

# The A<sup>”</sup> Dictionary of Machine Learning

Alexander Jung<sup>1</sup>, Konstantina Olioumtsevits<sup>1</sup>, Ekkehard Schnoor<sup>1</sup>,  
Tommi Flores Ryynänen<sup>1</sup>, Juliette Gronier<sup>2</sup>, Salvatore Rastelli<sup>1</sup>, and  
Mikko Seesto<sup>1</sup>

<sup>1</sup>Aalto University    <sup>2</sup>ENS Lyon

February 19, 2026



please cite as: A. Jung, K. Olioumtsevits, E. Schnoor, T. Ryynänen, J. Gronier, S. Rastelli, and M. Seesto, *The Aalto Dictionary of Machine Learning*. Espoo, Finland: Aalto University, 2026.

## Acknowledgment

This dictionary of machine learning evolved through the development and teaching of several courses, including CS-E3210 Machine Learning: Basic Principles, CS-C3240 Machine Learning, CS-E4800 Artificial Intelligence, CS-EJ3211 Machine Learning with Python, CS-EJ3311 Deep Learning with Python, CS-E4740 Federated Learning, and CS-E407507 Human-Centered Machine Learning. These courses were offered at Aalto University <https://www.aalto.fi/en>, to adult learners via The Finnish Institute of Technology (FITEch) <https://fitech.io/en/>, and to international students through the European University Alliance Unite! <https://www.aalto.fi/en/unite>.

We are grateful to the students who provided valuable feedback that helped to shape this dictionary.

This work was supported by

- the Research Council of Finland (grants 331197, 363624, 349966);
- the European Union (grant 952410);
- the Jane and Aatos Erkko Foundation (grant A835);
- Business Finland, as part of the project Forward-Looking AI Governance in Banking and Insurance (FLAIG).

# Contents

<b>List of Symbols</b>	<b>4</b>
<b>Mathematical Tools</b>	<b>22</b>
<b>Machine Learning Concepts</b>	<b>148</b>
<b>Reinforcement Learning</b>	<b>312</b>
<b>Machine Learning Systems</b>	<b>319</b>
<b>Machine Learning Regulation</b>	<b>325</b>
<b>Index</b>	<b>331</b>
<b>References</b>	<b>344</b>

# Lists of Symbols

## Sets and Functions

$a \in \mathcal{A}$  The object  $a$  is an element of the set  $\mathcal{A}$ .

---

$a := b$  We define  $a$  as  $b$ .

---

$|\mathcal{A}|$  The cardinality (i.e., number of elements) of a finite set  $\mathcal{A}$ .

---

$\mathcal{A} \subseteq \mathcal{B}$   $\mathcal{A}$  is a subset of  $\mathcal{B}$ .

---

$\mathcal{A} \subset \mathcal{B}$   $\mathcal{A}$  is a strict subset of  $\mathcal{B}$ .

---

$\mathcal{A} \times \mathcal{B}$  The Cartesian product of the sets  $\mathcal{A}$  and  $\mathcal{B}$ .

---

$\mathbb{N}$  The natural numbers  $1, 2, \dots$ .

---

$\mathbb{R}$  The real numbers  $x$  [1].

---

$\mathbb{R}_+$  The nonnegative real numbers  $x \geq 0$ .

---

$\mathbb{R}_{++}$  The positive real numbers  $x > 0$ .

---

$\{0, 1\}$  The set consisting of the two real numbers 0 and 1.

---

$[0, 1]$  The closed interval consisting of  $x \in \mathbb{R}$  for which  $0 \leq x \leq 1$ .

	The set of $\mathbf{w} \in \mathcal{C}$ minimizing the real-valued function $\arg \min_{\mathbf{w} \in \mathcal{C}} f(\mathbf{w})$ $f : \mathcal{C} \rightarrow \mathbb{R}.$
	See also: function.
$\mathbb{S}^{(n)}$	The set of unit-norm vectors in $\mathbb{R}^{n+1}$ . See also: norm, vector.
$\exp(a)$	The exponential function evaluated at the real number $a \in \mathbb{R}.$
	See also: function.
$\log a$	The logarithm of the positive number $a \in \mathbb{R}_{++}$ .
$f(\cdot) : \mathcal{A} \rightarrow \mathcal{B} : a \mapsto f(a)$	A function (or map) from a set $\mathcal{A}$ to a set $\mathcal{B}$ , which assigns to each input $a \in \mathcal{A}$ a well-defined output $f(a) \in \mathcal{B}$ . The set $\mathcal{A}$ is the domain of the function $f$ and the set $\mathcal{B}$ is the co-domain of $f$ . Machine learning (ML) aims to learn a function $h$ that maps features $\mathbf{x}$ of a data point to a prediction $h(\mathbf{x})$ for its label $y$ . See also: domain, co-domain.
$\text{epi}(f)$	The epigraph of a real-valued function $f : \mathbb{R}^d \rightarrow \mathbb{R}$ . See also: epigraph, function.
$(a_r)_{r \in \mathbb{N}}, (a^{(r)})_{r \in \mathbb{N}}, \{a^{(r)}\}_{r \in \mathbb{N}}$	A sequence of elements. See also: sequence.

---

$\mathbb{I}_{\mathcal{A}}(x)$  The indicator function of a set  $\mathcal{A}$  delivers  $f(x) = 1$  for any  $x \in \mathcal{A}$  and  $f(x) = 0$  otherwise.

See also: indicator function.

---

$\frac{\partial f(w_1, \dots, w_d)}{\partial w_j}$  The partial derivative (if it exists) of a real-valued function  $f : \mathbb{R}^d \rightarrow \mathbb{R}$  with respect to  $w_j$  [2, Ch. 9].

See also: partial derivative, function.

---

$\nabla f(\mathbf{w})$  The gradient of a differentiable real-valued function  $f : \mathbb{R}^d \rightarrow \mathbb{R}$  is the vector  $\nabla f(\mathbf{w}) = (\partial f / \partial w_1, \dots, \partial f / \partial w_d)^T \in \mathbb{R}^d$  [2, Ch. 9].

See also: gradient, differentiable, function, vector.

---

$\partial \mathcal{C}$  The boundary of a subset  $\mathcal{C}$  of some metric space.

See also: boundary, metric space.

---

$\text{Id}$  The identity operator.

See also: operator.

## Vector spaces

$\mathbf{x} = (x_1, \dots, x_d)^T$	A vector of length $d$ , with its $j$ th entry being $x_j$ . See also: vector.
$\mathbb{R}^d$	The set of all vectors $\mathbf{x} = (x_1, \dots, x_d)^T$ consisting of $d$ real-valued entries $x_1, \dots, x_d \in \mathbb{R}$ . See also: vector.
$\mathbf{I}_{l \times d}$	A generalized identity matrix with $l$ rows and $d$ columns. The entries of $\mathbf{I}_{l \times d} \in \mathbb{R}^{l \times d}$ are equal to 1 along the main diagonal and 0 otherwise. See also: matrix.
$\mathbf{I}_d, \mathbf{I}$	A square identity matrix of size $d \times d$ . If the size is clear from context, we drop the subscript. See also: matrix.
$\ \mathbf{x}\ _2$	The Euclidean (or $\ell_2$ ) norm of the vector $\mathbf{x} = (x_1, \dots, x_d)^T \in \mathbb{R}^d$ defined as $\ \mathbf{x}\ _2 := \sqrt{\sum_{j=1}^d x_j^2}$ . See also: norm, vector.
$\ \mathbf{x}\ $	Some norm of the vector $\mathbf{x} \in \mathbb{R}^d$ [3]. Unless otherwise specified, we mean the Euclidean norm $\ \mathbf{x}\ _2$ . See also: norm, vector, Euclidean norm.
$\mathbf{x}^T$	The transpose of a vector $\mathbf{x} \in \mathbb{R}^d$ is a matrix $\mathbf{x}^T \in \mathbb{R}^{1 \times d}$ with the vector as its single row. See also: transpose, vector, matrix.

The transpose of a matrix  $\mathbf{X} \in \mathbb{R}^{m \times d}$ . A square real-valued matrix  $\mathbf{X} \in \mathbb{R}^{m \times m}$  is called symmetric if  $\mathbf{X} = \mathbf{X}^T$ .

See also: transpose, matrix.

---

$\mathbf{X}^{-1}$  The inverse matrix of a matrix  $\mathbf{X} \in \mathbb{R}^{d \times d}$ .

See also: inverse matrix, matrix.

---

$\mathbf{0} = (0, \dots, 0)^T$  The vector in  $\mathbb{R}^d$  with each entry equal to zero.

See also: vector.

---

$\mathbf{1} = (1, \dots, 1)^T$  The vector in  $\mathbb{R}^d$  with each entry equal to one.

See also: vector.

---

$(\mathbf{v}^T, \mathbf{w}^T)^T$  The vector of length  $d + d'$  obtained by concatenating the entries of the vector  $\mathbf{v} \in \mathbb{R}^d$  with the entries of  $\mathbf{w} \in \mathbb{R}^{d'}$ .

See also: vector.

---

$\text{span}(\mathbf{B})$  The span of a matrix  $\mathbf{B} \in \mathbb{R}^{a \times b}$ , which is the subspace of all linear combinations of the columns of  $\mathbf{B}$  such that  $\text{span}(\mathbf{B}) = \{\mathbf{B}\mathbf{a} : \mathbf{a} \in \mathbb{R}^b\} \subseteq \mathbb{R}^a$ .

See also: matrix, subspace.

---

$\text{null}(\mathbf{A})$  The nullspace of a matrix  $\mathbf{A} \in \mathbb{R}^{a \times b}$ , which is the subspace of vectors  $\mathbf{a} \in \mathbb{R}^b$  such that  $\mathbf{A}\mathbf{a} = \mathbf{0}$ .

See also: nullspace, matrix, subspace, vector.

$\det(\mathbf{C})$  The determinant of the matrix  $\mathbf{C}$ .

See also: determinant, matrix.

---

$\text{tr}(\mathbf{C})$  The trace of the matrix  $\mathbf{C}$ .

See also: trace.

---

$\mathbf{A} \otimes \mathbf{B}$  The Kronecker product of the matrices  $\mathbf{A}$  and  $\mathbf{B}$  [4].

See also: Kronecker product, matrix.

---

Entrywise inequality between vectors  $\mathbf{a}, \mathbf{b} \in \mathbb{R}^d$ , i.e.,

$$\mathbf{a} \geq \mathbf{b} \quad a_j \geq b_j \quad \text{for } j = 1, \dots, d.$$

See also: vector.

---

A closed ball in some metric space that contains all points with a

$\overline{\mathcal{B}}_\varepsilon(\mathbf{x})$  distance from  $\mathbf{x}$  less or equal to  $\varepsilon$ .

See also: metric space.

---

The norm of a vector  $\mathbf{w} \in \mathcal{H}$  in a Hilbert space  $\mathcal{H}$ .

$\|\mathbf{w}\|_{\mathcal{H}}$  See also: norm, Hilbert space.

## Probability Theory

$\mathbf{x} \sim \mathbb{P}$  The random variable (RV)  $\mathbf{x}$  is distributed according to the probability distribution  $\mathbb{P}$  [5], [6].

See also: RV, probability distribution.

---

$\mathbb{E}_{\mathbb{P}}\{f(\mathbf{z})\}$  The expectation of an RV  $f(\mathbf{z})$  that is obtained by applying a deterministic function  $f$  to an RV  $\mathbf{z}$  whose probability distribution is  $\mathbb{P}$ . If the probability distribution is clear from context, we just write  $\mathbb{E}\{f(\mathbf{z})\}$ .

See also: expectation, RV, function, probability distribution.

---

$\text{cov}(x, y)$  The covariance between two real-valued RVs defined over a common probability space.

See also: covariance, RV, probability distribution.

---

$\mathbb{P}^{(\mathbf{x}, y)}$  A (joint) probability distribution of an RV whose realizations are data points with features  $\mathbf{x}$  and label  $y$ .

See also: probability distribution, RV, realization, data point, feature, label.

---

$\mathbb{P}^{(y|\mathbf{x})}$  A conditional probability distribution of an RV  $y$  given (or conditioned on) the value of another RV  $\mathbf{x}$  [7, Sec. 3.5].

See also: conditional probability distribution, RV.

---

$\mathbb{P}(\mathcal{A})$  The probability of the measurable event  $\mathcal{A}$ .

See also: probability, measurable, event.

	The moment generating function (MGF) of an RV $x$ .
$M_x(t)$	See also: probability distribution, probability density function (pdf).
$\mathbb{P}^{(\mathcal{D})}$	<p>The empirical distribution of a dataset <math>\mathcal{D}</math>.</p> <p>See also: empirical distribution, dataset, bootstrap.</p>
$\mathbb{P}^{(\mathbf{x}; \mathbf{w})}$	<p>A parameterized probability distribution of an RV <math>\mathbf{x}</math>. The probability distribution depends on a parameter vector <math>\mathbf{w}</math>. For example, <math>\mathbb{P}^{(\mathbf{x}; \mathbf{w})}</math> could be a multivariate normal distribution with the parameter vector <math>\mathbf{w}</math> given by the entries of the mean vector <math>\mathbb{E}\{\mathbf{x}\}</math> and the covariance matrix <math>\mathbb{E}\left\{ (\mathbf{x} - \mathbb{E}\{\mathbf{x}\})(\mathbf{x} - \mathbb{E}\{\mathbf{x}\})^T \right\}</math>.</p> <p>See also: probability distribution, parameter, probabilistic model.</p>
$\mathcal{N}(\mu, \sigma^2)$	<p>The probability distribution of a Gaussian random variable (Gaussian RV) <math>x \in \mathbb{R}</math> with mean (or expectation) <math>\mu = \mathbb{E}\{x\}</math> and variance <math>\sigma^2 = \mathbb{E}\{(x - \mu)^2\}</math>.</p> <p>See also: probability distribution, Gaussian RV.</p>
$\mathcal{N}(\boldsymbol{\mu}, \mathbf{C})$	<p>The multivariate normal distribution of a vector-valued Gaussian RV <math>\mathbf{x} \in \mathbb{R}^d</math> with mean (or expectation) <math>\boldsymbol{\mu} = \mathbb{E}\{\mathbf{x}\}</math> and covariance matrix <math>\mathbf{C} = \mathbb{E}\{(\mathbf{x} - \boldsymbol{\mu})(\mathbf{x} - \boldsymbol{\mu})^T\}</math>.</p> <p>See also: multivariate normal distribution, Gaussian RV.</p>
$\Delta^k$	<p>The probability simplex, which consists of all vectors <math>\mathbf{p} = (p_1, \dots, p_k)^T \in \mathbb{R}^k</math> with nonnegative entries that sum to one, i.e., <math>p_c \geq 0</math> for <math>c = 1, \dots, k</math> and <math>\sum_{c=1}^k p_c = 1</math>.</p> <p>See also: probability mass function (pmf).</p>

---

$H(x)$  The entropy of a discrete random variable (discrete RV)  $x$ .  
See also: entropy, discrete RV.

---

$\Omega$  A sample space of all possible outcomes of a random experiment.  
See also: event.

---

$\Sigma$  A collection of measurable subsets of a sample space  $\Omega$ .  
See also: sample space, event.

---

$\mathcal{P}$  A probability space that consists of a sample space  $\Omega$ , a  $\sigma$ -algebra  $\Sigma$  of measurable subsets of  $\Omega$ , and a probability distribution  $\mathbb{P}(\cdot)$ .  
See also: sample space, measurable, probability distribution.

## ML

---

$r$	An index $r = 1, 2, \dots$ that enumerates data points. See also: data point.
$m$	The number of data points in (i.e., the size of) a dataset. See also: data point, dataset.
$\mathcal{D}$	A dataset $\mathcal{D} = \{\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(m)}\}$ is a list of individual data points $\mathbf{z}^{(r)}$ , for $r = 1, \dots, m$ . See also: dataset, data point.
$d$	The number of features that characterize a data point. See also: feature, data point.
$x_j$	The $j$ th feature of a data point. The first feature is denoted by $x_1$ , the second feature $x_2$ , and so on. See also: data point, feature.
$\mathbf{x}$	The feature vector $\mathbf{x} = (x_1, \dots, x_d)^T$ of a data point. The vector's entries are the individual features of a data point. See also: feature vector, data point, vector, feature.
$\mathcal{X}$	The feature space $\mathcal{X}$ is the set of all possible values that the features $\mathbf{x}$ of a data point can take on. See also: feature space, feature, data point.

Instead of the symbol  $\mathbf{x}$ , we sometimes use  $\mathbf{z}$  as another symbol to denote a vector whose entries are the individual features of a data point. We need two different symbols to distinguish between raw and learned features [8, Ch. 9].

See also: vector, feature, data point.

---

$\mathbf{x}^{(r)}$  The feature vector of the  $r$ th data point within a dataset.  
See also: feature vector, data point, dataset.

---

$x_j^{(r)}$  The  $j$ th feature of the  $r$ th data point within a dataset.  
See also: feature, data point, dataset.

---

$\mathcal{B}$  A mini-batch (or subset) of randomly chosen data points.  
See also: batch, data point.

---

$B$  The size of (i.e., the number of data points in) a mini-batch.  
See also: data point, batch.

---

$y$  The label (or quantity of interest) of a data point.  
See also: label, data point.

---

$y^{(r)}$  The label of the  $r$ th data point.  
See also: label, data point.

---

$(\mathbf{x}^{(r)}, y^{(r)})$  The features and label of the  $r$ th data point.  
See also: feature, label, data point.

$\mathcal{Y}$  The label space  $\mathcal{Y}$  of an ML method consists of all potential label values that a data point can carry. The nominal label space might be larger than the set of different label values arising in a given dataset (e.g., a training set). ML problems (or methods) using a numeric label space, such as  $\mathcal{Y} = \mathbb{R}$  or  $\mathcal{Y} = \mathbb{R}^3$ , are referred to as regression problems (or methods). ML problems (or methods) that use a discrete label space, such as  $\mathcal{Y} = \{0, 1\}$  or  $\mathcal{Y} = \{\text{cat}, \text{dog}, \text{mouse}\}$ , are referred to as classification problems (or methods).

See also: label space, ML, label, data point, dataset, training set, regression, classification.

---

$\eta$  Learning rate (or step size) used by gradient-based methods.  
See also: learning rate, step size, gradient-based method.

---

$h(\cdot)$  A hypothesis map that maps the features of a data point to a prediction  $\hat{y} = h(\mathbf{x})$  for its label  $y$ .

See also: hypothesis, map, feature, data point, prediction, label.

---

$\mathcal{Y}^{\mathcal{X}}$  Given two sets  $\mathcal{X}$  and  $\mathcal{Y}$ , we denote by  $\mathcal{Y}^{\mathcal{X}}$  the set of all possible hypothesis maps  $h : \mathcal{X} \rightarrow \mathcal{Y}$ .

See also: hypothesis, map.

---

$\mathcal{H}$  A hypothesis space or model used by an ML method. The hypothesis space consists of different hypothesis maps  $h : \mathcal{X} \rightarrow \mathcal{Y}$  between which the ML method must choose.

See also: hypothesis space, model, ML, hypothesis, map.

$d_{\text{eff}}(\mathcal{H})$	The effective dimension of a hypothesis space $\mathcal{H}$ . See also: effective dimension, hypothesis space.
$B^2$	The squared bias of a learned hypothesis $\hat{h}$ , or its parameters. Note that $\hat{h}$ becomes an RV if it is learned from data points being RVs themselves. See also: bias, hypothesis, parameter, RV, data point.
$V$	The variance of a learned hypothesis $\hat{h}$ , or its parameters. Note that $\hat{h}$ becomes an RV if it is learned from data points being RVs themselves. See also: variance, hypothesis, parameter, RV, data point.
$L((\mathbf{x}, y), h)$	The loss incurred by predicting the label $y$ of a data point using the prediction $\hat{y} = h(\mathbf{x})$ . The prediction $\hat{y}$ is obtained by evaluating the hypothesis $h \in \mathcal{H}$ for the feature vector $\mathbf{x}$ of the data point. See also: loss, label, data point, prediction, hypothesis, feature vector.
$E_v$	The validation error of a hypothesis $h$ , which is its average loss incurred over a validation set. See also: validation error, hypothesis, loss, validation set.
$\hat{L}(h \mathcal{D})$	The empirical risk, or average loss, incurred by the hypothesis $h$ on a dataset $\mathcal{D}$ . See also: empirical risk, loss, hypothesis, dataset.

$E_t$  The training error of a hypothesis  $h$ , which is its average loss incurred over a training set.

See also: training error, hypothesis, loss, training set.

---

$t$  A discrete-time index  $t = 0, 1, \dots$  used to enumerate sequential events (or time instants).

See also: event.

---

$t$  An index that enumerates learning tasks within a multitask learning problem.

See also: learning task, multitask learning.

---

$\alpha$  A regularization parameter that controls the amount of regularization.

See also: regularization, parameter.

---

$\lambda_j(\mathbf{Q})$  The  $j$ th eigenvalue (sorted in either ascending or descending order) of a positive semi-definite (psd) matrix  $\mathbf{Q}$ . We also use the shorthand  $\lambda_j$  if the corresponding matrix is clear from context.

See also: eigenvalue, psd, matrix.

---

$\sigma(\cdot)$  The activation function used by an artificial neuron within an artificial neural network (ANN).

See also: activation function, ANN.

---

$\mathcal{R}_{\hat{y}}$  A decision region within a feature space.

See also: decision region, feature space.

**w** A parameter vector  $\mathbf{w} = (w_1, \dots, w_d)^T$  of a model, e.g., the weights of a linear model or an ANN.

See also: parameter, vector, model, weight, linear model, ANN.

---

**$h^{(\mathbf{w})}(\cdot)$**  A hypothesis map that involves tunable model parameters  $w_1, \dots, w_d$  stacked into the vector  $\mathbf{w} = (w_1, \dots, w_d)^T$ .

See also: hypothesis, map, model parameter, vector.

---

**$\phi(\cdot)$**  A feature map  $\phi : \mathcal{X} \rightarrow \mathcal{X}' : \mathbf{x} \mapsto \phi(\mathbf{x})$  that transforms the feature vector  $\mathbf{x}$  of a data point into a new feature vector  $\mathbf{x}' = \phi(\mathbf{x}) \in \mathcal{X}'$ .

See also: feature map.

---

**$k(\cdot, \cdot)$**  Given some feature space  $\mathcal{X}$ , a kernel is a map  $k : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{C}$  that is psd.

See also: feature space, kernel, map, psd.

---

**$\text{VCdim}(\mathcal{H})$**  The Vapnik–Chervonenkis dimension (VC dimension) of the hypothesis space  $\mathcal{H}$ .

See also: VC dimension, hypothesis space.

## Federated learning (FL)

An undirected graph whose nodes  $i \in \mathcal{V}$  represent devices within a federated learning network (FL network). The undirected weighted edges  $\mathcal{E}$  represent connectivity between devices and statistical similarities between their datasets and learning tasks.

See also: undirected graph, device, FL network, dataset, learning task.

---

A node that represents some device within an FL network.

$i \in \mathcal{V}$  The device can access a local dataset and train a local model.  
See also: device, FL network, local dataset, local model.

---

$\mathcal{G}^{(\mathcal{C})}$  The induced subgraph of  $\mathcal{G}$  using the nodes in  $\mathcal{C} \subseteq \mathcal{V}$ .

---

$\mathbf{L}^{(\mathcal{G})}$  The Laplacian matrix of a graph  $\mathcal{G}$ .  
See also: Laplacian matrix, graph.

---

$\mathbf{L}^{(\mathcal{C})}$  The Laplacian matrix of the induced graph  $\mathcal{G}^{(\mathcal{C})}$ .  
See also: Laplacian matrix, graph.

---

$\mathcal{N}^{(i)}$  The neighborhood of node  $i$  in a graph  $\mathcal{G}$ .  
See also: neighborhood, graph.

---

$d^{(i)}$  The weighted node degree  $d^{(i)} := \sum_{i' \in \mathcal{N}^{(i)}} A_{i,i'}$  of node  $i$ .  
See also: node degree.

---

$d_{\max}^{(\mathcal{G})}$  The maximum weighted node degree of a graph  $\mathcal{G}$ .  
See also: maximum, node degree, graph.

$\mathcal{D}^{(i)}$  The local dataset  $\mathcal{D}^{(i)}$  carried by node  $i \in \mathcal{V}$  of an FL network.

See also: local dataset, FL network.

$m_i$  The number of data points (i.e., the sample size) contained in the local dataset  $\mathcal{D}^{(i)}$  at node  $i \in \mathcal{V}$ .

See also: data point, sample size, local dataset.

$\mathbf{x}^{(i,r)}$  The features of the  $r$ th data point in the local dataset  $\mathcal{D}^{(i)}$ .

See also: feature, data point, local dataset.

$y^{(i,r)}$  The label of the  $r$ th data point in the local dataset  $\mathcal{D}^{(i)}$ .

See also: label, data point, local dataset.

$\mathbf{w}^{(i)}$  The local model parameters of device  $i$  within an FL network.

See also: model parameter, device, FL network.

$L_i(\mathbf{w})$  The local loss function used by device  $i$  to measure the usefulness of some choice  $\mathbf{w}$  for the local model parameters.

See also: loss function, device, model parameter.

$L^{(d)}(\mathbf{x}, h(\mathbf{x}), h'(\mathbf{x}))$  The loss incurred by a hypothesis  $h'$  on a data point with features  $\mathbf{x}$  and label  $h(\mathbf{x})$  that is obtained from another hypothesis.

See also: loss, hypothesis, data point, feature, label.

---

$\text{stack}\{\mathbf{w}^{(i)}\}_{i=1}^n$

The vector  $\left(\left(\mathbf{w}^{(1)}\right)^T, \dots, \left(\mathbf{w}^{(n)}\right)^T\right)^T \in \mathbb{R}^{dn}$  that is obtained by vertically stacking the local model parameters  $\mathbf{w}^{(i)} \in \mathbb{R}^d$  for  $i = 1, \dots, n$ .

See also: vector, model parameter.

---

$h^{(i)}$

A hypothesis  $h^{(i)} \in \mathcal{H}^{(i)}$  at some node  $i$  within an FL network.

See also: hypothesis, FL network.

---

$\hat{h}^{(i)}$

A learned hypothesis  $\hat{h}^{(i)} \in \mathcal{H}^{(i)}$ , obtained by some FL method, at some node  $i$  within an FL network.

See also: hypothesis, FL network.

## Mathematical Tools

**algebraic connectivity** The algebraic connectivity of an undirected graph is the second-smallest eigenvalue  $\lambda_2$  of its Laplacian matrix. An undirected graph is connected if and only if  $\lambda_2 > 0$  (see Fig. 1) [9], [10].



Fig. 1. Three examples of undirected graphs.

See also: undirected graph, eigenvalue, Laplacian matrix, connected.

**alternating direction method of multipliers (ADMM)** The ADMM is an iterative optimization method for solving a structured optimization problem. In particular, the ADMM can be used to solve an optimization problem of the following form:

$$\begin{aligned} \min_{\mathbf{x} \in \mathbb{R}^d, \mathbf{x}' \in \mathbb{R}^{d'}} \quad & f(\mathbf{x}) + g(\mathbf{x}') \\ \text{s.t.} \quad & \mathbf{A}\mathbf{x} - \mathbf{B}\mathbf{x}' = \mathbf{c} \end{aligned}$$

for given matrices  $\mathbf{A} \in \mathbb{R}^{p \times d}$  and  $\mathbf{B} \in \mathbb{R}^{p \times d'}$  and a given vector  $\mathbf{c} \in \mathbb{R}^p$ .

See also: optimization method, method of multipliers (MoM).

**augmented Lagrangian** The augmented Lagrangian is a modification of the standard Lagrangian that includes an additional quadratic penalty term for constraint violations. It is used in the MoM for iteratively solving constrained optimization problems.

See also: MoM.

**average node degree** The average node degree  $\bar{d}$  of a weighted undirected graph  $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{A})$  is the average of all node degrees, i.e.,  $\bar{d} = (1/n) \sum_{i \in \mathcal{V}} d^{(i)}$ .

See also: node degree, undirected graph, neighbor.

**Banach's fixed-point theorem** Banach's fixed-point theorem (also referred to as the contraction principle [2, Th. 9.23], [11]) states that every contractive operator  $\mathcal{F}$  on a complete metric space  $\mathcal{W}$  has a unique fixed point. Formally, let  $(\mathcal{W}, d(\cdot, \cdot))$  be a non-empty complete metric space and let  $\mathcal{F} : \mathcal{W} \rightarrow \mathcal{W}$  satisfy

$$d(\mathcal{F}\mathbf{w}, \mathcal{F}\mathbf{w}') \leq \kappa \cdot d(\mathbf{w}, \mathbf{w}') \quad \forall \mathbf{w}, \mathbf{w}' \in \mathcal{W}$$

for some constant  $\kappa \in [0, 1)$ . Then,  $\mathcal{F}$  has a unique fixed point, i.e., there exists a unique  $\mathbf{w}^* \in \mathcal{W}$  with  $\mathcal{F}\mathbf{w}^* = \mathbf{w}^*$ . Moreover, for any initial  $\mathbf{w}^{(0)} \in \mathcal{W}$ , the fixed-point iteration  $\mathbf{w}^{(t)} = \mathcal{F}\mathbf{w}^{(t-1)}$ , for  $t = 1, 2, \dots$ , converges to  $\mathbf{w}^*$  at a rate governed by  $\kappa$ . In particular,

$$d(\mathbf{w}^{(t)}, \mathbf{w}^*) \leq \kappa^t \cdot d(\mathbf{w}^{(0)}, \mathbf{w}^*) \quad \text{for } t = 1, 2, \dots$$



Fig. 2. Contractive operator  $\mathcal{F} : \mathcal{W} \rightarrow \mathcal{W}$  with a unique fixed point  $\mathbf{w}^*$  satisfying  $\mathcal{F}\mathbf{w}^* = \mathbf{w}^*$ .

See also: contractive operator, fixed-point iteration.

**basis** A basis of a finite-dimensional vector space  $\mathcal{V}$  is a set of linearly independent vectors  $\mathcal{B} = \{\mathbf{b}^{(1)}, \dots, \mathbf{b}^{(d)}\} \subseteq \mathcal{V}$  such that any vector  $\mathbf{v} \in \mathcal{V}$  can be expressed as a linear combination of the basis vectors  $\mathbf{b}^{(1)}, \dots, \mathbf{b}^{(d)}$ , i.e.,

$$\mathbf{v} = \sum_{j=1}^d \alpha_j \mathbf{b}^{(j)} \quad \text{for some } \alpha_1, \dots, \alpha_d.$$

The scalars  $\alpha_1, \dots, \alpha_d$  can be regarded as the coordinates of  $\mathbf{v}$  with respect to the basis  $\mathcal{B}$ . Any basis of  $\mathcal{V}$  has the same number  $d$  of elements, which is the dimension of  $\mathcal{V}$ .

See also: vector space, linearly independent, vector.

**boundary** Consider a subset  $\mathcal{C} \subseteq \mathbb{R}^n$ . The boundary of  $\mathcal{C}$ , denoted by  $\partial\mathcal{C}$ , is the set of all points  $\mathbf{x} \in \mathbb{R}^n$  such that every closed ball  $\overline{\mathcal{B}}_\varepsilon(\mathbf{x}) = \{\mathbf{y} \in \mathbb{R}^n : \|\mathbf{y} - \mathbf{x}\| \leq \varepsilon\}$  of radius  $\varepsilon > 0$  centered at  $\mathbf{x}$  intersects both  $\mathcal{C}$  and its complement  $\mathbb{R}^n \setminus \mathcal{C}$ .



Fig. 3. Boundary of the set  $\mathcal{C}$  and a closed ball  $\overline{\mathcal{B}}_\varepsilon(\mathbf{x})$  around  $\mathbf{x} \in \partial\mathcal{C}$ .

Note that the boundary  $\partial\mathcal{C}$  might contain elements that do not belong to  $\mathcal{C}$  itself. The notion of a boundary can be generalized to arbitrary metric spaces, including an undirected graph with a metric given by the length of the shortest path.

See also: metric space, neighborhood.

**Bregman divergence** The Bregman divergence induced by a convex, differentiable function  $\phi(\cdot)$  is defined as

$$D_\phi(\mathbf{w}, \mathbf{w}') := \phi(\mathbf{w}) - \phi(\mathbf{w}') - \langle \nabla\phi(\mathbf{w}'), \mathbf{w} - \mathbf{w}' \rangle$$

for  $\mathbf{w}, \mathbf{w}'$  in the domain  $\mathcal{W}$  of  $\phi$ . It measures the deviation of  $\phi(\mathbf{w})$  from its first-order Taylor approximation around  $\mathbf{w}'$  and is in general neither symmetric nor a metric.



Fig. 4. Contour lines  $\{\mathbf{w} \in \mathcal{W} : \phi(\mathbf{w}) = c\}$  for different values of  $c \in \mathbb{R}$  of a convex function  $\phi(\cdot)$  defined on a convex set  $\mathcal{W}$ . The density of the contour lines can be used to steer the learning rate used in a gradient step, i.e., if the current model parameter lies in a region with dense contour lines, a smaller learning rate is preferable.

For a twice differentiable  $\phi$ , the divergence  $D_\phi(\cdot, \cdot)$  behaves locally like a quadratic form:

$$D_\phi(\mathbf{w}, \mathbf{w}') \approx \frac{1}{2} (\mathbf{w} - \mathbf{w}')^T \nabla^2 \phi(\mathbf{w}') (\mathbf{w} - \mathbf{w}'),$$

which can be interpreted as a squared norm of the displacement  $\mathbf{w} - \mathbf{w}'$  induced by the Hessian  $\nabla^2 \phi(\mathbf{w}')$ .

See also: gradient step, gradient, inner product, proximal operator.

**Cauchy sequence** A Cauchy sequence is a sequence  $(\mathbf{x}^{(r)})_{r \in \mathbb{N}}$  in a metric space  $(\mathcal{X}, d(\cdot, \cdot))$  such that the elements  $\mathbf{x}^{(r)} \in \mathcal{X}$  become arbitrarily close to each other eventually. In other words [2, Definition 3.8],

$$\forall \epsilon > 0, \exists N \in \mathbb{N} \text{ such that } \forall r, r' \geq N, d(\mathbf{x}^{(r)}, \mathbf{x}^{(r')}) < \epsilon.$$

Fig. 5 shows a Cauchy sequence in the metric space  $(\mathbb{Q}, |\cdot|)$  of rational numbers.



Fig. 5. Cauchy sequence  $(x^{(r)})_{r \in \mathbb{N}}$  in the metric space  $(\mathbb{Q}, |\cdot|)$ . This sequence is generated by a fixed-point iteration used to approximate  $\sqrt{2}$ . For all  $r \geq N$ , the sequence elements lie within a band of width  $\varepsilon$ . Note that the sequence does not converge in  $\mathbb{Q}$ , since  $\sqrt{2} \notin \mathbb{Q}$  [2, Example 1.1].

See also: sequence, metric space, convergence.

**Cauchy-Schwarz inequality** For any vectors  $\mathbf{x}, \mathbf{x}'$  in an inner product space (i.e., a Hilbert space) over the field of real or complex numbers, the Cauchy-Schwarz inequality provides an upper bound on the (squared) absolute value (on  $\mathbb{R}$  or  $\mathbb{C}$ ) of their inner product. More precisely, it states that

$$|\langle \mathbf{x}, \mathbf{x}' \rangle|^2 \leq \langle \mathbf{x}, \mathbf{x} \rangle \langle \mathbf{x}', \mathbf{x}' \rangle.$$

Using the norm defined by the inner product, this can be expressed equivalently as  $|\langle \mathbf{x}, \mathbf{x}' \rangle| \leq \|\mathbf{x}\| \|\mathbf{x}'\|$ . Equality holds if and only if  $\{\mathbf{x}, \mathbf{x}'\}$  is linearly dependent.

See also: linearly independent, firmly non-expansive operator, Euclidean space, orthogonality condition.

**central limit theorem (CLT)** Consider a sequence of independent and

identically distributed (i.i.d.) RVs  $x^{(r)}$ , for  $r = 1, 2, \dots$ , each with mean zero and finite variance  $\sigma^2 > 0$ . The CLT states that the normalized sum

$$s^{(m)} := \frac{1}{\sqrt{m}} \sum_{r=1}^m x^{(r)}$$

converges in distribution to a Gaussian RV with mean zero and variance  $\sigma^2$  as  $m \rightarrow \infty$  [12, Proposition 2.17]. One elegant way to derive the CLT is via the characteristic function of the normalized sum  $s^{(m)}$ . Let  $\phi(t) = \mathbb{E}\{\exp(jtx)\}$  (with the imaginary unit  $j = \sqrt{-1}$ ) be the common characteristic function of each sum and  $x^{(r)}$ , and let  $\phi^{(m)}(t)$  denote the characteristic function of  $s^{(m)}$ . Define an operator  $\mathcal{T}$  acting on characteristic functions such that

$$\phi^{(m)}(t) = \mathcal{T}(\phi^{(m-1)})(t) := \phi\left(\frac{t}{\sqrt{m}}\right) \cdot \phi^{(m-1)}\left(\frac{\sqrt{m-1}}{\sqrt{m}}t\right).$$

This fixed-point iteration captures the effect of recursively adding an i.i.d. RV  $\mathbf{x}^{(m)}$  and rescaling. Iteratively applying  $\mathcal{T}$  leads to convergence of  $\phi^{(m)}(t)$  toward the fixed point

$$\phi^*(t) = \exp(-t^2\sigma^2/2),$$

which is the characteristic function of a Gaussian RV with mean zero and variance  $\sigma^2$ . Generalizations of the CLT allow for dependent or nonidentically distributed RVs [12, Sec. 2.8].



Fig. 6. Characteristic functions of normalized sums of i.i.d. RVs  $x^{(r)} \in \{-1, 1\}$  for  $r = 1, \dots, m$  compared to the Gaussian limit.

See also: RV, Gaussian RV.

**characteristic function** The characteristic function of a real-valued RV  $x$  is the following function [6, Sec. 26]:

$$\phi_x(t) := \mathbb{E}\exp(jtx) \quad \text{with } j = \sqrt{-1}.$$

The characteristic function uniquely determines the probability distribution of  $x$ .

See also: RV, probability distribution.

**Chebyshev's inequality** Consider a real-valued RV  $x$  for which the second moment  $\mathbb{E}\{x^2\}$  exists (and is finite). The existence of the second moment implies the existence of a finite expectation  $\mu := \mathbb{E}\{x\}$  and a finite variance  $\sigma^2 := \mathbb{E}\{(x-\mu)^2\}$  [13, Proposition 6.12]. Chebyshev's inequality refers to the following upper bound on the probability that

$x$  deviates from  $\mu$  by more than a given threshold  $\eta$  [14, Ch. 4]. In particular,

$$\mathbb{P}(|x - \mu| \geq \eta) \leq \frac{\sigma^2}{\eta^2} \quad \text{for any } \eta > 0.$$

This upper bound can be obtained by applying Markov's inequality to the new RV  $\tilde{x} := (x - \mu)^2$ .

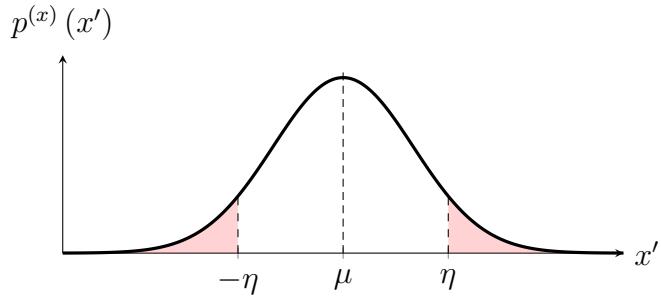


Fig. 7. Chebyshev's inequality provides an upper bound on the tail probability  $\mathbb{P}(|x - \mu| \geq \eta)$  (i.e., the shaded area) of a real-valued RV  $x$  with a finite second moment.

See also: expectation, Markov's inequality, concentration inequality.

**Chernoff bound** The Chernoff bound is a concentration inequality derived as a direct application of Markov's inequality [15, Ch. 2]. Let  $x$  be a real-valued RV such that its MGF  $M_x(t) = \mathbb{E}\{\exp(tx)\}$  exists for some  $t > 0$ . Applying Markov's inequality to the nonnegative RV  $\exp(tx)$  yields, for any  $\eta \in \mathbb{R}$ ,

$$\mathbb{P}(x \geq \eta) = \mathbb{P}(\exp(tx) \geq \exp(t\eta)) \leq \exp(-t\eta) \mathbb{E}\{\exp(tx)\}.$$

Note that this is actually an entire family of upper bounds, parameterized by all valid choices for  $t > 0$  (i.e.,  $M_x(t)$  must exist).

See also: concentration inequality, Markov's inequality, Chebyshev's inequality, Hoeffding's inequality.

**co-domain** The co-domain of a function  $f : \mathcal{U} \rightarrow \mathcal{V}$  is the set  $\mathcal{V}$  into which  $f$  maps elements of its domain  $\mathcal{U}$ .

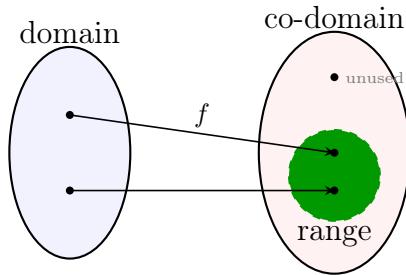


Fig. 8. Domain and co-domain of a function  $f : \mathcal{U} \rightarrow \mathcal{V}$ .

See also: domain, function, map.

**column space** The column space of a matrix  $\mathbf{A} \in \mathbb{R}^{m \times d}$ , denoted by  $\text{span}(\mathbf{A})$ , is the set of all linear combinations of the columns of  $\mathbf{A}$ . In other words,

$$\text{span}(\mathbf{A}) = \{\mathbf{A}\mathbf{w} : \mathbf{w} \in \mathbb{R}^d\}.$$

The column space  $\text{span}(\mathbf{A})$  of the matrix  $\mathbf{A}$  is a subspace of the Euclidean space  $\mathbb{R}^m$ .

See also: matrix, vector space.

**concentration inequality** Concentration inequality refers to an upper bound on the probability that an RV deviates more than a prescribed amount from its expectation [16].

See also: probability, RV, expectation, Markov's inequality, Chebyshev's inequality, Hoeffding's inequality, Chernoff bound.

**condition number** The condition number  $\kappa(\mathbf{Q}) \geq 1$  of a positive definite matrix  $\mathbf{Q} \in \mathbb{R}^{d \times d}$  is the ratio  $\alpha/\beta$ , where  $\alpha$  and  $\beta$  denote the largest and smallest eigenvalues of  $\mathbf{Q}$ , respectively. The condition number is useful for the analysis of ML methods. The computational complexity of gradient-based methods for linear regression depends crucially on the condition number of the matrix  $\mathbf{Q} = \mathbf{X}\mathbf{X}^T$ , where  $\mathbf{X}$  is the feature matrix of the training set. These methods converge faster when the condition number  $\kappa(\mathbf{Q})$  is close to 1.

See also: matrix, eigenvalue, gradient-based method.

**conditional expectation** Consider a numeric RV  $\mathbf{x} \in \mathbb{R}^d$  defined on a probability space  $(\Omega, \Sigma, \mathbb{P})$ . Let  $\Sigma' \subseteq \Sigma$  be a (sub-) $\sigma$ -algebra that represents partial information about the outcome of a random experiment. The conditional expectation of  $\mathbf{x}$  given (or conditioned on)  $\Sigma'$ , denoted by  $\mathbb{E}\{\mathbf{x} | \Sigma'\}$ , is a numeric RV that [6, Sec. 34], [17]: 1) is measurable with respect to  $\Sigma'$ ; and 2) satisfies

$$\int_{\mathcal{A}} \mathbb{E}\{\mathbf{x} | \Sigma'\} d\mathbb{P} = \int_{\mathcal{A}} \mathbf{x} d\mathbb{P} \quad \text{for any } \mathcal{A} \in \Sigma'.$$

Intuitively,  $\mathbb{E}\{\mathbf{x} | \Sigma'\}$  summarizes the average value of  $\mathbf{x}$  using only information contained in the (typically smaller)  $\sigma$ -algebra  $\Sigma'$  [6], [18], [19].

See also: probability space,  $\sigma$ -algebra, expectation.

**conditional probability distribution** Consider a stochastic process consisting of two RVs  $\mathbf{x}$  and  $y$  with probability distribution  $\mathbb{P}^{(\mathbf{x},y)}$ . The

conditional probability distribution of  $y$  given (or conditioned on)  $\mathbf{x}$  is denoted by  $\mathbb{P}^{(y|\mathbf{x})}$ . It is defined via the conditional expectations of the indicator functions of measurable sets in the  $\sigma$ -algebra generated by the RV  $y$  [6, Sec. 34], [20].

See also: probability distribution, conditional expectation.

**conditional probability mass function (conditional pmf)** Consider two discrete RVs  $y$  and  $x$  defined on the same probability space  $(\Omega, \mathcal{F}, \mathbb{P}(\cdot))$ . The conditional pmf of  $y$  given (or conditioned on)  $x$  is denoted by  $p^{(y|x)}(\cdot | \cdot)$  and is defined by

$$p^{(y|x)}(y' | x') := \mathbb{P}(y = y' | x = x')$$

for all realizations  $y', x'$  with  $\mathbb{P}(x = x') > 0$ . Equivalently, the conditional pmf can be expressed using conditional expectation as

$$p^{(y|x)}(y' | x') = \mathbb{E}\{\mathbb{I}_{y=y'} | \Sigma(x)\}(x'),$$

where  $\Sigma(x)$  denotes the  $\sigma$ -algebra generated by the RV  $x$ .

See also: probability space, pmf, conditional expectation.

**conjugate prior** A prior distribution  $\mathbb{P}^{(\mathbf{w})}$  over model parameters  $\mathbf{w}$  is called conjugate to a conditional probability distribution (referred to as the likelihood model)  $\mathbb{P}^{(\mathbf{x}|\mathbf{w})}$  if the resulting posterior distribution  $\mathbb{P}^{(\mathbf{w}|\mathbf{x})}$  belongs to the same parametric family as  $\mathbb{P}^{(\mathbf{w})}$  [21], [22]. One important example of a conjugate prior and likelihood model is obtained from the multivariate normal distribution. In particular, consider the prior  $\mathbb{P}^{(\mathbf{w})}$  is a multivariate normal distribution  $\mathcal{N}(\boldsymbol{\mu}, \mathbf{C})$  with some mean  $\boldsymbol{\mu}$  and

covariance matrix  $\mathbf{C}$ . For a likelihood model  $\mathbb{P}^{(\mathbf{x}|\mathbf{w})}$  that is a multivariate normal distribution with mean  $\mathbf{w}$  and covariance  $\mathbf{C}$ , the resulting posterior  $\mathbb{P}^{(\mathbf{w}|\mathbf{x})}$  is also a multivariate normal distribution [22, Appendix B].

See also: conditional probability distribution, multivariate normal distribution, probability distribution.

**conjugate transpose** The conjugate transpose of a matrix is obtained by transposing the matrix and taking the complex conjugate of each entry. For a matrix  $\mathbf{A} \in \mathbb{C}^{m \times d}$ , its conjugate transpose is denoted by  $\mathbf{A}^H \in \mathbb{C}^{d \times m}$  and is defined entrywise by

$$(\mathbf{A}^H)_{j,r} = \overline{(\mathbf{A})_{r,j}},$$

where  $\overline{(\cdot)}$  denotes complex conjugation.

See also: transpose, matrix, Hermitian.

**connected** An undirected graph  $\mathcal{G} = (\mathcal{V}, \mathcal{E})$  is connected if, for every non-empty subset  $\mathcal{V}' \subset \mathcal{V}$ , we can find at least one edge connecting a node in  $\mathcal{V}'$  with some node in  $\mathcal{V} \setminus \mathcal{V}'$ . We illustrate two examples of undirected graphs in Fig. 9.

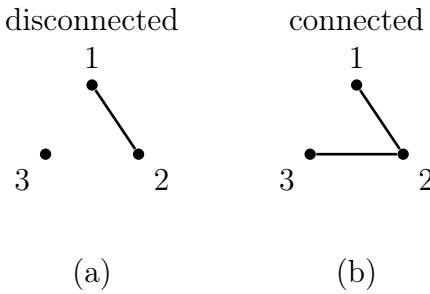


Fig. 9. (a) Disconnected graph. (b) Connected graph.

See also: undirected graph, algebraic connectivity.

**constrained optimization problem** A constrained optimization problem is an optimization problem that involves constraints on the optimization variable. For example, consider the optimization problem

$$\min_{\mathbf{w} \in \mathcal{W}} f(\mathbf{w}),$$

where the objective function  $f : \mathbb{R}^d \rightarrow \mathbb{R}$  is to be minimized over a constraint set  $\mathcal{W} \subseteq \mathbb{R}^d$ . Here, the constraint set  $\mathcal{W}$  restricts the feasible values of the optimization variable  $\mathbf{w}$ . If  $\mathcal{W} = \mathbb{R}^d$ , the above optimization problem reduces to an unconstrained optimization problem.

See also: optimization problem.

**continuous** A function  $f : \mathbb{R}^d \rightarrow \mathbb{R}$  is continuous at a point  $\mathbf{x}' \in \mathbb{R}^d$  if, for every  $\epsilon > 0$ , there is a  $\delta > 0$  such that, for all  $\mathbf{x} \in \mathbb{R}^d$  with  $\|\mathbf{x} - \mathbf{x}'\|_2 < \delta$ , it holds that  $|f(\mathbf{x}) - f(\mathbf{x}')| < \epsilon$  [2]. In other words, we can make  $f(\mathbf{x})$  arbitrarily close to  $f(\mathbf{x}')$  by choosing  $\mathbf{x}$  sufficiently close to  $\mathbf{x}'$ .

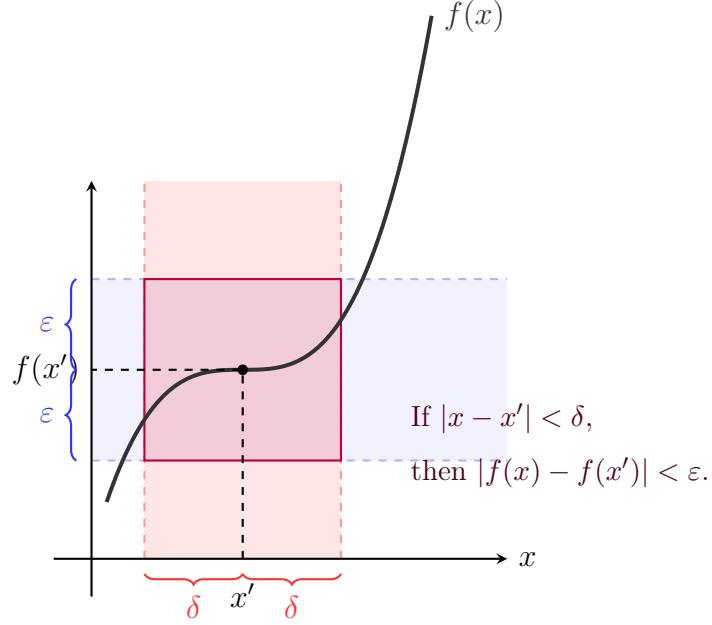


Fig. 10. Function  $f(x) = 0.3(x - 2)^3 + 2.5$ , continuous at every  $x'$ .

If  $f$  is continuous at every point  $\mathbf{x}' \in \mathbb{R}^d$ , then  $f$  is said to be continuous on  $\mathbb{R}^d$ . The notion of a continuous function can be naturally extended to functions between general metric spaces [2].

See also: Euclidean space, metric.

**contractive operator** An operator  $\mathcal{F} : \mathcal{V} \rightarrow \mathcal{V}$  on a normed space  $(\mathcal{V}, \|\cdot\|)$  is a contraction (or contractive) if, for some  $\kappa \in [0, 1)$ , [23], [24]

$$\|\mathcal{F}\mathbf{w} - \mathcal{F}\mathbf{w}'\| \leq \kappa \|\mathbf{w} - \mathbf{w}'\| \quad \text{holds for any } \mathbf{w}, \mathbf{w}' \in \mathcal{V}.$$

The notion of a contractive operator generalizes naturally from normed spaces to arbitrary metric spaces [23].

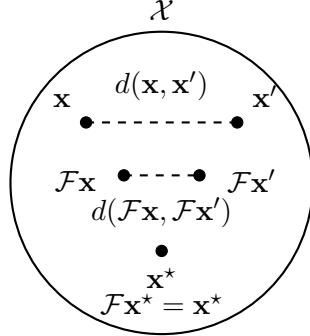


Fig. 11. Contractive operator  $\mathcal{F} : \mathcal{X} \rightarrow \mathcal{X}$  with a unique fixed point  $\mathbf{x}^*$  satisfying  $\mathcal{F}\mathbf{x}^* = \mathbf{x}^*$ . For any two points  $\mathbf{x}, \mathbf{x}'$  in the same space, the distance between their images  $\mathcal{F}\mathbf{x}$  and  $\mathcal{F}\mathbf{x}'$  is strictly smaller.

Intuitively, a contractive operator brings any two points from its domain closer together by at least a factor of  $\kappa$ .

See also: operator, metric space, Banach's fixed-point theorem, Bellman operator.

**convergence** Consider a sequence  $(a_r)_{r \in \mathbb{N}}$  with numeric values  $a_r \in \mathbb{R}$ . This sequence is said to converge to a value  $a^*$  if the values  $a_r$  become arbitrarily close to  $a^*$  for sufficiently large indices  $r$ . Mathematically speaking, the sequence converges to  $a^*$  if [1], [2]

$$\forall \epsilon > 0, \exists N \in \mathbb{N} : r > N \Rightarrow |a_r - a^*| < \epsilon.$$

We denote the convergence of a sequence to  $a^*$  by

$$\lim_{r \rightarrow \infty} a_r = a^*.$$

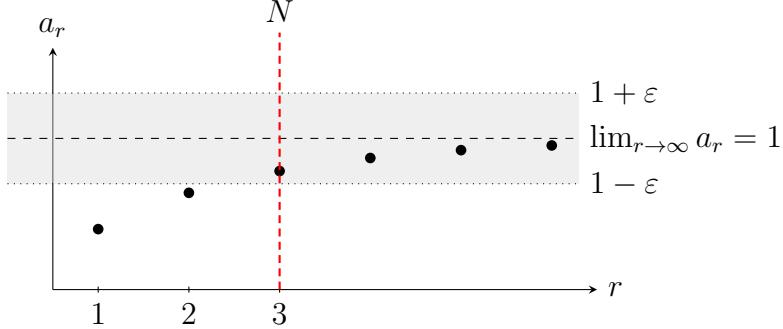


Fig. 12. Real-valued sequence  $(a_r)_{r \in \mathbb{N}}$  converging to the limit  $a^* = 1$ .

The concept of convergence of a real-valued sequence (where  $\mathcal{A} = \mathbb{R}$ ) extends naturally to a sequence in an arbitrary metric space  $\mathcal{A}$ . Indeed, we just need to replace the absolute difference  $|a_r - a^*|$  with the metric  $d(a_r, a^*)$ . Note that a sequence can only converge if it is a Cauchy sequence [2]. However, not every Cauchy sequence is converging unless the underlying metric space is complete.

See also: sequence, metric space, Cauchy sequence.

**convex** A subset  $\mathcal{C} \subseteq \mathbb{R}^d$  of the Euclidean space  $\mathbb{R}^d$  is referred to as convex if it contains the line segment between any two points  $\mathbf{x}, \mathbf{y} \in \mathcal{C}$  in that set, i.e.,

$$\alpha\mathbf{x} + (1 - \alpha)\mathbf{y} \in \mathcal{C} \quad \text{for any } \alpha \in [0, 1].$$

Similarly, a function  $f: \mathbb{R}^d \rightarrow \mathbb{R}$  is convex if its epigraph  $\{(\mathbf{w}^T, t)^T \in \mathbb{R}^{d+1} : t \geq f(\mathbf{w})\}$  is a convex set [25]. We illustrate one example of a convex set and a convex function in Fig. 13.



Fig. 13. (a) Convex set  $\mathcal{C} \subseteq \mathbb{R}^d$ . (b) Convex function  $f : \mathbb{R}^d \rightarrow \mathbb{R}$ .

See also: Euclidean space, function, epigraph.

**convex optimization** Convex optimization studies the formulation, properties, and efficient solution methods for convex optimization problems [25]. A convex optimization problem (defined on the Euclidean space  $\mathbb{R}^d$ ) consists of a convex objective function  $f : \mathbb{R}^d \rightarrow \mathbb{R}$  and a convex constraint set  $\mathcal{C}$  for the optimization variable  $\mathbf{w}$ . It can be written compactly as [25]

$$\min_{\mathbf{w} \in \mathcal{C}} f(\mathbf{w}).$$

Alternatively, a convex optimization problem can be expressed in terms of convex constraint functions  $g_1, \dots, g_k$  as

$$\begin{aligned} & \min_{\mathbf{w} \in \mathbb{R}^d} f(\mathbf{w}) \\ \text{s.t. } & g_r(\mathbf{w}) \leq 0, \quad r = 1, \dots, k. \end{aligned} \tag{1}$$

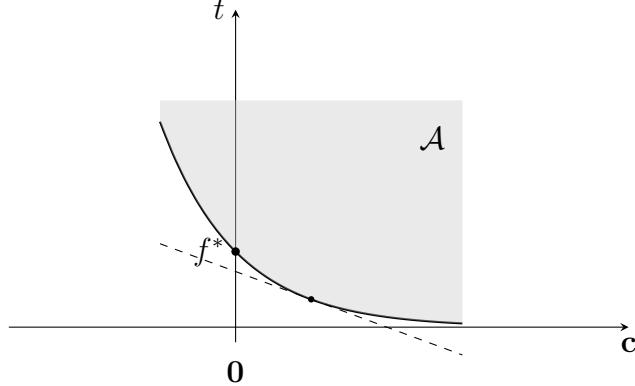


Fig. 14. Convex optimization problem (1) represented by a set  $\mathcal{A}$  that consists of objective values  $t$  and constraint values  $\mathbf{c} = (c_1, \dots, c_d)^T$  that are achievable, i.e.,  $f(\mathbf{w}) \leq t, g_1(\mathbf{w}) \leq c_1, \dots, g_k(\mathbf{w}) \leq c_k$  for some  $\mathbf{w} \in \mathbb{R}^d$ . The optimal value  $f^*$  of the optimization problem is the smallest  $t$  for which  $(\mathbf{0}, t) \in \mathcal{A}$ .

The formulation (1) lends, in turn, to the following epigraph form [25, Sec. 5.3]:

$$\inf \{t \in \mathbb{R} : (\mathbf{0}, t) \in \mathcal{A}\}$$

with the set

$$\begin{aligned} \mathcal{A} := \{(\mathbf{c}, t) \in \mathbb{R}^d \times \mathbb{R} : f(\mathbf{w}) \leq t, \\ g_r(\mathbf{w}) \leq c_r, r = 1, \dots, k \text{ for some } \mathbf{w} \in \mathbb{R}^d\}. \end{aligned}$$

It can be shown that, since  $f, g_1, \dots, g_k$  are convex functions,  $\mathcal{A}$  is a convex set [25, Ch. 2]. The set  $\mathcal{A}$  fully characterizes the optimization problem (1) and can be interpreted as the epigraph of the objective function  $f$  over the feasible region defined by the constraint functions

$g_1, \dots, g_k$ .

See also: convex, optimization problem, optimization method.

**convex optimization problem** See convex optimization.

**countable** A set is called countable if its elements can be put into a one-to-one correspondence with the natural numbers  $\mathbb{N} = \{1, 2, 3, \dots\}$  or with a finite subset of  $\mathbb{N}$  [26]. Equivalently, a set  $\mathcal{A}$  is countable if there exists an injective function  $f : \mathcal{A} \rightarrow \mathbb{N}$ .

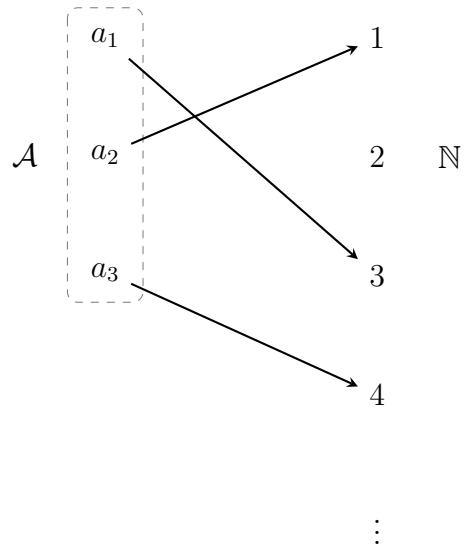


Fig. 15. Injective function mapping the elements of a finite set  $\mathcal{A}$  to the natural numbers  $\mathbb{N}$ , which implies that  $\mathcal{A}$  is countable.

Typical examples include the set of integers  $\mathbb{Z}$  and rational numbers  $\mathbb{Q}$ . In contrast, the set of real numbers  $\mathbb{R}$  is not countable, meaning no such one-to-one correspondence with  $\mathbb{N}$  exists.

See also: injective, function.

## Courant–Fischer–Weyl min–max characterization (CFW min–max characterization)

Consider a psd matrix  $\mathbf{Q} \in \mathbb{R}^{d \times d}$  with eigenvalue decomposition (EVD) (or spectral decomposition), i.e.,

$$\mathbf{Q} = \sum_{j=1}^d \lambda_j \mathbf{u}^{(j)} (\mathbf{u}^{(j)})^T.$$

Here, we use the ordered (in ascending order) eigenvalues

$$\lambda_1 \leq \dots \leq \lambda_n.$$

The CFW min–max characterization [3, Th. 8.1.2] represents the eigenvalues of  $\mathbf{Q}$  as the solutions to certain optimization problems. See also: psd, matrix, EVD, eigenvalue, optimization problem.

**covariance** The covariance between two real-valued RVs  $x$  and  $y$ , defined on a common probability space, measures their linear dependence. It is defined as

$$\text{cov}(x, y) = \mathbb{E}\{(x - \mathbb{E}\{x\})(y - \mathbb{E}\{y\})\}.$$

A positive covariance indicates that  $x$  and  $y$  tend to increase together, while a negative covariance suggests that one tends to increase as the other decreases. If  $\text{cov}(x, y) = 0$ , the RVs are said to be uncorrelated, though not necessarily statistically independent. See Fig. 16 for visual illustrations.

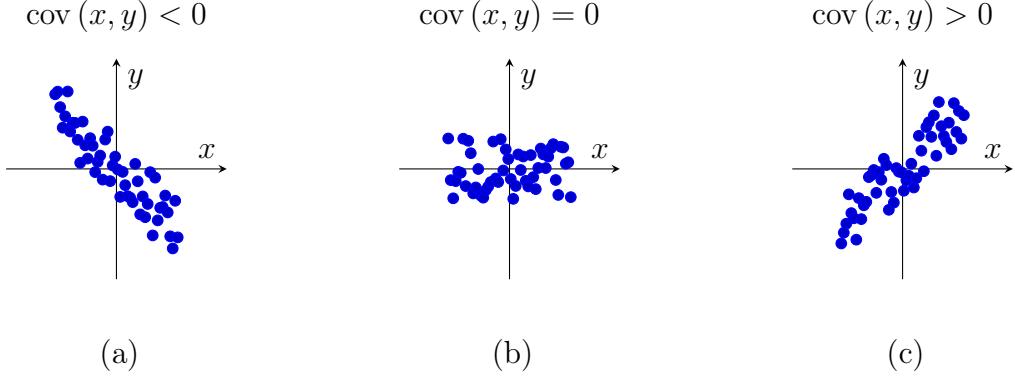


Fig. 16. Scatterplots illustrating realizations from three different probabilistic models for two RVs with different covariance values. (a) Negative. (b) Zero. (c) Positive.

See also: probabilistic model, expectation.

**covariance function** A covariance function characterizes the second-order dependence structure of a stochastic process. For a real-valued stochastic process  $\{x(t)\}_{t \in \mathcal{I}}$  with finite second moments, the covariance function is defined as [27]

$$\begin{aligned} \text{cov}^{(x)}(t, t') &:= \text{cov}(x(t), x(t')) \\ &= \mathbb{E}[(x(t) - \mathbb{E}[x(t)])(x(t') - \mathbb{E}[x(t')])], \quad t, t' \in \mathcal{I}. \end{aligned}$$

See also: covariance, function, stochastic process.

**covariance matrix** The covariance matrix of an RV  $\mathbf{x} \in \mathbb{R}^d$  is defined as the expectation (if it exists):

$$\mathbf{C}^{(\mathbf{x})} := \mathbb{E} \left\{ (\mathbf{x} - \mathbb{E}\{\mathbf{x}\})(\mathbf{x} - \mathbb{E}\{\mathbf{x}\})^T \right\}.$$

See also: covariance, matrix, RV.

**cumulative distribution function (cdf)** The cdf  $F^{(x)}(\eta)$  of a real-valued RV  $x$  is [28], [29]

$$F^{(x)}(\eta) := \mathbb{P}(x \leq \eta).$$

See also: RV, pdf, probability distribution.

**denoising autoencoder** A denoising autoencoder extends the basic autoencoder by perturbing (e.g., by intentionally adding noise to) the input vector (or feature vector) before feeding it into the autoencoder itself during the training [30, Sec. 14.2.2]. Once trained, we can use a denoising autoencoder to denoise a corrupted representation of a data point (see Fig. 17).

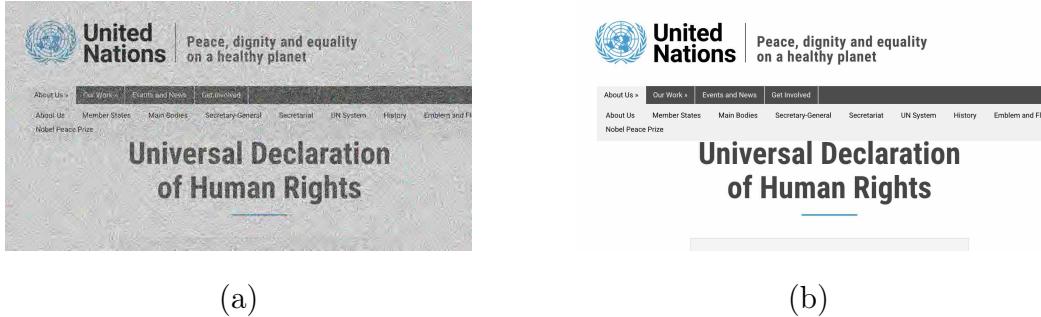


Fig. 17. A denoising autoencoder reads in (a) a corrupted (i.e., noisy) representation of a data point and computes a prediction for (b) the clean representation.

See also: autoencoder, diffusion method.

**derivative** See partial derivative.

**determinant** The determinant  $\det(\mathbf{A})$  of a square matrix  $\mathbf{A} = (\mathbf{a}^{(1)}, \dots, \mathbf{a}^{(d)}) \in \mathbb{R}^{d \times d}$  is a function of its columns  $\mathbf{a}^{(1)}, \dots, \mathbf{a}^{(d)} \in \mathbb{R}^d$ , i.e., it satisfies the following properties [31]:

- Normalized:

$$\det(\mathbf{I}) = 1.$$

- Multilinear:

$$\begin{aligned} \det(\mathbf{a}^{(1)}, \dots, \alpha\mathbf{u} + \beta\mathbf{v}, \dots, \mathbf{a}^{(d)}) &= \alpha \det(\mathbf{a}^{(1)}, \dots, \mathbf{u}, \dots, \mathbf{a}^{(d)}) \\ &\quad + \beta \det(\mathbf{a}^{(1)}, \dots, \mathbf{v}, \dots, \mathbf{a}^{(d)}). \end{aligned}$$

- Antisymmetric:

$$\det(\dots, \mathbf{a}^{(j)}, \dots, \mathbf{a}^{(j')}, \dots) = -\det(\dots, \mathbf{a}^{(j')}, \dots, \mathbf{a}^{(j)}, \dots).$$

We can interpret a matrix  $\mathbf{A}$  as a linear transformation on  $\mathbb{R}^d$ . The determinant  $\det(\mathbf{A})$  characterizes how volumes in  $\mathbb{R}^d$  (and their orientation) are altered by this transformation (see Fig. 18) [3], [32]. In particular,  $\det(\mathbf{A}) > 0$  preserves orientation,  $\det(\mathbf{A}) < 0$  reverses orientation, and  $\det(\mathbf{A}) = 0$  collapses volume entirely, indicating that  $\mathbf{A}$  is non-invertible. The determinant also satisfies  $\det(\mathbf{AB}) = \det(\mathbf{A}) \cdot \det(\mathbf{B})$ , and if  $\mathbf{A}$  is diagonalizable with eigenvalues  $\lambda_1, \dots, \lambda_d$ , then  $\det(\mathbf{A}) = \prod_{j=1}^d \lambda_j$  [33]. For the special cases  $d = 2$  (i.e., two-dimensional or 2-D) and  $d = 3$  (i.e., three-dimensional or 3-D), the determinant can be interpreted as an oriented area or volume spanned by the column vectors of  $\mathbf{A}$ .

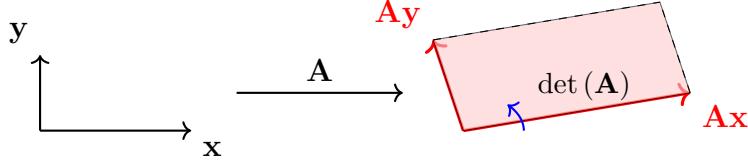


Fig. 18. We can interpret a square matrix  $\mathbf{A}$  as a linear transformation of  $\mathbb{R}^d$  into itself. The determinant  $\det(\mathbf{A})$  characterizes how this transformation alters an oriented volume.

See also: eigenvalue, inverse matrix.

**diagonalizable** A square matrix  $\mathbf{A} \in \mathbb{C}^{d \times d}$  is called diagonalizable if it is similar to a diagonal matrix [33], [34]. Formally,  $\mathbf{A}$  is diagonalizable if there exists an invertible matrix  $\mathbf{P} \in \mathbb{C}^{d \times d}$  such that

$$\mathbf{A} = \mathbf{P}\mathbf{D}\mathbf{P}^{-1},$$

where  $\mathbf{D} \in \mathbb{C}^{d \times d}$  is a diagonal matrix whose main diagonal entries are the eigenvalues of  $\mathbf{A}$ . A matrix  $\mathbf{A} \in \mathbb{R}^{d \times d}$  is diagonalizable if and only if it has  $d$  linearly independent eigenvectors [33].

See also: matrix, eigenvalue, EVD.

**differentiable** A real-valued function  $f : \mathbb{R}^d \rightarrow \mathbb{R}$  is differentiable if it can be approximated locally at any point by a linear function. The local linear approximation at the point  $\mathbf{x}$  is determined by the gradient  $\nabla f(\mathbf{x})$  [2].

See also: function, gradient.

**differential entropy** For an RV  $\mathbf{x} \in \mathbb{R}^d$  with a pdf  $p^{(x)}(\cdot)$ , the differential

entropy is defined as [35]

$$h(\mathbf{x}) := - \int_{\mathbf{x}' \in \mathbb{R}^d} \log p(\mathbf{x}') p^{(\mathbf{x})}(\mathbf{x}') d\mathbf{x}'.$$

Differential entropy can be negative and lacks some properties of entropy for discrete-valued RVs, such as invariance under a change of variables [35]. Among all RVs with a given mean  $\boldsymbol{\mu}$  and covariance matrix  $\mathbf{C}$ ,  $h(\mathbf{x})$  is maximized by  $\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}, \mathbf{C})$ .

See also: uncertainty, probabilistic model.

**dimension** The dimension  $\dim \mathcal{A}$  of a vector space  $\mathcal{A}$  is the cardinality of any basis of  $\mathcal{A}$  [36]. Strictly speaking, this definition applies only to finite-dimensional vector spaces, i.e., those that possess a finite basis.

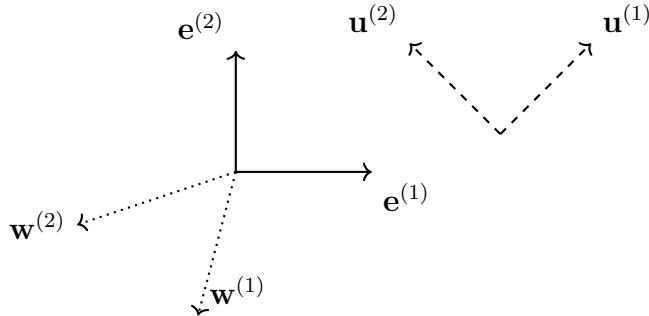


Fig. 19. Three bases,  $\{\mathbf{e}^{(1)}, \mathbf{e}^{(2)}\}$ ,  $\{\mathbf{u}^{(1)}, \mathbf{u}^{(2)}\}$ ,  $\{\mathbf{w}^{(1)}, \mathbf{w}^{(2)}\}$ , for the vector space  $\mathbb{R}^2$ .

For such spaces, all bases have the same cardinality, which is the dimension of the space [34, Ch. 2].

See also: vector space, basis.

**directed acyclic graph (DAG)** A DAG is a directed graph that contains no directed cycles. Formally, a DAG  $\mathcal{G} = (\mathcal{V}, \mathcal{E})$  satisfies that, for any

sequence of distinct nodes  $(i_1, \dots, i_k)$ , the presence of directed edges  $(i_1, i_2), (i_2, i_3), \dots, (i_{k-1}, i_k)$  implies that  $(i_k, i_1) \notin \mathcal{E}$ .



Fig. 20. (a) A DAG defined on three nodes  $\mathcal{V} = \{1, 2, 3\}$ . (b) Another directed graph on the same nodes that is not a DAG, since it contains a directed cycle.

The absence of directed cycles allows for a topological ordering of nodes such that all edges point from earlier to later nodes in this order. Several ML models, such as ANNs or decision trees, are naturally represented as DAGs.

See also: directed graph, ANN, decision tree.

**directed cycle** A directed cycle in a directed graph  $\mathcal{G} = (\mathcal{V}, \mathcal{E})$  is a sequence of distinct nodes  $(i_1, i_2, \dots, i_k)$  such that  $(i_1, i_2), (i_2, i_3), \dots, (i_{k-1}, i_k), (i_k, i_1) \in \mathcal{E}$ . In a directed cycle, following the direction of each edge eventually leads back to the starting node, creating a closed loop.

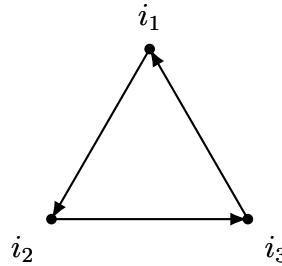


Fig. 21. A directed cycle consisting of three nodes connected in a closed loop.

The presence of a directed cycle prevents a directed graph from being a directed acyclic graph (DAG).

See also: directed graph, DAG.

**directed graph** A directed graph contains edges that have an orientation (or direction). Mathematically, a directed graph  $\mathcal{G} = (\mathcal{V}, \mathcal{E})$  consists of nodes  $\mathcal{V}$  and a set  $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$  of directed edges.

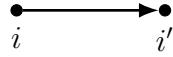


Fig. 22. The edges of a directed graph have an orientation (or direction). We can indicate the orientation by an arrow head.

We can represent a directed edge from node  $i \in \mathcal{V}$  to node  $i' \in \mathcal{V}$  by an ordered pair  $(i, i')$ . Directed graphs are widely used to model interconnected systems or networks, such as transportation systems, electronic circuits, and biological processes [37].

See also: graph.

**discrete random variable (discrete RV)** An RV, i.e., a function that maps the outcomes of a random experiment to elements of a measurable space  $\mathcal{X}$ , is referred to as discrete if its value space  $\mathcal{X}$  is countable [6].

See also: RV, probability, probability distribution.

**distance** The distance between two points in a metric space is the value of the metric evaluated at those points [38].

See also: metric space, Euclidean distance.

**domain** The domain of a function  $f : \mathcal{U} \rightarrow \mathcal{V}$  is the set  $\mathcal{U}$  from which  $f$  takes its inputs.

See also: function, co-domain, map.

**dual norm** Every norm  $\|\cdot\|$  defined on a Euclidean space  $\mathbb{R}^d$  has an associated dual norm, which is denoted by  $\|\cdot\|_*$  and defined as  $\|\mathbf{y}\|_* := \sup_{\|\mathbf{x}\| \leq 1} \mathbf{y}^T \mathbf{x}$ . The dual norm measures the largest possible inner product between  $\mathbf{y}$  and any vector in the unit ball of the original norm. For further details, see [25, Sec. A.1.6].

See also: norm, Euclidean space, vector.

**dual problem** Consider a constrained optimization problem, which we refer to as primal problem in what follows, of the following form:

$$\begin{aligned} & \min_{\mathbf{x} \in \mathbb{R}^d} f(\mathbf{x}) \\ \text{s.t. } & g_i(\mathbf{x}) \leq 0, \quad i = 1, \dots, k \\ & h_j(\mathbf{x}) = 0, \quad j = 1, \dots, l, \end{aligned}$$

and its associated Lagrange dual function  $q(\boldsymbol{\lambda}, \boldsymbol{\nu})$ . For any  $\boldsymbol{\lambda} \geq \mathbf{0}$ ,  $\boldsymbol{\nu} \in \mathbb{R}^l$ , and any  $\mathbf{x}$  that satisfies the constraints of the primal problem [25, Ch. 5],

$$q(\boldsymbol{\lambda}, \boldsymbol{\nu}) \leq f(\mathbf{x}).$$

Making this lower bound maximally tight amounts to the following optimization problem:

$$\max_{\boldsymbol{\lambda} \geq \mathbf{0}, \boldsymbol{\nu}} q(\boldsymbol{\lambda}, \boldsymbol{\nu}).$$

This optimization problem is referred to as the dual problem associated with the original primal problem. Fig. 23 illustrates the dual problem

for an optimization problem with a single inequality constraint. This optimization problem can be characterized by the set  $\mathcal{A} = \{(u, t) : f(\mathbf{x}) \leq t, g_1(\mathbf{x}) \leq u \text{ for some } \mathbf{x} \in \mathbb{R}^d\}$  [25, Sec. 5.3]. The dual problem amounts to finding a supporting hyperplane for the set  $\mathcal{A}$  with the largest vertical intercept.

