Lecture notes on:

Machine Learning for Physicists

"32nd Vietnam School of Physics: Particles and Dark Matter"

Bryan Zaldívar
Institute for Corpuscular Physics
Valencia, Spain.

July 2026

Contents

1 Introduction

What is machine learning?

A useful working definition is:

Machine learning is the problem of optimising over a space of functions or functionals in order to satisfy constraints imposed by data, equations, or physical principles.

This optimisation is usually associated with machine learning when it involves flexible parameterisations and iterative optimisation methods, for example neural networks, decision trees, or related models.

Machine-learning methods are not restricted to problems involving tabular datasets. They may also be used for:

- solving differential equations;
- learning or emulating expensive simulations;
- finding the ground state of a Hamiltonian;
- learning control strategies or policies;
- discovering low-dimensional structure in complex systems.

There are different ML paradigms. The most prominent ones are:

1.1 Supervised learning

In supervised learning, one is given a dataset

$$\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{N},$$

where each input $\boldsymbol{x}_i$ is accompanied by a target or label y_i.

The two most common supervised-learning tasks are (see Fig. 1):

Regression. The target y is continuous. Examples in physics include:

- regressing a particle or jet energy from detector-level observables;
- reconstructing the incident angle or momentum of a cosmic ray from an air shower;
- estimating a background contribution in a signal region from control-region data.

Classification. The target is categorical. Examples include:

- classifying different astrophysical source populations;
- separating signal from background events;
- identifying jet flavour or event topology at a collider.

The term *supervised* refers to the fact that the desired output is known for every training example.

1.2 Unsupervised learning

In unsupervised learning, the dataset contains inputs only:

$$\mathcal{D} = \{\boldsymbol{x}_i\}_{i=1}^{N}.$$

Typical tasks include:

- Clustering. The idea is to cluster the dataset into a predefined number of clusters. But the latter can be optimised as well.

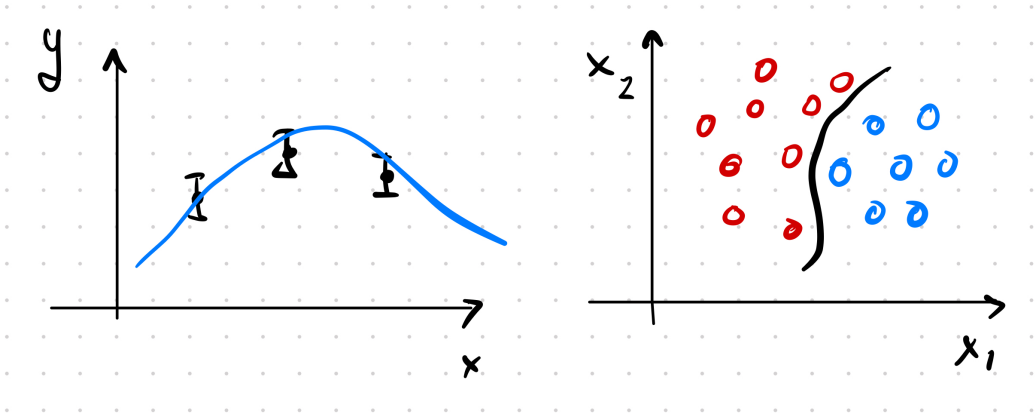

Figure 1: Supervised learning tasks: Regression (left) and Classification (right).

- Probability Density Estimation. Given a cloud of points, what is the value of the probability (density) at a given location?
- Dimensionality Reduction. Recognizing that the data may live in a lower-dimensional manifold (not necessarily linear), and transform the coordinate system such that unimportant dimensions may be discarded.

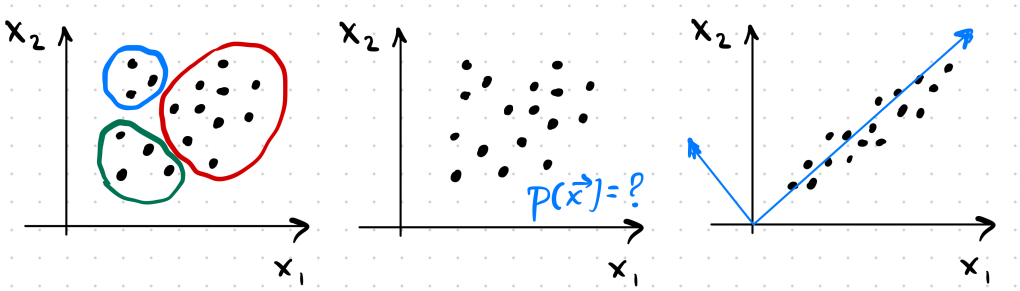

Figure 2: Unsupervised learning tasks: Clustering (left), Probability Density Estimation (centre), and Dimensionality Reduction (right).

Examples in physics include:

- estimating the distribution of collider backgrounds or astrophysical spectra;
- clustering detector hits or particles for jet reconstruction;
- applying principal-component analysis to Wilson coefficients in an effective-field-theory forecast.

1.3 Semi-supervised and reinforcement learning

Semi-supervised learning. Only a fraction of the data is labelled. The labelled subset is used to infer information about the unlabelled subset.

A representative example is the classification of γ-ray sources when spectra are available for many sources but reliable labels are known only for a smaller subset.

Reinforcement learning. This is the typical learning paradigm used by algorihms to play games (chess, go, ...). An agent (the algorithm) interacts with an environment (chess board, or some other system to interact with). At each step, it observes a state, chooses an action, and receives a reward. The goal is to learn a policy that maximises the expected cumulative reward.

An example in a physics context:

- the agent could be a trigger algorithm;
- the environment could be the incoming detector data stream;
- the reward could encode signal efficiency, background rejection, latency, or a combination thereof.

Other learning paradigms include active learning, self-supervised learning, and combinatorial optimisation problems such as Boolean satisfiability.

1.4 Machine learning and physics

Machine learning is fundamentally statistical. Observed data are treated as stochastic realisations of an underlying generative law.

For example,

$$y = f(\boldsymbol{x}; \boldsymbol{\theta}) + \epsilon, \qquad \epsilon \sim \mathcal{N}(0, \sigma^2),$$

implies

$$p(y \mid \boldsymbol{x}, \boldsymbol{\theta}) = \mathcal{N}\big(y \mid f(\boldsymbol{x}; \boldsymbol{\theta}), \sigma^2\big)\,.$$

The function f may be a physics model, a machine-learning model, or a combination of both. Machine learning is often most useful when it complements, rather than replaces, physical modelling.

