Introductory Lectures on Federated Learning

Dipl.-Ing. Dr.techn. Alexander Jung*

December 15, 2023

Abstract

We discuss theory and algorithms for federated learning (FL) from decentralized data with an intrinsic network structure, i.e., networked data. Such networked data consists of local datasets that are related by different notions of similarities. FL methods collaboratively train local (" personalized") models for each local dataset. These tailored local models are coupled via a network structure that reflects statistical similarities between local datasets. We use the network structure of data to guide this coupling via regularization techniques. In particular, we use different measures for the variation of local models as a regularization term. The resulting generalized total variation minimization provides a unifying design principle for practical FL algorithms. These algorithms solve generalized total variation via distributed optimization methods that are able to cope with limited computational resources and imperfections.

*AJ is currently Associate Professor for Machine Learning at Aalto University (Finland).
This work has been partially funded by the Academy of Finland (decision numbers 331197, 331197) and the European Union (grant number 952410).

Contents

Lists of Symbols

Sets and Functions

$a \in \mathcal{A}$	This statement indicates that the object a is an element of the set $\mathcal{A}$.
$a := b$	This statement defines a to be shorthand for b.
$\lvert\mathcal{A}\rvert$	The cardinality (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 $\mathcal{B}$.
$\mathbb{N}$	The set of natural numbers $1, 2, \ldots$.
$\mathbb{R}$	The set of real numbers x [2].
$\mathbb{R}_+$	The set of non-negative real numbers $x \geq 0$.
$\mathbb{R}_{++}$	The set of positive real numbers $x > 0$.
$h(\cdot): \mathcal{A} \to \mathcal{B} : a \mapsto h(a)$	A function (map) that accepts any element $a \in \mathcal{A}$ from a set $\mathcal{A}$ as input and delivers a well-defined element $h(a) \in \mathcal{B}$ of a set $\mathcal{B}$. The set $\mathcal{A}$ is the domain of the function h and the set $\mathcal{B}$ is the codomain of h. ML aims at finding (or learning) a function h ("hypothesis") that reads in the features $\mathbf{x}$ of a data point and delivers a prediction $h(\mathbf{x})$ for its label y.

$\{0, 1\}$	The binary set that consists of the two real numbers 0 and 1.
$[0, 1]$	The closed interval of real numbers x with $0 \leq x \leq 1$.
$\operatorname{argmin} f(\mathbf{w})$	The set of minimizers for a real-valued function $f(\mathbf{w})$.
$\log a$	The logarithm of the positive number $a \in \mathbb{R}_{++}$.

Matrices and Vectors

$\mathbf{I}_{l\times d}$	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 equal to 0 otherwise.
$\mathbf{I}$	A square identity matrix whose shape should be clear from the context.
$\mathbb{R}^d$	The set of vectors $\mathbf{x} = \big(x_1, \ldots, x_d\big)^T$ consisting of d real-valued entries $x_1, \ldots, x_d \in \mathbb{R}$.
$\mathbf{x} = \big(x_1, \ldots, x_d\big)^T$	A vector of length d. The jth entry of the vector is denoted x_j.
$\|\mathbf{x}\|_2$	The Euclidean (or "ℓ_2") norm of the vector $\mathbf{x} = \big(x_1, \ldots, x_d\big)^T \in \mathbb{R}^d$ given as $\|\mathbf{x}\|_2 := \sqrt{\sum_{j=1}^d x_j^2}$.
$\|\mathbf{x}\|$	Some norm of the vector $\mathbf{x} \in \mathbb{R}^d$ [3]. Unless specified otherwise, we mean the Euclidean norm $\|\mathbf{x}\|_2$.
$\mathbf{x}^T$	The transpose of a vector $\mathbf{x}$ that is considered a single column matrix. The transpose is a single-row matrix $\big(x_1, \ldots, x_d\big)$.
$\mathbf{X}^T$	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 $\mathbf{X} = \mathbf{X}^T$.
$\mathbf{0} = \big(0, \ldots, 0\big)^T$	A vector of zero entries.

$\left(\mathbf{v}^T, \mathbf{w}^T\right)^T$	The vector of length $d + d'$ obtained by concatenating the entries of vector $\mathbf{v} \in \mathbb{R}^d$ with the entries of $\mathbf{w} \in \mathbb{R}^{d'}$.
$\text{span}\{\mathbf{B}\}$	The span of a matrix $\mathbf{B} \in \mathbb{R}^{a \times b}$, which is the subspace of all linear combinations of columns of $\mathbf{B}$, $\text{span}\{\mathbf{B}\} = \left\{\mathbf{Ba} : \mathbf{a} \in \mathbb{R}^b\right\} \subseteq \mathbb{R}^a$.
$\mathbb{S}_+^d$	The set of all positive semi-definite (psd) matrices of size $d \times d$.
$\det(\mathbf{C})$	The determinant of the matrix $\mathbf{C}$.

Probability Theory

$\mathbb{E}_p\{f(\mathbf{z})\}$	The expectation of a function $f(\mathbf{z})$ of a RV $\mathbf{z}$ whose probability distribution is $p(\mathbf{z})$. If the probability distribution is clear from context we just write $\mathbb{E}\{f(\mathbf{z})\}$.
$p(\mathbf{x}, y)$	A (joint) probability distribution of a RV whose realizations are data points with features $\mathbf{x}$ and label y.
$p(\mathbf{x}\|y)$	A conditional probability distribution of a RV $\mathbf{x}$ given the value of another RV y [4, Sec. 3.5].
$p(\mathbf{x}; \mathbf{w})$	A parametrized probability distribution of a RV $\mathbf{x}$. The probability distribution depends on a parameter vector $\mathbf{w}$. For example, $p(\mathbf{x}; \mathbf{w})$ could be a multivariate normal distribution of a Gaussian RV $\mathbf{x}$ with the parameter vector $\mathbf{w}$ given by the entries of the mean vector $\mathbb{E}\{\mathbf{x}\}$ and the covariance matrix $\mathbb{E}\Big\{\big(\mathbf{x} - \mathbb{E}\{\mathbf{x}\}\big)\big(\mathbf{x} - \mathbb{E}\{\mathbf{x}\}\big)^T\Big\}$.
$\mathcal{N}(\mu, \sigma^2)$	The probability distribution of a scalar normal ("Gaussian") RV $x \in \mathbb{R}$ with mean (or expectation) $\mu = \mathbb{E}\{x\}$ and variance $\sigma^2 = \mathbb{E}\big\{(x-\mu)^2\big\}$.
$\mathcal{N}(\boldsymbol{\mu}, \mathbf{C})$	The multivariate normal distribution of a vector-valued Gaussian RV $\mathbf{x} \in \mathbb{R}^d$ with mean (or expectation) $\boldsymbol{\mu} = \mathbb{E}\{\mathbf{x}\}$ and covariance matrix $\mathbf{C} = \mathbb{E}\big\{\big(\mathbf{x} - \boldsymbol{\mu}\big)\big(\mathbf{x} - \boldsymbol{\mu}\big)^T\big\}$.

Machine Learning

r	An index $r = 1, 2, \ldots,$ that enumerates data points.
m	The number of data points in (the size of) a dataset.
$\mathcal{D}$	A dataset $\mathcal{D} = \{\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(m)}\}$ is a list of individual data points $\mathbf{z}^{(r)}$, for $r = 1, \ldots, m$.
d	Number of features that characterize a data point.
x_j	The jth feature of a data point. The first feature of a given data point is denoted x_1, the second feature x_2 and so on.
$\mathbf{x}$	The feature vector $\mathbf{x} = \big(x_1, \ldots, x_d\big)^T$ of a data point whose entries are the individual features of a data point.
$\mathcal{X}$	The feature space $\mathcal{X}$ is the set of all possible values that the features $\mathbf{x}$ of a data point can take on.
$\mathbf{z}$	Beside the symbol $\mathbf{x}$, we sometimes use $\mathbf{z}$ as another symbol to denote a vector whose entries are features of a data point. We need two different symbols to distinguish between "raw" or "original" and learnt features [1, Ch. 9].
$\mathbf{x}^{(r)}$	The feature vector of the rth data point within a dataset.
$x_j^{(r)}$	The jth feature of the rth data point within a dataset.
$\mathcal{B}$	A mini-batch (subset) of randomly chosen data points.
B	The size of (the number of data points in) a mini-batch.

y	The label (quantity of interest) of a data point.
$y^{(r)}$	The label of the rth data point.
$\big(\mathbf{x}^{(r)}, y^{(r)}\big)$	The features and label of the rth data point.
$\mathcal{Y}$	The label space $\mathcal{Y}$ of a ML method consists of all potential label values that a data point can have. We often use label spaces that are larger than the set of different label values arising in a give dataset (e.g., a training set). We refer to a ML problems (methods) using a numeric label space, such as $\mathcal{Y} = \mathbb{R}$ or $\mathcal{Y} = \mathbb{R}^3$, as regression problems (methods). ML problems (methods) that use a discrete label space, such as $\mathcal{Y} = \{0, 1\}$ or $\mathcal{Y} = \{$"cat","dog","mouse"$\}$ are referred to as classification problems (methods).
α	learning rate (step-size) used by gradient-based methods.
$h(\cdot)$	A hypothesis map that reads in features $\mathbf{x}$ of a data point and delivers a prediction $\hat{y} = h(\mathbf{x})$ for its label y.
$\mathcal{Y}^{\mathcal{X}}$	Given two sets $\mathcal{X}$ and $\mathcal{Y}$, we denote by $\mathcal{Y}^{\mathcal{X}}$ the set of all possible hypothesis maps $h : \mathcal{X} \to \mathcal{Y}$.
$\mathcal{H}$	A hypothesis space or model used by a ML method. The hypothesis space consists of different hypothesis maps $h : \mathcal{X} \to \mathcal{Y}$ between which the ML method has to choose .

B^2	The squared bias of a learnt hypothesis $\hat{h}$ delivered by a ML algorithm that is fed with data points which are modelled as realizations of RVs. If data is modelled as realizations of RVs, also the delivered hypothesis $\hat{h}$ is the realization of a RV.
V	The variance of the (parameters of the) hypothesis delivered by a ML algorithm. If the input data for this algorithm is interpreted as realizations of RVs, so is the delivered hypothesis a realization of a RV.
$L\left((\mathbf{x}, y), h\right)$	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 from evaluating the hypothesis $h \in \mathcal{H}$ for the feature vector $\mathbf{x}$ of the data point.
E_v	The validation error of a hypothesis h, which is its average loss incurred over a validation set.
$\widehat{L}\big(h\|\mathcal{D}\big)$	The empirical risk or average loss incurred by the predictions of hypothesis h for the data points in the dataset $\mathcal{D}$.
E_t	The training error of a hypothesis h, which is its average loss incurred over a training set.
t	A discrete-time index $t = 0, 1, \ldots$ used to enumerate a sequence to sequential events ("time instants").
t	An index that enumerates learning tasks within a multi-task learning problem.

λ	A regularization parameter that controls the amount of regularization.
$\lambda_j\big(\mathbf{Q}\big)$	The jth eigenvalue (sorted either ascending or descending) of a psd matrix $\mathbf{Q}$. We also use the shorthand λ_j if the corresponding matrix is clear from context.
$\sigma(\cdot)$	The activation function used by an artificial neuron within an artificial neural network (ANN).
$\mathcal{R}_{\hat{y}}$	A decision region within a feature space.
$\mathbf{w}$	A parameter vector $\mathbf{w} = \big(w_1, \ldots, w_d\big)^T$ whose entries are parameters of a model. These parameters could be feature weights in linear maps, the weights in ANNs or the thresholds used for splits in decision trees.
$h^{(\mathbf{w})}(\cdot)$	A hypothesis map that involves tunable model parameters $w_1, \ldots, w_d$ which are stacked into the vector $\mathbf{w} = \big(w_1, \ldots, w_d\big)^T$.
$\nabla f(\mathbf{w})$	The gradient of a differentiable real-valued function $f : \mathbb{R}^d \to \mathbb{R}$ is the vector $\nabla f(\mathbf{w}) = \big(\frac{\partial f}{\partial w_1}, \ldots, \frac{\partial f}{\partial w_d}\big)^T \in \mathbb{R}^d$ [5, Ch. 9].
$\phi(\cdot)$	A feature map $\phi : \mathcal{X} \to \mathcal{X}' : \mathbf{x} \mapsto \mathbf{x}' := \phi\big(\mathbf{x}\big) \in \mathcal{X}'$.

Federated Learning

$\mathcal{G} = (\mathcal{V}, \mathcal{E})$	Empirical graph whose nodes $i \in \mathcal{V}$ carry local datasets and local models.
$i \in \mathcal{V}$	A node in the empirical graph that represents a local dataset and a corresponding local model. It might also be useful to think of node i as a small computer that can collect data and execute computations to train ML models.
$\mathcal{D}^{(i)}$	The local dataset $\mathcal{D}^{(i)}$ at node $i \in \mathcal{V}$.
m_i	The number of data points (sample size) contained in the local dataset $\mathcal{D}^{(i)}$ at node $i \in \mathcal{V}$.
$\mathcal{N}^{(i)}$	The neighbourhood of the node i in an empirical graph is the set of other nodes that are connected by end edge with i.
$\mathbf{x}^{(i,r)}$	The feature vector of the r-th data point in the local dataset $\mathcal{D}^{(i)}$.
$y^{(i,r)}$	The label of the r-th data point in the local dataset $\mathcal{D}^{(i)}$ at node $i \in \mathcal{V}$.
$L^{(\mathrm{d})}\big(\mathbf{x}, h\big(\mathbf{x}\big), h'\big(\mathbf{x}\big)\big)$	The loss incurred by a "external" hypothesis h' on a data point with features $\mathbf{x}$ and predicted label $h\big(\mathbf{x}\big)$ that is obtained from some local hypothesis.

1 Introduction

Many important application domains generate decentralized collections of local datasets that are related via an intrinsic network structure [6]. Two timely application domains that generate such networked data are (i) the high-precision management of pandemics and (ii) the Internet of Things (IoT) [7, 8]. Here, local datasets are generated by smartphones, wearables or other IoT devices [9, 10]. These local datasets are related via physical contact networks, social networks [11], co-morbidity networks [12], or communication networks [13].

FL is an umbrella term for distributed optimization techniques to train machine learning (ML) models from decentralized collections of local datasets [14–18]. These methods implement computations, such as gradient computations (see Section 5) for ML model training, at the location of data generation. This design philosophy is different from a naive application of ML techniques which is to first collect all local datasets at a single location (computer). This pooled data is then fed into a conventional ML method such as linear regression.

The distributed training of ML models, at locations close to the actual data generation, can be beneficial in several aspects [19]:

- **Privacy.** FL methods are appealing for applications involving sensitive data (such as healthcare) as they do not require the exchange of raw data but only model (parameter) updates [16, 17]. By exchanging only model updates, FL methods are considered privacy-friendly in the sense of not leaking (too much) sensitive information that is contained in the

local datasets (see Section 12).

- **Robustness.** By relying on decentralized data and computation, FL methods offer robustness against hardware failures (such as "stragglers") and data poisonings (see Section 13).

- **Parallel Computing.** A main application domain for FL are mobile networks, consisting of humans equipped with smartphones. We can interpret a mobile network as a parallel computer which is constituted by smartphones that can communicate via radio links. This parallel hardware allows to speed up computational tasks such as the computation of gradients required to train ML models (see Section 5).

- **Trading Computation against Communication.** Consider a FL application where local datasets are generated by low-complexity devices at remote locations that cannot be easily accessed. The cost of communicating raw local datasets to some central unit (which then trains a single global ML model) might be much higher than the computational cost incurred by using the low-complexity devices to (partially) train ML models [20].

- **Personalization.** FL can be used to train personalized ML models for collections of local datasets, which might be generated by smartphones (and their users) [21]. A key challenge for ensuring personalization is the heterogeneity of local datasets [22, 23]. Indeed, the statistical properties of different local datasets might vary significantly such they cannot be well modelled as independent and identically distributed (i.i.d.). Each local dataset induces a separate learning task that consists of learning

useful parameter values for a local model. This course discusses FL methods to train personalized models via combining the information carried in decentralized and heterogeneous data (see Section 9.3 and Section 9.4).

1.1 Prerequisites

The main mathematical structure used to study and design FL algorithms is the Euclidean space $\mathbb{R}^d$. We therefore expect some famililartiy with the algebraic and geometric structure of $\mathbb{R}^d$. By algebraic structure of $\mathbb{R}^d$, we mean the (real) vector space obtained from the elements ("vectors") in $\mathbb{R}^d$ along with the usual definitions of vector addition and mulitplication by scalars in $\mathbb{R}$ [24, 25]. We will make heavy use of concepts from linear algebra to represent and maninpulate data and ML models.

The metric structure of $\mathbb{R}^d$ will be used to study the (convergence) behaviour of FL algorithms. In particular, we will study FL algorithms that are obtained as fixed-point iterations of some non-linear operator on $\mathbb{R}^d$ which depends on the data (distribution) and ML models used within a FL system. The computational properties (such as convergence speed) of these FL algorithms can then be characterized via the contraction properties of the underlying operator [26].

A main tool for the design the FL algorithms are varitions of gradient descent (GD). These gradient-based methods are based on approximating a differentiable function $f(\mathbf{x})$ locally by a linear function given by the gradient $\nabla f(\mathbf{x})$. We therefore expect some familiarity with multivariable calculus [5].

1.2 Related Courses

In what follows we briefly explain how this course CS-E4740 relates to selected courses at Aalto University.

- **CS-EJ3211 - Machine Learning with Python.** Teaches the application of basic ML methods using the Python package (library) `scikit-learn` [27]. CS-E4740 couples a network of basic ML methods using regularization techniques to obtain tailored (personalized) ML models for local datasets. This coupling is required to adaptive pool local datasets obtain sufficiently large training sets for the personalized ML models.

- **CS-E4510 - Distributed Algorithms.** Teaches basic mathematical tools for the study and design of distributed algorithms that are implemented via distributed systems [28]. FL is enabled by distributed algorithms to train ML models from decentralized data (see Section 10).

- **CS-C3240 - Machine Learning (spring 2022 edition).** Teaches basic theory of ML models and methods [1]. CS-E4740 combines the components of basic ML methods, such as data representation and models, with network models. In particular, instead of a single dataset and a single model (such as a decision tree), we will study networks of local datasets and local models.

- **ABL-E2606 - Data Protection.** This course discusses important legal constraints ("laws"), including the European general data protection regulation (GDPR), for the use of data and, in turn, for the design of trustworthy FL methods (see Part III)).

- **MS-C2105 - Introduction to Optimization.** This course teaches basic optimisation theory and how to model applications as (linear, integer, and non-linear) optimization problems. CS-E4740 uses optimization theory and methods to formulate FL problems (see Section 8) and design FL methods (see Section 10).

- **ELEC-E5424 - Convex Optimization.** This course teaches advanced optimisation theory for the important class of convex optimization problems [29]. Convex optimization theory and methods can be used for the study and design of FL algorithms.

1.3 Outline of the Course

Our course is roughly divided into three parts:

- Part I discusses the basic components and principles of (non-federated) ML: Section 2 introduces data, models and loss functions as three main components of ML. Section 3 explains how these components are combined via empirical risk minimization (ERM). The regularization of ERM, via manipulating its three main components, is discussed in Section 4. We then explain when and how to solve regularized ERM via simple GD methods in Section 5. Overall, Part I serves two main purposes: (i) to briefly recap basic concepts of ML in a simple centralized setting and (ii) highlight ML techniques (such as regularization) that are particulary relevant for the design and analysis of FL methods.

- Part II revolves around a main data structure, i.e., the empirical graph, and a main design principle, referred to as GTV minimization

(GTVMin), for the FL systems. Section 7 introduces an empirical graph as a representation for collections of local datasets and corresponding tailored models. The undirected and weighted edges of the empirical graph represent statistical similarities between local datasets. Section 8 formulates FL as an instance of regularized empirical risk minimization (RERM) which we refer to as GTVMin. GTVMin uses the variation of personalized model parameters across edges in the empirical graph as regularizer. We will see that GTVMin couples the training of tailored (or "personalized") ML models such that well-connected nodes (clusters) in the empirical graph will obtain similar trained models. Section 9 derives some main flavours of FL as special cases of GTVMin. Section 10 applies well-known optimization methods to GTVMin, resulting in FL algorithms for distributed training of tailored ("personalized") models.

- Part III discusses requirements and design aspects for trustworthy FL: Section 11 enumerates seven key requirements for trustworthy artificial intelligence (AI) that have been put forward by the European Union. We then discuss how FL algorithms can esnure privacy protection in Section 12. Section 13.2 discusses the vulnerability of FL methods against backdoor attacks.

1.4 Main Goal and Learning Outcomes

The main goal of the course is to teach students a mathematical toolbox for the analysis and design of FL algorithms. This toolbox revolves around the formulation of a given FL application as an optimization problem over

a undirected graph $\mathcal{G} = (i, i')$ whose nodes $i \in \mathcal{V}$ represent individual local datasets. We refer to this refer as the empirical graph of a collection of local datasets (see Section 7.1). The edges (i, i') of $\mathcal{G}$ connect local datasets with similar statistical properties.

FL applications can be formalized as instances of the generic optimization problem

$$\min_{\mathbf{w}^{(i)}} \sum_{i \in \mathcal{V}} L_i \left(\mathbf{w}^{(i)} \right) + \lambda \sum_{\{i,i'\} \in \mathcal{E}} A_{i,i'} d \left(\mathbf{w}^{(i)}, \mathbf{w}^{(i')} \right). \quad (1)$$

We refer to this problem as GTVMin and devote much of Part II to the discussion of its properties. The optimization variables $\mathbf{w}^{(i)}$ in (1) are local model parameters at the nodes $i \in \mathcal{V}$ of an empirical graph. The objective function in (1) consists of two components: The first component is a sum over all nodes of the loss values $L_i \left(\mathbf{w}^{(i)} \right)$ incurred by local model parameters at each node i. The second component is the sum of local model parameters varitions across the edges $\{i, i'\}$ of the empirical graph.

1.5 Acknowledgements

The author is grateful for feedback on the lecture notes received from: Dr. Shamsiiat Abdurakhmanova, Dr. Yu Tian, Dr. Olga Kuznetsova, Alexander Pavlyuk, Junyuan Fang, Matti Ahokas, Felix Knop, Iina Hyytiäinen, Maria Montoya Freire and Ariana Marta. Some of the figures have been prepared with the help of Dr. Yu Tian. Special thanks to Amirhossein Mohammadi for carefully proof-reading parts of the manuscript.

1.6 Exercises

Exercise 1.1. Parallel Computing (5 points). Assume that you have a single computer that is able to sum two numbers per second. You can assume that the computer has sufficient storage memory which can be read from and written to at very high speed. How long does the computer take to compute the sum of 100 numbers $\mathcal{D} = \{x^{(1)}, \ldots, x^{(100)}\}$.

Now assume that you have two identical such computers which are connected via a high-speed network. How much faster can we compute the sum of the numbers in $\mathcal{D}$ using these two computers? Hint: see [30, Sec. 1.2.3.].

Exercise 1.2. Communication Bottleneck (5 points). Consider two laptops, each having a webcam taking pictures with 1080×1080 pixels. These two laptops are connected via a wireless `Bluetooth` connection with average bitrate of $1 \cdot 10^6$ bits/s [31]. What is the maximum rate at which one laptop can record images and transmit it to the other laptop? You can ignore any compression techniques and assume that images are stored as bitmaps with each pixel encoded by three bytes (red, green and blue intensities on scale $0, \ldots, 255$).

Part I

Machine Learning: The Basics

2 Three Components of ML

ML revolves around computing accurate predictions for a quantity of interest. These predictions are computed by evaluating a hypothesis map $h(\mathbf{x})$ which is fed with the features $\mathbf{x}$ of a data point (see Section 2.1). The value $h(\mathbf{x})$ is a prediction for some quantity of interest which we refer to as the label of a data point.

In general, the hypothesis h is learnt or optimized based on the discrepancy between predictions and observations. The space of possible hypothesis maps, from which a ML method can choose from, is referred to as hypothesis space or model (see Section 2.2).

To choose or learn a useful hypothesis out of a hypothesis space we need a measure for the quality of the predictions obtained from a hypothesis. To this end, ML methods use loss functions to obtain a quantitative measure for the prediction errors (see Section 2.3).

Different ML methods use different choices for data points (their features and label), the hypothesis space (or model) and loss function. The design choices for these components are guided by computational aspects and statistical aspects. In what follows we discuss each of these design choices for a toy application which is to predict the maximum daytime temperature.

2.1 Design Choice: Data [1, Ch. 2.1]

We consider data as collections of elementary data points that carry relevant information. Each individual data point is characterized by several properties that we group into features and labels.

The features $\mathbf{x}$ are low-level properties that we can measure or compute easily in an automated fashion. Examples for such low-level properties are physical quantities that can be measured with sensing devices such as temperature, humidity or light intensity. Note that, by our (informal) definition of features, also any quantity that can be computed, e.g., by applying a pre-trained artificial neural network (ANN), from physical measurements are features. The feature space $\mathcal{X}$ collects all possible values that the features $\mathbf{x} \in \mathcal{X}$ of a data point can take on. This course focuses on ML methods that use an Euclidean space $\mathbb{R}^d$ as the feature space.

Besides their features, data points are characterized by labels y which are high-level properties or quantities of interest. The label space $\mathcal{Y}$ collects the possible values that the labels of a data point can take on, i.e., $y \in \mathcal{Y}$. ML applications involving a "numeric" label space $\mathcal{Y}$, such as $\mathcal{Y} = \mathbb{R}$, are referred to as regression problems. ML applications involving a finite label space $\mathcal{Y}$ are referred to as classification problems.

In contrast to features, determining the label of a data point is more difficult and typically requires to consult a human domain expert. Another source for the label of a data point might be the predictions obtained from a reference (or "teacher") model that has been trained on a different but related dataset. The federated learning (FL) methods discussed in part II use this approach to leverage similarities between learning tasks arising from datasets with similar statistical properties.

Note that the distinction between features and labels is blurry. Indeed, one and the same property of a data point might be considered as features in one application, while it might be useful to consider it as a label in another

application. Self-supervised learning methods exploit this design freedom to obtain large amounts of labeled data points required to train large ANNs [32].

Our definition of data points as elementary information-carrying objects is quite abstract and therefore flexible. Indeed, the definition of what we consider as data points is an important design choice. As an example, we could use data points to represent time periods such as days. One data point would then represent the day 12-Mar-2020. Another data point represents the day 12-Apr-2019. Instead of entire days, we could also define data points as shorter time-periods such as hours or 10-minute intervals.

After we have defined data points we are faced with two further design choices:

- **Which features to use? (feature selection).** Which properties should we use as features of a data point? In the above example, involving a data point 12-Apr-2019, we could use the maximum daytime temperatures during the previous two days as features. Another choice for the features would be to use these temperatures during the previous ten days as features.

- **Which label to use? (learning task).** What property of a data point is a quantity of interest or label y? For the above weather forecasting example, we could define the label as the average temperature during the next two days. Another choice for the label y would be the average temperature during the next ten days. We could also define several different labels $y_1, y_2, \ldots$, for the same data point. For example, y_j could be the average temperature during the next j days.

The design choices for the data points, their features and label must take into account statistical aspects, computational aspects and trustworthiness of the resulting ML method. For example, using a large number of features will typically result in a higher computational complexity of the resulting ML method. On the other hand, a large number of features might be necessary to enable learning an accurate hypothesis. However, to ensure explainability of the resulting ML method, we might prefer using few interpretable features [33].

The design choice for the labels of a data point must also take into account the trustworthiness of the resulting ML methods. If a data point represents a user of a social network or online marketplace, then we might avoid labels that allow to predict individual behaviour and, in turn, enable manipulation of the human user.

2.2 Design Choice: Model [1, Ch. 2.2]

Consider a ML application that involves data points, each characterized by features $\mathbf{x} \in \mathcal{X}$ and a label $y \in \mathcal{Y}$. The overall goal of ML is to find or learn a hypothesis map,

$$h : \mathcal{X} \to \mathcal{Y} : \mathbf{x} \mapsto h(\mathbf{x}). \tag{2}$$

that computes a prediction $\hat{y} = h(\mathbf{x})$ for the label $y \in \mathcal{Y}$ of a data point solely from its features $\mathbf{x} \in \mathcal{X}$.

Practical ML methods have only finite computational resources to evaluate a hypothesis map (2) and evaluate them (see Section 2.3). Any practical ML method must therefore restrict itself to a (tiny) subset of the set $\mathcal{Y}^{\mathcal{X}}$ that contains all possible hypothesis maps. This subset of computationally feasible ("affordable") hypothesis maps is referred to as the hypothesis space

(or model),

$$\mathcal{H} := \{h : h(\mathbf{x}) \text{ can be computed efficiently.}\} \tag{3}$$

As depicted in Figure 1, ML methods typically use a hypothesis space $\mathcal{H}$ that is a tiny subset of $\mathcal{Y}^{\mathcal{X}}$.

Similar to the features and labels used to characterize data points, also the hypothesis space underlying a ML method is a design choice. The design choice for the hypothesis space involves a trade-off between computational complexity and accuracy of the resulting ML methods [1, Ch. 2]. Note that the informal criterion "can be computed efficiently" depends on the available computational resources. Graphic processing units can efficiently compute matrix-vector operations that constitute a hypothesis map [34]. However, we can also implement ML methods using optical computers that realize ANNs using diffractive layers [35].

The hypothesis space should be large enough to contain at least one hypothesis that allows to accurately predict the label from the features of a data point. However, it should not be too large such that ML methods can efficiently search it for a useful hypothesis. Another limitation for the the size or dimension of the hypothesis space is the amount of available data (see Section 3.3). Beside the computational aspects and statistical aspects, a third design aspect for the model is the explainability of the resulting ML method [36, 37].

During the rest of this course, we will focus on FL methods that use linear models [1, Ch. 3]. In its most basic form, linear models can only be used for data points characterized by numeric features $x_1, \ldots, x_d \in \mathbb{R}$. It is convenient

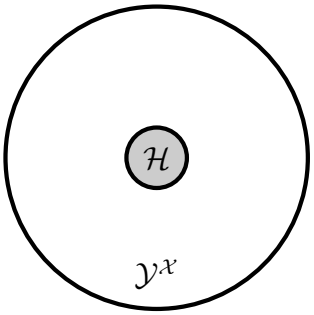

Figure 1: The hypothesis space $\mathcal{H}$ is a (typically very small) subset of the (typically very large) set $\mathcal{Y}^{\mathcal{X}}$ of all possible maps from the feature space $\mathcal{X}$ to the label space $\mathcal{Y}$.

to collect those features of a data point into a feature vector

$$\mathbf{x} := \big(x_1, \ldots, x_d\big)^T \in \mathbb{R}^d.$$

Formally, for a given number d of numeric features, a linear model is the hypothesis space which consists of all linear maps,

$$\mathcal{H}^{(d)} := \left\{ h(\mathbf{x}) = \mathbf{w}^T \mathbf{x} : \mathbf{w} \in \mathbb{R}^d \right\}. \tag{4}$$

Note that (122) defines an entire family of hypothesis spaces, which is indexed by the number d of features that are linearly combined to form the prediction $h(\mathbf{x})$. The design choice of d is guided by computational aspects (smaller d means less computation), statistical aspects (larger d might reduce prediction error) and interpretability (favouring a linear model with few features).

