# AI & Machine Learning for Genomic Data Science

Master in Genomic Data Science – Università di Pavia

Barbara Tarantino

AA 2025-2026

# Course Outline

- Day 1 → Python Foundations & AI in Medicine
- Day 2 → Core Machine Learning for Genomics
- Day 3 → Deep Learning Foundations (PyTorch)
- Day 4 → Computer Vision for Medicine
- Day 5 → Large Language Models & Clinical Text

Objective: to acquire theoretical and practical bases to apply AI to genomics, clinical images, medical text.

# Program - Day 2

## Core Machine Learning for Genomics

- Principles and workflows of Machine Learning applied to genomics
- Modelli lineari supervisionati (Linear & Logistic Regression)
- Scikit-learn: pipeline, cross-validation e grid search
- Tree and ensemble models (Decision Tree, Random Forest, XGBoost)
- Model interpretability (Feature Importance, SHAP, LIME)
- Limitations of Classic ML and Introduction to Deep Learning

# Practical organization – Day 2

Morning (9:30 – 12:30)

- Genomics ML workflow (→ model data → evaluation)
- Supervised Linear Models: Linear and Logistic Regression
- Train/test split, overfitting and key metrics
- Introductory notebook: Logistic Regression on clinical dataset

Afternoon (14:30 – 17:00)

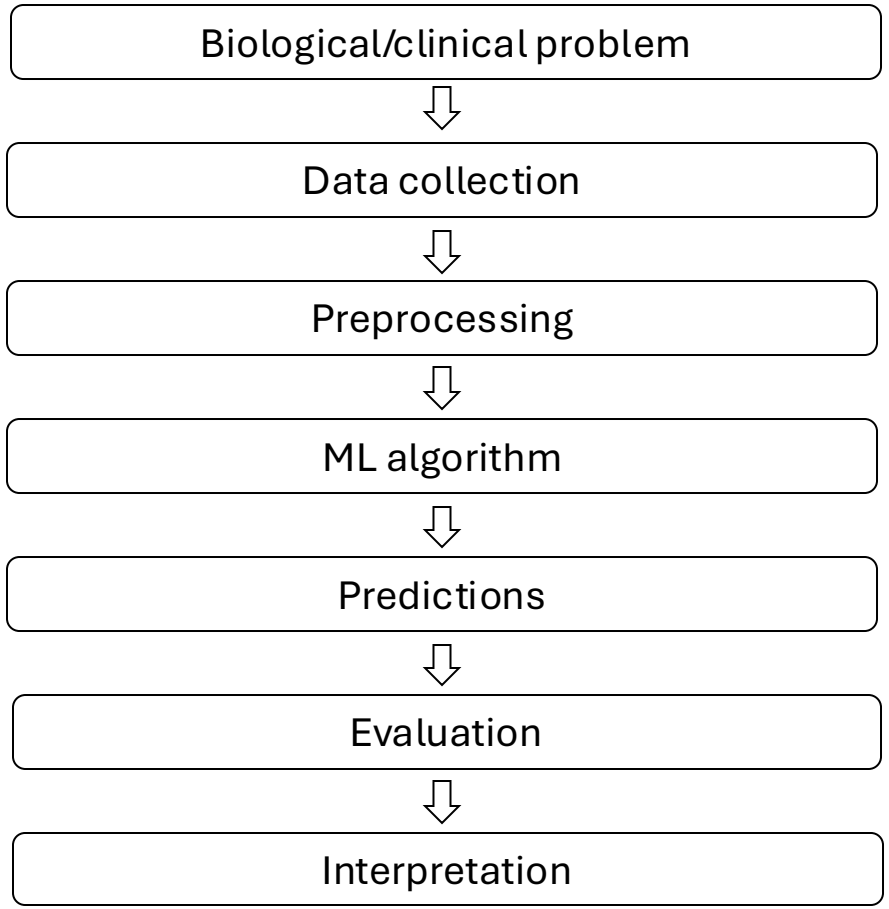
- Tree and ensemble models: Decision Tree, Random Forest, XGBoost
- Model interpretability (Feature Importance, SHAP, LIME)
- Application Notebook: Machine Learning Models Compared
- Discussion: Limitations of Classic ML and the Shift to Deep Learning

Recap + Q&A

# Section 1

Machine Learning Workflow

# Genomics ML Workflow




Clinical/biological question: e.g. predicting tumor subtype, response to therapy

Data types: clinical, genomic, image, text

Preparation steps: cleaning, normalization, train/split test

Choice of model: logistic regression, Random Forest, etc.

Output types: Classification (categories) or regression (numeric values)

Evaluation: performance metrics (accuracy, precision/recall, AUC, MSE)

Final interpretation: translation of results into useful knowledge

# Data characteristics

- High dimensionality: thousands of genes/variables, but few patients
- Noise and biological variability: individual differences and laboratory techniques influence the data
- Feature engineering: selection and transformation of variables to make data more informative
- Final goal: to build predictive models that are accurate, generalizable and clinically interpretable

# Data Types

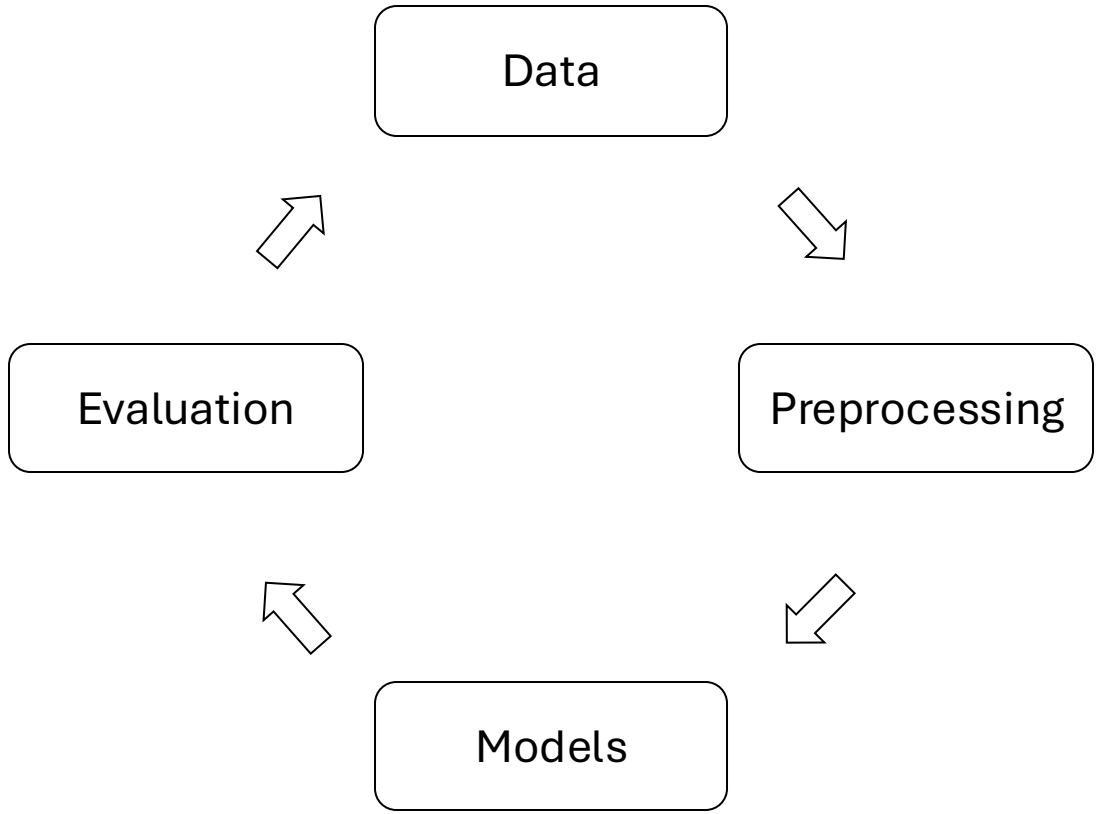
- Tabular data → numerical measurements (gene expression, genetic variants, clinical laboratory data).
- Image data → histopathology, radiology, digital medical images.
- Sequence and text data → DNA/RNA, proteins, electronic health records (EHRs).
- Multimodal data → combination of different sources (e.g. clinical + genomic + images).