Example 1: unknown or imperfectly modelled backgrounds. Suppose the data model is

$$f(\boldsymbol{x}; \boldsymbol{\theta}) = f_{\text{sig}}(\boldsymbol{x}; \boldsymbol{\theta}_{\text{DM}}) + f_{\text{bkg}}(\boldsymbol{x}; \boldsymbol{\theta}_{\text{bkg}}).$$

The signal model f_{sig} may be well motivated physically, while the background is difficult to model from first principles. One may then preserve the signal model and replace only the background contribution by a flexible machine-learning model. This is, keeping inference on the physics of interest (the signal) while treating the background in a physically agnostic way (e.g. as nuisance parameters).

Example 2: simulation emulation. If the physics model is accurate, but simulating it:

$$\boldsymbol{y} = \text{sim}(\boldsymbol{\theta}, \boldsymbol{\eta})$$

(where θ are the physical parameters of interest, and η are other required stochastic variables) is expensive, a "surrogate" ML model

$$\boldsymbol{y} \approx f_{\boldsymbol{w}}(\boldsymbol{\theta}, \boldsymbol{\eta})$$

(with optimizable parameters $\boldsymbol{w}$) may be trained to emulate it.

Example 3: "likelihood-free" inference. A simulator is formally a sampler of the data likelihood

$$p(\boldsymbol{y} \mid \boldsymbol{\theta})\ ,$$

even if the latter is unknown or intractable. A neural model can then be trained to approximate the likelihood $p(\boldsymbol{y}_{\text{obs}}|\boldsymbol{\theta})$, or the posterior $p(\boldsymbol{\theta}|\boldsymbol{y}_{\text{obs}})$, to make inference on the physical parameters $\boldsymbol{\theta}$ without the need to evaluate the likelihood (as opposed to likelihood-based methods as MCMC or standard Variational Inference). This method is called "Simulation-based Inference" (SBI), and it has been an established method nowadays in the communities of astrohpysics, cosmology and particle physics.

2 The "Learning" of ML

Learning and fitting

In the simplest supervised setting, learning reduces to fitting a parameterised function to data. More broadly, one may also learn:

- useful data representations;
- unknown probability distributions;
- strategies or policies.

So learning typically includes fitting, but its scope is arguably more generic.

2.1 Least squares

Consider a regression model $f(\boldsymbol{x};\boldsymbol{\theta})$. The least-squares objective is

$$C(\boldsymbol{\theta}) = \sum_{i=1}^{N} \left[y_i - f(\boldsymbol{x}_i;\boldsymbol{\theta})\right]^2 .$$

The fitted parameters are

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} C(\boldsymbol{\theta}).$$

For a linear model,

$$f(x;\boldsymbol{\theta}) = \theta_0 + \theta_1 x,$$

the optimum can be obtained analytically, as can be checked in any textbook of standard data analysis. More generally, an analytical solution exists whenever the model is linear in the parameters:

$$f(x;\boldsymbol{\theta}) = \sum_{m=0}^{M} \theta_m \phi_m(x),$$

where the basis functions $\phi_m(x)$ may themselves be highly nonlinear in x.

Indeed, define the design matrix

$$\Phi = \begin{pmatrix} 1 & \phi_1(x_1) & \cdots & \phi_M(x_1) \\ 1 & \phi_1(x_2) & \cdots & \phi_M(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \phi_1(x_N) & \cdots & \phi_M(x_N) \end{pmatrix}.$$

Then the optimum solution is:

$$\hat{\boldsymbol{\theta}} = (\Phi^\top \Phi)^{-1} \Phi^\top \boldsymbol{y},$$

provided that $\Phi^\top\Phi$ is invertible.

Homework 1. Derive the normal-equation solution above.

Least squares as maximum likelihood

Suppose

$$y_i \mid \boldsymbol{x}_i, \boldsymbol{\theta} \sim \mathcal{N}\big(f(\boldsymbol{x}_i; \boldsymbol{\theta}), \sigma^2\big) .$$

Assuming independent and identically distributed (i.i.d.) observations,

$$p(\boldsymbol{y} \mid X, \boldsymbol{\theta}) = \prod_{i=1}^{N} p(y_i \mid \boldsymbol{x}_i, \boldsymbol{\theta}) \ ,$$

where $\boldsymbol{y} = \{y_i\}$, $\boldsymbol{X} = \{\boldsymbol{x}_i\}$. The log-likelihood is

$$\log \mathcal{L}(\boldsymbol{\theta}) = -\frac{1}{2\sigma^2} \sum_{i=1}^{N} \left[y_i - f(\boldsymbol{x}_i; \boldsymbol{\theta})\right]^2 + \text{const.}$$

Therefore,

$$\arg\max_{\boldsymbol{\theta}} \log \mathcal{L}(\boldsymbol{\theta}) = \arg\min_{\boldsymbol{\theta}} C(\boldsymbol{\theta}).$$

Least squares is thus the maximum-likelihood estimator for a Gaussian noise model with constant variance.

2.2 Gradient descent

For nonlinear models, the maximum-likelihood estimator usually has no closed-form solution. One then resorts to numerical optimisation.

The "Gradient Descent" algorithm updates the parameters iteratively. At iteration t:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \nabla_{\boldsymbol{\theta}} C(\boldsymbol{\theta}_t),$$

where $\eta > 0$ is a hyper-parameter called the learning rate (see Fig. 3-left).

The algorithm converges to a stationary point,

$$\nabla_{\boldsymbol{\theta}} C(\boldsymbol{\theta}) = 0.$$

A large learning rate may overshoot the minimum, while a very small learning rate may make optimisation unnecessarily slow. Because non-convex objectives may contain local minima or saddle points, optimisation is often repeated from several random initialisations (See Fig. 4). Learning-rate schedules are also common:

$$\eta = \eta(t),$$

with $\eta(t)$ typically decreasing during training.

Stochastic gradient descent

For a dataset of size N, write

$$C(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^{N} C_i(\boldsymbol{\theta}).$$

Partition the data into B mini-batches of size ℓ_B, so that approximately $B = N/\ell_B$. Then

$$C(\boldsymbol{\theta}) = \frac{1}{B} \sum_{b=1}^{B} C_b(\boldsymbol{\theta}).$$