Each individual hypothesis in the linear model $\mathcal{H}^{(d)}$ is fully specified by a weight vector $\mathbf{w} \in \mathbb{R}^d$, which we indicate by the notation $h^{(\mathbf{w})}(\mathbf{x}) := \mathbf{w}^T \mathbf{x}$. Thus, the linear model (122) is an example for a parametrized model. Another example for a parametrized model is an ANN [1, Ch. 2, 3].

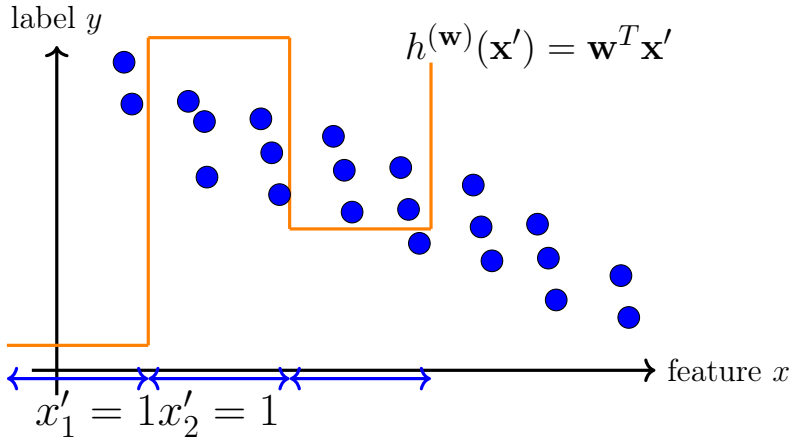

Figure 2: We can extend a linear model (122) by replacing the original feature x of a data point with transformed features $\mathbf{x}' = \mathbf{\Phi}(\mathbf{x})$. These transformed features are obtained by applying a feature map $\mathbf{\Phi}$ to the original features.

Upgrading Linear Models via Feature Maps. We can use the linear model $\mathcal{H}^{(d)}$ as a building block for constructing non-linear models. For example, we could use a linear model for the transformed features $\mathbf{x}' = \mathbf{\Phi}(\mathbf{x})$ (see Figure 2). The resulting model consists of all maps that are concatenations of linear maps with the feature transformation or map $\mathbf{\Phi}$. Kernel methods use feature maps obtained from a kernel function. The input layers of an ANN (whose output layer is linear) are another instance of a feature map (Figure 3).

Non-Numeric Features. We can use the linear model (122) only for data points whose features are numeric vectors $\mathbf{x} = (x_1, \ldots, x_d)^T \in \mathbb{R}^d$. In some application domains, such as natural language processing [38], there is no obvious natural choice for numeric features. However, since ML methods based on the hypothesis space (122) can be implemented efficiently using widely

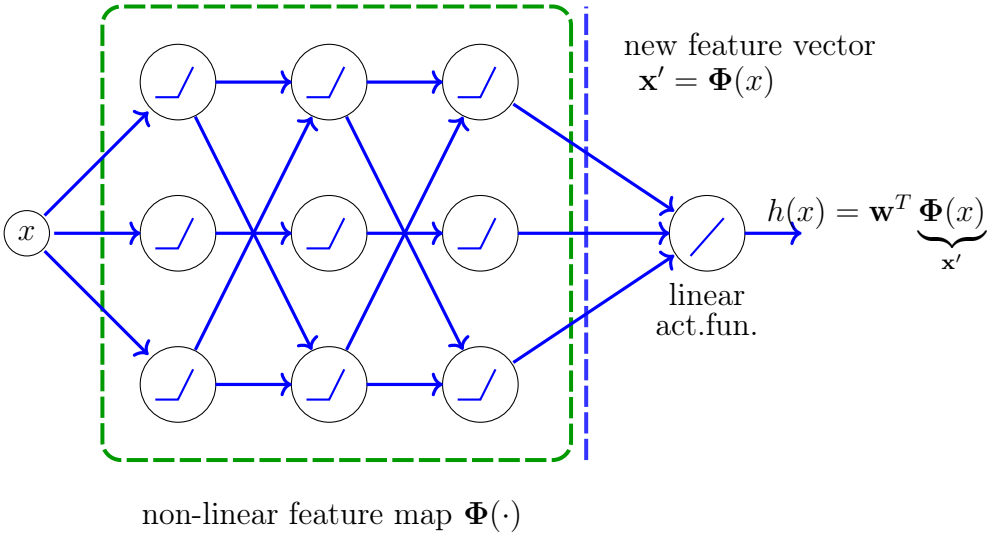

Figure 3: The hypothesis space spanned by an ANN (whose output layer is linear) coincides with the linear model applied to the activations $\mathbf{\Phi}(x)$ of the penultimate layer. We can interpret these activations as transformed features that are obtained by applying the input layers to the original features. These input layers implement a non-linear feature map $\mathbf{\Phi}(\cdot)$.

available software and hardware, it might be useful to construct numerical features even for non-numeric data. For text data, there has been significant progress recently on methods that map a human-generated text into sequences of vectors (see [38, Chap. 12] for more details). Moreover, [1, Sec. 9.3] shows how to construct numeric features for data points that are related by some notion of similarity.

2.3 Design Choice: Loss [1, Ch. 2.3]

ML methods find or learn a useful hypothesis out of an entire hypothesis space (or model) $\mathcal{H}$. To measure the quality of a hypothesis, ML methods use a loss function. Mathematically, a loss function is a map

$$L : \mathcal{X} \times \mathcal{Y} \times \mathcal{H} \to \mathbb{R}_+ : \big(\big(\mathbf{x}, y\big), h\big) \mapsto L\left((\mathbf{x}, y), h\right)$$

which assigns a pair consisting of a data point $(\mathbf{x}, y)$, itself characterized by features $\mathbf{x}$ and label y, and a hypothesis $h \in \mathcal{H}$ the non-negative real number $L\left((\mathbf{x}, y), h\right)$.

The loss value $L\left((\mathbf{x}, y), h\right)$ quantifies the discrepancy between the true label y and the prediction $h(\mathbf{x})$. A small (close to zero) value $L\left((\mathbf{x}, y), h\right)$ indicates a small discrepancy between the prediction $h(\mathbf{x})$ and the actual (or true) label y of a data point. Figure 4 depicts a loss function for a given data point, with features $\mathbf{x}$ and label y, as a function of the hypothesis $h \in \mathcal{H}$. Note that Figure 4 needs to be understood as a cartoon as we use the horizontal axis to represent a hypothesis space (or model) $\mathcal{H}$. Note that the elements of $\mathcal{H}$ are hypothesis maps $h : \mathcal{X} \to \mathcal{Y}$.

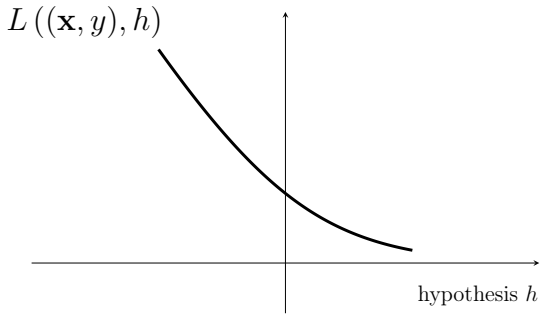

Figure 4: Some loss function $L\left((\mathbf{x}, y), h\right)$ for a fixed data point, with features $\mathbf{x}$ and label y, and varying hypothesis h. ML methods try to find (learn) a hypothesis that incurs minimum loss.

We could make Figure 4 more rigorous, by using the horizontal axis to represent a single numeric parameter of a hypothesis map such as its value $h(\mathbf{x})$ for the features of the considered data point (see, e.g., Figure 9). Another option would be to use the horizontal axis to represent the single model parameters of the linear model $\mathcal{H}^{(1)}$.

Much like the choice for the hypothesis space $\mathcal{H}$ used in a ML method, also the loss is a design choice. We will discuss some widely used examples for loss function in Section 2.3.1 and Section 2.3.2. Like data and model, the loss should be chosen in view of three design aspects

- computational aspects,
- statistical aspects,
- trustworthiness of the resulting ML method.

The choice for the loss function impacts the computational complexity of searching the hypothesis space for a hypothesis with minimum loss. Consider

a ML method that uses a parametrized hypothesis space and a loss function that is a convex and differentiable (smooth) function of the parameters of a hypothesis. In this case, searching for a hypothesis with small loss can be done efficiently using the gradient-based methods discussed in Section 5.

Some ML methods use a loss function that is neither convex nor differentiable. The minimization of such loss functions tends to be computationally more challenging (requiring more computation). More discussion about the computational complexities of different types of loss functions can be found in [1, Sec. 4.2.].

Beside computational aspects, the choice for the loss function should also take into account statistical aspects. For example, some loss functions result in ML methods that are more robust against outliers [1, Sec. 3.3]. We might also use probabilistic models for the data generated in an ML application to guide the design choice for the loss function. In particular, the maximum likelihood principle suggests to use the logarithm of a likelihood ("log-likelihood") function as a loss function. This likelihood function is determined by the (parametrized) probability distribution of the data points.

A third aspect for the choice of loss function is its explainability. Section 2.3.2 discusses loss functions to measure the quality of binary classifier. It seems natural to measure the quality of a binary classifier by the average number of wrongly classified data points, which is precisely the average 0/1 loss (8) (see Section 2.3.2).

In contrast to its appealing interpretation as error-rate, the computational aspects of the average 0/1 loss (which is equivalent to accuracy) are less pleasant. Minimizing the average 0/1 loss to learn an accurate hypothesis

amounts to a non-convex and non-smooth optimization problem which is typically computationally more challenging compared to minimizing a convex function.

Section 2.3.2 introduces the logistic loss as a computationally attractive alternative choice for the loss function in binary classification problems. Learning a linear hypothesis with minimum (average) logistic loss amounts to minimizing a differentiable convex function. Section 5 discusses practically useful gradient-based methods to minimize such functions.

To summarize, the computational aspects, the statistical aspects and the explainability of ML methods typically result in conflicting design goals for a loss function. A loss function that has favourable statistical properties might incur a high computational complexity of the resulting ML method. Loss functions that result in computationally efficient ML methods might not allow for an easy interpretation. For example, it is hard to grasp the meaning of a binary classifier achieving an average logistic loss of 10^{-1}. On the other hand, the value of the average 0/1 loss is often considered as an (approximate) error rate. It can be helpful to use different loss functions for the search of a good hypothesis (model training) and for its final evaluation (see Figure 5).

For example, in a binary classification problem, we might use the logistic loss to search for (learn) an accurate hypothesis using the optimization methods in Section 5. After having found (learnt) a hypothesis, we use the average 0/1 loss for the final performance evaluation. The 0/1 loss is appealing for this purpose as it can be interpreted as an error (or misclassification) rate. The loss function used for the final performance evaluation of a learnt hypothesis is sometimes referred to as metric.

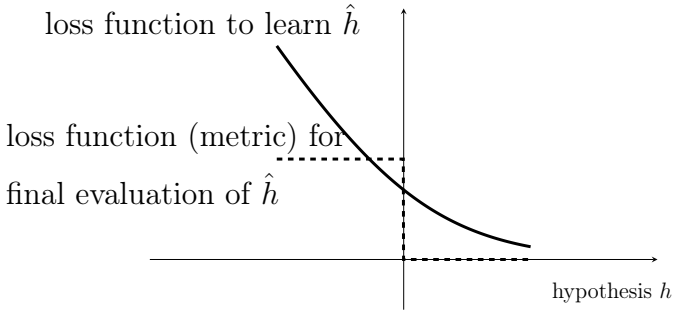

Figure 5: Two different loss functions for a given data point and varying hypothesis h. One of these loss functions (solid curve) is used to learn a good hypothesis by minimizing the loss. We use another loss function (dashed curve) to evaluate the performance of the learnt hypothesis $\hat{h}$.

2.3.1 Loss Functions for Numeric Labels

For ML problems involving data points with numeric labels $y \in \mathbb{R}$, a widely used (first) choice for the loss function is the squared error loss

$$L\left((\mathbf{x}, y), h\right) := \big(y - \underbrace{h(\mathbf{x})}_{=\hat{y}}\big)^2. \tag{5}$$

The squared error loss (5) depends on the features $\mathbf{x}$ only via the predicted label value $\hat{y} = h(\mathbf{x})$. We can evaluate the squared error loss solely using the prediction $h(\mathbf{x})$ and the true label value y. Besides the prediction $h(\mathbf{x})$, no other properties of the features $\mathbf{x}$ influence the value of the squared error loss.

We will slightly abuse notation and use the shorthand $L\left(y, \hat{y}\right)$ for any loss function that depends on the features $\mathbf{x}$ only via the predicted label $\hat{y} = h(\mathbf{x})$. Figure 6 depicts the squared error loss as a function of the prediction error

$y - \hat{y}$. It also depicts the (special case of the) Huber loss

$$L\left((\mathbf{x}, y), h\right) := \left|h(\mathbf{x}) - y\right|. \tag{6}$$

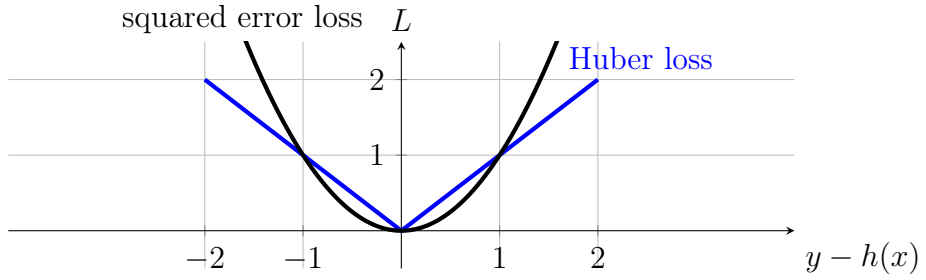

Figure 6: Two widely used loss functions to measure how well a hypothesis predicts a numeric label are the squared error loss (5) and the Huber loss (6). Note that, for a given hypothesis h, we can evaluate the loss function only if we know the features $\mathbf{x}$ and the label y of the data point.

The squared error loss (5) has appealing computational and statistical properties. For linear predictor maps $h(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$, the squared error loss is a convex and differentiable function of the parameter vector $\mathbf{w}$. This allows, in turn, to efficiently search for the optimal linear predictor using efficient iterative optimization methods (see Section 5). The squared error loss also has a useful interpretation in terms of a probabilistic model for the features and labels: Minimizing the squared error loss is equivalent to maximum likelihood estimation within a linear Gaussian model [39, Sec. 2.6.3].

Another loss function used in regression problems is the absolute error loss (6), which is a special case of the Huber loss [1, Sec. 3.3]. It can be shown that ML methods using absolute error loss are more robust against a few outliers in the training set compared to methods using squared error

loss (see [1, Sec. 3.3.]). The improved robustness comes at the expense of increased computational complexity of minimizing the non-differentiable loss (6) compared to the convex and differentiable squared error loss (5).

2.3.2 Loss Functions for Categorical Labels

Classification problems involve data points whose labels take on values from a discrete label space $\mathcal{Y}$. In what follows, unless stated otherwise, we focus on binary classification problems with label space $\mathcal{Y} = \{-1, 1\}$. Classification methods learn a hypothesis or classifier that maps the features $\mathbf{x}$ of a data point to a predicted label $\hat{y} \in \mathcal{Y}$.

It seems that we cannot use linear hypothesis maps $h^{(\mathbf{w})}(\mathbf{x}) := \mathbf{w}^T\mathbf{x}$ (see (122)) for a classification problem. Indeed, the function value $h(\mathbf{x})$ is a real-number and (most likely) does not belong to the label space $\mathcal{Y} = \{-1, 1\}$. However, a simple post-processing step turns a linear hypothesis $h^{(\mathbf{w})}$ into a binary classifier: We classify a data point as

$$\hat{y} = 1 \text{ if } h^{(\mathbf{w})}(\mathbf{x}) > 0 \text{ , and } \hat{y} = -1 \text{ otherwise.} \tag{7}$$

Thus, the sign of $h^{(\mathbf{w})}(\mathbf{x})$ determines the predicted label $\hat{y} \in \{-1, 1\}$. We will abuse notation and also refer to the linear map $h^{(\mathbf{w})}(\mathbf{x})$ as a binary classifier.

Besides its sign, we can also use the absolute value $|h^{(\mathbf{w})}(\mathbf{x})|$ to convey useful information. In particular, we might use the absolute value $|h^{(\mathbf{w})}(\mathbf{x})|$ as a measure for the confidence in the prediction $\hat{y}$ (7).

In principle, we can measure the quality of a hypothesis when used to classify data points using the squared error loss (5). However, the squared error loss is a poor quality measure for a binary classifier. Figure 7 illustrates

how the squared error loss of a linear hypothesis can be (highly) misleading for evaluating the performance of a binary classifier.

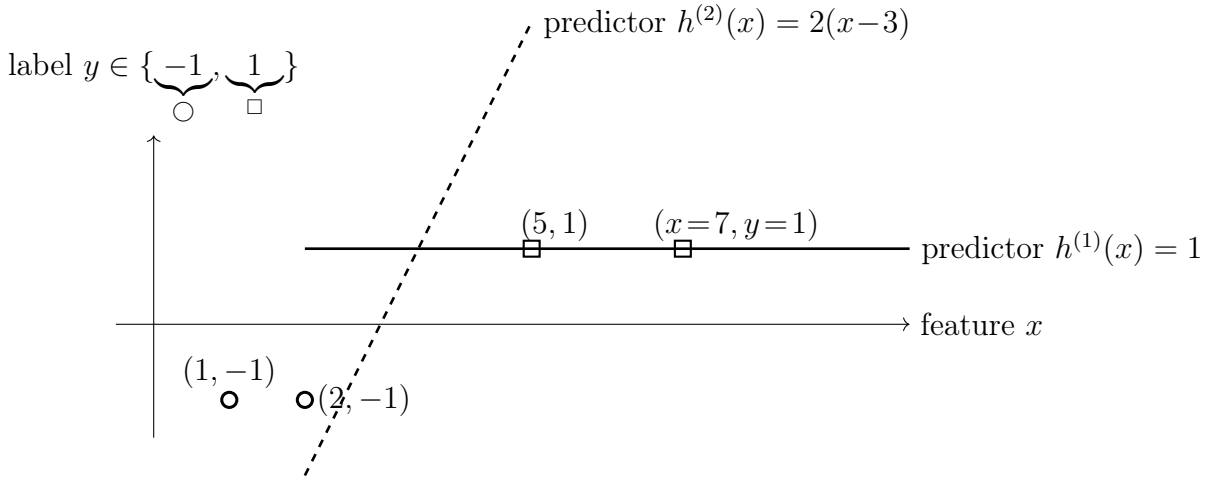

Figure 7: A training set consisting of four data points with binary labels $\hat{y}^{(r)} \in \{-1, 1\}$. Minimizing the squared error loss (5) would prefer the (poor) classifier $h^{(1)}$ over the (reasonable) classifier $h^{(2)}$.

Figure 7 depicts a dataset $\mathcal{D}$ consisting of $m = 4$ data points with binary labels $y^{(r)} \in \{-1, 1\}$, for $r = 1, \ldots, m$. The figure also depicts two different hypotheses $h^{(1)}(x)$ and $h^{(2)}(x)$ for classifying data points (via their signs).

The classifications $\hat{y}$ obtained from the hypothesis $h^{(2)}(x)$ would perfectly match the labels of the four training data points in Figure 7 since $h^{(2)}\big(x^{(r)}\big) \geq 0$ if and only if $y^{(r)} = 1$. In contrast, the classifications $\hat{y}^{(r)}$ obtained by thresholding $h^{(1)}(x)$ are incorrect for data points with $y = -1$.

Looking at $\mathcal{D}$, we might prefer using $h^{(2)}$ over $h^{(1)}$ to classify data points. However, the squared error loss $\big(y - h^{(2)}(x)\big)^2$ incurred by the (reasonable) classifier $h^{(2)}$ is much larger than the squared error loss incurred by the (poor)

classifier $h^{(1)}$. The squared error loss is typically a bad choice for assessing the quality of a hypothesis map that is used for classifying data points into different categories.

Generally speaking, we want the loss function to punish (deliver large values for) a hypothesis that results in a wrong classification ($\hat{y} \neq y$) with high confidence ($|h(\mathbf{x})|$ is large). On the other hand, a useful loss function should not punish (deliver small values for) a hypothesis that results a correct classification ($\hat{y} = y$) with high confidence ($|h(\mathbf{x})|$ is large). By its very definition, the squared error loss (5) yields always large values if the confidence $|h(\mathbf{x})|$ is large, no matter if the resulting classification (obtained after thresholding) is correct or wrong. This is clearly an undesirable property for a loss function that is used to score a hypothesis that is used to classify data points.

We now discuss some loss functions which have proven useful for assessing the quality of a hypothesis that is used to classify data points. Unless stated otherwise, the formulas for these loss functions are valid only if the label values are the real numbers -1 and 1, i.e., for the label space $\mathcal{Y} = \{-1, 1\}$. These formulas need to be modified accordingly if different label values are used. For example, instead of the label space $\mathcal{Y} = \{-1, 1\}$, we could equally well use the label space $\mathcal{Y} = \{0, 1\}$, or $\mathcal{Y} = \{\square, \triangle\}$ or $\mathcal{Y} = \{$ "Class 1", "Class 2"$\}$.

A natural choice for the loss function can be based on the requirement that a reasonable hypothesis should deliver correct classifications, $\hat{y} = y$ for any data point. This suggests to learn a hypothesis $h(\mathbf{x})$ with small 0/1 loss

$$L\left((\mathbf{x}, y), h\right) := \begin{cases} 1 & \text{if } y \neq \hat{y} \\ 0 & \text{else,} \end{cases} \quad \text{with } \hat{y} = \begin{cases} 1 & \text{if } h(\mathbf{x}) \geq 0 \\ 0 & \text{else.} \end{cases} \tag{8}$$

Figure 9 illustrates the 0/1 loss (8) for a data point with features $\mathbf{x}$ and label $y = 1$ as a function of the value $h(\mathbf{x})$. The 0/1 loss is equal to zero if the hypothesis yields a correct classification $\hat{y} = y$. For a wrong classification $\hat{y} \neq y$, the 0/1 loss is equal to one.

The 0/1 loss (8) is conceptually appealing when data points are interpreted as realizations of independent and identically distributed (i.i.d.) random variables (RVs) with common probability distribution $p(\mathbf{x}, y)$ (see Section 3.1). Given m realizations $\left(\mathbf{x}^{(1)}, y^{(1)}\right), \ldots, \left(\mathbf{x}^{(m)}, y^{(m)}\right)$ of these i.i.d. RVs, with high probability

$$(1/m) \sum_{r=1}^{m} L\left((\mathbf{x}^{(r)}, y^{(r)}), h\right) \approx p(y \neq h(\mathbf{x})) \tag{9}$$

for sufficiently large sample size m.

A precise formulation of the approximation (9) can be obtained from the law of large numbers [40, Section 1]. We can apply the law of large numbers since the loss values $L\left((\mathbf{x}^{(r)}, y^{(r)}), h\right)$ are realizations of i.i.d. RVs. It is customary to indicate the average 0/1 loss of a hypothesis h indirectly via the accuracy $1 - (1/m) \sum_{r=1}^{m} L\left((\mathbf{x}^{(r)}, y^{(r)}), h\right)$.

In view of (9), the 0/1 loss (8) seems a very natural choice for assessing the quality of a classifier if our goal is to enforce correct classifications $\hat{y} = y$. The appealing statistical aspects of the 0/1 loss come at the expense of an increased computational complexity. Indeed, for a given data point $(\mathbf{x}, y)$, the 0/1 loss (8) is non-convex and non-differentiable when viewed as a function of the hypothesis h. Thus, ML methods that use the 0/1 loss to learn a hypothesis require advanced optimization methods to solve the resulting learning problem (see [1, Sec. 3.8.]).

To avoid the non-convexity of the 0/1 loss (8) we can approximate it by a convex "surrogate" loss function. One such approximation of the 0/1 loss is the hinge loss

$$L\left((\mathbf{x},y),h\right) := \max\{0, 1 - yh(\mathbf{x})\}. \tag{10}$$

Figure 9 depicts the hinge loss (10) as a function of the hypothesis $h(\mathbf{x})$. The hinge loss (10) becomes minimal (zero) for a correct classification ($\hat{y} = y$) with sufficient confidence $|h(\mathbf{x})| \geq 1$. For a wrong classification ($\hat{y} \neq y$), the hinge loss increases monotonically with increasing confidence $|h(x)|$ in the wrong classification.

While the hinge loss avoids the non-convexity of the 0/1 loss, it still is a non-differentiable function of $h(\mathbf{x})$. Optimizing a non-differentiable loss function is typically harder, requiring more iterations of iterative methods, compared to optimizing a differentiable loss function [41, 42]. Figure 8 depicts an example of a differentiable and a non-differentiable convex loss function, respectively. Iterative optimization methods can typically only explore the loss function locally around a current choice for the model parameters. For the non-differentiable loss function in Figure 8, such algorithms cannot determine how far the current model parameters are from the minimum.

Besides the 0/1 loss and the hinge loss, another popular loss function for binary classification problems is the logistic loss

$$L\left((\mathbf{x},y),h\right) := \log(1 + \exp(-yh(\mathbf{x}))). \tag{11}$$

The logistic loss (11) is used in logistic regression [1, Ch. 3] to measure the usefulness of a linear hypothesis map $h^{(\mathbf{w})}(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$.

Figure 9 depicts the logistic loss (11) as a function of the hypothesis $h(\mathbf{x})$. The logistic loss (11) is a convex and differentiable function of $h(\mathbf{x})$. For a

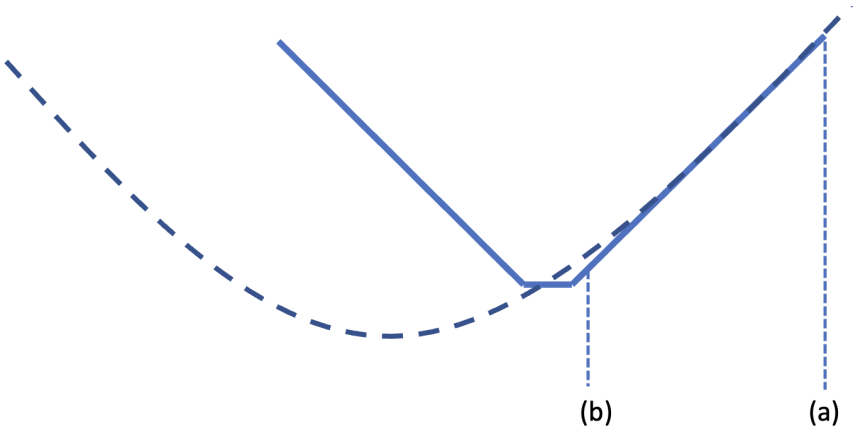

Figure 8: Two examples of loss functions that are convex functions of model parameters. If we can only explore the local shape of a non-differentiable loss function around the current model parameters, it is impossible to decide if we are currently in (a) or (b).

correct classification ($\hat{y} = y$), the logistic loss (11) decreases monotonically with increasing confidence $h(\mathbf{x})$. For a wrong classification ($\hat{y} \neq y$), the logistic loss increases monotonically with increasing confidence $|h(\mathbf{x})|$ in the wrong classification.

Both, the hinge loss (10) as well as the logistic loss (11) are convex functions of the weight vector $\mathbf{w} \in \mathbb{R}^d$ for a linear hypothesis map $h^{(\mathbf{w})}(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$. In contrast to the hinge loss, the logistic loss (11) is also a differentiable function of the $\mathbf{w}$.

The convex and differentiable logistic loss function can be minimized using gradient-based methods (see Section 5). In contrast, we cannot use gradient-based methods to minimize the hinge loss since it is not differentiable (it does not have a gradient everywhere). However, we can apply a generalization of gradient descent (GD) which is known as subgradient descent [42]. Subgradient

descent is obtained from GD by generalizing the concept of a gradient to that of a subgradient [41, 43].

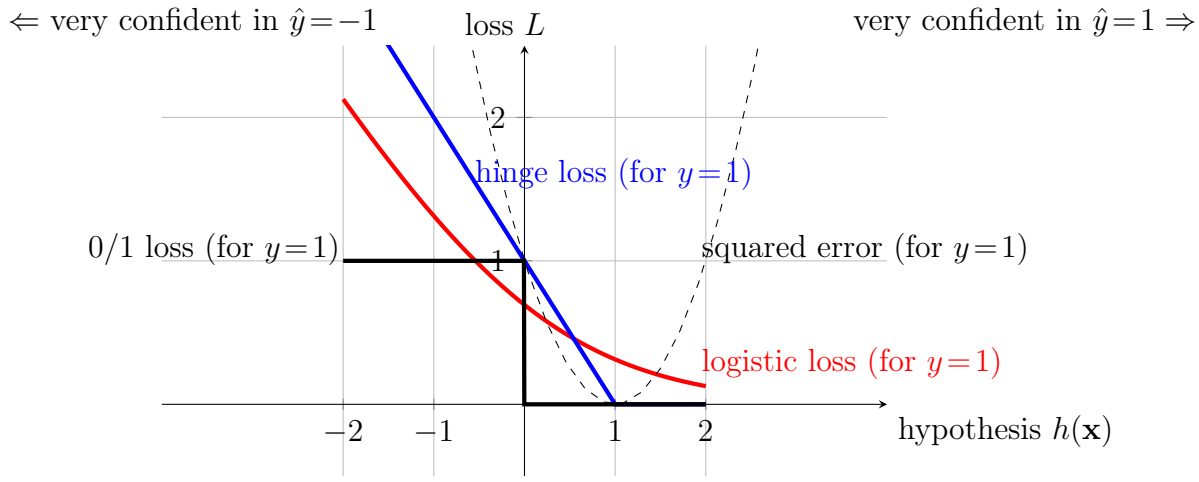

Figure 9: The solid curves depict three widely-used loss functions for measuring the performance of a binary classifier $h^{(\mathbf{w})}(\mathbf{x})$ with label space $\mathcal{Y} = \{-1, 1\}$. A data point with features $\mathbf{x}$ and label $y = 1$ is classified as $\hat{y} = 1$ if $h(\mathbf{x}) \geq 0$ and classified as $\hat{y} = -1$ if $h(\mathbf{x}) < 0$. We can interpret the absolute value $|h(\mathbf{x})|$ as the confidence in the classification $\hat{y}$. The more confident we are in a correct classification ($\hat{y}=y=1$), i.e, the more positive $h(\mathbf{x})$, the smaller the loss. Note that each of the three loss functions for binary classification tends monotonically towards 0 for increasing $h(\mathbf{x})$. The dashed curve depicts the squared error loss (5), which increases for increasing $h(\mathbf{x})$.

2.4 Exercises

Exercise 2.1. Data Imputation (5 points). Consider the dataset available under `https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_wine.html#sklearn.datasets.load_wine`. This dataset consists of data points that represent wine samples. Each data point is characterized by $d = 13$ features $x_1, \ldots, x_d$. Let us assume that the feature value x_1 is missing for several data points. We could then define feature x_1 as the label of a data point and try to predict it from the remaining features. Try to find a coefficients $w_2, \ldots, w_{13}$ such that $\underbrace{y}_{x_1} \approx \sum_{j=2}^{d} w_j x_j$.

Exercise 2.2. Design Aspects (5 points). What are three main design aspects for the hypothesis space (model) of a ML method?

Exercise 2.3. 0/1 loss (5 points). Why is the 0/1 loss rarely used as a criterion for learning (optimizing) a classifier?