# Practical challenges

- Overfitting: The model learns the training data too well and fails to generalize to new patients.
- Generalization: it is necessary to test the models on independent data to verify their robustness.
- Reproducibility: use pipelines and standard procedures (e.g. scikit-learn) to be able to replicate the results.
- Interpretability: in the clinical field it is not enough to predict, you need to understand why the model makes a certain decision.

# The Iterative ML Cycle

- Non-linear process: the data, model, and evaluation phases are repeated several times
- Continuous feedback: Results drive changes to preprocessing or model choice
- Objective: to improve robustness, avoid overfitting and increase generalization



# Objectives of ML in Genomics

- Prediction: Using available data to estimate a future risk or outcome
  (e.g., likelihood of response to a therapy)

- Classification: assign samples or patients to defined categories
  (e.g. tumor subtypes)

- Size reduction: compress high-dimensional data while keeping relevant information
  (e.g. from thousands of genes to a few principal components)

- Knowledge discovery: identifying hidden patterns or new biomarkers

# Section 2

Supervised linear models

# Supervised Learning: definition

- Supervised learning: The model learns from labeled data (, ):
  - = independent variables (input, feature)
  - = dependent variable (output, target)

- Objective: To estimate a function that connects inputs and outputs:

$$\hat{Y} = f(X; \theta)$$

- Where:
  - = model (rule learned from data)
  - = parameters of the model to be estimated

Purpose: to obtain correct predictions also on new data (generalization).

# Supervised Learning: workflow

- Data collection and preparation → input, output.
- Train set → used to estimate model parameters.
- Test set → used to evaluate its generalization.
- Loss function → measures the difference between prediction and reality.
- Optimization → search for parameters that minimize loss.
- Final output → model that can predict on new data.

# Regression

- Definition: Supervised learning task in which the output variable is continuous.
- Objective: To estimate a function that approximates the relationship between input and output:

$$\hat{Y} = f(X; \theta), \hat{Y} \in R$$

- Application examples: predicting blood pressure, the quantitative expression of a gene.
- Common methods: Linear Regression, Regression Trees.

# Classification

- Definition: Supervised learning task in which the output variable takes on discrete values.
- Objective: To estimate a function such that:

$$\hat{Y} = f(X; \theta), \hat{Y} \in \{1,2, \dots, K\}$$

- with = number of classes.
- Application examples: distinguishing healthy vs sick patients, predicting the tumor subtype (ALL vs AML).
- Common methods: Logistic Regression, Decision Trees, Random Forest.

# Linear Regression

- Definition: Predictive model for continuous output variables.

- General formula:

$$\hat{Y} = \beta_0 + \beta_1 X_1 + \cdots + \beta p X p$$

- Interpretation:
  - $\beta_0$ = intercepts → average value of when all = 0
  - $\beta_j$ = coefficient of the variable → how much it changes if it increases by 1 unit$X_j$ $X_j$

- Example: Predicting blood pressure as a function of age and BMI.

- Limit: assumes a linear relationship between inputs and outputs.

# Logistic Regression

- Definition: Model for categorical output variables (e.g., sick vs. healthy).
- Output: A probability between 0 and 1.
- Formula:

$$P(Y = 1 \mid X) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 X_1 + \cdots \beta_p X_p)}}$$

- Interpretation:
  - The linear combination of the data is transformed by a sigmoid function → "S" curve.$\beta_0 + \beta_1 X_1 + \cdots$
  - If the probability > threshold (e.g. 0.5) → the model predicts class 1 (sick).
- Example: Probability that a sample belongs to a tumor subtype given gene expression profile.

# Loss and Parameters

- A supervised model is defined by its parameters (e.g. coefficients).
- The loss function () measures the error between prediction and reality.
- Purpose: Finding minimizing loss:$\theta^*$

$$\theta^* = \arg\min_{\theta} L(\theta)$$

To do this, mathematical tools (derivatives and gradients) are needed.

# Linear Regression (analytical estimation)

- Linear regression
  - Loss: Mean *squared error* $L(\beta) = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \widehat{Y_i})^2$
  - Closed analytical solution (least squares): $\hat{\beta} = (X^T X)^{-1} X^T Y$

- Logistic regression
  - Loss: log-loss (maximum negative likelihood): , where $L(\beta)$ $= -\frac{1}{n}\sum_{i=1}^{n}[Y_i \log(\widehat{p_i}) + (1 - Y_i)\log(1 - \widehat{p_i})]$ $\widehat{p}_i = P(Y = 1 \mid X_i)$
  - No closed formulas → need iterative methods of optimization.

# Derivative in 1D

- Consider a loss function with only one parameter.$L(\theta)$
- The derivative measures the instantaneous change of as .$\frac{dL}{d\theta} L\ \theta$
- Interpretation:
  - $\frac{dL}{d\theta} > 0$→ increases if it increases (positive slope).$L\theta$
  - $\frac{dL}{d\theta} < 0$→ decreases if it increases (negative slope).$L\ \theta$
  - $\frac{dL}{d\theta} = 0$→ stationary point (minimum, maximum, or saddle).

