Machine Learning for Astrophysicists: An Introduction

Giuseppe Riccio

Astronomical Observatory of Capodimonte - Napoli

DaME (Data Mining & Exploration) started as a collaboration between:

- Astronomical Observatory of Capodimonte
- University of Naples, Federico II
- Caltech

created in **2007** order to deep dive into the field of Astroinformatics by applying ML techniques to Astrophysics

A Band of Fools

Stefano Cavuoti, Giuseppe Riccio, Giuseppe Angora, Ylenia Maruccia **- INAF OACN**

Giuseppe Longo, Massimo Brescia, Demetra De Cicco, Maurizio Paolillo, Lorenzo Santo **- Unina, Federico II**

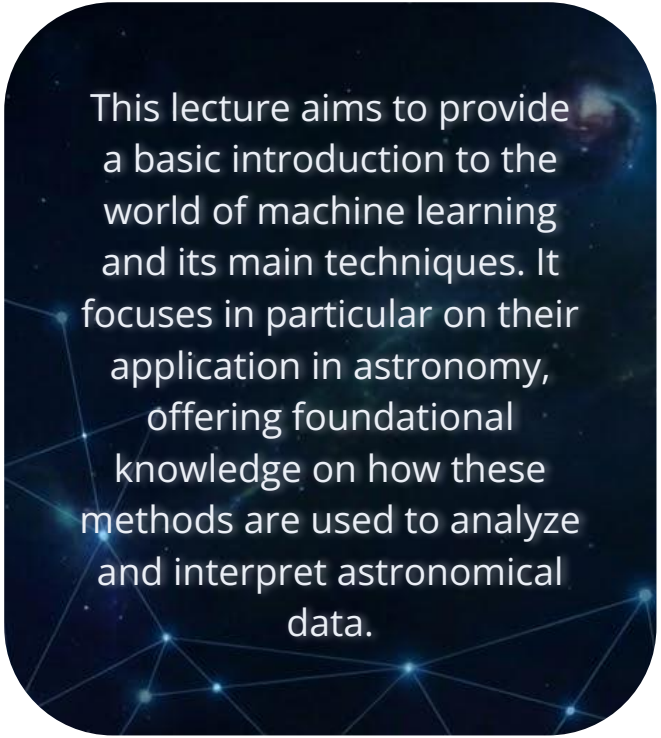
Past members (Fixed Term, Fellowship, PhD, Master Students...):
Giovanni Albano, Valeria Amaro, Marianna Annunziatella, Massimo Benedetto, Sabrina Checola, Raffaele D'Abrusco, Maurizio D'Addona, Pierluigi D'Andrea, Giovanni D'Angelo, Michele Delli Veneri, Virgilio De Stefano, Luna Di Colandrea, Alessandro Di Guido, Antonio D'Isanto, Lars Doorenbos, Francesco Esposito, Pamela Esposito, Michelangelo Fiore, Mauro Garofalo, Domenico Guarino, Marisa Guglielmo, Omar Laurino, Francesco Manna, Alfonso Nocella, Luca Pellecchia, Carlo Enrico Petrillo, Oleksandra Razim, Sandro Riccardi, Bojan Skordovski, Andrea Solla, Olena Torbaniuk, Gianluca Tutino, Civita Vellucci, Giovanni Vebber, Giuseppe Sarracino, Natale De Bonis, Simone Vaccaro **- Around the World**

**We few, we happy few, we band of brothers;
For he today that sheds his blood with me shall be my brother**

**Henry V
William Shakespeare**

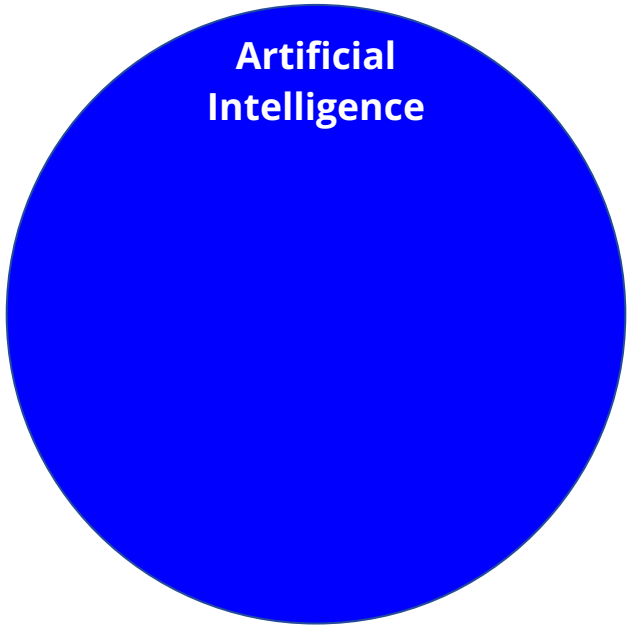
About this lecture: Scope & Outline

- Introduction
  - Welcome to the world of Artificial Intelligence
  - From Turing to ChatGPT
  - What is Machine Learning, and what is not
  - Why Machine Learning
- The Data
  - The Data Tsunami
  - From Big Data to Astroinformatics
  - Data in Astronomy
  - Issues
- Basic Concepts of ML
  - Approaching a ML project
  - Pro&Cons
  - ML paradigms
- Supervised Learning
  - Definitions
  - Deep into the workflow
  - Evaluation metrics
  - Basic models
- Unsupervised Learning
  - An overview
  - Evaluation metrics
  - Basic models
- Deep Learning in a nutshell
- Final Tips



Introduction

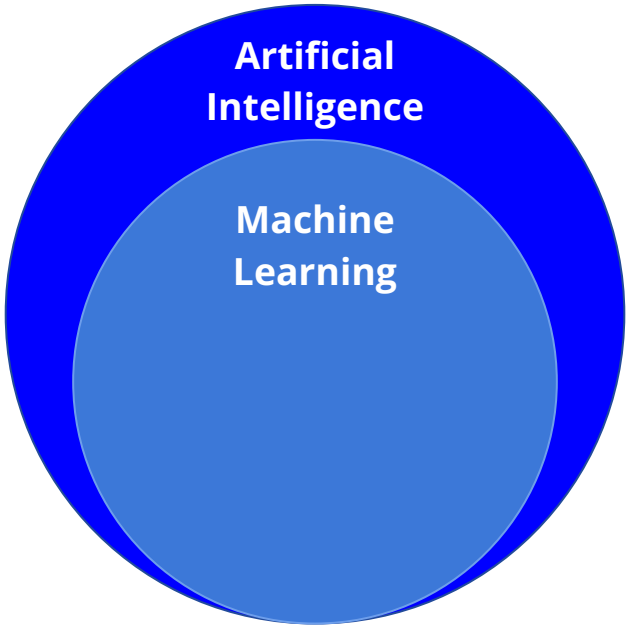
What area are we moving in?



"Artificial intelligence (AI) is technology that enables computers and machines to simulate human learning, comprehension, problem solving, decision making, creativity and autonomy."

- General term for very large and rapidly developing field.
- No strict definition, but often used when machines perform tasks that could only be solved by humans or are very difficult and assumed to require "intelligence".
- Started in the 1940s – when the computer was invented. Turing and von Neumann immediately asked: If we can formalize computation, can we use that to formalize "thinking"?
- Includes ML, NLP, computer vision, robotics, planning, search, intelligent agents, …
- Sometimes misused as a "hype" term for ML or … basic data analysis.
- Or people refer to the fascinating developments in the area of foundation models

What area are we moving in?



Machine Learning

"Field of study that gives computers the ability to learn without being explicitly programmed"

Arthur Samuel - 1959 (?)

What is ML… in two words

Machine learning is a branch of informatics that allows software to **learn** to find solutions to specific tasks from numerical data without being **explicitly** programmed to do so.

What is ML… in two words

Machine learning is a branch of informatics that allows software to **learn** to find solutions to specific tasks from numerical data without being **explicitly** programmed to do so.

Learning means acquiring knowledge that can be applied in the future. So we teach the computer something so it can repeat it in the future without our intervention.

"A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E."

Tom Mitchell, Carnegie Mellon University, 1998

What is ML... in two words

Machine learning is a branch of informatics that allows software to **learn** to find solutions to specific tasks from numerical data without being **explicitly** programmed to do so.

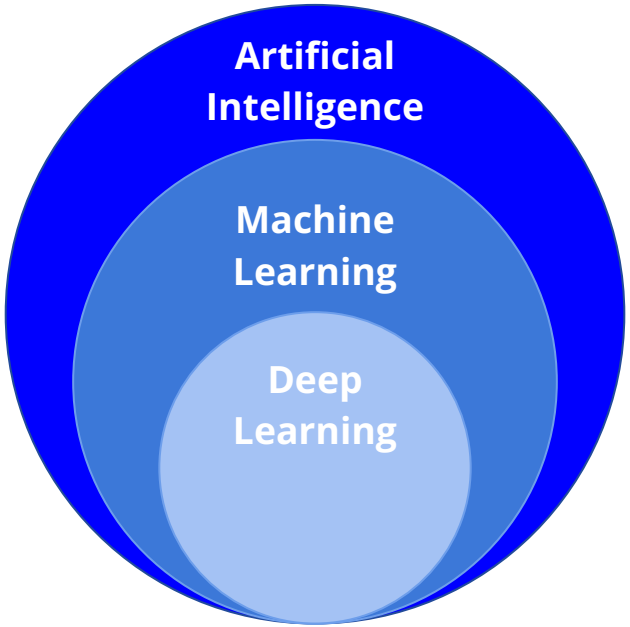
Learning means acquiring knowledge that can be applied in the future. So we teach the computer something so it can repeat it in the future without our intervention.

"A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E."

Tom Mitchell, Carnegie Mellon University, 1998

Human intervention is necessary for the computer to learn, but this intervention doesn't consist of explicitly telling the computer how to behave. Instead, we give the computer a sort of "key" (algorithms), so it can interpret the data on its own. From there, it will proceed on its own, and it will then decide what to do when it needs to put its learning into practice.

What area are we moving in?

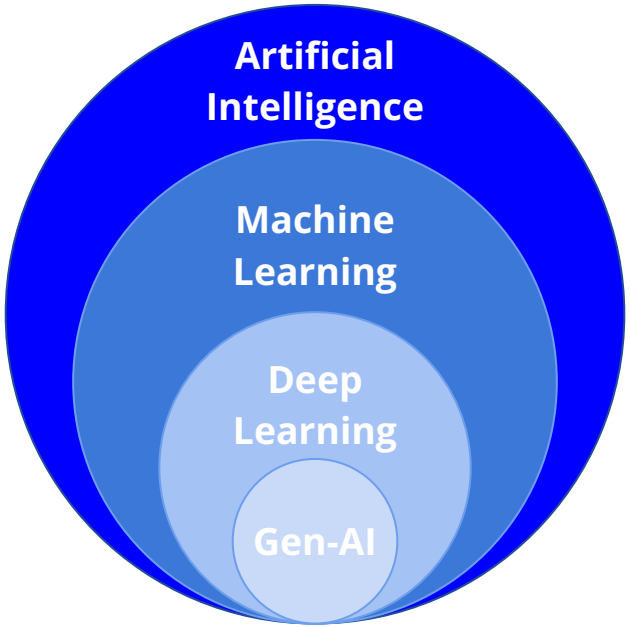


Deep Learning

Deep learning is a **subset** of machine learning that uses multilayered neural networks, called **deep neural networks**, that more closely simulate the complex decision-making power of the **human brain**.

Because deep learning **doesn't require human intervention**, it enables machine learning at a tremendous scale. It is well suited to natural language processing (NLP), computer vision, and other tasks that involve the fast, accurate identification complex patterns and relationships in large amounts of data. Some form of deep learning powers most of the artificial intelligence (AI) applications **in our lives today**.

What area are we moving in?



Gen-AI

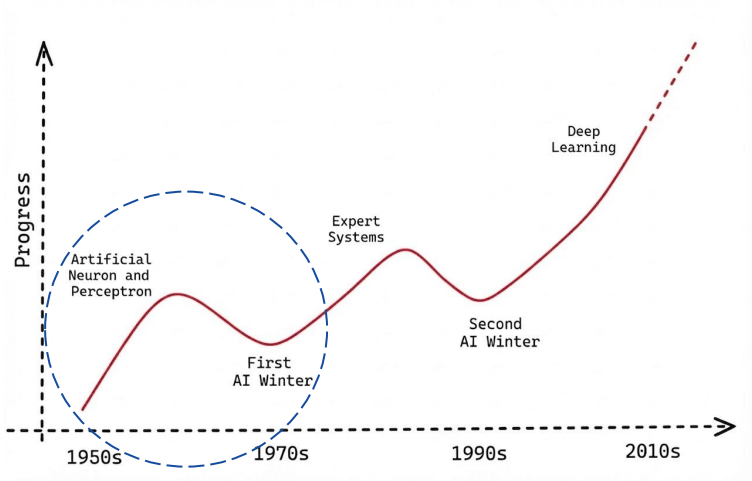
Generative artificial intelligence is a subfield of artificial intelligence that uses **generative models** to generate text, images, videos, audio, software code or other forms of data. These models **learn the underlying patterns and structures of their training data**, and use them to generate new data in response to input, which often takes the form of natural language prompts.

AI vs ML vs DL

Aspect	Artificial Intelligence (AI)	Machine Learning (ML)	Deep Learning (DL)
Definition	AI refers to the broader concept of creating machines that can perform tasks requiring human-like intelligence.	ML involves the development of algorithms that enable computers to learn from data and improve over time.	DL is a subset of ML that uses artificial neural networks with multiple layers (deep architectures) to learn representations of data.
Approach	Mimics human intelligence and behavior to perform tasks.	Learns patterns from data and makes predictions or decisions.	Learns representations of data through hierarchical layers of abstraction.
Data Size	Can handle both small and large datasets.	Can handle both small and large datasets.	Particularly effective with large volumes of data.
Complexity	Can be complex and may involve various approaches, including ML and DL.	Can range from simple linear models to complex deep neural networks.	Utilizes complex neural network architectures with multiple layers.
Interpretability	May lack interpretability due to the complexity of AI systems.	Depends on the complexity of the ML model; simpler models may be more interpretable.	Often considered less interpretable due to the hierarchical nature of deep neural networks.
Training Time	Can vary widely depending on the complexity of the AI system.	Training time depends on the complexity of the ML model and the size of the dataset.	Can be time-consuming, especially with large datasets and complex architectures.
Hardware	Can run on various hardware platforms, including CPUs and GPUs.	Can run on CPUs and GPUs, with specialized hardware (e.g., TPUs) available for ML tasks.	Often requires GPUs or specialized hardware accelerators for training and inference.

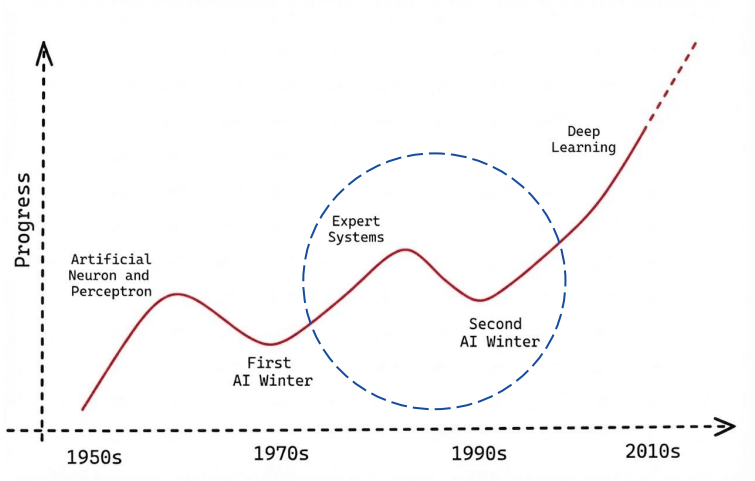
History of AI

- **1950**
  - **Alan Turing** publishes Computing Machinery and Intelligence. In this paper, Turing famous for breaking the German ENIGMA code during WWII and often referred to as the "father of computer science" asks the following question: **"Can machines think?"**
    From there, he offers a test, now famously known as the "Turing Test," where a human interrogator would try to distinguish between a computer and human text response. While this test has undergone much scrutiny since it was published, it remains an important part of the history of AI, and an ongoing concept within philosophy as it uses ideas around linguistics.
- **1956**
  - John McCarthy coins the term "**artificial intelligence**" at the first-ever AI conference at **Dartmouth College**. (McCarthy went on to invent the Lisp language.) Later that year, Allen Newell, J.C. Shaw and Herbert Simon create the Logic Theorist, the first-ever running AI computer program.
- **1967**
  - Frank Rosenblatt builds the **Mark 1 Perceptron**, the first computer based on a neural network that "learned" through trial and error. Just a year later, Marvin Minsky and Seymour Papert publish a book titled Perceptrons, which becomes both the landmark work on neural networks and, at least for a while, an argument against future neural network research initiatives.
- **1970s**
  - First **"AI winter"**: unrealistic expectations, limited computing power, little practical progress compared to media hype led to a **loss of interest** and **drastic funding cuts**



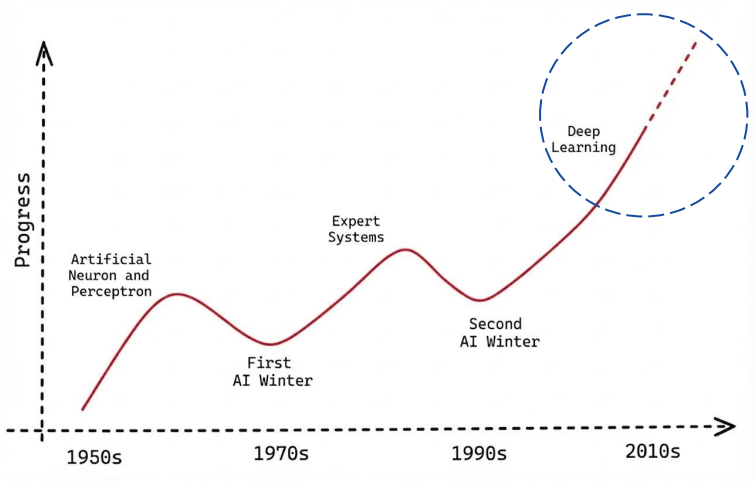
History of AI

- **1980s**
  - Revival of interest thanks to the development of **Expert Systems** and **Neural networks**, which use a backpropagation algorithm to train itself, became widely used in AI applications (medicine, engineering and finance).
- **End ‘80s - Begin ‘90s**
  - Second **“AI winter”** : **serious limitations** of expert systems, which have proven difficult to update, very rigid, and incapable to effectively managing uncertainty or new and unexpected situations, in the face of **excessive development and maintenance costs**
- **Half ‘90s**
  - Growing availability of data and increasing computing power pave the way for the success of **Machine Learning**.
- **1997**
  - IBM's Deep Blue beats then world chess champion Garry Kasparov, in a chess match (and rematch).
- **2004**
  - John McCarthy writes a paper, What Is Artificial Intelligence?, and proposes an often-cited definition of AI. By this time, the era of big data and cloud computing is underway, enabling organizations to manage ever-larger data estates, which will one day be used to train AI models.



History of AI

- **2011**
  - IBM Watson® beats champions Ken Jennings and Brad Rutter at Jeopardy! Also, around this time, data science begins to emerge as a popular discipline.
- **2015**
  - Baidu's Minwa supercomputer uses a special deep neural network called a **convolutional neural network** to identify and categorize images with a higher rate of accuracy than the average human.
- **2016**
  - DeepMind's AlphaGo program, powered by a deep neural network, beats Lee Sodol, the world champion Go player, in a five-game match. The victory is significant given the huge number of possible moves as the game progresses (over 14.5 trillion after just four moves). Later, Google purchases DeepMind for a reported USD 400 million.
- **2022**
  - A rise in large language models or **LLMs**, such as OpenAI's ChatGPT, creates an enormous change in performance of AI and its potential to drive enterprise value. With these new generative AI practices, deep-learning models can be pre-trained on large amounts of data.

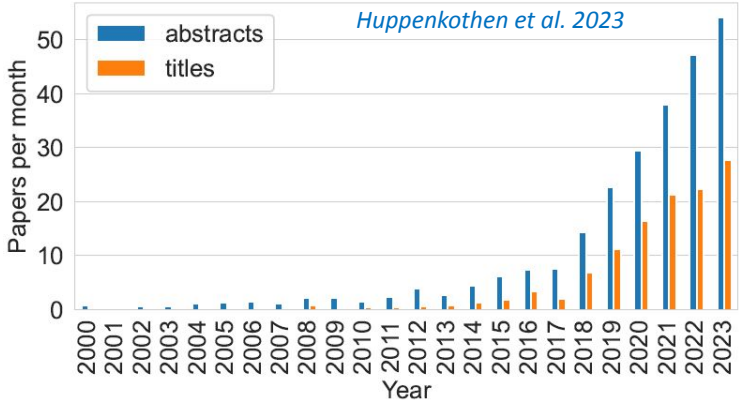


History of AI…nowadays

AI & Astrophysics (more widely, Science)

As AI is becoming a standard approach in Astrophysics, there is an increasing and urgent need for **best practices** on how to build and implement well-posed solutions, how to apply them and report their results.

Data challenges are a common practice internally to large communities involved into big survey projects, often including errors and lack of uniformity in the training data, causing data analysis mismatched with “traditional” methods and the need of careful cross validation.



At least, **four main criteria** should be followed by an AI approach in Astrophysics, but more in general in any human science field:

Reproducibility: results should be invariant w.r.t. data training/prediction choice and code/data made publicly accessible

Interpretability: methods should be explainable and transparent in terms of both pipeline and data flow process

Accuracy: methods should provide accurate results, i.e. unbiased, uniform and rigorous in terms of their validation

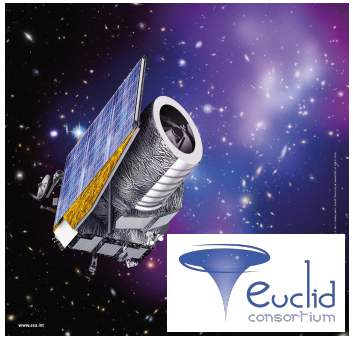
Flexibility: methods should demonstrate usefulness and non-ambiguous advantages w.r.t. traditional approaches

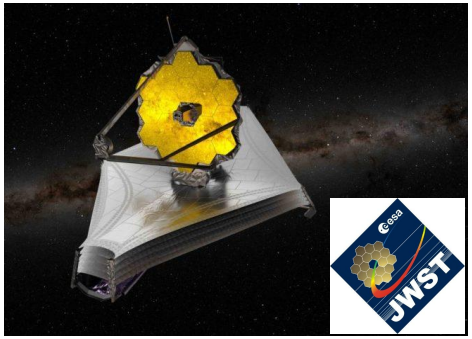
Why AI in Astronomy

In the last two decades, Astronomy has been the scene of the realization of panchromatic surveys, with sophisticated instruments acquiring a huge amount of exceptional quality data

SOME NUMBERS…

- ESA Euclid : ~100 GB/day for 6 years → 200 TB
- Rubin/LSST : ~30 TB/night for 10 years → >100 PB
- JWST : ~30GB/day for 10 years (and more)
- Pan-STARRS : >100 TB of data
- GAIA : ~1 PB in 5 year
- SKA : 100 Pbytes/day – 3 EBytes/year
- KiDS, DES, Herschel-ATLAS, Hi-GAL, E-ELT…





The Data

The problem:
Data-Rich Astronomy

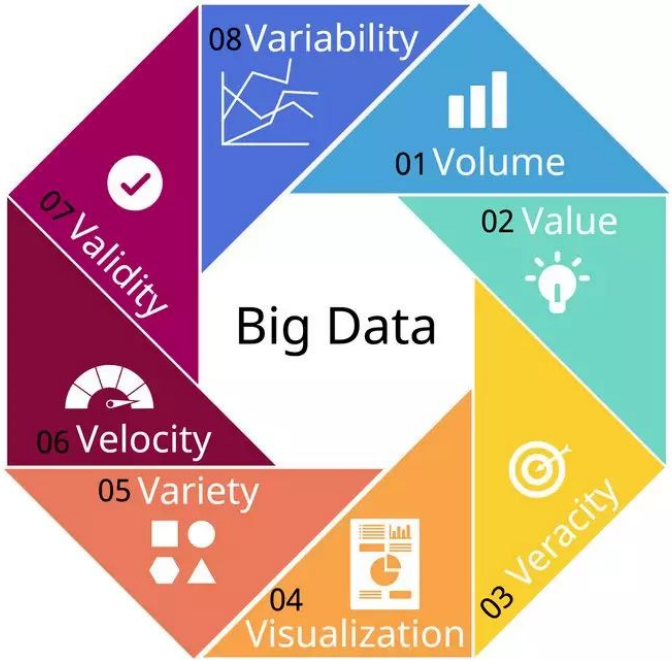
The power of (Big) Data

If we collect enough data with a high-dimensional parameter space within any domain of study, then we would have a *consistent* statistical model for that domain: **ALL ANSWERS ARE AVAILABLE!**

credits: *M. Brescia*

The power of (Big) Data

If we collect enough data with a high-dimensional parameter space within any domain of study, then we would have a *consistent* statistical model for that domain: **ALL ANSWERS ARE AVAILABLE!**

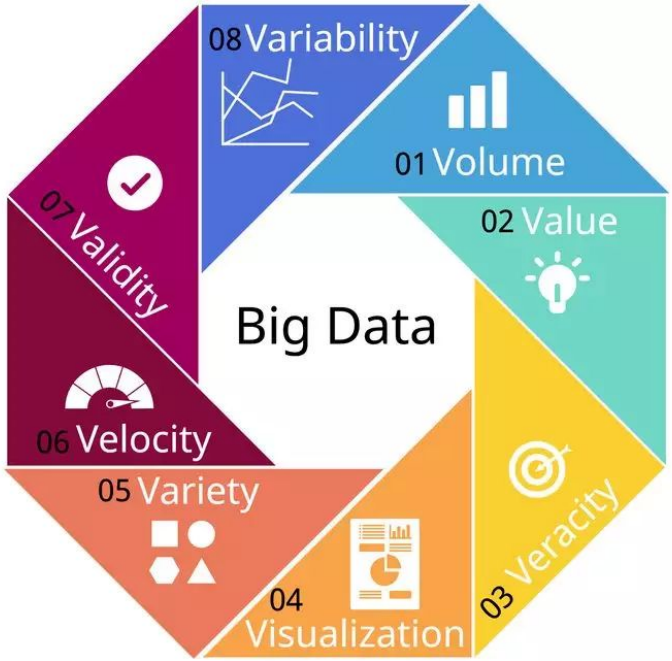


What is a datum?

credits: *M. Brescia*

The power of (Big) Data

If we collect enough data with a high-dimensional parameter space within any domain of study, then we would have a *consistent* statistical model for that domain: **ALL ANSWERS ARE AVAILABLE!**



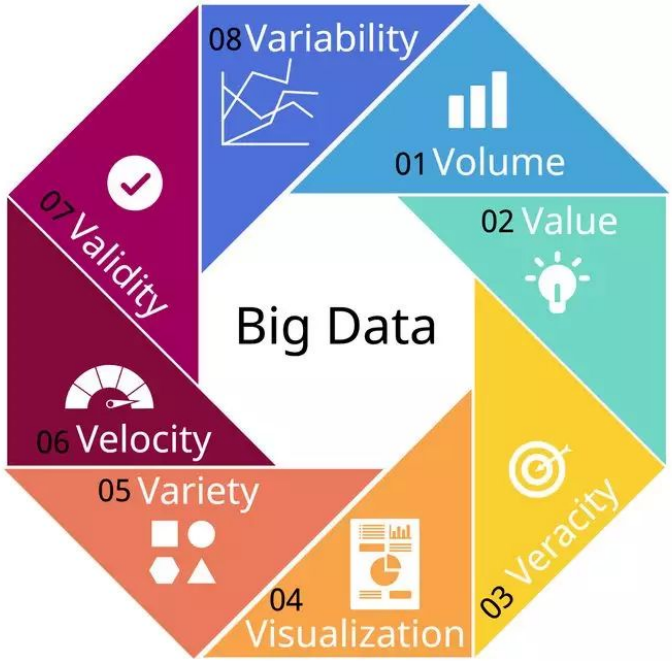
What is a datum?

- *a piece of information*
- *a fixed starting point of a scale or operation*
- *a real thing used for knowledge inference*
- *something for reasoning*

credits: *M. Brescia*

The power of (Big) Data

If we collect enough data with a high-dimensional parameter space within any domain of study, then we would have a *consistent* statistical model for that domain: **ALL ANSWERS ARE AVAILABLE!**



credits: *M. Brescia*

What is a datum?

- *a piece of information*
- *a fixed starting point of a scale or operation*
- *a real thing used for knowledge inference*
- *something for reasoning*

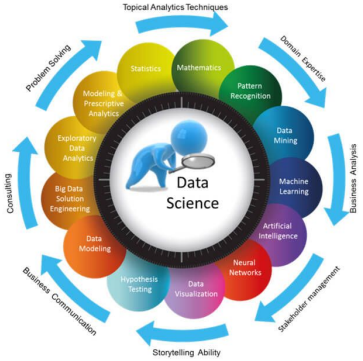
In a word...

SOLUTION

Each datum is a **solution** to the domain problem, thus in a Big Data regime we have an order of ~10^{20} solutions, each one in a multi-D parameter space!

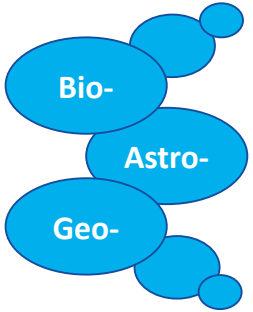
From Big Data Science to X-Informatics

Everything we want to know about a domain is specified and encoded within the data (i.e. **data-driven**).



Therefore, the final goal of any Big Data Science is to find those encodings, patterns and knowledge nuggets.

Big Data pushes for ICT in any Science domain X

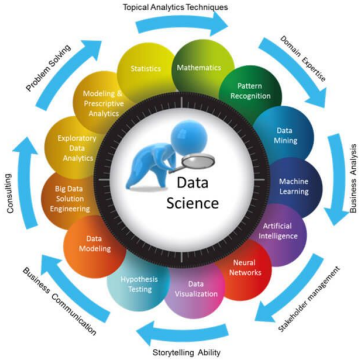


Big Data Science means:

- HPC/HTC demanding
- Efficient cross-correlation
- Self-adaptive learning machines

From Big Data Science to X-Informatics

Everything we want to know about a domain is specified and encoded within the data (i.e. **data-driven**).



Therefore, the final goal of any Big Data Science is to find those encodings, patterns and knowledge nuggets.

X-Informatics

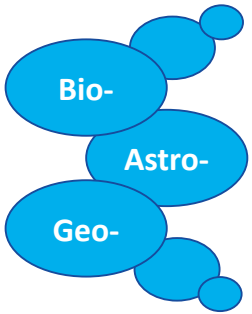
A set of new disciplines arisen from the **4th paradigm** of Science

Data-driven Science =
Scientific Knowledge Discovery in Databases

"One of the greatest challenges for 21st-century science is how we respond to this new era of data intensive science [...] This is recognized as the 4th paradigm beyond experimental and theoretical research and computer simulations of natural phenomena - one that requires new tools, techniques, and ways of working."

