x_i is the i-th element of the vector x

$X_{i,j}$ it the element in the row i, column j of matrix X

$X_{i,.}$ is the vector representing the i-th row of matrix X

$X_{.,j}$ is the vector representing the j-th column of matrix X

Matrix Operations

Element-wise addition. Results in a matrix, only allowed between matrices with the same dimension.

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} + \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix} = \begin{pmatrix} 1+5 & 2+6 \\ 3+7 & 4+8 \end{pmatrix} = \begin{pmatrix} 6 & 8 \\ 10 & 12 \end{pmatrix}$$

Scalar multiplication. Same as with vectors. Results in a matrix, where each element is multiplied by the scalar.

$$5 \cdot \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} = \begin{pmatrix} 1 \cdot 5 & 2 \cdot 5 \\ 3 \cdot 5 & 4 \cdot 5 \end{pmatrix} = \begin{pmatrix} 5 & 10 \\ 15 & 20 \end{pmatrix}$$

Matrix-vector multiplication. Results in a vector, which is the sum of the scalar-vector product between each element of the vector and the corresponding column of the matrix.

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \cdot \begin{pmatrix} 5 \\ 6 \end{pmatrix} = 5 \begin{pmatrix} 1 \\ 3 \end{pmatrix} + 6 \begin{pmatrix} 2 \\ 4 \end{pmatrix} = \begin{pmatrix} 17 \\ 39 \end{pmatrix}$$

Matrix Multiplication

Matrix multiplication is **only defined** between two matrices where the number of columns of the first matrix matches the number of rows of the second matrix.

Matrix multiplication is **not** commutative!

Let A be a matrix of dimension $n \times m$ and B a matrix with dimension $m \times p$. Then, the result of the product $A \cdot B$ is a matrix with dimension $n \times p$, where the element in position i, j is the dot product of the ith row of A with the jth column of B.

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \cdot \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix} = \begin{pmatrix} 1 \cdot 5 + 2 \cdot 7 & 1 \cdot 6 + 2 \cdot 8 \\ 3 \cdot 5 + 4 \cdot 7 & 3 \cdot 6 + 4 \cdot 8 \end{pmatrix} = \begin{pmatrix} 19 & 22 \\ 43 & 50 \end{pmatrix}$$

Matrix Inverse

Let I_n be the **identity matrix** of dimension n. The identity matrix is a matrix consisting of all zeros, except for the main diagonal, where all elements are equal to 1.

$$I_1 = 1, \qquad I_2 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \qquad I_3 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Given an $n \times n$ matrix A, its **inverse** A^{-1} is a matrix such that:

$$A \cdot A^{-1} = I_n$$

Computing the inverse of a matrix is a computationally expensive operation and is rarely performed directly.

Functions, Derivatives, Gradients

Given two non-empty sets **A** and **B**, a function is a relationship defined from **A** to **B** such that each element of **A** is associated with one and only one element in **B**.

Given a function $f: D \subset R \rightarrow R$ defined by $y = f(x)$, its first **derivative** is defined by:

$$y = f'(x) = \frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Given a function $f: D \subset R^n \rightarrow R$ defined by $y = f(x_1, x_2, x_3, \dots, x_n)$, its **gradient** is the n-dimensional **vector** ∇f defined by:

$$\nabla f(x_1, x_2, x_3, \dots, x_n) = \begin{pmatrix} \frac{\partial f(x_1, x_2, x_3, \dots, x_n)}{\partial x_1} \\ \dots \\ \frac{\partial f(x_1, x_2, x_3, \dots, x_n)}{\partial x_n} \end{pmatrix}$$

Functions, Derivatives, Gradients

Given a function $f: D \subset R^n \rightarrow R$ defined by $y = f(x_1, x_2, x_3, \dots, x_n)$, its **gradient** is the n-dimensional **vector** ∇f defined by :

$$\nabla f(x_1, x_2, x_3, \dots, x_n) = \begin{pmatrix} \frac{\partial f(x_1, x_2, x_3, \dots, x_n)}{\partial x_1} \\ \dots \\ \frac{\partial f(x_1, x_2, x_3, \dots, x_n)}{\partial x_n} \end{pmatrix}$$

where $\frac{\partial f(x_1,x_2,x_3,\dots,x_n)}{\partial x_1}$ is the **partial derivative** of f with respect to x_1.

The partial derivative is the derivative taken with respect to one variable while keeping all other variables constant.

Linear Algebra & Calculus

This is a brief introduction to a broad and captivating topic.

While it is not required for this class, you are welcome to explore the video series by ThreeBlueOneBrown if you wish to dig deeper:

- Linear Algebra
- Calculus

Some News

Rescheduling

Lectures have been scheduled

- Wednesday **26th February**.
- Wednesday **5th March**.

All the lectures from now on will be 5 hours long and will be held in **Buonarroti**.

Project Work – Deadline & Rules

The project work must be submitted **before** the start of the written exam. Late submissions will result in 0 points, irrespective of the quality of the work.

Submission and completion of the written exam **cannot** be done separately.

If you retake the written exam, you may **resubmit** the same project work, but the mark will remain unchanged.

Alternatively, you may submit a **different** project work when retaking the written exam.

1. Business Understanding

- Workshop
- Evaluation metrics
- Literature review

2. Data Gathering

- Source discovery
- Data preparation
- Quality assessment

3. Data Exploration

- Explorative analysis
- Data cleaning

4. Modelling

- Feature engineering
- Model design
- Model implementation

5. Evaluation

- Performance evaluation
- Time evaluation

6. Deployment

- Execution scheduling
- Performance tuning
- Integration

Linear Regression

Context

Supervised learning: we are given both inputs and outputs, and the model should capture the relationship between them.

Assumptions:

- The relationship between input and output is (approximately) linear
- There is no uncertainty on the inputs
- The uncertainty on the outputs is constant
- All the samples are independent from each other
- No features are collinear

Problem statement

We measure the weight and height of 25000 female students aged 18.

For other students, we will be given the weight, but we will **not** be provided the height.

We need to build a model to **predict** the height given the weight.

Height	Weight
167.1	51.3
181.6	61.9
176.3	69.4
173.3	64.6
172.2	65.5
…	…

Linear regression model

n: the number of samples

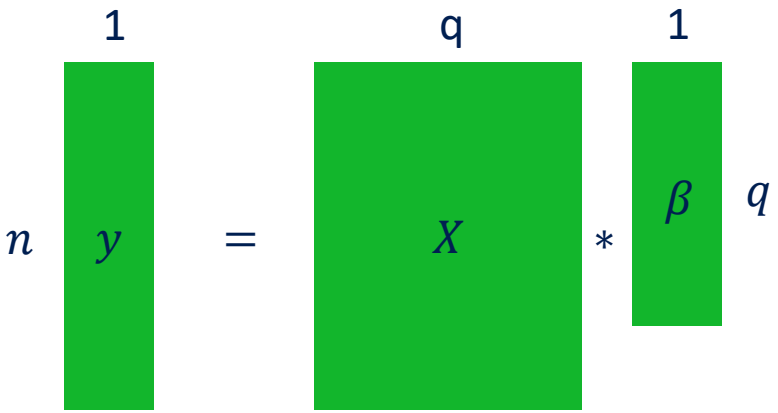
q: the number of features

$y \in R^n$: the output vector

$X \in R^{n,q}$: the input matrix, with n rows and q columns

$\beta \in R^q$: the parameter vector

Model equation: $y = X\beta = \beta_1 X_{.,1} + \beta_2 X_{.,2} + \cdots + \beta_q X_{.,q}$



Example

We measure the weight and height of 25000 female students aged 18.

We need to build a model to predict the height given the weight.

$n = 25000$: the number of samples

$y = \begin{pmatrix} 167.1 \\ 181.6 \\ 176.3 \\ \dots \end{pmatrix} \in R^{25000}$: the output vector

$X = \begin{bmatrix} 51.3 & 1 \\ 61.9 & 1 \\ 69.4 & 1 \\ \dots & \dots \end{bmatrix} \in R^{25000,2}$: the input matrix, with 25000 rows and 2 columns

$\beta = \begin{pmatrix} \beta_1 \\ \beta_2 \end{pmatrix} \in R^2$: the parameter vector

Linear regression training

$$y = X\beta$$

$$\begin{pmatrix} 167.1 \\ 181.6 \\ 176.3 \\ \dots \end{pmatrix} \approx \begin{bmatrix} 51.3 & 1 \\ 61.9 & 1 \\ 69.4 & 1 \\ \dots & \dots \end{bmatrix} \begin{pmatrix} \beta_1 \\ \beta_2 \end{pmatrix}$$

Find β such that βX is ***as close as possible*** to y.

Linear regression loss function

Find β such that βX is *as close as possible* to y.

We want to minimize the mean squared error between y and βX. This is why this method is often called Ordinary Least Squares (OLS).

The **loss function** is:

$$L(\beta) = \frac{1}{n}\|y - X\beta\|_2^2 = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \beta \cdot X_{i,.}\right)^2$$

Linear regression training

$$y = X\beta$$

$$\begin{pmatrix} 167.1 \\ 181.6 \\ 176.3 \\ \dots \end{pmatrix} \approx \begin{bmatrix} 51.3 & 1 \\ 61.9 & 1 \\ 69.4 & 1 \\ \dots & \dots \end{bmatrix} \begin{pmatrix} \beta_1 \\ \beta_2 \end{pmatrix}$$

Find β such that βX is *as close as possible* to y. That is, find β such that

$$L(\beta) = \frac{1}{n}\|y - \beta X\|_2^2 = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \beta \cdot X_{i,.}\right)^2$$

is **minimal**.

Linear regression training

Find β such that βX is *as close as possible* to y. That is, find β such that

$$L(\beta) = \frac{1}{n}\|y - X\beta\|_2^2 = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \beta \cdot X_{i,.}\right)^2$$

is minimal.

$L(\beta)$ is **convex**, so the optimization problem can be solved easily. Moreover, it is **differentiable**, so we know that the optimal β, denoted $\hat{\beta}$, can be found by solving $\nabla L(\beta) = 0$, where 0 is a vector with the same shape of β.

The solution is

$$\hat{\beta} = (X^T X)^{-1} X^T y$$

Note: inverting a matrix is a computationally expensive operation, so no library implements this formula.

Reminder

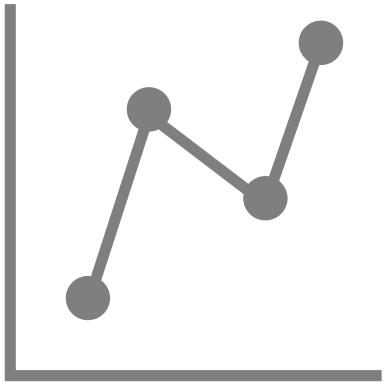
You will **not** be expected to memorize formulas—nobody does. Google is your ally.

However, it is useful to know **when a formula exists** and when more complex procedures are required to find a solution.

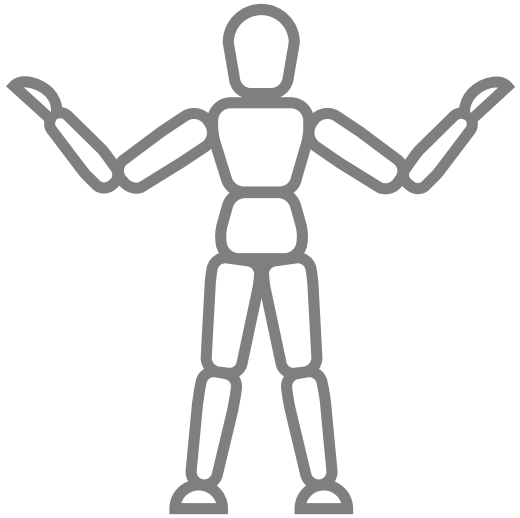
Everything in life is an optimization problem, but not every optimization problem has an easy solution.

Doubt

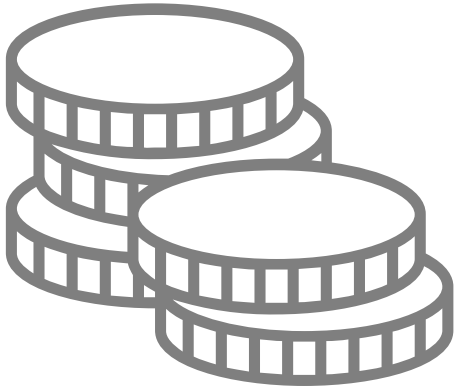
Data



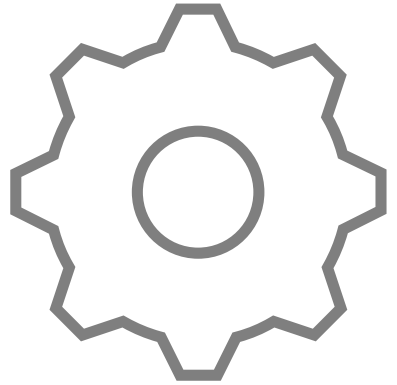
Model



Loss



Learning algorithm



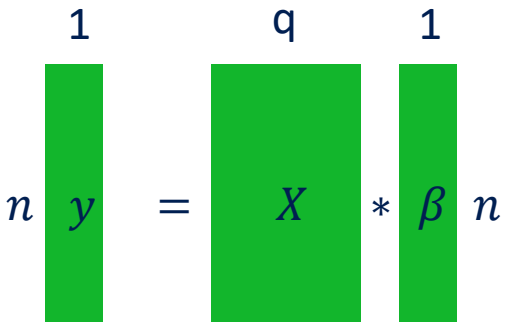
Discussion

How can each slot of the framework be filled in the case of linear regression?

Data

Height	Weight
167.1	51.3
181.6	61.9
176.3	69.4
173.3	64.6
172.2	65.5
…	…

Model



Loss

$$L(\beta) = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \beta \cdot X_{i,.}\right)^2$$

Learning algorithm

$$\hat{\beta} = (X^T X)^{-1} X^T y$$

What is a linear regression model?

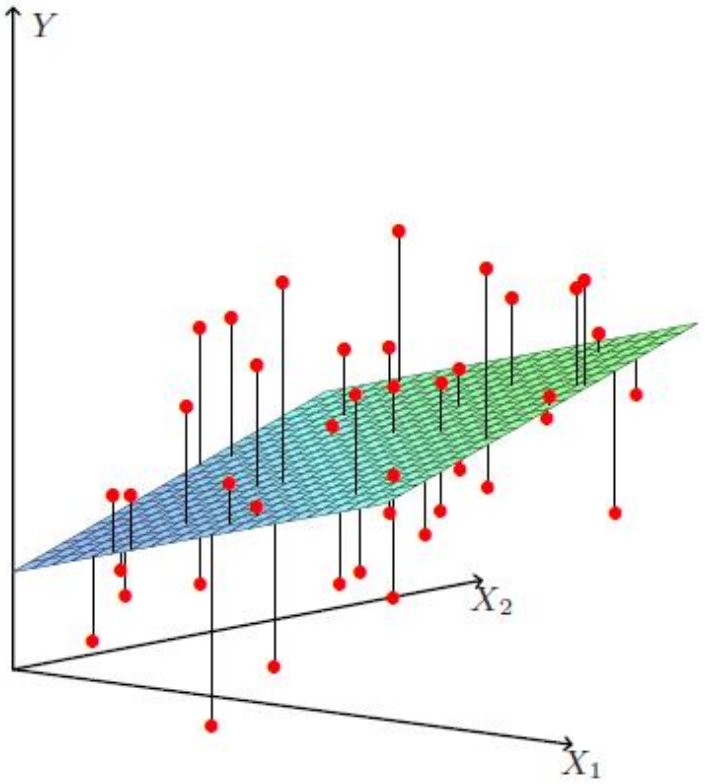


FIGURE 3.1. *Linear least squares fitting with* $X \in \mathbb{R}^2$. *We seek the linear function of* X *that minimizes the sum of squared residuals from* Y.

From T. Hastie et al, Elements of Statistical Learning

Discussion

How can we evaluate an ML model?

How do we evaluate the model?

We can use a linear regression model to **explore** the feature of a dataset. That's common in econometrics.

However, we are usually interested in making predictions.

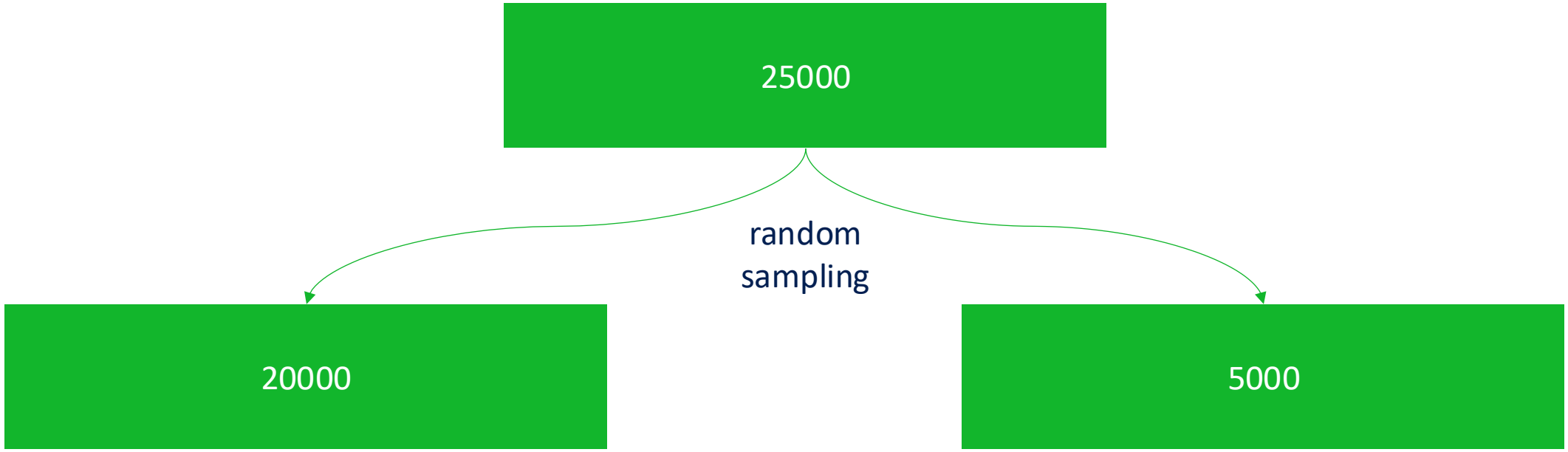
To make good predictions, the model must be able to **generalize**.

If we use all the data to train the model, we cannot evaluate how well it works on other data.

So, we need to **hold out** some data.

We define a **train** and a **test** set.

How do we evaluate the model?



Train set – used for parameter optimization

Test set – used for evaluation **only**

How do we evaluate the model?

20000

5000

Train set – used for parameter optimization

Test set – used for evaluation **only**

$$\hat{\beta} = (X^T X)^{-1} X^T y$$

$$\hat{y} = X\hat{\beta}$$

$$M = \sum_i |\hat{y}_i - y_i|$$ (this is a metric)

How do we use the model?

When a **new sample** x arrives, we only know the weight.

We need to predict the height.

So, we use our model:

$$\hat{y} = \hat{\beta} \cdot x$$

to predict the height, where:

$$x = \begin{pmatrix} weight \\ 1 \end{pmatrix}$$

Interpreting linear regression coefficients

We measure the weight and height of 25000 female students aged 18.

For other students, we will be given the weight, but we will **not** be provided the height.

We need to build a model to **predict** the height given the weight.

Assume the fitted model is:

$$\hat{y} = X\hat{\beta}$$

with $\hat{\beta} = \binom{0.46}{146}$.

The first coefficient means that, on average, a 1kg increase in weight leads to an increase of 0.46 cm in height.

How to evaluate uncertainty?

The most common method to evaluate uncertainty in a linear regression model is to resort to its **statistical interpretation**.

The data-generating process is:

$$y = X\beta + \epsilon$$

where ϵ is a **random vector** of size n.

As per the assumption of the model, ϵ_i, ϵ_j are **independent** for each i and j, $i \neq j$.

Moreover, the mean of ϵ is assumed to be **zero**.

Usually, ϵ_i is also assumed to be **Gaussian** for each i.

Note that y is a random vector, while X is deterministic.

How to evaluate uncertainty?

Reminder:

$$\hat{\beta} = (X^T X)^{-1} X^T y$$
$$\hat{y} = X\hat{\beta}$$

but now $\hat{\beta}$ and $\hat{y}$ are **random** vectors!

Assuming ϵ is zero-mean and the variance of each ϵ_i is σ_ϵ^2, we can compute mean and variance of the prediction $\hat{y}_i$ originated from a new sample x. We get:

$$E[\hat{y}] = y$$
$$Var[\hat{y}_i] = \sigma_\epsilon^2 x^T (X^T X)^{-1} x$$
$$Var[y_i - \hat{y}_i] = \sigma_\epsilon^2 (I + x^T (X^T X)^{-1} x)$$

Using the assumptions on the distribution of ϵ, and thus of $\hat{y}$, we can then compute **confidence intervals** on parameters and **prediction intervals** on predictions.

How to evaluate uncertainty?

If σ_ϵ^2 is **known**, one can show that $\hat{y}_i$ is Gaussian and that the prediction interval can be computed as:

$$\left[\hat{y}_i - z_c\sqrt{\sigma_\epsilon^2(1 + x^T(X^TX)^{-1}x)}; \hat{y}_i + z_c\sqrt{\sigma_\epsilon^2(1 + x^T(X^TX)^{-1}x)}\right]$$

where z_c is the quantile of the standard Gaussian distribution for the desired confidence.

For a 95% confidence, it is 1.96.

If σ_ϵ^2 is not known, it can be estimated by computing the variance of $\hat{\epsilon} = \hat{y} - y$ on the train set.

In this case, however, t_c, the quantile of a **Student's T distribution** with $n - q$ degrees of freedom, must be used.

How to evaluate uncertainty?

While prediction intervals are for… predictions, confidence intervals are for parameters.