In supervised models, the minima correspond to the optimal parameters.$L(\theta)$

# Multi-dimensional gradient

- If it depends on several parameters, partial derivatives are used.$L(\theta)\theta = (\theta_1, \ldots, \theta_p)$,

- The gradient is the vector:

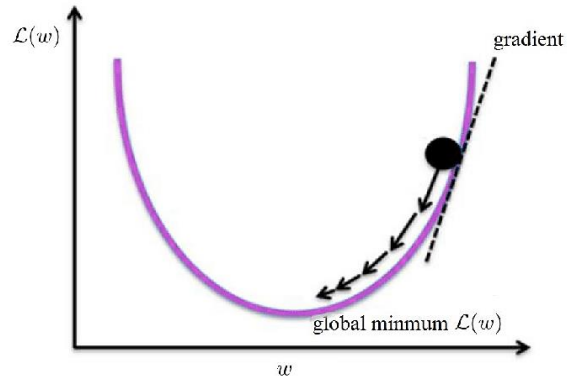
$$\nabla L(\theta) = \left(\frac{\partial L}{\partial \theta_1}, \ldots, \frac{\partial L}{\partial \theta_p}\right)$$

- Each partial derivative measures how the loss varies by changing only one parameter.
- The gradient combines all of this information in a single direction that represents the overall variation.

# Gradient Insight

- Consider a loss function (,) that depends on multiple parameters.$\theta_1\theta_2$

- We can imagine it as a 3D surface:
  - horizontal plane = parameters ($\theta_1$,$\theta_2$)
  - height = value of the loss .




- At each point, the gradient ∇():
  - is a vector that indicates the fastest growth direction of the function;
  - its length tells how steep the climb is in that direction.

- To minimize the loss, we move in the opposite direction to the gradient.

# Gradient Descent (GD)

- Idea: If the gradient indicates the steepest climb, we take steps in the opposite direction to minimize the error.
- Iterative parameter update:

$$\theta^{(t+1)} = \theta^{(t)} - \eta \nabla L\left(\theta^{(t)}\right)$$

- Where:
  - $\eta > 0$: learning rate, check the stride length.
  - $\nabla L\left(\theta^{(t)}\right)$: gradient calculated at step $.t$
- Geometric intuition: like descending a mountain → at each step you choose the steepest descent.
- Repeating the process → converges to a minimum of the loss function.

# Stochastic Gradient Descent (SGD)

- Classical GD problem: compute over the entire dataset → expensive with a lot of data.$\nabla L(\theta)$
- SGD idea: Estimate the gradient using a random sample or mini-batch.
- Update:

$$\theta^{(t+1)} = \theta^{(t)} - \eta \, \nabla L_{batch}\left(\theta^{(t)}\right)$$

- Advantages:
  - Faster computation.
  - Suitable for large datasets.
  - Stochastic noise can promote generalization.
- Note: In practice, Mini-Batch SGD (32–512 samples) is almost always used.

# From fitting to generalization

- We have seen how:
  - Build a supervised model.
  - Estimate parameters (analytical or iterative).
  - Minimize loss (GD, SGD).

- But reducing the error on training is not enough.
  A good model must generalize to new data.

- Now: evaluation techniques → Train/Test split, Overfitting, Cross-Validation.

# Train/Test Split

- Purpose: To estimate the generalization of a supervised model.

- Formal procedure:
  1. We divide the dataset ={(, )}, into two disjoint sets:$X_i Y_i i = 1, \ldots, n$
     - Training set (≈70–80%) → used to estimate parameters.$D_{train}$
     - Test sets (≈20–30%) → used only to evaluate the model.$D_{test}$
  2. The model is built exclusively by .$f(X;\theta) D_{train}$
  3. The loss is measured on the test.$L_{test}(\theta)$

- Principle: a good model must not only minimize, but keep it low.$L_{train}(\theta) L_{test}(\theta)$

# Underfitting e Overfitting

- Underfitting

Model too simple → doesn't capture the structure of the data:
$L_{train}(\theta)$ high, high$L_{test}(\theta)$

- Overfitting

Too complex model → also adapts to noise:
$L_{train}(\theta)$ low, high$L_{test}(\theta)$

- Desired equilibrium (generalization)

Model that keeps both errors low:
$L_{train}(\theta)$ low, low$L_{test}(\theta)$

# Bias–Varianza Trade-off

- Total error decomposable into:

$$\text{Errore totale} = \text{Bias}^2 + \text{Varianza} + \text{Rumore}$$

- Bias = systematic error → too high = underfitting.
- Variance = sensitivity to data → too high = overfitting.
- Noise = intrinsic variability, which cannot be eliminated.

Objective: to find the balance point that minimizes total error.

# Cross-Validation (k-fold)

- Procedure:

1. Divide into blocks.
2. For each iteration:
   - train the model on ,$D \backslash D_j$
   - Evaluate on .$D_j$
3. Average performance on test sets.

- Advantages:
  - Any data used for both training and testing.
  - A more stable rating than a single split.

- Common case: =5 or =10.

# Strategies to reduce overfitting

- Augmenting data
  - More samples → the model learns the actual structure, not the noise.
  - Techniques: collection of new data or *data augmentation*.

- Feature selection
  - Too many irrelevant variables increase the complexity of the model.
  - Eliminating predictors reduces variance and improves generalization.

- Regularization
  - A penalty term is added to the loss function:

$$L_{reg}(\theta) = L(\theta) + \lambda \cdot Pen(\theta)$$

  - Ridge (L2): penalizes → coeff. small but not zero. $\sum \theta_j^2$
  - Lasso (L1): penalizes → some coeff. exactly 0 → selection of variables. $\sum |\theta_j|$
  - Effect: simpler models and less sensitive to noise.

# Early Stopping

- In iterative methods (e.g. Gradient Descent, neural networks):
  - $L_{train}(\theta)$ it decreases continuously.
  - $L_{test}(\theta)$ it decreases at the beginning, then increases (overfitting).

- Early stopping
  - You monitor a validation set during training.
  - You stop training when it starts to grow.$L_{val}(\theta)$

- Objective
  - Stop the model at the equilibrium point → $L_{train}(\theta)\ \ basso$, $L_{test}(\theta)\ \ basso$
  - Maximize generalization capacity.

# From fitting to evaluation

- So far, we've seen how:
  - Build and train a supervised model.
  - Reduce overfitting (regularization, early stopping, cross-validation).

- Now we need a criterion to measure the performance of the model.

- Evaluation metrics allow us to compare models and understand which one generalizes best.

Next step: metrics for regression and classification.