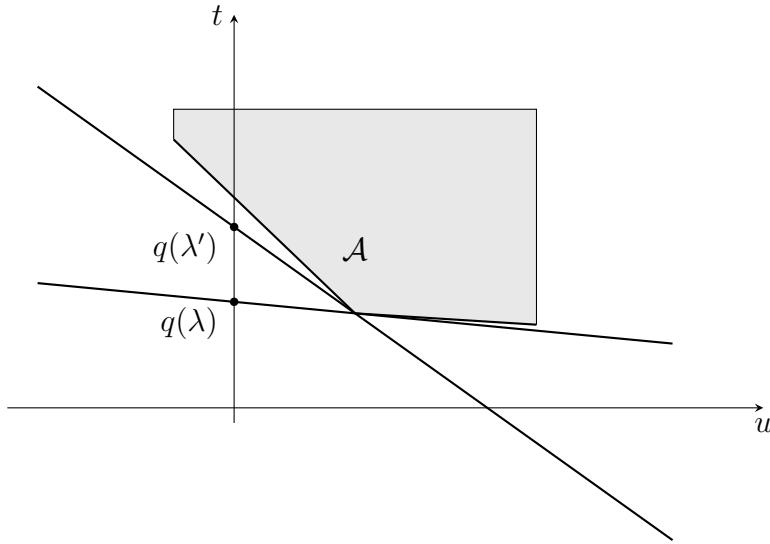


Fig. 23. The dual problem of a constrained optimization problem is to find a supporting hyperplane with the largest vertical intercept [25, Sec. 5.3].

See also: optimization problem, Lagrangian.

**duality gap** Consider a constrained optimization problem with Lagrangian

$$L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\nu}) = f_0(\mathbf{x}) + \sum_{i=1}^k \lambda_i f_i(\mathbf{x}) + \sum_{j=1}^l \nu_j g_j(\mathbf{x}).$$

The dual problem is defined as

$$\max_{\boldsymbol{\lambda} \in \mathbb{R}_+^k, \boldsymbol{\nu} \in \mathbb{R}^l} \underbrace{q(\boldsymbol{\lambda}, \boldsymbol{\nu})}_{:=\inf_{\mathbf{x}} L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\nu})} .$$

Let  $p^*$  denote the optimal value of the primal problem and  $d^*$  the optimal value of the associated dual problem. The duality gap is defined as

$$p^* - d^* \geq 0.$$

The duality gap is always nonnegative, even for non-convex and non-smooth optimization problems. When the duality gap is zero, i.e.,  $p^* = d^*$ , strong duality is said to hold [25, Ch. 5].

See also: optimization problem, dual problem.

**eigenvalue** We refer to a number  $\lambda \in \mathbb{R}$  as an eigenvalue of a square matrix  $\mathbf{A} \in \mathbb{R}^{d \times d}$  if there exists a nonzero vector  $\mathbf{x} \in \mathbb{R}^d \setminus \{\mathbf{0}\}$  such that  $\mathbf{Ax} = \lambda\mathbf{x}$  (see Fig. 24).

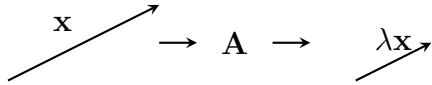


Fig. 24. This vector is the eigenvector corresponding to the eigenvalue  $\lambda$ .

See also: matrix, eigenvector.

**eigenvalue decomposition (EVD)** An EVD for a square matrix  $\mathbf{A} \in \mathbb{R}^{d \times d}$  is a factorization of the form

$$\mathbf{A} = \mathbf{V}\Lambda\mathbf{V}^{-1}.$$

The columns of the matrix  $\mathbf{V} = (\mathbf{v}^{(1)}, \dots, \mathbf{v}^{(d)})$  are the eigenvectors of the matrix  $\mathbf{A}$ . The diagonal matrix  $\Lambda = \text{diag}\{\lambda_1, \dots, \lambda_d\}$  contains the eigenvalues  $\lambda_j$  corresponding to the eigenvectors  $\mathbf{v}^{(j)}$ . Matrices that

allow for an EVD are referred to as diagonalizable.

See also: matrix, eigenvector, eigenvalue, diagonalizable.

**eigenvector** An eigenvector of a matrix  $\mathbf{A} \in \mathbb{R}^{d \times d}$  is a nonzero vector  $\mathbf{x} \in \mathbb{R}^d \setminus \{\mathbf{0}\}$  such that  $\mathbf{Ax} = \lambda\mathbf{x}$  with some eigenvalue  $\lambda$ . Eigenvectors to the eigenvalue  $\lambda$  span a subspace of  $\mathbb{R}^d$ , namely the nullspace of  $\mathbf{A} - \lambda\mathbf{I}$ .

See also: matrix, vector, eigenvalue.

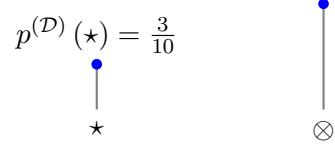
**empirical distribution** Consider a dataset  $\mathcal{D} = \{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$  consisting of  $m$  distinct data points, each characterized by the feature vector  $\mathbf{x}^{(r)} \in \mathcal{X}$  for  $r = 1, \dots, m$ . For a given  $\sigma$ -algebra  $\Sigma$  over the feature space  $\mathcal{X}$ , the empirical distribution of  $\mathcal{D}$  is the probability distribution  $\mathbb{P}^{(\mathcal{D})}$  defined via

$$\mathbb{P}^{(\mathcal{D})}(\mathcal{A}) = (1/m) |\{r : \mathbf{x}^{(r)} \in \mathcal{A}\}| \quad \text{for any event } \mathcal{A} \in \Sigma.$$

In other words, the empirical distribution assigns to any measurable set  $\mathcal{A} \in \Sigma$  the fraction of data points in  $\mathcal{D}$  that fall into  $\mathcal{A}$ . If the feature space is ordered, the empirical distribution can also be characterized by its empirical cumulative distribution function (cdf):

$$F^{(\mathcal{D})}(\mathbf{x}) = (1/m) |\{r : \mathbf{x}^{(r)} \preceq \mathbf{x}\}| \quad \text{for any } \mathbf{x} \in \mathcal{X},$$

where  $\preceq$  denotes the ordering relation on  $\mathcal{X}$ .



$$\mathcal{D} = (\star, \star, \star, \otimes, \otimes, \otimes, \otimes, \otimes, \otimes, \otimes)$$

Fig. 25. A dataset  $\mathcal{D}$  consisting of  $m = 10$  data points, each characterized by a value from the finite feature space  $\mathcal{X} = \{\star, \otimes\}$ . The empirical pmf  $p^{(\mathcal{D})}(\mathbf{x})$  assigns to each possible value  $\mathbf{x} \in \mathcal{X}$  the fraction of data points in  $\mathcal{D}$  whose feature takes on this value. Here, three out of ten data points take on the feature value  $\star$ , resulting in  $p^{(\mathcal{D})}(\star) = 3/10$ .

If the feature space  $\mathcal{X}$  is finite, the empirical distribution of  $\mathcal{D}$  can also be characterized by the empirical pmf:

$$p^{(\mathcal{D})}(\mathbf{x}) = (1/m)|\{r : \mathbf{x}^{(r)} = \mathbf{x}\}| \quad \text{for any } \mathbf{x} \in \mathcal{X}.$$

See also:  $\sigma$ -algebra, probability distribution.

**entropy** Entropy quantifies the uncertainty or unpredictability associated with an RV [35]. For a discrete RV  $x$  taking on values in a finite set  $\mathcal{S} = \{x_1, \dots, x_k\}$  with a pmf  $p^{(x)}(x_c)$  ( $= \mathbb{P}(x = x_c)$ ), the entropy is defined as [35]

$$H(x) := - \sum_{c=1}^k p^{(x)}(x_c) \log p^{(x)}(x_c).$$

For a given set of values  $\mathcal{S}$ , the entropy is maximized for a uniformly distributed RV, where  $p^{(x)}(x_c) = 1/k$ . The minimal entropy, which is

zero, is obtained when  $p^{(x)}(x_c) = 1$  for some  $x_c \in \mathcal{S}$ . Differential entropy generalizes the concept of entropy from discrete RVs to continuous RVs. See also: uncertainty, probabilistic model.

**epigraph** The epigraph of a real-valued function  $f : \mathbb{R}^n \rightarrow \mathbb{R} \cup \{+\infty\}$  is the set of points lying on or above its graph (see Fig. 26), i.e.,

$$\text{epi}(f) = \{(\mathbf{x}, t) \in \mathbb{R}^n \times \mathbb{R} \mid f(\mathbf{x}) \leq t\}.$$

A function is convex if and only if its epigraph is a convex set [25], [39].

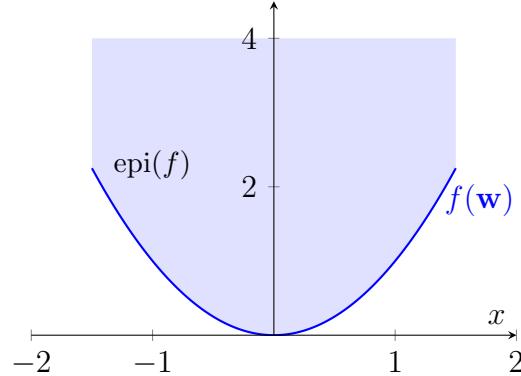


Fig. 26. Epigraph of the function  $f(x) = x^2$  (i.e., the shaded area).

See also: function, convex.

**Erdős–Rényi graph (ER graph)** An ER graph [40], [41] is a probabilistic model for graphs defined over a given node set  $i = 1, \dots, n$ . One way to define the ER graph is via the collection of i.i.d. binary RVs  $b^{\{i,i'\}} \in \{0, 1\}$  for each pair of different nodes  $i, i'$ . A specific realization of an ER graph contains an edge  $\{i, i'\}$  if and only if  $b^{\{i,i'\}} = 1$ . The ER graph is parameterized by the number  $n$  of nodes and the probability

$$\mathbb{P}(b^{(\{i,i'\})} = 1).$$

See also: graph, probabilistic model, i.i.d., RV, realization, probability.

**estimator** Machine learning systems (ML systems) are often analyzed using a probabilistic model  $\mathbb{P}^{(\mathbf{x}; \mathbf{w})}$  for the generation of data  $\mathbf{x}$  [8]. The true value of the model parameters  $\mathbf{w}$  is typically unknown and needs to be estimated. An estimator is a measurable function  $h(\mathbf{x})$  that reads in the data  $\mathbf{x}$  and delivers an estimate (or approximation)  $\hat{\mathbf{w}} = h(\mathbf{x})$  of the true value of the model parameters  $\mathbf{w}$  [42], [43].

See also: model parameter, measurable, function.

**Euclidean distance** The Euclidean distance between two vectors  $\mathbf{w}, \mathbf{w}' \in \mathbb{R}^d$  is the Euclidean norm  $\|\mathbf{w} - \mathbf{w}'\|_2$  of the difference  $\mathbf{w} - \mathbf{w}'$ .

See also: vector, Euclidean space.

**Euclidean norm** The Euclidean norm (or 2-norm, or  $\ell_2$ -norm)  $\|\mathbf{w}\|_2$  of a vector  $\mathbf{w} = (w_1, \dots, w_d) \in \mathbb{R}^d$  is defined as

$$\|\mathbf{w}\|_2 := \sqrt{\sum_{j=1}^d w_j^2}.$$

The Euclidean norm is distinct among all norms on  $\mathbb{R}^d$ , in the sense that it is induced by the inner product  $\mathbf{w}^T \mathbf{v}$  [1], [25], [44]. In other words,  $\|\mathbf{w}\|_2 = \sqrt{\mathbf{w}^T \mathbf{w}}$ .

See also: norm, vector, metric, Euclidean space, Hilbert space.

**Euclidean space** The Euclidean space  $\mathbb{R}^d$  of dimension  $d \in \mathbb{N}$  consists of vectors  $\mathbf{x} = (x_1, \dots, x_d)$  with  $d$  real-valued entries  $x_1, \dots, x_d \in \mathbb{R}$ . Such a Euclidean space is equipped with a geometric structure defined

by the inner product  $\mathbf{x}^T \mathbf{x}' = \sum_{j=1}^d x_j x'_j$  between any two vectors  $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^d$  [2].

See also: vector.

**event** Consider an RV  $\mathbf{x}$ , defined on some probability space, which takes values in a measurable space  $\mathcal{X}$ . An event  $\mathcal{A} \subseteq \mathcal{X}$  is a subset of  $\mathcal{X}$  such that the probability  $\mathbb{P}(\mathbf{x} \in \mathcal{A})$  is well defined. In other words, the preimage  $\mathbf{x}^{-1}(\mathcal{A})$  of an event belongs to the underlying  $\sigma$ -algebra, i.e., the preimage is a measurable subset of the sample space [1], [6], [17]. Roughly speaking, an event represents a set of possible outcomes of some process. One example of such a process could also be the treatment of a health-care patient.

See also: RV, data point, independent and identically distributed assumption (i.i.d. assumption), probabilistic model.

**expectation–maximization (EM)** The EM algorithm [45] is an iterative optimization method for approximately solving certain maximum likelihood optimization problems that are difficult to solve directly [22, Sec. 9.4], [46, Sec. 11.4.7]. To motivate the EM algorithm and explain its construction, consider an ML application involving a single observed data point with feature  $x \in \mathcal{X}$ , where  $\mathcal{X}$  is a finite feature space. The data generation is modeled via a probabilistic model that consists of an RV  $x'$  with a pmf  $p^{(x')}(\cdot; \mathbf{w})$ . Here, the actual model parameters  $\mathbf{w} \in \mathcal{W}$ —used for the data generation via sampling from  $p^{(x')}(\cdot; \mathbf{w})$ —are unknown. A widely used approach for estimating these model parameters is via the

solutions of the following maximum likelihood problem:

$$\min_{\mathbf{w} \in \mathcal{W}} -\log p^{(x')}(\mathbf{x}; \mathbf{w}). \quad (2)$$

For some probabilistic models, such as a Gaussian mixture model (GMM), this optimization problem can be difficult to solve directly. As a work-around, one can often introduce an auxiliary attribute  $y \in \mathcal{Y}$ , generated via some RV  $y'$ , such that the corresponding probabilistic model  $p^{(x',y')}(\cdot, \cdot; \mathbf{w})$  yields the following, much easier maximum likelihood problem:

$$\min_{\mathbf{w} \in \mathcal{W}} -\log p^{(x',y')}(\mathbf{x}, y; \mathbf{w}). \quad (3)$$

The attribute  $y$  is introduced solely to simplify (3), but it is not observed in practice—only the feature  $x$  is available. Thus, we cannot solve (3) directly, as we do not know which value  $y$  to plug into the pmf  $p^{(x',y')}(\mathbf{x}, y; \mathbf{w})$ . The EM method resolves this dilemma by alternating between the following two steps: 1) an E-step in which a “soft” estimate of the auxiliary attribute  $y$  is computed in the form of the posterior distribution  $p^{(y'|x')}(\cdot; \widehat{\mathbf{w}})$  using the current choice  $\widehat{\mathbf{w}}$  for the model parameters; and 2) an M-step in which a surrogate objective function derived from this posterior distribution is minimized. The completion of these two steps constitutes one full iteration of the EM method. In more detail, the E-step produces the following function:

$$Q(\mathbf{w} | \widehat{\mathbf{w}}) := - \sum_{y \in \mathcal{Y}} p^{(y'|x')}(\mathbf{y}; \widehat{\mathbf{w}}) \log \left( p^{(x',y')}(\mathbf{x}, \mathbf{y}; \mathbf{w}) / p^{(y'|x')}(\mathbf{y}; \widehat{\mathbf{w}}) \right),$$

and the M-step minimizes  $Q(\mathbf{w} | \widehat{\mathbf{w}})$  over  $\mathbf{w} \in \mathcal{W}$ . This function satisfies the following two key properties [22, Sec. 9.4], [46, Sec. 11.4.7]: 1) upper

bound  $Q(\mathbf{w} \mid \hat{\mathbf{w}}) \geq -\log p^{(x')}(x; \mathbf{w})$  for all  $\mathbf{w} \in \mathcal{W}$ ; and 2) tightness  $Q(\hat{\mathbf{w}} \mid \hat{\mathbf{w}}) = -\log p^{(x')}(x; \hat{\mathbf{w}})$ . To summarize, during each iteration, EM minimizes an upper-bounding surrogate objective function that is tight at the current iterate  $\hat{\mathbf{w}}$ . Thus, EM is a majorize–minimize (MM) method for approximately solving (2). The above construction and analysis of EM can be extended to more general settings involving multiple data points and infinite feature spaces such as  $\mathbb{R}^d$  (see [46, Sec. 11.4.7] for further details).

See also: maximum likelihood, optimization problem, probabilistic model.

**exponential family** An exponential family is a particular probabilistic model for a given sample space  $\mathcal{X}$  of possible outcomes. It is defined by the specification of a base measure  $h(\mathbf{x})$  and a vector of sufficient statistics  $\mathbf{T}(\mathbf{x})$ . A distribution belongs to the exponential family if its pdf (or pmf) can be written as

$$p(\mathbf{x}; \mathbf{w}) = h(\mathbf{x}) \exp(\mathbf{w}^T \mathbf{T}(\mathbf{x}) - A(\mathbf{w})),$$

where  $\mathbf{w}$  denotes the natural parameter and  $A(\mathbf{w})$  is the log-partition function ensuring normalization  $\int_{\mathbf{x}} p(\mathbf{x}; \mathbf{w}) d\mathbf{x} = 1$ . Exponential family distributions arise as maximum-entropy solutions under constraints on the expected values of the sufficient statistics. Thus, we can view them as the least informative probability distributions for given expectations of the sufficient statistics [35, Ch. 12].

See also: probability distribution, pdf, pmf.

**field** A field  $(\mathbb{F}, +, \cdot)$  is an algebraic structure generalizing the real and complex numbers. Formally, a field consists of a set  $\mathbb{F}$  equipped with two binary operations, an addition  $\oplus$  and a multiplication  $\odot$ , where  $(\mathbb{F}, \oplus)$  is an abelian group with an identity element 0 and  $(\mathbb{F} \setminus \{0\}, \odot)$  is an abelian group with an identity element 1. Furthermore, it satisfies distributivity, i.e.,  $a \odot (b \oplus c) = (a \odot b) \oplus (a \odot c)$  for all  $a, b, c \in \mathbb{F}$ . The most common examples are  $(\mathbb{R}, +, \cdot)$  and  $(\mathbb{C}, +, \cdot)$ , with addition and multiplication in the usual sense. Fields provide the set of scalars required to define a vector space.

See also: group, vector space, subspace, Euclidean space, inner product.

**firmly non-expansive operator** An operator  $\mathcal{F} : \mathcal{H} \rightarrow \mathcal{H}$  defined on a Hilbert space  $\mathcal{H}$  is called firmly non-expansive if it satisfies

$$\|\mathcal{F}\mathbf{w} - \mathcal{F}\mathbf{w}'\|_{\mathcal{H}}^2 \leq \langle \mathbf{w} - \mathbf{w}', \mathcal{F}\mathbf{w} - \mathcal{F}\mathbf{w}' \rangle \quad \text{for any } \mathbf{w}, \mathbf{w}' \in \mathcal{H}.$$

By the Cauchy-Schwarz inequality, any firmly non-expansive operator is necessarily also a non-expansive operator. Every fixed-point iteration that uses a firmly non-expansive operator converges to a fixed point of the operator [23].

See also: fixed-point iteration, contractive operator.

**fixed point** Consider some operator  $\mathcal{F} : \mathcal{H} \rightarrow \mathcal{H}$  defined on a Hilbert space  $\mathcal{H}$ . A vector  $\hat{\mathbf{w}} \in \mathcal{H}$  is called a fixed point of the operator  $\mathcal{F}$  if it satisfies

$$\mathcal{F}\hat{\mathbf{w}} = \hat{\mathbf{w}}.$$

In other words, applying the operator  $\mathcal{F}$  to its fixed point  $\hat{\mathbf{w}}$  returns the same vector  $\hat{\mathbf{w}}$ . Finding a fixed point of a suitable operator  $\mathcal{F}$  is a

common approach to solving various optimization problems (e.g., an instance of empirical risk minimization (ERM)). A popular method for computing approximations of a fixed point is the fixed-point iteration. See also: fixed-point iteration.

**fixed-point characterization** The solution of optimization problems arising in ML can often be characterized by a fixed-point equation. As a case in point, consider an optimization problem with a smooth and convex objective function  $f(\mathbf{w})$ . Each minimizer  $\hat{\mathbf{w}}$  of  $f(\mathbf{w})$  satisfies the fixed-point equation  $\hat{\mathbf{w}} = \mathcal{F}^{(\eta)}\hat{\mathbf{w}}$  with  $\mathcal{F}^{(\eta)} : \mathbf{w} \mapsto \mathbf{w} - \eta \nabla f(\hat{\mathbf{w}})$  [25]. Here,  $\eta > 0$  is an arbitrary positive constant and  $\nabla f(\mathbf{w})$  denotes the gradient of  $f(\mathbf{w})$ . Once we have found a fixed-point equation with an operator  $\mathcal{F}$  that is a contractive operator, we can use a fixed-point iteration to compute a solution of the underlying optimization problem. In the above example, the resulting fixed-point iteration coincides with gradient descent (GD) for minimizing  $f(\mathbf{w})$  [25].

See also: fixed-point equation, fixed-point iteration.

**fixed-point equation** A fixed-point equation is an equation of the following form:

$$\mathbf{w} = \mathcal{F}(\mathbf{w}),$$

where  $\mathcal{F} : \mathcal{H} \rightarrow \mathcal{H}$  is an operator defined on a Hilbert space  $\mathcal{H}$ . Solving a fixed-point equation amounts to finding the fixed points of  $\mathcal{F}$ . Many optimization problems, including instances of ERM, can be cast in this form. For example, minimizing a smooth convex function  $f$  is equivalent

to solving the following fixed-point equation:

$$\mathbf{w} = \mathcal{F}\mathbf{w} \quad \text{with } \mathcal{F} : \mathbf{w} \mapsto \mathbf{w} - \eta \nabla f(\mathbf{w}).$$

Here,  $\eta > 0$  can be chosen freely. The above fixed-point equation is nothing but the zero-gradient condition for the minimizer of  $f$  [25]. Similarly, one can reformulate the optimality conditions (i.e., the Karush–Kuhn–Tucker conditions (KKT conditions)) of optimization problems with constraints as a fixed-point equation [25], [47].

See also: fixed point, optimization problem.

**fixed-point iteration** A fixed-point iteration is an iterative method for solving a fixed-point equation, which, in an ML context, often arises from an optimization problem. In case of operators  $\mathcal{F} : \mathbb{R}^d \rightarrow \mathbb{R}^d$ , it constructs a sequence  $\mathbf{w}^{(0)}, \mathbf{w}^{(1)}, \dots$  in  $\mathbb{R}^d$  by repeatedly applying an operator  $\mathcal{F}$ , i.e.,

$$\mathbf{w}^{(t+1)} = \mathcal{F}\mathbf{w}^{(t)} \quad \text{for } t = 0, 1, \dots \quad (4)$$

The operator  $\mathcal{F}$  is chosen such that any of its fixed points is a solution  $\hat{\mathbf{w}}$  to the given optimization problem. For example, given a differentiable and convex function  $f(\mathbf{w})$ , the fixed points of the operator  $\mathcal{F} : \mathbf{w} \mapsto \mathbf{w} - \nabla f(\mathbf{w})$  coincide with the minimizers of  $f(\mathbf{w})$ . In general, for a given optimization problem with solution  $\hat{\mathbf{w}}$ , there are many different operators  $\mathcal{F}$  whose fixed points are  $\hat{\mathbf{w}}$ . Clearly, we should use an operator  $\mathcal{F}$  in (4) that reduces the distance (with respect to the Euclidean norm or another norm) to a solution such that

$$\underbrace{\|\mathbf{w}^{(t+1)} - \hat{\mathbf{w}}\|_2}_{\stackrel{(4)}{=} \|\mathcal{F}\mathbf{w}^{(t)} - \mathcal{F}\hat{\mathbf{w}}\|_2} \leq \|\mathbf{w}^{(t)} - \hat{\mathbf{w}}\|_2.$$

Thus, we require  $\mathcal{F}$  to be at least a non-expansive operator, i.e., the iteration (4) should not result in worse model parameters that have a larger distance to a solution  $\hat{\mathbf{w}}$ . Furthermore, each iteration (4) should also make some progress, i.e., reduce the distance to a solution  $\hat{\mathbf{w}}$ . This requirement can be made precise using the notion of a contractive operator [48], [49]. The operator  $\mathcal{F}$  is a contractive operator (with respect to the Euclidean norm) if, for some  $\kappa \in [0, 1]$ ,

$$\|\mathcal{F}\mathbf{w} - \mathcal{F}\mathbf{w}'\|_2 \leq \kappa \|\mathbf{w} - \mathbf{w}'\|_2 \quad \text{holds for any } \mathbf{w}, \mathbf{w}'.$$

For a contractive operator  $\mathcal{F}$ , the fixed-point iteration (4) generates a sequence  $\mathbf{w}^{(t)}$  that converges quite rapidly. In particular [2, Th. 9.23],

$$\|\mathbf{w}^{(t)} - \hat{\mathbf{w}}\|_2 \leq \kappa^t \|\mathbf{w}^{(0)} - \hat{\mathbf{w}}\|_2.$$

Here,  $\|\mathbf{w}^{(0)} - \hat{\mathbf{w}}\|_2$  is the distance between the initialization  $\mathbf{w}^{(0)}$  and the solution  $\hat{\mathbf{w}}$ . It turns out that a fixed-point iteration (4) with a firmly non-expansive operator  $\mathcal{F}$  is guaranteed to converge to a fixed point of  $\mathcal{F}$  [48, Corollary 5.16]. Fig. 27 depicts examples of a non-expansive operator, a firmly non-expansive operator, and a contractive operator. All of these operators are defined on the one-dimensional (1-D) space  $\mathbb{R}$ . Another example of a firmly non-expansive operator is the proximal operator of a convex function [24], [48].

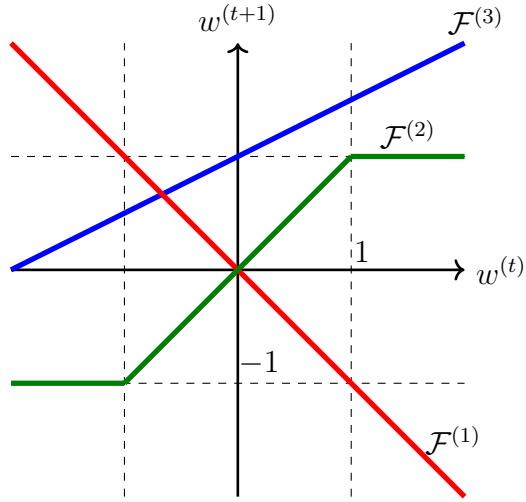


Fig. 27. Examples of a non-expansive operator  $\mathcal{F}^{(1)}$ , a firmly non-expansive operator  $\mathcal{F}^{(2)}$ , and a contractive operator  $\mathcal{F}^{(3)}$ .

See also: contractive operator, proximal operator, Bellman operator, Banach's fixed-point theorem, policy evaluation, value iteration.

**full-rank** A matrix  $\mathbf{A} \in \mathbb{R}^{m \times d}$  is full-rank if it has maximum rank [36]. For a tall matrix, i.e., when  $d < m$ , being full-rank means that its rank is equal to  $d$ .

$$\begin{array}{ll}
\mathbf{A} = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} & \mathbf{B} = \begin{pmatrix} 1 & 2 \\ 2 & 4 \end{pmatrix} \\
\text{full-rank square} & \text{rank-deficient square} \\
\\
\mathbf{C} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{pmatrix} & \mathbf{D} = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 4 & 6 \end{pmatrix} \\
\text{full-rank tall matrix} & \text{rank-deficient wide matrix}
\end{array}$$

Fig. 28. Examples of full-rank and rank-deficient matrices.

A square matrix is full-rank if and only if it is invertible.

See also: matrix, rank, dimension, linear map, column space.

**function** A function between two sets  $\mathcal{U}$  and  $\mathcal{V}$  assigns each element  $u \in \mathcal{U}$  exactly one element  $f(u) \in \mathcal{V}$  [2]. We write this as

$$f : \mathcal{U} \rightarrow \mathcal{V} : u \mapsto f(u),$$

where  $\mathcal{U}$  is the domain and  $\mathcal{V}$  the co-domain of  $f$ . That is, a function  $f$  defines a unique output  $f(u) \in \mathcal{V}$  for every input  $u \in \mathcal{U}$  (see Fig. 29).

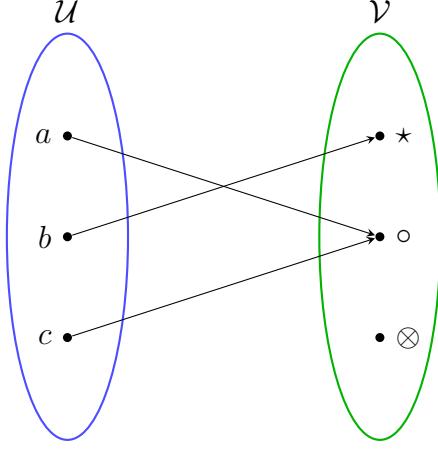


Fig. 29. A function  $f: \mathcal{U} \rightarrow \mathcal{V}$  mapping each element of the domain  $\mathcal{U} = \{a, b, c\}$  to exactly one element of the co-domain  $\mathcal{V} = \{\star, \circ, \otimes\}$ .

See also: domain, co-domain, output.

**Gaussian** See Gaussian RV.

**Gaussian process (GP)** A GP is a collection of RVs  $\{f(\mathbf{x})\}_{\mathbf{x} \in \mathcal{X}}$  indexed by input values  $\mathbf{x}$  from some input space  $\mathcal{X}$  such that, for any finite subset  $\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)} \in \mathcal{X}$ , the corresponding RVs  $f(\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)})$  have a joint multivariate normal distribution

$$f(\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}) \sim \mathcal{N}(\boldsymbol{\mu}, \mathbf{K}).$$

For a fixed input space  $\mathcal{X}$ , a GP is fully specified (or parameterized) by: 1) a mean function  $\mu(\mathbf{x}) = \mathbb{E}\{f(\mathbf{x})\}$ ; and 2) a covariance function  $k(\mathbf{x}, \mathbf{x}') = \mathbb{E}\{(f(\mathbf{x}) - \mu(\mathbf{x}))(f(\mathbf{x}') - \mu(\mathbf{x}'))\}$ .

Example: We can interpret the temperature distribution across Finland (at a specific point in time) as the realization of a GP  $f(\mathbf{x})$ , where each

input  $\mathbf{x} = (\text{lat}, \text{lon})$  denotes a geographic location. Temperature observations from Finnish Meteorological Institute (FMI) weather stations provide values  $f(\mathbf{x})$  at specific locations (see Fig. 30). A GP allows us to predict the temperature nearby FMI weather stations and to quantify the uncertainty of these predictions.

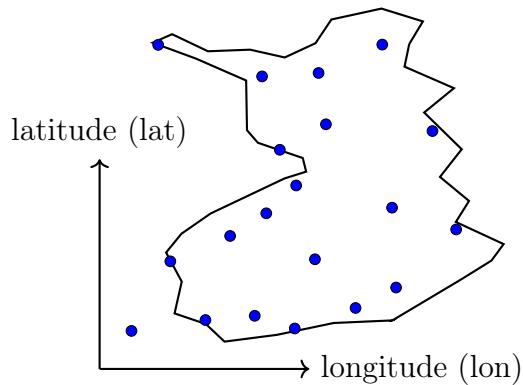


Fig. 30. For a given point in time, we can interpret the current temperature distribution over Finland as a realization of a GP indexed by geographic coordinates and sampled at FMI weather stations. The weather stations are indicated by blue dots.

See also: multivariate normal distribution, uncertainty, Gaussian RV.

**Gaussian random variable (Gaussian RV)** A standard Gaussian RV is a real-valued RV  $x$  with a pdf [7], [18], [29]

$$p^{(x)}(\eta) = \frac{1}{\sqrt{2\pi}} \exp(-\eta^2/2).$$

Given a standard Gaussian RV  $x$ , we can construct a general Gaussian RV  $x'$  with mean  $\mu$  and variance  $\sigma^2$  via  $x' := \sigma x + \mu$ . The probability

distribution of a Gaussian RV is referred to as normal distribution, denoted by  $\mathcal{N}(\mu, \sigma^2)$ .

A Gaussian random vector  $\mathbf{x} \in \mathbb{R}^d$  with covariance matrix  $\mathbf{C}$  and mean  $\boldsymbol{\mu}$  can be constructed as [18], [29], [50]

$$\mathbf{x} := \mathbf{A}\mathbf{z} + \boldsymbol{\mu},$$

where  $\mathbf{z} := (z_1, \dots, z_d)^T$  is a vector of i.i.d. standard Gaussian RVs, and  $\mathbf{A} \in \mathbb{R}^{d \times d}$  is any matrix satisfying  $\mathbf{A}\mathbf{A}^T = \mathbf{C}$ . The probability distribution of a Gaussian random vector is referred to as the multivariate normal distribution, denoted by  $\mathcal{N}(\boldsymbol{\mu}, \mathbf{C})$ .

We can interpret a Gaussian random vector  $\mathbf{x} = (x_1, \dots, x_d)^T$  as a stochastic process indexed by the set  $\mathcal{I} = \{1, \dots, d\}$ . A Gaussian process (GP) is a stochastic process over an arbitrary index set  $\mathcal{I}$  such that any restriction to a finite subset  $\mathcal{I}' \subseteq \mathcal{I}$  yields a Gaussian random vector [27].

Gaussian RVs are widely used probabilistic models in the statistical analysis of ML methods. Their significance arises partly from the central limit theorem (CLT), which provides conditions under which the average of many independent RVs (not necessarily Gaussian themselves) tends toward a Gaussian RV [19].

The multivariate normal distribution is also distinct in that it represents maximum uncertainty. Among all vector-valued RVs with a given covariance matrix  $\mathbf{C}$ , the RV  $\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}, \mathbf{C})$  maximizes differential entropy [35, Th. 8.6.5]. This makes GPs a natural choice for capturing uncertainty or the lack of (domain) knowledge.

See also: multivariate normal distribution, GP, probabilistic model,

CLT, differential entropy.

**geometric median (GM)** The GM of a set of input vectors  $\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}$  in  $\mathbb{R}^d$  is a point  $\mathbf{z} \in \mathbb{R}^d$  that minimizes the sum of distances to the vectors [25] such that

$$\mathbf{z} \in \arg \min_{\mathbf{y} \in \mathbb{R}^d} \sum_{r=1}^m \|\mathbf{y} - \mathbf{x}^{(r)}\|_2. \quad (5)$$

Fig. 31 illustrates a fundamental property of the GM: If  $\mathbf{z}$  does not coincide with any of the input vectors, then the unit vectors pointing from  $\mathbf{z}$  to each  $\mathbf{x}^{(r)}$  must sum to zero—this is the zero-subgradient (optimality) condition for (5). It turns out that the solution to (5) cannot be arbitrarily pulled away from trustworthy input vectors as long as they are the majority [51, Th. 2.2].

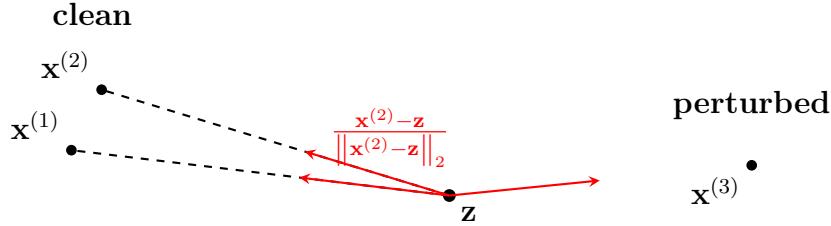


Fig. 31. Consider a solution  $\mathbf{z}$  of (5) that does not coincide with any of the input vectors. The optimality condition for (5) requires that the unit vectors from  $\mathbf{z}$  to the input vectors sum to zero.

See also: vector, subgradient.

**gradient** For a real-valued function  $f : \mathbb{R}^d \rightarrow \mathbb{R} : \mathbf{w} \mapsto f(\mathbf{w})$ , if a vector  $\mathbf{g}$  exists such that  $\lim_{\mathbf{w} \rightarrow \mathbf{w}'} f(\mathbf{w}) - (f(\mathbf{w}') + \mathbf{g}^T(\mathbf{w} - \mathbf{w}')) / \|\mathbf{w} - \mathbf{w}'\| = 0$ ,

it is referred to as the gradient of  $f$  at  $\mathbf{w}'$ . If it exists, the gradient is unique and denoted by  $\nabla f(\mathbf{w}')$  or  $\nabla f(\mathbf{w})|_{\mathbf{w}'}$  [2]. For a differentiable function, the gradient is the vector whose entries are the partial derivatives of  $f$  with respect to each input component, such that  $\nabla f(\mathbf{w}) = [\partial f / \partial x_1, \dots, \partial f / \partial x_d]^T$ . Geometrically, the gradient points in the direction of steepest ascent.

See also: function, vector, differentiable, partial derivative, gradient-based method, GD, zero-gradient condition, Hessian.

**gradient step** Given a differentiable real-valued function  $f(\cdot) : \mathbb{R}^d \rightarrow \mathbb{R}$  and a vector  $\mathbf{w} \in \mathbb{R}^d$ , the gradient step updates  $\mathbf{w}$  by adding the scaled negative gradient  $\nabla f(\mathbf{w})$  to obtain the new vector (see Fig. 32) as follows:

$$\hat{\mathbf{w}} := \mathbf{w} - \eta \nabla f(\mathbf{w}). \quad (6)$$

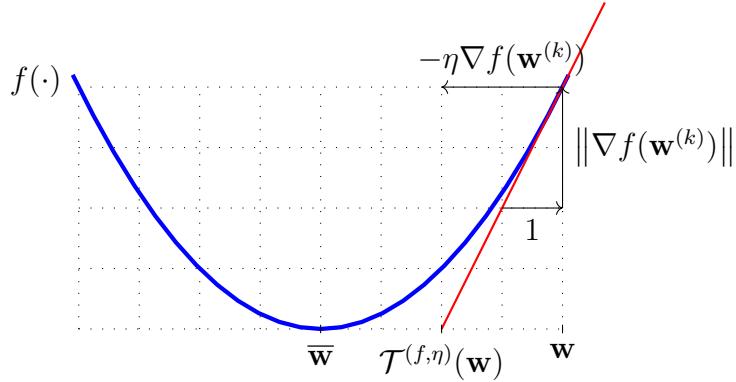


Fig. 32. The basic gradient step (6) maps a given vector  $\mathbf{w}$  to the updated vector  $\mathbf{w}'$ .

More formally, the gradient step amounts to applying the operator

$$\mathcal{T}^{(f,\eta)}(\cdot) : \mathbb{R}^d \rightarrow \mathbb{R}^d : \mathbf{w} \mapsto \mathbf{w} - \eta \nabla f(\mathbf{w}).$$

For a convex function  $f(\cdot)$  that is sufficiently smooth and a sufficiently small learning rate  $\eta > 0$ , one can verify that  $\mathcal{T}^{(f,\eta)}$  becomes a firmly non-expansive operator [23, Corollary 18.16]. Moreover, for a strongly convex and smooth function  $f(\cdot)$  and an appropriate choice of learning rate, one can verify that  $\mathcal{T}^{(f,\eta)}$  is a contractive operator [52, Th. 2.1.14]. In these cases, gradient-based methods are instances of a fixed-point iteration, since the operator  $\mathcal{T}^{(f,\eta)}$  has a fixed point that coincides with the minimizer of  $f(\cdot)$  (see zero-gradient condition). Note that the gradient step (6) optimizes locally—in a neighborhood whose size is determined by the step size  $\eta$ —a linear approximation to the function  $f(\cdot)$ . A natural generalization of (6) is to locally optimize the function itself—instead of its linear approximation—such that

$$\hat{\mathbf{w}} = \arg \min_{\mathbf{w}' \in \mathbb{R}^d} f(\mathbf{w}') + \frac{1}{\eta} \|\mathbf{w} - \mathbf{w}'\|_2^2. \quad (7)$$

We intentionally use the same symbol  $\eta$  for the parameter in (7) as we used for the step size in (6). The larger the  $\eta$  we choose in (7), the more progress the update will make toward reducing the function value  $f(\hat{\mathbf{w}})$ . Note that, much like the gradient step (6), the update (7) also defines an operator that depends on the function  $f(\cdot)$  and on the learning rate  $\eta$ . For a convex function  $f(\cdot)$ , this operator is known as the proximal operator of  $f(\cdot)$  [24].

See also: differentiable, gradient, step size, proximal operator, subgradient descent, fixed-point equation.

**graph** A graph  $\mathcal{G} = (\mathcal{V}, \mathcal{E})$  consists of a node set  $\mathcal{V}$  and an edge set  $\mathcal{E}$ . Each edge  $e \in \mathcal{E}$  is characterized by the nodes to which it is connected and in what precise sense. For example, an edge of a directed graph is leaving one node and pointing to another node. An edge of an undirected graph connects two nodes without any sense of direction [37], [53]. In principle, there can also be several (parallel) edges that are connected to the same nodes in the same way [53]. Moreover, edges may connect a node to itself, resulting in so-called self-loops [37]. A simple undirected graph contains no parallel edges and no self-loops [54]. Each edge  $e \in \mathcal{E}$  of a simple undirected graph can be identified with a set of two nodes  $i, i'$ .

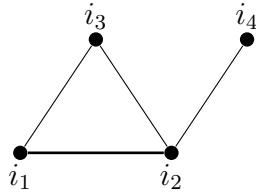


Fig. 33. A simple undirected graph with four nodes  $\mathcal{V} = \{i_1, i_2, i_3, i_4\}$  and four edges  $\mathcal{E} = \{\{i_1, i_2\}, \{i_2, i_3\}, \{i_3, i_1\}, \{i_2, i_4\}\}$ .

Weighted graphs assign a numerical value  $A_e$ , referred to as edge weight, to each edge  $e \in \mathcal{E}$ .

See also: map, edge weight.

**graph of a function** Given a function  $f : \mathcal{X} \rightarrow \mathcal{Y}$  with domain  $\mathcal{X}$  and co-domain  $\mathcal{Y}$ , the graph of  $f$  is the subset of  $\mathcal{X} \times \mathcal{Y}$  defined as

$$\{(\mathbf{x}, f(\mathbf{x})) : \mathbf{x} \in \mathcal{X}\}.$$

The graph of a function provides a geometric representation that is widely used in analysis, topology, and optimization [55], [56].

See also: function, epigraph.

**group** A group  $(G, \diamond)$  is an algebraic structure [57, Ch. 1] consisting of a set  $G$  and a binary operation  $\diamond$  that assigns to any two elements  $a, b \in G$  another element  $a \diamond b \in G$ . For  $G$  to be a group, it must satisfy associativity (i.e.,  $(a \diamond b) \diamond c = a \diamond (b \diamond c)$  for all  $a, b, c \in G$ ), the existence of an identity element  $e \in G$  with  $a \diamond e = e \diamond a = a$  for all  $a \in G$ , and the existence of an inverse  $a^{-1} \in G$  for every element  $a \in G$  that satisfies  $a^{-1}a = aa^{-1} = e$ . A group is called abelian or commutative if  $a \diamond b = b \diamond a$  for all  $a, b \in G$ . Basic examples are the additive group  $(\mathbb{R}, +)$  and the multiplicative group  $(\mathbb{R} \setminus \{0\}, \cdot)$ ; similarly,  $(\mathbb{C}, +)$  and  $(\mathbb{C} \setminus \{0\}, \cdot)$  are defined using addition and multiplication on the real (complex) numbers in the usual sense. More complicated group structures exist in matrices [57], such as rotation matrices useful in ML for data augmentation.

See also: vector space, subspace, inverse matrix, field.

**halfspace** See hyperplane.

**Hermitian** A square matrix  $\mathbf{A} \in \mathbb{C}^{d \times d}$  is Hermitian if it coincides with its conjugate transpose, i.e.,  $\mathbf{A} = \mathbf{A}^H$ . Trivially, a Hermitian matrix is also a normal matrix.

See also: normal matrix.

**Hessian** Consider a function  $f : \mathbb{R}^d \rightarrow \mathbb{R}$  for which the second-order partial

derivatives exist at  $\mathbf{x}'$ . Then, the Hessian  $\nabla^2 f(\mathbf{x}')$  of  $f$  at  $\mathbf{x}$  is defined as the matrix of second-order partial derivatives of  $f$  at  $\mathbf{x}'$ , i.e.,

$$\nabla^2 f(\mathbf{x}') = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_d} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_d} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_d \partial x_1} & \frac{\partial^2 f}{\partial x_d \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_d^2} \end{pmatrix}.$$

If the second-order partial derivatives are continuous in a neighborhood around  $\mathbf{x}'$ , then the Hessian is a symmetric matrix, i.e.,  $\partial^2 f / \partial x_j \partial x_{j'} = \partial^2 f / \partial x_{j'} \partial x_j$  for all  $j, j'$  [2]. If additionally  $f$  is convex, then the Hessian is a psd matrix [25].

The Hessian  $\nabla^2 f(\mathbf{x}')$  can be used to compute a quadratic function

$$q(\mathbf{x}) = \frac{1}{2}(\mathbf{x} - \mathbf{x}')^T \underbrace{\nabla^2 f(\mathbf{x}')}_{\text{Hessian}} (\mathbf{x} - \mathbf{x}') + (\mathbf{x} - \mathbf{x}')^T \underbrace{\nabla f(\mathbf{x}')}_{\text{gradient}} + f(\mathbf{x}')$$

that approximates  $f$  locally around  $\mathbf{x}'$  (see also Fig. 34).

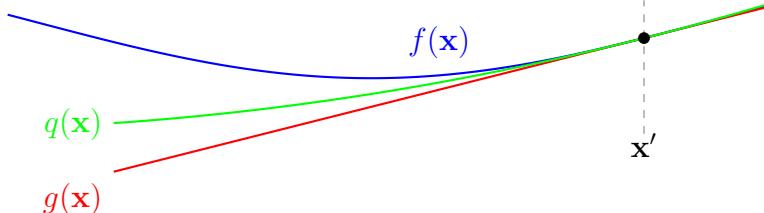


Fig. 34. A function  $f(\mathbf{x})$  that is sufficiently smooth at a point  $\mathbf{x}'$  can be locally approximated by a quadratic function  $q(\mathbf{x})$ , which provides a more accurate approximation compared to a linear function  $g(\mathbf{x})$ .

See also: function, matrix, quadratic function, differentiable.

**Hilbert space** A Hilbert space  $\mathcal{H}$  is a complete inner product space. Thus,  $\mathcal{H}$  is a vector space equipped with an inner product  $\langle \cdot, \cdot \rangle$ . The inner product induces a norm  $\|\cdot\|_{\mathcal{H}}$  via

$$\|\mathbf{w}\|_{\mathcal{H}} = \sqrt{\langle \mathbf{w}, \mathbf{w} \rangle}.$$

Furthermore,  $\mathcal{H}$  is complete, in the sense that every Cauchy sequence  $(\mathbf{w}^{(r)})_{r \in \mathbb{N}}$  in  $\mathcal{H}$  converges to a limit  $\lim_{r \rightarrow \infty} \mathbf{w}^{(r)}$  that is also contained in  $\mathcal{H}$ .

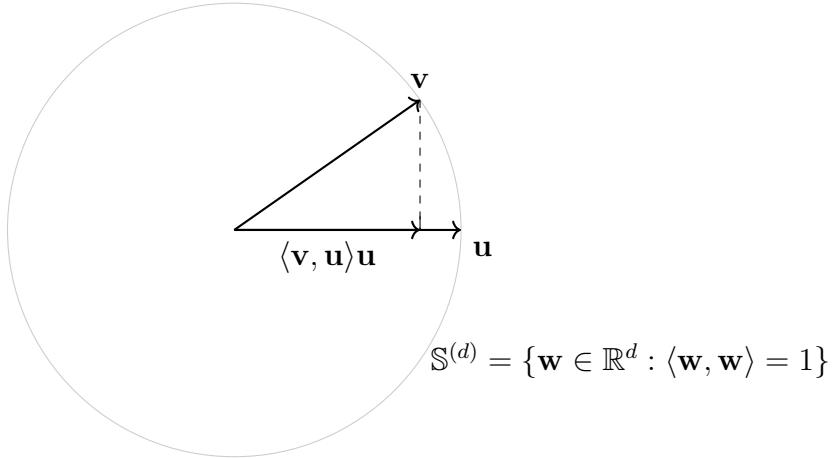


Fig. 35. For two unit-norm vectors  $\mathbf{u}, \mathbf{v} \in \mathbb{S}^{(d)} \subseteq \mathbb{R}^d$ , the inner product  $\langle \mathbf{u}, \mathbf{v} \rangle$  is the expansion coefficient for the projection of  $\mathbf{v}$  onto the subspace  $\{c\mathbf{u} : c \in \mathbb{R}\}$  spanned by  $\mathbf{u}$ . The absolute value  $|\langle \mathbf{u}, \mathbf{v} \rangle|$  measures the norm of this projection.

One important example of a Hilbert space is the Euclidean space  $\mathbb{R}^d$  with the inner product  $\langle \mathbf{w}, \mathbf{w}' \rangle = \mathbf{w}^\top \mathbf{w}'$ .

See also: inner product, vector space.

**Hoeffding's inequality** Hoeffding's inequality [58] is a fundamental concentration inequality that provides an upper bound on the probability that a sum (or average) of independent, bounded RVs deviates from its mean by more than some threshold. Let  $x_1, \dots, x_n$  be independent real-valued RVs  $x_i$  taking values in  $[a_i, b_i] \subset \mathbb{R}$ , and  $S_n := \sum_{i=1}^n x_i$  and  $\mathbb{E}\{S_n\} = \sum_{i=1}^n \mathbb{E}\{x_i\}$ . Then, Hoeffding's inequality [15, Th. 2.2.6] states that

$$\mathbb{P}(|S_n - \mathbb{E}\{S_n\}| \geq t) \leq 2 \exp \left( -\frac{2t^2}{\sum_{i=1}^n (b_i - a_i)^2} \right) \quad \forall t > 0.$$

Hoeffding's inequality typically provides sharper bounds than Chebyshev's inequality, but it is more restrictive by assuming bounded RVs. In the context of ML, Hoeffding's inequality can be used for the analysis of ERM or MAB methods.

See also: concentration inequality, probability, RV, Chebyshev's inequality, expectation, probability space, Markov's inequality.

**hyperplane** A hyperplane is a  $(d - 1)$ -dimensional affine subspace of a  $d$ -dimensional vector space. In the context of a Euclidean space  $\mathbb{R}^d$ , a hyperplane is a set of the following form:

$$\{\mathbf{x} \in \mathbb{R}^d : \mathbf{w}^\top \mathbf{x} = b\},$$

where  $\mathbf{w} \in \mathbb{R}^d \setminus \{0\}$  is a normal vector and  $b \in \mathbb{R}$  is an offset. Such a hyperplane divides  $\mathbb{R}^d$  into two halfspaces:

$$\{\mathbf{x} \in \mathbb{R}^d : \mathbf{w}^\top \mathbf{x} > b\} \quad \text{and} \quad \{\mathbf{x} \in \mathbb{R}^d : \mathbf{w}^\top \mathbf{x} < b\}.$$

Hyperplanes arise as the decision boundaries of linear classifiers.

See also: subspace, vector space, Euclidean space, decision boundary, inner product.

**independent and identically distributed (i.i.d.)** A collection of RVs  $\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(m)}$  defined on the same probability space  $(\Omega, \Sigma, \mathbb{P}(\cdot))$  is referred to as i.i.d. if each  $\mathbf{z}^{(r)}$  follows the same probability distribution, and the RVs are mutually independent. That is, for any collection of events  $\mathcal{A}_1, \dots, \mathcal{A}_m \in \Sigma$ , we have

$$\mathbb{P}(\mathbf{z}^{(1)} \in \mathcal{A}_1, \dots, \mathbf{z}^{(m)} \in \mathcal{A}_m) = \prod_{r=1}^m \mathbb{P}(\mathbf{z}^{(r)} \in \mathcal{A}_r).$$

Probability theory also guarantees the existence and construction of independent copies of RVs [5, Th. 2.19].

See also: RV, probability distribution, event, data point, i.i.d. assumption.

**indicator function** Consider some process that can result in different possible outcomes (e.g., survival or death of a patient). Such a process can be modeled as a random experiment with sample space  $\Omega$  containing all possible outcomes. The indicator function  $\mathbb{I}_{\mathcal{A}}$  of an event  $\mathcal{A} \subseteq \Omega$  is a function defined as [7]

$$\mathbb{I}_{\mathcal{A}}(\omega) = \begin{cases} 1 & \text{if } \omega \in \mathcal{A}, \\ 0 & \text{if } \omega \notin \mathcal{A}. \end{cases}$$

The notion of an indicator function is not limited to outcomes of a random experiment. Ultimately, it is just a principled way to represent

a set  $\mathcal{A}$  by a function  $\mathbb{I}_{\mathcal{A}}$  [25]. For a probability space  $(\Omega, \Sigma, \mathbb{P}(\cdot))$  and  $\mathcal{A} \in \Sigma$ , the indicator function  $\mathbb{I}_{\mathcal{A}}$  is an RV and it holds that  $\mathbb{E}\{\mathbb{I}_{\mathcal{A}}\} = \mathbb{P}(\mathcal{A})$ .

See also: outcome, random experiment, function.

**infimum (or greatest lower bound)** The infimum of a set of real numbers is the largest number that is less than or equal to every element in the set. More formally, a real number  $a$  is the infimum of a set  $\mathcal{A} \subseteq \mathbb{R}$  if: 1)  $a$  is a lower bound of  $\mathcal{A}$ ; and 2) no number larger than  $a$  is a lower bound of  $\mathcal{A}$ . Every non-empty set of real numbers that is bounded below has an infimum, even if it does not contain its infimum as an element [2, Sec. 1.4].

**injective** A function  $f : \mathcal{U} \rightarrow \mathcal{V}$  is injective if it maps distinct elements of its domain to distinct elements of its co-domain, i.e., if  $f(u_1) = f(u_2)$  implies  $u_1 = u_2$  for all  $u_1, u_2 \in \mathcal{U}$  [26]. Equivalently, no two different function inputs are mapped to the same function output.

See also: function.

**inner product** Consider a vector space  $\mathcal{X}$  over the field  $\mathbb{F}$ , where  $\mathbb{F}$  is either the field of real numbers  $\mathbb{R}$  or the field of complex numbers  $\mathbb{C}$ . An inner product in  $\mathcal{X}$  is a function

$$\langle \cdot, \cdot \rangle : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{F}$$

that satisfies the following properties for all  $\mathbf{x}, \mathbf{x}', \mathbf{x}'' \in \mathcal{X}$  and all scalars  $\alpha \in \mathbb{F}$  [34]:

- Conjugate symmetry:  $\langle \mathbf{x}, \mathbf{x}' \rangle = \overline{\langle \mathbf{x}', \mathbf{x} \rangle}$ ;

- Linearity in the first argument:  $\langle \alpha\mathbf{x} + \mathbf{x}'', \mathbf{x}' \rangle = \alpha\langle \mathbf{x}, \mathbf{x}' \rangle + \langle \mathbf{x}'', \mathbf{x}' \rangle$ ;
- Positive-definiteness:  $\langle \mathbf{x}, \mathbf{x} \rangle \geq 0$ , with equality if and only if  $\mathbf{x} = \mathbf{0}$ .

The pair  $(\mathcal{X}, \langle \cdot, \cdot \rangle)$  is called an inner product space. Each inner product induces a norm via  $\|\mathbf{x}\| := \sqrt{\langle \mathbf{x}, \mathbf{x} \rangle}$  for all  $\mathbf{x} \in \mathcal{X}$ , which in turn induces a metric via  $d(\mathbf{x}, \mathbf{x}') := \|\mathbf{x} - \mathbf{x}'\|$  for all  $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$ .

See also: field, norm, vector, Hilbert space, orthogonality condition, metric space.

**integrable** A measurable function  $f : \Omega \rightarrow \mathbb{R}$  defined on a measure space  $(\Omega, \Sigma, \mu)$  is called integrable if the Lebesgue integral of its absolute value is finite, i.e.,

$$\int_{\Omega} |f(x)| d\mu < \infty.$$

In this case, the Lebesgue integral  $\int_{\Omega} f(x) d\mu$  is well defined and finite. An RV  $x$  defined on the sample space of a probability space  $(\Omega, \Sigma, \mathbb{P})$  is integrable if

$$\mathbb{E}\{|x|\} = \int_{\Omega} |x(\omega)| d\mathbb{P} < \infty,$$

which is equivalent to the existence of the expectation  $\mathbb{E}\{x\}$  (i.e., it is finite).

See also: measure space, measure.

**inverse matrix** An inverse matrix  $\mathbf{A}^{-1}$  is defined for a square matrix  $\mathbf{A} \in \mathbb{R}^{n \times n}$  that is of full-rank, meaning its columns are linearly independent. In this case,  $\mathbf{A}$  is said to be invertible, and its inverse satisfies

$$\mathbf{A}\mathbf{A}^{-1} = \mathbf{A}^{-1}\mathbf{A} = \mathbf{I}.$$

A square matrix is invertible if and only if its determinant is nonzero. Inverse matrices are fundamental in solving systems of linear equations and in the closed-form solution of linear regression [32], [59]. The concept of an inverse matrix can be extended to matrices that are not square or do not have full-rank. One may define a “left inverse”  $\mathbf{B}$  satisfying  $\mathbf{BA} = \mathbf{I}$  or a “right inverse”  $\mathbf{C}$  satisfying  $\mathbf{AC} = \mathbf{I}$ . For general rectangular or singular matrices, the Moore–Penrose pseudoinverse  $\mathbf{A}^+$  provides a unified concept of a generalized inverse matrix [3].

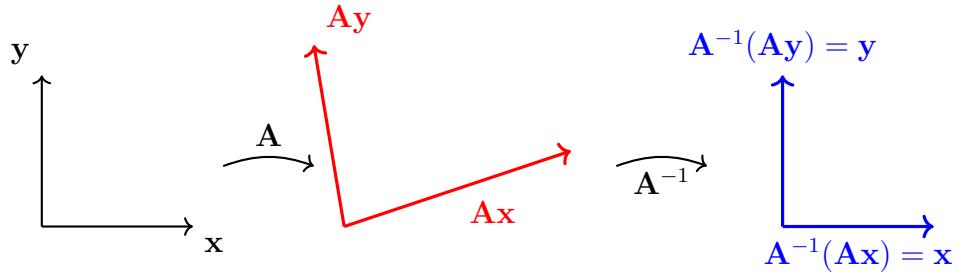


Fig. 36. A matrix  $\mathbf{A}$  represents a linear transformation of  $\mathbb{R}^2$ . The inverse matrix  $\mathbf{A}^{-1}$  represents the inverse transformation.

See also: matrix, determinant, linear regression, pseudoinverse.

**Jacobian matrix** The Jacobian matrix of a vector-valued function

$$\mathbf{f} : \mathbb{R}^d \rightarrow \mathbb{R}^{d'} : \mathbf{x} \mapsto \mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), \dots, f_{d'}(\mathbf{x}))^T$$

is the matrix constituted by all first-order partial derivatives  $\mathbf{f}$  [2], [60].

More explicitly,

$$D_f(\mathbf{x}) := \begin{bmatrix} \frac{\partial f_1}{\partial x_1}(\mathbf{x}) & \cdots & \frac{\partial f_1}{\partial x_d}(\mathbf{x}) \\ \vdots & \ddots & \vdots \\ \frac{\partial f_{d'}}{\partial x_1}(\mathbf{x}) & \cdots & \frac{\partial f_{d'}}{\partial x_d}(\mathbf{x}) \end{bmatrix}.$$

The Jacobian matrix represents the best linear approximation of  $\mathbf{f}$  around a given argument vector  $\mathbf{x}$ . Note that the  $i$ th row of the Jacobian matrix coincides with the transpose of the gradient of one component function  $f_i$  evaluated at  $\mathbf{x}$ .

See also: matrix, function, partial derivative, gradient.

**Johnson–Lindenstrauss lemma (JL lemma)** The JL lemma describes conditions for the existence of a feature map  $\Phi : \mathbb{R}^d \rightarrow \mathbb{R}^{d'}$  with  $d' \ll d$  such that the pairwise Euclidean distance between feature vectors of a finite dataset is approximately preserved [15], [61], [62]. Consider a dataset  $\mathcal{D} = \{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$  with data points characterized by feature vectors in  $\mathbb{R}^d$ . Then, for any  $d'$  that satisfies

$$d' \geq \frac{4 \ln(m)}{\varepsilon^2/2 - \varepsilon^3/3} \quad \text{for some } 0 < \varepsilon < 1,$$

there is a feature map  $\Phi$  such that [63]

$$(1-\varepsilon) \left\| \mathbf{x}^{(r)} - \mathbf{x}^{(r')} \right\|_2 \leq \left\| \Phi(\mathbf{x}^{(r)}) - \Phi(\mathbf{x}^{(r')}) \right\|_2 \leq (1+\varepsilon) \left\| \mathbf{x}^{(r)} - \mathbf{x}^{(r')} \right\|_2 \quad (8)$$

for all  $\mathbf{x}^{(r)}, \mathbf{x}^{(r')} \in \mathcal{D}$ .

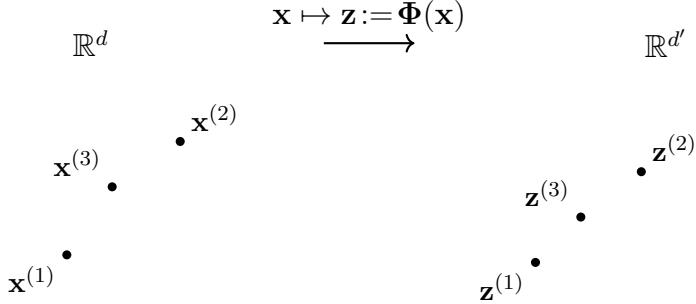


Fig. 37. The JL lemma offers precise conditions that guarantee the existence of a feature map  $\Phi : \mathbb{R}^d \rightarrow \mathbb{R}^{d'}$  such that pairwise Euclidean distances between (the feature vectors of) data points are approximately preserved. Roughly speaking,  $\Phi$  maps neighboring points in the original feature space to neighboring points in the new feature space.

The feature map  $\Phi$  can be obtained from a random matrix  $\mathbf{A} \in \mathbb{R}^{d' \times d}$  whose entries are i.i.d. Gaussian RVs  $A_{i,j} \sim \mathcal{N}(0, 1/d')$ . It can be shown that the feature map  $\mathbf{x} \mapsto \underbrace{\mathbf{Ax}}_{\Phi(\mathbf{x})}$  satisfies (8) with probability at least  $1 - 1/m$  [63].

See also: matrix, norm, vector space, Euclidean space, dimensionality reduction, principal component analysis (PCA).

**Karush–Kuhn–Tucker conditions (KKT conditions)** Consider a constrained optimization problem with Lagrangian

$$L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\nu}) = f(\mathbf{x}) + \sum_{i=1}^k \lambda_i g_i(\mathbf{x}) + \sum_{j=1}^l \nu_j h_j(\mathbf{x})$$

with differentiable functions  $f, g_1, \dots, g_k, h_1, \dots, h_l$ . The KKT conditions are a system of equations and inequalities for the primal and dual

variables  $(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\nu})$  [25, Sec. 5.5.3]:

$$\begin{aligned}
\nabla_{\mathbf{x}} L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\nu}) &= \mathbf{0} && \text{(stationarity)} \\
g_i(\mathbf{x}) &\leq 0, \quad i = 1, \dots, k && \text{(primal feasibility)} \\
h_j(\mathbf{x}) &= 0, \quad j = 1, \dots, l && \text{(primal feasibility)} \\
\lambda_i &\geq 0, \quad i = 1, \dots, k && \text{(dual feasibility)} \\
\lambda_i g_i(\mathbf{x}) &= 0, \quad i = 1, \dots, k && \text{(complementary slackness).}
\end{aligned}$$

If the primal problem results in a zero duality gap, any optimal choice  $(\mathbf{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\nu}^*)$  for the primal and dual variables must satisfy the KKT conditions. Conversely, if the primal problem is convex and some regularity conditions hold, any choice  $(\mathbf{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\nu}^*)$  that satisfies the KKT conditions is also optimal for the primal and dual problems [25]. The KKT conditions can be written as a fixed-point equation and can, in turn, be used to construct fixed-point iterations for solving the primal and dual problems [25].

See also: optimization problem, duality gap, convex, dual problem, fixed-point iteration.

**kernel (linear map)** See nullspace.

**Kullback–Leibler divergence (KL divergence)** The KL divergence is a quantitative measure of how different a probability distribution  $\mathbb{P}^{(1)}$  is from another probability distribution  $\mathbb{P}^{(2)}$ , which is defined on the same probability space [35]. The KL divergence  $D^{(\text{KL})}(\mathbb{P}^{(1)}, \mathbb{P}^{(2)})$  is well defined only if  $\mathbb{P}^{(1)}$  is absolutely continuous with respect to  $\mathbb{P}^{(2)}$ .

Special case: If  $\mathbb{P}^{(1)}$  and  $\mathbb{P}^{(2)}$  are described by the pmfs  $p^{(x)}(\cdot)$  and

$p^{(y)}(\cdot)$  on a sample space  $\Omega$ , then

$$D^{(\text{KL})}(p^{(x)}(\cdot), p^{(y)}(\cdot)) := \sum_{\omega \in \Omega} p^{(x)}(\omega) \log \frac{p^{(x)}(\omega)}{p^{(y)}(\omega)}.$$

In this discrete setting, the KL divergence coincides with the Bregman divergence generated by the negative entropy function [64, Sec. 4.3]

$$\phi(p) := \sum_{\omega \in \Omega} p^{(x)}(\omega) \log p^{(x)}(\omega).$$

See also: probability distribution, Bregman divergence.

**Lagrange dual function** Consider a constrained optimization problem of the following form:

$$\begin{aligned} & \min_{\mathbf{x} \in \mathbb{R}^d} f(\mathbf{x}) \\ \text{s.t. } & g_i(\mathbf{x}) \leq 0, \quad i = 1, \dots, k \\ & h_j(\mathbf{x}) = 0, \quad j = 1, \dots, l. \end{aligned}$$

A useful tool for the analysis of such an optimization problem is the Lagrange dual function:

$$q : \mathbb{R}_+^k \times \mathbb{R}^l : q(\boldsymbol{\lambda}, \boldsymbol{\nu}) = \inf_{\mathbf{x} \in \mathbb{R}^d} L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\nu}),$$

where  $L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\nu}) := f(\mathbf{x}) + \sum_{i=1}^k \lambda_i g_i(\mathbf{x}) + \sum_{j=1}^l \nu_j h_j(\mathbf{x})$  denotes the Lagrangian of the original optimization problem. Fig. 38 illustrates the function  $q(\lambda)$  obtained for an optimization problem with a single inequality constraint. For a given value  $\lambda \geq 0$ ,  $q(\lambda)$  is the vertical

intercept of a supporting hyperplane with normal vector  $(\lambda, 1)$  to the set  $\mathcal{A} = \{(u, t) : f(\mathbf{x}) \leq t, g_1(\mathbf{x}) \leq u \text{ for some } \mathbf{x} \in \mathbb{R}^d\}$  [25, Sec. 5.3].

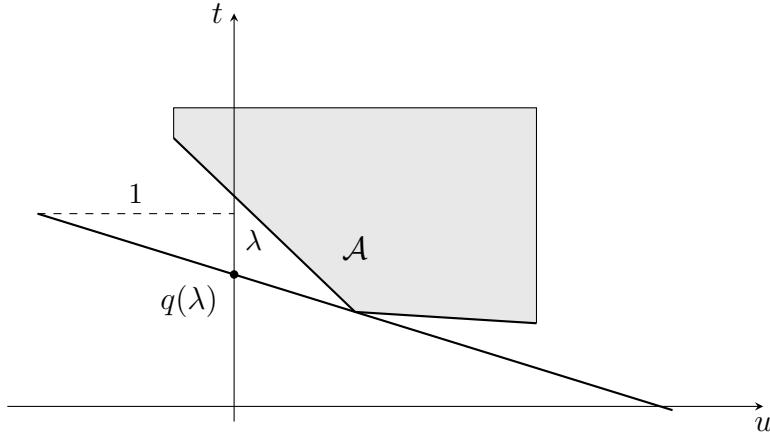


Fig. 38. The Lagrange dual function  $q(\lambda)$ , for a given  $\lambda \geq 0$ , is the vertical intercept of a supporting hyperplane with normal vector  $(\lambda, 1)$  to the set  $\mathcal{A} = \{(u, t) : f(\mathbf{x}) \leq t, g_1(\mathbf{x}) \leq u \text{ for some } \mathbf{x} \in \mathbb{R}^d\}$  [25, Sec. 5.3].

See also: optimization problem, function, Lagrangian, supporting hyperplane.

**Lagrangian** Consider a constrained optimization problem of the following form:

$$\begin{aligned} & \min_{\mathbf{w} \in \mathbb{R}^d} f(\mathbf{w}) \\ \text{s.t. } & g_i(\mathbf{w}) \leq 0, \quad i = 1, \dots, k \\ & h_j(\mathbf{w}) = 0, \quad j = 1, \dots, l. \end{aligned}$$

Here,  $f : \mathbb{R}^d \rightarrow \mathbb{R}$  is the objective function,  $g_i : \mathbb{R}^d \rightarrow \mathbb{R}$ ,  $i = 1, \dots, k$ , are the inequality constraint functions, and  $h_j : \mathbb{R}^d \rightarrow \mathbb{R}$ ,  $j = 1, \dots, l$ ,

are the equality constraint functions. The Lagrangian of the above optimization problem is defined as [25]

$$L(\mathbf{w}, \boldsymbol{\lambda}, \boldsymbol{\nu}) := f(\mathbf{w}) + \sum_{i=1}^k \lambda_i g_i(\mathbf{w}) + \sum_{j=1}^l \nu_j h_j(\mathbf{w}).$$

Here,  $\boldsymbol{\lambda} \in \mathbb{R}_+^k$  (i.e.,  $\lambda_i \geq 0$  for all  $i = 1, \dots, k$ ) and  $\boldsymbol{\nu} \in \mathbb{R}^l$  are the multipliers associated with the inequality and equality constraints, respectively. The Lagrangian is a useful tool for the analysis of constrained optimization problems and the design of optimization methods. For example, the Lagrangian allows us to define a dual problem, which provides a lower bound for the optimal value of a constrained optimization problem. This lower bound, in turn, can be used to construct a stopping criterion for iterative optimization methods [25]. Fig. 39 provides a geometric interpretation of the Lagrangian for a constrained optimization problem with a single inequality constraint ( $k = 1$ ) and no equality constraints ( $l = 0$ ) (see [25, Sec. 5.3]). Here, the Lagrangian  $L(\mathbf{w}, \lambda)$  is the vertical intercept of a straight line with slope  $-\lambda$  that passes through a point  $(g_1(\mathbf{w}), f(\mathbf{w}))$ .

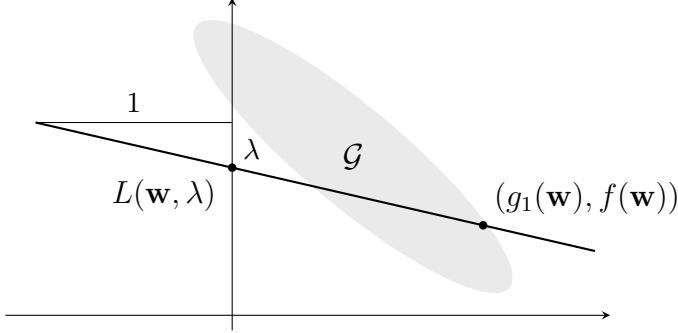


Fig. 39. An optimization problem with an objective function  $f(\mathbf{w})$  and a single inequality constraint  $g_1(\mathbf{w}) \leq 0$  can be represented by a set  $\mathcal{G} := \{(g_1(\mathbf{w}), f(\mathbf{w})) : \mathbf{w} \in \mathbb{R}^d\}$ . The value of the Lagrangian  $L(\mathbf{w}, \lambda)$  is the vertical intercept of a line with slope  $-\lambda$  that passes through the point  $(g_1(\mathbf{w}), f(\mathbf{w})) \in \mathcal{G}$  [25].

See also: optimization problem, convex optimization, dual problem.

**Laplacian matrix** The structure of a graph  $\mathcal{G}$ , with nodes  $i = 1, \dots, n$ , can be analyzed using the properties of special matrices that are associated with  $\mathcal{G}$ . One such matrix is the graph Laplacian matrix  $\mathbf{L}^{(\mathcal{G})} \in \mathbb{R}^{n \times n}$ , which is defined for an undirected and weighted graph [65], [66]. It is defined elementwise as (see Fig. 40)

$$L_{i,i'}^{(\mathcal{G})} := \begin{cases} -A_{i,i'}, & \text{for } i \neq i', \{i, i'\} \in \mathcal{E}, \\ \sum_{i'' \neq i} A_{i,i''}, & \text{for } i = i', \\ 0, & \text{else.} \end{cases}$$

Here,  $A_{i,i'}$  denotes the edge weight of an edge  $\{i, i'\} \in \mathcal{E}$ .

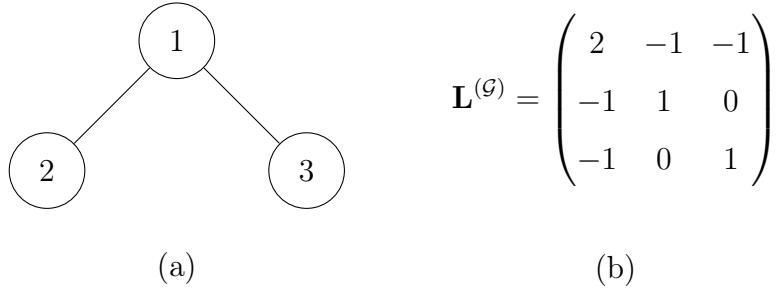


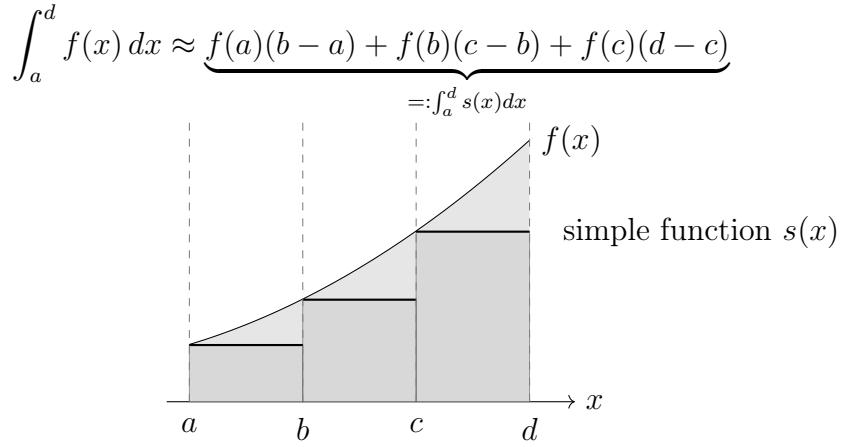
Fig. 40. (a) Some undirected graph  $\mathcal{G}$  with three nodes  $i = 1, 2, 3$ . (b) The Laplacian matrix  $\mathbf{L}^{(\mathcal{G})} \in \mathbb{R}^{3 \times 3}$  of  $\mathcal{G}$ .

See also: graph, matrix, edge weight.

**law of large numbers** The law of large numbers refers to the convergence of the average of an increasing (large) number of i.i.d. RVs to the mean of their common probability distribution. Different instances of the law of large numbers are obtained by using different notions of convergence [29].

See also: convergence, i.i.d., RV, mean, probability distribution.

**Lebesgue integral** The Lebesgue integral assigns each integrable function  $f : \mathbb{R}^d \rightarrow \mathbb{R}$  a number  $\int_{\mathbb{R}^d} f(\mathbf{x}) d\mathbf{x}$  that is referred to as the integral of  $f$ . The integral of  $f$  can be interpreted as the volume that is enclosed by the function  $f$  in the space  $\mathbb{R}^{d+1}$ . We can compute it by increasingly accurate approximations using simple functions [1, Ch. 1].



It is useful to think of the Lebesgue integral as a function that maps an integrable function  $f$  to the value of its integral:

$$f \mapsto \int_{\mathbb{R}^d} f(\mathbf{x}) d\mathbf{x}.$$

The precise definition of this function, whose domain consists of the integrable functions, is a cornerstone of measure theory [1, Ch. 1].

See also: function.