We can compute:

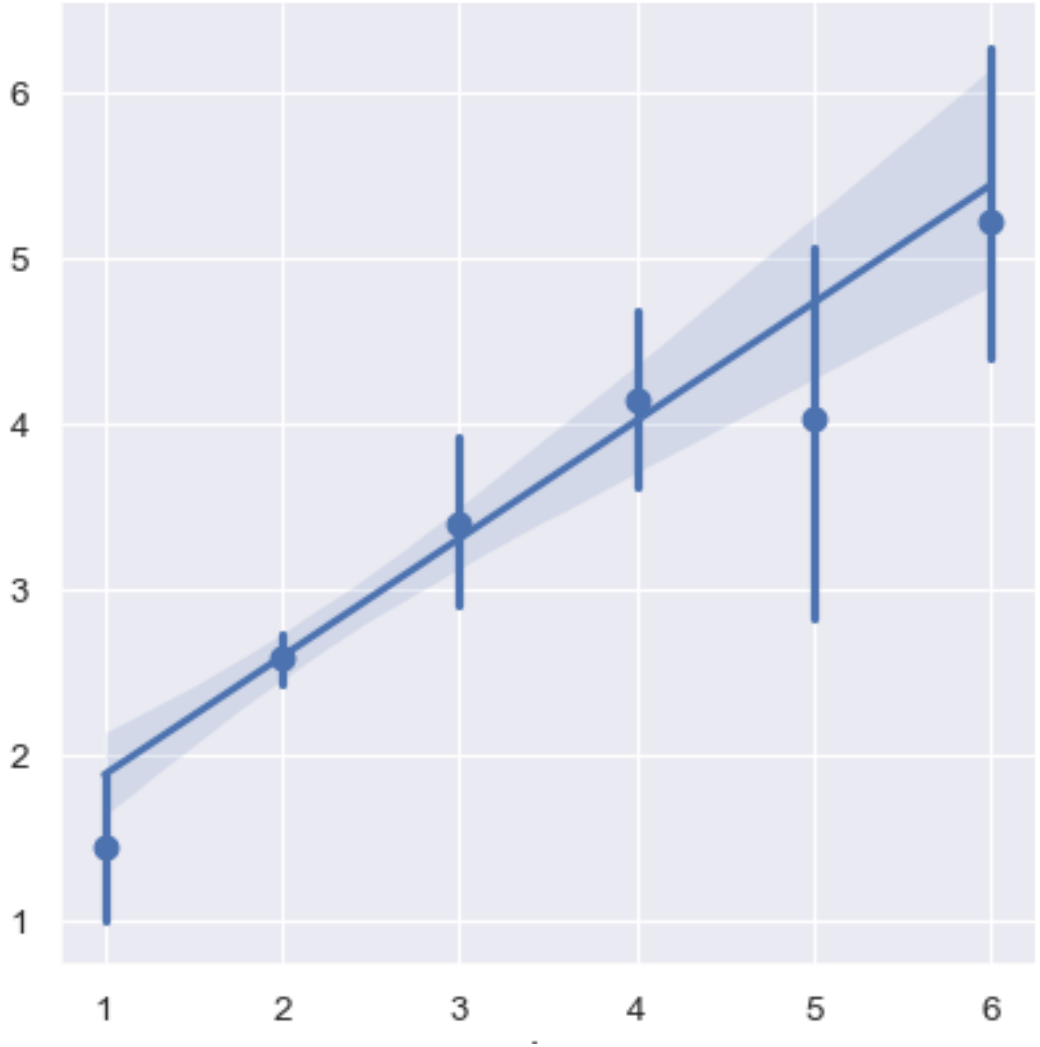
$$E\left[\hat{\beta}\right] = \beta$$
$$Var\left[\hat{\beta}\right] = \sigma_\epsilon^2 (X^T X)^{-1}$$

The variance of $\hat{\beta}$ is a **matrix**, we are concerned with the elements on the diagonal, which represent the variance of each $\hat{\beta}_i$.

Then, we can compute the **confidence intervals** for each $\hat{\beta}_i$ as:

$$\left[\widehat{\beta_i} - z_c\sqrt{\sigma_\epsilon^2 (X^T X)_{ii}^{-1}};\ \widehat{\beta_i} + z_c\sqrt{\sigma_\epsilon^2 (X^T X)_{ii}^{-1}}\right]$$

How to evaluate uncertainty?



What are confidence intervals?

A confidence interval provides an idea about the uncertainty of the estimate.

If the estimate was repeated **infinite times**, with all the datasets generated by the same generation process $y = X\beta + \epsilon$, the α-level confidence interval is the interval where the estimated $\hat{\beta}$ will fall a fraction α of the times.

Reminder

You will **not** be expected to memorise formulas—nobody does. Google is your friend.

However, it is important to understand that **such formulas exist and how they originate**.

Recognizing the limitations and merits of frameworks for uncertainty estimation is a crucial topic.

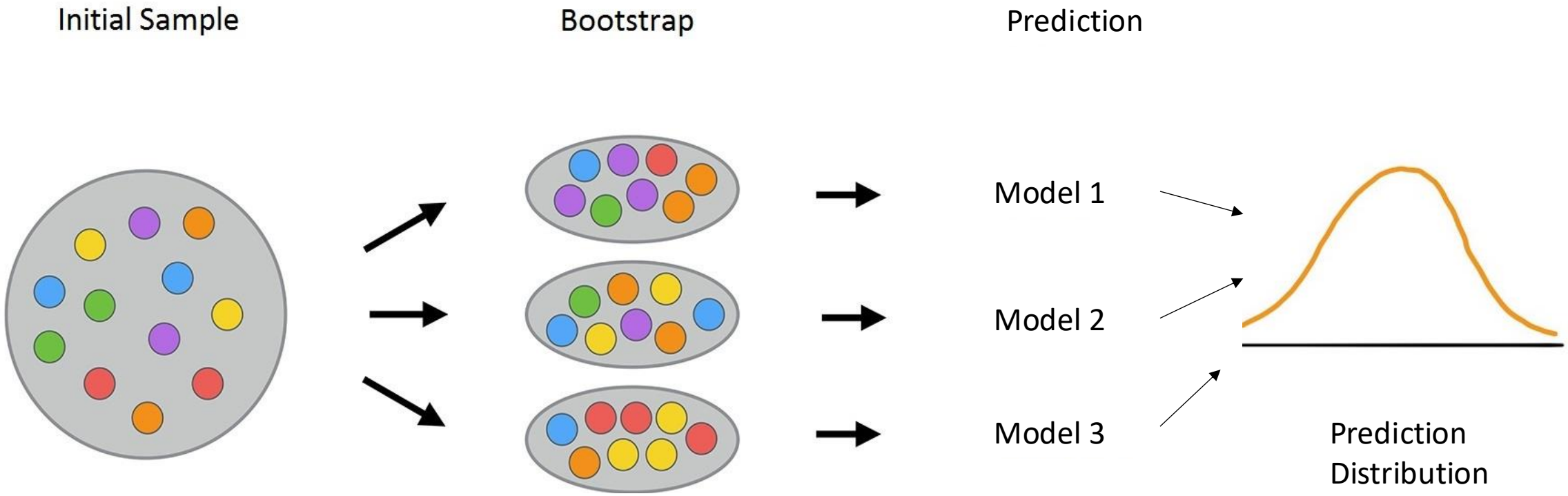
Beyond parametric statistics

Many models cannot support easy statistical interpretations.

So, we need more general methods to estimate prediction uncertainty.

Bootstrap is a common one.

Bootstrap


Initial Sample
Bootstrap
Prediction
Model 1
Model 2
Model 3
Prediction
Distribution

Feature engineering

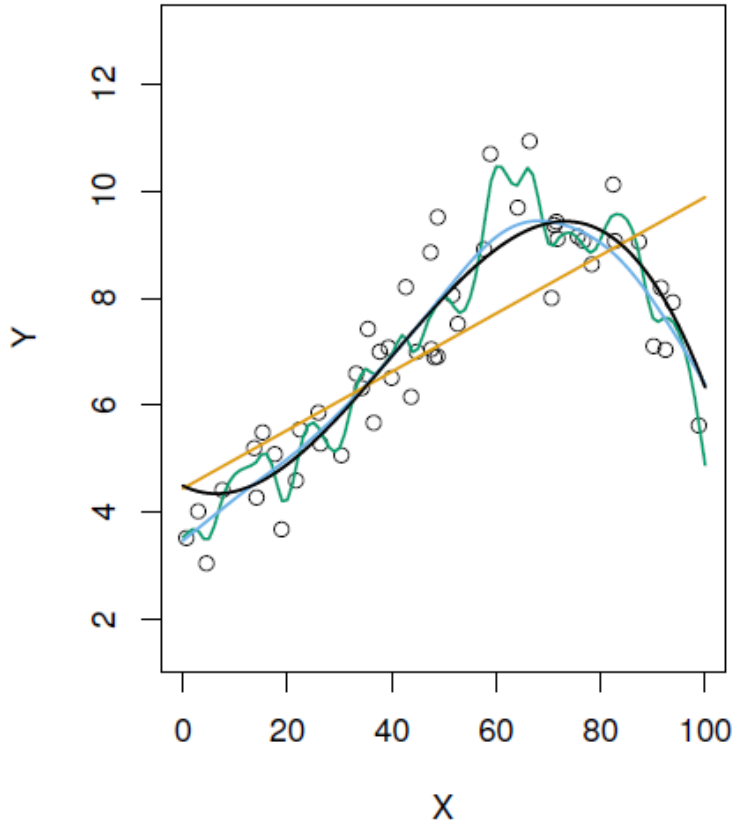
We want to create a model to fit the dataset in the picture.

A straight line such as $y = \beta_0 + \beta_1 x$ will not cut it.

So, we may try and make the model more complex.

$$y = \beta_0 + \beta_1 x + \beta_2 x^2$$

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3$$

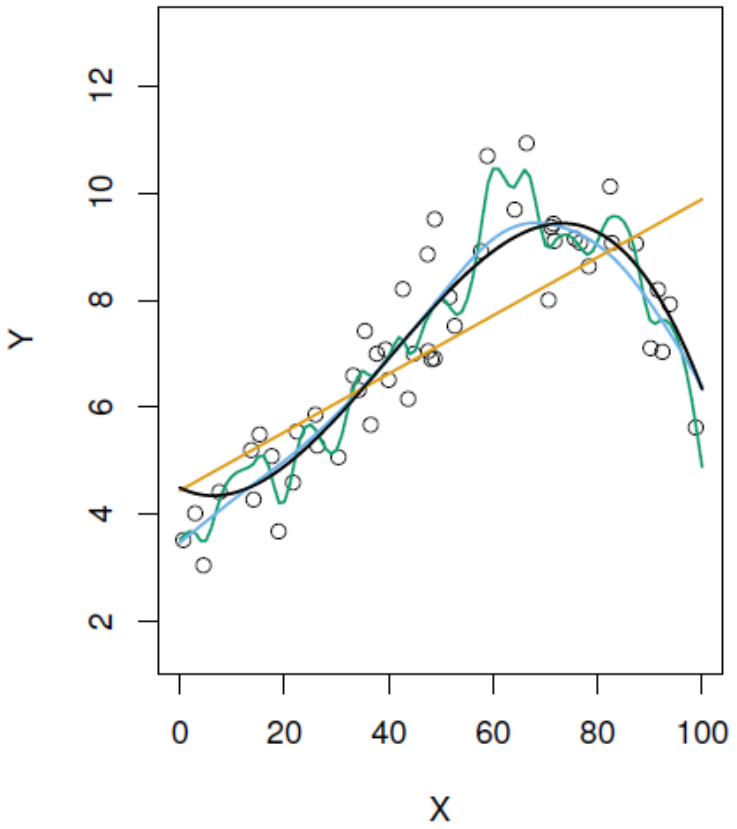


Feature engineering

The more complex the model, the better it can follow the training data.

However, it is **not guaranteed** that a more complex model will outperform a simpler one on new samples.

Note that the models with squares and cubes are **still linear** and the optimal β can still be found with the least squares formula.



Bias-variance tradeoff

Let us consider the statistical interpretation of linear regression:

$$y = X\beta + \epsilon$$

Given a new sample (x, y), one can show that the **expected mean squared error** of the prediction $\hat{y}$ is:

$$E_D[(y - \hat{y})^2] = E_D[y - \hat{y}]^2 + Var_D[\hat{y}] + \sigma_\epsilon^2$$

- $E_D[y - \hat{y}] = y - E_D[\hat{y}]$ is the **bias** of the prediction
- $Var_D[\hat{y}] = E_D[(\hat{y} - E_D[\hat{y}])^2]$ is the **variance** of the prediction
- σ_ϵ^2 is the **irreducible error**

The expected values are computed over all possible **datasets** D.

See https://stats.stackexchange.com/questions/204115/understanding-bias-variance-tradeoff-derivation for a formal proof

Bias-variance tradeoff

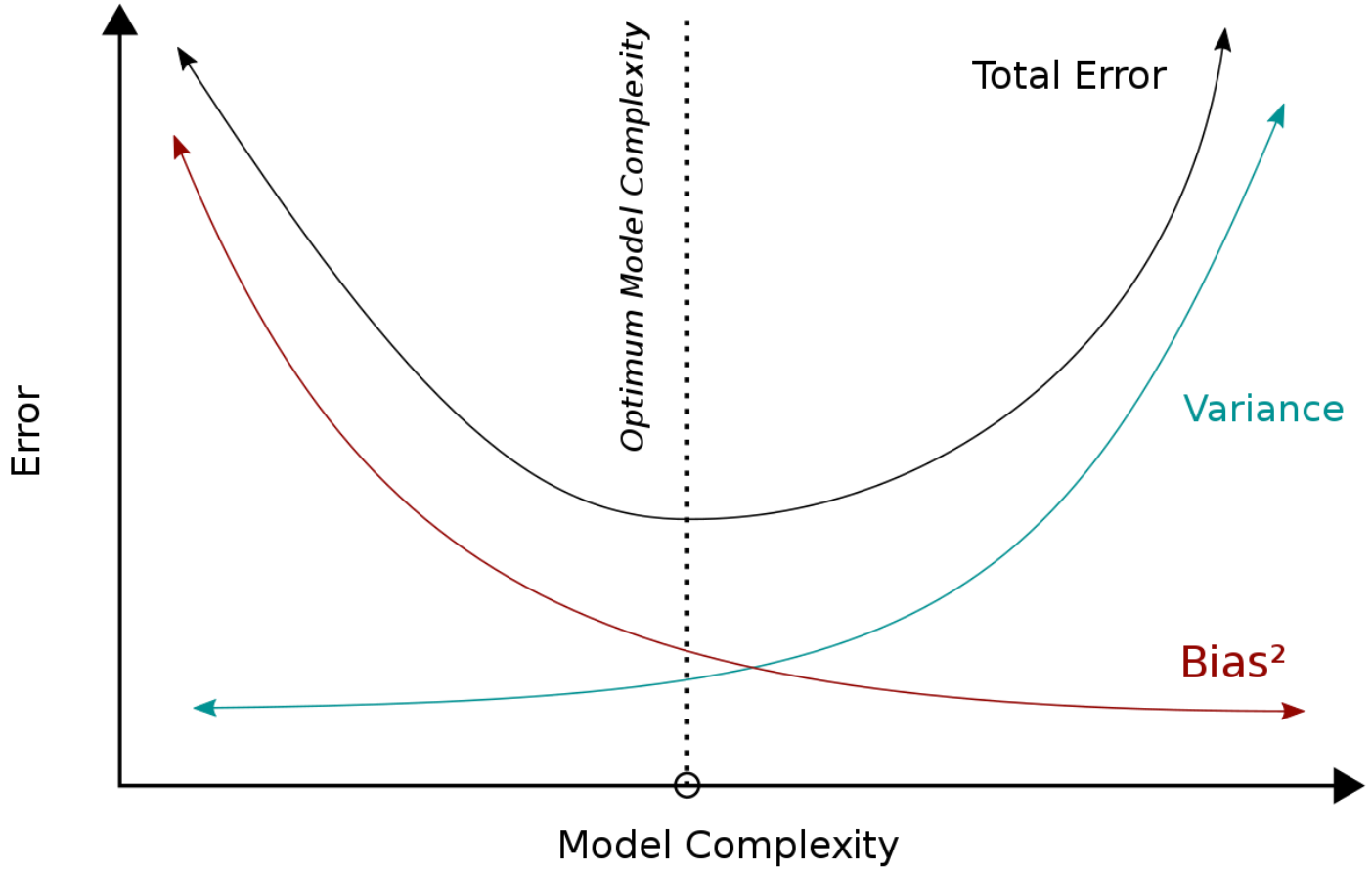
The bias measures the capability of the model to adapt to the training data.

The variance measures the variability of the sensitivity of the prediction to changes in the training data.

The more complex and flexible a model is, the lower the bias, the higher the variance.

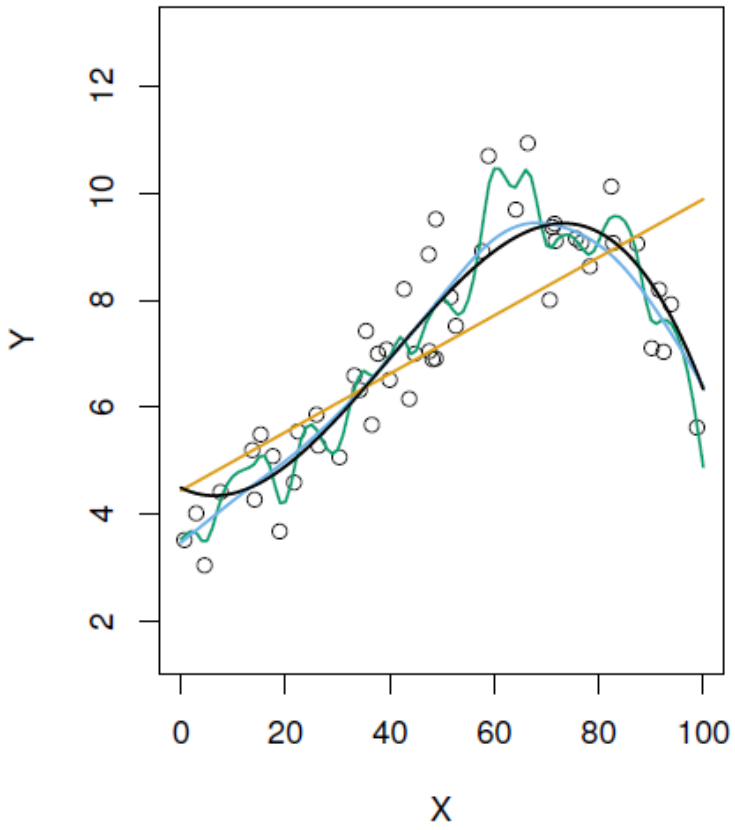
Theoretically, one wants low bias and low variance. However, there is a **tradeoff** between the two.

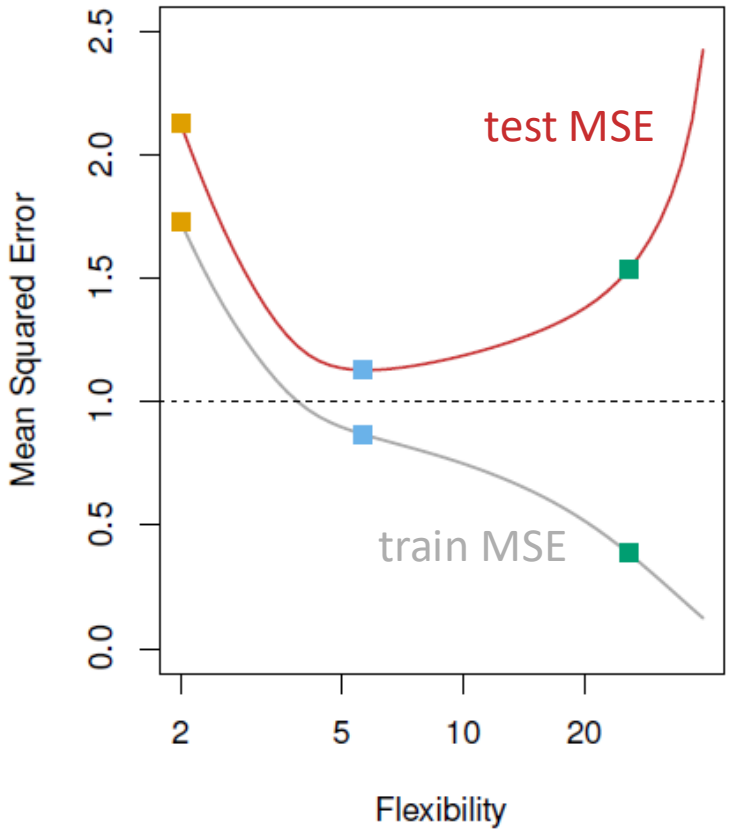
Bias-variance tradeoff

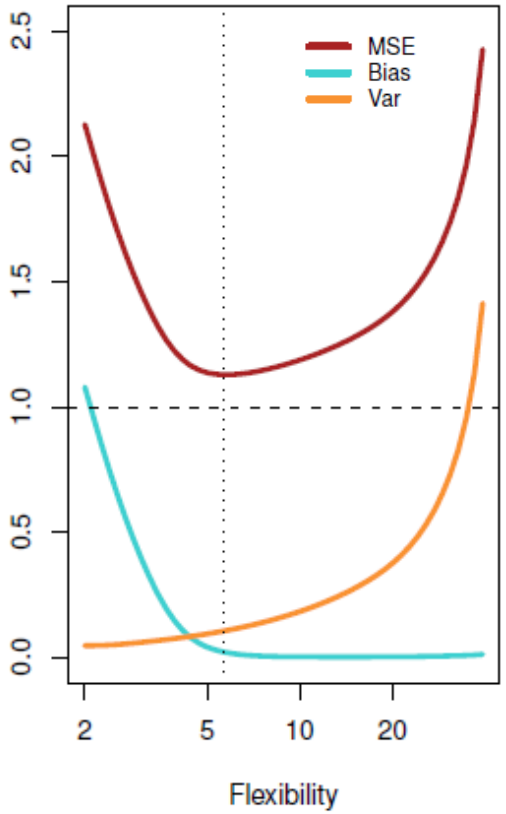


From https://en.wikipedia.org/wiki/Bias%E2%80%93variance_tradeoff

Bias-variance tradeoff





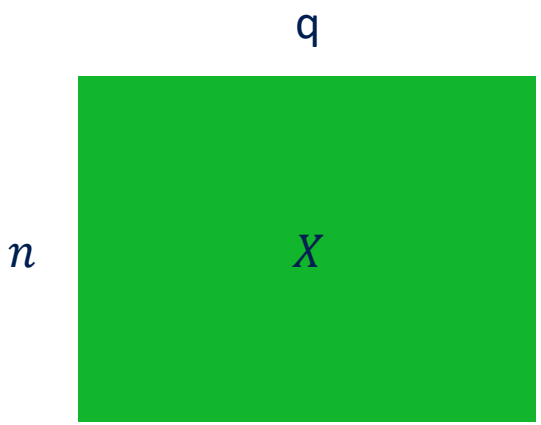


From James et al, Introduction to Statistical Learning

Problems with linear regression

1. What happens if the number of features q is greater than the number of samples n?

2. Can we control the flexibility of the model without adding or deleting features?



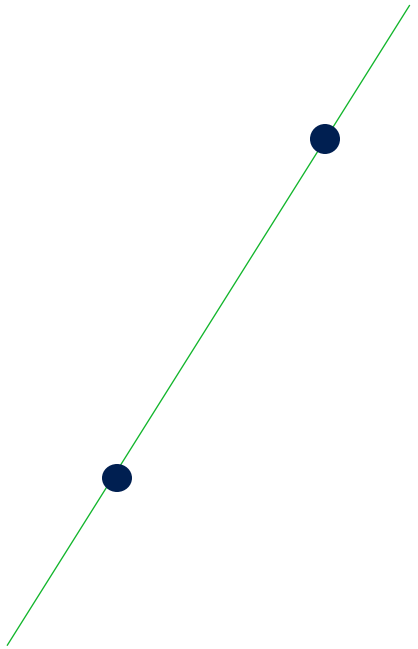
The case $n < q$

If $n < q$, or even when n is about the same size as q, we do not have enough **information** to identify the model parameters.

To understand this, think about curve fitting.

To identify a single straight line, which has two parameters ($y = \beta_0 + \beta_1 x$), we need at least two points.

To identify a parabola ($y = \beta_0 + \beta_1 x + \beta_2 x^2$), we need three. And so on.



Controlling flexibility

We would like to control the flexibility of a model without having to add or remove features.

We can control for the size of the parameters!

By setting a maximum “budget” the model can use for its parameters, it will be bound to use **smaller parameters** – thus providing smoother behaviour – or to use **fewer of them**.

Regularization

We change the loss function of the linear regression.

$$L_{(r)}(\beta) = \frac{1}{n}\|y - \beta X\|_2^2 + \lambda\|\beta\|_2^2 = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \beta \cdot X_{i,.}\right)^2 + \lambda\sum_{i=1}^{q}\beta_i^2$$

$$L_{(l)}(\beta) = \frac{1}{n}\|y - \beta X\|_2^2 + \lambda|\beta| = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \beta \cdot X_{i,.}\right)^2 + \lambda\sum_{i=1}^{q}|\beta_i|$$

The first loss results in a **ridge regression**, the second in a **LASSO model**.

Ridge regression

The optimal parameters for a ridge regression can be computed in **closed form**.

$$\hat{\beta}_{(r)} = (X^T X + \lambda I)^{-1} X^T y$$

where I is an identity matrix.