# Valuation Metrics - Regression

- Mean Squared Error (MSE):

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

- Root Mean Squared Error (RMSE):

$$RMSE = \sqrt{MSE}$$

- Mean Absolute Error (MAE):

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|\, y_i - \hat{y}_i \,|$$

- $R^2$ (Coefficient of Determination):

$$MAE = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

# Evaluation metrics - Classification

- Accuracy

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

- Precision

$$Precision = \frac{TP}{TP + FP}$$

- Recall (Sensitivity, True Positive Rate)

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

- Specificity (True Negative Rate)

$$Specificity = \frac{TN}{TN + FP}$$

- F1-score

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

| | | **Predicted class** | |
|---|---|---|---|
| | | $P$ | $N$ |
| **Actual Class** | $P$ | True Positives (TP) | False Negatives (FN) |
| | $N$ | False Positives (FP) | True Negatives (TN) |

# Section 3

Scikit-learn

# Introduction to scikit-learn

- Open-source library for machine learning in Python.
- Simple → unified APIs (`.fit()`, `.predict()`, `.score()`).
- Consistent → same structure for all models.
- Modular → components (preprocessing, models, metrics) that can be combined.

Installation:

```
pip install scikit-learn
```

Simple import:

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
```

# Internal structure and ML workflow

| **Workflow ML** | **Modulo scikit-learn** |
|---|---|
| Dataset | `sklearn.datasets` |
| Preprocessing | `sklearn.preprocessing` |
| Model selection | `sklearn.model_selection` |
| Model | `sklearn.linear_model, tree, ensemble, svm, cluster` |
| Metrics | `sklearn.metrics` |

Each module ↔ is a step in the supervised learning cycle.

# Preprocessing

Purpose: Make the data model-friendly.

Main Classes:

- `StandardScaler` → standardizza feature (media = 0, varianza = 1):

$$x_i' = \frac{x_i - \mu}{\sigma}$$

- `MinMaxScaler` → carries values in the range: $[0, 1]$

$$x_i' = \frac{x_i - \min(x)}{\max(x) - \min(x)}$$

- `OneHotEncoder` → transforms categorical variables into binary:

$$x = "rosso" \Rightarrow [1,0,0], \qquad x = "verde" \Rightarrow [0,1,0]$$

# Preprocessing

Methods:

- `.fit(X)` → calculates transformation parameters (, min, max, categories).$\mu, \sigma$
- `.transform(X)` → applies the transform.
- `.fit_transform(X)` → both steps in one.

Attributes:

- `mean_, scale_` (per scaler).
- `categories_` (for encoding).

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train) # calculate μ and σ on train
X_test = scaler.transform(X_test) # use same values
```

# Model Selection

Purpose: Evaluate and choose robust models.

Main Classes/Functions:

- `train_test_split` → divide dataset in training e test.
- `cross_val_score` → applica k-fold cross-validation.

Methods and attributes:

- `train_test_split` → returns 4 arrays: .$(X_{train'}\, X_{test'}\, y_{train'}\, y_{test})$
- `cross_val_score:`
  - Input: model, data, number of folds.
  - Output: array of scores for each fold.

Formula CV (media score):

$$CV\_score = \frac{1}{k}\sum_{i=1}^{k} score_i$$

# Model Selection

```
from sklearn.model_selection import train_test_split, cross_val_score

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)


from sklearn.linear_model import LogisticRegression

model = LogisticRegression(max_iter=1000)

scores = cross_val_score(model, X_train, y_train, cv=5)

print(scores, scores.mean())
```

# Grid Search

- Technique for finding the best hyperparameters of a model.
- Try all combinations of values in a predefined grid.
- Use cross-validation to estimate average performance.

```
from sklearn.model_selection import GridSearchCV
GridSearchCV(estimator, param_grid, cv)
```

Methods and attributes:

- `.fit(X, y)` → trains multiple models with all combinations.
- `best_params_` → better hyperparameters.
- `best_score_` → best average CV score.
- `.predict(X)` → predictions with the best model.

Selection criterion formula:

$$best\_params = \arg\max_{\theta \in grid} CV\_score(\theta)$$

# Grid Search

```
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression

param_grid = {"C": [0.1, 1, 10]} # values to test
grid = GridSearchCV(LogisticRegression(max_iter=1000), param_grid, cv=5)

grid.fit(X_train, y_train)

print("Best parameters:", grid.best_params_)
print("Best score:", grid.best_score_)
```

# Machine Learning Models

Each algorithm is a class → becomes a model object with methods and attributes.

Main Classes:

- `LinearRegression` (linear regression)
- `LogisticRegression` (binary/multiclass classification)
- `DecisionTreeClassifier / DecisionTreeRegressor`
- `RandomForestClassifier / RandomForestRegressor`
- SVC (Support Vector Classifier) / SVR (regression with SVM)

Methods common to all models

- `tag.fit(X_train, y_train)` → trains (parameter estimation).
- `.predict(X_test)` → predictions.
- `.predict_proba(X_test)` → probability (probabilistic classification only).
- `.score(X, y)` → metrica rapida (accuracy o $R^2$).

# Machine Learning Models

Key attributes for each model

Lineari (Linear/Logistic Regression):

- `coef_` → estimated coefficients.
- `intercept_` → constant term.
- `classes_` (Logistic only) → class labels.

Alberi (DecisionTree) / Random Forest:

- `feature_importances_` → importance of variables.
- `tree_` → structure of the trained tree.

Random Forest:

- `estimators_` → list of decision trees.
- `feature_importances_` → importanza media delle feature.

# Machine Learning Models

```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)

print("Coefficients:", model.coef_)
print("Intercetta:", model.intercept_)
print("Classes:", model.classes_)
```

# Valuation metrics

Purpose: To evaluate the performance of a model.

Each function takes real vs predicted values as input and returns a numerical score.

Regression:

```
from sklearn.metrics import mean_squared_error, r2_score
mse = mean_squared_error(y_test, y_pred)
r2  = r2_score(y_test, y_pred)
```

Classification:

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
acc = accuracy_score(y_test, y_pred)
Prec = precision_score(y_test, y_pred)
rec = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
```

# Final Comparison

| **Theoretical concept (Supervised)** | **Implementation in scikit-learn** |
|---|---|
| Data $X, y$ | `numpy / pandas / sklearn.datasets` |
| Preprocessing | `StandardScaler, OneHotEncoder` |
| Train/Test split | `train_test_split(X, y)` |
| Cross-validation | `cross_val_score(model, X, y)` |
| Model definition $f_\theta$ | `model = LogisticRegression()` |
| Parameter estimation $\theta$ | `model.fit(X_train, y_train)` |
| Prediction $\hat{y}$ | `model.predict(X_test)` |
| Leak function $L(\theta)$ | `sklearn.metrics (MSE, accuracy, ecc.)` |
| General evaluation | `GridSearchCV, cross_val_score` |

# Summary exercises

- Aprire la cartella
  **D2 – Core Machine Learning for Genomics → notebooks/**
- Select
  **LogisticRegression_sklearn_Notebook.ipynb**
- Open the file with **Google Colab**.
- Run the cells with **Shift + Enter** to view results and graphs.

Colab allows you to integrate **code, text and output** in an interactive and collaborative environment directly online.

# Section 4

Tree and Ensemble Models

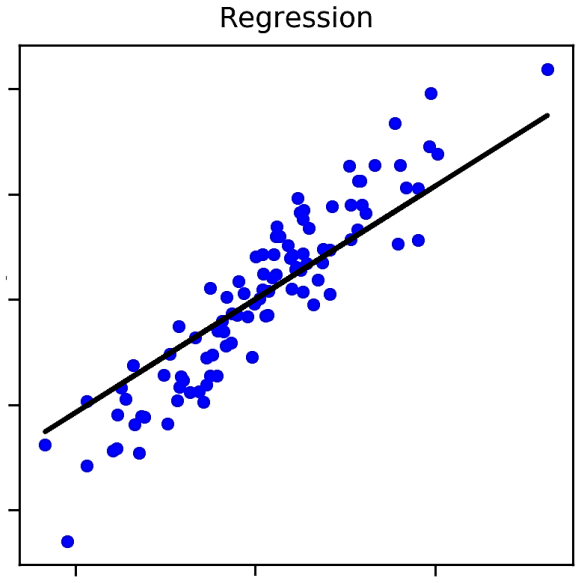
# Linear regression

- Linear regression is a model that predicts a continuous numerical value from input variables.

- The relationship is described as a linear combination:

$$\hat{Y} = \beta_0 + \beta_1 X_1 + \cdots + \beta p X p$$

- Assumption: The relationship between input and output is alway straight line in variable space.
- Limitation: in biology, relationships are rarely linear → loss of information.

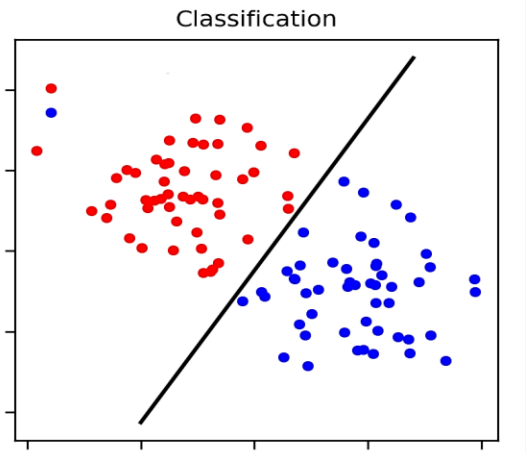


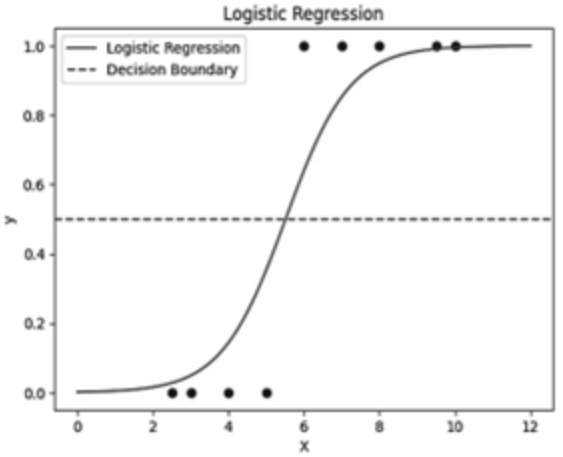
# Linear classification

- Logistic regression is a generalized linear model used for binary classification (e.g., sick vs. healthy).
- Linearly combine the input variables and transform the result into probability using the sigmoid function:

$$P(Y = 1 \mid X) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 X_1 + \cdots \beta_p X_p)}}$$

- The decision boundary is defined by:
  $= 0\beta_0 + \beta_1 X_1 + \cdots + \beta p X p$
- It means that classes are separable by a line or hyperplane.



# Limitations of Linear Models

- Linear models assume that the relationship between input and output is simple (straight or flat).

- In biological/genomic data this almost never happens:
  - A gene alone does not clearly distinguish two conditions.
  - Diseases are multifactorial (multiple genes and environmental factors together).

- Result: linearity leads to simple but inaccurate models, both for classification and regression.

# Limitations of Linear Models

- Complex interactions: The contribution of one gene may depend on the level of another gene → nonlinear relationship.
- Threshold effects: a gene affects the phenotype only if it exceeds a certain expression.
- Biological noise: individual and technical variability makes the data less regular.
- These phenomena cannot be captured by a linear combination of variables.

# From linear to nonlinear

- In summary:
  - Regression: linear model → prediction line; Nonlinear regression → more flexible curves that follow the data better.
  - Classification: linear problem → classes separable with a line; nonlinear problem → need a curved boundary.
- In genomic/clinical data, the nonlinear case almost always prevails.

- Formally:
  - Linear case → exists such that: = $0\beta_0 + \beta_1 X_1 + \cdots + \beta p X p$
  - Nonlinear case → there is no such linear combination.

- We need a model capable of representing non-linear relationships.

# Beyond linear models: flexible frontiers

- In general, the input-output relationship may require more complex functions:

$$f(x_1, x_2, \dots, x_p) = 0$$

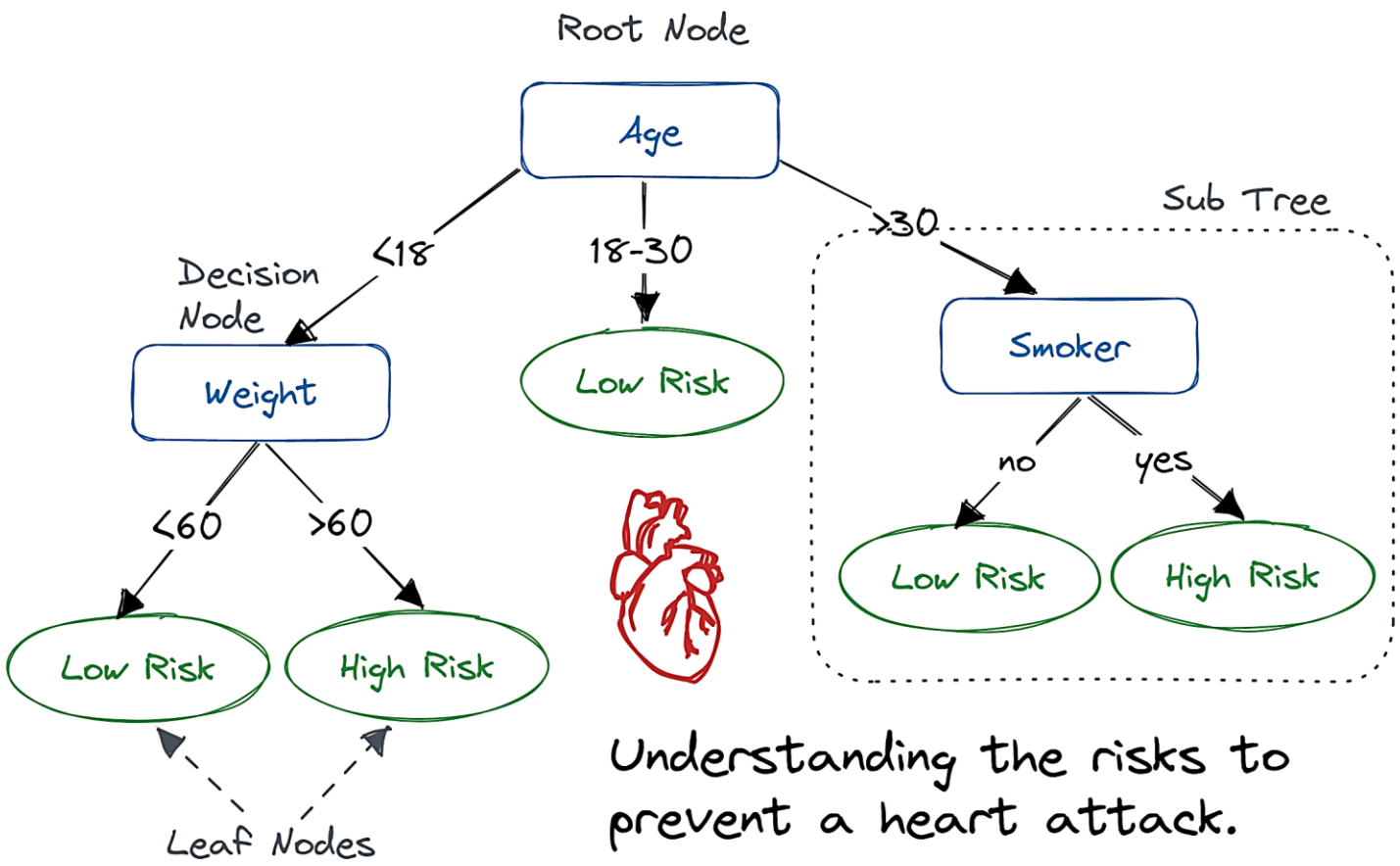
- Examples of nonlinear models:
  - Decision Trees → divide the data space into sub-regions based on variable values.
  - Random Forest → combine many independent trees (bagging) to achieve more stable and robust borders.
  - Boosting (AdaBoost, Gradient Boosting, XGBoost, LightGBM, CatBoost) → build trees sequentially, each one optimizes the errors of the previous one.

- These methods allow you to represent:
  - Classification → non-linear decision boundaries.
  - Regression → nonlinear prediction curves.
- Result: ability to capture the complex interactions typical of genomics and medicine.

# Decision Tree: concetto base

- Decision trees are non-linear models that progressively subdivide the data space.
- Each node of the tree represents a condition on a variable (e.g. value greater than or less than a threshold).
- The data is divided into increasingly homogeneous subsets.
- The terminal nodes (leaves) contain the final prediction:
  - Class → in classification.
  - Mean value → in regression.

# Decision Tree: concetto base



# Decision Tree: How it chooses divisions

- A decision tree must decide how to divide the data at each node.
- The goal is to obtain nodes that contain samples that are as homogeneous as possible.
- Key concept: The quality of a division depends on the order/purity level of the resulting nodes.
- The more "pure" the child nodes are (i.e. they almost all belong to the same class), the better the division.

# Decision Tree: Node Purity Measurements

- A node contains a number of samples belonging to different classes.