**likelihood function** The likelihood function of a probabilistic model with parameters  $\mathbf{w}$  is obtained by evaluating the pdf (or pmf) for the observed dataset [42, 67]. More precisely, consider a dataset  $\mathcal{D}' = \{\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(m)}\}$  that is assumed to be generated via i.i.d. RVs with a common pdf  $p(\mathbf{z}; \mathbf{w})(\cdot)$ . The common pdf is parametrized by the model parameters  $\mathbf{w}$ . The likelihood function is then defined by  $L^{(\mathcal{D}')}(w) = \prod_{r=1}^m p(\mathbf{z}; \mathbf{w})(\mathbf{z}^{(r)})$ . The likelihood function can be used to construct a loss function for ERM, which results in maximum likelihood.

See also: probabilistic model, maximum likelihood, loss function.

**linear map** A linear map  $f : \mathbb{R}^d \rightarrow \mathbb{R}^m$  is a function that satisfies additivity, i.e.,  $f(\mathbf{x} + \mathbf{y}) = f(\mathbf{x}) + f(\mathbf{y})$ , and homogeneity, i.e.,  $f(c\mathbf{x}) = cf(\mathbf{x})$ , for all vectors  $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$  and scalars  $c \in \mathbb{R}$ . In particular,  $f(\mathbf{0}) = \mathbf{0}$ . Any linear map can be represented as a matrix multiplication  $f(\mathbf{x}) = \mathbf{Ax}$  for some matrix  $\mathbf{A} \in \mathbb{R}^{m \times n}$ . The collection of real-valued linear maps (where  $m = 1$ ), for a given dimension  $d$ , constitutes a linear model. The notion of a linear map can be generalized from the domain  $\mathbb{R}^d$  and co-domain  $\mathbb{R}^m$  to arbitrary vector spaces.

See also: map, function, vector, matrix, linear model.

**linear operator** A linear operator is an operator  $\mathcal{F}$  whose domain and co-domain are vector spaces and which satisfies [68]

$$\mathcal{F}(\alpha\mathbf{w} + \beta\mathbf{w}') = \alpha\mathcal{F}(\mathbf{w}) + \beta\mathcal{F}(\mathbf{w}')$$

for all  $\mathbf{w}, \mathbf{w}'$  in the domain and all scalars  $\alpha, \beta$ .

See also: operator, vector space, matrix.

**linearly dependent** A subset  $\{\mathbf{a}^{(1)}, \dots, \mathbf{a}^{(d)}\} \in \mathcal{V}$  of a vector space  $\mathcal{V}$  over a field  $\mathbb{F}$  is linearly dependent if it is not linearly independent. This occurs if there exists a set of scalars  $\alpha_1, \dots, \alpha_d \in \mathbb{F}$ , not all equal to zero, such that

$$\sum_{j=1}^d \alpha_j \mathbf{a}^{(j)} = \mathbf{0}.$$

Intuitively, this means that at least one vector in the set can be expressed as a (non-trivial) linear combination of the others.

See also: vector space, linearly independent, vector.

**linearly independent** A subset  $\{\mathbf{a}^{(1)}, \dots, \mathbf{a}^{(d)}\} \in \mathcal{V}$  of a vector space is linearly independent if there is no nontrivial linear combination of these vectors that equals the zero vector [36]. In other words,

$$\sum_{j=1}^d \alpha_j \mathbf{a}^{(j)} = \mathbf{0} \quad \text{implies} \quad \alpha_1 = \alpha_2 = \dots = \alpha_d = 0.$$

See also: vector space, vector, dimension, basis, linearly dependent.

**Lipschitz continuity** A function between two metric spaces is Lipschitz continuous if there exists a constant  $L > 0$  such that the distance between the function values at any two points is at most  $L$  times the distance between those points [38].

See also: function, vector.

**majorize-minimize (MM)** Consider an optimization problem  $\min_{\mathbf{w} \in \mathcal{W}} f(\mathbf{w})$  with some complicated (potentially non-convex and non-smooth) objective function. One important example of such an optimization problem is ERM, which is used to learn the model parameters of a nonlinear model. An MM method is an iterative optimization method that constructs a sequence  $\mathbf{w}^{(1)}, \mathbf{w}^{(2)}, \dots \in \mathcal{W}$  of model parameters as follows [22], [69], [70] (see also Fig. 41): During the  $t$ th iteration, the objective function  $f(\cdot)$  is approximated by another function  $g(\cdot; \mathbf{w}^{(t)})$ . This approximation must be an upper bound for (i.e., must majorize) the original objective function, i.e.,  $g(\mathbf{w}; \mathbf{w}^{(t)}) \geq f(\mathbf{w})$  for all  $\mathbf{w} \in \mathcal{W}$ , and it must be tight for  $\mathbf{w}^{(t)}$ , i.e.,  $g(\mathbf{w}^{(t)}; \mathbf{w}^{(t)}) = f(\mathbf{w}^{(t)})$ . The new model

parameters  $\mathbf{w}^{(t+1)}$  are then obtained by minimizing the approximation, i.e.,  $\mathbf{w}^{(t+1)} \in \arg \min_{\mathbf{w} \in \mathcal{W}} g(\mathbf{w}; \mathbf{w}^{(t)})$ .

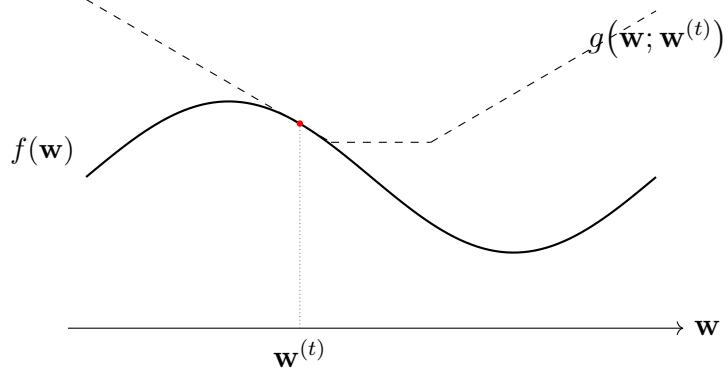


Fig. 41. The construction of model parameters based on the iterative MM method.

Similar to gradient-based methods, the MM principle is also based on approximating an objective function locally, around the current model parameters, and then optimizing this approximation to obtain new model parameters. However, the construction of local approximations is very different. While gradient-based methods use linear functions for these approximations, MM methods can use nonlinear functions as long as they are upper bounds for the original objective function.

See also: gradient-based method, expectation–maximization (EM).

**map** We use the term map as a synonym for function.

See also: function.

**marginal distribution** A marginal distribution refers to the probability distribution of a subset of the RVs belonging to a stochastic process [6, p.

266], [29]. For example, consider two RVs  $(x, y)$  with a joint probability distribution  $\mathbb{P}^{(x,y)}$ . This joint probability distribution fully determines the marginal distributions of  $x$  and  $y$ . The process of computing the marginal distribution of a subset of RVs is referred to as marginalization [71].

See also: probability distribution, RV, stochastic process, marginalization.

**marginalization** The term marginalization refers to the process of computing the probability distribution of a subset of RVs from the joint probability distribution of a stochastic process [71].

See also: probability distribution, RV, marginal distribution.

**Markov chain** A Markov chain is a stochastic process  $\{X_t\}_{t \in \mathbb{N}}$  defined on a common probability space and using the index set  $\mathbb{N}$ . The RV  $X_t$  might represent (the generation of) a state of a physical system at the time instant  $t$ . The defining property of a Markov chain is the Markov property [17], [29], [72]. For all  $t \in \mathbb{N}$ ,

$$\mathbb{P}^{(X_{t+1}|X_t, \dots, X_1)} = \mathbb{P}^{(X_{t+1}|X_t)}.$$

In other words, the conditional probability distribution of the next state  $X_{t+1}$  depends on the past  $X_t, X_{t-1}, \dots, X_1$  only through the current state  $X_t$ . The concept of a Markov chain can be generalized from discrete time (with index set  $\mathbb{N}$ ) to continuous time (with index set  $\mathbb{R}$ ) [72].

See also: stochastic process, conditional probability distribution, Markov decision process (MDP).

**Markov property** See Markov chain.

**Markov's inequality** Consider a real-valued nonnegative RV  $x$  for which the expectation  $\mathbb{E}\{x\}$  exists. Markov's inequality provides an upper bound on the probability  $\mathbb{P}(x \geq a)$  that  $x$  exceeds a given positive threshold  $a > 0$ . In particular,

$$\mathbb{P}(x \geq a) \leq \frac{\mathbb{E}\{x\}}{a} \quad \text{holds for any } a > 0.$$

This inequality can be verified by noting that  $\mathbb{P}(x \geq a)$  is the expectation  $\mathbb{E}\{g(x)\}$  with the following indicator function:

$$g : \mathbb{R} \rightarrow \mathbb{R} : x' \mapsto \mathbb{I}_{\{x \geq a\}}(x').$$

As illustrated in Fig. 42, for any positive  $a > 0$ ,

$$g(x') \leq x'/a \quad \text{for all } x' \in \mathbb{R}.$$

This implies Markov's inequality via the monotonicity property of the Lebesgue integral [13, p. 50].

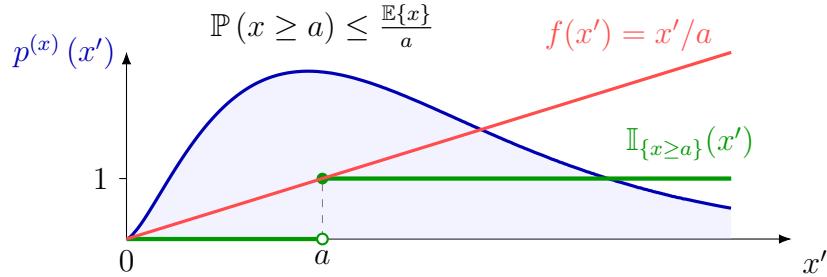


Fig. 42. The expectation  $\mathbb{E}\{x\}$  and the probability  $\mathbb{P}(x \geq a)$  of a nonnegative RV  $x$  with a pdf  $p^{(x)}(\cdot)$  can be obtained via Lebesgue integrals of  $f(x') = x'/a$  and  $g(x') = \mathbb{I}_{\{x \geq a\}}(x')$ , respectively.

See also: expectation, probability, concentration inequality.

**matrix** A matrix of size  $m \times d$  is a 2-D array of numbers, which is denoted by

$$\mathbf{A} = \begin{pmatrix} A_{1,1} & A_{1,2} & \dots & A_{1,d} \\ A_{2,1} & A_{2,2} & \dots & A_{2,d} \\ \vdots & \vdots & \ddots & \vdots \\ A_{m,1} & A_{m,2} & \dots & A_{m,d} \end{pmatrix} \in \mathbb{R}^{m \times d}.$$

Here,  $A_{r,j}$  denotes the matrix entry in the  $r$ th row and the  $j$ th column. Matrices are useful representations of various mathematical objects [36], including the following:

- Systems of linear equations: We can use a matrix to represent a system of linear equations

$$\begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} \text{ compactly as } \mathbf{Aw} = \mathbf{y}.$$

One important example of systems of linear equations is the optimality condition for the model parameters within linear regression.

- Linear maps: Consider a  $d$ -dimensional vector space  $\mathcal{U}$  and a  $m$ -dimensional vector space  $\mathcal{V}$ . If we fix a basis  $\mathbf{u}^{(1)}, \dots, \mathbf{u}^{(d)}$  for  $\mathcal{U}$  and a basis  $\mathbf{v}^{(1)}, \dots, \mathbf{v}^{(m)}$  for  $\mathcal{V}$ , each matrix  $\mathbf{A} \in \mathbb{R}^{m \times d}$  naturally defines a linear map  $\alpha : \mathcal{U} \rightarrow \mathcal{V}$  (see Fig. 43) such that

$$\mathbf{u}^{(j)} \mapsto \sum_{r=1}^m A_{r,j} \mathbf{v}^{(r)}.$$

- Datasets: We can use a matrix to represent a dataset. Each row corresponds to a single data point, and each column corresponds to a specific feature or label of a data point.

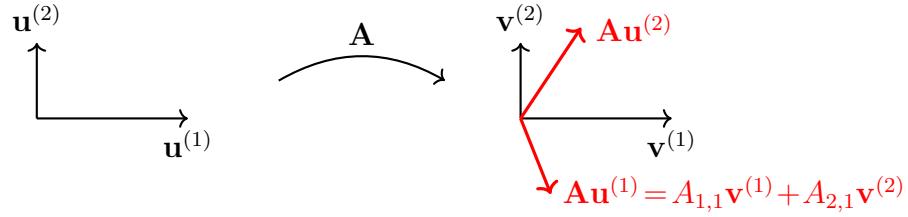


Fig. 43. A matrix  $\mathbf{A}$  defines a linear map between two vector spaces.

See also: linear map, dataset, linear model.

**maximum** The maximum of a set  $\mathcal{A} \subseteq \mathbb{R}$  of real numbers is the greatest element in that set if such an element exists. A set  $\mathcal{A}$  has a maximum if it is bounded above and attains its supremum (or least upper bound) [2, Sec. 1.4].

See also: supremum.

**maximum likelihood** Consider data points  $\mathcal{D} = \{\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(m)}\}$  that are interpreted as the realizations of i.i.d. RVs with a common probability distribution  $\mathbb{P}^{(\mathcal{D}; \mathbf{w})}$ , which depends on the model parameters  $\mathbf{w} \in \mathcal{W} \subseteq \mathbb{R}^n$ . Maximum likelihood methods learn model parameters  $\mathbf{w}$  by maximizing the pdf (or pmf)  $p^{(\mathcal{D}; \mathbf{w})}(\mathcal{D}') = \prod_{r=1}^m p^{(\mathbf{z}; \mathbf{w})}(\mathbf{z}^{(r)})$  of the observed dataset  $\mathcal{D}'$ . Thus, the maximum likelihood estimator is a solution to the optimization problem  $\max_{\mathbf{w} \in \mathcal{W}} p^{(\mathcal{D}; \mathbf{w})}(\mathcal{D}')$ .

See also: probability distribution, optimization problem, probabilistic model.

**mean** The mean of an RV  $\mathbf{x}$ , which takes on values in a Euclidean space  $\mathbb{R}^d$ , is its expectation  $\mathbb{E}\{\mathbf{x}\}$ . It is defined as the Lebesgue integral of  $\mathbf{x}$

with respect to the underlying probability distribution  $\mathbb{P}(\cdot)$  (e.g., see [2] or [6]), i.e.,

$$\mathbb{E}\{\mathbf{x}\} = \int_{\mathbb{R}^d} \mathbf{x} dP(\mathbf{x}).$$

We also use the term to refer to the sample mean of a finite dataset  $\mathcal{D} = \{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)} \in \mathbb{R}^d\}$ . However, these two definitions are essentially the same. Indeed, we can use a dataset to construct a discrete RV  $\tilde{\mathbf{x}}^{(\mathcal{D})} = \mathbf{x}^{(I)}$  on the sample space  $\{1, \dots, m\}$ . Here, the index  $I$  is chosen uniformly at random, i.e.,  $\mathbb{P}(I = r) = 1/m$  for all  $r = 1, \dots, m$ . The mean of  $\tilde{\mathbf{x}}^{(\mathcal{D})}$  is precisely the average  $(1/m) \sum_{r=1}^m \mathbf{x}^{(r)}$ . For an RV with a finite second-order moment, i.e.,  $\mathbb{E}\{\|\mathbf{x}\|_2^2\}$  is well defined and finite, the mean is characterized as the solution of the following risk minimization problem [7]:

$$\mathbb{E}\{\mathbf{x}\} = \arg \min_{\mathbf{c} \in \mathbb{R}^d} \mathbb{E}\{\|\mathbf{x} - \mathbf{c}\|_2^2\}.$$

For the RV  $\tilde{\mathbf{x}}^{(\mathcal{D})}$ , associated with a dataset  $\mathcal{D}$ , this optimization problem reduces to ERM with squared error loss on  $\mathcal{D}$ .

See also: RV, expectation, probability distribution, ERM.

**measurable** Consider a random experiment, such as recording the air temperature at an FMI weather station. The corresponding sample space  $\Omega$  consists of all possible outcomes  $\omega$  (e.g., all possible temperature values in degree Celsius). In many ML applications, we are not interested in the exact outcome  $\omega$ , but only whether it belongs to a subset  $\mathcal{A} \subseteq \Omega$  (e.g., determining whether the temperature is below zero degrees). We call such a subset  $\mathcal{A}$  measurable if it is possible to decide, for any outcome  $\omega$ , whether  $\omega \in \mathcal{A}$  (see Fig. 44).

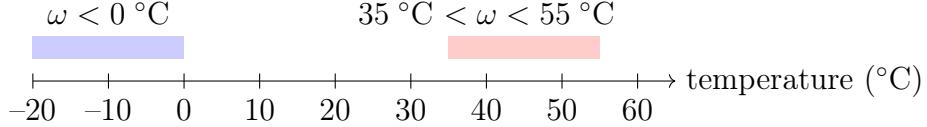


Fig. 44. A sample space constituted by all possible temperature values  $\omega$  that can occur at an FMI station. Two measurable subsets of temperature values, denoted by  $\mathcal{A}^{(1)}$  and  $\mathcal{A}^{(2)}$ , are highlighted. For any actual temperature value  $\omega$ , it is possible to determine (via some equipment) whether  $\omega \in \mathcal{A}^{(1)}$  and whether  $\omega \in \mathcal{A}^{(2)}$ .

In principle, measurable sets could be chosen freely (e.g., depending on the resolution of the measuring equipment). However, it is often useful to impose certain completeness requirements on the collection of measurable sets. For example, the sample space itself should be measurable, and the union of two measurable sets should also be measurable. These completeness requirements can be formalized via the concept of a  $\sigma$ -algebra (or  $\sigma$ -field) [1], [6], [17]. A measurable space is a pair  $(\mathcal{X}, \mathcal{F})$  that consists of an arbitrary set  $\mathcal{X}$  and a collection  $\mathcal{F}$  of measurable subsets of  $\mathcal{X}$  that form a  $\sigma$ -algebra.

See also: sample space, outcome,  $\sigma$ -algebra, probability.

**measure** A measure  $\mu$  on a set  $\Omega$  equipped with a  $\sigma$ -algebra  $\Sigma$  is a function  $\mu : \Sigma \rightarrow [0, \infty)$  that assigns a nonnegative value to each measurable set  $\mathcal{A} \in \Sigma$  such that [2], [6], [73]: 1)  $\mu(\emptyset) = 0$ ; and 2) for any countable

collection  $\{\mathcal{A}_i\}_{i=1}^{\infty}$  of pairwise disjoint sets in  $\Sigma$ ,

$$\mu\left(\bigcup_{i=1}^{\infty} A_i\right) = \sum_{i=1}^{\infty} \mu(A_i),$$

which is referred to as “countable additivity”.

See also: measurable, countable.

**measure space** A measure space is a triple  $(\Omega, \Sigma, \mu)$  consisting of a set  $\Omega$ , a  $\sigma$ -algebra  $\Sigma$  of subsets of  $\Omega$ , and a measure  $\mu : \Sigma \rightarrow [0, \infty)$ .

The measure  $\mu$  assigns a nonnegative number to each measurable set  $\mathcal{A} \in \Sigma$ , generalizing the notions of length, area, or volume in Euclidean spaces [2], [73]. Measure spaces provide the mathematical foundation for the Lebesgue integral or the definition of RVs as measurable mappings between measure spaces. A probability space is a special case of a measure space where the total measure of the sample space is normalized to one, i.e.,  $\mu(\Omega) = 1$ . In this case,  $\mu$  is called a probability distribution.

See also: measurable, probability space, probability distribution.

**median** A median  $\text{med}(x)$  of a real-valued RV  $x$  is any number  $M \in \mathbb{R}$  such that  $\mathbb{P}(x \leq M) \geq 1/2$  and  $\mathbb{P}(x \geq M) \geq 1/2$  (see Fig. 45) [42].

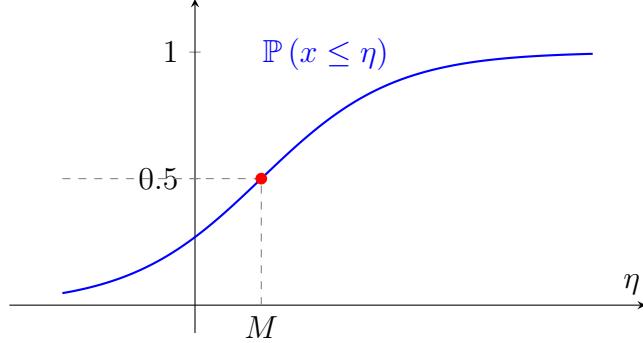


Fig. 45. The median of a real-valued RV is any number  $M$  that partitions  $\mathbb{R}$  into two rays with equal probability.

We can define the median  $\text{med}(\mathcal{D})$  of a dataset  $\mathcal{D} = \{x^{(1)}, \dots, x^{(m)} \in \mathbb{R}\}$  via a specific RV  $\tilde{x}^{(\mathcal{D})}$  that is naturally associated with  $\mathcal{D}$ . In particular, this RV is defined on the sample space  $\{1, \dots, m\}$  via  $\tilde{x}^{(\mathcal{D})} := x^{(I)}$ . Here, the index  $I$  is chosen uniformly at random, i.e.,  $\mathbb{P}(I = r) = 1/m$  for all  $r = 1, \dots, m$ . If the RV  $x$  is integrable, any median of  $x$  solves the following optimization problem:

$$\min_{x' \in \mathbb{R}} \mathbb{E}|x - x'|.$$

For the above RV  $\tilde{x}$  (constructed from a dataset  $\mathcal{D}$ ), this optimization problem is ERM on  $\mathcal{D}$  using absolute error loss. Like the mean, the median of a dataset  $\mathcal{D}$  can also be used to estimate parameters of an underlying probabilistic model. Compared with the mean, the median is more robust to outliers. For example, the median of a dataset  $\mathcal{D}$  with more than one data point does not change even if we arbitrarily increase the largest element of  $\mathcal{D}$  (see Fig. 46). In contrast, the mean will increase arbitrarily.

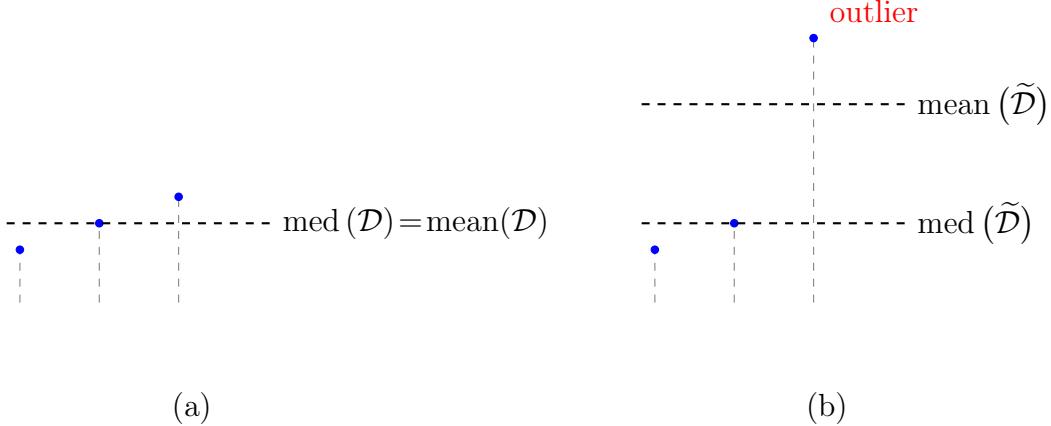


Fig. 46. The median is robust against outlier contamination. (a) Original dataset  $\mathcal{D}$ . (b) Noisy dataset  $\tilde{\mathcal{D}}$  including an outlier.

See also: mean, outlier, robustness, least absolute deviation regression.

**method of multipliers (MoM)** The MoM is an iterative optimization method for solving a constrained optimization problem of the following form [74]:

$$\begin{aligned} \min_{\mathbf{x} \in \mathbb{R}^d} \quad & f(\mathbf{x}) \\ \text{s.t. } \mathbf{A}\mathbf{x} = \mathbf{b}. \end{aligned}$$

Here,  $f : \mathbb{R}^d \rightarrow \mathbb{R}$  denotes the objective function,  $\mathbf{A} \in \mathbb{R}^{m \times d}$  is a given matrix, and  $\mathbf{b} \in \mathbb{R}^m$  is a given vector. The MoM is based on the augmented Lagrangian

$$L_\rho(\mathbf{x}, \mathbf{y}) = f(\mathbf{x}) + \mathbf{y}^\top \mathbf{A}\mathbf{x} + \frac{\rho}{2} \|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2,$$

where  $\mathbf{y}$  denotes the vector of Lagrange multipliers and  $\rho > 0$  is a penalty parameter. The MoM constructs a sequence of estimates  $(\mathbf{x}^{(1)}, \mathbf{y}^{(1)}), \dots$

that converge to a solution of the optimization problem. In particular, during each iteration  $t$ , the current estimates  $\mathbf{x}^{(k)}, \mathbf{y}^{(k)}$  are updated as follows:

$$\begin{aligned}\mathbf{x}^{(k+1)} &= \arg \min_{\mathbf{x} \in \mathbb{R}^d} L_\rho(\mathbf{x}, \mathbf{y}^{(k)}), \\ \mathbf{y}^{(k+1)} &= \mathbf{y}^{(k)} + \rho (\mathbf{A}\mathbf{x}^{(k+1)} - \mathbf{b}).\end{aligned}$$

The MoM can be written as a fixed-point iteration of the following form:

$$(\mathbf{x}^{(k+1)}, \mathbf{y}^{(k+1)}) = \mathcal{F}(\mathbf{x}^{(k)}, \mathbf{y}^{(k)})$$

with

$$\begin{aligned}\mathcal{F} : \mathbb{R}^d \times \mathbb{R}^m &\rightarrow \mathbb{R}^d \times \mathbb{R}^m : (\mathbf{x}, \mathbf{y}) \mapsto (\mathbf{x}', \mathbf{y} + \rho(\mathbf{A}\mathbf{x}' - \mathbf{b})) \\ \text{with } \mathbf{x}' &= \arg \min_{\mathbf{x} \in \mathbb{R}^d} L_\rho(\mathbf{x}, \mathbf{y}).\end{aligned}$$

See also: optimization problem, augmented Lagrangian, fixed-point iteration.

**metric** A metric is a quantitative measure used to compare objects. In mathematics, a metric measures the distance between two points in a space and must follow specific rules, i.e., the distance is always nonnegative, zero only if the points are the same, symmetric, and it satisfies the triangle inequality [2]. In the context of ML, the term metric refers to a quantitative measure of how well a model performs (somewhat similar to a loss function). Examples include accuracy, precision, and the average 0/1 loss on a test set [30], [22]. The term loss function is typically used in the context of model training, while

the term metric is used in the context of model validation.

See also: accuracy, precision, validation, loss.

**metric space** A metric space is a set  $\mathcal{X}$  equipped with a function (referred to as a metric)  $d(\cdot, \cdot) : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}_+$  that satisfies the following requirements for all  $\mathbf{x}, \mathbf{x}', \mathbf{x}'' \in \mathcal{X}$ :

1. Nonnegativity:  $d(\mathbf{x}, \mathbf{x}') \geq 0$ ;
2. Identity:  $d(\mathbf{x}, \mathbf{x}') = 0$  if and only if  $\mathbf{x} = \mathbf{x}'$ ;
3. Symmetry:  $d(\mathbf{x}, \mathbf{x}') = d(\mathbf{x}', \mathbf{x})$ ;
4. Triangle inequality:  $d(\mathbf{x}, \mathbf{x}'') \leq d(\mathbf{x}, \mathbf{x}') + d(\mathbf{x}', \mathbf{x}'')$ .

Formally, a metric space is a pair  $(\mathcal{X}, d(\cdot, \cdot))$  that satisfies the above requirements [38].

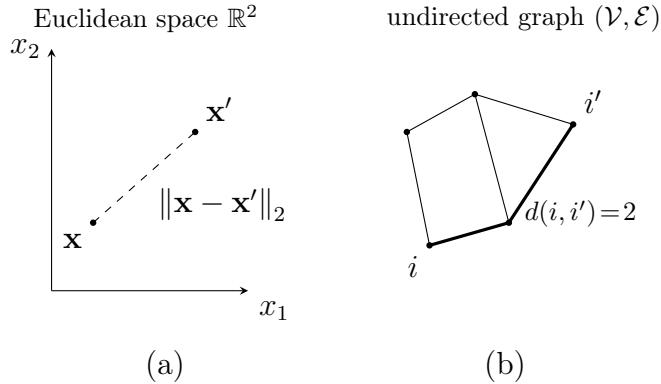


Fig. 47. Examples of metric spaces. (a) Euclidean space  $\mathbb{R}^2$  with the Euclidean distance as a metric. (b) Undirected graph  $(\mathcal{V}, \mathcal{E})$  with the shortest-path distance as a metric.

A prominent example of a metric space is the Euclidean space equipped

with a metric given by the Euclidean distance  $d(\mathbf{x}, \mathbf{x}') = \|\mathbf{x} - \mathbf{x}'\|_2$ . Another well-known example of a metric space is an undirected graph  $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ , with the metric  $d(i, i')$  defined by the length of the shortest path connecting nodes  $i$  and  $i'$ .

See also: Euclidean space, undirected graph, feature space.

**minimum** Given a set of real numbers, the minimum is the smallest of those numbers. Note that for some sets, such as the set of negative real numbers, the minimum does not exist.

**mirror descent** Mirror descent is an iterative optimization method obtained by generalizing the gradient step. The gradient step for minimizing a differentiable objective function  $f : \mathbb{R}^d \rightarrow \mathbb{R}$  can be written as

$$\mathbf{w}^{(k+1)} = \arg \min_{\mathbf{w} \in \mathbb{R}^d} \langle \nabla f(\mathbf{w}^{(k)}), \mathbf{w} \rangle + \frac{1}{2\eta} \|\mathbf{w} - \mathbf{w}^{(k)}\|_2^2.$$

Thus, a gradient step minimizes a linearization of  $f(\cdot)$  penalized by a scaled squared Euclidean norm  $\frac{1}{\eta} \|\mathbf{w} - \mathbf{w}^{(k)}\|_2^2$ . The scaling factor is the inverse of the learning rate  $\eta$  used in the gradient step. Mirror descent replaces the squared Euclidean norm with a Bregman divergence  $D_\phi(\cdot, \cdot)$  induced by a strictly convex function  $\phi(\cdot)$ . This mirror map is typically defined on a convex set  $\mathcal{W} \subseteq \mathbb{R}^d$  and is differentiable and strictly convex on the interior of  $\mathcal{W}$  [64]. The resulting update becomes

$$\mathbf{w}^{(k+1)} = \arg \min_{\mathbf{w} \in \mathcal{W}} \langle \nabla f(\mathbf{w}^{(k)}), \mathbf{w} \rangle + \frac{1}{\eta} D_\phi(\mathbf{w}, \mathbf{w}^{(k)}).$$

See also: gradient step, Bregman divergence, inner product, zero-gradient condition, proximal operator.

**moment generating function (MGF)** Consider the MGF  $M_x(t)$  of a real-valued RV  $x$ , which is defined as  $M_x(t) = \mathbb{E}\{\exp(t \cdot x)\}$  for any  $t \in \mathbb{R}$  for which this expectation exists [6, Sec. 21]. As its name indicates, the MGF allows us to compute the moments  $\mathbb{E}\{x^k\}$  for  $k \in \mathbb{N}$ . In particular, the  $k$ th moment is obtained by evaluating the  $k$ th derivative of  $M_x(t)$  for  $t = 0$ , i.e.,  $\mathbb{E}\{x^k\} = M_x^{(k)}(0)$ . This fact can be verified by the following identities:

$$\begin{aligned} M_x(t) &= \mathbb{E}\{\exp(t \cdot x)\} \\ &\stackrel{(a)}{=} \mathbb{E}\left\{\sum_{k=0}^{\infty} \frac{t^k}{k!} x^k\right\} \\ &\stackrel{(b)}{=} \sum_{k=0}^{\infty} \frac{t^k}{k!} \mathbb{E}\{x^k\}. \end{aligned}$$

Here, step (a) is due to the Taylor series expansion of  $\exp(t \cdot x)$  and step (b) is valid when the MGF exists for all  $t$  in some interval  $(-t_0, t_0)$  [6, p. 278].

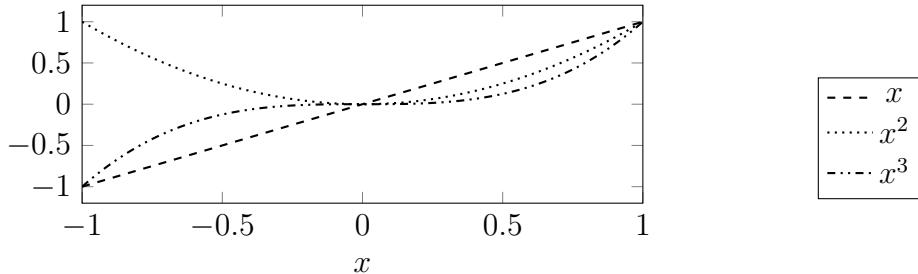


Fig. 48. The first few powers of an RV  $x$ . The MGF encodes the moments of  $x$ , which are the expectations of the powers  $x^k$  for  $k = 1, 2, \dots$ .

The MGF is a useful tool for the study of sums of independent RVs. As a case in point, if  $x$  and  $y$  are independent RVs, then the MGF of

their sum  $z = x + y$  typically satisfies  $M_z(t) = M_x(t)M_y(t)$ , i.e., the MGF of the sum is typically the pointwise product of the individual MGFs [6, p. 280].

See also: RV, expectation.

**multivariate normal distribution** The multivariate normal distribution, which is denoted by  $\mathcal{N}(\boldsymbol{\mu}, \mathbf{C})$ , is a fundamental probabilistic model for numerical feature vectors of fixed dimension  $d$ . It defines a family of probability distributions over vector-valued RVs  $\mathbf{x} \in \mathbb{R}^d$  [7], [18], [50]. Each distribution in this family is fully specified by its mean vector  $\boldsymbol{\mu} \in \mathbb{R}^d$  and covariance matrix  $\mathbf{C} \in \mathbb{R}^{d \times d}$ . When the covariance matrix  $\mathbf{C}$  is invertible, the corresponding probability distribution is characterized by the following pdf:

$$p(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^d \det(\mathbf{C})}} \exp \left[ -\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu})^T \mathbf{C}^{-1} (\mathbf{x} - \boldsymbol{\mu}) \right].$$

Note that this pdf is only defined when  $\mathbf{C}$  is invertible. More generally, any RV  $\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}, \mathbf{C})$  admits the following representation:

$$\mathbf{x} = \mathbf{A}\mathbf{z} + \boldsymbol{\mu},$$

where  $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$  is a standard normal random vector and  $\mathbf{A} \in \mathbb{R}^{d \times d}$  satisfies  $\mathbf{A}\mathbf{A}^\top = \mathbf{C}$ . This representation remains valid even when  $\mathbf{C}$  is singular, in which case  $\mathbf{A}$  is not full-rank [75, Ch. 23]. The family of multivariate normal distributions is exceptional among probabilistic models for numerical quantities, at least for the following reasons. First, the family is closed under affine transformations, i.e.,

$$\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}, \mathbf{C}) \text{ implies } \mathbf{B}\mathbf{x} + \mathbf{c} \sim \mathcal{N}(\mathbf{B}\boldsymbol{\mu} + \mathbf{c}, \mathbf{B}\mathbf{C}\mathbf{B}^T).$$

Second, the probability distribution  $\mathcal{N}(\mathbf{0}, \mathbf{C})$  maximizes the differential entropy among all distributions with the same covariance matrix  $\mathbf{C}$  [35]. See also: probabilistic model, probability distribution, standard normal random vector, differential entropy, Gaussian RV.

**neighborhood** Consider some metric space  $\mathcal{X}$  with a metric  $d : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}_+$ .

The neighborhood of a point  $\mathbf{x} \in \mathcal{X}$  is the set of other points having a sufficiently small distance to  $\mathbf{x}$ . For example, the  $\epsilon$ -neighborhood of  $\mathbf{x}$  is defined as

$$\{\mathbf{x}' \in \mathcal{X} : d(\mathbf{x}, \mathbf{x}') \leq \epsilon\}.$$

If  $\mathcal{X}$  is an undirected graph, which is a special case of a metric space, the neighborhood of a node  $i \in \mathcal{V}$  is the set of its neighbors.

See also: metric, neighbor.

**Newton's method** Newton's method is an iterative optimization method for finding local minima or maxima of a differentiable objective function  $f(\mathbf{w})$ . Like gradient-based methods, Newton's method also computes a new estimate  $\hat{\mathbf{w}}_{t+1}$  by optimizing a local approximation of  $f(\mathbf{w})$  around the current estimate  $\hat{\mathbf{w}}_t$ . In contrast to gradient-based methods, which use the gradient to build a local linear approximation, Newton's method uses the Hessian matrix to build a local quadratic approximation. In particular, starting from an initial estimate  $\hat{\mathbf{w}}_0$ , Newton's method iteratively updates the estimate as follows:

$$\hat{\mathbf{w}}_{t+1} = \hat{\mathbf{w}}_t - (\nabla^2 f(\hat{\mathbf{w}}_t))^{-1} \nabla f(\hat{\mathbf{w}}_t) \quad \text{for } t = 0, 1, \dots$$

Here,  $\nabla f(\hat{\mathbf{w}}_t)$  is the gradient, and  $\nabla^2 f(\mathbf{w}^{(t)})$  is the Hessian of the

objective function  $f$ . Since using a quadratic function as a local approximation is more accurate than using a linear function (which is a special case of a quadratic function), Newton's method tends to converge faster than gradient-based methods (see Fig. 49). However, this faster convergence comes at the increased computational complexity of the iterations. Indeed, each iteration of Newton's method requires the inversion of the Hessian.

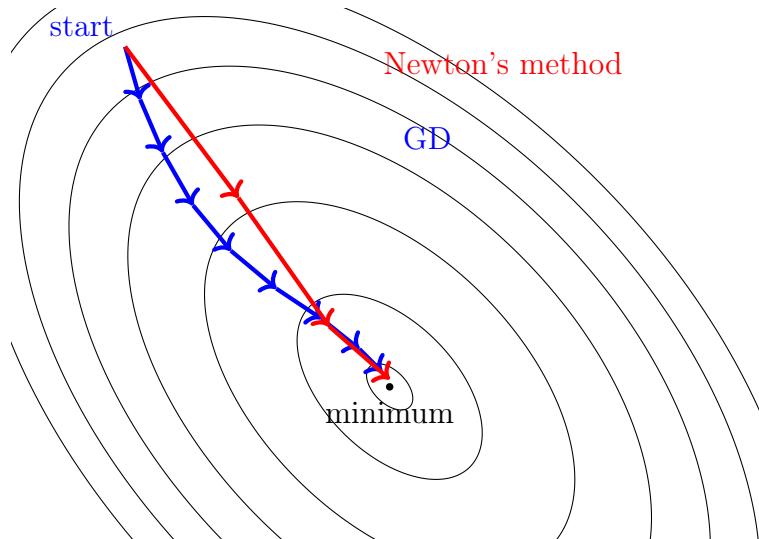


Fig. 49. Comparison of GD (blue) and Newton's method (red) paths toward the minimum of a loss function.

See also: optimization method, gradient, Hessian, GD.

**node degree** The degree  $d^{(i)}$  of a node  $i \in \mathcal{V}$  in an undirected graph is the number of its neighbors, i.e.,  $d^{(i)} := |\mathcal{N}^{(i)}|$ . For a weighted undirected graph  $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{A})$ , we can alternatively define the (weighted) node degree as the sum of the weights of all edges connected to node  $i$ , i.e.,

$$d^{(i)} = \sum_{i' \in \mathcal{V}} A_{i,i'} \quad [76].$$

See also: undirected graph, neighbor, average node degree.

**non-expansive operator** An operator  $\mathcal{F} : \mathcal{V} \rightarrow \mathcal{V}$  defined on a normed space  $(\mathcal{V}, \|\cdot\|)$  is called non-expansive if it does not increase distances. In other words,

$$\|\mathcal{F}\mathbf{w} - \mathcal{F}\mathbf{w}'\| \leq \|\mathbf{w} - \mathbf{w}'\| \quad \text{for any } \mathbf{w}, \mathbf{w}' \in \mathcal{V}.$$

Non-expansiveness is typically not sufficient to guarantee convergence of a fixed-point iteration that uses  $\mathcal{F}$  (see Fig. 50).

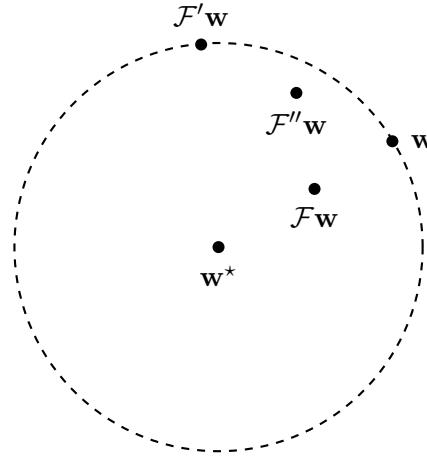


Fig. 50. The result of applying a contractive operator  $\mathcal{F}$ , a non-expansive operator  $\mathcal{F}'$ , and a firmly non-expansive operator  $\mathcal{F}''$ . These operators have the common fixed point  $\mathbf{w}^*$ .

See also: fixed-point iteration, contractive operator.

**non-smooth** We refer to a function as non-smooth if it is not smooth [52].

See also: function, smooth.

**norm** A norm is a function that maps each (vector) element of a vector space to a nonnegative real number. Formally, a norm  $\|\cdot\|$  on a vector space  $\mathcal{X}$  over a field  $\mathbb{F}$  is a function  $\|\cdot\| : \mathcal{X} \rightarrow \mathbb{R}_+$  that satisfies the following conditions [33] for all  $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$  and  $\alpha \in \mathbb{F}$ :

- Definiteness:  $\|\mathbf{x}\| = 0 \iff \mathbf{x} = \mathbf{0}$ ;
- Homogeneity:  $\|\alpha \mathbf{x}\| = |\alpha| \cdot \|\mathbf{x}\|$ ;
- Triangle inequality:  $\|\mathbf{x} + \mathbf{x}'\| \leq \|\mathbf{x}\| + \|\mathbf{x}'\|$ .

Intuitively, norms measure the length or size of a vector. Norms can also be used to measure distances between vectors, as they define a metric by  $d(\mathbf{x}, \mathbf{x}') = \|g\|_{\text{euclidean}} \mathbf{x} - \mathbf{x}'$ , making  $(\mathcal{X}, d(\cdot, \cdot))$  a metric space. A prominent family of norms used in ML is the  $\ell_p$ -norms for  $p \geq 1$ , defined for a vector  $\mathbf{x} \in \mathbb{R}^d$  as  $\|\mathbf{x}\|_p = \left( \sum_{i=1}^d |x_i|^p \right)^{1/p}$ . Important instances include the  $\ell_1$ -norm, the  $\ell_2$ -norm (or Euclidean norm), and the  $\ell_\infty$ -norm  $\|\mathbf{x}\|_\infty = \max_i |x_i|$ , which are illustrated by the geometry of their respective unit spheres in Fig. 51. In ML, norms are fundamental, for instance for defining a loss function or a regularizer.

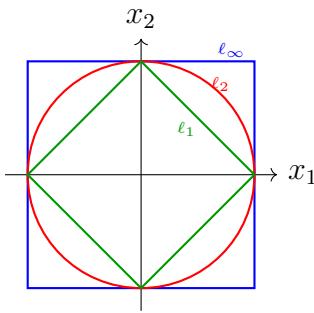


Fig. 51. Unit spheres  $\mathbb{S}^{(1)}$  with respect to different  $\ell_p$ -norms for  $p = 1, 2, \infty$ .

See also: Euclidean space, Hilbert space, inner product, linear regression, Lasso.

**normal distribution** See Gaussian RV, multivariate normal distribution, CLT.

**normal matrix** A normal matrix is a square matrix  $\mathbf{A} \in \mathbb{C}^{d \times d}$  that commutes with its conjugate transpose, i.e.,  $\mathbf{A}\mathbf{A}^H = \mathbf{A}^H\mathbf{A}$ . Normal matrices admit an orthonormal basis of eigenvectors and are unitarily diagonalizable.

See also: matrix, diagonalizable.

**normal vector** See hyperplane.

**normed space** A normed space is a vector space  $\mathcal{V}$  equipped with a norm  $\|\cdot\|$ . Formally, it is denoted as the pair  $(\mathcal{V}, \|\cdot\|)$ . Every normed space is a metric space with the induced metric defined by  $d(\mathbf{x}, \mathbf{x}') = \|\mathbf{x} - \mathbf{x}'\|$  for all  $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$ . In ML, normed spaces are a common framework for feature spaces and parameter spaces, most typically the Euclidean spaces  $\mathbb{R}^d$  with the Euclidean norm.

See also: norm, vector space, metric space, Euclidean space, Hilbert space.

**nullspace** The nullspace of a matrix  $\mathbf{A} \in \mathbb{R}^{d' \times d}$ , denoted by  $\text{null}(\mathbf{A})$ , is the set of all vectors  $\mathbf{n} \in \mathbb{R}^d$  such that

$$\mathbf{A}\mathbf{n} = \mathbf{0}.$$

Consider a feature learning method that uses the matrix  $\mathbf{A}$  to transform a feature vector  $\mathbf{x} \in \mathbb{R}^d$  of a data point into a new feature vector

$\mathbf{z} = \mathbf{Ax} \in \mathbb{R}^{d'}$ . The nullspace  $\text{null}(\mathbf{A})$  characterizes all directions in the original feature space  $\mathbb{R}^d$  along which the transformation  $\mathbf{Ax}$  remains unchanged. In other words, adding any vector from the nullspace to a feature vector  $\mathbf{x}$  does not affect the transformed representation  $\mathbf{z}$ . This property can be exploited to enforce invariances in the predictions (computed from  $\mathbf{Ax}$ ). Fig. 52 illustrates one such invariance. It shows rotated versions of two handwritten digits, which approximately lie along 1-D curves in the original feature space. These curves are aligned with a direction vector  $\mathbf{n} \in \mathbb{R}^d$ . To ensure that the trained model is invariant to such rotations, we can choose the transformation matrix  $\mathbf{A}$  such that  $\mathbf{n} \in \text{null}(\mathbf{A})$ . This ensures that  $\mathbf{Ax}$ , and hence the resulting prediction, is approximately insensitive to rotations of the input image.

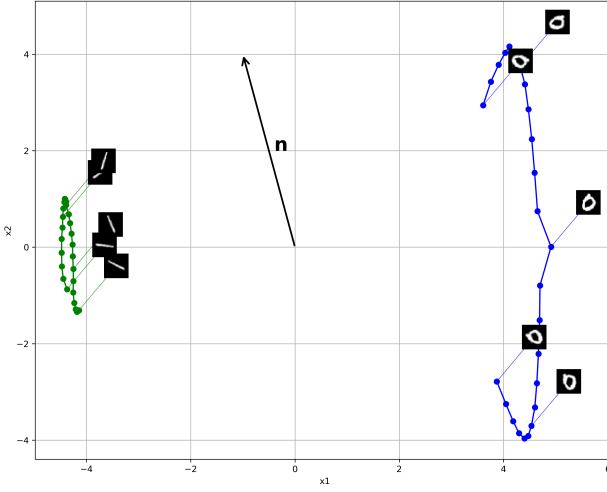


Fig. 52. Rotated handwritings of two different digits. The rotations are approximately aligned along straight lines parallel to the vector  $\mathbf{n}$ . For a binary classifier distinguishing between these digits, a natural choice is a linear feature map  $\mathbf{x} \mapsto \mathbf{Ax}$  with a matrix  $\mathbf{A}$  whose nullspace contains  $\mathbf{n}$ , i.e.,  $\mathbf{n} \in \text{null}(\mathbf{A})$ .

See also: matrix, feature map, feature learning.

Python demo: [click me](#)

**operator** An operator is a function whose domain and co-domain have a specific mathematical structure such as a vector space, a Hilbert space, or a metric space [48], [68]. Many ML methods involve operators whose domain and co-domain are Euclidean spaces.

See also: function, vector space, Hilbert space.

**optimization** See convex optimization, optimization method, optimization problem.

**optimization method** An optimization method is an algorithm that takes a representation of an optimization problem as input and computes an (approximate) solution as its output. A central example of an optimization problem in ML is ERM. By applying an appropriate optimization method to ERM, we obtain a concrete learning algorithm [25], [52], [77].

See also: algorithm, optimization problem.

**optimization problem** An optimization problem is a mathematical structure consisting of an objective function  $f : \mathcal{U} \rightarrow \mathcal{V}$  defined over an optimization variable  $\mathbf{w} \in \mathcal{U}$ , together with a feasible set  $\mathcal{W} \subseteq \mathcal{U}$ . The co-domain  $\mathcal{V}$  is assumed to be ordered, meaning that for any two elements  $\mathbf{a}, \mathbf{b} \in \mathcal{V}$ , we can determine whether  $\mathbf{a} \leq \mathbf{b}$ . Here, “ $\leq$ ” denotes a general partial order relation, which may differ from the standard numerical order on real numbers [25, Sec. 2.4]. The goal in optimization is to find those values  $\mathbf{w} \in \mathcal{W}$  for which the objective  $f(\mathbf{w})$  is extremal—i.e., minimal or maximal [25], [52], [77].

See also: objective function.

**orthogonality condition** Consider a linear subspace  $\mathcal{W} \subseteq \mathbb{R}^d$  and a vector  $\mathbf{w} \in \mathbb{R}^d$ . A vector  $\mathbf{w}^* \in \mathcal{W}$  is the projection  $P_{\mathcal{W}}(\mathbf{w})$  if and only if

$$\mathbf{v}^T (\mathbf{w} - \mathbf{w}^*) = 0 \quad \text{for all } \mathbf{v} \in \mathcal{W}.$$

In other words, the projection error  $\mathbf{w} - \mathbf{w}^*$  is orthogonal to the subspace  $\mathcal{W}$ . The notion of orthogonality can be extended to general Hilbert spaces, i.e., vector spaces with inner product.

See also: vector, projection, Euclidean space.

**outcome** Outcome is one possible result of a physical process. Such a process could be the observation of a physical phenomenon, a computation performed by an algorithm, or a random experiment [6].

See also: sample space.

**partial derivative** Consider a real-valued function  $f : \mathbb{R}^d \rightarrow \mathbb{R} : \mathbf{x} \mapsto f(\mathbf{x})$ .

The partial derivative of  $f(\mathbf{x})$  with respect to the entry  $x_j$  measures how  $f$  changes when  $x_j$  varies while all other entries  $x_{j'}$ , for  $j' \in [d] \setminus \{j\}$ , are held fixed. It is defined as [2]

$$\frac{\partial f(\mathbf{x})}{\partial x_j} = \lim_{\varepsilon \rightarrow 0} \frac{f(x_1, \dots, x_j + \varepsilon, \dots, x_d) - f(x_1, \dots, x_j, \dots, x_d)}{\varepsilon}.$$

Note that the partial derivative is only defined if this limit exists. For a differentiable function, the partial derivatives of  $f$  are the entries of the gradient  $\nabla f$ .

See also: function, derivative, gradient.

**positive semi-definite (psd)** A (real-valued) symmetric matrix  $\mathbf{Q} = \mathbf{Q}^T \in \mathbb{R}^{d \times d}$  is referred to as psd if  $\mathbf{x}^T \mathbf{Q} \mathbf{x} \geq 0$  for every vector  $\mathbf{x} \in \mathbb{R}^d$ . A psd matrix  $\mathbf{Q}$  admits a spectral decomposition with nonnegative eigenvalues  $\lambda_1, \dots, \lambda_d \geq 0$  [33], [34]. The notion of being psd can be extended from matrices to (real-valued) symmetric kernel maps  $k : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$  (with  $k(\mathbf{x}, \mathbf{x}') = k(\mathbf{x}', \mathbf{x})$ ) as follows: For any finite set of feature vectors  $\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}$ , the resulting matrix  $\mathbf{Q} \in \mathbb{R}^{m \times m}$  with entries  $Q_{r,r'} = k(\mathbf{x}^{(r)}, \mathbf{x}^{(r')})$  is psd [78].

See also: symmetric matrix, kernel.

**posterior (prediction)** The study and design of ML methods is often based on a probabilistic model for the data generation process. Within a probabilistic model, we view (the generation of) a data point with features  $\mathbf{x}$  and label  $y$  as an RV with probability distribution  $\mathbb{P}^{(\mathbf{x},y)}$ . It turns out that the optimal prediction for the label  $y$ , given the feature vector  $\mathbf{x}$ , is fully determined by the conditional probability distribution  $\mathbb{P}^{(y|\mathbf{x})}$  of  $y$  given (or conditioned on)  $\mathbf{x}$  [22], [29], [42].

See also: probabilistic model, conditional probability distribution, posterior distribution.

**posterior distribution** In the probabilistic analysis of a ML system, one typically distinguishes between observed data points and unknown parameters of a probabilistic model. The posterior distribution is the conditional probability distribution of the unknown model parameters, given (or conditioned on) the observed data points [22].

See also: conditional probability distribution, prior distribution, probability distribution.

**preimage** Consider a function  $f: \mathcal{U} \rightarrow \mathcal{V}$  between two sets. The preimage  $f^{-1}(\mathcal{B})$  of a subset  $\mathcal{B} \subseteq \mathcal{V}$  is the set of all inputs  $u \in \mathcal{U}$  that are mapped into  $\mathcal{B}$  by  $f$ , i.e.,

$$f^{-1}(\mathcal{B}) := \{u \in \mathcal{U} \mid f(u) \in \mathcal{B}\}.$$

The preimage is well defined even if the function  $f$  is non-invertible [2].

See also: function.

**prior distribution** In the probabilistic analysis of an ML system, one typically distinguishes between observed data points and unknown param-

eters of a probabilistic model. A prior distribution is a postulated probability distribution over the possible choices of these model parameters before (i.e., prior to) observing any data [22]. Given a conditional probability distribution of the data given the model parameters, we update the prior to obtain the posterior distribution of the model parameters.

See also: probability distribution, model parameter, posterior distribution.

**probabilistic model** A probabilistic model for the generation of data points consists of RVs with a joint probability distribution [7]. This joint probability distribution typically involves parameters (or model parameters) that are either chosen manually or learned via statistical inference methods such as maximum likelihood estimation [42].

See also: model, data point, RV, probability distribution, parameter, maximum likelihood, realization.

**probabilistic principal component analysis (PPCA)** PPCA extends basic PCA by using a probabilistic model for data points. Using a probabilistic model allows us to cast dimensionality reduction as an estimation problem that can be solved using EM [79].

See also: PCA, probabilistic model, dimensionality reduction, EM.

**probability density function (pdf)** The pdf  $p^{(x)}(\cdot)$  of a continuous real-valued RV  $x \in \mathbb{R}$  allows us to compute the probability  $\mathbb{P}(x \in \mathcal{B})$  (of the event  $\mathcal{B} \subseteq \mathbb{R}$ ) via a Lebesgue integral as follows [7, Ch. 3]:

$$\mathbb{P}(x \in \mathcal{B}) = \int_{\mathcal{B}} p^{(x)}(\eta) d\eta.$$

This definition extends naturally to a (continuous) vector-valued RV  $\mathbf{x} \in \mathbb{R}^d$ , as the Lebesgue integral is defined for  $\mathbb{R}^d$  with any dimension  $d$ .

See also: RV, probability, vector, probability distribution, measurable.

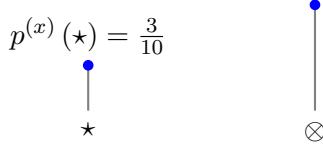
**probability distribution** To analyze ML methods, it can be useful to interpret data points as i.i.d. realizations of an RV. The typical properties of such data points are then governed by the probability distribution of this RV. The probability distribution of a binary RV  $y \in \{0, 1\}$  is fully specified by the probabilities  $\mathbb{P}(y = 0)$  and  $\mathbb{P}(y = 1) = 1 - \mathbb{P}(y = 0)$ . The probability distribution of a real-valued RV  $x \in \mathbb{R}$  might be specified by a pdf  $p(x)$  such that  $\mathbb{P}(x \in [a, b]) \approx p(a)|b-a|$ . In the most general case, a probability distribution is defined by a probability measure [6], [18].

See also: i.i.d., realization, RV, probability, pdf.

**probability mass function (pmf)** The pmf of a discrete RV  $x$  is a function  $p^{(x)}(\cdot) : \mathcal{X} \rightarrow [0, 1]$  that assigns to each possible value  $x' \in \mathcal{X}$  of the RV  $x$  the probability  $p^{(x)}(x') = \mathbb{P}(x' = x)$  [29]. Fig. 53 illustrates the pmf of a discrete RV  $x$ .

$$\mathcal{D}'' = (\otimes, \otimes, \otimes, \star, \otimes, \star, \otimes, \otimes, \star, \otimes)$$

$$\mathcal{D}' = (\otimes, \star, \otimes, \star, \otimes, \otimes, \star, \otimes, \otimes, \otimes)$$



$$\mathcal{D} = (\star, \star, \star, \otimes, \otimes, \otimes, \otimes, \otimes, \otimes, \otimes)$$

Fig. 53. The pmf  $p^{(x)}(\cdot)$  of a discrete RV  $x$  taking values in the set  $\mathcal{X} = \{\star, \otimes\}$ . Three datasets are also shown whose relative frequencies of data points match this pmf exactly. Such datasets could arise as realizations of i.i.d. RVs sharing a common pmf  $p^{(x)}(\cdot)$ .

A pmf always satisfies  $\sum_{x' \in \mathcal{X}} p^{(x)}(x') = 1$ . We can view a pmf as representing a collection of (sufficiently large) datasets. This collection contains any  $\mathcal{D} = \{x^{(1)}, \dots, x^{(m)}\}$ , with the relative frequencies of every value  $x' \in \mathcal{X}$  being close to the corresponding pmf value  $p^{(x)}(x')$ :

$$\frac{|r \in \{1, \dots, m\} : x^{(r)} = x'|}{m} \approx p^{(x)}(x').$$

Note that requiring relative frequencies to be close to the pmf values implies that the empirical entropy of such a dataset is close to the entropy of the pmf  $p^{(x)}(\cdot)$ . Information theory refers to the collection of such datasets as the typical set corresponding to the pmf  $p^{(x)}(\cdot)$  [35]. A main result of information theory states that a dataset generated by i.i.d. sampling from  $p^{(x)}(\cdot)$  belongs, with high probability, to the typical set with respect to  $p^{(x)}(\cdot)$  [35, Th. 3.1.2].

See also: discrete RV, probability, probability distribution, probabilistic model.

**probability simplex** The probability simplex  $\Delta^{k-1}$  is the set of all vectors in  $\mathbb{R}^k$  with nonnegative entries that sum to one [25]. Each element of  $\Delta^{k-1}$  represents a pmf of an RV  $y \in \{1, \dots, k\}$ .

See also: probability, vector, pmf, RV.

**probability space** A probability space is a mathematical structure that allows us to reason about a random experiment, e.g., the observation of a physical phenomenon. Formally, a probability space  $\mathcal{P}$  is a triplet  $(\Omega, \mathcal{F}, \mathbb{P}(\cdot))$  where

- $\Omega$  is a sample space containing all possible outcomes of a random experiment;
- $\mathcal{F}$  is a  $\sigma$ -algebra, i.e., a collection of subsets of  $\Omega$  (called events) that satisfies certain closure properties under set operations;
- $\mathbb{P}(\cdot)$  is a probability distribution, i.e., a function that assigns a probability  $\mathbb{P}(\mathcal{A}) \in [0, 1]$  to each event  $\mathcal{A} \in \mathcal{F}$ . This function must satisfy  $\mathbb{P}(\Omega) = 1$  and  $\mathbb{P}(\bigcup_{i=1}^{\infty} \mathcal{A}_i) = \sum_{i=1}^{\infty} \mathbb{P}(\mathcal{A}_i)$  for any countable sequence of pairwise disjoint events  $\mathcal{A}_1, \mathcal{A}_2, \dots$  in  $\mathcal{F}$ .

Probability spaces provide the foundation of probabilistic models that can be used to study the behavior of ML methods [6], [18], [19].

See also: probability, random experiment, sample space, event, probability distribution, function, probabilistic model, ML.

**projected gradient descent (projected GD)** Consider an ERM-based method that uses a parameterized model with parameter space  $\mathcal{W} \subseteq \mathbb{R}^d$ . Even if the objective function of ERM is smooth, we cannot use basic GD, as it does not take into account constraints on the optimization variable (i.e., the model parameters). Projected GD extends basic GD to address this issue. A single iteration of projected GD consists of first taking a gradient step and then projecting the result back onto the parameter space. See Fig. 54 for a visual illustration.

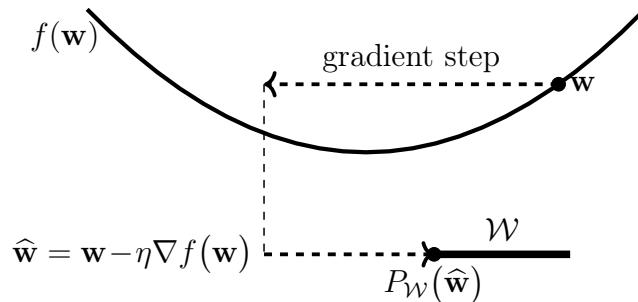


Fig. 54. Projected GD augments a basic gradient step with a projection back onto the constraint set  $\mathcal{W}$ .

See also: ERM, model, parameter space, objective function, smooth, GD, model parameter, gradient step, projection.

**projection** Consider a bounded subset  $\mathcal{W} \subseteq \mathbb{R}^d$  of the  $d$ -dimensional Euclidean space. We define the projection  $P_{\mathcal{W}}(\mathbf{w})$  of a vector  $\mathbf{w} \in \mathbb{R}^d$  onto  $\mathcal{W}$  as

$$P_{\mathcal{W}}(\mathbf{w}) = \arg \min_{\mathbf{w}' \in \mathcal{W}} \|\mathbf{w} - \mathbf{w}'\|_2.$$

In other words,  $P_{\mathcal{W}}(\mathbf{w})$  is the vector in  $\mathcal{W}$  that is closest to  $\mathbf{w}$ . The projection is only well defined for subsets  $\mathcal{W}$  for which the above

minimum exists [25].

See also: Euclidean space, vector, minimum.

**proximable** A convex function for which the proximal operator can be computed efficiently is sometimes referred to as proximable or simple [80].

See also: convex, function, proximal operator.

**proximal operator** Given a convex and continuous function  $f(\mathbf{w}')$ , we define its proximal operator as [24], [48]

$$\text{prox}_f(\mathbf{w}) := \arg \min_{\mathbf{w}' \in \mathbb{R}^d} \left[ f(\mathbf{w}') + (1/2) \|\mathbf{w} - \mathbf{w}'\|_2^2 \right].$$

It is often necessary to evaluate the proximal operator for a scaled version  $\eta f$ , with some factor  $\eta > 0$ ,

$$\text{prox}_{\eta f(\cdot)}(\mathbf{w}) := \arg \min_{\mathbf{w}' \in \mathbb{R}^d} \left[ f(\mathbf{w}') + \frac{1}{2\eta} \|\mathbf{w} - \mathbf{w}'\|_2^2 \right] \quad \text{with } \eta > 0.$$

The factor  $\eta$  controls the relative importance of the function  $f$  and the penalty term in the optimization problem defining the proximal operator. From an algorithm design perspective, the role of the scaling factor  $\eta$  is similar to the learning rate in gradient-based methods. As illustrated in Fig. 55, evaluating the proximal operator amounts to minimizing a penalized variant of  $f$ . The penalty term is the scaled squared Euclidean distance to a given vector  $\mathbf{w}$  (which is the input to the proximal operator). The proximal operator can be interpreted as a generalization of the gradient step, which is defined for a smooth convex function  $f(\mathbf{w}')$ . Indeed, taking a gradient step with step size  $\eta$  at the

current vector  $\mathbf{w}$  is the same as applying the (scaled, by  $\eta$ ) proximal operator of the function  $\tilde{f}(\mathbf{w}') = (\nabla f(\mathbf{w}))^T(\mathbf{w}' - \mathbf{w})$ .

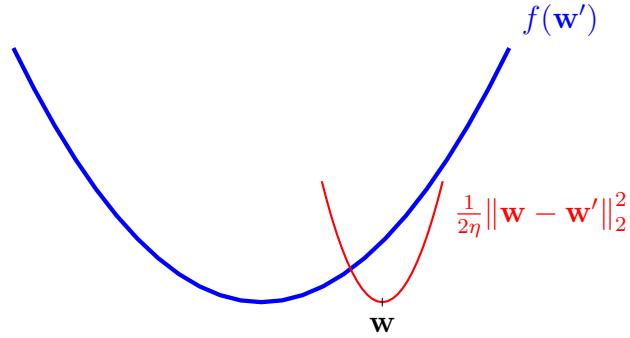


Fig. 55. The proximal operator updates a vector  $\mathbf{w}$  by minimizing a penalized version of the function  $f$ . The penalty term is the scaled squared Euclidean distance between the optimization variable  $\mathbf{w}'$  and the given vector  $\mathbf{w}$ .

See also: convex, function, gradient step.

**pseudocontractive operator** An operator  $\mathcal{F} : \mathcal{H} \rightarrow \mathcal{H}$  on a Hilbert space  $\mathcal{H}$  is called pseudocontractive if [23]

$$\|\mathcal{F}\mathbf{w} - \mathcal{F}\mathbf{w}'\|^2 \leq \|\mathbf{w} - \mathbf{w}'\|^2 + \|(\text{Id} - \mathcal{F})\mathbf{w} - (\text{Id} - \mathcal{F})\mathbf{w}'\|^2$$

holds for all  $\mathbf{w}, \mathbf{w}' \in \mathcal{H}$ , where  $\text{Id}$  denotes the identity operator. This condition is equivalent to the monotonicity of the operator  $\text{Id} - \mathcal{F}$  [23, Example 20.8]. Pseudocontractive operators include non-expansive operators and contractive operators as special cases.

Example: Let  $\mathcal{H} = \mathbb{R}$  and define

$$\mathcal{F}(w) = w - \phi(w),$$

where  $\phi : \mathbb{R} \rightarrow \mathbb{R}$  is any nondecreasing function. Then,  $\text{Id} - \mathcal{F} = \phi$  is monotone, and hence  $\mathcal{F}$  is pseudocontractive [23]. For instance, choosing  $\phi(w) = \max\{0, w\}$  yields a pseudocontractive operator that is neither contractive nor non-expansive.

Alternative meaning [81], [82]: In the literature on asynchronous and parallel optimization, an operator  $\mathcal{F}$  with a fixed point  $\mathbf{w}^*$  is called pseudocontractive if there exists  $\kappa \in (0, 1)$  such that [81], [82]

$$\|\mathcal{F}\mathbf{w} - \mathbf{w}^*\| \leq \kappa \|\mathbf{w} - \mathbf{w}^*\| \quad \text{for all } \mathbf{w} \in \mathcal{H}. \quad (9)$$

This notion expresses contraction with respect to the fixed point and implies linear convergence of the corresponding fixed-point iteration  $\mathbf{w}^{(t+1)} = \mathcal{F}\mathbf{w}^{(t)}$  [82, Proposition 1.2].

Example: Consider the operator  $\mathcal{F} : \mathbb{R} \rightarrow \mathbb{R}$  defined by

$$\mathcal{F}(w) = \begin{cases} \frac{1}{2}w, & w \neq 1, \\ 0.9, & w = 1. \end{cases}$$

The unique fixed point is  $w^* = 0$ , and  $|\mathcal{F}(w) - w^*| \leq 0.9|w - w^*|$  holds for all  $w \in \mathbb{R}$ , so  $\mathcal{F}$  is pseudocontractive in the sense of (9). However,  $\mathcal{F}$  fails to be a non-expansive operator, since arbitrarily small perturbations around  $w = 1$  can produce large changes in  $\mathcal{F}(w)$ .

See also: operator, Hilbert space, fixed point.

**pseudoinverse** The Moore–Penrose pseudoinverse  $\mathbf{A}^+$  of a matrix  $\mathbf{X} \in \mathbb{R}^{m \times d}$  generalizes the notion of an inverse matrix [3]. The pseudoinverse arises naturally in ridge regression for a dataset with feature matrix  $\mathbf{X}$  and

label vector  $\mathbf{y}$  [83, Ch. 3]. The model parameters learned by ridge regression are given by

$$\hat{\mathbf{w}}^{(\alpha)} = (\mathbf{X}^T \mathbf{X} + \alpha \mathbf{I})^{-1} \mathbf{X}^T \mathbf{y}, \quad \alpha > 0.$$

We can then define the pseudoinverse  $\mathbf{X}^+ \in \mathbb{R}^{d \times m}$  via the limit [84, Ch. 3]:

$$\lim_{\alpha \rightarrow 0^+} \hat{\mathbf{w}}^{(\alpha)} = \mathbf{X}^+ \mathbf{y}.$$

See also: matrix, inverse matrix, ridge regression.

**random experiment** A random experiment is a physical (or abstract) process that produces an outcome  $\omega$  from a set  $\Omega$  of possibilities. This set of all possible outcomes is referred to as the sample space of the experiment. The key characteristic of a random experiment is that its outcome is unpredictable (or uncertain). Any measurement or observation of the outcome is an RV, i.e., a function of the outcome  $\omega \in \Omega$ . Probability theory uses a probability space as a mathematical structure for the study of random experiments. A key conceptual property of a random experiment is that it can be repeated under identical conditions. Strictly speaking, repeating a random experiment a given number of  $m$  times defines a new random experiment. The outcomes of this new experiment are length- $m$  sequences of outcomes from the original experiment (see Fig. 56). While the outcome of a single experiment is uncertain, the long-run behavior of the outcomes of repeated experiments tends to become increasingly predictable. This informal claim can be made

precise via fundamental results of probability theory, such as the law of large numbers and the CLT.

new random experiment with  $\Omega' = \Omega \times \dots \times \Omega$

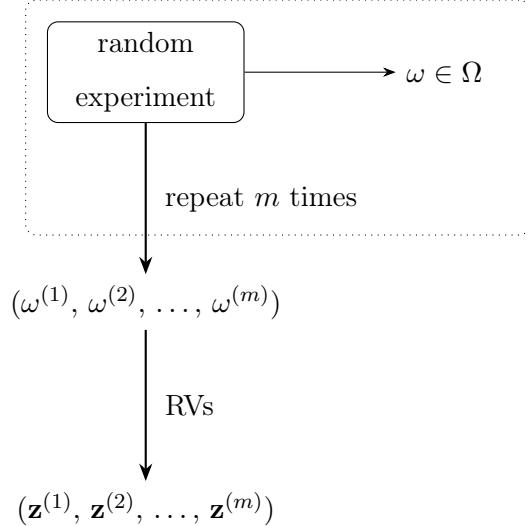


Fig. 56. A random experiment produces an outcome  $\omega \in \Omega$  from a set of possibilities (i.e., a sample space)  $\Omega$ . Repeating the experiment  $m$  times yields another random experiment whose outcomes are sequences  $(\omega^{(1)}, \omega^{(2)}, \dots, \omega^{(m)}) \in \Omega \times \dots \times \Omega$ . One example of a random experiment arising in many ML applications is the gathering of a training set  $\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(m)}$ .

Examples for random experiments arising in ML applications include the following:

- Data collection: The data points collected in ERM-based methods can be interpreted as RVs, i.e., as functions of the outcome  $\omega \in \Omega$

of a random experiment.

- Stochastic gradient descent (SGD) uses a random experiment at each iteration to select a subset of the training set.
- Privacy protection methods use random experiments to perturb the outputs of an ML method to ensure DP.

See also: outcome, sample space, RV, probability, probability space.

**random geometric graph (RGG)** An RGG is a probabilistic model for graphs built from nodes randomly placed in a metric space. Given a metric space, the RGG is characterized by the number of nodes, the connection radius, and the probability distribution describing the node placement. More precisely, for a node set  $i = 1, \dots, n$ , each node  $i$  is assigned to a random position  $\mathbf{x}^{(i)} \in \mathcal{X}$ , typically as realizations of i.i.d. RVs taking values in a set  $\mathcal{X}$ . Together with a metric, this forms a metric space, often with the metric induced by a norm. A specific realization of an RGG contains an edge  $\{i, i'\}$  if and only if the distance between the nodes with respect to the metric is smaller than some threshold, i.e., when  $d(\mathbf{x}^{(i)}, \mathbf{x}^{(i')}) \leq r$  for some threshold  $r > 0$ , as illustrated in Fig. 57.

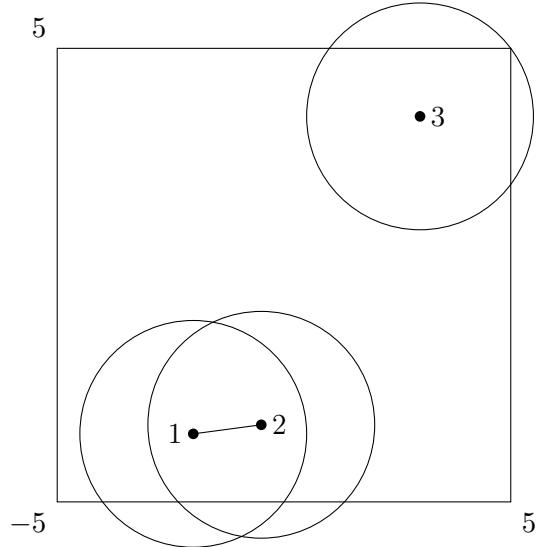


Fig. 57. Illustration of an RGG with  $\mathcal{X} = [-5, 5] \times [-5, 5] \subset \mathbb{R}^2$  and radius  $r = 2.5$  (with respect to the Euclidean distance), where nodes 1 and 2 (corresponding to  $(-2.0, -3.5)^T$  and  $(-0.5, -3.3)^T$ , within radius  $r = 2.5$ ) are connected, while node 3 (corresponding to  $(3.0, 3.5)^T$ ) has no other node within that distance.

See also: graph, stochastic block model (SBM), Erdős–Rényi graph (ER graph).

**random variable (RV)** An RV is a function that maps the outcomes of a random experiment to elements of a measurable space [6], [18]. Mathematically, an RV is a function  $x : \Omega \rightarrow \mathcal{X}$  whose domain is the sample space  $\Omega$  of a probability space and whose co-domain is a measurable space  $\mathcal{X}$ . Different types of RVs include

- binary RVs, which map each outcome to an element of a binary

set (e.g.,  $\{-1, 1\}$  or  $\{\text{cat}, \text{no cat}\}$ );

- discrete RVs, which take on values in a countable set (which can be finite or countably infinite);
- real-valued RVs, which take on values in the real numbers  $\mathbb{R}$ ;
- vector-valued RVs, which map outcomes to the Euclidean space  $\mathbb{R}^d$ .

Probability theory uses the concept of measurable spaces to rigorously define and study the properties of collections of RVs [6].

See also: function, random experiment, sample space, probability space, vector, Euclidean space, probability, measurable.

**rank** The rank of a matrix  $\mathbf{A} \in \mathbb{R}^{m \times d}$ , denoted by  $\text{rank } \mathbf{A}$ , is the maximum number of linearly independent columns of  $\mathbf{A}$  [36]. Equivalently, the rank can be defined as the dimension of the column space  $\text{span}(\mathbf{A}) = \{\mathbf{Aw} \text{ for some } \mathbf{w} \in \mathbb{R}^d\}$ . The rank of a matrix  $\mathbf{A} \in \mathbb{R}^{m \times d}$  can neither exceed the number of rows nor the number of columns of  $\mathbf{A}$  [59], [85], i.e.,  $\text{rank } \mathbf{A} \leq \min\{m, d\}$ .

See also: matrix, dimension, column space, linear map.

**rank-deficient** A matrix  $\mathbf{A} \in \mathbb{R}^{m \times d}$  is rank-deficient if it is not full-rank, i.e., when  $\text{rank } \mathbf{A} < \min\{m, d\}$ .

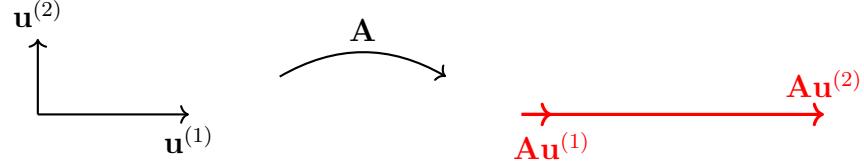


Fig. 58. Example of a rank-deficient matrix  $\mathbf{A} \in \mathbb{R}^{2 \times 2}$ .

In linear regression, the solution of the ERM problem is not unique whenever the feature matrix  $\mathbf{X}$  is such that the matrix  $\mathbf{X}^\top \mathbf{X}$  is rank-deficient.

See also: full-rank, dimension, vector space.

**Rényi divergence** The Rényi divergence measures the (dis)similarity between two probability distributions [86].

See also: probability distribution.

**reproducing kernel Hilbert space (RKHS)** An RKHS  $\mathcal{H}$  is a special type of Hilbert space that consists of functions defined on a set  $\mathcal{X}$  [87]. The defining property of an RKHS is that point evaluations are continuous linear functionals. Each RKHS is associated with a kernel

$$k : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R},$$

satisfying the reproducing property

$$f(\mathbf{x}) = \langle f, k(\mathbf{x}, \cdot) \rangle_{\mathcal{H}} \quad \text{for all } f \in \mathcal{H}, \mathbf{x} \in \mathcal{X}.$$

RKHSs arise naturally in the analysis of kernel methods, including kernel-ridge regression, support vector machine (SVM), and GP regression

[46], [78]. Another application of RKHSs is for the analysis of minimum variance estimation [88].

See also: Hilbert space, kernel.