Now the formula can be solved also in the case $n < q$.

Moreover, the higher λ, the more the parameters $\hat{\beta}_{(r)}$ will be shrunk towards zero.

Ridge regression

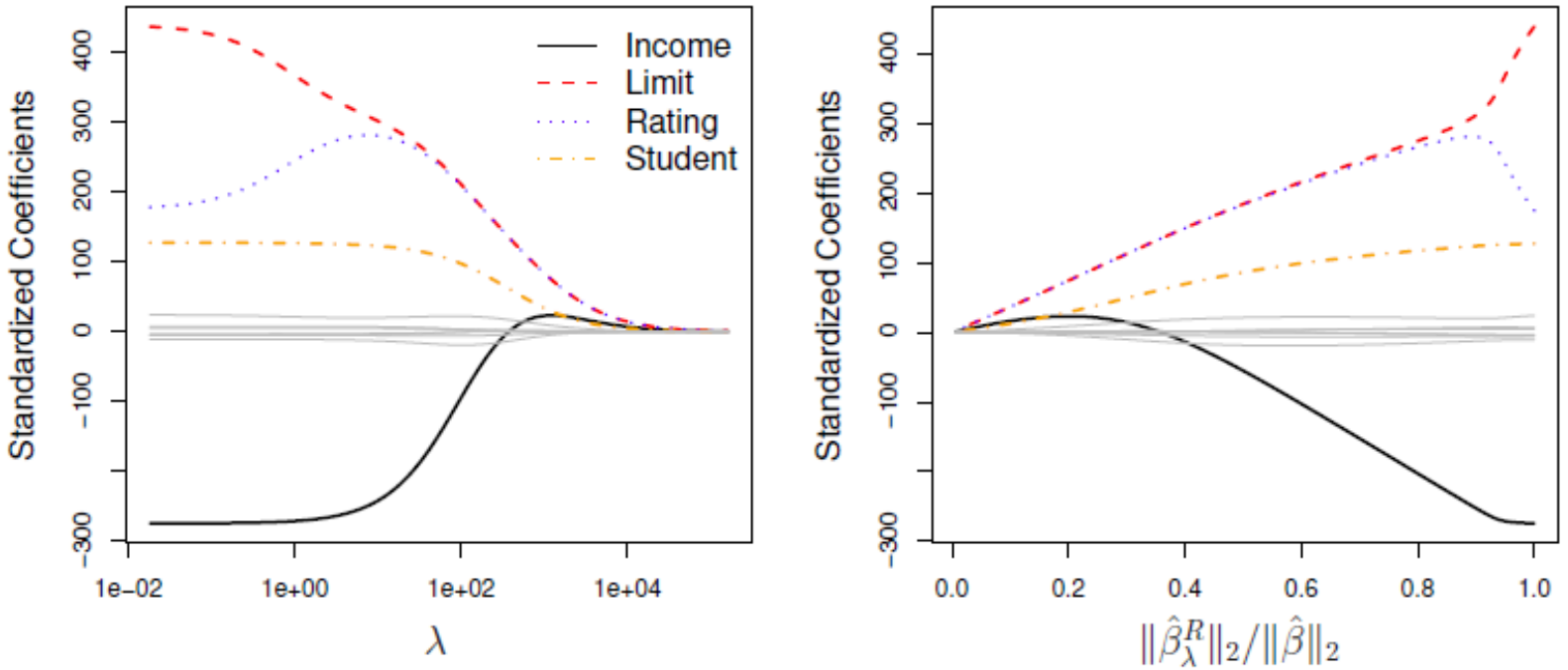


FIGURE 6.4. *The standardized ridge regression coefficients are displayed for the* `Credit` *data set, as a function of* λ *and* $\|\hat{\beta}^R_\lambda\|_2/\|\hat{\beta}\|_2$.

From James et al, Introduction to Statistical Learning

LASSO regression

In the general case, LASSO does not have a closed-form solution for the parameter vector, as the loss function is not differentiable.

However, algorithms exist to find the solution in an **efficient way**.

The effect of LASSO is to force some of the parameters to **be zero**, therefore it can be used as a feature selection method.

The higher λ, the more parameters in $\hat{\beta}_{(l)}$ will become zero.

Same as Ridge, LASSO allows for model training in the case $n < q$.

LASSO regression

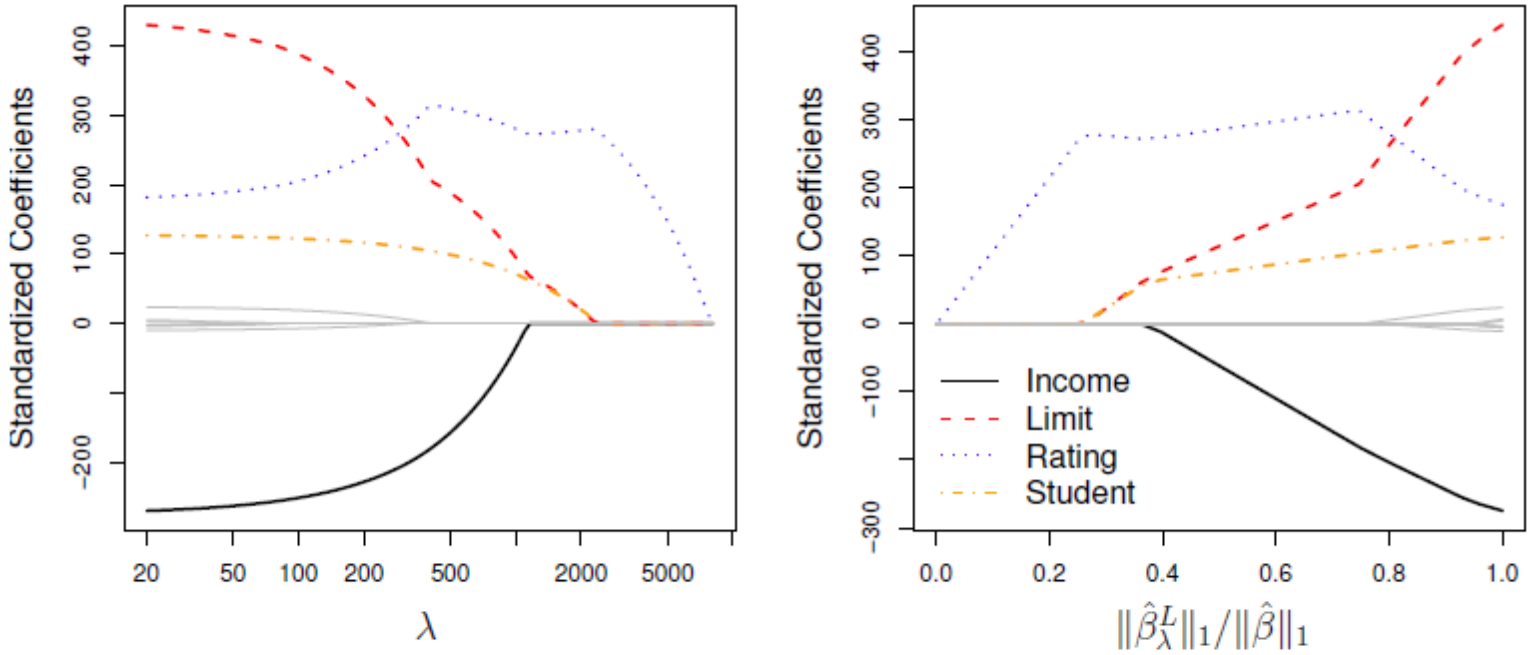


FIGURE 6.6. *The standardized lasso coefficients on the* `Credit` *data set are shown as a function of* λ *and* $\|\hat{\beta}_\lambda^L\|_1/\|\hat{\beta}\|_1$.

From James et al, Introduction to Statistical Learning

Hyperparameters

What is the best value of λ for a specific problem?

There is nothing in the training algorithm to tell.

The variable λ is an **hyperparameter**, that is a value in the model which cannot be optimized by the training algorithm but must be chosen by the developer.

A common technique to choose λ is cross-validation.

Ridge vs LASSO

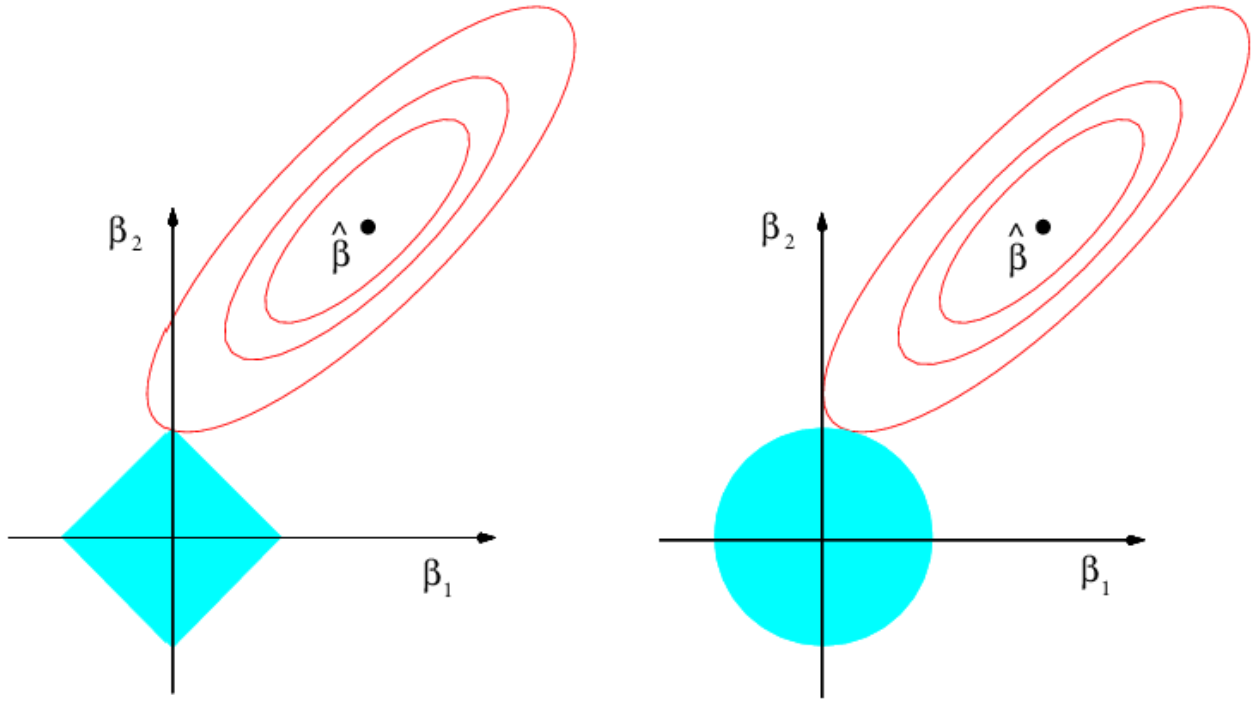


FIGURE 6.7. *Contours of the error and constraint functions for the lasso* (left) *and ridge regression* (right). *The solid blue areas are the constraint regions,* $|\beta_1| + |\beta_2| \leq s$ *and* $\beta_1^2 + \beta_2^2 \leq s$, *while the red ellipses are the contours of the RSS.*

From James et al, Introduction to Statistical Learning

Open Notebook in Colab

Logistic Regression

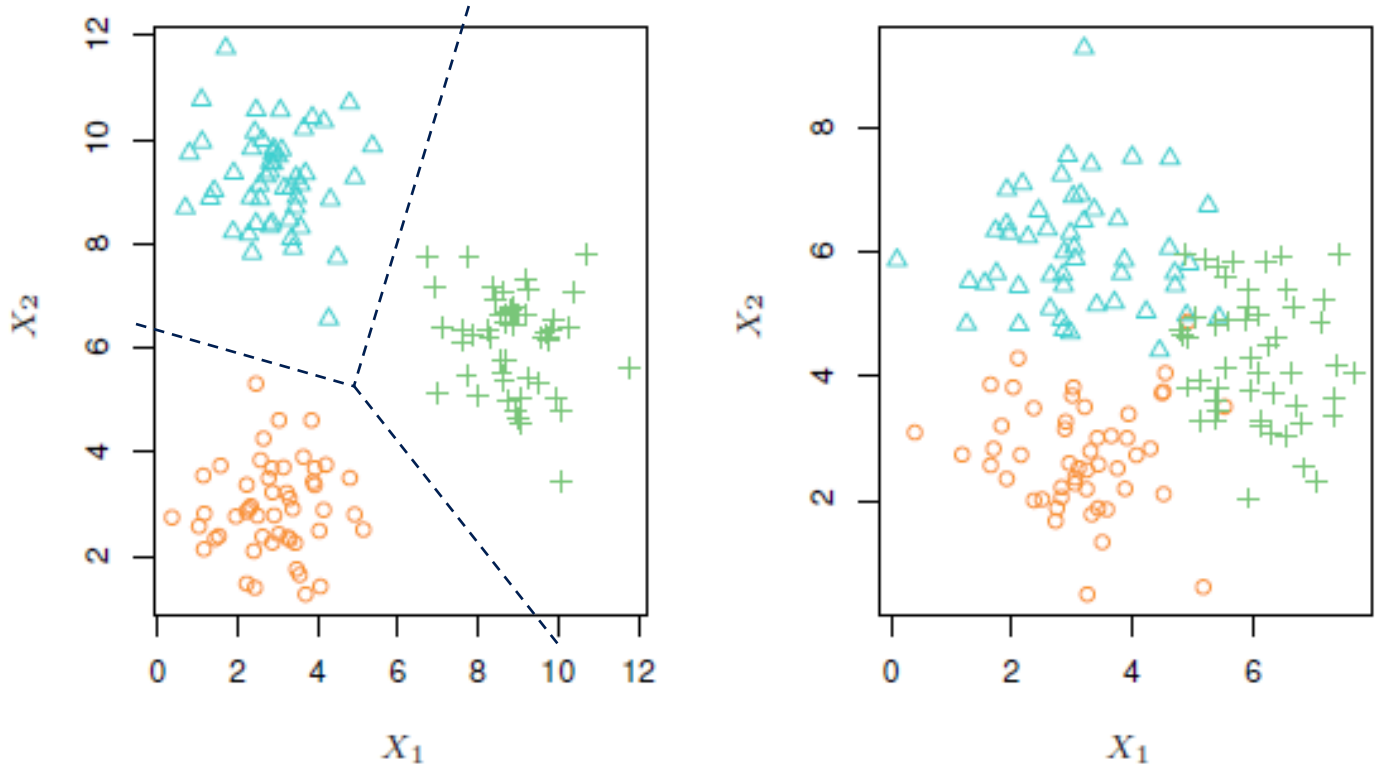
Context

Supervised learning: we are given both inputs and outputs, and the model should capture the relationship between them.

Assumptions:

- Classes are linearly separable in the input space
- All the samples are independent from each other
- No features are collinear

Linearly separable



From James et al, Introduction to Statistical Learning

Problem statement

We measure the weight, height, and sex of 25000 students aged 18.

For other students, we will be given the height and weight, but we will not be provided the sex (male / female).

We need to build a model to predict the sex (male / female) given the height and weight.

Height	Weight	Sex
167.1	51.3	M
181.6	61.9	M
176.3	69.4	F
173.3	64.6	M
172.2	65.5	F
…	…	…

About categorical variables

We know that models need numbers. Therefore, M/F must be converted to numbers.

How can we do it?

We can associate 1 to M and 0 to F, or vice-versa, arbitrarily.

However, using 0, 1, 2, 3, 4 for linear regression features is usually a **bad idea**. Why?

Discussion

Why isn’t a linear regression well-suited for a **classification** problem?

Why not linear regression?

We now have a quantitative output variable.

Why shouldn’t we use linear regression?

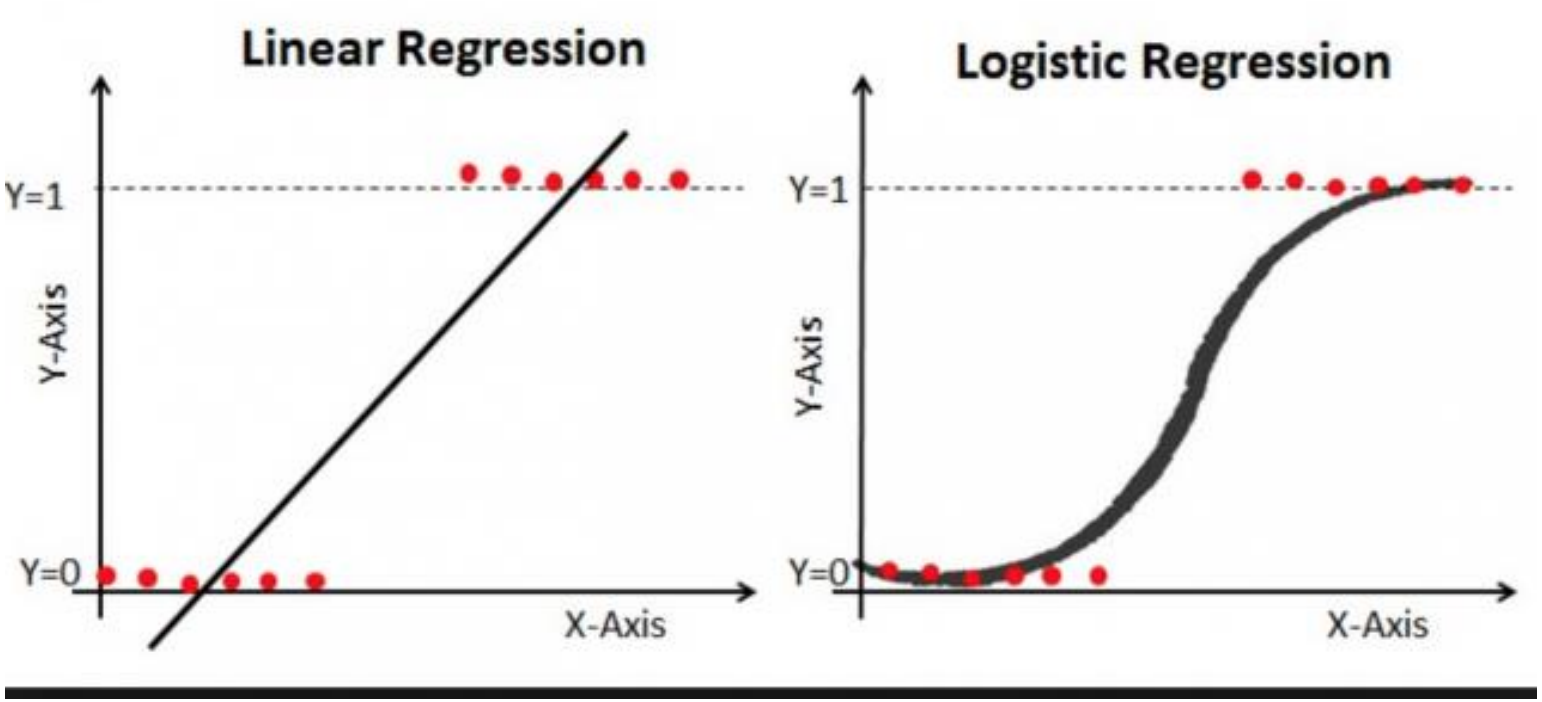
We want to predict the probability for each sample, that is a number between 0 and 1. A linear regression can predict every number.

We pick the model

$$\log \frac{P(Y = 1|X = x)}{P(Y = 0|X = x)} = \beta x$$

Logistic regression is a linear regression where the target are the **log-odds-ratios**, or logits.

Why not linear regression?



Example

We measure the weight, height, and sex of 25000 students aged 18.

We need to build a model to predict the sex (male / female) given the height and weight.

$n = 25000$: the number of samples

$y = \begin{pmatrix} 1 \\ 1 \\ 0 \\ \dots \end{pmatrix} \in \{0, 1\}^{25000}$: the output vector

$X = \begin{bmatrix} 167.1 & 51.3 & 1 \\ 181.6 & 61.9 & 1 \\ \dots & \dots & \dots \end{bmatrix} \in R^{25000,3}$: the input matrix, with 25000 rows and 3 columns

$\beta = \begin{pmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \end{pmatrix} \in R^3$: the parameter vector

Logistic regression training

$$\log \frac{P(Y = 1|X = x)}{P(Y = 0|X = x)} = \beta x$$

$$\mathrm{P(Y = 1|X = x)} = \frac{e^{\beta x}}{1 + e^{\beta x}}$$

Find β such that the ***probability of observing the training data is maximized***.

This is the idea of Maximum Likelihood.

The loss function of linear regression can also be interpreted as the solution of a Maximum Likelihood problem.

Logistic regression loss function

$$\mathrm{l}(\beta) = \prod_{i:y_i=1} P(\mathrm{Y}=1|\mathrm{X}=\mathrm{x}) * \prod_{i:y_i=0} 1 - P(\mathrm{Y}=1|\mathrm{X}=\mathrm{x})$$

The maximum of a function is the maximum of its log and the log of a product is the sum of the logs.

$$\mathrm{l}(\beta) = \sum_{i:y_i=1} \log P(\mathrm{Y}=1|\mathrm{X}=\mathrm{x}) + \sum_{i:y_i=0} \log(1 - P(\mathrm{Y}=1|\mathrm{X}=\mathrm{x}))$$

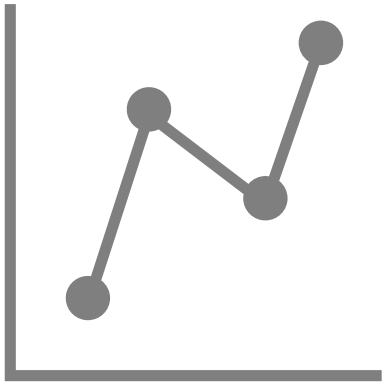
$$\mathrm{l}(\beta) = \sum_{i=1}^{n} y_i \log \frac{e^{\beta x}}{1+e^{\beta x}} + (1-y_i) \log \frac{1}{1+e^{\beta x}}$$

The loss function should be minimized, so we choose

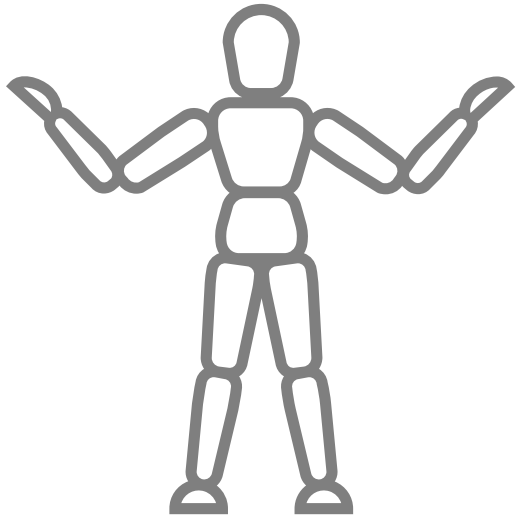
$$\mathrm{L}(\beta) = -\sum_{i=1}^{n} y_i \log \frac{e^{\beta x}}{1+e^{\beta x}} + (1-y_i) \log \frac{1}{1+e^{\beta x}}$$

The minimum can be found with numeric algorithms.