- To decide whether a node is "good," you measure how homogeneous (pure) or mixed (impure) the classes within it are.
- The purity measurement depends on the type of problem:

**Regression**

- There are no classes → measuring the variance of values: $Var(node) = \frac{1}{N} \sum_{i=1}^{N} (y_i - \bar{y})^2$
- A knot is pure if all values are equal.
- Variance → low if the values are similar, high if very dispersed.

# Decision Tree: Node Purity Measurements

**Classification**

- You use functions based on the proportions of the classes:
  - Let = number of classes.$C$
  - Let = proportion of samples of the class in the node:$p_c c$ $p_c = \frac{n.\ campioni\ della\ classe\ c}{n.\ totale\ campioni\ nel\ nodo}$

- Gini index: $G(p) = 1 - \sum_{c=1}^{C} p_c^2$
- Entropy: $\mathrm{H}(p) = -\sum_{c=1}^{C} p_c \log(p_c)$

- Gini/Entropy → low if the node is dominated by a single class.
- High if the classes are mixed.

# Decision Tree: Split Choice

- At each node, the tree must decide which variable and threshold to use to divide the data.
- Objective: to obtain child nodes that are more homogeneous than the parent node. Procedure:

1. For each candidate variable and threshold, → a measure of impurities is calculated.
   - Classification: Gini or Entropy.
   - Regression: Variance of values.
2. The impurity of the child nodes is compared with that of the parent node.
3. You choose the split that leads to the greatest improvement.

$$Information\ Gain\ (IG) = H(nodo\ padre) - \sum_i \frac{N_i}{N} H(nodo\ figlio_i)$$

- $N_i$= number of samples in the child node, = total in the parent node, H(·) = measure of impurities (Gini/Entropy by classification, Variance by regression).$N$
- If the child nodes are more homogeneous than the parent node → is high → the split is considered good.

# Decision Tree Limits

- Overfitting: Shafts that are too deep adapt to the noise of the training set.
- Instability: Small variations in the data generate very different structures.
- Structural bias: preference for variables with numerous possible splits.
- Poor generalization skills: useful for introducing non-linearities, but weak on complex problems.

Conclusion: Decision Trees are simple and interpretable, but not very robust → motivate the use of ensemble methods (Random Forest, Boosting).

# Beyond individual trees: ensemble

- A single tree is a high-variance model: strong sensitivity to training data.
- Ensemble methods reduce variance by combining multiple base models.
- General definition:

$$\hat{f}(x) = \frac{1}{M} \sum_{m=1}^{M} f_m(x)$$

- where are the individual predictors (trees). $f_m(x)$
- Two main approaches:
  - Bagging: independent predictors, trained in parallel on bootstrap (*Random Forest*) samples.
  - Boosting: Sequential predictors, each built to reduce the residual error of the previous ones.

Key concept: The combination of weak models produces a more accurate and stable predictor.

# Random Forest: general idea

- A *Random Forest* is a set of decision trees trained on different versions of the dataset.
- Si basa sul principio del bagging (bootstrap aggregating):
  - random subsets are extracted from the original dataset (with replacement);$B$
  - each generates an independent tree.
- Final prediction = average or grade of the trees:

$$\hat{f}(x) = \frac{1}{B}\sum_{b=1}^{B} f_b\,(x)$$

- where is the prediction of the tree .$f_b(x)b$
- Effect: Reduced variance and increased stability.

# Random Forest: specific operation

- In addition to bagging, Random Forest introduces randomness in variables:
  - At each split, only randomly chosen variables are considered.$m < p$
  - This reduces the correlation between trees and increases diversity.

Full steps:

1. Extract a bootstrap sample of the dataset.
2. Build a decision tree on the sample.
3. At each split only random variables → be evaluated.$m$
4. Repeat for trees.$B$
5. Aggregate predictions:
   - Classification → majority vote.
   - Regression → average of values.

# Final prediction (aggregation)

- Classification: each tree produces a class; Random Forest chooses the class with the most votes (majority vote).

$$\hat{C}(x) = arg\max_{c} \sum_{b=1}^{B} I(T_b(x) = c)$$

where is the indicator function.$I(\cdot)$

- Regression: each tree produces a numerical value; the Random Forest calculates the average.

$$\hat{f}(x) = \frac{1}{B} \sum_{b=1}^{B} T_b\,(x)$$

Key idea: The combination reduces variance and makes the model more stable.

# Random Forest in Python (scikit-learn)

In Python, la Random Forest è implementata in scikit-learn (`sklearn.ensemble`):

- `RandomForestClassifier` for Classification
- `RandomForestRegressor` for Regression

Main Parameters:

- `n_estimators`: Number of trees
- `max_features`: number of variables considered at each split (default = in classification, in regression).$\sqrt{p}p/3$
- `max_depth`: maximum depth of the tree.
- `random_state`: Status for reproducibility.

# Random Forest in Python (scikit-learn)

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import RandomForestClassifier

# Random Forest Model (regression)
rf_reg = RandomForestRegressor(
    n_estimators=100,
    max_features="sqrt",
    random_state=42
)

# Random Forest Model (Classification)
rf_class = RandomForestClassifier(
      n_estimators=100,
      max_features="sqrt",
      random_state=42
)
```