**sample** In the context of ML, a sample is a finite sequence (of length  $m$ ) of data points  $\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(m)}$ . The number  $m$  is called the sample size. ERM-based methods use a sample to train a model (or learn a hypothesis) by minimizing the average loss (i.e., the empirical risk) over that sample. Since a sample is defined as a sequence, the same data point may appear more than once. By contrast, some authors in statistics define a sample as a set of data points, in which case duplicates are not allowed [89], [90]. These two views (i.e., sequence versus set) can be reconciled by regarding a sample as a sequence of feature–label pairs  $(\mathbf{x}^{(1)}, y^{(1)}), \dots, (\mathbf{x}^{(m)}, y^{(m)})$ . The  $r$ th pair consists of the features  $\mathbf{x}^{(r)}$  and the label  $y^{(r)}$  of an unique underlying data point  $\tilde{\mathbf{z}}^{(r)}$ . While the underlying data points  $\tilde{\mathbf{z}}^{(1)}, \dots, \tilde{\mathbf{z}}^{(m)}$  are unique, some of them can have identical features and labels.

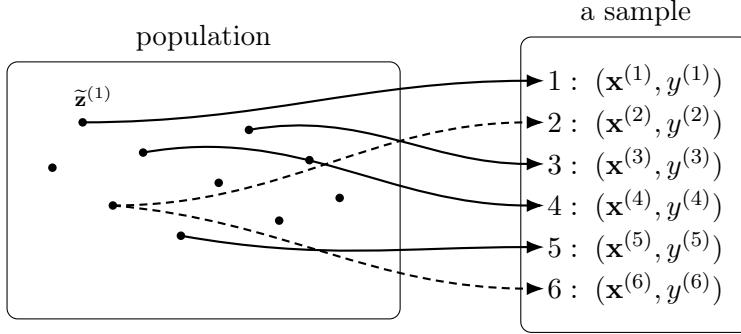


Fig. 59. A sample viewed as a finite sequence. Each element of this sample consists of the feature vector and the label of a data point from an underlying population. The same data point may occur more than once in the sample.

For the analysis of ML methods, it is common to interpret (the generation of) a sample as the realization of a stochastic process indexed by  $\{1, \dots, m\}$ . A widely used assumption is the i.i.d. assumption, where sample elements  $(\mathbf{x}^{(r)}, y^{(r)})$ , for  $r = 1, \dots, m$ , are i.i.d. RVs with a common probability distribution.

See also: sequence, i.i.d. assumption, dataset.

**sample covariance matrix** Consider a dataset consisting of data points characterized by feature vectors  $\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)} \in \mathbb{R}^d$ . The sample covariance matrix of  $\mathcal{D}$  is defined as the covariance matrix with respect to the empirical distribution  $\mathbb{P}^{(\mathcal{D})}$  induced by  $\mathcal{D}$ . It is given explicitly by

$$\widehat{\mathbf{C}} = \frac{1}{m} \sum_{r=1}^m (\mathbf{x}^{(r)} - \widehat{\mathbf{m}})(\mathbf{x}^{(r)} - \widehat{\mathbf{m}})^T.$$

Here, we use the sample mean  $\widehat{\mu}$ .

See also: covariance, matrix, RV.

**Schur decomposition** Every square matrix  $\mathbf{A} \in \mathbb{C}^{d \times d}$  admits a Schur decomposition as follows [3, Th. 7.1.3]:

$$\mathbf{A} = \mathbf{U}\mathbf{T}\mathbf{U}^H.$$

Here,  $\mathbf{U} \in \mathbb{C}^{d \times d}$  is a unitary matrix (i.e.,  $\mathbf{U}^H\mathbf{U} = \mathbf{I}$ ) and  $\mathbf{T}$  is upper triangular with the eigenvalues of  $\mathbf{A}$  on its diagonal. Carefully note that the Schur decomposition exists also for a matrix  $\mathbf{A}$  that is not diagonalizable. The identity

$$(\mathbf{U}^{(1)})^H \mathbf{A} \mathbf{U}^{(1)} = \begin{pmatrix} (\mathbf{x}^{(1)})^H \\ (\mathbf{X}^{(2)})^H \end{pmatrix} \mathbf{A} \begin{pmatrix} \mathbf{x}^{(1)} & \mathbf{X}^{(2)} \end{pmatrix} = \begin{pmatrix} \lambda_1 & \mathbf{b}^H \\ 0 & \mathbf{A}^{(1)} \end{pmatrix}$$

represents the first step in the construction of the Schur decomposition. Every matrix  $\mathbf{A} \in \mathbb{C}^{n \times n}$  has at least one eigenvalue  $\lambda_1$  with a unit-norm eigenvector  $\mathbf{x}^{(1)}$ ,  $\mathbf{A}\mathbf{x}^{(1)} = \lambda_1\mathbf{x}^{(1)}$ . This eigenvector allows us to decompose  $\mathbf{A}$  as shown above. Here, we extend  $\mathbf{x}^{(1)}$  to an orthonormal basis  $\mathbf{Q}^{(1)} = (\mathbf{x}^{(1)} | \mathbf{X}^{(2)})$  and use  $\mathbf{b}^H := (\mathbf{x}^{(1)})^H \mathbf{A} \mathbf{X}^{(2)}$  and  $\mathbf{A}^{(1)} := (\mathbf{X}^{(2)})^H \mathbf{A} \mathbf{X}^{(2)}$ . Applying the same construction recursively to  $\mathbf{A}^{(1)}$  yields the Schur decomposition.

See also: matrix, eigenvalue, EVD.

**sequence** A sequence is an ordered collection of values from a set  $\mathcal{A}$ . For example, a sequence of values from the set  $\mathcal{A} = \{\star, \otimes\}$  could be

$$a = (\star, \otimes, \star, \star, \otimes, \dots).$$

Formally, a sequence  $a$  is a function [2]

$$a : \mathbb{N} \rightarrow \mathcal{A} : r \mapsto a_r.$$

We denote a sequence by  $(a_r)_{r \in \mathbb{N}}$  or  $(a^{(r)})_{r \in \mathbb{N}}$ . Sometimes we also use the notation  $\{a^{(r)}\}_{r \in \mathbb{N}}$ . Note that the same value  $a \in \mathcal{A}$  can appear multiple times in the sequence at different positions  $r$ . Sequences are fundamental for the study of ML methods, for instance when describing successive iterates  $\{\mathbf{w}^{(t)}\}_{t \in \mathbb{N}}$  of an iterative algorithm updating a parameter vector.

We can also use a sequence to represent an infinite dataset

$$\mathcal{D} = \{ (\mathbf{x}^{(1)}, y^{(1)}), (\mathbf{x}^{(2)}, y^{(2)}), \dots \}.$$

See also: function, dataset, convergence, Cauchy sequence.

**$\sigma$ -algebra** Consider a random experiment with a sample space  $\Omega$ . A  $\sigma$ -algebra (or  $\sigma$ -field)  $\Sigma$  is a collection of subsets of  $\Omega$  with the following properties [1], [6], [17]:

- The empty set  $\emptyset$  and the entire sample space  $\Omega$  belong to  $\Sigma$ , i.e.,  $\emptyset \in \Sigma$  and  $\Omega \in \Sigma$ .
- If a set  $\mathcal{A}$  belongs to  $\Sigma$ , then its complement  $\Omega \setminus \mathcal{A}$  also belongs to  $\Sigma$ , i.e.,  $\mathcal{A} \in \Sigma$  implies  $\Omega \setminus \mathcal{A} \in \Sigma$ .
- If a countable collection of sets  $\mathcal{A}_1, \mathcal{A}_2, \dots$  belongs to  $\Sigma$ , then their union also belongs to  $\Sigma$ , i.e.,  $\mathcal{A}_1, \mathcal{A}_2, \dots \in \Sigma$  implies  $\bigcup_{i=1}^{\infty} \mathcal{A}_i \in \Sigma$ .

See also: sample space, RV, probability space.

**$\sigma$ -field** See  $\sigma$ -algebra.

**simple function** A simple function  $f : \mathbb{R}^d \rightarrow \mathbb{R}$  is a measurable function that takes on only finitely many values. In other words,

$$f(\mathbf{x}) = \sum_{c=1}^k \alpha_c \mathbb{I}_{C^{(c)}}(\mathbf{x}),$$

where  $\mathbb{I}_{\mathcal{C}}$  denotes the indicator function of a subset  $\mathcal{C} \subset \mathbb{R}^d$  and  $\alpha_1, \dots, \alpha_k \in \mathbb{R}$  are arbitrary coefficients. The subsets  $\mathcal{C}^{(1)}, \dots, \mathcal{C}^{(k)}$  in the above decomposition must be measurable and must form a partition of  $\mathbb{R}^d$ .

See also: measurable, Lebesgue integral.

**singular value** See singular value decomposition (SVD).

**singular value decomposition (SVD)** The SVD for a matrix  $\mathbf{A} \in \mathbb{R}^{m \times d}$  is a factorization of the following form:

$$\mathbf{A} = \mathbf{V}\Lambda\mathbf{U}^T$$

with orthonormal matrices  $\mathbf{V} = (\mathbf{v}^{(1)}, \dots, \mathbf{v}^{(m)}) \in \mathbb{R}^{m \times m}$  and  $\mathbf{U} = (\mathbf{u}^{(1)}, \dots, \mathbf{u}^{(d)}) \in \mathbb{R}^{d \times d}$  [3] (see Fig. 60).

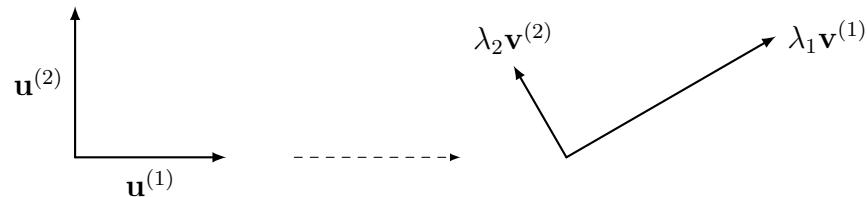


Fig. 60. Orthonormal matrices  $\mathbf{V}$  and  $\mathbf{U}$ .

The matrix  $\Lambda \in \mathbb{R}^{m \times d}$  is only nonzero along the main diagonal, whose entries  $\Lambda_{j,j}$  are nonnegative and referred to as singular values.

See also: matrix.

**smooth** A real-valued function  $f : \mathbb{R}^d \rightarrow \mathbb{R}$  is smooth if it is differentiable and its gradient  $\nabla f(\mathbf{w})$  is continuous at all  $\mathbf{w} \in \mathbb{R}^d$  [52], [64]. A smooth

function  $f$  is referred to as  $\beta$ -smooth if the gradient  $\nabla f(\mathbf{w})$  is Lipschitz continuous with Lipschitz constant  $\beta$ , i.e.,

$$\|\nabla f(\mathbf{w}) - \nabla f(\mathbf{w}')\| \leq \beta \|\mathbf{w} - \mathbf{w}'\| \quad \text{for any } \mathbf{w}, \mathbf{w}' \in \mathbb{R}^d.$$

The constant  $\beta$  quantifies the smoothness of the function  $f$ , i.e., the smaller the  $\beta$ , the smoother  $f$  is. Optimization problems with a smooth objective function can be solved effectively by gradient-based methods. Indeed, gradient-based methods approximate the objective function locally around a current choice  $\mathbf{w}$  using its gradient. This approximation works well if the gradient does not change too rapidly. We can make this informal claim precise by studying the effect of a single gradient step with step size  $\eta = 1/\beta$  (see Fig. 61).

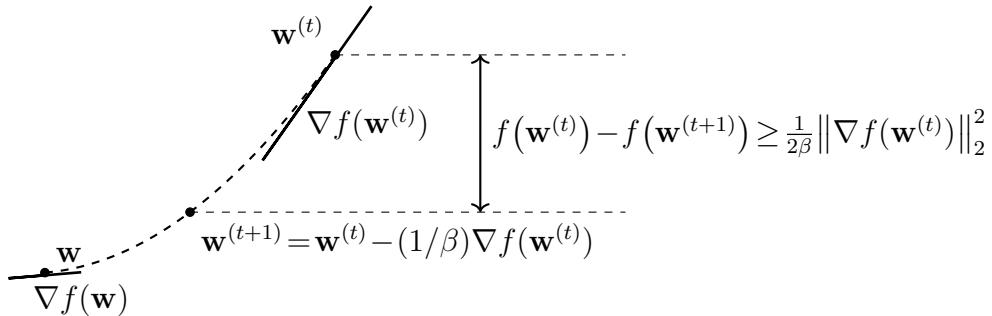


Fig. 61. Consider an objective function  $f(\mathbf{w})$  that is  $\beta$ -smooth. Taking a gradient step, with step size  $\eta = 1/\beta$ , decreases the objective by at least  $1/2\beta \|\nabla f(\mathbf{w}^{(t)})\|_2^2$  [52], [64], [91]. Note that the step size  $\eta = 1/\beta$  becomes larger for smaller  $\beta$ . Thus, for smoother objective functions (i.e., those with smaller  $\beta$ ), we can take larger steps.

See also: function, differentiable, gradient, gradient-based method.

**spectral decomposition** An EVD of a normal matrix is called a spectral decomposition. It has the special property that the eigenvectors of the normal matrix are orthonormal. This also works the other way, i.e., any matrix that has orthonormal eigenvectors is a normal matrix.

See also: EVD, normal matrix, eigenvector.

**spectrum** The spectrum of a linear operator  $\mathcal{F}$  is the set of values  $\lambda$  for which the operator  $\mathcal{F} - \lambda \text{Id}$  fails to be invertible [92]. The notion of a spectrum can then be naturally extended to any kind of object for which an associated linear operator can be defined. For example, the spectrum of a square matrix  $\mathbf{A} \in \mathbb{R}^{d \times d}$  is obtained via the spectrum of the associated linear operator [59]. As another example, we can define the spectrum of a stochastic process via a linear operator constructed from its covariance function [27].

See also: linear operator, matrix, stochastic process.

**standard deviation** The standard deviation of a real-valued RV  $x$  is defined as the square root of its variance, i.e.,  $\sqrt{\mathbb{E}\{(x - \mathbb{E}\{x\})^2\}}$ .

See also: RV, variance, expectation.

**standard normal random vector** A standard normal random vector is an RV  $\mathbf{x} = (x_1, \dots, x_d)^T$  whose entries are i.i.d. Gaussian RVs  $x_j \sim \mathcal{N}(0, 1)$ . The probability distribution of a standard normal random vector is a special case of a multivariate normal distribution  $\mathbf{x} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ . See also: vector, i.i.d., Gaussian RV, multivariate normal distribution, RV.

**state** A state is a mathematical representation of the minimal information

needed to characterize a system at a given time such that, together with the system dynamics, it suffices to predict the system’s future behavior [93], [94].

See also: feature vector, hypothesis, action.

**state space** The state space of a system is constituted by all possible states of the system at any point in time.

See also: feature vector, hypothesis, action.

**stochastic** We refer to a method as stochastic if it involves a random component or is governed by probabilistic laws. ML methods use randomness to reduce computational complexity (e.g., see SGD) or to capture uncertainty in probabilistic models.

See also: SGD, uncertainty, probabilistic model.

**stochastic process** A stochastic process is a collection of RVs defined on a common probability space and indexed by some set  $\mathcal{I}$  [18], [29], [95]. The index set  $\mathcal{I}$  typically represents time or space, allowing us to represent random phenomena that evolve across time or spatial dimensions—for example, sensor noise or financial time series. Stochastic processes are not limited to temporal or spatial settings. For instance, random graphs such as the ER graph or the SBM can also be viewed as stochastic processes. Here, the index set  $\mathcal{I}$  consists of node pairs that index RVs whose values encode the presence or weight of an edge between two nodes. Moreover, stochastic processes naturally arise in the analysis of stochastic algorithms, such as SGD, which construct a sequence of RVs.

See also: RV, SBM, SGD, uncertainty, probabilistic model.

**strictly convex** A real-valued function  $f : \mathcal{U} \rightarrow \mathbb{R}$  with convex domain  $\mathcal{U} \subset \mathbb{R}^d$  is strictly convex if, for any two distinct  $\mathbf{x}, \mathbf{y} \in \mathcal{U}$  and any  $\lambda \in (0, 1)$ , it satisfies [25, Definition 3.1.1]

$$f(\lambda\mathbf{x} + (1 - \lambda)\mathbf{y}) < \lambda f(\mathbf{x}) + (1 - \lambda)f(\mathbf{y}).$$

Furthermore, if  $f$  is differentiable, this condition is equivalent to

$$f(\mathbf{y}) > f(\mathbf{x}) + \nabla f(\mathbf{x})^T(\mathbf{y} - \mathbf{x})$$

for any  $\mathbf{x} \neq \mathbf{y}$ , which implies that  $f(\cdot)$  admits a unique minimizer on any convex subset of its domain. Unlike strongly convex functions, strictly convex functions do not require a uniform quadratic lower bound.

See also: convex, strongly convex.

**strongly convex** A function  $f : \mathbb{R}^d \rightarrow \mathbb{R} \cup \{+\infty\}$  is  $\mu$ -strongly convex (with parameter  $\mu > 0$ ) if [48]

$$f(\lambda\mathbf{x} + (1 - \lambda)\mathbf{y}) \leq \lambda f(\mathbf{x}) + (1 - \lambda)f(\mathbf{y}) - \frac{\mu}{2}\lambda(1 - \lambda)\|\mathbf{x} - \mathbf{y}\|_2^2$$

for all  $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$  and  $\lambda \in [0, 1]$ . Equivalently,  $f$  is  $\mu$ -strongly convex if for every  $\mathbf{g} \in \partial f(\mathbf{x})$ , [64, Sec. 3.4]

$$f(\mathbf{y}) \geq f(\mathbf{x}) + \mathbf{g}^\top(\mathbf{y} - \mathbf{x}) + \frac{\mu}{2}\|\mathbf{y} - \mathbf{x}\|_2^2 \quad \text{for all } \mathbf{x}, \mathbf{y} \in \mathbb{R}^d.$$

If  $f$  is differentiable, this reduces to [91, Sec. B.1.1]

$$f(\mathbf{y}) \geq f(\mathbf{x}) + \nabla f(\mathbf{x})^\top(\mathbf{y} - \mathbf{x}) + \frac{\mu}{2}\|\mathbf{y} - \mathbf{x}\|_2^2,$$

and if  $f$  is twice differentiable,

$$\nabla^2 f(\mathbf{x}) \succeq \mu \mathbf{I}, \quad \text{for all } \mathbf{x} \in \mathbb{R}^d.$$

Strong convexity implies uniqueness of the minimizer of  $f(\mathbf{x})$  and enables linear convergence rates for gradient-based methods.

See also: function, convex, differentiable, strictly convex.

**subgradient** For a real-valued function  $f : \mathbb{R}^d \rightarrow \mathbb{R} : \mathbf{w} \mapsto f(\mathbf{w})$ , a vector  $\mathbf{a}$  such that  $f(\mathbf{w}) \geq f(\mathbf{w}') + (\mathbf{w} - \mathbf{w}')^T \mathbf{a}$  is referred to as a subgradient of  $f$  at  $\mathbf{w}'$  [39], [77].

See also: function, vector.

**subspace** A subset of a vector space  $\mathcal{V}$  is a subspace of  $\mathcal{V}$  if it is also a vector space with respect to the same operations as  $\mathcal{V}$ . Fig. 62 illustrates this for the vector space  $\mathbb{R}^2$  (over the field  $\mathbb{R}$ ).

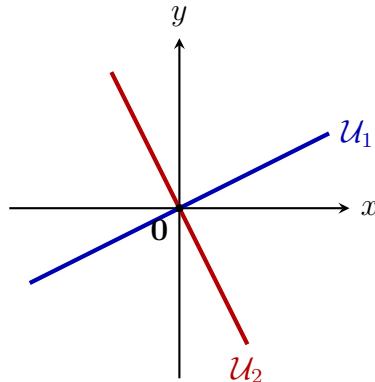


Fig. 62. Any  $\mathbf{x} \in \mathbb{R}^2$  defines a subspace of  $\mathcal{U}_x \subset \mathbb{R}^2$  by  $\mathcal{U}_x = \{\lambda \cdot \mathbf{x} : \lambda \in \mathbb{R}\} \subset \mathbb{R}^2$ . For  $\mathbf{x} \neq \mathbf{0}$ , this corresponds to straight lines through the origin (illustrated by  $\mathcal{U}_1$  and  $\mathcal{U}_2$ ). For the zero vector  $\mathbf{x} = \mathbf{0}$ , this leads to the trivial subspace  $\{\mathbf{0}\}$  just containing the zero vector.

See also: vector space, dimension, basis.

**supporting hyperplane** Let  $\mathcal{C} \subseteq \mathbb{R}^d$  be a non-empty set. A hyperplane

$$\{\mathbf{x} \in \mathbb{R}^d : \langle \mathbf{a}, \mathbf{x} \rangle = b\}, \quad \mathbf{a} \in \mathbb{R}^d \setminus \{\mathbf{0}\}, \quad b \in \mathbb{R}$$

is called a supporting hyperplane of  $\mathcal{C}$  if

$$\mathbf{a}^T \mathbf{x} \leq b \quad \text{for all } \mathbf{x} \in \mathcal{C}$$

and there exists at least one point  $\mathbf{x}^*$  in the boundary of  $\mathcal{C}$  such that  $\mathbf{a}^T \mathbf{x}^* = b$  [25]. The vector  $\mathbf{a}$  is then a normal vector to the supporting hyperplane, pointing toward the halfspace that does not contain the set  $\mathcal{C}$  (see [25, Sec. 2.5.2]).

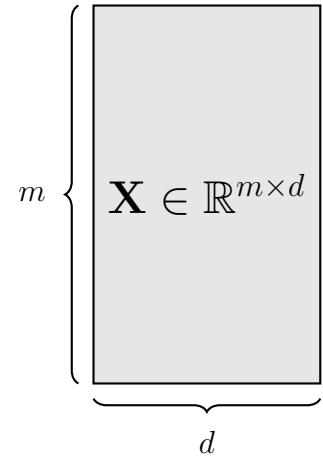
See also: hyperplane, boundary.

**supremum (or least upper bound)** The supremum of a set of real numbers is the smallest number that is greater than or equal to every element in the set. More formally, a real number  $a$  is the supremum of a set  $\mathcal{A} \subseteq \mathbb{R}$  if: 1)  $a$  is an upper bound of  $\mathcal{A}$ ; and 2) no number smaller than  $a$  is an upper bound of  $\mathcal{A}$ . Every non-empty set of real numbers that is bounded above has a supremum, even if it does not contain its supremum as an element [2, Sec. 1.4].

**symmetric matrix** A symmetric matrix is a square matrix  $\mathbf{A}$  with real-valued entries that is equal to its transpose, i.e.,  $\mathbf{A} = \mathbf{A}^T$ . Every symmetric matrix is a normal matrix.

See also: transpose, normal matrix.

**tall matrix** A matrix  $\mathbf{X} \in \mathbb{R}^{m \times d}$  is referred to as tall if it has more rows than columns, i.e., when  $m > d$ .



See also: matrix, wide matrix.

**total variation** See generalized total variation (GTV).

**trace** The trace  $\text{tr}(\mathbf{A})$  of a square matrix  $\mathbf{A} \in \mathbb{R}^{d \times d}$  is the sum of its diagonal entries [36]. Formally, it is the following linear map:

$$\text{tr}(\cdot) : \mathbb{R}^{d \times d} \rightarrow \mathbb{R} : \mathbf{A} \mapsto \sum_{j=1}^d A_{j,j}.$$

It satisfies the cyclic property  $\text{tr}(\mathbf{AB}) = \text{tr}(\mathbf{BA})$  for any matrices  $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{d \times d}$  [34, p. 301].

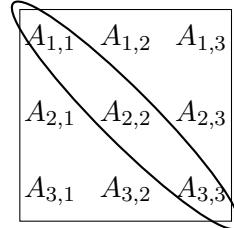


Fig. 63. The trace  $\text{tr}(\mathbf{A})$  of a  $3 \times 3$  matrix  $\mathbf{A} \in \mathbb{R}^{3 \times 3}$  is the sum of three main diagonal entries  $A_{1,1}, A_{2,2}, A_{3,3}$ .

Furthermore, if  $\mathbf{A}$  has eigenvalues  $\lambda_1, \dots, \lambda_d$  (each repeated according to its algebraic multiplicity), then

$$\text{tr}(\mathbf{A}) = \sum_{j=1}^d \lambda_j.$$

This identity follows from the invariance of the trace under similarity transformations [34, Ch. 10].

See also: matrix, eigenvalue.

**transpose** The transpose of a real-valued matrix is obtained by exchanging rows and columns. For a matrix  $\mathbf{A} \in \mathbb{R}^{m \times d}$ , its transpose is denoted by  $\mathbf{A}^T$  and satisfies  $(\mathbf{A}^T)_{j,j'} = \mathbf{A}_{j',j}$ .

See also: matrix, symmetric matrix.

**typical set** See pmf.

**unbiased estimator** An estimator for the parameters of a probabilistic model is unbiased if its expectation equals the true value of the parameters [42], [43].

See also: estimator, parameter, expectation.

**undirected graph** See graph.

**unitary matrix** A square matrix  $\mathbf{U} \in \mathbb{C}^{d \times d}$  is called unitary if its conjugate transpose (or Hermitian transpose)  $\mathbf{U}^H$  is also its inverse matrix, i.e., if

$$\mathbf{U}^H \mathbf{U} = \mathbf{U} \mathbf{U}^H = \mathbf{I}.$$

Equivalently, the columns (and rows) of a unitary matrix form an orthonormal basis of  $\mathbb{C}^d$  with respect to the standard inner product [33],

[34].

See also: matrix.

**variance** The variance of a real-valued RV  $x$  is defined as the expectation  $\mathbb{E}\{(x - \mathbb{E}\{x\})^2\}$  of the squared difference between  $x$  and its expectation  $\mathbb{E}\{x\}$ . We extend this definition to vector-valued RVs  $\mathbf{x}$  as  $\mathbb{E}\{\|\mathbf{x} - \mathbb{E}\{\mathbf{x}\}\|_2^2\} = \text{tr}(\mathbf{C}^{(\mathbf{x})})$ , i.e., the sum of the variances of each entry of  $\mathbf{x}$ , which can be written compactly as the trace of the covariance matrix  $\mathbf{C}^{(\mathbf{x})}$  of  $\mathbf{x}$ .

See also: RV, expectation, vector.

**vector** A vector is an element of a vector space. In the context of ML, a particularly important example of a vector space is the Euclidean space  $\mathbb{R}^d$ , where  $d \in \mathbb{N}$  is the (finite) dimension of the space. A vector  $\mathbf{x} \in \mathbb{R}^d$  can be represented as a list or 1-D array of real numbers, i.e.,  $x_1, \dots, x_d$  with  $x_j \in \mathbb{R}$  for  $j = 1, \dots, d$ . The value  $x_j$  is the  $j$ th entry of the vector  $\mathbf{x}$ . It can also be useful to view a vector  $\mathbf{x} \in \mathbb{R}^d$  as a function that maps each index  $j \in \{1, \dots, d\}$  to a value  $x_j \in \mathbb{R}$ , i.e.,  $\mathbf{x} : j \mapsto x_j$ . This perspective is particularly useful for the study of kernel methods.

See Fig. 64 for the two views of a vector.

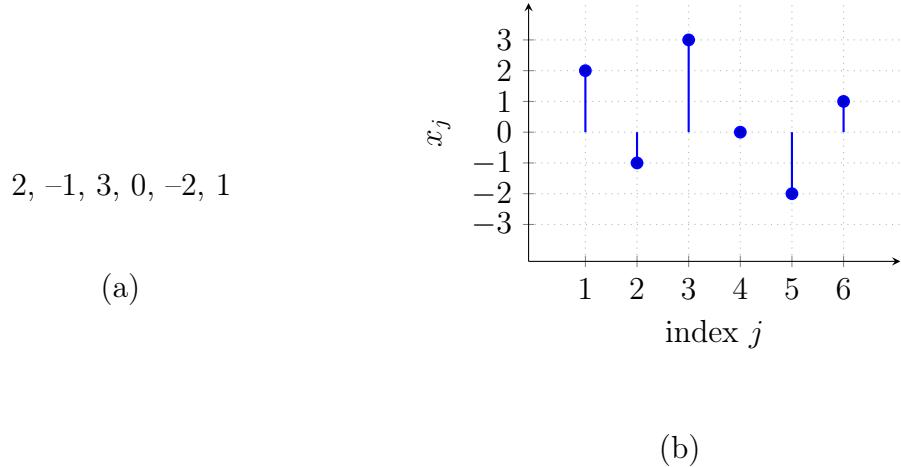


Fig. 64. Two equivalent views of a vector  $\mathbf{x} = (2, -1, 3, 0, -2, 1)^T \in \mathbb{R}^6$ . (a) As a numeric array. (b) As a map  $j \mapsto x_j$ , where  $j \in \{1, \dots, 6\}$ .

See also: vector space, Euclidean space, basis, matrix, linear map.

**vector space** A vector space  $\mathcal{V}$  (also called linear space) is a collection of elements, called vectors, along with the following two operations (see also Fig. 65): 1) addition (denoted by  $\mathbf{v} + \mathbf{w}$ ) of two vectors  $\mathbf{v}, \mathbf{w}$ ; and 2) multiplication (denoted by  $c \cdot \mathbf{v}$ ) of a vector  $\mathbf{v}$  with a scalar  $c$  that belongs to some number field (such as the real numbers  $\mathbb{R}$  or the complex numbers  $\mathbb{C}$ ). The defining property of a vector space is that it is closed under two specific operations. First, if  $\mathbf{v}, \mathbf{w} \in \mathcal{V}$ , then  $\mathbf{v} + \mathbf{w} \in \mathcal{V}$ . Second, if  $\mathbf{v} \in \mathcal{V}$  and  $c \in \mathbb{R}$ , then  $c\mathbf{v} \in \mathcal{V}$ .

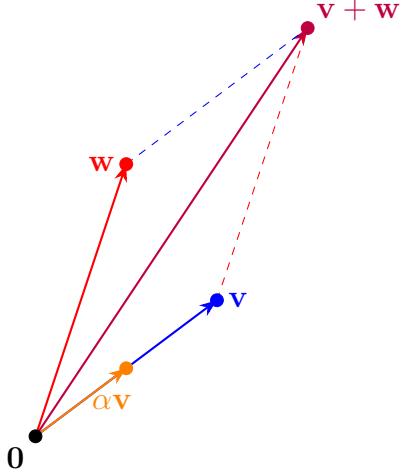


Fig. 65. A vector space  $\mathcal{V}$  is a collection of vectors such that scaling and adding them always yields another vector in  $\mathcal{V}$ .

A common example of a vector space is the Euclidean space  $\mathbb{R}^n$ , which is widely used in ML to represent datasets. We can also use  $\mathbb{R}^n$  to represent, either exactly or approximately, the hypothesis space used by an ML method. Another example of a vector space, which is naturally associated with every probability space  $(\Omega, \mathcal{F}, \mathbb{P}(\cdot))$ , is the collection of all real-valued RVs  $x : \Omega \rightarrow \mathbb{R}$  [1], [13].

See also: vector, Euclidean space, basis, Hilbert space, linear model, linear map.

**weighted graph** A weighted graph is a graph whose edges are assigned numeric weights. Typically, these edge weights are nonnegative real numbers. For example, if a graph represents a road network with nodes corresponding to intersections and edges representing road segments,

the edge weight could represent the capacity (measured in maximum vehicles per hour) of the road segment [37].

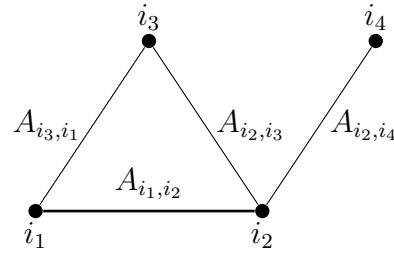


Fig. 66. A weighted graph with four nodes  $\mathcal{V} = \{i_1, i_2, i_3, i_4\}$  and four edges  $\mathcal{E} = \{\{i_1, i_2\}, \{i_2, i_3\}, \{i_3, i_1\}, \{i_2, i_4\}\}$ . Each edge is assigned a weight.

See also: graph.

**wide matrix** A matrix  $\mathbf{X} \in \mathbb{R}^{m \times d}$  is referred to as wide if it has more columns than rows, i.e., when  $d > m$ .

$$m \left\{ \underbrace{\mathbf{X} \in \mathbb{R}^{m \times d}}_{d \quad (d > m)} \right.$$

See also: matrix, tall matrix, Johnson–Lindenstrauss lemma (JL lemma), random projection.

# Machine Learning Concepts

**absolute error loss** Consider a data point with features  $\mathbf{x} \in \mathcal{X}$  and numeric label  $y \in \mathbb{R}$ . As its name suggests, the absolute error loss incurred by a hypothesis  $h : \mathcal{X} \rightarrow \mathbb{R}$  is defined as

$$L((\mathbf{x}, y), h) = |y - h(\mathbf{x})|.$$

Fig. 67 depicts the absolute error loss for a fixed data point with feature vector  $\mathbf{x}$  and label  $y$ . It also indicates the loss values incurred by two different hypotheses  $h'$  and  $h''$ . Similar to the squared error loss, the absolute error loss is also a convex function of the prediction  $\hat{y} = h(\mathbf{x})$ . However, in contrast to the squared error loss, the absolute error loss is non-smooth, as it is not differentiable at the optimal prediction  $\hat{y} = y$ . This property makes ERM-based methods using the absolute error loss computationally more demanding [52], [96]. To build intuition, it is useful to consider the two hypotheses depicted in Fig. 67. Just by inspecting the slope of  $L$  around  $h'(\mathbf{x})$  and  $h''(\mathbf{x})$ , it is impossible to determine whether we are very close to the optimum (at  $h'$ ) or still far away (at  $h''$ ). As a result, any optimization method that is based on local approximations of the loss function (such as subgradient descent) must use a decreasing learning rate to avoid overshooting when approaching the optimum. This required decrease in learning rate tends to slow down the convergence of the optimization method. Besides the increased computational complexity, using absolute error loss in ERM can be beneficial in the presence of outliers in the training set. In contrast to the squared error loss, the slope of the absolute error loss does not

increase with increasing prediction error  $y - h(\mathbf{x})$ . As a result, the effect of introducing an outlier with large prediction error on the solution  $\hat{h}$  of ERM with absolute error loss is much smaller compared with the effect on the solution of ERM with squared error loss.

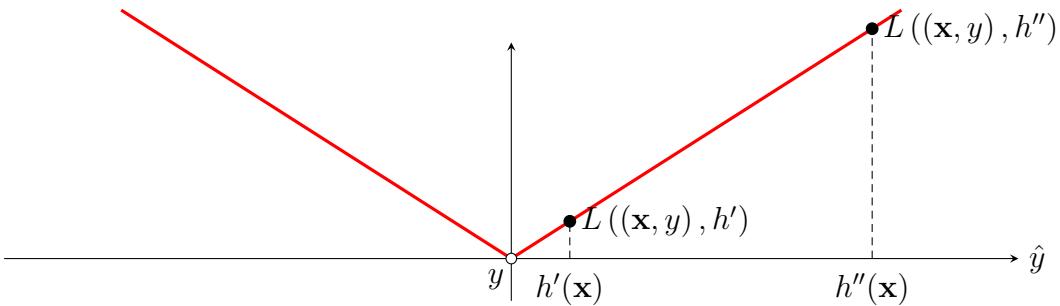


Fig. 67. For a data point with numeric label  $y \in \mathbb{R}$ , the absolute error  $|y - h(\mathbf{x})|$  can be used as a loss function to guide the learning of a hypothesis  $h$ .

See also: data point, feature, label, loss, ERM, subgradient descent, least absolute deviation regression.

**accuracy** Consider data points characterized by features  $\mathbf{x} \in \mathcal{X}$  and a categorical label  $y$  that takes on values from a finite label space  $\mathcal{Y}$ . The accuracy of a hypothesis  $h : \mathcal{X} \rightarrow \mathcal{Y}$ , when applied to the data points in a dataset  $\mathcal{D} = \{(\mathbf{x}^{(1)}, y^{(1)}), \dots, (\mathbf{x}^{(m)}, y^{(m)})\}$ , is then defined as  $1 - (1/m) \sum_{r=1}^m L^{(0/1)}((\mathbf{x}^{(r)}, y^{(r)}), h)$  using the 0/1 loss  $L^{(0/1)}(\cdot, \cdot)$ .

See also: 0/1 loss, loss, metric.

**activation** The output of an artificial neuron within an ANN is referred to as its activation. In particular, the activation is obtained by applying a

(typically nonlinear) activation function to a weighted sum of its inputs.

See also: ANN, deep net.

**activation function** Each artificial neuron within an ANN is assigned an activation function  $\sigma(\cdot)$  that maps a weighted combination of the neuron inputs  $x_1, \dots, x_d$  to a single output value  $a = \sigma(w_1x_1 + \dots + w_dx_d)$ . Note that each neuron is parameterized by the weights  $w_1, \dots, w_d$ . See also: ANN, activation, rectified linear unit (ReLU).

**adaptive boosting (AdaBoost)** AdaBoost is a specific boosting algorithm that combines base learners sequentially [97], [98], [99]. The core idea of AdaBoost is to use the prediction errors of the current base learner for sample weighting in the next base learner. In particular, the  $t$ th base learner learns a hypothesis  $\hat{h}^{(t)}$  by weighted ERM with sample weights  $q^{(r)}$ . The prediction errors of  $\hat{h}^{(t)}$  are then used to update the sample weights by increasing the weights of data points that have been predicted poorly (i.e., with large loss) by  $\hat{h}^{(t)}$ . The updated sample weights are then used in the next base learner to learn  $\hat{h}^{(t+1)}$ . The ultimate hypothesis  $\tilde{h}^{(R)}$  delivered after  $R$  iterations is a linear combination of the hypotheses  $\hat{h}^{(1)}, \dots, \hat{h}^{(R)}$ . AdaBoost can be interpreted as a generalized gradient step [83, Ch. 10.4]

$$\tilde{h}^{(t)} = \tilde{h}^{(t-1)} + \eta^{(t)}\hat{h}^{(t)}.$$

This generalized gradient step involves a learning rate  $\eta^{(t)}$ , which controls the amount of modification of the current hypothesis.

See also: boosting, sample weighting, ERM.

**algorithm** An algorithm is a precise, step-by-step specification for producing

an output from a given input within a finite number of well-defined computational steps [100]. For example, a gradient-based method for linear regression is an algorithm that explicitly describes how to map a given training set into model parameters through a sequence of gradient steps. The precise form of an algorithm depends on the available computational infrastructure. For example, if this infrastructure allows us to compute an inverse matrix, then we can define a linear regression algorithm using the normal equations. In contrast, if the computational infrastructure only allows basic arithmetic operations (i.e., multiplication and addition), the normal equations need to be somehow translated into a sequence of arithmetic operations (e.g., as in gradient-based methods). To study algorithms rigorously, we can represent (or approximate) them by different mathematical structures [101]. One approach is to represent an algorithm as a collection of possible executions. Each individual execution is then a sequence of the following form:

input,  $s_1, s_2, \dots, s_T$ , output.

This sequence starts from an input and progresses via intermediate steps until an output is delivered. Crucially, an algorithm encompasses more than just a mapping from input to output; it also includes intermediate computational steps  $s_1, \dots, s_T$ .

See also: output, linear regression, training set, model parameter, gradient step, model, stochastic.

**application programming interface (API)** An API is a formal mechanism that allows software components to interact in a structured and

modular way [102]. In the context of ML, APIs are commonly used to provide access to a trained ML model. Users—whether humans or machines—can submit the feature vector of a data point and receive a corresponding prediction. Suppose a trained ML model is defined as  $\hat{h}(x) := 2x + 1$ . Through an API, a user can input  $x = 3$  and receive the output  $\hat{h}(3) = 7$  without knowledge of the detailed structure of the ML model or its training. In practice, the model is typically deployed on a server connected to the Internet. Clients send requests containing feature values to the server, which responds with the computed prediction  $\hat{h}(\mathbf{x})$ . APIs promote modularity in ML system design, i.e., one team can develop and train the model, while another team handles integration and user interaction. Publishing a trained model via an API also offers practical advantages. For instance, the server can centralize computational resources that are required to compute predictions. Furthermore, the internal structure of the model remains hidden—which is useful for protecting intellectual property or trade secrets. However, APIs are not without risk. Techniques such as model inversion can potentially reconstruct a model from its predictions using carefully selected feature vectors.

See also: ML, model, feature vector, data point, prediction, feature, model inversion.

**area under the curve (AUC)** The AUC is a quantitative measure of the usefulness of a binary classifier [103]. It is defined (using the natural measure of the Euclidean space  $\mathbb{R}^2$ ) as the area under the receiver operating characteristic (ROC) curve.

See also: classifier, Euclidean space, ROC.

**artificial intelligence (AI)** AI refers to systems that behave rationally, in the sense of maximizing a long-term reward. The ML-based approach to AI is to train a model to predict optimal actions. These predictions are computed from observations about the state of the environment. The choice of loss function sets AI applications apart from more basic ML applications. Artificial intelligence systems (AI systems) rarely have access to a labeled training set that allows the average loss to be measured for any possible choice of model parameters. Instead, AI systems use observed reward signals to estimate the loss incurred by the current choice of model parameters.

See also: ML, AI system, reinforcement learning (RL).

**artificial neural network (ANN)** An ANN is a graphical (signal-flow) representation of a hypothesis that maps features of a data point at its input to a prediction for the corresponding label at its output. The fundamental computational unit of an ANN is the artificial neuron, which applies an activation function  $\sigma(\cdot)$  to the sum of its inputs. The output of a neuron can be used either as the final output of the ANN or as an input to other neurons. A key design aspect of an ANN is its connectivity structure (or architecture), i.e., which neuron outputs are connected to which neuron inputs. As illustrated in Fig. 68, we can represent an ANN as a DAG.

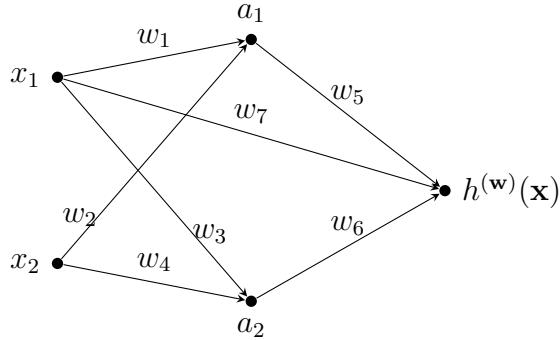


Fig. 68. An ANN can be represented as a weighted DAG with nodes that correspond to neurons or features of a data point. Features can be viewed as trivial neurons without input and with a fixed output given by the feature value. The weighted directed edges indicate how neuron outputs are used as inputs to other neurons. The edge weights are tunable model parameters and are used to scale the inputs to the neurons. The output of some neurons is used as the prediction  $h^{(w)}(\mathbf{x})$ .

One widely used type of ANN is deep nets where neurons form consecutive layers. In a deep net, the outputs of neurons in a given layer are typically only connected to the inputs of the neurons in a consecutive layer. Sometimes it is useful to add shortcut or skip connections that directly connect the outputs of neurons in one layer to the inputs of neurons in a nonconsecutive layer [30], [104].

See also: label, activation function, deep net, layer.

**attack** An attack on an ML system refers to an intentional action—either active or passive—that compromises the system’s integrity, availability, or confidentiality. Active attacks involve perturbing components such as

datasets (via data poisoning) or communication links between devices within an ML application. Passive attacks, such as privacy attacks, aim to infer sensitive attributes without modifying the system. Depending on their goal, we distinguish among denial-of-service attacks, backdoor attacks, and privacy attacks.

See also: data poisoning, privacy attack, sensitive attribute, denial-of-service attack, backdoor.

**attention** Some ML applications involve data points composed of smaller units, referred to as tokens. For example, a sentence consists of words, an image of pixel patches, and a network of nodes. In general, the tokens that constitute a single data point are not independent of one another. Instead, each token of a data point depends on (or pays attention to) specific other tokens. Probabilistic models provide a principled framework for representing and analyzing such dependencies [105]. Attention mechanisms use a more direct approach without explicit reference to a probabilistic model. The idea is to represent the relationship between two tokens  $i$  and  $i'$  using a parameterized function  $f^{(\mathbf{w})}(i, i')$ , where the parameters  $\mathbf{w}$  are learned via a variant of ERM. Practical attention mechanisms differ in their precise choice of attention model  $f^{(\mathbf{w})}(i, i')$  as well as in the precise ERM variant used to learn the parameters  $\mathbf{w}$ . One widely used family of attention mechanisms defines the parameters  $\mathbf{w}$  in terms of two vectors associated with each token  $i$ , i.e., a query vector  $\mathbf{q}^{(i)}$  and a key vector  $\mathbf{k}^{(i')}$ . For a given token  $i$  with query  $\mathbf{q}^{(i)}$  and another token  $i'$  with key  $\mathbf{k}^{(i')}$ , the quantity  $(\mathbf{q}^{(i)})^\top \mathbf{k}^{(i')}$  indicates the extent to which token  $i$  attends to (or depends on) token  $i'$  (see Fig.

69).

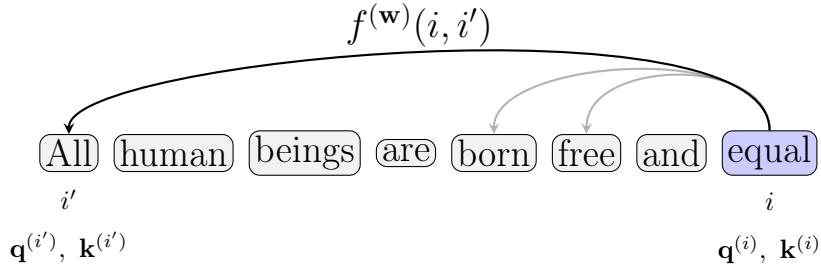


Fig. 69. Attention mechanisms learn a parameterized function  $f^{(\mathbf{w})}(i, i')$  to measure how much token  $i$  attends to token  $i'$ . One widely used construction of  $f^{(\mathbf{w})}(i, i')$  uses query and key vectors, denoted by  $\mathbf{q}^{(i)}$  and  $\mathbf{k}^{(i)}$ , assigned to each token  $i$  [106].

See also: token, function, natural language processing (NLP).

**autoencoder** An autoencoder is an ML method that simultaneously learns an encoder map  $h \in \mathcal{H}$  and a decoder map  $h^* \in \mathcal{H}^*$ . Different autoencoders use different models  $\mathcal{H}, \mathcal{H}^*$ , e.g., ANNs with different architectures. The special case of an autoencoder using (vector-valued) linear models for  $\mathcal{H}, \mathcal{H}^*$  results in PCA.

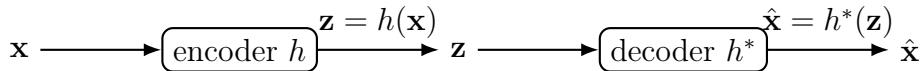


Fig. 70. Autoencoder with an encoder  $h$  mapping  $\mathbf{x} \mapsto \mathbf{z}$  and a decoder  $h^*$  mapping  $\mathbf{z} \mapsto \hat{\mathbf{x}}$ .

The training of the encoder and decoder can be implemented via ERM using a loss that measures the deviation of the reconstructed feature

vector  $h^*(h(\mathbf{x}))$  from the original feature vector  $\mathbf{x}$ .

See also: encoder, feature learning, dimensionality reduction.

**backdoor** A backdoor attack refers to the intentional manipulation of an ML training process. The attacker might perturb the training set (i.e., through data poisoning) or the optimization method used by an ERM-based method. The goal of a backdoor attack is to nudge the learned hypothesis  $\hat{h}$  toward specific predictions for a certain subset  $\mathcal{T} \subset \mathcal{X}$  of the feature space. Any feature vector  $\mathbf{x} \in \mathcal{T}$  serves as a key (or trigger) to unlock a backdoor, in the sense of delivering anomalous predictions. The trigger pattern  $\mathcal{T}$  and corresponding anomalous prediction  $\hat{h}(\mathbf{x})$ , for  $\mathbf{x} \in \mathcal{T}$ , are only known to the attacker.

See also: attack, data poisoning.

**backpropagation** Backpropagation is an algorithm for computing the gradient  $\nabla_{\mathbf{w}} f(\mathbf{w})$  of an objective function  $f(\mathbf{w})$  that depends on the model parameters  $\mathbf{w}$  of an ANN. One example of such an objective function is the average loss incurred by the ANN on a batch of data points. This algorithm is a direct application of the chain rule from calculus to efficiently compute partial derivatives of the loss function with respect to the model parameters. Backpropagation consists of two consecutive phases, also illustrated in Fig. 71. The first phase includes the forward pass, where a batch of data points is fed into the ANN. The ANN processes the input through its layers using its current weights, ultimately producing a prediction at its output. The prediction of the batch is compared to the true label using a loss function, which quantifies

the prediction error. The second phase includes the backward pass (i.e., backpropagation), where the error is backpropagated through the ANN layers. The obtained partial derivatives with respect to the ANN parameters  $w_1, \dots, w_d$  constitute the gradient  $\nabla f(\mathbf{w})$ , which can be used, in turn, to implement a gradient step.

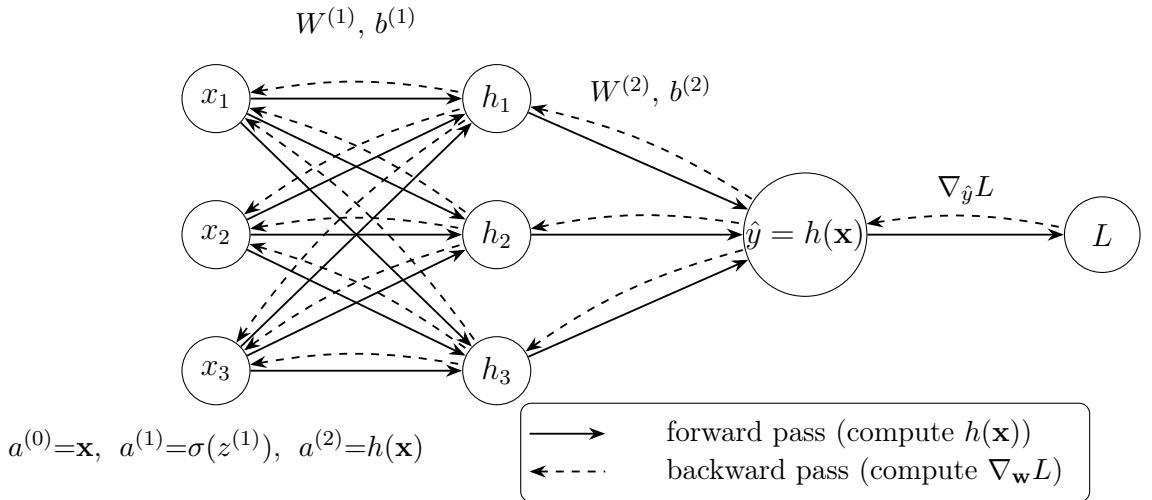


Fig. 71. Solid arrows show the forward pass (i.e., data flow and loss calculation), while dashed arrows show the gradient correction flow during the backward pass for updating the parameters  $W^{(x)}, b^{(x)}$ .

See also: ANN, loss function, GD, optimization method.

**bagging** Bagging is an ensemble technique where base learners use perturbed copies

$$\tilde{\mathcal{D}}^{(1)}, \dots, \tilde{\mathcal{D}}^{(M)}$$

of the original training set  $\mathcal{D}$  [107]. Each base learner delivers a poten-

tially different hypothesis

$$\hat{h}^{(1)}, \dots, \hat{h}^{(M)}.$$

The hypothesis delivered by the overall method is obtained by aggregating the hypotheses  $\hat{h}^{(1)}, \dots, \hat{h}^{(M)}$  using some aggregation rule. For classification methods, the rule is typically a majority vote, while for regression methods, it amounts to averaging.

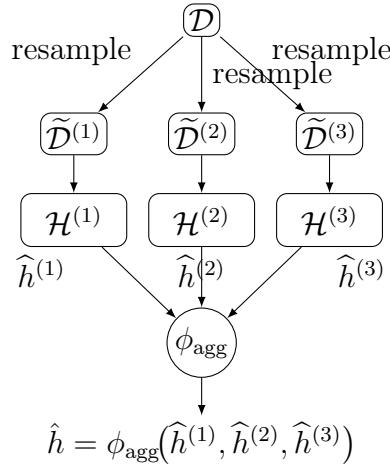


Fig. 72. An example of bagging where three base learners use perturbations  $\tilde{\mathcal{D}}^{(1)}, \dots, \tilde{\mathcal{D}}^{(3)}$  of the original training set  $\mathcal{D}$  to learn the hypotheses  $\hat{h}^{(1)}, \dots, \hat{h}^{(3)}$ . The final hypothesis is obtained by aggregating these individual hypotheses via some aggregation rule  $\phi_{\text{agg}}$ .

See also: ensemble, robustness, bootstrap.

**base learner** A base learner is an ML method that is part of an ensemble method.

See also: ensemble, bagging, stacking, boosting.

**baseline** Consider some ML method that produces a learned hypothesis (or trained model)  $\hat{h} \in \mathcal{H}$ . We evaluate the quality of a trained model by computing the average loss on a test set. But how can we assess whether the resulting test set performance is sufficiently good? How can we determine if the trained model performs close to optimal such that there is little point in investing more resources (for data collection or computation) to improve it? To this end, it is useful to have a reference (or baseline) level against which we can compare the performance of the trained model.

Such a reference value might be obtained from human performance, e.g., the misclassification rate of dermatologists who diagnose cancer from visual inspection of skin [108]. Another source for a baseline is an existing, but for some reason unsuitable, ML method. For example, the existing ML method might be computationally too expensive for the intended ML application. Nevertheless, its test set error can still serve as a baseline. Another, somewhat more principled, approach to constructing a baseline is via a probabilistic model. In many cases, given a probabilistic model  $p(\mathbf{x}, y)$ , we can precisely determine the minimum achievable risk among any hypotheses (not even required to belong to the hypothesis space  $\mathcal{H}$ ) [42].

This minimum achievable risk (referred to as the Bayes risk) is the risk of the Bayes estimator for the label  $y$  of a data point, given its features  $\mathbf{x}$ . Note that, for a given choice of loss function, the Bayes estimator (if it exists) is completely determined by the probability distribution  $\mathbb{P}$  [42, Ch. 4]. However, computing the Bayes estimator and Bayes risk

presents two main challenges. First, the probability distribution  $\mathbb{P}$  is unknown and must be estimated from observed data. Second, even if  $\mathbb{P}$  were known, computing the Bayes risk exactly may be computationally infeasible [109]. A widely used probabilistic model is the multivariate normal distribution  $(\mathbf{x}, y) \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$  for data points characterized by numeric features and labels. Here, for the squared error loss, the Bayes estimator is given by the posterior mean  $\mu_{y|\mathbf{x}}$  of the label  $y$ , given the features  $\mathbf{x}$  [18], [42]. The corresponding Bayes risk is given by the posterior variance  $\sigma_{y|\mathbf{x}}^2$  (see Fig. 73).

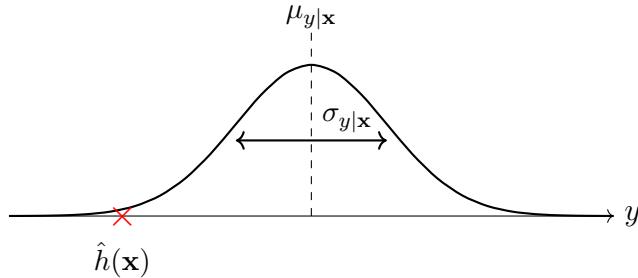


Fig. 73. If the features and the label of a data point are drawn from a multivariate normal distribution, we can achieve the minimum risk (under squared error loss) by using the Bayes estimator  $\mu_{y|\mathbf{x}}$  to predict the label  $y$  of a data point with features  $\mathbf{x}$ . The corresponding minimum risk is given by the posterior variance  $\sigma_{y|\mathbf{x}}^2$ . We can use this quantity as a baseline for the average loss of a trained model  $\hat{h}$ .

See also: Bayes risk, Bayes estimator.

**batch** In the context of SGD, a batch refers to a randomly chosen subset

of the overall training set. We use the data points in this subset to estimate the gradient of training error and, in turn, to update the model parameters.

See also: SGD, training set, data point, gradient, training error, model parameter.

**batch learning** In batch learning (also known as offline learning), the ML model is trained on the entire dataset in a single training iteration, instead of updating it incrementally as data arrive. All available data are inputted into a learning algorithm, resulting in a model that can make predictions. Since these datasets tend to be large, training is computationally expensive and time-consuming, so it is typically performed offline. After learning, the model will be static and will not adapt to new data automatically. Updating the model with new information requires retraining the model entirely. Once the model has been trained, it is launched into production where it cannot be updated. Training a model can take many hours, so many models in production settings are updated cyclically on a periodic schedule when the data distribution is stable. For example, a retail analytics team could retrain their demand forecast model every Sunday using the previous week's sales data to predict next week's demand. If a system needs to be constantly updated to rapidly changing data, such as in stock price prediction, a more adaptable solution such as online learning is necessary.

See also: batch, model, dataset, online learning.

**Bayes estimator** Consider a probabilistic model with a joint probability

distribution  $p(\mathbf{x}, y)$  over the features  $\mathbf{x}$  and the label  $y$  of a data point. For a given loss function  $L(\cdot, \cdot)$ , we refer to a hypothesis  $h$  as a Bayes estimator if its risk  $\mathbb{E}\{L((\mathbf{x}, y), h)\}$  is the minimum achievable risk [42]. Note that whether a hypothesis qualifies as a Bayes estimator depends on the underlying probability distribution and the choice for the loss function  $L(\cdot, \cdot)$ .

See also: probabilistic model, hypothesis, estimator, risk.

**Bayes risk** Consider a probabilistic model for an ML application where each data point is interpreted as an RV. The probabilistic model includes a probability distribution for the features  $\mathbf{x}$  and label  $y$  of a data point. The Bayes risk is the minimum possible risk that can be achieved by any hypothesis  $h : \mathcal{X} \rightarrow \mathcal{Y}$ . Any hypothesis that achieves the Bayes risk is referred to as a Bayes estimator [42].

See also: probabilistic model, risk, Bayes estimator.

**bias** Consider an ERM-based ML method that learns a hypothesis  $\hat{h} \in \mathcal{H}$  from a given training set. The analysis of the ML method is often based on a probabilistic model (such as the i.i.d. assumption) for the data generation. Here, data points and, in turn, the learned hypothesis  $\hat{h}$  are viewed as (realizations of) RVs. Any property  $\theta(\hat{h})$  of  $\hat{h}$ , such as specific model parameters in a parametric model or the prediction error  $y - \hat{h}(\mathbf{x})$  for a fixed data point, then also becomes an RV. The squared bias of a numeric property  $\theta(\hat{h}) \in \mathbb{R}^r$  is [22], [83]

$$B^2 := \|\mathbb{E}\{\theta(\hat{h})\} - \theta(\bar{h})\|_2^2.$$

Here,  $\bar{h}$  is a reference hypothesis, which could be defined by  $\bar{h}(\mathbf{x}) = y$

for a fixed test data point with feature vector  $\mathbf{x}$  and label  $y$ .

See also: probabilistic model, i.i.d., RV, estimation error.

**binary classification** Binary classification refers to classification with two

labels. The labels are usually defined as  $\{-1, 1\}$  or  $\{0, 1\}$ .

See also: classification, label, data point, feature.

**binary cross-entropy (BCE)** The BCE loss is a special case of cross-entropy loss for a binary classification problem.

See also: cross-entropy, classification.

**boosting** Boosting is an iterative optimization method to learn an accurate hypothesis map (or strong learner) by sequentially combining less accurate base learners (referred to as weak learners) [97], [98], [99], [83, Ch. 10]. Boosting can be understood as a generalization of gradient-based methods for ERM using parametric models and smooth loss functions [110]. In particular, starting from an initialization  $\tilde{h}$ , boosting methods construct a sequence of hypotheses  $\tilde{h}^{(t)}$ ,  $t = 1, \dots$ , via a generalized gradient step

$$\tilde{h}^{(t)} = \tilde{h}^{(t-1)} + \eta^{(t)} \hat{h}^{(t)}.$$

Here,  $\eta^{(t)}$  denotes a learning rate and  $\hat{h}^{(t)}$  is provided by the  $t$ th base learner. Comparing the above update with the plain gradient step suggests that we view  $\hat{h}^{(t)}$  as a (negative) generalized gradient. Boosting methods differ in their choice of base learners for computing the generalized gradients  $\hat{h}^{(t)}$ .

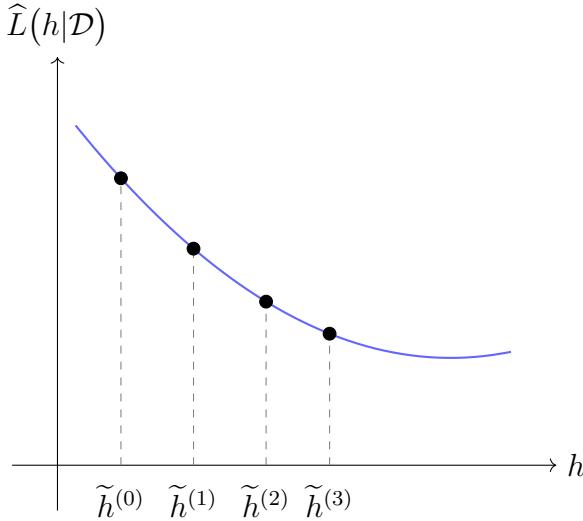


Fig. 74. Boosting methods construct a sequence of hypothesis maps via a generalized gradient step. This generalized gradient step uses the output of base learners.

See also: ensemble, adaptive boosting (AdaBoost), gradient boosting.

**bootstrap** For the analysis of ML methods, it is often useful to interpret a given dataset  $\mathcal{D} = \{\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(m)}\}$  as a collection of (realizations of) i.i.d. RVs with common probability distribution  $\mathbb{P}$ . In practice, the probability distribution  $\mathbb{P}$  is unknown and must be estimated from  $\mathcal{D}$ . The idea of the bootstrap method is to use the empirical distribution  $\mathbb{P}^{(\mathcal{D})}$  of  $\mathcal{D}$  as an estimator for  $\mathbb{P}$  [83]:

$$\frac{1}{m} |r : \mathbf{z}^{(r)} \in \mathcal{A}| \approx \mathbb{P}(\mathcal{A}).$$

Repeatedly sampling from the empirical distribution, which is equivalent to sampling with replacement from  $\mathcal{D}$  [111], results in new datasets

$\mathcal{D}^{(1)}, \dots, \mathcal{D}^{(B)}$ , each containing  $m$  data points. We then use each of those datasets for model training (e.g., via ERM), resulting in the learned hypotheses  $\hat{h}^{(1)}, \dots, \hat{h}^{(B)}$ . We can use these learned hypotheses to estimate important characteristics of an ML method such as bias, variance, or generalization gap [83].

See also: i.i.d., RV, probability distribution, histogram.

**bootstrap aggregation** See bagging.

**Brier score** The Brier score evaluates probabilistic predictions for binary outcomes. Consider  $m$  data points, indexed by  $r = 1, \dots, m$ , each representing a binary outcome  $y^{(r)} \in \{0, 1\}$ . Let  $\hat{y}^{(r)} \in [0, 1]$  denote the predicted probability of success for the  $r$ th data point. The Brier score is defined as the average squared deviation between the predicted probability  $\hat{y}^{(r)}$  and the observed outcome  $y^{(r)}$  [112], i.e.,

$$\frac{1}{m} \sum_{r=1}^m (\hat{y}^{(r)} - y^{(r)})^2.$$

See also: prediction, outcomes, probability.

**classification** Classification is the task of determining a discrete-valued label  $y$  for a given data point, based solely on its features  $\mathbf{x}$ . The label  $y$  belongs to a finite set, such as  $y \in \{-1, 1\}$  or  $y \in \{1, \dots, 19\}$ , and represents the category to which the corresponding data point belongs.

See also: label, data point, feature.

**classifier** A classifier is a hypothesis (i.e., a map)  $h(\mathbf{x})$  used to predict a label taking on values from a finite label space. We might use the function

value  $h(\mathbf{x})$  itself as a prediction  $\hat{y}$  for the label. However, it is customary to use a map  $h(\cdot)$  that delivers a numeric quantity. The prediction is then obtained by a simple thresholding step. For example, in a binary classification problem with a label space  $\mathcal{Y} \in \{-1, 1\}$ , we might use a real-valued hypothesis map  $h(\mathbf{x}) \in \mathbb{R}$  as a classifier. A prediction  $\hat{y}$  can then be obtained via thresholding:

$$\hat{y} = 1 \quad \text{for } h(\mathbf{x}) \geq 0 \quad \text{and} \quad \hat{y} = -1 \quad \text{otherwise.} \quad (10)$$

We can characterize a classifier by its decision regions  $\mathcal{R}_a$  for every possible label value  $a \in \mathcal{Y}$ .

See also: hypothesis, classification, decision region.

**cluster** A cluster is a subset of data points that are more similar to each other than to the data points outside the cluster. The quantitative measure of similarity between data points is a design choice. If data points are characterized by Euclidean feature vectors  $\mathbf{x} \in \mathbb{R}^d$ , we can define the similarity between two data points via the Euclidean distance between their feature vectors. An example of such clusters is shown in Fig. 75.

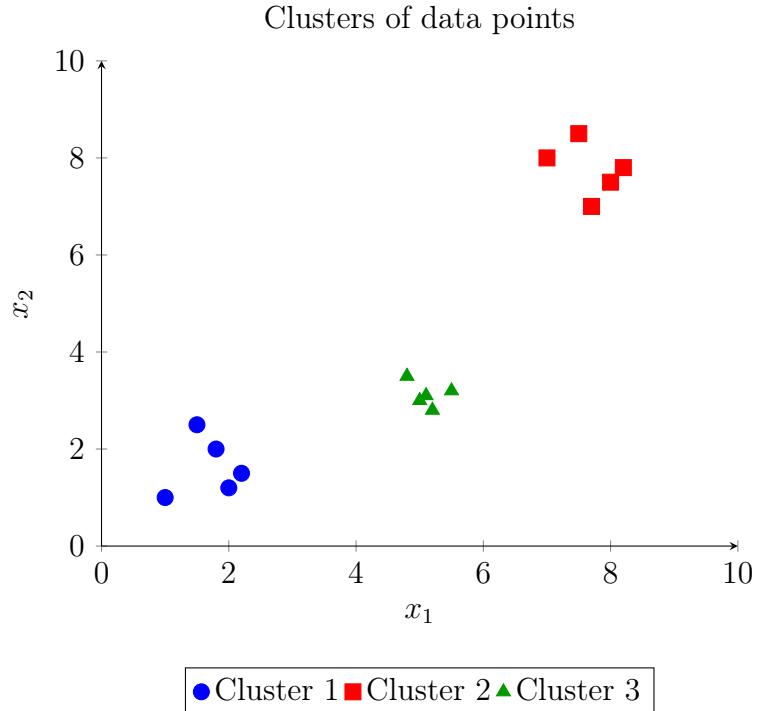


Fig. 75. Illustration of three clusters in a 2-D feature space. Each cluster groups data points that are more similar to each other than to those in other clusters based on the Euclidean distance.

See also: data point, feature vector, Euclidean distance, feature space.

**cluster centroid** Clustering methods decompose a given dataset into a few clusters. Different clustering methods use different representations for these clusters. If data points are characterized by numerical feature vectors  $\mathbf{x} \in \mathbb{R}^d$ , we can use some vector  $\boldsymbol{\mu} \in \mathbb{R}^d$ , referred to as cluster centroid, to represent a cluster. For example, if a cluster consists of a set of data points, we use the average of their feature vectors as a cluster centroid. However, there are also other choices for how to construct

a cluster centroid, e.g., using the geometric median (GM) [113] or via non-Euclidean geometry [114].

See also: clustering, feature vector,  $k$ -means.

**clustered federated learning (CFL)** CFL trains local models for the devices in an FL application by using a clustering assumption, i.e., the devices of an FL network form clusters. Two devices in the same cluster generate local datasets with similar statistical properties. CFL pools the local datasets of devices in the same cluster to obtain a training set for a cluster-specific model. Generalized total variation minimization (GTVMin) clusters devices implicitly by enforcing approximate similarity of model parameters across well-connected nodes of the FL network.

See also: FL, clustering assumption, FL network, cluster, graph clustering.

**clustering** Clustering methods decompose a given set of data points into a few subsets, which are referred to as clusters. Each cluster consists of data points that are more similar to each other than to data points outside the cluster. Different clustering methods use different measures for the similarity between data points and different forms of cluster representations. The clustering method  $k$ -means uses the average feature vector of a cluster (i.e., the cluster mean) as its representative. A popular soft clustering method based on GMM represents a cluster by a multivariate normal distribution.

See also: cluster,  $k$ -means, soft clustering, GMM.

**clustering assumption** The clustering assumption postulates that data

points in a dataset form a (small) number of groups or clusters. Data points in the same cluster are more similar to each other than those outside the cluster [115]. We obtain different clustering methods by using different notions of similarity between data points.

See also: clustering, data point, dataset, cluster.

**clustering error** Consider a clustering method that decomposes a given dataset into clusters. The clustering error is a quantitative measure of the usefulness of the clusters. Different clustering methods use different choices for the clustering error. For example, the hard clustering method  $k$ -means measures the clustering error via the average squared Euclidean distance between the feature vector of a data point and the nearest cluster centroid (see Fig. 76). Another construction for the clustering error can be based on a probabilistic model such as the GMM where the cluster centroids are parameters of the underlying probability distribution.



Fig. 76. For data points with numeric feature vectors, we can use the average squared Euclidean distance to the nearest cluster centroids as a measure of the clustering error.

See also:  $k$ -means, probabilistic model, GMM, maximum likelihood.

**computational aspect** By computational aspects of an ML method, we mainly refer to the computational resources required for its implementation. For example, if an ML method uses iterative optimization techniques to solve ERM, then its computational aspects include: 1) how many arithmetic operations are needed to implement a single iteration (i.e., a gradient step); and 2) how many iterations are needed to obtain useful model parameters. One important example of an iterative optimization technique is GD.

See also: ML, ERM, gradient step, model parameter, GD.

**concept activation vector (CAV)** Consider a deep net, consisting of several hidden layers, trained to predict the label of a data point from its feature vector. One way to explain the behavior of the trained deep net is by using the activations of a hidden layer as a new feature vector  $\mathbf{z}$ . We then probe the geometry of the resulting new feature space by applying the deep net to data points that represent a specific concept  $\mathcal{C}$ . By applying the deep net also to data points that do not belong to this concept, we can train a binary linear classifier  $g(\mathbf{z})$  that distinguishes between concept and non-concept data points based on the activations of the hidden layer. The resulting decision boundary is a hyperplane whose normal vector is the CAV [116] for the concept  $\mathcal{C}$ .

See also: deep net, linear model, trustworthy artificial intelligence (trustworthy AI), interpretability, transparency.

**confusion matrix** Consider a finite dataset with  $m$  data points, each characterized by a feature vector  $\mathbf{x}$  and a label  $y \in \mathcal{Y}$  with a finite label

space  $\mathcal{Y} = \{1, \dots, k\}$ . For a given hypothesis  $h$ , the confusion matrix is a  $k \times k$  matrix where each row corresponds to a specific value of the true label  $y \in \mathcal{Y}$  and each column to a specific value of the prediction  $h(\mathbf{x}) \in \mathcal{Y}$ . The matrix entry in the  $c$ th row and  $c'$ th column is the number of data points with the true label  $y = c$  that are predicted as  $h(\mathbf{x}) = c'$ . The sum of the main diagonal entries is the number of correctly classified data points, i.e., those for which  $y = h(\mathbf{x})$ . Summing the off-diagonal entries results in the total number of data points that are misclassified by  $h$ .

See also: label, label space, hypothesis, matrix, classification.

**convex clustering** Consider a dataset  $\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)} \in \mathbb{R}^d$ . Convex clustering learns vectors  $\mathbf{w}^{(1)}, \dots, \mathbf{w}^{(m)}$  by minimizing:

$$\sum_{r=1}^m \|\mathbf{x}^{(r)} - \mathbf{w}^{(r)}\|_2^2 + \alpha \sum_{i,i' \in \mathcal{V}} \left\| \mathbf{w}^{(i)} - \mathbf{w}^{(i')} \right\|_p.$$

Here,  $\|\mathbf{u}\|_p := (\sum_{j=1}^d |u_j|^p)^{1/p}$  denotes the  $p$ -norm (for  $p \geq 1$ ). It turns out that many of the optimal vectors  $\widehat{\mathbf{w}}^{(1)}, \dots, \widehat{\mathbf{w}}^{(m)}$  coincide. A cluster then consists of those data points  $r \in \{1, \dots, m\}$  with identical  $\widehat{\mathbf{w}}^{(r)}$  [117], [118].

See also: dataset, convex, clustering, vector, norm, cluster, data point.

**coreset** A coresset is a small subset of a larger dataset that approximates certain properties of the original dataset [119]. The construction of a coresset typically involves selecting representative data points and assigning them weights to reflect their importance in the original dataset (Fig. 77).

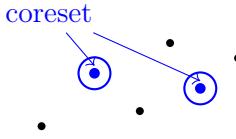


Fig. 77. A coresset (highlighted in blue) is a small subset of a larger dataset.

Coresets are particularly useful for ML applications (such as clustering) involving large datasets, as they allow for efficient computation while preserving the essential characteristics of the data [120].

See also: dataset, data point, clustering.

**covariate** See feature.

**cross-entropy** Consider a multi-class classification problem with a feature space  $\mathcal{X}$  and a finite label space  $\mathcal{Y} = \{1, \dots, k\}$ . A data point with a feature vector  $\mathbf{x}$  is represented as a pmf  $\mathbf{y} = (y_1, \dots, y_k)^T$  over  $\mathcal{Y}$ , where  $y_c$  denotes the probability that a randomly chosen data point with feature vector  $\mathbf{x}$  has label  $c$ . A hypothesis  $h(\mathbf{x})$  outputs a predicted pmf  $\hat{\mathbf{y}} = (\hat{y}_1, \dots, \hat{y}_k)^T$ . The associated cross-entropy loss is [35]

$$L((\mathbf{x}, \hat{\mathbf{y}}), h) := - \sum_{c=1}^k y_c \log \hat{y}_c.$$

The cross-entropy loss quantifies the dissimilarity between the true pmf  $\mathbf{y}$  and the predicted pmf  $\hat{\mathbf{y}}$ . It is also a measure of the expected number of bits required to encode labels drawn from the true pmf  $\mathbf{y}$  when using a coding scheme optimized for the predicted pmf  $\hat{\mathbf{y}}$  [35].

Note that, for binary classification (with  $k = 2$ ), the cross-entropy loss reduces to the logistic loss when employing a parametric model

with model parameters  $\mathbf{w}$  such that  $\hat{y}_2/\hat{y}_1 = \exp(\mathbf{w}^T \mathbf{x})$ . Moreover, the representation (16) of logistic loss requires encoding the label space  $\{1, 2\}$  using the values  $-1$  and  $1$ .

See also: classification, pmf, logistic loss.

**data** In the context of ML, the term data is often used as a synonym for dataset [89], [90]. The ISO/IEC 2382:2015 standard [121] defines data as a

reinterpretable representation of information in a formalized manner suitable for communication, interpretation, or processing.

See also: dataset, data point, sample.

**data augmentation** Data augmentation methods add synthetic data points to an existing set of data points. These synthetic data points are obtained by perturbations (e.g., adding noise to physical measurements) or transformations (e.g., rotations of images) of the original data points. These perturbations and transformations are such that the resulting synthetic data points should still have the same label. As a case in point, a rotated cat image is still a cat image even if their feature vectors (obtained by stacking pixel color intensities) are very different (see Fig. 78). Data augmentation can be an efficient form of regularization.

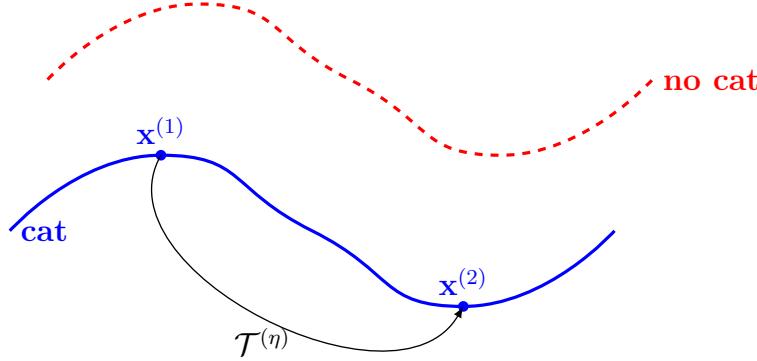


Fig. 78. Data augmentation exploits intrinsic symmetries of data points in some feature space  $\mathcal{X}$ . We can represent a symmetry by an operator  $\mathcal{T}^{(\eta)} : \mathcal{X} \rightarrow \mathcal{X}$ , parameterized by some number  $\eta \in \mathbb{R}$ . For example,  $\mathcal{T}^{(\eta)}$  might represent the effect of rotating a cat image by  $\eta$  degrees. A data point with feature vector  $\mathbf{x}^{(2)} = \mathcal{T}^{(\eta)}(\mathbf{x}^{(1)})$  must have the same label  $y^{(2)} = y^{(1)}$  as a data point with feature vector  $\mathbf{x}^{(1)}$ .

See also: data, data point, label, feature vector, stacking, regularization, feature space.