Jim Gray (Turing Award 1998)

Big Data pushes for ICT in any Science domain X



Big Data Science means:

- HPC/HTC demanding
- Efficient cross-correlation
- Self-adaptive learning machines

Astroinformatics

Basically, it contains all of the components of Data Science, in their astronomical applications

Characterize the known

Feature selection, Parameter space analysis

Assign the new from the known

Supervised learning, Regression, Classification

Explore the unknown

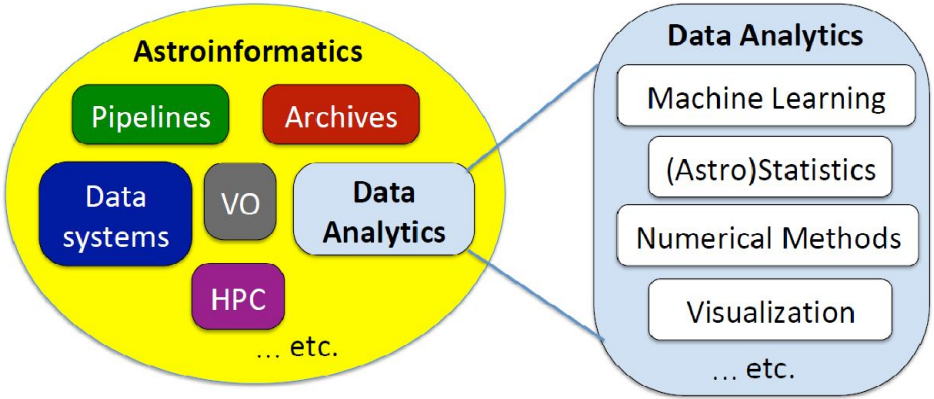
Clustering, unsupervised learning

Discover the unknown

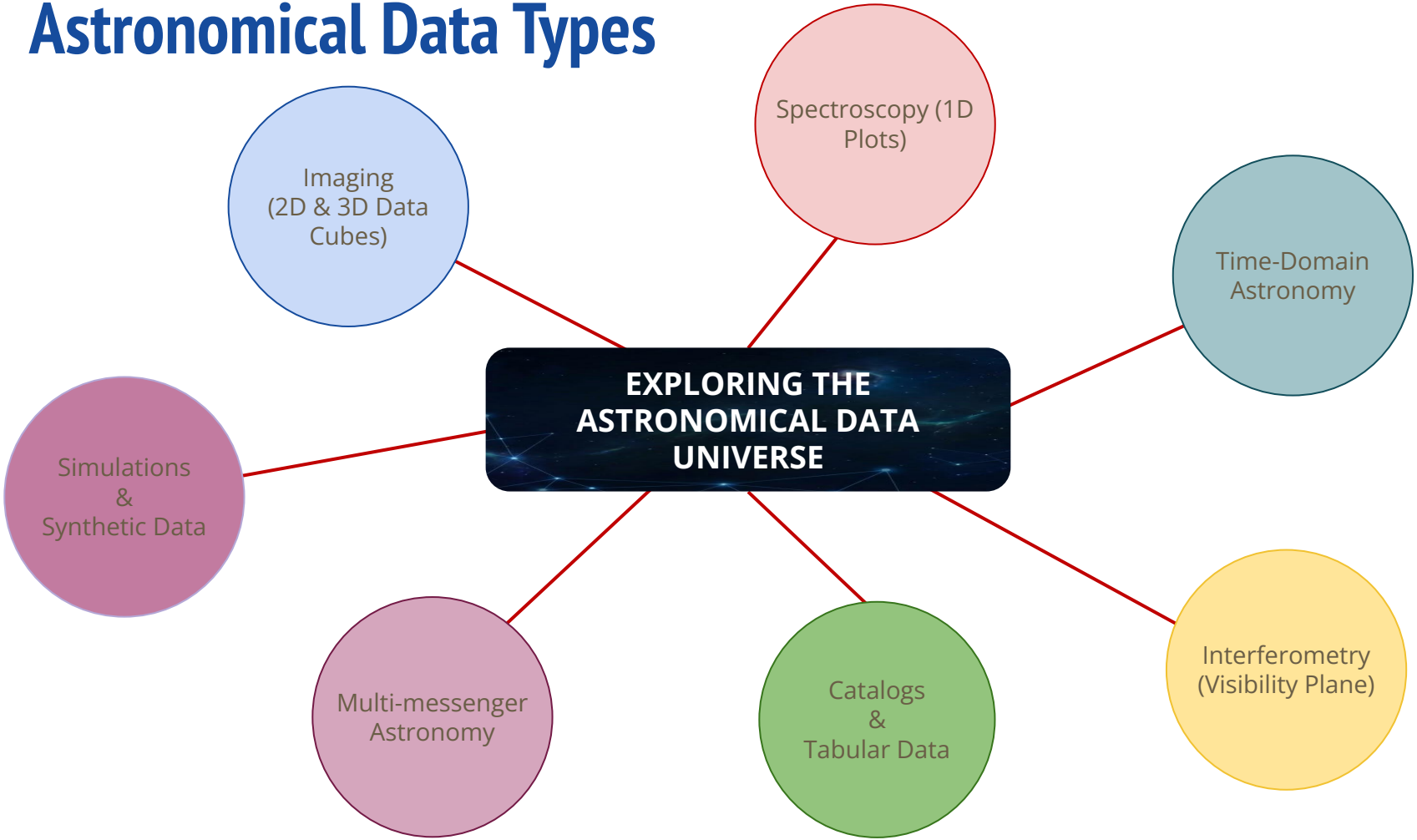
Outlier detection and analytics (serendipity)

Benefits of very large datasets

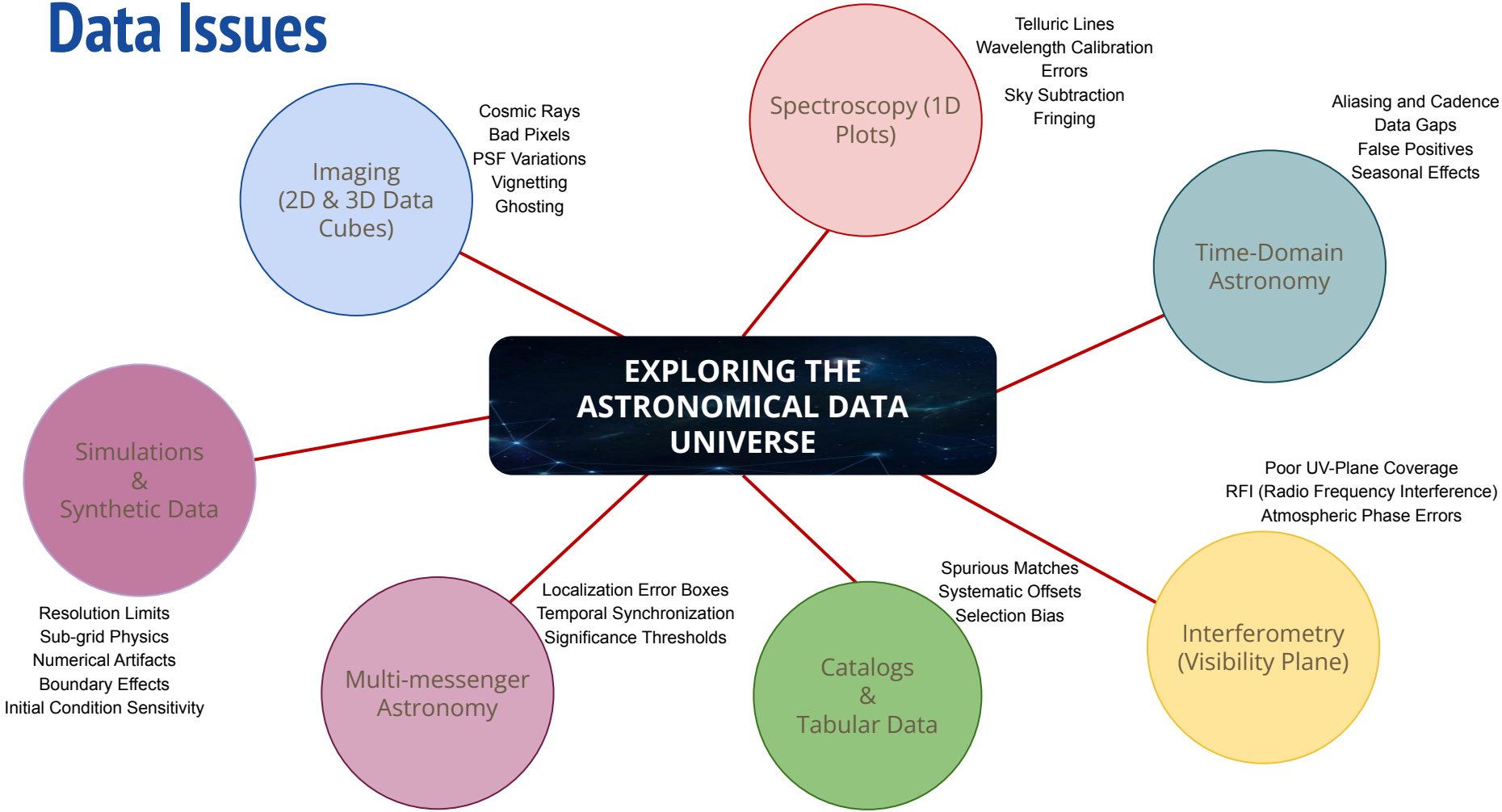
Statistics of “typical” events, cross-correlation, automated search for “rare” events



Astronomical Data Types



Data Issues


EXPLORING THE ASTRONOMICAL DATA UNIVERSE
Imaging (2D & 3D Data Cubes)
Cosmic Rays
Bad Pixels
PSF Variations
Vignetting
Ghosting
Spectroscopy (1D Plots)
Telluric Lines
Wavelength Calibration Errors
Sky Subtraction
Fringing
Time-Domain Astronomy
Aliasing and Cadence
Data Gaps
False Positives
Seasonal Effects
Simulations & Synthetic Data
Resolution Limits
Sub-grid Physics
Numerical Artifacts
Boundary Effects
Initial Condition Sensitivity
Multi-messenger Astronomy
Localization Error Boxes
Temporal Synchronization
Significance Thresholds
Catalogs & Tabular Data
Spurious Matches
Systematic Offsets
Selection Bias
Interferometry (Visibility Plane)
Poor UV-Plane Coverage
RFI (Radio Frequency Interference)
Atmospheric Phase Errors

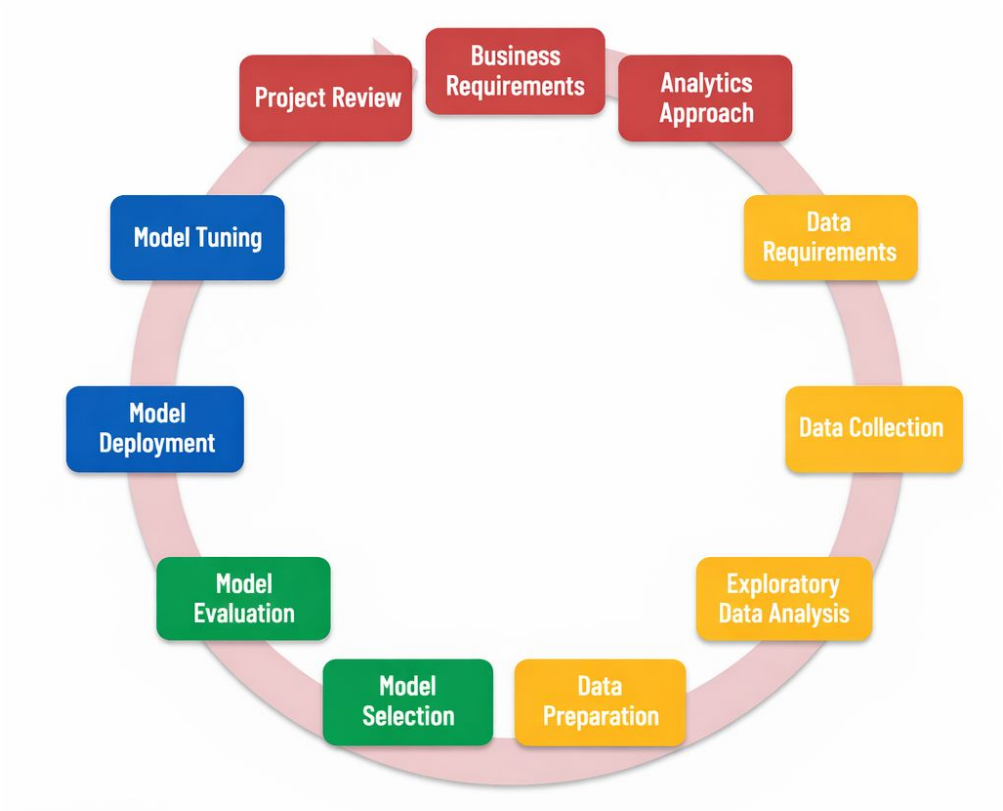
Basic Concepts of ML

What is ML… a summary

ML “consists in the development of **algorithms** that allow an **automatic adaptation of computers capabilities** to solve the assigned problem based on empirical data. […] Such techniques embed the intrinsic **data-driven learning** capability to explore huge amounts of **multi-dimensional data** by searching for **hidden correlations** within the data parameter space. […] A ML method can be thought as a learner which exploits examples of data to comprehend characteristics, discerns patterns, uncover anomalies and associations from the unknown underlying data distributions” [Bengio, 2009]

- **algorithms** → ML is a branch of informatics
- **automatic adaptation of computers capabilities** → improves its performance by “training” on data
- **data-driven learning** → research starts from available data, not from physical theory
- **multi-dimensional data** → born to handle with a huge amount and variety of data
- **hidden correlations** → able to find correlations not accessible to human

Approaching a ML project

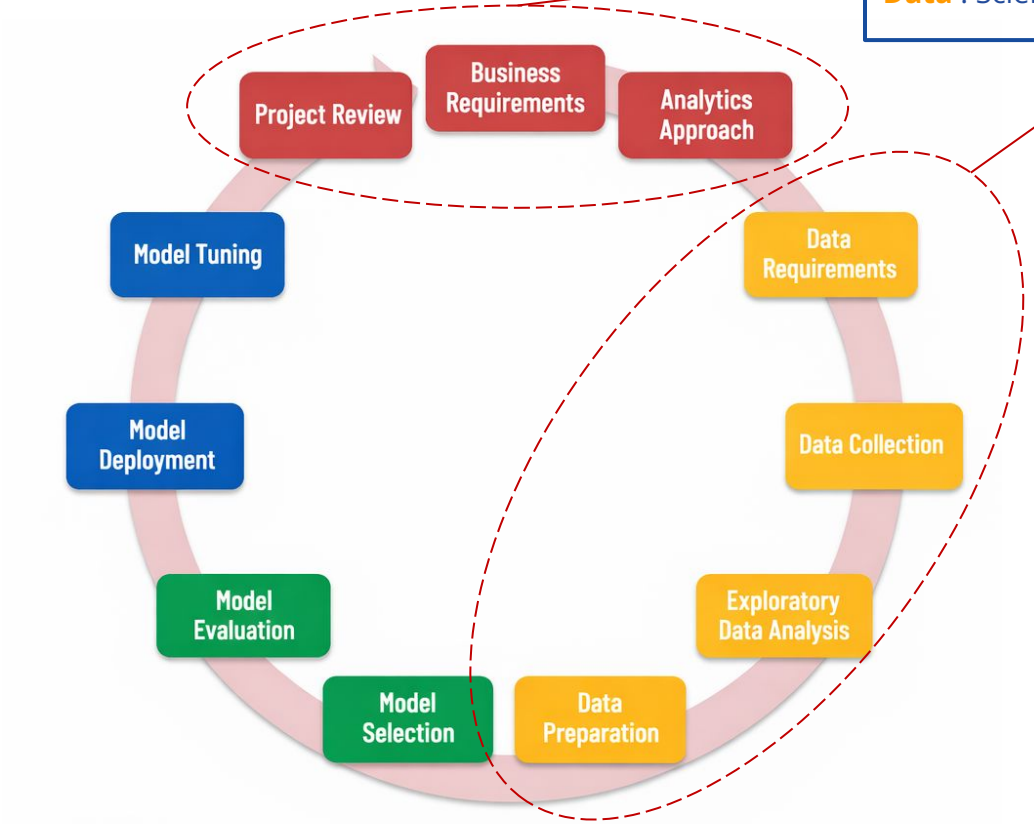


credits: *Christopher S. Penn | @cspenn | @TrustInsights*
cspenn.com | TrustInsights.ai

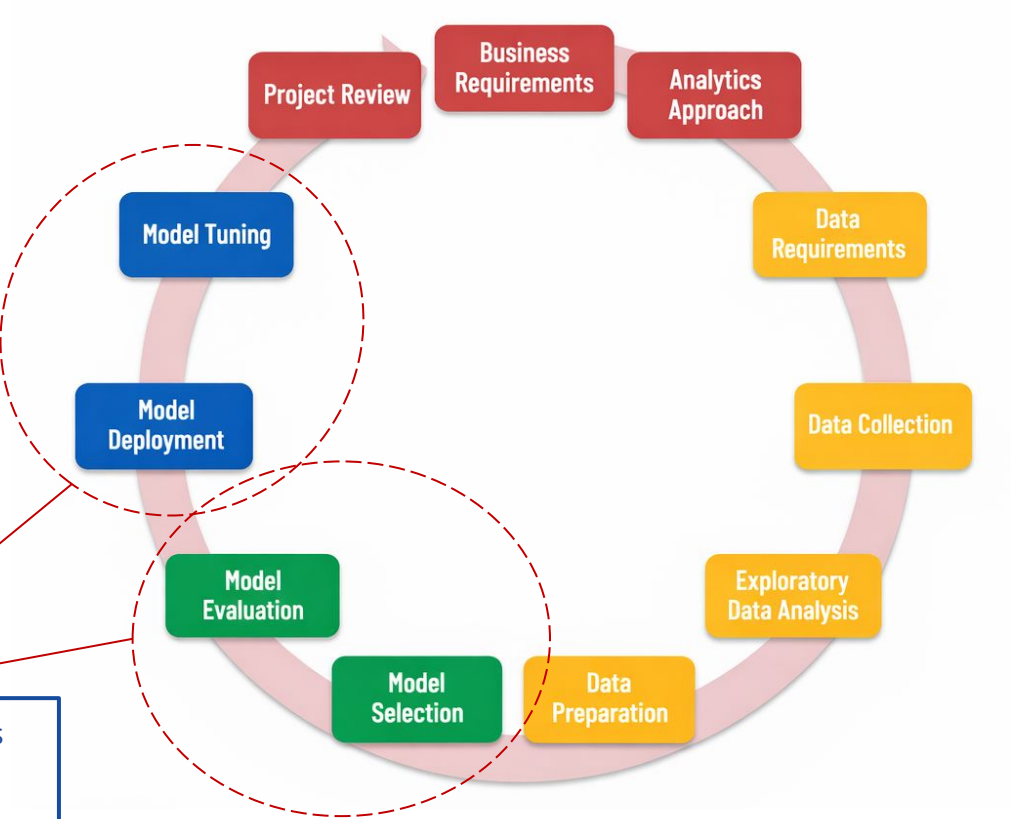
Approaching a ML project

Planning : Scientists

Data : Scientists + DB + Statistics (+ML)



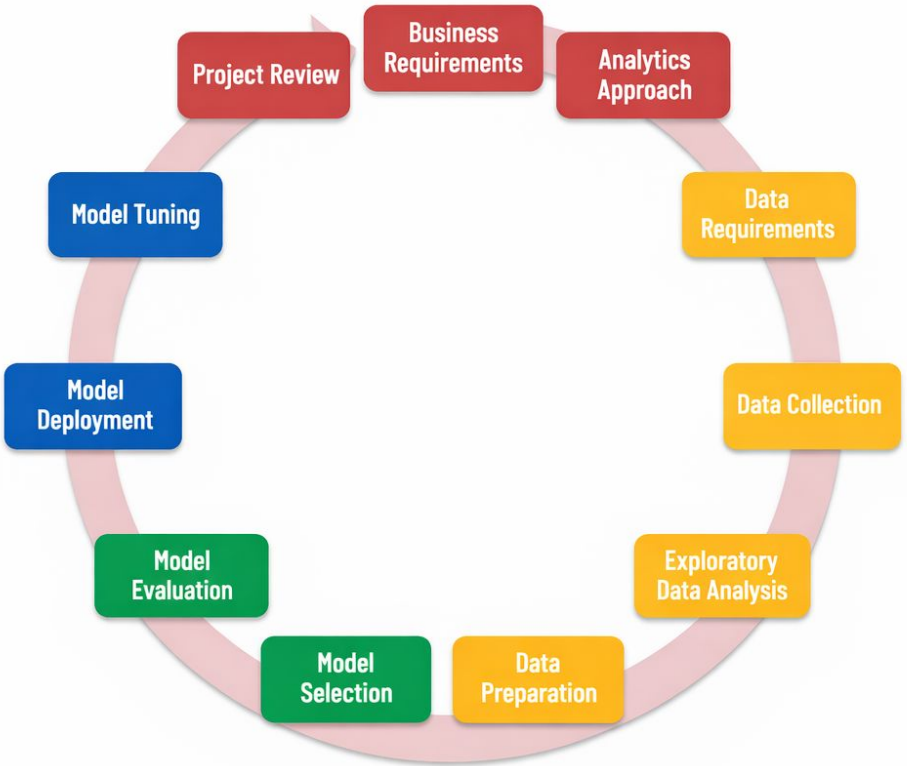
Approaching a ML project



Development : ML + Scientists

Deployment : Scientists + ML

Approaching a ML project



Planning : Nothing to say 😉

Data Requirements, Collection and Preliminary Data Analysis :

- Essential raw material.
- Relevance and representativeness are crucial.

Data Preparation : Cleaning, transformation and organization

- Missing data handling
- Normalization
- Encoding
- Dataset splitting

Model Selection :

- Parameters optimization
- Learning and refining
- Hyperparameters tuning

Model Evaluation :

- Evaluation on test set
- Measuring generalization ability

Deployment : Integration into real systems/decision-making processes

Model Tuning : Observe, monitor metrics and tuning to improve model performance

Machine Learning Pros & Cons

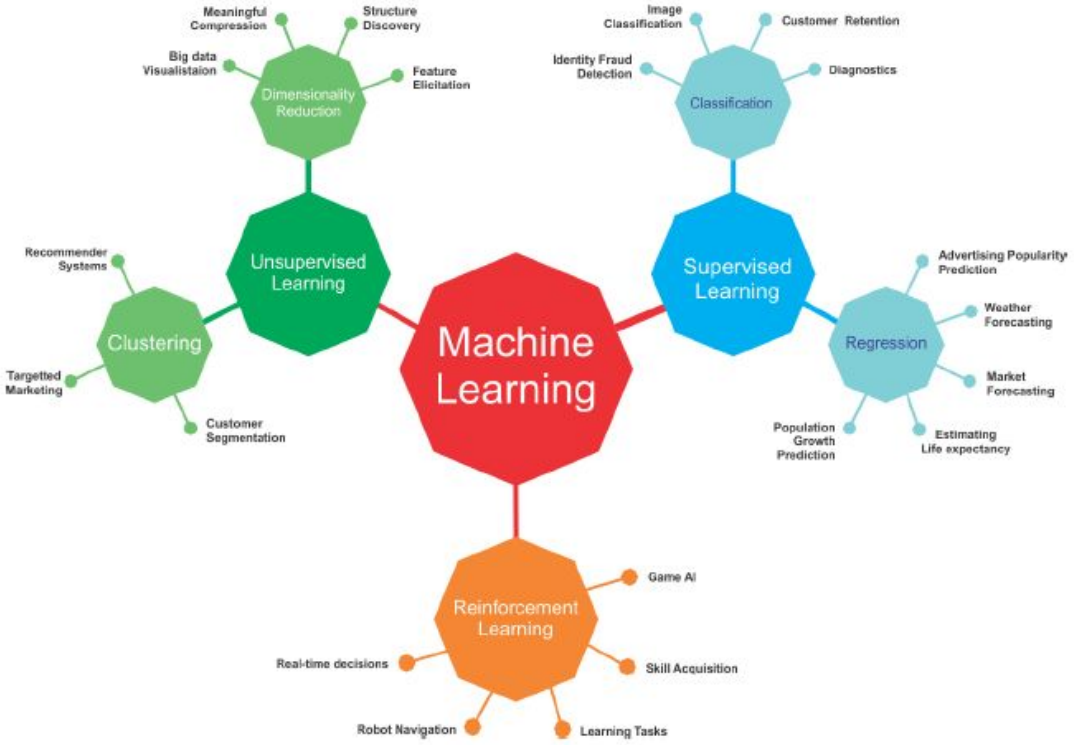
PROs

- **Process Automation** : ML enables systems to make decisions and adapt to changes without the need for direct human intervention
- **Predictive Ability** : Provides powerful tools to predict future outcomes by analyzing historical data
- **Continuous learning** : Models can be updated with new data, improving their performance over time
- **Cross-sector applicability** : ML techniques can be used in various sectors, such as medicine, agriculture, marketing, industry... and astronomy!

CONs

- **Requires large amounts of data** : Model performance depends heavily on the availability of relevant and representative data.
- **Risk of overfitting** : Overly complex models may memorize training data and fail to generalize well to new data.
- **Data bias** : If the training dataset is biased or unbalanced, the model may produce inaccurate or discriminatory results.
- **Limited interpretability** : Many complex models, such as deep neural networks, are difficult to interpret and explain.
- **Computational costs** : Training and using advanced models can require significant hardware resources and high processing times.

Machine Learning paradigms

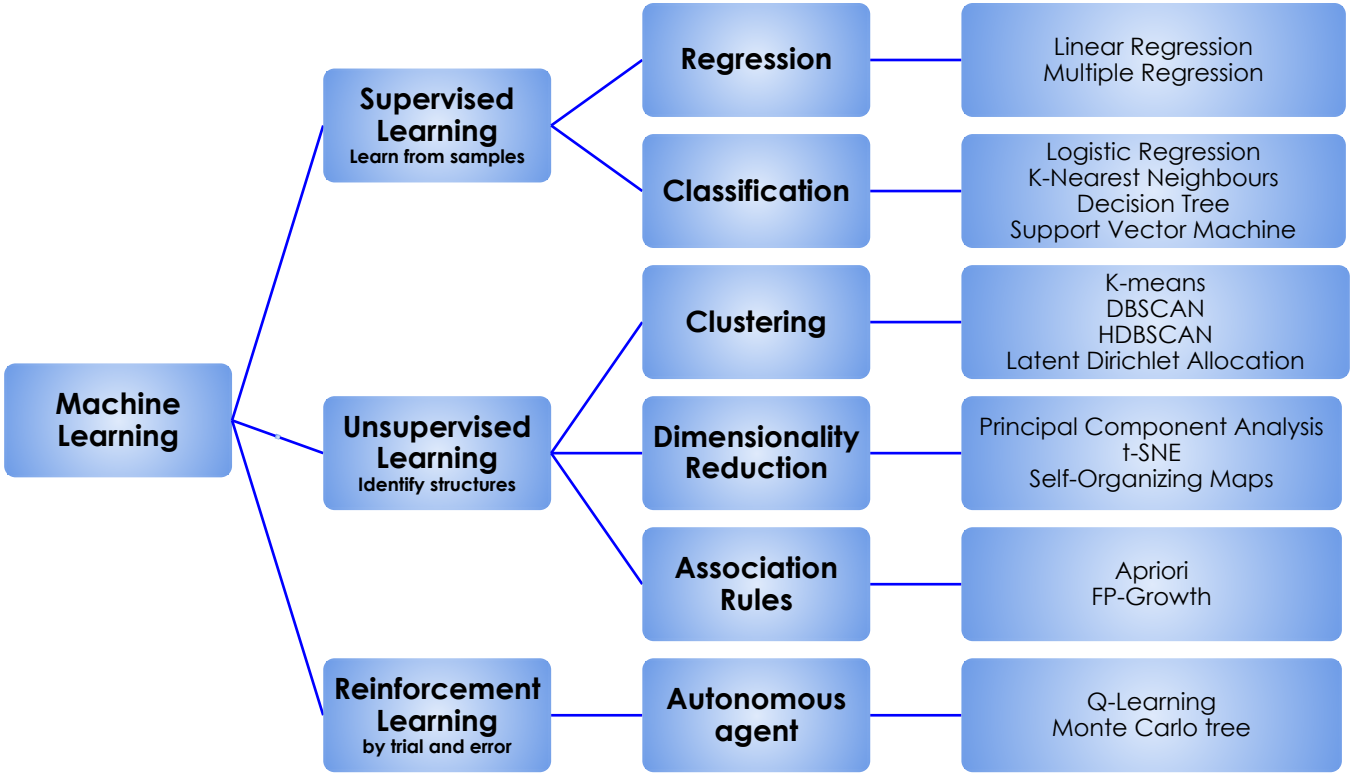


- **Supervised learning** trains a model using labeled data where each input has a known correct output. The model learns by comparing its predictions with these correct answers and improves over time. It is used for both **classification** and **regression** problems

- **Unsupervised learning** works with unlabeled data where no correct answers or categories are provided. The model's job is to find the data, hidden patterns, similarities or groups on its own. This is useful in scenarios where labeling data is difficult or impossible. Common applications are **clustering** and **association**

- **Reinforcement Learning** trains an agent to make decisions by interacting with an environment. Instead of being told the correct answers, agent learns **by trial and error** method and gets rewards for good actions and penalties for bad ones. Over time it develops a strategy to maximize rewards and achieve goals. This approach is good for problems having sequential decision making such as **robotics, gaming and autonomous systems**.

Machine Learning paradigms



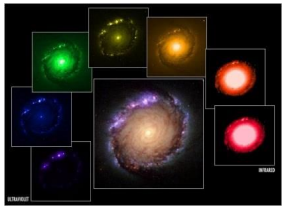
Glossary

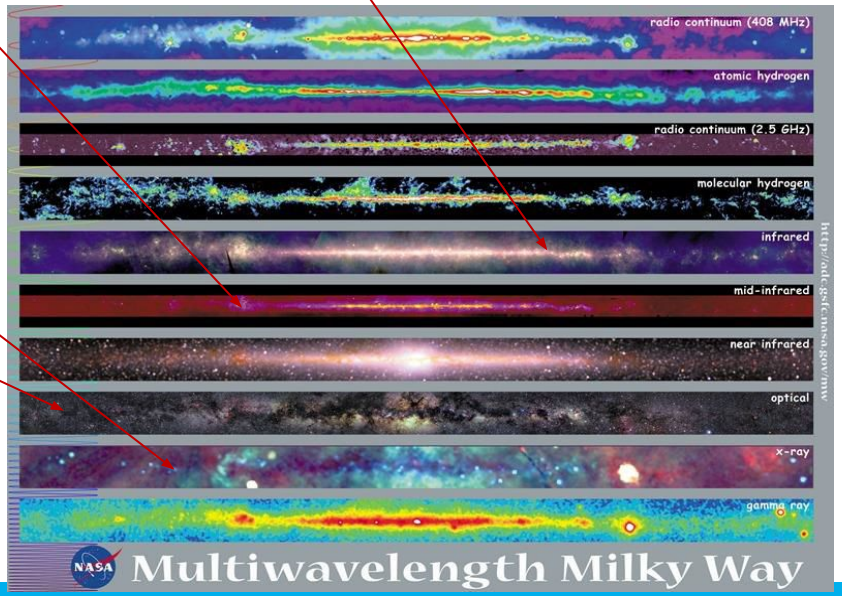
- **Data** can be tables, images, streaming vectors.
  They may be represented under the form of numbers, percentages, pixel values, literals, strings, probabilities, any other entity giving an information on a physical/conceptual/simulated event or phenomena of our world
- **Dataset** is a set of samples representing a problem.
  All samples must be expressed in a uniform way (i.e. same dimensions and representation)
- **Pattern** is a sequence of symbols/values identifying a single sample of any dataset
- **Feature** is an atomic element of a pattern, i.e. a number or symbol representing one characteristic of the pattern (carrier of hidden information)
- **Target (supervised dataset)** is usually a label or a set of labels representing the solution (desired/known output) of a single pattern.
  If unknown or missing, the pattern belongs to the unsupervised category of datasets
- **Base of Knowledge or KB (Knowledge Base)** is the ensemble of datasets in which the patterns contain the target (known solutions to a real problem) It is always available for supervised ML
- **Model** is a mathematical function f that maps input features to an output or prediction. It is defined by its structure and its parameters $\boldsymbol{\theta}$, which are optimized during training.

Examples of BoK

Dataset: set of galaxies observed by a space telescope. Each **pattern** has 15 **features** (different emission flux wavelength) + one **target** (redshift)

UMAG, GMAG, RMAG, IMAG, ZMAG, nuv, fuv, YMAG, JMAG, HMAG, KMAG, w1, w2, w3, w4, **zspec**
20.38, 20.46, 20.32, 20.09, 20.04, 0.65, 3.21, 19.28, 18.963, 19.286, 17.505, 16.828, 15.238, 12.238, 8.579, 1.824
19.465, 19.368, 19.193, 19.015, 0.219, 1.397, 18.29, 17.76, 16.97, 15.77, 14.26, 13.2, 10.76, 8.158, 0.459, 1.934
17.995, 17.934, 17.873, 1.865, 0.132, 16.863, 16.597, 15.902, 14.75, 13.33, 12.28, 9.5, 7.37, 0.478, 20.49, 2.247
20.13, 20.36, 1.43, 4.22,19.906, 19.409, 18.427, 17.935, 17.076, 15.589, 12.619, 8.863, 1.4365, 8.15, 0.45, 1.93



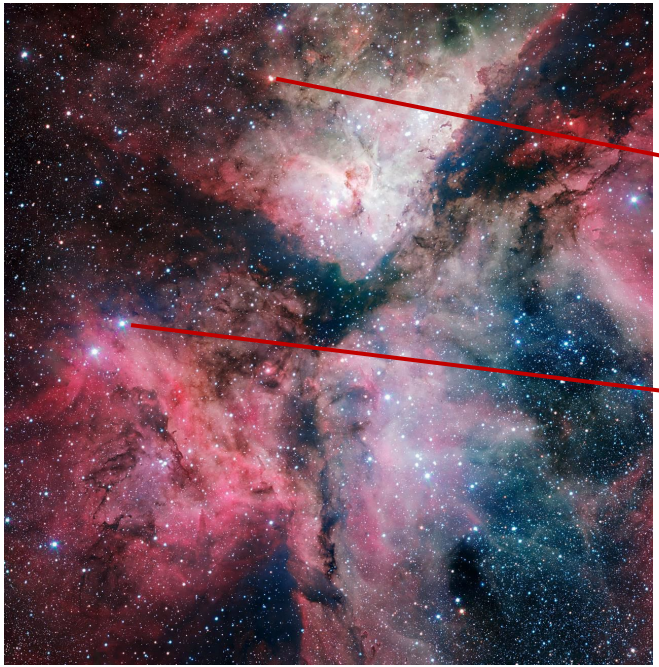


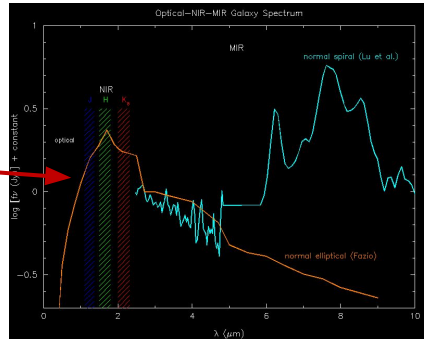
Usually such **BoK** is made by hundreds of thousands of galaxies (**patterns**)

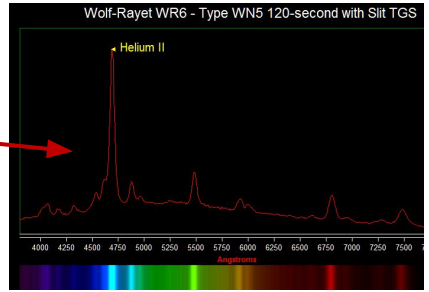
The ML problem is to learn to **predict** their redshift for new objects observed in further space missions

Examples of BoK

- **Dataset:** large multi band image of a nebula, million of patterns of galaxies and stars (their spectra)
- **Features:** peaks in the object spectrum
- **Target:** the type of object
- **ML Problem (supervised):** learn to **classify** objects (such as star/galaxy separation)







Supervised Learning

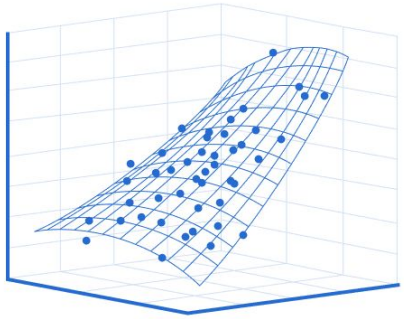
Supervised ML definitions

In supervised machine learning, we have a set of data points or observations (described by **features**) for which we know the desired output (**target**), expressed in terms of categorical classes, numerical or logical variables, or as a general observed description of a real-world problem (**labels**). The desired output provides a certain level of supervision, as it is used by the **learning model** to adjust parameters or make decisions that allow it to predict the correct output for new data.

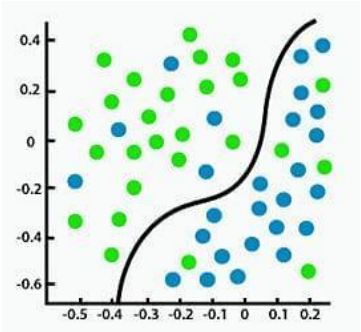
The two most basic tasks are **regression** and **classification**, depending on whether the target is numerical or categorical. When the algorithm is able to correctly predict observations, we call it a ***classifier*** or ***regressor***.

Some classifiers are also able to provide results in a more probabilistic sense, that is, a probability that a data point belongs to a class. We usually refer to this model behavior as *probabilistic classification*.