# Bagging vs Boosting

- Bagging (Bootstrap Aggregating):
  - Independent models are trained on bootstrap samples (with replacement).
  - Aggregate prediction (mean by regression, grade by classification).
  - Objective: Reduce variance → more stable estimates.

- Boosting:
  - Models trained sequentially.
  - Each new model corrects the errors of the previous one.
  - Goal: Reduce bias → more accurate model.

Bagging = stability (variance↓).

Boosting = iterative correction (bias↓).

# Boosting: general formulation

- Objective: to build an additive model as the sum of weak models:

$$F_M(x) = \sum_{m=1}^{M} \gamma_m \, h_m(x)$$

- Where:
  - $h_m(x)$= weak pattern (shallow tree).
  - $\gamma_m$= weight of the model, which measures how much each tree corrects the errors of the previous ones

- Weights are chosen to minimize the loss function (e.g., log-loss), → trees that improve the prediction the most receive higher weights.$\gamma_m$

- Difference from bagging: the models are not independent, but built in sequence to progressively improve prediction.

# Gradient Boosting: Intuition

- Residuals = residual errors of the current model.
- The new tree tries to predict these mistakes.
- The update follows the direction of the loss gradient (a concept already seen in Gradient Descent).
- It works by both regression and classification:
  - Regression → MSE, MAE.
  - Classification → log-loss, binomial deviance.

Intuition: each tree corrects the errors of the previous one → progressive improvement until convergence.

# Modern Boosting: Top Libraries

- Gradient Boosting (scikit-learn)
  - Basic implementation → teaching and small/medium datasets.
  - `sklearn.ensemble.GradientBoostingClassifier`
  - `sklearn.ensemble.GradientBoostingRegressor`

- XGBoost (Extreme Gradient Boosting)
  - Optimized and regularized version.
  - High accuracy, widely used in research and competitions.
  - `xgboost. XGBClassifier`
  - `xgboost. XGBRegressor`

# Ensemble methods: comparison

- Decision Tree
  - Simple, interpretable.
  - Easy overfitting, low stability.
- Random Forest (Bagging)
  - Many independent shafts on bootstrap samples.
  - Reduces variance → more stable.
- Boosting (Gradient Boosting, XGBoost)
  - Trees in sequence → each one corrects the errors of the previous ones.
  - Reduces bias → more accurate.

Both (RF and Boosting) turn weak shafts into strong patterns, but differently.

# Section 5

Interpretability of Models

# Feature Importance classica

- In tree models, each variable can be evaluated based on its contribution to the quality of the splits.

- Most common methods:
  - Mean Decrease in Impurity (MDI)
    - Importance = average reduction of impurities (Gini, entropy, MSE) brought by the splits that use that variable.
  - Mean Decrease in Accuracy (MDA)
    - Importance = loss of accuracy if the variable is excluded/permuted.

# Feature Importance classica

- Importance values are normalized:

$$FI_j = \frac{\sum_{t \in split(X_j)} \Delta L_t}{\sum_{k=1}^{p} \sum_{t \in split(X_j)} \Delta L_t}$$

Where:

- $t \in split(x_j)$→ all tree splits that use the variable $x_j$
- $\Delta L_t$→ reduction of the loss function (e.g. Gini, entropy, MSE) due to the split function
- $p$→ total number of features in the model

In practice, it measures how much the variable contributes, on average, to reducing impurities or improving the accuracy of the model compared to the others.$FI_j x_j$

# Classic Feature Importance: Interpretation and Limitations

Interpretation

- Variables with higher FI → more contribution to splits.
- Useful for understanding which features drive predictions.
- In genomics→ identifies relevant genes/variants.

Limits

- Bias towards variables with high cardinality (many distinct values → inflated importance).
- Dataset dependent (changes with samples/variables).
- Only global → does not explain decisions on individual patients/cases.

*Classical FI = useful but limited → modern XAI methods are needed.*

# Beyond feature importance: XAI

- XAI = Explainable AI → methods to make complex models (RF, Boosting, DL) interpretable.

Objectives

- Global → understand which variables matter most *on average* on the dataset.
- Local → explain *why* the model made a certain prediction on a single sample/patient.
- Reliability → increase trust and scientific validation in biomedical contexts.

Why it is needed in genomics

- Identify key genes/variants.
- Explain individual risk (patient-specific predictions).

# Modern XAI Methods

- Permutation Importance → assesses the degradation in performance if the values of a variable are mixed.
- Partial Dependence Plots (PDPs) → show the mean relationship between a variable and the prediction.
- SHAP (Shapley Additive exPlanations) → distributes the output among variables based on their contribution (game theory).
- LIME (Local Interpretable Model-agnostic Explanations) → creates a simple local model to explain a single prediction.