**data imputation** See missing data.

**data matrix** See tabular data.

**data minimization principle** European data protection regulation includes a data minimization principle. This principle requires a data controller to limit the collection of personal information to what is directly relevant and necessary to accomplish a specified purpose. The data should be retained only for as long as necessary to fulfill that purpose [122, Article

5(1)(c)], [123].

See also: data.

**data normalization** Data normalization refers to transformations applied to the feature vectors of data points to improve the ML method's statistical aspects or computational aspects. For example, in linear regression with gradient-based methods using a fixed learning rate, convergence depends on controlling the norm of feature vectors in the training set. A common approach is to normalize feature vectors such that their norm does not exceed one [8, Ch. 5].

See also: gradient-based method, learning rate, feature map.

**data parallelism** Data parallelism refers to a widely used class of distributed optimization methods for training an ML model. Here, each participating device stores a full replica of the model parameters but processes a disjoint subset of the global dataset [74]. Compared to model parallelism, which distributes the model parameters across devices, data parallelism distributes the computational workload associated with large datasets. It is especially effective when the model fits into the memory of a single device, but the dataset or training complexity would be prohibitive without parallel processing.

See also: model parallelism, horizontal federated learning (HFL).

**data point** A data point is any object that conveys information [35]. Examples include students, radio signals, trees, images, RVs, real numbers, or proteins. We describe data points of the same type by two categories of properties. The first category includes features that are measurable

or computable properties of a data point. These attributes can be automatically extracted or computed using sensors, computers, or other data collection systems. For a data point that represents a patient, one feature could be the body weight. The second category includes labels that are higher level facts (or quantities of interest)—that is, facts that typically require human expertise or domain knowledge to determine rather than being directly measurable—associated with the data point. For a data point that represents a patient, a cancer diagnosis provided by a physician would serve as the label. Fig. 79 depicts an image as an example of a data point along with its features and labels. Importantly, what constitutes a feature or a label is not inherent to the data point itself—it is a design choice that depends on the specific ML application.



A single data point.

Features:

- $x_1, \dots, x_d$ : Color intensities of all image pixels.
- $x_{d+1}$ : Time-stamp of the image capture.
- $x_{d+2}$ : Spatial location of the image capture.

Labels:

- $y_1$ : Number of cows depicted.
- $y_2$ : Number of wolves depicted.
- $y_3$ : Condition of the pasture (e.g., healthy, overgrazed).

Fig. 79. Illustration of a data point consisting of an image. We can use different properties of the image as features and higher level facts about the image as labels.

The distinction between features and labels is not always clear-cut. A property that is considered a label in one setting (e.g., a cancer diagnosis) may be treated as a feature in another setting—particularly if reliable automation (e.g., via image analysis) allows it to be computed without human intervention. ML broadly aims to predict the label of a data point based on its features.

See also: data, feature, label, dataset.

**data poisoning** Data poisoning refers to the intentional manipulation (or fabrication) of data points to maliciously steer the training of an ML model [124], [125]. Data poisoning attacks take various forms, including backdoor and denial-of-service attacks. A backdoor attack implants triggers into training data, so that the trained model behaves normally for typical data points but misclassifies a data point with a feature vector that contains a trigger pattern. A denial-of-service attack degrades the trained model’s overall performance by injecting mislabeled or adversarial examples to prevent effective learning. Data poisoning is particularly harmful in decentralized or distributed ML settings (such as FL), where training data cannot be centrally verified.

See also: attack, backdoor, denial-of-service attack, trustworthy AI.

**dataset** A dataset is a set of distinct data points. Strictly speaking, a dataset is an unordered collection of data points that does not contain any repetitions. However, in ML literature, the term dataset is often used as a synonym for sample, i.e., a sequence (or finite list) of data points that may contain repetitions. ML methods use datasets for

model training and validation. The notion of a dataset is broad, i.e., data points may represent concrete physical entities (such as humans or animals) or abstract objects (such as numbers). For illustration, Fig. 80 depicts a dataset whose data points are cows.



Fig. 80. A cow herd somewhere in the Alps.

Quite often, an ML engineer does not have direct access to the underlying dataset. For instance, accessing the dataset in Fig. 80 would require visiting the cow herd. In practice, we work with a more convenient representation (or approximation) of the dataset. Various mathematical models have been developed for this purpose [126], [127], [128], [129]. One of the most widely used is the relational model, which organizes data as a table (or relation) [126], [130]. A table consists of rows and columns, where each row corresponds to a single data point, and each column represents a specific attribute of a data point. ML methods typically interpret these attributes as features or as a label of a data point. As an illustration, Table I shows a relational representation of the dataset from Fig. 80. In the relational model, the order of rows is immaterial, and each attribute (i.e., column) is associated with a domain that specifies the set of admissible values. In ML applications,

these attribute domains correspond to the feature space and the label space.

TABLE I  
A RELATION (OR TABLE) THAT REPRESENTS THE DATASET IN FIG. 80

Name	Weight	Age	Height	Stomach temperature
Zenzi	100	4	100	25
Berta	140	3	130	23
Resi	120	4	120	31

While the relational model is useful for the study of many ML applications, it may be insufficient regarding the requirements for trustworthy AI. Modern approaches like datasheets for datasets provide more comprehensive documentation, including details about the data collection process, intended use, and other contextual information [131].

See also: data point, sample, data, feature, feature space, label space.

**decision boundary** Consider a hypothesis map  $h$  that reads in a feature vector  $\mathbf{x} \in \mathbb{R}^d$  and delivers a value from a finite set  $\mathcal{Y}$ . The decision boundary of  $h$  is the set of vectors  $\mathbf{x} \in \mathbb{R}^d$  that lie between different decision regions. More precisely, a vector  $\mathbf{x}$  belongs to the decision boundary if and only if each neighborhood  $\{\mathbf{x}' : \|\mathbf{x} - \mathbf{x}'\| \leq \varepsilon\}$ , for any  $\varepsilon > 0$ , contains at least two vectors with different function values.

See also: hypothesis, map, feature vector, boundary, vector, decision region, neighborhood, function.

**decision region** Consider a hypothesis map  $h$  that delivers values from a finite set  $\mathcal{Y}$ . For each label value (i.e., category)  $a \in \mathcal{Y}$ , the hypothesis  $h$  determines a subset of feature values  $\mathbf{x} \in \mathcal{X}$  that result in the same output  $h(\mathbf{x}) = a$ . We refer to this subset as a decision region of the hypothesis  $h$ .

See also: hypothesis, map, label, feature.

**decision tree** A decision tree is a flowchart-like representation of a hypothesis map  $h$ . More formally, a decision tree is a DAG containing a root node that reads in the feature vector  $\mathbf{x}$  of a data point. The root node then forwards the data point to one of its child nodes based on some elementary test on the features  $\mathbf{x}$ . If the receiving child node is not a leaf node, i.e., it has child nodes itself, it represents another test. Based on the test result, the data point is forwarded to one of its descendants. This testing and forwarding of the data point is continued until the data point ends up in a leaf node without any children. See Fig. 81 for visual illustrations.

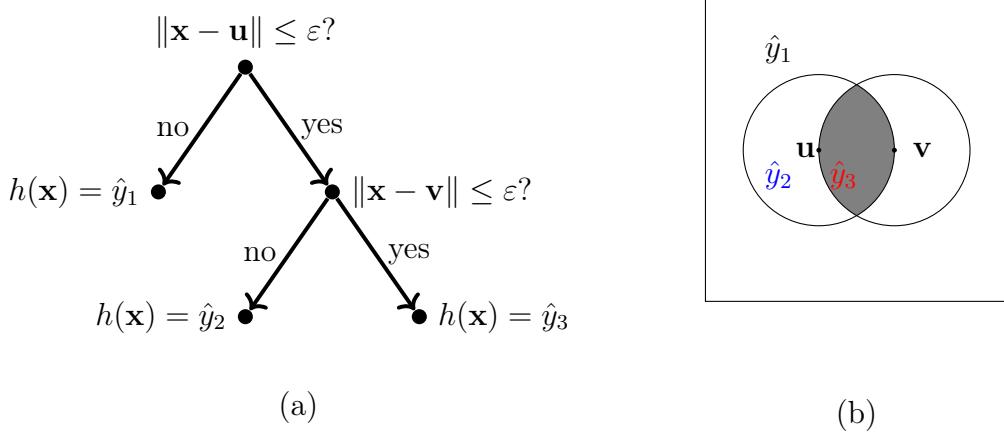


Fig. 81. (a) A decision tree is a flowchart-like representation of a piecewise constant hypothesis  $h : \mathcal{X} \rightarrow \mathbb{R}$ . Each piece is a decision region  $\mathcal{R}_{\hat{y}} := \{\mathbf{x} \in \mathcal{X} : h(\mathbf{x}) = \hat{y}\}$ . The depicted decision tree can be applied to numeric feature vectors, i.e.,  $\mathcal{X} \subseteq \mathbb{R}^d$ . It is parameterized by the threshold  $\varepsilon > 0$  and the vectors  $\mathbf{u}, \mathbf{v} \in \mathbb{R}^d$ . (b) A decision tree partitions the feature space  $\mathcal{X}$  into decision regions. Each decision region  $\mathcal{R}_{\hat{y}} \subseteq \mathcal{X}$  corresponds to a specific leaf node in the decision tree.

See also: decision region.

**deep learning** See deep net.

**deep net** A deep net is an ANN with a (relatively) large number of hidden layers. Deep learning is an umbrella term for ML methods that use a deep net as their model [30].

See also: ANN, layer, deep learning, model, large language model (LLM).

**degree of belonging** Degree of belonging is a number that indicates the

extent to which a data point belongs to a cluster [8, Ch. 8]. The degree of belonging can be interpreted as a soft cluster assignment. Soft clustering methods can encode the degree of belonging with a real number in the interval  $[0, 1]$ . Hard clustering is obtained as the extreme case when the degree of belonging only takes on values 0 or 1.

See also: data point, cluster, soft clustering, hard clustering.

**denial-of-service attack** A denial-of-service attack aims (e.g., via data poisoning) to steer the training of a model such that it performs poorly for typical data points.

See also: attack, data poisoning, model, data point.

### **density-based spatial clustering of applications with noise (DBSCAN)**

DBSCAN refers to a clustering algorithm for data points that are characterized by numeric feature vectors. Like  $k$ -means and soft clustering via GMM, DBSCAN also uses the Euclidean distances between feature vectors to determine the clusters. However, in contrast to  $k$ -means and GMM, DBSCAN uses a different notion of similarity between data points. DBSCAN considers two data points as similar if they are connected via a sequence (i.e., path) of nearby intermediate data points. Thus, DBSCAN might consider two data points as similar (and therefore belonging to the same cluster) even if their feature vectors have a large Euclidean distance.

See also: clustering,  $k$ -means, GMM, cluster, graph.

**design matrix** The term design matrix is a synonym for the feature matrix, particularly used in statistics [90], [132]. It collects the feature vectors of

the data points in a dataset that is used for model training or validation.

See also: feature matrix, feature vectors, data point, dataset.

**device** A physical system that can store and process data. In the context of ML, the term typically refers to a computer capable of reading data points from different sources and using them to train an ML model [133].  
See also: data, ML, data point, model.

**diagnosis** Consider an ERM-based method that resulted in a trained model (or learned hypothesis)  $\hat{h} \in \mathcal{H}$ . We can diagnose the method by comparing the training error  $E_t$  with the validation error  $E_v$  incurred by  $\hat{h}$  on the training set and the validation set.

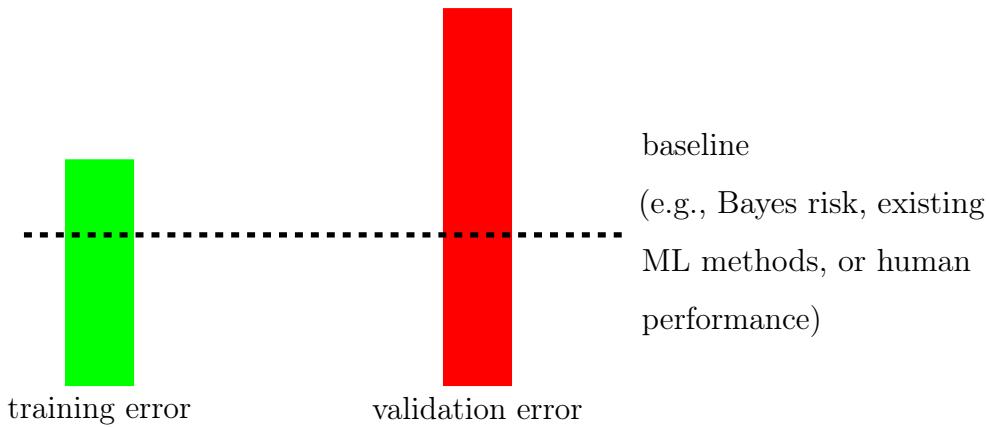


Fig. 82. We can diagnose an ERM-based ML method by comparing its training error with its validation error. Ideally, both are on the same level as a baseline.

See also: baseline, validation,  $k$ -fold cross-validation ( $k$ -fold CV), gener-

alization.

**differential privacy (DP)** Consider some ML method  $\mathcal{A}$  that reads in a dataset (e.g., the training set used for ERM) and delivers some output  $\mathcal{A}(\mathcal{D})$ . The output could be either the learned model parameters or the predictions for specific data points. DP is a precise measure of privacy leakage incurred by revealing the output. Roughly speaking, an ML method is differentially private if the probability distribution of the output  $\mathcal{A}(\mathcal{D})$  remains largely unchanged when the sensitive attribute of one data point in the training set is changed. Note that DP builds on a probabilistic model for an ML method, i.e., we interpret its output  $\mathcal{A}(\mathcal{D})$  as the realization of an RV. The randomness in the output can be ensured by intentionally adding the realization of an auxiliary RV (i.e., adding noise) to the output of the ML method.

See also: output, privacy leakage, sensitive attribute, privacy attack, privacy funnel.

**diffusion method** A diffusion method is a generative ML method that trains a model to approximate the reversal of a gradual stochastic corruption process [134], [135]. Diffusion methods train models using a combination of a forward (diffusion) process and a learned reverse map. In the forward process, random noise is incrementally added to clean representations of a data point, such as an image, an audio recording, or other types of data. Similar to a denoising autoencoder, the trained model can be applied to a noisy representation of a data point. The resulting prediction is a denoised representation. What

sets diffusion methods apart from generic denoising autoencoders is the gradual nature of the corruption process used for model training.

See also: denoising autoencoder, training.

**dimensionality reduction** Dimensionality reduction refers to methods that learn a transformation  $h : \mathbb{R}^d \rightarrow \mathbb{R}^{d'}$  of a (typically large) set of raw features  $x_1, \dots, x_d$  into a smaller set of informative features  $z_1, \dots, z_{d'}$ .

Using a smaller set of features is beneficial in several ways:

- Statistical benefit: It typically reduces the risk of overfitting, as reducing the number of features often reduces the effective dimension of a model.
- Computational benefit: Using fewer features means less computation for the training of ML models. As a case in point, linear regression methods need to invert a matrix whose size is determined by the number of features.
- Visualization: Dimensionality reduction is also instrumental for data visualization. For example, we can learn a transformation that delivers two features  $z_1, z_2$ , which we can use, in turn, as the coordinates of a scatterplot. Fig. 83 depicts the scatterplot of handwritten digits that are placed using transformed features. Here, the data points are naturally represented by a large number of greyscale values (one value for each pixel).

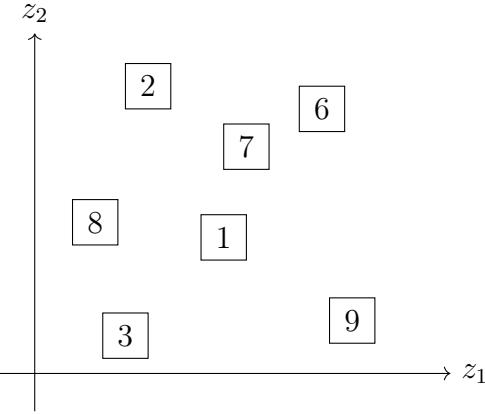


Fig. 83. Example of dimensionality reduction: High-dimensional image data (e.g., high-resolution images of handwritten digits) embedded into 2-D using learned features  $(z_1, z_2)$  and visualized in a scatterplot.

See also: overfitting, autoencoder, random projection, PCA, JL lemma.

**discrepancy** Consider an FL application with networked data represented by an FL network. FL methods use a discrepancy measure to compare hypothesis maps from local models at nodes  $i, i'$ , connected by an edge in the FL network.

See also: FL, FL network, local model.

**distributed algorithm** A distributed algorithm is an algorithm designed for a special type of computer, i.e., a collection of interconnected computing devices (or nodes). These devices communicate and coordinate their local computations by exchanging messages over a network [136], [82]. Unlike a classical algorithm, which is implemented on a single device, a distributed algorithm is executed concurrently on multiple devices with computational capabilities. Similar to a classical algorithm, a

distributed algorithm can be modeled as a set of potential executions. However, each execution in the distributed setting involves both local computations and message-passing events. A generic execution might look as follows:

$$\begin{aligned} \text{Node 1: } & \text{input}_1, s_1^{(1)}, s_2^{(1)}, \dots, s_{T_1}^{(1)}, \text{output}_1; \\ \text{Node 2: } & \text{input}_2, s_1^{(2)}, s_2^{(2)}, \dots, s_{T_2}^{(2)}, \text{output}_2; \\ & \vdots \\ \text{Node N: } & \text{input}_N, s_1^{(N)}, s_2^{(N)}, \dots, s_{T_N}^{(N)}, \text{output}_N. \end{aligned}$$

Each device  $i$  starts from its own local input and performs a sequence of intermediate computations  $s_t^{(i)}$  at discrete-time instants  $t = 1, \dots, T_i$ . These computations may depend on both the previous local computations at the device and the messages received from other devices. One important application of distributed algorithms is in FL where a network of devices collaboratively trains a personal model for each device.

See also: algorithm, device, event, FL, model.

**early stopping** Consider an ERM-based method that uses an iterative optimization method (such as GD) to learn model parameters by minimizing the empirical risk on a given training set. Early stopping refers to terminating the iterations even if they still substantially decrease the empirical risk on the training set. Instead of monitoring the objective function (which is the empirical risk on the training set), early stopping monitors the validation error incurred by the model parameters in each iteration. Early stopping can be interpreted as an implementation

of regularization via model pruning. Indeed, terminating an iterative optimization method after a small number of iterations restricts the set of model parameters that can be reached from the initialization (see Fig. 84).

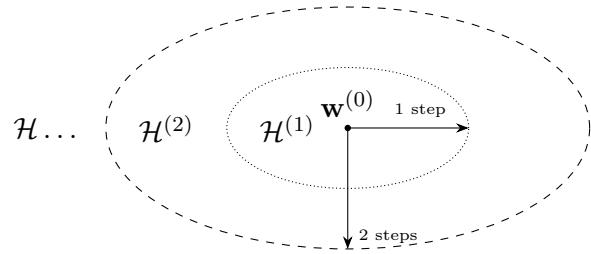


Fig. 84. A gradient-based method for ERM using a hypothesis space  $\mathcal{H}$  defines a nested sequence of effective hypothesis spaces  $\mathcal{H}^{(1)} \subseteq \mathcal{H}^{(2)} \subseteq \dots \subseteq \mathcal{H}$ . The effective hypothesis space  $\mathcal{H}^{(t)}$  is determined by all model parameters that can be reached from the initialization  $\mathbf{w}^{(0)}$  within  $t$  gradient steps.

See also: regularization, gradient-based method, overfitting.

**edge weight** Each edge  $\{i, i'\}$  of an FL network is assigned a nonnegative edge weight  $A_{i,i'} \geq 0$ . A zero edge weight  $A_{i,i'} = 0$  indicates the absence of an edge between nodes  $i, i' \in \mathcal{V}$ .

See also: FL network.

**effective dimension** The effective dimension  $d_{\text{eff}}(\mathcal{H})$  of an infinite hypothesis space  $\mathcal{H}$  is a measure of its size. Loosely speaking, the effective dimension is equal to the effective number of independent tunable model parameters. These parameters might be the coefficients used in a linear map or the weights and bias terms of an ANN.

See also: hypothesis space, model parameter, ANN.

**embedding** An embedding is a map  $h$  that represents data points using the elements (or vectors) of a given vector space. The map is constructed such that similar objects are represented by similar vectors, according to some metric in the vector space. ML methods learn the map  $h \in \mathcal{H}$ , out of a model  $\mathcal{H}$ , by optimizing a quantitative measure of how well it captures the intrinsic structure of a given training set. For data points belonging to a metric space, a widely used approach is to minimize the reconstruction error incurred by the embedding [30], [137, Ch. 13].

See also: vector space, feature learning.

**empirical risk** The empirical risk  $\widehat{L}(h|\mathcal{D})$  of a hypothesis on a dataset  $\mathcal{D}$  is the average loss incurred by  $h$  when applied to the data points in  $\mathcal{D}$ .

See also: risk, hypothesis, dataset, loss, data point.

**empirical risk minimization (ERM)** ERM is the optimization problem of selecting a hypothesis  $\hat{h} \in \mathcal{H}$  that minimizes the average loss (or empirical risk) on a training set  $\mathcal{D}$ . The hypothesis is chosen from a hypothesis space (or model)  $\mathcal{H}$ . The dataset  $\mathcal{D}$  is referred to as training set. A plethora of ERM-based ML methods is obtained for different design choices for the dataset, model, and loss [8, Ch. 3]. Fig. 85 illustrates ERM for a linear model and data points that are characterized by a single feature  $x$  and a label  $y$ . The hypothesis  $h$  is a linear map that predicts the label of a data point as a linear function of its feature  $x$ , i.e.,  $h(x) = w_1x + w_0$ , where  $w_1$  and  $w_0$  are the model parameters of the hypothesis  $h$ . The ERM problem is to find the model

parameters  $w_1$  and  $w_0$  that minimize the average loss (or empirical risk) incurred by the hypothesis  $h$  on the training set  $\mathcal{D}$ .

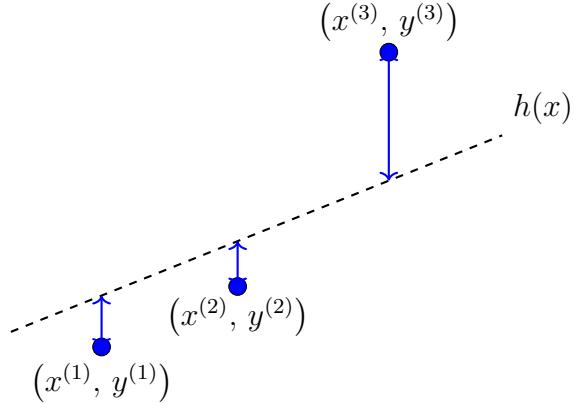


Fig. 85. ERM learns a hypothesis  $h \in \mathcal{H}$  out of a model  $\mathcal{H}$  by minimizing the average loss (or empirical risk)  $1/m \sum_{r=1}^m L((\mathbf{x}^{(r)}, y^{(r)}), h)$  incurred on a training set  $\mathcal{D}$ .

See also: optimization problem, loss, empirical risk, training set, optimization method.

**encoder** See autoencoder.

**ensemble** An ensemble method combines multiple ML methods, each of those referred to as a base learner, to improve overall performance. The base learners can be ERM-based using different choices for the loss, model, and training set. By aggregating the predictions of base learners, ensemble methods can often achieve better performance than any single base learner. The aggregation can amount to averaging the

predictions of base learners (in regression) or using a majority vote (for classification methods).

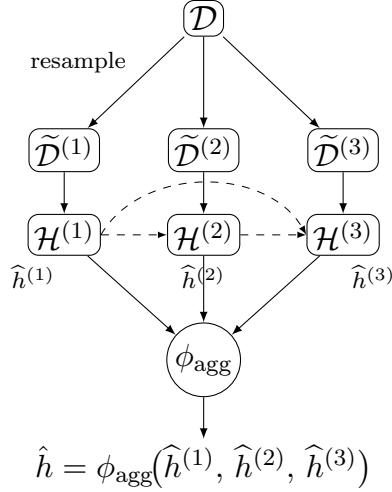


Fig. 86. A generic ensemble with three base learners, each using ERM to learn  $\mathcal{H}^{(j)} \in \mathcal{H}^{(j)}$  based on the training set  $\tilde{\mathcal{D}}^{(j)}$ . A base learner might also use the output of other base learners. The final hypothesis  $\hat{h}$  is obtained by aggregating the hypotheses generated by the base learners.

Different ensemble methods use different constructions for the base learners. For example, bagging methods (such as a random forest) use random sampling to construct slightly different training sets for each base learner. On the other hand, boosting methods run the base learners sequentially, i.e., each base learner tries to correct the prediction errors of the previous ones. A third family of ensemble methods is stacking, where base learners are trained on the same training set but with different models.

See also: bagging, boosting, stacking.

**epoch** An epoch represents one complete pass of the entire training set through some learning algorithm. It refers to the point at which a model has processed every data point in the training set once. Training a model usually requires multiple epochs, since each iteration allows the model to refine the parameters and improve predictions. The number of epochs is something predefined by the user, and thus a hyperparameter, which plays a crucial role in determining how the model will generalize to unseen data. Too few epochs will result in underfitting, while too many epochs can result in overfitting.

See also: training set, algorithm, model, data point, parameter, prediction, underfitting, overfitting.

**estimation error** Consider data points, each with feature vector  $\mathbf{x}$  and label  $y$ . In some applications, we can model the relation between the feature vector and the label of a data point as  $y = \bar{h}(\mathbf{x}) + \varepsilon$ . Here, we use some true underlying hypothesis  $\bar{h}$  and a noise term  $\varepsilon$ , which summarizes any modeling or labeling errors. The estimation error incurred by an ML method that learns a hypothesis  $\hat{h}$ , e.g., using ERM, is defined as  $\hat{h}(\mathbf{x}) - \bar{h}(\mathbf{x})$  for some feature vector. For a parametric hypothesis space, which consists of hypothesis maps determined by model parameters  $\mathbf{w}$ , we can define the estimation error as  $\Delta\mathbf{w} = \hat{\mathbf{w}} - \bar{\mathbf{w}}$  [83], [43].

See also: data point, feature vector, label, hypothesis, ML, ERM, hypothesis space, map, model parameter.

**expectation** Consider a numeric feature vector  $\mathbf{x} \in \mathbb{R}^d$  that we interpret

as the realization of an RV with probability distribution  $p(\mathbf{x})$ . The expectation of  $\mathbf{x}$  is defined as the integral  $\mathbb{E}\{\mathbf{x}\} := \int \mathbf{x} p(\mathbf{x})$ . Note that the expectation is only defined if this integral exists, i.e., if the RV is integrable [2], [6], [73]. Fig. 87 illustrates the expectation of a scalar discrete RV  $x$  that takes on values from a finite set only.

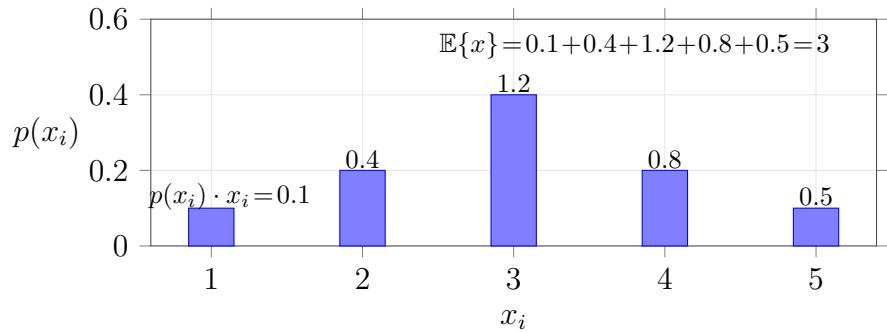


Fig. 87. The expectation of a discrete RV  $x$  is obtained by summing its possible values  $x_i$ , weighted by the corresponding probability  $p(x_i) = \mathbb{P}(x = x_i)$ .

See also: feature vector, realization, RV, probability distribution, probability.

**expert** ML aims to learn a hypothesis  $h$  that accurately predicts the label of a data point based on its features. We measure the prediction error using some loss function. Ideally, we want to find a hypothesis that incurs minimal loss on any data point. We can make this informal goal precise via the i.i.d. assumption and by using the Bayes risk as the baseline for the (average) loss of a hypothesis. An alternative approach to obtaining a baseline is to use the hypothesis  $h'$  learned by an existing ML method. We refer to this hypothesis  $h'$  as an expert [138]. Regret

minimization methods learn a hypothesis that incurs a loss comparable to that of the best expert [138], [139].

See also: loss function, baseline, regret.

**explainability** We define the (subjective) explainability of an ML method as the level of simulatability [140] of the predictions delivered by an ML system to a human user. Quantitative measures of the (subjective) explainability of a trained model can be constructed by comparing its predictions with the predictions provided by a user on a test set [140], [141]. Alternatively, we can use probabilistic models for data and measure the explainability of a trained ML model via the conditional (or differential) entropy of its predictions, given the user's predictions [142], [143].

See also: trustworthy AI, regularization.

**explainable empirical risk minimization (EERM)** EERM is an instance of structural risk minimization (SRM) that adds a regularization term to the average loss in the objective function of ERM. The regularization term is chosen to favor hypothesis maps that are intrinsically explainable for a specific user. This user is characterized by their predictions provided for the data points in a training set [141].

See also: SRM, regularization, ERM, training set.

**explainable machine learning (XML)** XML methods aim to complement each prediction with an explanation of how the prediction has been obtained. The construction of an explicit explanation might not be necessary if the ML method uses a sufficiently simple (or interpretable)

model [144].

See also: prediction, explanation, ML, model.

**explanation** One approach to enhance the transparency of an ML method for its human user is to provide an explanation alongside the predictions delivered by the method. Explanations can take different forms. For instance, they may consist of human-readable text or quantitative indicators, such as feature importance scores for the individual features of a given data point [145]. Alternatively, explanations can be visual—for example, intensity maps that highlight image regions that drive the prediction [146]. Fig. 88 illustrates two types of explanations. The first is a local linear approximation  $g(\mathbf{x})$  of a nonlinear trained model  $\hat{h}(\mathbf{x})$  around a specific feature vector  $\mathbf{x}'$ , as used in the method LIME. The second form of explanation depicted in the figure is a sparse set of predictions  $\hat{h}(\mathbf{x}^{(1)}), \hat{h}(\mathbf{x}^{(2)}), \hat{h}(\mathbf{x}^{(3)})$  at selected feature vectors, offering concrete reference points for the user.

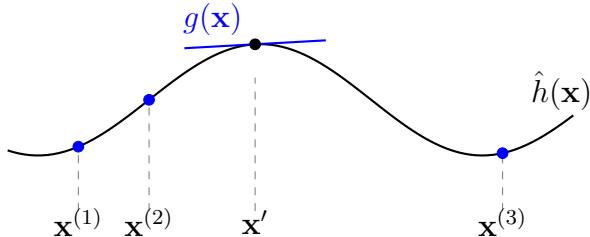


Fig. 88. A trained model  $\hat{h}(\mathbf{x})$  can be explained locally at some point  $\mathbf{x}'$  by a linear approximation  $g(\mathbf{x})$ . For a differentiable  $\hat{h}(\mathbf{x})$ , this approximation is determined by the gradient  $\nabla \hat{h}(\mathbf{x}')$ . Another form of explanation could be the function values  $\hat{h}(\mathbf{x}^{(r)})$  for  $r = 1, 2, 3$ .

See also: ML, prediction, feature, data point, classification.

**feature** A feature of a data point is one of its properties that can be measured or computed easily without the need for human supervision. For example, if a data point is a digital image (e.g., stored as a .jpeg file), then we could use the red–green–blue (RGB) intensities of its pixels as features. Another example is shown in Fig. 89, where the signal samples of a finite-duration audio signal are used as its features.

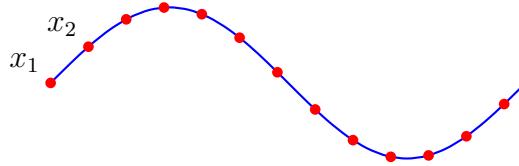


Fig. 89. An audio signal (blue waveform) and its discretized signal samples (red dots) that can be used as its features  $x_1, \dots, x_d$ .

Domain-specific synonyms for the term feature are "covariate," "explanatory variable," "independent variable," "input (variable)," "predictor (variable)," or "regressor" [89], [147], [148].

See also: data point.

**feature learning** Consider an ML application with data points characterized by raw features  $\mathbf{x} \in \mathcal{X}$ . Feature learning refers to the task of learning a map

$$\Phi : \mathcal{X} \rightarrow \mathcal{X}' : \mathbf{x} \mapsto \mathbf{x}'$$

that reads in the features  $\mathbf{x} \in \mathcal{X}$  of a data point and delivers new features  $\mathbf{x}' \in \mathcal{X}'$  from a new feature space  $\mathcal{X}'$ . Different feature learning methods

are obtained for different design choices of  $\mathcal{X}, \mathcal{X}'$ , for a hypothesis space  $\mathcal{H}$  of potential maps  $\Phi$ , and for a quantitative measure of the usefulness of a specific  $\Phi \in \mathcal{H}$ . For example, PCA uses  $\mathcal{X} := \mathbb{R}^d$ ,  $\mathcal{X}' := \mathbb{R}^{d'}$  with  $d' < d$ , and a hypothesis space

$$\mathcal{H} := \{\Phi : \mathbb{R}^d \rightarrow \mathbb{R}^{d'} : \mathbf{x}' := \mathbf{F}\mathbf{x} \text{ with some } \mathbf{F} \in \mathbb{R}^{d' \times d}\}.$$

PCA measures the usefulness of a specific map  $\Phi(\mathbf{x}) = \mathbf{F}\mathbf{x}$  by the minimum linear reconstruction error incurred on a dataset such that

$$\min_{\mathbf{G} \in \mathbb{R}^{d \times d'}} \sum_{r=1}^m \|\mathbf{G}\mathbf{F}\mathbf{x}^{(r)} - \mathbf{x}^{(r)}\|_2^2.$$

See also: feature, feature space, hypothesis space, PCA.

**feature map** A feature map refers to a function

$$\Phi : \mathcal{X} \rightarrow \mathcal{X}', \quad \mathbf{x} \mapsto \mathbf{x}'$$

that transforms a feature vector  $\mathbf{x} \in \mathcal{X}$  of a data point into a new feature vector  $\mathbf{x}' \in \mathcal{X}'$ , where  $\mathcal{X}'$  is typically different from  $\mathcal{X}$ . The transformed representation  $\mathbf{x}'$  is often more useful than the original  $\mathbf{x}$ . For instance, the geometry of data points may become more linear in  $\mathcal{X}'$ , allowing the application of a linear model to  $\mathbf{x}'$ . This idea is central to the design of kernel methods [78]. Other benefits of using a feature map include reducing overfitting and improving interpretability [149]. A common use case is data visualization, where a feature map with two output dimensions allows the representation of data points in a 2-D scatterplot. Some ML methods employ trainable feature maps, whose

parameters are learned from data. An example is the use of hidden layers in a deep net, which act as successive feature maps [150]. A principled way to train a feature map is through ERM, using a loss function that measures reconstruction quality, e.g.,  $L = \|\mathbf{x} - r(\mathbf{x}')\|^2$ , where  $r(\cdot)$  is a trainable map that attempts to reconstruct  $\mathbf{x}$  from the transformed feature vector  $\mathbf{x}'$ .

See also: feature, map, kernel method, feature learning, PCA.

**feature matrix** Consider a dataset  $\mathcal{D}$  with  $m$  data points with feature vectors  $\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)} \in \mathbb{R}^d$ . It is convenient to collect the individual feature vectors into the feature matrix:

$$\mathbf{X} := (\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)})^T = \begin{pmatrix} x_1^{(1)} & x_2^{(1)} & \cdots & x_d^{(1)} \\ x_1^{(2)} & x_2^{(2)} & \cdots & x_d^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{(m)} & x_2^{(m)} & \cdots & x_d^{(m)} \end{pmatrix} \in \mathbb{R}^{m \times d}.$$

Note that the feature matrix is of size  $m \times d$ , i.e., it has  $m$  rows and  $d$  columns.

See also: dataset, data point, feature vector, feature, matrix.

**feature space** The feature space of a given ML application or method is constituted by all potential values that the feature vector of a data point can take on. For data points described by a fixed number  $d$  of numerical features, a common choice for the feature space is the Euclidean space  $\mathbb{R}^d$ . However, the mere presence of  $d$  numeric features does not imply that  $\mathbb{R}^d$  is the most appropriate representation of the feature space. Indeed, the numerical features might be assigned to data points in a

largely arbitrary or random manner, resulting in data points that are randomly scattered throughout  $\mathbb{R}^d$  without any meaningful geometric structure. Feature learning methods try to learn a transformation of the original (potentially nonnumeric) features to ensure a more meaningful arrangement of data points in  $\mathbb{R}^d$ . Three examples of feature spaces are shown in Fig. 90.

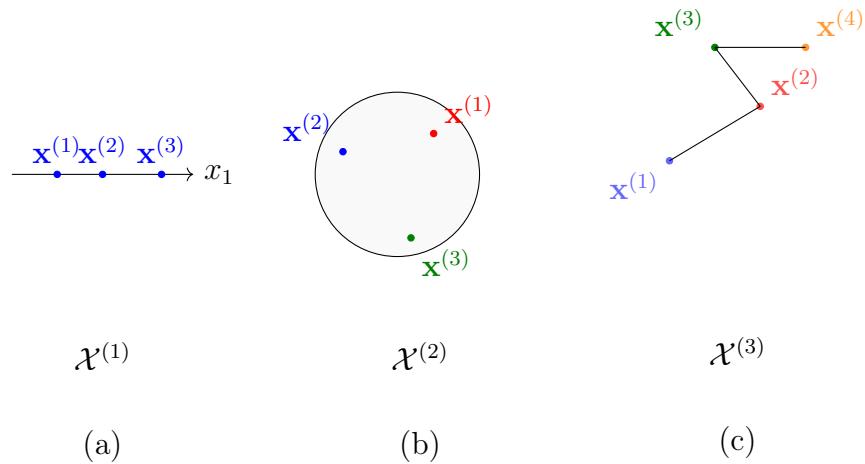


Fig. 90. Three different feature spaces. (a) A linear space  $\mathcal{X}^{(1)} = \mathbb{R}$ . (b) A bounded convex set  $\mathcal{X}^{(2)} \subseteq \mathbb{R}^2$ . (c) A discrete space  $\mathcal{X}^{(3)}$  whose elements are nodes of an undirected graph.

See also: feature vector, Euclidean space.

**feature vector** Feature vector refers to a vector  $\mathbf{x} = (x_1, \dots, x_d)^T$  whose entries are individual features  $x_1, \dots, x_d$ . Many ML methods use feature vectors that belong to some finite-dimensional Euclidean space  $\mathbb{R}^d$ . For some ML methods, however, it can be more convenient to work with feature vectors that belong to an infinite-dimensional vector space (e.g.,

see kernel method).

See also: feature, vector, ML, Euclidean space, vector space.

**federated averaging (FedAvg)** FedAvg refers to a family of iterative FL algorithms. It uses a server-client setting and alternates between clientwise local models retraining, followed by the aggregation of updated model parameters at the server [151], [152]. The local update at client  $i = 1, \dots, n$  at time  $t$  starts from the current model parameters  $\mathbf{w}^{(t)}$  provided by the server and typically amounts to executing few iterations of SGD. After completing the local updates, they are aggregated by the server (e.g., by averaging them). Fig. 91 illustrates the execution of a single iteration of FedAvg.

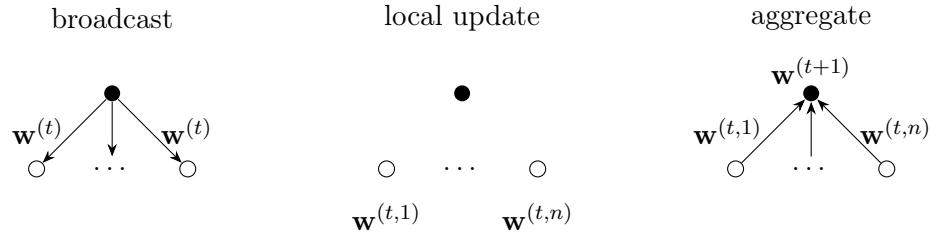


Fig. 91. Illustration of a single iteration of FedAvg, which consists of broadcasting model parameters by the server, performing local updates at clients, and aggregating the updates by the server.

See also: FL, algorithm, local model, SGD.

**federated gradient descent (FedGD)** FedGD refers to an FL distributed algorithm that can be implemented as message passing across an FL network [152].

See also: FL, distributed algorithm, FL network, gradient step, gradient-based method.

**federated learning (FL)** FL is an umbrella term for ML methods that train models in a collaborative fashion using decentralized data and computation.

See also: ML, model, data.

**federated learning network (FL network)** An FL network is a mathematical model for a federated learning system (FL system) that consists of interconnected devices. These devices are represented by the nodes  $\mathcal{V}$  of an undirected weighted graph

$$\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{A}).$$

An edge  $\{i, i'\} \in \mathcal{E}$  indicates collaborations between two devices  $i, i' \in \mathcal{V}$ . The edge weights  $A_{i,i'} > 0$  quantify the extent of collaborations, which may be related to a communication link capacity, statistical similarity between local datasets, or both. Each device  $i \in \mathcal{V}$  has potentially access to a local dataset  $\mathcal{D}^{(i)}$  and might train a local model  $\mathcal{H}^{(i)}$ , e.g., using ERM-based methods. Many popular FL methods are obtained by coupling the training of these local models across the edges of the FL network [152]. This coupling can be implemented in different ways, e.g., using a penalty term that enforces similarity between the model parameters of neighboring devices, or using the predictions of neighboring devices to augment the local datasets. Fig. 92 illustrates an FL network with four devices.

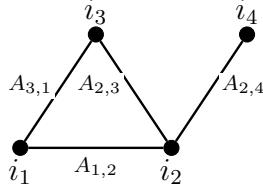


Fig. 92. An FL network with nodes  $\mathcal{V} = \{i_1, i_2, i_3, i_4\}$  representing four devices of an FL system.

The FL network specifies which devices can interact and to what extent.

See also: FL, device, graph, GTVMin.

**federated proximal (FedProx)** FedProx refers to an iterative FL algorithm that alternates between separately training local models and combining the updated local model parameters. In contrast to federated averaging (FedAvg), which uses SGD to train local models, FedProx uses a proximal operator for the training [153].

See also: FL, algorithm, local model, model parameter, FedAvg, SGD, proximal operator.

**federated stochastic gradient descent (FedSGD)** FedSGD refers to an FL distributed algorithm that can be implemented as message passing across an FL network [152].

See also: FL, distributed algorithm, FL network, gradient step, gradient-based method, SGD.

**FedRelax** FedRelax refers to a GTVMin-based FL method for training local models at the devices of an FL network [152]. It is a distributed algorithm that implements a nonlinear variant of the Jacobi method to

solve GTVMin.

See also: FL, distributed algorithm.

**Finnish Meteorological Institute (FMI)** The FMI is a government agency responsible for gathering and reporting weather data in Finland.

See also: data.

**flow-based clustering** Flow-based clustering groups the nodes of an undirected graph by applying  $k$ -means clustering to nodewise feature vectors. These feature vectors are built from network flows between carefully selected sources and destination nodes [154].

See also: clustering, graph,  $k$ -means, feature vector.

**Gaussian mixture model (GMM)** A GMM is a probabilistic model for (the generation of) data points with numeric feature vectors  $\mathbf{x} \in \mathbb{R}^d$  [22], [46]. It assumes that each data point is generated by first drawing a latent cluster index  $I \in \{1, \dots, k\}$  according to cluster probabilities

$$\mathbb{P}(I = c) = p_c, \quad \sum_{c=1}^k p_c = 1.$$

Conditioned on  $I = c$ , the feature vector  $\mathbf{x}$  is drawn from a multivariate normal distribution  $\mathbb{P}^{(c)} = \mathcal{N}(\boldsymbol{\mu}^{(c)}, \mathbf{C}^{(c)})$ . The resulting marginal distribution of  $\mathbf{x}$  is therefore a weighted sum of multivariate normal distributions, i.e.,

$$\mathbb{P} = \sum_{c=1}^k p_c \mathcal{N}(\boldsymbol{\mu}^{(c)}, \mathbf{C}^{(c)}).$$

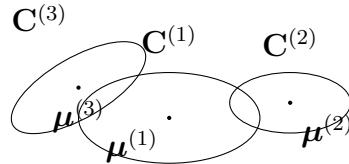


Fig. 93. Illustration of a GMM with three components.

A GMM is parameterized by the cluster-specific probability  $p_c$ , mean  $\mu^{(c)}$ , and covariance matrix  $\mathbf{C}^{(c)}$  for  $c = 1, \dots, k$ .

See also: probabilistic model, multivariate normal distribution, clustering.

**generalization** Generalization refers to the ability of a model trained on a training set to make accurate predictions on new unseen data points. This is a central goal of ML and AI, i.e., to learn patterns that extend beyond the training set. Most ML systems use ERM to learn a hypothesis  $\hat{h} \in \mathcal{H}$  by minimizing the average loss over a training set of data points  $\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(m)}$ , which is denoted by  $\mathcal{D}^{(t)}$ . However, success on the training set does not guarantee success on unseen data—this discrepancy is the challenge of generalization.

To study generalization mathematically, we need to formalize the notion of “unseen” data. A widely used approach is to assume a probabilistic model for data generation, such as the i.i.d. assumption. Here, we interpret data points as independent RVs with an identical probability distribution  $p(\mathbf{z})$ . This probability distribution, which is assumed fixed but unknown, allows us to define the risk of a trained model  $\hat{h}$  as the

expected loss:

$$\bar{L}(\hat{h}) = \mathbb{E}_{\mathbf{z} \sim p(\mathbf{z})} \{ L(\hat{h}, \mathbf{z}) \}.$$

The difference between risk  $\bar{L}(\hat{h})$  and empirical risk  $\widehat{L}(\hat{h}|\mathcal{D}^{(t)})$  is known as the generalization gap. Tools from probability theory, such as concentration inequalities and uniform convergence, allow us to bound this gap under certain conditions [155].

**Generalization without probability:** Probability theory is one way to study how well a model generalizes beyond the training set, but it is not the only way. Another option is to use simple deterministic changes to the data points in the training set. The basic idea is that a good model  $\hat{h}$  should be robust, i.e., its prediction  $\hat{h}(\mathbf{x})$  should not change much if we slightly change the features  $\mathbf{x}$  of a data point  $\mathbf{z}$ . For example, an object detector trained on smartphone photos should still detect the object if a few random pixels are masked [156]. Similarly, it should deliver the same result if we rotate the object in the image [150]. See Fig. 94 for a visual illustration.

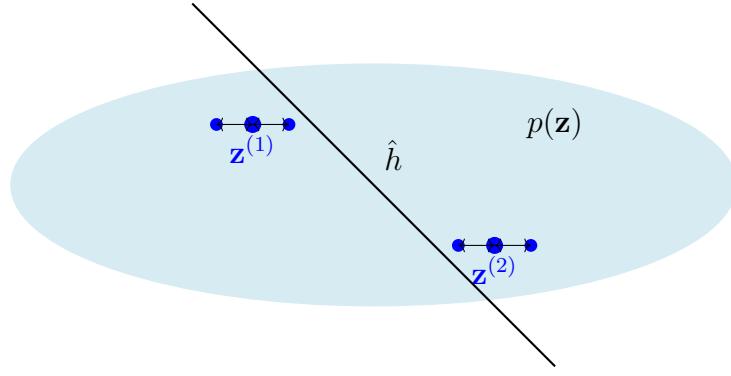


Fig. 94. Two data points  $\mathbf{z}^{(1)}, \mathbf{z}^{(2)}$  that are used as a training set to learn a hypothesis  $\hat{h}$  via ERM. We can evaluate  $\hat{h}$  outside  $\mathcal{D}^{(t)}$  either by an i.i.d. assumption with some underlying probability distribution  $p(\mathbf{z})$  or by perturbing the data points.

See also: ERM, i.i.d. assumption, overfitting, validation.

**generalization gap** Generalization gap is the difference between the performance of a hypothesis  $h \in \mathcal{H}$  on the training set  $\mathcal{D}^{(t)}$  and its performance on data points outside  $\mathcal{D}^{(t)}$ . We can make this notion precise by using a probabilistic model that allows us to compute the risk (or expected loss)  $\bar{L}(\hat{h})$  of a hypothesis  $h$ .

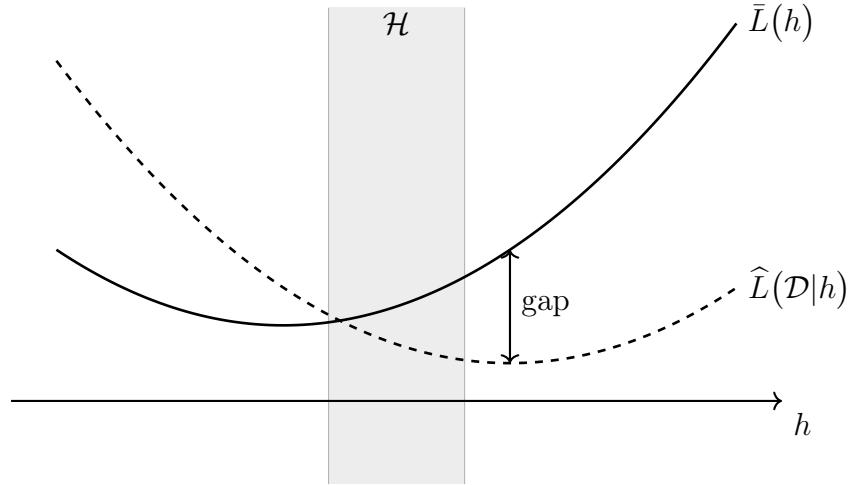


Fig. 95. The generalization gap can be defined as the difference between the risk  $\bar{L}(h)$  and the average loss (or empirical risk)  $\hat{L}(h|\mathcal{D}^{(t)})$  computed on a training set.

In practice, the probability distribution underlying this expectation is unknown. Thus, we need to estimate the expectation based on observed data points. Validation techniques use different constructions of a validation set, which is different from the training set, to estimate the generalization gap.

See also: generalization, validation, ERM, loss function.

**generalized additive model (GAM)** A GAM is obtained from a linear model by replacing the original features  $x_j$ , for  $j = 1, \dots, d$ , of a data point with nonlinear functions  $\phi_j(x_j)$  [157]. More formally, a GAM

consists of hypothesis maps of the form

$$h(\mathbf{x}) = w_0 + \sum_{j=1}^d w_j \phi_j(x_j).$$

See also: linear model, feature, function.

**generalized total variation (GTV)** GTV is a measure of the variation of trained local models  $h^{(i)}$  (or their model parameters  $\mathbf{w}^{(i)}$ ) assigned to the nodes  $i = 1, \dots, n$  of an undirected weighted graph  $\mathcal{G}$  with edges  $\mathcal{E}$ . Given a measure  $d^{(h,h')}$  of the discrepancy between hypothesis maps  $h, h'$ , the GTV is

$$\sum_{\{i,i'\} \in \mathcal{E}} A_{i,i'} d^{(h^{(i)}, h^{(i')})}.$$

Here,  $A_{i,i'} > 0$  denotes the weight of the undirected edge  $\{i, i'\} \in \mathcal{E}$ .

See also: local model, model parameter, graph, discrepancy, hypothesis, map.

**generalized total variation minimization (GTVMMin)** GTVMMin is an instance of regularized empirical risk minimization (RERM) using the GTV of local model parameters as a regularizer [158].

See also: RERM, GTV, regularizer.

**gradient boosting** Gradient boosting is a boosting algorithm that learns a hypothesis  $\tilde{h}$  by sequentially combining the hypotheses  $\hat{h}^{(t)}$  [83, Algorithm 10.3], [110]. Similar to AdaBoost, gradient boosting uses a generalized gradient step to combine the results of the base learners:

$$\tilde{h}^{(t)} = \tilde{h}^{(t-1)} - \eta^{(t)} \hat{h}^{(t)},$$

where the generalized gradient  $\hat{h}t$  is constructed from the  $t$ th base learner. The difference between AdaBoost and gradient boosting is in the construction of  $\hat{h}t$ . While AdaBoost uses weighted ERM for this construction, gradient boosting uses ERM on a modified training set. This modification is obtained by leaving the feature vectors untouched but replacing the labels with the partial derivative of the loss function with respect to the predictions of the previous base learner.

See also: boosting, AdaBoost, GD.

**gradient descent (GD)** GD is an iterative method for finding the minimum of a differentiable function  $f : \mathbb{R}^d \rightarrow \mathbb{R}$ . GD generates a sequence of estimates  $\mathbf{w}^{(0)}, \mathbf{w}^{(1)}, \mathbf{w}^{(2)}, \dots$  that (ideally) converge to a minimum of  $f$ . At each iteration  $k$ , GD refines the current estimate  $\mathbf{w}^{(k)}$  by taking a step in the direction of the steepest descent of a local linear approximation. This direction is given by the negative gradient  $\nabla f(\mathbf{w}^{(k)})$  of the function  $f$  at the current estimate  $\mathbf{w}^{(k)}$ . The resulting update rule is given by

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - \eta \nabla f(\mathbf{w}^{(k)}), \quad (11)$$

where  $\eta > 0$  is a suitably small step size. For a suitably chosen step size  $\eta$ , the update typically reduces the function value, i.e.,  $f(\mathbf{w}^{(k+1)}) < f(\mathbf{w}^{(k)})$ . Fig. 96 illustrates a single GD step.

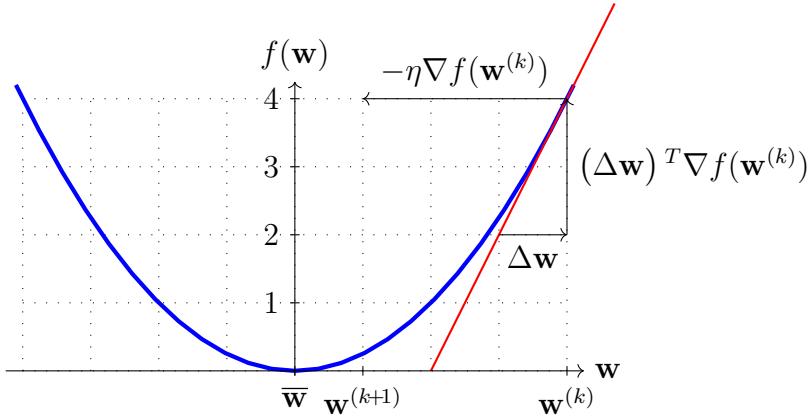


Fig. 96. A single gradient step (11) toward the minimizer  $\bar{\mathbf{w}}$  of  $f(\mathbf{w})$ .

See also: minimum, differentiable, gradient, step size, gradient step.

**gradient-based method** A gradient-based method is an iterative technique for finding the minimum (or maximum) of a differentiable objective function  $f(\mathbf{w})$  of the model parameters  $\mathbf{w}$ . Such a method constructs a sequence of approximations to an optimal choice for  $\mathbf{w}$ . As the name indicates, a gradient-based method uses the gradients of the objective function evaluated during previous iterations to construct new, (hopefully) improved model parameters. One important example of a gradient-based method is GD.

See also: gradient, differentiable, objective function, optimization method, GD.

**graph clustering** Graph clustering aims to cluster data points that are represented as the nodes of a graph  $\mathcal{G}$ . The edges of  $\mathcal{G}$  represent pairwise similarities between data points. We can sometimes quantify

the extent of these similarities by an edge weight [154], [65].

See also: graph, clustering, data point, edge weight.

**graph neural network (GNN)** A GNN is a special type of ANN that is defined via a given graph. In a GNN, node-associated representations are updated by aggregating and transforming embeddings of neighboring nodes [134], [159], [160].

See also: ANN, graph.

**hard clustering** Hard clustering refers to the task of partitioning a given set of data points into (a few) nonoverlapping clusters. This requirement allows us to represent a cluster by a subset of data points, i.e., precisely those belonging to the cluster. In contrast to hard clustering, soft clustering methods allow for overlapping clusters and specify, for each data point, a numeric degree of belonging to each cluster. Hard clustering is an extreme case of soft clustering where the degrees of belonging take only two values, indicating either no belonging or full belonging. For data points characterized by numeric feature vectors  $\mathbf{x} \in \mathbb{R}^d$ , a widely used hard clustering method is  $k$ -means. Any hard clustering method for numeric feature vectors  $\mathbf{x} \in \mathbb{R}^d$  can be adapted for nonnumerical data using feature learning methods. One important example of this approach is spectral clustering, where data points have a similarity structure in the form of an undirected graph. The nodes of this graph represent data points while undirected (possibly weighted) edges represent similarities (and their extend) between data points. We can then use the entries of the eigenvectors of the Laplacian matrix as

numeric features for each data point.

See also: clustering, data point, cluster,  $k$ -means.

**high-dimensional regime** The high-dimensional regime of ERM is characterized by the effective dimension of the model being larger than the sample size, i.e., the number of (labeled) data points in the training set. For example, linear regression methods operate in the high-dimensional regime whenever the number  $d$  of features used to characterize data points exceeds the number of data points in the training set. Another example of ML methods that operate in the high-dimensional regime is large ANNs, which have far more tunable weights (and bias terms) than the total number of data points in the training set. High-dimensional statistics is a recent main thread of probability theory that studies the behavior of ML methods in the high-dimensional regime [16], [161].

See also: ERM, effective dimension, overfitting, regularization.

**hinge loss** Consider a data point characterized by a feature vector  $\mathbf{x} \in \mathbb{R}^d$  and a binary label  $y \in \{-1, 1\}$ . The hinge loss incurred by a real-valued hypothesis map  $h(\mathbf{x})$  is defined as

$$L((\mathbf{x}, y), h) := \max\{0, 1 - yh(\mathbf{x})\}. \quad (12)$$

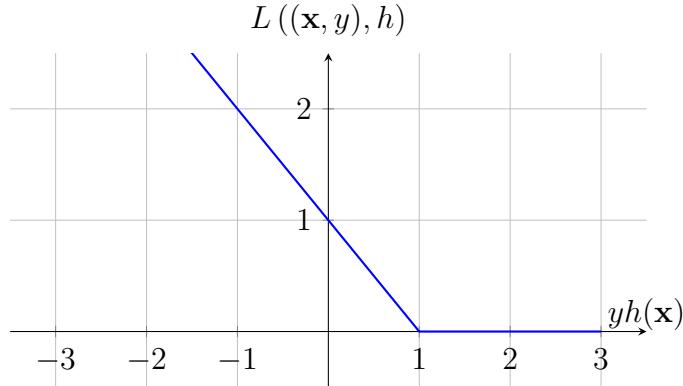


Fig. 97. The hinge loss incurred by the prediction  $h(\mathbf{x}) \in \mathbb{R}$  for a data point with label  $y \in \{-1, 1\}$ . A regularized variant of the hinge loss is used by the SVM [162].

See also: SVM, classification, classifier.

**histogram** Consider a dataset  $\mathcal{D}$  that consists of  $m$  data points  $\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(m)}$ , each of them belonging to some cell  $[-U, U] \times \dots \times [-U, U] \subseteq \mathbb{R}^d$  with side length  $U$ . We partition this cell evenly into smaller elementary cells with side length  $\Delta$ . The histogram of  $\mathcal{D}$  assigns each elementary cell to the corresponding fraction of data points in  $\mathcal{D}$  that fall into this elementary cell. A visual example of such a histogram is provided in Fig. 98.

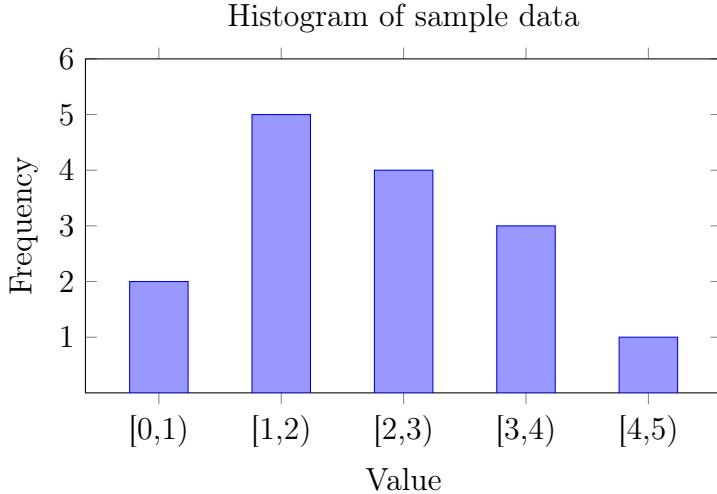


Fig. 98. A histogram consists of the fractions of data points that fall within different value ranges (i.e., bins). Each bar height shows the count of samples in the corresponding interval.

See also: dataset, data point, sample.

**horizontal federated learning (HFL)** HFL uses local datasets constituted by different data points but uses the same features to characterize them [163]. For example, weather forecasting uses a network of spatially distributed weather (observation) stations. Each weather station measures the same quantities, such as daily temperature, air pressure, and precipitation. However, different weather stations measure the characteristics or features of different spatiotemporal regions. Each spatiotemporal region represents an individual data point, each characterized by the same features (e.g., daily temperature or air pressure).

See also: semi-supervised learning (SSL), FL, vertical federated learning

(VFL).

**Huber loss** The Huber loss unifies the squared error loss and the absolute error loss.

See also: loss, squared error loss, absolute error loss.

**Huber regression** Huber regression [164] refers to ERM-based methods that use the Huber loss as a measure of the prediction error. Two important special cases of Huber regression are least absolute deviation regression and linear regression. Tuning the threshold parameter of the Huber loss allows the user to trade the robustness of the absolute error loss against the computational benefits of the smooth squared error loss.

See also: least absolute deviation regression, linear regression, absolute error loss, squared error loss.

**hyperparameter** A hyperparameter associated with an ML method is a quantity that is used to select among a family of models. Typical examples include the learning rate used in a gradient-based method, the number of features used in a linear model, or the maximum depth of a decision tree. The usefulness of a specific hyperparameter choice can be assessed via validation. Similar to learning (or tuning) model parameters by ERM on a training set, we can learn (or tune) hyperparameters via minimizing the validation error. Thus, in a sense, hyperparameters are higher-level model parameters that are learned via a higher-level form of ERM, i.e., minimizing the validation error obtained by the trained model with a given hyperparameter value.

See also: model, validation, model parameter.

**hypothesis** A hypothesis refers to a map (or function)  $h : \mathcal{X} \rightarrow \mathcal{Y}$  from the feature space  $\mathcal{X}$  to the label space  $\mathcal{Y}$ . Given a data point with features  $\mathbf{x}$ , we use a hypothesis map  $h$  to estimate (or approximate) the label  $y$  using the prediction  $\hat{y} = h(\mathbf{x})$ .

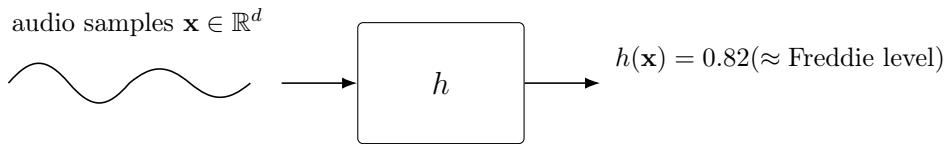


Fig. 99. A hypothesis  $h : \mathcal{X} \rightarrow \mathcal{Y}$  maps the features  $\mathbf{x} \in \mathcal{X}$  of a data point to a prediction  $h(\mathbf{x}) \in \mathcal{Y}$  of the label. For example, the ML application <https://freddiemeter.withyoutube.com/> uses the samples of an audio recording as features to predict how closely a person's singing resembles that of Freddie Mercury.

ML is all about learning (or finding) a hypothesis map  $h$  such that  $y \approx h(\mathbf{x})$  for any data point (with features  $\mathbf{x}$  and label  $y$ ). Practical ML methods, limited by finite computational resources, must restrict learning to a subset of all possible hypothesis maps. This subset is called the hypothesis space or simply the model underlying the method.

See also: map, function, prediction, model.

**hypothesis space** A hypothesis space is a mathematical model that characterizes the learning capacity of an ML method. The goal of such a method is to learn a hypothesis map that maps features of a data point to a prediction of its label. Given a finite amount of computational

resources, a practical ML method typically explores only a restricted set of all possible maps from the feature space to the label space. Such a restricted set is referred to as a hypothesis space  $\mathcal{H}$  underlying the ML method (see Fig. 100). For the analysis of a given ML method, the choice of a hypothesis space  $\mathcal{H}$  is not unique, i.e., any superset containing all maps the method can learn is also a valid hypothesis space.

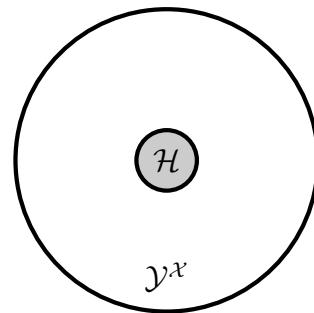


Fig. 100. The hypothesis space  $\mathcal{H}$  of an ML method is a (typically very small) subset of the (typically very large) set  $\mathcal{Y}^{\mathcal{X}}$  of all possible maps from the feature space  $\mathcal{X}$  into the label space  $\mathcal{Y}$ .

On the other hand, from an ML engineering perspective, the hypothesis space  $\mathcal{H}$  is a design choice for ERM-based methods. This design choice can be guided by the available computational resources and statistical aspects. For instance, if efficient matrix operations are feasible and a roughly linear relation exists between features and labels, a linear model can be a useful choice for  $\mathcal{H}$ .

See also: hypothesis, model, map, linear model.

**image segmentation** Image segmentation refers to the task of clustering the pixels of an image into a few segments [165], [166]. Each segment is a subset (or cluster) of pixels that are similar to each other in terms of color, texture, or other visual properties.

See also: clustering.

### **independent and identically distributed assumption (i.i.d. assumption)**

The i.i.d. assumption is a widely used probabilistic model for the generation of data points. In particular, data points are represented as i.i.d. RVs.

See also: i.i.d., probabilistic model, data point, RV.

**inference** In the context of ML, inference refers to the process of evaluating a learned hypothesis (or trained model)  $\hat{h}(\mathbf{x})$  based on the features of a data point [22], [46]. A basic ML workflow starts with model training and then uses the trained model for inference.

See also: model, loss, ERM.

**input vector** The term input vector is often used as a synonym for the feature vector of a data point. In settings where data points arise from a dynamical system observed over time, features are obtained from measuring input variables. These input variables are then used by ML methods to predict the system's output (which is a label in ML terminology).

See also: vector, feature vector, data point, output.

**interpretability** An ML method is interpretable for a human user if they can comprehend the decision process of the method. One approach

to develop a precise definition of interpretability is via the concept of simulatability, i.e., the ability of a human to mentally simulate the model behavior [140], [143], [167], [168], [169]. The idea is as follows: If a human user understands an ML method, then they should be able to anticipate its predictions on a test set. We illustrate such a test set in Fig. 101, which also depicts two learned hypotheses  $\hat{h}$  and  $\hat{h}'$ . The ML method producing the hypothesis  $\hat{h}$  is interpretable to a human user familiar with the concept of a linear map. Since  $\hat{h}$  corresponds to a linear map, the user can anticipate the predictions of  $\hat{h}$  on the test set. In contrast, the ML method delivering  $\hat{h}'$  is not interpretable, because its behavior is no longer aligned with the user's expectations.

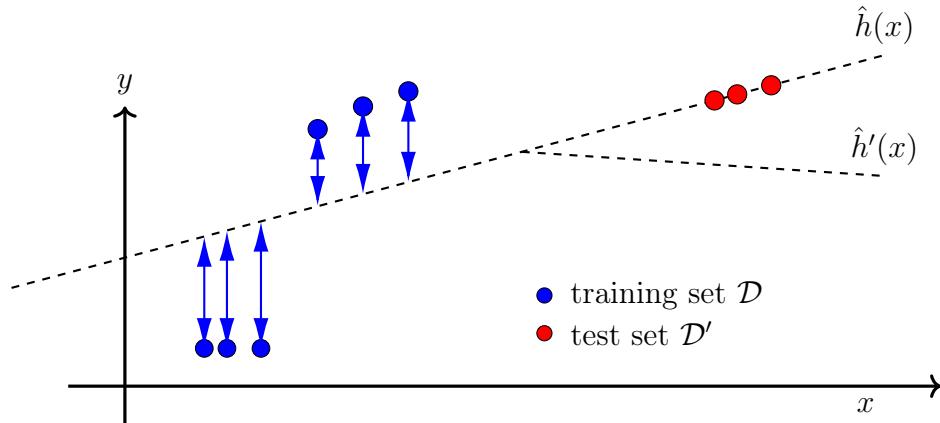


Fig. 101. We can assess the interpretability of trained ML models  $\hat{h}$  and  $\hat{h}'$  by comparing their predictions to pseudo-labels generated by a human user for  $\mathcal{D}'$ .

The notion of interpretability is closely related to the notion of explain-

ability, as both aim to make ML methods more understandable for humans. In the context of Fig. 101, interpretability of an ML method  $\hat{h}$  requires that the human user can anticipate its predictions on an arbitrary test set. This contrasts with explainability, where the user is provided explanations to better understand the predictions of  $\hat{h}$  on a specific test set  $\mathcal{D}'$ . These explanations can be saliency maps or by pointing out reference examples from the training set.

See also: explainability, trustworthy AI, regularization, LIME.

**iteration** The elementary computational step during the execution of an algorithm is referred to as iteration [100], [170]. For example, the elementary computational step of gradient-based methods is a gradient step. More generally, the elementary computational step of a fixed-point iteration is the evaluation of an underlying operator  $\mathcal{F}$ , which might vary across iterations (see Fig. 102). Many important ML algorithms, including Lloyd's algorithm and GD, are fixed-point iterations.

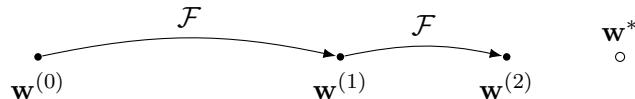


Fig. 102. A fixed-point iteration consists of the repeated application of an operator  $\mathcal{F}$  with some fixed point  $\mathbf{w}^*$ , i.e.,  $\mathcal{F}\mathbf{w}^* = \mathbf{w}^*$ .

See also: algorithm, gradient step, GD.

**Jacobi method** The Jacobi method is an algorithm for solving systems of linear equations (i.e., a linear system) of the form  $\mathbf{Ax} = \mathbf{b}$ . Here,

$\mathbf{A} \in \mathbb{R}^{d \times d}$  is a square matrix with nonzero main diagonal entries. The method constructs a sequence  $\mathbf{x}^{(0)}, \mathbf{x}^{(1)}, \dots$  by updating each entry of  $\mathbf{x}^{(t)}$  as follows:

$$x_i^{(t+1)} = \frac{1}{a_{ii}} \left( b_i - \sum_{j \neq i} a_{ij} x_j^{(t)} \right).$$

Note that all entries  $x_1^{(k)}, \dots, x_d^{(k)}$  are updated simultaneously. The above iteration converges to a solution, i.e.,  $\lim_{t \rightarrow \infty} \mathbf{x}^{(t)} = \mathbf{x}$ , under certain conditions on the matrix  $\mathbf{A}$ , e.g., being strictly diagonally dominant or symmetric positive definite [3], [36], [59]. Jacobi-type methods are appealing for large linear systems due to their parallelizable structure [82]. We can interpret the Jacobi method as a fixed-point iteration. Indeed, using the decomposition  $\mathbf{A} = \mathbf{D} + \mathbf{R}$ , with  $\mathbf{D}$  being the diagonal of  $\mathbf{A}$ , allows us to rewrite the linear equation  $\mathbf{Ax} = \mathbf{b}$  as a fixed-point equation:

$$\mathbf{x} = \underbrace{\mathbf{D}^{-1}(\mathbf{b} - \mathbf{Rx})}_{\mathcal{F}\mathbf{x}},$$

which leads to the iteration  $\mathbf{x}^{(t+1)} = \mathbf{D}^{-1}(\mathbf{b} - \mathbf{Rx}^{(t)})$ .

As an example, for the linear equation  $\mathbf{Ax} = \mathbf{b}$ , where

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix},$$

the Jacobi method updates each component of  $\mathbf{x}$  as follows:

$$\begin{aligned} x_1^{(k+1)} &= \frac{1}{a_{11}} \left( b_1 - a_{12}x_2^{(k)} - a_{13}x_3^{(k)} \right), \\ x_2^{(k+1)} &= \frac{1}{a_{22}} \left( b_2 - a_{21}x_1^{(k)} - a_{23}x_3^{(k)} \right), \\ x_3^{(k+1)} &= \frac{1}{a_{33}} \left( b_3 - a_{31}x_1^{(k)} - a_{32}x_2^{(k)} \right). \end{aligned}$$

See also: algorithm, matrix, fixed-point iteration, optimization method.

**$k$ -fold cross-validation ( $k$ -fold CV)** A  $k$ -fold CV is a method for evaluating the generalization gap of an ERM-based ML method. The idea is to divide a dataset  $\mathcal{D}$  evenly into  $k$  subsets (or folds)  $\mathcal{D}^{(1)}, \dots, \mathcal{D}^{(k)}$ .

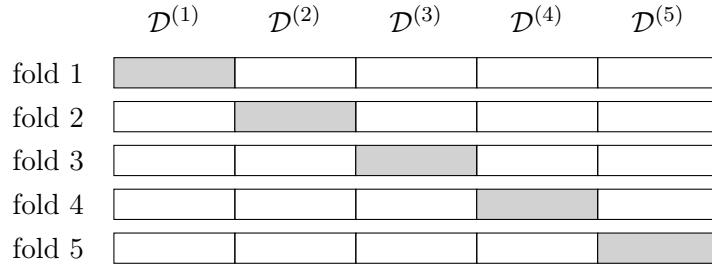


Fig. 103. In  $k$ -fold CV, the available dataset  $\mathcal{D}$  is evenly divided into  $k$  folds  $\mathcal{D}^{(1)}, \dots, \mathcal{D}^{(k)}$ . Each fold is used once as a validation set, while the remaining  $k - 1$  folds form the training set.

For each fold  $b = 1, \dots, k$ , we train the model on the union of all folds except  $\mathcal{D}^{(b)}$  and validate it on  $\mathcal{D}^{(b)}$ . The overall performance is obtained by averaging the validation results across all  $k$  folds.

See also: validation, validation error.

**$k$ -means** The  $k$ -means principle is an optimization-based approach to the clustering of data points with a numeric feature vector [8, Ch. 8]. As a hard clustering approach,  $k$ -means partitions a dataset into  $k$  disjoint subsets (or clusters), which are indexed by  $c = 1, \dots, k$ . Each cluster  $\mathcal{C}$  is characterized by the average feature vector of data points that belong to it. This average (or mean) feature vector is referred to as the cluster centroid  $\mathbf{w}^{(c)}$ . A visual illustration is provided in Fig. 104.

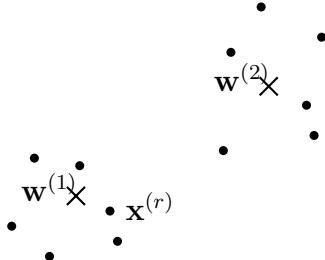


Fig. 104. A scatterplot of data points, indexed by  $r = 1, \dots, m$  and characterized by feature vectors  $\mathbf{x}^{(r)} \in \mathbb{R}^2$ . The scatterplot also includes two cluster centroids  $\mathbf{w}^{(1)}, \mathbf{w}^{(2)} \in \mathbb{R}^2$ .

In general, solving the  $k$ -means optimization problem exactly is challenging (or NP-hard) [171]. However, there are simple iterative methods for finding approximately optimal cluster centroids. One such method is referred to as Lloyd's algorithm.

See also: hard clustering, cluster, Lloyd's algorithm.

**k-nearest neighbors regression (KNNR)** KNNR is a widely used instance of locally weighted learning (LWL) for regression [8], [22]. To make a prediction for a given data point  $\mathbf{z}$ , KNNR identifies the  $k$  data points in the training set that are closest to  $\mathbf{z}$  according to some metric. The prediction is then obtained as the average of the labels of these  $k$  nearest neighbors (NNs).

See also: LWL, regression, data point, training set, NN.

**kernel (kernel method)** Consider a set of data points, each represented by a feature vector  $\mathbf{x} \in \mathcal{X}$ , where  $\mathcal{X}$  denotes the feature space. A (real-valued) kernel is a function  $k : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$  that assigns to every pair of

feature vectors  $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$  a real number  $k(\mathbf{x}, \mathbf{x}')$ . This value is typically interpreted as a similarity measure between  $\mathbf{x}$  and  $\mathbf{x}'$ . The defining property of a kernel is that it is symmetric, i.e.,  $k(\mathbf{x}, \mathbf{x}') = k(\mathbf{x}', \mathbf{x})$ , and that for any finite set of feature vectors  $\mathbf{x}_1, \dots, \mathbf{x}_n \in \mathcal{X}$ , the matrix

$$\mathbf{K} = \begin{pmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & k(\mathbf{x}_1, \mathbf{x}_2) & \dots & k(\mathbf{x}_1, \mathbf{x}_n) \\ k(\mathbf{x}_2, \mathbf{x}_1) & k(\mathbf{x}_2, \mathbf{x}_2) & \dots & k(\mathbf{x}_2, \mathbf{x}_n) \\ \vdots & \vdots & \ddots & \vdots \\ k(\mathbf{x}_n, \mathbf{x}_1) & k(\mathbf{x}_n, \mathbf{x}_2) & \dots & k(\mathbf{x}_n, \mathbf{x}_n) \end{pmatrix} \in \mathbb{R}^{n \times n}$$

is psd. A kernel naturally defines a transformation of a feature vector  $\mathbf{x}$  into a function  $\mathbf{z} = k(\mathbf{x}, \cdot)$ . The function  $\mathbf{z}$  maps an input  $\mathbf{x}' \in \mathcal{X}$  to the value  $k(\mathbf{x}, \mathbf{x}')$ . We can view the function  $\mathbf{z}$  as a new feature vector that belongs to a feature space  $\mathcal{X}'$  that is typically different from  $\mathcal{X}$ . This new feature space  $\mathcal{X}'$  has a particular mathematical structure, i.e., it is a reproducing kernel Hilbert space (RKHS) [78], [162]. Since  $\mathbf{z}$  belongs to a RKHS, which is a vector space, we can interpret it as a generalized feature vector. Note that a finite-length feature vector  $\mathbf{x} = (x_1, \dots, x_d)^T \in \mathbb{R}^d$  can be viewed as a function  $\mathbf{x} : \{1, \dots, d\} \rightarrow \mathbb{R}$  that assigns a real value to each index  $j \in \{1, \dots, d\}$ .