Exercise 2.4. Perfect Fit ? (5 points). For a prescribed sample size m, can you craft m data points characterized by feature vectors $\mathbf{x}^{(r)} \in \mathbb{R}^d$ and binary label $y^{(r)} \in \{-1, 1\}$ such that it is impossible to find a linear hypothesis $h^{(\mathbf{w})}(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$ that correctly predicts the label of each data point in $\mathcal{D}$, i.e., there is no choice for the model parameters $\mathbf{w}$ such that

$$\begin{aligned} &h^{(\mathbf{w})}(\mathbf{x}^{(r)}) \geq 0 \text{ for each data point with } y^{(r)} = 1, \text{ and} \\ &h^{(\mathbf{w})}(\mathbf{x}^{(r)}) < 0 \text{ for each data point with } y^{(r)} = -1 \text{ ?} \end{aligned} \tag{12}$$

Exercise 2.5. Formulas for Logistic loss (5 points). Modify the formula (11) for the logistic loss, which is valid only for label space $\mathcal{Y} = \{-1, 1\}$, to

measure the quality of hypothesis map for a binary classification problem with label space $\mathcal{Y} = \{\square, \triangle\}$.

Exercise 2.6. Decision Tree as Linear Model (5 points). A decision tree is an important example for a non-linear model, i.e., a hypothesis space that contains non-linear hypothesis maps. Consider data points characterized by a single features $x \in \mathbb{R}$ and numeric label y. Discuss how a decision tree, with maximum tree depth 4 and decision nodes implementing threshold tests with thresholds 10, 5, and -4 can be obtained by combining a feature map with a linear model $\mathcal{H}^{(d)}$ (d can be larger than 1).

3 A Design Principle for ML

ML methods learn a hypothesis out of a given hypothesis space (or model) $\mathcal{H}$ that incurs minimum loss when applied to arbitrary data points. To make this informal goal precise we need to specify what we mean by an "arbitrary" data point. One approach to define the notion of "arbitrary" data point is to use a probabilistic model.

Section 3.1 introduces a widely used probabilistic model which is referred to as the i.i.d. assumption. NOTE: the i.i.d. assumption is only a conceptual device for studying ML methods. In practice, we rarely know if the observed data points are really i.i.d. (unless we generate them with an ideal random number generator).

The i.i.d. assumption allows to define the expected loss or risk as a performance measure for a given hypothesis. Section 3.2 introduces empirical risk minimization (ERM) as a main design principle for ML. The simple idea of ERM is to approximate the risk of a hypothesis by its average loss on a training set.

Section 3.3 discusses conditions for the training set and hypothesis space such that ERM is likely to succeed.

3.1 The i.i.d. Assumption

Consider a ML application that involves a dataset

$$\mathcal{D} = \left\{ \left(\mathbf{x}^{(1)}, y^{(1)}\right), \ldots, \left(\mathbf{x}^{(m)}, y^{(m)}\right) \right\}.$$

The (arguably) most widely-used probabilistic model in ML is to interpret the data points $\left(\mathbf{x}^{(r)}, y^{(r)}\right)$ as realizations of i.i.d. RVs with a common probability

distribution $p(\mathbf{x}, y)$. This probabilistic model is sometimes referred to as an "i.i.d. assumption" and denoted by

$$\big(\mathbf{x}^{(r)}, y^{(r)}\big) \overset{\text{i.i.d.}}{\sim} p(\mathbf{x}, y). \tag{13}$$

The probabilistic model (13) enables us to measure the quality of a hypothesis $h \in \mathcal{H}$ by the expected loss or risk [44]

$$\bar{L}\big(h\big) := \mathbb{E}\big\{L\left((\mathbf{x}, y), h\right)\big\} = \int_{\mathbf{x},y} L\left((\mathbf{x}, y), h\right) dp(\mathbf{x}, y). \tag{14}$$

The risk (14) of h is the expected value of the loss $L\left((\mathbf{x}, y), h\right)$ incurred when applying the hypothesis h to (the realization of) a random data point with features $\mathbf{x}$ and label y. Note that the computation of the risk (14) requires the joint probability distribution $p(\mathbf{x}, y)$ of the (random) features and label of data points. It seems reasonable to learn a hypothesis h^* that incurs minimum risk,

$$h^* \in \operatorname*{argmin}_{h \in \mathcal{H}} \underbrace{\bar{L}\big(h\big)}_{=\mathbb{E}\left\{L((\mathbf{x},y),h)\right\}}. \tag{15}$$

Any hypothesis that solves (15), i.e., that incurs minimal risk, is referred to as a Bayes estimator [44, Chapter 4].

Thus, once we know the underlying probability distribution, the only challenge for learning the optimal hypothesis is the efficient (numerical) solution of the optimization problem (14). Efficient methods to solve the optimization problem (14) include variational methods and random sampling (Monte Carlo) methods [44, 45].

If we do not know the probability distribution underlying the data points, we cannot compute the risk and, in turn, cannot use (15) as a learning

principle. However, we can approximate (or estimate) the risk by an average over the data points in a dataset. We define the empirical risk of a hypothesis $h \in \mathcal{H}$ incurred on a dataset $\mathcal{D}$ as

$$\widehat{L}\big(h|\mathcal{D}\big) = (1/m) \sum_{r=1}^{m} L\left((\mathbf{x}^{(r)}, y^{(r)}), h\right). \tag{16}$$

The empirical risk of the hypothesis $h \in \mathcal{H}$ is the average loss on the data points in $\mathcal{D}$. Note that the empirical risk depends on three components, the loss function $L(\cdot,\cdot)$, the hypothesis h and the (features and labels of the) data points in the dataset $\mathcal{D}$.

If the data points used to compute the empirical risk (16) are modelled as realizations of i.i.d. RVs whose common probability distribution is $p(\mathbf{x}, y)$, basic results of probability theory tell us that

$$\mathbb{E}\big\{L\left((\mathbf{x}, y), h\right)\big\} \approx \underbrace{(1/m) \sum_{r=1}^{m} L\left((\mathbf{x}^{(r)}, y^{(r)}), h\right)}_{\stackrel{(16)}{=} \widehat{L}\big(h|\mathcal{D}\big)} \text{ for sufficiently large } m. \tag{17}$$

The approximation error in (17) can be quantified by different variants of the law of large numbers [4, 40, 46] or large-deviation bounds [47].

3.2 Empirical Risk Minimization

Many ML methods are motivated by (17) which suggests that a hypothesis with small empirical risk (16) will also result in a small expected loss or risk. The minimum risk is achieved by the Bayes estimator of the label y, given the features $\mathbf{x}$. However, to actually compute the optimal estimator we need the (joint) probability distribution $p(\mathbf{x}, y)$ of features $\mathbf{x}$ and label y. This joint

probability distribution is typically unknown and must be estimated from observed data points.

Many practical ML methods are numerical optimization algorithms to solve ERM

$$\widehat{h} := \operatorname*{argmin}_{h \in \mathcal{H}} \underbrace{(1/m) \sum_{r=1}^{m} L\left(\left(\mathbf{x}^{(r)}, y^{(r)}\right), h\right)}_{\widehat{L}\left(h|\mathcal{D}\right)}. \tag{18}$$

The objective function of ERM is the empirical risk (16) which, in turn, approximates the risk $\bar{L}\big(h\big)$ via the law of large numbers (17). Note that ERM (18) is parametrized by three components: the dataset $\mathcal{D}$, the hypothesis space $\mathcal{H}$ and loss function $L\left(\cdot,\cdot\right)$. The Python library `scikit-learn` provides implementations of (18) for different choices for the hypothesis space $\mathcal{H}$ and loss function $L\left(\cdot,\cdot\right)$ [27].

The objective function of ERM depends on the dataset

$$\mathcal{D} = \left\{\left(\mathbf{x}^{(1)}, y^{(1)}\right), \ldots, \left(\mathbf{x}^{(m)}, y^{(m)}\right)\right\}.$$

In general, $\widehat{L}\big(h|\mathcal{D}\big)$ (viewed as a function of $h \in \mathcal{H}$) depends on each of the feature vectors $\mathbf{x}^{(r)}$ and labels $y^{(r)}$ in $\mathcal{D}$. By the i.i.d. assumption (13), each data point $\left(\mathbf{x}^{(r)}, y^{(r)}\right)$ in $\mathcal{D}$ is a realization of a RV with (unknown) probability distribution $p(\mathbf{x}, y)$. This probability distribution crucially affects the computational aspects and statistical aspects of ML methods that use ERM (18).

In general, we do not have (much) control over the validity of the i.i.d. assumption, let alone the underlying probability distribution $p(\mathbf{x}, y)$. However, we can use principled statistical tests to verify if the i.i.d. assumption is likely to be valid [48]. However, we can make different design choices for the

data points (their features $\mathbf{x}$ and label y), hypothesis space $\mathcal{H}$ and the loss function that define ERM. Different ML methods use different choices for these components of ERM which, in turn, result in different computational aspects and statistical aspects of ERM-based ML methods (see [1, Ch. 3]).

By computational aspects of ERM, we mainly refer to the amount of computation required to find (approximate) solutions to (18). In general, this amount depends on the shape of the empirical risk, viewed as a function of the hypothesis (or its parameters) (see Figure 10). This informal claim is made precise in Section 5 where we study the number of iterations required by GD methods for linear regression. For linear regression, the shape of the objective function in (18) is determined by the eigenvalues of a matrix constructed from the feature vectors in (18).

The statistical aspects of (18) include the probability distribution of the solutions to (18) and their deviation from the Bayes estimator (15). The statistical aspects of (18) also include its robustness against violations of the i.i.d. assumption. Different choices for the model and loss function used in (18) yield different levels of robustness for the resulting ML method [49].

For linear regression, which uses a linear model $h^{(\mathbf{w})}(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$ and squared error loss (5), the generic ERM (18) specializes to

$$\widehat{\mathbf{w}} := \operatorname*{argmin}_{\mathbf{w}\in\mathbb{R}^d} \underbrace{(1/m)\sum_{r=1}^{m}\left(\mathbf{w}^T\mathbf{x}^{(r)} - y^{(r)}\right)^2}_{\widehat{L}\left(h^{(\mathbf{w})}|\mathcal{D}\right)}. \tag{19}$$

3.3 How Much Training is Needed?

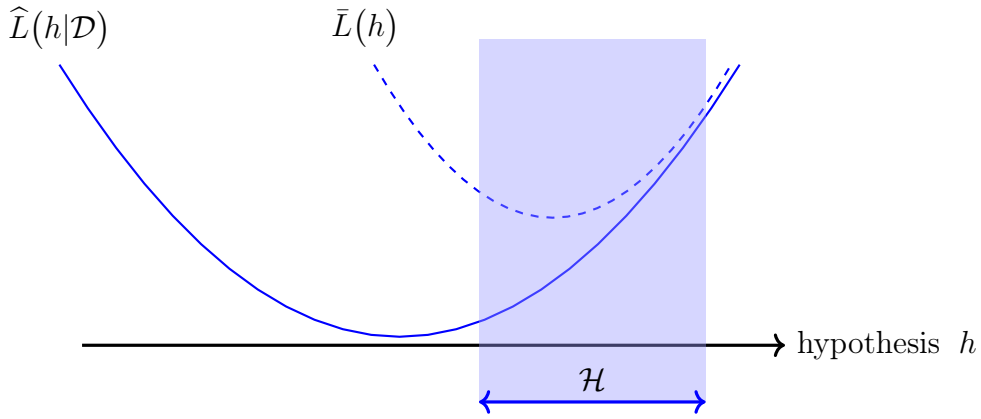

Figure 10: ERM (18) learns a hypothesis, out of a hypothesis space $\mathcal{H}$, by minimizing the empirical risk $\widehat{L}\big(h|\mathcal{D}\big)$ over a training set $\mathcal{D}$. However, the ultimate goal is to learn a hypothesis which accurately predicts the label of any data point and, in turn, has a small risk $\bar{L}\big(h\big)$. Thus, ERM can be expected to deliver a useful hypothesis if the deviation between the empirical risk and the risk is small (uniformly over $h \in \mathcal{H}$).

We have obtained ERM by approximating the risk $\bar{L}\big(h\big)$ with the empirical risk $\widehat{L}\big(h|\mathcal{D}\big)$ (see Figure 10). The risk is the ultimate performance criterion but cannot be computed without knowing the underlying probability distribution (see Section 3.1). Instead, we minimize the empirical risk on a given training set constituted by m data points. Thus, we learn a hypothesis $\hat{h}$ by solving ERM (18). Any solution of (18) can be used as the learnt hypothesis $\hat{h}$.

The solutions of ERM will only deliver a useful hypothesis (with small risk) if two conditions are satisfied:

- the training error $\widehat{L}\big(\hat{h}|\mathcal{D}\big)$ of the learnt hypothesis is small, and,

- the deviation between $\widehat{L}\big(\hat{h}|\mathcal{D}\big)$ and risk $\bar{L}\big(\hat{h}\big)$ is small.

We can test the first condition by inspecting the optimal value of ERM. However, we cannot verify the second condition without further assumptions. Indeed, the deviation between empirical risk and risk depends on the statistical properties of the data points and how many of them are used in the training set for ERM. Moreover, the deviation will also depend on the hypothesis space from which ERM learns the hypothesis $\hat{h}$. We will next analyze this dependency by using a simple probabilistic model that builds on the i.i.d. assumption from Section 3.1.

In what follows, we will interpret any data point, including those in the training set $\mathcal{D}$ of ERM (18), as realizations of i.i.d. RVs with a multivariate normal distribution. In particular, the features $\mathbf{x} \in \mathbb{R}^d$ of data points are realizations of a multivariate normal distribution, $\mathbf{x} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. The label of any data point is related to the features via a linear Gaussian model,

$$y = \overline{\mathbf{w}}^T \mathbf{x} + \varepsilon, \text{ with } \varepsilon \sim \mathcal{N}(0, \sigma^2), \mathbf{x} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}). \tag{20}$$

The linear Gaussian model (20) postulates that the label of a data point is obtained as a noisy linear combination of its features. We form this linear combination using the entries of a true underlying (but unknown to the ML method) weight vector $\overline{\mathbf{w}} \in \mathbb{R}^d$. The additive noise ε in (20) is a realization of a zero-mean Gaussian RV with variance σ^2.

We learn a linear hypothesis $\hat{h}$ via the following instance of ERM (18),

$$\hat{h} \in \operatorname*{argmin}_{h^{(\mathbf{w})} \in \mathcal{H}^{(d')}} (1/m) \sum_{r=1}^{m} \big(y^{(r)} - h^{(\mathbf{w})}\big(\mathbf{x}^{(r)}\big)\big)^2. \tag{21}$$

The linear model $\mathcal{H}^{(d')}$ consists of all linear functions of the first $d' \in \{1, \ldots, d\}$ features $x_1, \ldots, x_{d'}$ of a data point. Thus, we allow the ERM instance (21)

to use only a subset of the d available features $\mathbf{x} = \big(x_1, \ldots, x_d\big)^T$. The feature vectors $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)}$ of data points are modelled as i.i.d. realizations $\mathbf{x}^{(r)} \overset{\text{i.i.d.}}{\sim} \mathcal{N}(\mathbf{0}, \mathbf{I})$.

Note that the objective function in (21) involves the features and labels of data points in $\mathcal{D}$. Since we interpret those as realizations of RVs, also the learnt hypothesis $\hat{h}$ and its risk $\bar{L}\big(\hat{h}\big)$ become realizations of RVs. For a sufficiently large training set, such that $m > d' + 1$, the expectation of the risk $\bar{L}\big(\hat{h}\big)$ is [1, Ch. 6.3],

$$\mathbb{E}\{\bar{L}\big(\hat{h}\big)\} = B^2 + \underbrace{\big(B^2 + \sigma^2\big)d'/(m - d' - 1)}_{\text{variance } V} + \sigma^2. \tag{22}$$

Here, we used the bias term

$$B^2 = \sum_{j=d'+1}^{d} \overline{w}_j^2. \tag{23}$$

According to (22), the expected risk $\mathbb{E}\{\bar{L}\big(\hat{h}\big)\}$ depends on three components:

- the bias term B^2, which depends on the true underlying weight vector $\overline{\mathbf{w}}$ (not under control of ML engineer) and the dimension d' of the linear model $\mathcal{H}^{(d')}$. The hyper-parameter d' can be chosen freely from $\{1, \ldots, d\}$.

- the variance term V which is small if m is significantly larger than d'. As a rule of thumb, the sample size m should be ten times larger than the effective dimension d' of the linear model,

$$m \geq 10 \times d'. \tag{24}$$

- the noise floor σ^2 which is due to the intrinsic noise in the (assumed) relation (20) between features and label of a data point. Under our probabilistic model (20), no ML method can learn a hypothesis with (expected) risk smaller than σ^2.

We can interpret the component $B^2 + V$ in (22) as a measure for the estimation error incurred by using (21) (via its equivalent form (25)) as an estimator for the true underlying $\overline{\mathbf{w}}$ in (20). If we use the linear model $\mathcal{H}^{(d')}$ with $d' = d$, which implies that the bias vanishes $B^2 = 0$, the estimation error becomes $\sigma^2 d'/(m - d' - 1)$ [50, 51].

The third component σ^2 in (22) arises from the noise contained in the true label y (according to our probabilistic model (20)), which we predict by the learnt hypothesis $h^{(\widehat{\mathbf{w}})}(\mathbf{x})$.

Note that (22) is the expected risk of a specific ML method, i.e., using ERM (21) to learn the hypothesis $\hat{h}$. One might wonder if there is a better ML method that could learn a better linear hypothesis $h'(\mathbf{x}) = \big(\mathbf{w}'\big)^T \mathbf{x}$, having smaller expected risk: $\mathbb{E}\{\bar{L}\big(h'\big)\} < \mathbb{E}\{\bar{L}\big(\hat{h}\big)\}$? It turns out that this is not the case and that the hypothesis learnt via (21) achieves minimum expected risk. Indeed, the expected risk of any linear hypothesis learnt from m data points that follow the probabilistic model (20) is lower bounded by (22) [52].

To summarize, using a specific probabilistic model (20) we analyzed the expected risk $\mathbb{E}\big\{\bar{L}\big(\hat{h}\big)\big\}$ of the linear hypothesis $\hat{h}$ learnt via ERM (21). We decomposed $\mathbb{E}\big\{\bar{L}\big(\hat{h}\big)\big\}$ into different components, one of them being proportional to the ratio $d'/(m - d' - 1)$.

To ensure a small $\mathbb{E}\big\{\bar{L}\big(\hat{h}\big)\big\}$, the size m of the training set used by ERM (21) should be significantly larger (say, ten times) than the dimension d' of the

linear model used in (21). While our analysis only applies to linear models, their results provide useful intuition for non-linear models such as ANN. For example, we can (locally) approximate non-linear models by linear models to determine their effective dimension d' [1, Ch. 2].

3.4 Exercises

Exercise 3.1. Why Probability Theory? (5 points) Try to justify the use of probabilistic models in ML. In particular, why is it useful to interpret data points (such as the rows in a spreadsheet) as realizations of RVs.

Exercise 3.2. Trivial Objective in ERM (5 points) Consider some arbitrary dataset which is used to learn a hypothesis out of $\mathcal{H}$ via ERM (18). For which combinations of loss function and hypothesis space does ERM not depend at all on the dataset.

Exercise 3.3. Data Leakage via ERM (5 points) Consider a dataset $\mathcal{D}$ with m data points, each characterized by a single integer-valued feature $x \in \mathbb{Z}$ and a real-valued label $y \in \mathbb{R}$. We try to learn a hypothesis which maps every integer $x \in \mathbb{Z}$ to a predicted label $h(x) \in \mathbb{R}$. The hypothesis space $\mathcal{H}$ is constituted by all maps $h : \mathbb{Z} \to \mathbb{R}$, which could be interpreted as discrete-time signals [53]. The learning is based on minimizing the average squared error loss $\widehat{L}\big(h|\mathcal{D}\big)$ over all hypothesis maps. Is it possible to perfectly recover the dataset just from knowing the values of $\widehat{L}\big(h|\mathcal{D}\big)$ for all $h \in \mathcal{H}$?

Exercise 3.4. ERM for linear regression (5 points) Show that the solution of (21) is the linear hypothesis $h^{(\widehat{\mathbf{w}})}(\mathbf{x}) = \big(\widehat{\mathbf{w}}\big)^{T}\mathbf{x}$ with weight vector $\widehat{\mathbf{w}} \in \mathbb{R}^{d'}$ being a solution to

$$\widehat{\mathbf{w}} \in \operatorname*{argmin}_{\mathbf{w}\in\mathbb{R}^{d'}} \sum_{r=1}^{m} \Big(y^{(r)} - \sum_{j=1}^{d'} w_j x_j^{(r)}\Big)^2. \tag{25}$$

Exercise 3.5. Uniqueness in ERM for linear regression (5 points) Discuss sufficient conditions on the training set such that (25) has a unique solution.

Exercise 3.6. Estimation Error and Expected Risk (5 points) Consider a set of m data points whose features and label are realizations of RVs with a probability distribution defined via (20). We use ERM (25), with $d' = d$ to learn the weights of a linear hypothesis map. We can interpret the solution $\widehat{\mathbf{w}}$ of (25) as an estimator for the true underlying weights $\overline{\mathbf{w}}$ in (20). The performance of this estimator can be measured via the estimation error $\mathbb{E}\big\{ \|\widehat{\mathbf{w}} - \overline{\mathbf{w}}\|_2^2 \big\}$. Discuss the relation between this estimation error and the expected risk $\mathbb{E}\{\bar{L}\big(h^{(\widehat{\mathbf{w}})}\big)\}$.

Exercise 3.7. Fundamental Limit (5 points) Consider data points characterized by a feature vector $\mathbf{x} = \big(x_1, \ldots, x_5\big)^T$ and a numeric label $y \in \mathbb{R}$. Assume that the label of a data point is related to its features via a linear Gaussian model (see (20)),

$$y = \sum_{r=1}^{5} \overline{w}_j x_j + \varepsilon.$$

Here, $\overline{w}_j$ are fixed weights and $\varepsilon \sim \mathcal{N}(0, 5)$ is a realization of a zero-mean Gaussian RV with variance 5. Is there a ML method that can learn a hypothesis with risk no larger than 5 from an training set with $m = 10$ data points (which are i.i.d. realizations of probability distribution $p(\mathbf{x}, y)$)?

Exercise 3.8. Perfect Fit ? (5 points) Consider the dataset $\mathcal{D}$ that consists of $m = 20$ data points, each characterized by a single numeric feature and label given as

$$x^{(r)} = \sin\big(4\pi r/m\big) \text{ , and } y^{(r)} = \cos\big(\pi r/(2m)\big) \text{ for } r = 1, \ldots, m.$$

If we use a sufficiently large hypothesis space $\mathcal{H}$, is it possible to find a

hypothesis map $\hat{h} \in \mathcal{H}$ that perfectly fits the data points in $\mathcal{D}$ in the sense of

$$\hat{h}\big(x^{(r)}\big) = y^{(r)} \text{ for each } r = 1, \ldots, m \text{ ?}$$

4 Regularization

The idea of ERM is to approximate the expected loss or risk of a hypothesis by the empirical risk $\widehat{L}\big(h|\mathcal{D}\big)$ (18) over a training set $\mathcal{D}$. Whenever this approximation fails to be accurate, the solution of ERM might be a hypothesis that performs poorly outside the training set used in (18). Figure 10 illustrates the difference between the risk and the empirical risk as its approximation.

Section 4.1 discusses overfitting as an extreme case where the empirical risk (the training error) of a learnt hypothesis is negligible while its risk is unacceptably large. Section 4.2 introduces regularization techniques that reduce the discrepancy between empirical risk and risk by modifying either the data, model or loss function used by a ERM-based ML method. Section 4.3 explains how to use regularization to couple the training of different local models. This coupling of the training of local models is at the heart of the FL methods discussed in part II.

4.1 Overfitting

Consider a ERM-based ML method that learns a hypothesis from a model $\mathcal{H}$ by minimizing the average loss incurred on a training set (see (18)). The behaviour of such a method can often be (partially) predicted by comparing the size of the training set with the size of the model $\mathcal{H}$. But how to measure the size of the training set and the size of a model ?

An obvious measure for the size of the training set used in ERM (18) is the number m of data points it contains. The number of data points in a dataset is useful measure of its size only if the data points can be (accurately)

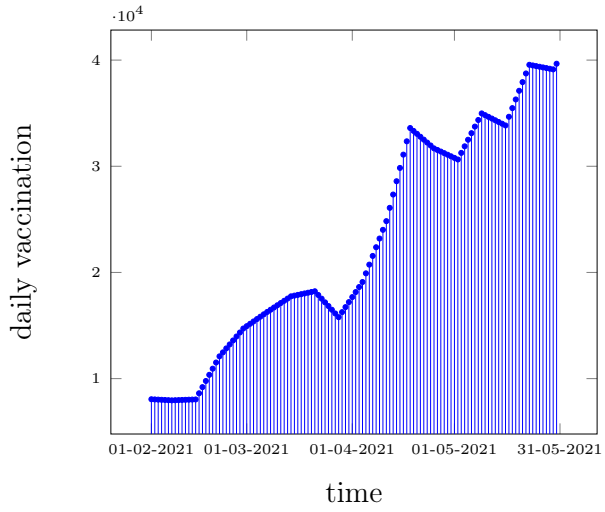

Figure 11: Time series of daily vaccinations which we might use to define data points.

modelled as realizations of i.i.d. RVs. However, for some ML applications, the number of data points in a dataset is not a very useful measure for its size. Consider data points obtained by the values of a time series, such as the daily vaccinations collected during each day (see Figure 11)

If the time series (or its trend) does not change too rapidly, several consecutive data points will be almost identical (or at least strongly correlated). To measure the size of a dataset obtained by sampling a (correlated) time series, we should also take into account the typical duration during which the time series is approximately constant [54]. However, in what follows we tacitly assume that data points can be reasonably well modelled as realizations of i.i.d. RVs such that their number is a useful measure for the size of dataset.

It seems far less obvious how to measure the size of a hypothesis space that contains infinitely many hypothesis maps. Probably the most widely

used concept to measure the size of infinite hypothesis spaces is the Vapnik–Chervonenkis (VC) dimension. A simplified variant of the VC dimension, referred to as the effective dimension $d_{\rm eff}(\mathcal{H})$ has been introduced in [1, Ch. 2]. Loosely speaking, $d_{\rm eff}(\mathcal{H})$ is the maximum number of data points (obtained as i.i.d. realizations of a "well-behaved" probability distribution) such that there is always at least one $h \in \mathcal{H}$ that perfectly fits these data points.

It is instructive to evaluate the effective dimension for the linear model $\mathcal{H}^{(d)}$ that consists of linear hypothesis maps that read in d features of a data point. Here, it can be shown that $d_{\rm eff}\big(\mathcal{H}^{(d)}\big) = d$ (see Exercise 4.5). Another important example is polynomial regression for data points characterized by a single feature x which uses the hypothesis space [1, Sec. 3.2.]

$$\mathcal{H}_d^{(\rm poly)} := \{h(x) = \sum_{j=0}^{d-1} x^j w_j\}.$$

The effective dimension of $\mathcal{H}_d^{(\rm poly)}$ is $d_{\rm eff}\Big(\mathcal{H}_d^{(\rm poly)}\Big) = d$.

The ratio $d_{\rm eff}(\mathcal{H})/m$ between effective dimension of the model $\mathcal{H}$ and the size m of the training set underlying ERM is often a useful indicator for overfitting. For example, if $d_{\rm eff}(\mathcal{H})/m > 1$ then ERM (18) is likely to overfit the training set. Indeed, by the very definition of the effective dimension $d_{\rm eff}(\mathcal{H})$, for (almost) any given training set

$$\big\{\big(\mathbf{x}^{(1)}, y^{(1)}\big), \ldots, \big(\mathbf{x}^{(m)}, y^{(m)}\big)\big\} \text{ of size } m \leq d_{\rm eff}(\mathcal{H}),$$

there is a hypothesis $\hat{h} \in \mathcal{H}$ that perfectly fits the training set, i.e., $y^{(r)} = \hat{h}\big(\mathbf{x}^{(r)}\big)$ for $r = 1, \ldots, m$. To sum up, ERM with a training set of size $m \leq d_{\rm eff}(\mathcal{H})$, will (almost always) learn a hypothesis $\hat{h}$ with vanishing training

error.[1]

Figure 12 depicts the dependence of the training error $\widehat{L}\big(\hat{h}|\mathcal{D}\big)$ and the risk $\bar{L}\big(\hat{h}\big)$ of the hypothesis $\hat{h}$ delivered by ERM using a model with effective dimension $d_{\text{eff}}\left(\mathcal{H}\right)$ and a training set $\mathcal{D}$ of size m.

Consider a fixed size m of the training set and let us vary the effective dimension $d_{\text{eff}}\left(\mathcal{H}\right)$ of the model, e.g., by using a varying number of features or using varying number of hidden layers for a ANN. If we increase $d_{\text{eff}}\left(\mathcal{H}\right)$, the training error typically decreases as we can choose from a larger hypothesis to fit the data points in the training set.

As soon as $d_{\text{eff}}\left(\mathcal{H}\right) > m$, the training error vanishes but at the same time, the risk of the learnt hypothesis increases. This regime will typically lead to overfitting by learning a hypothesis that perfectly predicts the labels of data points in the training set but delivers poor predictions for data points outside the training set.

If we use a too small model with $d_{\text{eff}}\left(\mathcal{H}\right) \ll m$, we might not be able to find any suitable hypothesis $h \in \mathcal{H}$ that accurately predicts the labels of data points from their features. Here, ERM tends to deliver a hypothesis $\hat{h}$ with large training error $\widehat{L}\big(\hat{h}|\mathcal{D}\big)$ and large risk $\bar{L}\big(\hat{h}\big)$. We say that the ERM-based method is underfitting the training set.

Note that ensuring $d_{\text{eff}}\left(\mathcal{H}\right)/m \leq 1$ requires to use at least $m \geq d_{\text{eff}}\left(\mathcal{H}\right)$ data points in the training set for ERM. However, it might be infeasible to collect such an amount of data points for modern ML methods that use high-dimensional models (such as deep ANNs) with $d_{\text{eff}}\left(\mathcal{H}\right)$ being in the order

[1] If we use a loss function that is zero when the prediction $\hat{y} = h(\mathbf{x})$ coincides with the label y of a data point.

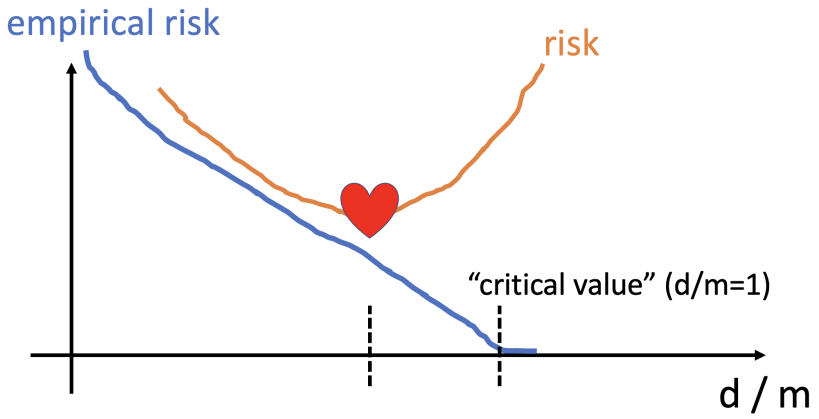

Figure 12: The typical dependence of the training error and the risk of the hypothesis $\hat{h}$ delivered by ERM using a model with effective dimension $d_{\text{eff}}(\mathcal{H})$ and a training set of size m.

of billions [38].