SHAP and LIME are the most popular methods for ensemble and deep learning models today.

# SHAP: Theory and Intuition

- SHAP (SHAP (*SHapley Additive exPlanations*) measures **how much each variable contributes to the prediction** of the model, **for each individual observation**.
- So **SHAP values are calculated observation by observation**, and then they can be averaged to obtain a global view.

$$\phi_j(x_i) = \sum_{S \subseteq F \setminus \{j\}} \frac{|S|!(p-|S|-1)!}{p!} \left[ f_{S \cup \{j\}}(x_i) - f_S(x_i) \right]$$

- $x_i$: the observation (sample) for which we are explaining the prediction.
- $S$: A set of features that are partially used by the model.
- $f_S(x_i)$: Prediction using only features in .$S$
- $f_{S \cup \{j\}}(x_i)$: Prediction by adding the .$X_j$
- $\phi_j(x_i)$: average contribution of the variable to the prediction for $X_j$*that specific observation*.

# SHAP: Theory and Intuition

**Interpretation:**

1. **Local:** SHAP explains *why* the model made that single prediction

- Each observation gets **its own set of SHAP values** → *local explanation*.
  - $\phi_j(x_i) > 0$: the variable increases the probability (e.g. more risk).
  - $\phi_j(x_i) < 0$: the variable reduces it (e.g. less risk).

**2. Global: By** aggregating the SHAP values, we understand *which features matter most* in the overall model.

- By **averaging the absolute values** of each variable over all observations
  → we obtain *the global importance* of the features.

# LIME: theory and intuition

- LIME (*Local Interpretable Model-agnostic Explanations*) explains **a single prediction** by constructing a small, simple model that mimics the behavior of the complex model near that point.

$$\min_{g \in G} L(f, g, \pi_x) + \Omega(g)$$

- $f$: A complex model to explain.
- $g$: local and interpretable model (e.g. linear regression).
- $L(f, g, \pi_x)$: error between the predictions of and , with the points $f g \pi_x$ **close to** $x$.
- $\Omega(g)$: Penalties for keeping $g$ **it simple and readable**.

**Procedure:**

- Similar **points are** $x$ created to (small variations).
- The predictions of the complex model are calculated.
- He trains to understand $g$ **which features matter most** around $.x$

**Result:**
LIME shows the features that have most influenced *that specific prediction*.

# SHAP vs LIME

| Aspect | SHAP (SHapley Additive exPlanations) | LIME (Local Interpretable Model-agnostic Explanations) |
|---|---|---|
| **Objective** | It measures the average contribution of each variable to the prediction. | Estimate which features affect a single prediction. |
| **Unit of analysis** | Calculated **for each observation** → each sample has its own SHAP values. | Analyze **one prediction at a time** with a local model. |
| **Theoretical principle** | Based on **game theory** (Shapley values). | Based on **local approximation** of the complex model. |
| **Method** | Compare the prediction with and without each variable. | Generate points close to and train an interpretable model.$x\ g$ |
| **Type of explanation** | **Local: Explains** a single prediction.<br>**Global: Average** of SHAP values → overall importance. | **Local only: Explain** the decision for a single point. |
| **Interpretation** | $\phi_j > 0$: increases prediction (e.g. more risk).<br>$\phi_j < 0$: reduces it (e.g. less risk). | The coefficients of show the effect of features around $.gx$ |
| **Advantages** | Coherent and additive explanations.<br>Both local and global vision. | Simpler and faster; Useful for quick analysis. |
| **Limits** | More computationally expensive on complex models. | Less stable: Changes if local data changes. |

SHAP = more complete (global + local, robust) - LIME = simpler and faster (local only).

# Interpretability in Python

- **Feature importance classica**
  - RF/GBM: attributo `.feature_importances_` in scikit-learn.

```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier().fit(X, y)
rf.feature_importances_
```

- **Permutation Importance**
  - `sklearn.inspection.permutation_importance`
- **Partial Dependence Plots (PDP)**
  - `sklearn.inspection.PartialDependenceDisplay`
- **SHAP**
  - `shap → shap. TreeExplainer(rf).shap_values(X)`
- **LIME**
  - `lime → lime.lime_tabular. LimeTabularExplainer`

# Section 6

From the limits of ML to Deep Learning

# Limitations of classic ML

- Manual feature engineering → often requires hand-extracting features from raw data.
- Limited capacity → classical models (RF, boosting) do not easily capture very complex interactions.
- Scalability → difficulty in adapting to very large or unstructured data (images, genomic sequences, texts).
- Generalization → strong dependence on feature selection and preprocessing.

# Why Deep Learning Is Needed

- Machine learning of features → DL builds representations directly from data.
- Highly flexible models → neural networks approximate very complex nonlinear functions.
- Adaptability to heterogeneous data → images, sequences, text, multi-omics data.
- Improved current performance (state of the art) in many biomedical fields → imaging diagnosis, sequence prediction, clinical NLP.

# Key Differences: Classic ML vs DL

| **Aspect** | **ML classico (RF, Boosting, etc.)** | **Deep Learning (NN, CNN, Transformers, etc.)** |
|---|---|---|
| Feature extraction | Handbook, expert-based | Automatica (feature learning) |
| Modeling skills | Limited, interpretable models | High, complex models |
| Data type | Structured (tabular) | Structured + unstructured (images, sequences, text) |
| Required data | Medium-small | Large datasets needed |
| Interpretability | Major | More difficult (black-box) |

# Towards Deep Learning in Genomics

1. Genomic data = high dimensionality (thousands of genes/variants).
2. Nonlinear and complex relationships between genes, environment and phenotype.
3. DL as a natural extension:
    1. Neural networks → capture nonlinear relationships.
    2. CNN → local patterns (DNA sequences, histological images).
    3. RNNs/Transformers → long-range addictions (gene sequences, medical records).

Deep learning does not replace classic ML, but complements it when data is complex and abundant.

# Summary exercises

- Aprire la cartella
  **D2 – Core Machine Learning for Genomics → notebooks/**
- Select
  **MLmodels_Notebook.ipynb**
- Open the file with **Google Colab**.
- Run the cells with **Shift + Enter** to view results and graphs.

Colab allows you to integrate **code, text and output** in an interactive and collaborative environment directly online.

# Conclusion Day 2

Today, we introduced the fundamental principles of Machine Learning applied to genomics:

- Linear models (regression and classification).
- Decision trees and ensemble models (Random Forest, Boosting).
- Evaluation and interpretability metrics (Feature Importance, XAI: SHAP and LIME).

We discussed the limitations of classical ML and the shift towards Deep Learning, which we will explore in the next meeting (Day 3).

See you tomorrow!