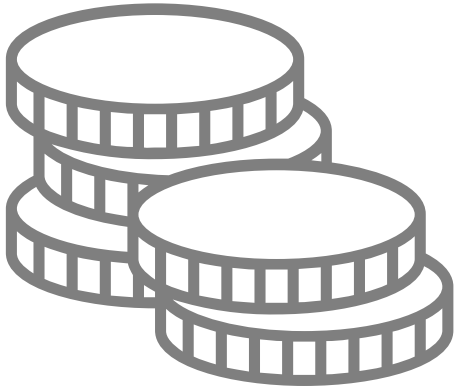
Data



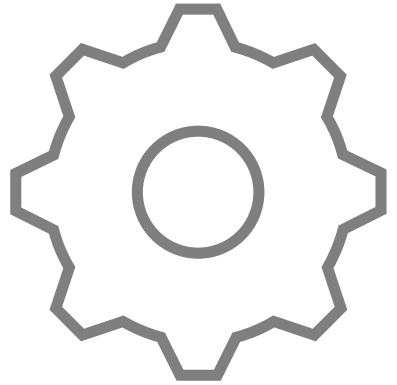
Model



Loss



Learning algorithm



Data

Height	Weight	Sex
167.1	51.3	M
181.6	61.9	M
176.3	69.4	F
173.3	64.6	M
172.2	65.5	F
…	…	…

Model

$$\mathrm{P}(\mathrm{Y}=1|\mathrm{X}=\mathrm{x})=\frac{e^{\beta x}}{1+e^{\beta x}}$$

Loss

$$\mathrm{L}(\beta)=-\sum_{i=1}^{n} y_i \log\frac{e^{\beta x}}{1+e^{\beta x}}+ +(1-y_i)\log\frac{1}{1+e^{\beta x}}$$

Learning algorithm

Numeric optimization of $L(\beta)$

Interpreting logistic regression coefficients

$$\log \frac{P(Y = 1|X = x)}{P(Y = 0|X = x)} = \beta x$$

The coefficients of a logistic regression bring information about the **log-odds**.

For each unit increase of x:

- If β is positive, the odds of the sample x being of class 1 increases by a factor $e^{\beta x}$
- If β is negative, the odds of the sample x being of class 1 decreases by a factor $e^{\beta x}$

Regularization

Exactly as in the **linear regression**, the loss function of the logistic regression can also be updated with a **penalization** on the size of the coefficients.

Similar to the case of linear regression, a ridge penalization shrinks all the parameters towards zero, while a LASSO penalty makes some coefficients exactly zero.

Similar to the case of linear regression, regularization is particularly useful when the number of samples is lower than the number of features.

Open Notebook in Colab

Tree Models

Why going beyond linear models?

Linear models are **simple**, **interpretable**, and quite **stable**.

However, they struggle to capture complex relationships.

And they require a lot of manual feature engineering.

Sometimes, we may need more **flexible** models.

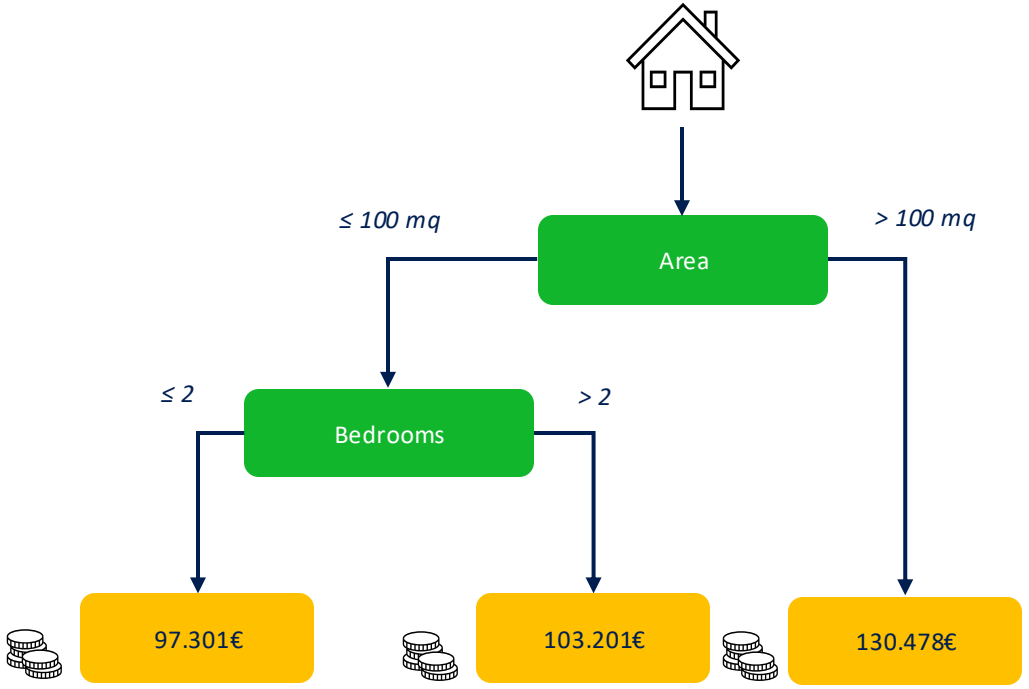
A simple regression tree

We want to predict the **price of a house**.

We collect data for 1000 houses in the same region.

Assume we consider only two features: the area and the number of bedrooms.

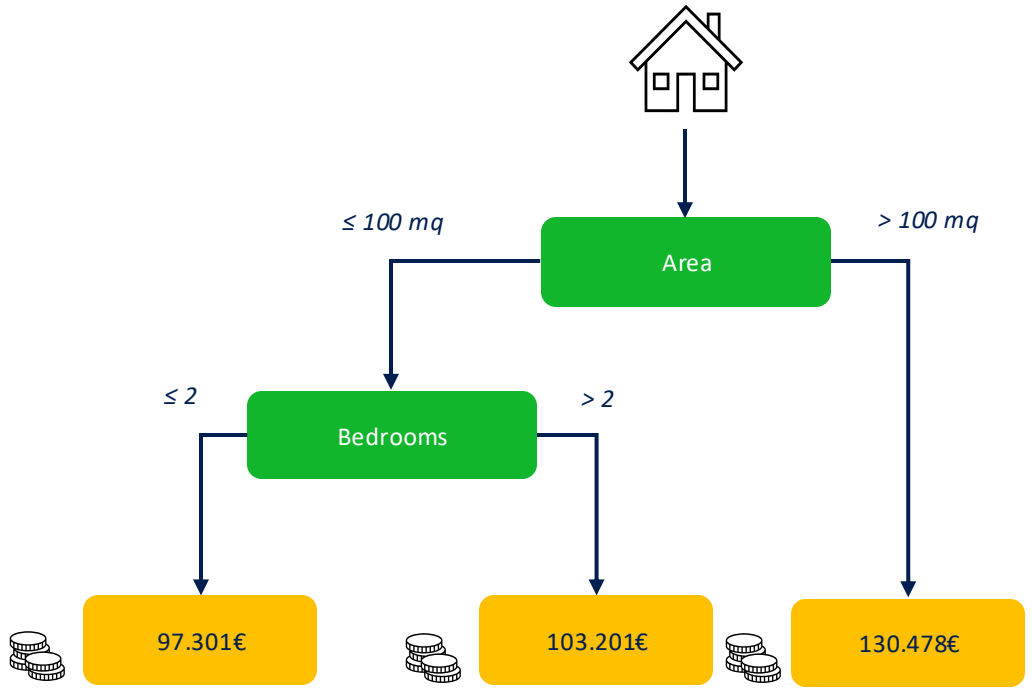
We can build a very simple tree.



Tree jargon

The green boxes are called internal nodes. They split the input space in homogeneous subspaces.

The yellow boxes are called leaves (or terminal nodes). Each leaf represent a region associated with a particular prediction.

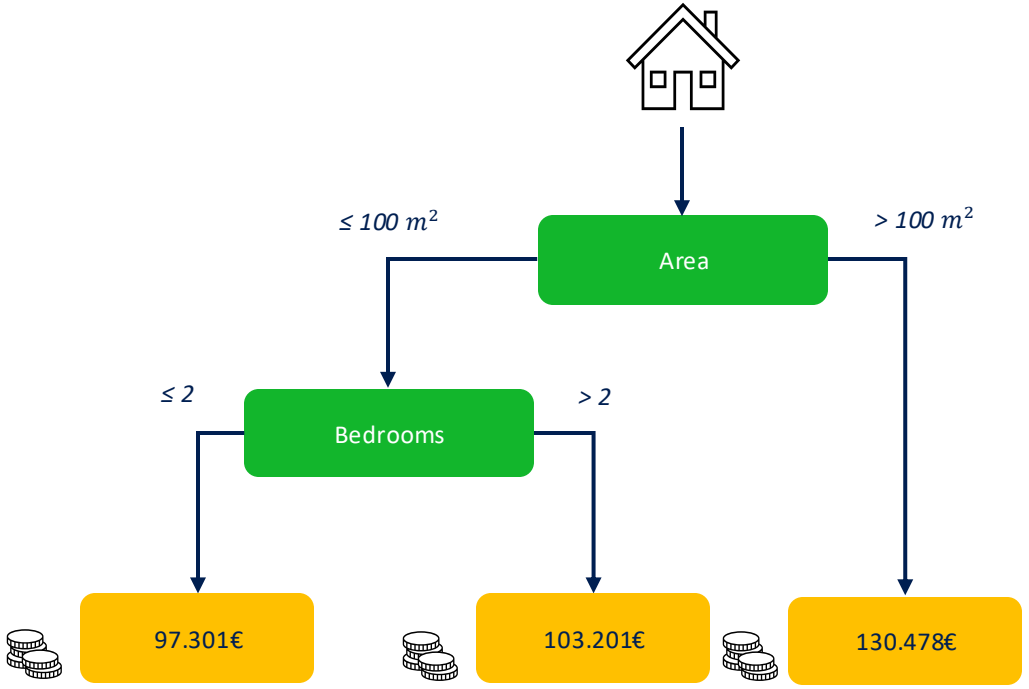


Interpreting the tree

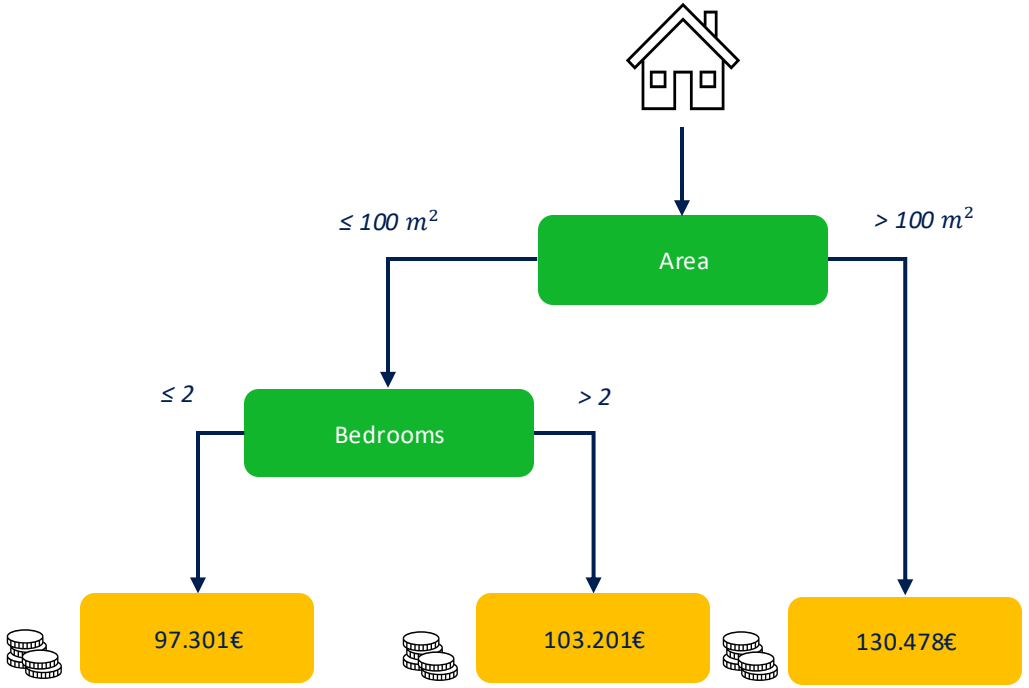
In order to understand the price of a house, we must first look at its area.

If it is above 100 m^2, we can provide a good-enough estimation without applying further rules.

If it is below 100 m^2 we need to check the number of bedrooms. Then, we can provide an estimation.



Splitting the input space?

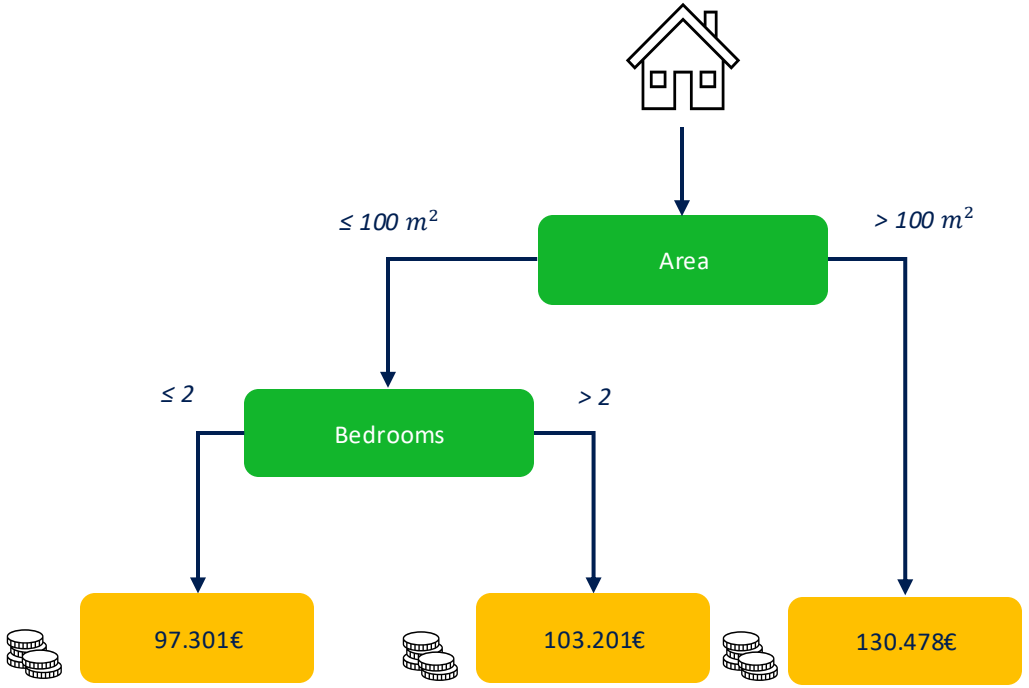


How to grow a tree

The tree is just a suitable representation.

The real purpose is to divide the input space into a **finite number r of regions**.

The input space is the set of all possible values for the q features. Therefore, it is normally a subset of R^q.



How to grow a tree

We want the regions to be non-overlapping and we want the samples inside the same region to have outputs as **similar** as possible.

We are going to take the average of the training samples in the same region and use it as a **prediction** for each new sample falling in that region.

How to grow a tree

Theoretically, the regions could be of any shape. Practically, they are **boxes** (i.e., they are delimited by constants).

So, we want to find the regions $R_1, \dots, R_r$ such that

$$L = \sum_{j=1}^{r} \sum_{x_i \in R_j} \left(\hat{y}_{R_j} - y_i \right)^2$$

is **minimal**.

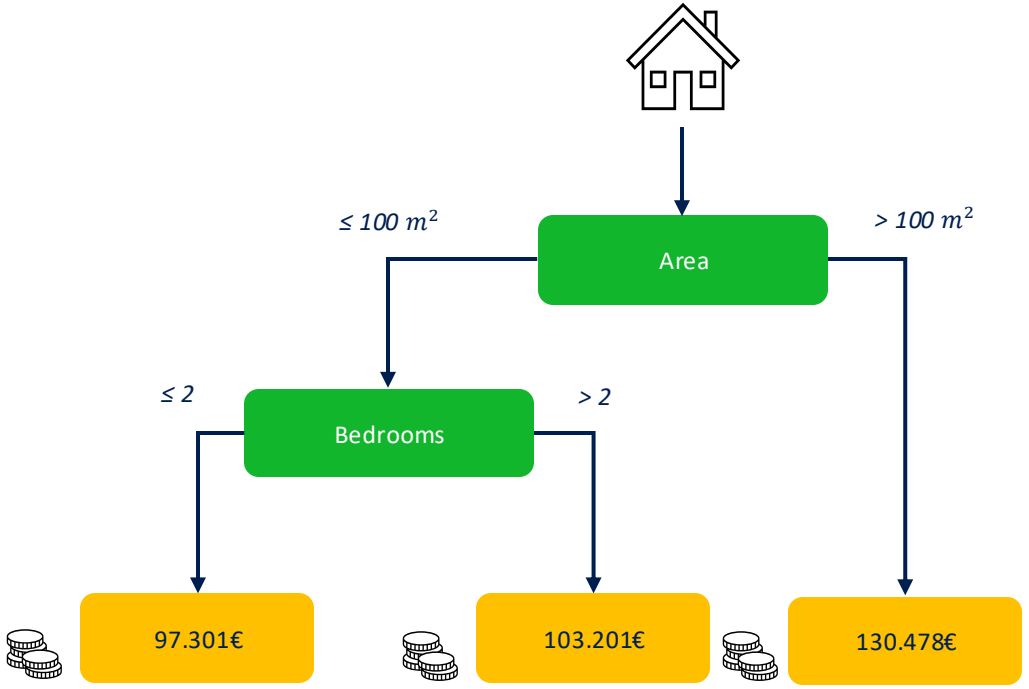
How to grow a tree

However, there is **no efficient way** to solve the problem over all possible regions. The problem of learning a globally optimum tree is **NP-complete**.

Therefore, a **top-down greedy approach** is usually adopted.

Top-down? The input space is split recursively, by starting from the top of the tree and going down.

Greedy? At each split, the optimal constant is chosen, without caring for subsequent splits.

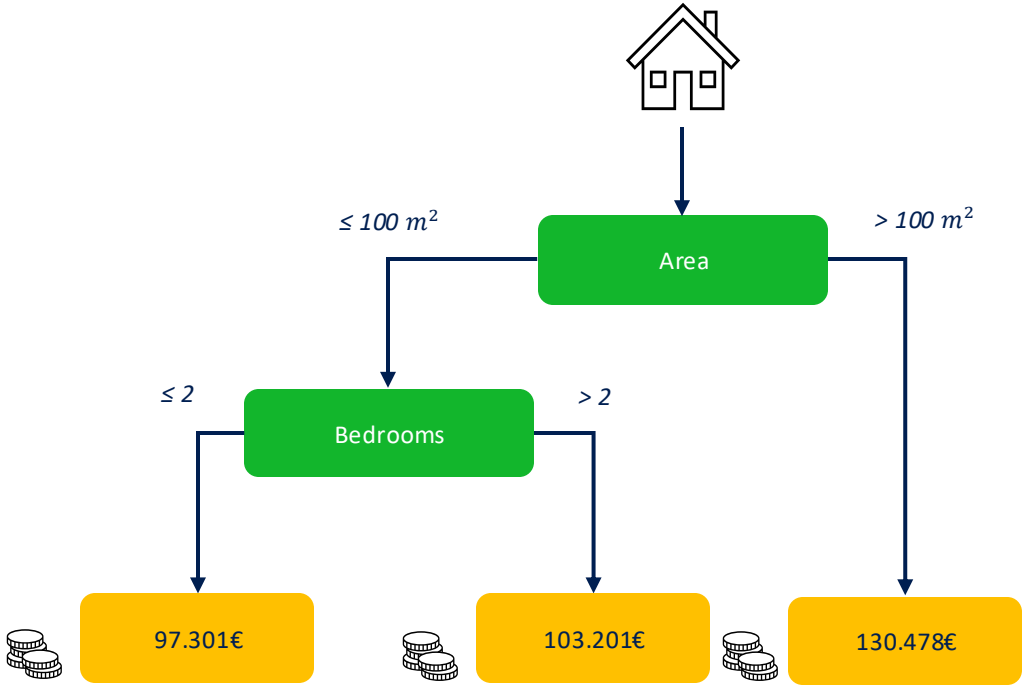


How to grow a tree

So, we start from the entire input space. For each feature, we compute the constant which leads to the maximum drop in the loss function. This can be done efficiently.

The feature which achieves the **maximum overall drop** in the loss function is considered and the split is performed.

Then, the process is repeated for each of the regions just created until a **stopping criterion** is met.



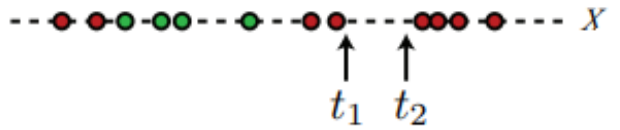
Splitting?

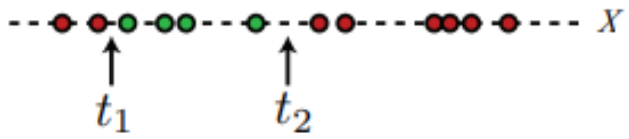
There is a finite number of features and, for each categorical feature, a finite number of possible splits. So, we can simply **try them all**!

But what about numerical features?

Only a **finite number** of thresholds is meaningful – the midpoints of the gap between the ordered values of each variable.

Moreover, other heuristics can be applied: in classification trees, only gaps between samples of different classes matter.





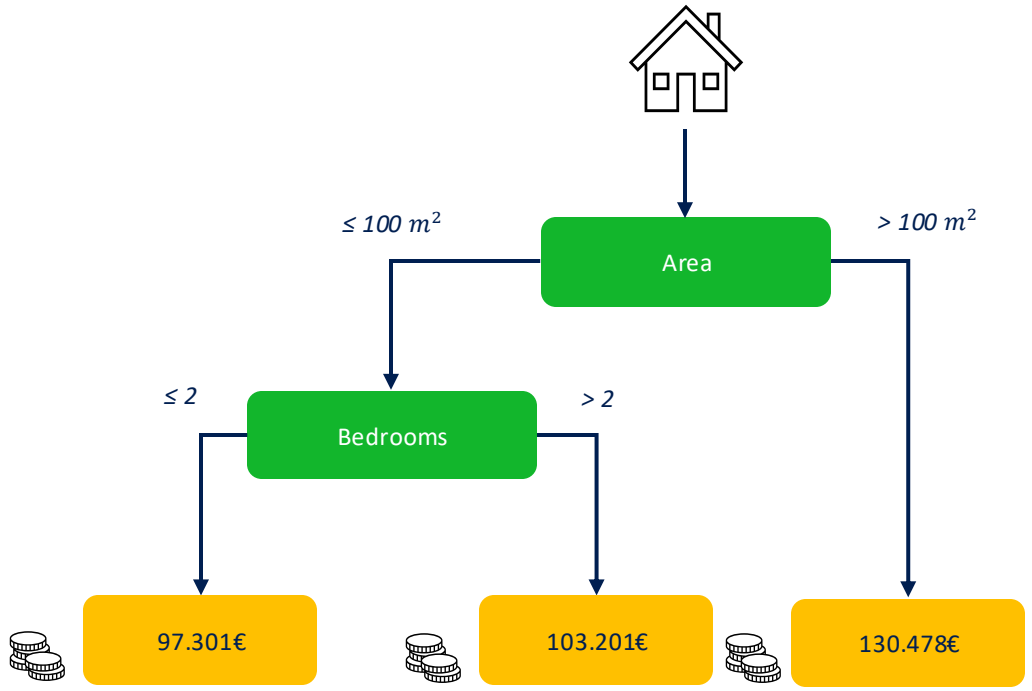
Stopping criterion?

Single-node stopping rules:

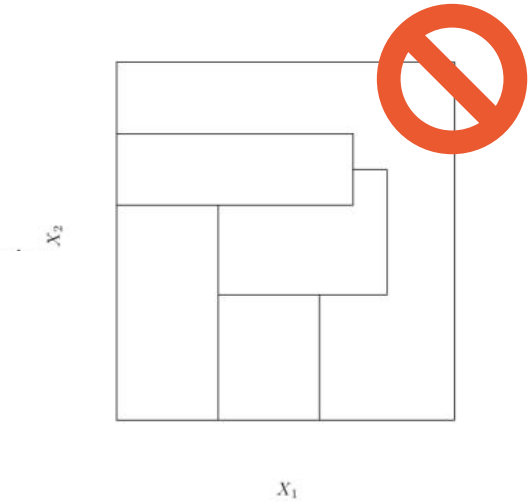
- Don’t split a node if all the samples have the **same target**.
- Don’t split a node if all the samples have the **same features**.