The next section gives a glimpse on regularization techniques for modifying a given model $\mathcal{H}$ and training set $\mathcal{D}^{(\text{train})}$ to decrease the ratio $d_{\text{eff}}(\mathcal{H})/m$.These techniques either reduce $d_{\text{eff}}(\mathcal{H})$ (model pruning) or increase the size m of the training set (data augmentation) without the need to collect new labeled data points.

4.2 Regularization via Data, Model and Loss

Regularization techniques modify ERM (18) to favour a hypothesis that does not perform worse outside the training set. According to Section 4.1, this requires that the ratio $d_{\text{eff}}(\mathcal{H})/m$ between the effective dimension of the model and the size of the training set is sufficiently small (below one).

Obviously, we can decrease this ratio either by using a smaller model ("model pruning") or by using a larger training set.

In general, we can implement regularization via each of the three main ML components, either as

- data augmentation (see Figure 13): we enlarge the training set $\mathcal{D}$ in (18) by adding new data points obtained by perturbing features or labels.

- model pruning: we modify (18) by reducing the optimization domain, which is the hypothesis space $\mathcal{H}$, to a (tiny) subset $\mathcal{H}' \subseteq \mathcal{H}$,

$$\widehat{h} := \operatorname*{argmin}_{h \in \mathcal{H}'} \underbrace{(1/m) \sum_{r=1}^{m} L\left(\left(\mathbf{x}^{(r)}, y^{(r)}\right), h\right)}_{\widehat{L}\left(h|\mathcal{D}\right)}. \tag{26}$$

- loss penalization: modify the loss function in (18) by adding a scaled regularization term,

$$\widehat{h} := \operatorname*{argmin}_{h \in \mathcal{H}} \underbrace{(1/m) \sum_{r=1}^{m} L\left(\left(\mathbf{x}^{(r)}, y^{(r)}\right), h\right)}_{\widehat{L}\left(h|\mathcal{D}\right)} + \lambda \mathcal{R}\big\{h\big\}. \tag{27}$$

  Two well-known instances of (27) are ridge regression and least absolute shrinkage and selection operator (Lasso). These two methods learn a linear hypothesis map $h^{(\mathbf{w})} = \mathbf{w}^T \mathbf{x}$ by minimizing (27) for the squared error loss and regularizer $\mathcal{R}\big\{h^{(\mathbf{w})}\big\} = \|\mathbf{w}\|_2^2$ (ridge regression) or $\mathcal{R}\big\{h^{(\mathbf{w})}\big\} = \|\mathbf{w}\|_1$ (Lasso).

These three approaches to regularization are essentially equivalent. As a case in point, Figure 14 illustrates the equivalence between data augmentation

and loss penalization. Indeed, the empirical risk incurred over an enlarged dataset obtained by data augmentation coincides with the empirical risk on the original dataset using a penalized loss function. This penalty term corresponds to the average loss on the augmented data points.

Equivalence ridge regression and data augmentation. Consider linear regression methods that learn the weight vector of a linear hypothesis map $h^{(\mathbf{w})}(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$ by minimizing the average squared error loss on a training set. We augment $\mathcal{D}$ by adding B realizations of i.i.d. Gaussian RVs with zero mean and covariance matrix $\sigma^2\mathbf{I}$. This results in the augmented dataset $\mathcal{D}'$ which contains each data point of $\mathcal{D}$ along with its B perturbations. As shown in [1, Sec. 7.3.], for $B \to \infty$, the average squared error loss $\widehat{L}\big(h^{(\mathbf{w})}|\mathcal{D}'\big)$ on the augmented dataset $\mathcal{D}'$ tends towards $\widehat{L}\big(h^{(\mathbf{w})}|\mathcal{D}\big) + \sigma^2 \left\|\mathbf{w}\right\|_2^2$. Note that $\widehat{L}\big(h^{(\mathbf{w})}|\mathcal{D}\big) + \sigma^2 \left\|\mathbf{w}\right\|_2^2$ is an instance of loss penalization (see (27)) using the regularizer $\mathcal{R}\big\{h^{(\mathbf{w})}\big\} = \left\|\mathbf{w}\right\|_2^2$. Note that $\widehat{L}\big(h^{(\mathbf{w})}|\mathcal{D}\big) + \sigma^2 \left\|\mathbf{w}\right\|_2^2$ is precisely the average loss used by ridge regression to learn a linear hypothesis map [1, Sec. 3].

Equivalence ridge regression and model pruning. Consider a training set that consists of m data points $\big(\mathbf{x}^{(r)}, y^{(r)}\big)$, for $r = 1, \ldots, m$. Ridge regression learns the weight vector of a linear hypothesis map $h^{(\mathbf{w})}(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$ by solving the unconstrained optimization problem

$$\min_{\mathbf{w}\in\mathbb{R}^d} (1/m) \sum_{r=1}^{m} \big(y^{(r)} - \mathbf{w}^T\mathbf{x}^{(r)}\big)^2 + \lambda \left\|\mathbf{w}\right\|_2^2 \text{ with some } \lambda > 0. \tag{28}$$

Let us denote the unique solution of (28) by $\widehat{\mathbf{w}}^{(\lambda)}$ and its squared Euclidean norm by $\rho^{(\lambda)} := \big\|\widehat{\mathbf{w}}^{(\lambda)}\big\|_2^2$. Trivially, $\widehat{\mathbf{w}}^{(\lambda)}$ is also a solution to the constrained

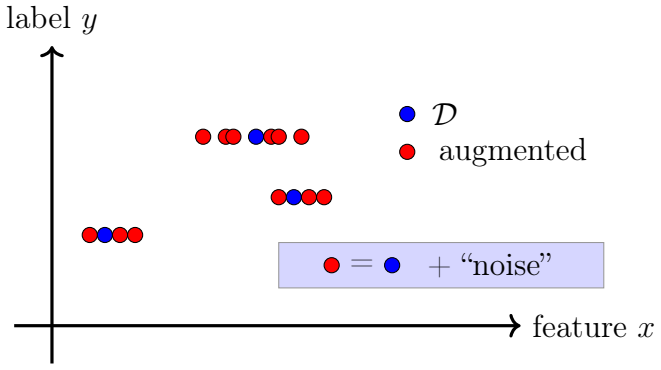

Figure 13: Data augmentation adds new data points to a given dataset $\mathcal{D}$. These new data points are obtained by perturbing features and labels of the data points in $\mathcal{D}$.

optimization problem

$$\min_{\mathbf{w}\in\mathbb{R}^d} (1/m) \sum_{r=1}^{m} \big(y^{(r)} - \mathbf{w}^T \mathbf{x}^{(r)}\big)^2 \quad \text{s.t. } \|\mathbf{w}\|_2^2 \leq \rho^{(\lambda)}. \tag{29}$$

Note that (29) is actually an instance of ERM (18) using the pruned linear model $\mathcal{H}^{(\lambda)} := \{h^{(\mathbf{w})} \in \mathcal{H}^{(d)} : \|\mathbf{w}\|_2^2 \leq \rho^{(\lambda)}\} \subseteq \mathcal{H}^{(d)}$.

4.3 Federated Learning via Regularization

Section 10 will present FL algorithms that use regularization to couple the training of several different local models. The main building block of these algorithms is the regularized empirical risk minimization (RERM) instance

$$\widehat{\mathbf{w}} := \operatorname*{argmin}_{\mathbf{w}\in\mathbb{R}^d} \underbrace{\widehat{L}\big(h^{(\mathbf{w})}|\mathcal{D}\big)}_{\text{training error}} + (\lambda/2) \underbrace{\|\mathbf{w}' - \mathbf{w}\|^2}_{\text{deviation from } \mathbf{w}'}. \tag{30}$$

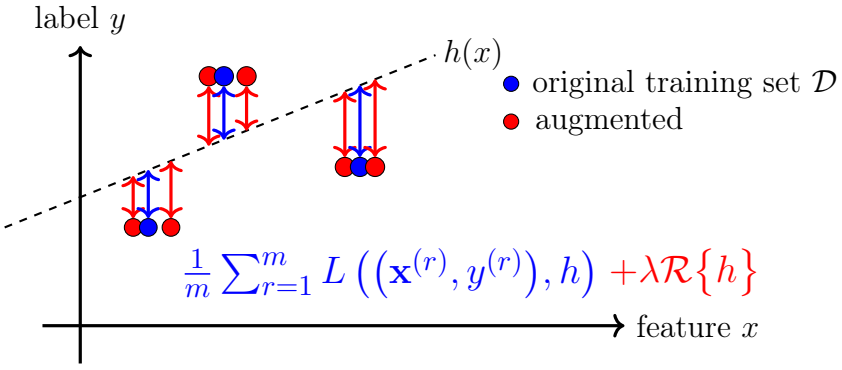

Figure 14: Equivalence of data augmentation and loss penalization techniques for regularization.

The model parameters $\widehat{\mathbf{w}}$ obtained from (30) balances between two (conflicting) goals. On one hand, we want to minimize the empirical risk $\widehat{L}\big(h^{(\mathbf{w})}|\mathcal{D}\big)$. On the other hand, the learnt model parameters $\widehat{\mathbf{w}}$ should not deviate too much from given reference model parameters $\mathbf{w}'$.

The RERM (30) will be used as the elementary computation of FL algorithms in Section 10. In these algorithms, the vector $\mathbf{w}'$ in (30) represents (an aggregation of) model parameters for different (but similar) learning tasks.

Section 7 introduces the empirical graph of a collection of local datasets and corresponding learning tasks that are related by a notion of pairwise similarities. The (weighted) edges of the empirical graph represent (the level of) similarities between local datasets and corresponding learning tasks. Each node of the empirical graph solves RERM (30) to update its current local model parameters. Here, the vector $\mathbf{w}'$ in (30) is an aggregation of the current local model parameters at neighbouring nodes (see Section 10).

Note that solving RERM (30) is to evaluate the proximity operator

$\mathbf{prox}_{f,\lambda}(\mathbf{w}')$ of the empirical risk $f(\mathbf{w}) = \widehat{L}\big(h^{(\mathbf{w})}|\mathcal{D}\big)$, viewed as a function of the model parameters $\mathbf{w}$. This interpretation provides a strong conceptual link between the FL methods developed in Section 8.2 and proximal algorithms [55].

4.4 Exercises

Exercise 4.1. Regularization via data augmentation (5 points). Consider the RERM (30) using some labeled dataset

$$\mathcal{D} = \left\{ \left(\mathbf{x}^{(1)}, y^{(1)}\right), \ldots, \left(\mathbf{x}^{(m)}, y^{(m)}\right) \right\},$$

linear model $\mathcal{H}^{(d)}$ and squared error loss (5). Show that (30) is equivalent to plain ERM (18) for linear regression applied to another dataset $\mathcal{D}'$ which is obtained by adding data points, with carefully chosen features and labels, to the original dataset $\mathcal{D}$.

Exercise 4.2. Proximity operator for linear regression (5 points). Consider the update (30) for the specific choice of squared error loss and linear model. Show that this special case of (30) has always a unique solution.

Exercise 4.3. Uniqueness of Ridge regression (5 points). Provide a rigorous proof for the existence and uniqueness of the solution to the unconstrained optimization problem (28).

Exercise 4.4. Ridge regression as model pruning (5 points). Provide a rigorous proof for the equivalence (in both directions!) between the unconstrained optimization problem (28) and the constrained optimization problem (29).

Exercise 4.5. Effective Dimension of Linear model (10 points). Consider data points with d numeric features $\mathbf{x} = \big(x_1 \ldots, x_d\big)^T \in \mathbb{R}^d$ and a numeric label $y \in \mathbb{R}$. Show that with probability one there is a linear hypothesis map $h(\mathbf{x}) = \sum_{j=1}^{d} w_j x_j$ that perfectly fits $m \leq d$ data points,

$$\big(\mathbf{x}^{(1)}, y^{(1)}\big), \ldots, \big(\mathbf{x}^{(m)}, y^{(m)}\big) \overset{\text{i.i.d.}}{\sim} p(\mathbf{x}, y). \tag{31}$$

In other words, these data points are realizations of i.i.d. RVs with a common probability distribution $p(\mathbf{x}, y)$ that is required to be continuous but arbitrary otherwise. The requirement for $p(\mathbf{x}, y)$ to be continuous is important. Indeed, try to find a probability distribution $p(\mathbf{x}, y)$ that is not continuous and for which it is impossible (with probability one) to find a linear map that perfectly fits the data points in (31).

Exercise 4.6. Convex quadratic function (5 points). Consider a convex quadratic function $f(\mathbf{w}) = \mathbf{w}^T \mathbf{Q} \mathbf{w} + \mathbf{q}^T \mathbf{w} + q$ with some positive semi-definite (psd) matrix $\mathbf{Q}$, vector $\mathbf{q}$ and scalar q. Discuss conditions on $\mathbf{Q}$, $\mathbf{q}$ and q such that $f(\mathbf{w})$ attains its minimum for some vector $\widehat{\mathbf{w}}$. Discuss also conditions that ensure that this minimum is attained (it at all) only at a single vector.

5 Gradient Methods

Section 3.2 formulated ML as ERM, which is a mathematical optimization problem [29, 56]. The stastistical and computational properties of this optimization problem crucially depends on the design choices for the hypothesis space and loss function of a ML method.

Section 3.3 discussed statistical properties of the solutions to ERM (18). In this section, we focus on the computational aspects of ERM, i.e., how to actually compute (approximate) solutions of (18). In particular, we discuss gradient-based methods for solving the ERM arising from combinations of a parametrized model and a differentiable loss function [1, Ch. 3].

We will illustrate the key principles of gradient-based methods using linear regression, i.e., learning a linear hypothesis map to predict the numeric label of a data point from its numeric features. The restriction to linear regression allows for a rather complete analysis of gradient-based methods which also provides intuition for the behaviour of gradient-based methods applied to other combinations of a parametrized hypothesis space (such as ANNs) and loss function (such as logistic loss).

The core of any gradient-based methods, such as gradient descent (GD), is a gradient step (see Section 5.1). A gradient step updates the current choice for model parameters by the scaled negative gradient of the empirical risk (viewed as a function of model parameters). We can interpret the gradient step as the constrained minimization of a local linear approximation of the empirical risk. Another interpretation of the gradient step is that of minimizing a local approximation to the objective function that is a quadratic function (see Exercise 5.10).

Some ML methods require the model parameters to belong to a subset (a constraint set) of the Euclidean space. The resulting ERM is then a constrained optimization problem. Projected GD (see Section 5.2) handles such constraints on the model parameters by projecting the result of a gradient step back into the constraint set.

Section 5.3 discusses the effects of inexact gradient steps due to different types of errors in the gradient computation. Numerical errors might arise from constraints on computational resources such as a finite communication rate between computational units. Approximation errors might arise when using different objective functions for the implementation and for the analysis of GD methods. It might seem strange at first to not use the same objective function for the execution and the analysis of GD methods. However, introducing this freedom might allow to generalize the analysis of GD beyond the (important but limited) class of smooth and strongly convex objective functions.

Note that the objective function of ERM is an approximation to the risk (viewed as a function of the model parameters). In particular, the gradient of the empirical risk is a stochastic approximation (or "estimate") for the gradient of the risk. Stochastic gradient descent (SGD) methods (see Section 5.4) are obtained from GD by taking into account the stochastic nature of the gradient errors.

5.1 Gradient Descent

Let us rewrite ERM (19) for linear regression as

$$\widehat{\mathbf{w}} \in \operatorname*{argmin}_{\mathbf{w} \in \mathbb{R}^d} f(\mathbf{w}) := (1/m) \sum_{r=1}^{m} \big(y^{(r)} - \mathbf{w}^T \mathbf{x}^{(r)}\big)^2. \tag{32}$$

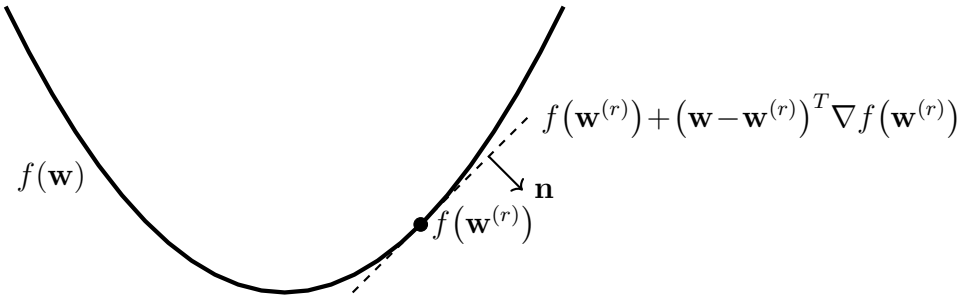

Figure 15: We can approximate a differentiable function $f(\mathbf{w})$ locally around a point $\mathbf{w}^{(r)} \in \mathbb{R}^d$ using a hyperplane. The normal vector $\mathbf{n} = (\nabla f\big(\mathbf{w}^{(r)}\big), -1)^T \in \mathbb{R}^{d+1}$ of this approximating hyperplane is determined by the gradient $\nabla f\big(\mathbf{w}^{(r)}\big)$ [5].

Our goal is to compute (at least approximately) an optimal parameter vector $\widehat{\mathbf{w}}$ that achieves the minimum objective value in (32). Note that the objective function $f(\mathbf{w})$ in (32) is a differentiable function of the model parameters $\mathbf{w}$. In particular, we can compute a gradient $\nabla f(\mathbf{w})$ for any model parameters $\mathbf{w} \in \mathbb{R}^d$ [25],

$$\nabla f(\mathbf{w}) = -(2/m) \sum_{r=1}^{m} \mathbf{x}^{(r)} \big(y^{(r)} - \mathbf{w}^T \mathbf{x}^{(r)}\big). \tag{33}$$

We use the gradient (33) to construct a sequence of model parameters $\mathbf{w}^{(0)}, \mathbf{w}^{(1)}, \ldots$. To this end, we start from an initial choice for the model

parameters $\mathbf{w}^{(0)} \in \mathbb{R}^d$ and repeat the gradient step

$$\begin{aligned}\mathbf{w}^{(r+1)} &:= \mathbf{w}^{(r)} - \alpha_r \nabla f\big(\mathbf{w}^{(r)}\big) \\ &\overset{(33)}{=} \mathbf{w}^{(r)} + (2\alpha_r/m) \sum_{r=1}^{m} \mathbf{x}^{(r)} \big(\underbrace{y^{(r)}}_{\text{label}} - \underbrace{\big(\mathbf{w}^{(r)}\big)^T \mathbf{x}^{(r)}}_{\text{prediction}} \big) \\ &= \mathbf{w}^{(r)} + 2\alpha_r \mathbf{q} - 2\alpha_r \mathbf{Q}\mathbf{w}^{(r)} \\ &\qquad \text{with } \mathbf{q} := (1/m) \sum_{r=1}^{m} \mathbf{x}^{(r)} y^{(r)},\ \mathbf{Q} := (1/m) \sum_{r=1}^{m} \mathbf{x}^{(r)} \big(\mathbf{x}^{(r)}\big)^T. \quad (34)\end{aligned}$$

Here, we used a non-negative learning rate $\alpha_r \geq 0$ that might vary between consecutive gradient steps. The gradient step (34) is constituted by matrix/vector multiplications and additions. Such operations can be implemented rather efficiently via numerical linear algebra software and hardware [25, 57].

5.1.1 GD for Linear Regression

To turn the gradient step (34) into a practical ML algorithm, we need to

- choose a suitable value for the step-size or learning rate α_r in (34). The learning rate must not be too large to avoid overshooting or moving away from the optimum (see Figure 16-(b)). On the other hand, if the learning rate is chosen too small, the gradient step makes too little progress towards the solutions of (32) (see Figure 16-(a)). Note that in practice we can only afford to repeat the gradient step for a finite number of iterations.

- define a stopping criterion to decide when to stop iterating (34).

One approach to choose the learning rate is to start with some initial value (first guess) and monitor the decrease of the objective function. If this decrease

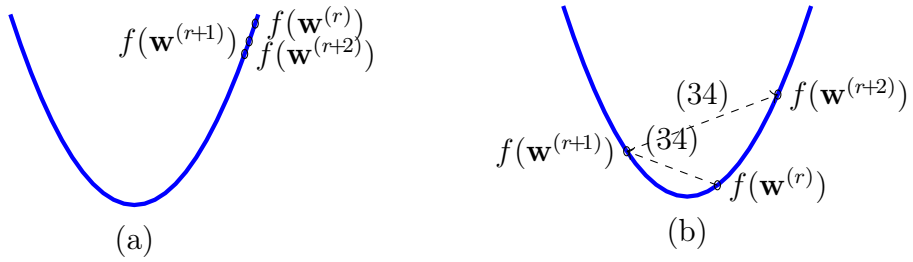

Figure 16: Effect of choosing bad values for the learning rate α in the GD step (34). (a) If the learning rate α in the GD step (34) is chosen too small, the iterations make very little progress towards the optimum or even fail to reach the optimum at all. (b) If the learning rate α is chosen too large the gradient step might "overshoot" such that the iterates $\mathbf{w}^{(r)}$ might diverge from the optimum (it might then happen that $f(\mathbf{w}^{(r+1)}) > f(\mathbf{w}^{(r)})$!).

does not agree with the decrease predicted by the gradient, we decrease the learning rate by a constant factor. After we decrease the learning rate we re-consider the decrease of the objective function. We repeat this procedure until a sufficient decrease of the objective function is achieved [41, Sec 6.1].

Another approach to setting the learning rate in (34) is to use a prescribed sequence α_r of learning rates that vary across successive gradient steps. The sequence α_r, for $r = 1, 2, \ldots$, is also referred to as a learning rate schedule [58]. In particular, convergence of (34) is guaranteed for any learning rate schedule that satisfies [41, Sec. 6.1]

$$\lim_{r\to\infty} \alpha_r = 0, \ \sum_{r=1}^{\infty} \alpha_r = \infty \ , \text{ and } \ \sum_{r=1}^{\infty} \alpha_r^2 < \infty. \tag{35}$$

The first condition (35) requires that the learning rate eventually become sufficiently small to avoid overshooting. The third condition (35) requires

that the learning rate decays sufficiently fast. Note that the first and third condition in (35) could be satisfied by the trivial learning rate schedule $\alpha_r = 0$ which is clearly not useful as the gradient step has no effect. This trivial schedule is prohibited by the middle condition of (35) which requires that the learning rate α_r is large enough such that the gradient step makes sufficient progress.

For the stopping criterion we might use a fixed number $r_{\max}$ of iterations. This hyper-parameter $r_{\max}$ might be dictated by limited resources (such as computational time) for implementing GD or tuned via validation techniques. Another choice for the stopping criterion could be obtained from monitoring the decrease in the objective value $f\big(\mathbf{w}^{(r)}\big)$, i.e., we stop repeating the GD step (34) whenever $\big|f\big(\mathbf{w}^{(r)}\big) - f\big(\mathbf{w}^{(r+1)}\big)\big| \leq \eta$ for a given tolerance η. The tolerance level η is a hyper-parameter of the resulting ML method which could be chosen via validation techniques.

The above techniques for choosing the learning rate α and deciding when to stop the GD steps might be useful in practice. However, they might result in sub-optimal use of computational resources for situations where we have more detailed knowledge about the objective function $f(\mathbf{w})$ in (32) and its gradient (see (34)). Indeed, the choice for the learning rate α and the stopping criterion can be guided by the eigenvalues of the matrix (see (34))

$$\mathbf{Q} = (1/m) \sum_{r=1}^{m} \mathbf{x}^{(r)} \big(\mathbf{x}^{(r)}\big)^T = (1/m) \mathbf{X}^T \mathbf{X}. \tag{36}$$

Here we used the feature matrix $\mathbf{X} := \big(\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)}\big)^T \in \mathbb{R}^{m \times d}$. It can be shown that the matrix $\mathbf{Q}$ is positive semi-definite (psd) which implies that all

of its eigenvalues are real-valued and non-negative,

$$\lambda_1(\mathbf{Q}) \leq \ldots \leq \lambda_d(\mathbf{Q}). \tag{37}$$

It turns out that a GD step (34) reduces the distance $\left\|\mathbf{w}^{(r)} - \widehat{\mathbf{w}}\right\|_2$ to $\widehat{\mathbf{w}}$ (the solution of (32)) by a constant factor [41, Ch. 6],

$$\left\|\mathbf{w}^{(r+1)} - \widehat{\mathbf{w}}\right\|_2 \leq \kappa^{(\alpha_r)}(\mathbf{Q}) \left\|\mathbf{w}^{(r)} - \widehat{\mathbf{w}}\right\|_2. \tag{38}$$

Here, we used the contraction factor

$$\kappa^{(\alpha)}(\mathbf{Q}) := \max\{|1 - \alpha 2\lambda_1|, |1 - \alpha 2\lambda_d|\}, \tag{39}$$

which depends on the learning rate and the matrix $\mathbf{Q}$ (36) (via its eigenvalues (37)). The matrix $\mathbf{Q}$ is determined by the feature vectors of the training set in (32), over which we have little (or no) control. However, we can ensure $\kappa^{(\alpha)}(\mathbf{Q}) < 1$ if we choose the learning rate in (34) such that $\alpha_r < 1/\lambda_d$.

For a contraction factor $\kappa^{(\alpha)}(\mathbf{Q}) < 1$, a sufficient condition on number $r^{(\eta)}$ of gradient steps required to ensure an optimization error $\left\|\mathbf{w}^{(r+1)} - \widehat{\mathbf{w}}\right\|_2 \leq \eta$ is (see (38))

$$r^{(\eta)} \geq \log\left(\left\|\mathbf{w}^{(0)} - \widehat{\mathbf{w}}\right\|_2 / \eta\right) / \log\left(1/\kappa^{(\alpha)}(\mathbf{Q})\right). \tag{40}$$

The relation (38) suggests that smaller values of the contraction factor $\kappa^{(\alpha)}(\mathbf{Q})$ result in faster convergence of gradient steps towards a solution of (32). For a given matrix $\mathbf{Q}$ (36), we can minimize this contraction factor using the learning rate (see Figure 17)

$$\alpha^{(*)} := \frac{1}{\lambda_1 + \lambda_d}. \tag{41}$$

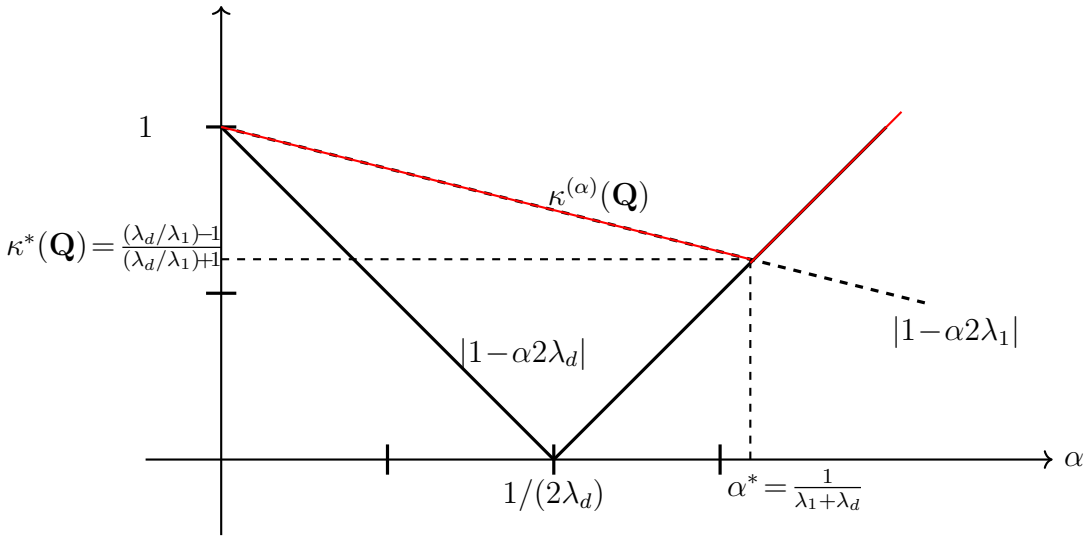

Figure 17: For a given matrix $\mathbf{Q}$ (36), the contraction factor $\kappa^{(\alpha)}(\mathbf{Q})$ varies with the learning rate α. The matrix $\mathbf{Q}$ is determined by the feature vectors of the data points in the ERM (32).

Inserting the optimal learning rate (41) into (38),

$$\left\|\mathbf{w}^{(r+1)} - \widehat{\mathbf{w}}\right\|_2 \leq \underbrace{\frac{(\lambda_d/\lambda_1)-1}{(\lambda_d/\lambda_1)+1}}_{:=\kappa^*(\mathbf{Q})} \left\|\mathbf{w}^{(r)} - \widehat{\mathbf{w}}\right\|_2 . \tag{42}$$

Note that the formula (42) is only valid (meaningful) if the feature vectors $\mathbf{x}^{(r)}$ are such that $\lambda_1 > 0$, i.e., the matrix $\mathbf{Q}$ (36) is invertible. If we cannot guarantee that $\lambda_1 > 0$, we can ensure converge of gradient steps by using a fixed learning rate $\alpha_r = \alpha$ that satisfies [59, Thm. 2.1.14]

$$0 < \alpha < 1/\lambda_d. \tag{43}$$

Note that both, the optimal learning rate (41) and the optimal contraction factor

$$\kappa^{*}(\mathbf{Q}) := \frac{(\lambda_d/\lambda_1) - 1}{(\lambda_d/\lambda_1) + 1} \tag{44}$$

depend on the eigenvalues of the matrix $\mathbf{Q}$ (36). This matrix, in turn, is determined by the feature vectors $\mathbf{x}^{(r)}$ of the data points used in the ERM (19).

In general we have only little (or no) control over the statistical properties of the features $\mathbf{x}^{(r)}$. However, we might be able to control the eigenvalues (37) of the matrix (36) via using a feature map as a pre-processing step. Two important examples for such a feature map are the centering (removing sample mean) and normalization (enforcing unit norm) of features [1, Ch. 5]. These simple transformations ensure that the eigenvalues of the matrix (36) (using the transformed features) are contained between lower and upper bounds L and U,

$$0 < L \leq \lambda_1 \leq \lambda_d \leq U. \tag{45}$$

We can then replace the precise eigenvalues λ_1 and λ_d in (41) and (44) with the bounds L and U, respectively. Moreover, the sufficient condition (43) becomes

$$0 < \alpha < 1/U. \tag{46}$$

According to (42), the ideal case is when all eigenvalues are identical which leads, in turn, to a contraction factor $\kappa^{*}(\mathbf{Q}) = 0$ (a single gradient step arrives at a solution of (32)!). In this regard, we should use a feature map that results in a matrix $\mathbf{Q}$ (obtained from (36) using the transformed feature vectors) being a scaled identity matrix.

Note that the matrix (36) can be interpreted as an approximation for the covariance matrix of a probability distribution from which the features are drawn i.i.d. (see 3.1). Some authors refer to a RV with covariance matrix being a scaled identity as "white" and, in turn, a feature map that aims at making the covariance matrix a scaled identity as a "whitening" of "decorrelation" transformation [60].