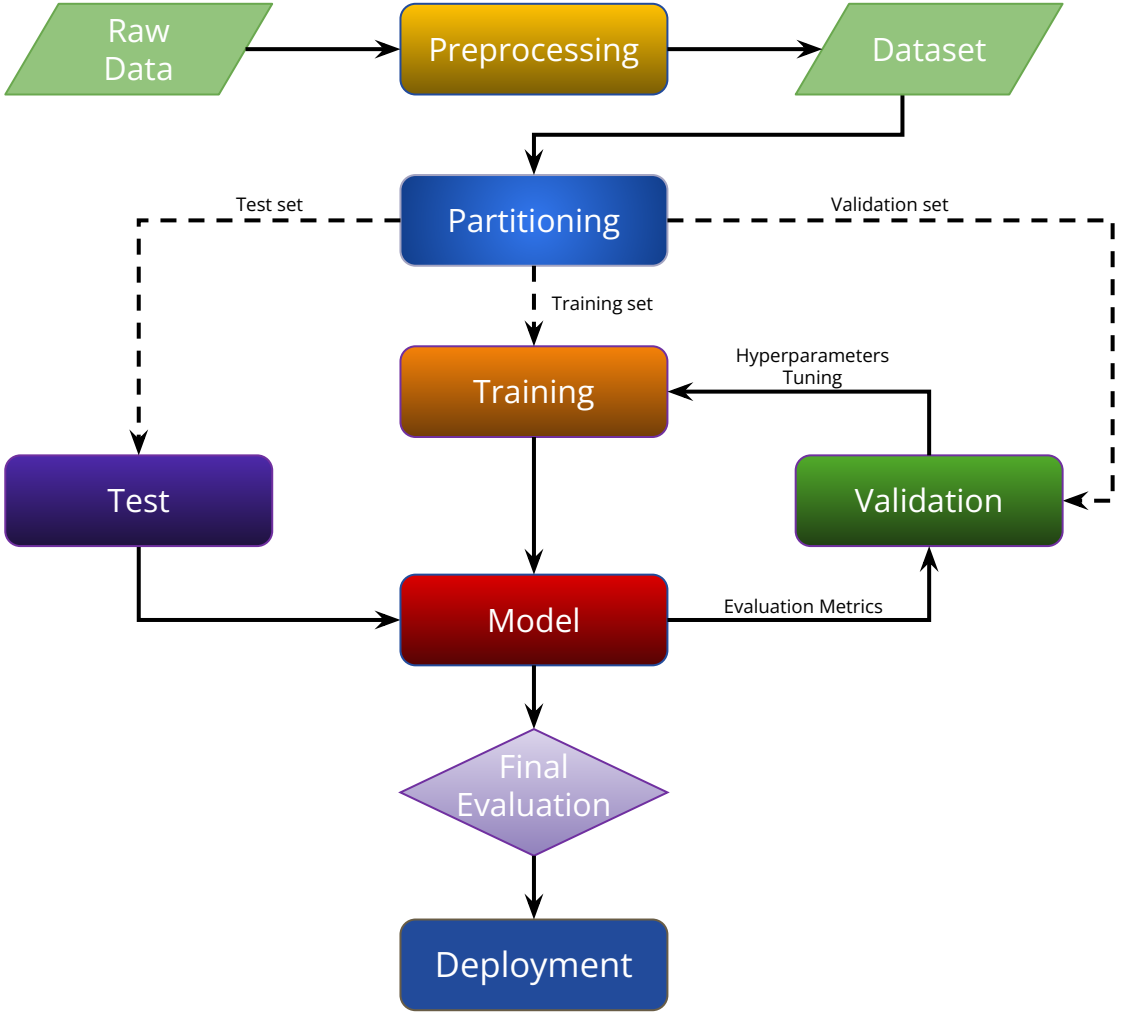
Regression: $\mathcal{Y} \subseteq \mathbb{R}$



Classification: $\mathcal{Y} = \{C_1, ..., C_g\}$



Workflow

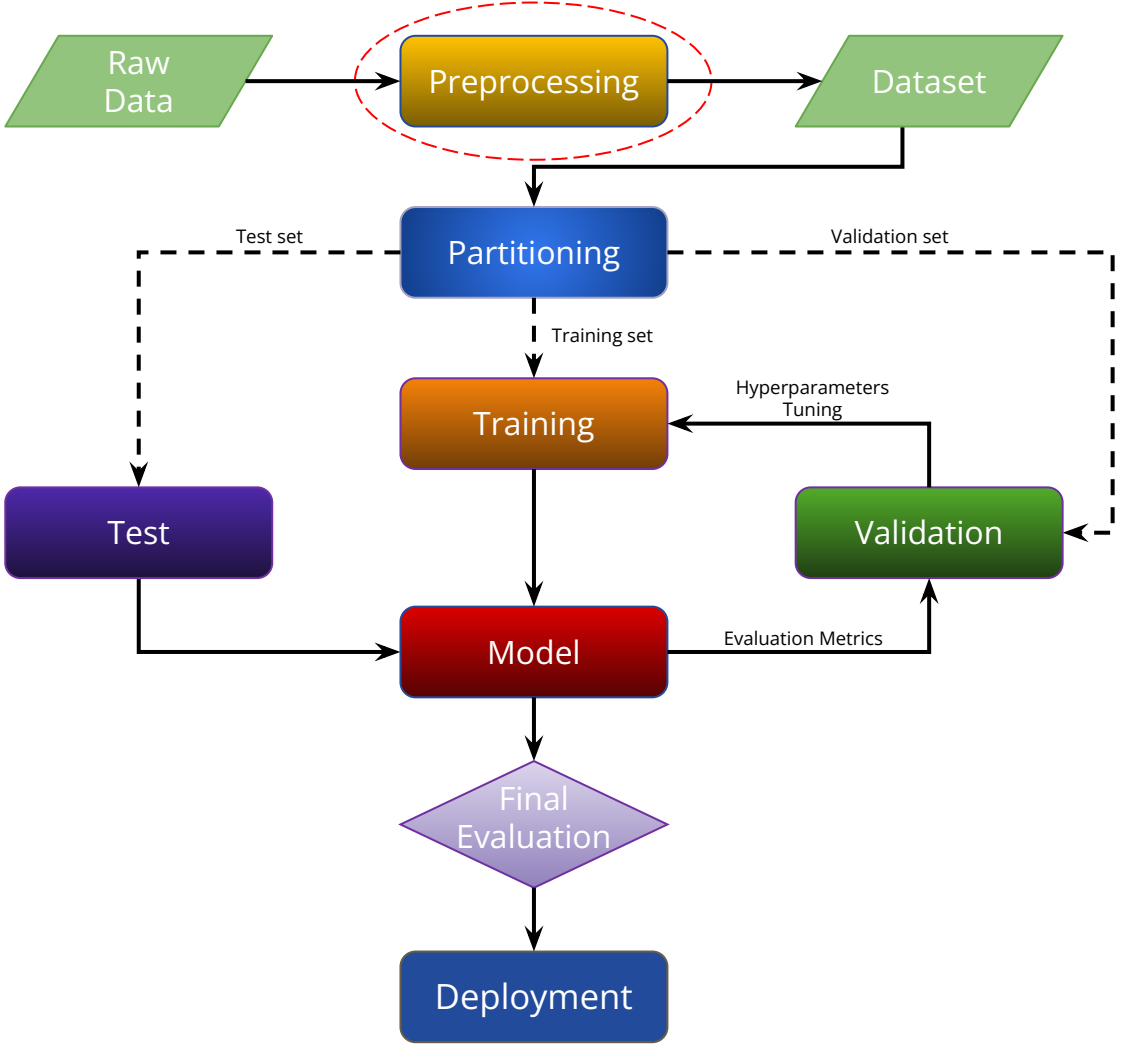

Raw Data
Preprocessing
Dataset
Partitioning
Test set
Validation set
Training set
Training
Hyperparameters Tuning
Test
Validation
Model
Evaluation Metrics
Final Evaluation
Deployment

Workflow

Preprocessing

Build input patterns appropriate for feeding into our supervised learning algorithm. This includes scaling and preparation of data:

- Cleaning
- Encoding
- Feature Scaling - Normalization - Standardization
- Missing values handling



Workflow

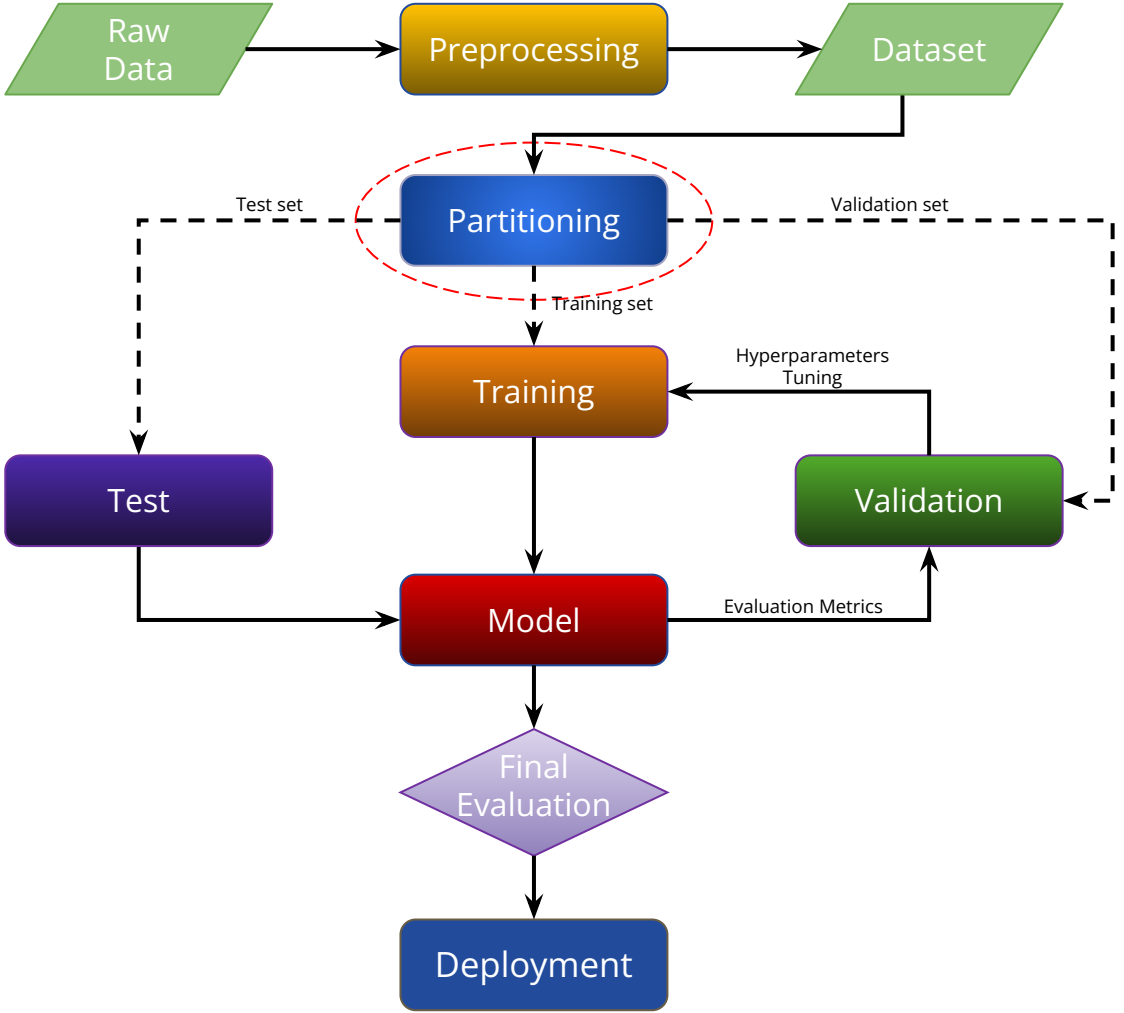
Partitioning

Create data sets for training and evaluation, **randomly** splitting the universe of data patterns (**Knowledge Base**).

Training Set: used to train the model, i.e. to estimate its parameters

Validation Set: used to optimize the training and to to choose the best setup

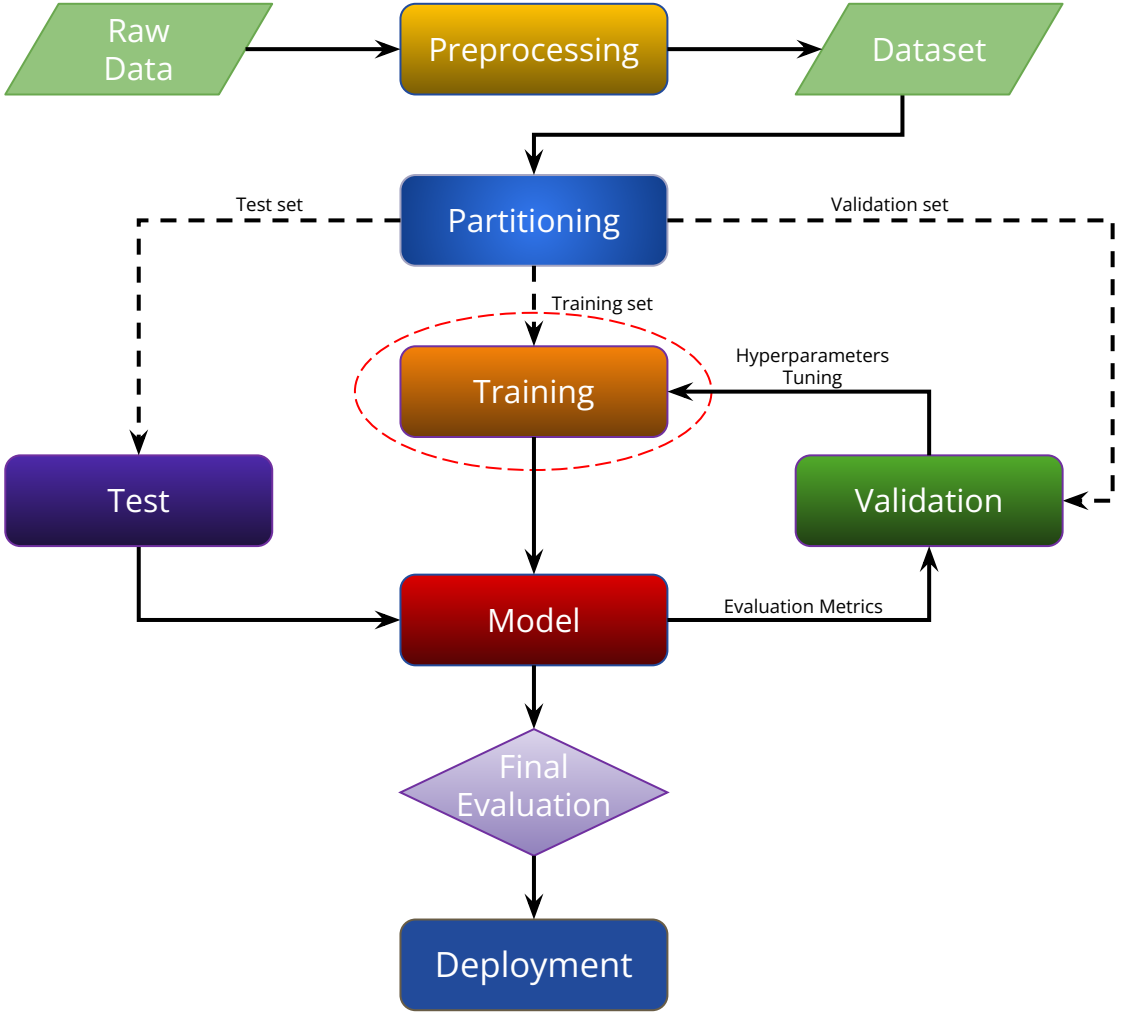
Test Set: to independently measure the final performance of the network



Workflow

Training

We execute the model on the training data set. The output result consists of a **model** that (in the successful case) has learned **how to predict** the outcome when new unknown data are submitted

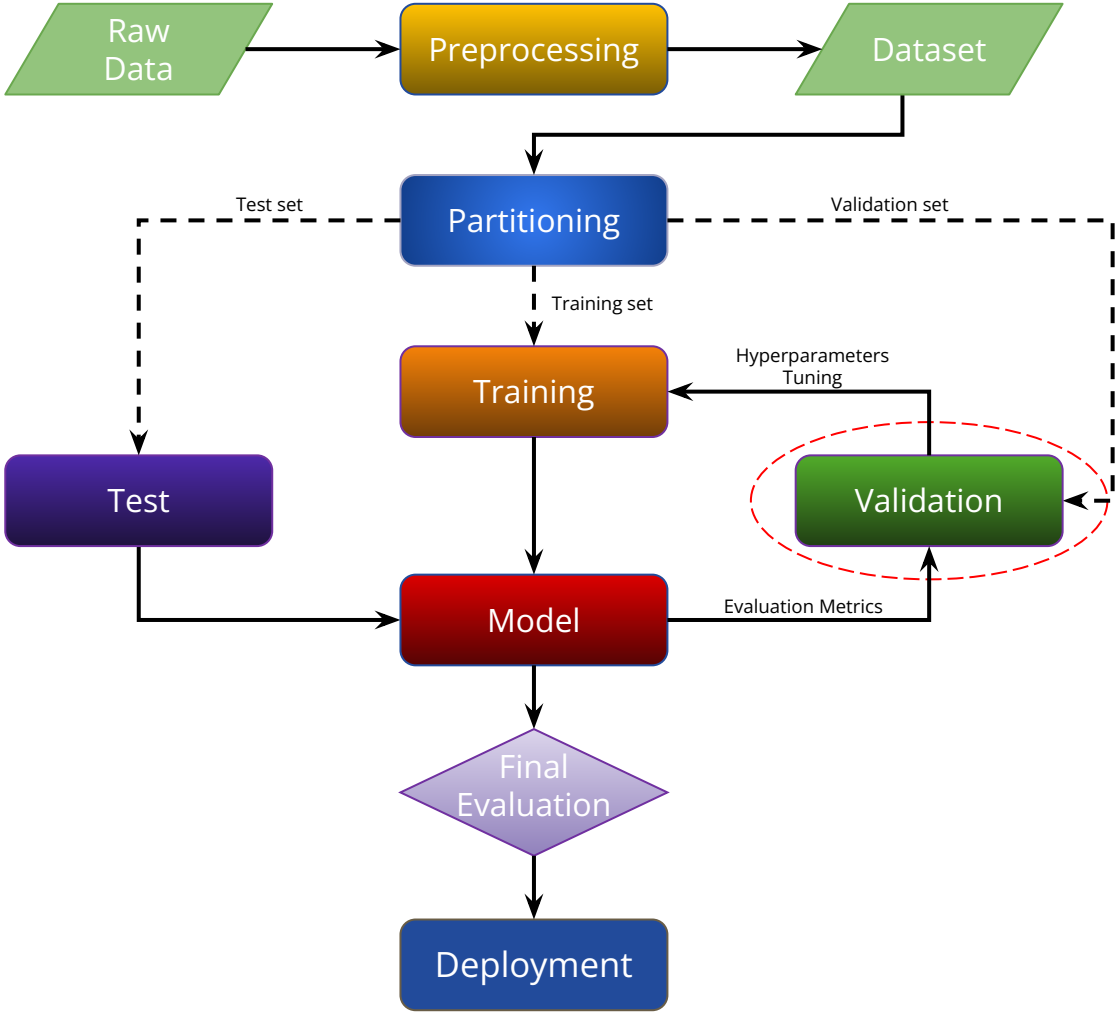


Workflow

Validation

After we have created the model, it is of course required a test of **its performance** accuracy, completeness and contamination (or its dual, the purity). It is particularly crucial to do this on data that the model has **not seen yet**. This is main reason why on previous steps we separated the data set into training patterns and a subset of the data not used for training.

If the classification error of the validation set is higher than the training error, then we have to go back and adjust model parameters.

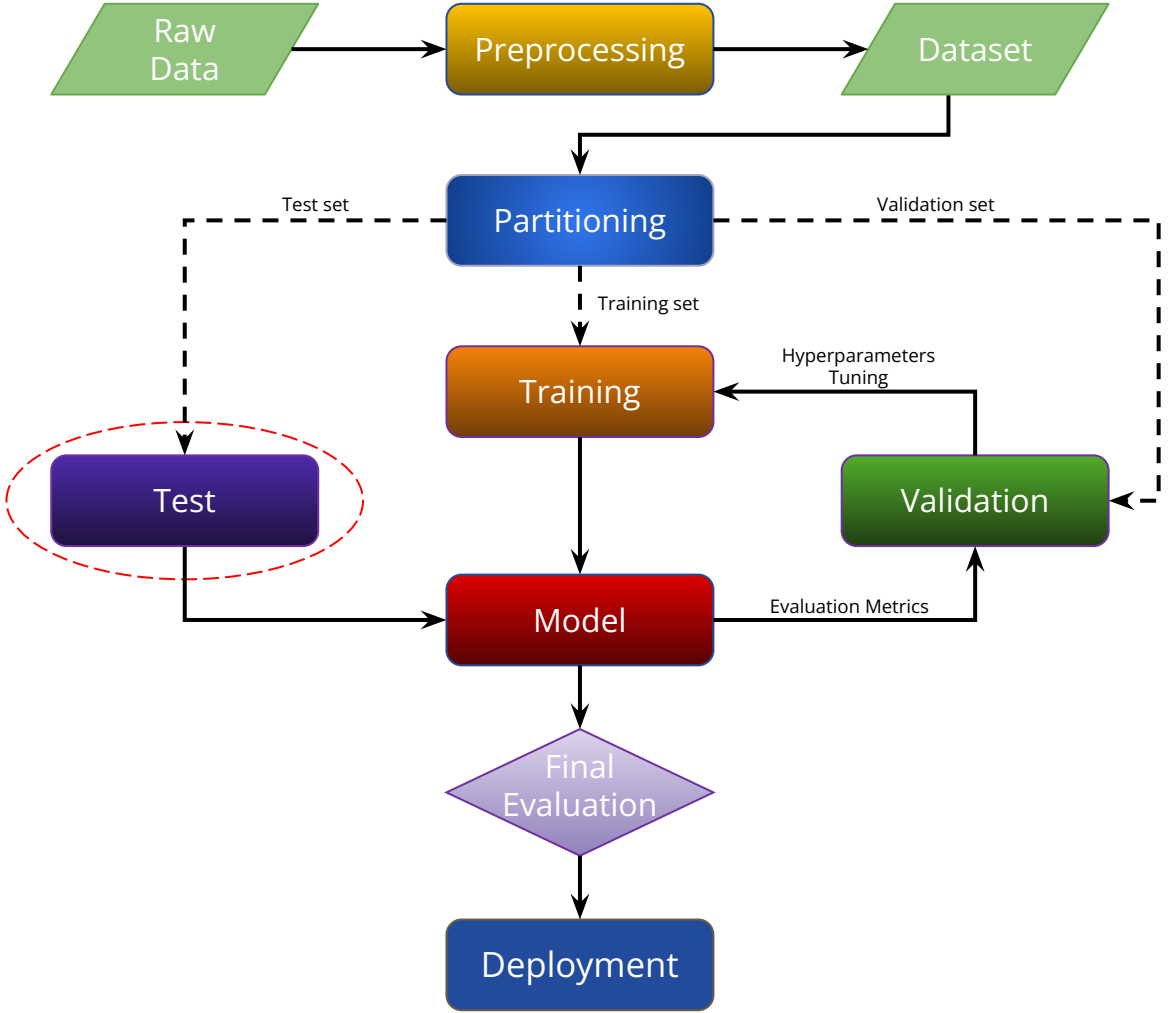


Workflow

Test

In this phase we can **verify and measure** the **generalization** capabilities of the model.

It is very easy to learn every single combination of input vectors and their mappings to the output as observed on the training data, and we can achieve a very low error in doing that, but *how does the very same rules or mappings perform on new data that may have different input to output mappings?*



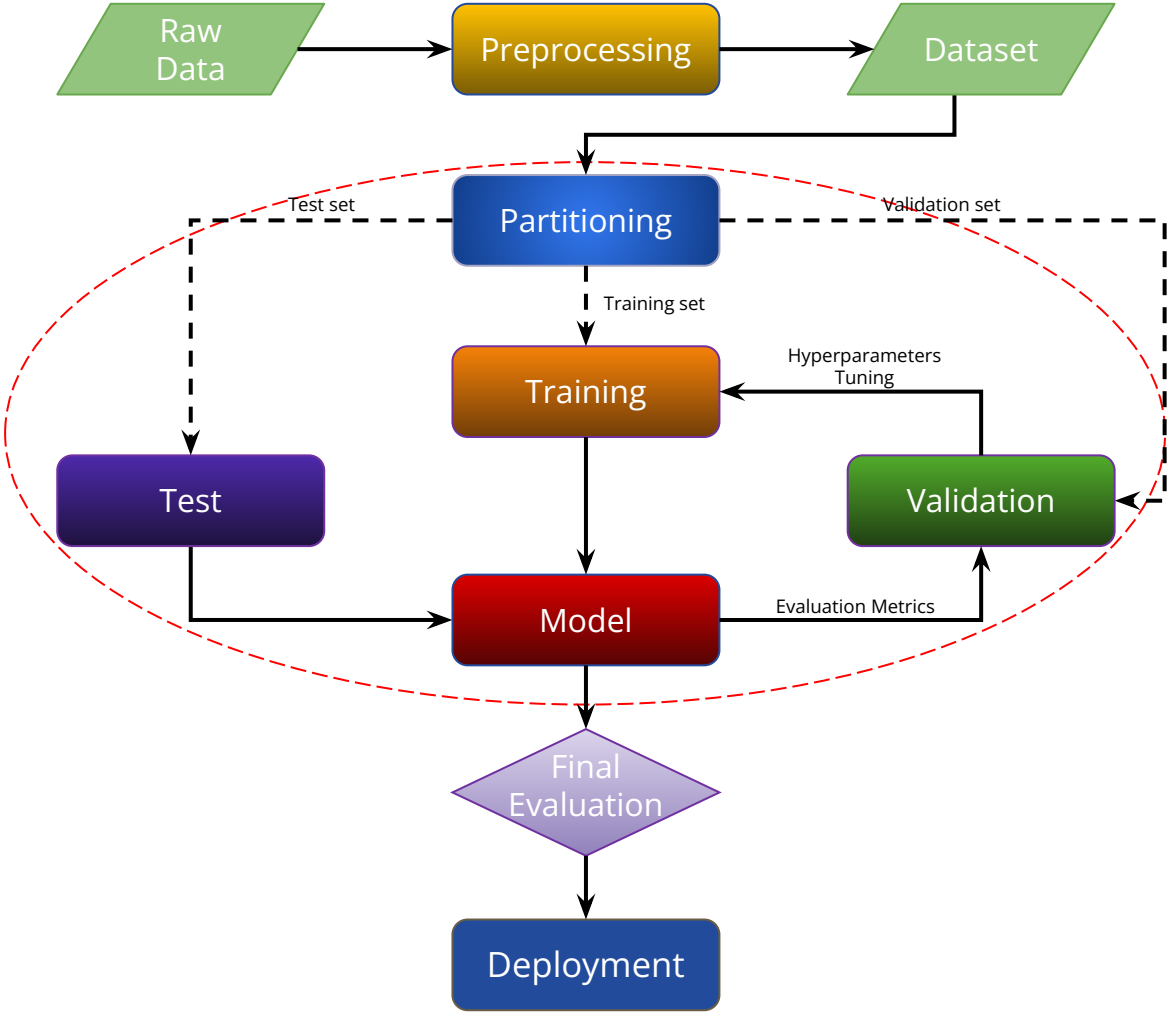
Workflow

A good practice

Initial conditions (dataset partitioning, internal random seed, network initialization) can significantly influence the model definition.

For a more robust model evaluation, it is strongly recommended to run the ML steps a sufficient number of times to generate reliable statistics.

How often does this actually occur?

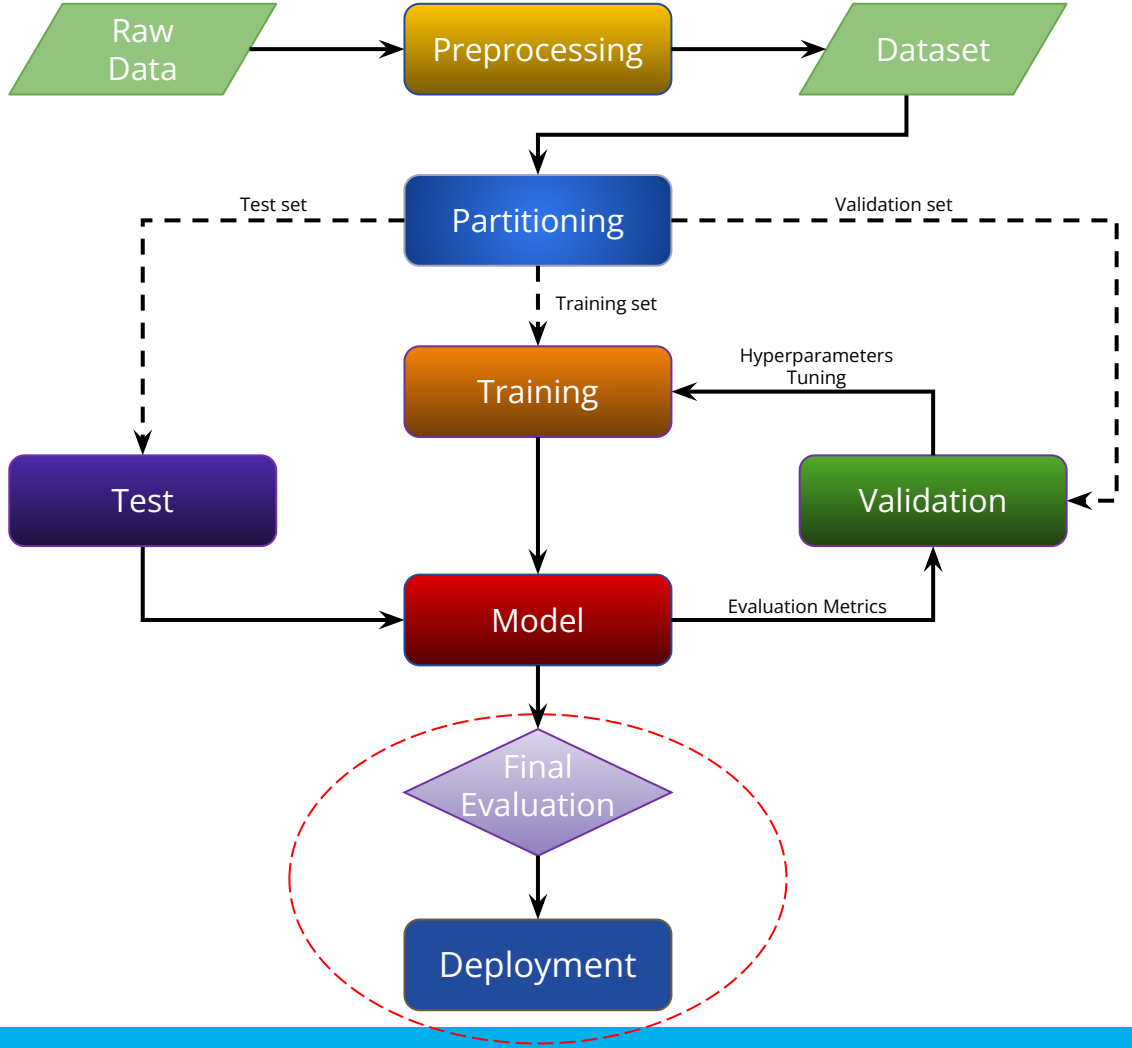


Workflow

Deployment... or not

If validation and test were successful, the model has **correctly learned** the underlying real problem. So far we can proceed to use the model to classify/predict new data.

Otherwise...

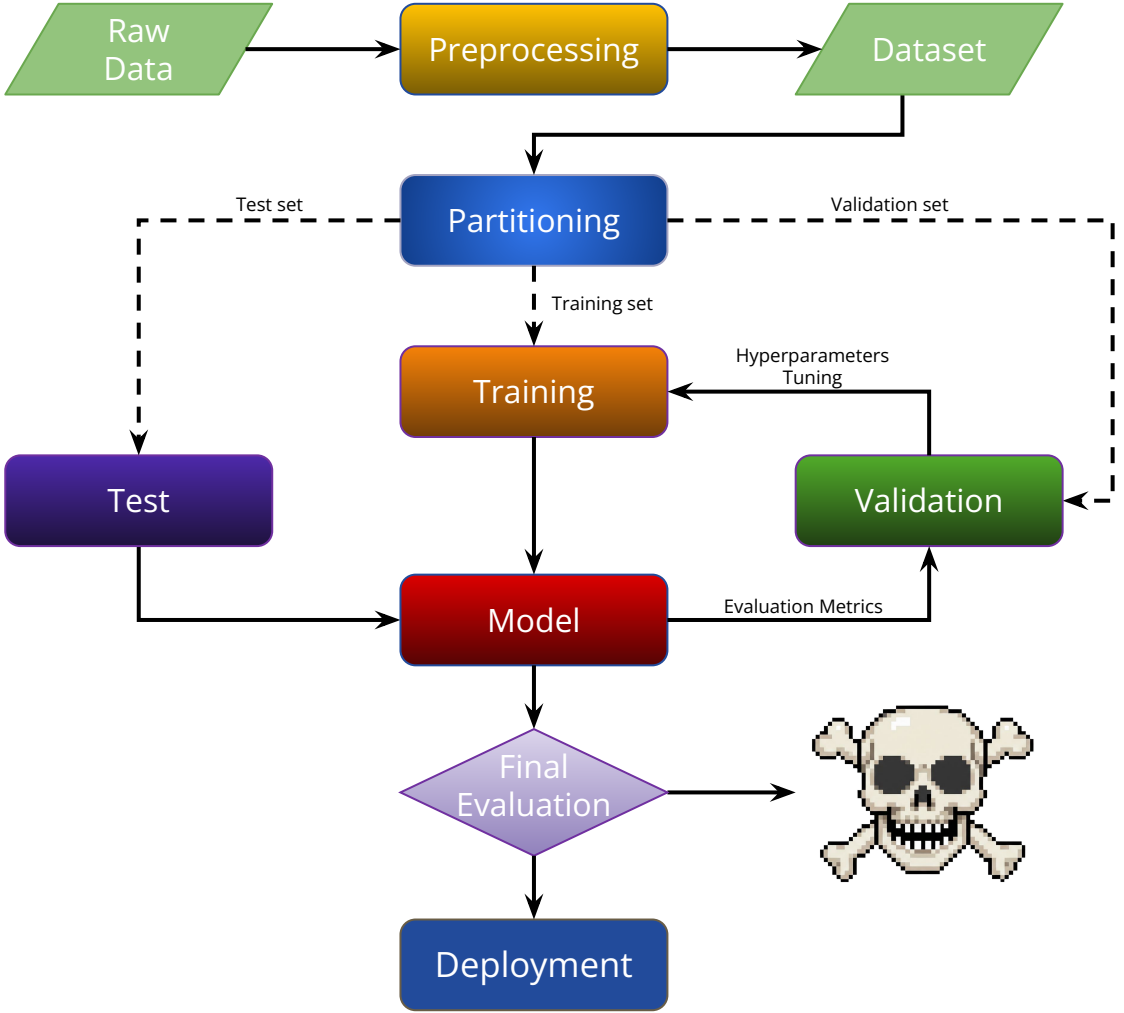


Workflow

Deployment... or not

If validation and test were successful, the model has **correctly learned** the underlying real problem. So far we can proceed to use the model to classify/predict new data.

Otherwise...



Data Preprocessing

Data preprocessing is the first step in any data analysis or machine learning pipeline. It involves cleaning, transforming and organizing raw data to ensure it is accurate, consistent and ready for modeling. It has a big impact on model building such as:

- Clean and well-structured data allows models to learn meaningful patterns rather than noise.
- Properly processed data prevents misleading inputs, leading to more reliable predictions.
- Organized data makes it simpler to create useful inputs for the model, enhancing model performance.
- Organized data supports better Exploratory Data Analysis (EDA), making patterns and trends more interpretable.

Data Cleaning

- **Handling Missing Data**: Impute missing values or remove rows/columns with excessive missing data.
- **Removing Duplicates**: Identify and eliminate duplicate records.
- **Outlier Detection**: Detect and handle outliers that can skew results.

Data Transformation

- **Encoding Categorical Data**: Convert categorical text data into numeric format.
- **Normalization**: Scaling numerical features to a range of 0 to 1.
- **Standardization**: Scaling data to have a mean of 0 and a standard deviation of 1

Data Reduction

- **Dimensionality Reduction**: Reduce the number of features to simplify models, reduce noise, and improve computational efficiency.
- **Sampling** : Select a representative subset from a large dataset to reduce computational complexity and improve efficiency in analysis, without losing essential information

Feature Engineering: Create new features from existing ones to better represent the underlying problem.

Missing Data

Missing data are a **common problem** in real data, due to multiple reasons: **data sources, instrumental limits and acquisition issues**.

Some Astronomical Case Studies

Data Sources (Cross-matching)

- Issue: Merging disparate catalogs (e.g., matching Infrared data from JWST with Optical data from Hubble).
- Gap: Objects emitting outside a specific instrument's spectral range appear as "missing" in the unified dataset.

Instrumental Limits

- Sensitivity: Faint sources falling below the Noise Floor of the CCD sensor.
- Saturation: Extremely bright stars "blinding" pixels, preventing an accurate numerical readout of the flux

Acquisition Issues

- Environmental: Cloud cover or atmospheric turbulence (poor seeing) interrupting time-series observations
- Hardware: Cosmic ray strikes corrupting image pixels or temporary signal loss during data downlink from deep space

Why it matters for ML

Mathematical Error: Most algorithms cannot process *NaN* or *Null* values.
Selection Bias: Simply deleting missing entries might systematically exclude the most distant or unique objects in the universe.

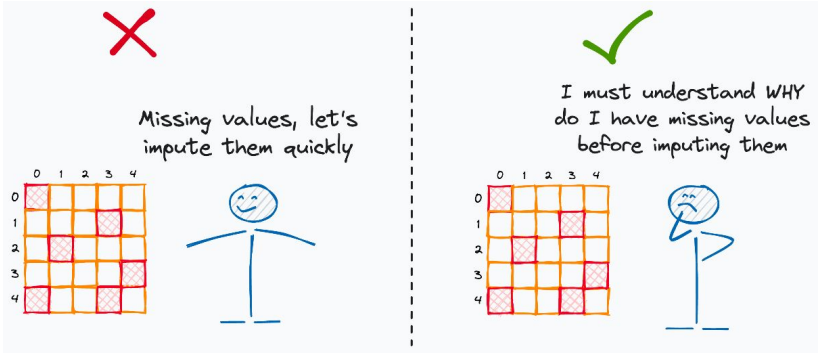
Missing Data - Imputation

The process of replacing missing or inconsistent data with substituted values is defined **Imputation**.

By using statistical techniques, imputation aims to create a complete dataset that preserves the original's integrity, ensuring that algorithms and analyses can function without the bias or information loss caused by simply deleting incomplete records.

Data imputation can be approached in various ways, depending on the **complexity** of the dataset and the **specific nature** of the missing data

Critical Considerations
When choosing an approach, it is essential that the estimated value respects **Instrumental Limits** to avoid introducing physically impossible data into the system. Furthermore, every imputed value should be validated through a **Cross-check** with any reference catalogs to ensure scientific consistency.



Missing Data - Imputation

Simple (Univariate) Imputation
This approach replaces missing values using only the information contained within the variable itself, without considering relationships with other columns.

Mean/Median: Replaces the missing value with the mean (for normal numerical data) or the median (if there are outliers) of the column.

Mode: Primarily used for categorical variables, inserting the most frequently occurring value.

Constant Value: Fills gaps with a predefined value (e.g., "0" or "Unknown").