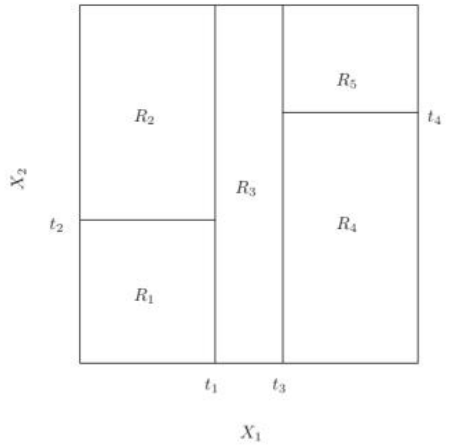
Common stopping criteria are:

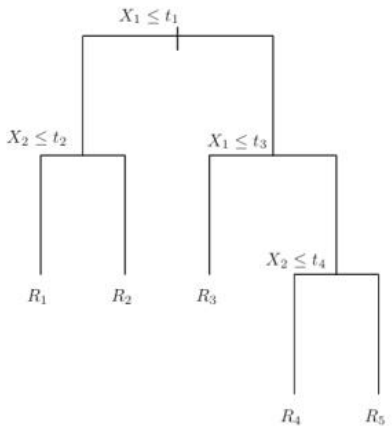
- A maximum depth of the tree
- A minimum number in each leaf
- A minimum number of samples to be considered for a split
- A minimum drop in loss function to be considered for a split

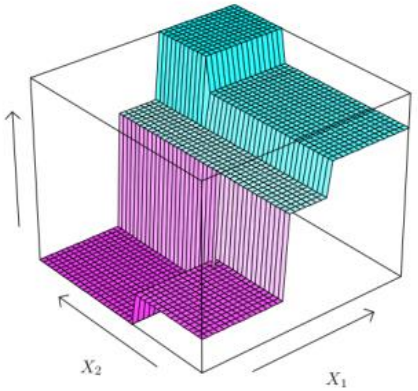


A tree model









From https://hastie.su.domains/MOOC-Slides/trees.pdf

Pruning trees

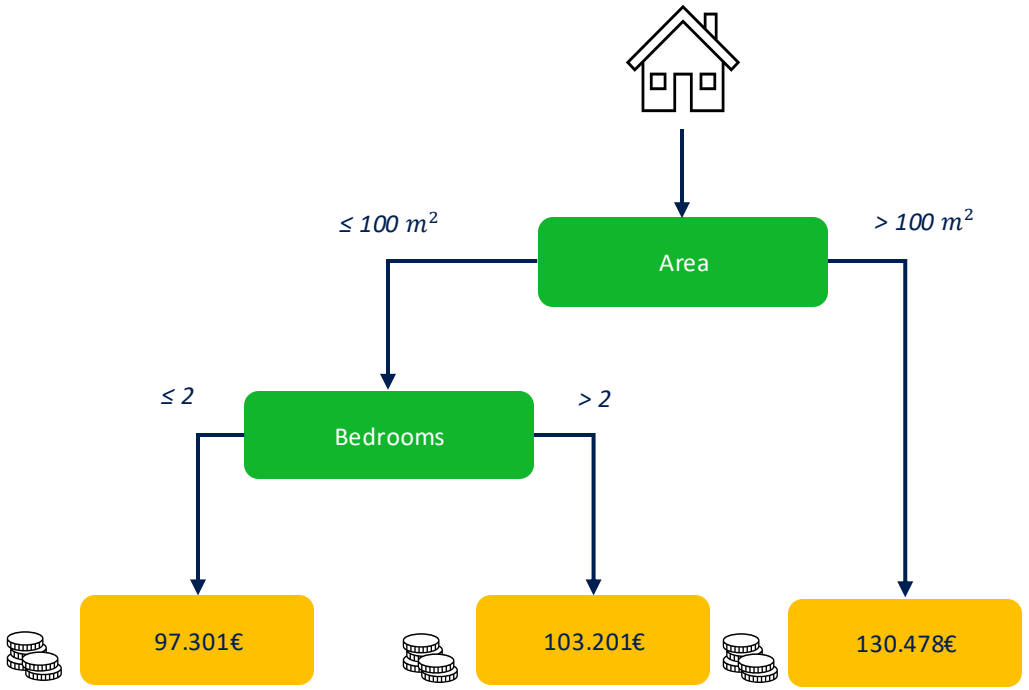
A common strategy to improve the performance of a tree model is to let it overgrow and then perform **pruning**.

Assume you have grown a deep and large tree T_0. Choose some values for $\lambda \in R$. To each value of λ a pruned tree T can be associated such that the following function is minimized:

$$\sum_{j=1}^{l} \sum_{x_i \in R_j} \left(\hat{y}_{R_j} - y_i \right)^2 + \lambda l$$

where l is the number of leaves in the reduced tree T.

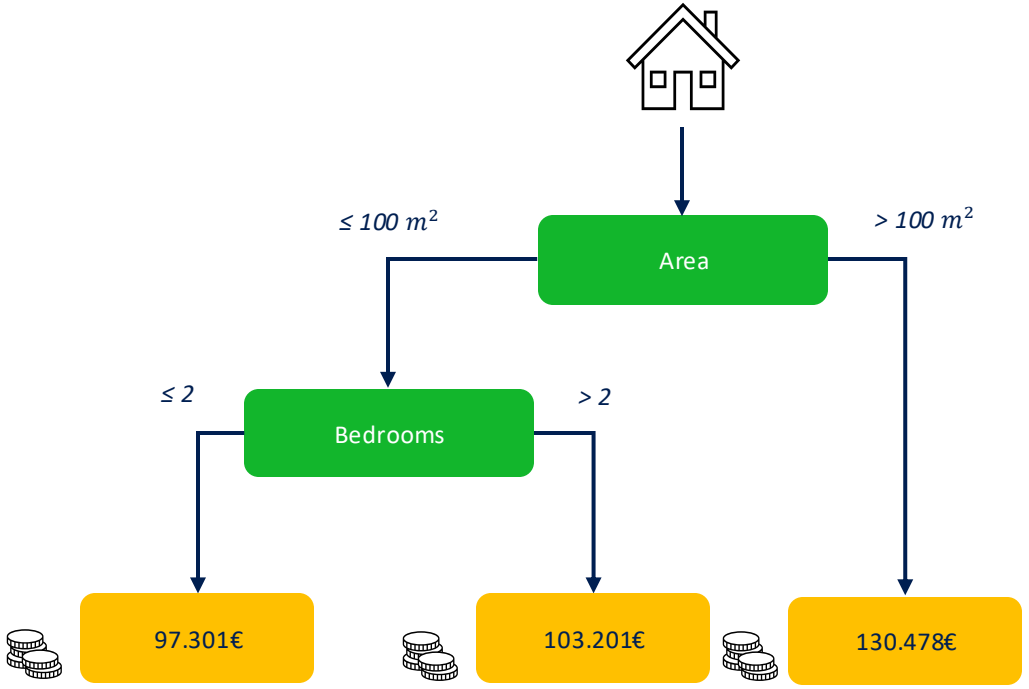
Pruning can be seen as a form of regularization.



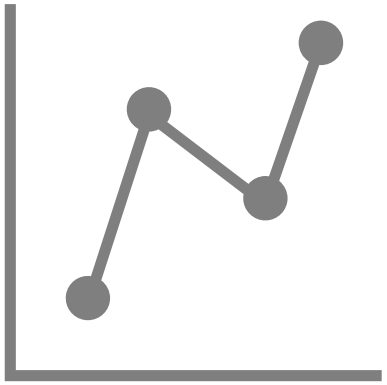
Pruning trees

As the regularization parameters in LASSO or Ridge, λ here is also an **hyperparameter**, that is a value that the training algorithm cannot optimize.

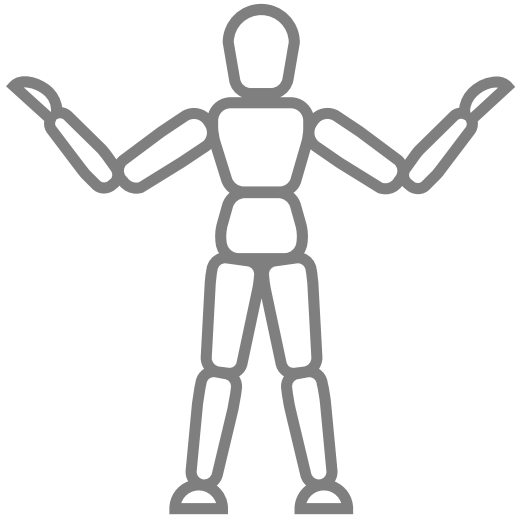
It must be picked by other methods, such as cross-validation.



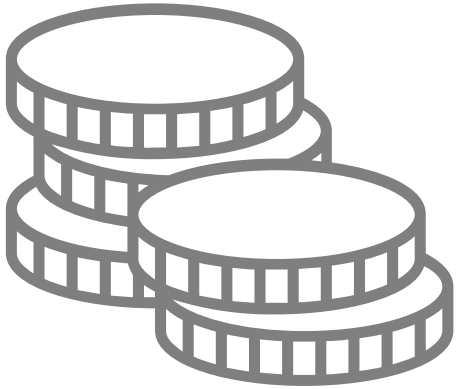
Data



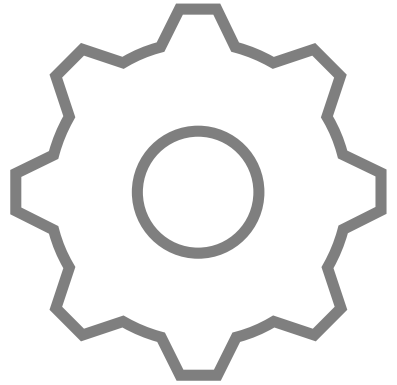
Model



Loss



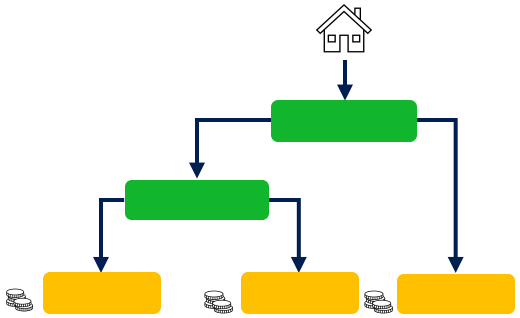
Learning algorithm



Data

Area	Bedrooms	Price
102	2	150k
64	1	72k
72	2	70k
80	3	112k
110	3	198k
…	…	…

Model



Loss

$$L = \sum_{j=1}^{r} \sum_{x_i \in R_j} \left(\hat{y}_{R_j} - y_i \right)^2$$

Learning algorithm

For all practical purpose, a greedy, top-down algorithm.

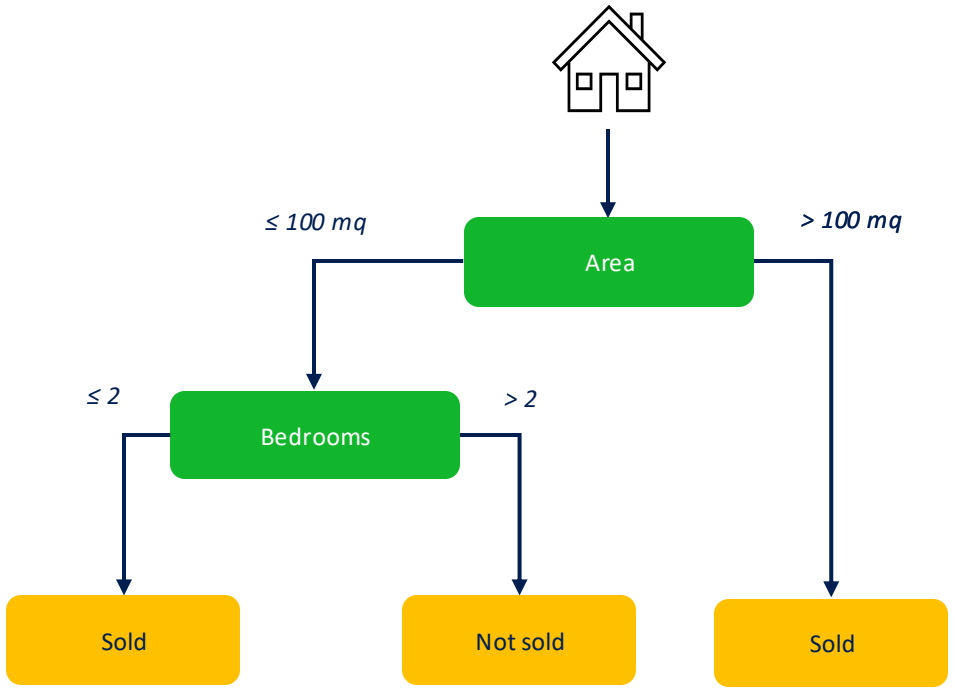
A simple classification tree

We want to predict **whether a house** will be sold in the next 6 months.

We collect data for 1000 houses in the same region.

Assume we consider only two features: the area and the number of bedrooms.

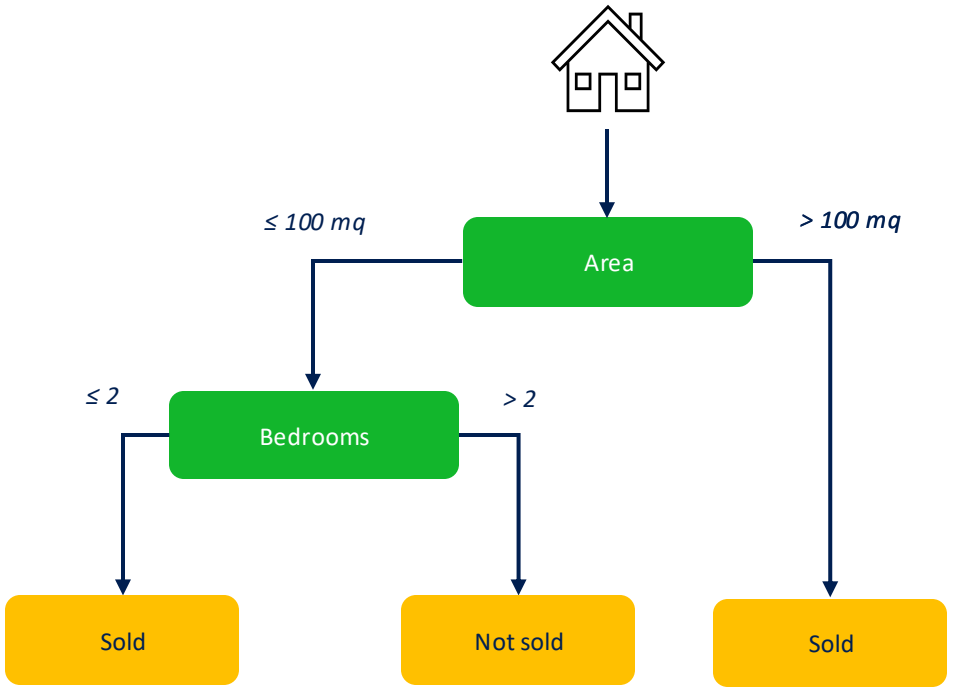
We can build a very simple tree.



Classification trees

They work in a very similar way to the regression trees, with 2 key differences:

- the **loss function**
- the prediction is the **most common class** for the train set in the region where a new sample falls



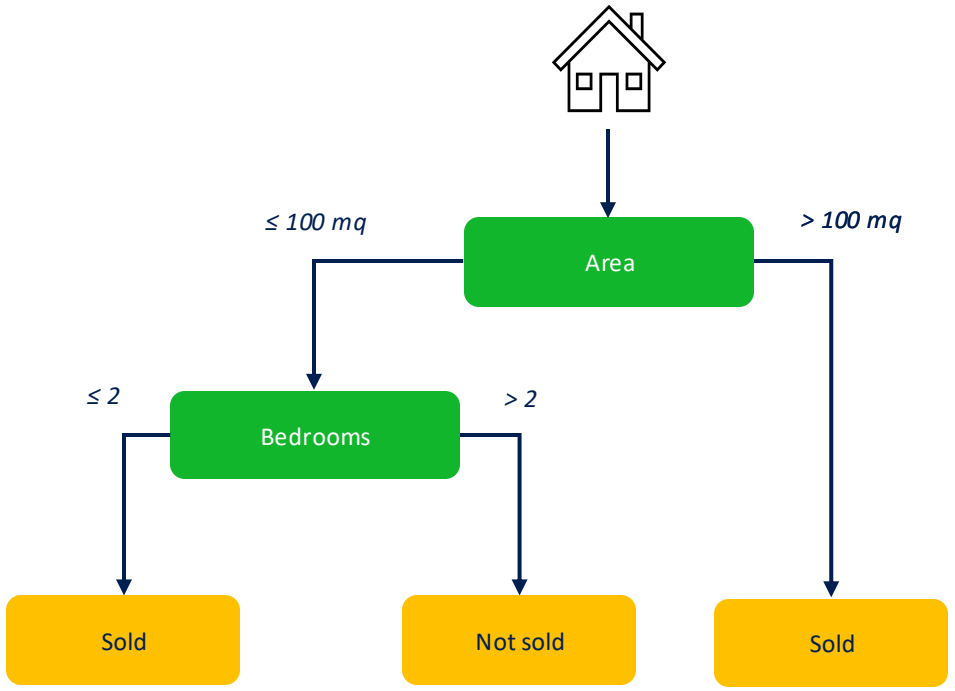
Loss function

The squared error is no longer a good loss function.

We need something that measures the homogeneity of the classes of the samples.

Two common choices are:

- Gini index
- Entropy



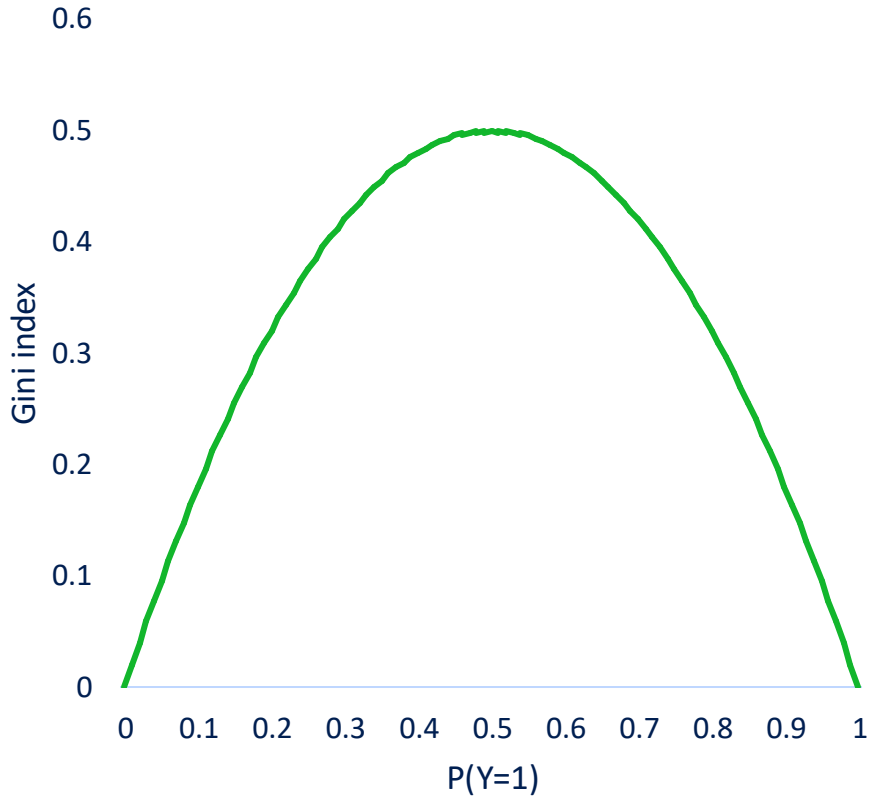
Gini index

For a random variable Y with k classes $c = 1, \dots, k$ the Gini index is:

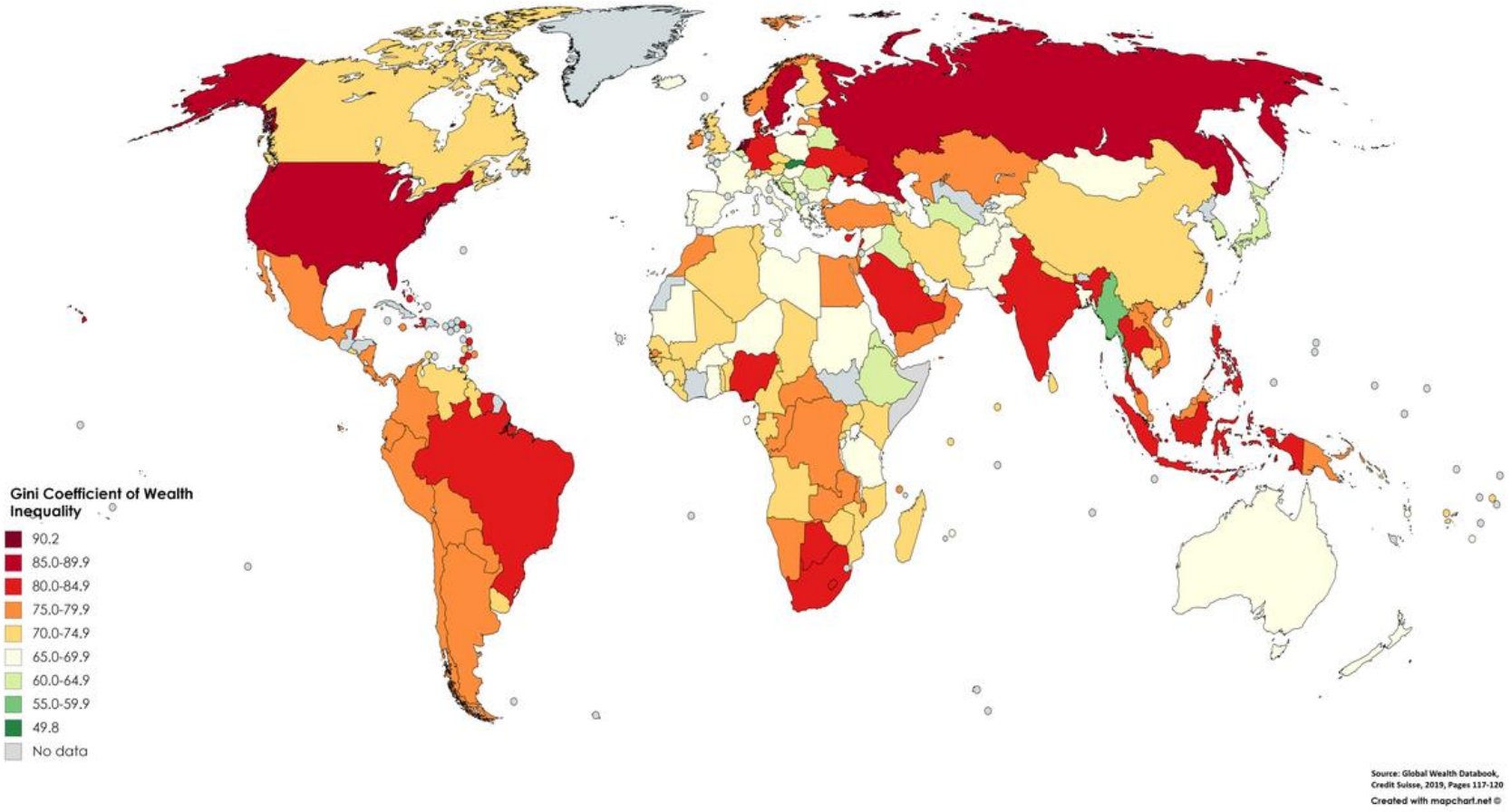
$$G = \sum_{c=1}^{k} P(Y = c)\,(1 - P(Y = c))$$

In a tree, the probability $P(Y = c)$ is estimated as the fraction of samples of class c in region R_j and then Gini indexes are summed over all the regions.