Stochastic gradient descent replaces the full gradient by a mini-batch estimate:

$$\nabla C(\boldsymbol{\theta}) \approx \nabla C_b(\boldsymbol{\theta}).$$

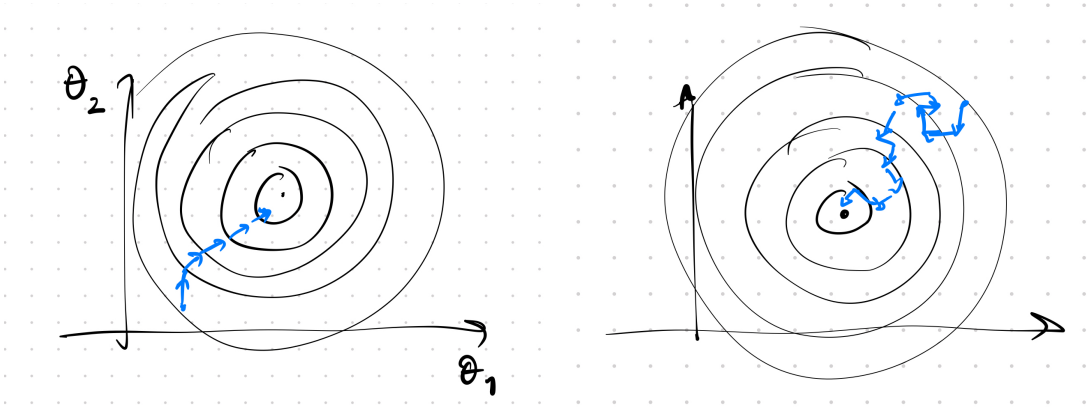

Figure 3: Left) Standard Gradient Descent (GD) evolution in a two-dimensional parameter space. The cost function is shown in contours around the minimum. Right) The same, for Stochastic Gradient Descent.

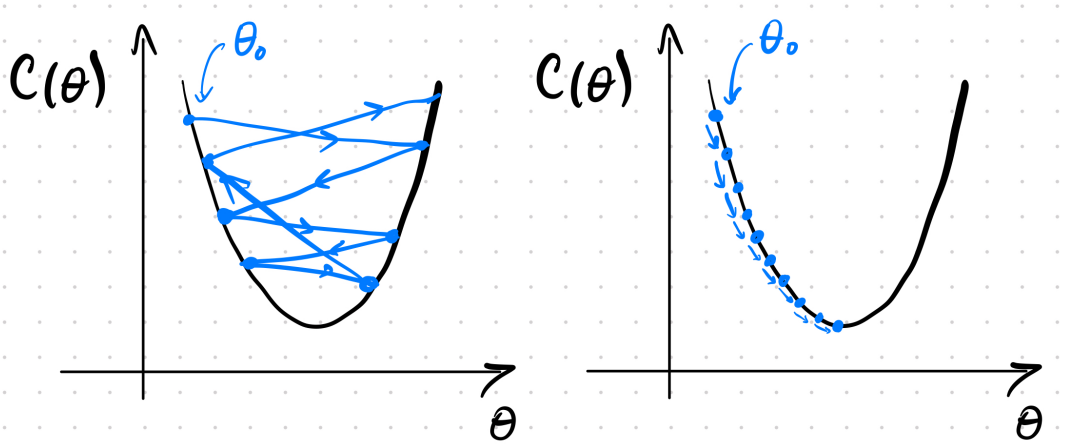

Figure 4: Standard Gradient Descent evolution. This figure shows the Cost function in a one-dimensional parameter space, with too large (small) values of the learning rates shown in the left (right) panel.

This implies that at every iteration, since the parameter update does not use the full cost function, it points towards a random direction (see Fig. 3-right). However, if the mini-batch is sampled uniformly, we have:

$$\mathbb{E}_b[\nabla C_b(\boldsymbol{\theta})] = \nabla C(\boldsymbol{\theta}) \ ,$$

so on average it converges to the same minimum. The method is computationally cheaper per step (since the sum is over $\ell_B \ll N$ terms), and the stochasticity may help the optimiser escape shallow local minima or saddle regions.

3 Overfitting

A very flexible model may fit the training data almost perfectly while failing to reproduce the data-generating mechanism. See Fig. 5 (the red curve). Such a model has learned statistical fluctuations or noise. It performs well on the training sample but poorly on new data: it fails to generalise.

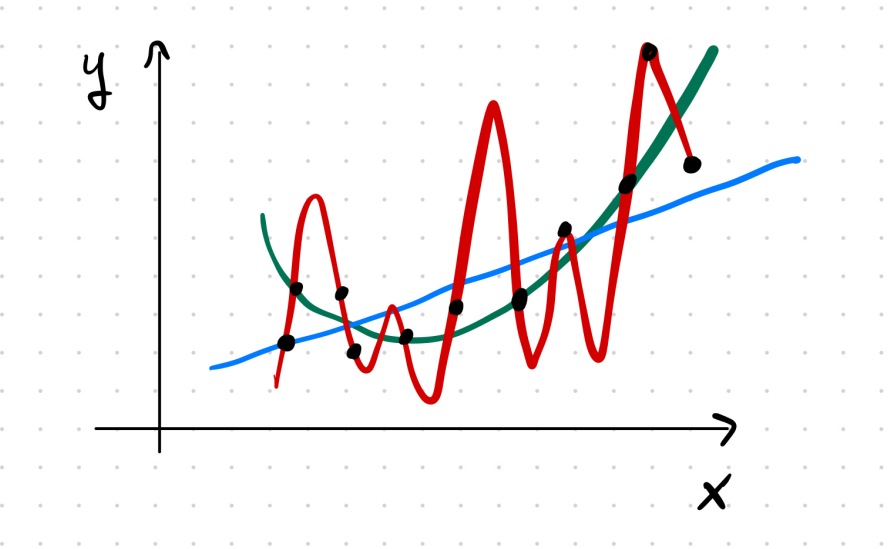

Figure 5: Schematic illustration of model complexity and overfitting. Given the data, a linear model (blue) seems to underfit them, while the more flexible model (red) seems to overfit. The parabola (green) seems to be the model closest to the true data generating function, and thus the one with best generalization power.

Bias–variance trade-off

Let the data-generating process be

$$y = f(x) + \epsilon, \qquad \mathbb{E}[\epsilon] = 0, \qquad \mathrm{Var}(\epsilon) = \sigma_\epsilon^2.$$

Let $\hat{f}_{\mathcal{D}}(x)$ be the model fitted on a dataset $\mathcal{D}$. At a fixed test point x_*, consider

$$\mathrm{MSE}(x_*) = \mathbb{E}_{\mathcal{D},y_*}\left[\left(y_* - \hat{f}_{\mathcal{D}}(x_*)\right)^2\right].$$

This decomposes as

$$\mathrm{MSE}(x_*) = \underbrace{\left[f(x_*) - \mathbb{E}_{\mathcal{D}}\hat{f}_{\mathcal{D}}(x_*)\right]^2}_{\text{bias}^2} + \underbrace{\mathrm{Var}_{\mathcal{D}}\left[\hat{f}_{\mathcal{D}}(x_*)\right]}_{\text{variance}} + \underbrace{\sigma_\epsilon^2}_{\text{irreducible noise}}.$$

Low-complexity models tend to have high bias and low variance. High-complexity models tend to have low bias and high variance.

Homework 2. Derive the bias–variance decomposition.