ORIGINAL DATA

Object ID	u-band (m)	g-band (m)	r-band (m)	i-band (m)	z-band (m)	Redshift (z_spec)	Mass (log M*)	Radius (R_e pc)
GAL-001	14.5	22.0	26.0	18.0	19.0	1.123	4.53	4.0
GAL-002	NaN	22.0	22.0	18.1	20.0	1.001	3.99	6.0
GAL-003	14.5	22.0	18.0	NaN	20.0	NaN	3.76	6.0
GAL-004	14.5	22.0	22.2	18.1	19.0	1.876	NaN	3.0
GAL-005	14.5	22.0	18.0	18.2	21.0	1.050	4.33	2.0
GAL-006	14.5	22.0	12.3	18.1	NaN	1.444	3.34	NaN
GAL-007	14.5	18.0	13.0	15.0	19.0	NaN	3.07	4.5
GAL-008	NaN	22.0	13.0	18.1	18.1	1.999	4.13	10.0
GAL-009	14.5	18.0	NaN	18.1	21.0	1.250	3.75	8.8
GAL-010	14.5	22.0	13.0	19.0	20.0	1.618	3.66	1.0

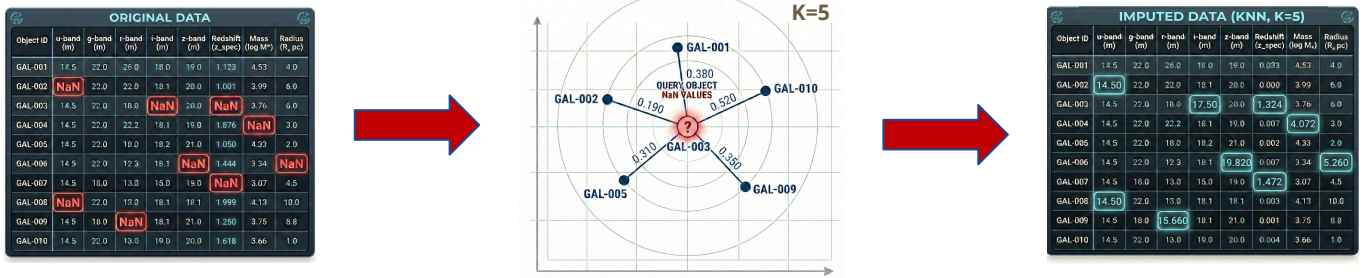
IMPUTED DATA (MEAN REPLACEMENT)

Object ID	u-band (m)	g-band (m)	r-band (m)	i-band (m)	z-band (m)	Redshift (z_spec)	Mass (log M_*)	Radius (R_e pc)
GAL-001	14.5	22.0	26.0	18.0	19.0	1.123	4.53	4.0
GAL-002	14.50	22.0	22.0	18.1	20.0	1.001	3.99	6.0
GAL-003	14.5	22.0	18.0	18.06	20.0	1.420	3.76	6.0
GAL-004	14.5	22.0	22.2	18.1	19.0	1.876	3.82	3.0
GAL-005	14.5	22.0	18.0	18.2	21.0	1.050	4.33	2.0
GAL-006	14.5	22.0	12.3	18.1	19.46	1.444	3.34	5.03
GAL-007	14.5	18.0	13.0	15.0	19.0	1.420	3.07	4.5
GAL-008	14.50	22.0	13.0	18.1	18.1	1.999	4.13	10.0
GAL-009	14.5	18.0	17.50	18.1	21.0	1.250	3.75	8.8
GAL-010	14.5	22.0	13.0	19.0	20.0	1.618	3.66	1.0

Missing Data - Imputation

Multivariate Imputation (Predictive Imputation)
This method is more sophisticated because it models the missing value as a function of the other variables available in the dataset.

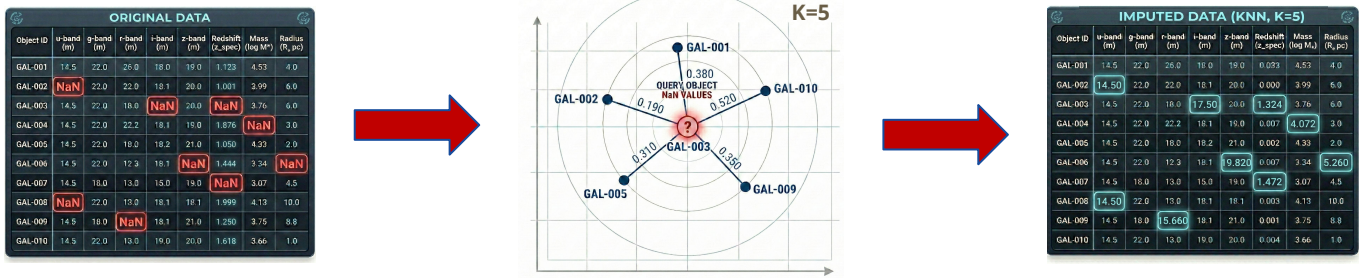
K-Nearest Neighbors (KNN): Identifies the "K" rows most similar to the one with the missing data and calculates a weighted average to fill the gap.



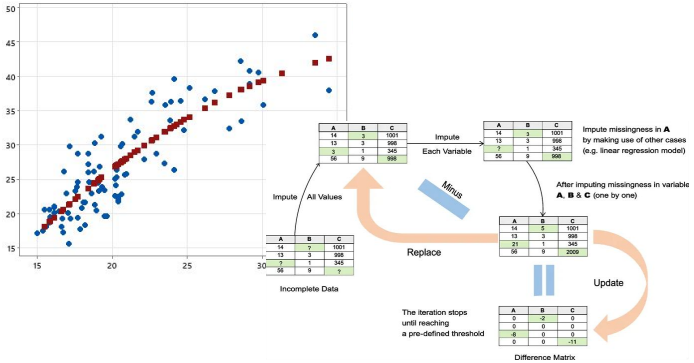
Missing Data - Imputation

Multivariate Imputation (Predictive Imputation)
This method is more sophisticated because it models the missing value as a function of the other variables available in the dataset.

K-Nearest Neighbors (KNN): Identifies the "K" rows most similar to the one with the missing data and calculates a weighted average to fill the gap.



Regression: Uses a mathematical model to predict the missing value based on its correlation with other variables.



MICE (Multivariate Imputation by Chained Equations): An iterative process that models each variable with missing data based on the others, repeating the cycle multiple times to stabilize the estimates.

Missing Data - Imputation

Time-Based Approaches (Time-Series Imputation)
Specific to sequential data or time series where chronological order is fundamental.

Last Observation Carried Forward (LOCF): Uses the last known value before the gap to fill it.

Linear Interpolation: Extrapolates a plausible value by drawing a straight line between the point preceding and the one following the missing data.

Time-Based Approaches (Time-Series Imputation)
Unlike deterministic methods, this approach introduces **random noise** to the substituted value.

- Purpose: It preserves the natural dispersion of the data, ensuring the dataset doesn't become "too perfect" or uniform.
- Relevance: This is crucial when operating near **Instrumental Limits**, where background noise is a real physical factor.

Neural (Generative) Imputation
This modern approach utilizes **Deep Learning** architectures, such as **Autoencoders**, to reconstruct the signal.

- Mechanism: The model learns the complex, non-linear hidden structure of the data to "fill" gaps with high precision.
- Function: It acts as a powerful **Cross-check tool**, ensuring the reconstructed data remains consistent across different volumes or catalogs.

Encoding

Data encoding is the fundamental process of transforming **categorical variables** (text, labels, or categories) into **numerical formats**. Since most Machine Learning algorithms rely on mathematical and matrix operations, they cannot directly process text strings like "Spiral" or "Elliptical".

In essence, encoding acts as a translator between human language and mathematical models.

Some Encoding Examples

- **Label Encoding**: assigns a unique integer to each category (e.g., Red=0, Green=1, Blue=2). It is useful when categories have an intrinsic order, but it can confuse the model into thinking that the value "2" is "greater" than "0" in a quantitative sense.
- **One-Hot Encoding**: creates a new column (dummy variable) for each category, inserting a **1** if the category is present and a **0** otherwise. It is the standard method for nominal data without a specific order, as it avoids introducing artificial numerical hierarchies.
- **Ordinal Encoding**: similar to Label Encoding, but specifically used when there is a logical order between categories (e.g., "Low", "Medium", "High" become 1, 2, 3).
- **Binary Encoding**: first transforms categories into integers and then converts those numbers into binary code. It is very efficient for handling variables with many categories (high cardinality), reducing memory space compared to One-Hot Encoding.

Feature Scaling

Feature scaling is a crucial data preprocessing step in machine learning that transforms independent variables to **a similar range or distribution**, preventing features with larger magnitudes from **dominating** model training.

Why Feature Scaling is Necessary

- **Algorithm Performance:** Algorithms calculating distances (KNN, SVM, K-Means) or using gradient descent (Linear Regression, Neural Networks) require scaled data to function correctly.
- **Preventing Bias:** If features have vastly different ranges (e.g., age 0-100 vs. income 0-1,000,000), models may overemphasize larger values.
- **Faster Convergence:** Gradient-based methods reach optimal solutions faster when features are on similar scales.

Key Scaling Techniques

1. **Normalization (Min-Max Scaling):** Rescales data to a fixed range, usually. This is useful when the distribution is not Gaussian and for algorithms that require input in a bounded interval.
2. **Standardization (Z-score Normalization):** Transforms data to have a mean (μ) of 0 and a standard deviation (σ) of 1. It is ideal for data with Gaussian distributions and is less sensitive to outliers.
3. **Robust Scaling:** Scales data based on percentiles (IQR), making it ideal for datasets with significant outliers

Algorithm Type	Scaling Required?	Recommended Technique	Why
K-Nearest Neighbors	✅ Yes	Standard or Min-Max	Relies on distance calculation
K-Means Clustering	✅ Yes	Standard or Min-Max	Uses Euclidean distance
Support Vector Machines	✅ Yes	Standard	Features should be on same scale
Linear Regression	⚠️ Recommended	Standard	Improves convergence, interpretability
Logistic Regression	⚠️ Recommended	Standard	Helps gradient descent converge faster
Neural Networks	✅ Yes	Min-Max or Standard	Activation functions work best with scaled data
Decision Trees	❌ No	None needed	Makes splits based on feature values
Random Forest	❌ No	None needed	Tree-based, scale-invariant
XGBoost/LightGBM	❌ No	None needed	Tree-based, handles scales naturally
Naive Bayes	❌ No	None needed	Probability-based
Principal Component Analysis	✅ Yes	Standard	Variance-based, needs consistent scale

Python scikit-learn package:
MinMaxScaler, StandardScaler & RobustScaler

Normalization vs Standardization

SCENARIO	METHOD
You need to constrain the data within a specific **range**	**Normalization**
You want to maintain the original unit **interpretation**	**Normalization**
The distribution of the data is not necessarily **Gaussian**	**Normalization**
You need to compare and analyze features with **different units and scales**	**Standardization**
You want to reduce the impact of **outliers**	**Standardization**

$$x_{std}^{(i)} = \frac{x^{(i)} - \mu_x}{\sigma_x}$$

standardization

$$x_{norm}^{(i)} = \frac{x^{(i)} - \mathbf{x}_{min}}{\mathbf{x}_{max} - \mathbf{x}_{min}}$$

min-max scaling ("normalization")

	input	standardized	normalized
0	0	-1.46385	0.0
1	1	-0.87831	0.2
2	2	-0.29277	0.4
3	3	0.29277	0.6
4	4	0.87831	0.8
5	5	1.46385	1.0

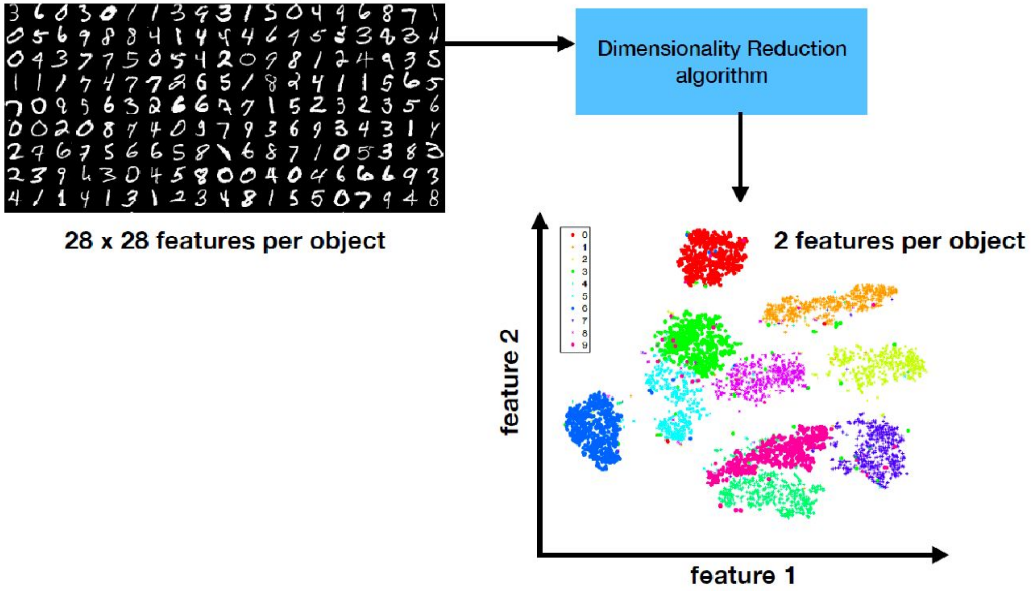
	NORMALIZATION	STANDARDIZATION
PROS	- Fairness and Balance - Improved Comparability	- Improved Interpretability - Enhanced Comparability - Normal Distribution Assumption - Robustness to Outliers
CONS	- Loss of Original Value Interpretation - Sensitivity to Outliers	- Loss of Original Unit Interpretation - Dependency on Normality Assumption

Dimensionality Reduction

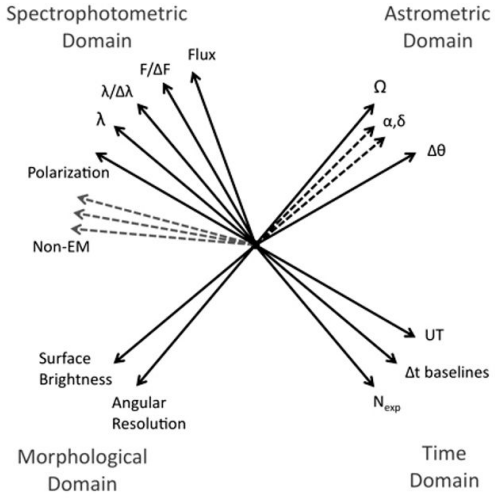
Dimensionality reduction is a machine learning technique that **reduces** the number of input variables (features) in a dataset while retaining essential information

Why do we need Dimensionality Reduction

- **“Practical”:**
  - Improve performance of ML algorithms: original features can be correlated and redundant, most algorithms cannot handle thousands of features
  - Compressing data (e.g, SKA)
- **“Artistic”:**
  - Data visualization and interpretation
  - Uncover complex trends
  - Look for “unknown unknowns”



Parameter Space Exploration (PSE)



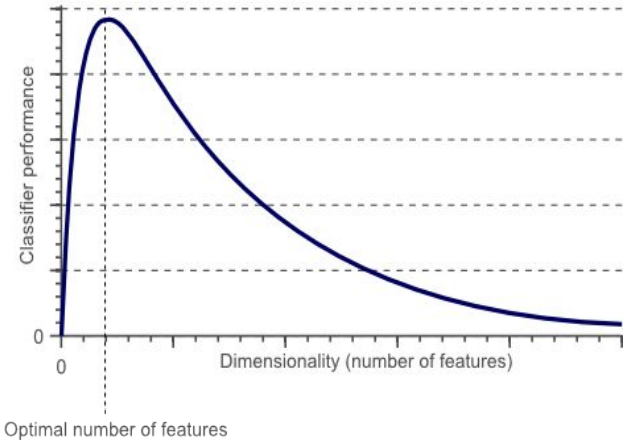
By definition, ML is based on self-adaptive techniques. A priori, real world data are intrinsically carriers of embedded information, hidden by noise.

In almost all cases the signal-to-noise (S/N) ratio is low and the amount of data prevents the human exploration. We don't know which feature of a pattern is much more carrier of useful information and how and where the correlation of features gives the best knowledge of the solution.

Curse of dimensionality

The larger the size of Parameter Space (PS), the higher the distance among data, thus decreasing the correlation among them within the PS.

More features ⇒ Larger PS ⇒ Data Sparseness ⇒
⇒ Less point similarities ⇒ Less efficient predictive models



Dimensionality Reduction Techniques

Linear Techniques

Assumption that the data resides, or can be approximated, in a linear subspace of the original space.

Principal Component Analysis (PCA)

PCA is a classic technique that **projects** data into a new coordinate system constructed to maximize variance along the first components.

Through a spectral analysis of the covariance matrix, PCA identifies the orthogonal directions along which the data exhibits greater variability, thus reducing dimensionality **without significantly losing overall information**.

This linear projection **preserves the global relationships** between data points and allows for easy interpretation of the results, but is limited in the presence of nonlinear structures.

Non Linear Techniques

Designed to process data that lie on curved and more complex manifolds, attempting to preserve local or global relationships that are not necessarily linear.

T-distributed stochastic neighbor embedding (t-SNE)

t-SNE does not rely on linear transformations, but constructs a low-dimensional representation while attempting to **preserve the local structure of the data**: nearby points in high-dimensional space must remain close together even in the reduced representation.

The result is a representation that, while losing precision in global distances and interpretability of axes, is extremely effective at **revealing groups, clusters, and local structures**, making it particularly useful for the visual exploration of complex data.

Principal Component Analysis (PCA)

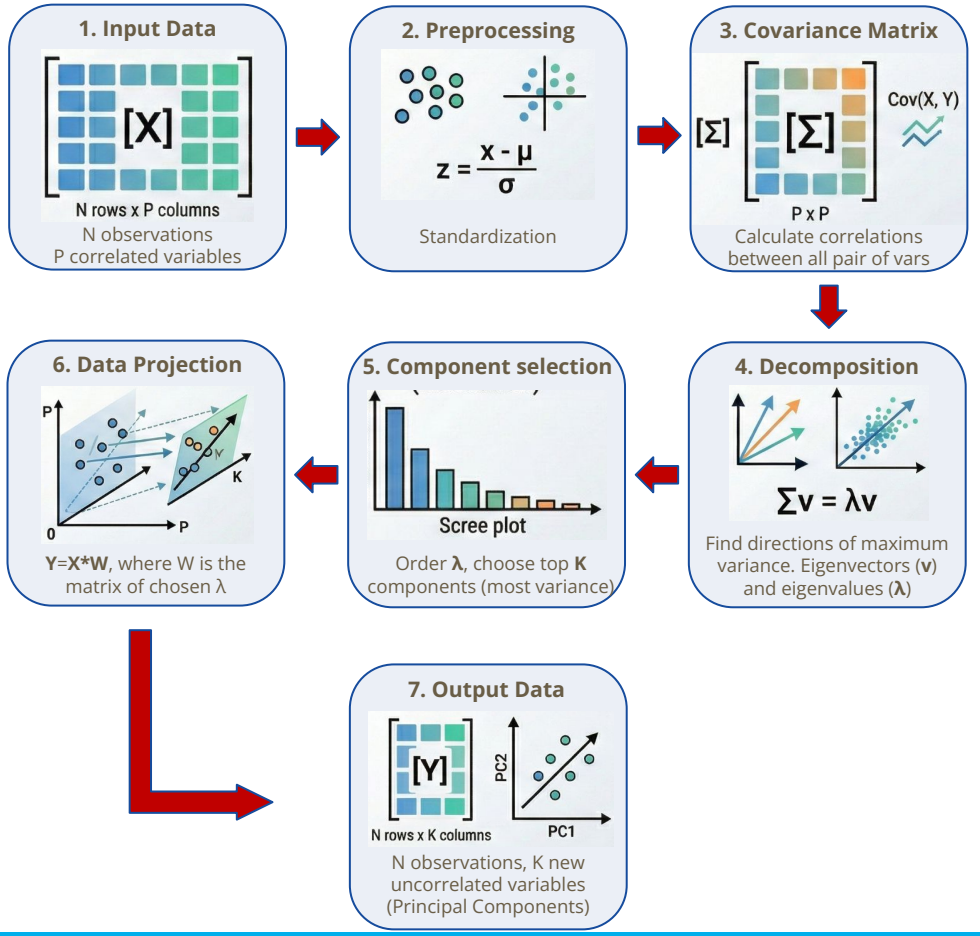
The main goal of PCA is to reduce the number of variables in a data set while **preserving as much information as possible.**

PROS

- It removes correlated features, thus correcting for the phenomenon known as **multicollinearity**
- It improves the **performance** of machine learning algorithms
- It reduces the so-called **overfitting**, a phenomenon in which a model learns the training data too well, including noise and irrelevant details, losing the ability to **generalize** to new data

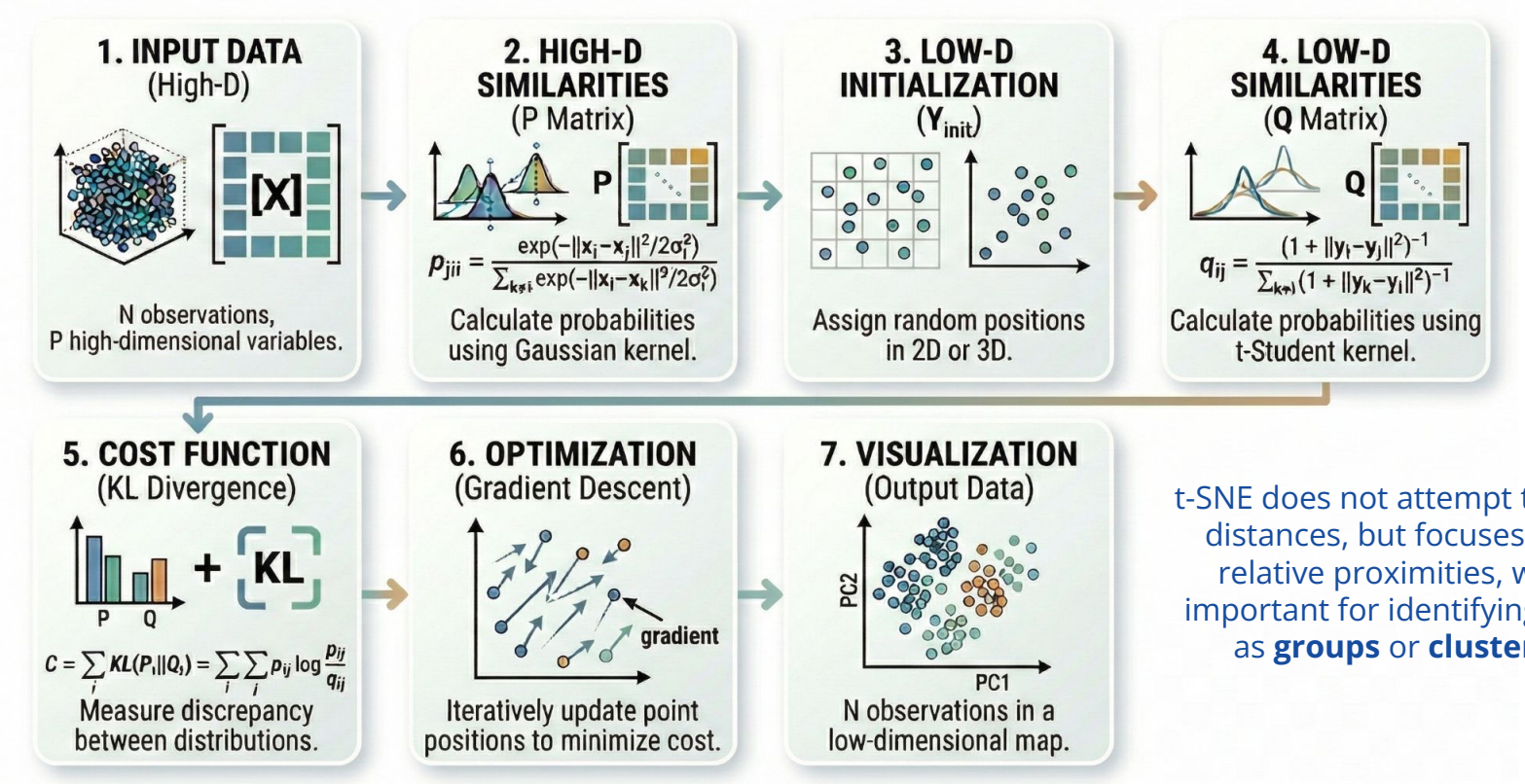
CONS

- The independent variables are **less interpretable**, because the PCA components are a linear combination of the original features
- It can cause **information loss** if you do not pay attention to the component number
- Since PCA is an algorithm that aims to maximize variance, it requires **scaling** the features a priori.



T-distributed stochastic neighbor embedding (t-SNE)

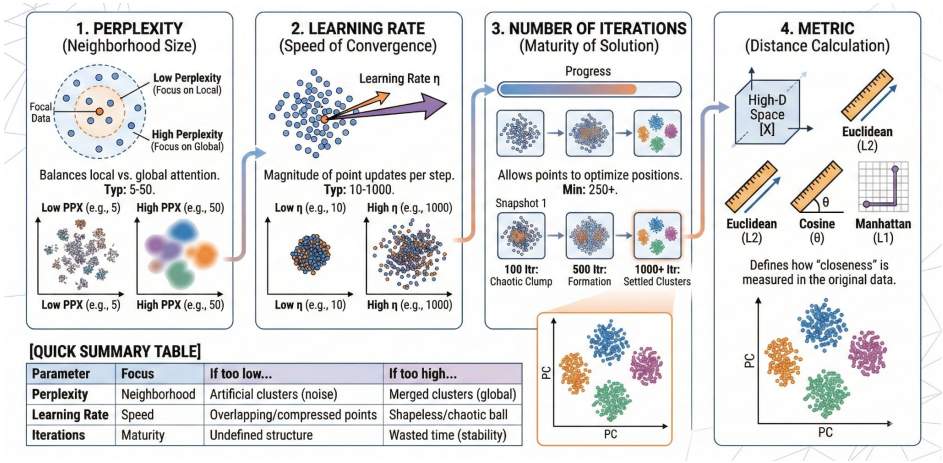
The goal of t-SNE is to preserve **local relationships** between data: if two points are close together in high-dimensional space, they should remain close together in low-dimensional space as well.



t-SNE does not attempt to preserve global distances, but focuses on maintaining relative proximities, which are more important for identifying structures such as **groups** or **clusters** in the data.

t-SNE critical parameters

- **Perplexity**: This is the most critical parameter. It defines the number of "neighbors" each point considers when constructing the map.
  - What it does: It balances the algorithm's attention between the local and global structure of the data.
  - Typical values: Usually between 5 and 50.
  - Effect: A perplexity that is too low generates small, dispersed clusters (even where they don't exist); one that is too high can merge distinct clusters into a single mass.
- **Learning Rate**: This determines the magnitude of the points' movement in the low-dimensional space during each optimization iteration (Gradient Descent).
  - What it does: If it is too high, the points "bounce" around, and the data appears as a dense, shapeless ball. If it is too low, the points remain compressed and fail to separate correctly.
  - Typical values: Often between 10 and 1000. Many modern implementations (like scikit-learn) use an automatic value based on the number of samples.
- **Number of Iterations**: t-SNE is an iterative process. It needs time to allow the points to "settle" into their optimal positions.
  - What it does: It defines how many times the algorithm updates the positions.
  - Minimum value: Generally at least 250–1000 iterations. If you see a "donut" shape or points that are still very contracted, you probably need more iterations.
- **Metric (Distance Metric)**: This defines how "closeness" is calculated between points in the original dataset.
  - Options: The most common is Euclidean, but for specific data (like text or genomic data), you might use Cosine or Manhattan distance.



[QUICK SUMMARY TABLE]

Parameter	Focus	If too low...	If too high...
Perplexity	Neighborhood	Artificial clusters (noise)	Merged clusters (global)
Learning Rate	Speed	Overlapping/compressed points	Shapeless/chaotic ball
Iterations	Maturity	Undefined structure	Wasted time (stability)

Data Sampling

Data sampling in machine learning is the process of **selecting a representative subset** from a larger dataset to train models efficiently, **reducing computational costs while maintaining accuracy**. It solves issues with large datasets, speeds up training, and helps balance imbalanced data

Why Data Sampling is Necessary

- **Efficiency:** Working with a subset reduces memory usage and speeds up training time for large datasets.
- **Balancing Classes:** In tasks with unequal class distributions, sampling ensures the model doesn't ignore the minority class.
- **Data Representation:** Proper sampling ensures the subset maintains the statistical characteristics of the original, large dataset

Which one is the best?

Obviously, There is no **universally best method**: it depends on the **context**.

Stratified Sampling is often preferable to ensure representativeness.

Key Scaling Techniques

1. **Simple Random Sampling:** Every data point has an equal chance of being selected, ideal for reducing bias in large, homogeneous datasets.
2. **Stratified Sampling:** The population is divided into subgroups (strata) based on shared characteristics, and samples are drawn from each to ensure representation.
3. **Systematic Sampling:** Every *n*-th element is selected from an ordered list, which is useful for maintaining order while sampling.
4. **Cluster Sampling:** The dataset is divided into clusters (e.g., geographically), and entire clusters are randomly selected.
5. **Oversampling/Undersampling** Used for imbalanced data to either increase the minority class instances or decrease the majority class instances.

Feature Engineering

We can think of feature engineering as the broader creative and strategic process that aims to **improve the quality of data** fed to a machine learning model.

Its main goal is to transform raw data into features that better represent the underlying problem and make it easier for algorithms to learn.

Why it is important

1. **Improve model performance:** Well-engineered features can reveal hidden relationships in the data and make it easier for the algorithm to learn and make accurate predictions.
2. **Reduces complexity**: Creating more meaningful features can allow for simpler, more interpretable models.
3. **Handles nonlinear data**: Through appropriate transformations, nonlinear relationships in the data can be made linear, making it easier for some algorithms to learn.
4. **Provides insight into the problem**: The feature creation process often leads to a better understanding of the data and the problem you are trying to solve.

It is not just about applying algorithms, but it requires domain **knowledge**, **creativity** and **experimentation**.

Feature Selection vs Extraction

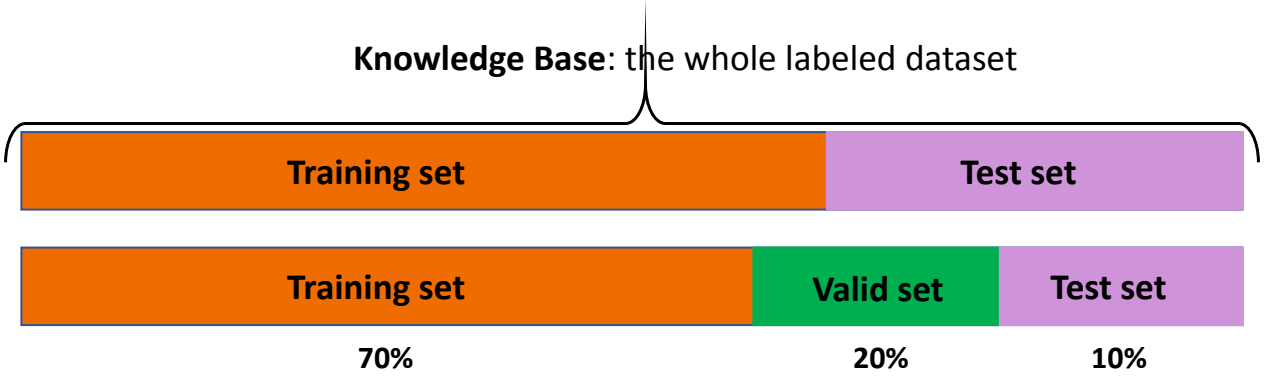
Feature	Selection	Extraction
What it does	Chooses informative features from existing ones	Creates new features from existing ones
Analogy	Picking key ingredients	Combining ingredients in a new way
Benefits	Faster training, interpretability, avoids overfitting	Captures hidden patterns, improves accuracy
Challenges	Choosing selection method, discarding information	Designing methods, increases dimensionality

An example in astronomy

Feature Extraction: adding **colours** to the dataset by combining magnitudes

Feature Selection: choose between **colours** and **magnitudes**, both present into the dataset

Dataset Partitioning



Training Set: used to train the model, i.e. to estimate the parameters. During this phase, the network parameters (weights and biases) are adapted according to the loss function

Validation Set: this set can be used for two, mutually excluding, reasons:

i) at the end of the training, to choose the best hyper-parameter setup, to select the best set of features;

ii) during the training, to update some hyper-parameters or to apply regularization techniques.

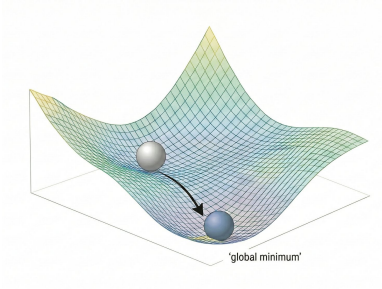
Test Set: to measure the performance of the network independently, and to investigate if the network achieved a good level of generalization.

Training

The training phase is the "learning" engine of machine learning. It is the iterative process where an algorithm analyzes data to find patterns, adjusting its internal parameters to map inputs to the correct outputs.

The goal is to optimize internal parameters (weights and biases), in order to minimize a **loss function**, which measures the discrepancy between the model's predictions and the actual values.

The model parameters are **optimized** using numerical optimization algorithms, such as **gradient descent** or its more efficient variants (e.g. **stochastic gradient descent** or **Adam**).



Gradient descent is a first-order iterative optimization algorithm used to find the **local minimum** of a differentiable cost function to minimize error. By calculating the **negative gradient** of the function, the algorithm determines the steepest downhill direction to update parameters (weights and biases), with step sizes controlled by a **learning rate** (α)

Most common loss functions

Regression:

Mean Squared Error $$MSE(y, \bar{y}) = \frac{1}{N}\sum_{i=1}^{N}(y(\mathbf{x}_i) - \bar{y}(\mathbf{x}_i))^2$$

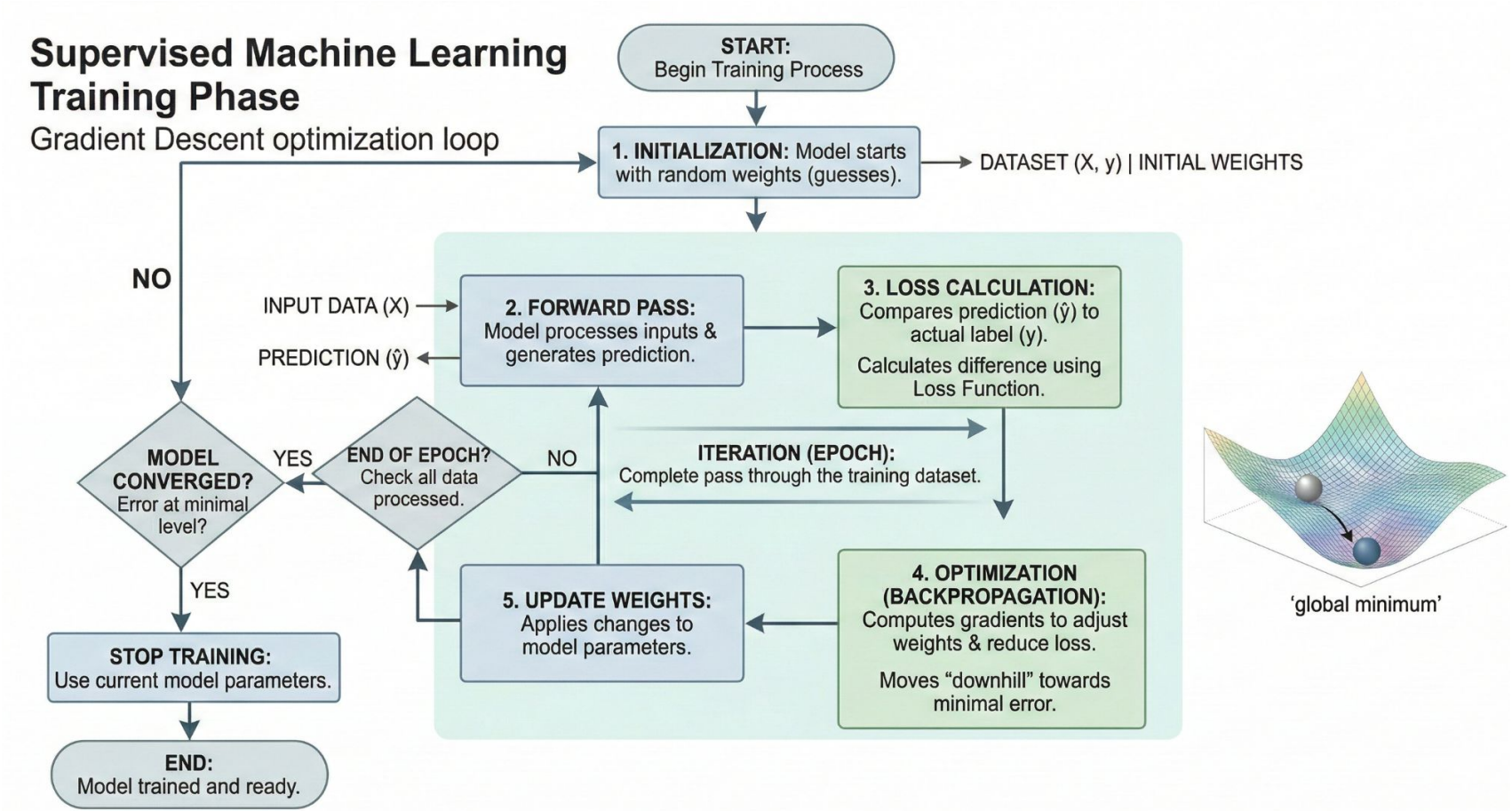
Mean Absolute Error $$MAE(y, \bar{y}) = \frac{1}{N}\sum_{i=1}^{N}|y(\mathbf{x}_i) - \bar{y}(\mathbf{x}_i)|$$

Classification: (Binary) Cross-Entropy

2 classes $$\mathcal{H}(y, \bar{y}) = -\frac{1}{N}\sum_{i=1}^{N}\bar{y}(\mathbf{x}_i)\cdot\log(y(\mathbf{x}_i)) + (1-\bar{y}(\mathbf{x}_i))\cdot\log(1-y(\mathbf{x}_i))$$

M classes $$\mathcal{H}(y, \bar{y}) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M}\bar{y}_j(\mathbf{x}_i)\cdot\log(y_j(\mathbf{x}_i))$$

Training


Supervised Machine Learning
Training Phase
Gradient Descent optimization loop
START:
Begin Training Process
1. INITIALIZATION: Model starts with random weights (guesses).
DATASET (X, y) | INITIAL WEIGHTS
NO
INPUT DATA (X)
PREDICTION (ŷ)
2. FORWARD PASS:
Model processes inputs & generates prediction.
3. LOSS CALCULATION:
Compares prediction (ŷ) to actual label (y).
Calculates difference using Loss Function.
MODEL CONVERGED?
Error at minimal level?
YES
END OF EPOCH?
Check all data processed.
NO
ITERATION (EPOCH):
Complete pass through the training dataset.
YES
5. UPDATE WEIGHTS:
Applies changes to model parameters.
4. OPTIMIZATION (BACKPROPAGATION):
Computes gradients to adjust weights & reduce loss.
Moves "downhill" towards minimal error.
STOP TRAINING:
Use current model parameters.
END:
Model trained and ready.
'global minimum'

Validation

The trained model is used to predict responses for observations in a **second dataset** to evaluate its performance and refine the hyperparameters, aiming to find the optimal balance: ensuring the model **is complex enough to capture the underlying patterns (avoiding underfitting)** while preventing it from becoming **overly sensitive to training-specific fluctuations (overfitting)**

Bias-Variance Tradeoff

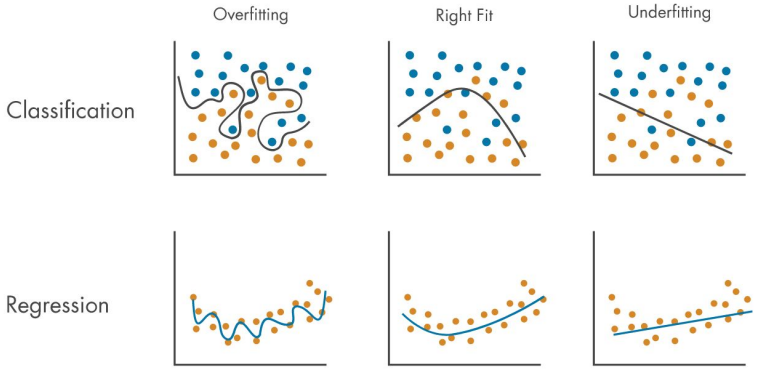
- **Bias** : represents the prediction error of the model due to its simplifications

  High bias (rigid model) → **underfitting**

- **Variance :** represents the prediction error of the model due to its sensitivity to the training data.