Like variance, the Gini measures how much **variability** there is in a distribution. However, it only applies to categorical variables.



Gini index – off topic



From https://en.wikipedia.org/wiki/Gini_coefficient

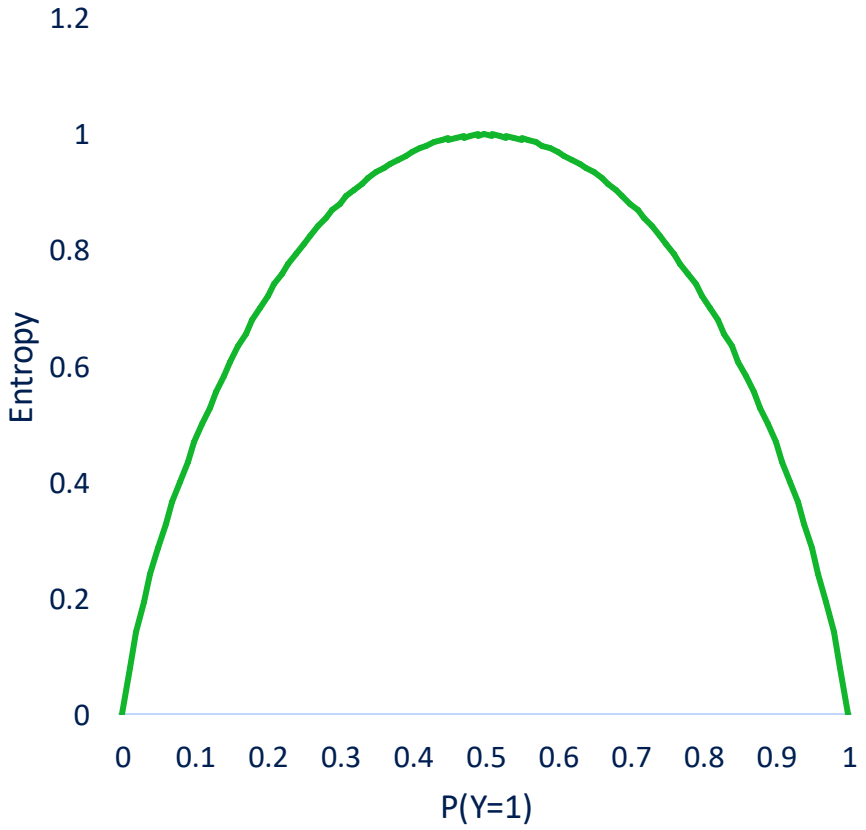
Entropy

For a random variable Y with k classes $c = 1, \dots, k$ the entropy is:

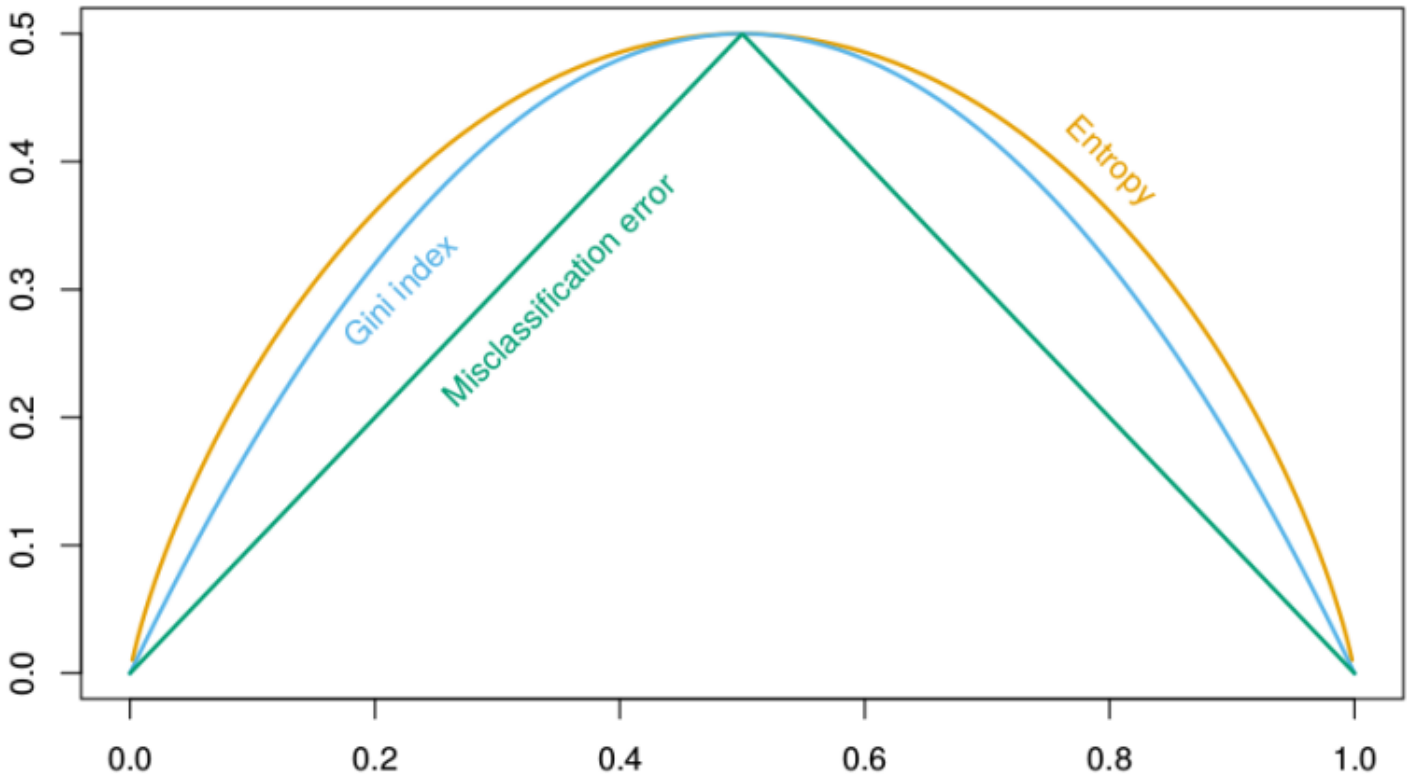
$$H = -\sum_{c=1}^{k} P(Y = c) \log_2 P(Y = c)$$

Similar to the Gini index, it measures the **variability** of a **categorical** distribution.

Interpretation: H is the expected minimum number of bits to encode a randomly drawn value from the random variable Y.

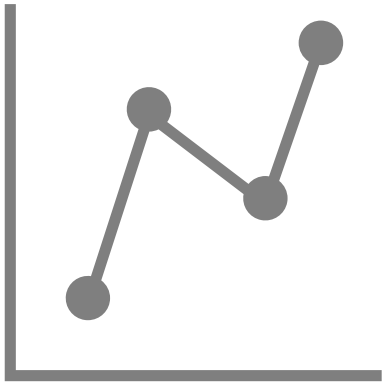


Why Gini and entropy?

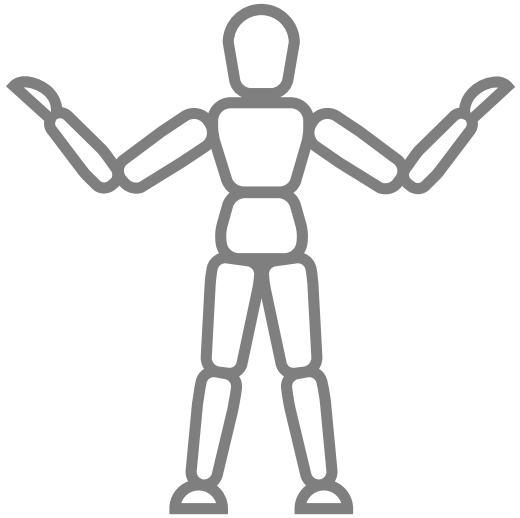


Both Gini index and entropy push for values **very close to 0 or 1** – that is for very homogeneous leaves.

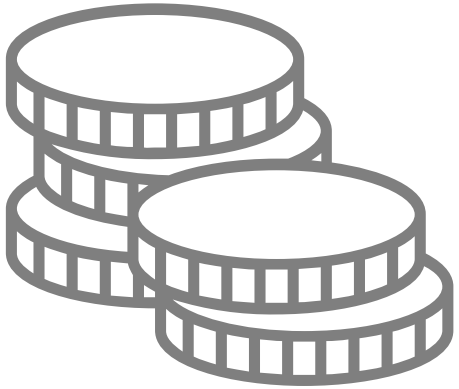
Data



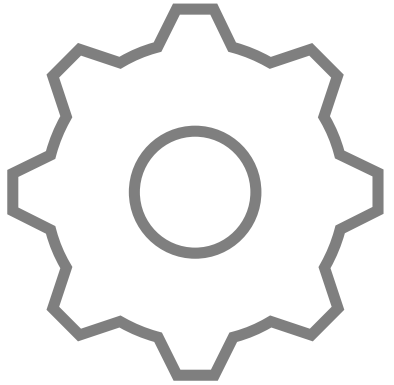
Model



Loss



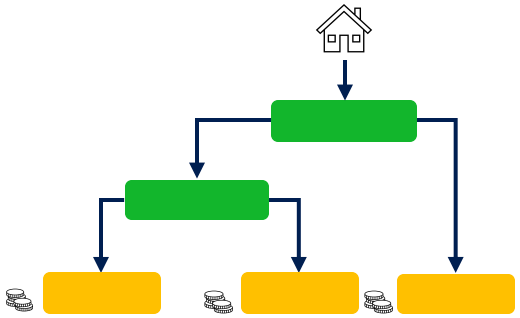
Learning algorithm



Data

Area	Bedrooms	Outcome
102	2	Sold
64	1	Not sold
72	2	Sold
80	3	Sold
110	3	Not sold
…	…	…

Model



Loss

Gini index or entropy

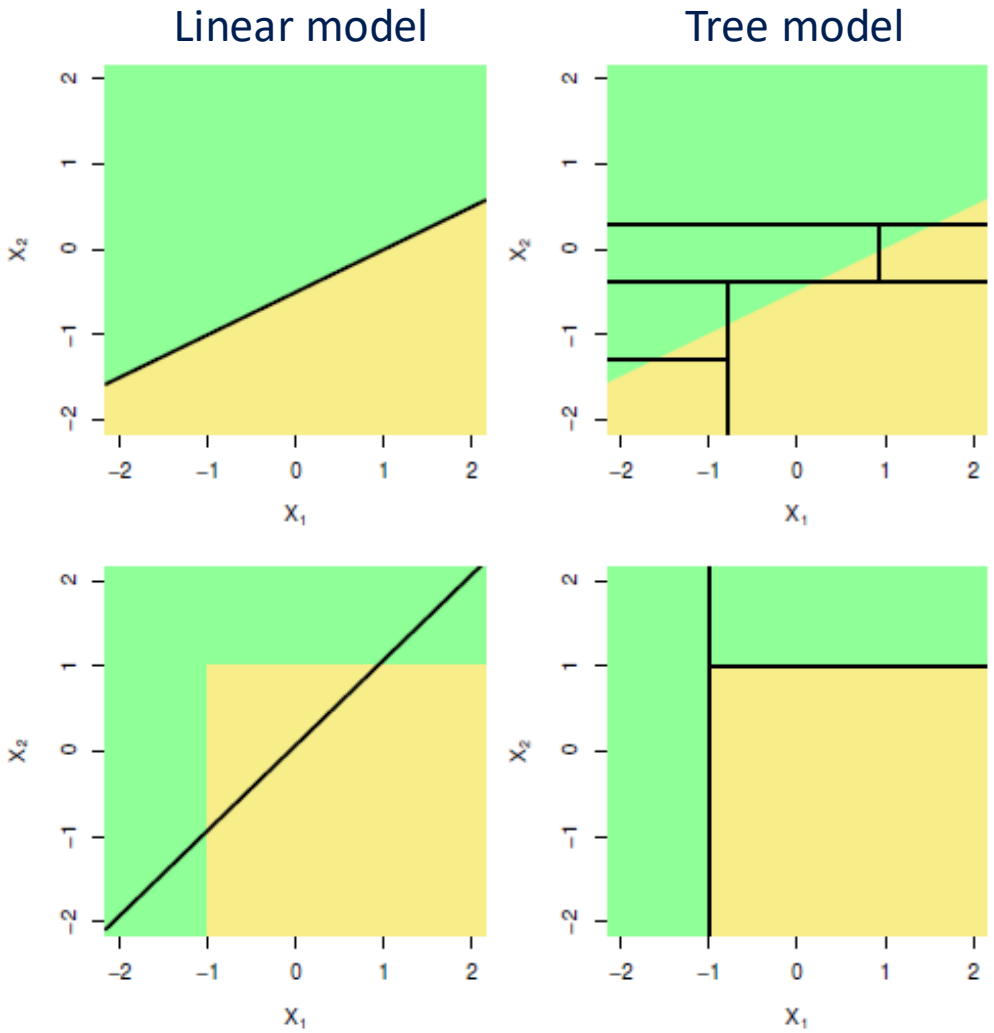
Learning algorithm

For all practical purpose, a greedy, top-down algorithm.

Tree vs linear models

The real decision boundary is linear

The real decision boundary is a box



From James et al, Introduction to Statistical Learning

Why trees?

- **Easy** to interpret and explain
- Can handle both quantitative and qualitative features
- Can predict both quantitative and qualitative outputs
- Quite computationally efficient

Why not trees?

Trees have a very **large variance**.

This means that small changes in the train set lead to big changes in the models.

Due to this issue, in many test cases, trees show an overall **poor performance** compared to other models.

Go to https://scikit-learn.org/stable/modules/tree.html

From the Last Lecture…

1. Business Understanding

- Workshop
- Evaluation metrics
- Literature review

2. Data Gathering

- Source discovery
- Data preparation
- Quality assessment

3. Data Exploration

- Explorative analysis
- Data cleaning

4. Modelling

- Feature engineering
- Model design
- Model implementation

5. Evaluation

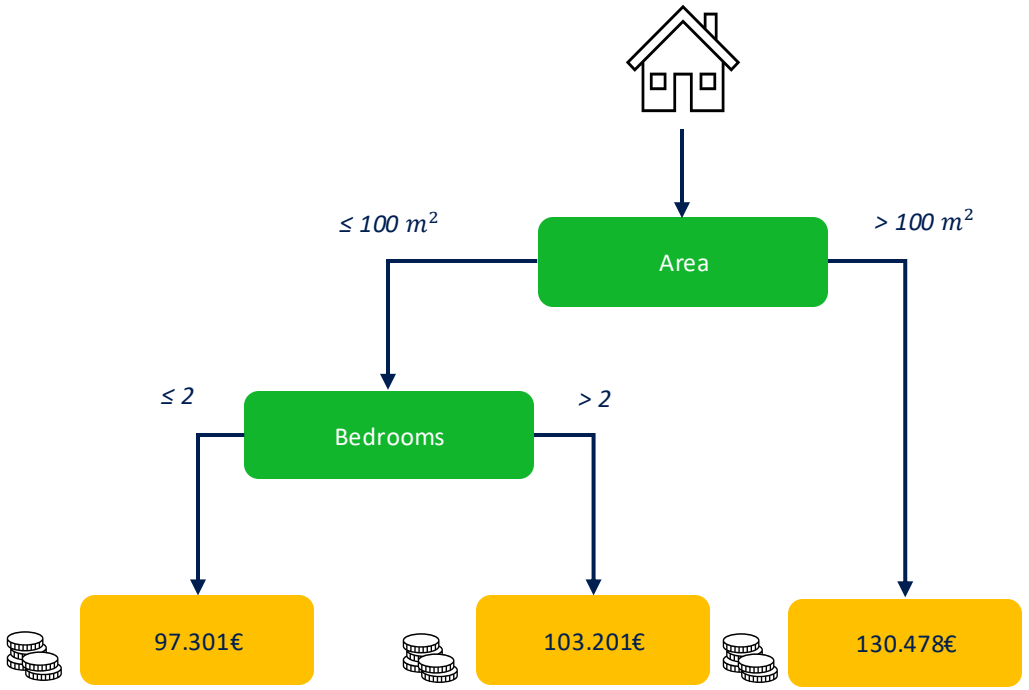
- Performance evaluation
- Time evaluation

6. Deployment

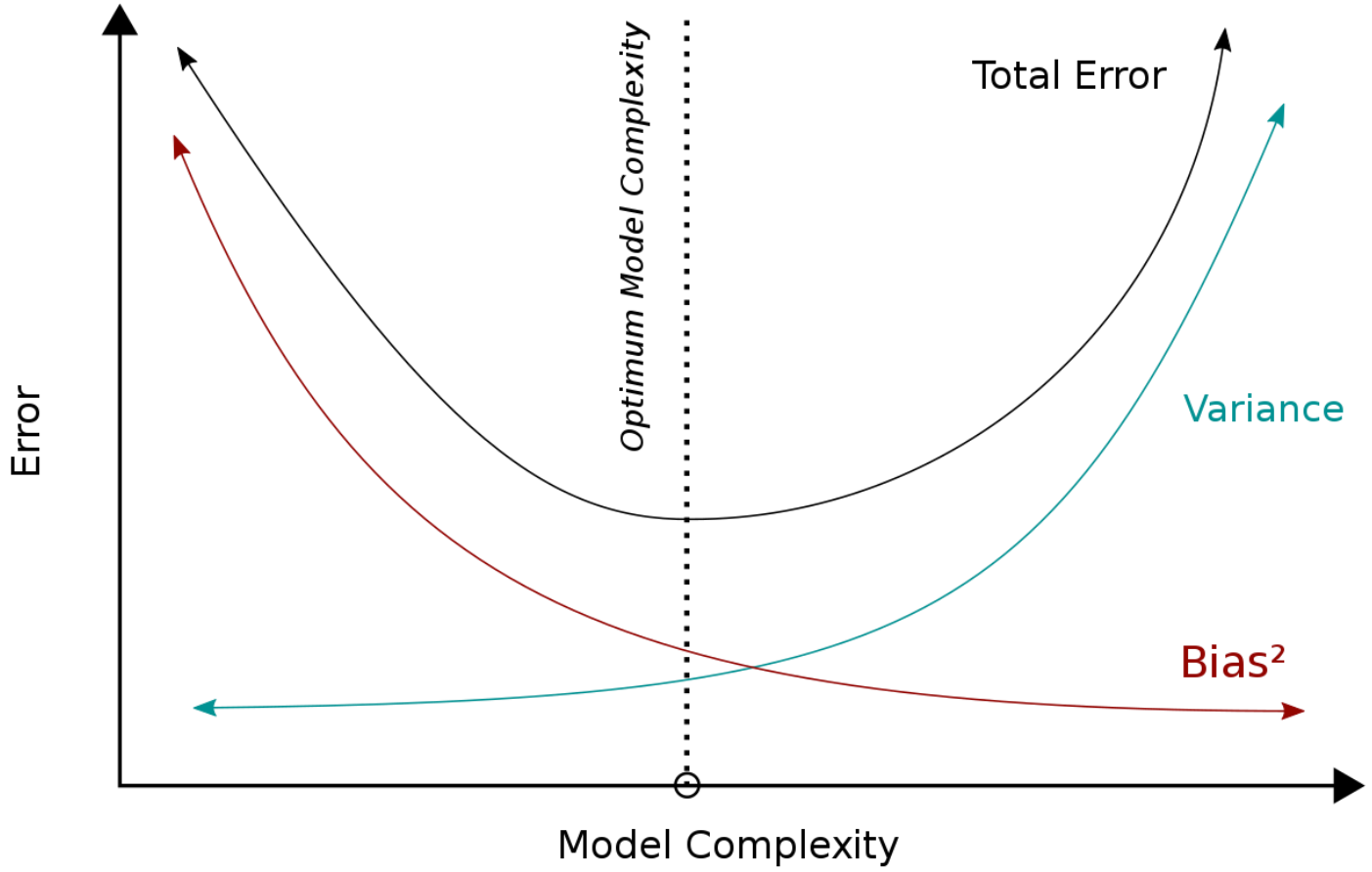
- Execution scheduling
- Performance tuning
- Integration

Trees


130.478€
100 m^2
Area
97.301€
103.201€
0
2
Bedrooms


≤ 100 m^2
Area
> 100 m^2
≤ 2
Bedrooms
> 2
97.301€
103.201€
130.478€

Bias-variance tradeoff



From https://en.wikipedia.org/wiki/Bias%E2%80%93variance_tradeoff

Random Forest

From statistics…

Let $X_1, \dots X_n$ be independent random variables with variance σ^2.

Then, their average

$$\frac{1}{n}\sum_{i=1}^{n} X_i$$

Has variance $\frac{\sigma^2}{n}$.

Averaging independent random variables reduces the variance.

So, why don't we average the predictions of trees?

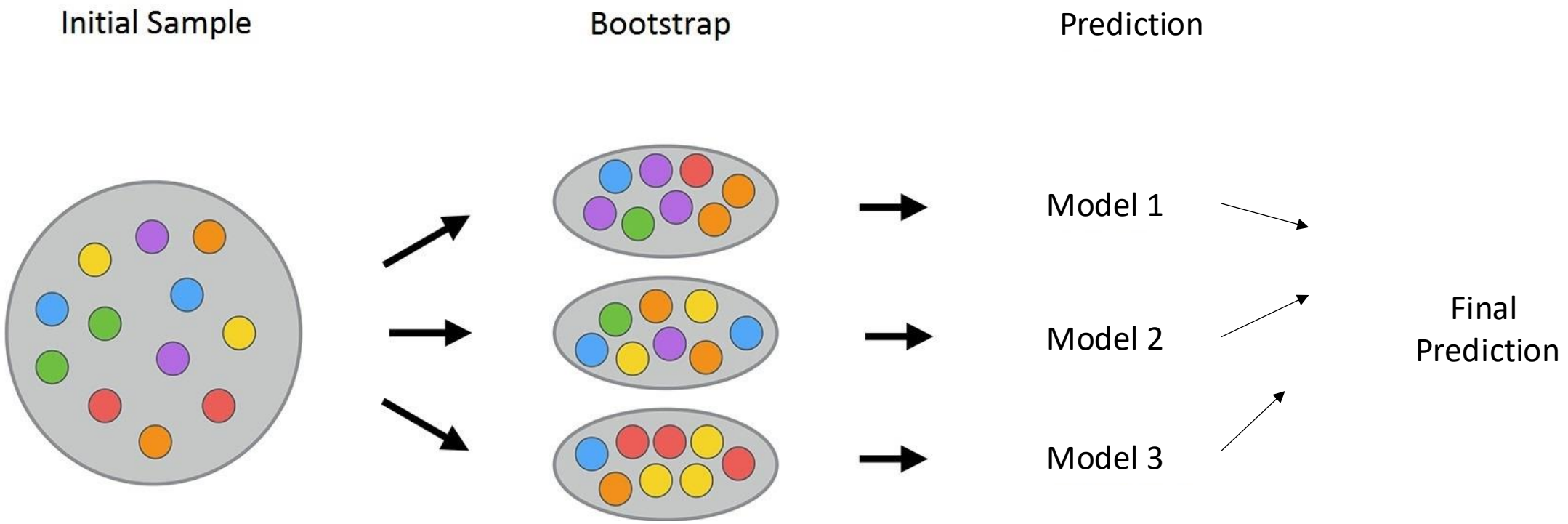
From statistics…

Unfortunately, we must fit **many models** on a **single dataset**.