5.1.2 GD for Ridge Regression

So far we have discussed the properties of GD for solving the ERM (32) arising in linear regression. Let us now study how the behaviour of GD changes by adding a regularization term to the average loss in (32). Thus, we use GD to learn the parameters of a linear hypothesis by solving ridge regression

$$\min_{\mathbf{w}\in\mathbb{R}^d} f^{(\lambda)}(\mathbf{w}) := (1/m)\sum_{r=1}^{m}\left(y^{(r)} - \mathbf{w}^T\mathbf{x}^{(r)}\right)^2 + \lambda\left\|\mathbf{w}\right\|_2^2. \tag{47}$$

The gradient of the objective function $f^{(\lambda)}(\mathbf{w})$ can be computed via

$$\nabla f^{(\lambda)}(\mathbf{w}) = -(2/m)\sum_{r=1}^{m}\mathbf{x}^{(r)}\left(y^{(r)} - \mathbf{w}^T\mathbf{x}^{(r)}\right) + 2\lambda\mathbf{w}. \tag{48}$$

Note that (48) and (47) reduce to (33) and (32), respectively, for the specific choice $\lambda = 0$. The GD step for minimizing $f^{(\lambda)}$ becomes

$$\begin{aligned}\mathbf{w}^{(r+1)} &= \mathbf{w}^{(r)} + 2\alpha_r\mathbf{q} - 2\alpha_r\mathbf{Q}^{(\lambda)}\mathbf{w}^{(r)} \\ &\text{with } \mathbf{q} := (1/m)\sum_{r=1}^{m}\mathbf{x}^{(r)}y^{(r)},\ \mathbf{Q}^{(\lambda)} := \lambda\mathbf{I} + (1/m)\mathbf{X}\mathbf{X}^T.\end{aligned} \tag{49}$$

Here, we used the feature matrix $\mathbf{X} := \left(\mathbf{x}^{(1)},\ldots,\mathbf{x}^{(m)}\right)^T$.

The GD step (49) for ridge regression has exactly the same form as the GD step (34) for linear regression. The convergence analysis (38) also applies

to (49). In particular, the convergence speed of (49) is crucially influenced by the eigenvalues

$$0 \leq \tilde{\lambda}_1 \leq \ldots \leq \tilde{\lambda}_d \tag{50}$$

of the matrix $\mathbf{Q}^{(\lambda)}$. Basic results from linear algebra tell us that the eigenvalues (50) of the matrix $\mathbf{Q}^{(\lambda)} = \mathbf{Q} + \lambda \mathbf{I}$ and the eigenvalues (37) of the matrix $\mathbf{Q} = (1/m)\mathbf{X}\mathbf{X}^T$ are related via $\tilde{\lambda}_j = \lambda_j + \lambda$ [61, Cor. 4.3.15]. This implies, in turn, that $\kappa^{(*)}\big(\mathbf{Q}^{(\lambda)}\big) \leq \kappa^{(*)}\big(\mathbf{Q}\big)$ and $\lim_{\lambda \to \infty} \kappa^{(*)}\big(\mathbf{Q}^{(\lambda)}\big) = 0$ (see (44)).

Thus, the stronger the regularization (larger λ) in (47), the faster the gradient steps converge to a solution (47). The beneficial computational aspects of using a large λ might come at the cost of detrimental statistical aspects. In particular, ridge regression using a large λ might learn a poor hypothesis $\hat{h}(\mathbf{x}) = \big(\widehat{\mathbf{w}}^{(\lambda)}\big)^T \mathbf{x}$, with $\widehat{\mathbf{w}}^{(\lambda)}$ denoting the solution of (47). Indeed, using excessive regularization (too large λ) might incur a large bias (see Section 3.3 and [1, Ch. 6]).

5.2 Projected Gradient Descent

Section 5.1 discussed methods for learning the parameter vector $\mathbf{w}$ of a linear hypothesis by ERM (32). For some applications it might be useful to constrain the parameter vector $\mathbf{w}$ to belong to a subset $\mathcal{C} \subset \mathbb{R}^d$. Indeed, restricting the parameter vectors to $\mathcal{C}$ is a form of model pruning which can help to avoid overfitting (see Section 4).

Another use case for constraining the model parameter are distributed FL methods. In particular, Section 10.3 discusses a FL method for coupling the training of local model parameters $\mathbf{w}^{(i)}$, for $i \in \{1, \ldots, n\}$ such that they eventually coincide. This coupling can be implemented by requiring the

stacked parameter vector $\mathbf{w} = \big(\mathbf{w}^{(1)}, \ldots, \mathbf{w}^{(n)}\big)^T$ to belong to the subset

$$\mathcal{C} = \Big\{ \big(\mathbf{w}^{(1)}, \ldots, \mathbf{w}^{(n)}\big)^T : \mathbf{w}^{(1)} = \ldots = \mathbf{w}^{(n)} \Big\}.$$

Let us now show how to adapt the basic GD step (34) to solve the ERM

$$f^* = \min_{\mathbf{w} \in \mathcal{C}} f(\mathbf{w}) := (1/m) \sum_{r=1}^{m} \big(y^{(r)} - \mathbf{w}^T \mathbf{x}^{(r)}\big)^2. \tag{51}$$

We assume that the constraint set $\mathcal{C} \subseteq \mathbb{R}^d$ is such that we can efficiently compute the projection

$$P_{\mathcal{C}}\big(\mathbf{w}\big) = \operatorname*{argmin}_{\mathbf{w}' \in \mathcal{C}} \|\mathbf{w} - \mathbf{w}'\|_2 \text{ for any } \mathbf{w} \in \mathbb{R}^d. \tag{52}$$

A suitable modification of the gradient step (34) to solve the constrained ERM (51) is [41]

$$\begin{aligned} \mathbf{w}^{(r+1)} &:= P_{\mathcal{C}}\big(\mathbf{w}^{(r)} - \alpha \nabla f\big(\mathbf{w}^{(r)}\big)\big) \\ &\overset{(33)}{=} P_{\mathcal{C}}\big(\mathbf{w}^{(r)} + (2\alpha/m) \sum_{r=1}^{m} \mathbf{x}^{(r)} \big(y^{(r)} - \big(\mathbf{w}^{(r)}\big)^T \mathbf{x}^{(r)}\big)\big) \\ &= P_{\mathcal{C}}\big(\mathbf{w}^{(r)} + \mathbf{q} - \mathbf{Q}\mathbf{w}^{(r)}\big) \text{ with } \mathbf{q} := (2\alpha/m) \sum_{r=1}^{m} \mathbf{x}^{(r)} y^{(r)}, \\ &\qquad\qquad \mathbf{Q} := (2\alpha/m) \sum_{r=1}^{m} \mathbf{x}^{(r)} \big(\mathbf{x}^{(r)}\big)^T. \end{aligned} \tag{53}$$

The projected GD step (53) amounts to first computing an ordinary GD step $\mathbf{w}^{(r)} \mapsto \mathbf{w}^{(r)} - \alpha \nabla f\big(\mathbf{w}^{(r)}\big)$ and then projecting back to the constraint set $\mathcal{C}$. Note that we re-obtain the basic GD step (34) from the projected GD step (53) for the extreme case where $\mathcal{C} = \mathbb{R}^d$.

The approaches for choosing the learning rate α in - and defining a stopping criterion for - the basic GD step (34) explained in Section 5.1 can also be

used for the projected GD step (53). Moreover, somewhat surprisingly, the convergence speed of the projected GD is characterized by the very same bound (38) as we derived for the unconstrained GD in Section 5.1 [41, Ch. 6]. However, the bound (38) is only telling about the number of GD steps required to achieve a guaranteed level of sub-optimality $\left|f\left(\mathbf{w}^{(r)}\right) - f^{*}\right|$. Each projected GD step (53) might require significantly more computation than the basic GD step, as it requires to compute the projection (52).

5.3 Perturbed Gradient Descent

Consider the gradient step (34) for solving the ERM (32) arising in linear regression. In what follows we assume that the feature vectors $\mathbf{x}^{(r)}$ used in the ERM (32) are such that the matrix $\mathbf{Q}$ (see (36)) is invertible. Thus, the smallest eigenvalue of the matrix $\mathbf{Q}$ is positive: $\lambda_1 > 0$. Moreover, we use a fixed learning rate $\alpha_r = \alpha$ in (34) which is sufficiently small such that the contraction factor (39) satisfies $\kappa^{(\alpha)}\left(\mathbf{Q}\right) < 1$.

In practice, we are often not able to compute all the quantities in (34) exactly. For example, the sum $\sum_{r=1}^{m} \mathbf{x}^{(r)}\left(y^{(r)} - \left(\mathbf{w}^{(r)}\right)^{T}\mathbf{x}^{(r)}\right)$ might require to gather data points from distributed storage locations. These storage locations might become unavailable during the computation (34) due to software or hardware failures (e.g., limited connectivity).

To take into account different sources of imperfections during the compu-

tation of (34), we introduce an error term,

$$\begin{aligned}\mathbf{w}^{(r+1)} &:= \mathbf{w}^{(r)} - \alpha \nabla f\big(\mathbf{w}^{(r)}\big) + \boldsymbol{\varepsilon}^{(r)} \\ &\stackrel{(33)}{=} \mathbf{w}^{(r)} + (2\alpha/m) \sum_{r=1}^{m} \mathbf{x}^{(r)} \big(y^{(r)} - \big(\mathbf{w}^{(r)}\big)^T \mathbf{x}^{(r)}\big) + \boldsymbol{\varepsilon}^{(r)} \\ &= \mathbf{w}^{(r)} + 2\alpha \mathbf{q} - 2\alpha \mathbf{Q} \mathbf{w}^{(r)} + \boldsymbol{\varepsilon}^{(r)} \text{ with } \mathbf{q} := (1/m) \sum_{r=1}^{m} \mathbf{x}^{(r)} y^{(r)}, \\ & \qquad \mathbf{Q} := (1/m) \sum_{r=1}^{m} \mathbf{x}^{(r)} \big(\mathbf{x}^{(r)}\big)^T. \end{aligned} \tag{54}$$

A straight-forward analysis of the perturbed gradient step (54) yields the following bound on the deviation between $\mathbf{w}^{(r)}$ and the solution $\widehat{\mathbf{w}}$ of (32) (see, e.g., [62])

$$\big\|\mathbf{w}^{(r)} - \widehat{\mathbf{w}}\big\|_2 \leq \big(\kappa^{(\alpha)}\left(\mathbf{Q}\right)\big)^r \big\|\mathbf{w}^{(0)} - \widehat{\mathbf{w}}\big\|_2 + \sum_{r'=1}^{r} \big(\kappa^{(\alpha)}\left(\mathbf{Q}\right)\big)^{r'} \Big\|\boldsymbol{\varepsilon}^{(r-r')}\Big\|_2. \tag{55}$$

This bound applies for any number of iterations $r = 1, 2. \ldots$ of the perturbed gradient step (54).

5.4 Stochastic Gradient Descent

Section 5.1 explained how to use gradient steps to construct a sequence of model parameters that are (hopefully) increasingly accurate approximations to a solution of ERM (32) for linear regression. Each gradient step (34) for linear regression requires to compute the sum

$$-(2/m) \sum_{r=1}^{m} \mathbf{x}^{(r)} \Big(y^{(r)} - \big(\mathbf{w}^{(r)}\big)^T \mathbf{x}^{(r)}\Big) \stackrel{(33)}{=} \nabla f\big(\mathbf{w}^{(r)}\big). \tag{56}$$

The sum (56) depends on the data points used in the ERM (32) via their feature vectors $\mathbf{x}^{(r)}$ and labels $y^{(r)}$, for $r = 1, \ldots, m$. Moreover, the sum (56)

also depends on the current choice $\mathbf{w}^{(r)}$ for the model parameters (which we hope to improve by the gradient step). Note that we need to compute a sum of the form (56) for each single gradient step (34).

Computing the sum in (56) might be computationally challenging (or even impossible) for several reasons:

- **Big Data.** Computing the sum (56) might become infeasible for very large training sets, with m in the order of billions. We might then simply not have enough time to compute the sum (56) within the available time budget for completing a gradient step. For example, if we use GD methods to learn optimal steering of a self-driving car, we might need to execute the gradient steps in real-time.

- **Infrastructure Cost.** Second, for decentralized data which is stored in different data centres located all over the planet, the sum (56) would require an excessive amount of network resources.

- **Finite Bandwidths.** Moreover, the finite transmission rate of communication networks limits the speed with which the gradient steps (34) can be executed in real-time (e.g., while steering an autonomous vehicle).

The idea of SGD (and its variants) is to replace the exact gradient $\nabla f(\mathbf{w})$ (56) in (34) by an approximation $g(\mathbf{w}) \approx \nabla f(\mathbf{w})$,

$$\mathbf{w}^{(r+1)} = \mathbf{w}^{(r)} - \alpha_r g\big(\mathbf{w}^{(r)}\big). \tag{57}$$

We hope that the approximation $g\big(\mathbf{w}^{(r)}\big)$ can be computed quicker than the exact gradient (56) while being sufficiently accurate.

The word "stochastic" in the name SGD hints already at the use of probabilistic models in the construction of the gradient approximation $g\big(\mathbf{w}\big) \approx \nabla f\big(\mathbf{w}\big)$. The most basic form of SGD constructs the approximation $g(\mathbf{w})$ by replacing the sum in (56) with a randomly selected component,

$$g(\mathbf{w}) := -2\mathbf{x}^{(\hat{l})}\Big(y^{(\hat{l})} - \mathbf{w}^T\mathbf{x}^{(\hat{l})}\Big). \tag{58}$$

The index $\hat{l}$ is chosen randomly from the set $\{1,\ldots,m\}$. The resulting SGD step (or iteration) updates given model parameters $\mathbf{w}^{(r)}$ to obtain a new (hopefully improved) model parameters

$$\mathbf{w}^{(r+1)} \overset{(57),(58)}{=} \mathbf{w}^{(r)} + \alpha_r 2\mathbf{x}^{(\hat{l}_r)}\Big(y^{(\hat{l}_r)} - \big(\mathbf{w}^{(r)}\big)^T\mathbf{x}^{(\hat{l}_r)}\Big). \tag{59}$$

Every update (59) uses a "fresh" randomly chosen index $\hat{l}_r$. The indices $\hat{l}_1, \hat{l}_2, \ldots$, are realizations of i.i.d. RVs with common probability distribution

$$p(\hat{l}_r = r) = 1/m \text{ , for each } r \in \{1,\ldots,m\}.$$

The update (59) avoids the computation of the sum (56) but instead only requires the computation of a random index $\hat{l}_r$. The resulting computational savings might be significant when the training set consists of a large number of data points.

The saving in computational complexity of SGD comes at the cost of introducing the approximation error (or "gradient noise")

$$\Delta g^{(r)} := \nabla f(\mathbf{w}^{(r)}) - g\big(\mathbf{w}^{(r)}\big). \tag{60}$$

Indeed, we can interpret the basic SGD update (57) as a perturbed gradient step (54) with perturbation given by the scaled gradient noise (60),

$$\mathbf{w}^{(r+1)} := \mathbf{w}^{(r)} - \alpha_r \nabla f\big(\mathbf{w}^{(r)}\big) + \boldsymbol{\varepsilon}^{(r)} \text{ with } \boldsymbol{\varepsilon}^{(r)} = \alpha_r \cdot \Delta g^{(r)}. \tag{61}$$

Note that the perturbation in the SGD update (61) is the product of learning rate α_r and the gradient noise (60). Thus, it seems that we should use a very small learning rate α_r to attenuate the gradient noise. However, we cannot make the learning rate arbitrarily small (the extreme case would be $\alpha_r = 0$) since the SGD updates (57) would then make too little progress towards the solution of (32) (see Figure 16-(a)). One useful choice for the learning rate of SGD is $\alpha_r = 1/r$ [63].

Mini-Batch SGD. Different variants of SGD have been developed with the aim to reduce the gradient noise $\Delta g^{(r)}$ in the SGD step (61). One such variant, which is referred to as mini-batch SGD, uses more than one (randomly chosen) data point for the construction of a gradient approximation,

$$g\big(\mathbf{w}\big) = -(2/B) \sum_{r' \in \mathcal{B}} \mathbf{x}^{(r')} \big(y^{(r')} - \big(\mathbf{w}\big)^T \mathbf{x}^{(r')}\big). \tag{62}$$

For each new iteration of SGD (57), mini-batch SGD generates a new batch

$$\mathcal{B} = \Big\{ \Big(\mathbf{x}^{(\hat{l}_1)}, y^{(\hat{l}_1)}\Big), \ldots, \Big(\mathbf{x}^{(\hat{l}_B)}, y^{(\hat{l}_B)}\Big) \Big\}$$

of B randomly selected data points is generated. The indices $\hat{l}_1, \ldots, \hat{l}_B$ are chosen randomly from $\{1, \ldots, m\}$.

The batch size B is an important hyper-parameter of mini-batch SGD that controls the statistical properties of the gradient approximation (62). Increasing B will reduce the variance of the stochastic approximation (62). Two special cases of mini-batch SGD are basic SGD (which is obtained for $B = 1$) and basic GD (which is obtained for $B = m$).

Online Learning. We can use SGD (57) also in ML applications where data points $\big(\mathbf{x}^{(t)}, y^{(t)}\big)$ are gathered sequentially over time $t = 1, 2, \ldots$. For

example, data points could be the snapshots of the on-board camera installed on a self-driving car. During each time instant t, the camera delivers a new data point $\big(\mathbf{x}^{(t)}, y^{(t)}\big)$ that results in a new component $f^{(t)}\big(\mathbf{w}\big) := \big(y^{(t)} - \mathbf{w}^{T}\mathbf{x}^{(t)}\big)^{2}$ for the objective function in (32).

We can use the SGD step (57) to obtain an online learning method (see [1, Sec. 4.7]). The idea is to use the gradient of the new component $f^{(t)}\big(\mathbf{w}\big) = \big(y^{(t)} - \mathbf{w}^{T}\mathbf{x}^{(t)}\big)^{2}$ as an estimate for the gradient of the objective function (32). This online version of SGD computes, at each time instant t,

$$\mathbf{w}^{(t+1)} := \mathbf{w}^{(t)} + 2\alpha_{t}\mathbf{x}^{(t)}\Big(y^{(t)} - \big(\mathbf{w}^{(t)}\big)^{T}\mathbf{x}^{(t)}\Big). \tag{63}$$

We can use the online gradient step (63) also to learn model parameters from a given training set $\mathcal{D}$ under an i.i.d. assumption. Indeed, we can identify the time index t with the sample index r that enumerates the data points in $\mathcal{D}$. Since the data point $\big(\mathbf{x}^{(t)}, y^{(t)}\big) \in \mathcal{D}$ is a realization of a RV with probability distribution $p(\mathbf{x}, y)$, the gradient $\nabla f^{(t)}\left(\mathbf{w}\right) = -\mathbf{x}^{(t)}\big(y^{(t)} - \mathbf{w}^{T}\mathbf{x}^{(t)}\big)$ is an unbiased estimator for the gradient $\nabla \bar{L}\big(\mathbf{w}\big)$ of the risk $\bar{L}\big(\mathbf{w}\big) = \mathbb{E}\left\{\big(y - \mathbf{w}^{T}\mathbf{x}\big)^{2}\right\}$.

5.5 Exercises

Exercise 5.1. Positive Semidefinite (5 points). Develop a rigorous proof for the fact that the matrix $\mathbf{Q}$ defined in (34) is psd.

Exercise 5.2. Optimality Condition (10 points). Show that any solution $\widehat{\mathbf{w}}$ of ERM (32) must satisfy

$$\mathbf{Q}\widehat{\mathbf{w}} = \mathbf{q} \text{ , with vector } \mathbf{q} \text{ and matrix } \mathbf{Q} \text{ defined in (34).}$$

Exercise 5.3. GD along a Parabola (2 points). Consider GD $w^{(r+1)} = w^{(r)} - \alpha \nabla f\big(w^{(r)}\big)$ with a fixed learning rate α applied to the objective function $f(w) = (3/2)w^2 + 4w - 10^{10}$. Characterize the set of values for the learning rate α such that $w^{(r)}$ converges to an optimum $\overline{w} \in \operatorname{argmin}_{w \in \mathbb{R}} f(w)$.

Exercise 5.4. Matrix Algebra (2 points). Provide a rigorous proof for the second equality in (36).

Exercise 5.5. Learning rate Schedule (5 points). Find an example for a learning rate schedule α_r that satisfies all three conditions in (35).

Exercise 5.6. Projecting onto Ray (5 points). Try to develop a closed-form expression for the projection (52) of a vector $\mathbf{w} \in \mathbb{R}^3$ onto a constraint set being a ray,

$$\mathcal{C} = \big\{c\big(1,1,1\big)^T : c \in \mathbb{R}_+\big\}.$$

Exercise 5.7. Projecting onto Interval (5 points). Try to develop a closed-form expression for the projection (52) of a real number $w \in \mathbb{R}$ (which can be identified with a vector $\mathbf{w}$ of length one) onto the constraint set $\mathcal{C} = [-4, 1]$ which is a closed interval.

Exercise 5.8. Projecting onto Ball (5 points). Try to develop a closed-form expression for the projection (52) of a vector $\mathbf{w} \in \mathbb{R}^d$ onto the constraint set $\mathcal{C} = \{\mathbf{w}' \in \mathbb{R}^d : \|\mathbf{w}'\|_2 \leq 3\}$.

Exercise 5.9. Unbiased (5 points). Due to the randomly chosen index $\hat{l}$, the gradient approximation (58) in SGD becomes (the realization of) a RV. This implies, in turn, that the gradient noise (60) is also (the realization of) a RV. What is the mean vector of this RV ?

Exercise 5.10. Local Quadratic Approximation (5 points). Consider a differentiable function $f(\cdot) : \mathbb{R}^d \rightarrow \mathbb{R}$ and some vector $\mathbf{w}^{(r)} \in \mathbb{R}^d$. The gradient step

$$\mathbf{w}^{(r+1)} := \mathbf{w}^{(r)} - \alpha_r \nabla f\big(\mathbf{w}^{(r)}\big). \tag{64}$$

computes the updated vector $\mathbf{w}^{(r)}$ (that hopefully satisfies $f\big(\mathbf{w}^{(r+1)}\big) < f\big(\mathbf{w}^{(r)}\big)$). Show that the updated vector delivered by the gradient step (64) minimizes the quadratic function

$$\tilde{f}^{(r)}\big(\mathbf{w}\big) := f\big(\mathbf{w}^{(r)}\big) + \mathbf{w}^T \nabla f\big(\mathbf{w}^{(r)}\big) + (1/(2\alpha_r)) \big\|\mathbf{w} - \mathbf{w}^{(r)}\big\|_2^2 .$$

6 Implementing ML

We have now disbussed all the building blocks for the implementation of ML algorithms. Ater choosing a data representation (their features and label), a model (or hypothesis space) and loss function, we can appy gradient methods to solve ERM. From a software engineeering perspective, a ML algorithm is a program that is able to solve ERM (18). Gradient methods

require the underlying computer to be able to compute the gradient $\nabla f(\mathbf{w})$ of the empirical risk $f(\mathbf{w})$. Note that the function value $f(\mathbf{w}) = \widehat{L}\big(h^{(\mathbf{w})}|\mathcal{D}\big)$ is the empirical risk incurred by the hypothesis $h^{(\mathbf{w})}$ obtained from model parameters $\mathbf{w}$.

Our first algorithm is obtained by applying the basic GD mehod from Section 5 to ERM.

Algorithm 1 A Gradient Based ML Algorithm

Input: dataset $\mathcal{D} = \big\{\big(\mathbf{x}^{(1)}, y^{(1)}\big), \ldots, \big(\mathbf{x}^{(m)}, y^{(m)}\big)\big\}$; loss function $L\big((\mathbf{x}, y), h^{(\mathbf{w})}\big)$ learning rate schedule α_r; stoppting criterion

Output: learnt model parameters $\widehat{\mathbf{w}}$

Initialize: $r{:=}0$; $\widehat{\mathbf{w}}$

1: **while** stopping criterion is not satisfied **do**

2: update model parameters by a GD step

$$\mathbf{w}^{(r+1)} := \mathbf{w}^{(r)} - \alpha_r \nabla f\big(\mathbf{w}^{(r)}\big) \text{ with } f(\mathbf{w}) := \widehat{L}\big(h^{(\mathbf{w})}|\mathcal{D}\big) \tag{65}$$

3: $r{:=}r{+}1$

4: **end while**

5:

Part II

Federated Learning

7 Networked Data and Models

Many important application domains, such as healthcare, generate collections of local datasets. Consider different healthcare providers including hospitals, blood laboratories or x-ray institutes. Each of these individual providers maintain their own local patient databases. These local datasets are related via statistical similarities: the frequency of flu patients at nearby hospitals might be correlated.

Section 7.1 introduces the empirical graph as a useful representation of collections of local datasets along with their similarities. The empirical graph might be designed manually by using domain expertise that provides measures for the similarity between local datasets. Another source for the empirical graph might be basic statistics of the local datasets such as the fraction of shared data points. For some application domains, it might be useful to learn the empirical graph using probabilistic models for the local datasets (see Section 8.5).

Section 7.2 augments the empirical graph by assigning a separate local hypothesis space (or local model) to each node. We could train each local model separately using the corresponding local dataset as training set in ERM (see Section 3.2). However, the local dataset might be too small to train a (high-dimensional) model such as a large ANN or a linear model with many features (see Section 3.3).

We can use the network structure of the empirical graph to pool local datasets to obtain a sufficiently large training set to train each local model via ERM. The pooling of local datasets relies on a clustering assumption that we discuss in Section 7.3. Roughly speaking, we assume that local datasets form

clusters of homogeneous data: The local datasets in the same cluster can be considered (approximately) as obtained by i.i.d. draws from a cluster-wise probability distribution. It seems then natural to pool local datasets in the same cluster to obtain a larger training set for the corresponding local models.

Since we typically do not know the exact cluster structure of local datasets, we cannot implement the above pooling approach directly. Instead, we use the network structure of the empirical graph to couple the training of local models such that their model parameters are (approximately) constant over clusters. Section 8 puts forward regularization techniques to enforce similar local model parameters at well-connected nodes. Forcing the local model parameters at nodes in the same cluster to be identical is equivalent to the pooling of the local datasets at these nodes.

7.1 The Empirical Graph

An empirical graph is an undirected weighted graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ whose nodes $\mathcal{V} := \{1, \ldots, n\}$ represent local datasets $\mathcal{D}^{(i)}$, for $i \in \mathcal{V}$. Each node $i \in \mathcal{V}$ of the empirical graph $\mathcal{G}$ carries a separate local dataset $\mathcal{D}^{(i)}$. To build intuition, think of a local dataset $\mathcal{D}^{(i)}$ as a labelled dataset

$$\mathcal{D}^{(i)} := \left\{ \left(\mathbf{x}^{(i,1)}, y^{(i,1)}\right), \ldots, \left(\mathbf{x}^{(i,m_i)}, y^{(i,m_i)}\right) \right\}. \tag{66}$$

Here, $\mathbf{x}^{(i,r)}$ and $y^{(i,r)}$ denote, respectively, the features and the label of the rth data point in the local dataset $\mathcal{D}^{(i)}$. Note that the size m_i of the local dataset might vary between different nodes $i \in \mathcal{V}$. Figure 18 depicts an example for an empirical graph.

An undirected edge $\{i, i'\} \in \mathcal{E}$ between two different nodes $i, i' \in \mathcal{V}$ indicates

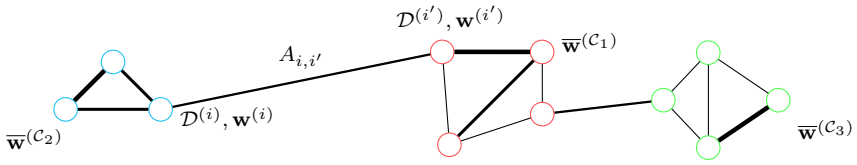

Figure 18: Example of an empirical graph whose nodes $i \in \mathcal{V}$ carry local datasets $\mathcal{D}^{(i)}$ and local models that are parametrized by local model parameters $\mathbf{w}^{(i)}$. The nodes $\mathcal{V}$ partitioned into three disjoint clusters $\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3$, with cluster-wise optimal model parameters $\overline{\mathbf{w}}^{(\mathcal{C})}$ (see Section 7.3).

that the local datasets $\mathcal{D}^{(i)}$ and $\mathcal{D}^{(i')}$ have similar statistical properties. We quantify the level of similarity by a positive edge weight $A_{i,i'} > 0$. The neighbourhood of a node $i \in \mathcal{V}$ is $\mathcal{N}^{(i)} := \{i' \in \mathcal{V} : \{i, i'\} \in \mathcal{E}\}$, and its size is referred to as node degree $d^{(i)} := |\mathcal{N}^{(i)}|$. In what follows, we only consider empirical graphs that contain edges between different nodes and, in turn, have no loops [64]. Thus, each edge $e \in \mathcal{E}$ of an empirical graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ can be identified with a set $\{i, i'\} \subseteq \mathcal{V}$ that consists of two different nodes $i, i' \in \mathcal{V}$.

Our (rather) informal notion of similarity could be made precise via a probabilistic model that interprets the local dataset $\mathcal{D}^{(i)}$ as realizations of RVs with some parametrized probability distribution $p^{(i)}\big(\mathcal{D}^{(i)}; \mathbf{w}^{(i)}\big)$. The similarity between local datasets $\mathcal{D}^{(i)}$ and $\mathcal{D}^{(i')}$ could then be defined in terms of distance $\big\|\mathbf{w}^{(i)} - \mathbf{w}^{(i')}\big\|_2$ between the parameters of the probability distributions.

The undirected edges $\{i, i'\} \in \mathcal{E}$ of an empirical graph encode a symmetric notion of similarity between local datasets. If the local dataset $\mathcal{D}^{(i)}$ at node i

is similar to the local dataset $\mathcal{D}^{(i')}$ at node i', then also the local dataset $\mathcal{D}^{(i')}$ is similar to the local dataset $\mathcal{D}^{(i)}$.

The empirical graph of networked data is a design choice which is guided by computational aspects and statistical aspects of the resulting ML method. For example, using an empirical graph with a small number of edges ("sparse graphs") typically results in a smaller computational complexity. Indeed, the amount of computation required by the FL methods developed in Section 10 is proportional to the number of edges in the empirical graph. On the other hand, the empirical graph should contain sufficient number of edges between nodes that carry statistically similar local datasets. This allows regularization techniques to adaptively pool local datasets into clusters of (approximately) homogeneous data (see Section 7.3).

7.2 Networked Models

Consider data with empirical graph $\mathcal{G}$ whose nodes $i \in \mathcal{V}$ carry local datasets $\mathcal{D}^{(i)}$. We will study FL applications where each node $i \in \mathcal{V}$ also carries a local model $\mathcal{H}^{(i)}$. For each node $i \in \mathcal{V}$, we wish to learn a useful hypothesis $\widehat{h}^{(i)}$ from a local hypothesis space $\mathcal{H}^{(i)}$. The learnt hypothesis should incur a small average loss over a local dataset $\mathcal{D}^{(i)}$.

The collection of local models $\mathcal{H}^{(i)}$, for each node $i \in \mathcal{V}$, constitutes a networked model $\mathcal{H}^{(\mathcal{G})}$ over an empirical graph $\mathcal{G}$. Formally, a networked model is a map

$$\mathcal{H}^{(\mathcal{G})} : i \mapsto \mathcal{H}^{(i)} \text{ for each node } i \in \mathcal{V}. \tag{67}$$

It is important to note that different nodes might carry different local models $\mathcal{H}^{(i)}$. For example, $\mathcal{H}^{(i)}$ might be a linear model $\mathcal{H}^{(d)}$, while $\mathcal{H}^{(i')}$ might be a

decision tree for some other node $i' \neq i$.