  High variance (too complex), the model "follows" every single data point → **overfitting**

1. Evaluate model complexity
2. Increase dataset size
3. Regularization
4. Cross-Validation
5. Feature Selection

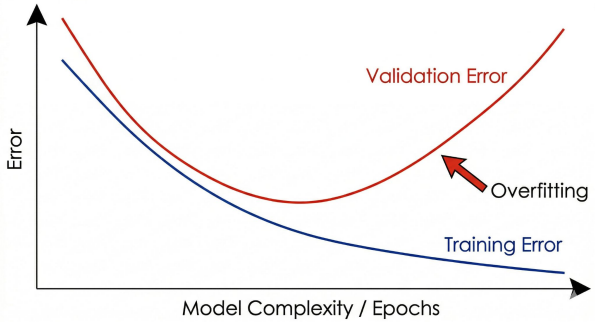


Regularization

In simple terms, regularizing a model means **changing its learning behavior** during the training phase.

Most used approaches

Early stopping
Stop training when performance on validation set stops improving



L1 and L2 Regularization
Add a penalty term (L1 or L2) to the loss function to simplify the model

L1 Regularization

$$\text{Cost} = \sum_{i=0}^{N}(y_i - \sum_{j=0}^{M} x_{ij}W_j)^2 + \lambda \sum_{j=0}^{M} |W_j|$$

L2 Regularization

$$\text{Cost} = \sum_{i=0}^{N}(y_i - \sum_{j=0}^{M} x_{ij}W_j)^2 + \lambda \sum_{j=0}^{M} W_j^2$$

Loss function

Regularization Term

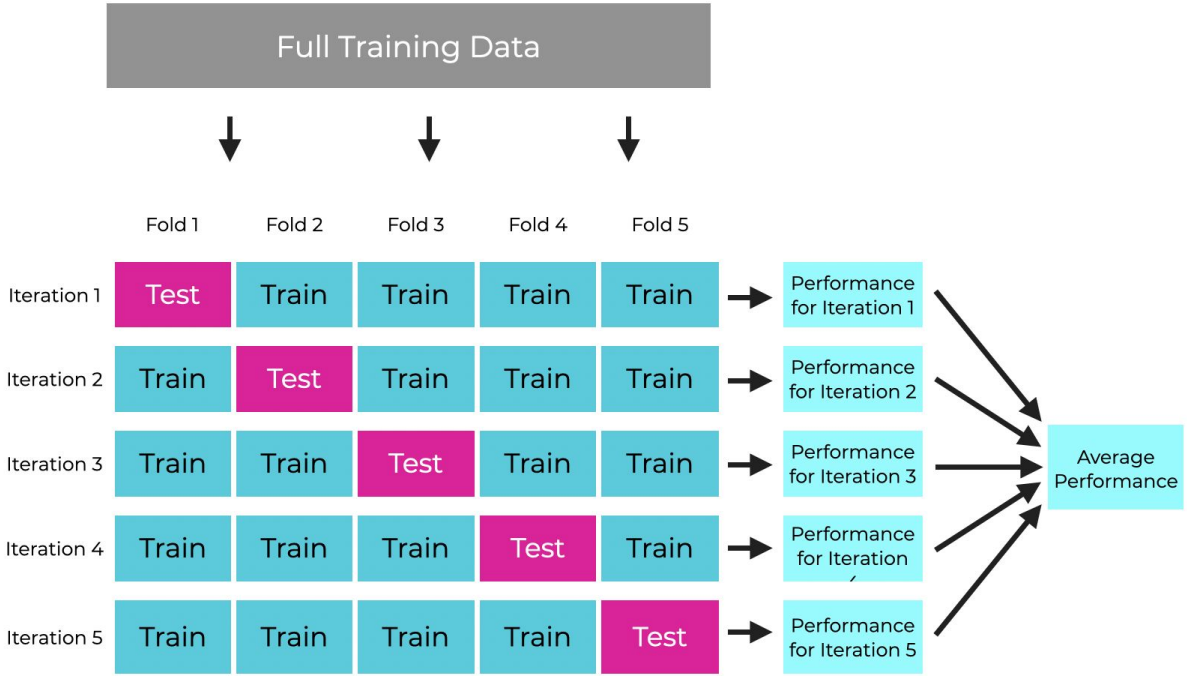
For many other approaches, see **Goodfellow +2016**

Cross-validation

A model can and must be evaluated **several times with different configurations** to understand which is the best performing condition. When the dataset has **limited size**, the cross-validation is used to fulfill the role of the validation set.

The most common is **k-fold cross-validation**: it involves splitting the training dataset into **multiple parts** (or "**folds**") of similar size and training k times, each time using k-1 folds for training and 1 fold for validation. Finally, the performance is **averaged** over the k experiments.

This approach allows for a **more robust** evaluation of the model's performance, reducing the risk that the choice of a single validation set will influence the result too much.



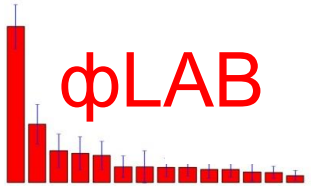
Feature Selection

We have already encountered the feature selection when discussing data reduction during the pre-processing phase. The same approach can be also applied during the validation phase to discard features with less informative value

Some ML algorithms (for example, Random Forest) provide "*for free*" an estimation of **feature importance**, i.e. a measure of the contribution of a feature to the ML task.

Other tools analyze what a ML model have learnt and assign an importance to each involved feature:

- **PHiLAB** (**P**arameter **H**andling **i**nvestigation **LAB**oratory) *Brescia et al. 2019, Angora et al. 2019, Delli Veneri et al. 2019*

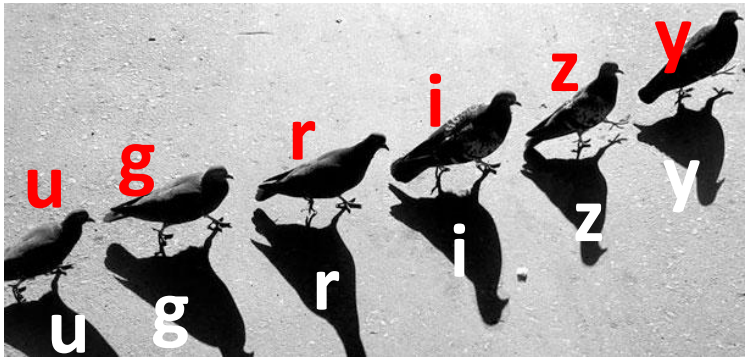


- **SHAP** (**SH**apley **A**dditive ex**P**lanations) *Lundberg and Lee 2017, NIPS 2017 Conference, USA*

PHiLab - Parameter Handling investigation LABoratory

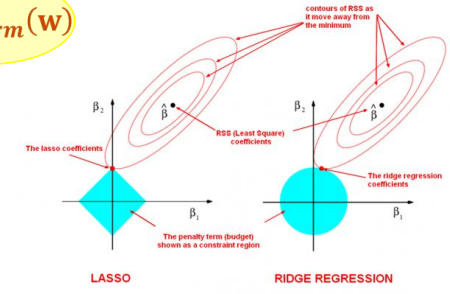
t is a hybrid model based on the combination of two techniques: **Boruta** and **LASSO regularization** by exploiting **Random Forest model** as feature relevance computing engine.

Boruta shadow features as noise threshold



LASSO regression to shrink features

$$min_f \sum_{i=1}^{n} L(f(x)) + \lambda \mathbf{L}_{1-norm}(\mathbf{w})$$

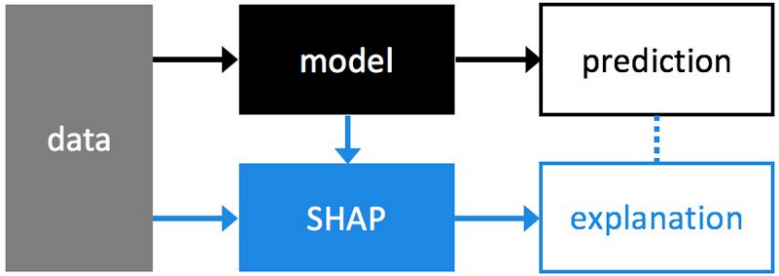


- Rejected features by Boruta are **definitely removed** from dataset;
- High relevant features found by Boruta are **definitely kept** in the dataset;
- The features classified as «weak relevant» by Boruta (if any) **are passed** to Lasso regression for a further importance evaluation and final keeping/rejecting decision

SHAP - SHapley Additive exPlanations

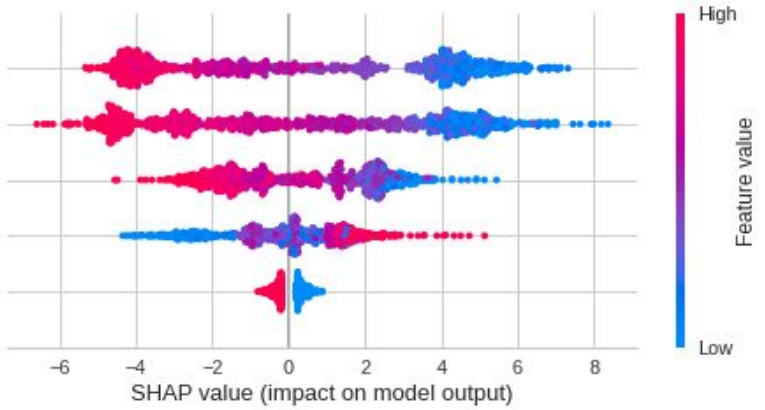
SHAP is a method based on cooperative game theory (**Shapley values**) used to explain the output of any machine learning model. In the context of feature selection, SHAP not only explains whether a variable is important, but also **how** and **to what extent** each variable contributes to the prediction's deviation from the mean.

Shapley's values calculate the importance of a feature by comparing what a model predicts **with and without this feature**. However, since the order in which a model sees the features can affect its predictions, this is done **in all possible ways**, so that the features are compared fairly.



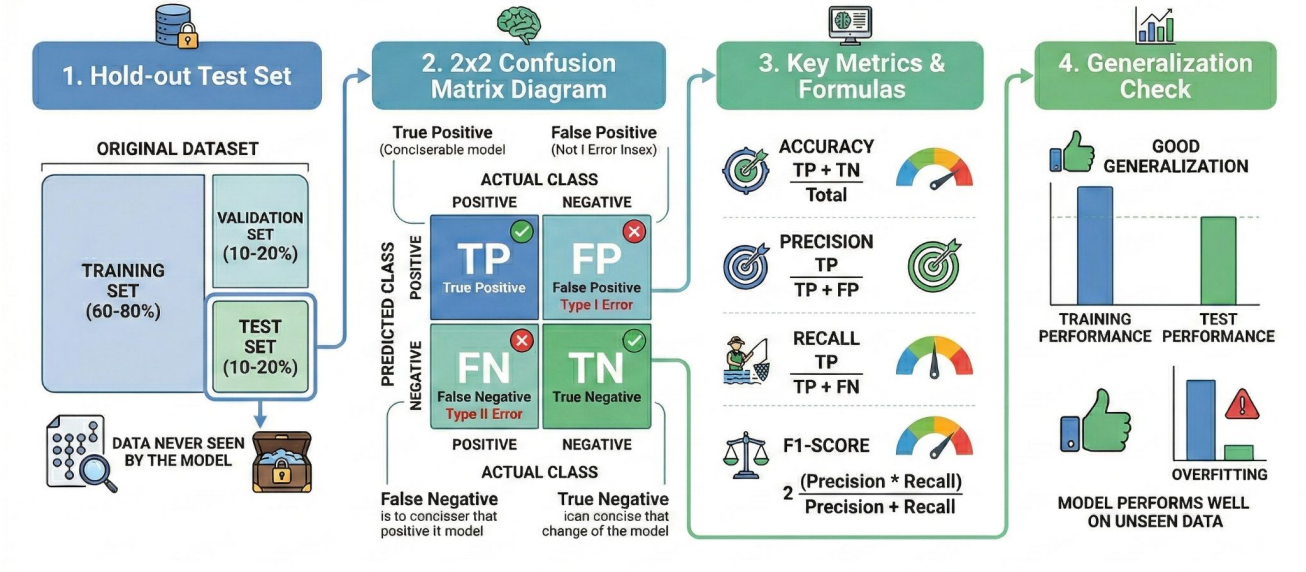
How does Beeswarm plot help prevent overfitting?

- **Randomly impacted features** : Red and blue points randomly mixed both to the right and left of zero indicates that feature has **no clear relationship** with the output. It's likely just adding variance (noise) to the model.
- **Dominant outliers** : If you see only a few isolated points far from zero, the model may have learned a **specific rule** from very little data, a classic sign of overfitting.



Test

The testing phase represents the **moment of truth** in a machine learning experiment. After training the model and optimizing its parameters, testing serves to evaluate how the system performs on **"real" data** it has **never seen before**.



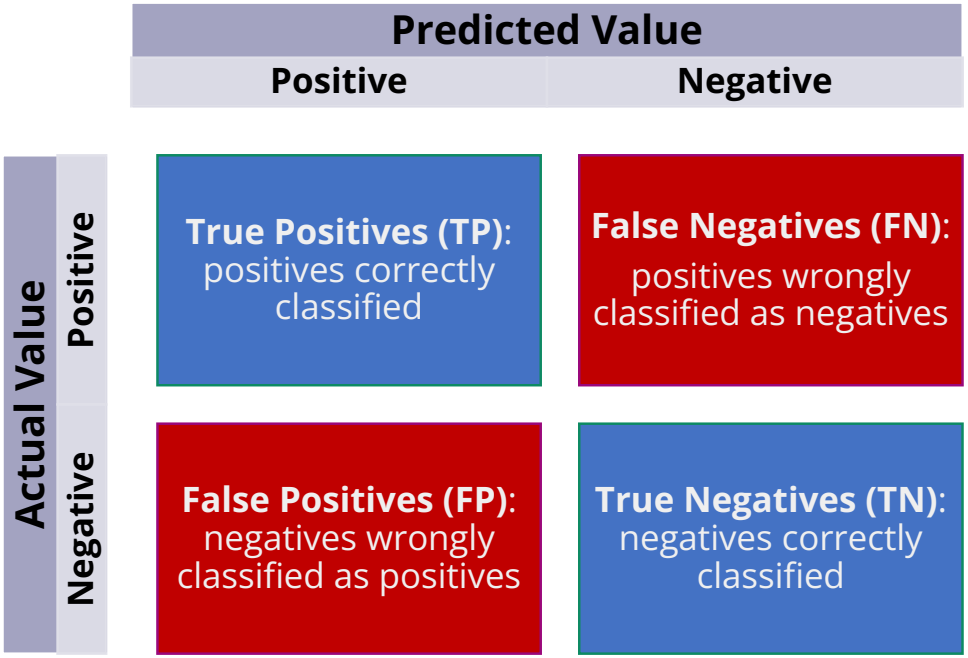
Performance evaluation tools
(Classification)

- Confusion Matrix
  - Accuracy
  - Precision
  - Recall
  - F1-Score
  - ROC curve
    - AUC

The ability of the model to maintain good performance on data never seen before is what we call **generalization**.

Metrics : Classification

Confusion matrix



The confusion matrix allows you to visualize not only **how accurate** a model is, but also **the nature of the errors** it makes, distinguishing between false positives and false negatives.

This is especially important in applications where the consequences of errors are **different**.

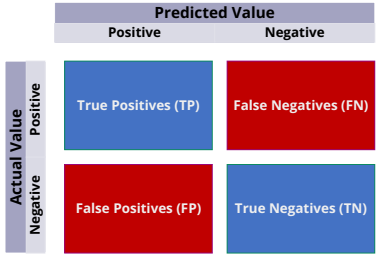
For example, in a medical context, a false negative (failing to recognize a disease) can be more serious than a false positive (diagnosing a disease when it doesn't exist).

Accuracy

$$AE = \frac{TP + TN}{TP + TN + FP + FN}$$

Average score for all the included classes

It provides a general idea of the model's accuracy. However, accuracy can be misleading, especially with **unbalanced** datasets where one class dominates.



Purity/Precision

$$pur = \frac{TP}{TP + FP} = \frac{TP}{P}$$

"How many of all the cases predicted as positive by the model actually turned out to be positive?"

It's related to the **contamination rate**, how pure is the sample selected by the network

It is important when trying to reduce FP as much as possible

Completeness/Recall

$$compl = \frac{TP}{TP + FN}$$

It is the **True Positive Rate**, that is, the proportion of TP detected compared to all actual positive cases.

"Of all the galaxies actually present in space (my sample), how many did my model find?"

F1-score

$$F1 = 2 \cdot \frac{pur \cdot compl}{pur + compl}$$

Combine precision and recall into a single metric to balance the trade-off (harmonic mean between purity and completeness)

It provides a better sense of a model's overall performance, especially for unbalanced datasets.

"What is the overall balance of my model between the ability to avoid false alarms (Precision) and the ability to avoid missing anything (Recall)?"

Example : 100 objects, 99 star & 1 galaxy, galaxy classified as star.

How good is the model?

Metric	Formula	Value	Meaning
Accuracy	$AE = \frac{TP + TN}{TP + TN + FP + FN}$	**99%**	**Deceptive**. It seems like a success...
Completeness / Recall	$compl = \frac{TP}{TP + FN}$		
Precision / Purity	$pur = \frac{TP}{TP + FP} = \frac{TP}{P}$		
F1-Score	$F1 = 2 \cdot \frac{pur \cdot compl}{pur + compl}$		

Example : 100 objects, 99 star & 1 galaxy, galaxy classified as star.

How good is the model?

Metric	Formula	Value	Meaning
Accuracy	$AE = \frac{TP + TN}{TP + TN + FP + FN}$	**99%**	**Deceptive**. It seems like a success...
Completeness / Recall	$compl = \frac{TP}{TP + FN}$	**0%**	**Total failure**. You didn't catch the object you were looking for.
Precision / Purity	$pur = \frac{TP}{TP + FP} = \frac{TP}{P}$	**Undefined**	0/0. Since you didn't predict any galaxies, we don't know how accurate you would be.
F1-Score	$F1 = 2 \cdot \frac{pur \cdot compl}{pur + compl}$	**0**	Harmonic mean that collapses because the recall is zero.

Metrics : Classification

$$\text{Recall} = \frac{TP}{TP+FN} \qquad \text{False Positive Rate} = \frac{FP}{FP+TN}$$

ROC (Receiver Operating Characteristic) curve

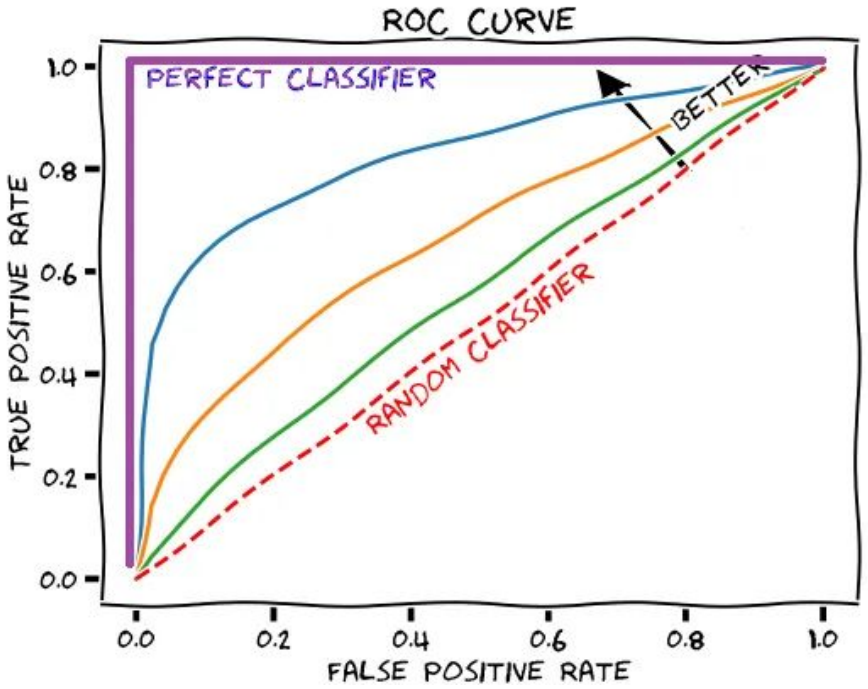
it is a diagram where the **True Positive Rate** (i.e. the recall/completeness) and the **False Positive Rate** (i.e. the contamination, 1 – purity) are plotted by varying a probability threshold.

The ROC curve shows how the recall and false positive rate vary as the model's decision threshold changes.

Ideally, a good model tends to have a curve that approaches the upper left corner of the graph (high recall, low false positive rate).

Area Under the (ROC) Curve : number between 0 and 1 which quantifies the area under the ROC curve, indicating the model's ability to discriminate between positive and negative classes.

AUC = 1 → Perfect model
AUC = 0 → Coin toss

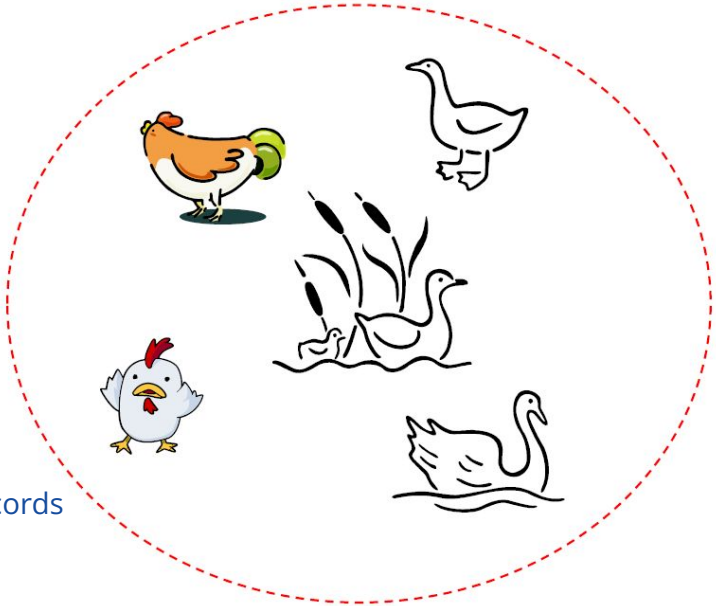


Metrics : Regression

Unlike classification problems, where labels are discrete, regression model evaluation focuses on the **distance** between model predictions and actual values.

Metric	Formula	Description	When to use
MAE **Mean Absolute Error**	$MAE = \frac{\sum_{i=1}^{n} \lvert(y_i - \hat{y}_i)\rvert}{n}$	Average of the absolute differences between the predicted values and the actual observation values.	All individual differences are weighted equally in the average → Low sensitivity to outliers
MSE **Mean Squared Error**	$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$	Measures the average of the squares of the errors.	It heavily penalizes large errors → High sensitivity to outliers
RMSE **Root Mean Squared Error**	$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$	The square root of MSE, bringing the units back to the original scale	Use when you want to penalize large errors but still need the result to be in the same unit as the target.
R^2 score **Coefficient of Determination**	$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$	Represents the proportion of variance for a dependent variable explained by the model.	Use to understand how well the model fits the data compared to a simple mean baseline.
MAPE **Mean Absolute Percentage Error**	$MAPE = \frac{\sum_{i=1}^{n}\left\lvert\frac{y_i - \hat{y}_i}{y_i}\right\rvert}{n} \times 100\%$	Measures the average of the absolute percentage errors for each prediction	It tells you, on average, how far off your predictions are in terms of a percentage of the actual values

KNN - K-Nearest Neighbours

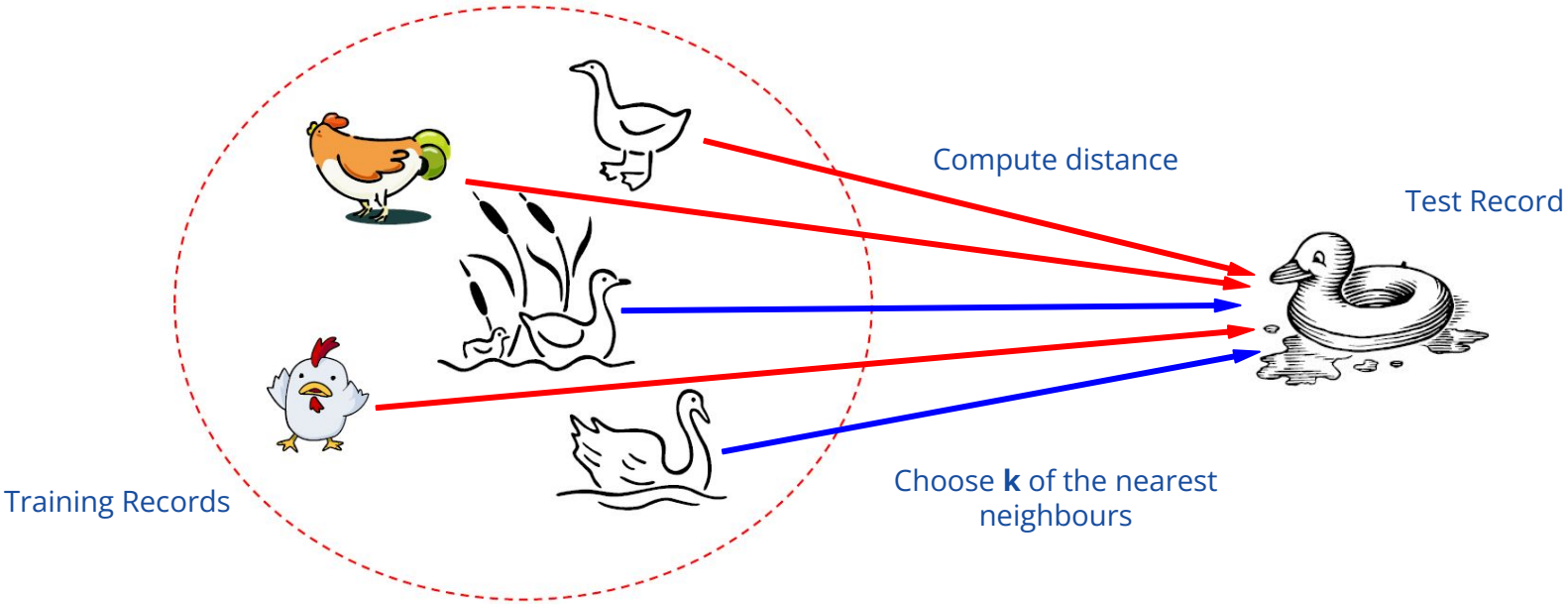


Training Records

Test Record

KNN - K-Nearest Neighbours

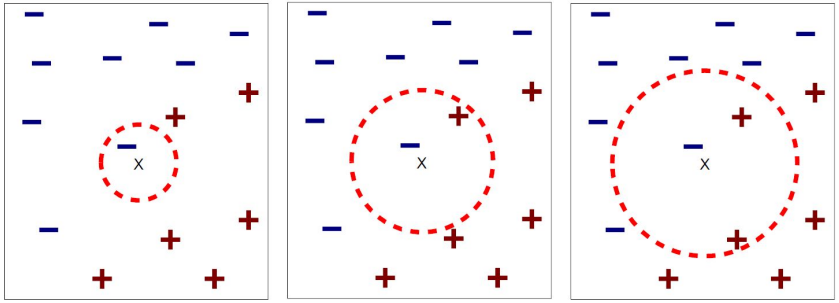
Basic Idea : If it walks like a duck, quacks like a duck, then it's probably a duck



KNN - K-Nearest Neighbours

Requirements

- The set of **stored records**
- The value of **k**, the number of nearest neighbors to retrieve
- **Distance Metric** to compute distance between records

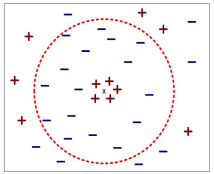


(a) 1-nearest neighbor (b) 2-nearest neighbor (c) 3-nearest neighbor

KNN - Configuration

Parameter Choice

- If k is too small, sensitive to noise points (**overfitting**)
- If k is too large, neighborhood may include points from other classes (**underfitting**)

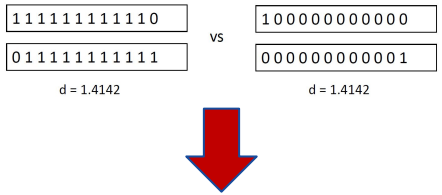


Distance Calculation

- Euclidean Distance

$$d(p,q) = \sqrt{\sum_i (p_i - q_i)^2}$$

- Can be used to weigh the majority vote in classification ($w=1/d^2$)
- High dimensional data → curse of dimensionality
- Can produce counter intuitive results



Normalization the vectors to unit length

KNN - Pro & Cons

- **Algorithmic Transparency and Simplicity**: k-NN is a non-parametric, instance-based learning method that relies on the intuitive principle of feature-space proximity.
- **Lazy Learning Paradigm**: Since the algorithm does not construct an explicit internal model during the training phase is very fast. This allows for immediate integration of new observations.
- **Native Multiclass Support**: The architecture inherently facilitates multiclass classification problems without the need for specialized decomposition strategies.
- **Local Decision Boundaries**: k-NN is particularly effective when the data exhibits high local density

- **Hyperparameter Selection k**: The choice of the **k** parameter is critical for the model's bias-variance tradeoff.
- **Mandatory Feature Scaling**: The distance metric is highly sensitive to the magnitude of the features. Normalization or Standardization are required, features with larger numerical ranges will disproportionately influence the distance calculation, leading to biased results.
- **Memory Constraints**: Although the training phase is instantaneous, the algorithm must retain the entire reference dataset in RAM to perform real-time comparisons. This poses significant hardware scalability challenges for large-scale datasets.

- **Computational Latency at Inference**: The primary bottleneck is the high complexity during the prediction phase. As the cardinality of the dataset increases, the time required to compute distances to every training instance becomes prohibitive for real-time applications.
- **The Curse of Dimensionality:** In high-dimensional feature spaces, the volume of the space increases exponentially, causing data points to become sparsely distributed.
- **Sensitivity to Class Imbalance:** With unbalanced datasets, the majority vote mechanism tends to favor the dominant class. Without implementing techniques such as distance-weighting or synthetic oversampling, the model will consistently misclassify rare but significant minority instances.

KNN and Outliers

Sensitivity to Outliers and Stochastic Noise

Susceptibility to Local Anomalies: k-NN is highly sensitive to the presence of outliers, particularly when the hyperparameter **k** is small. A single mislabeled or anomalous data point (an outlier) can create a "wrong" decision island in the feature space. This is because the algorithm lacks a global model to "smooth out" these local irregularities.

Impact on Distance Metrics: Outliers with extreme values in one or more dimensions can significantly distort the distance calculations. In Euclidean space, the distance is heavily influenced by large differences in a single coordinate, meaning an outlier can "pull" the classification of a query point toward itself, even if it is not representative of the local neighborhood.

The "Voter Contamination" Effect: Even with a larger **k**, a cluster of outliers can contaminate the "neighborhood" of a query point. If the ratio of outliers to legitimate neighbors is high within the radius of **k**, the majority vote will be compromised, leading to a localized failure of the model's predictive power.

How to "defend" k-NN from Outliers?

Increasing k: Shifting from a low **k** to a higher value (e.g., 10–15) creates a "filtering effect" that shields the model from isolated anomalies. By expanding the neighborhood, the influence of a single outlier is diluted by the majority vote of more representative data points.

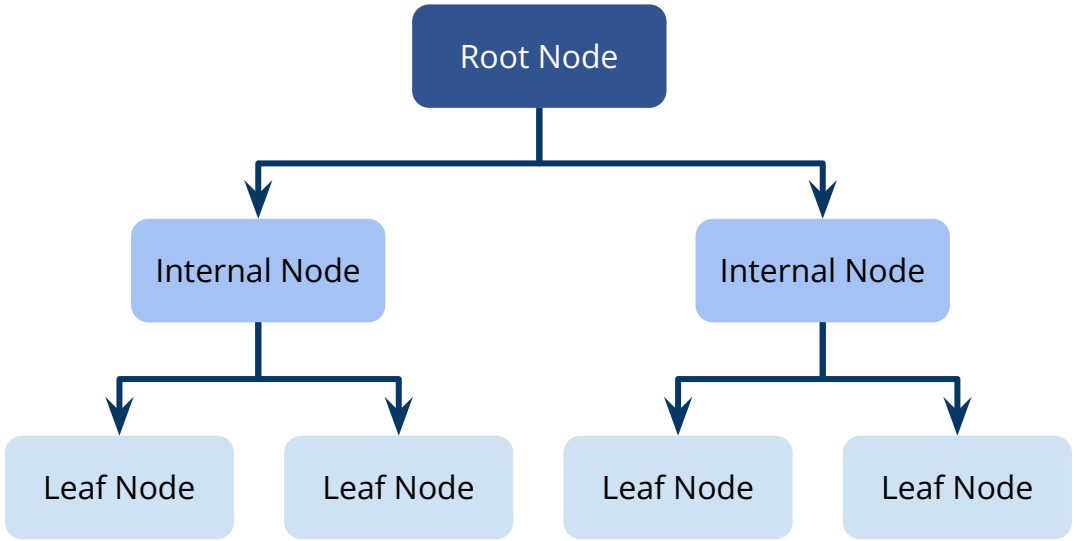
Using Weighted Distance: You can implement a distance-weighted approach where closer neighbors have a higher influence on the result than more distant ones. However, proceed with caution: this can be a double-edged sword if an outlier happens to be the point geographically closest to your query.