Worst, the models should be independent – or at least not highly correlated.

Bootstrap to the rescue!

Bootstrap


Initial Sample
Bootstrap
Prediction
Model 1
Model 2
Model 3
Final
Prediction

Bootstrap

From a single original dataset of size n we create k dataset of the same size.

The samples for each bootstrapped dataset are drawn from the original dataset **with repetition**.

One can show that, for large n, each bootstrapped dataset contains about $1 - \frac{1}{e} \approx 63\%$ of unique samples.

So, we have about 37% of data to be used as test set for each tree.

Bootstrap

Why 63% of the original samples?

Consider a specific sample from the original dataset of size n. The probability that it is not selected in a **random draw** is:

$$1 - \frac{1}{n}$$

But to create our bootstrapped dataset we need to perform $\boldsymbol{n}$ **draws**. So, the probability that the observation is not selected for n times is:

$$\left(1 - \frac{1}{n}\right)^n$$

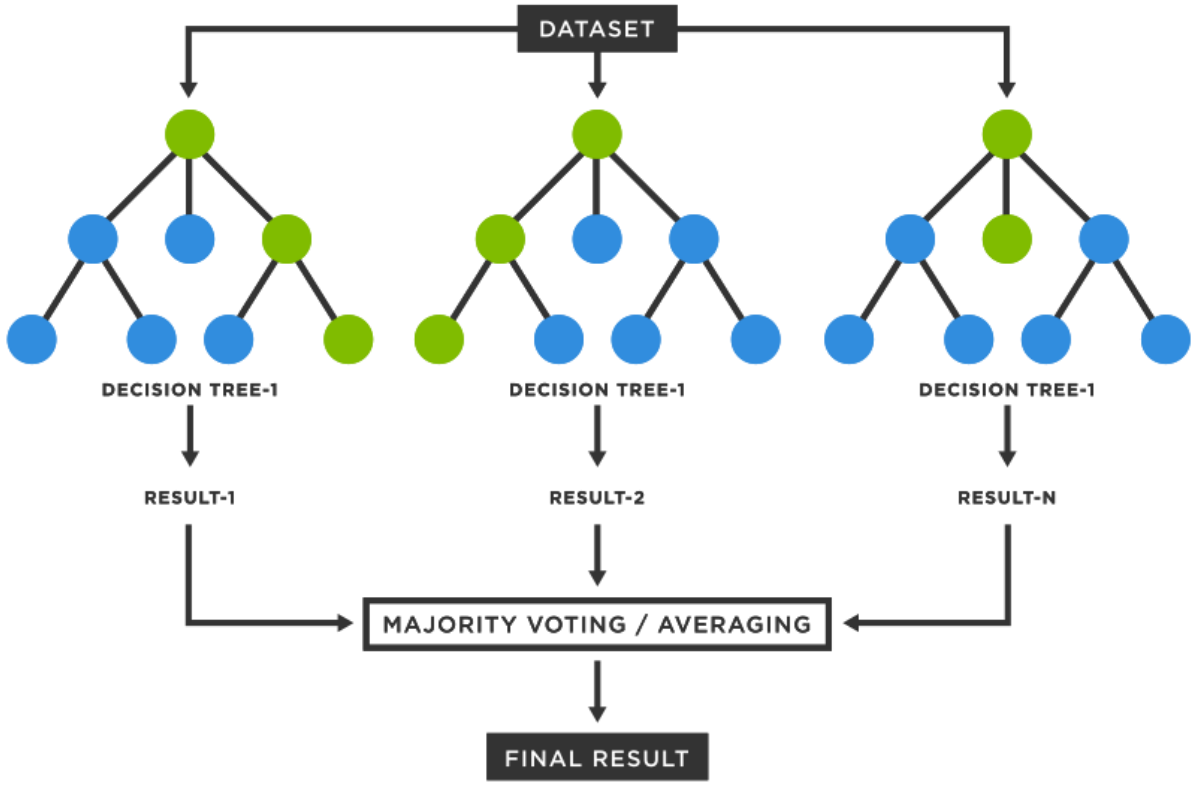
Assuming the dataset is large:

$$\lim_{n \to +\infty} \left(1 - \frac{1}{n}\right)^n = \lim_{n \to +\infty} \left(1 + \frac{-1}{n}\right)^n = e^{-1} \approx 0.37$$

Bagging

Once a good number (usually 100+) trees have been grown from bootstrapped datasets, their predictions on new samples are:

- the **average** of the prediction of single trees, in case of regression
- the **most common** class, in case of classification



Feature Bagging

In order to produce predictions as independent as possible, random forest does not perform bagging only on the samples.

It does the same on **features**.

Each tree is grown with a subset of the features of the original dataset (how much? That's an hyperparameter).

This process is called **feature bagging**.

Why Feature Bagging?

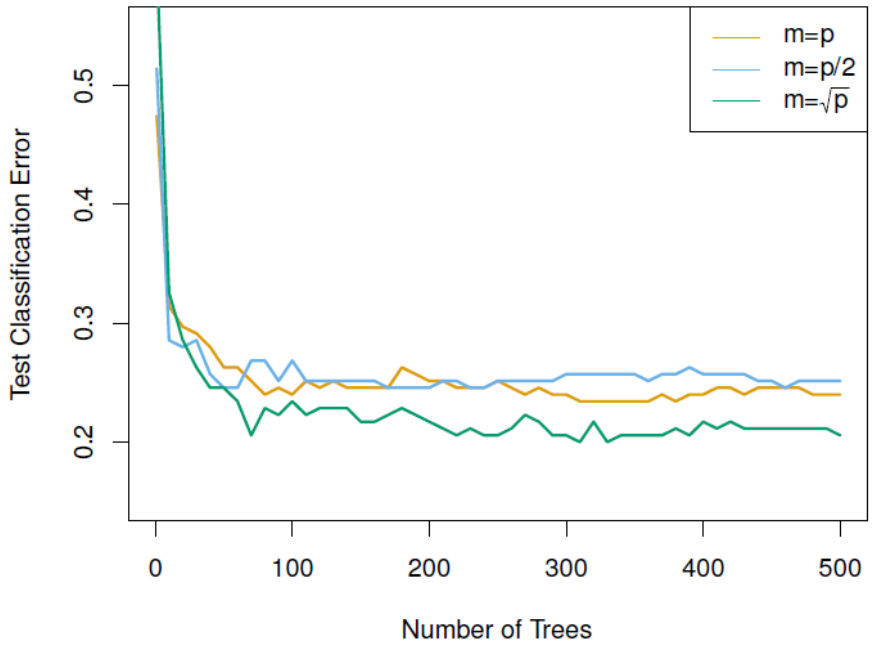
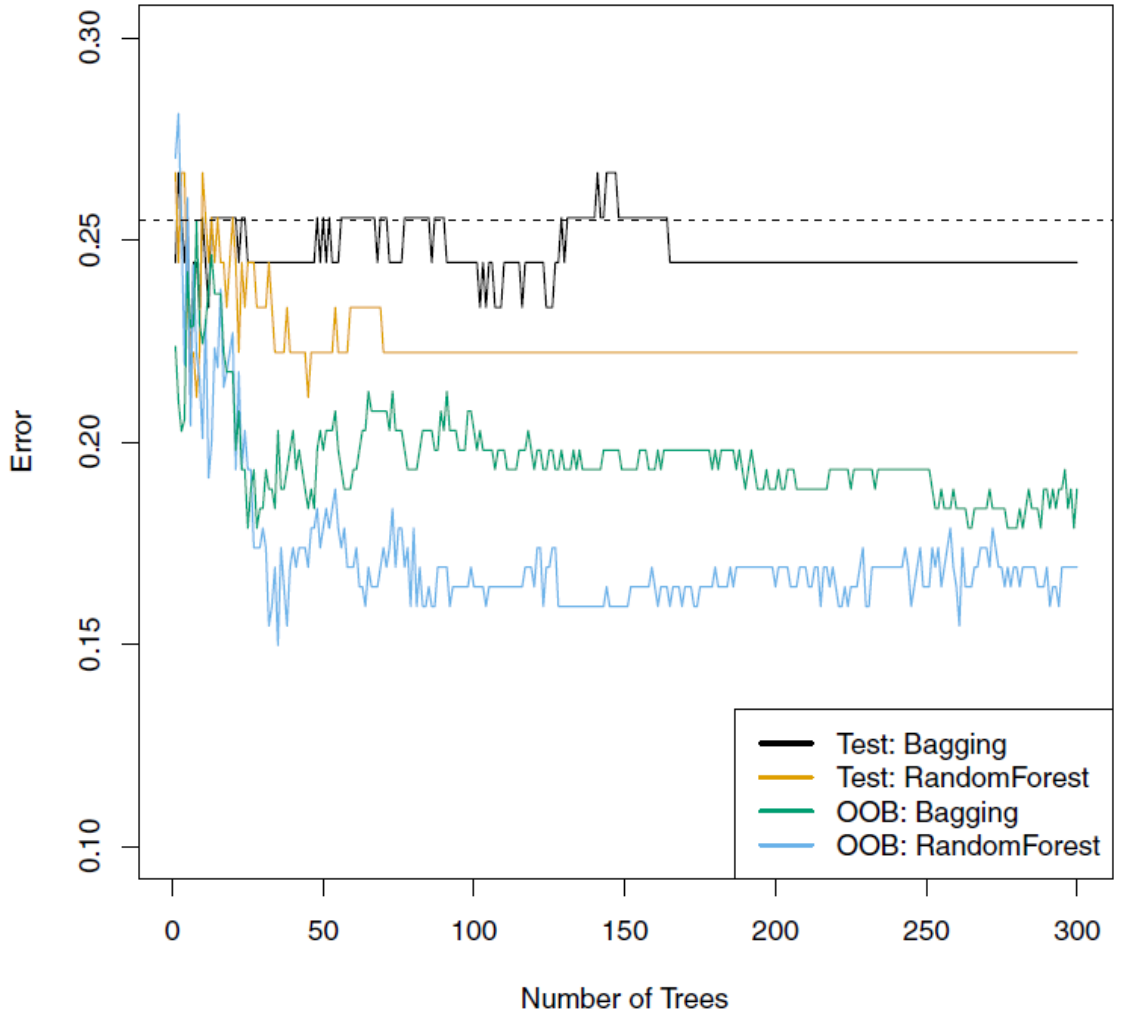


FIGURE 8.10. *Results from random forests for the 15-class gene expression data set with $p = 500$ predictors. The test error is displayed as a function of the number of trees. Each colored line corresponds to a different value of m, the number of predictors available for splitting at each interior tree node. Random forests ($m < p$) lead to a slight improvement over bagging ($m = p$). A single classification tree has an error rate of 45.7%.*

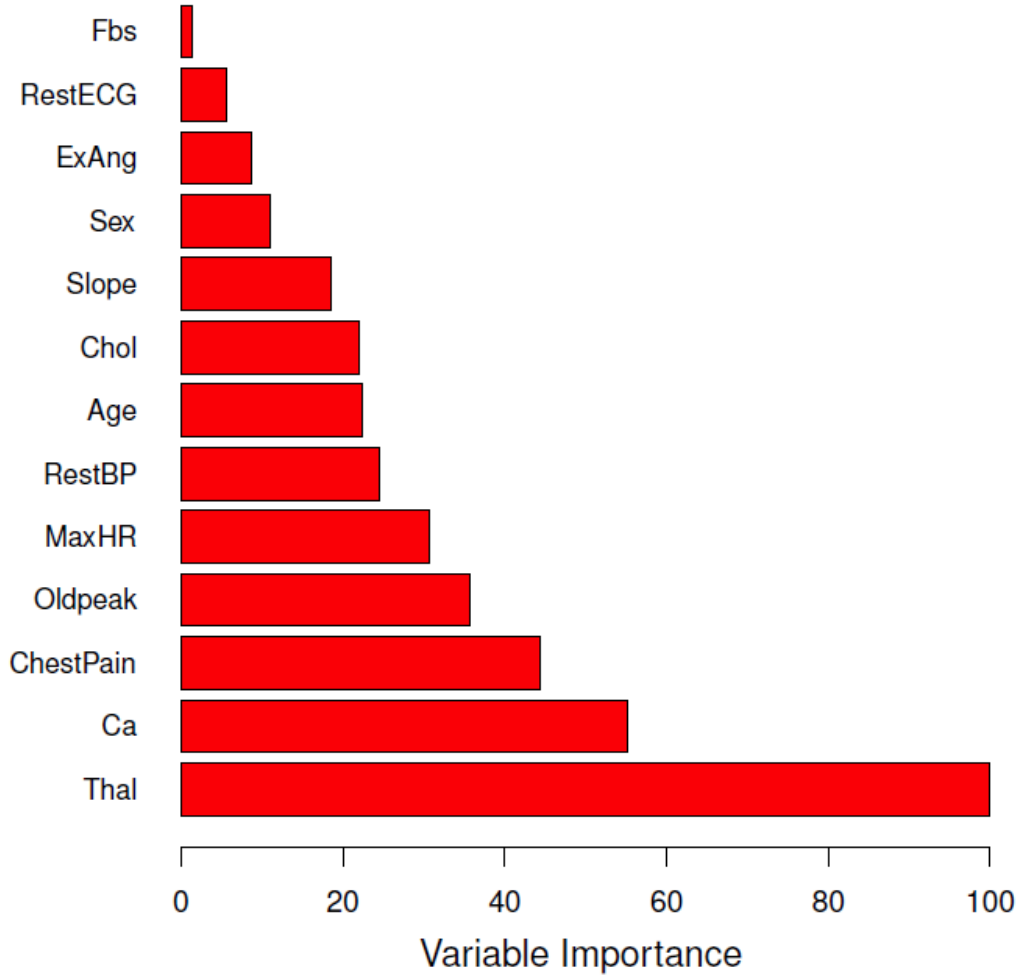
From James et al, Introduction to Statistical Learning

Why Random Forest?



From James et al, Introduction to Statistical Learning

Interpreting Random Forest



From James et al, Introduction to Statistical Learning

Interpreting Random Forest

A random forest model cannot be interpreted as easily as a single tree.

However, we can leverage the **decrease in the loss function** at each split.

We note how much the loss function decreases in each tree for each split involving a specific variable. Then, we average over all the trees.

This way, we get a measure of the impact of each feature in shaping the tree.

This is called *mean decrease impurity (MDI).*

Interpreting Random Forest

However, it is well known that MDI is **biased** towards the features with more values or with higher variance (as they are involved in a larger number of splits).

An alternative method is *permutation importance*.

We train the random forest q times, where q is the number of features, and each time we **randomly permute** one feature, so that its values are no longer related to the target.

Then, we rank the features according to the **drop in out-of-sample accuracy** of the model.

A word of caution…

Any feature importance method works on a **specific instance of a model**.

Therefore, it can tell how much a feature is useful for that specific instance of the model. It does NOT provide any clue about the value of the feature in absolute terms.

Never trust feature importance from a bad model.

See https://scikit-learn.org/stable/modules/permutation_importance.html#permutation-importance

Another word of caution…

Assume you have two highly correlated features in your dataset.

If you apply permutation importance, when you “remove” a variable, the model will simply use the other one!

So, both the variables will seem useless, until you remove both and the result deteriorates massively.

Always pay attention to correlated variables when applying feature importance methods.

See https://scikit-learn.org/stable/modules/permutation_importance.html#permutation-importance

Open Notebook in Colab

Gradient Boosting

Why boosting?

Bagging fits many different models on different datasets.

Therefore, the process can be performed in parallel and the information from different models can be aggregated only ***a posteriori***.

But what if we used the information from each model to **improve the training** of the next one?

Boosting Models

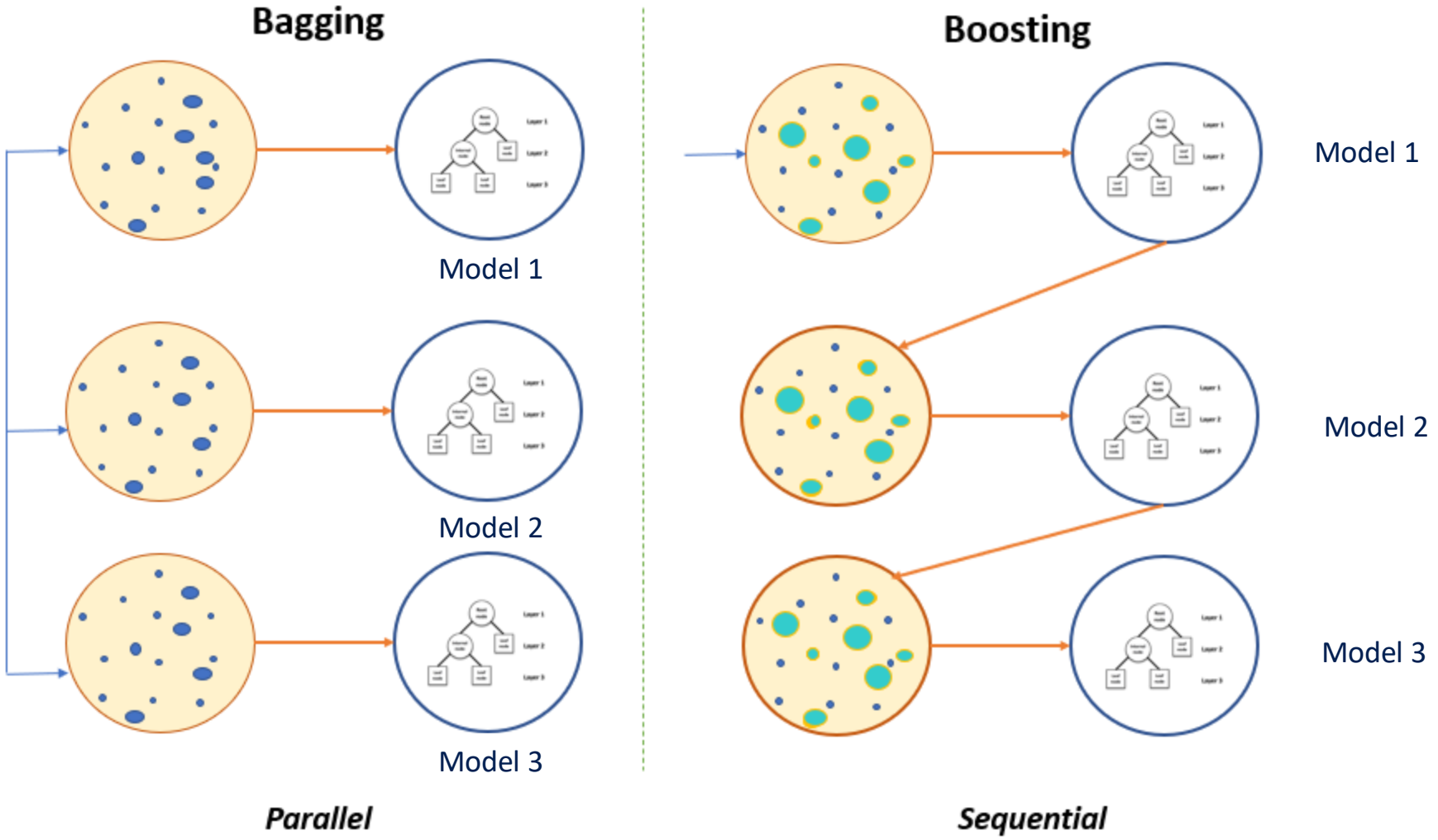
Boosting models are considered among the most effective for tabular data.

There are many different boosting algorithms: we will only discuss the main ideas.

The most advanced implementations are provided by:

- **XGBoost**
- **LightGBM** (by Microsoft)

Boosting


Bagging
Model 1
Model 2
Model 3
Parallel
Boosting
Model 1
Model 2
Model 3
Sequential

Why boosting?

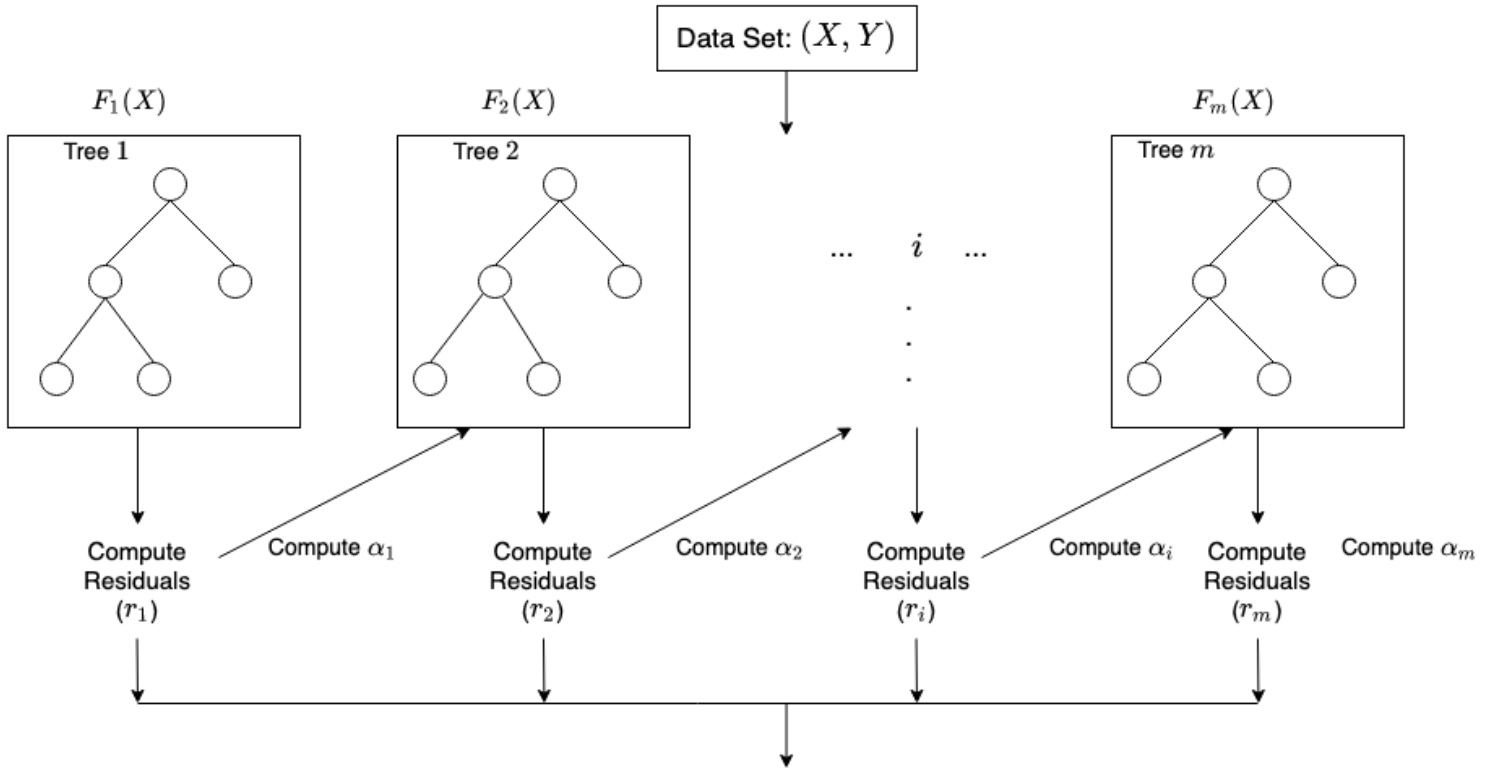
The main idea behind boosting is to make each tree focus on the samples which were **worst predicted** by the previous ones.

Each tree can be a rather poor predictor on the whole dataset, as long as it improves on some samples – i.e., on some regions on the input space.