See also: feature vector, feature space, Hilbert space, kernel method.

**kernel method** A kernel method is an ML method that uses a kernel  $k$  to map the original (i.e., raw) feature vector  $\mathbf{x}$  of a data point to a new (transformed) feature vector  $\mathbf{z} = k(\mathbf{x}, \cdot)$  [78], [162]. The motivation for transforming the feature vectors is that, by using a suitable kernel, the data points have a more "pleasant" geometry in the transformed feature space. For example, in a binary classification problem, using

transformed feature vectors  $\mathbf{z}$  might allow us to use linear models, even if the data points are not linearly separable in the original feature space (see Fig. 105).

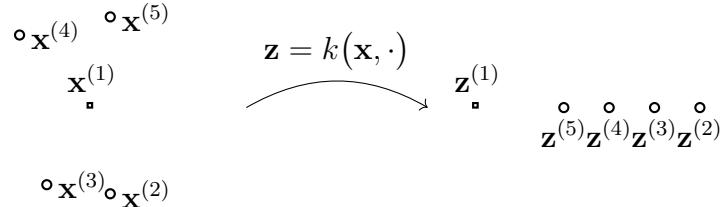


Fig. 105. Five data points characterized by feature vectors  $\mathbf{x}^{(r)}$  and labels  $y^{(r)} \in \{\circ, \blacksquare\}$  for  $r = 1, \dots, 5$ . With these feature vectors, there is no way to separate the two classes by a straight line (representing the decision boundary of a linear classifier). In contrast, the transformed feature vectors  $\mathbf{z}^{(r)} = k(\mathbf{x}^{(r)}, \cdot)$  allow us to separate the data points using a linear classifier.

See also: kernel, feature vector, feature space, linear classifier.

**Kronecker product** The Kronecker product of two matrices  $\mathbf{A} \in \mathbb{R}^{m \times n}$  and  $\mathbf{B} \in \mathbb{R}^{p \times q}$  is a block matrix denoted by  $\mathbf{A} \otimes \mathbf{B}$  and defined as [3], [33]

$$\mathbf{A} \otimes \mathbf{B} = \begin{pmatrix} a_{11}\mathbf{B} & \cdots & a_{1n}\mathbf{B} \\ \vdots & \ddots & \vdots \\ a_{m1}\mathbf{B} & \cdots & a_{mn}\mathbf{B} \end{pmatrix} \in \mathbb{R}^{mp \times nq}.$$

The Kronecker product is a special case of the tensor product for matrices and is widely used in multivariate statistics, linear algebra, and structured ML models. It satisfies the identity  $(\mathbf{A} \otimes \mathbf{B})(\mathbf{x} \otimes \mathbf{y}) =$

$(\mathbf{Ax}) \otimes (\mathbf{By})$  for vectors  $\mathbf{x}$  and  $\mathbf{y}$  of compatible dimensions.

See also: matrix, ML, model, vector.

**label** A label is a higher-level fact or quantity of interest associated with a data point. For example, if the data point is an image, the label could indicate whether the image contains a cat [89], [147], [148].

See also: data point, label space.

**label space** In an ML application, each data point is described by a set of features together with an associated label. The set of all admissible label values is called the label space, denoted by  $\mathcal{Y}$ . Importantly,  $\mathcal{Y}$  may include values that no observed data point has as its label value. To a large extent, the choice of  $\mathcal{Y}$  is up to the ML engineer and depends on the problem formulation. Fig. 106 shows some examples of label spaces that are commonly used in ML applications.

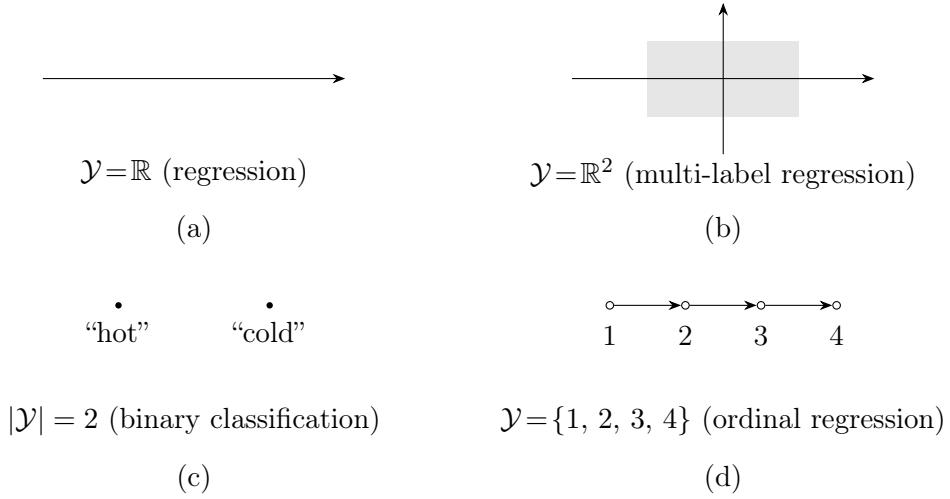


Fig. 106. Examples of label spaces and the corresponding types of ML. (a) Regression. (b) Multi-label regression. (c) Binary classification. (d) Ordinal regression.

The choice of the label space  $\mathcal{Y}$  determines the type of ML methods appropriate for the application at hand. Regression methods use the  $\mathcal{Y} = \mathbb{R}$ , while binary classification methods use a label space  $\mathcal{Y}$  that consists of two different elements, i.e.,  $|\mathcal{Y}| = 2$ . Ordinal regression methods use a finite, ordered set of label values, e.g.,  $\mathcal{Y} = \{1, 2, 3, 4\}$  with the natural ordering  $1 < 2 < 3 < 4$ .

See also: data point, label, regression, classification.

**label vector** Given a dataset of  $m$  labeled data points

$$(\mathbf{x}^{(1)}, y^{(1)}) , \dots , (\mathbf{x}^{(m)}, y^{(m)}) ,$$

it is convenient to collect the corresponding labels into a single label vector  $\mathbf{y} := (y_1, \dots, y_m)^T$  [22], [83].

See also: dataset, labeled data point, label, data point.

**labeled data point** A data point whose label is known or has been determined by some means that might require human labor.

See also: data point, label.

**large language model (LLM)** An LLM is an umbrella term for ML methods that use high-dimensional ML models (with billions of model parameters) trained on large collections of text data. LLMs are used to analyze or generate sequences of tokens that constitute text data. Many current LLMs use some variant of a transformer that is trained via self-supervised learning, i.e., the training is based on the task of predicting a few words that are intentionally removed from a large text corpus. Thus, we can construct labeled data points simply by selecting some words from a given text as labels and the remaining words as features of data points. This construction requires very little human supervision and allows for generating sufficiently large training sets for LLMs.

See also: token, transformer, NLP.

**layer** A deep net is an ANN that consists of consecutive layers, indexed by  $\ell = 1, 2, \dots, L$ . The  $\ell$ th layer consists of artificial neurons  $a_1^{(\ell)}, \dots, a_{d^{(\ell)}}^{(\ell)}$  with the layer width  $d^{(\ell)}$ . Each of these artificial neurons evaluates an activation function for a weighted sum of the outputs (or activations) of the previous layer  $\ell - 1$ . The input to layer  $\ell = 1$  is formed from weighted sums of the features of the data point for which the deep net computes a prediction. The outputs of the neurons in layer  $\ell$  are then,

in turn, used to form the inputs for the neurons in the next layer. The final (output) layer consists of a single neuron whose output is used as the prediction delivered by the deep net.

See also: deep net, ANN.

**learning rate** Consider an iterative ML method for finding or learning a useful hypothesis  $h \in \mathcal{H}$ . Such an iterative method repeats similar computational (update) steps that adjust or modify the current hypothesis to obtain an improved hypothesis. A key parameter of an iterative method is the learning rate. The learning rate controls the extent to which the current hypothesis can be modified during a single iteration. Consider, for example, the gradient step [8, Ch. 5]

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \nabla f(\mathbf{w}^{(t)}) \quad (13)$$

of a gradient-based method for ERM, where the objective function  $f(\mathbf{w})$  is the empirical risk incurred by  $h^{(\mathbf{w})}$  on a training set. Given the current model parameters  $\mathbf{w}^{(t)}$  at iteration  $t$ , the gradient step produces updated model parameters  $\mathbf{w}^{(t+1)}$  by moving in the opposite direction of the gradient  $\nabla f(\mathbf{w}^{(t)})$ .

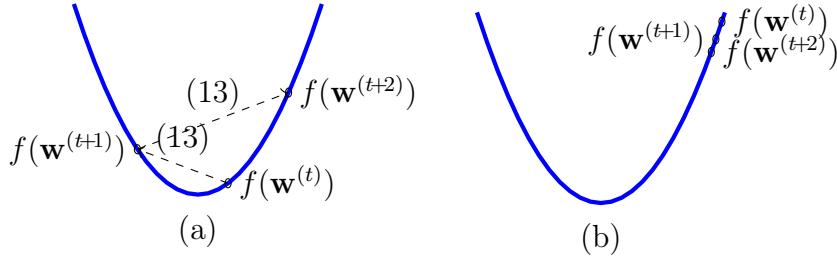


Fig. 107. Effect of using an inadequate learning rate  $\eta$  in the gradient step (13).

(a) If  $\eta$  is too large, the gradient steps can “overshoot” such that the iterates  $\mathbf{w}^{(t)}$  diverge away from the optimum, i.e.,  $f(\mathbf{w}^{(t+1)}) > f(\mathbf{w}^{(t)})$ . (b) If  $\eta$  is too small, the gradient steps make too little progress towards the optimum within the available number of iterations (due to limited computational budget).

See also: ML, hypothesis, parameter, GD, SGD, projected gradient descent (projected GD), step size.

**learning task** Consider a dataset  $\mathcal{D}$  consisting of multiple data points  $\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(m)}$ . For example,  $\mathcal{D}$  can be a collection of images in an image database. A learning task is defined by specifying those properties (or attributes) of a data point that are used as its features and labels. Given a choice of model  $\mathcal{H}$  and loss function, a learning task leads to an instance of ERM and can thus be represented by the associated objective function  $\widehat{L}(h|\mathcal{D})$  for  $h \in \mathcal{H}$ . Importantly, multiple distinct learning tasks can be constructed from the same dataset by selecting different sets of features and labels (see Fig. 108).



An image showing cows grazing in the Austrian countryside.

Task 1 (regression):

Features are the RGB values of all image pixels, and the label is the number of cows depicted.

Task 2 (classification):

Features include the average green intensity of the image, and the label indicates whether cows should be moved to another location (i.e., yes/no).

Fig. 108. Two learning tasks constructed from a single image dataset. These tasks differ in feature selection and choice of label (i.e., the objective), but are both derived from the same dataset.

Different learning tasks arising from the same underlying dataset are often coupled. For example, when a probabilistic model is used to generate data points, statistical dependencies among different labels induce dependencies among the corresponding learning tasks. In general,

solving learning tasks jointly, e.g., using multitask learning methods, tends to be more effective than solving them independently (thereby ignoring dependencies among learning tasks) [172], [173], [174].

See also: multitask learning, label space.

**least absolute deviation regression** Least absolute deviation regression is an instance of ERM using the absolute error loss. It is a special case of Huber regression. For the parametric model  $h^{(w)}(\mathbf{x}) = w$ , ERM with absolute error loss is solved by the median (see Fig. 109). Using squared error loss instead for the same parametric model makes ERM compute the mean.

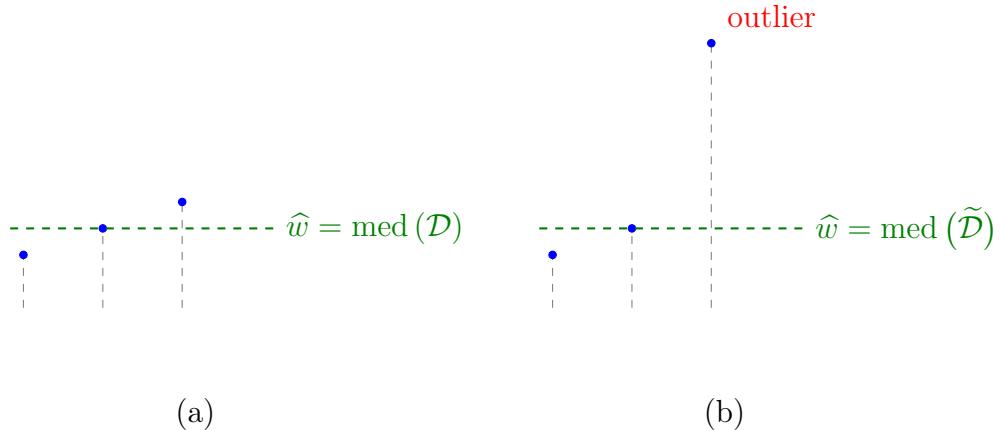


Fig. 109. For the simple parametric model  $h^{(w)}(\mathbf{x}) = w$ , ERM with absolute error loss amounts to computing the median. (a) Original dataset  $\mathcal{D}$ . (b) Noisy dataset  $\tilde{\mathcal{D}}$  including an outlier.

See also: ERM, absolute error loss, Huber regression.

**least absolute shrinkage and selection operator (Lasso)** The Lasso is

an instance of explainable empirical risk minimization (EERM) [83]. It learns the weights  $\mathbf{w}$  of a linear map  $h(\mathbf{x}) = \mathbf{w}^T \mathbf{x}$  from a training set. Lasso is obtained from linear regression by adding the scaled  $\ell_1$ -norm  $\alpha \|\mathbf{w}\|_1$  to the average squared error loss incurred on the training set [175]. Using the  $\ell_1$ -norm as a regularizer, instead of the squared  $\ell_2$ -norm used in ridge regression, encourages the learned weights to have many entries set to zero [16], [161].

See also: EERM, weight, linear map, training set, linear regression, norm, squared error loss.

**least squares** Least squares refers to ERM-based methods that use the average squared error loss

$$\frac{1}{m} \sum_{r=1}^m (y^{(r)} - h(\mathbf{x}^{(r)}))^2$$

on a training set  $\mathcal{D} = \{ (\mathbf{x}^{(1)}, y^{(1)}) , \dots , (\mathbf{x}^{(m)}, y^{(m)}) \}$  to measure the quality of a hypothesis map  $h \in \mathcal{H}$ . We obtain different least squares methods by using different models in ERM. For example, the least squares variant of a linear model is a least squares method that uses a linear model.

See also: ERM, squared error loss, linear model, linear regression.

**leave-one-out cross-validation (LOO-CV)** LOO-CV is a special case of  $k$ -fold CV where the validation set is of size one, i.e., a single data point.

See also:  $k$ -fold CV, validation, validation error.

**linear classifier** Consider data points characterized by numeric features  $\mathbf{x} \in \mathbb{R}^d$  and a label  $y \in \mathcal{Y}$  from some finite label space  $\mathcal{Y}$ . A linear

classifier is characterized by having decision regions that are separated by hyperplanes in  $\mathbb{R}^d$  [8, Ch. 2].

See also: data point, feature, label, label space, classifier, decision region.

**linear discriminant analysis (LDA)** LDA is a classical feature learning method [22], [176]. In the context of binary classification problems, LDA seeks a linear feature map  $\phi^{(\mathbf{w})} : \mathbb{R}^d \rightarrow \mathbb{R} : \mathbf{x} \mapsto \mathbf{w}^T \mathbf{x}$  such that the new feature  $\phi^{(\mathbf{w})}(\mathbf{x})$  optimally allows us to predict the label of a data point.

See also: feature learning, dimensionality reduction, self-supervised learning.

**linear least squares** Linear least squares refers to the variant of linear regression that uses the squared error loss to measure the quality of a linear hypothesis map. Conversely, it can also be viewed as the variant of least squares that restricts the hypothesis to a linear model. In particular, linear least squares learns the parameters  $\mathbf{w}$  of a linear hypothesis  $h^{(\mathbf{w})}(\mathbf{x}) = \mathbf{w}^T \mathbf{x}$  by solving the following optimization problem:

$$\min_{\mathbf{w} \in \mathbb{R}^d} \|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2. \quad (14)$$

Here, the feature matrix is  $\mathbf{X} = (\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)})$  and the label vector is  $\mathbf{y} = (y^{(1)}, \dots, y^{(m)})^T$ . Both are constructed from the training set

$$\mathcal{D} = \{(\mathbf{x}^{(1)}, y^{(1)}), \dots, (\mathbf{x}^{(m)}, y^{(m)})\}.$$

The optimization problem in (14) admits a clear geometric interpretation, i.e., we seek the vector  $\mathbf{X}\hat{\mathbf{w}}$  in the column space of  $\mathbf{X}$  that is closest to

the label vector  $\mathbf{y}$  (see Fig. 110) [25, Ch. 8]. A necessary and sufficient condition for  $\hat{\mathbf{w}}$  to minimize (14) is the normal equations:

$$\mathbf{X}^T \mathbf{X} \hat{\mathbf{w}} = \mathbf{X}^T \mathbf{y}.$$

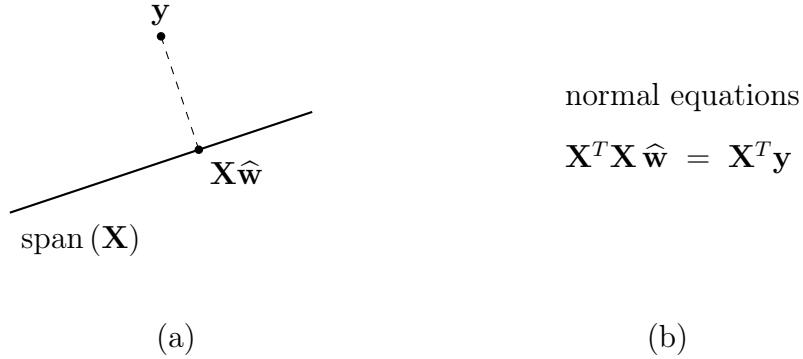


Fig. 110. Linear least squares has both geometric and algebraic interpretations.

(a) Geometrically, it finds the orthogonal projection of the label vector  $\mathbf{y}$  onto the column space of the feature matrix  $\mathbf{X}$  [25, Ch. 8]. (b) Algebraically, it solves a linear system known as normal equations.

See also: least squares, linear regression, squared error loss, linear model, ERM.

**linear model** Consider an ML application involving data points, each represented by a numeric feature vector  $\mathbf{x} \in \mathbb{R}^d$ . A linear model defines a hypothesis space consisting of all real-valued linear maps from  $\mathbb{R}^d$  to  $\mathbb{R}$  such that

$$\mathcal{H}^{(d)} := \left\{ h : \mathbb{R}^d \rightarrow \mathbb{R} \mid h(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} \text{ for some } \mathbf{w} \in \mathbb{R}^d \right\}.$$

Each value of  $d$  defines a different hypothesis space, corresponding to the number of features used to compute the prediction  $h(\mathbf{x})$ . The choice of

$d$  is often guided not only by computational aspects (e.g., fewer features reduce computation) and statistical aspects (e.g., more features typically reduce bias and risk), but also by interpretability. A linear model using a small number of well-chosen features is generally considered more interpretable [144], [149]. The linear model is attractive because it can typically be trained using scalable convex optimization methods [77], [83]. Moreover, linear models often permit rigorous statistical analysis, including fundamental limits on the minimum achievable risk [16]. They are also useful for analyzing more complex nonlinear models such as ANNs. For instance, a deep net can be viewed as the composition of a feature map—implemented by the input and hidden layers—and a linear model in the output layer. Similarly, a decision tree can be interpreted as applying a one-hot-encoded feature map based on decision regions, followed by a linear model that assigns a prediction to each region. More generally, any trained model  $\hat{h} \in \mathcal{H}$  that is differentiable at some  $\mathbf{x}'$  can be locally approximated by a linear map  $g(\mathbf{x})$ . Fig. 111 illustrates such a local linear approximation, defined by the gradient  $\nabla \hat{h}(\mathbf{x}')$ . Note that the gradient is only defined where  $\hat{h}$  is differentiable. To ensure robustness in the context of trustworthy AI, one may prefer models whose associated map  $\hat{h}$  is Lipschitz continuous. A classic result in mathematical analysis—Rademacher’s theorem—states that if  $\hat{h}$  is Lipschitz continuous with some constant  $L$  over an open set  $\Omega \subseteq \mathbb{R}^d$ , then  $\hat{h}$  is differentiable almost everywhere in  $\Omega$  [177, Th. 3.1].

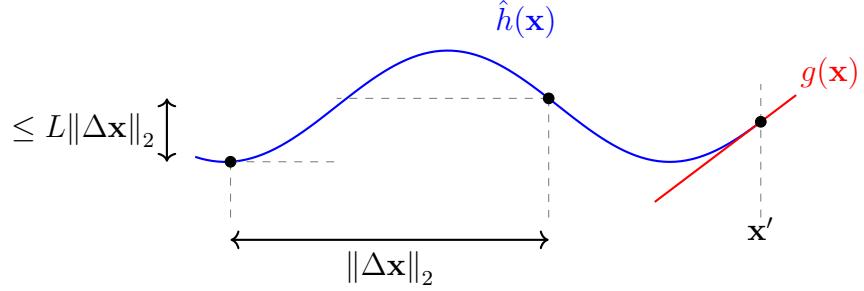


Fig. 111. A trained model  $\hat{h}(\mathbf{x})$  that is differentiable at a point  $\mathbf{x}'$  can be locally approximated by a linear map  $g \in \mathcal{H}^{(d)}$ . This local approximation is determined by the gradient  $\nabla \hat{h}(\mathbf{x}')$ .

See also: model, hypothesis space, linear map, interpretability, LIME.

**linear regression** Linear regression methods learn a linear hypothesis map

$h^{(\mathbf{w})}(\mathbf{x}) = \mathbf{w}^T \mathbf{x}$  that is used to predict the numeric label  $y \in \mathbb{R}$  of a data point based on its numeric feature vectors  $\mathbf{x} = (x_1, \dots, x_d)^T \in \mathbb{R}^d$ .

The least-squares variant of linear regression measures the quality of a linear hypothesis map via the average squared error loss incurred on a training set

$$(\mathbf{x}^{(1)}, y^{(1)}) , \dots , (\mathbf{x}^{(m)}, y^{(m)}) .$$

As an instance of ERM, linear (least-squares) regression learns the model parameters  $\mathbf{w}$  by solving the following optimization problem:

$$\min_{\mathbf{w} \in \mathbb{R}^d} \frac{1}{m} \sum_{r=1}^m (y^{(r)} - \mathbf{w}^T \mathbf{x}^{(r)})^2 .$$

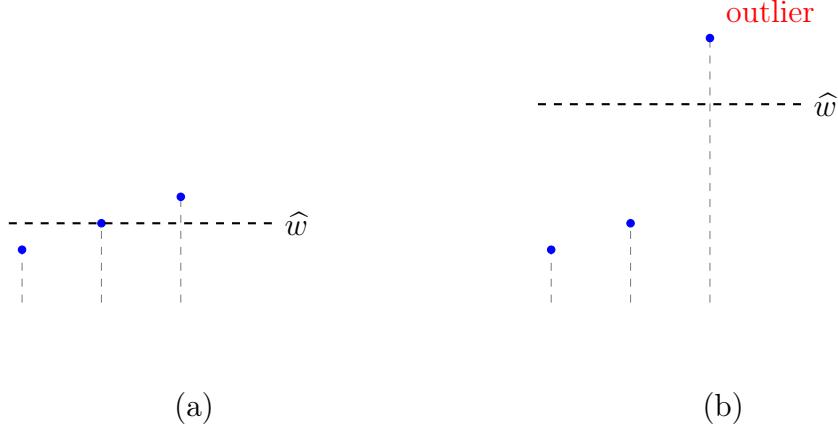


Fig. 112. For a linear model with  $d = 1$  and using the trivial feature  $x = 1$  for any data point, linear regression reduces to computing the average  $\hat{w} = (1/m) \sum_{r=1}^m y^{(r)}$ . (a) A clean training set and the resulting parameter (given by the average). (b) A perturbed dataset (including an outlier) and the resulting parameter.

We can rewrite the above optimization problem more compactly using the feature matrix  $\mathbf{X} = (\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)})^T \in \mathbb{R}^{m \times d}$  and the label vector  $\mathbf{y} = (y^{(1)}, \dots, y^{(m)})^T \in \mathbb{R}^m$ . This allows us to rewrite the above optimization problem as

$$\min_{\mathbf{w} \in \mathbb{R}^d} \frac{1}{m} \|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2.$$

By the zero-gradient condition, a necessary and sufficient condition for a vector  $\hat{\mathbf{w}}$  to be a solution to the above optimization problem is the linear system of equations [36]:

$$\mathbf{X}^T \mathbf{X} \widehat{\mathbf{w}} = \mathbf{X}^T \mathbf{y}. \quad (15)$$

Instead of solving (15) directly (via computing the inverse matrix or pseudoinverse), many ML methods use variants of GD to construct a sequence  $\mathbf{w}^{(0)}, \mathbf{w}^{(1)}, \dots$  of increasingly accurate approximations of a solution  $\hat{\mathbf{w}}$  to (15). These gradient-based methods can be interpreted as a fixed-point iteration for the following reformulation of (15):

$$(\mathbf{I} - \mathbf{A}\mathbf{X}^T\mathbf{X})\hat{\mathbf{w}} + \mathbf{X}^T\mathbf{y} = \hat{\mathbf{w}} \quad \text{with some invertible matrix } \mathbf{A}.$$

This equation is solved by a vector  $\hat{\mathbf{w}}$  if and only if this vector also solves (15). The optimality condition (15) is also useful for the study of the stability of linear regression. Ideally, we would like the solutions of (15) to be insensitive to small perturbations of the training set. We can capture these perturbations via a perturbed feature matrix  $\tilde{\mathbf{X}} = \mathbf{X} + \Delta\mathbf{X}$  and perturbed label vector  $\tilde{\mathbf{y}} = \mathbf{y} + \Delta\mathbf{y}$ . Here,  $\Delta\mathbf{X}$  and  $\Delta\mathbf{y}$  represent small perturbations to the feature vectors and labels of the data points in the original training set. Matrix perturbation theory allows us to evaluate how much the solutions of the perturbed linear regression problem [3, Sec. 2.6]

$$\tilde{\mathbf{X}}^T\tilde{\mathbf{X}}\tilde{\mathbf{w}} = \tilde{\mathbf{X}}^T\tilde{\mathbf{y}}$$

deviate from the solutions  $\tilde{\mathbf{w}}$  of the original linear regression problem.

See also: regression, ERM, linear model.

**Lloyd's algorithm** Lloyd's algorithm [178] is an iterative optimization method for finding cluster centroids that are approximately optimal for the  $k$ -means objective function. Lloyd's algorithm alternates between: 1) updating the cluster assignment of each data point based on the nearest current cluster centroid; and 2) recalculating the cluster centroids

given the updated cluster assignments [178].

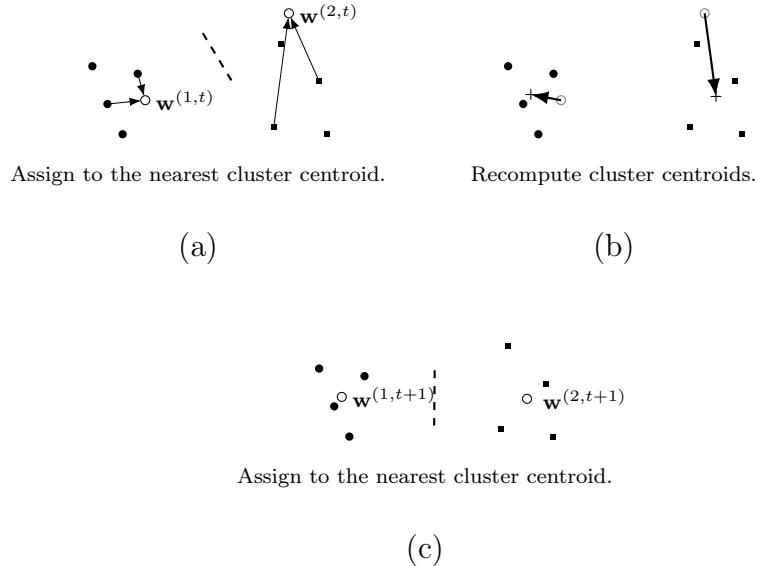


Fig. 113. Lloyd's algorithm alternates between (a) and (c) assigning data points to the nearest cluster centroid and, in turn, (b) recomputing the cluster centroids based on the new cluster assignments.

See also: cluster centroid,  $k$ -means, clustering.

**local dataset** The concept of a local dataset is in between the concept of a data point and a dataset. A local dataset consists of several individual data points characterized by features and labels. In contrast to a single dataset used in basic ML methods, a local dataset is also related to other local datasets via different notions of similarity. These similarities might arise from probabilistic models or communication infrastructure and are encoded in the edges of an FL network.

See also: dataset, data point, feature, label, ML, probabilistic model, FL network.

**local interpretable model-agnostic explanations (LIME)** Consider a trained model (or learned hypothesis)  $\hat{h} \in \mathcal{H}$ , which maps the feature vector of a data point to the prediction  $\hat{y} = \hat{h}$ . LIME is a technique for explaining the behavior of  $\hat{h}$  locally around a data point with feature vector  $\mathbf{x}^{(0)}$  [149]. The explanation is given in the form of a local approximation  $g \in \mathcal{H}'$  of  $\hat{h}$  (see Fig. 114). This approximation can be obtained by an instance of ERM with a carefully designed training set. In particular, the training set consists of data points with feature vectors centered around  $\mathbf{x}^{(0)}$  and the (pseudo-)label  $\hat{h}(\mathbf{x})$ . Note that we can use a different model  $\mathcal{H}'$  for the approximation from the original model  $\mathcal{H}$ . For example, we can use a decision tree to locally approximate a deep net. Another widely used choice for  $\mathcal{H}'$  is the linear model.

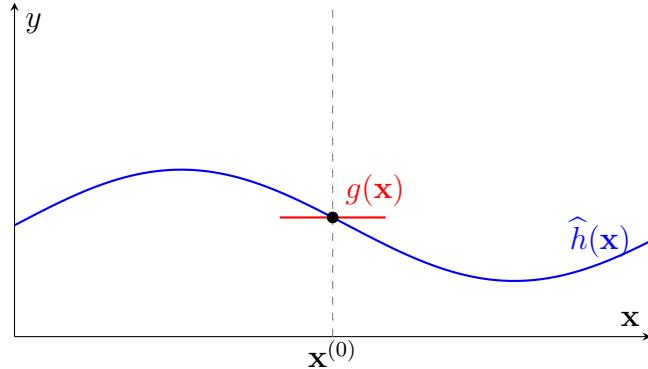


Fig. 114. To explain a trained model  $\hat{h} \in \mathcal{H}$ , around a given feature vector  $\mathbf{x}^{(0)}$ , we can use a local approximation  $g \in \mathcal{H}'$ .

See also: model, explanation, ERM, training set, label, decision tree, deep net, linear model.

**local model** Consider a collection of devices that are represented as nodes  $\mathcal{V}$  of an FL network. A local model  $\mathcal{H}^{(i)}$  is a hypothesis space assigned to a node  $i \in \mathcal{V}$ . Different nodes can have different hypothesis spaces, i.e., in general,  $\mathcal{H}^{(i)} \neq \mathcal{H}^{(i')}$  for different nodes  $i, i' \in \mathcal{V}$ .

See also: device, FL network, model, hypothesis space.

**locally weighted learning (LWL)** LWL is a nonparametric ML method that makes predictions for a given data point  $\mathbf{z}$  based on a weighted combination of the labels of the data points in the training set. The weights reflect the similarity between the data point  $\mathbf{z}$  and each data point in the training set. A widely used instance of LWL is the k-nearest neighbors regression (KNNR) [8], [22], [179].

See also: nonparametric, data point, training set, KNNR.

**logistic loss** Consider a data point characterized by features  $\mathbf{x}$  and a binary label  $y \in \{-1, 1\}$ . We use a real-valued hypothesis  $h$  to predict the label  $y$  from the features  $\mathbf{x}$ . The logistic loss incurred by this prediction is defined as [22]

$$L((\mathbf{x}, y), h) := \log(1 + \exp(-yh(\mathbf{x}))). \quad (16)$$

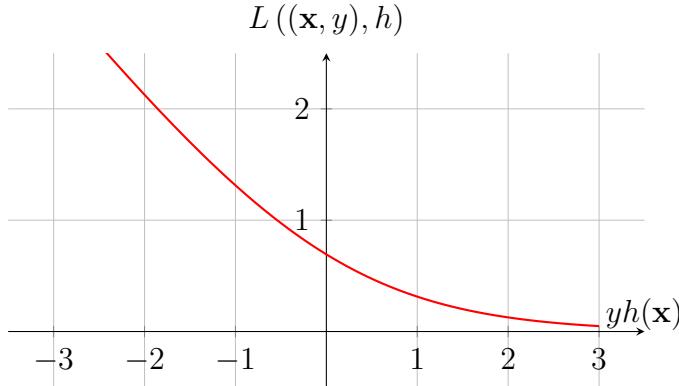


Fig. 115. The logistic loss incurred by the prediction  $h(\mathbf{x}) \in \mathbb{R}$  for a data point with label  $y \in \{-1, 1\}$ .

Note that the expression (16) for the logistic loss applies only for the label space  $\mathcal{Y} = \{-1, 1\}$  and when using the thresholding rule (10).

See also: loss, classification, classifier, linear model.

**logistic regression** Logistic regression learns a linear hypothesis map (or classifier)  $h(\mathbf{x}) = \mathbf{w}^T \mathbf{x}$  to predict a binary label  $y$  based on the numeric feature vector  $\mathbf{x}$  of a data point [22], [83]. The quality of a linear hypothesis map is measured by the average logistic loss on some labeled data points (i.e., the training set).

See also: regression, hypothesis, map, classifier, label, feature vector, data point, logistic loss, labeled data point, training set.

**loss** ML methods use a loss function  $L(\mathbf{z}, h)$  to measure the error incurred by applying a specific hypothesis to a specific data point. With a slight abuse of notation, we use the term loss for both the loss function  $L$

itself and the specific value  $L(\mathbf{z}, h)$  for a data point  $\mathbf{z}$  and hypothesis  $h$ .

See also: loss function, empirical risk.

**loss function** A loss function is a map:

$$L : \mathcal{X} \times \mathcal{Y} \times \mathcal{H} \rightarrow \mathbb{R}_+ : ((\mathbf{x}, y), h) \mapsto L((\mathbf{x}, y), h).$$

It assigns a nonnegative real number (i.e., the loss)  $L((\mathbf{x}, y), h)$  to a pair that consists of a data point, with features  $\mathbf{x}$  and label  $y$ , and a hypothesis  $h \in \mathcal{H}$ . The value  $L((\mathbf{x}, y), h)$  quantifies the discrepancy between the true label  $y$  and the prediction  $h(\mathbf{x})$ . Lower (closer to zero) values  $L((\mathbf{x}, y), h)$  indicate a smaller discrepancy between prediction  $h(\mathbf{x})$  and label  $y$ . Fig. 116 depicts a loss function for a given data point, with features  $\mathbf{x}$  and label  $y$ , as a function of the hypothesis  $h \in \mathcal{H}$ .

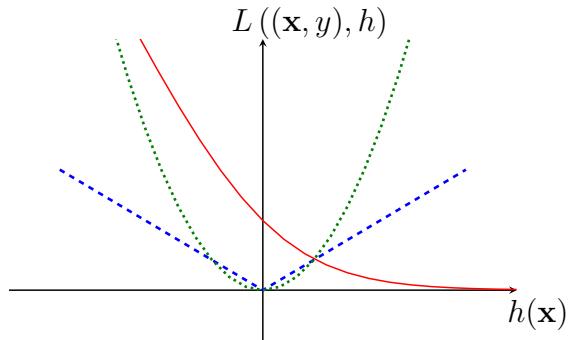


Fig. 116. Some loss function  $L((\mathbf{x}, y), h)$  for a fixed data point, with feature vector  $\mathbf{x}$  and label  $y$ , and a varying hypothesis  $h$ . ML methods try to find (or learn) a hypothesis that incurs minimal loss.

See also: loss, label, feature vector, ERM.

**machine learning (ML)** ML methods aim to learn (or find) a useful hypothesis map  $\hat{h} \in \mathcal{H}$  out of a model  $\mathcal{H}$ . The learned  $\hat{h}$  is used to compute a prediction  $\hat{y} = \hat{h}(\mathbf{x})$  for the label  $y$  of a data point. The learning process is guided by a quantitative measure of the loss incurred when the predictions obtained from the learned hypothesis differ from the actual label  $y$ . Different ML methods use different design choices for this quantitative measure (or loss function) as well as different choices for the model and the data points (i.e., their features and labels) [8, Ch. 3].

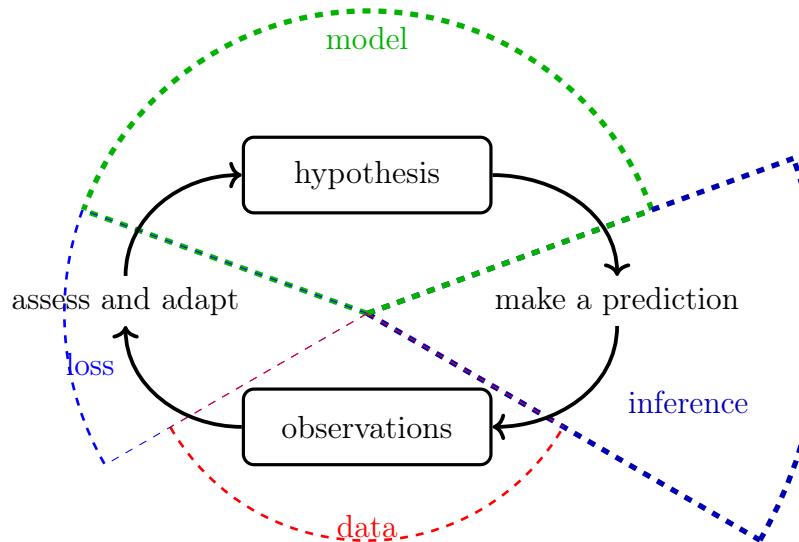


Fig. 117. ML learns a hypothesis out of a model (or hypothesis space) by trying to minimize the loss incurred by the predictions for the labels of data points. The predictions are computed solely from the features of the data points.

Another distinction between ML methods is how they access data points during learning. For example, some methods have access to a complete dataset during training, which allows them to use ERM [8], [180]. In contrast, online learning methods access data sequentially and, in turn, update the learned hypothesis whenever a new data point arrives [138], [139], [181].

See also: model, loss, data, ERM, online learning.

**machine unlearning** Consider an ML method that learns a hypothesis  $\hat{h}$  via ERM on a training set  $\mathcal{D}$ . The learned hypothesis can reveal information about  $\mathcal{D}$ , which is exploited by privacy attacks such as model inversion. Machine unlearning refers to techniques that modify  $\hat{h}$ , so that it is harder to infer properties of individual data points in  $\mathcal{D}$  [182]. Machine unlearning helps to meet legal requirements for privacy protection in AI systems [122].

See also: model inversion, privacy protection, general data protection regulation (GDPR).

**mean absolute error (MAE)** The MAE of a hypothesis is the average absolute error loss computed over a given dataset. In theoretical analyses, MAE also denotes the expected absolute error loss, i.e., the corresponding risk.

See also: absolute error loss, risk.

**mean squared error (MSE)** The MSE of a hypothesis is the average squared error loss computed over a given dataset. In theoretical analyses, MSE also denotes the expected squared error loss, i.e., the corresponding

risk.

See also: squared error loss, risk.

**mean squared estimation error (MSEE)** Consider an ML method that learns model parameters  $\hat{\mathbf{w}}$  based on some dataset  $\mathcal{D}$ . If we interpret the data points in  $\mathcal{D}$  as i.i.d. realizations of an RV  $\mathbf{z}$ , we define the estimation error  $\Delta\mathbf{w} := \hat{\mathbf{w}} - \bar{\mathbf{w}}$ . Here,  $\bar{\mathbf{w}}$  denotes the true model parameters of the probability distribution of  $\mathbf{z}$ . The MSEE is defined as the expectation  $\mathbb{E}\{\|\Delta\mathbf{w}\|^2\}$  of the squared Euclidean norm of the estimation error [42], [43].

See also: RV, estimation error, probabilistic model, squared error loss.

**membership inference attack** Consider an ML method that learns a hypothesis via ERM on a training set. Membership inference attack is a form of privacy attack where an adversary tries to determine whether a particular data point was part of the training set. The attacker typically queries  $\hat{h}$  with candidate feature vectors  $\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(B)}$ , and infers the membership status of a given data point based on the predictions  $\hat{h}(\mathbf{x}^{(1)}), \dots, \hat{h}(\mathbf{x}^{(B)})$  [183].

See also: attack, privacy attack.

**missing data** Consider a dataset constituted by data points collected via some physical device. Due to imperfections and failures, some of the feature or label values of data points might be corrupted or simply missing. Data imputation aims to estimate these missing values [184]. We can interpret data imputation as an ML problem where the label of a data point is the value of the corrupted feature.

See also: feature, label.

**model (machine learning)** The study and design of ML methods is often based on a mathematical model [185]. One of the most widely used examples of a mathematical model for ML is a hypothesis space. A hypothesis space consists of hypothesis maps that are used by an ML method to predict labels from the features of data points. Another important type of mathematical model is a probabilistic model, which consists of probability distributions that describe how data points are generated. Unless stated otherwise, we use the term model to refer specifically to the hypothesis space underlying an ML method. We illustrate one example of a hypothesis space and a probabilistic model in Fig. 118.

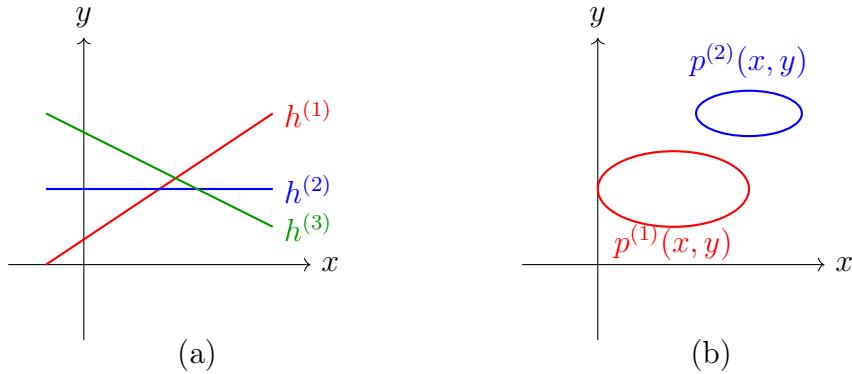


Fig. 118. Two types of mathematical models used in ML. (a) A hypothesis space consisting of three linear maps. (b) A probabilistic model consisting of a probability distribution over the plane spanned by the possible values of the feature and label of a data point.

See also: hypothesis space, probabilistic model, probability distribution.

**model inversion** A model inversion is a form of privacy attack on an ML system. An adversary seeks to infer sensitive attributes of individual data points by exploiting partial access to a trained model  $\hat{h} \in \mathcal{H}$ . This access typically consists of querying the model for predictions  $\hat{h}(\mathbf{x})$  using carefully chosen inputs. Basic model inversion techniques have been demonstrated in the context of facial image classification, where images are reconstructed using the (gradient of) model outputs combined with auxiliary information such as a person's name [186] (see Fig. 119).

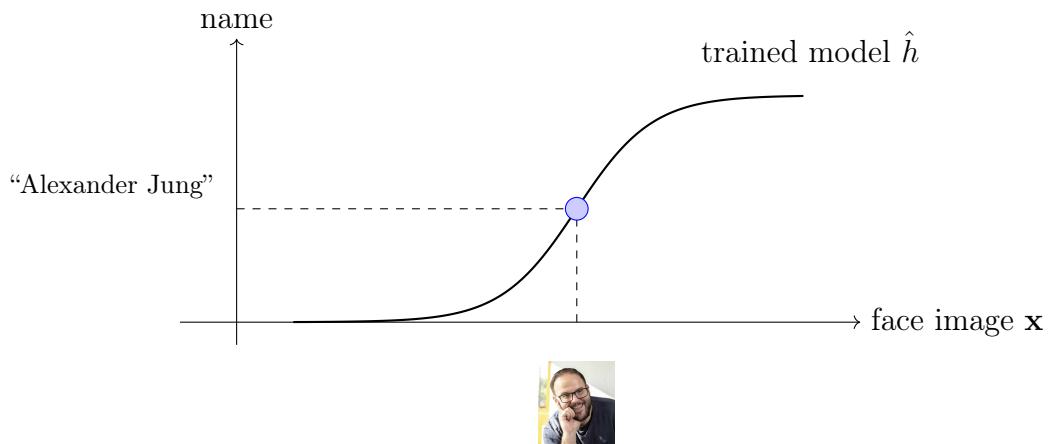


Fig. 119. Model inversion techniques implemented in the context of facial image classification.

See also: model, privacy attack, ML, sensitive attribute, data point, prediction, classification, gradient, trustworthy AI, privacy protection.

**model parallelism** Model parallelism refers to a particular class of dis-

tributed optimization methods used to train an ML model. Here, different devices store and process disjoint subsets of the model parameters. This approach contrasts with data parallelism, where each device maintains a full replica of the model parameters while processing disjoint subsets of the global dataset. Model parallelism allows us to train an ML model whose parameters cannot fit into the memory of a single device. One key application of model parallelism is the training of extremely large ANNs, such as transformer models with billions of model parameters.

See also: VFL.

**model parameter** The elements of a parametric model are specified by quantities that are referred to as model parameters. In the context of ML, a parametric model consists of hypothesis maps that are specified by a list of model parameters  $w_1, w_2, \dots, w_d$ . It is often convenient to stack these model parameters into a vector  $\mathbf{w} = (w_1, \dots, w_d)^T \in \mathbb{R}^d$ .

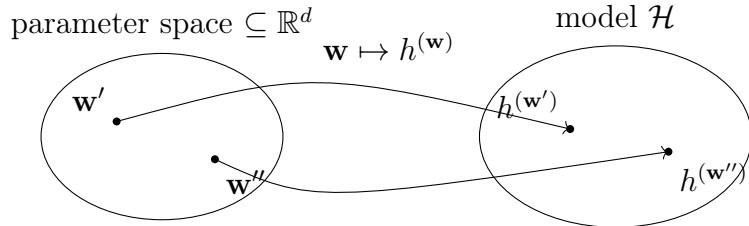


Fig. 120. The model parameters  $\mathbf{w}$  select a well-defined hypothesis  $h^{(\mathbf{w})}$  out of the model  $\mathcal{H}$ .

We can think of model parameters as an identifier for a hypothesis map, similar to how a social security number identifies a person.

See also: model, parameter, hypothesis, map.

**model selection** In ML, model selection refers to the process of choosing between different candidate models. In its most basic form, model selection amounts to the following steps [8, Ch. 6]:

- 1) training each candidate model;
- 2) computing the validation error for each trained model;
- 3) choosing the model with the smallest validation error.

See also: ML, model, training, validation error.

**multi-label classification** Multi-label classification problems and methods use data points that are characterized by several labels. As an example, consider a data point representing a picture with two labels. One label indicates the presence of a human in this picture and another label indicates the presence of a car.

See also: label, classification, data point.

**multilayer perceptron (MLP)** An MLP is a type of ANN that is composed of multiple layers of affine transformations followed by pointwise nonlinear activation functions [30]. Mathematically, an MLP implements a composition of nonlinear operators of the form  $\mathbf{w} \mapsto \sigma(\mathbf{Aw} + \mathbf{b})$ , where  $\mathbf{A}$  and  $\mathbf{b}$  are learnable model parameters and the activation function  $\sigma$  is applied elementwise.

See also: ANN, activation function.

**multitask learning** Multitask learning aims to leverage relations between different learning tasks. Consider two learning tasks obtained from the

same dataset of webcam snapshots. The first task is to predict the presence of a human, while the second task is to predict the presence of a car. It may be useful to use the same deep net structure for both tasks and only allow the weights of the final output layer to be different.

See also: learning task, dataset, deep net, weight, layer.

**mutual information (MI)** The MI  $I(\mathbf{x}; y)$  between two RVs  $\mathbf{x}, y$  defined on the same probability space is given by [35]

$$I(\mathbf{x}; y) := \mathbb{E} \left\{ \log \frac{p(\mathbf{x}, y)}{p(\mathbf{x})p(y)} \right\}.$$

It is a measure of how well we can estimate  $y$  based solely on  $\mathbf{x}$ . A large value of  $I(\mathbf{x}; y)$  indicates that  $y$  can be well predicted solely from  $\mathbf{x}$ . This prediction could be obtained by a hypothesis learned by an ERM-based ML method.

See also: RV, probability space, prediction, hypothesis, ERM, ML.

**natural language processing (NLP)** NLP studies ML methods for the analysis and generation of human language. Typical NLP tasks include text classification, machine translation, sentiment analysis, and question answering. Modern NLP systems represent language as sequences of tokens and train models that capture contextual dependencies, such as attention-based methods.

See also: token, attention.

**nearest neighbor (NN)** NN methods learn a hypothesis  $h : \mathcal{X} \rightarrow \mathcal{Y}$  whose function value  $h(\mathbf{x})$  is solely determined by the NNs within a given dataset. Different methods use different metrics for determining the

NNs. If data points are characterized by numeric feature vectors, we can use their Euclidean distances as the metric.

See also: metric, neighbor.

**neighbor** A neighbor of a node  $i \in \mathcal{V}$  within an undirected graph is a node  $i' \in \mathcal{V} \setminus \{i\}$  that is connected via an edge to node  $i$ .

See also: undirected graph, connected.

**nested cross-validation (nested CV)** Nested CV is a method of extending the  $k$ -fold CV from training and validation sets to also cover the test set. Instead of simply choosing a single test set from the data, two loops are run, i.e., the outer and the inner loop. The outer loop uses  $k$ -fold CV to separate a test set from the data, and the inner loop again uses  $k$ -fold CV to separate the remaining data into training and validation sets. Doing this decreases the variance and bias in the results.

See also:  $k$ -fold CV, leave-one-out cross-validation (LOO-CV), validation, validation error.

**network Lasso** The network Lasso is a special case of GTVMin, obtained by using a norm-based discrepancy measure for comparing local model parameters [187]. It can also be viewed as a generalization of the Lasso method to datasets and models with an intrinsic network structure.

See also: Lasso, GTVMin, GTV, RERM, regularizer.

**networked data** Networked data consist of local datasets that are related by some notion of pairwise similarity. We can represent networked data using a graph whose nodes carry local datasets and whose edges encode pairwise similarities. An example of networked data can be found in FL

applications where local datasets are generated by spatially distributed devices.

See also: data, local dataset, graph, FL, device.

**networked exponential families (nExpFam)** nExpFam is a collection of exponential families, each of them assigned to a node of an FL network. The model parameters are coupled via the network structure by requiring them to have a small GTV [188].

See also: FL network, model parameter, GTV.

**networked federated learning (NFL)** NFL refers to methods that learn personalized models in a distributed fashion. These methods learn from local datasets that are related by an intrinsic network structure.

See also: model, local dataset, FL.

**networked model** A networked model over an FL network  $\mathcal{G} = (\mathcal{V}, \mathcal{E})$  assigns a local model (i.e., a hypothesis space) to each node  $i \in \mathcal{V}$  of the FL network  $\mathcal{G}$ .

See also: model, FL network, local model, hypothesis space.

**nonparametric** Nonparametric ML methods do not assume a fixed model structure with a finite number of model parameters. Instead, the complexity of the learned hypothesis can grow with the number of data points in the training set [8], [22]. Examples of nonparametric ML methods include LWL and GPs.

See also: ML, LWL, GP.

**normal equations** Consider linear least squares, which learns the parame-

ters of a linear model by minimizing the average squared error loss on a training set  $\mathcal{D}$ . The learned parameters  $\hat{\mathbf{w}}$  are characterized by the linear system of equations, i.e.,

$$\mathbf{X}^T \mathbf{X} \hat{\mathbf{w}} = \mathbf{X}^T \mathbf{y}. \quad (17)$$

Here,  $\mathbf{X} = (\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)})^T \in \mathbb{R}^{m \times d}$  is the feature matrix of  $\mathcal{D}$  and  $\mathbf{y} = (y^{(1)}, \dots, y^{(m)})^T \in \mathbb{R}^m$  is the label vector of  $\mathcal{D}$ . This linear system of equations is referred to as normal equations, as it amounts to an orthogonality condition. Indeed, (17) can be rewritten as

$$\mathbf{X}^T (\mathbf{X} \hat{\mathbf{w}} - \mathbf{y}) = \mathbf{0}.$$

This means that the prediction error vector  $\mathbf{X} \hat{\mathbf{w}} - \mathbf{y}$  is orthogonal to the columns of  $\mathbf{X}$  and, in turn, to the subspace  $\text{span}(\mathbf{X})$  spanned by them.

See also: linear least squares, model parameter.

**objective function** An objective function is a map that assigns a numeric objective value  $f(\mathbf{w})$  to each choice  $\mathbf{w}$  of some variable that we want to optimize (see Fig. 121). In the context of ML, the optimization variable could be the model parameters of a hypothesis  $h^{(\mathbf{w})}$ . Common objective functions include the risk (i.e., expected loss) or the empirical risk (i.e., average loss over a training set). ML methods apply optimization techniques, such as gradient-based methods, to find the choice  $\mathbf{w}$  with the optimal value (e.g., the minimum or the maximum) of the objective function.

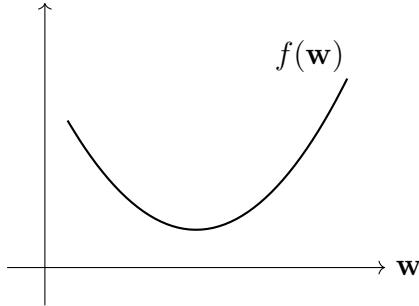


Fig. 121. An objective function maps each possible value  $\mathbf{w}$  of an optimization variable, such as the model parameters of an ML model, to a value  $f(\mathbf{w})$  that measures the usefulness of  $\mathbf{w}$ .

See also: loss, empirical risk, ERM, optimization problem.

**online algorithm** An online algorithm processes input data incrementally, receiving data points sequentially and making decisions or producing outputs (or decisions) immediately without having access to the entire input in advance [138], [139]. Unlike an offline algorithm, which has the entire input available from the start, an online algorithm must handle uncertainty about future inputs and cannot revise past decisions. Similar to an offline algorithm, we represent an online algorithm formally as a collection of possible executions. However, the execution sequence for an online algorithm has a distinct structure as follows:

$$\text{in}_1, s_1, \text{out}_1, \text{in}_2, s_2, \text{out}_2, \dots, \text{in}_T, s_T, \text{out}_T.$$

Each execution begins from an initial state (i.e.,  $\text{in}_1$ ) and proceeds through alternating computational steps, outputs (or decisions), and inputs. Specifically, at step  $t$ , the algorithm performs a computational

step  $s_t$ , generates an output  $\text{out}_t$ , and then subsequently receives the next input (data point)  $\text{in}_{t+1}$ . A notable example of an online algorithm in ML is online gradient descent (online GD), which incrementally updates model parameters as new data points arrive.

See also: algorithm, data, data point, uncertainty, ML, online GD, model parameter, online learning.

**online gradient descent (online GD)** Consider an ML method that learns model parameters  $\mathbf{w}$  from some parameter space  $\mathcal{W} \subseteq \mathbb{R}^d$ . The learning process uses data points  $\mathbf{z}^{(t)}$  that arrive at consecutive time instants  $t = 1, 2, \dots$ . Let us interpret the (generation of) data points  $\mathbf{z}^{(t)}$  as i.i.d. RVs with a common probability distribution  $\mathbb{P}^{(\mathbf{z})}$ . The risk  $\mathbb{E}\{L(\mathbf{z}, \mathbf{w})\}$  of a hypothesis  $h^{(\mathbf{w})}$  can then (under mild conditions) be obtained as the limit:

$$\lim_{T \rightarrow \infty} \frac{1}{T} \sum_{t=1}^T L(\mathbf{z}^{(t)}, \mathbf{w}).$$

We might use this limit as the objective function for learning the model parameters  $\mathbf{w}$ . Unfortunately, the above limit can only be evaluated if we wait infinitely long in order to collect all data points. However, many ML applications require methods that learn online, i.e., as soon as a new data point  $\mathbf{z}^{(t)}$  arrives at time  $t$ , we update the current model parameters  $\mathbf{w}^{(t)}$ . Note that the new data point  $\mathbf{z}^{(t)}$  contributes the component  $L(\mathbf{z}^{(t)}, \mathbf{w})$  to the risk. As its name suggests, online GD updates  $\mathbf{w}^{(t)}$  via a (projected) gradient step such that

$$\mathbf{w}^{(t+1)} := P_{\mathcal{W}}(\mathbf{w}^{(t)} - \eta_t \nabla_{\mathbf{w}} L(\mathbf{z}^{(t)}, \mathbf{w})). \quad (18)$$

Note that (18) is a gradient step for the current component  $L(\mathbf{z}^{(t)}, \cdot)$

of the risk. The update (18) ignores all previous components  $L(\mathbf{z}^{(t')}, \cdot)$  for  $t' < t$ . It might therefore happen that, compared with  $\mathbf{w}^{(t)}$ , the updated model parameters  $\mathbf{w}^{(t+1)}$  increase the retrospective average loss  $\sum_{t'=1}^{t-1} L(\mathbf{z}^{(t')}, \cdot)$ . However, for a suitably chosen learning rate  $\eta_t$ , online GD can be shown to be optimal in practically relevant settings. By optimal, we mean that the model parameters  $\mathbf{w}^{(T+1)}$  delivered by online GD after observing  $T$  data points  $\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(T)}$  are at least as good as those delivered by any other learning method [139], [189].

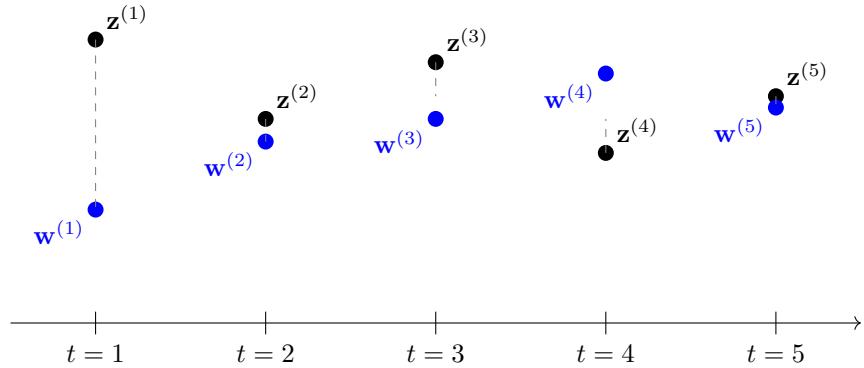


Fig. 122. An instance of online GD that updates the model parameters  $\mathbf{w}^{(t)}$  using the data point  $\mathbf{z}^{(t)} = x^{(t)}$  arriving at time  $t$ . This instance uses the squared error loss  $L(\mathbf{z}^{(t)}, w) = (x^{(t)} - w)^2$ .

See also: objective function, GD, gradient step, online learning.

**online learning** Some ML methods are designed to process data in a sequential manner, updating their model parameters one at a time, as new data points become available. A typical example is time-series data, such as daily minimum and maximum temperatures recorded by

an FMI weather station. These values form a chronological sequence of observations. During each time step  $t$ , online learning methods update (or refine) the current hypothesis  $h^{(t)}$  (or model parameters  $\mathbf{w}^{(t)}$ ) based on the newly observed data point  $\mathbf{z}^{(t)}$ .

See also: online GD, online algorithm.

**optimism in the face of uncertainty** ML methods learn model parameters  $\mathbf{w}$  according to some performance criterion  $\bar{f}(\mathbf{w})$ . However, they usually cannot access  $\bar{f}(\mathbf{w})$  directly but rely on an estimate (or approximation)  $f(\mathbf{w})$  of  $\bar{f}(\mathbf{w})$ . As a case in point, ERM-based methods use the average loss on a given dataset (i.e., the training set) as an estimate for the risk of a hypothesis. Using a probabilistic model, one can construct a confidence interval  $[l^{(\mathbf{w})}, u^{(\mathbf{w})}]$  for each choice  $\mathbf{w}$  for the model parameters. One simple construction is  $l^{(\mathbf{w})} := f(\mathbf{w}) - \sigma/2$ ,  $u^{(\mathbf{w})} := f(\mathbf{w}) + \sigma/2$ , with  $\sigma$  being a measure of the (expected) deviation of  $f(\mathbf{w})$  from  $\bar{f}(\mathbf{w})$ . We can also use other constructions for this interval as long as they ensure that  $\bar{f}(\mathbf{w}) \in [l^{(\mathbf{w})}, u^{(\mathbf{w})}]$  with a sufficiently high probability. An optimist chooses the model parameters according to the most favorable—yet still plausible—value  $\tilde{f}(\mathbf{w}) := l^{(\mathbf{w})}$  of the performance criterion (see Fig. 123). Two examples of ML methods that use such an optimistic construction of an objective function are SRM [155, Ch. 11] and upper confidence bound (UCB) methods for sequential decision making [190, Sec. 2.2].

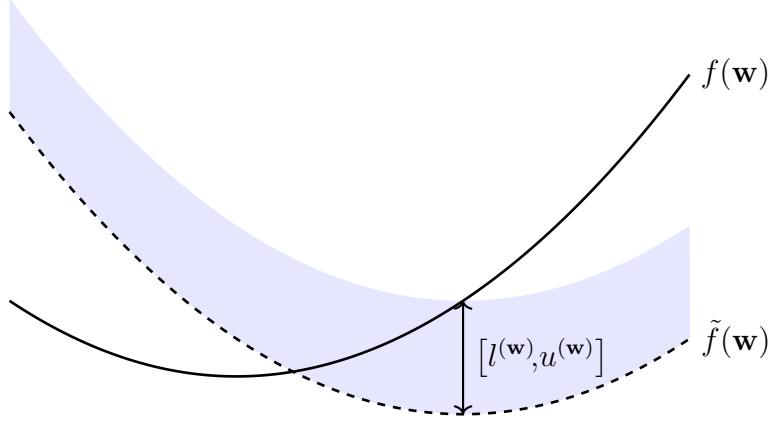


Fig. 123. ML methods learn model parameters  $\mathbf{w}$  by using some estimate of  $f(\mathbf{w})$  for the ultimate performance criterion  $\bar{f}(\mathbf{w})$ . Using a probabilistic model, one can use  $f(\mathbf{w})$  to construct confidence intervals  $[l^{(\mathbf{w})}, u^{(\mathbf{w})}]$ , which contain  $\bar{f}(\mathbf{w})$  with high probability. The best plausible performance measure for a specific choice  $\mathbf{w}$  of model parameters is  $\tilde{f}(\mathbf{w}) := l^{(\mathbf{w})}$ .

See also: objective function, UCB, optimization method, gradient-based method.

**outlier** Many ML methods are motivated by the i.i.d. assumption, which interprets data points as realizations of i.i.d. RVs with a common probability distribution. The i.i.d. assumption is useful for applications where the statistical properties of the data generation process are stationary (or time-invariant) [95]. However, in some applications, the data consist of a majority of regular data points that conform with the i.i.d. assumption as well as a small number of data points that have fundamentally different statistical properties compared with the regular

data points. We refer to a data point that substantially deviates from the statistical properties of most data points as an outlier. Different methods for outlier detection use different measures of this deviation. Statistical learning theory studies fundamental limits on the ability to mitigate outliers reliably [191], [192].

See also: robustness, stability, Huber regression, probabilistic model.

**output** The term output is sometimes used as a synonym for the label of a data point [46].

See also: label, data point.

**output vector** The term output vector is used as a synonym for the label vector of a dataset [46].

See also: output, label vector, dataset.

**overfitting** Consider an ML method that uses ERM to learn a hypothesis with the minimum empirical risk on a given training set. Such a method is overfitting the training set if it learns a hypothesis with an empirical risk on the training set that is significantly smaller than the average loss outside the training set. In other words, if an ML method overfits, it has a large generalization gap.

See also: ERM, generalization, validation, generalization gap.

**parameter** The parameter of an ML model is a tunable (i.e., learnable or adjustable) quantity that allows us to choose between different hypothesis maps. For example, the linear model  $\mathcal{H} := \{h^{(\mathbf{w})} : h^{(\mathbf{w})}(x) = w_1x + w_2\}$  consists of all hypothesis maps  $h^{(\mathbf{w})}(x) = w_1x + w_2$  with

a particular choice for the parameters  $\mathbf{w} = (w_1, w_2)^T \in \mathbb{R}^2$ . Another example of a model parameter is the weights assigned to a connection between two neurons of an ANN.

See also: ML, model, hypothesis, map, linear model, weight, ANN.

**parameter space** The parameter space  $\mathcal{W}$  of an ML model  $\mathcal{H}$  is the set of all feasible choices for the model parameters (see Fig. 124). Many important ML methods use a model that is parameterized by vectors of the Euclidean space  $\mathbb{R}^d$ . Two widely used examples of parameterized models are linear models and deep nets. The parameter space is then often a subset  $\mathcal{W} \subseteq \mathbb{R}^d$ , e.g., all vectors  $\mathbf{w} \in \mathbb{R}^d$  with a norm smaller than one.

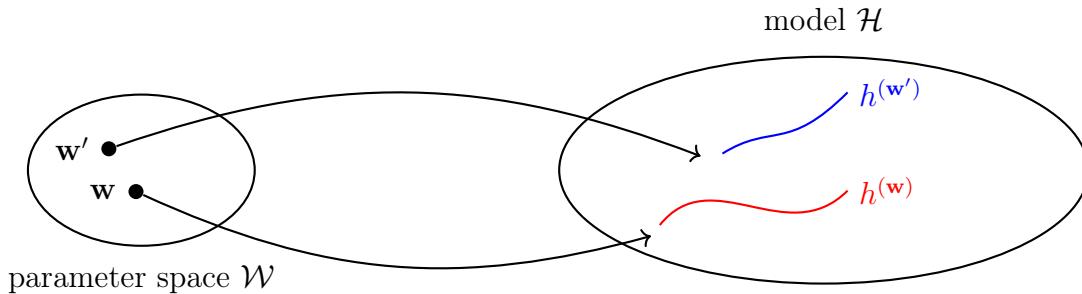


Fig. 124. The parameter space  $\mathcal{W}$  of an ML model  $\mathcal{H}$  consists of all feasible choices for the model parameters. Each choice  $\mathbf{w}$  for the model parameters selects a hypothesis map  $h^{(\mathbf{w})} \in \mathcal{H}$ .

See also: parameter, model, model parameter.

**parametric model** A parametric model is a mathematical model characterized by a finite set of variable quantities called parameters. An

important example is the probabilistic model consisting, for a given dimension  $d$ , of all multivariate normal distributions (on sample space  $\mathbb{R}^d$ ) with some mean vector  $\mu \in \mathbb{R}^d$  and covariance matrix  $\mathbf{C} \in \mathbb{R}^{d \times d}$ .

In the context of ML, a parametric model defines a hypothesis space  $\mathcal{H}$  parameterized by a finite number of model parameters. Each hypothesis  $h \in \mathcal{H}$  is uniquely identified by a list of model parameters  $w_1, w_2, \dots, w_d$  (see Fig. 124). We can stack these parameters into a vector  $\mathbf{w} \in \mathbb{R}^d$ . Two widely used examples of parametric models are the linear model and the ANN. The corresponding parameter space is typically a subset of  $\mathbb{R}^d$ .

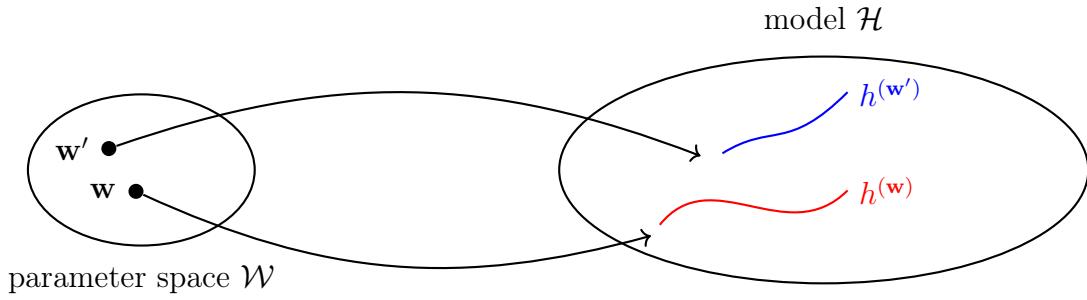


Fig. 125. The parameter space  $\mathcal{W}$  of an ML model  $\mathcal{H}$  consists of all feasible choices for the model parameters. Each choice  $\mathbf{w}$  for the model parameters selects a hypothesis map  $h^{(\mathbf{w})} \in \mathcal{H}$ .

See also: model, model parameter, parameter space.

**penalty term** Consider an ERM-based ML method that learns model parameters  $\mathbf{w}$  by minimizing the average loss (or empirical risk)  $\widehat{L}(\mathbf{w}|\mathcal{D})$  on a training set  $\mathcal{D}$ . To avoid overfitting and control the generalization

gap, it is common to augment the objective function  $\hat{L}(\mathbf{w}|\mathcal{D})$  with a penalty term  $\alpha\mathcal{R}\{\mathbf{w}\}$ . We refer to the resulting modified ERM as RERM.

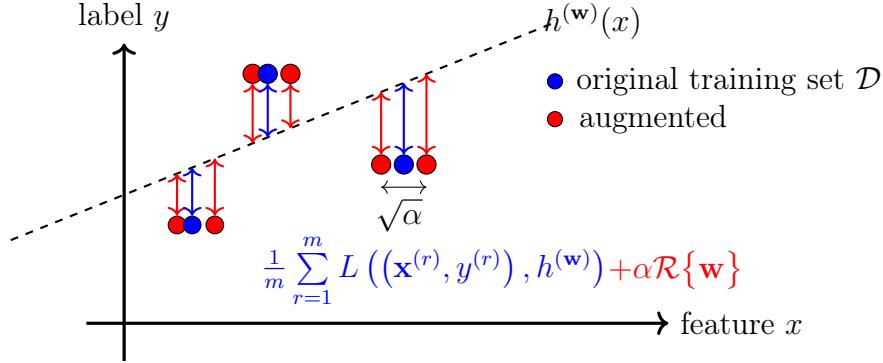


Fig. 126. Adding a penalty term  $\alpha\mathcal{R}\{\mathbf{w}\}$  to the objective function in ERM is equivalent to including perturbations of the training set  $\mathcal{D}$  during training. The regularization parameter  $\alpha$  controls the extent of the perturbations.

The penalty term depends only on the model parameters but not on the data points in the training set. For some combinations of model and loss function, the penalty term can be obtained as the average loss incurred on (an infinite number of) perturbed copies of the training set. In other words, adding a penalty term in ERM can be viewed as a form of data augmentation.

See also: overfitting, RERM, regularization, data augmentation.

**perceptron** The perceptron is one of the oldest ML algorithms; it was developed by Frank Rosenblatt in 1957 [193]. The perceptron algorithm works on data with numeric features  $\mathbf{x}_r \in \mathbb{R}^d$  and binary labels

$y_r \in \{-1, 1\}$  and returns a linear classifier. It is guaranteed to find such a linear classifier if the classes are linearly separable. As the algorithm tries to find a hyperplane  $g(\mathbf{x}) := \mathbf{w}^\top \mathbf{x}$  that separates the classes, i.e.,  $\text{sgn}(\mathbf{w}^\top \mathbf{x}_r) = y_r \forall r \in \{1, \dots, m\}$ , at each iteration it selects a sample  $r'$  that is misclassified, i.e.,  $\text{sgn}(\mathbf{w}^\top \mathbf{x}'_{r'}) \neq y_{r'}$ . Then, the weights  $\mathbf{w}$  are updated by  $\mathbf{w} \leftarrow \mathbf{w} + y_{r'} \mathbf{x}_{r'}$ . While this does not guarantee that the sample is now correctly classified, the error margin has decreased.

See also: probabilistic model, maximum likelihood, optimization problem.

**perplexity** The perplexity of an RV  $x$  is defined as  $e^{H_x}$ .

See also: entropy.

**polynomial regression** Polynomial regression is an instance of ERM that learns a polynomial hypothesis map to predict a numeric label based on the numeric features of a data point. For data points characterized by a single numeric feature, polynomial regression uses the hypothesis space  $\mathcal{H}_d^{(\text{poly})} := \{h(x) = \sum_{j=0}^{d-1} x^j w_j\}$ . The quality of a polynomial hypothesis map is measured using the average squared error loss incurred on a set of labeled data points (which we refer to as the training set).

See also: regression, ERM, squared error loss.

**precision** Precision is a metric commonly used in binary classification for the assessment of trained models. It measures the proportion of correctly predicted data points among those with a positive label [194], [195].

See also: metric, classification, confusion matrix, recall.

**prediction** A prediction is an estimate or approximation for some quantity

of interest. ML revolves around learning or finding a hypothesis map  $h$  that reads in the features  $\mathbf{x}$  of a data point and delivers a prediction  $\hat{y} := h(\mathbf{x})$  for its label  $y$ .

See also: ML, hypothesis, map, feature, data point, label.

**predictor** A predictor is a real-valued hypothesis map. Given a data point with features  $\mathbf{x}$ , the value  $h(\mathbf{x}) \in \mathbb{R}$  is used as a prediction for the true numeric label  $y \in \mathbb{R}$  of the data point.

See also: hypothesis, map, data point, feature, prediction, label.

**principal component analysis (PCA)** Consider a dataset

$$\mathcal{D} = \{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$$

consisting of data points characterized by feature vectors  $\mathbf{x}^{(r)} \in \mathbb{R}^d$  for  $r = 1, \dots, m$ . PCA determines, for a given number  $d' < d$ , a linear feature map

$$\Phi^{(\mathbf{W})} : \mathbb{R}^d \rightarrow \mathbb{R}^{d'} : \mathbf{x} \mapsto \mathbf{W}\mathbf{x}$$

such that the new feature vectors  $\mathbf{z}^{(r)} = \mathbf{W}\mathbf{x}^{(r)}$  allow us to reconstruct the original features with minimum linear reconstruction error [8], [22], [83]:

$$\min_{\mathbf{R} \in \mathbb{R}^{d \times d'}} \sum_{r=1}^m \|\mathbf{x}^{(r)} - \mathbf{R}\mathbf{W}\mathbf{x}^{(r)}\|_2^2.$$

We can view PCA as a form of ERM using the loss function  $L(\mathbf{x}, \mathbf{W}) = \|\mathbf{x} - \widehat{\mathbf{R}}\mathbf{W}\mathbf{x}\|_2^2$  with a reconstruction matrix  $\widehat{\mathbf{R}}$  that achieves the above minimum reconstruction error. It turns out that this ERM problem can be solved by a matrix  $\mathbf{W} = (\mathbf{u}^{(1)}, \dots, \mathbf{u}^{(d')})^T$  whose rows are given by

$d'$  eigenvectors corresponding to the  $d'$  largest eigenvalues of the matrix:

$$\widehat{\mathbf{Q}} = \frac{1}{m} \sum_{r=1}^m \mathbf{x}^{(r)} (\mathbf{x}^{(r)})^T = \mathbf{X}^T \mathbf{X}.$$

Note that  $\widehat{\mathbf{Q}}$  coincides with the sample covariance matrix of  $\mathcal{D}$  if its sample mean vanishes. The psd matrix  $\widehat{\mathbf{Q}}$  allows for an EVD of the following form [33], [85]:

$$\widehat{\mathbf{Q}} = \sum_{j=1}^d \lambda_j \mathbf{u}^{(d)} (\mathbf{u}^{(d)})^T = \begin{pmatrix} \mathbf{u}^{(1)} & \dots & \mathbf{u}^{(d)} \end{pmatrix} \begin{pmatrix} \lambda_1 & & 0 \\ & \ddots & \\ 0 & & \lambda_d \end{pmatrix} \begin{pmatrix} (\mathbf{u}^{(1)})^T \\ \vdots \\ (\mathbf{u}^{(d)})^T \end{pmatrix}.$$

This decomposition consists of decreasing nonnegative eigenvalues  $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_d \geq 0$  and corresponding eigenvectors  $\mathbf{u}^{(1)}, \dots, \mathbf{u}^{(d)}$  that form an orthonormal basis of  $\mathbb{R}^d$ .

See also: feature map, feature learning, dimensionality reduction, EVD.

**privacy attack** A privacy attack on an ML system aims to infer sensitive attributes of individuals by exploiting partial access to a trained ML model. One form of a privacy attack is model inversion.

See also: attack, sensitive attribute, model inversion, trustworthy AI, GDPR.

**privacy funnel** The privacy funnel is a method for learning a feature map that provides privacy-friendly features for a data point [196].