Preventive Cleaning (Pre-processing): This is often the most robust solution. Use outlier detection techniques to identify and remove suspicious data points before the k-NN induction phase**.**

Decision Trees

A Decision Tree is a supervised machine learning algorithm used for both classification and regression. Its structure resembles an upside-down tree, where decisions are made by following **a series of logical splits** based on data features.

In simple terms, a Decision Tree is a set of **nested if-else conditions** that help in making decisions based on input features. Basically it mimics **human decision-making**: "If condition X is true, then check condition Y; otherwise, go to Z.



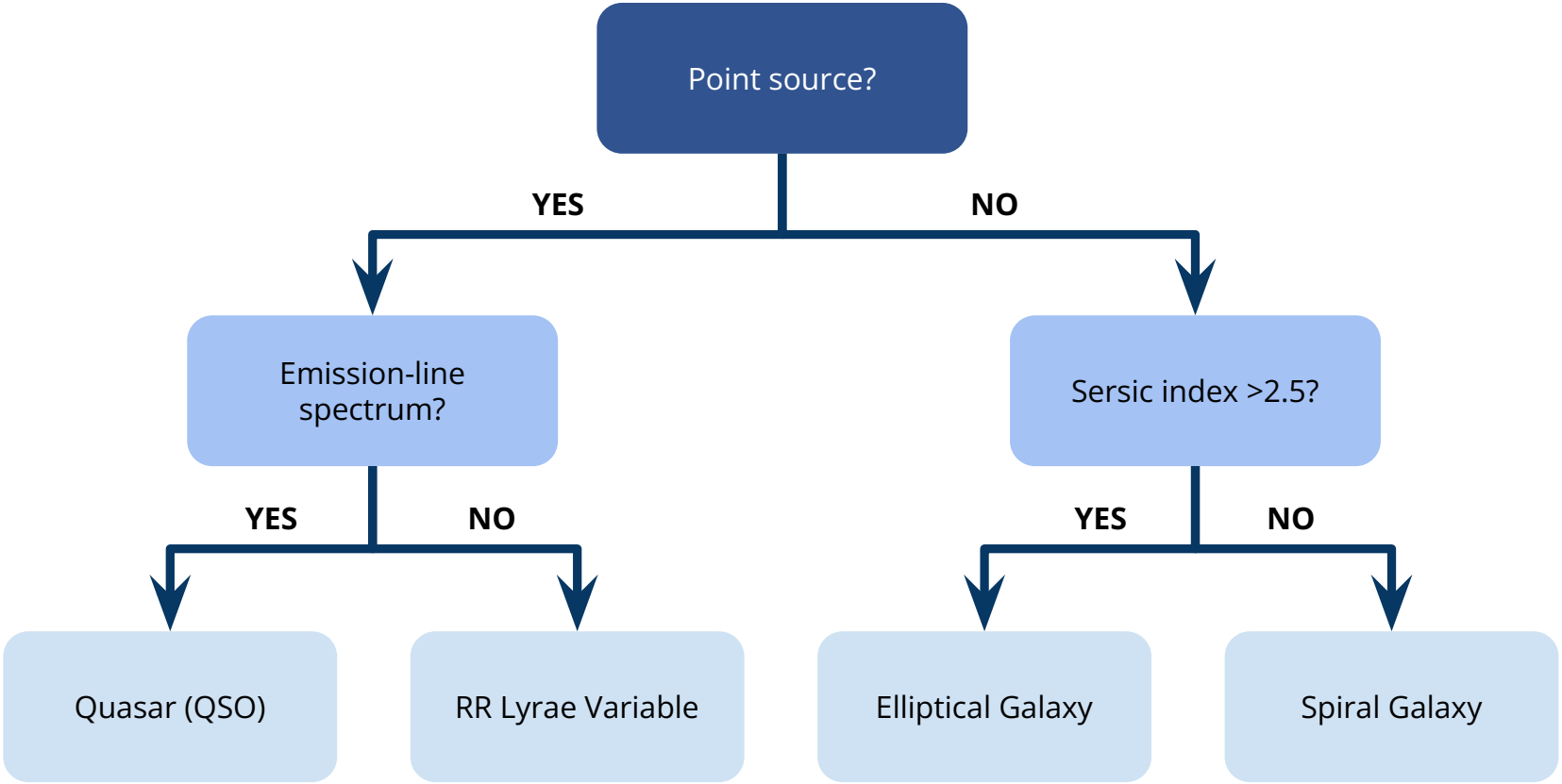
Root node: The topmost node representing the entire dataset. It asks the most important question (the feature that provides the most information gain).

Splits/Branches: These represent the outcomes of a test (e.g., "Yes" or "No"). They connect nodes.

Internal Nodes: Decision points where further questions are asked about specific variables.

Leaf Nodes: The terminal nodes that provide the final prediction (the class label or the numerical value). No further splits occur here.

An astronomical example



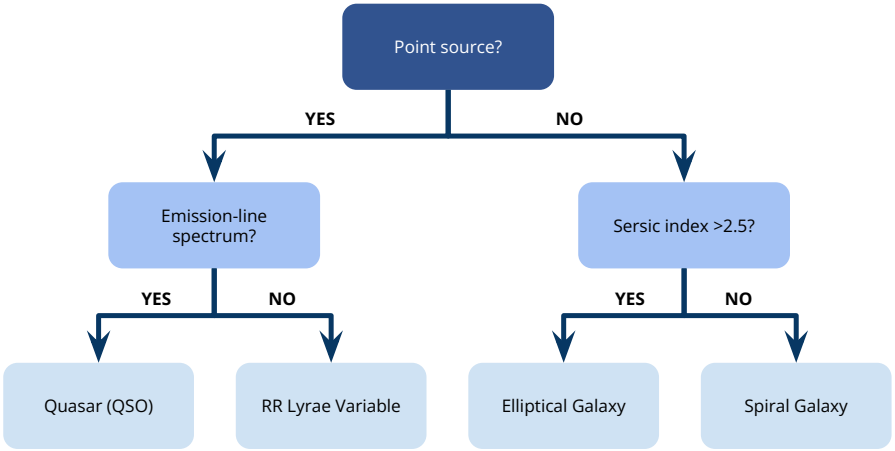
Split Criteria

The determination of the Root Node represents the **most critical phase** in the construction of a Decision Tree, as it requires the identification of the feature that maximizes the initial separation of the global dataset into more homogeneous subsets.

1. Systematic Feature Evaluation (Exhaustive Search)

The algorithm operates according to a physics-agnostic approach, meaning it does not possess a priori knowledge of astrophysical principles; instead, it relies on iterative statistical optimization across the entire feature space.

- **Feature Space Analysis**: The model systematically evaluates every available parameter, including photometric magnitudes, the Sersic Index, morphological indicators, and spectroscopic data.
- **Threshold Optimization**: For continuous variables, the algorithm identifies candidate split points by evaluating every possible numerical interval to determine which threshold (e.g., n > 2.5) yields the highest predictive power.



2. Objective Functions for Impurity Reduction (Gini vs. Entropy)

For every potential question, the algorithm calculates the "confusion" that would remain in the resulting child nodes. Two primary metrics guide this choice:

Gini Impurity (G): Measures the probability of a random observation being incorrectly classified based on the distribution of labels in the node. A "pure" node, such as one exclusively containing Quasars, results in G = 0.

$$I_G = 1 - \sum_{i=1}^{M} p_i^2$$

Entropy (H): Measures the degree of "disorder" or uncertainty within the group. The algorithm's ultimate goal is to minimize this value.

$$H = -\sum_{i=1}^{M} p_i log_2 p_i$$

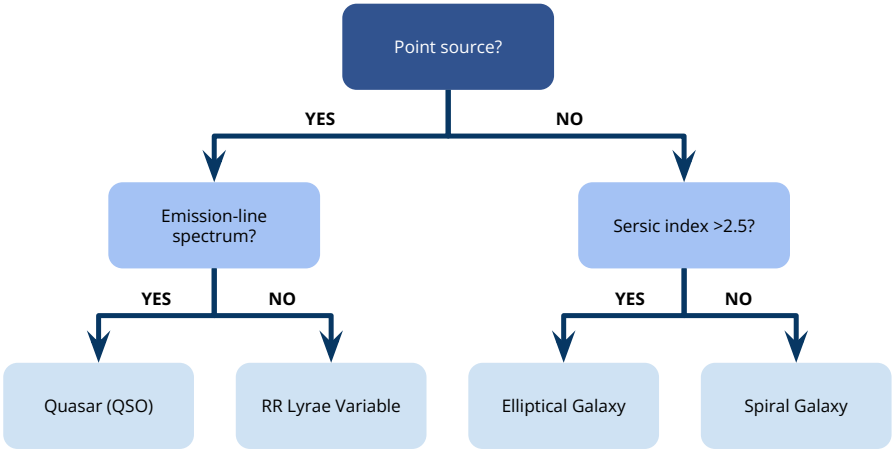
Split Criteria

The determination of the Root Node represents the **most critical phase** in the construction of a Decision Tree, as it requires the identification of the feature that maximizes the initial separation of the global dataset into more homogeneous subsets.

3. Selection of "Point Source?" as the Optimal Root Node

In the context of astronomical classification, the morphological distinction between point-like and extended sources is statistically selected as the primary split for several reasons:

- **Maximization of Information Gain IG**: Segregating sources based on their Point Spread Function (PSF) profile provides the most significant reduction in global impurity, effectively separating compact objects (stars/quasars) from diffuse structures (galaxies).
- **Minimization of Cross-Class Contamination**: This partition fundamentally eliminates the risk of misclassifying high-redshift elliptical galaxies as stellar or quasi-stellar objects early in the decision flow.
- **Algorithmic Efficiency:** Compared to secondary spectral or color-based traits, the point-versus-extended dichotomy generates the most homogeneous (pure) clusters at the initial level of recursive partitioning.



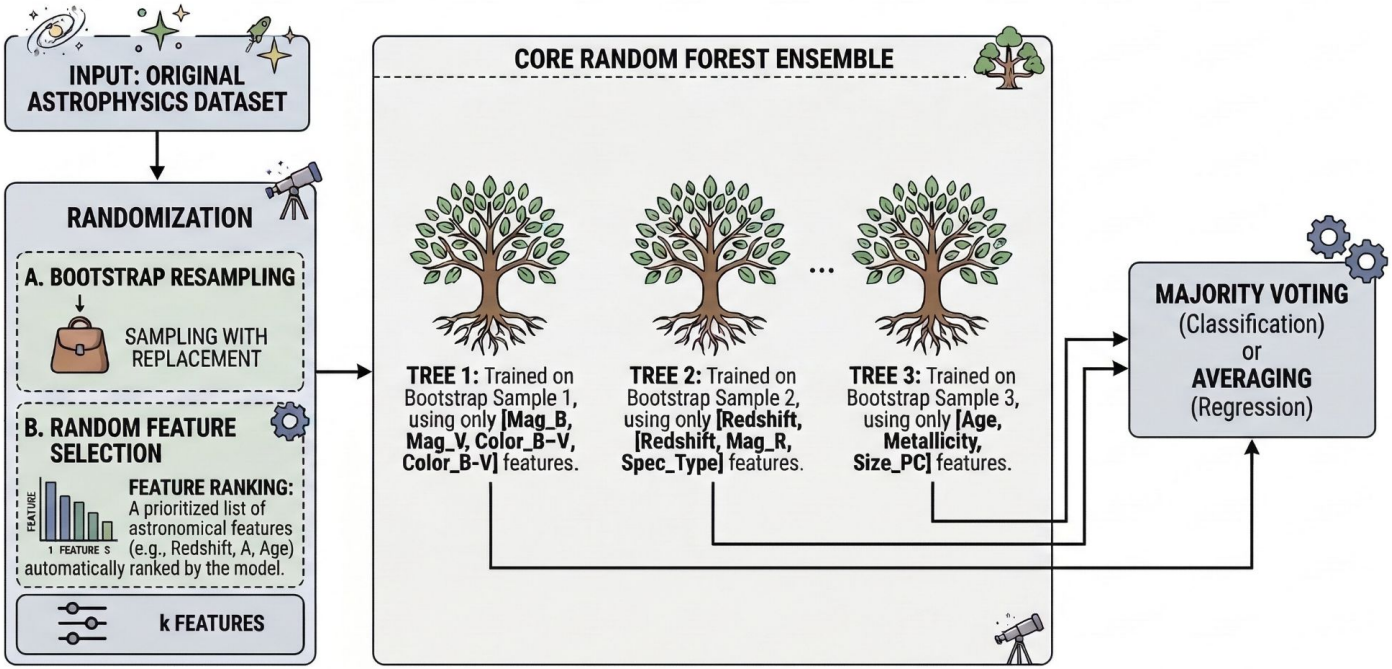
$$IG(T, a) = \mathrm{H}(T) - \mathrm{H}(T|a)$$

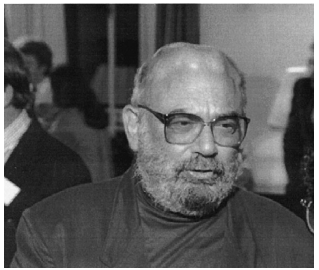
4. Recursive Partitioning and Iterative Optimization

This systematic evaluation is subsequently applied **recursively** to every resulting internal node. At each stage of the hierarchy, the algorithm operates exclusively on the increasingly specific **subset of data** reaching that particular junction. While the **global feature space remains accessible**, the model iteratively identifies the most pertinent indicators to further refine the classification within these localized partitions. This process continues until **a convergence criterion is satisfied**, such as achieving maximum node purity or reaching a predefined terminal depth, effectively isolating the Leaf Nodes as the final output of the decision flow.

Random Forest

While Decision Trees are intuitive, their construction logic is inherently **greedy**: they make the best local split at each node without considering the global optimum. This 'shortsightedness' leads to fragile models and **overfitting**. To solve this, we introduce the Random Forest: an **ensemble** architecture that replaces the myopia of a single tree with the collective wisdom of hundreds, significantly improving robustness and generalization.





Leo Breiman, 1928 - 2005

Leo Breiman (2001)
"Random Forests"
Machine Learning, 45, 5-32

Bagging

To overcome the **greedy** limitations of a single tree, Random Forest employs a dual strategy: **Bagging** (**B**ootstrap **Agg**regat**ing**) to diversify the data samples and **Feature Sampling** to force the exploration of different variables, ensuring a robust and de-correlated ensemble.

1. Bootstrap (Sampling with Replacement)

A statistical technique where multiple training sets of size *N* are generated from an original dataset of size *N* by sampling with replacement. This ensures that each sample is slightly different, containing **duplicates** and leaving out about one-third of the data (**Out-Of-Bag**), which is essential for building a diverse and robust ensemble.

Example

- Imagine a bag containing 100 galaxies.
- For each tree, we draw 100 galaxies at random, but with a rule: every time we pick one, we record it and **put it back in the bag** (sampling with replacement).
- **The Result**: Each tree sees **a slightly different version of the data** - some galaxies appear multiple times, while others don't appear at all. This creates the "*diversity*" needed to prevent overfitting.

Out-Of-Bag (OOB) observations

They can be used as *internal test set* to evaluate performance without a separate cross-validation

2. Aggregating (Combining Results)

Once all trees have been trained on their respective bootstrap samples, we combine their predictions:

- **For Classification**: We use **Majority Voting**. For example, if 80 trees say "Spiral Galaxy" and 20 say "Elliptical," the final output is "Spiral."
- **For Regression**: We simply take the Average of all the trees' predictions.

The Winning Formula: Bagging + Feature Sampling

A Random Forest is more than just Bagging. It adds **Feature Sampling**: at each node split, the algorithm only considers **a random subset of features**. This '**de-correlates**' the trees. Even if one variable is overwhelmingly strong, the forest is forced to look at other dimensions, turning a collection of diverse, 'weak' perspectives into a powerful, stable ensemble.

Feature Importance & Ranking

While the model randomly selects variables for each tree, it simultaneously calculates how much each individual feature **contributes** to reducing the global error.

- **Measuring Importance**: Every time a variable (e.g., Redshift) is used for a split, the algorithm calculates the purity gain (or variance reduction) achieved.
- **Scientific Ranking**: At the end of the training process, we obtain a ranked list of the most influential variables.

Why is this fundamental in Astrophysics?

1. **Physical Interpretation**: It is not enough to know that a galaxy is elliptical; we need to understand which parameters (color, mass, metallicity) led to that classification.
2. **Dimensionality Reduction**: It allows us to discard "background noise" in the catalog, focusing only on the physical quantities that are truly informative.
3. **Model Validation**: If the Feature Ranking prioritizes variables that physical theory considers irrelevant, we know the model might have captured data biases.

Feature Sampling is what makes a **Random Forest unique**: it prevents dominant variables (like *Luminosity*) from taking over every tree. Without it, your ensemble would just be a collection of identical, highly correlated copies, losing the power of diversity.

A Final Comparison

Feature	KNN	Decision Tree	Random Forest
Conceptual Simplicity	🟢 Intuitive (**Proximity**)	🟢 Very Simple	🟡 Moderate (**Ensemble**)
Interpretability	🟡 Only for low **k**	🟢 High (**Visual**)	🔴 Low (**Black Box**)
Training Speed	🟢 Instant (**Lazy**)	🟢 Fast	🔴 Slow (**Multiple trees**)
Prediction Speed	🔴 Slow (**Calculates distances**)	🟢 Instant	🟡 Moderate
Robustness to Outliers	🔴 Highly sensitive	🟢 Robust	🟢 Very Robust
Overfitting Risk	🟡 High if **k** is small	🔴 Very High	🟢 Low (**Due to Bagging**)
Missing Values Handling	🔴 Requires Imputation	🟢 Handles them well	🟢 Handles them well
Data Scaling Required	🔴 Mandatory (**Standardize**)	🟢 Not required	🟢 Not required
Memory Usage	🔴 High (**Stores dataset**)	🟢 Low	🟡 Moderate/High
General Accuracy	🟡 Medium	🟡 Medium	🟢 Very High

Unsupervised Learning

Overview

Unsupervised Learning represents a machine learning paradigm in which the system extracts inferential structures from a dataset devoid of labels or a predefined target variable.Unlike the supervised paradigm, where the objective is the approximation of a mapping function $f(x) = y$ based on known input-output pairs, in unsupervised learning, the algorithm operates exclusively on the feature space $\boldsymbol{X}$ to model the underlying probability distribution or to identify its topological properties.

Operational Mechanism

The algorithm acts as a statistical explorer that analyzes **density**, **variance**, and **distance** (metrics) between observations. The goal is not the prediction of a value, but rather the **discovery of emerging patterns** or the synthesis of information through internal optimization criteria (such as minimizing intra-cluster variance or reconstruction error).

Overview

Unsupervised Learning represents a machine learning paradigm in which the system extracts inferential structures from a dataset devoid of labels or a predefined target variable. Unlike the supervised paradigm, where the objective is the approximation of a mapping function $f(x) = y$ based on known input-output pairs, in unsupervised learning, the algorithm operates exclusively on the feature space $\boldsymbol{X}$ to model the underlying probability distribution or to identify its topological properties.

Operational Mechanism

The algorithm acts as a statistical explorer that analyzes **density**, **variance**, and **distance** (metrics) between observations. The goal is not the prediction of a value, but rather the **discovery of emerging patterns** or the synthesis of information through internal optimization criteria (such as minimizing intra-cluster variance or reconstruction error).

Clustering (Group Analysis)

Clustering aims to partition the dataset into subsets (clusters) such that the similarity between elements of the same group is maximized, while similarity between elements of different groups is minimized.

- **Astrophysical Example**: Taxonomic classification of celestial sources based on multidimensional photometric properties, useful for identifying stellar populations with similar evolutionary traits without pre-assigned classes.
- **Common Algorithms**: K-Means (centroid-based), DBSCAN (density-based), Hierarchical Clustering.

Overview

Unsupervised Learning represents a machine learning paradigm in which the system extracts inferential structures from a dataset devoid of labels or a predefined target variable. Unlike the supervised paradigm, where the objective is the approximation of a mapping function $f(x) = y$ based on known input-output pairs, in unsupervised learning, the algorithm operates exclusively on the feature space $\boldsymbol{X}$ to model the underlying probability distribution or to identify its topological properties.

Operational Mechanism

The algorithm acts as a statistical explorer that analyzes **density**, **variance**, and **distance** (metrics) between observations. The goal is not the prediction of a value, but rather the **discovery of emerging patterns** or the synthesis of information through internal optimization criteria (such as minimizing intra-cluster variance or reconstruction error).

Clustering (Group Analysis)

Clustering aims to partition the dataset into subsets (clusters) such that the similarity between elements of the same group is maximized, while similarity between elements of different groups is minimized.

- **Astrophysical Example**: Taxonomic classification of celestial sources based on multidimensional photometric properties, useful for identifying stellar populations with similar evolutionary traits without pre-assigned classes.
- **Common Algorithms**: K-Means (centroid-based), DBSCAN (density-based), Hierarchical Clustering.

Dimensionality Reduction

You already know it...

Association Rules (Association Rule Learning)

This involves discovering probabilistic relationships and co-occurrences between variables in transactional datasets or relational databases.

- **Example**: Identifying systemic correlations between different chemical indicators in planetary atmospheres to infer underlying physical processes.

Overview

Unsupervised Methods (UM) are applied **without any a priori knowledge**... They cluster the data relying on their intrinsic statistical properties. The understanding only takes place through labeling (**very limited Knowledge Base**).

"a blind man in a dark room - looking for a black cat - which may be not there"
Charles Bowen

to be honest it is full of cats, the problem is to find the cat interesting us...

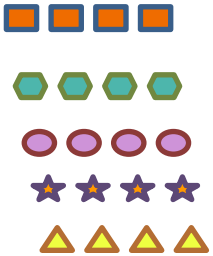
Unsupervised Methods:

- need little or none a-priori knowledge;
- do not reproduce biases present in the KB;
- require more complex error evaluation (through complex statistics);
- are computationally intensive;
- are not user friendly
  ...more an art than a science; i.e. lot of experience required

Overview

The World

Clustering

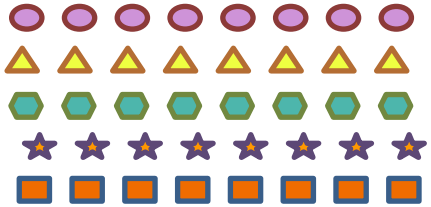


Clusters

Analysis of results

Analysis of results

Analysis of results

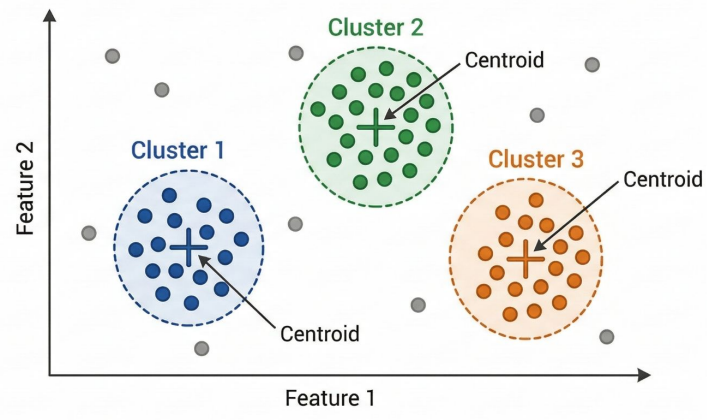


New Knowledge

Clustering

Clustering is the core activity of unsupervised Machine Learning and serves as a fundamental pillar for **Knowledge Discovery in Databases (KDD)**

Given a dataset ***X*** consisting of objects x_i, the objective is to assign each point to a finite system of ***k*** subsets (clusters), maximizing internal similarity within groups and minimizing similarity between different groups



Distance metrics constitute the **primary objective function** guiding iterative partitioning during the training phase and, simultaneously, serve as the **analytical benchmark** to quantify the structural validity and coherence of the resulting clusters in the absence of a priori labels.

Metrics

Metrics are primarily used to measure two concepts: **compactness** (how close the points in the same group are) and **separation** (how far apart the groups are from each other). Metrics depends on the underlying geometry and spatial relationships of the input parameter space, defined by the distance function chosen (Euclidean, Mahalanobis, Geodesic…)

Compactness - Intra-cluster Metrics

These metrics measure the **internal cohesion** or density of a single cluster C_k. The goal during training is typically to **minimize** these values to ensure that points within a group are as similar as possible.

- **Centroid-based Distance (d_{ce})**: Calculates the average distance of all patterns in the cluster from their geometric center (centroid).
- **Mean-based Distance (d_{av})**: Measures the average of all pairwise distances between the objects within the cluster. This is computationally expensive for large datasets.
- **Closest Neighbor-based Distance (d_{mn})**: Focuses on local density by evaluating the minimum distance between points within the same cluster

$$d_{ce}(k) = \frac{1}{N_k} \sum_{i \in C_k} |x_i - CE_k|$$

$$d_{av}(k) = \frac{\sum_{i,j \in C_k, i \neq j} |x_i - x_j|}{N_k(N_k - 1)}$$

$$d_{mn}(k) = \frac{\sum_{i \in C_k} \min_{j \in C_k, j \neq i} |x_i - x_j|}{N_k}$$

Separation - Inter-cluster Metrics

These metrics quantify the isolation or **"gap" between two distinct clusters**, C_p and C_t. Effective clustering algorithms aim to **maximize** these distances to ensure that different categories are clearly distinguishable.

- **Single-linkage Distance (D_{sl} or D_{min})**: Defined as the minimum distance between the two closest objects belonging to different clusters. While simple, it is highly sensitive to noise and outliers.
- **Complete-linkage Distance (D_{cl} or D_{max})**: Measures the maximum distance between the two most distant objects in the respective clusters. It requires clusters to be very compact and well-separated to be effective.
- **Centroid-based Distance (D_{ce})**: The geometric distance calculated specifically between the centroids of two clusters.
- **Mean-based Distance (D_{av})**: The geometric distance calculated by averaging the distances between all of points of the clusters

$$D_{sl}(p,t) = \min_{i \in C_p, j \in C_t} |x_i - x_j|$$

$$D_{cl}(p,t) = \max_{i \in C_p, j \in C_t} |x_i - x_j|$$

$$D_{ce}(p,t) = |CE_p - CE_t|$$

$$D_{av}(p,t) = \frac{\sum_{i \in C_p} \sum_{j \in C_t} |x_i - x_j|}{N_p N_t}$$

Metrics - Silhouette score

Silhouette score

The Silhouette Score measures the balance between an observation *cohesion* (similarity to its own cluster) and its *separation* (difference from other clusters). Unlike global indices that look at centroids, the Silhouette provides a local evaluation of structural coherence.

$$s(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))}$$

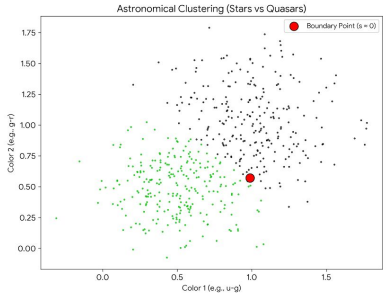
a(i) **(Cohesion)**
The average distance between point *i* and all other points within the same cluster. This represents intra-cluster compactness.

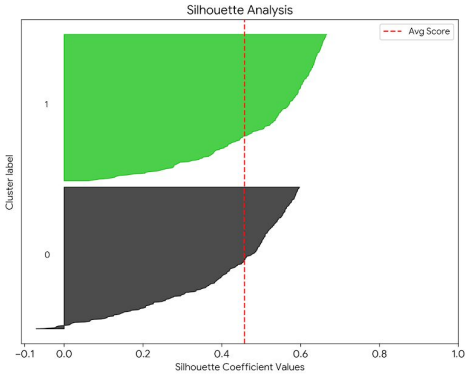
b(i) **(Separation)**
The average distance between point *i* and all points in the nearest cluster to which *i* does not belong.

- **Close to +1**: Indicates the point is far from neighboring clusters and perfectly integrated into its own.
- **Close to 0**: Indicates the point lies on the decision boundary between two clusters (overlapping zone).
- **Close to -1**: Indicates the point has likely been assigned to the wrong cluster.

Star - Quasar clustering example

Two partially overlapping celestial populations. The Red Point (Boundary Point) is physically located in a region where the two clusters "touch." Geometrically, it is difficult to definitively state whether it belongs to the green or the black cluster.





The Silhouette Plot

Each horizontal line represents the score of a single object.The Boundary Point corresponds to one of the very short bars near zero. This value near 0 mathematically indicates uncertainty: the point is nearly equidistant from both clusters.

Negative Values: Misclassified objects—stars that have colors more similar to quasars, and vice versa.

Red Dashed Line: This represents the global average score. If the average is low, it implies that the two populations are not physically well-separated in the provided parameter space.

Metrics - Davies-Bouldin Index (DBI)

- computationally efficient and significantly faster than the Silhouette Score
- misleadingly poor score for non-spherical cluster

The DBI measures the **ratio** between **intra-cluster scatter** and **inter-cluster separation**. The objective is to identify a data partition where clusters are **highly compact** (low internal variance) and **well-separated** from one another (high distance between groups).

The lower the DBI value, the better the clustering.

A value approaching 0 represents a nearly perfect partition where clusters are highly concentrated and infinitely distant from each other.

$$DB = \frac{1}{K}\sum_{i=1}^{K}\max_{j\neq i}\left(\frac{S_i + S_j}{M_{ij}}\right)$$

S_i (Scatter): The average distance of all points within cluster *i* from its centroid (representing intra-cluster compactness).
M_{ij} (Separation): The geometric distance between the centroids of cluster *i* and cluster *j* (representing inter-cluster separation).

For every cluster *i*, the index identifies the specific cluster *j* that is "most similar" (the one that yields the highest ratio).
This makes the DBI a conservative metric, as it focuses on the worst-case scenario for separation.

Metrics - Davies-Bouldin Index (DBI)

- computationally efficient and significantly faster than the Silhouette Score
- misleadingly poor score for non-spherical cluster

The DBI measures the **ratio** between **intra-cluster scatter** and **inter-cluster separation**. The objective is to identify a data partition where clusters are **highly compact** (low internal variance) and **well-separated** from one another (high distance between groups).

The lower the DBI value, the better the clustering.
A value approaching 0 represents a nearly perfect partition where clusters are highly concentrated and infinitely distant from each other.

$$DB = \frac{1}{K}\sum_{i=1}^{K}\max_{j\neq i}\left(\frac{S_i + S_j}{M_{ij}}\right)$$

S_i (Scatter): The average distance of all points within cluster *i* from its centroid (representing intra-cluster compactness).
M_{ij} (Separation): The geometric distance between the centroids of cluster *i* and cluster *j* (representing inter-cluster separation).

For every cluster *i*, the index identifies the specific cluster *j* that is "most similar" (the one that yields the highest ratio).
This makes the DBI a conservative metric, as it focuses on the worst-case scenario for separation.

Star - Quasar clustering example

To reconnect with our previous example, let's examine how the Davies-Bouldin Index (DBI) evaluates the same dataset from a global, centroid-based perspective

Scenario A: Optimal Clustering (K=2)
The algorithm correctly identifies the two primary physical populations.

C_1 - Stars: Highly dense and compact → S_{stars} = 0.2

C_2 - Quasars: Slightly more dispersed → $S_{quasars}$ = 0.4

Distance between centroids M_{12} = 2.0.

DBI = 0.3 ↓

Metrics - Davies-Bouldin Index (DBI)

computationally efficient and significantly faster than the Silhouette Score

misleadingly poor score for non-spherical cluster

The DBI measures the **ratio** between **intra-cluster scatter** and **inter-cluster separation**. The objective is to identify a data partition where clusters are **highly compact** (low internal variance) and **well-separated** from one another (high distance between groups).

The lower the DBI value, the better the clustering.

A value approaching 0 represents a nearly perfect partition where clusters are highly concentrated and infinitely distant from each other.

$$DB = \frac{1}{K}\sum_{i=1}^{K}\max_{j\neq i}\left(\frac{S_i + S_j}{M_{ij}}\right)$$

S_i (Scatter): The average distance of all points within cluster *i* from its centroid (representing intra-cluster compactness).
M_{ij} (Separation): The geometric distance between the centroids of cluster *i* and cluster *j* (representing inter-cluster separation).

For every cluster *i*, the index identifies the specific cluster *j* that is "most similar" (the one that yields the highest ratio).
This makes the DBI a conservative metric, as it focuses on the worst-case scenario for separation.

Star - Quasar clustering example

To reconnect with our previous example, let's examine how the Davies-Bouldin Index (DBI) evaluates the same dataset from a global, centroid-based perspective

Scenario A: Optimal Clustering (K=2)
The algorithm correctly identifies the two primary physical populations.

C_1 - Stars: Highly dense and compact → S_{stars} = 0.2

C_2 - Quasars: Slightly more dispersed → $S_{quasars}$ = 0.4

Distance between centroids M_{12} = 2.0.

DBI = 0.3 ↓

Scenario B: Over-clustering (K=3)
Suppose the model is forced to create a third cluster, splitting the Quasar group into two (C_2, C_3).

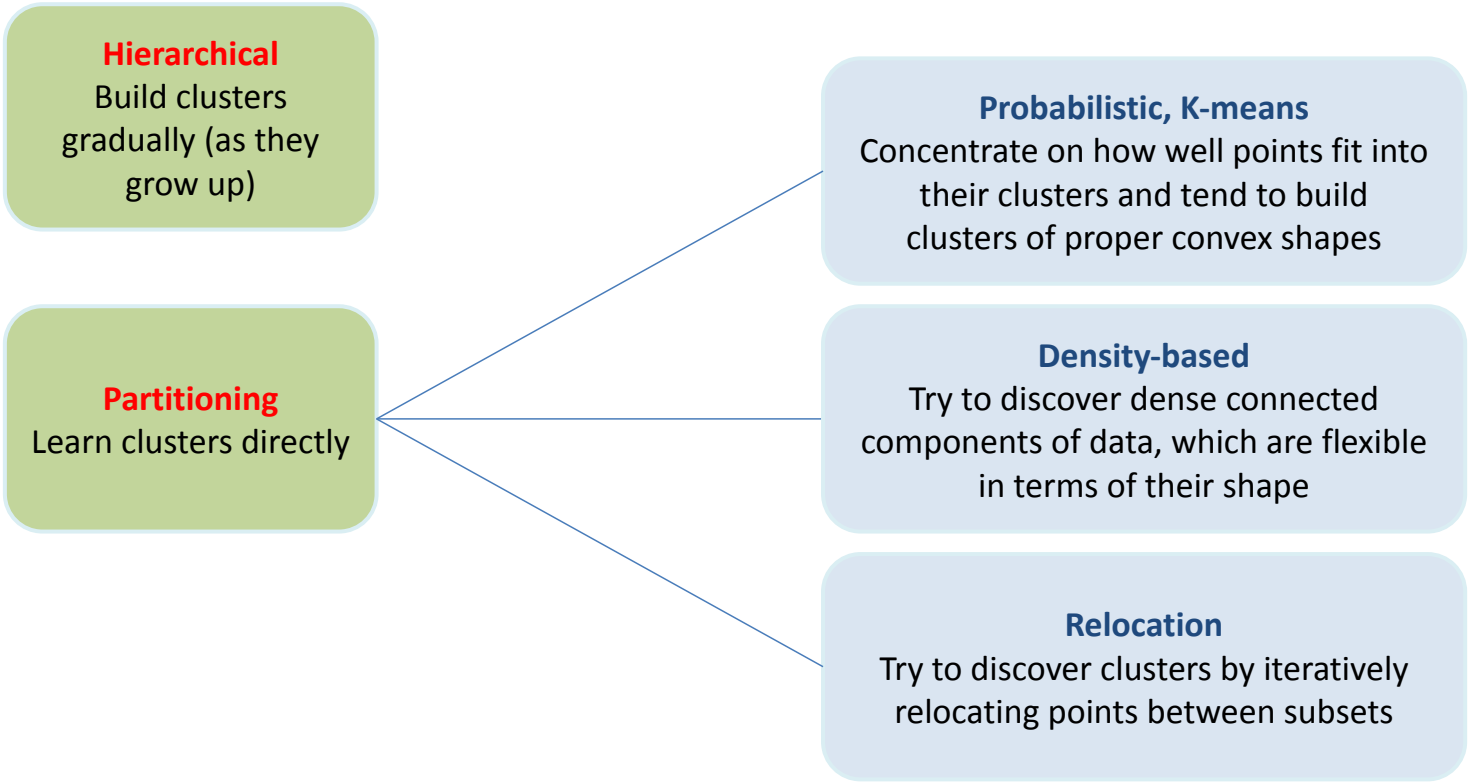
Compactness (*S*): the individual groups are even smaller and more compact ***S*~0.15**

Distance between centroids (*M*): the new centroids (2 and 3) are now very close to each other because they belong to the same parent population → M_{23} = 0.5

DBI = 0.88 ↑

DBI can be used to plot a graph against varying values of **K**. The point where the DBI reaches its **minimum** identifies **the number of physically distinct populations present in your data**.