A traditional physics diagnostic

A common heuristic is to inspect

$$\frac{\chi^2}{\text{d.o.f.}}, \qquad \chi^2 = \sum_i \frac{[y_i - f(x_i;\boldsymbol{\theta})]^2}{\sigma_i^2}.$$

A value much smaller than one may indicate overfitting, although it may also result from overestimated uncertainties.

This criterion has important limitations:

- the effective number of degrees of freedom may be unclear for nonlinear models;
- parameter degeneracies may reduce the number of independent directions in function space;
- evaluating χ^2 on the same data used for fitting does not directly test generalisation.

3.1 Training and test splitting

A more direct approach is to split the data into disjoint training and test sets.

A typical procedure is:

1. shuffle the data;
2. reserve, for example, 20% for testing;
3. fit each candidate model on the training data;
4. evaluate each model on the test data;
5. select the model with the best test performance;
6. optionally refit the selected model on all available data.

The training and test sets should be representative of the same underlying distribution and should cover the relevant domain.

This is a very efficient heuristic method to control overfitting since, precisely by keeping part of the dataset outside the training, the generalization capability of the model can be directly estimated with the test dataset. See Fig. 6.

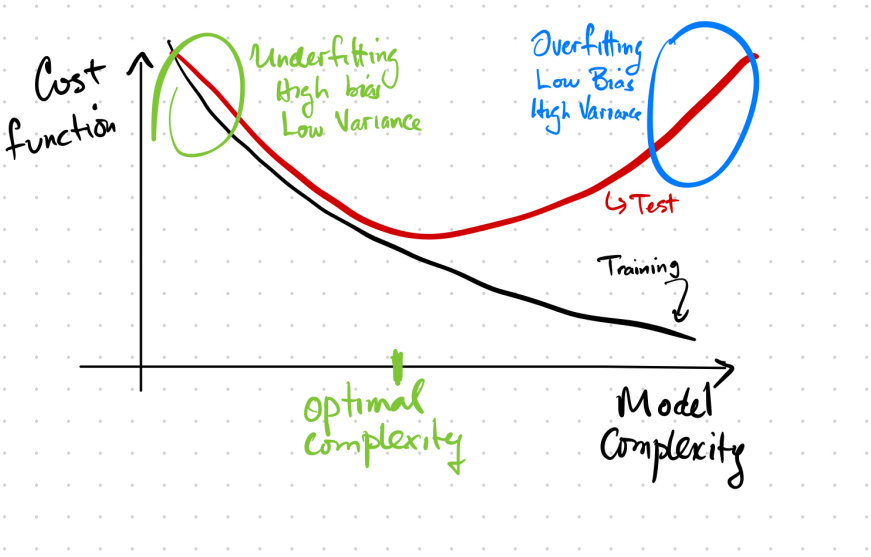

Figure 6: Schematic illustration of the behavior of the Cost as a function of the model complexity, for training dataset (black) and test dataset (red). The optimal complexity lies at intermediate values, where the cost/error on the test set is minimum. More (less) complex models lead to overfitting (underfitting).

K-fold cross-validation

To reduce dependence on one arbitrary split, divide the dataset into K folds:

$$\mathcal{D} = \mathcal{D}_1 \cup \cdots \cup \mathcal{D}_K.$$

For each k, use $\mathcal{D}_k$ as the validation set and the remaining folds as the training set. The model is selected by minimising the average validation error:

$$\frac{1}{K} \sum_{k=1}^{K} C_{\text{val}}^{(k)}.$$

Typical choices are $K = 5$ or $K = 10$.

3.2 Regularisation

For linear regression,

$$\hat{\boldsymbol{\theta}} = (\Phi^\top \Phi)^{-1} \Phi^\top \boldsymbol{y}.$$

As the number of basis functions approaches the number of data points, $\Phi^\top \Phi$ may become ill-conditioned or singular, generating a wiggly behavior (See Fig. 7, red curve). Large coefficients may then combine through delicate cancellations, producing oscillatory fits.

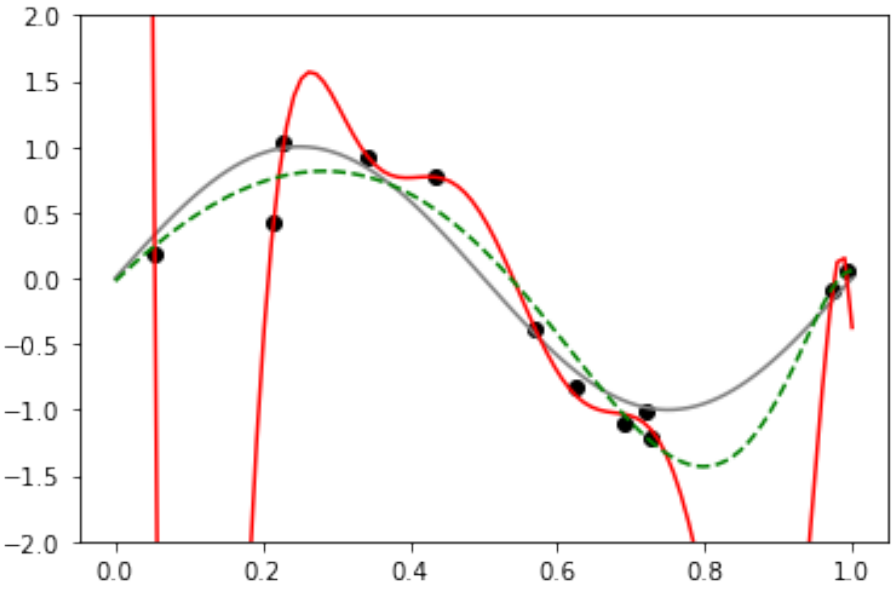

Figure 7: Example of a regression problem. Data points are a noisy realization of the generating function shown in gray. A polynomial of degree 9 is fitted via MSE and shown in red. The same polynomial is fitted via Ridge loss, and shown in green dashed.

In order to prevent large values of the parameters, Ridge regression modifies the objective:

$$C_{\text{ridge}}(\boldsymbol{\theta}) = \frac{1}{2} \sum_{i=1}^{N} [y_i - f(x_i; \boldsymbol{\theta})]^2 + \frac{\alpha}{2} \boldsymbol{\theta}^\top \boldsymbol{\theta}.$$

The resulting estimator is

$$\hat{\boldsymbol{\theta}}_{\text{ridge}} = (\Phi^\top \Phi + \alpha I)^{-1} \Phi^\top \boldsymbol{y}.$$

Often the intercept is excluded from the penalty by replacing I with a diagonal matrix whose intercept entry is zero. The resulting fit is clearly smoother, even for overparametrized models (See Fig. 7, green curve).

Regularisation also applies to nonlinear models, including neural networks, although the interpretation is less transparent. In general, regularisation introduces an *inductive bias*: it makes some hypotheses preferable to others.