See also: feature, data point, GDPR, DP.

**privacy leakage** Consider an ML application that processes a dataset  $\mathcal{D}$  and delivers some output, such as the predictions obtained for new

data points. Privacy leakage arises if the output carries information about a private (or sensitive) feature of a data point of  $\mathcal{D}$  (such as a human). Based on a probabilistic model for the data generation, we can measure the privacy leakage via the MI between the output and the sensitive feature. Another quantitative measure of privacy leakage is DP. The relations between different measures of privacy leakage have been studied in the literature (see [197]).

See also: MI, DP, privacy attack, GDPR.

**privacy protection** Consider some ML method  $\mathcal{A}$  that reads in a dataset  $\mathcal{D}$  and delivers some output  $\mathcal{A}(\mathcal{D})$ . The output could be the learned model parameters  $\hat{\mathbf{w}}$  or the prediction  $\hat{h}(\mathbf{x})$  obtained for a specific data point with features  $\mathbf{x}$ . Many important ML applications involve data points representing humans. Each data point is characterized by features  $\mathbf{x}$ , potentially a label  $y$ , and a sensitive attribute  $s$  (e.g., a recent medical diagnosis). Roughly speaking, privacy protection means that it should be impossible to infer, from the output  $\mathcal{A}(\mathcal{D})$ , any of the sensitive attributes of data points in  $\mathcal{D}$ . Mathematically, privacy protection requires non-invertibility of the map  $\mathcal{A}(\mathcal{D})$ . In general, just making  $\mathcal{A}(\mathcal{D})$  non-invertible is typically insufficient for privacy protection. We need to make  $\mathcal{A}(\mathcal{D})$  sufficiently non-invertible.

See also: ML, dataset, model parameter, prediction, data point, feature, label, sensitive attribute, map.

**probability** We assign a probability value, typically chosen in the interval  $[0, 1]$ , to each event that can occur in a random experiment [6], [7], [20],

[73].

See also: event, random experiment.

**Q-learning** Q-learning is a popular RL algorithm that learns an optimal policy by estimating the optimal action-value function (or Q-function) [93].

See also: RL, fixed-point iteration.

**quadratic function** A quadratic function is a function  $f : \mathbb{R}^d \rightarrow \mathbb{R}$  of the following form:

$$f(\mathbf{w}) = \mathbf{w}^T \mathbf{Q} \mathbf{w} + \mathbf{q}^T \mathbf{w} + a$$

with some matrix  $\mathbf{Q} \in \mathbb{R}^{d \times d}$ , vector  $\mathbf{q} \in \mathbb{R}^d$ , and scalar  $a \in \mathbb{R}$ .

See also: function, matrix, vector.

**Rademacher complexity** Similar to the VC dimension, the Rademacher complexity [198] is a quantitative measure of the size of a hypothesis space  $\mathcal{H}$ . It is based on the empirical Rademacher complexity, which is defined for a given dataset  $\mathcal{D}$  as

$$\mathcal{R}_{\mathcal{D}}(\mathcal{H}) = \mathbb{E}_{\varepsilon_1, \dots, \varepsilon_m} \sup_{h \in \mathcal{H}} \frac{1}{m} \sum_{r=1}^m \varepsilon_r h(\mathbf{x}^{(r)}).$$

Here, the expectation is taken with respect to the RVs  $\varepsilon_1, \dots, \varepsilon_m$ , which are i.i.d. and take values in  $\{-1, +1\}$  with equal probability  $1/2$ . The Rademacher complexity of  $\mathcal{H}$  is then defined as the expectation of the empirical Rademacher complexity of a random dataset  $\mathcal{D} = \{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$  that consists of  $m$  i.i.d. RVs  $\mathbf{x}^{(r)} \in \mathcal{X}$  for  $r = 1, \dots, m$ .

See also: VC dimension, hypothesis space, generalization, ML, effective dimension.

**random forest** A random forest is a set of different decision trees. Each of these decision trees is obtained by fitting a perturbed copy of the original dataset.

See also: decision tree, dataset.

**random projection** A random projection uses a random wide matrix  $\mathbf{A} \in \mathbb{R}^{d' \times d}$ , with  $d' < d$ , to map a feature vector  $\mathbf{x} \in \mathbb{R}^d$  to a shorter feature vector  $\mathbf{Ax} \in \mathbb{R}^{d'}$ . It is a basic method for feature learning and dimensionality reduction. The projection matrix  $\mathbf{A}$  is typically generated entrywise by i.i.d. RVs with a common probability distribution  $\mathbb{P}$ . For a broad class of such probability distributions, a random projection approximately preserves pairwise Euclidean distances between feature vectors of a given finite dataset. The celebrated JL lemma guarantees the existence of such a distance-preserving dimensionality reduction map but does not itself involve randomness. Random projections provide a probabilistic construction that realizes this guarantee with high probability. Roughly speaking, for many relevant applications, random projections preserve the most relevant information contained in the original (typically very long) feature vector. Fig. 127 illustrates this behavior for an RGB image. The left panel shows the original image. The middle panel shows a masked image where a randomly selected five percent of the original pixels are kept, and the remaining pixels are set to a fixed light-gray color. The right panel shows the result of a simple

reconstruction based on repeated averaging of nearby retained pixels.

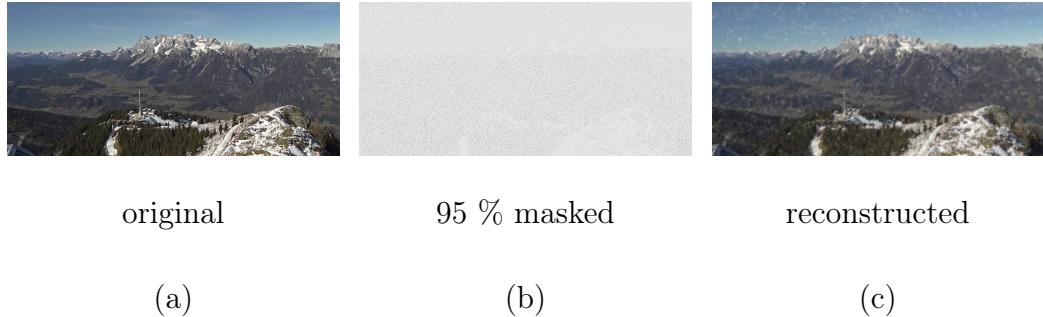


Fig. 127. Illustration of a random projection in the form of removing (or masking) all image pixels, except those in a small random subset. (a) The left panel shows the original RGB image. (b) The middle panel shows a version with only a random five percent subset of pixels retained. (c) The right panel shows a simple convolution-based reconstruction that diffuses information from the known pixels into masked regions.

See also: feature learning, dimensionality reduction, JL lemma.

Python demo: [click me](#)

**realization** Consider an RV  $\mathbf{x}$  that maps each outcome  $\omega \in \Omega$  of a probability space to an element  $a$  of a measurable space  $\mathcal{N}$  [2], [6], [73]. A realization of  $\mathbf{x}$  is any element  $\mathbf{a} \in \mathcal{N}$  such that there exists an element  $\omega' \in \Omega$  with  $\mathbf{x}(\omega') = \mathbf{a}$ .

See also: RV, outcome, probability space, measurable.

**recall** Recall is a metric commonly used in binary classification for the assessment of trained models. It is the ratio between the number of true positives (i.e., correctly predicted positive data points) and the

number of data points with a positive label [194], [195].

See also: metric, classification, confusion matrix, precision.

**receiver operating characteristic (ROC)** Consider a label space  $\mathcal{Y} = \{-1, 1\}$  and a classifier that uses a real-valued hypothesis  $h(\mathbf{x})$ . For a given threshold  $\eta \in \mathbb{R}$ , the ultimate prediction is  $\hat{y} = 1$  if  $h(\mathbf{x}) \geq \eta$  and  $\hat{y} = -1$  otherwise. On a test set  $\mathcal{D}^{(\text{test})}$ , we compute, for each value of  $\eta$ , the following two quantities: 1) true positive rate  $\text{TPR}^{(\eta)}$ ; and 2) false positive rate  $\text{FPR}^{(\eta)}$ . The ROC curve is the following function [199]:

$$\text{ROC} : \mathbb{R} \rightarrow \mathbb{R}^2 : \eta \mapsto (\text{TPR}^{(\eta)}, \text{FPR}^{(\eta)}).$$

One important characteristic of the ROC curve is the area under the curve (AUC).

See also: classifier, AUC.

**rectified linear unit (ReLU)** The ReLU is a popular choice for the activation function of a neuron within an ANN. It is defined as  $\sigma(z) = \max\{0, z\}$ , with  $z$  being the weighted input of the artificial neuron.

See also: activation function, ANN.

**recurrent neural network (RNN)** An RNN is a specific type of ANN that is designed for processing data that consist of a sequence of tokens. An RNN maintains an internal hidden state that is updated recurrently as new tokens are processed. This recurrent dependence allows information to propagate across time steps, making RNNs suitable for tasks such as speech recognition, language modeling, or time series prediction. However, their inherently sequential computation limits parallelization

and is challenging for gradient-based methods. Variants like the long short-term memory (LSTM) and gated recurrent unit (GRU) mitigate these problems.

See also: ANN, token.

**regression** Regression problems revolve around the prediction of a numeric label solely from the features of a data point [8, Ch. 2].

See also: prediction, label, feature, data point.

**regularization** A key challenge of modern ML applications is that they often use large models, which have an effective dimension in the order of billions. Training a high-dimensional model using basic ERM-based methods is prone to overfitting, i.e., the learned hypothesis performs well on the training set but poorly outside the training set. Regularization refers to modifications of a given instance of ERM in order to avoid overfitting, i.e., to ensure that the learned hypothesis does not perform much worse outside the training set. There are three routes for implementing regularization:

- 1) Model pruning: We prune the original model  $\mathcal{H}$  to obtain a smaller model  $\mathcal{H}'$ . For a parametric model, the pruning can be implemented via constraints on the model parameters (such as  $w_1 \in [0.4, 0.6]$  for the weight of feature  $x_1$  in linear regression).
- 2) Loss penalization: We modify the objective function of ERM by adding a penalty term to the training error. The penalty term estimates how much higher the expected loss (or risk) is compared with the average loss on the training set.

- 3) Data augmentation: We can enlarge the training set  $\mathcal{D}$  by adding perturbed copies of the original data points in  $\mathcal{D}$ . One example for such a perturbation is to add the realization of an RV to the feature vector of a data point.

Fig. 128 illustrates the above three routes to regularization. These routes are closely related and sometimes fully equivalent. Data augmentation using Gaussian RVs to perturb the feature vectors in the training set of linear regression has the same effect as adding the penalty  $\lambda\|\mathbf{w}\|_2^2$  to the training error (which is nothing but ridge regression). The decision on which route to use for regularization can be based on the available computational infrastructure. For example, it might be much easier to implement data augmentation than model pruning.

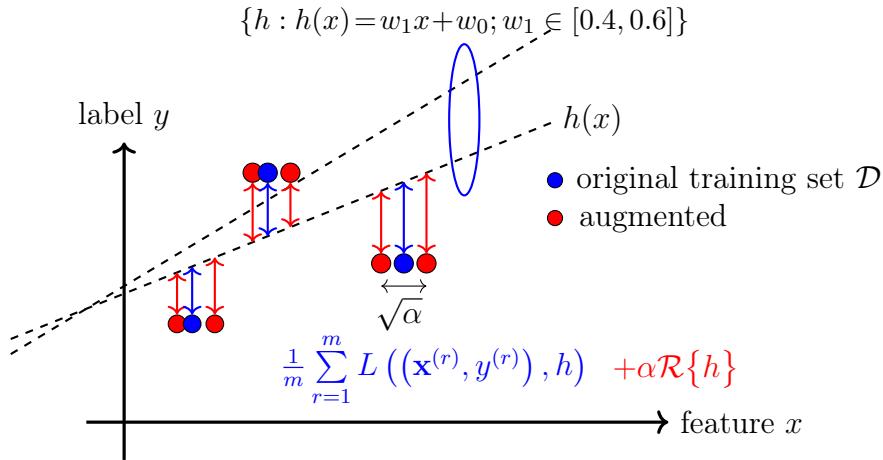


Fig. 128. Three approaches to regularization: 1) data augmentation; 2) loss penalization; and 3) model pruning (via constraints on model parameters).

See also: overfitting, data augmentation, validation, ridge regression, Lasso, model selection.

**regularized empirical risk minimization (RERM)** Basic ERM learns a hypothesis (or trains a model)  $h \in \mathcal{H}$  based solely on the empirical risk  $\widehat{L}(h|\mathcal{D})$  incurred on a training set  $\mathcal{D}$ . To make ERM less prone to overfitting, we can implement regularization by including a (scaled) regularizer  $\mathcal{R}\{h\}$  in the learning objective. This leads to RERM such that

$$\hat{h} \in \arg \min_{h \in \mathcal{H}} \widehat{L}(h|\mathcal{D}) + \alpha \mathcal{R}\{h\}. \quad (19)$$

The parameter  $\alpha \geq 0$  controls the regularization strength. For  $\alpha = 0$ , we recover standard ERM without regularization. As  $\alpha$  increases, the learned hypothesis is increasingly biased toward small values of  $\mathcal{R}\{h\}$ . The component  $\alpha \mathcal{R}\{h\}$  in the objective function of (19) can be intuitively understood as a surrogate for the increased average loss that may occur when predicting labels for data points outside the training set. This intuition can be made precise in various ways. For example, consider a linear model trained using squared error loss and the regularizer  $\mathcal{R}\{h\} = \|\mathbf{w}\|_2^2$ . In this setting,  $\alpha \mathcal{R}\{h\}$  corresponds to the expected increase in loss caused by adding Gaussian RVs to the feature vectors in the training set [8, Ch. 3]. A principled construction for the regularizer  $\mathcal{R}\{h\}$  arises from approximate upper bounds on the generalization error. The resulting RERM instance is known as SRM [200, Sec. 7.2].

See also: ERM, regularization, loss, SRM.

**regularized loss minimization (RLM)** See RERM.

**regularizer** A regularizer assigns each hypothesis  $h$  from a hypothesis space  $\mathcal{H}$  a quantitative measure  $\mathcal{R}\{h\}$  conveying to what extent its prediction errors might differ on data points on and outside a training set. Ridge regression uses the regularizer  $\mathcal{R}\{h\} := \|\mathbf{w}\|_2^2$  for linear hypothesis maps  $h^{(\mathbf{w})}(\mathbf{x}) := \mathbf{w}^T \mathbf{x}$  [8, Ch. 3]. Lasso uses the regularizer  $\mathcal{R}\{h\} := \|\mathbf{w}\|_1$  for linear hypothesis maps  $h^{(\mathbf{w})}(\mathbf{x}) := \mathbf{w}^T \mathbf{x}$  [8, Ch. 3].

See also: ridge regression, Lasso, loss, objective function.

**relational model** A relational model is a mathematical representation of data. The core idea is to organize data as a collection of tables (or relations) [126], [130]. A table consists of rows and columns, where each row corresponds to a single data point, and each column represents a specific attribute of a data point. ML methods use these attributes as the features and label of a data point. Table II shows a relational representation of a dataset that consists of cows. In the relational model, the order of rows is immaterial, and each attribute (i.e., column) is associated with a domain that specifies the set of admissible values. In ML applications, these attribute domains correspond to the feature space and the label space.

TABLE II

A RELATION (OR TABLE) THAT REPRESENTS THE DATASET IN FIG. 80

Name	Weight	Age	Height	Stomach temperature
Zenzi	100	4	100	25
Berta	140	3	130	23
Resi	120	4	120	31

See also: data, data point, features, label, dataset.

**response** The term response is sometimes used as a synonym for the label of a data point [83].

See also: label, data point, target.

**response vector** The term response vector is used as a synonym for the label vector of a dataset [83].

See also: response, label vector, dataset, target vector.

**reward** A reward refers to some observed (or measured) quantity that allows us to estimate the loss incurred by the prediction (or decision) of a hypothesis  $h(\mathbf{x})$ . For example, in an ML application to self-driving vehicles,  $h(\mathbf{x})$  could represent the current steering direction of a vehicle. We could construct a reward from the measurements of a collision sensor that indicate if the vehicle is moving toward an obstacle. We define a low reward for the steering direction  $h(\mathbf{x})$  if the vehicle moves dangerously toward an obstacle.

See also: loss, MAB, RL.

**ridge regression** Consider a regression problem where the goal is to learn a hypothesis  $h(\mathbf{w})$  for predicting the numeric label of a data point based on its feature vector. Ridge regression learns the parameters  $\mathbf{w}$  by minimizing the penalized average squared error loss. The average squared error loss is measured on a set of labeled data points (i.e., the training set)

$$(\mathbf{x}^{(1)}, y^{(1)}) , \dots , (\mathbf{x}^{(m)}, y^{(m)}) .$$

The penalty term is the scaled squared Euclidean norm  $\alpha \|\mathbf{w}\|_2^2$  with a regularization parameter  $\alpha > 0$ . The purpose of the penalty term is regularization, i.e., to prevent overfitting in the high-dimensional regime, where the number of features  $d$  exceeds the number of data points  $m$  in the training set. For the training of a linear model, adding  $\alpha \|\mathbf{w}\|_2^2$  to the average squared error loss is equivalent to computing the average squared error loss on an augmented training set.

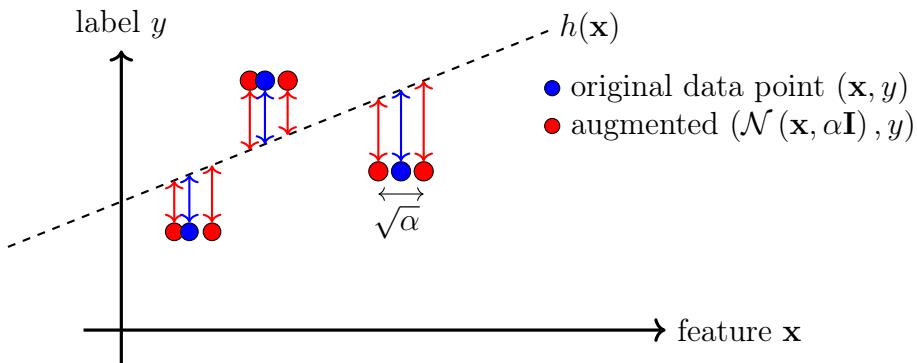


Fig. 129. For a linear model, adding the penalty term  $\alpha \|\mathbf{w}\|_2^2$  to the objective function in ERM is equivalent to ERM on an augmented dataset.

This augmented training set is obtained by replacing each data point  $(\mathbf{x}^{(r)}, y^{(r)})$  in the original training set by the realization of infinitely many i.i.d. RVs whose probability distribution is centered around  $(\mathbf{x}^{(r)}, y^{(r)})$ .

See also: regression, regularization, map, data augmentation.

**risk** Consider a hypothesis  $h$  used to predict the label  $y$  of a data point based on its features  $\mathbf{x}$ . We measure the quality of a particular prediction using a loss function  $L((\mathbf{x}, y), h)$ . If we interpret data points as the realizations of i.i.d. RVs, the  $L((\mathbf{x}, y), h)$  also becomes the realization of an RV. The i.i.d. assumption allows us to define the risk of a hypothesis as the expected loss  $\mathbb{E}\{L((\mathbf{x}, y), h)\}$ . Note that the risk of  $h$  depends on both the specific choice for the loss function and the probability distribution of the data points.

See also: i.i.d. RV, i.i.d. assumption, loss, probability distribution.

**risk stratification** Risk stratification assigns data points to risk groups (or strata) by quantizing the prediction  $\hat{h}(\mathbf{x})$  obtained from a trained risk prognosis model. Typical choices for these strata include low, intermediate, and high risk [201], [202].

See also: stratification, prediction, clustering.

**robustness** Robustness is a key requirement for trustworthy AI. It refers to the property of an ML system to maintain acceptable performance even when subjected to different forms of perturbations. These perturbations may affect the features of a data point in order to manipulate the prediction delivered by a trained ML model. Robustness also includes the stability of ERM-based methods against perturbations of the training

set. Such perturbations can occur within data poisoning attacks.

See also: trustworthy AI, stability, data poisoning, attack.

**sample mean** The sample mean  $\mathbf{m} \in \mathbb{R}^d$  for a given dataset, with feature vectors  $\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)} \in \mathbb{R}^d$ , is defined as

$$\mathbf{m} = \frac{1}{m} \sum_{r=1}^m \mathbf{x}^{(r)}.$$

See also: sample, mean, dataset, feature vector.

**sample size** The sample size is the number of individual data points contained in a sample or dataset. Consider an ERM-based method that uses a training set  $\mathcal{D}$  with sample size  $m$  and a model  $\mathcal{H}$  with effective dimension  $d_{\text{eff}}(\mathcal{H})$ . If the training set can be well approximated by the i.i.d. assumption, then the ratio between  $m$  and  $d_{\text{eff}}(\mathcal{H})$  can be a useful indicator of the occurrence of overfitting [8, Ch. 6].

See also: data point, dataset.

**sample space** A sample space is the set of all possible outcomes of a random experiment [6], [7], [28], [29].

See also: probability space.

**sample weighting** Consider an ERM-based method that learns a hypothesis by minimizing the average loss on a training set. In its basic form, ERM treats all data points equally important. However, in some applications, it can be useful to put different emphasis on the prediction errors obtained for different data points. For example, if a data point is

considered an outlier we should reduce its influence on the learned hypothesis. We can implement this idea by assigning a nonnegative weight  $q^{(r)}$  to each data point  $(\mathbf{x}^{(r)}, y^{(r)})$  in the training set. This results in the following weighted ERM principle:

$$\min_{h \in \mathcal{H}} \sum_{r=1}^m q^{(r)} L((\mathbf{x}^{(r)}, y^{(r)}), h). \quad (20)$$

Fig. 130 illustrates the concept of a training set of three data points that contribute unequally to the empirical risk.

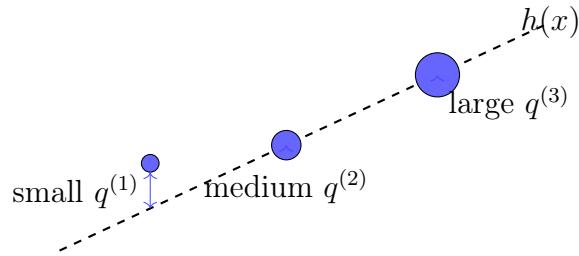


Fig. 130. Sample weighting assigns each data point of a training set a weight  $q^{(r)}$ . Assigning a small weight (such as  $q^{(1)}$  in this example) to a data point decreases its influence on the hypothesis learned via solving (20).

See also: ERM, outlier, AdaBoost.

**scatterplot** A visualization technique that depicts data points using markers in a 2-D plane. Fig. 131 depicts an example of a scatterplot.

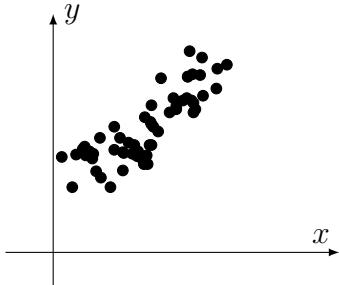


Fig. 131. A scatterplot with circle markers, where the data points represent daily weather conditions in Finland. Each data point is characterized by its minimum daytime temperature  $x$  as the feature and its maximum daytime temperature  $y$  as the label. The temperatures have been measured at the FMI weather station Helsinki Kaisaniemi during 1 September 2024—28 October 2024.

A scatterplot can enable the visual inspection of data points that are naturally represented by feature vectors in high-dimensional spaces.

See also: data point, minimum, feature, maximum, label, FMI, feature vector, dimensionality reduction.

**self-supervised learning** Self-supervised learning uses some of the features of a data point as its label. For example, if a data point consists of a sentence within a text document, we can use the last word of the sentence as the label that is to be predicted from all the previous words, which form the features of the data point. A main application of self-supervised learning is in NLP for the training of LLMs from large collections of text data.

See also: feature, label, LLM.

**semi-supervised learning (SSL)** SSL methods use unlabeled data points

to support the learning of a hypothesis from labeled data points [115]. This approach is particularly useful for ML applications that offer a large number of unlabeled data points, but only a limited number of labeled data points.

See also: data point, hypothesis, labeled data point, ML.

**sensitive attribute** ML revolves around learning a hypothesis map that allows us to predict the label of a data point from its features. In some applications, we must ensure that the output delivered by an ML system does not allow us to infer sensitive attributes of a data point. Which part of a data point is considered a sensitive attribute is a design choice that varies across different application domains.

See also: ML, hypothesis, map, label, data point, feature.

**sensitivity** See recall.

**similarity graph** Some ML applications generate data points that are related by a domain-specific notion of similarity. These similarities can be represented conveniently using a similarity graph  $\mathcal{G} = (\mathcal{V} := \{1, \dots, m\}, \mathcal{E})$ . The node  $r \in \mathcal{V}$  represents the  $r$ th data point. Two nodes are connected by an undirected edge if the corresponding data points are similar.

See also: ML, data point, graph, connected.

**skip connection** Consider a deep net with neurons that are organized in consecutive layers. A skip connection links the output of a neuron in some layer to the input of a neuron in a nonconsecutive layer [104].

See also: deep net, DAG.

**soft clustering** Soft clustering refers to the task of partitioning a given set of data points into (a few) overlapping clusters. Each data point is assigned to several different clusters with varying degrees of belonging. Soft clustering methods determine the degree of belonging (or soft cluster assignment) for each data point and each cluster. A principled approach to soft clustering for data points characterized by numerical feature vectors is via a probabilistic model such as the GMM. The conditional probability of a data point belonging to a specific mixture component is then a natural choice for the degree of belonging. GMM soft clustering methods can be applied to nonnumeric data by using feature learning methods to provide numerical features (such as in spectral clustering).

See also: clustering, cluster, degree of belonging, GMM, spectral clustering.

**soft label** Consider a classification problem where data points are characterized by features  $\mathbf{x}$  and labels from a finite label space  $\mathcal{Y} = \{1, \dots, k\}$ . Some ML applications involve data points that have almost identical features but different labels. In such cases, instead of assigning to each data point a single label  $y \in \mathcal{Y}$ , it can be more useful to assign an entire probability distribution over the label space. This probability distribution can be represented as a pmf  $\mathbf{y} = (y_1, \dots, y_k)^T \in \Delta^k$ . We can view the entries  $y_c$ , for  $c = 1, \dots, k$ , as soft labels of a data point. Mathematically, the soft label  $y_c$  is the probability that a randomly chosen data point with feature vector  $\mathbf{x}$  has label  $c$ .

See also: classification, pmf, cross-entropy.

**softmax function** The softmax function  $S : \mathbb{R}^k \rightarrow \Delta^{k-1}$  maps an arbitrary input vector  $\mathbf{w} \in \mathbb{R}^k$  (often referred to as logits) onto the probability simplex. The  $i$ th component of the output is defined using the exponential function as

$$S(\mathbf{w})_i = \frac{\exp(w_i)}{\sum_{j=1}^k \exp(w_j)} \quad \text{for } i = 1, \dots, k. \quad (21)$$

Since the resulting components are non-negative and sum to one, the output of the softmax function can be interpreted as a pmf over  $k$  distinct outcomes. The softmax function is a standard tool in ML for modeling discrete probability distributions. For instance, in an ANN-based classifier, it is typically applied to the outputs of the final layer to obtain the predicted probability distribution over the possible class label. Similarly, in RL with a discrete action space, a policy can be defined using a softmax function to describe the probability distribution for the next action to take.

See also: probability simplex, pmf, ANN, classifier, layer, policy.

**spectral clustering** Spectral clustering is a particular instance of graph clustering, i.e., it clusters data points represented as the nodes  $i = 1, \dots, n$  of a graph  $\mathcal{G}$ . Spectral clustering uses the eigenvectors of the Laplacian matrix  $\mathbf{L}^{(\mathcal{G})}$  to construct feature vectors  $\mathbf{x}^{(i)} \in \mathbb{R}^d$  for each node (i.e., for each data point)  $i = 1, \dots, n$ . We can feed these feature vectors into Euclidean space-based clustering methods, such as  $k$ -means or soft clustering via GMM. Roughly speaking, the feature vectors of nodes belonging to a well-connected subset (or cluster) of nodes in  $\mathcal{G}$  are located nearby in the Euclidean space  $\mathbb{R}^d$  (see Fig. 132).

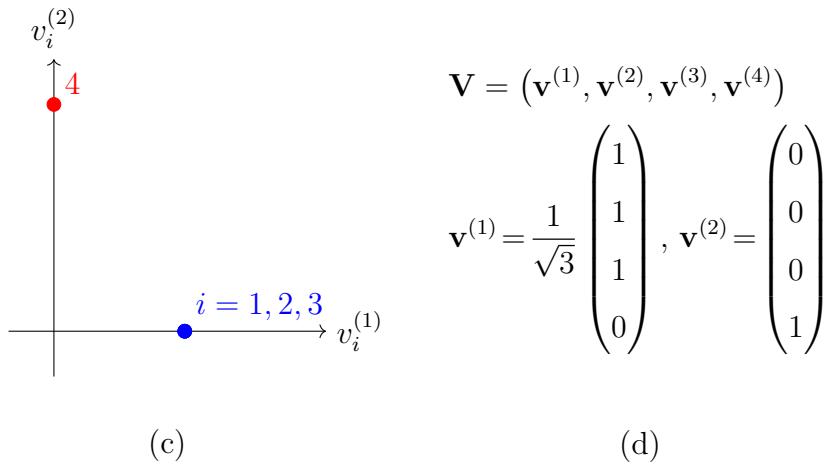
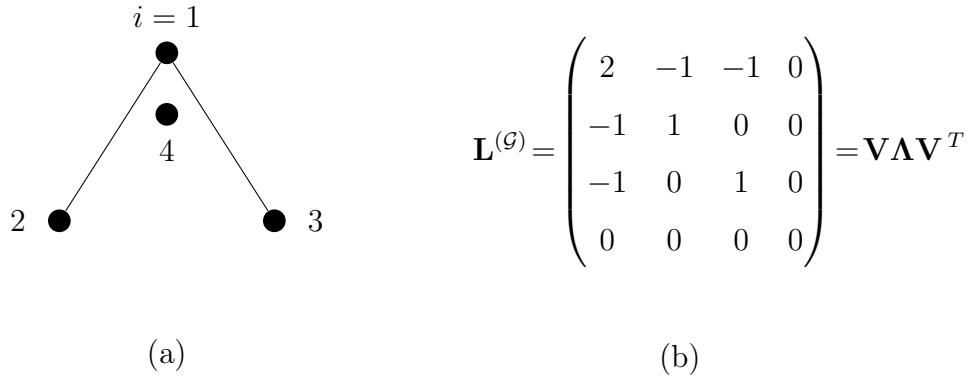


Fig. 132. (a) An undirected graph  $\mathcal{G}$  with four nodes  $i = 1, 2, 3, 4$ , each representing a data point. (b) The Laplacian matrix  $\mathbf{L}^{(\mathcal{G})} \in \mathbb{R}^{4 \times 4}$  and its EVD. (c) A scatterplot of data points using the feature vectors  $\mathbf{x}^{(i)} = (v_i^{(1)}, v_i^{(2)})^T$ . (d) Two eigenvectors  $\mathbf{v}^{(1)}, \mathbf{v}^{(2)} \in \mathbb{R}^d$  corresponding to the eigenvalue  $\lambda = 0$  of the Laplacian matrix  $\mathbf{L}^{(\mathcal{G})}$ .

See also: clustering, graph clustering, Laplacian matrix, eigenvalue.

**spectrogram** A spectrogram represents the time-frequency distribution of the energy of a time signal  $x(t)$ . Intuitively, it quantifies the amount of signal energy present within a specific time segment  $[t_1, t_2] \subseteq \mathbb{R}$  and frequency interval  $[f_1, f_2] \subseteq \mathbb{R}$ . Formally, the spectrogram of a signal is defined as the squared magnitude of its short-time Fourier transform (STFT) [203]. Fig. 133 depicts a time signal along with its spectrogram.

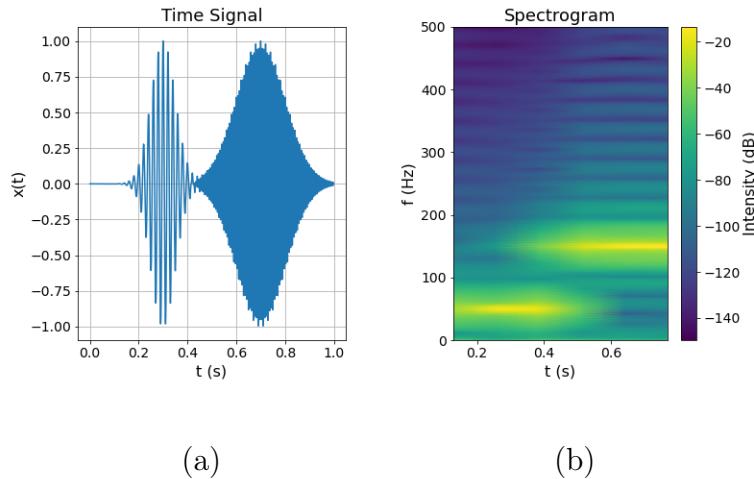


Fig. 133. (a) A time signal consisting of two modulated Gaussian pulses. (b) An intensity plot of the spectrogram.

The intensity plot of its spectrogram can serve as an image of a signal. A simple recipe for audio signal classification is to feed this signal image into deep nets originally developed for image classification and object detection [204]. It is worth noting that, beyond the spectrogram, several alternative representations exist for the time-frequency distribution of

signal energy [205], [206].

See also: classification, deep net.

**squared error loss** The squared error loss measures the prediction error of a hypothesis  $h$  when predicting a numeric label  $y \in \mathbb{R}$  from the features  $\mathbf{x}$  of a data point. It is defined as

$$L((\mathbf{x}, y), h) := (y - \underbrace{h(\mathbf{x})}_{=\hat{y}})^2.$$

See also: loss, prediction, hypothesis, label, feature, data point.

**stability** Mathematically, an ML method is a map  $\mathcal{A}$  from a given dataset  $\mathcal{D}$  to an output  $\mathcal{A}(\mathcal{D})$ . As a case in point, consider an ERM-based ML method that maps a training set  $\mathcal{D}$  to the learned model parameters  $\mathcal{A}(\mathcal{D}) = \hat{\mathbf{w}}$ , which achieve the minimum average loss on the training set. Instead of the learned model parameters, the output  $\mathcal{A}(\mathcal{D})$  could also be the predictions obtained from the trained model. Stability refers to the desirable property of  $\mathcal{A}$  that small changes in the input dataset  $\mathcal{D}$  result in small changes in the output  $\mathcal{A}(\mathcal{D})$ . The notion of stability is intimately related to the notion of generalization. In particular, there are formal notions of stability that allow us to bound the generalization gap (see [155, Ch. 13]). To build intuition, consider the three datasets depicted in Fig. 134, each of which is equally likely under the same data-generating probability distribution. Since the optimal model parameters are determined by this underlying probability distribution, an accurate ML method  $\mathcal{A}$  should return the same (or very similar) output  $\mathcal{A}(\mathcal{D})$  for all three datasets. In other words, any useful  $\mathcal{A}$  must be robust to

variability in sample realizations from the same probability distribution, i.e., it must be stable.

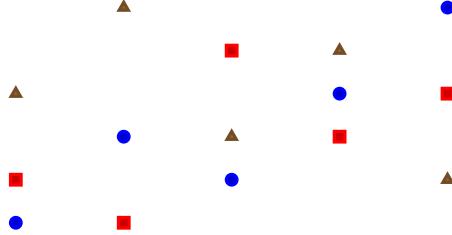


Fig. 134. Three datasets  $\mathcal{D}^{(*)}$ ,  $\mathcal{D}^{(\square)}$ , and  $\mathcal{D}^{(\triangle)}$ , each sampled independently from the same data-generating probability distribution. A stable ML method should return similar outputs when trained on any of these datasets.

See also: generalization, robustness.

**stacking** Stacking is one of the main types of ensemble methods. In stacking, a finite number  $M$  of base learners are trained on the same dataset but with different models  $\mathcal{H}^{(j)}$  or loss functions  $L^{(j)}$  for  $j = 1, \dots, M$  [83, Ch. 8.8], [207], [208]. The  $j$ th base learner delivers a learned hypothesis  $\hat{h}^{(j)} \in \mathcal{H}^{(j)}$ . The final prediction for a data point is obtained by aggregating the predictions of the base learners via an aggregation rule  $\phi_{\text{agg}}$ , such as majority voting for classification or averaging for regression. We can interpret stacking as a form of feature learning, where each base learner extracts a new feature. The aggregation rule can be obtained by another instance of ERM that learns a hypothesis map  $\phi_{\text{agg}} \in \mathcal{H}$  from a meta-model  $\mathcal{H}$ . The hypothesis  $\phi_{\text{agg}}$  is applied to the transformed feature vector

$$\Phi(\mathbf{x}) = (\hat{h}^{(1)}(\mathbf{x}), \dots, \hat{h}^{(M)}(\mathbf{x}))^T.$$

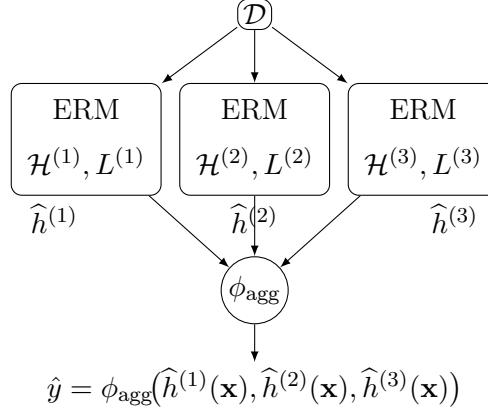


Fig. 135. Three base learners using ERM with different models and loss functions to obtain learned hypotheses  $\hat{h}^{(1)}, \hat{h}^{(2)}, \hat{h}^{(3)}$ . For a data point with feature vector  $\mathbf{x}$ , each base learner delivers a prediction  $\hat{h}^{(j)}(\mathbf{x})$  for  $j = 1, 2, 3$ . These predictions are then used as new features for an aggregation rule  $\phi_{\text{agg}}$  that delivers the overall prediction  $\hat{y}$ . The aggregation rule can be obtained by training a meta-model  $\mathcal{H}$ .

See also: ensemble, bagging.

**statistical aspect** By statistical aspects of an ML method, we refer to (properties of) the probability distribution of its output under a probabilistic model for the data fed into the method.

See also: ML, probability distribution, probabilistic model, data.

**step size** See learning rate.

**stochastic algorithm** A stochastic algorithm uses a random mechanism during its execution. For example, SGD uses a randomly selected subset of data points to compute an approximation for the gradient of

an objective function. We can represent a stochastic algorithm by a stochastic processes, i.e., the possible execution sequence is the possible outcomes of a random experiment [7], [209], [210].

See also: stochastic, algorithm, SGD, data point, gradient, objective function, stochastic process, random experiment, optimization method, gradient-based method.

**stochastic block model (SBM)** The SBM [211] is a probabilistic generative model for an undirected graph  $\mathcal{G} = (\mathcal{V}, \mathcal{E})$  with a given set of nodes  $\mathcal{V}$  [212]. In its most basic variant, the SBM generates a graph by first randomly assigning each node  $i \in \mathcal{V}$  to a cluster index  $c_i \in \{1, \dots, k\}$ . A pair of different nodes in the graph is connected by an edge with probability  $p_{i,i'}$  that depends solely on the labels  $c_i, c_{i'}$ . The presence of edges between different pairs of nodes is statistically independent.

See also: model, graph, cluster, probability, label.

**stochastic gradient descent (SGD)** SGD is obtained from GD by replacing the gradient of the objective function with a stochastic approximation. A main application of SGD is to train a parameterized model via ERM on a training set  $\mathcal{D}$  that is either very large or not readily available (e.g., when data points are stored in a database distributed globally). To evaluate the gradient of the empirical risk (as a function of the model parameters  $\mathbf{w}$ ), we need to compute a sum  $\sum_{r=1}^m \nabla_{\mathbf{w}} L(\mathbf{z}^{(r)}, \mathbf{w})$  over all data points in the training set. We obtain a stochastic approximation to the gradient by replacing the sum  $\sum_{r=1}^m \nabla_{\mathbf{w}} L(\mathbf{z}^{(r)}, \mathbf{w})$  with a sum  $\sum_{r \in \mathcal{B}} \nabla_{\mathbf{w}} L(\mathbf{z}^{(r)}, \mathbf{w})$  over a randomly chosen subset  $\mathcal{B} \subseteq \{1, \dots, m\}$

(see Fig. 136). We often refer to these randomly chosen data points as a batch. The batch size  $|\mathcal{B}|$  is an important parameter of SGD. SGD with  $|\mathcal{B}| > 1$  is referred to as mini-batch SGD [181].

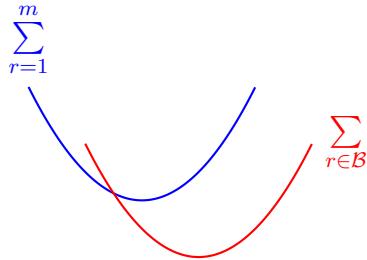


Fig. 136. SGD for ERM approximates the gradient by replacing the sum  $\sum_{r=1}^m \nabla_{\mathbf{w}} L(\mathbf{z}^{(r)}, \mathbf{w})$  over all data points in the training set (indexed by  $r = 1, \dots, m$ ) with a sum over a randomly chosen subset  $\mathcal{B} \subseteq \{1, \dots, m\}$ .

See also: GD, gradient, objective function, stochastic, model, ERM, training set, data point, empirical risk, function, model parameter, batch, parameter.

**stopping criterion** Many ML methods use iterative algorithms that construct a sequence of model parameters in order to minimize the training error. For example, gradient-based methods iteratively update the parameters of a parametric model, such as a linear model or a deep net. Given a finite amount of computational resources, we need to stop updating the parameters after a finite number of iterations. A stopping criterion is any well-defined condition for deciding when to stop updating.

See also: algorithm, gradient-based method.

**stratification** The process of splitting a dataset into subsets, so called strata, according to some key attribute is called stratification [89], [90], [213]. The goal is to ensure that an ML method performs well for each stratum defined by these attributes. For example, in a medical dataset, we may want to stratify a patient dataset by age groups to ensure that an ML model performs well across all age groups.

When splitting a dataset into a training set and a validation set, stratification ensures that both sets have similar distributions of the key attribute. Without stratification, using a small validation set may underrepresent or even completely miss data points with a rare attribute, leading to misleading performance estimates. See Fig. 137 for a visual illustration.

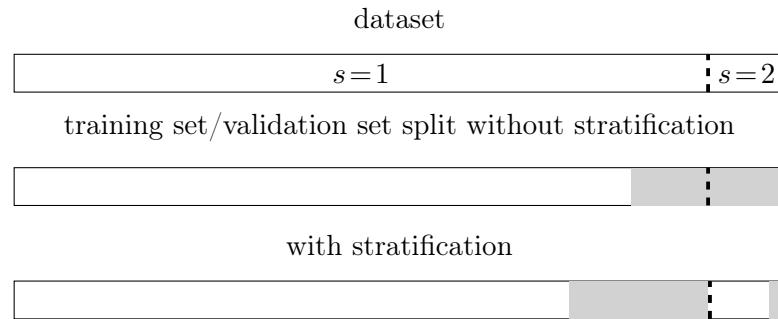


Fig. 137. Stratification ensures that both the training set and the validation set (shaded grey) have similar distributions of a binary key attribute  $s$ . In other words, with stratification, both strata (i.e.,  $s = 1$  and  $s = 2$  based on the key attribute) allocate the same validation proportion—20% of their own width.

See also: stratum, validation,  $k$ -fold CV.

**stratum** A stratum is a subset of data points that all share a common property (which could be a feature or a label). For example, in a weather dataset, all measurements from the same FMI weather station form one stratum.

Example (CSV snippet):

```
time, station, value, unit
2023-06-01 12:00, Helsinki, 18.2, degree Celsius
2023-06-01 13:00, Helsinki, 18.5, degree Celsius
2023-06-01 14:00, Helsinki, 19.0, degree Celsius
2023-06-01 12:00, Oulu, 12.1, degree Celsius
2023-06-01 13:00, Oulu, 12.4, degree Celsius
2023-06-01 14:00, Oulu, 12.7, degree Celsius
2023-06-01 12:00, Tampere, 15.3, degree Celsius
2023-06-01 13:00, Tampere, 15.6, degree Celsius
2023-06-01 14:00, Tampere, 16.0, degree Celsius
```

Here, the rows for each station (i.e., Helsinki, Oulu, Tampere) represent different strata.

See also: data point, dataset, stratification.

**structural risk minimization (SRM)** SRM is an instance of RERM with which the model  $\mathcal{H}$  can be expressed as a countable union of submodels such that  $\mathcal{H} = \bigcup_{n=1}^{\infty} \mathcal{H}^{(n)}$ . Each submodel  $\mathcal{H}^{(n)}$  permits the derivation of an approximate upper bound on the generalization error incurred when applying ERM to train  $\mathcal{H}^{(n)}$ . These individual bounds—one for each submodel—are then combined to form a regularizer used in the

RERM objective. These approximate upper bounds (one for each  $\mathcal{H}^{(n)}$ ) are then combined to construct a regularizer for RERM [155, Sec. 7.2].

See also: RERM, model, generalization, ERM, regularizer, risk.

**subgradient descent** Subgradient descent is a generalization of GD that does not require differentiability of the function to be minimized. This generalization is obtained by replacing the concept of a gradient with that of a subgradient. Similar to gradients, subgradients allow us to construct local approximations of an objective function. The objective function might be the empirical risk  $\widehat{L}(h^{(\mathbf{w})} | \mathcal{D})$  viewed as a function of the model parameters  $\mathbf{w}$  that select a hypothesis  $h^{(\mathbf{w})} \in \mathcal{H}$ .

See also: subgradient, generalization, GD, function, gradient, objective function, empirical risk, model parameter, hypothesis.

**support vector machine (SVM)** The SVM is a binary classification method that learns a linear hypothesis map. Thus, like linear regression and logistic regression, it is also an instance of ERM for the linear model. However, the SVM uses a different loss function from the one used in those methods. As illustrated in Fig. 138, it aims to maximally separate data points from the two different classes in the feature space (i.e., maximum margin principle). Maximizing this separation is equivalent to minimizing a regularized variant of the hinge loss (12) [22], [162], [214].

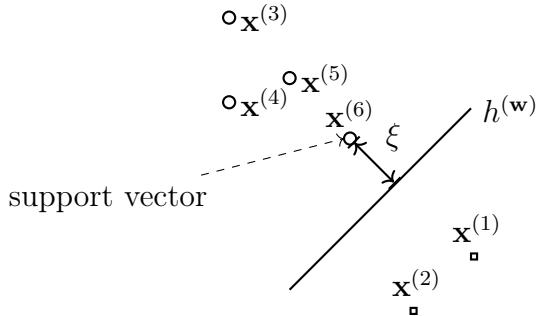


Fig. 138. The SVM learns a hypothesis (or classifier)  $h^{(\mathbf{w})}$  with minimal average soft-margin hinge loss. Minimizing this loss is equivalent to maximizing the margin  $\xi$  between the decision boundary of  $h^{(\mathbf{w})}$  and each class of the training set.

The above basic variant of SVM is only useful if the data points from different categories can be (approximately) linearly separated.

See also: classification, linear model, classifier, hinge loss.

**tabular data** Tabular data consist of data points that are characterized by a common set of attributes. These attributes can be used as the features or labels of data points. If the attributes are numeric, we can represent a dataset by a data matrix  $\mathbf{D} \in \mathbb{R}^{m \times d}$ , where each of the  $m$  rows corresponds to a single data point, and each of the  $d$  columns represents a specific attribute [132].

See also: data, data point.

**target** The term target is sometimes used as a synonym for the label of a data point [30], [22].

See also: label, data point.

**target vector** The term target vector is used as a synonym for the label vector of a dataset [30], [22].

See also: target, label vector, dataset.

**test set** A set of data points that have been used neither to train a model (e.g., via ERM) nor to choose between different models in a validation set.

See also: data point, model, ERM, validation set.

**token** A token is a basic unit of information obtained by splitting a sequence of symbols, such as a text string, into smaller parts. In NLP, tokens often correspond to words, subwords, or characters that form the features of a data point. Tokenization transforms raw text (e.g., “The cat sleeps”) into a sequence of tokens (e.g., [“The”, “cat”, “sleeps”]), which can then be mapped to numerical feature vectors.

See also: sequence, feature vector.

**training** In the context of ML, training refers to the process of learning a useful hypothesis  $\hat{h}$  out of a model  $\mathcal{H}$ . The training of a model  $\mathcal{H}$  is guided by the loss incurred on a set of data points, which serve as the training set. For parametric models, where each hypothesis  $h^{(\mathbf{w})}$  is characterized by a specific choice for the model parameters, training amounts to finding an optimal choice for the model parameters  $\mathbf{w}$ . A widely-used approach to training is ERM, which learns a hypothesis by minimizing the average loss incurred on a training set. One of the main challenges in ML is to control the discrepancy between the loss incurred

on the training set and the loss incurred on other (unseen) data points.

See also: model, loss, ERM.

**training error** Training error is the average loss of a hypothesis when predicting the labels of the data points in a training set. We sometimes also refer to training error as the minimal average loss that is achieved by a solution of ERM.

See also: loss, hypothesis, label, data point, training set, ERM.

**training set** A training set is a dataset  $\mathcal{D}$  that consists of some data points used in ERM to learn a hypothesis  $\hat{h}$ . The average loss of  $\hat{h}$  on the training set is referred to as the training error. The comparison of the training error with the validation error of  $\hat{h}$  allows us to diagnose the ML method and informs how to improve the validation error (e.g., using a different hypothesis space or collecting more data points) [8, Sec. 6.6].

See also: training, dataset, data point, ERM, hypothesis, loss, training error, validation error, ML, hypothesis space.

**transfer learning** Transfer learning aims at leveraging information obtained while solving an existing learning task to solve another learning task.

See also: learning task, multitask learning

**transformer** In the context of ML, the term transformer refers to an ANN that uses some form of attention mechanism to capture dependencies among tokens [106]. The attention mechanism is what sets transformers apart from previous models used for sequential data such as recurrent neural networks (RNNs). A transformer ANN often combines several

attention layers via more traditional layer architectures.

See also: attention, NLP.

**uncertainty** In the context of ML, uncertainty refers to the presence of multiple plausible outcomes or explanations based on available data. For example, the prediction  $\hat{h}(\mathbf{x})$  produced by a trained ML model  $\hat{h}$  often reflects a range of possible values for the true label of a given data point. The broader this range, the greater the associated uncertainty. Probability theory allows us to represent, quantify, and reason about uncertainty in a mathematically rigorous manner.

See also: probabilistic model, risk, entropy, variance.

**underfitting** Consider an ML method applying ERM to learn a hypothesis that minimizes the empirical risk on a given training set. The method is said to underfit if it fails to achieve a sufficiently low empirical risk on the training set. Underfitting typically occurs when the chosen model is too simple to capture the underlying relationship between features and labels.

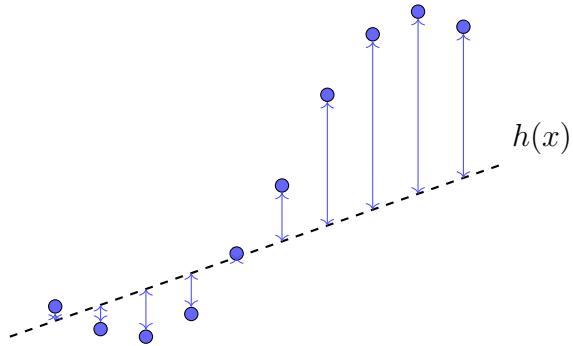


Fig. 139. No linear hypothesis  $h$  can capture the relationship between features and labels for the depicted training set. Thus, any method that uses a linear model will underfit this training set.

For example, an ML method using a linear model on data with a highly nonlinear relationship between features and labels will not be able to learn a hypothesis with small average loss on the training set, let alone a low risk.

See also: training set, model, risk, overfitting.

**upper confidence bound (UCB)** Consider an ML application that requires selecting, at each time step  $t$ , an action  $a_t$  from a finite set of alternatives  $\mathcal{A}$ . The utility of selecting action  $a_t$  is quantified by a numeric reward signal  $r^{(a_t)}$ . A widely used probabilistic model for this type of sequential decision-making problem is the stochastic multiarmed bandit (stochastic MAB) setting [190]. In this model, the reward  $r^{(a)}$  is viewed as the realization of an RV with unknown mean  $\mu^{(a)}$ . Ideally, we would always choose the action with the largest expected reward  $\mu^{(a)}$ , but these means are unknown and must be estimated from observed

data. Simply choosing the action with the largest estimate  $\hat{\mu}^{(a)}$  can lead to suboptimal outcomes due to estimation uncertainty. The UCB strategy addresses this by selecting actions not only based on their estimated means but also by incorporating a term that reflects the uncertainty in these estimates—favoring actions with a high-potential reward and high uncertainty. Theoretical guarantees for the performance of UCB strategies, including logarithmic regret bounds, are established in [190]. See also: action, reward, stochastic MAB, uncertainty, regret, optimism in the face of uncertainty.

**validation** Consider a hypothesis  $\hat{h}$  that has been learned via some ML method, e.g., by solving ERM on a training set  $\mathcal{D}$ . Validation refers to the process of evaluating the loss incurred by the hypothesis  $\hat{h}$  on a set of data points that are not contained in the training set  $\mathcal{D}$ . This set of data points is called the validation set. The average loss of  $\hat{h}$  on the validation set is referred to as the validation error. An example of validation is shown in Fig. 140.

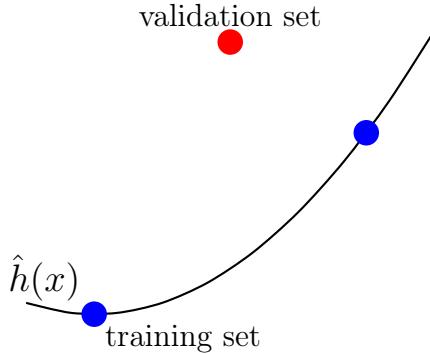


Fig. 140. Illustration of validation. The blue points represent the data points in the training set, while the red point represents a data point in the validation set. The hypothesis  $\hat{h}$  (black curve) fits the data points in the training set perfectly, but incurs a large loss on the data point in the validation set.

See also: training set, validation set, validation error, overfitting, generalization,  $k$ -fold CV, LOO-CV.

**validation error** Consider a hypothesis  $\hat{h}$  that is obtained by some ML method, e.g., using ERM on a training set. The average loss of  $\hat{h}$  on a validation set, which is different from the training set, is referred to as the validation error.

See also: hypothesis, ML, ERM, training set, loss, validation set, validation.

**validation set** A set of data points used to estimate the risk of a hypothesis  $\hat{h}$  that has been learned by some ML method (e.g., solving ERM). The average loss of  $\hat{h}$  on the validation set is referred to as the validation error and can be used to diagnose an ML method (see [8, Sec. 6.6]).

The comparison between training error and validation error can inform directions for the improvement of the ML method (such as using a different hypothesis space).

See also: data point, risk, hypothesis, ML, ERM, loss, validation, validation error, training error, hypothesis space,  $k$ -fold CV, LOO-CV.

**Vapnik–Chervonenkis dimension (VC dimension)** The statistical properties of an ERM-based method depend critically on the expressive capacity of its hypothesis space (or model)  $\mathcal{H}$ . A standard measure of this capacity is the VC dimension  $\text{VCdim}(\mathcal{H})$  [215]. Formally, it is the largest integer  $m$  such that there exists a dataset  $\mathcal{D} = \{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\} \subseteq \mathcal{X}$  that can be perfectly classified (or shattered) by some  $h \in \mathcal{H}$ . Formally, this means that for every one of the  $2^m$  possible assignments of binary labels to each feature vector in  $\mathcal{D}$ , there exists some hypothesis  $h \in \mathcal{H}$  that realizes this labeling. Intuitively, the VC dimension quantifies how well  $\mathcal{H}$  can fit arbitrary label assignments, and thus captures its approximate power. It plays a central role in deriving bounds on the generalization gap. Fig. 141 illustrates the definition of the VC dimension for a linear model  $\mathcal{H}^{(2)}$  with  $d = 2$  features. Fig. 141(a) and 141(b) show the same set of three noncollinear feature vectors under two different binary labelings. In both cases, a separating hyperplane exists that realizes the labeling. Since this holds for all  $2^3 = 8$  possible binary labelings of the three feature vectors, the set is shattered. Fig. 141(c) depicts four feature vectors with a specific labeling. No linear separator can correctly classify all data points in this case. Thus,  $\text{VCdim}(\mathcal{H}^{(2)}) = 3$ .

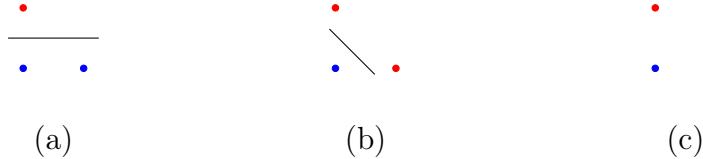


Fig. 141. Illustration of the VC dimension for a linear model  $\mathcal{H}^{(2)}$  that is used to learn a linear classifier in the feature space  $\mathbb{R}^2$ .

More generally, for a linear model  $\mathcal{H}^{(d)}$ , the VC dimension equals  $d + 1$ .

In other words, for linear models, the VC dimension essentially matches the dimension of the underlying parameter space  $\mathbb{R}^d$ . For more complex hypothesis spaces, such as decision trees or ANNs, the relation between VC dimension and the dimension of the feature space is far less direct. In these cases, alternative complexity measures, such as the Rademacher complexity, can be more useful for analyzing ERM-based methods.

See also: hypothesis space, Rademacher complexity, generalization, ML, effective dimension.

**vertical federated learning (VFL)** In VFL, different devices have access to different features of the same set of data points [216]. Formally, the underlying global dataset is

$$\mathcal{D}^{(\text{global})} := \{(\mathbf{x}^{(1)}, y^{(1)}), \dots, (\mathbf{x}^{(m)}, y^{(m)})\}.$$

We denote by  $\mathbf{x}^{(r)} = (x_1^{(r)}, \dots, x_{d'}^{(r)})^T$ , for  $r = 1, \dots, m$ , the complete feature vectors for the data points. Each device  $i \in \mathcal{V}$  observes only a subset  $\mathcal{F}^{(i)} \subseteq \{1, \dots, d'\}$  of features, resulting in a local dataset  $\mathcal{D}^{(i)}$

with feature vectors

$$\mathbf{x}^{(i,r)} = (x_{j_1}^{(r)}, \dots, x_{j_d}^{(r)})^T.$$

Some of the devices may also have access to the labels  $y^{(r)}$ , for  $r = 1, \dots, m$ , of the global dataset (see Fig. 142).

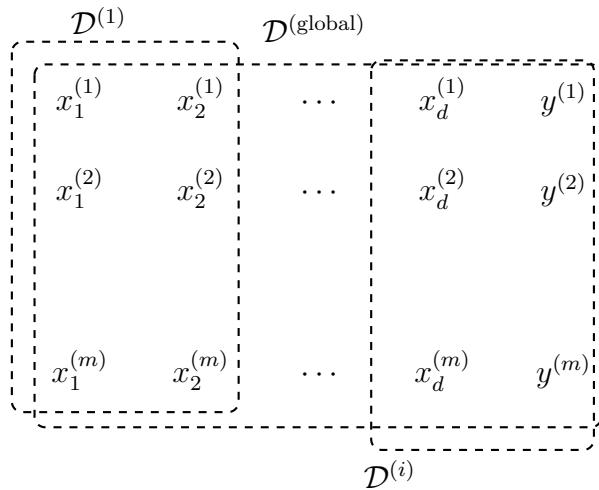


Fig. 142. VFL uses local datasets that are derived from the data points of a common global dataset. The local datasets differ in the choice of features used to characterize the data points.

One potential application of VFL is to enable collaboration between different healthcare providers. Each provider collects distinct types of measurements—such as blood values, electrocardiography, and lung X-rays—for the same patients. Another application is a national social insurance system, where health records, financial indicators, consumer behavior, and mobility data are collected by different institutions. VFL

enables joint learning across these parties while allowing well-defined levels of privacy protection. We can view VFL as a specific form of model parallelism.

See also: privacy protection, FL, model parallelism.

**weight** Consider a parameterized hypothesis space  $\mathcal{H}$ . We use the term weights for numeric model parameters that are used to scale features or their transformations in order to compute  $h^{(\mathbf{w})} \in \mathcal{H}$ . A linear model uses weights  $\mathbf{w} = (w_1, \dots, w_d)^T$  to compute the linear combination  $h^{(\mathbf{w})}(\mathbf{x}) = \mathbf{w}^T \mathbf{x}$ . Weights are also used in ANNs to form linear combinations of features or the outputs of neurons in hidden layers (see Fig. 143).

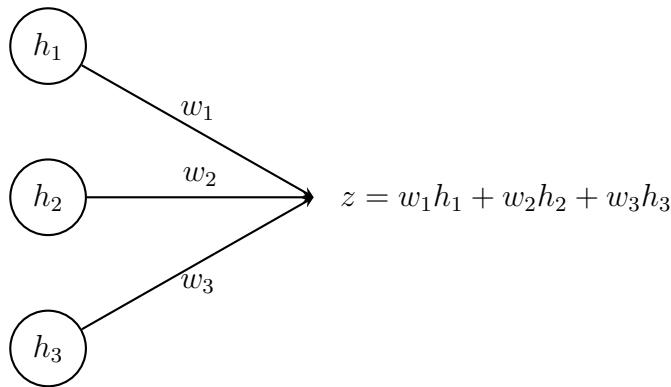


Fig. 143. A section of an ANN that contains a hidden layer with outputs (or activations)  $h_1$ ,  $h_2$ , and  $h_3$ . These outputs are combined linearly to compute  $z$ , which can be used either as output of the ANN or as input to another layer.

See also: hypothesis space, model parameter, feature, linear model,

ANN, layer, activation.

**weighted least squares** Weighted least squares refers to ERM-based methods that use the weighted average squared error loss

$$\frac{1}{m} \sum_{r=1}^m q^{(r)} (y^{(r)} - h(\mathbf{x}^{(r)}))^2$$

on a training set  $\mathcal{D} = \{ (\mathbf{x}^{(1)}, y^{(1)}) , \dots , (\mathbf{x}^{(m)}, y^{(m)}) \}$  to measure the quality of a hypothesis map  $h \in \mathcal{H}$ . The weights  $q^{(1)}, \dots, q^{(m)} \in \mathbb{R}_+$  allow us to emphasize or de-emphasize the contribution of individual data points in the training set. Ideally, we assign a small weight  $q^{(r)}$  to the  $r$ th data point if it is an outlier (see Fig. 144). We obtain different weighted least squares methods by using different models in ERM.

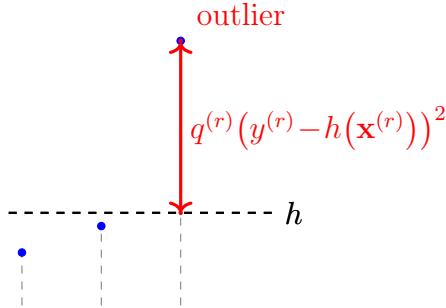


Fig. 144. Weighted least squares can be used to mitigate the effect of outlier data points in a training set.

See also: ERM, squared error loss, linear regression, linear model.

**zero-gradient condition** Consider the unconstrained optimization problem  $\min_{\mathbf{w} \in \mathbb{R}^d} f(\mathbf{w})$  with a smooth and convex objective function  $f(\mathbf{w})$ . A

necessary and sufficient condition for a vector  $\hat{\mathbf{w}} \in \mathbb{R}^d$  to solve this problem is that the gradient  $\nabla f(\hat{\mathbf{w}})$  is the zero vector  $\nabla f(\hat{\mathbf{w}}) = \mathbf{0}$  (see Fig. 145).

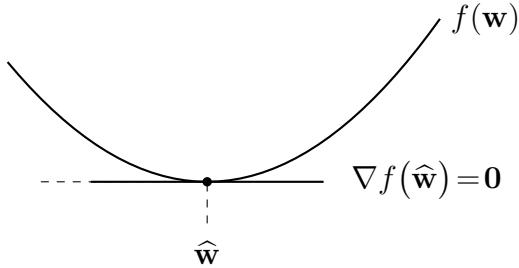


Fig. 145. A vector  $\hat{\mathbf{w}}$  solves the optimization problem if the gradient satisfies  $\nabla f(\hat{\mathbf{w}}) = \mathbf{0}$ .

In other words [25, p. 140],

$$\nabla f(\hat{\mathbf{w}}) = \mathbf{0} \Leftrightarrow f(\hat{\mathbf{w}}) = \min_{\mathbf{w} \in \mathbb{R}^d} f(\mathbf{w}).$$

By defining the gradient operator  $\nabla f : \mathbb{R}^d \rightarrow \mathbb{R}^d$ , we can rewrite the zero-gradient condition as a fixed-point equation:

$$(\mathcal{I} - \alpha \nabla f) \hat{\mathbf{w}} = \hat{\mathbf{w}}.$$

Here,  $\mathcal{I}$  denotes the identity operator (i.e.,  $\mathcal{I}(\mathbf{w}) = \mathbf{w}$ ) and  $\alpha$  is an arbitrary positive number.

See also: optimization problem, smooth, convex, objective function, vector, gradient.

**0/1 loss** The 0/1 loss  $L^{(0/1)}((\mathbf{x}, y), h)$  measures the quality of a classifier  $h(\mathbf{x})$  that delivers a prediction  $\hat{y}$  (e.g., via thresholding (10)) for the

label  $y$  of a data point with features  $\mathbf{x}$ . It is equal to 0 if the prediction is correct, i.e.,  $L^{(0/1)}((\mathbf{x}, y), h) = 0$  when  $\hat{y} = y$ . It is equal to 1 if the prediction is wrong, i.e.,  $L^{(0/1)}((\mathbf{x}, y), h) = 1$  when  $\hat{y} \neq y$ .

See also: loss, classifier, prediction, accuracy, label, data point, feature.

## Reinforcement Learning

**action** An action refers to a decision taken by an AI system at a given time step  $t$  that influences the observed reward signal. The actions are elements of an action space  $\mathcal{A}$  and are typically denoted by  $a_t \in \mathcal{A}$ . The action  $a_t$  is selected based on the feature vector  $\mathbf{x}^{(t)}$  (which collects all available observations) and the current hypothesis  $h^{(t)}$ . RL uses online learning methods to learn a hypothesis  $h^{(t)}$  that predicts a (nearly) optimal action. The usefulness of the prediction  $a_t$  is evaluated indirectly through the resulting reward signal  $r^{(t)}$ . In the special case of an MAB, the set of possible actions is finite and each action corresponds to selecting one arm. In more general RL settings, the action space may be continuous.

See also: reward, hypothesis, RL, MAB, loss function.

**action space** See action.

**agent** An agent denotes any system that implements some form of online learning [93], [217]. During each time step  $t$ , an agent receives a feature vector  $\mathbf{x}^{(t)}$  that provides (typically incomplete) information about the underlying state of the system. The agent then applies its current hypothesis  $h^{(t)}$  to select an action  $a^{(t)} = h^{(t)}(\mathbf{x}^{(t)})$ . It then receives a reward  $r^{(t)}$  that quantifies the usefulness of this action and, in turn, guides the adaptation of its hypothesis (or model parameter).

See also: hypothesis, action, reward, RL.

**Bellman operator** The Bellman operator  $\mathcal{F}$  associated with an MDP is

defined on the space of all value functions. In particular, it maps a value function  $v : \mathcal{S} \rightarrow \mathbb{R}$  to another value function  $v' : \mathcal{S} \rightarrow \mathbb{R}$  as follows:

$$v'(s) = \max_{a \in \mathcal{A}} \left( \mathbb{E}\{r(s, a) | s, a\} + \gamma \mathbb{E}\{v(s') | s, a\} \right),$$

where  $\gamma \in (0, 1)$  is a discount factor and  $s'$  is the next state generated according to the transition function  $s' \sim \mathbb{P}(s' | s, a)$ . The state-value function  $v^*$  of the optimal policy  $\pi^*$  is a fixed point of the Bellman operator,  $v^* = \mathcal{F}v^*$ . This fixed-point equation naturally lends itself to the value iteration method for computing the state-value function of an optimal policy. Besides the Bellman operator associated with an MDP, there is also a Bellman operator  $\mathcal{F}^{(\pi)}$  associated with a policy  $\pi$ . In this case, the Bellman operator is defined as

$$\mathcal{F}^{(\pi)}v(s) = \mathbb{E}\{r(s, a) | s, a\} + \gamma \mathbb{E}\{v(s') | s, a\},$$

where  $s' \sim \mathbb{P}(s' | s, a)$  and  $a$  is selected according to  $\pi$ . The state-value function  $v_\pi$  is a fixed point of  $\mathcal{F}^{(\pi)}$ ,  $v_\pi = \mathcal{F}^{(\pi)}v_\pi$ . This fixed-point equation can be solved by a fixed-point iteration that is known as policy evaluation. The Bellman operator is named after Richard Bellman, who introduced it in the context of dynamic programming [218]. The Bellman operator is a key concept in RL and is used to derive algorithms for solving MDPs, such as value iteration and policy iteration [93].

See also: MDP, value function, policy, value iteration, contractive operator, Banach's fixed-point theorem.

**environment (reinforcement learning)** An environment denotes the external system with which an agent interacts over time. During each

time step  $t$ , the environment provides the agent with a feature vector  $\mathbf{x}^{(t)}$  and, in response to the action  $a^{(t)}$  by the agent, a reward  $r^{(t)}$  [93]. See also: agent, feature vector, reward.

**Markov decision process (MDP)** An MDP is a mathematical structure for the study of RL. Formally, an MDP is a stochastic process that is defined by a specific choice for

- a state space  $\mathcal{S}$ ;
- an action space  $\mathcal{A}$ ;
- a transition function  $\mathbb{P}(s' | s, a)$  specifying the conditional probability distribution  $\mathbb{P}^{(s'|s,a)}$  over the next state  $s' \in \mathcal{S}$ , given the current state  $s \in \mathcal{S}$  and action  $a \in \mathcal{A}$ ;
- a reward function  $r(s, a) \in \mathbb{R}$  that assigns a numerical reward to each state-action pair  $(s, a)$ .

For a given policy  $\pi$ , these components define the probability distribution of a sequence

$$s_1, a_1, r_1, s_2, a_2, r_2, \dots, s_t, a_t, r_t$$

of RVs. The defining property of an MDP is the Markov property. That is, at time instant  $t$ , the conditional probability distribution of the next state  $s_{t+1}$  and reward  $r_t$  depends on the past only via the current state  $s_t$  and action  $a_t$ . RL methods try to learn a policy  $\pi$  that maximizes the expected return:

$$\mathbb{E} \left\{ \sum_{t=1}^{\infty} \gamma^{t-1} r_t \mid s_1 \right\}.$$

The conditioning on the initial state  $s_1$  indicates that the expected return is evaluated by following the policy  $\pi$  from a given initial state. The expected return involves the discount factor  $\gamma \in (0, 1)$  that determines the relative importance of future rewards compared to the immediate reward. The discount factor  $\gamma$  is typically fixed for a given MDP and controls the trade-off between short-term and long-term reward. MDPs are widely used in robotics, game playing, and autonomous systems to model decision-making problems where an agent interacts with an environment to achieve a goal [93], [94], [219].

See also: RL, stochastic process, function, reward.

**multiarmed bandit (MAB)** An MAB is a precise formulation of a sequential decision-making task under uncertainty. At each time step  $t$ , one must choose an action from a finite action space  $\mathcal{A}$ . Choosing action  $a$  at time  $t$  yields a reward  $r^{(a,t)}$ . Each MAB induces an ML problem, i.e., to learn a hypothesis that predicts the optimal action  $a_t$  at time  $t$ . This prediction must be based on the actions and rewards received up to time  $t - 1$  [93], [190].

See also: reward, regret.

**policy (reinforcement learning)** A policy is a function that specifies how the next action  $a_t$  in an MDP is chosen when the current state is  $s_t$ . Typically, a policy is stochastic, meaning that it defines a conditional probability distribution  $\mathbb{P}^{(a|s)}$  over the actions for a given current state. We can view a policy also as a hypothesis that uses features derived from the current state to predict the best next action [93].

See also: action, MDP, state.

**policy evaluation (reinforcement learning)** Policy evaluation refers to computing the state-value function  $v_\pi$  of a given policy  $\pi$  in an MDP. One widely used method, referred to as iterative policy evaluation, is based on the characterization of  $v_\pi$  as a fixed point of the Bellman operator  $\mathcal{F}^{(\pi)}$ . In particular, starting from an initial value function  $v_0$ , we iteratively apply the Bellman operator  $\mathcal{F}^{(\pi)}$  to obtain a sequence of value functions  $v_1, v_2, \dots$  as follows:

$$v_{t+1} = \mathcal{F}^{(\pi)}v_t, \quad t = 0, 1, 2, \dots$$

Under mild conditions, this fixed-point iteration converges to  $v_\pi$  as  $t \rightarrow \infty$  [93, Sec. 4.2].

See also: policy, state-value function, MDP.

**regret** The regret of a hypothesis  $h$  relative to another hypothesis  $h'$ , which serves as a baseline, is the difference between the loss incurred by  $h$  and the loss incurred by  $h'$  [138]. The baseline hypothesis  $h'$  is also referred to as an expert.

See also: baseline, loss, expert.

**reinforcement learning (RL)** RL refers to an online learning setting where we can only evaluate the usefulness of a single hypothesis (i.e., a specific choice of model parameters) at each time step  $t$ . In particular, RL methods apply the current hypothesis  $h^{(t)}$  to the feature vector  $\mathbf{x}^{(t)}$  of the newly received data point to predict the next action. The usefulness

of the resulting prediction  $h^{(t)}(\mathbf{x}^{(t)})$  is quantified by a reward signal  $r^{(t)}$  (see Fig. 146).

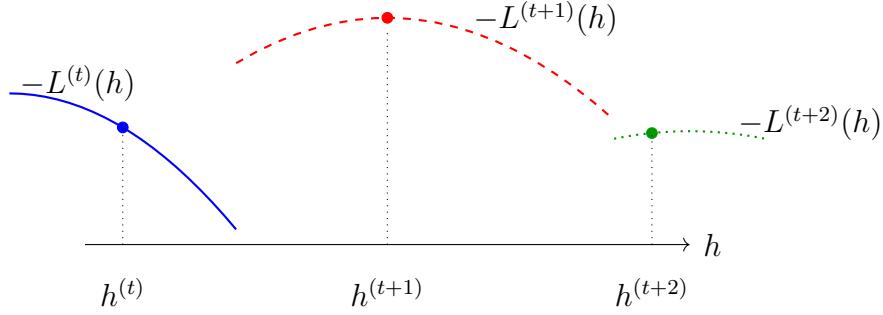


Fig. 146. Three consecutive time steps  $t, t + 1, t + 2$  with corresponding loss functions  $L^{(t)}, L^{(t+1)}, L^{(t+2)}$ . During time step  $t$ , an RL method can evaluate the loss function only for one specific hypothesis  $h^{(t)}$ , resulting in the reward signal  $r^{(t)} = -L^{(t)}(h^{(t)})$ .

In general, the reward depends also on the previous predictions  $h^{(t')}(x^{(t')})$  for  $t' < t$ . The goal of RL is to learn  $h^{(t)}$ , for each time step  $t$ , such that the (possibly discounted) cumulative reward is maximized [8], [93].

See also: reward, loss function, ML.

**state-value function** For a given MDP, any policy  $\pi$  naturally induces a value function  $v_\pi : \mathcal{S} \rightarrow \mathbb{R}$ . The value  $v_\pi(s)$  is the expected return when the MDP starts in a given state  $s \in \mathcal{S}$  and actions are selected according to  $\pi$ .

See also: MDP, value function, state.

**stochastic multiarmed bandit (stochastic MAB)** A stochastic MAB is

a stochastic process that is obtained from an MAB. In particular, the reward  $r^{(a,t)}$  is modeled as an RV with an unknown probability distribution  $\mathbb{P}^{(r^{(a,t)})}$  [139], [190]. In the simplest setting, the probability distribution  $\mathbb{P}^{(r^{(a,t)})}$  does not depend on  $t$ , i.e., it is time invariant.

See also: reward, regret.

**value function** In the context of an MDP, a value function  $v : \mathcal{S} \rightarrow \mathbb{R}$  assigns to each state  $s \in \mathcal{S}$  a real number  $v(s)$  that quantifies the long-term desirability of being in state  $s$ .

See also: state.

**value iteration** Consider an MDP with the associated Bellman operator  $\mathcal{F}$ . The state-value function  $v^*$  of the optimal policy is a fixed point of  $\mathcal{F}$ , i.e.,  $v^* = \mathcal{F}v^*$ . Value iteration is the fixed-point iteration for computing  $v^*$  by repeatedly applying  $\mathcal{F}$  to an initial value function  $v_0$  [93, Sec. 4.4].

See also: state-value function, fixed-point iteration, value function.

# Machine Learning Systems

**automaton** An automaton is a mathematical representation of a computing device whose behavior is described by a set of internal states, a memory structure, and a state-transition rule. Formally, an automaton consists of a state space, a set of admissible memory configurations, and a transition function that specifies how the current state and memory are updated in response to inputs [101]. The notion of an automaton is useful for the analysis of algorithms, such as those used in ML methods [100]. Collections of interacting automata can be used to study distributed algorithms, where each automaton represents a device that executes local computations and communicates with other devices [82], [220].

See also: device, state, algorithm.