We measure the usefulness of a particular hypothesis $h \in \mathcal{H}^{(i)}$ using a local loss function

$$L_i(\cdot) : \mathcal{H}^{(i)} \to \mathbb{R}_+ : h \mapsto L_i(h). \tag{68}$$

If node $i \in \mathcal{V}$ carries a local dataset of the form (66), we might define a local loss function via the average loss

$$L_i(h) := (1/m_i) \sum_{r=1}^{m_i} L\left(\left(\mathbf{x}^{(i,r)}, y^{(i,r)}\right), h\right). \tag{69}$$

The FL methods developed in Section 8 and Section 10 can be applied also to loss functions that are not of the form (69). Examples for such more general loss functions arise in unsupervised ML applications such as clustering or dimensionality reduction [1, Ch. 8,9].

As with the loss function used in basic ML methods (see Section 2), the (design) choice for the local loss functions must take into account the computational aspects and statistical aspects of the resulting FL methods. There are often design goal conflicts arising from computational aspects (e.g., requiring a small number of gradient steps to find an approximate minimizer) and statistical aspects (e.g., robustness against outliers in the training set).

Our focus will be on parametrized networked models where each $\mathcal{H}^{(i)}$ is parametrized by a common finite-dimensional Euclidean space $\mathbb{R}^d$. This setting covers some widely-used ML models such as (regularized) generalized linear models or linear time series models [65–69]. For a parametrized networked model, the local hypothesis $h^{(i)}$ at any node $i \in \mathcal{V}$ is fully determined by local model parameters $\mathbf{w}^{(i)} \in \mathbb{R}^d$. We find it convenient to collect local

parameters $\mathbf{w}^{(i)}$ as the values of a map $\mathbf{w} : i \mapsto \mathbf{w}^{(i)}$. These maps constitute the networked model parameter space

$$\mathcal{W} := \Big\{ \mathbf{w} : i \mapsto \mathbf{w}^{(i)} \text{ such that } h^{(\mathbf{w}^{(i)})} \in \mathcal{H}^{(i)} \Big\}. \tag{70}$$

The usefulness of a specific choice for the local model parameter $\mathbf{w}^{(i)}$ is measured by the local loss function $L_i\left(\mathbf{w}^{(i)}\right) := L_i\left(h^{(\mathbf{w}^{(i)})}\right)$. The local loss functions $L_i\left(\mathbf{w}^{(i)}\right)$ are an important design choice within FL systems. This design choice is related to the design choice for the local datasets and (the parametrization of) local models $\mathcal{H}^{(i)}$ (see (69)).

As for other ML components (see Section 2), there are statistical aspects and computational aspects guiding the choice for the local loss functions. Statistically, we want the loss function to favour local model parameters that result in a robust and accurate hypothesis for each node. Computationally, we want a local loss function that can be minimized efficiently using the available computational resources (see Section 5 and Section 10).

The FL methods in Section 10.3 and 10.1 require local loss functions that are differentiable and allow for efficient computation of their gradient at arbitrary points. Indeed, these FL methods use GD steps as their elementary computation.

The FL method in Section 10.2 requires that the local loss function $L_i\left(\mathbf{w}^{(i)}\right)$ allows for efficient solution of the regularized problem (see Section 4.3)

$$\min_{\mathbf{w}^{(i)} \in \mathbb{R}^d} L_i\left(\mathbf{w}^{(i)}\right) + \lambda \|\mathbf{w}^{(i)} - \mathbf{w}'\|_2^2 \text{ with some } \mathbf{w}' \in \mathbb{R}^d. \tag{71}$$

Note that the basic computation (71) used by the FL method in Section 10.2 is itself an optimization problem which might be solved by the gradient-based

methods in Section 5.

7.3 Clustering Assumption

Many applications generate local datasets which do not carry sufficient statistical power to guide learning of model parameters $\mathbf{w}^{(i)}$ (see Section 3.3). As a case in point, consider a local dataset $\mathcal{D}^{(i)}$ of the form (66), with feature vectors $\mathbf{x}^{(r)} \in \mathbb{R}^d$ with $m_i \ll d$. We would like to learn the parameter vector $\mathbf{w}^{(i)}$ of a linear hypothesis $h(\mathbf{x}) = \mathbf{x}^T \mathbf{w}^{(i)}$.

Linear regression methods [1, Sec. 3.1] learn the parameter vector by minimizing the average squared error loss

$$\min_{\mathbf{w}^{(i)}} L_i\left(\mathbf{w}^{(i)}\right) = (1/m_i) \sum_{r=1}^{m_i} \left(y^{(r)} - \left(\mathbf{w}^{(i)}\right)^T \mathbf{x}^{(r)}\right)^2.$$

However, for $m_i \ll d$ (the high-dimensional regime) the above minimum is not unique and might also provide a poor hypothesis. In particular, the model parameters $\widehat{\mathbf{w}}$ that achieve the minimum average loss over the training set $\mathcal{D}^{(i)}$ (and therefore are delivered by ERM-based methods) might incur unacceptably large prediction errors for data points outside the training set $\mathcal{D}^{(i)}$ [1, Ch. 6].

Training linear models in the high-dimensional regime requires regularization techniques such as those used by ridge regression or Lasso [39]. These methods implement regularization by adding a penalty term to the average loss of a hypothesis incurred over a training set. This penalty term is an approximation or estimate for the increase in average loss when the hypothesis is applied to data points outside the training set.

Section 8 proposes different measures for the variation of local hypothesis

maps $h^{(i)}$ over edges in the empirical graph. The resulting measures are then used as a penalty term to regularize the training of local models. We will see that adding this penalty term results in an adaptive pooling of local datasets that form a cluster with a common choice for an optimal hypothesis (or model parameters).

Pooling local datasets to form clusters only makes sense if they have similar statistical properties that can be captured by a common hypothesis. In particular, we require the local loss functions at nodes $i \in \mathcal{C}$ in the same cluster $\mathcal{C}$ to have approximately identical minimizers.

Assumption 1 (Clustering (informal))**.** *Consider local datasets $\mathcal{D}^{(i)}$ represented by the nodes of an empirical graph $\mathcal{G}$. There is a partition*

$$\mathcal{P} = \{\mathcal{C}_1, \ldots, \mathcal{C}_{|\mathcal{P}|}\} \text{ with } \mathcal{C}_c \cap \mathcal{C}_{c'} = \emptyset \text{ and } \mathcal{V} = \mathcal{C}_1 \cup \ldots \cup \mathcal{C}_{|\mathcal{P}|}, \tag{72}$$

of the nodes $\mathcal{V}$ into disjoint clusters $\mathcal{C}_c \in \mathcal{P}$. The local datasets at the nodes in the same cluster conform to an i.i.d. assumption (see Section 3.1) with a cluster-specific probability distribution $p^{(c)}(\mathbf{x}, y)$ for $c = 1, \ldots, k$.

Assume we would know which nodes belong to the same cluster $\mathcal{C}$. It would then be sensible to pool the local datasets, or equivalently add the corresponding local loss functions (69), and learn model parameters by minimizing the resulting cluster-wise loss function,

$$\overline{\mathbf{w}}^{(\mathcal{C})} = \operatorname*{argmin}_{\mathbf{v} \in \mathbb{R}^d} f^{(\mathcal{C})}(\mathbf{v}) \text{ with } f^{(\mathcal{C})}(\mathbf{v}) := \sum_{i \in \mathcal{C}} \underbrace{L_i(\mathbf{v})}_{:=(1/m_i)\sum_{(\mathbf{x},y)\in\mathcal{D}^{(i)}} L\left((\mathbf{x},y),h^{(\mathbf{v})}\right)}. \tag{73}$$

Note that (73) can only be implemented if we know the cluster $\mathcal{C}$ of local datasets for which the i.i.d. assumption is (approximately) valid. However, it

is often not known a-priori if two local datasets $\mathcal{D}^{(i)}, \mathcal{D}^{(i')}$ belong to the same cluster $\mathcal{C}$. Rather, we might only know if they are statistically similar which is indicated by an edge $\{i, i'\} \in \mathcal{E}$ (with weight $A_{i,i'}$) in the empirical graph.

To obtain a practicable approximation to the cluster-wise learning (73), we use regularization to couple the minimizing of individual local loss functions. This regularization will be implemented via measures for the variation of local model parameters across the weighted edges of the empirical graph. Section 8 defines generalized total variation (GTV) as a measure for the variation of local model parameters. Using GTV as a regularizer yields a flexible design principle for FL methods. The resulting FL methods use the weighted edges in the empirical graph to couple the training of local models by enforcing small variations of local model parameters across edges.

Consider local datasets that form clusters according to Assumption 1. Ideally, we would pool local datasets in the same cluster and learn separate model parameters for each cluster via (73). However, we do not know the clusters in general but only the empirical graph which allows to compute the GTV of local model parameters. If nodes in the same cluster are connected by sufficiently many edges and only few edges connect nodes in different clusters, regularization via GTV approximates (73) [68, 70–72].

We emphasize that Assumption 1 is only a modelling assumption which might not be valid for the FL application at hand. Moreover, even when Assumption 1 is valid, the empirical graph, either provided by a domain expert or graph learning methods (see Section 8.5), does not reflect well the actual cluster structure (72). It is therefore important to always validate the local hypothesis maps (or local model parameters) obtained by FL methods (see

Section 10) that require an empirical graph that conforms with a clustering assumption (see Assumption 1).

7.4 Exercises

Exercise 7.1. Get Weather Data (5 points). Construct an empirical graph $\mathcal{G}^{(\mathrm{FMI})}$ for the data collected at Finnish Meteorological Institute (FMI) weather stations. The empirical graph should contain at least $n = 100$ nodes which represent different FMI weather stations. The local dataset $\mathcal{D}^{(i)}$ at node i is constituted by $m = 5$ data points

$$\left(x^{(i,1)}, y^{(i,1)}\right), \ldots, \left(x^{(i,m)}, y^{(i,m)}\right)$$

that represent the daily minimum and maximum temperatures during five consecutive days starting with 31 January 2014. Use the latitude and longitude of weather stations to determine pairwise geodesic distances between them. Place an edge between two nodes $i, i' \in \mathcal{V}$ if either weather station i is among the four nearest neighbours of weather station i' or vice versa. The empirical graph should be stored as a `networkx.Graph` containing the local datasets as node attributes (in the form of `numpy` arrays).

Exercise 7.2. Testing Clustering Assumption (5 points). Consider the empirical graph $\mathcal{G}^{(\mathrm{FMI})}$ constructed in Exercise 7.1. Apply two different clustering methods to partition the nodes in $\mathcal{G}^{(\mathrm{FMI})}$. First, partition the nodes $\mathcal{V}$ using k-means by using the stacked minimum and maximum daytime temperatures as a feature vector $\mathbf{x}^{(i)}$ (of length $2 \cdot 5 = 10$) for each node $i \in \mathcal{V}$. Compare the resulting partition with the partition obtained from spectral graph clustering by using uniform edge weights $A_{i,i'} = 1$ for each edge $\{i, i'\} \in \mathcal{E}$. Discuss your choices for the hyper-parameters of each clustering method.

Exercise 7.3. A Notion of Similarity (10 points). Consider nodes $\mathcal{V} = \{1, \ldots, n\}$ that carry local datasets $\mathcal{D}^{(i)}$ and identical linear local models $\mathcal{H}^{(i)} = \mathcal{H}^{(d)}$, for $i \in \mathcal{V}$, each using the same number d of features. We then train each local model by ERM on the corresponding local dataset,

$$\widehat{h}^{(i)} := \operatorname*{argmin}_{h \in \mathcal{H}^{(i)}} \widehat{L}\big(h|\mathcal{D}^{(i)}\big).$$

A notion of similarity between two different nodes $i, i' \in \mathcal{V}$ might be

$$\text{“node } i \in \mathcal{V} \text{ is similar to } i' \in \mathcal{V} \setminus \{i\}\text{” if } \widehat{L}\big(\widehat{h}^{(i)}|\mathcal{D}^{(i')}\big) \leq 2\widehat{L}\big(\widehat{h}^{(i)}|\mathcal{D}^{(i)}\big). \quad (74)$$

Thus, node i is considered similar to node i' if the model trained on local dataset $\mathcal{D}^{(i)}$ incurs an average loss $\widehat{L}\big(\widehat{h}^{(i)}|\mathcal{D}^{(i')}\big)$ on the local dataset $\mathcal{D}^{(i')}$ that does not exceed twice the (optimized!) training error $\widehat{L}\big(\widehat{h}^{(i)}|\mathcal{D}^{(i)}\big)$. Is this notion of similarity symmetric, i.e., does “node i is similar to node i'” always imply that “node i' is similar to node i” ? Or can you provide a counter-example which proves that the similarity notion is not symmetric and would therefore require the empirical graph to have directed edges?

8 A Design Principle for FL

Section 7.2 introduced the empirical graph whose nodes carry local datasets and corresponding local models. This section develops a flexible design principle for FL agorithms to train local models at nodes of an empirical graph by combining the information carried by local datasets with the information in their network structure. The network structure of local datasets is encoded in the weighted edges of the empirical graph.

Our design principle couples the training of local models by enforcing approximately identical local model parameters at nodes that carry similar local datasets. Section 8.1 defines the GTV as a measure for the discrepancy between local model parameters at nodes that carry (statistically) similar local datasets.

Section 8.2 introduces GTV minimization (GTVMin) to learn local model parameters that optimally balance between their empirical risk incurred on local datasets and their GTV. Section 8.3 discusses several useful interpretations of GTVMin. Section 8.4 extends GTV and GTVMin from parametric to heterogeneous non-parametric local models such as decision trees. The basic idea is to measure the variation of trained local models via the discrepancy between their predictions on a common test set.

8.1 Generalized Total Variation

Consider a parametrized networked model with local model parameters $\mathbf{w}^{(i)} \in \mathbb{R}^d$, for each node $i \in \mathcal{V}$, that constitute networked model parameters $\mathbf{w} \in \mathcal{W}$. We measure the variation of local model parameters over the edge $e = \{i, i'\} \in$

$\mathcal{E}$ by the difference

$$\mathbf{u}^{(e)} := \mathbf{w}^{(e_+)} - \mathbf{w}^{(e_-)} \text{ with } e_+ := \min\{i, i'\}, e_- := \max\{i, i'\}.$$

A quantitative measure for the overall variation of the networked model parameters $\mathbf{w}$ is the GTV

$$\|\mathbf{w}\|_{\rm GTV} := \sum_{e \in \mathcal{E}} A_{i,i'} \phi\big(\mathbf{u}^{(e)}\big) \text{ with penalty function } \phi(\cdot) : \mathbb{R}^d \to \mathbb{R}. \quad (75)$$

Note that GTV (75) represents an entire ensemble of variation measures. This ensemble is parametrized by the penalty function $\phi(\cdot)$ which we only require to be convex but arbitrary otherwise. Two popular choices are $\phi(\mathbf{v}) := \|\mathbf{v}\|_2$, which is used by network Lasso [73], and $\phi(\mathbf{v}) := (1/2)\,\|\mathbf{v}\|_2^2$ which is used by "MOCHA" [18]. Another recent FL method based on GTV regularization uses the choice $\phi(\mathbf{v}) := \|\mathbf{v}\|_1$ [74].

Different choices for the penalty function offer different trade-offs between computational aspects and statistical aspects of the resulting FL algorithms (see Section 10). As a case in point, the network Lasso (which uses $\phi(\mathbf{u}) = \|\mathbf{u}\|_2$) is computationally more challenging than MOCHA (which used penalty $\phi(\mathbf{u}) = (1/2)\|\mathbf{u}\|_2^2$). On the other hand, network Lasso is more accurate in learning models for data with specific network structures (such as chains) that are challenging for method that use the smooth penalty $\phi(\mathbf{v}) := (1/2)\|\mathbf{v}\|_2^2$ [75, 76].

8.2 Generalized Total Variation Minimization

We can enforce similar local model parameters $\mathbf{w}^{(i)} \approx \mathbf{w}^{(i')}$, at nodes i, i' connected by edge $\{i, i'\}$ with large weight $A_{i,i'}$, by favouring a small GTV

$\|\mathbf{w}\|_{\rm GTV}$. The extreme case of vanishing GTV is achieved by using identical local model parameters $\mathbf{w}^{(i)} = \mathbf{a}$ for all nodes $i \in \mathcal{V}$. However, we also need to take into account the local loss functions $L_i\left(\mathbf{w}^{(i)}\right)$ whose minimizers differ for nodes in different clusters (see Assumption 1).

It seems reasonable to learn local model parameters $\mathbf{w}^{(i)}$ by balancing between (the sum of) local loss functions and GTV (75). The optimal balance is obtained as the solutions of GTVMin

$$\widehat{\mathbf{w}} \in \arg\min_{\mathbf{w}\in\mathcal{W}} \sum_{i\in\mathcal{V}} L_i\left(\mathbf{w}^{(i)}\right) + \lambda\|\mathbf{w}\|_{\rm GTV}. \tag{76}$$

The regularization parameter $\lambda > 0$ in (76) steers the preference towards local model parameters $\mathbf{w}^{(i)}$ with small GTV instead of those incurring small total loss $\sum_{i\in\mathcal{V}} L_i\left(\mathbf{w}^{(i)}\right)$. The choice of λ can be guided by cross validation [1, Ch. 8], [39] or by characterizing the cluster structure of solutions to (76) (see Section 7.3). The cluster structure of GTVMin solutions and its dependence on λ has been studied in a recent line of work [68, 70, 72].

By increasing the value of λ in (76) we can enforce the resulting local model parameters $\widehat{\mathbf{w}}^{(i)}$ to be constant over increasingly larger subsets of nodes. Choosing λ larger than some critical value, that depends on the shape of the local loss functions and the network structure of $\mathcal{G}$, results in $\widehat{\mathbf{w}}^{(i)}$ being constant over all nodes $i \in \mathcal{V}$. Thus, we effectively train a single global model for all nodes (see Sections 9.1 and 9.2).

GTVMin (76) unifies and considerably extends some well-known methods for distributed optimization and learning. In particular, the network Lasso [73] is obtained from (76) for the choice $\phi(\mathbf{v}) := \|\mathbf{v}\|_2$. The MOCHA method [18] is obtained from (76) for the choice $\phi(\mathbf{v}) := \|\mathbf{v}\|_2^2$. Another special case of (76), obtained for the choice $\phi(\mathbf{v}) := \|\mathbf{v}\|_1$, has been studied recently [74].

8.3 Interpretations

We next discuss some useful relations between GTVMin (76) and basic ML techniques.

8.3.1 Regularized Empirical Risk Minimization

Note that GTVMin (76) is an instance of RERM (27) that uses GTV (75) for the regularizer. We measure the empirical risk incurred by the networked model parameters $\mathbf{w} \in \mathcal{W}$ by the sum of the local loss functions $\sum_{i \in \mathcal{V}} L_i \left(\mathbf{w}^{(i)} \right)$. The interpretation of GTVMin as a form of RERM suggests to apply optimization methods for structured optimization problems to FL applications that can be formulated as GTVMin [77].

8.3.2 Multi-task Learning

Consider networked data with empirical graph $\mathcal{G}$ whose nodes $i \in \mathcal{V}$ carry local datasets $\mathcal{D}^{(i)}$. Each node also carries a local model $\mathcal{H}^{(i)}$ which is trained from $\mathcal{D}^{(i)}$. The training of the local model $\mathcal{H}^{(i)}$ is a separatelearning task, one for each node $i \in \mathcal{V}$. GTVMin couples these learning tasks via requiring a small GTV of local model parameters $\mathbf{w}^{(i)}$. Thus, we can interpret GTVMin as a special case of multi-task learning [78, 79].

8.3.3 Few-Shot Learning

[80]

8.3.4 Clustering

It can be shown that the solutions of GTVMin are approximately piecewise constant over well-connected subsets of nodes in the empirical graph [65, 68, 70, 75]. Thus, we can interpret GTVMin as a generalization of graph clustering methods such as spectral clustering or flow-based clustering [81, 82].

In contrast to basic graph clustering methods, the clustering delivered by GTVMin is determined not only by the network structure of the empirical graph. Indeed, the resulting cluster structure also depends on the local loss functions $L_i(\cdot)$ which are, in turn, determined by the local datasets $\mathcal{D}^{(i)}$.

We obtain convex clustering [83, 84] as special case of GTVMin (76) with the local loss functions

$$L_i\left(\mathbf{w}^{(i)}\right) = \left\|\mathbf{w}^{(i)} - \mathbf{x}^{(i)}\right\|_2^2, \text{ for all nodes } i \in \mathcal{V} \tag{77}$$

and GTV penalty $\phi(\mathbf{u}) = \|\mathbf{u}\|_p$ being a p-norm $\|\mathbf{u}\|_p := \left(\sum_{j=1}^{d} |u_j|^p\right)^{1/p}$ with some $p \geq 1$. The feature vectors $\mathbf{x}^{(i)}$ in (77) characterize data points that we wish to cluster. Each cluster is formed by all data points $\mathbf{x}^{(i)}$ that have identical local model parameters $\mathbf{w}^{(i)}$.

8.3.5 Locally Weighted Learning

Another interpretation of GTVMin is that of locally weighted learning [85]. Indeed, the local model parameters delivered by GTVMin are approximately constant over well-connected subsets of nodes in the empirical graph [65, 68, 70, 75]. Thus, for each node i in a given cluster $\mathcal{C}$, the solution $\widehat{\mathbf{w}}^{(i)}$ of GTV approximates the solution $\overline{\mathbf{w}}^{(\mathcal{C})}$ of (73), $\widehat{\mathbf{w}}^{(i)} \approx \overline{\mathbf{w}}^{(\mathcal{C})}$.

Note that the cluster-wise optimization (73) can be written as a locally

weighted learning problem [85, Sec. 3.1.2]

$$\overline{\mathbf{w}}^{(\mathcal{C})} = \operatorname*{argmin}_{\mathbf{w} \in \mathbb{R}^d} \sum_{i' \in \mathcal{V}} \rho_{i'} L_{i'} \left(\mathbf{w} \right). \tag{78}$$

The locally-weighted learning problem (78) involves the weights $\rho_{i'}$ which are equal to 1 if $i' \in \mathcal{C}$ and equal to 0 otherwise.

8.4 Non-Parametric Models

So far, we focused on networked federated learning (NFL) methods for local models that are parametrized by local model parameters $\mathbf{w}^{(i)}$, for each node $i \in \mathcal{V}$. However, many important ML methods such as decision trees are non-parametric. We will now modify GTVMin for parametric local models to non-parametric local models (hypothesis spaces) $\mathcal{H}^{(i)}$ for nodes $i \in \mathcal{V}$. To this end, we first extend the concept of GTV to obtain a measure for the variation of local hypothesis maps $h^{(i)} \in \mathcal{H}^{(i)}$ that belong to non-parametric local models $\mathcal{H}^{(i)}$ such as decision trees.

We measure the variation between $h^{(i)}$ and $h^{(i')}$, across an edge $\{i, i'\} \in \mathcal{E}$, via the discrepancy between their predictions $h^{(i)}\big(\mathbf{x}\big), h^{(i')}\big(\mathbf{x}\big)$ for the data points in a common test-set

$$\mathcal{D}^{(\text{test})} = \Big\{ \mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m')} \Big\}. \tag{79}$$

The test-set $\mathcal{D}^{(\text{test})}$ must be shared with each node i, i' of an edge $\{i, i'\}$. Each of these two nodes can then compute the discrepancy

$$d\left(h^{(i)}, h^{(i')}\right) := (1/m') \sum_{r=1}^{m'} L^{(\text{d})}\left(\mathbf{x}^{(r)}, h^{(i')}\big(\mathbf{x}^{(r)}\big), h^{(i)}\big(\mathbf{x}^{(r)}\big)\right). \tag{80}$$

In principle, we can use different test-sets to compute the discrepancy over different edges in the empirical graph.

We are now in the position to generalize the GTV (75) from parametric local models to non-parametric local models. This generalization is obtained by summing the discrepancies (80) over all edges $\mathcal{E}$,

$$\mathrm{GTV}\left\{h\right\} := \sum_{\{i,i'\} \in \mathcal{E}} A_{i,i'} d\left(h^{(i)}, h^{(i')}\right). \tag{81}$$

We stress that (81) defines an entire ensemble of GTV measures $\mathrm{GTV}\left\{h\right\}$. This ensemble is obtained by different choices for the loss function $L^{(\mathrm{d})}\left(\cdot,\cdot,\cdot\right)$ used to compute the discrepancy $d\left(h^{(i)}, h^{(i')}\right)$ (80). This loss function depends on the hypothesis maps h, h' only via their predictions $h(\mathbf{x}), h'(\mathbf{x})$.

It is important to note that the loss function $L^{(\mathrm{d})}\left(\cdot,\cdot,\cdot\right)$ in (80) can be different from the loss function $L_i\left(\cdot\right)$ (69) used to train a local model. For example, the local models might be trained using a loss function for multi-class classification into 10 different classes while the loss function $L^{(\mathrm{d})}\left(\cdot,\cdot,\cdot\right)$ might be the squared error loss.

We obtain GTVMin for non-parametric local models by using $\mathrm{GTV}\left\{h\right\}$ as regularization term in RERM,

$$\widehat{h} \in \operatorname*{argmin}_{h \in \mathcal{H}^{(\mathcal{G})}} \sum_{i \in \mathcal{V}} L_i\left(h^{(i)}\right) + \lambda \mathrm{GTV}\left\{h\right\}. \tag{82}$$

Here, we use a networked hypothesis space $\mathcal{H}^{(\mathcal{G})} = \mathcal{H}^{(1)} \times \ldots \times \mathcal{H}^{(n)}$ whose elements are maps that assign a hypothesis $h^{(i)} \in \mathcal{H}^{(i)}$ to each node $i \in \mathcal{V}$. Note that (82) is parametrized by the choice for the loss function used to compute the discrepancy (80). Different choices for the loss function in (80) result in different computational aspects and statistical aspects of (82). If we

use the squared error loss to measure the discrepancy (80), (82) specializes to

$$\widehat{h} \in \operatorname*{argmin}_{h \in \mathcal{H}^{(\mathcal{G})}} \sum_{i \in \mathcal{V}} L_i \left(h^{(i)} \right) + (\lambda/m') \sum_{\{i,i'\} \in \mathcal{E}} A_{i,i'} \sum_{r=1}^{m'} \Big(h^{(i)}\big(\mathbf{x}^{(r)}\big) - h^{(i')}\big(\mathbf{x}^{(r)}\big) \Big)^2 . \quad (83)$$

8.5 Graph Learning Methods

We have introduced the empirical graph as a representation of collections of local datasets and their similarity structure.The empirical graph is often a design choice that should be guided by computational aspects and statistical aspects. Computationally, we prefer a sparse empirical graph with few edges. Indeed, the FL methods presented in Section 10 can be implemented as message passing over the empirical graph which means that the amount of computation is proportional to the number of edges. Statistically, we want the empirical graph to reflect an underlying cluster structure (see Assumption 1). This might require to have a sufficiently large number of edges connecting nodes in the same cluster.

8.5.1 Similarity Measures from Statistics

An edge between nodes $i, i' \in \mathcal{V}$ should represent a statistical similarity between local datasets $\mathcal{D}^{(i)}$ and $\mathcal{D}^{(i')}$. To make this notion precise we might use the i.i.d. assumption from Section 3.1 for each local dataset. In particular, we interpret the data points in $\mathcal{D}^{(i)}$ as realizations of i.i.d. RVs with common probability distribution $p^{(i)}\big(\mathbf{x}, y\big)$. The probability distributions $p^{(i)}\big(\mathbf{x}, y\big)$ can be different for different nodes $i \in \mathcal{V}$. However, we will tacitly assume that the features and labels of data points belong to the same feature space $\mathcal{X}$ (e.g., $\mathcal{X} = \mathbb{R}^d$) and to the same label space $\mathcal{Y}$ (e.g., $\mathcal{Y} = \mathbb{R}$) for all local datasets.

We denote the composition of feature space and label space by $\mathcal{Z} := \mathcal{X} \times \mathcal{Y}$.

We can then define the presence of an edge (and its weight) between nodes $i, i' \in \mathcal{V}$ using various measures for the distance $D\big(p^{(i)}, p^{(i')}\big)$ between $p^{(i)}\big(\mathbf{x}, y\big)$ and $p^{(i')}\big(\mathbf{x}, y\big)$. One example for such a distance measure is the Kullback-Leibler divergence which can be defined for any pair of probability distributions under mild technical conditions.

In particular, the KL divergence between two multivariate normal distributions $p^{(i)}, p^{(i')}$, each having mean $\boldsymbol{\mu}^{(i)}, \boldsymbol{\mu}^{(i')} \in \mathbb{R}^d$ and a non-singular (invertible) covariance matrix $\mathbf{C}^{(i)}, \mathbf{C}^{(i')}$, respectively, is

$$D^{(\mathrm{KL})}\big(p^{(i)}, p^{(i')}\big) = (1/2)\Big(\mathrm{tr}\big\{\big(\mathbf{C}^{(i')}\big)^{-1}\mathbf{C}^{(i)}\big\} - d \\ + \big(\boldsymbol{\mu}^{(i')} - \boldsymbol{\mu}^{(i)}\big)^T\big(\mathbf{C}^{(i')}\big)^{-1}\big(\boldsymbol{\mu}^{(i')} - \boldsymbol{\mu}^{(i)}\big) \\ + \log\big[\det\Big(\mathbf{C}^{(i')}\Big)/\det\big(\mathbf{C}^{(i)}\big)\,\big]\Big). \tag{84}$$

Note that the KL divergence (84) is not symmetric, i.e., $D^{(\mathrm{KL})}\big(p^{(i)}, p^{(i')}\big) \neq D^{(\mathrm{KL})}\big(p^{(i')}, p^{(i)}\big)$ in general. However, our definition of an empirical graph uses undirected edges. To obtain a symmetric distance measure, we can use the average

$$D\big(p^{(i)}, p^{(i')}\big) := (1/2)\big(D^{(\mathrm{KL})}\big(p^{(i)}, p^{(i')}\big) + D^{(\mathrm{KL})}\big(p^{(i')}, p^{(i)}\big)\big).$$

In practice, we typically do not know the underlying probability distribution underlying a $\mathcal{D}^{(i)}$, including its mean vector and covariance matrix. However, we can estimate these parameters using the sample mean and the

sample covariance matrix,

$$\widehat{\boldsymbol{\mu}}^{(i)} = (1/m)\sum_{r=1}^{m} \mathbf{x}^{(i,r)} \text{ (sample mean)}$$

$$\widehat{\mathbf{C}}^{(i)} = (1/m)\sum_{r=1}^{m} (\mathbf{x}^{(i,r)} - \widehat{\boldsymbol{\mu}}^{(i)})(\mathbf{x}^{(i,r)} - \widehat{\boldsymbol{\mu}}^{(i)})^T \text{ (sample covariance).} \quad (85)$$

We can plug in the estimates (85) into (84) to obtain a distance measure $\widehat{D}\big(p^{(i)}, p^{(i')}\big)$ that can be computed directly from local datasets.

Another measure for the (dis-)similarity between probability distribution is the Wasserstein distance [86]. This distance is a metric on the space of probability distributions defined on a metric space $\mathcal{Z}$ with metric $d(P, Q)$. It is given, for two probability distribution P and Q, as

$$W_p(P, Q) = \left(\inf_{\gamma \in \Gamma(P,Q)} \int_{\mathcal{Z}\times\mathcal{Z}} d(\mathbf{z}, \mathbf{z}')^p d\gamma(\mathbf{z}, \mathbf{z}') \right)^{1/p}.$$

Here, $p \geq 1$ denotes a tunable parameter and $\Gamma(P, Q)$ is the set of all joint probability distributions $\gamma(\mathbf{z}, \mathbf{z}')$ on $\mathcal{Z} \times \mathcal{Z}$ whose marginals are P and Q. More precisely,

$$\begin{aligned} \Gamma(P, Q) = \{\gamma(\mathbf{z}, \mathbf{z}') : & \int_{\mathcal{Z}\times\mathcal{Z}} \gamma(\mathbf{z}, \mathbf{z}') d\mathbf{z} d\mathbf{z}' = 1, \\ & \int_{\mathcal{Z}} \gamma(\mathbf{z}, \mathbf{z}') d\mathbf{z} = P(\mathbf{z}'), \\ & \int_{\mathcal{Z}} \gamma(\mathbf{z}, \mathbf{z}') d\mathbf{z}' = Q(\mathbf{z})\}. \end{aligned}$$

Note that the Wasserstein distance is only defined for probability distribution on a metric space. In contrast, the KL divergence does not require the data points to belong to a metric space.