4 Classification

4.1 Problem definition

In classification,

$$\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{N},$$

where

$$y_i \in \{1, \ldots, K\}$$

is a categorical label.

The goal is to assign a label to a new point $\boldsymbol{x}_*$. As a result, the algorithms compute (explicitly or implicitly) a "decision boundary" between the classes. See Fig. 8.

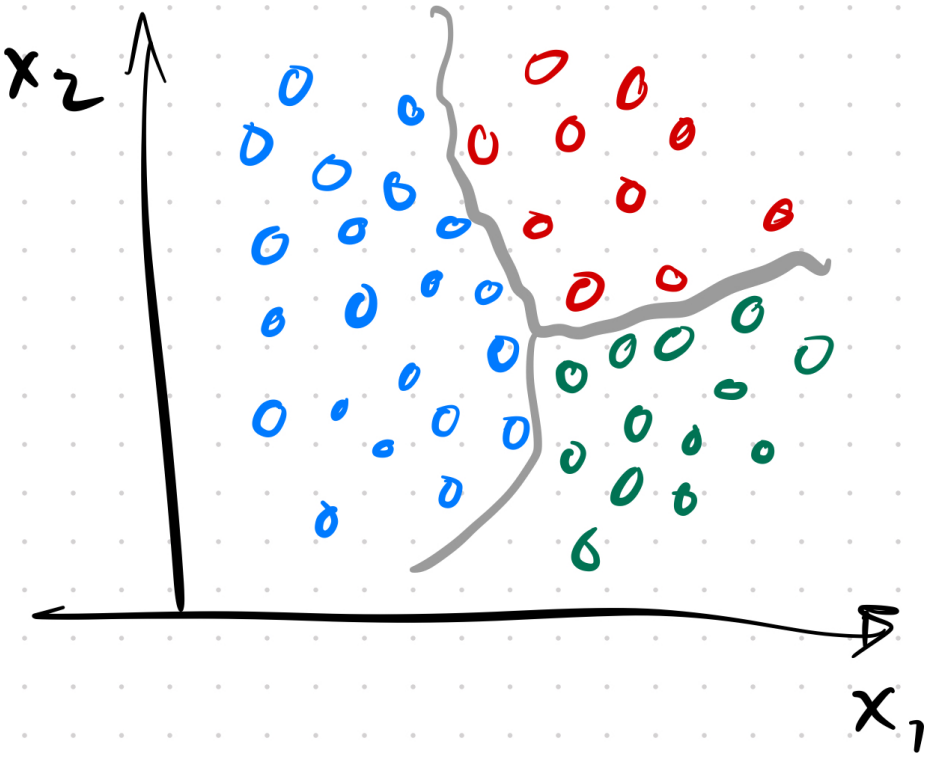

Figure 8: Illustration of a 3-class classification problem, and the resulting decision boundary (gray). In realistic problems, there is typically overlap between the clouds of points, so a perfect classification is in practice very challenging.

4.2 Three broad modelling strategies

Discriminative functions. These directly learn a decision boundary or a hard class-assignment rule. Basic decision trees fall naturally into this category.

Probabilistic discriminative models. These learn posterior class probabilities

$$p(C_k \mid \boldsymbol{x}).$$

Logistic regression and neural classifiers are standard examples.

Probabilistic generative models. These model the class-conditional distribution

$$p(\boldsymbol{x} \mid C_k)$$

first, and then use Bayes' theorem:

$$p(C_k \mid \boldsymbol{x}) = \frac{p(\boldsymbol{x} \mid C_k)p(C_k)}{\sum_j p(\boldsymbol{x} \mid C_j)p(C_j)},$$

to find the class probabilities. Naive Bayes, linear discriminant analysis, and quadratic discriminant analysis are examples in this category. They are called *generative* because the class-conditional distribution can be sampled to generate synthetic examples.

4.3 Probabilistic classification

For regression, a commonly adopted likelihood is

$$p(y \mid \boldsymbol{x}, \boldsymbol{\theta}) = \mathcal{N}\big(y \mid f(\boldsymbol{x}; \boldsymbol{\theta}), \sigma^2\big)\,,$$

as we saw previously. For classification, the target is categorical, so the appropriate likelihood changes.

Binary classification

Encode the two classes as

$$y \in \{0, 1\}.$$

This is a Bernoulli random variable, whose probability (mass) distribution satisfies

$$p(y \mid \mu) = \mu^y (1 - \mu)^{1-y},$$

where

$$\mu = p(y = 1).$$

A probabilistic classifier models precisely this parameter:

$$\mu(\boldsymbol{x}) = f(\boldsymbol{x}; \boldsymbol{\theta}) = p(y = 1 \mid \boldsymbol{x}, \boldsymbol{\theta}),$$

using a parametric function $f(\boldsymbol{x}; \boldsymbol{\theta})$ with optimizable parameters $\boldsymbol{\theta}$. Assuming independent and identically distributed (*iid*) data, the likelihood for the data set is thus:

$$\mathcal{L}(\boldsymbol{\theta}) = \prod_{i=1}^{N} f(\boldsymbol{x}_i; \boldsymbol{\theta})^{y_i} \left[1 - f(\boldsymbol{x}_i; \boldsymbol{\theta})\right]^{1-y_i}.$$

The negative log-likelihood is the so-called binary cross-entropy:

$$-\log \mathcal{L}(\boldsymbol{\theta}) = -\sum_{i=1}^{N} \left[y_i \log f(\boldsymbol{x}_i; \boldsymbol{\theta}) + (1 - y_i) \log \left(1 - f(\boldsymbol{x}_i; \boldsymbol{\theta})\right)\right].$$

The sigmoid function from Bayes' theorem

For two classes,

$$p(C_1 \mid \boldsymbol{x}) = \frac{p(\boldsymbol{x} \mid C_1)p(C_1)}{p(\boldsymbol{x} \mid C_1)p(C_1) + p(\boldsymbol{x} \mid C_2)p(C_2)}.$$

Define the log-odds

$$a(\boldsymbol{x}) = \log \frac{p(\boldsymbol{x} \mid C_1)p(C_1)}{p(\boldsymbol{x} \mid C_2)p(C_2)}.$$

Then

$$p(C_1 \mid \boldsymbol{x}) = \frac{1}{1 + \exp[-a(\boldsymbol{x})]} = \sigma(a(\boldsymbol{x})).$$

The sigmoid function maps

$$\sigma : \mathbb{R} \to (0, 1).$$

This is the reason why most of the probabilistic classifiers model use the sigmoid function as the output for binary classification.