Main Clustering Taxonomy



Kmeans

KMeans is an unsupervised clustering algorithm that groups data based on **distances**. The basic idea is to **divide** a data set into distinct clusters, with each point **belonging to the cluster whose centroid is closest**.

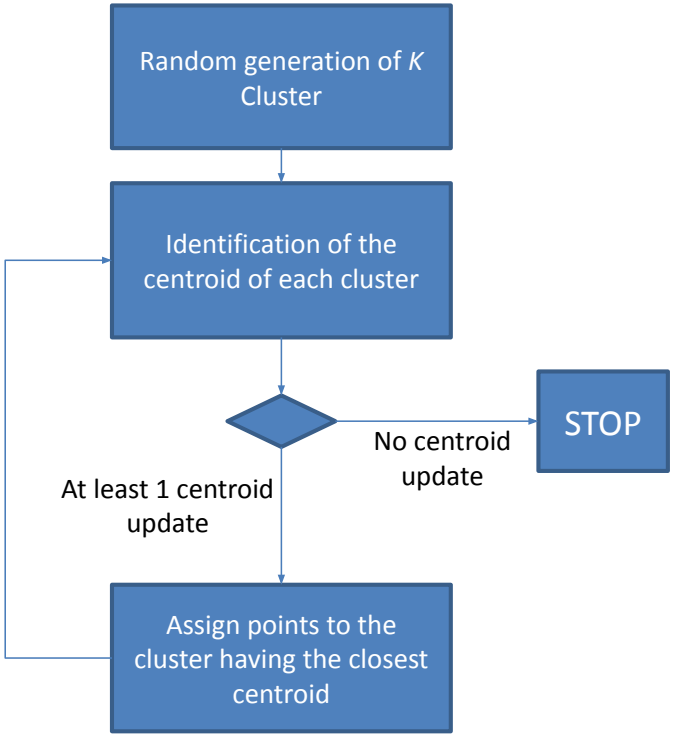
PROS

- ✓ **Computational Efficiency**: It is exceptionally fast. Its time complexity is linearly proportional to the number of patterns, making it ideal for Astronomical Big Data
- ✓ **Scalability**: It handles millions of records efficiently, provided there is enough RAM to store the dataset
- ✓ **Intuition and Interpretability**: The concept of a "centroid" is easy to explain, and results are immediately interpretable within a physical or parameter space
- ✓ **Guaranteed Convergence**: The algorithm always converges to a result, providing a stable (though not necessarily global) partition.

CONS

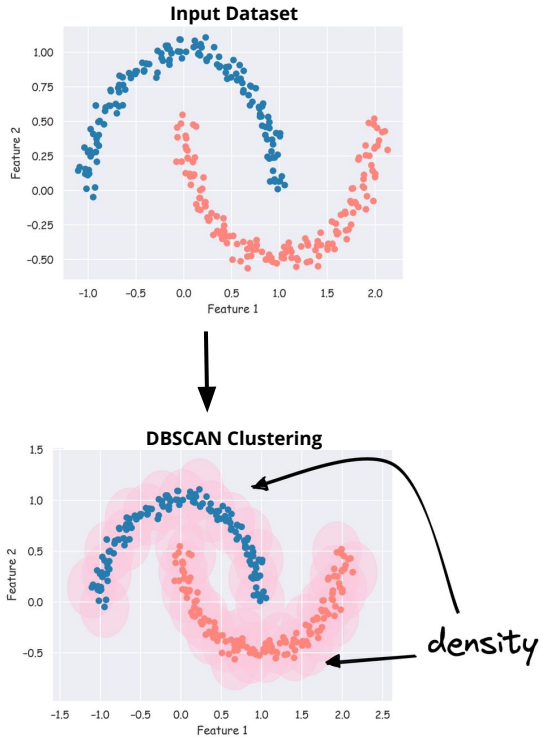
- x **Predefined *K***: K-Means does not "discover" the number of groups; the user must specify *K* beforehand
- x **Geometric Bias (Spherical Clusters)**: Since K-Means relies on the Euclidean distance, it inherently favors spherical/circular clusters
- x **Initialization Sensitivity**: If the initial centroids are poorly placed by chance, the algorithm can get stuck in a local minimum. This is often mitigated using the **K-Means++** initialization method
- **Sensitivity to Outliers**: Since centroids are calculated as the mean of the points, a single extreme outlier (e.g., a measurement error or a rare transient event) can "pull" the centroid away from the cluster's core, distorting the partition

REALLY A CON?



DBSCAN

DBSCAN (**Density-Based Spatial Clustering of Applications with Noise**) differs radically from K-Means because it does not look for centroids, but models the **local density** of the data. The idea is to group points based on "density", that is points close together in a high-density region and separated by lower-density regions.



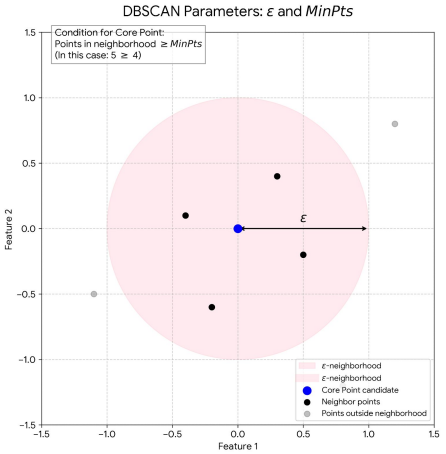
Model Hyperparameters

- **ε** : The scanning radius that defines the neighborhood around a specific point
- ***MinPts*** : The minimum number of points required within the **ε**-radius for that region to be considered "dense."

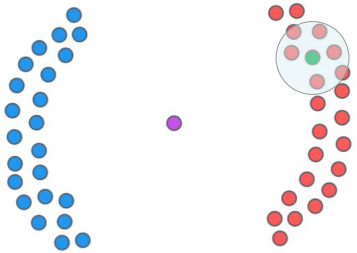
Classification of Points

The algorithm classifies every object in the dataset into one of three rigorous categories:

1. **Core Point (p_c)** : A point that contains at least *MinPts* within its ε-neighborhood. These points act as the "engine" that expands the cluster
2. **Border Point (p_b)** : A point that has fewer than *MinPts* in its neighborhood but falls within the ε-radius of a Core Point. These points delineate the cluster's edges
3. **Noise Point (Outlier)** : A point that is neither a Core Point nor a Border Point. These are automatically ignored by the clustering process

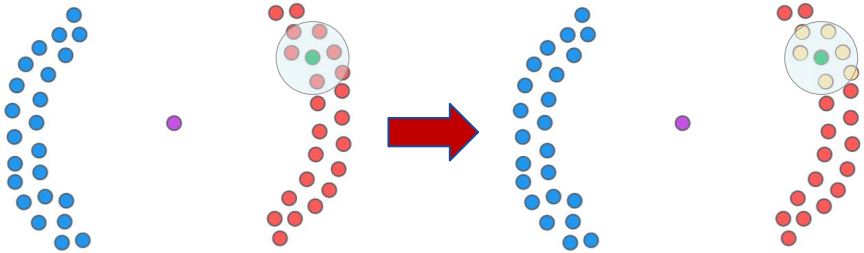


DBSCAN - How does it works?



1. Select a random unvisited point *p* and draw a circle with radius ε

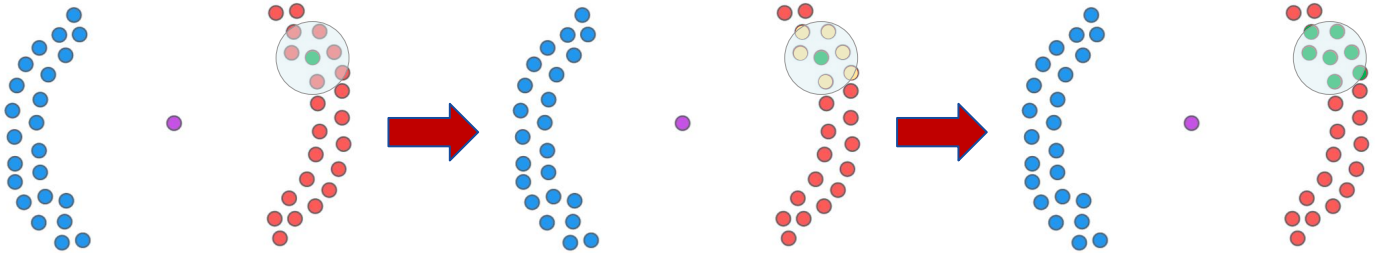
DBSCAN - How does it works?



1. Select a random unvisited point *p* and draw a circle with radius ε

2. Count how many points fall inside the circle (N_e)

DBSCAN - How does it works?

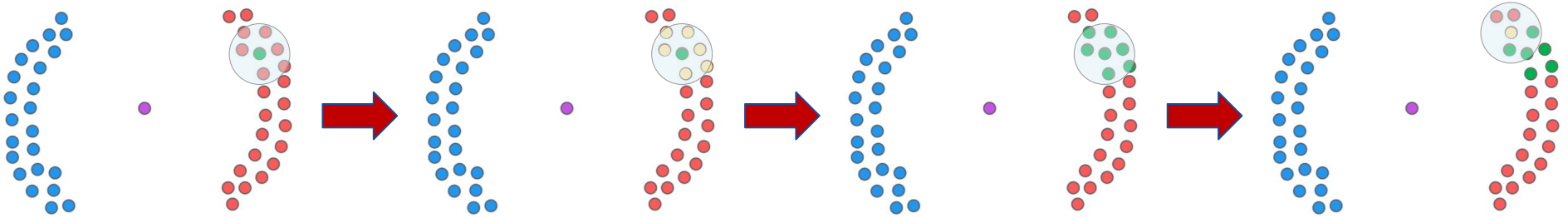


1. Select a random unvisited point *p* and draw a circle with radius ε

2. Count how many points fall inside the circle (N_e)

3. If N_e > *MinPts*, then the point is a **Core Point** and a cluster is created containing all the points ∈ N_e

DBSCAN - How does it works?



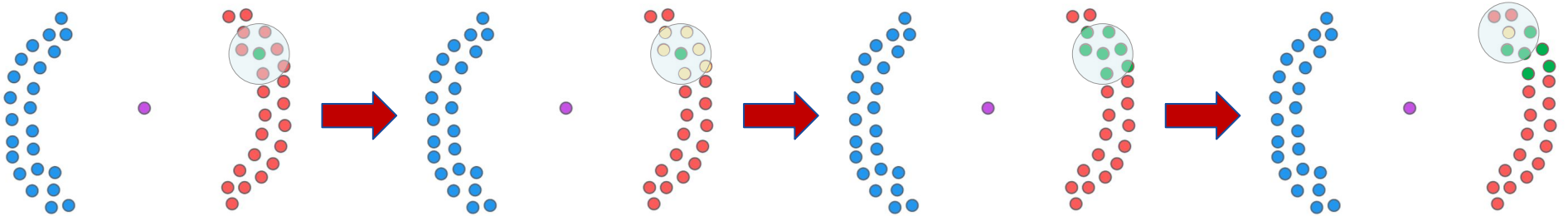
1. Select a random unvisited point *p* and draw a circle with radius ε

2. Count how many points fall inside the circle (N_e)

3. If N_e > *MinPts*, then the point is a **Core Point** and a cluster is created containing all the points ∈ N_e

4. Check the other points in the cluster

DBSCAN - How does it works?

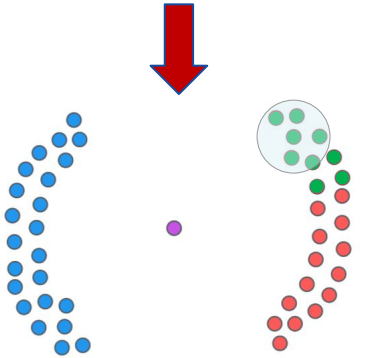


1. Select a random unvisited point *p* and draw a circle with radius ε

2. Count how many points fall inside the circle (N_e)

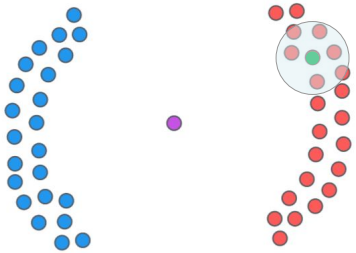
3. If $N_e > MinPts$, then the point is a **Core Point** and a cluster is created containing all the points $\in N_e$

4. Check the other points in the cluster

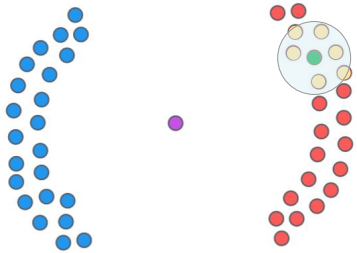


5. If a point is itself a Core Point, all its neighbors are included in the current cluster

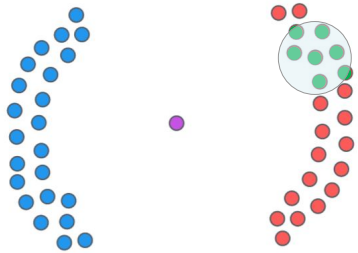
DBSCAN - How does it works?



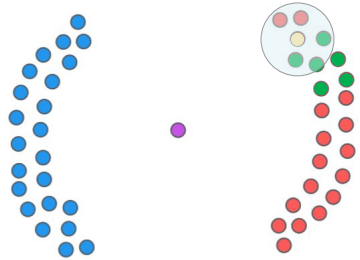
1. Select a random unvisited point *p* and draw a circle with radius ε



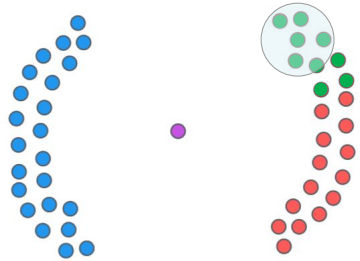
2. Count how many points fall inside the circle (N_e)



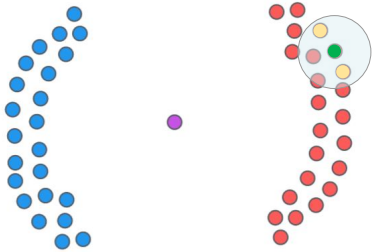
3. If $N_e > MinPts$, then the point is a **Core Point** and a cluster is created containing all the points $\in N_e$



4. Check the other points in the cluster

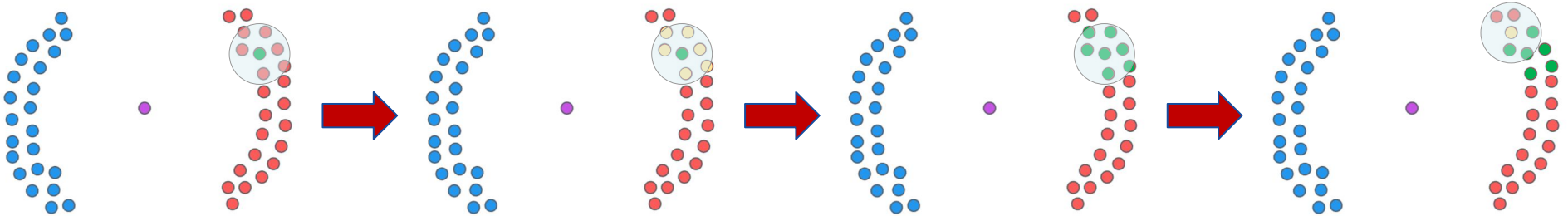


5. If a point is itself a Core Point, all its neighbors are included in the current cluster



6. If the checked point has $N_e < MinPts$ but it's inside a radius of a Point Core, then it is a **Border Point**. The cluster check stops

DBSCAN - How does it works?



1. Select a random unvisited point *p* and draw a circle with radius ε

2. Count how many points fall inside the circle (N_e)

3. If $N_e > MinPts$, then the point is a **Core Point** and a cluster is created containing all the points $\in N_e$

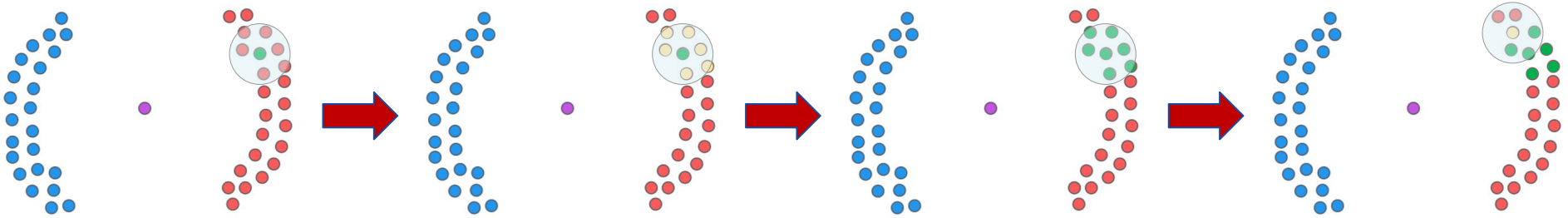
4. Check the other points in the cluster

5. If a point is itself a Core Point, all its neighbors are included in the current cluster

6. If the checked point has $N_e < MinPts$ but it's inside a radius of a Point Core, then it is a **Border Point**. The cluster check stops

7. Restart check with a new random unvisited point and a new cluster to populate

DBSCAN - How does it works?



1. Select a random unvisited point *p* and draw a circle with radius ε

2. Count how many points fall inside the circle (N_e)

3. If $N_e > MinPts$, then the point is a **Core Point** and a cluster is created containing all the points $\in N_e$

4. Check the other points in the cluster

5. If a point is itself a Core Point, all its neighbors are included in the current cluster

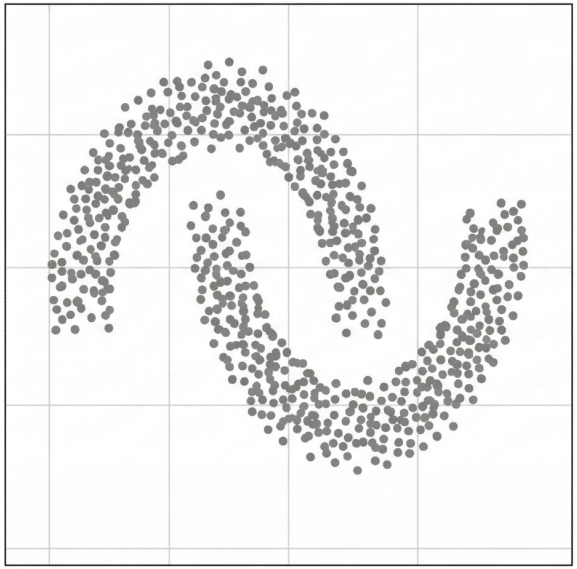
6. If the checked point has $N_e < MinPts$ but it's inside a radius of a Point Core, then it is a **Border Point**. The cluster check stops

7. Restart check with a new random unvisited point and a new cluster to populate

8. If a point is neither a Core Point nor a Border Point, it is labeled as **Noise Point**

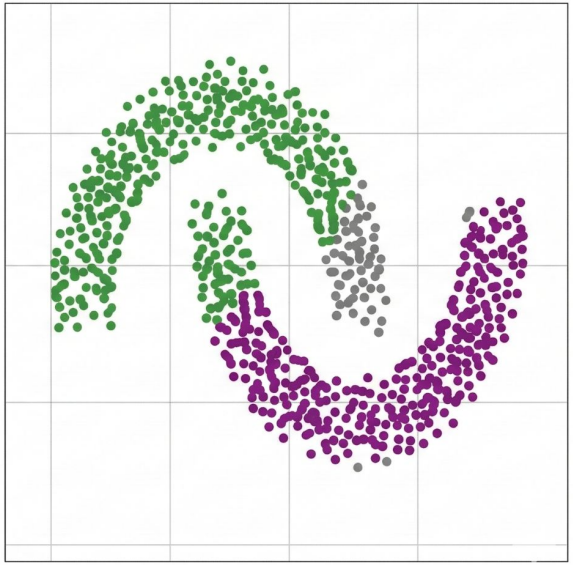
DBSCAN - An easy test

INPUT Data

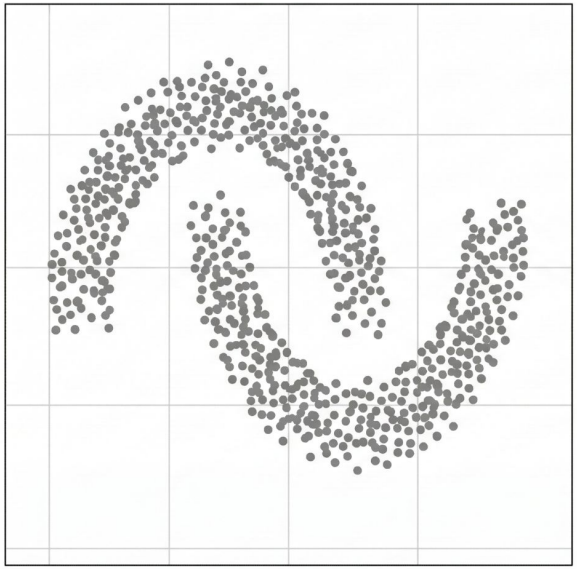


DBSCAN - An easy test

KMeans Clustering

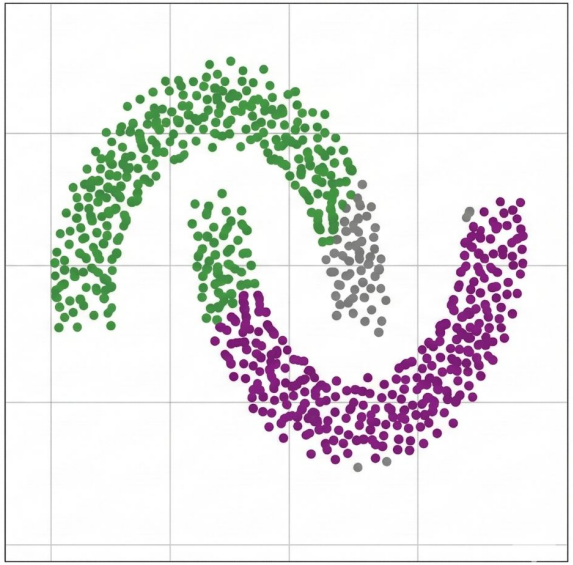


INPUT Data

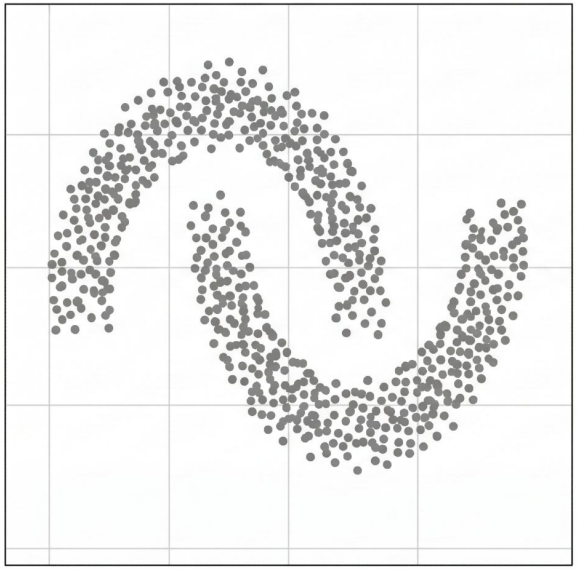


DBSCAN - An easy test

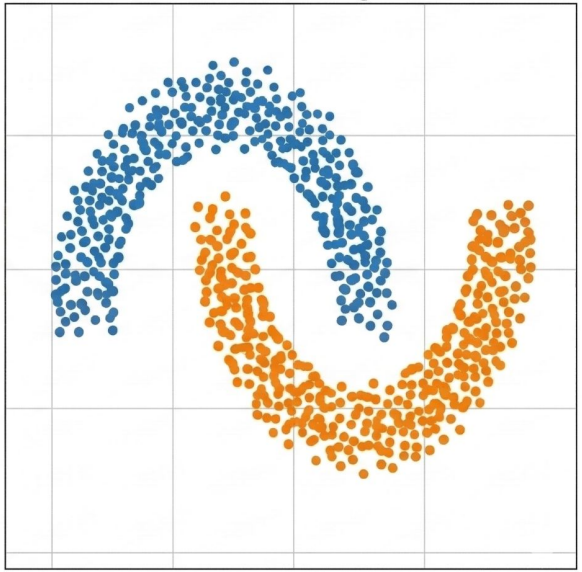
KMeans Clustering



INPUT Data



DBSCAN Clustering



DBSCAN - Pro & Cons

PROS

- ✓ **Discovery of Arbitrary Morphologies**: Unlike K-Means, which is biased toward discovering hyperspherical clusters, DBSCAN can identify clusters of **arbitrary and non-convex shapes**. This is achieved by following local density paths rather than calculating distances to a central point.
- ✓ **Automatic Determination of Cluster Count**: The algorithm does not require the user to pre-specify the number of clusters **(k)**. The number of discovered clusters is an emergent property of the data density and the chosen parameters (**ε** and ***MinPts***).
- ✓ **Robustness to Noise and Outliers**: DBSCAN is intrinsically designed to handle noise. Points that do not satisfy the density criteria and are not reachable from any core point are explicitly classified as "Noise". This prevents outliers from **distorting the cluster geometry**.
- ✓ **Interpretability of Parameters**: The parameters **ε** (distance threshold) and ***MinPts*** (density threshold) have clear physical and domain-specific interpretations, which are particularly valuable in spatial and geographical information systems (GIS).

CONS

- x **Vulnerability to Varying Densities**: This is arguably the most significant limitation. Since DBSCAN utilizes a global **ε** parameter, it fails to effectively cluster datasets where different clusters exhibit significantly **different local densities**. A radius that captures a sparse cluster may cause multiple dense clusters to merge erroneously.
- x **The Curse of Dimensionality**: In high-dimensional spaces, the "distance" between points becomes **less meaningful** (the distance to the nearest neighbor approaches the distance to the farthest neighbor). Consequently, density-based metrics lose their discriminative power unless dimensionality reduction (e.g., PCA) is performed first.
- x **High Sensitivity to Parameterization**: The algorithm's performance is highly sensitive to the choice of **ε**. Minor variations in the radius can lead to radically different results, transitioning from classifying most points as noise to merging distinct clusters.
- x **Non-Deterministic Boundary Assignment**: While the identification of core points and noise is deterministic, the assignment of border points may depend on the **processing order** of the data if a point is reachable from multiple clusters.

HDBSCAN - Hierarchical DBSCAN

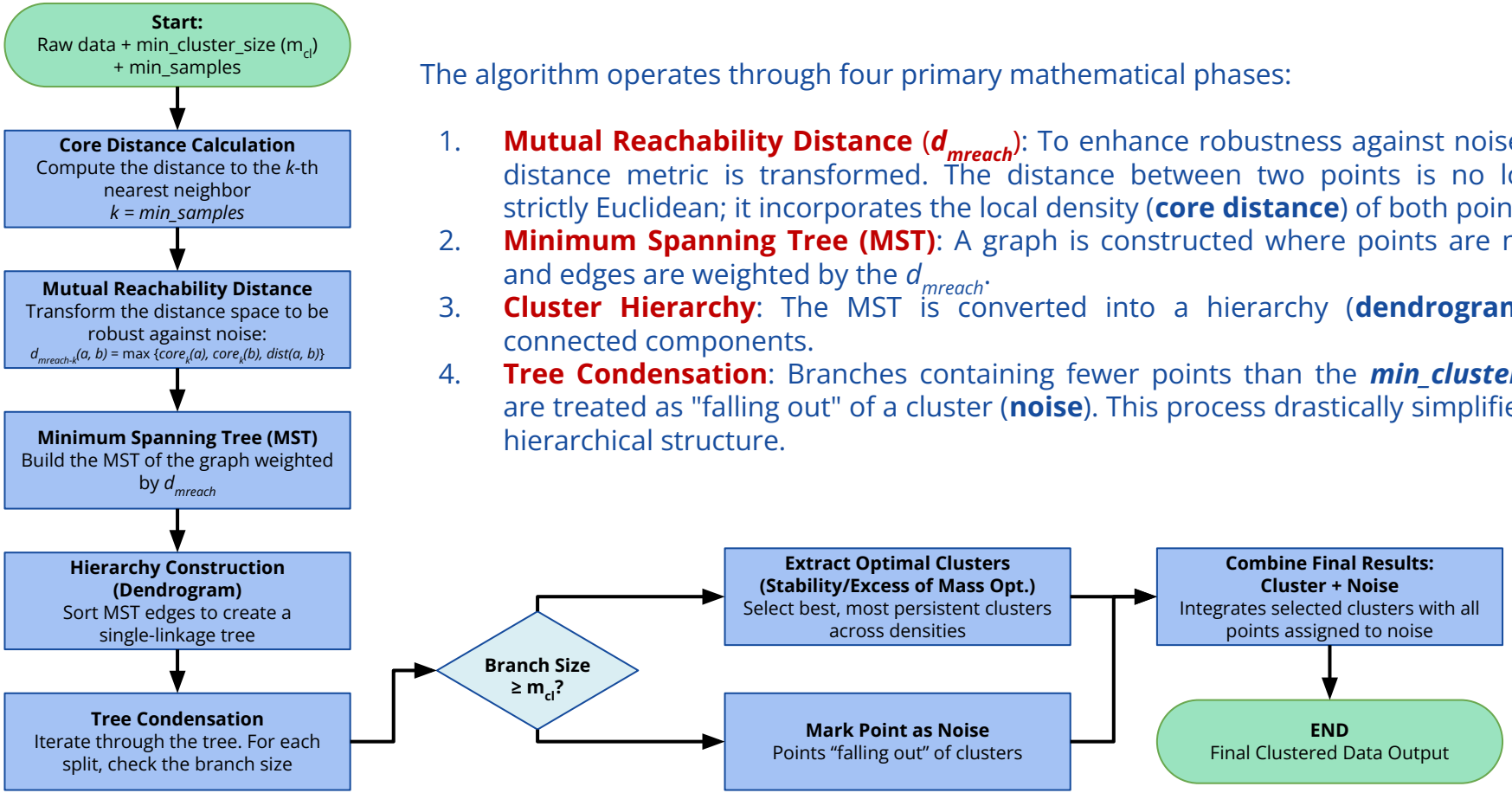
The DBSCAN Limitation: The primary drawback of DBSCAN is the reliance on a global **ε** (radius). In datasets featuring clusters with significantly different densities, a single **ε** value is insufficient: it either merges dense clusters or classifies sparse ones as noise.

The HDBSCAN Solution: HDBSCAN eliminates the requirement for a fixed **ε**. Instead of searching for clusters at a specific density level, the algorithm explores **all possible density scales simultaneously**.

Key Parameters

- **min_cluster_size** : the minimum threshold of points required to form a cluster. If a set of points is smaller than this value, it will be labeled as noise or merged with other clusters. This approach makes the algorithm more robust and less sensitive to manual parameter tuning compared to its predecessor
- **min_samples** : defines how "dense" a point must be to avoid being considered an outlier. Higher values make HDBSCAN more conservative (more points are labeled as noise).

HDBScan - Hierarchical DBSCAN



The algorithm operates through four primary mathematical phases:

1. **Mutual Reachability Distance (d_{mreach})**: To enhance robustness against noise, the distance metric is transformed. The distance between two points is no longer strictly Euclidean; it incorporates the local density (**core distance**) of both points
2. **Minimum Spanning Tree (MST)**: A graph is constructed where points are nodes and edges are weighted by the d_{mreach}.
3. **Cluster Hierarchy**: The MST is converted into a hierarchy (**dendrogram**) of connected components.
4. **Tree Condensation**: Branches containing fewer points than the ***min_cluster_size*** are treated as "falling out" of a cluster (**noise**). This process drastically simplifies the hierarchical structure.

HDBScan - Hierarchical DBSCAN

Start:
Raw data + min_cluster_size (m_{cl})
+ min_samples

↓

Core Distance Calculation
Compute the distance to the *k*-th nearest neighbor
k = min_samples

↓

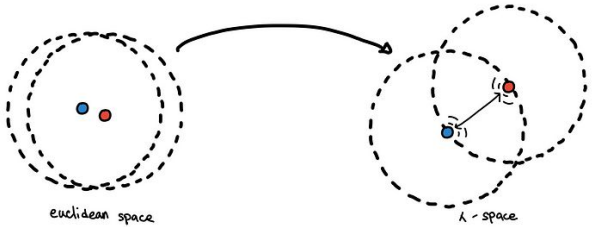
Mutual Reachability Distance
Transform the distance space to be robust against noise:
$d_{mreach\text{-}k}(a, b) = \max\{core_k(a), core_k(b), dist(a, b)\}$

The **core distance** of a point *p* is defined as the **Euclidean distance** to its *k*-th nearest neighbor, where *k* is governed by the *min_samples* parameter. This metric serves as a **proxy for local density**: lower core distance values characterize observations situated within regions of high spatial concentration, while elevated values denote areas of rarefaction, reflecting the necessity to expand the search neighborhood to fulfill the minimum proximity requirement.

The **Mutual Reachability Distance** is calculated between each pair of points (p,q):

$$d_{mreach-k}(a,b) = \max\{core_k(a), core_k(b), dist(a,b)\}$$

It implements a **metric repulsion mechanism** designed to enhance algorithmic robustness against stochastic noise. In high-density local contexts, this measure converges toward the intrinsic Euclidean distance; conversely, in sparse regions, the distance undergoes a **constrained dilation** governed by the respective core distance values. This transformation ensures the spatial segregation of outliers, thereby preserving the topological integrity of dense clusters during the aggregation process.