Given a measure $\widehat{D}\big(p^{(i)}, p^{(i')}\big)$ for the (dis-)similarity between any two local datasets $\mathcal{D}^{(i)}$, $\mathcal{D}^{(i')}$ to construct an empirical graph (with nodes $\mathcal{V} = \{1, \ldots, n\}$) in several ways.

- **Thresholding.** Add an edge between nodes $i, i' \in \mathcal{V}$ if $\widehat{D}\big(p^{(i)}, p^{(i')}\big)$ is below a prescribed threshold η,

$$\{i, i'\} \in \mathcal{E} \text{ if and only if } \widehat{D}\big(p^{(i)}, p^{(i')}\big) < \eta. \tag{86}$$

- **Nearest neighbours.** Fix a number $k \in \{0, \ldots, n-1\}$, then iterate over all nodes $i \in \mathcal{V}$ (in some order), adding k edges between i and k nodes $i' \in \mathcal{V} \setminus \{i\}$ with smallest distance $\widehat{D}\big(p^{(i)}, p^{(i')}\big)$. The hyper-parameter k roughly determines the maximum node degree for any node $i \in \mathcal{V}$ in the empirical graph.

- **Fully connected weighted empirical graph.** Add an edge $\{i, i'\}$ between every pair $i, i' \in \mathcal{V}$ of two distinct nodes. Determine the edge weight $A_{i,i'}$ based on $\widehat{D}\big(p^{(i)}, p^{(i')}\big)$,

$$A_{i,i'} = g\big(\widehat{D}\big(p^{(i)}, p^{(i')}\big)\big) \text{ with some profile } g(\cdot) : \mathbb{R}_+ \to \mathbb{R}_+. \tag{87}$$

  One popular choice for the profile in (87) is $g(a) := \exp\big(-a^2\big)$ [81, Sec. 8], [87]. The threshold rule (86) is the extreme case of (87) which is obtained for the specific profile function $g(a) = 1$ if $a < \eta$ and $g(a) = 0$ if $a \geq \eta$.

8.5.2 Graph Learning via GTVMin

Section 8 defines GTV as a (family of) measure(s) for the variation of local model parameters across edges of the empirical graph. The FL methods in Section 10 are all based on optimizing the GTV, viewed as a function of the local model parameters. However, if we have sufficiently large local datasets $\mathcal{D}^{(i)}$, we might be able to learn useful model parameters separately using

ordinary ML methods [27]. We can then use these model parameters to learn the edges of the empirical graph by minimizing the GTV, viewed as a function of the empirical graph [88–90]. This learning approach might also leverage structural constrains on the empirical graph such as maximum node degree or the shape of the spectrum of the Laplacian matrix.

8.5.3 Graph Learning as Model Selection

We can interpret the empirical graph as a tuning (hyper-) parameter for GTVMin (76). Thus, learning the empirical graph is actually a model selection problem [1, Ch. 6]. In particular, we can choose between different candidates $\mathcal{G}^{1)}, \ldots, \mathcal{G}^{(M)}$ for the empirical graph by solving GTVMin for each candidate and then computing validation errors of each learnt hypothesis map $\widehat{h}^{(i)}$, for $i \in \mathcal{V}$. We then pick the empirical graph which results in the smallest (average) validation error.

8.5.4 Metric Learning

Section 8.5.1 discussed how to use similarity measures between probability distribution to construct the edges and their weights of an empirical graph. An alternative perspective is to determine a similarity measure that is more directly tied to the ultimate goal of a FL method, i.e., to train local models. We could try to learn a similarity measure between two local datasets by evaluating the effect of combining them, e.g., pooling them, in order to better train a local model. A similar idea has been used for few-shot learning [80].

8.6 Exercises

Exercise 8.1. Nearest Neighbour Graph (5 points). One approach to learn an empirical graph $\mathcal{G}^{(\mathrm{nn})}$ for a collection of n local datasets is to first compute a measure for the pairwise distances between local datasets and then iterating over all nodes i and connected it with the k nearest nodes (see Section 8.5). Here, $k \in \{0, \ldots, n-1\}$ is a hyper-parameter that controls the number of edges in the resulting empirical graph. Can we be sure that each node i in $\mathcal{G}^{(\mathrm{nn})}$ has at most k neighbours ?

9 Main Flavours of Federated Learning

Let us now discuss how different choices for the empirical graph and local loss functions used in GTVMin result in main flavours of FL [91]. The most basic FL setup is obtained when the empirical graph is a star with the centre node acting as a server or coordinator (see Section 9.1). Section 9.2 explains decentralized FL where an arbitrary empirical graph is used for message passing implementations of FL algorithms to collaboratively train a single global model.

Note that GTVMin uses a regularization parameter λ to control the (strength of the) coupling between the local models at nodes that are connected by an edge in the empirical graph. Section 9.3 discusses clustered FL which uses this parameter to steer the clustering structure of the learnt local models.

One extreme case of clustered FL is when the empirical graph becomes one single cluster. This corresponds to decentralized FL of a global model that is shared among all nodes. Another extreme case is personalized FL, which is discussed in Section 9.4, where each node in the empirical graph becomes a separate cluster.

Section 9.5 discusses vertical FL from different local datasets that correspond to the same data points but use (have access to) different features. Section 9.6 discusses horizonal FL from local datasets that contain different data points but use the same features to characterize them [92, Ch. 4].

9.1 Centralized Federated Learning

Probably the most basic FL setup uses a server which is connected by separate links to each client. The clients generate local datasets which are used to train a single global model $\mathcal{H}$. FL methods use the server to maintain the current model parameters [14]. These global model parameters are broadcasted to the clients which compute model parameters updates based on their local datasets (e.g., using gradient steps). These local updates are sent back to the server which aggregates them to obtain new global model parameters.

The above server-based (centralized) FL setup correspond to GTVMin for an empirical graph being a star graph (see Figure 19).

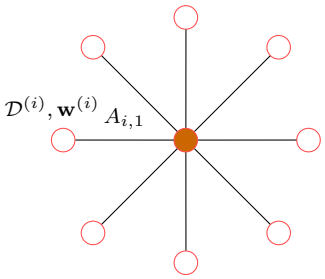

Figure 19: Star graph $\mathcal{G}^{(\text{star})}$ with centre node representing a server and peripheral nodes representing clients that generate local datasets.

9.2 Decentralized Federated Learning

[93] empirical graph is arbitrary; GTVMin parameter chosen so large that all local model parameters equal;

9.3 Clustered Federated Learning

Clustered FL uses different forms of a clustering assumption that requires local datasets, and their associated learning tasks, to form clusters (see Assumption 1) [22, 23, 78, 94, 95]. Local datasets belonging to the same cluster have similar statistical properties and, in turn, similar optimal parameter values for the corresponding local models. Our recent work offers a precise characterization of the cluster structure and local loss functions that allow FL methods from Section 10 to pool local datasets that form a cluster of statistically homogeneous local datasets [70]. This characterization also provides some guidance for choosing the regularization parameter λ in GTVMin (76).

9.4 Personalized Federated Learning

Using a sufficiently small regularization parameter in GTVMin (76) allows the local model parameters to vary over the nodes $i \in \mathcal{V}$. However, these trained personalized local models might still be coupled partially. For example, if local models are deep ANNs we might enforce the parameters of input layers to be identical while the parameters for the deeper layers might be different for each local dataset.

The partial parameter sharing for local models can be implemented in many different ways [96, Sec. 4.3.]. We can use GTVMin (76) for partial parameter sharing by using a specific choice for the GTV penalty function. In particular, we could construct the penalty function as a combination of two terms,

$$\phi\big(\mathbf{w}^{(i)} - \mathbf{w}^{(i')}\big) := \lambda^{(1)}\phi^{(1)}\big(\mathbf{w}^{(i)} - \mathbf{w}^{(i')}\big) + \lambda^{(2)}\phi^{(2)}\big(\mathbf{w}^{(i)} - \mathbf{w}^{(i')}\big). \tag{88}$$

Each component $\phi^{(1)}$, $\phi^{(2)}$ measures different parts (subsets) of the variation $\mathbf{w}^{(i)} - \mathbf{w}^{(i')}$ of local model parameters at connected nodes $\{i, i'\} \in \mathcal{E}$.

Moreover, we might use different regularization strengths $\lambda^{(1)}$ and $\lambda^{(2)}$ for different penalty components in (88) to enforce different subsets of the model parameters to be clustered with different granularity (cluster size). For local models being deep ANNs, we might want to enforce the low-level layers (closer to the input) to have same model parameters (weights and bias terms), while deeper (closer to the output) layers can have different model parameters. Figure 9.4 illustrates this setting for local models constituted by ANNs with a single hidden layer. Yet another technique for partial sharing of model parameters is to train a hyper-model which, in turn, is used to initialize the training of local models [97].

For heterogeneous collections of local models, including non-parametric local models such as decision trees, we can implement a partial coupling by carefully choosing the loss function in the discrepancy measure (80). In particular, we might use a loss function $L^{(\mathrm{d})}(\cdot, \cdot, \cdot)$ that is different from the loss used to measure the quality of a hypothesis $h^{(i)}$. Loosely speaking, the loss function $L^{(\mathrm{d})}(\cdot, \cdot, \cdot)$ might not be directly related to the ultimate learning task of the local models (which could be binary classification), but instead correspond to an auxiliary task (such as dimensionality reduction) which partially couples local models.

9.5 Vertical Federated Learning

Vertical FL refers to applications with local datasets that are constituted by the same data points but characterizing them with different features [98].

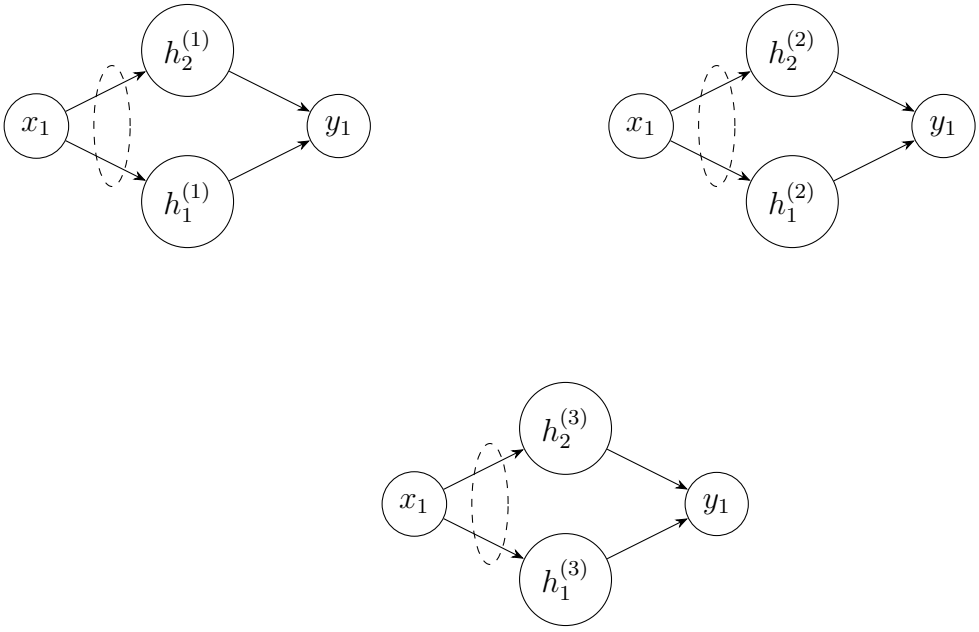

Figure 20: Personalized FL for local models being ANNs with one hidden layer. We couple the training of the hidden layer weights using GTVMin but let the output weights of a local model be freely trained on the local datasets.

In particular, vertical FL applications revolve around an underlying global dataset

$$\mathcal{D}^{(\text{global})} := \left\{ \left(\mathbf{x}^{(1)}, y^{(1)} \right), \ldots, \left(\mathbf{x}^{(m)}, y^{(m)} \right) \right\}.$$

Each data point in the global dataset is characterized by d' features $\mathbf{x}^{(r)} = \left(x_1^{(r)}, \ldots, x_{d'}^{(r)} \right)^T$. The global dataset can only be accessed indirectly via local datasets that use different subsets of the feature vectors $\mathbf{x}^{(r)}$.

Formally, the local dataset $\mathcal{D}^{(i)}$ contains the pairs $\left(\mathbf{x}^{(i,r)}, y^{(i,r)} \right)$, for $r = 1, \ldots, m$. The labels $y^{(i,r)}$ are identical to the labels in the global dataset, $y^{(i,r)} = y^{(r)}$. The feature vectors $\mathbf{x}^{(i,r)}$ are obtained by a subset $\mathcal{F}^{(i)} := \{j_1, \ldots, j_d\}$ of the original d' features in $\mathbf{x}^{(r)}$,

$$\mathbf{x}^{(i,r)} = \left(x_{j_1}^{(r)}, \ldots, x_{j_d}^{(r)} \right)^T.$$

9.6 Horizontal Federated Learning

Horizontal FL refers to applications with local datasets $\mathcal{D}^{(i)}$, for $i \in \mathcal{V}$ that contain data points characterized by the same features [99]. The local datasets $\mathcal{D}^{(i)}$ are obtained by different subsets of an underlying global dataset

$$\mathcal{D}^{(\text{global})} := \left\{ \left(\mathbf{x}^{(1)}, y^{(1)} \right), \ldots, \left(\mathbf{x}^{(m)}, y^{(m)} \right) \right\}.$$

The local dataset $\mathcal{D}^{(i)}$ is constituted by the data points of $\mathcal{D}^{(\text{global})}$ with indices in $\{r_1, \ldots, r_{m_i}\}$,

$$\mathcal{D}^{(i)} := \left\{ \left(\mathbf{x}^{(r_1)}, y^{(r_1)} \right), \ldots, \left(\mathbf{x}^{(r_{m_i})}, y^{(r_{m_i})} \right) \right\}.$$

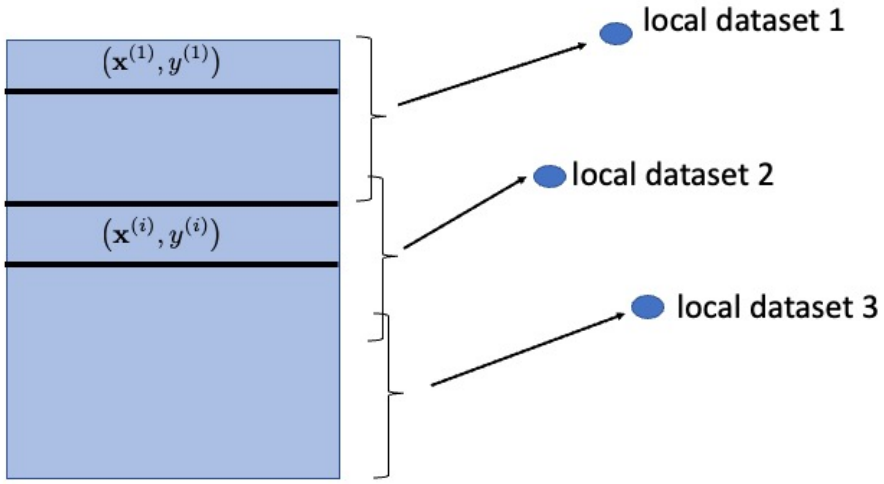

Figure 21: Horizontal FL uses the same features to characterize data points in different local datasets. Different local datasets are constituted by different subsets of an underlying global dataset. As an example consider local datasets that are constituted by patient records using the same type of personal information (social security number, home address). The underlying global dataset consists of the union of all patient records.

10 Federated Learning Algorithms

Section 8 introduced GTVMin as a design principle for FL methods that couple the training of local models on their corresponding local datasets. We obtain FL algorithms by the application of optimization methods to solve GTVMin. In particular, we will use distributed optimization methods to solve GTVMin in a decentralized fashion. The resulting FL algorithms are a special case of distributed algorithms that are implemented by message passing over the edges of the empirical graph [28]. Thus, we use the empirical graph not only to encode statistical similarites between local datasets but also as a computational model for the distributed FL algorithms presented in this chapter.

Throughout this chapter we will focus on the special case of GTVMin (76) for the penalty function $\phi(\mathbf{u}) = \|\mathbf{u}\|_2^2$,

$$\widehat{\mathbf{w}} \in \operatorname*{arg\,min}_{\mathbf{w} \in \mathcal{W}} \sum_{i \in \mathcal{V}} L_i \left(\mathbf{w}^{(i)} \right) + \lambda \sum_{\{i,i'\} \in \mathcal{E}} A_{i,i'} \left\| \mathbf{w}^{(i)} - \mathbf{w}^{(i')} \right\|_2^2. \tag{89}$$

We obtain different FL methods by applying different iterative optimization methods to solve GTVMin (89). These optimization methods construct a sequence

$$\widehat{\mathbf{w}}_0, \widehat{\mathbf{w}}_1, \ldots$$

of local model parameters that are (hopefully) increasingly accurate approximations of the solutions $\widehat{\mathbf{w}}$ to (89).

Section 10.1 applies gradient-based methods from Section 5 to solve (89). The resulting gradient step can be implemented as a message passing over the empirical graph.

Section 10.2 applies block-coordinate minimization to solve (89). The idea is to iteratively update each local model parameters $\mathbf{w}^{(i)}$ by minimizing (89) for given (fixed) local model parameters $\mathbf{w}^{(i')}$ at other nodes $i' \in \mathcal{V} \backslash \{i\}$ [30,41].

10.1 FedSGD

Consider the GTVMin instance (89) for local loss functions $L_i\left(\mathbf{w}^{(i)}\right)$ being differentiable. For differentiable local loss functions, also the overall objective function $f(\mathbf{w})$ in (89) is differentiable with gradient

$$\nabla f(\mathbf{w}) = \big(\mathbf{g}^{(1)}, \ldots, \mathbf{g}^{(n)}\big)^T \text{ with } \mathbf{g}^{(r)} := \nabla L_i\left(\mathbf{w}^{(i)}\right) + 2\lambda \sum_{i' \in \mathcal{N}^{(i)}} A_{i,i'}\big(\mathbf{w}^{(i)} - \mathbf{w}^{(i')}\big). \tag{90}$$

The gradient step for solving (89) is then

$$\begin{aligned} \widehat{\mathbf{w}}_{r+1}^{(i)} &:= \widehat{\mathbf{w}}_{r}^{(i)} - \alpha_{r,i} \nabla f(\widehat{\mathbf{w}}_{r}^{(i)}) \\ &\overset{(90)}{=} \widehat{\mathbf{w}}_{r}^{(i)} - \alpha_{r,i} \Bigg(\nabla L_i\left(\widehat{\mathbf{w}}_{r}^{(i)}\right) + 2\lambda \sum_{i' \in \mathcal{N}^{(i)}} A_{i,i'}\big(\widehat{\mathbf{w}}_{r}^{(i)} - \widehat{\mathbf{w}}_{r}^{(i')}\big) \Bigg). \end{aligned} \tag{91}$$

The gradient step (91) needs to be executed in parallel at all nodes $i \in \mathcal{V}$. To execute the gradient step at node $i \in \mathcal{V}$ it needs to have access to the current local model parameters $\widehat{\mathbf{w}}_{r}^{(i')}$ at the neighbours $i' \in \mathcal{N}^{(i)}$.

Note that the learning rate $\alpha_{r,i}$ in (91) might be different for different iterations r or at different nodes i. Thus, we might tailor the learning rate $\alpha_{r,i}$ to the local geometry of the empirical graph at node i. For example, we might scale the $\alpha_{r,i}$ by the (inverse of the) node degree $d^{(i)} = \big|\mathcal{N}^{(i)}\big|$.

Local Linear Regression. Let us specialize the gradient step (91) for training local linear models. Here, we learn a linear hypothesis map $\widehat{h}^{(i)}\big(\mathbf{x}\big) = \big(\widehat{\mathbf{w}}^{(i)}\big)^T \mathbf{x}$ with small average squared error loss on the local dataset

$\mathcal{D}^{(i)} = \big\{\big(\mathbf{x}^{(i,1)}, y^{(i,1)}\big), \ldots, \big(\mathbf{x}^{(i,m_i)}, y^{(i,m_i)}\big)\big\}$. The corresponding local loss function is

$$L_i\big(\mathbf{w}^{(i)}\big) := (1/m_i)\sum_{r=1}^{m_i}\big(y^{(i,r)} - \big(\mathbf{w}^{(i)}\big)^T\mathbf{x}^{(i,r)}\big)^2 \text{ at each node } i \in \mathcal{V}. \quad (92)$$

The gradient of the loss function (92) is

$$\begin{aligned}\nabla L_i\big(\mathbf{w}^{(i)}\big) &:= -(2/m_i)\sum_{r=1}^{m_i}\mathbf{x}^{(i,r)}\big(y^{(i,r)} - \big(\mathbf{w}^{(i)}\big)^T\mathbf{x}^{(i,r)}\big) \\ &= -(2/m_i)\big(\mathbf{X}^{(i)}\big)^T\big(\mathbf{y}^{(i)} - \mathbf{X}^{(i)}\mathbf{w}^{(i)}\big). \end{aligned} \quad (93)$$

Here, we used the feature matrix $\mathbf{X}^{(i)} \in \mathbb{R}^{m_i \times d}$ and label vector $\mathbf{y}^{(i)} \in \mathbb{R}^{m_i}$ at node $i \in \mathcal{V}$,

$$\mathbf{X}^{(i)} := \big(\mathbf{x}^{(i,1)}, \ldots, \mathbf{x}^{(i,m_i)}\big)^T, \text{ and } \mathbf{y}^{(i)} := \big(y^{(i,1)}, \ldots, y^{(i,m_i)}\big)^T. \quad (94)$$

As discussed in Section 5.4, computing the sum in (93) might be infeasible for some applications. The idea of SGD is to approximate the gradient by

$$g\big(\mathbf{w}^{(i)}\big) := -(2/m_i)\sum_{r\in\mathcal{B}}\mathbf{x}^{(i,r)}\big(y^{(i,r)} - \big(\mathbf{w}^{(i)}\big)^T\mathbf{x}^{(i,r)}\big)\Big(\approx \nabla L_i\big(\mathbf{w}^{(i)}\big)\Big). \quad (95)$$

As in Section 5.4, the approximation $g\big(\mathbf{w}^{(i)}\big)$ is constructed with a subset (batch) $\mathcal{B} = \Big\{\Big(\mathbf{x}^{(\hat{l}_1)}, y^{(\hat{l}_1)}\Big), \ldots, \Big(\mathbf{x}^{(\hat{l}_B)}, y^{(\hat{l}_B)}\Big)\Big\}$ of B randomly chosen (indices of) data points in $\mathcal{D}^{(i)}$.

We obtain Algorithm 2 by inserting (95) into the gradient step (91).

Algorithm 2 FedSGD for Local Linear Regression

Input: empirical graph $\mathcal{G}$; GTV parameter λ; learning rate $\alpha_{r,i}$

local dataset $\mathcal{D}^{(i)} = \big\{\big(\mathbf{x}^{(i,1)}, y^{(i,1)}\big), \ldots, \big(\mathbf{x}^{(i,m_i)}, y^{(i,m_i)}\big)\big\}$ for each node i;

Output: weights $\widehat{\mathbf{w}}^{(i)}$ of linear hypothesis $h(\mathbf{x}) = \big(\widehat{\mathbf{w}}^{(i)}\big)^{T}\mathbf{x}$ at each node $i \in \mathcal{V}$

Initialize: $r := 0$; $\widehat{\mathbf{w}}_0^{(i)} := \mathbf{0}$

1: **while** stopping criterion is not satisfied **do**

2: **for** all nodes $i \in \mathcal{V}$ (simultaneously) **do**

3: share local parameters $\widehat{\mathbf{w}}_r^{(i)}$ with all neighbours $i' \in \mathcal{N}^{(i)}$

4: generate a new random batch $\mathcal{B}^{(i)} := \{r_1, \ldots, r_B\}$

5: construct feature matrix $\widetilde{\mathbf{X}}^{(i)} \in \mathbb{R}^{B \times d}$ and label vector $\widetilde{\mathbf{y}}^{(i)} \in \mathbb{R}^{B}$,

$$\widetilde{\mathbf{X}}^{(i)} := \big(\mathbf{x}^{(i,r_1)}, \ldots, \mathbf{x}^{(i,r_B)}\big)^{T}, \text{ and } \widetilde{\mathbf{y}}^{(i)} := \big(y^{(i,r_1)}, \ldots, y^{(i,r_B)}\big)^{T}. \tag{96}$$

6: update local model parameters via (see (91))

$$\widehat{\mathbf{w}}_{r+1}^{(i)} := \widehat{\mathbf{w}}_r^{(i)} + 2\alpha_{r,i}\Big((1/m_i)\big(\widetilde{\mathbf{X}}^{(i)}\big)^{T}\big(\widetilde{\mathbf{y}}^{(i)} - \widetilde{\mathbf{X}}^{(i)}\widehat{\mathbf{w}}_r^{(i)}\big) - \lambda \sum_{i' \in \mathcal{N}^{(i)}} A_{i,i'}\big(\widehat{\mathbf{w}}_r^{(i)} - \widehat{\mathbf{w}}_r^{(i')}\big)\Big). \tag{97}$$

7: **end for**

8: $r := r+1$

9: **end while**

10: $\widehat{\mathbf{w}}^{(i)} := \widehat{\mathbf{w}}_r^{(i)}$ for all nodes $i \in \mathcal{V}$

10.2 FedRelax

We now apply block-coordinate minimization [30, 41] to solve GTVMin (89). To this end, we rewrite(89) as

$$\widehat{\mathbf{w}} \in \arg\min_{\mathbf{w}\in\mathcal{W}} \underbrace{\sum_{i\in\mathcal{V}} f^{(i)}(\mathbf{w})}_{:=f^{(\mathrm{GTV})}(\mathbf{w})}$$
$$\text{with } f^{(i)}(\mathbf{w}) := L_i\left(\mathbf{w}^{(i)}\right) + (\lambda/2)\sum_{i'\in\mathcal{N}^{(i)}} A_{i,i'}\left\|\mathbf{w}^{(i)}-\mathbf{w}^{(i')}\right\|_2^2. \quad (98)$$

The objective function in (98) decomposes into node-wise functions $f^{(i)}(\mathbf{w})$, for each $i \in \mathcal{V}$. Block-coordinate minimization exploits this structure to decouple the optimization of local model parameters $\widehat{\mathbf{w}}^{(i)}$.

We next discuss a basic variant of block-coordinate minimization to solve (98). Given the current local model parameters $\widehat{\mathbf{w}}_r$, we compute (hopefully improved) updated local model parameters $\widehat{\mathbf{w}}_{r+1}^{(i)}$ by minimizing $f^{(\mathrm{GTV})}(\cdot)$ along $\mathbf{w}^{(i)}$ starting from $\widehat{\mathbf{w}}_r$,

$$\begin{aligned}\widehat{\mathbf{w}}_{r+1}^{(i)} &\in \operatorname*{argmin}_{\mathbf{w}^{(i)}\in\mathbb{R}^d} f^{(\mathrm{GTV})}\left(\widehat{\mathbf{w}}_{r+1}^{(1)},\ldots,\widehat{\mathbf{w}}_{r+1}^{(i+1)},\mathbf{w}^{(i)},\widehat{\mathbf{w}}_{r}^{(i+1)},\ldots\right)\\ &\overset{(98)}{=} \operatorname*{argmin}_{\mathbf{w}^{(i)}\in\mathbb{R}^d} f^{(i)}\left(\widehat{\mathbf{w}}_{r+1}^{(1)},\ldots,\widehat{\mathbf{w}}_{r+1}^{(i-1)},\mathbf{w}^{(i)},\widehat{\mathbf{w}}_{r}^{(i+1)},\ldots\right)\\ &\overset{(98)}{=} \operatorname*{argmin}_{\mathbf{w}^{(i)}\in\mathbb{R}^d} L_i\left(\mathbf{w}^{(i)}\right) + \lambda\sum_{i'\in\mathcal{N}^{(i)}} A_{i,i'}\left\|\mathbf{w}^{(i)}-\widehat{\mathbf{w}}_{r^{(i,i')}}^{(i')}\right\|_2^2. \quad (99)\end{aligned}$$

Here, we used the short-hand

$$r^{(i,i')} = r \text{ for } i < i', \text{ and } r^{(i,i')} = r+1 \text{ for } i' < i. \quad (100)$$

Algorithm 3 repeats the update (99) simultaneously at all nodes $i \in \mathcal{V}$. The updates (99) are repeated until a stopping criterion is met, e.g., a maximum

number r' of updates at each node. Another stopping criterion can be obtained from monitoring the decrease of the objective function.

Algorithm 3 FedRelax for Parametric Local models

Input: empirical graph $\mathcal{G}$; local loss functions $L_i(\cdot)$, GTV parameter λ

Output: learnt local model parameters $\widehat{\mathbf{w}}^{(i)}$

Initialize: $r:=0$; $\widehat{\mathbf{w}}_0^{(i)}:=\mathbf{0}$

1: **while** stopping criterion is not satisfied **do**

2: **for** nodes $i=1,2,\ldots,$ **do**

3: update local model parameters (see (99), (100))

$$\widehat{\mathbf{w}}_{r+1}^{(i)} := \operatorname*{argmin}_{\mathbf{w}^{(i)} \in \mathbb{R}^d} L_i\left(\mathbf{w}^{(i)}\right) + (\lambda/2) \sum_{i' \in \mathcal{N}^{(i)}} A_{i,i'} \left\| \mathbf{w}^{(i)} - \widehat{\mathbf{w}}_{r^{(i,i')}}^{(i')} \right\|_2^2 \tag{101}$$

4: share local model parameters $\widehat{\mathbf{w}}_{r+1}^{(i)}$ with all neighbours $i' \in \mathcal{N}^{(i)}$

5: **end for**

6: $r:=r+1$

7: **end while**

8: $\widehat{\mathbf{w}}^{(i)} := \widehat{\mathbf{w}}_r^{(i)}$ for all nodes $i \in \mathcal{V}$

Non-Parametric Models. Algorithm 3 can only be used for parametric local models $\mathcal{H}^{(i)}$ such as linear regression or ANNs with a fixed number d of model parameters at each node $i \in \mathcal{V}$. We can extend the scope of Algorithm 3 to non-parametric models by applying block-coordinate minimization to the non-parametric GTVMin variant (82) instead of (89). We obtain Algorithm 4 by replacing the update in step 3 of Algorithm 3 with

$$\widehat{h}_{r+1}^{(i)} \in \operatorname*{argmin}_{h \in \mathcal{H}^{(i)}} L_i\left(h^{(i)}\right) + (\lambda/2) \sum_{i' \in \mathcal{N}^{(i)}} A_{i,i'} \underbrace{d\left(h^{(i)}, \widehat{h}_r^{(i')}\right)}_{\text{see (80)}}. \tag{102}$$

The update (102) uses the discrepancy (80) between local models to measure the GTV (81). Note that Algorithm 4 defines a large family of FL methods.