Multiclass classification

For K classes, use the so-called "one-hot encoding":

$$\boldsymbol{y}_i = (y_{i1}, \ldots, y_{iK}), \qquad y_{ik} \in \{0, 1\}, \qquad \sum_{k=1}^{K} y_{ik} = 1.$$

I.e. for a given categorical output point $y_i = k$, its value is encoded in a K-component binary vector $\boldsymbol{y}_i$, where $y_{ik} = 1$, and $y_{ij} = 0 \; \forall j \neq k$. A categorical distribution is

$$p(\boldsymbol{y} \mid \boldsymbol{\mu}) = \prod_{k=1}^{K} \mu_k^{y_k}, \qquad \sum_{k=1}^{K} \mu_k = 1,$$

depending on a parameter vector $\boldsymbol{\mu}$ whose components give

$$\mu_k(\boldsymbol{x}) = p(C_k \mid \boldsymbol{x}, \boldsymbol{\theta}).$$

A multiclass classifier thus models precisely this parameter vector, $\mu_k(\boldsymbol{x}) = f_k(\boldsymbol{x}; \boldsymbol{\theta})$. The likelihood is

$$\mathcal{L}(\boldsymbol{\theta}) = \prod_{i=1}^{N} \prod_{k=1}^{K} f_k(\boldsymbol{x}_i; \boldsymbol{\theta})^{y_{ik}},$$

and the cross-entropy loss is

$$-\log \mathcal{L}(\boldsymbol{\theta}) = -\sum_{i=1}^{N} \sum_{k=1}^{K} y_{ik} \log f_k(\boldsymbol{x}_i; \boldsymbol{\theta}).$$

The multiclass analogue of the sigmoid is the softmax:

$$f_k(\boldsymbol{x}; \boldsymbol{\theta}) = \frac{\exp[a_k(\boldsymbol{x}; \boldsymbol{\theta})]}{\sum_{j=1}^{K} \exp[a_j(\boldsymbol{x}; \boldsymbol{\theta})]}.$$

We discuss below some relevant models for classification.

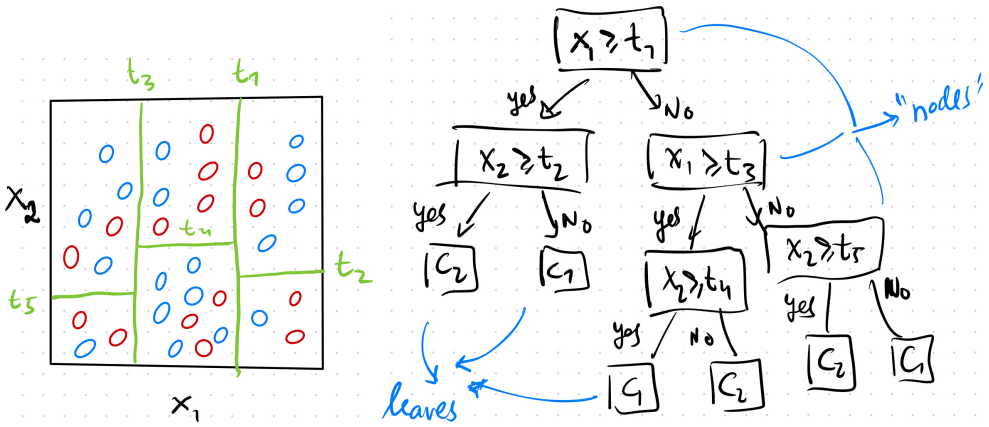

Figure 9: Left) Schematic partition of the input space in a binary-classification problem. Right) Corresponding "decision tree".

4.4 Representative models

4.4.1 Decision trees

Decision Trees is an example of a discriminative function (non probabilistic) classifier. A decision tree recursively partitions input space using binary tests such as

$$x_d < t.$$

At each stage, the split is chosen greedily: it is locally optimal given the current tree.
For a node corresponding to region R_m, define

$$p_{mk} = \frac{1}{N_m} \sum_{\boldsymbol{x}_i \in R_m} \mathbb{I}(y_i = k),$$

where N_m is the number of samples in the node.

The Gini impurity is

$$G(m) = \sum_{k=1}^{K} p_{mk}(1 - p_{mk}) = 1 - \sum_{k=1}^{K} p_{mk}^2.$$

For a candidate split j producing left and right child nodes,

$$G_j = \pi_L G(L) + \pi_R G(R),$$

where

$$\pi_{L,R} = \frac{N_{L,R}}{N_m}.$$

The selected split is

$$j_* = \arg\min_j G_j.$$

For a feature x_d, split candidates are typically chosen at midpoints between consecutive observed values:

$$t_\ell = \frac{x_d^{(\ell)} + x_d^{(\ell+1)}}{2}.$$

4.4.2 Gaussian generative classification

This is a probabilistic generative approach, since it models the distribution of the data given the class. In particular, these models assume

$$p(\boldsymbol{x} \mid C_k) = \mathcal{N}(\boldsymbol{x} \mid \boldsymbol{\mu}_k, \Sigma_k).$$

For two classes, the log-odds are

$$a(\boldsymbol{x}) = \log \frac{p(\boldsymbol{x} \mid C_1)\pi_1}{p(\boldsymbol{x} \mid C_2)\pi_2},$$

where $\pi_k \equiv p(C_k)$ are the priors. Expanding the Gaussian densities gives a quadratic function of $\boldsymbol{x}$:

$$a(\boldsymbol{x}) = -\frac{1}{2}\boldsymbol{x}^\top \left(\Sigma_1^{-1} - \Sigma_2^{-1}\right) \boldsymbol{x} + \boldsymbol{w}^\top \boldsymbol{x} + w_0,$$

where $\boldsymbol{w}$ and w_0 depend on the parameters $\boldsymbol{\mu}_k, \Sigma_k$ of the Gaussians, and the priors π_k respectively. Noting that the classification boundary for equal class probability is $a(\boldsymbol{x}) = 0$ (indeed, $\sigma(a = 0) = 0.5$ implies $p(C_1|\boldsymbol{x}) = p(C_0|\boldsymbol{x}) = 0.5$), then:

- if $\Sigma_1 = \Sigma_2$, the quadratic term vanishes and the decision boundary is linear;
- if $\Sigma_1 \neq \Sigma_2$, the decision boundary is quadratic.

For K classes, the maximum-likelihood estimators are

$$\hat{\pi}_k = \frac{N_k}{N},$$

$$\hat{\boldsymbol{\mu}}_k = \frac{1}{N_k} \sum_{i:y_i=k} \boldsymbol{x}_i,$$

and

$$\hat{\Sigma}_k = \frac{1}{N_k} \sum_{i:y_i=k} (\boldsymbol{x}_i - \hat{\boldsymbol{\mu}}_k)(\boldsymbol{x}_i - \hat{\boldsymbol{\mu}}_k)^\top.$$

Homework 4. Derive the maximum-likelihood estimators for π_k, $\boldsymbol{\mu}_k$, and Σ_k.