HDBScan - Hierarchical DBSCAN

Start:
Raw data + min_cluster_size (m_{cl}) + min_samples

↓

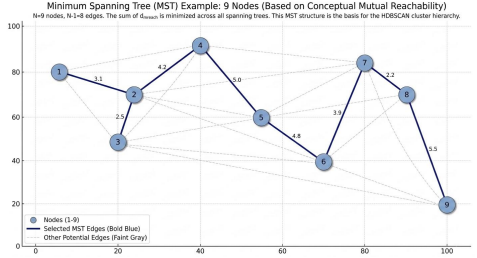
Core Distance Calculation
Compute the distance to the *k*-th nearest neighbor
k = min_samples

↓

Mutual Reachability Distance
Transform the distance space to be robust against noise:
$d_{mreach\text{-}k}(a, b) = \max\{core_k(a), core_k(b), dist(a, b)\}$

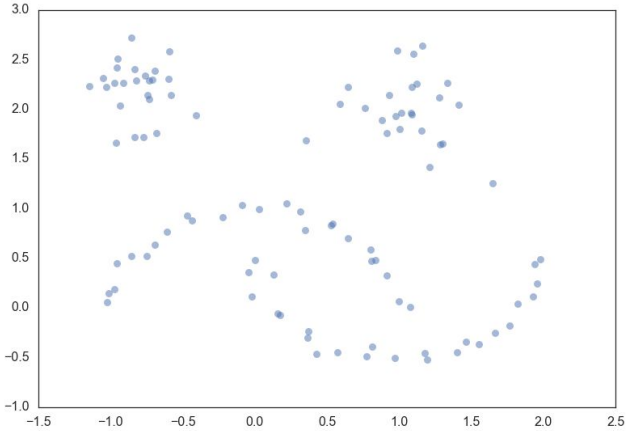
↓

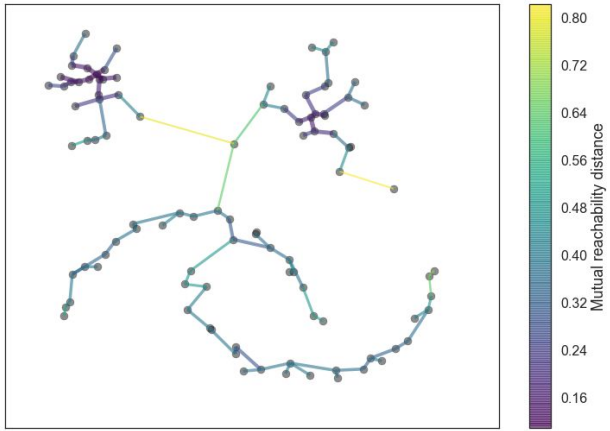
Minimum Spanning Tree (MST)
Build the MST of the graph weighted by d_{mreach}



The Minimum Spanning Tree (MST) is not constructed using standard Euclidean distances; instead, it is built upon d_{mreach}). This fundamentally alters the geometry of the tree:

- **Edge Weighting**: Each edge connecting two points, *a* and *b*, is assigned a **weight** equal to their mutual reachability distance $d_{mreach}(a, b)$.
- **Density Effect**: Since $d_{mreach}(a, b)$ incorporates core distances, the MST **prioritized connections within dense regions** (where weights are minimal) before addressing points in sparse areas.
- **Noise Resilience**: Outliers, characterized by significantly higher core distances, are "**pushed**" to the periphery of the tree with exceptionally long edges. This facilitates their identification and eventual isolation from the primary cluster structures.1





HDBScan - Hierarchical DBSCAN

Start:
Raw data + min_cluster_size (m_{cl})
+ min_samples

↓

Core Distance Calculation
Compute the distance to the *k*-th nearest neighbor
k = min_samples

↓

Mutual Reachability Distance
Transform the distance space to be robust against noise:
$d_{mreach\text{-}k}(a, b) = \max\{core_k(a), core_k(b), dist(a, b)\}$

↓

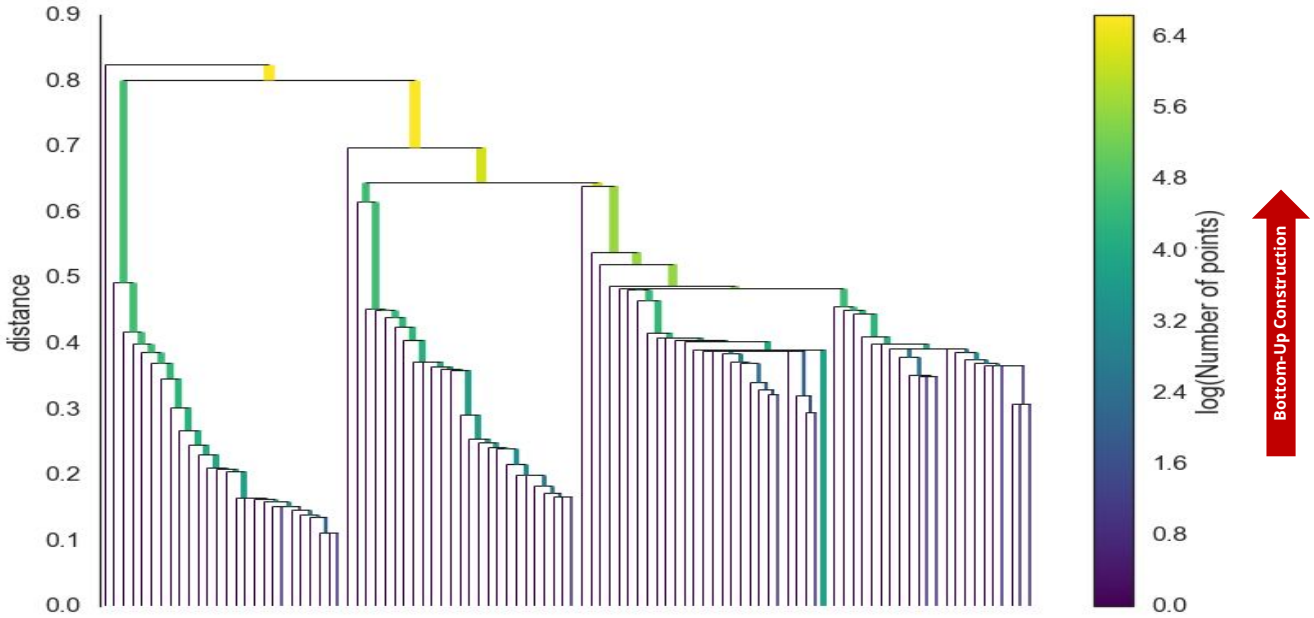
Minimum Spanning Tree (MST)
Build the MST of the graph weighted by d_{mreach}

↓

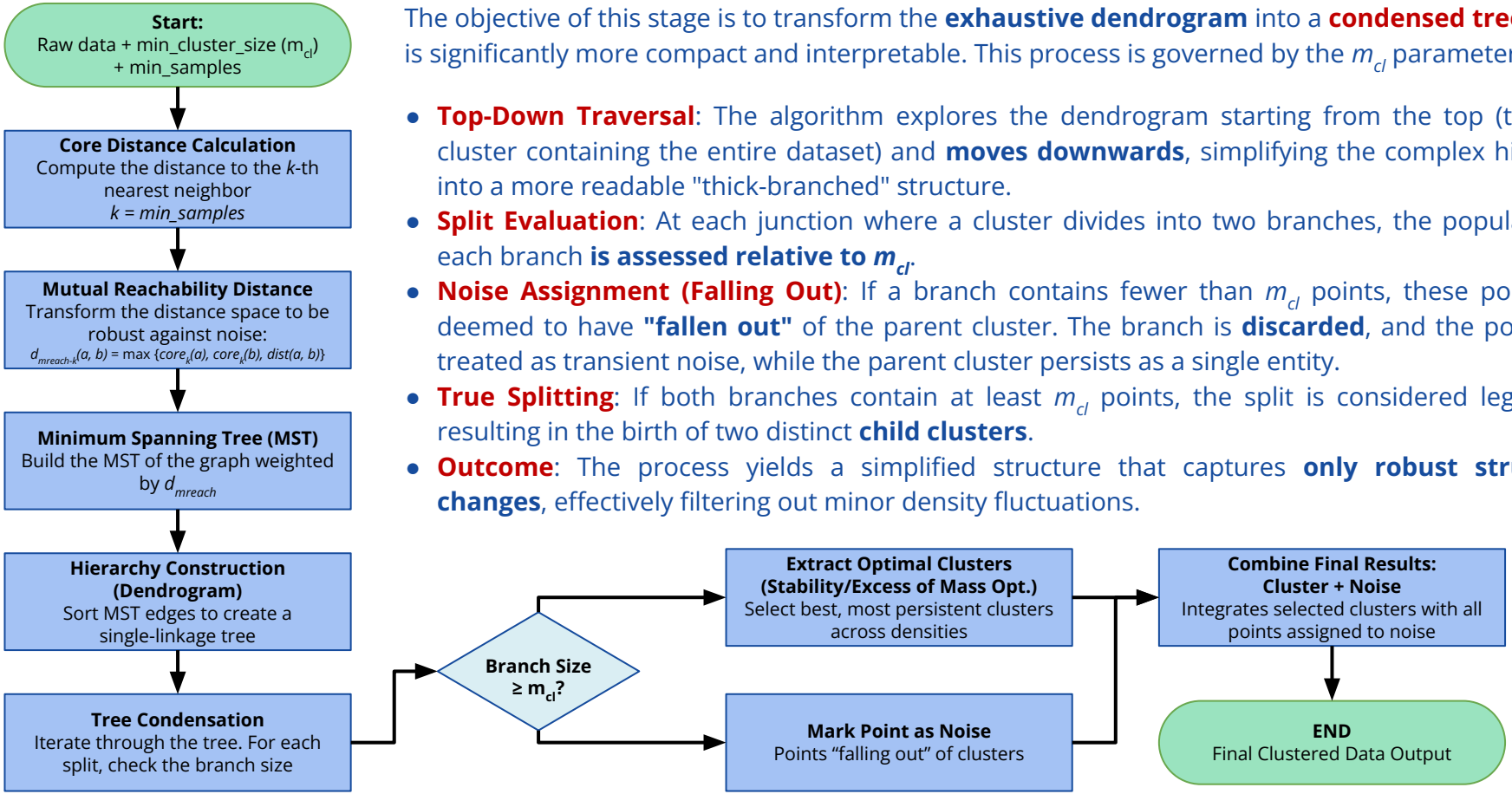
Hierarchy Construction (Dendrogram)
Sort MST edges to create a single-linkage tree

The MST serves as the foundational **precursor** to the dendrogram.

1. **Edge Sorting**: All edges within the MST are prioritized and **sorted** in ascending order of weight.
2. **Bottom-Up Construction**: The hierarchy is initiated from the shortest edges (representing the highest density/proximity), where points merge first to form the initial micro-clusters.
3. **Component Aggregation**: As edges with progressively higher weights are incorporated, these components merge until the entire dataset is unified into a single global cluster.



HDBScan - Hierarchical DBSCAN



The objective of this stage is to transform the **exhaustive dendrogram** into a **condensed tree**, which is significantly more compact and interpretable. This process is governed by the m_{cl} parameter:

- **Top-Down Traversal**: The algorithm explores the dendrogram starting from the top (the root cluster containing the entire dataset) and **moves downwards**, simplifying the complex hierarchy into a more readable "thick-branched" structure.
- **Split Evaluation**: At each junction where a cluster divides into two branches, the population of each branch **is assessed relative to m_{cl}**.
- **Noise Assignment (Falling Out)**: If a branch contains fewer than m_{cl} points, these points are deemed to have **"fallen out"** of the parent cluster. The branch is **discarded**, and the points are treated as transient noise, while the parent cluster persists as a single entity.
- **True Splitting**: If both branches contain at least m_{cl} points, the split is considered legitimate, resulting in the birth of two distinct **child clusters**.
- **Outcome**: The process yields a simplified structure that captures **only robust structural changes**, effectively filtering out minor density fluctuations.

HDBScan - Hierarchical DBSCAN

Start:
Raw data + min_cluster_size (m_{cl}) + min_samples

Core Distance Calculation
Compute the distance to the *k*-th nearest neighbor
k = min_samples

Mutual Reachability Distance
Transform the distance space to be robust against noise:
$d_{mreach\text{-}k}(a, b) = \max\{core_k(a), core_k(b), dist(a, b)\}$

Minimum Spanning Tree (MST)
Build the MST of the graph weighted by d_{mreach}

Hierarchy Construction (Dendrogram)
Sort MST edges to create a single-linkage tree

Tree Condensation
Iterate through the tree. For each split, check the branch size

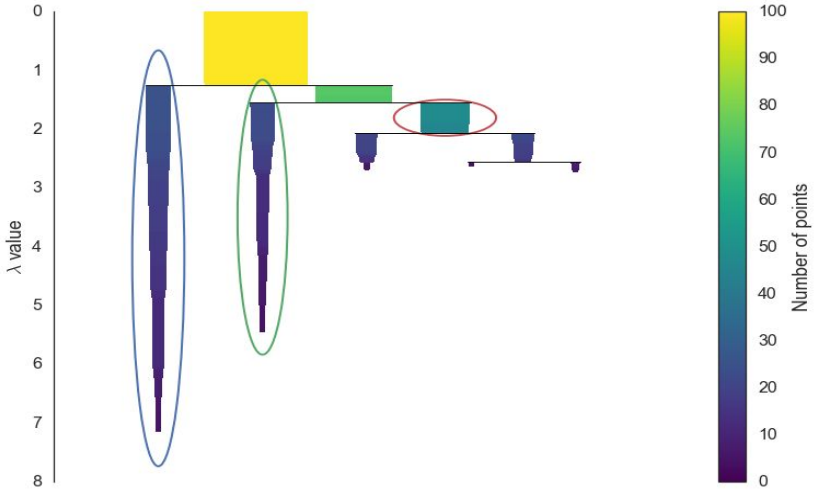
Branch Size ≥ m_{cl}?

Extract Optimal Clusters (Stability/Excess of Mass Opt.)
Select best, most persistent clusters across densities

Mark Point as Noise
Points "falling out" of clusters

Combine Final Results: Cluster + Noise
Integrates selected clusters with all points assigned to noise

END
Final Clustered Data Output



Each branch is characterized by its own **mass**.
The **Excess of Mass (EOM)** algorithm effectively calculates the area of these branches using two dimensions:

- **Base (Width)**: Represents the **number of points** belonging to the cluster at a specific density level
- **Height (λ)**: Represents the **persistence** of the group as density increases ($\lambda = 1/d_{mreach}$)

A cluster is considered "**stable**" if it manages to maintain a large population (width) over a significant density range (height).

HDBScan - Hierarchical DBSCAN

Start:
Raw data + min_cluster_size (m_{cl}) + min_samples

Core Distance Calculation
Compute the distance to the *k*-th nearest neighbor
k = min_samples

Mutual Reachability Distance
Transform the distance space to be robust against noise:
$d_{mreach\text{-}k}(a, b) = \max\{core_k(a), core_k(b), dist(a, b)\}$

Minimum Spanning Tree (MST)
Build the MST of the graph weighted by d_{mreach}

Hierarchy Construction (Dendrogram)
Sort MST edges to create a single-linkage tree

Tree Condensation
Iterate through the tree. For each split, check the branch size

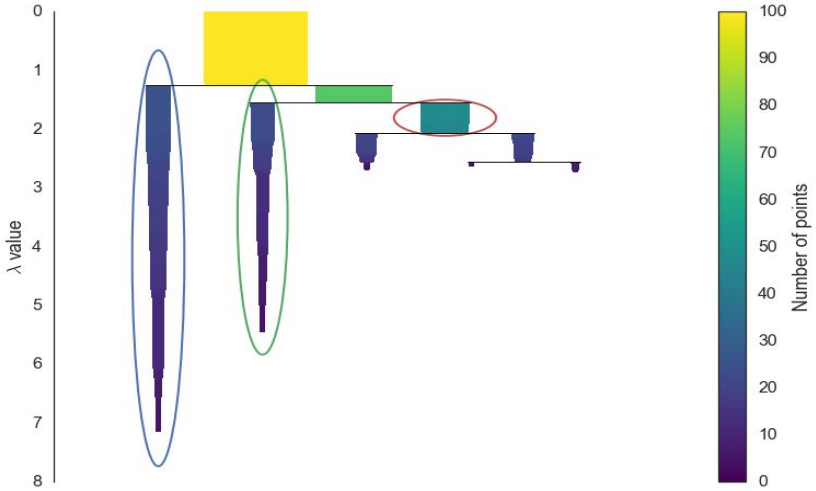
Branch Size ≥ m_{cl}?

Extract Optimal Clusters (Stability/Excess of Mass Opt.)
Select best, most persistent clusters across densities

Mark Point as Noise
Points "falling out" of clusters

Combine Final Results: Cluster + Noise
Integrates selected clusters with all points assigned to noise

END
Final Clustered Data Output



For every node in the tree, HDBSCAN compares the **Stability (S)** of the Parent cluster against the **sum of the Stabilities** of its Children:

- ***S(Parent) > ∑ S(Children)*** : The parent is deemed more stable and persistent. The algorithm **selects the parent** and "clears" or discards the children's branches.
- ***S(Parent) < ∑ S(Children)*** : The children, collectively, represent the data structure more effectively. The algorithm **retains the children** and sets the parent's stability equal to the sum of its children's stabilities for the next comparison up the tree.

HDBScan - Hierarchical DBSCAN

Start:
Raw data + min_cluster_size (m_{cl}) + min_samples

Core Distance Calculation
Compute the distance to the *k*-th nearest neighbor
k = min_samples

Mutual Reachability Distance
Transform the distance space to be robust against noise:
$d_{mreach\text{-}k}(a, b) = \max\{core_k(a), core_k(b), dist(a, b)\}$

Minimum Spanning Tree (MST)
Build the MST of the graph weighted by d_{mreach}

Hierarchy Construction (Dendrogram)
Sort MST edges to create a single-linkage tree

Tree Condensation
Iterate through the tree. For each split, check the branch size

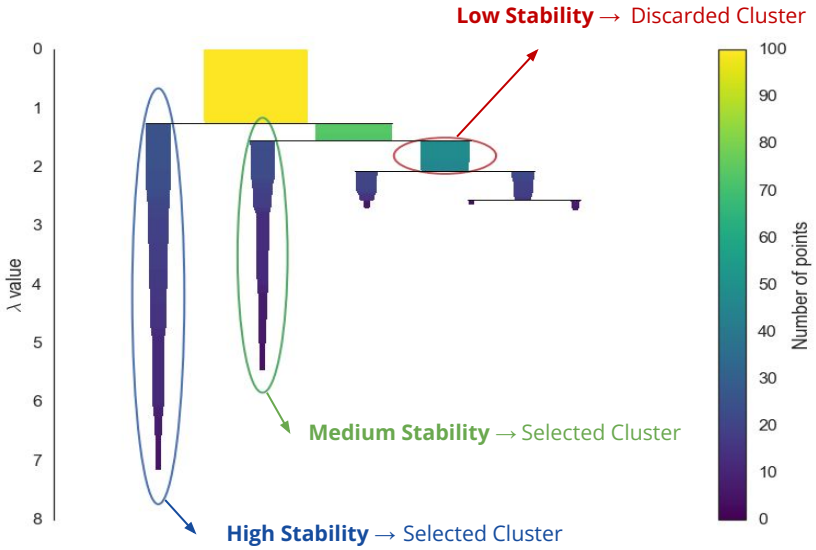
Branch Size ≥ m_{cl}?

Extract Optimal Clusters (Stability/Excess of Mass Opt.)
Select best, most persistent clusters across densities

Mark Point as Noise
Points "falling out" of clusters

Combine Final Results: Cluster + Noise
Integrates selected clusters with all points assigned to noise

END
Final Clustered Data Output



Optimization

HDBSCAN maximizes **global stability**. Using an **optimization equation**, the algorithm selects the set of non-overlapping clusters that provides the **greatest overall "area"** (persistence), automatically discarding **temporary and insignificant density fluctuations**.

HDBSCAN - Pro & Cons

PROS

- ✓ **Variable Density Detection**: The algorithm excels at identifying clusters with significantly different density levels within the same dataset. It does not require all groups to be equally compact to be recognized.
- ✓ **Reduced Arbitrary Parameterization**: Instead of relying on a fixed distance threshold, HDBSCAN uses the ***min_cluster_size*** parameter. This value is highly intuitive, as it refers to the minimum size a group must reach to be considered a relevant cluster.
- ✓ **Advanced Noise Filtering**: By utilizing **Mutual Reachability Distance**, HDBSCAN is extremely robust against **outliers**. Points in low-density areas are mathematically "pushed away" from cluster cores, preventing them from distorting the shape of the groups.
- ✓ **Hierarchical Insight**: The output is not just a flat set of labels, but a complete hierarchy (condensed tree). This allows users to understand how data is organized at different levels of detail and how clusters "emerge" or "dissolve" as density changes.
- ✓ **Mathematical Stability** : Final cluster selection is based on persistence through the **Excess of Mass (EOM)** criterion. The algorithm selects groups that "survive" the longest during the condensation process, ensuring statistically significant results.

CONS

- x **Computational and Memory Cost**: Constructing the **Minimum Spanning Tree (MST)** and subsequently condensing the hierarchy requires significant resources. For extremely large datasets (millions of records), processing time and RAM usage can become **critical**.
- x **Sensitivity to Dimensionality**: The algorithm suffers from the "**curse of dimensionality**". In spaces with hundreds of features, distances between points tend to become uniform, making it difficult to distinguish dense regions from background noise.
- x **Distance Metric Dependency**: Performance is heavily dependent on the choice of **distance metric** (e.g., Euclidean, Manhattan). An incorrect metric choice relative to the nature of the data can lead to clusters that lack logical meaning.
- x **Difficulty with "Overlapping" Clusters**: If two clusters have very similar densities and are connected by a thin "trail" of points, HDBSCAN may struggle to separate them correctly, tending to merge them into a single structure unless the split is very distinct.
- x **Interpretational Complexity**: For non-technical users, understanding concepts such as λ, condensed trees, or Excess of Mass stability can be more challenging than simpler, more linear clustering methods.

A Final Comparison

Feature	K-means	DBSCAN	HDBSCAN
Conceptual Simplicity	🟢 Very Simple (**Centroids**)	🟡 Moderate (**Density**)	🔴 Complex (**Hierarchical**)
Parameter Tuning	🟡 Moderate (***k***)	🔴 Difficult (**ε**, ***MinPts***)	🟢 Simple (m_{cl})
Robustness to Outliers	🔴 Highly sensitive	🟢 Robust (**Labels noise**)	🟢 Very Robust (**Robust** d_{mreach})
Cluster Shapes	🔴 Spherical only	🟢 Arbitrary shapes	🟢 Arbitrary shapes
Varying Densities	🔴 Fails	🔴 Fails (**Fixed ε**)	🟢 Excellent (**Adaptive**)
Data Scaling Required	🔴 Mandatory	🔴 Mandatory	🔴 Mandatory
Computational Speed	🟢 Very Fast	🟡 Moderate	🔴 Slower (**MST creation**)
Overfitting Risk	🟡 Moderate	🟡 High (**if ε is small**)	🟢 Low (**Stability Optimization**)
Manual Cutoff	🔴 Required (***k***)	🔴 Required (**ε**)	🟢 Automated (**EOM**)

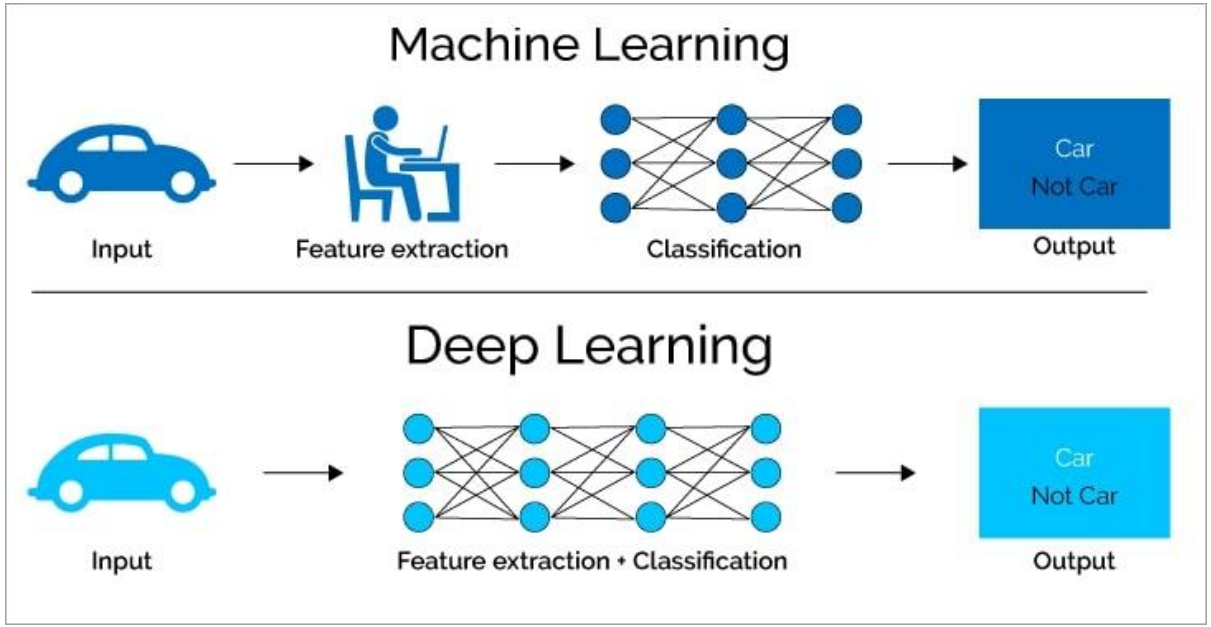
Deep Learning in a nutshell

Deep Learning vs Machine Learning

Automated Feature Extraction: In traditional ML, we define the features (e.g., color indices, Sérsic parameters). In Deep Learning, the network autonomously discovers morphological or spectral descriptors directly from raw data.

Performance Scaling: While classical algorithms reach a performance plateau, Deep Learning models keep improving as dataset sizes increase (ideal for massive surveys like LSST or Euclid).

End-to-End Learning: You don't need a pipeline of multiple disjoint tools; a single neural network maps the raw CCD signal directly to a scientific classification.

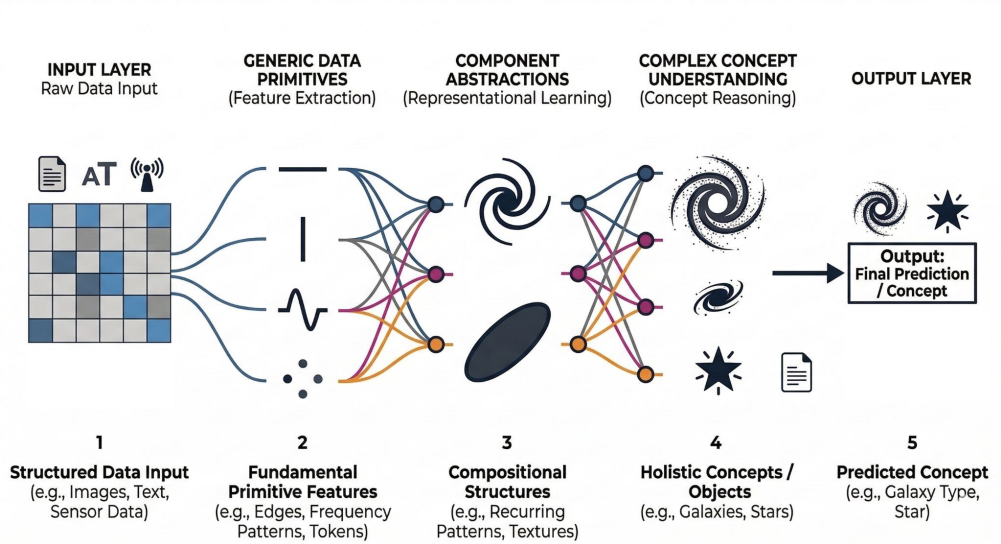


The Hierarchical Architecture

Deep Learning leverages **artificial neural networks**, computational architectures designed for high-dimensional data mapping and automated feature representation and simulating the human brain behavior. These networks consist of **stacked layers of interconnected nodes (neurons)**, where each connection applies a mathematical transformation to the incoming signal.

A **canonical neural network** is composed of an input layer, one or more hidden layers for feature extraction, and an output layer for final inference. The 'depth' of a network is defined by the number of hidden layers; increasing depth enables the model to capture **hierarchical, multi-scale features**—essential for interpreting complex astronomical datasets.

1. **Input Layer**: Receives the raw signal.
2. **Hidden Layers**: The "brain" of the network, extracting increasing levels of abstraction:
    - **Low-level layers**: Detect gradients, point sources (stars), and edges.
    - **Mid-level layers**: Combine edges into structures (spiral arms, flares, tidal tails).
    - **High-level layers**: Identify complex objects (galaxy mergers, gravitational lenses).
3. **Output Layer**: Provides the final prediction

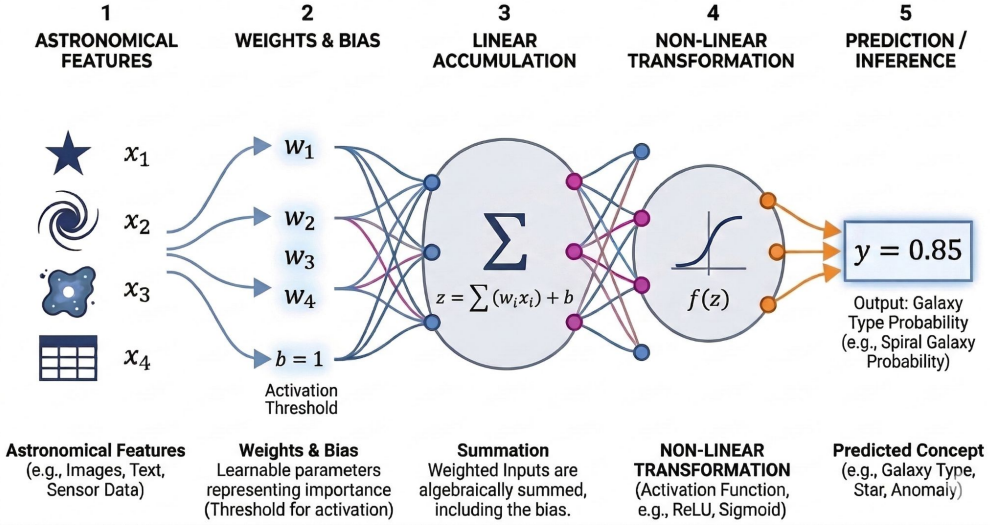


The Mathematical Unit: The Neuron

A neuron is essentially a mathematical function that **weighs the importance of incoming information**. Basically, it is a sequence of linear operations followed by a nonlinearity.

Weighted Sum: Each input ***x*** is multiplied by a Weight (***w***), representing its importance. A Bias (***b***) is added to adjust the activation threshold. They must be optimized.

Activation Function: To learn non-linear patterns (as physics is rarely linear), the output is 'filtered' by an activation function (*ReLU, Sigmoid, Tanh…*), allowing the network to decide which information is '**scientifically relevant**'. Without this phase, the network would be a simple linear equation, unable to model cosmic complexity.



From the Neuron to the Network: The Training Flow

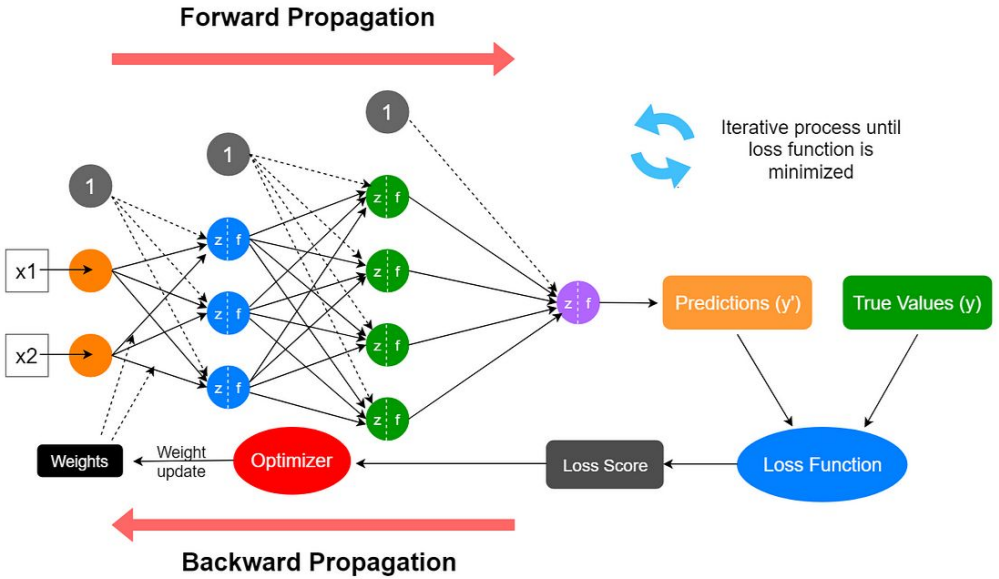
Forward Pass: Data flows through the network to produce a guess (e.g., a photometric redshift).

Loss Function: Measures the "distance" between the guess and the physical truth (**Ground Truth**). It is the numerical representation of the error. The goal is to navigate the system toward the global minimum of this error surface within a high-dimensional parameter space.

Backpropagation: The error is sent backward through the network to identify which neurons contributed most to the mistake:

1. Calculation of the partial derivative of the Loss with respect to every single weight ***∂L/∂w***
2. Once the direction of the slope (**the Gradient**) is known, the **optimizer** updates the weights with a step proportional to the slope itself, regulated by the **Learning Rate** (***η***)

$$w_{new} = w_{old} - \eta \cdot \nabla L$$



The right choice

Network Type	Architecture	Astronomical Use Cases
ANN / MLP	Multi-layer Perceptron	**Tabular catalogs**, stellar parameter estimation (e.g., Teff, logg).
CNN	Convolutional NN	**Imaging**: Morphology classification, Gravitational Lensing detection, Radio-map cleaning.
RNN / LSTM	Recurrent NN	**Time Series**: Light curve analysis, Transient classification, Pulsar timing.
Transformers	Attention-based	**Spectroscopy**: Spectral line identification, Cross-matching across large surveys.
GAN / Diffusion	Generative Models	**Simulation**: Sky Mock generation, image de-noising, and super-resolution.

CNN - Convolutional Neural Network

Convolutional Neural Networks (CNNs) are architectures specifically designed for processing **grid-like topology data**. By leveraging local filters and parameter sharing, CNNs automatically extract a hierarchy of spatial features—ranging from low-level edges to complex morphologies—ensuring **high computational efficiency and translation invariance**, both of which are fundamental properties for astronomical image analysis.

1. Convolution (Feature Extraction)

This is the core of the network. Small matrices (e.g., 3x3 or 5x5) slide across the astronomical image to detect specific patterns:

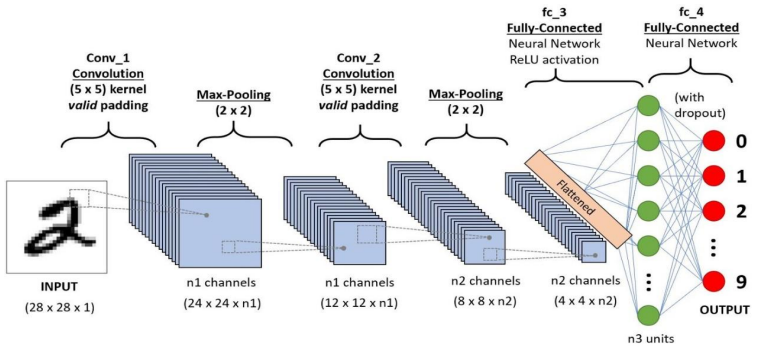
- **Initial Layers**: Detect low-level features such as edges, intensity gradients, and point sources.
- **Deep Layers**: Combine these signals to recognize complex structures like spiral arms, galactic bulges, gravitational lenses, or tidal tails.

2. Pooling (Downsampling)

Pooling layers **reduce the spatial dimensions** of the data while retaining the most salient information. This grants the network **Translation Invariance**: the model will recognize a galaxy regardless of its specific coordinates on the CCD sensor.

3. Final Classification (Fully Connected Layers)

Once the image is reduced to a vector of high-level "physical concepts," a series of **dense layers** (similar to the single neuron model) performs the final inference to determine the object's class or estimate its parameters (e.g., redshift, mass).



Why CNNs for Astronomy?

- **Spatial Hierarchy**: CNNs treat images as 2D structures, maintaining the context between neighboring pixels rather than treating them as independent variables.
- **Automatic Feature Engineering**: There is no need for manual calculation of Sersic profiles or Petrosian radii; the network "learns" the most discriminative morphological markers directly from the raw data.
- **Scalability for Big Data**: CNNs are the gold standard for massive surveys (e.g., LSST, Euclid) where manual visual inspection is no longer feasible.

Deep Learning Pros & Cons

PROs

- **Automated Feature Extraction** : This is the primary advantage. You no longer need to manually define which physical parameters are important (e.g., color indices or Sersic profiles). The network autonomously "learns" the most discriminative features directly from the raw data.
- **Universal Approximation** : Mathematically, a sufficiently deep DNN can approximate any continuous function. This makes it unrivaled at modeling highly non-linear and complex physical phenomena.
- **Scalability with Big Data** : Unlike classical models that plateau in performance as data volume increases, DNNs continue to improve as the dataset grows. This is vital for next-generation surveys like LSST (Vera Rubin) or Euclid
- **Multimodal Integration** : They can seamlessly integrate heterogeneous data sources (e.g., CCD images + tabular metadata + spectra) into a single, coherent predictive model

CONs

- **"Black Box" Nature (Interpretability)** : It is often difficult to decipher why a network reached a specific conclusion. In a scientific context where causality is paramount, this is a significant hurdle. However, techniques like SHAP values are helping to mitigate this.
- **Data Hunger** :To perform reliably, DNNs require thousands (or millions) of labeled examples. If your sample of rare astronomical objects is small, the network may fail to generalize effectively.
- **Computational Expense** : Training requires dedicated hardware (GPUs/TPUs) and significant energy consumption. It is not always the most efficient solution for simpler, low-dimensional tasks.
- **Overfitting and Fragility** : With millions of parameters, the network can easily "memorize" the noise in the training set rather than learning the underlying physics, leading to failure when applied to new, unseen data.

Final Takeaways

We have navigated the **end-to-end Machine Learning pipeline** tailored for astronomical research. While this overview couldn't be exhaustive — given the scale of the field — it aims to establish a more **conscious and rigorous framework** for our collaborative exploration

ML model should not be a “black box”
A lower loss is meaningless if we cannot extract the underlying physics. Prioritize models that allow for scientific insight.

Robustness is Key of Validation
Never rely on a single train/test split. Use rigorous cross-validation to ensure your results aren't just a statistical fluke.

Data as the Truth-Source
The model is a mirror of its training set. If the input is biased or incomplete, the model will faithfully reproduce those same flaws.

Preprocessing is the Core
In astronomy, the real discovery often happens during signal cleaning and artifact removal. Preprocessing is where domain expertise meets the algorithm.

The Scientist’s Prerogative
You are the domain expert. If a model’s output contradicts fundamental physical laws, the scientist—not the algorithm—has the final word.

Quality > Quantity
More data does not inherently mean better science. Adding noisy or poorly calibrated samples only dilutes the information bottleneck.

"Because a lower Loss function doesn't always mean better Physics…"

THANK YOU FOR YOUR ATTENTION