**checkpoint** A checkpoint is a saved representation of the state of a running computation [221]. In an ML system, a checkpoint typically includes the current model parameters during model training [222].

See also: state, training.

**checkpointing** Checkpointing is a fault-tolerance mechanism that periodically creates checkpoints by saving the state of a running computation to persistent storage. Checkpointing is essential for fault-tolerant executions on revocable resources such as spot instances [221], [223].

See also: checkpoint, spot instance.

**cloud computing** Cloud computing is a computing paradigm in which computational resources such as processing, storage, and networking are

provided as on-demand services over a communication network [224], [225], [226]. In ML, cloud computing systems are commonly used to host large datasets and to execute ML algorithms. In contrast to FL systems, cloud computing typically centralizes data and computation within provider-managed data centers.

See also: FL system, ML system.

**cross-sectional data** Cross-sectional data consist of data points whose attributes are measured once, without explicitly modeling temporal evolution [227]. In ML, cross-sectional data arise in image classification, where each image is treated as an individual data point [83].

See also: data, data point.

**dynamical system** A dynamical system is an abstract system whose output depends on an internal state that evolves over time according to a state-update rule [228]. In discrete time, a dynamical system is commonly described by an iteration of the form  $\mathbf{s}^{(t+1)} = \mathcal{F}\mathbf{s}^{(t)}$ , where  $\mathbf{s}^{(t)}$  denotes the state at time  $t$  and  $\mathcal{F}$  is a state-transition map. In continuous time, dynamical systems are described by differential equations.

See also: output, state.

**early exit (deep learning)** Early exit methods refer to computational strategies for evaluating the prediction of a deep net. The idea is to terminate inference before evaluating all layers of a deep net [229].

See also: prediction, deep net, inference.

**edge computing** Edge computing refers to the placement of computation and data storage close to the sources of data generation, such as sensors, mobile devices, or embedded systems, rather than in centralized data centers [230]. In ML, edge computing supports low-latency inference and reduced communication by executing parts of ML algorithms on or near data-generating devices [231].

See also: cloud computing, FL system, ML system.

**edge device** An edge device operates at or near the edge of a communication network [230]. The term edge refers to the periphery of the network, where data is produced and first processed. In ML and, in particular, in FL, an edge device typically corresponds to a node of an FL network. Each edge device stores local data and implements parts of the machine learning pipeline (ML pipeline), such as data preprocessing, local model training, or inference [232].

See also: device, edge computing.

**federated learning system (FL system)** An FL system is a distributed ML system in which multiple computational devices collaborate to train ML models without sharing their raw local data. An FL system is characterized by a communication network that specifies which devices can exchange information. Conceptually, an FL system is distinct from an FL algorithm [224]. The system specifies the participating entities, their interconnections, and execution constraints, while the algorithm specifies the update rules for local and global model parameters [220], [233]. Typical information exchanged in an FL system includes model

parameters or gradient information, but not raw data.

See also: FL, ML system, algorithm, FL network.

**longitudinal data** Longitudinal data consist of data points whose attributes are measured repeatedly over time [234]. In ML, longitudinal data are common in applications such as healthcare, where patient measurements are taken at multiple time points [235].

See also: data, data point.

**machine learning as a service (MLaaS)** MLaaS refers to a cloud computing service model in which ML capabilities are provided to users via standardized network interfaces. In this model, the cloud provider manages the underlying computing infrastructure, data storage, and software platforms, while users access functionality such as model training and inference without direct control over physical resources [225].

See also: cloud computing, ML system.

**machine learning pipeline (ML pipeline)** The term ML pipeline refers to a composition (i.e., concatenation) of several functions within an ML system. The individual functions include data preprocessing, feature learning, model training, and inference. By combining them, an ML system turns raw data into predictions [236].

See also: function, ML system.

**machine learning system (ML system)** An ML system consists of computational devices that can gather and store data, execute algorithms, and exchange information via communication networks. Examples of

the exchanged information include data or updates of model parameters. Conceptually, an ML system is distinct from an ML algorithm, i.e., an algorithm specifies the abstract computational procedure (e.g., an optimization method), while the system specifies how this procedure is realized in practice [101], [233], [237]. Examples of algorithms executed by devices within an ML system include gradient-based methods for solving ERM problems.

See also: ML, device.

**mobile device** A mobile device is a portable computing device equipped with computational, storage, sensing, and wireless communication capabilities [238], [239]. Examples of mobile devices include smartphones, tablets, or wearables. Mobile devices can act as data sources and provide computational infrastructure for edge computings or FL systems.

See also: edge computing, FL system, ML system.

**preprocessing** Preprocessing refers to the set of operations applied to raw data before they are fed into an ML algorithm [83]. The goal of preprocessing is to transform the data into a form that is more suitable for follow-up stages of an ML pipeline. Typical preprocessing steps include cleaning corrupted or missing values, normalizing or scaling features, or encoding categorical variables [236].

See also: data, ML pipeline, feature.

**spot instance** A spot instance is a type of cloud computing service that provides computational resources at a reduced cost but without guarantees

on availability [223]. In particular, a spot instance may be revoked by the provider at any time, which requires the use of checkpointing [240]. See also: cloud computing, checkpointing.

# Machine Learning Regulation

**artificial intelligence system (AI system)** The EU Artificial Intelligence

Act (AI Act) [241] defines an AI system as a machine-based system that is designed to operate with varying levels of autonomy and that may exhibit adaptiveness (e.g., model retraining) after deployment. AI systems compute predictions that can influence environments or decisions [242]. In line with this definition, regulatory obligations and risk classifications apply at the level of the AI system rather than at the level of individual models or algorithms. The system-level view emphasizes that properties such as robustness, fairness, and transparency emerge from the interaction of models, data, and operational context, rather than from isolated components.

See also: AI, robustness, transparency.

**automated decision-making** Automated decision-making refers to ML ap-

plications that use predictions delivered by a trained model directly (i.e., without human involvement) to make decisions affecting individuals. Under the GDPR, individuals have the right not to be subject to decisions based solely on automated processing, when these decisions produce legal or similarly significant effects, unless appropriate safeguards (e.g., human oversight, contestability, or explicit consent) are implemented.

See also: ML, GDPR.

**deep fake** Deep fakes are synthetic media generated or substantially modified by an AI system such that it falsely appears to depict a real person,

object, or event. Deep fakes are typically produced using generative methods, trained to imitate visual, audio, or audiovisual characteristics of real data. From a system perspective, deep fakes are characterized by a deliberate mismatch between the observable content and its true origin, which can lead to deception, misinformation, or manipulation.

See also: AI system.

**FAIR principles** The FAIR principles are guidelines for scientific data management. The aim is to make research artifacts findable, accessible, interoperable, and reusable [243]. FAIR-compliant metadata is a key enabler of auditability, as it supports the traceability and reproducible inspection of an ML system [244].

See also: data, trustworthy AI.

**general data protection regulation (GDPR)** The GDPR was enacted by the European Union (EU), effective from 25 May 2018 [122]. It safeguards the privacy and data rights of individuals in the EU. The GDPR has significant implications for how data are collected, stored, and used in ML applications. Key provisions include the following:

- Data minimization principle: ML systems should only use the necessary amount of personal data for their purpose.
- Transparency and explainability: ML systems should enable their users to understand how the systems make decisions that impact the users.

- Data subject rights: Users should get an opportunity to access, rectify, and delete their personal data, as well as to object to automated decision-making and profiling.
- Accountability: Organizations must ensure robust data security and demonstrate compliance through documentation and regular audits.

See also: data, ML, data minimization principle, transparency, explainability.

**high-risk artificial intelligence system (high-risk AI system)** A subset of AI systems is classified as high-risk due to its potential to significantly impact safety, fundamental rights, or critical societal functions. High-risk AI systems are subject to stringent regulatory requirements under the EU AI Act, including conformity assessments, risk management, transparency obligations, and post-market monitoring [242]. Examples of high-risk AI systems include those used in critical infrastructure, education, employment, law enforcement, and biometric identification.

See also: AI system, transparency.

**personal data** Personal data are any information relating to an identified or identifiable natural person (i.e., the data subject). A natural person is identifiable if they can be identified, directly or indirectly, in particular by reference to an identifier such as a name, an identification number, location data, an online identifier, or one or more factors specific to the physical, physiological, genetic, mental, economic, cultural, or social

identity of that person [122]. In ML systems, personal data may occur in training data, model inputs, intermediate representations (e.g., feature vectors or embeddings), or model outputs, provided that the information relates to an identifiable natural person. The EU AI Act does not introduce a separate definition of personal data; instead, whenever an AI system processes personal data, the GDPR definition and obligations apply in full.

See also: data, AI system, GDPR.

**profiling** Profiling aims at identifying patterns and making inferences about individuals based on their data. Profiling techniques use ML methods to predict individuals' performance at work, economic situation, health, or personal preferences. Profiling is instrumental in targeted advertising, credit scoring, fraud detection, and personalized services. The GDPR imposes strict requirements on organizations that engage in profiling activities to ensure that individuals' rights are protected [122].

See also: data, GDPR.

**SHapley Additive exPlanations (SHAP)** SHAP is a post hoc method for explaining the prediction  $\hat{h}(\mathbf{x})$  of a trained model  $\hat{h} \in \mathcal{H}$  at a given feature vector  $\mathbf{x} = (x_1, \dots, x_d)^T$ . SHAP values are computed after model training and can be used to analyze the relative importance of different features for the prediction  $\hat{h}(\mathbf{x})$  [145].

See also: prediction, training.

**transparency** Transparency is a fundamental requirement for trustworthy

AI [245]. In the context of ML methods, transparency is often used interchangeably with explainability [142], [246]. However, in the broader scope of AI systems, transparency extends beyond explainability and includes providing information about the system’s limitations, reliability, and intended use. In medical diagnosis systems, transparency requires disclosing the confidence level for the predictions delivered by a trained model. In credit scoring, AI-based lending decisions should be accompanied by explanations of contributing factors, such as income level or credit history. These explanations allow humans (e.g., a loan applicant) to understand and contest automated decisions. Some ML methods inherently offer transparency. For example, logistic regression provides a quantitative measure of classification reliability through the value  $|h(\mathbf{x})|$ . Decision trees are another example, as they allow human-readable decision rules [144]. Transparency also requires a clear indication when a user is engaging with an AI system. For example, AI-powered chatbots should notify users that they are interacting with an automated system rather than a human. Furthermore, transparency encompasses comprehensive documentation detailing the purpose and design choices underlying the AI system. For instance, model datasheets [131] and AI system cards [247] help practitioners understand the intended use cases and limitations of an AI system [248].

See also: trustworthy AI, explainability.

**trustworthy artificial intelligence (trustworthy AI)** Besides the computational aspects and statistical aspects, a third main design aspect of ML methods is their trustworthiness [249]. The EU has put forward

seven key requirements (KRs) for trustworthy AI (which typically build on ML methods) [250]:

- 1) KR1 – Human agency and oversight;
- 2) KR2 – Technical robustness and safety;
- 3) KR3 – Privacy and data governance;
- 4) KR4 – Transparency;
- 5) KR5 – Diversity, nondiscrimination and fairness;
- 6) KR6 – Societal and environmental well-being;
- 7) KR7 – Accountability.

See also: computational aspect, statistical aspect, ML, AI, robustness, data, transparency.

# Index

- 0/1 loss, 310  
 $\sigma$ -algebra, 134  
 $\sigma$ -field, 134  
 $k$ -fold cross-validation ( $k$ -fold CV), 224  
 $k$ -means, 224  
absolute error loss, 148  
accuracy, 149  
action, 312  
action space, 312  
activation, 149  
activation function, 150  
adaptive boosting (AdaBoost), 150  
agent, 312  
algebraic connectivity, 22  
algorithm, 150  
alternating direction method of multi-  
pliers (ADMM), 22  
application programming interface (API)  
area under the curve (AUC), 152  
artificial intelligence (AI), 153  
artificial intelligence system (AI sys-  
tem), 325  
artificial neural network (ANN), 153  
attack, 154  
attention, 155  
augmented Lagrangian, 23  
autoencoder, 156  
automated decision-making, 325  
automaton, 319  
average node degree, 23  
backdoor, 157  
backpropagation, 157  
bagging, 158  
Banach's fixed-point theorem, 23  
base learner, 159  
baseline, 160  
basis, 24  
batch, 161  
batch learning, 162  
Bayes estimator, 162  
Bayes risk, 163  
Bellman operator, 312  
bias, 163  
binary classification, 164  
binary cross-entropy (BCE), 164  
boosting, 164  
bootstrap, 165

bootstrap aggregation, 166  
boundary, 24  
Bregman divergence, 25  
Brier score, 166  
  
Cauchy sequence, 26  
Cauchy-Schwarz inequality, 27  
central limit theorem (CLT), 28  
characteristic function, 29  
Chebyshev's inequality, 29  
checkpoint, 319  
checkpointing, 319  
Chernoff bound, 30  
classification, 166  
classifier, 166  
cloud computing, 319  
cluster, 167  
cluster centroid, 168  
clustered federated learning (CFL),  
    169  
clustering, 169  
clustering assumption, 169  
clustering error, 170  
co-domain, 31  
column space, 31  
computational aspect, 171  
  
concentration inequality, 31  
concept activation vector (CAV), 171  
condition number, 32  
conditional expectation, 32  
conditional probability distribution,  
    32  
conditional probability mass function  
    (conditional pmf), 33  
confusion matrix, 171  
conjugate prior, 33  
conjugate transpose, 34  
connected, 34  
constrained optimization problem, 35  
continuous, 35  
contractive operator, 36  
convergence, 37  
convex, 38  
convex clustering, 172  
convex optimization, 39  
convex optimization problem, 41  
coreset, 172  
countable, 41  
Courant–Fischer–Weyl min–max char-  
acterization (CFW min–max  
characterization), 42

covariance, 42  
covariance function, 43  
covariance matrix, 43  
covariate, 173  
cross-entropy, 173  
cross-sectional data, 320  
cumulative distribution function (cdf),  
44  
data, 174  
data augmentation, 174  
data imputation, 175  
data matrix, 175  
data minimization principle, 175  
data normalization, 176  
data parallelism, 176  
data point, 176  
data poisoning, 179  
dataset, 179  
decision boundary, 181  
decision region, 182  
decision tree, 182  
deep fake, 325  
deep learning, 183  
deep net, 183  
degree of belonging, 183  
denial-of-service attack, 184  
denoising autoencoder, 44  
density-based spatial clustering of ap-  
plications with noise (DBSCAN),  
184  
derivative, 45  
design matrix, 184  
determinant, 45  
device, 185  
diagnosis, 185  
diagonalizable, 46  
differentiable, 46  
differential entropy, 46  
differential privacy (DP), 186  
dimension, 47  
dimensionality reduction, 187  
directed acyclic graph (DAG), 47  
directed cycle, 48  
directed graph, 49  
discrepancy, 188  
discrete random variable (discrete RV),  
49  
distance, 49  
distributed algorithm, 188  
domain, 50

- dual norm, 50
- duality gap, 52
- dynamical system, 320
- early exit (deep learning), 320
- early stopping, 189
- edge computing, 321
- edge device, 321
- edge weight, 190
- effective dimension, 190
- eigenvalue, 52
- eigenvalue decomposition (EVD), 52
- eigenvector, 53
- embedding, 191
- empirical distribution, 53
- empirical risk, 191
- empirical risk minimization (ERM), 191
  - FAIR principles, 326
  - feature, 198
  - feature learning, 198
  - feature map, 199
  - feature matrix, 200
  - feature space, 200
  - feature vector, 201
  - federated averaging (FedAvg), 202
  - federated gradient descent (FedGD), 202
- encoder, 192
- ensemble, 192
- entropy, 54
- environment, 313
- epigraph, 55
- epoch, 194
- Erdős–Rényi graph (ER graph), 55
- estimation error, 194
- estimator, 56
- Euclidean distance, 56
- Euclidean norm, 56
- Euclidean space, 56
- event, 57
- expectation, 194
- expectation–maximization (EM), 57
- expert, 195
- explainability, 196
- explainable empirical risk minimization (EERM), 196
- explainable machine learning (XML), 196
- explanation, 197
- exponential family, 59

federated learning (FL),	203	general data protection regulation (GDPR),
federated learning network (FL net-		326
work),	203	generalization, 206
federated learning system (FL system),		generalization gap, 208
321		generalized additive model (GAM),
federated proximal (FedProx),	204	209
federated stochastic gradient descent		generalized total variation (GTV), 210
(FedSGD),	204	generalized total variation minimiza-
FedRelax,	204	tion (GTVMin), 210
field,	60	geometric median (GM), 69
Finnish Meteorological Institute (FMI),		gradient, 69
205		gradient boosting, 210
firmly non-expansive operator,	60	gradient descent (GD), 211
fixed point,	60	gradient step, 70
fixed-point characterization,	61	gradient-based method, 212
fixed-point equation,	61	graph, 72
fixed-point iteration,	62	graph clustering, 212
flow-based clustering,	205	graph neural network, 213
full-rank,	64	graph of a function, 72
function,	65	group, 73
Gaussian,	66	halfspace, 73
Gaussian mixture model (GMM),	205	hard clustering, 213
Gaussian process (GP),	66	Hermitian, 73
Gaussian random variable (Gaussian		Hessian, 73
RV),	67	high-dimensional regime, 214

high-risk artificial intelligence system	input vector, 220
(high-risk AI system), 327	integrable, 79
Hilbert space, 75	interpretability, 220
hinge loss, 214	inverse matrix, 79
histogram, 215	iteration, 222
Hoeffding's inequality, 75	Jacobi method, 222
horizontal federated learning (HFL),	Jacobian matrix, 80
216	Johnson–Lindenstrauss lemma (JL lemma),
Huber loss, 217	81
Huber regression, 217	k-nearest neighbors regression (KNNR),
hyperparameter, 217	225
hyperplane, 76	Karush–Kuhn–Tucker conditions (KKT
hypothesis, 218	conditions), 82
hypothesis space, 218	kernel (kernel method), 225
image segmentation, 220	kernel (linear map), 83
independent and identically distributed	kernel method, 226
(i.i.d.), 77	Kronecker product, 227
independent and identically distributed	Kullback–Leibler divergence (KL di-
assumption (i.i.d. assumption),	vergence), 83
220	label, 228
indicator function, 77	label space, 228
inference, 220	label vector, 229
infimum (or greatest lower bound), 78	labeled data point, 230
injective, 78	Lagrange dual function, 84
inner product, 78	

Lagrangian, 86  
Laplacian matrix, 87  
large language model (LLM), 230  
law of large numbers, 88  
layer, 230  
learning rate, 231  
learning task, 232  
least absolute deviation regression, 234  
least absolute shrinkage and selection operator (Lasso), 234  
least squares, 235  
leave-one-out cross-validation (LOO-CV), 235  
Lebesgue integral, 88  
likelihood function, 89  
linear classifier, 235  
linear discriminant analysis (LDA), 236  
linear least squares, 236  
linear map, 90  
linear model, 237  
linear operator, 90  
linear regression, 239  
linearly dependent, 90  
linearly independent, 91  
Lipschitz continuity, 91  
Lloyd's algorithm, 241  
local dataset, 242  
local interpretable model-agnostic explanations (LIME), 243  
local model, 244  
locally weighted learning (LWL), 244  
logistic loss, 244  
logistic regression, 245  
longitudinal data, 322  
loss, 245  
loss function, 246  
machine learning (ML), 247  
machine learning as a service (MLaaS), 322  
machine learning pipeline (ML pipeline), 322  
machine learning system (ML system), 322  
machine unlearning, 248  
majorize-minimize (MM), 91  
map, 92  
marginal distribution, 92  
marginalization, 93  
Markov chain, 93

Markov decision process (MDP), 314  
Markov property, 94  
Markov's inequality, 94  
matrix, 95  
maximum, 96  
maximum likelihood, 96  
mean, 96  
mean absolute error (MAE), 248  
mean squared error (MSE), 248  
mean squared estimation error (MSEE), 249  
measurable, 97  
measure, 98  
measure space, 99  
median, 99  
membership inference attack, 249  
method of multipliers (MoM), 101  
metric, 102  
metric space, 103  
minimum, 104  
mirror descent, 104  
missing data, 249  
mobile device, 323  
model (machine learning), 250  
model inversion, 251  
model parallelism, 251  
model parameter, 252  
model selection, 253  
moment generating function (MGF), 105  
multi-label classification, 253  
multiarmed bandit (MAB), 315  
multilayer perceptron, 253  
multitask learning, 253  
multivariate normal distribution, 106  
mutual information (MI), 254  
natural language processing (NLP), 254  
nearest neighbor (NN), 254  
neighbor, 255  
neighborhood, 107  
nested cross-validation (nested CV), 255  
network Lasso, 255  
networked data, 255  
networked exponential families (nExp-Fam), 256  
networked federated learning (NFL), 256  
networked model, 256

Newton's method, 107	outlier, 262
node degree, 108	output, 263
non-expansive operator, 109	output vector, 263
non-smooth, 109	overfitting, 263
nonparametric, 256	parameter, 263
norm, 110	parameter space, 264
normal distribution, 111	parametric model, 264
normal equations, 256	partial derivative, 115
normal matrix, 111	penalty term, 265
normal vector, 111	perceptron, 266
normed space, 111	perplexity, 267
nullspace, 111	personal data, 327
objective function, 257	policy, 315
online algorithm, 258	policy evaluation, 316
online gradient descent (online GD), 259	polynomial regression, 267
online learning, 260	positive semi-definite (psd), 115
operator, 113	posterior (prediction), 116
optimism in the face of uncertainty, 261	posterior distribution, 116
optimization, 113	precision, 267
optimization method, 114	prediction, 267
optimization problem, 114	predictor, 268
orthogonality condition, 114	preimage, 116
outcome, 115	preprocessing, 323
	principal component analysis (PCA), 268

prior distribution, 116	Rademacher complexity, 271
privacy attack, 269	random experiment, 125
privacy funnel, 269	random forest, 272
privacy leakage, 269	random geometric graph (RGG), 127
privacy protection, 270	random projection, 272
probabilistic model, 117	random variable (RV), 128
probabilistic principal component analysis (PPCA), 117	rank, 129
probability, 270	rank-deficient, 129
probability density function (pdf), 117	realization, 273
probability distribution, 118	recall, 273
probability mass function (pmf), 118	receiver operating characteristic (ROC), 274
probability simplex, 120	rectified linear unit (ReLU), 274
probability space, 120	recurrent neural network (RNN), 274
profiling, 328	regression, 275
projected gradient descent (projected GD), 121	regret, 316
projection, 121	regularization, 275
proximable, 122	regularized empirical risk minimization (RERM), 277
proximal operator, 122	regularized loss minimization (RLM), 278
pseudocontractive operator, 123	regularizer, 278
pseudoinverse, 124	reinforcement learning (RL), 316
Q-learning, 271	relational model, 278
quadratic function, 271	reproducing kernel Hilbert space (RKHS),
Rényi divergence, 130	

130	singular value, 135
response, 279	singular value decomposition (SVD), 135
response vector, 279	skip connection, 285
reward, 279	smooth, 135
ridge regression, 280	soft clustering, 286
risk, 281	soft label, 286
risk stratification, 281	softmax function, 287
robustness, 281	spectral clustering, 287
sample, 131	spectral decomposition, 137
sample covariance matrix, 132	spectrogram, 289
sample mean, 282	spectrum, 137
sample size, 282	spot instance, 323
sample space, 282	squared error loss, 290
sample weighting, 282	stability, 290
scatterplot, 283	stacking, 291
Schur decomposition, 133	standard deviation, 137
self-supervised learning, 284	standard normal random vector, 137
semi-supervised learning (SSL), 284	state, 137
sensitive attribute, 285	state space, 138
sensitivity, 285	state-value function, 317
sequence, 133	statistical aspect, 292
SHapley Additive exPlanations (SHAP), 328	step size, 292
similarity graph, 285	stochastic, 138
simple function, 134	stochastic algorithm, 292

stochastic block model (SBM), 293  
stochastic gradient descent (SGD), 293  
stochastic multiarmed bandit (stochastic MAB), 317  
stochastic process, 138  
stopping criterion, 294  
stratification, 295  
stratum, 296  
strictly convex, 139  
strongly convex, 139  
structural risk minimization (SRM), 296  
subgradient, 140  
subgradient descent, 297  
subspace, 140  
support vector machine (SVM), 297  
supporting hyperplane, 141  
supremum (or least upper bound), 141  
symmetric matrix, 141  
tabular data, 298  
tall matrix, 141  
target, 298  
target vector, 299  
test set, 299  
token, 299  
total variation, 142  
trace, 142  
training, 299  
training error, 300  
training set, 300  
transfer learning, 300  
transformer, 300  
transparency, 328  
transpose, 143  
trustworthy artificial intelligence (trustworthy AI), 329  
typical set, 143  
unbiased estimator, 143  
uncertainty, 301  
underfitting, 301  
undirected graph, 143  
unitary matrix, 143  
upper confidence bound (UCB), 302  
validation, 303  
validation error, 304  
validation set, 304  
value function, 318  
value iteration, 318  
Vapnik–Chervonenkis dimension (VC dimension), 305

variance, 144  
vector, 144  
vector space, 145  
vertical federated learning (VFL), 306  
  
weight, 308  
weighted graph, 146  
weighted least squares, 309  
wide matrix, 147  
  
zero-gradient condition, 309

## References

- [1] W. Rudin, *Real and Complex Analysis*, 3rd ed. New York, NY, USA: McGraw-Hill, 1987.
- [2] W. Rudin, *Principles of Mathematical Analysis*, 3rd ed. New York, NY, USA: McGraw-Hill, 1976.
- [3] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 4th ed. Baltimore, MD, USA: The Johns Hopkins Univ. Press, 2013.
- [4] G. H. Golub and C. F. Van Loan, “An analysis of the total least squares problem,” *SIAM J. Numer. Anal.*, vol. 17, no. 6, pp. 883–893, Dec. 1980, doi: 10.1137/0717073.
- [5] A. Klenke, *Probability Theory: A Comprehensive Course*, 3rd ed. Cham, Switzerland: Springer Nature, 2020.
- [6] P. Billingsley, *Probability and Measure*, 2nd ed. New York, NY, USA: Wiley, 1986.
- [7] D. P. Bertsekas and J. N. Tsitsiklis, *Introduction to Probability*, 2nd ed. Belmont, MA, USA: Athena Scientific, 2008.
- [8] A. Jung, *Machine Learning: The Basics*. Singapore, Singapore: Springer Nature, 2022.
- [9] F. R. K. Chung, *Spectral Graph Theory*. Providence, RI, USA: American Mathematical Society, 1997.

- [10] D. A. Spielman, *Spectral and Algebraic Graph Theory (Incomplete Draft)*. Ebook, 2025, Accessed: December 9, 2025. [Online]. Available: <http://cs-www.cs.yale.edu/homes/spielman/sagt>
- [11] C. T. Kelley, *Iterative Methods for Linear and Nonlinear Equations*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 1995.
- [12] A. W. van der Vaart, *Asymptotic Statistics*. Cambridge, U.K.: Cambridge Univ. Press, 1998.
- [13] G. B. Folland, *Real Analysis: Modern Techniques and Their Applications*, 2nd ed. New York, NY, USA: Wiley, 1999.
- [14] W. Feller, *An Introduction to Probability Theory and Its Applications*, vol. 1, 3rd ed. New York, NY, USA: Wiley, 1967.
- [15] R. Vershynin, *High-Dimensional Probability: An Introduction with Applications in Data Science*. Cambridge, U.K.: Cambridge Univ. Press, 2018.
- [16] M. J. Wainwright, *High-Dimensional Statistics: A Non-Asymptotic Viewpoint*. Cambridge, U.K.: Cambridge Univ. Press, 2019.
- [17] R. Durrett, *Probability: Theory and Examples*, 4th ed. Cambridge, U.K.: Cambridge Univ. Press, 2010.
- [18] R. M. Gray, *Probability, Random Processes, and Ergodic Properties*, 2nd ed. New York, NY, USA: Springer Science+Business Media, 2009.

- [19] S. Ross, *A First Course in Probability*, 9th ed. Boston, MA, USA: Pearson Education, 2014.
- [20] O. Kallenberg, *Foundations of Modern Probability*. New York, NY, USA: Springer-Verlag, 1997.
- [21] D. Barber, *Bayesian Reasoning and Machine Learning*. Cambridge, U.K.: Cambridge Univ. Press, 2012.
- [22] C. M. Bishop, *Pattern Recognition and Machine Learning*. New York, NY, USA: Springer Science+Business Media, 2006.
- [23] H. H. Bauschke and P. L. Combettes, *Convex Analysis and Monotone Operator Theory in Hilbert Spaces*. New York, NY, USA: Springer Science+Business Media, 2011.
- [24] N. Parikh and S. Boyd, “Proximal algorithms,” *Found. Trends Optim.*, vol. 1, no. 3, pp. 127–239, Jan. 2014, doi: 10.1561/2400000003.
- [25] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge, U.K.: Cambridge Univ. Press, 2004.
- [26] P. R. Halmos, *Naive Set Theory*. New York, NY, USA: Springer-Verlag, 1974.
- [27] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning*. Cambridge, MA, USA: MIT Press, 2006.
- [28] R. B. Ash, *Probability and Measure Theory*, 2nd ed. San Diego, CA, USA: Academic, 2000.

- [29] A. Papoulis and S. U. Pillai, *Probability, Random Variables, and Stochastic Processes*, 4th ed. New York, NY, USA: McGraw-Hill Higher Education, 2002.
- [30] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.
- [31] H. J. Dirschmid, *Tensors and Fields*, (in German). Vienna, Austria: Springer-Verlag, 1996.
- [32] G. Strang, *Computational Science and Engineering*. Wellesley, MA, USA: Wellesley-Cambridge Press, 2007.
- [33] R. A. Horn and C. R. Johnson, *Matrix Analysis*, 2nd ed. New York, NY, USA: Cambridge Univ. Press, 2013.
- [34] S. Axler, *Linear Algebra Done Right*, 3rd ed. Cham, Switzerland: Springer Nature, 2015.
- [35] T. M. Cover and J. A. Thomas, *Elements of Information Theory*, 2nd ed. Hoboken, NJ, USA: Wiley, 2006.
- [36] G. Strang, *Introduction to Linear Algebra*, 5th ed. Wellesley, MA, USA: Wellesley-Cambridge Press, 2016.
- [37] M. E. J. Newman, *Networks: An Introduction*. New York, NY, USA: Oxford Univ. Press, 2010.
- [38] H. H. Sohrab, *Basic Real Analysis*, 2nd ed. Birkhäuser, 2014.

- [39] D. P. Bertsekas, A. Nedic, and A. E. Ozdaglar, *Convex Analysis and Optimization*. Belmont, MA, USA: Athena Scientific, 2003.
- [40] P. Erdős and A. Rényi, “On random graphs I,” *Publ. Math. Debrecen*, vol. 6, no. 3–4, pp. 290–297, 1959, doi: 10.5486/PMD.1959.6.3-4.12.
- [41] E. N. Gilbert, “Random graphs,” *Ann. Math. Statist.*, vol. 30, no. 4, pp. 1141–1144, Dec. 1959.
- [42] E. L. Lehmann and G. Casella, *Theory of Point Estimation*, 2nd ed. New York, NY, USA: Springer-Verlag, 1998.
- [43] S. M. Kay, *Fundamentals of Statistical Signal Processing: Estimation Theory*. Upper Saddle River, NJ, USA: Prentice-Hall, 1993.
- [44] P. R. Halmos, *Finite-Dimensional Vector Spaces*. New York, NY, USA: Springer-Verlag, 1974.
- [45] A. P. Dempster, N. M. Laird, and D. B. Rubin, “Maximum likelihood from incomplete data via the EM algorithm,” *J. Roy. Statist. Soc.: Ser. B (Methodological)*, vol. 39, no. 1, pp. 1–22, Sep. 1977, doi: 10.1111/j.2517-6161.1977.tb01600.x.
- [46] K. P. Murphy, *Machine Learning: A Probabilistic Perspective*. Cambridge, MA, USA: MIT Press, 2012.
- [47] A. Chambolle and T. Pock, “An introduction to continuous optimization for imaging,” *Acta Numer.*, vol. 25, pp. 161–319, May 2016, doi: 10.1017/S096249291600009X.

- [48] H. H. Bauschke and P. L. Combettes, *Convex Analysis and Monotone Operator Theory in Hilbert Spaces*, 2nd ed. New York, NY, USA: Springer Science+Business Media, 2017.
- [49] V. I. Istrățescu, *Fixed Point Theory: An Introduction*. Dordrecht, The Netherlands: D. Reidel, 1981.
- [50] A. Lapidoth, *A Foundation in Digital Communication*. Cambridge, U.K.: Cambridge Univ. Press, 2009.
- [51] H. P. Lopuhaä and P. J. Rousseeuw, “Breakdown points of affine equivariant estimators of multivariate location and covariance matrices,” *Ann. Statist.*, vol. 19, no. 1, pp. 229–248, Mar. 1991, doi: 10.1214/aos/1176347978.
- [52] Y. Nesterov, *Introductory Lectures on Convex Optimization: A Basic Course*. Boston, MA, USA: Kluwer Academic, 2004.
- [53] R. T. Rockafellar, *Network Flows and Monotropic Optimization*. Belmont, MA, USA: Athena Scientific, 1998.
- [54] R. J. Wilson, *Introduction to Graph Theory*, 5th ed. Harlow, U.K.: Pearson Education, 2010.
- [55] J. R. Munkres, *Topology*, 2nd ed. Upper Saddle River, NJ, USA: Prentice-Hall, 2000.
- [56] R. T. Rockafellar, *Convex Analysis*. Princeton, NJ, USA: Princeton Univ. Press, 1970.

- [57] E. Artin, *Geometric Algebra*. New York, NY, USA: Dover, 2016.
- [58] W. Hoeffding, “Probability inequalities for sums of bounded random variables,” *J. Amer. Statistical Assoc.*, vol. 58, no. 301, pp. 13–30, 1963, doi: 10.1080/01621459.1963.10500830.
- [59] R. A. Horn and C. R. Johnson, *Topics in Matrix Analysis*. Cambridge, U.K.: Cambridge Univ. Press, 1991.
- [60] A. Holder and J. Eichholz, *An Introduction to Computational Science*. Cham, Switzerland: Springer Nature, 2019.
- [61] M. Burr, S. Gao, and F. Knoll, “Optimal bounds for Johnson-Lindenstrauss transformations,” *J. Mach. Learn. Res.*, vol. 19, no. 73, pp. 1–22, Oct. 2018. [Online]. Available: <http://jmlr.org/papers/v19/18-264.html>
- [62] W. B. Johnson and J. Lindenstrauss, “Extensions of Lipschitz mappings into a Hilbert space,” *Contemporary Math.*, vol. 26, pp. 189–206, 1984.
- [63] S. Dasgupta and A. Gupta, “An elementary proof of a theorem of Johnson and Lindenstrauss,” *Random Struct. Algorithms*, vol. 22, no. 1, pp. 60–65, Nov. 2003, doi: 10.1002/rsa.10073.
- [64] S. Bubeck, “Convex optimization: Algorithms and complexity,” *Found. Trends Mach. Learn.*, vol. 8, no. 3–4, pp. 231–357, Nov. 2015, doi: 10.1561/2200000050.
- [65] U. von Luxburg, “A tutorial on spectral clustering,” *Statist. Comput.*, vol. 17, no. 4, pp. 395–416, Dec. 2007, doi: 10.1007/s11222-007-9033-z.

- [66] A. Y. Ng, M. I. Jordan, and Y. Weiss, “On spectral clustering: Analysis and an algorithm,” in *Adv. Neural Inf. Process. Syst.*, T. Dietterich, S. Becker, and Z. Ghahramani, Eds., vol. 14, 2001, pp. 849–856. [Online]. Available: [https://papers.nips.cc/paper\\_files/paper/2001/hash/801272ee79cfde7fa5960571fee36b9b-Abstract.html](https://papers.nips.cc/paper_files/paper/2001/hash/801272ee79cfde7fa5960571fee36b9b-Abstract.html)
- [67] G. Casella and R. L. Berger, *Statistical Inference*, 2nd ed. Pacific Grove, CA: Duxbury, 2002.
- [68] N. Dunford and J. T. Schwartz, *Linear Operators, Part I: General Theory*. New York, NY, USA: Wiley, 1988.
- [69] K. Lange, *MM Optimization Algorithms*. Philadelphia, PA, USA: SIAM, 2016.
- [70] D. R. Hunter and K. Lange, “A tutorial on MM algorithms,” *Amer. Statistician*, vol. 58, no. 1, pp. 30–37, 2004, doi: 10.1198/0003130042836.
- [71] M. J. Wainwright and M. I. Jordan, “Graphical models, exponential families, and variational inference,” *Found. Trends Mach. Learn.*, vol. 1, no. 1–2, pp. 1–305, Nov. 2008, doi: 10.1561/2200000001.
- [72] J. R. Norris, *Markov Chains*. New York, NY, USA: Cambridge Univ. Press, 1997.
- [73] P. R. Halmos, *Measure Theory*. New York, NY, USA: Springer-Verlag, 1974.
- [74] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, “Distributed optimization and statistical learning via the alternating direction method

of multipliers,” *Found. Trends Mach. Learn.*, vol. 3, no. 1, pp. 1–122, Jul. 2011, doi: 10.1561/2200000016.

- [75] A. Lapidoth, *A Foundation in Digital Communication*, 2nd ed. Cambridge, U.K.: Cambridge Univ. Press, 2017.
- [76] T. Opsahl, F. Agneessens, and J. Skvoretz, “Node centrality in weighted networks: Generalizing degree and shortest paths,” *Social Netw.*, vol. 32, no. 3, pp. 245–251, Jul. 2010, doi: 10.1016/j.socnet.2010.03.006.
- [77] D. P. Bertsekas, *Nonlinear Programming*, 2nd ed. Belmont, MA, USA: Athena Scientific, 1999.
- [78] B. Schölkopf and A. J. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. Cambridge, MA, USA: MIT Press, 2002.
- [79] M. E. Tipping and C. M. Bishop, “Probabilistic principal component analysis,” *J. Roy. Statist. Soc.: Ser. B (Statist. Methodology)*, vol. 61, no. 3, pp. 611–622, 1999, doi: 10.1111/1467-9868.00196.
- [80] L. Condat, “A primal–dual splitting method for convex optimization involving lipschitzian, proximable and linear composite terms,” *J. Optim. Theory Appl.*, vol. 158, no. 2, pp. 460–479, Aug. 2013, doi: 10.1007/s10957-012-0245-9.
- [81] H. R. Feyzmahdavian and M. Johansson, “Asynchronous iterations in optimization: New sequence results and sharper algorithmic guarantees,” *J. Mach. Learn. Res.*, vol. 24, no. 158, pp. 1–75, Apr. 2023. [Online]. Available: <https://www.jmlr.org/papers/v24/22-0555.html>

- [82] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods*. Belmont, MA, USA: Athena Scientific, 2015.
- [83] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, 2nd ed. New York, NY, USA: Springer Science+Business Media, 2009.
- [84] A. Ben-Israel and T. N. E. Greville, *Generalized Inverses: Theory and Applications*, 2nd ed. New York, NY, USA: Springer-Verlag, 2003.
- [85] C. D. Meyer, *Matrix Analysis and Applied Linear Algebra*. Philadelphia, PA, USA: SIAM, 2000.
- [86] I. Csiszar, “Generalized cutoff rates and Renyi’s information measures,” *IEEE Trans. Inf. Theory*, vol. 41, no. 1, pp. 26–34, Jan. 1995, doi: 10.1109/18.370121.
- [87] N. Aronszajn, “Theory of reproducing kernels,” *Trans. Am. Math. Soc.*, vol. 68, no. 3, pp. 337–404, May 1950.
- [88] A. Jung, S. Schmutzhard, and F. Hlawatsch, “The RKHS approach to minimum variance estimation revisited: Variance bounds, sufficient statistics, and exponential families,” *IEEE Trans. Inf. Theory*, vol. 60, no. 7, pp. 4050–4065, Jul. 2014.
- [89] B. S. Everitt and A. Skrondal, *The Cambridge Dictionary of Statistics*, 4th ed. Cambridge, U.K.: Cambridge Univ. Press, 2010.
- [90] G. Upton and I. Cook, *A Dictionary of Statistics*, 3rd ed. Oxford, U.K.: Oxford Univ. Press, 2014.

- [91] D. P. Bertsekas, *Convex Optimization Algorithms*. Belmont, MA, USA: Athena Scientific, 2015.
- [92] V. Müller, *Spectral Theory of Linear Operators: and Spectral Systems in Banach Algebras*. Basel, Switzerland: Birkhäuser Verlag, 2003.
- [93] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. Cambridge, MA, USA: MIT Press, 2018.
- [94] D. P. Bertsekas, *Dynamic Programming and Optimal Control*, vol. 2, 3rd ed. Belmont, MA, USA: Athena Scientific, 2007.
- [95] P. J. Brockwell and R. A. Davis, *Time Series: Theory and Methods*, 2nd ed. New York, NY, USA: Springer-Verlag, 1991.
- [96] S. Sra, S. Nowozin, and S. J. Wright, Eds., *Optimization for Machine Learning*. Cambridge, MA, USA: MIT Press, 2012.
- [97] G. Ridgeway, D. Madigan, and T. S. Richardson, “Boosting methodology for regression problems,” in *Proc. 7th Int. Workshop Artif. Intell. Statist.*, D. Heckerman and J. Whittaker, Eds., vol. R2, Jan. 1999. [Online]. Available: <https://proceedings.mlr.press/r2/ridgeway99a.html>
- [98] R. E. Schapire, “A brief introduction to boosting,” in *Proc. 16th Int. Joint Conf. Artif. Intell.*, vol. 2, 1999, pp. 1401–1406. [Online]. Available: <https://www.ijcai.org/Proceedings/99-2/Papers/103.pdf>
- [99] H. Drucker, “Improving regressors using boosting techniques,” in *Proc. 14th Int. Conf. Mach. Learn.*, D. H. Fisher, Ed., Jul. 1997, pp. 107–115.

- [100] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 4th ed. Cambridge, MA, USA: MIT Press, 2022.
- [101] M. Sipser, *Introduction to the Theory of Computation*, 3rd ed. Andover, U.K.: Cengage Learning, 2013.
- [102] L. Richardson and M. Amundsen, *RESTful Web APIs*. Sebastopol, CA, USA: O'Reilly Media, 2013.
- [103] J. A. Hanley and B. J. McNeil, “The meaning and use of the area under a receiver operating characteristic (ROC) curve.” *Radiol.*, vol. 143, no. 1, pp. 29–36, Apr. 1982, doi: 10.1148/radiology.143.1.7063747.
- [104] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *2016 IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778, doi: 10.1109/CVPR.2016.90.
- [105] D. M. Blei, A. Y. Ng, and M. I. Jordan, “Latent dirichlet allocation,” *J. Mach. Learn. Res.*, vol. 3, pp. 993–1022, Jan. 2003.
- [106] A. Vaswani et al., “Attention is all you need,” in *Adv. Neural Inf. Process. Syst.*, I. Guyon, U. von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30, 2017, pp. 5998–6008. [Online]. Available: [https://papers.nips.cc/paper\\_files/paper/2017/hash/3f5ee243547dee91fdb053c1c4a845aa-Abstract.html](https://papers.nips.cc/paper_files/paper/2017/hash/3f5ee243547dee91fdb053c1c4a845aa-Abstract.html)
- [107] L. Breiman, “Bagging predictors,” *Mach. Learn.*, vol. 24, no. 2, pp. 123–140, Aug. 1996, doi: 10.1007/BF00058655.

- [108] M. P. Salinas et al., “A systematic review and meta-analysis of artificial intelligence versus clinicians for skin cancer diagnosis,” *npj Digit. Med.*, vol. 7, no. 1, May 2024, Art. no. 125, doi: 10.1038/s41746-024-01103-x.
- [109] G. F. Cooper, “The computational complexity of probabilistic inference using bayesian belief networks,” *Artif. Intell.*, vol. 42, no. 2–3, pp. 393–405, Mar. 1990, doi: 10.1016/0004-3702(90)90060-D.
- [110] J. H. Friedman, “Greedy function approximation: A gradient boosting machine,” *Ann. Statist.*, vol. 29, no. 5, pp. 1189–1232, Oct. 2001, doi: 10.1214/aos/1013203451.
- [111] B. Efron and R. J. Tibshirani, *An Introduction to the Bootstrap*. New York, NY, USA: Chapman & Hall, 1993.
- [112] G. W. Brier, “Verification of forecasts expressed in terms of probability,” *Monthly Weather Rev.*, vol. 78, no. 1, pp. 1–3, Jan. 1950, doi: 10.1175/1520-0493(1950)078<0001:VOFEIT>2.0.CO;2.
- [113] H.-S. Park and C.-H. Jun, “A simple and fast algorithm for K-medoids clustering,” *Expert Syst. Appl.*, vol. 36, no. 2, Part 2, pp. 3336–3341, Mar. 2009, doi: 10.1016/j.eswa.2008.01.039.
- [114] A. Banerjee, S. Merugu, I. S. Dhillon, and J. Ghosh, “Clustering with Bregman divergences,” *J. Mach. Learn. Res.*, vol. 6, no. 58, pp. 1705–1749, Oct. 2005. [Online]. Available: <http://jmlr.org/papers/v6/banerjee05b.html>
- [115] O. Chapelle, B. Schölkopf, and A. Zien, Eds., *Semi-Supervised Learning*. Cambridge, MA, USA: MIT Press, 2006.

- [116] B. Kim et al., “Interpretability beyond feature attribution: Quantitative testing with concept activation vectors (TCAV),” in *Proc. 35th Int. Conf. Mach. Learn.*, J. Dy and A. Krause, Eds., vol. 80, 2018, pp. 2668–2677. [Online]. Available: <https://proceedings.mlr.press/v80/kim18d.html>
- [117] D. Sun, K.-C. Toh, and Y. Yuan, “Convex clustering: Model, theoretical guarantee and efficient algorithm,” *J. Mach. Learn. Res.*, vol. 22, no. 9, pp. 1–32, Jan. 2021. [Online]. Available: <http://jmlr.org/papers/v22/18-694.html>
- [118] K. Pelckmans, J. De Brabanter, J. A. K. Suykens, and B. De Moor, “Convex clustering shrinkage,” presented at the PASCAL Workshop Statist. Optim. Clustering Workshop, 2005.
- [119] C. Chai et al., “Efficient coreset selection with cluster-based methods,” in *Proc. 29th ACM SIGKDD Conf. Knowl. Discovery Data Mining*, Aug. 2023, pp. 167–178, doi: 10.1145/3580305.3599326.
- [120] M. F. Balcan, S. Ehrlich, and Y. Liang, “Distributed k-means and k-median clustering on general topologies,” in *Adv. Neural Inf. Process. Syst.*, C. J. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, Eds., vol. 26, 2013, pp. 1995–2003. [Online]. Available: [https://papers.nips.cc/paper\\_files/paper/2013/hash/7f975a56c761db6506eca0b37ce6ec87-Abstract.html](https://papers.nips.cc/paper_files/paper/2013/hash/7f975a56c761db6506eca0b37ce6ec87-Abstract.html)
- [121] International Organization for Standardization and International Electrotechnical Commission, “Information technology — Vocabulary,”

ISO/IEC 2382:2015, 2015, Accessed: September 21, 2025. [Online]. Available: <https://www.iso.org/standard/63598.html>

- [122] European Parliament and Council of the European Union, “Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation) (Text with EEA relevance),” Official Journal of the European Union, L 119/1, May 4, 2016, Accessed: July, 2025. [Online]. Available: <https://eur-lex.europa.eu/eli/reg/2016/679/oj>
- [123] European Parliament and Council of the European Union, “Regulation (EU) 2018/1725 of the European Parliament and of the Council of 23 October 2018 on the protection of natural persons with regard to the processing of personal data by the Union institutions, bodies, offices and agencies and on the free movement of such data, and repealing Regulation (EC) No 45/2001 and Decision No 1247/2002/EC (Text with EEA relevance),” Official Journal of the European Union, L 295/39, Nov. 21, 2018, Accessed: December 1, 2025. [Online]. Available: <https://eur-lex.europa.eu/eli/reg/2018/1725/oj>
- [124] X. Liu, H. Li, G. Xu, Z. Chen, X. Huang, and R. Lu, “Privacy-enhanced federated learning against poisoning adversaries,” *IEEE Trans. Inf. Forensics Security*, vol. 16, pp. 4574–4588, 2021, doi: 10.1109/TIFS.2021.3108434.
- [125] J. Zhang, B. Chen, X. Cheng, H. T. T. Binh, and S. Yu, “PoisonGAN:

Generative poisoning attacks against federated learning in edge computing systems,” *IEEE Internet Things J.*, vol. 8, no. 5, pp. 3310–3322, Mar. 2021, doi: 10.1109/JIOT.2020.3023126.

- [126] A. Silberschatz, H. F. Korth, and S. Sudarshan, *Database System Concepts*, 7th ed. New York, NY, USA: McGraw-Hill Education, 2019.
- [127] S. Abiteboul, R. Hull, and V. Vianu, *Foundations of Databases*. Reading, MA, USA: Addison-Wesley, 1995.
- [128] S. Hoberman, *Data Modeling Made Simple: A Practical Guide for Business and IT Professionals*, 2nd ed. Basking Ridge, NJ, USA: Technics Publications, 2009.
- [129] R. Ramakrishnan and J. Gehrke, *Database Management Systems*, 3rd ed. New York, NY, USA: McGraw-Hill, 2002.
- [130] E. F. Codd, “A relational model of data for large shared data banks,” *Commun. ACM*, vol. 13, no. 6, pp. 377–387, Jun. 1970, doi: 10.1145/362384.362685.
- [131] T. Gebru et al., “Datasheets for datasets,” *Commun. ACM*, vol. 64, no. 12, pp. 86–92, Nov. 2021, doi: 10.1145/3458723.
- [132] B. S. Everitt, *The Cambridge Dictionary of Statistics*, 2nd ed. Cambridge, U.K.: Cambridge Univ. Press, 2002.
- [133] D. A. Patterson and J. L. Hennessy, *Computer Organization and Design: The Hardware/Software Interface*, 5th ed. San Francisco, CA, USA: Morgan Kaufmann, 2013.

- [134] C. M. Bishop and H. Bishop, *Deep Learning: Foundations and Concepts*. Cham, Switzerland: Springer Nature, 2024.
- [135] S. J. D. Prince, *Understanding Deep Learning*. Cambridge, MA, USA: MIT Press, 2023.
- [136] G. Tel, *Introduction to Distributed Algorithms*, 2nd ed. Cambridge, U.K.: Cambridge Univ. Press, 2000.
- [137] C. M. Bishop and H. Bishop, *Deep Learning: Foundations and Concepts*. Cham, Switzerland: Springer Nature, 2024.
- [138] N. Cesa-Bianchi and G. Lugosi, *Prediction, Learning, and Games*. New York, NY, USA: Cambridge Univ. Press, 2006.
- [139] E. Hazan, “Introduction to online convex optimization,” *Found. Trends Optim.*, vol. 2, no. 3–4, pp. 157–325, Aug. 2016, doi: 10.1561/2400000013.
- [140] J. Colin, T. Fel, R. Cadène, and T. Serre, “What I cannot predict, I do not understand: A human-centered evaluation framework for explainability methods,” in *Adv. Neural Inf. Process. Syst.*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., vol. 35, 2022, pp. 2832–2845. [Online]. Available: [https://proceedings.neurips.cc/paper\\_files/paper/2022/hash/13113e938f2957891c0c5e8df811dd01-Abstract-Conference.html](https://proceedings.neurips.cc/paper_files/paper/2022/hash/13113e938f2957891c0c5e8df811dd01-Abstract-Conference.html)
- [141] L. Zhang, G. Karakasidis, A. Odnoblyudova, L. Dogruel, Y. Tian, and A. Jung, “Explainable empirical risk minimization,” *Neural Comput. Appl.*, vol. 36, no. 8, pp. 3983–3996, Mar. 2024, doi: 10.1007/s00521-023-09269-3.

- [142] A. Jung and P. H. J. Nardelli, “An information-theoretic approach to personalized explainable machine learning,” *IEEE Signal Process. Lett.*, vol. 27, pp. 825–829, 2020, doi: 10.1109/LSP.2020.2993176.
- [143] J. Chen, L. Song, M. J. Wainwright, and M. I. Jordan, “Learning to explain: An information-theoretic perspective on model interpretation,” in *Proc. 35th Int. Conf. Mach. Learn.*, J. Dy and A. Krause, Eds., vol. 80, 2018, pp. 883–892. [Online]. Available: <https://proceedings.mlr.press/v80/chen18j.html>
- [144] C. Rudin, “Stop explaining black box machine learning models for high-stakes decisions and use interpretable models instead,” *Nature Mach. Intell.*, vol. 1, no. 5, pp. 206–215, May 2019, doi: 10.1038/s42256-019-0048-x.
- [145] C. Molnar, *Interpretable Machine Learning: A Guide for Making Black Box Models Explainable*, 3rd ed. Ebook, 2025, Accessed: August 1, 2025. [Online]. Available: <https://christophm.github.io/interpretable-ml-book/>
- [146] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, “Grad-CAM: Visual explanations from deep networks via gradient-based localization,” in *2017 IEEE Int. Conf. Comput. Vis.*, 2017, pp. 618–626, doi: 10.1109/ICCV.2017.74.
- [147] D. N. Gujarati and D. C. Porter, *Basic Econometrics*, 5th ed. New York, NY, USA: McGraw-Hill/Irwin, 2009.
- [148] Y. Dodge, Ed., *The Oxford Dictionary of Statistical Terms*. New York, NY, USA: Oxford Univ. Press, 2003.

- [149] M. T. Ribeiro, S. Singh, and C. Guestrin, “Why should i trust you?: Explaining the predictions of any classifier,” in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2016, pp. 1135–1144, doi: 10.1145/2939672.2939778.
- [150] S. Mallat, “Understanding deep convolutional networks,” *Philos. Trans. Roy. Soc. A*, vol. 374, no. 2065, Apr. 2016, Art. no. 20150203, doi: 10.1098/rsta.2015.0203.
- [151] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, “Communication-efficient learning of deep networks from decentralized data,” in *Proc. 20th Int. Conf. Artif. Intell. Statist.*, A. Singh and J. Zhu, Eds., vol. 54, 2017, pp. 1273–1282. [Online]. Available: <https://proceedings.mlr.press/v54/mcmahan17a.html>
- [152] A. Jung, *Federated Learning: From Theory to Practice*. Singapore, Singapore: Springer Nature, 2026.
- [153] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, “Federated optimization in heterogeneous networks,” in *Proc. Mach. Learn. Syst.*, I. Dhillon, D. Papailiopoulos, and V. Sze, Eds., vol. 2, 2020. [Online]. Available: [https://proceedings.mlsys.org/paper\\_files/paper/2020/hash/1f5fe83998a09396ebe6477d9475ba0c-Abstract.html](https://proceedings.mlsys.org/paper_files/paper/2020/hash/1f5fe83998a09396ebe6477d9475ba0c-Abstract.html)
- [154] Y. SarcheshmehPour, Y. Tian, L. Zhang, and A. Jung, “Flow-based clustering and spectral clustering: A comparison,” in *2021 55th Asilomar Conf. Signals, Syst., Comput.*, M. B. Matthews, Ed., 2021, pp. 1292–1296, doi: 10.1109/IEEECONF53345.2021.9723162.

- [155] S. Shalev-Shwartz and S. Ben-David, *Understanding Machine Learning: From Theory to Algorithms*. New York, NY, USA: Cambridge Univ. Press, 2014.
- [156] J. Su, D. V. Vargas, and K. Sakurai, “One pixel attack for fooling deep neural networks,” *IEEE Trans. Evol. Comput.*, vol. 23, no. 5, pp. 828–841, Oct. 2019, doi: 10.1109/TEVC.2019.2890858.
- [157] T. Hastie and R. Tibshirani, “Generalized Additive Models,” *Statistical Sci.*, vol. 1, no. 3, pp. 297–310, Aug. 1986, doi: 10.1214/ss/1177013604.
- [158] Y. SarcheshmehPour, Y. Tian, L. Zhang, and A. Jung, “Clustered federated learning via generalized total variation minimization,” *IEEE Trans. Signal Process.*, vol. 71, pp. 4240–4256, 2023, doi: 10.1109/TSP.2023.3322848.
- [159] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, “Geometric deep learning: Going beyond Euclidean data,” *IEEE Signal Process. Mag.*, vol. 34, no. 4, pp. 18–42, Jul. 2017, doi: 10.1109/MSP.2017.2693418.
- [160] T. N. Kipf and M. Welling, “Semi-supervised classification with graph convolutional networks,” in *5th Int. Conf. Learn. Representations*, Feb. 2017. [Online]. Available: <https://openreview.net/forum?id=SJU4ayYgl>
- [161] P. Bühlmann and S. van de Geer, *Statistics for High-Dimensional Data: Methods, Theory and Applications*. Berlin, Germany: Springer-Verlag, 2011.

- [162] C. H. Lampert, “Kernel methods in computer vision,” *Found. Trends Comput. Graph. Vis.*, vol. 4, no. 3, pp. 193–285, Sep. 2009, doi: 10.1561/0600000027.
- [163] Q. Yang, Y. Liu, Y. Cheng, Y. Kang, T. Chen, and H. Yu, “Horizontal federated learning,” in *Federated Learning*. Cham, Switzerland: Springer Nature, 2020, ch. 4, pp. 49–67.
- [164] P. J. Huber, “Robust estimation of a location parameter,” in *Breakthroughs in Statistics: Methodology and Distribution*. New York, NY, USA: Springer-Verlag, 1992, ch. 35, pp. 492–518.
- [165] A. Jung and Y. SarcheshmehPour, “Local graph clustering with network Lasso,” *IEEE Signal Process. Lett.*, vol. 28, pp. 106–110, 2021, doi: 10.1109/LSP.2020.3045832.
- [166] C. R. Wren, A. Azarbayejani, T. Darrell, and A. P. Pentland, “Pfinder: Real-time tracking of the human body,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 19, no. 7, pp. 780–785, Jul. 1997, doi: 10.1109/34.598236.
- [167] F. Doshi-Velez and B. Kim, “Towards a rigorous science of interpretable machine learning,” arXiv preprint arXiv:1702.08608, Mar. 2017. [Online]. Available: <https://arxiv.org/abs/1702.08608>
- [168] P. Hase and M. Bansal, “Evaluating explainable AI: Which algorithmic explanations help users predict model behavior?” in *Proc. 58th Annu. Meeting Assoc. Comput. Linguistics*, D. Jurafsky, J. Chai, N. Schluter, and J. Tetreault, Eds., Jul. 2020, pp. 5540–5552. [Online]. Available: <https://aclanthology.org/2020.acl-main.491>

- [169] Z. C. Lipton, “The mythos of model interpretability: In machine learning, the concept of interpretability is both important and slippery,” *Queue*, vol. 16, no. 3, pp. 31–57, Jun. 2018, doi: 10.1145/3236386.3241340.
- [170] J. Kleinberg and E. Tardos, *Algorithm Design*. Boston, MA, USA: Addison-Wesley, 2006.
- [171] M. Mahajan, P. Nimbhorkar, and K. Varadarajan, “The planar k-means problem is NP-hard,” in *WALCOM: Algorithms and Computation*, S. Das and R. Uehara, Eds. Berlin, Heidelberg, Germany: Springer-Verlag, 2009, pp. 274–285.
- [172] R. Caruana, “Multitask learning,” *Mach. Learn.*, vol. 28, pp. 41–75, Jul. 1997, doi: 10.1023/A:1007379606734.
- [173] A. Jung, G. Hannak, and N. Goertz, “Graphical lasso based model selection for time series,” *IEEE Signal Process. Lett.*, vol. 22, no. 10, pp. 1781–1785, Oct. 2015, doi: 10.1109/LSP.2015.2425434.
- [174] A. Jung, “Learning the conditional independence structure of stationary time series: A multitask learning approach,” *IEEE Trans. Signal Process.*, vol. 63, no. 21, Nov. 2015, doi: 10.1109/TSP.2015.2460219.
- [175] R. Tibshirani, “Regression shrinkage and selection via the lasso,” *J. Roy. Statist. Soc.: Ser. B (Methodological)*, vol. 58, no. 1, pp. 267–288, 1996, doi: 10.1111/j.2517-6161.1996.tb02080.x.
- [176] R. A. Fisher, “The use of multiple measurements in taxonomic problems,” *Ann. Eugenics*, vol. 7, no. 2, pp. 179–188, Sep. 1936, doi: 10.1111/j.1469-1809.1936.tb02137.x.

- [177] J. Heinonen, “Lectures on Lipschitz analysis,” Dept. Math. Statist., Univ. Jyväskylä, Jyväskylä, Finland, Rep. 100, 2005. [Online]. Available: <http://www.math.jyu.fi/research/reports/rep100.pdf>
- [178] S. Lloyd, “Least squares quantization in PCM,” *IEEE Trans. Inf. Theory*, vol. 28, no. 2, pp. 129–137, Mar. 1982.
- [179] C. G. Atkeson, A. W. Moore, and S. Schaal, “Locally weighted learning,” *Artif. Intell. Rev.*, vol. 11, no. 1–5, pp. 11–73, Feb. 1997, doi: 10.1023/A:1006559212014.
- [180] T. Hastie, R. Tibshirani, and M. Wainwright, *Statistical Learning with Sparsity: The Lasso and Generalizations*. Boca Raton, FL, USA: CRC Press, 2015.
- [181] L. Bottou, “On-line learning and stochastic approximations,” in *On-Line Learning in Neural Networks*, D. Saad, Ed. New York, NY, USA: Cambridge Univ. Press, 1999, ch. 2, pp. 9–42.
- [182] Y. Cao and J. Yang, “Towards making systems forget with machine unlearning,” in *2015 IEEE Symp. Secur. Privacy*, 2015, pp. 463–480, doi: 10.1109/SP.2015.35.
- [183] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, “Membership inference attacks against machine learning models,” in *2017 IEEE Symp. Secur. Privacy*, 2017, pp. 3–18, doi: 10.1109/SP.2017.41.
- [184] K. Abayomi, A. Gelman, and M. Levy, “Diagnostics for multivariate imputations,” *J. Roy. Statist. Soc.: Ser. C (Appl. Statist.)*, vol. 57, no. 3, pp. 273–291, Jun. 2008, doi: 10.1111/j.1467-9876.2007.00613.x.

- [185] E. A. Bender, *An Introduction to Mathematical Modeling*. New York, NY, USA: Wiley, 1978.
- [186] M. Fredrikson, S. Jha, and T. Ristenpart, “Model inversion attacks that exploit confidence information and basic countermeasures,” in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2015, pp. 1322–1333, doi: 10.1145/2810103.2813677.
- [187] D. Hallac, J. Leskovec, and S. Boyd, “Network lasso: Clustering and optimization in large graphs,” in *Proc. 21st ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2015, pp. 387–396, doi: 10.1145/2783258.2783313.
- [188] A. Jung, “Networked exponential families for big data over networks,” *IEEE Access*, vol. 8, pp. 202 897–202 909, Nov. 2020, doi: 10.1109/ACCESS.2020.3033817.
- [189] A. Rakhlin, O. Shamir, and K. Sridharan, “Making gradient descent optimal for strongly convex stochastic optimization,” in *Proc. 29th Int. Conf. Mach. Learn.*, J. Langford and J. Pineau, Eds., 2012, pp. 449–456. [Online]. Available: <https://icml.cc-Conferences/2012/papers/261.pdf>
- [190] S. Bubeck and N. Cesa-Bianchi, “Regret analysis of stochastic and non-stochastic multi-armed bandit problems,” *Found. Trends Mach. Learn.*, vol. 5, no. 1, pp. 1–122, Dec. 2012, doi: 10.1561/2200000024.
- [191] M. Kearns and M. Li, “Learning in the presence of malicious errors,” *SIAM J. Comput.*, vol. 22, no. 4, pp. 807–837, Aug. 1993, doi: 10.1137/0222052.

- [192] G. Lugosi and S. Mendelson, “Robust multivariate mean estimation: The optimality of trimmed mean,” *Ann. Statist.*, vol. 49, no. 1, pp. 393–410, Feb. 2021, doi: 10.1214/20-AOS1961.
- [193] F. Rosenblatt, “The perceptron: A probabilistic model for information storage and organization in the brain,” *Psychol. Rev.*, vol. 65, no. 6, pp. 386–408, Nov. 1958, doi: 10.1037/h0042519.
- [194] B. Juba and H. S. Le, “Precision-recall versus accuracy and the role of large data sets,” in *Proc. AAAI Conf. Artif. Intell.*, vol. 33, no. 01, Jul. 2019, pp. 4039–4048, doi: 10.1609/aaai.v33i01.33014039.
- [195] C. J. van Rijsbergen, *Information Retrieval*, 2nd ed. London, U.K.: Butterworth-Heinemann, 1979.
- [196] A. Makhdoumi, S. Salamatian, N. Fawaz, and M. Médard, “From the information bottleneck to the privacy funnel,” in *2014 IEEE Inf. Theory Workshop*, 2014, pp. 501–505, doi: 10.1109/ITW.2014.6970882.
- [197] A. Ünsal and M. Önen, “Information-theoretic approaches to differential privacy,” *ACM Comput. Surv.*, vol. 56, no. 3, Oct. 2023, Art. no. 76, doi: 10.1145/3604904.
- [198] P. L. Bartlett and S. Mendelson, “Rademacher and gaussian complexities: Risk bounds and structural results,” *J. Mach. Learn. Res.*, vol. 3, no. Nov., pp. 463–482, 2002. [Online]. Available: <https://www.jmlr.org/papers/v3/bartlett02a.html>
- [199] T. Fawcett, “An introduction to ROC analysis,” *Pattern Recognit. Lett.*, vol. 27, no. 8, pp. 861–874, Jun. 2006, doi: 10.1016/j.patrec.2005.10.010.

- [200] S. Shalev-Shwartz and A. Tewari, “Stochastic methods for  $\ell_1$  regularized loss minimization,” in *Proc. 26th Annu. Int. Conf. Mach. Learn.*, L. Bottou and M. Littman, Eds., Jun. 2009, pp. 929–936.
- [201] G. S. Collins and K. G. M. Moons, “Comparing risk prediction models.” *BMJ*, vol. 344, May 2012, Art. no. e3186, doi: 10.1136/bmj.e3186.
- [202] E. W. Steyerberg, *Clinical Prediction Models: A Practical Approach to Development, Validation, and Updating*, 2nd ed. Cham, Switzerland: Springer Nature, 2019.
- [203] L. Cohen, *Time-Frequency Analysis*. Upper Saddle River, NJ, USA: Prentice-Hall, 1995.
- [204] J. Li, L. Han, X. Li, J. Zhu, B. Yuan, and Z. Gou, “An evaluation of deep neural network models for music classification using spectrograms,” *Multimedia Tools Appl.*, vol. 81, no. 4, pp. 4621–4647, Feb. 2022, doi: 10.1007/s11042-020-10465-9.
- [205] B. Boashash, Ed., *Time Frequency Signal Analysis and Processing: A Comprehensive Reference*. Oxford, U.K.: Elsevier, 2003.
- [206] S. Mallat, *A Wavelet Tour of Signal Processing: The Sparse Way*, 3rd ed. Burlington, MA, USA: Academic, 2009.
- [207] D. H. Wolpert, “Stacked generalization,” *Neural Netw.*, vol. 5, no. 2, pp. 241–259, 1992, doi: 10.1016/S0893-6080(05)80023-1.
- [208] Z.-H. Zhou, *Ensemble Methods: Foundations and Algorithms*. New York, NY, USA: Chapman & Hall/CRC Press, 2012.

- [209] R. Motwani and P. Raghavan, *Randomized Algorithms*. Cambridge, U.K.: Cambridge Univ. Press, 1995.
- [210] R. G. Gallager, *Stochastic Processes: Theory for Applications*. New York, NY, USA: Cambridge Univ. Press, 2013.
- [211] P. W. Holland, K. B. Laskey, and S. Leinhardt, “Stochastic blockmodels: First steps,” *Social networks*, vol. 5, no. 2, pp. 109–137, 1983.
- [212] E. Abbe, “Community detection and stochastic block models: Recent developments,” *J. Mach. Learn. Res.*, vol. 18, no. 177, pp. 1–86, Apr. 2018. [Online]. Available: <http://jmlr.org/papers/v18/16-480.html>
- [213] Organisation for Economic Co-operation and Development (OECD), *OECD Glossary of Statistical Terms*. Paris, France: OECD Publishing, 2008. [Online]. Available: <https://doi.org/10.1787/9789264055087-en>
- [214] N. Cristianini and J. Shawe-Taylor, *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. New York, NY, USA: Cambridge Univ. Press, 2000.
- [215] V. N. Vapnik and A. Y. Chervonenkis, “On the uniform convergence of relative frequencies of events to their probabilities,” in *Measures of Complexity: Festschrift for Alexey Chervonenkis*. Cham, Switzerland: Springer, 2015, ch. 3, pp. 11–30.
- [216] Q. Yang, Y. Liu, Y. Cheng, Y. Kang, T. Chen, and H. Yu, “Vertical federated learning,” in *Federated Learning*. Cham, Switzerland: Springer Nature, 2020, ch. 5, pp. 69–81.

- [217] S. J. Russel and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. Upper Saddle River, NJ, USA: Prentice-Hall, 2010.
- [218] R. Bellman, *Dynamic Programming*. Princeton, NJ, USA: Princeton Univ. Press, 1957.
- [219] D. P. Bertsekas, *Dynamic Programming and Optimal Control*, vol. 1, 4th ed. Belmont, MA, USA: Athena Scientific, 2017.
- [220] N. A. Lynch, *Distributed Algorithms*. San Francisco, CA, USA: Morgan Kaufmann, 1996.
- [221] E. N. M. Elnozahy, L. Alvisi, Y.-M. Wang, and D. B. Johnson, “A survey of rollback-recovery protocols in message-passing systems,” *ACM Comput. Surv.*, vol. 34, no. 3, pp. 375–408, Sep. 2002, doi: 10.1145/568522.568525.
- [222] M. Abadi et al., “TensorFlow: A system for large-scale machine learning,” in *Proc. 12th USENIX Symp. Operating Syst. Des. Implementation*, 2016, pp. 265–283.
- [223] M. Mazzucco and M. Dumas, “Achieving performance and availability guarantees with spot instances,” in *Proc. 2011 IEEE Int. Conf. High Perform. Comput. Commun.*, Sep. 2011, pp. 296–303, doi: 10.1109/HPCC.2011.46.
- [224] M. van Steen and A. S. Tanenbaum, *Distributed Systems*, 3rd ed. Ebook, 2017. [Online]. Available: <https://www.distributed-systems.net/index.php/books/ds3/>

- [225] D. E. Comer, *The Cloud Computing Book: The Future of Computing Explained*. Boca Raton, FL, USA: CRC Press, 2021.
- [226] M. Armbrust et al., “A view of cloud computing,” *Commun. ACM*, vol. 53, no. 4, pp. 50–58, Apr. 2010, doi: 10.1145/1721654.1721672.
- [227] J. M. Wooldridge, *Introductory Econometrics: A Modern Approach*, 8th ed., international ed. Mason, OH, USA: Cengage Learning, 2025.
- [228] S. H. Strogatz, *Nonlinear Dynamics and Chaos: With Applications to Physics, Biology, Chemistry, and Engineering*, 2nd ed. Boca Raton, FL, USA: CRC Press, 2015.
- [229] S. Teerapittayanon, B. McDaniel, and H. T. Kung, “BranchyNet: Fast inference via early exiting from deep neural networks,” in *2016 23rd Int. Conf. Pattern Recognit.*, 2016, pp. 2464–2469, doi: 10.1109/ICPR.2016.7900006.
- [230] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, “Edge computing: Vision and challenges,” *IEEE Internet Things J.*, vol. 3, no. 5, pp. 637–646, Oct. 2016, doi: 10.1109/JIOT.2016.2579198.
- [231] T. N. Gia, L. Qingqing, J. P. Queralta, Z. Zou, H. Tenhunen, and T. Westerlund, “Edge AI in smart farming IoT: CNNs at the edge and fog computing with LoRa,” in *2019 IEEE AFRICON*, 2019, pp. 1–6, doi: 10.1109/AFRICON46755.2019.9134049.
- [232] O. Hashash, S. Sharafeddine, Z. Dawy, A. Mohamed, and E. Yaacoub, “Energy-aware distributed edge ML for mHealth applications with strict

- latency requirements,” *IEEE Wireless Commun. Lett.*, vol. 10, no. 12, pp. 2791–2794, Dec. 2021, doi: 10.1109/LWC.2021.3117876.
- [233] D. E. Knuth, *The Art of Computer Programming*, vol. 1: *Fundamental Algorithms*, 3rd ed. Reading, MA, USA: Addison-Wesley, 1997.
- [234] G. M. Fitzmaurice, N. M. Laird, and J. H. Ware, *Applied Longitudinal Analysis*, 2nd ed. Hoboken, NJ, USA: Wiley, 2011.
- [235] C. H. Mallinckrodt and I. Lipkovich, *Analyzing Longitudinal Clinical Trial Data: A Practical Guide*. Boca Raton, FL, USA: Chapman & Hall/CRC Press, 2017.
- [236] H. Hapke and C. Nelson, *Building Machine Learning Pipelines: Automating Model Life Cycles with TensorFlow*. Sebastopol, CA, USA: O'Reilly Media, 2020.
- [237] A. S. Tanenbaum, *Modern Operating Systems*, 3rd ed. Upper Saddle River, NJ, USA: Prentice-Hall, 2008.
- [238] N. D. Lane and P. Georgiev, “Can deep learning revolutionize mobile sensing?” in *Proc. 16th Int. Workshop Mobile Comput. Syst. Appl.*, Feb. 2015, pp. 117–122, doi: 10.1145/2699343.2699349.
- [239] V. Tsoukas, A. Gkogkidis, E. Boumpa, and A. Kakarountas, “A review on the emerging technology of TinyML,” *ACM Comput. Surv.*, vol. 56, no. 10, Jun. 2024, Art. no. 259, doi: 10.1145/3661820.
- [240] S. Khatua and N. Mukherjee, “A novel checkpointing scheme for Amazon

EC2 spot instances,” in *Proc. 13th IEEE/ACM Int. Symp. Cluster, Cloud, Grid Comput.*, May 2013, pp. 180–181, doi: 10.1109/CCGrid.2013.71.

- [241] European Commission, “Proposal for a regulation of the European Parliament and of the Council laying down harmonised rules on artificial intelligence (Artificial Intelligence Act) and amending certain Union legislative acts,” COM/2021/206 final, Apr. 21, 2021, Accessed: December 16, 2025. [Online]. Available: <https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=celex:52021PC0206>
- [242] European Parliament and Council of the European Union, “Regulation (EU) 2024/1689 of the European Parliament and of the Council of 13 June 2024 laying down harmonised rules on artificial intelligence and amending Regulations (EC) No 300/2008, (EU) No 167/2013, (EU) No 168/2013, (EU) 2018/858, (EU) 2018/1139 and (EU) 2019/2144 and Directives 2014/90/EU, (EU) 2016/797 and (EU) 2020/1828 (Artificial Intelligence Act) (Text with EEA relevance),” Official Journal of the European Union, L series, Jul. 12, 2024, Accessed: July 2025. [Online]. Available: <https://eur-lex.europa.eu/eli/reg/2024/1689/oj/eng>
- [243] M. D. Wilkinson et al., “The FAIR Guiding Principles for scientific data management and stewardship,” *Scientific Data*, vol. 3, Mar. 2016, Art. no. 160018, doi: 10.1038/sdata.2016.18.
- [244] S. Samuel, F. Löffler, and B. König-Ries, “Machine learning pipelines: Provenance, reproducibility and FAIR data principles,” in *Provenance and Annotation of Data and Processes: 8th and 9th International Provenance and Annotation Workshop, IPAW 2020 + IPAW 2021, Virtual Event*,

*July 19–22, 2021, Proceedings*, B. Glavic, V. Braganholo, and D. Koop, Eds. Cham, Switzerland: Springer Nature, 2021, pp. 226–230.

- [245] High-Level Expert Group on Artificial Intelligence, “Ethics guidelines for trustworthy AI,” European Commission, Apr. 8, 2019. [Online]. Available: <https://digital-strategy.ec.europa.eu/en/library/ethics-guidelines-trustworthy-ai>
- [246] C. Gallesse, “The AI Act proposal: A new right to technical interpretability?” *SSRN Electron. J.*, Feb. 2023. [Online]. Available: <https://ssrn.com/abstract=4398206>
- [247] M. Mitchell et al., “Model cards for model reporting,” in *Proc. Conf. Fairness, Accountability, Transparency*, 2019, pp. 220–229, doi: 10.1145/3287560.3287596.
- [248] K. Shahriari and M. Shahriari, “IEEE standard review — Ethically aligned design: A vision for prioritizing human wellbeing with artificial intelligence and autonomous systems,” in *2017 IEEE Canada Int. Humanitarian Technol. Conf.*, 2017, pp. 197–201, doi: 10.1109/IHTC.2017.8058187.
- [249] D. Pfau and A. Jung, “Engineering trustworthy AI: A developer guide for empirical risk minimization,” arXiv preprint arXiv:2410.19361, Nov. 2024. [Online]. Available: <https://arxiv.org/abs/2410.19361>
- [250] High-Level Expert Group on Artificial Intelligence, “The assessment list for trustworthy artificial intelligence (ALTAI): For self assessment,” European Commission, Jul. 17, 2020.

[Online]. Available: <https://digital-strategy.ec.europa.eu/en/library/assessment-list-trustworthy-artificial-intelligence-altai-self-assessment>