Naive Bayes

Naive Bayes is a particular implementation of a generative classifier, defined by assuming conditional independence of the features given the class:

$$p(\boldsymbol{x} \mid C_k) = \prod_{d=1}^{D} p(x_d \mid C_k).$$

Once the class-conditional distributions and class priors are estimated,

$$p(C_k \mid \boldsymbol{x}) \propto p(\boldsymbol{x} \mid C_k)\pi_k.$$

The prediction rule is

$$\hat{C}(\boldsymbol{x}) = \arg\max_k p(C_k \mid \boldsymbol{x}).$$

4.4.3 Logistic regression

Logistic regression is a probabilistic discriminative model.

For binary classification,

$$p(C_1 \mid \boldsymbol{x}) = \sigma(\boldsymbol{w}^\top \boldsymbol{x} + w_0).$$

For multiclass classification,

$$p(C_k \mid \boldsymbol{x}) = \frac{\exp\left(\boldsymbol{w}_k^\top \boldsymbol{x} + w_{k0}\right)}{\sum_{j=1}^{K} \exp\left(\boldsymbol{w}_j^\top \boldsymbol{x} + w_{j0}\right)}.$$

Unlike linear models of regression optimized via least squares, the optimum parameters of this linear classification model generally do not admit a closed-form solution via the cross entropy loss. Instead, They are obtained by numerical optimisation, for example gradient descent (or others methods, like quasi-Newton).

4.5 Neural Networks

A generalised linear regression model can be written as

$$f(\boldsymbol{x}) = \sum_{h=1}^{H} v_h \phi_h(\boldsymbol{x}).$$

For multiclass classification,

$$f_k(\boldsymbol{x}) = g_k\left(\sum_{h=1}^{H} v_{kh} \phi_h(\boldsymbol{x})\right),$$

where g_k is the identity for regression or a sigmoid/softmax-type output map for classification.

These models are linear in the chosen features ϕ_h (and Logistic Regression is an example of it). The idea of a neural network is to increase the flexibility of this model by making the features themselves trainable:

$$\phi_h(\boldsymbol{x}; \boldsymbol{w}_h) = s(\boldsymbol{w}_h^\top \boldsymbol{x} + b_h),$$

where s is a nonlinear activation function and $\boldsymbol{w}, b$ are trainable parameters.

A one-hidden-layer network is therefore defined as

$$\boxed{f_k(\boldsymbol{x}) = g_k\left(\sum_{h=1}^{H} v_{kh} s(\boldsymbol{w}_h^\top \boldsymbol{x} + b_h) + c_k\right).}$$

This function admits the typical graphical representation shown in Fig. 10.

Matrix notation

Let

$$X \in \mathbb{R}^{N \times D}$$

contain N input vectors with D features.

For one hidden layer, the neural network predictions can be written as:

$$F(X) = G\Big(S(XW + \mathbf{1}\boldsymbol{b}^\top)V + \mathbf{1}\boldsymbol{c}^\top\Big),$$

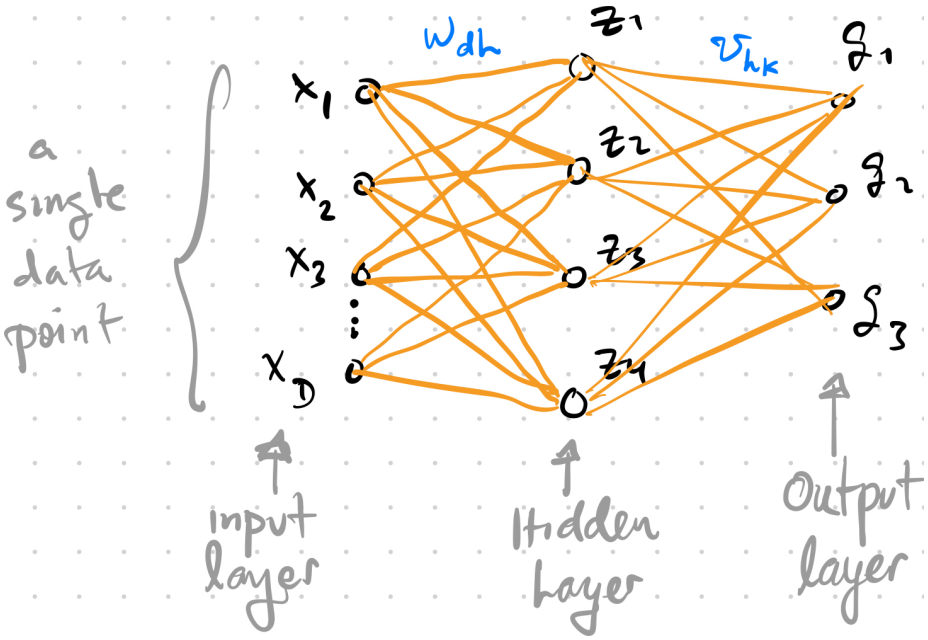

Figure 10: A fully connected feed-forward neural network.

where

$$W \in \mathbb{R}^{D\times H}, \qquad V \in \mathbb{R}^{H\times K},$$

are the matrices of parameters ("weights"), together with the parameters $\boldsymbol{b}, \boldsymbol{c}$.

For two hidden layers,

$$F(X) = G\Big(S_2\left[S_1(XW_1 + \mathbf{1}\boldsymbol{b}_1^\top)W_2 + \mathbf{1}\boldsymbol{b}_2^\top\right]V + \mathbf{1}\boldsymbol{c}^\top\Big).$$

This nested structure extends naturally to more hidden layers networks. A neural net with many hidden layers is called a "deep network". If there are many neurons/units in the hidden layers, the network is said to be "wide".

Activation functions

Common activation functions include:

$$\sigma(x) = \frac{1}{1+e^{-x}},$$

$$\tanh(x),$$

and

$$\mathrm{ReLU}(x) = \max(0, x).$$

Many activation choices are expressive enough in principle. Their practical differences mainly concern optimisation, smoothness, saturation, sparsity, and numerical stability.

Universal approximation

A feed-forward neural network with one hidden layer and a finite, but potentially large, number of neurons can approximate any continuous function on a compact domain under mild conditions on the activation function. This the famous "Universal Approximation Theorem" by Cybenko (1989).

This is an expressivity result, not an optimisation guarantee. It does not imply that:

- the required network is small;

- gradient-based training will find the desired parameters;
- the model will generalise well;
- the approximation is sample-efficient.

Architectures and physical symmetries

Neural-network architecture can encode known physical symmetries and constraints.

Translations. Convolutional neural networks are translation equivariant:

$$F(T_a x) = T_a F(x),$$

at least up to boundary effects and implementation details.

Rotations. Equivariant architectures can be constructed for rotation groups such as SO(2) and SO(3).

Permutations. Graph neural networks and set-based architectures are natural for systems of identical particles, where the output should be invariant or equivariant under permutations.