As each tree has only a limited weight in the final prediction, the boosting model **learns slowly**.

Go to https://inria.github.io/scikit-learn-mooc/python_scripts/ensemble_adaboost.html

Simplified XGBoost



$$F_m(X) = F_{m-1}(X) + \alpha_m h_m(X, r_{m-1}),$$

where α_i, and r_i are the regularization parameters and residuals computed with the i^{th} tree respectfully, and h_i is a function that is trained to predict residuals, r_i using X for the i^{th} tree. To compute α_i we use the residuals computed, r_i and compute the following: $arg \min_{\alpha} = \sum_{i=1}^{m} L(Y_i, F_{i-1}(X_i) + \alpha h_i(X_i, r_{i-1}))$ where $L(Y, F(X))$ is a differentiable loss function.

See https://aws.amazon.com/it/what-is/boosting/

Simplified XGBoost

Let $\hat{y}_i = 0$ and $r_i = y_i$ for all samples $i = 1 \dots n$ in the training set.

Denote with $\hat{y} = \hat{f}(x)$ the tree-based model, and $L(\hat{f}, x, y) = \sum_i \left(\hat{f}(x_i) - y_i\right)^2$ the loss function. Set a number b of trees to be trained.

For j $= 1 \dots b$, repeat the following steps:

1. Fit a tree $\hat{t}_j(x)$ with feature matrix X and target vector r – **note, r not y**.
2. Update the overall model $\hat{f}(x) = \hat{f}(x) + \lambda \hat{t}_j(x)$, where λ is an hyperparameter, which may be dependent on the step.
3. Update the **residuals** $r = r - \lambda \hat{t}_j(x)$

Once all the b trees have been trained, the final model is:

$$\hat{f}(x) = \sum_{j=1}^{b} \lambda \hat{t}_j(x)$$

See https://aws.amazon.com/it/what-is/boosting/

Simplified XGBoost

But what if the loss function is different?

You can apply **the same procedure**! Observe that $r = \hat{f}(x) - y$ is the **gradient of the loss function** with respect to the predictions. With a generic loss function…

For j $= 1 \ldots b$, repeat the following steps:

1. Fit a tree $\hat{t}_j(x)$ with feature matrix X and target vector $r = -\nabla L(\hat{f}, x, y)$.
2. Update the overall model $\hat{f}(x) = \hat{f}(x) + \lambda \hat{t}_j(x)$, where λ is an hyperparameter , which may be dependent on the step.

Once all the b trees have been trained, the final model is:

$$\hat{f}(x) = \sum_{j=1}^{b} \lambda \hat{t}_j(x)$$

See https://aws.amazon.com/it/what-is/boosting/

Simplified XGBoost

Gradient boosting can be seen as **a gradient descent in the space of the models**. Each step improves the model's ability to minimise the loss function by incorporating an additional block into the prediction function.

Many improvements have been proposed to the basic gradient boosting algorithm.

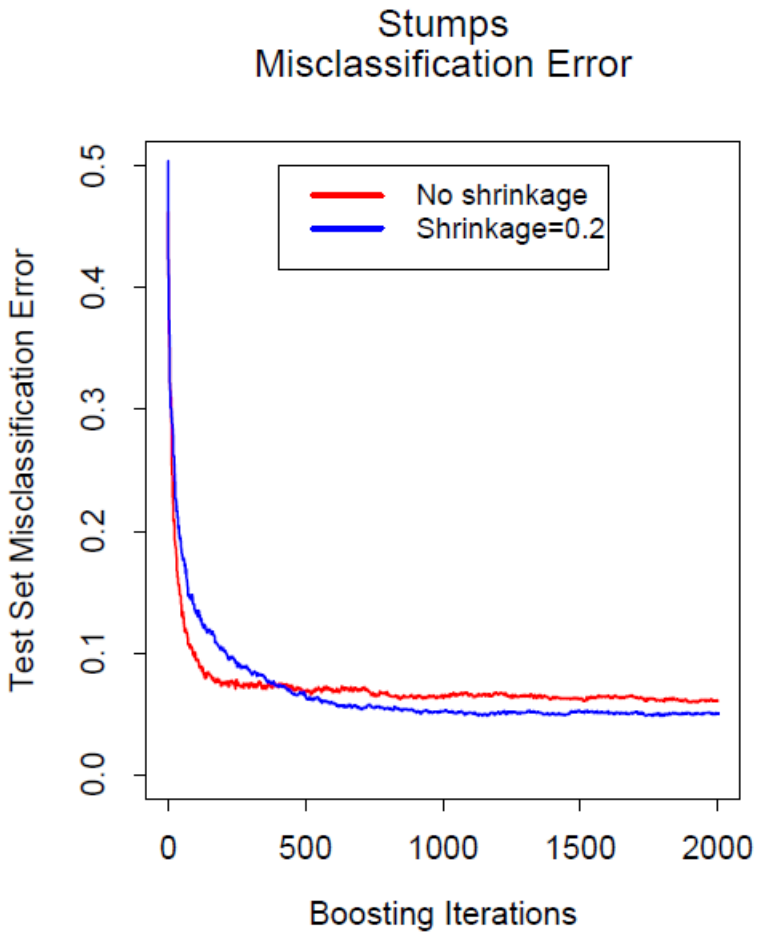
The most important are:

- Adding the variable **learning rate** as a hyperparameter depending on the step
- Training each tree on a different **subsample** of the initial dataset
- Adding regularization

Learning rate

Instead of using the rule $\hat{f}_j(x) = \hat{f}_{j-1}(x) + \hat{t}_j(x)$, we choose $\hat{f}_j(x) = \hat{f}_{j-1}(x) + \lambda\hat{t}_j(x)$, where λ is a positive hyperparameter.

It was shown empirically that the introduction of the learning rate **improves the out-of-sample error**, at the price of a **slower training**. The smaller λ, the larger the number of trees required to get to the same performance.



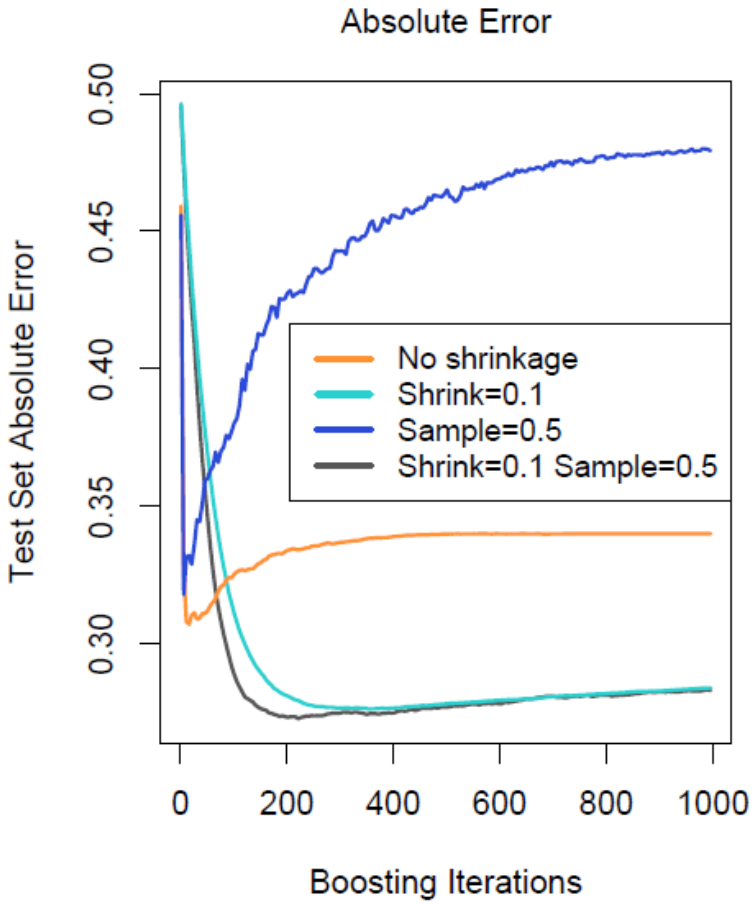
From Hastie et al, Elements of Statistical Learning

Sampling & Bagging

Instead of training each tree on the same dataset, at every iteration we sample a fraction s of it. The fraction s is also a hyperparameter.

Again, it was shown empirically that sampling **prevents overfitting** and helps convergence. This is a similar effect to the one observed with **stochastic gradient descent**.

As random forest, Gradient Boosting can also apply bagging.



From Hastie et al, Elements of Statistical Learning

Boosting hyperparameters

There are three main **hyperparameters** in boosting trees:

- **the number of trees**: if it is too large, boosting models can overfit
- **the learning rate**: if the learning rate is small, many trees may be needed to get a decent performance – however, if it is too large, the benefits of slow learning may be lost
- **the stopping criterion in each tree**: often gradient boosting uses very simple base trees, with low depth and a large number of samples in each leaf

Interpreting gradient boosting

Being based on trees, Gradient Boosting features the **same feature importance metrics** described for Random Forest.

Warning: the same issues apply.

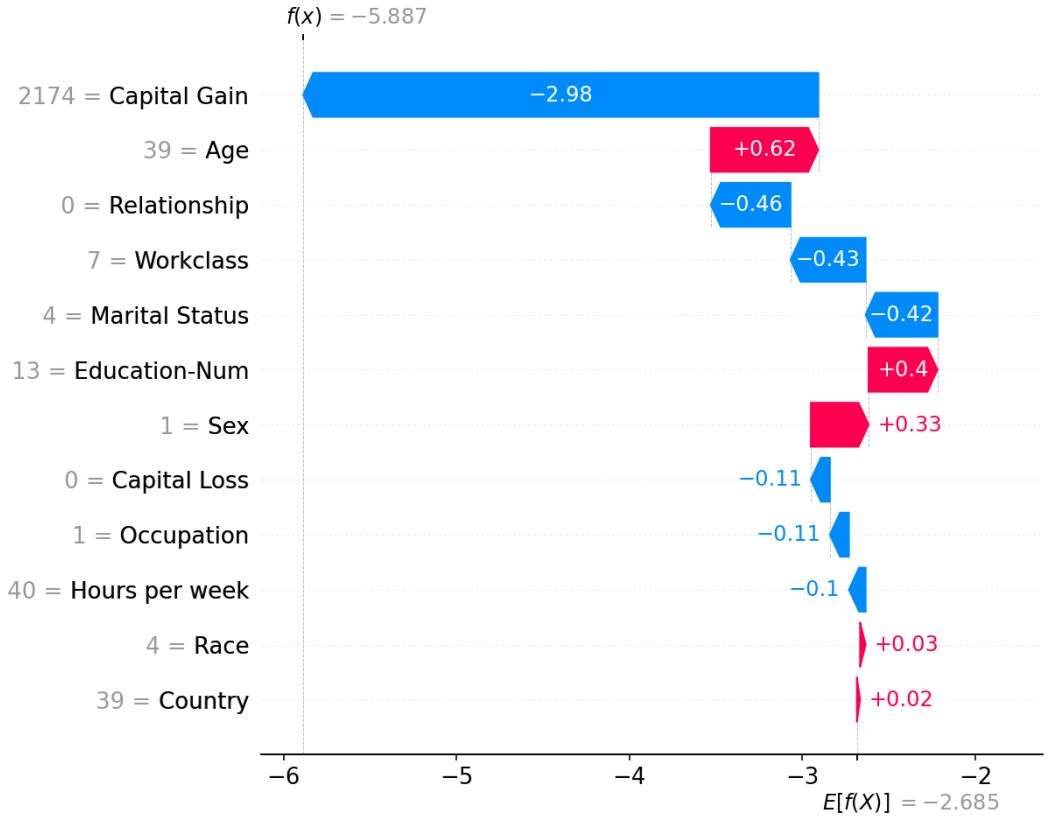
SHAP

SHAP: Key Ideas.

SHAP (Shapley Additive Explanation) is a method used to **approximate** Shapley values, which measure the contribution of each feature to a specific model prediction.

It is **local**, as it can be calculated for each observation.

It is **additive**, as the sum of the SHAP values equals the difference between the base value, the prediction without any informative feature and the actual prediction.

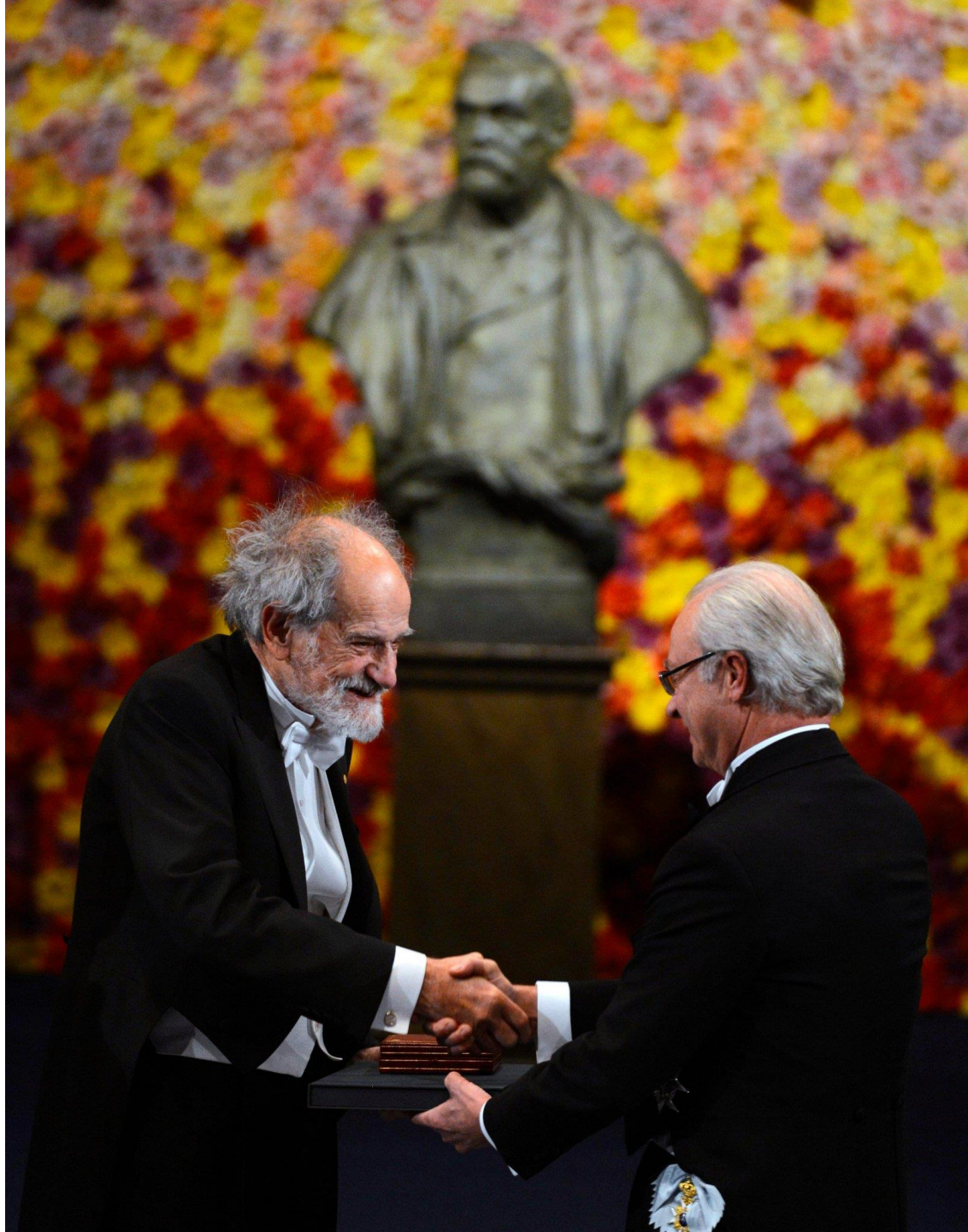


Shapley Values?

How do we **fairly** distribute money to players who win a game together, based on their contributions?

Let’s define by fair a method that is both:

- **Efficient**: all the money must be distributed.
- **Consistent**: if a player contributes more than another, they must get more money.

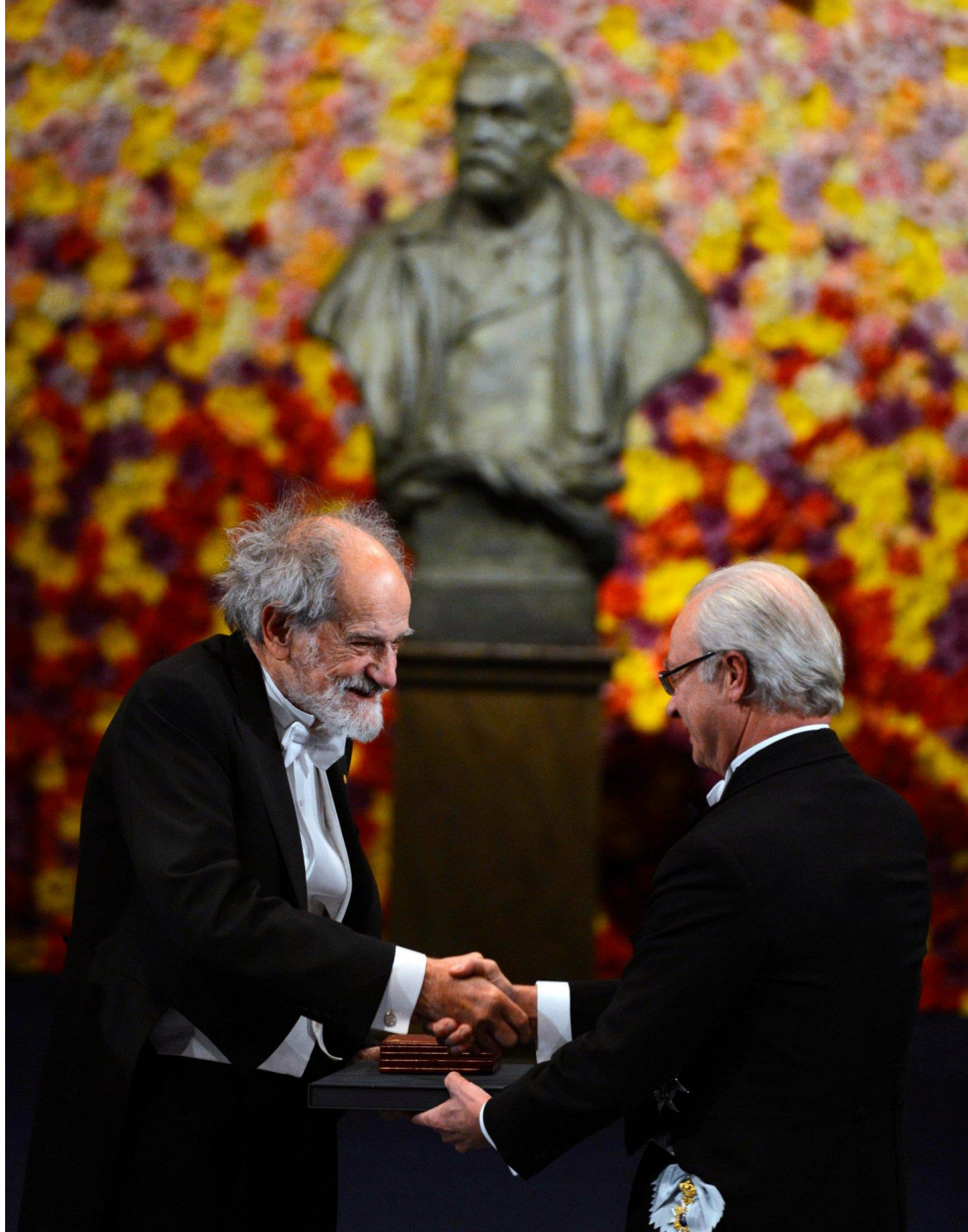


Shapley Values?

A theorem demonstrates that there is **one and only one** method to distribute the money fairly.

This involves splitting the money based on the average contribution of each player across **all possible games with every subset of players**, regardless of the order.

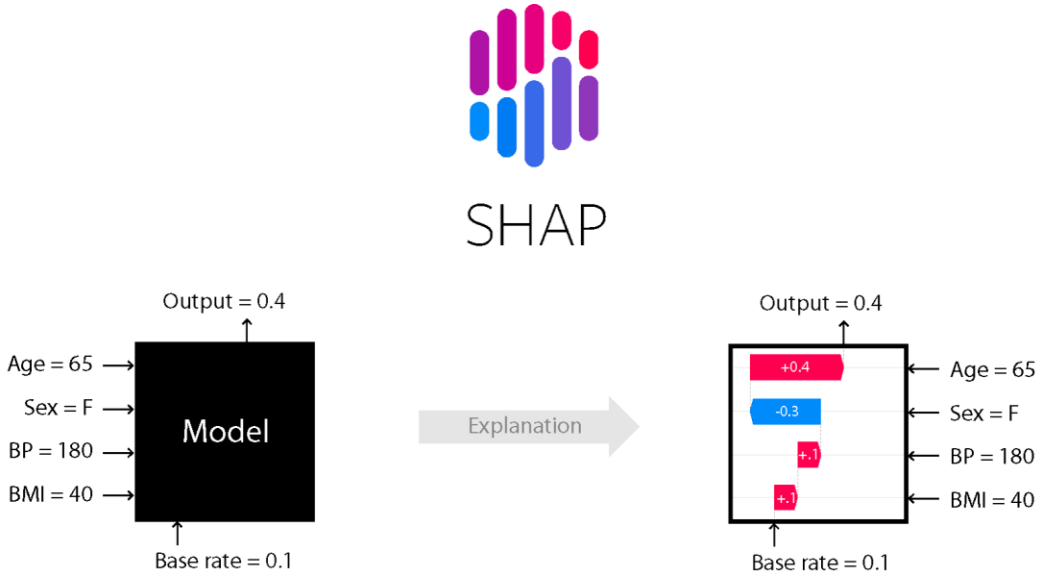
The **share** allocated to each player using this method is the **Shapley value**.



Two Problems

How can **all possible combinations** be considered?

How can the **same model** be trained **without** certain features?



Discussion

How can we solve these two problems?

Two Problems

How can all possible combinations be considered?

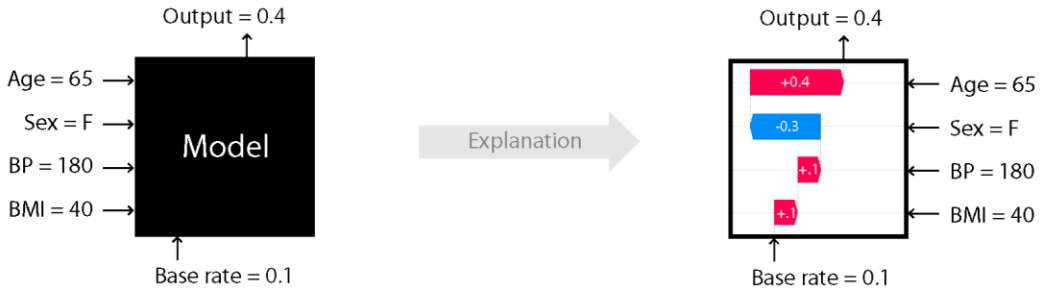
Sample! Consider only **a subset** of all possible combinations.

How can the same model be trained without certain features?

Sample! Replace the feature's value with another **random value** from the same variable.

This approach can be very problematic. Why?

SHAP



What is SHAP, Really?

A collection of **smart algorithms** to approximate Shapley values.

That is, the only **fair** way of computing feature contribution is black-box models.

SHAP

Interpretable ML

Explainability remains a significant topic and an ongoing challenge yet to be fully resolved.

Feel free to dive deeper with the book Interpretable Machine Learning by Christoph Molnar or the following talks:

- Adi Watzman, SHAP and Machine Learning Explainability
- Emanuele Fabbiani, From SHAP to EBM

Open Notebook in Colab


The End