Lorentz transformations. Lorentz-equivariant architectures can be designed for relativistic particle-physics applications.

Gauge transformations. Gauge-equivariant neural networks can preserve local gauge symmetries by construction.

Architectural constraints act as inductive biases and can improve data efficiency, extrapolation, and physical consistency.

4.6 Classification metrics

To evaluate the performance of classifiers, different metrics can be used, providing different levels of granularity and convenience to different situations.

Accuracy. This is the simplest metric, globally defined across the whole dataset. It is the fraction of correctly classified examples:

$$\text{accuracy} = \frac{\#\{\text{correct predictions}\}}{N}.$$

Confusion matrix. For binary classification, one commonly reports:

$$\begin{pmatrix} \text{TP} & \text{FN} \\ \text{FP} & \text{TN} \end{pmatrix},$$

or its row-normalised version. In this matrix, rows represent the true labels (say, class 1 on top, class 0 at the bottom), while columns represent the predicted labels by the classifier. *TP* is thus the True Positives, i.e. the number of points correctly classified as class 1; *FP*, the False Positives, are the number of points incorrectly classified as class 1; *FN*: False Negatives, are the number of points incorrectly classified as class 0, and *TN*: True Negatives, are the number of points correctly classified as class 0.

The true-positive rate and false-positive rate are

$$\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}}, \qquad \text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}}.$$

The confusion matrix can be straightforwardly built for multiclass classification, although the entries are no longer interpreted as TP, FP, FN, TN, which is an inherently binary concept.

ROC curve. Again for binary classification, given a score $s(\boldsymbol{x}) = p(C_1 \mid \boldsymbol{x})$, classify as C_1 when

$$s(\boldsymbol{x}) > t.$$

Varying the threshold t produces the receiver operating characteristic curve:

$$\text{TPR}(t) \quad \text{versus} \quad \text{FPR}(t).$$

See Fig. 11-left, whose behavior can be understood by looking at the distribution of predictions (Fig. 11-right. At $t = 1$, almost all events are assigned to the negative class, so both rates TPR and FPR are near zero. On the other extreme, at $t = 0$, almost all are assigned to the positive class, so both rates are near one. For an intermediate value $0 < t < 1$, we typically have TPR $\geq$ FPR, since the region $s(x) \geq t$ contains more probability mass of class 1 than of class 0. A random classifier would give a ROC along the diagonal, TPR = FPR, while a perfect classifier (see red curve in Fig. 11-left) would give $TPR = 1\ \forall t$.

A summary of the ROC curve is its area under the curve (AUC), with AUC = 1 for a perfect classifier.

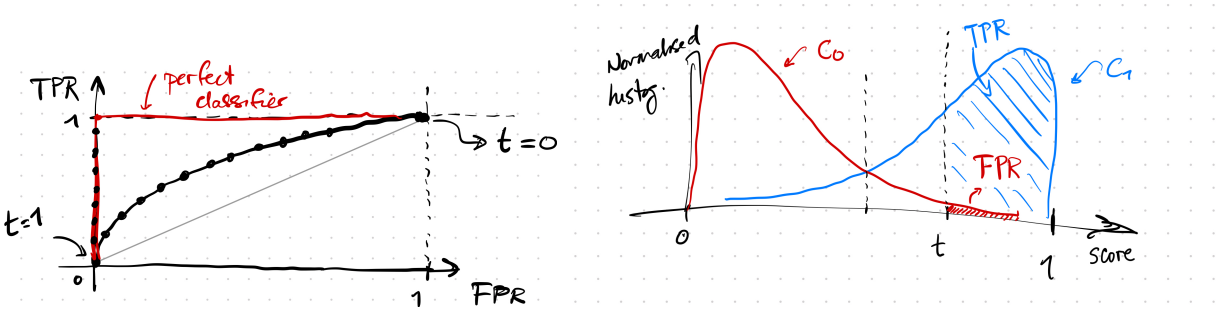

Figure 11: Left) Receiver operating characteristic (ROC) curve. Right) Score distributions (model predictions) for true signal –class 1– points (blue), and true background –class 0– points (red).

A generalization of the ROC curve for the multiclass case is not direct, since there is no inherent notion of TPR and FPR. However, practitioners usually adopt a "one-vs-the-rest" approach, where they represent the K-class classification prediction with K ROC curves, one per class k, where the negative class is the cumulative of the other classes $\forall j \neq k$.

Other metrics relevant to physics analyses

Two more classification metrics turn out to be closely connected to standard physics analyses (at least in particle physics):

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}, \quad \text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}, \tag{1}$$

noting that Recall = TPR.

In the context of signal-vs-background discrimination tasks, note that:

- TP: selected signal events (passing the cut t)
- FP: background events passing the cut t; i.e. this is the background contamination.
- FN: signal events not passing the cut t.

Then, Precision is a measure of the signal "purity":

$$\text{Precision} = \frac{S}{S+B} = \frac{S/B}{1+S/B} \Rightarrow \frac{S}{B} = \frac{\text{Precision}}{1 - \text{Precision}}$$

such that larger Precision is equivalent to larger S/B ratio.

On the other hand, Recall is the fraction of selected (true) signal events, so:

$$\text{Recall} = \epsilon_S : \quad \text{signal efficiency.}$$

Both Precision and Recall are very convenient metrics in common physics problems with a large class imbalance. For example, typical signal-vs-background analyses contain much more background events than signal events. Note that the ROC curve may look good even when Precision is very small. For example:

$$\begin{aligned} N_S &= 100, \quad N_B = 10^6 \\ \text{assume} &: \quad \text{TPR} = 0.8, \quad \text{FPR} = 10^{-3}, \quad \text{for some } t \\ &\Rightarrow \quad \text{TP} = 0.8 \times 100 = 80, \quad \text{FP} = 10^{-3} \times 10^6 = 1000. \\ &\Rightarrow \quad \text{Precision} = \frac{80}{80 + 1000} \simeq 0.07. \end{aligned}$$

Typically in physics, we require a given signal efficiency, say, $\epsilon_S = 0.7$, and we ask how much background rejection R_B our classifier can achieve:

$$R_B = \frac{1}{\epsilon_B}, \quad \epsilon_B = \frac{\text{FP}}{\text{FP} + \text{TN}} = \text{FPR}.$$

The procedure is then, simply, to extract the value $t = t_{0.7}$ along the ROC curve giving $\epsilon_S = \text{TPR} \simeq 0.7$, and then check the corresponding value of $\text{FPR}(t_{0.7})$.

5 Main Bibliography

1. *Pattern Recognition and Machine Learning.* C. M. Bishop, Springer, 2006.

2. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction (2nd edition).* T Hastie, R. Tibshirani, J. Friedman, Springer 2008.