Algorithm 4 FedRelax

Input: empirical graph $\mathcal{G}$ with edge weights $A_{i,i'}$; local loss functions $L_i(\cdot)$; GTV parameter λ; discrepancy measure $d\left(h^{(i)}, \widehat{h}_r^{(i')}\right)$ for $i, i' \in \mathcal{V}$

Initialize: $r := 0$; $\widehat{h}_0^{(i)} \equiv 0$ for all nodes $i \in \mathcal{V}$

1: **while** stopping criterion is not satisfied **do**

2: **for** nodes $i = 1, 2, \ldots$ **do**

3: update local hypothesis $\widehat{h}_r^{(i)}$ as follows (see (100)):

$$\widehat{h}_{r+1}^{(i)} \in \operatorname*{argmin}_{h^{(i)} \in \mathcal{H}^{(i)}} \left[L_i\left(h^{(i)}\right) + \lambda \sum_{i' \in \mathcal{N}^{(i)}} A_{i,i'} d\left(h^{(i)}, \widehat{h}_{r^{(i,i')}}^{(i')}\right) \right] \tag{103}$$

4: share local hypothesis $\widehat{h}_{r+1}^{(i)}$ with all neighbours $i' \in \mathcal{N}^{(i)}$

5: **end for**

6: $r := r+1$

7: **end while**

This family is parametrized by design choices for the discrepancy measures $d\left(h^{(i)}, \widehat{h}_r^{(i')}\right)$, for each $i, i' \in \mathcal{V}$ with $\{i, i'\} \in \mathcal{E}$, and local models $\mathcal{H}^{(i)}$ for $i \in \mathcal{V}$. The main restriction for these design choices is that the resulting update (103) must allow for an efficient implementation.

For the extreme case when $\lambda = 0$, the update (103) reduces to the training of the local model $\mathcal{H}^{(i)}$ which can be implemented by optimization methods for solving ERM (18) [27]. For some choices of discrepancy measure we can reformulate (103) as ERM (see (18)) on an augmented dataset. We next discuss one such choice for the local models that are trained using squared

error loss.

We obtain Algorithm 5 as a special case of Algorithm 4 when using the squared error loss to measure both, the prediction error of a local hypothesis on the local dataset and their discrepancies across an edge $\{i, i'\} \in \mathcal{E}$ in the empirical graph. In particular, we measure the discrepancy (80) between hypothesis maps $h^{(i)}$ and $h^{(i')}$ by the average squared difference $\big(h^{(i)}\big(\mathbf{x}^{(r)}\big) - h^{(i')}\big(\mathbf{x}^{(r)}\big)\big)^2$ of their predictions on a common test-set $\mathcal{D}'$ (see (83)). Note that this common test-set, which is an input parameter to Algorithm 5, needs to be shared with all nodes $i \in \mathcal{V}$ in the empirical graph.

Algorithm 5 FedRelax Least-Squares Regression

Input: empirical graph $\mathcal{G}$ with edge weights $A_{i,i'}$; test-set $\mathcal{D}' = \big\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m')}\big\}$; local models $\mathcal{H}^{(i)}$; GTV parameter λ

Initialize: $r := 0$; $\widehat{h}_0^{(i)} \equiv 0$ for all nodes $i \in \mathcal{V}$

1: **while** stopping criterion is not satisfied **do**

2: **for** nodes $i = 1, 2, \ldots$ **do**

3: update hypothesis $\widehat{h}_r^{(i)}$ as follows (see (100)):

$$\widehat{h}_{r+1}^{(i)} \in \operatorname*{argmin}_{h^{(i)} \in \mathcal{H}^{(i)}} \Bigg[(1/m_i) \sum_{(\mathbf{x},y) \in \mathcal{D}^{(i)}} \big(y - h^{(i)}\big(\mathbf{x}\big)\big)^2 + (\lambda/m') \sum_{i' \in \mathcal{N}^{(i)}} A_{i,i'} \sum_{r=1}^{m'} \Big(h^{(i)}\big(\mathbf{x}^{(r)}\big) - \widehat{h}_{r^{(i,i')}}^{(i')}\big(\mathbf{x}^{(r)}\big)\Big)^2 \Bigg] \tag{104}$$

4: share predictions $\Big\{\widehat{h}_{r+1}^{(i)}\big(\mathbf{x}\big)\Big\}_{\mathbf{x} \in \mathcal{D}'}$, with neighbours $i' \in \mathcal{N}^{(i)}$

5: **end for**

6: $r := r+1$

7: **end while**

The update (104) in step (3) of Algorithm 5 is nothing but an instance of ERM using the local model $\mathcal{H}^{(i)}$ and an augmentation $\widetilde{\mathcal{D}^{(i)}}$ of the local dataset $\mathcal{D}^{(i)}$,

$$\widetilde{\mathcal{D}^{(i)}} := \mathcal{D}^{(i)} \cup \bigcup_{i' \in \mathcal{N}^{(i)}} \mathcal{D}_{i'}^{(\text{test})}. \tag{105}$$

The augmented dataset $\widetilde{\mathcal{D}^{(i)}}$ includes a separate copy $\mathcal{D}_{i'}^{(\text{test})}$ of the test-set for each neighbour $i' \in \mathcal{N}^{(i)}$. Each of these $d^{(i)}$ consists of data points with same feature vectors as those in $\mathcal{D}^{(\text{test})}$ but different labels. The labels of the data points in $\mathcal{D}_{i'}^{(\text{test})}$ are the predictions $\widehat{h}_{r+1}^{(i')}\big(\mathbf{x}\big)$ obtained from the current hypothesis $\widehat{h}_{r+1}^{(i')}$.

Note that Algorithm 5 uses the same test-set $\mathcal{D}^{(\text{test})}$ for all edges to compute the discrepancy of local hypothesis maps. However, for each edge $e = \{i, i'\}$ only the predictions of the local hypothesis maps $\widehat{h}_{r}^{(i')}$, $\widehat{h}_{r+1}^{(i)}$ are compared on the test-set. Moreover, the discrepancy between these predictions is also weighted by the edge weight $A_{i,i'}$.

10.3 FedAvg

Consider a networked dataset with empirical graph $\mathcal{G}$ whose nodes carry local datasets and local models $\mathcal{H}^{(i)} = \mathcal{H}$ parametrized by local model parameters $\mathbf{w}^{(i)} \in \mathbb{R}^d$. In some FL applications we want to train a single global model $\mathcal{H}$ from the local datasets. This is equivalent to training the local models with the constraint of identical local model parameters,

$$\widehat{\mathbf{w}} \in \arg\min_{\mathbf{w} \in \mathcal{C}} \sum_{i \in \mathcal{V}} L_i\left(\mathbf{w}^{(i)}\right) \\ \text{with } \mathcal{C} = \left\{\mathbf{w} : \mathbf{w}^{(i)} = \mathbf{w}^{(i')} \text{ for any edge } \{i, i'\} \in \mathcal{E}\right\}. \tag{106}$$

For differentiable local loss functions, we can solve (106) using projected GD (see Section 5.2). In particular, the projection of networked model parameters $\mathbf{w} = \left(\mathbf{w}^{(1)}, \ldots, \mathbf{w}^{(n)}\right)^T$ on the constraint set $\mathcal{C}$ of (106) is given as

$$P_{\mathcal{C}}\left(\mathbf{w}\right) = \left(\mathbf{v}, \ldots, \mathbf{v}\right)^T \text{ with } \mathbf{v} := (1/n) \sum_{i \in \mathcal{V}} \mathbf{w}^{(i)}. \tag{107}$$

The resulting projected gradient step for solving (106) is

$$\widehat{\mathbf{w}}^{(i)}_{r+1/2} := \widehat{\mathbf{w}}^{(i)}_{r} - \alpha_{i,r} \nabla L_i\left(\widehat{\mathbf{w}}^{(i)}_{r}\right) \qquad \text{(local update at client } i\text{)} \tag{108}$$

$$\widehat{\mathbf{w}}^{(i)}_{r+1} := (1/n) \sum_{i \in \mathcal{V}} \widehat{\mathbf{w}}^{(i)}_{r+1/2} \qquad \text{(server averages)} \tag{109}$$

The averaging step (109) might take much longer to execute than the local update step (108). Indeed, (109) typically requires transmission of local model parameters from every client $i \in \mathcal{V}$ to a server or central computing unit. Thus, after the client $i \in \mathcal{V}$ has computed the local gradient step (108) it must wait until the server has collected the updates $\widehat{\mathbf{w}}^{(i)}_{r+1/2}$ from all clients and send back their average to $i \in \mathcal{V}$.

To avoid any waste of computational resources, when clients wait until they receive $\widehat{\mathbf{w}}_{r+1}^{(i)}$ back from the server, we might replace the single local gradient step (108) by several such gradient steps. Instead of using several local gradient steps, we could also use a local minimization of $L_i(\cdot)$ around $\widehat{\mathbf{w}}_r^{(i)}$,

$$\widehat{\mathbf{w}}_{r+1/2}^{(i)} := \operatorname*{argmin}_{\mathbf{v}\in\mathbb{R}^d} \left[L_i(\mathbf{v}) + \lambda_{i,r} \left\| \mathbf{v} - \widehat{\mathbf{w}}_r^{(i)} \right\|^2 \right]. \tag{110}$$

We obtain Algorithm 6 from projected GD for solving (106) by replacing the gradient step (108) with the local minimization (110). The parameter $\lambda_{i,r}$ controls the size of the neighbourhood around $\widehat{\mathbf{w}}_r^{(i)}$ over which (110) optimizes the local loss function $L_i(\cdot)$. Note that the local minimization step (110) is an instance of the RERM (30).

Let us spell out the special case of Algorithm 6 obtained when learning local linear models by minimizing the average squared error loss (5) incurred on the local dataset

$$\mathcal{D}^{(i)} := \left\{ \left(\mathbf{x}^{(i,1)}, y^{(i,1)}\right), \ldots, \left(\mathbf{x}^{(i,m_i)}, y^{(i,m_i)}\right) \right\}.$$

The resulting Algorithm 7 performs regularized linear regression on each local dataset. The regularization term is constructed from the average of the current local model parameters and therefore couples the training of local linear models.

Interpretation of FedAvg as Extreme Case of FedSGD. The popular federated averaging (FedAvg) method is obtained from (89) for $\lambda \to \infty$. Indeed, for sufficiently large λ, the penalty term in (89) dominates, enforcing the learnt local model parameters $\widehat{\mathbf{w}}^{(i)}$ to be nearly identical, $\widehat{\mathbf{w}}^{(i)} \approx \widehat{\mathbf{w}}^{(i')}$ for any two nodes i, i' belonging to the same connected component of the

Algorithm 6 Federated Averaging (FedAvg)

Input: client list $\mathcal{V}$

Server. (available under `fljung.cs.aalto.fi`)

Initialize. $r := 0$

1: **while** stopping criterion is not satisfied **do**

2: receive local model parameters $\mathbf{w}^{(i)}$ for all clients $i \in \mathcal{V}$

3: update global model parameters

$$\overline{\mathbf{w}}\,[r] := \left(1/\left|\mathcal{V}\right|\right) \sum_{i \in \mathcal{V}} \mathbf{w}^{(i)}.$$

4: send new global model parameters $\overline{\mathbf{w}}\,[r]$ to all clients $i \in \mathcal{V}$

5: $r := r+1$

6: **end while**

Client. (some $i \in \mathcal{V}$)

1: **while** stopping criterion is not satisfied **do**

2: receive global model parameters $\overline{\mathbf{w}}\,[r]$ from server

3: update local model parameters

$$\mathbf{w}^{(i)} := \underset{\mathbf{v} \in \mathbb{R}^d}{\operatorname{argmin}} \left[L_i\left(\mathbf{v}\right) + \lambda \left\| \mathbf{v} - \overline{\mathbf{w}}\,[r] \right\|^2 \right]. \tag{111}$$

4: send $\mathbf{w}^{(i)}$ to `fljung.cs.aalto.fi`

5: **end while**

Algorithm 7 FedAvg for linear regression

Input: client list $\mathcal{V}$

Server. (available under `fljung.cs.aalto.fi`)

Initialize. $r := 0$

1: **while** stopping criterion is not satisfied **do**

2: receive local model parameters $\mathbf{w}^{(i)}$ from all clients $i \in \mathcal{V}$

3: update global model parameters

$$\overline{\mathbf{w}}\,[r] := \big(1/\big|\mathcal{V}\big|\big) \sum_{i \in \mathcal{V}} \mathbf{w}^{(i)}.$$

4: send updated global model parameters $\overline{\mathbf{w}}\,[r]$ to all clients $i \in \mathcal{V}$

5: $r := r+1$

6: **end while**

Client. (at some node $i \in \mathcal{V}$)

1: **while** stopping criterion is not satisfied **do**

2: receive global model parameters $\overline{\mathbf{w}}\,[r]$ from server

3: update local model parameters by RERM (see (30))

$$\mathbf{w}^{(i)} := \underset{\mathbf{v} \in \mathbb{R}^d}{\operatorname{argmin}} \left[(1/m_i) \sum_{r=1}^{m_i} \left(\mathbf{v}^T \mathbf{x}^{(i,r)} - y^{(i,r)} \right)^2 + \lambda \left\| \mathbf{v} - \overline{\mathbf{w}}\,[r] \right\|_2^2 \right].$$

4: send $\mathbf{w}^{(i)}$ to `fljung.cs.aalto.fi`

5: **end while**

empirical graph $\mathcal{G}$. Thus, for sufficiently large λ, we can approximate (89) by (106).

10.4 Asynchronous Computation

The FL methods presented in Section 10.2 - Section 10.3 assume a perfectly synchronous mode of computation. As a case in point, consider the synchronous mode of operation required by Algorithm 3. In particular,the step 4 of Algorithm 3 has to be carried simultaneously at all nodes. Only when step 4 has been completed at all nodes, Algorithm 3 can continue with executing the update 101. However, what happens if some of the nodes $i \in \mathcal{V}$ fail to complete step 4 of Algorithm 3?

10.5 Exercises

Exercise 10.1. FedRelax uses Proximity Operator (5 points) Show that the coordinate minimization update (99) is equivalent to the evaluation of the proximity operator $\mathbf{prox}_{L_i(\cdot),\rho}(\mathbf{w}')$ for a specific vector $\mathbf{w}' \in \mathbb{R}^d$ and $\rho > 0$.

Exercise 10.2. FedAvg Linear Regression (5 points) Discuss how step 3 of Algorithm 7 could be computed using the `LinearRegression.fit()` method provided by the Python package `scikit-learn`.

Exercise 10.3. GTVMin for Local Linear Regression (5 points) Consider the GTVMin instance (89) with local loss functions given by the average squared error loss of a linear hypothesis map with local model parameters $\mathbf{w}^{(i)}$. Show that the objective function in (89) can then be written as a convex quadratic function $f(\mathbf{w}) = \mathbf{w}^T\mathbf{Q}\mathbf{w} + \mathbf{q}^T\mathbf{w} + q$ with networked model parameters $\mathbf{w} := \big(\mathbf{w}^{(1)}, \ldots, \mathbf{w}^{(n)}\big)^T$. Identify the psd matrix $\mathbf{Q}$, vector $\mathbf{q}$ and scalar q in terms of the feature vectors $\mathbf{x}^{(i,r)}$ and labels $y^{(i,r)}$ of the data points in the local datasets.

Part III

Trustworthy Federated Learning

11 Requirements for Trustworthy AI

FL is an important subfield of artificial intelligence (AI). As part of their AI strategy, the European Commission set up the High-Level Expert Group on Artificial Intelligence(AI HLEG) in 2018. This group put forward seven key requirements for trustworthy AI [100, 101]:

1. **KR1 - Human Agency and Oversight.** "*..AI systems should support human autonomy and decision-making, as prescribed by the principle of respect for human autonomy. This requires that AI systems should both act as enablers to a democratic, flourishing and equitable society by supporting the user's agency and foster fundamental rights, and allow for human oversight...*" [101, p.15]

2. **KR2 - Technical Robustness and Safety.** "*...Technical robustness requires that AI systems be developed with a preventative approach to risks and in a manner such that they reliably behave as intended while minimising unintentional and unexpected harm, and preventing unacceptable harm. ...*' [101, p.16].

3. **KR3 - Privacy and Data Governance.** "*..privacy, a fundamental right particularly affected by AI systems. Prevention of harm to privacy also necessitates adequate data governance that covers the quality and integrity of the data used...*" [101, p.17].

4. **KR4 - Transparency.** "*...This requirement is closely linked with the principle of explicability and encompasses transparency of elements*

relevant to an AI system: the data, the system and the business models." [101, p.18].

5. **KR5 - Diversity Non-Discrimination and Fairness.** "*...we must enable inclusion and diversity throughout the entire AI system's life cycle...this also entails ensuring equal access through inclusive design processes as well as equal treatment.*" [101, p.18].

6. **KR6 - Societal and Environmental Well-Being.** "*...Sustainability and ecological responsibility of AI systems should be encouraged, and research should be fostered into AI solutions addressing areas of global concern, such as for instance the Sustainable Development Goals.*" [101, p.19].

7. **KR7 - Accountability.** "*...mechanisms be put in place to ensure responsibility and accountability for AI systems and their outcomes, both before and after their development, deployment and use.*" [101, p. 19]

We next discuss in a step-by-step fashion how each of these requirements guides the design choices in the FL methods of Section 10. These design choices include the local models, local loss functions and the discrepancy measure (see (80)) used in GTVMin (see Section 8).

11.1 Human Agency and Oversight

Section 10 derived FL algorithms by applying optimization methods to (instances of) GTVMin (76). The computational and statistical properties of these algorithms crucially depend on the design choices for the building blocks

of GTVMin. One important point of human oversight are these design choices, including the local dataset, local models and local loss functions, that allow for human oversight. For example, we might not be allowed to use senstive properties of data points as their label which we aim to predict.

11.2 Technical Robustness and Safety

11.3 Privacy and Data Governance

11.4 Transparency

11.5 Diversity, Non-Discrimination and Fairness

11.6 Societal and Environmental Well-Being

11.7 Accountability

12 Privacy Protection

Any FL system involves the sharing of information between different clients which hold local datasets. Indeed, if there would be no information shared about a the local dataset of some client, then we could safely drop this client from the FL system. Algorithm 2 shares information via passing on the gradients of local loss functions. Algorithm 6 shares information via sharing of local model parameters updates and Algorithm 4 shares information via the predictions of local models on a common test set. The test-set used in FedRelaxDT contains non-sensitive synthetic data points. However, sharing the predictions of a trained local model at node i might reveal senstive information contained in the local dataset $\mathcal{D}^{(i)}$.

A main application domain of FL is healthcare where local datasets are collected at different healthcare providers such as hospitals or laboratories [102]. Let us assume that local datasets contain data points that represent human beings. The features of data points are different bio-physical measurements and diagnosis reports. The label might be the occurrence of some disease such as diabetes.

Some of these features and labels stored in a local dataset might carry sensitive information (e.g., presence of some infection) and cannot be shared beyond the owner of the local dataset (which could be a blood lab). We refer to such properties as private and need to ensure that they are not be shared beyond the owner of the local dataset. However, the (non-sensitive part of the) information contained in the local dataset might be helpful to train a high-dimensional ML model for data-driven diagnosis or personalized medication.

The FL methods discussed in Section 10 allow to share the information contained in local datasets to train local models in a privacy-friendly fashion. By privacy-friendly, we mean that no information about private properties (either a feature or a label) of a data point is revealed to anybody beyond the owner (or controller) of the local dataset. As a case in point, we will study the information flow during the iterations of Algorithm 3. Sections 12.2 and 12.3 will then explain modifications of Algorithm 3 to avoid the leakage of sensitive information. Note that Algorithm 3 only applies to parametrized hypothesis spaces which contain hypothesis maps $h^{(\mathbf{w}^{(i)})}$ that are fully determined by a parameter vector $\mathbf{w}^{(i)}$.

Figure 22 illustrates the data (and information) flow arising at node i during a single iteration of Algorithm 3. During each iteration of Algorithm 3, each node $i \in \mathcal{V}$ computes the update (99) and shares the updated local parameters $\widehat{\mathbf{w}}_{r+1}^{(i)}$ with its neighbours $\mathcal{N}^{(i)}$. It is the sharing of the updated parameters in step 4 of Algorithm 3 that might cause leakage of sensitive information.

We next discuss two different approaches to avoid this leakage of sensitive information. The first approach in Section 12.2 is to perturb (e.g., by adding noise) the update $\widehat{\mathbf{w}}_{r+1}^{(i)}$ before sharing it with the neighbours $\mathcal{N}^{(i)}$. Section 12.3 presents a complementary approach which is to directly perturb the features and labels of the data points in the local dataset $\mathcal{D}^{(i)}$.

12.1 Privacy Leakage of Gradients

Deep Leakage from Gradients Ligeng Zhu, Zhijian Liu, Song Han

Exchanging gradients is a widely used method in modern multi-node

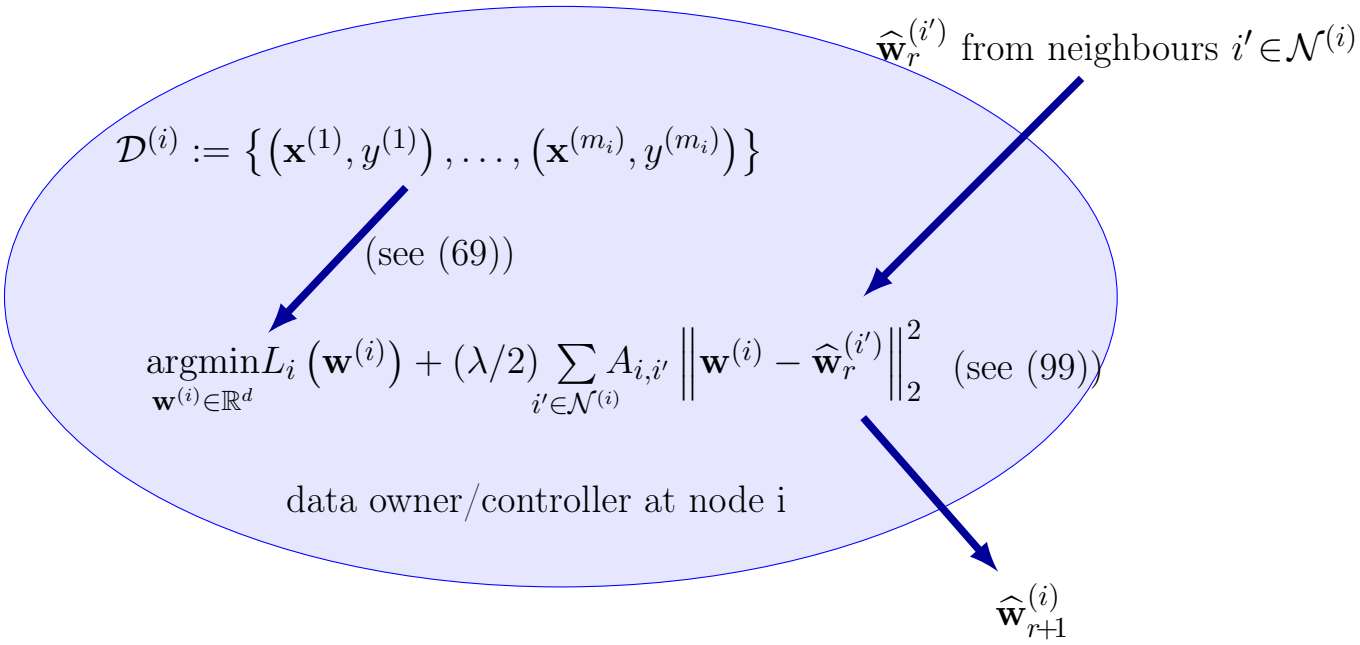

Figure 22: Data flow at node $i \in \mathcal{V}$ during a single iteration of Algorithm 3.

machine learning system (e.g., distributed training, collaborative learning). For a long time, people believed that gradients are safe to share: i.e., the training data will not be leaked by gradient exchange. However, we show that it is possible to obtain the private training data from the publicly shared gradients. We name this leakage as Deep Leakage from Gradient and empirically validate the effectiveness on both computer vision and natural language processing tasks. Experimental results show that our attack is much stronger than previous approaches: the recovery is pixel-wise accurate for images and token-wise matching for texts. We want to raise people's awareness to rethink the gradient's safety. Finally, we discuss several possible strategies to prevent such deep leakage. The most effective defense method is gradient pruning.

12.2 Adding Noise

We can formalize the notion of privacy leakage and privacy protection with a probabilistic model (see Section 3.1). To this end, we interpret data points in the local dataset $\mathcal{D}^{(i)}$ as realizations of RVs. This implies, in turn, that the updates $\widehat{\mathbf{w}}_r^{(i)}$ are realizations of RVs. Moreover, a private feature $x_j^{(r)}$ becomes a realization of a RV.

If the updates $\widehat{\mathbf{w}}_r^{(i)}$ could be used by an adversary to estimate or guess the value of $x_j^{(r)}$, they would leak information about private (sensitive) features. To quantify privacy leakage we can use the concept of mutual information. The mutual information between two RVs measures how much it helps to observe one of these RVs in order to estimate the other RV (see Section 15).

To avoid leakage of sensitive information, which is carried by the private feature $x_j^{(r)}$, we need to ensure a small mutual information $I\left(x_j^{(r)}; \widehat{\mathbf{w}}_r^{(i)}\right)$. This

mutual information is determined by the joint probability distribution of the local datasets $\mathcal{D}^{(i)}$, for $i \in \mathcal{V}$, and the parameters of Algorithm 3. If $I\left(x_j^{(r)}; \widehat{\mathbf{w}}_r^{(i)}\right)$ is too large, we replace the updates $\widehat{\mathbf{w}}_r^{(i)}$ in steps 3 and 4 of Algorithm 3 by the perturbed updates [103, 104]

$$\widetilde{\mathbf{w}}_r^{(i)} := \widehat{\mathbf{w}}_r^{(i)} + \sigma \boldsymbol{\varepsilon}^{(i,r)}. \tag{112}$$

Here, $\boldsymbol{\varepsilon}^{(i,r)}$ are realizations of i.i.d. RVs with common probability distribution $\mathcal{N}(\mathbf{0}, \mathbf{I})$. The parameter σ in (112) controls the amount of privacy protection offered by (112). Increasing the value of σ results in increased privacy protection, however, at the cost of deteriorating accuracy of the hypothesis obtained from $\widetilde{\mathbf{w}}_r^{(i)}$. We summarize the resulting modification of Algorithm 3 in Algorithm 8.

Algorithm 8 FedRelax with Perturbed Updates

Input: empirical graph $\mathcal{G}$; local loss functions $L_i(\cdot)$, GTV parameter λ; privacy protection level σ

Output: learnt local model parameters $\widehat{\mathbf{w}}^{(i)}$

Initialize: $r := 0$; $\widehat{\mathbf{w}}_0^{(i)} := \mathbf{0}$

1: **while** stopping criterion is not satisfied **do**

2: **for** all nodes $i \in \mathcal{V}$ (simultaneously) **do**

3: share local model parameters $\widetilde{\mathbf{w}}_r^{(i)}$ with neighbours $i' \in \mathcal{N}^{(i)}$

4: noisy update of local model parameters

$$\widetilde{\mathbf{w}}_{r+1}^{(i)} := \underset{\mathbf{w}^{(i)} \in \mathbb{R}^d}{\operatorname{argmin}} \left[L_i\left(\mathbf{w}^{(i)}\right) + (\lambda/2) \sum_{i' \in \mathcal{N}^{(i)}} A_{i,i'} \left\| \mathbf{w}^{(i)} - \widetilde{\mathbf{w}}_r^{(i')} \right\|_2^2 \right] + \sigma \boldsymbol{\varepsilon}^{(i,r)}. \quad (113)$$

5: **end for**

6: $r := r+1$

7: **end while**

8: $\widehat{\mathbf{w}}^{(i)} := \widetilde{\mathbf{w}}_r^{(i)}$ for all nodes $i \in \mathcal{V}$

12.3 Feature Perturbation

Let us now describe another modification of Algorithm 3 that protects a private feature $x_j^{(r)}$ of data points in the local dataset $\mathcal{D}^{(i))}$. Instead of perturbing the result of the updates 3 in Algorithm 3, we directly perturb the features of the data points in $\mathcal{D}^{(i)}$. The perturbed features result, in turn, in a perturbed local loss function $\tilde{L}_i(\cdot)$.

To perturb the features of the data points in the local dataset $\mathcal{D}^{(i)}$, we apply a feature map [1, Sec. 9.5]

$$\mathbf{\Phi} : \mathbf{x} \mapsto \mathbf{z} = \mathbf{\Phi}(\mathbf{x}). \tag{114}$$

A privacy-preserving feature map (114) delivers transformed features $\mathbf{z}$ that do not allow to predict (infer) the private feature (too well). However, the new features $\mathbf{z}$ should still allow to learn a hypothesis $h^{(\mathbf{w}^{(i)})}$ that accurately predicts the label of a data point from its privacy-preserving features $\mathbf{z}$. Such a hypothesis can be found via ERM (see Section 3.2), i.e., by minimizing the average loss

$$\tilde{L}_i\left(\mathbf{w}^{(i)}\right) := (1/m_i) \sum_{r=1}^{m_i} L\left(\left(\mathbf{\Phi}\left(\mathbf{x}^{(r)}\right), y^{(r)}\right), h^{(\mathbf{w}^{(i)})}\right). \tag{115}$$

We obtain Algorithm 9 from Algorithm 3 by replacing the local loss function $L_i(\cdot)$ with the perturbed version $\tilde{L}_i(\cdot)$ (115).

Examples for privacy-preserving feature maps (114) that could be used for Algorithm 9 include

- "information obfuscation" by adding noise $\mathbf{\Phi}(\mathbf{x}) = \mathbf{x}$+noise (see [104, Fig. 3])

Algorithm 9 FedRelax with Perturbed Features

Input: empirical graph $\mathcal{G}$; feature map $\mathbf{\Phi}$, local dataset $\mathcal{D}^{(i)}$ for each node $i \in \mathcal{V}$, GTV parameter λ

Output: learnt local model parameters $\widehat{\mathbf{w}}^{(i)}$

Initialize: $r := 0$; $\widehat{\mathbf{w}}_0^{(i)} := \mathbf{0}$

1: **while** stopping criterion is not satisfied **do**

2: **for** all nodes $i \in \mathcal{V}$ (simultaneously) **do**

3: share local parameters $\widehat{\mathbf{w}}_r^{(i)}$ with all neighbours $i' \in \mathcal{N}^{(i)}$

4: update local parameters

$$\widehat{\mathbf{w}}_{r+1}^{(i)} := \operatorname*{argmin}_{\mathbf{w}^{(i)} \in \mathbb{R}^d} \Bigg[(1/m_i) \sum_{r=1}^{m_i} L\left(\left(\mathbf{\Phi}\left(\mathbf{x}^{(r)}\right), y^{(r)} \right), h^{(\mathbf{w}^{(i)})} \right) + (\lambda/2) \sum_{i' \in \mathcal{N}^{(i)}} A_{i,i'} \left\| \mathbf{w}^{(i)} - \widehat{\mathbf{w}}_r^{(i')} \right\|_2^2 \Bigg] \tag{116}$$

5: **end for**

6: $r := r+1$

7: **end while**

8: $\widehat{\mathbf{w}}^{(i)} := \widehat{\mathbf{w}}_r^{(i)}$ for all nodes $i \in \mathcal{V}$

- linear map $\mathbf{\Phi}(\mathbf{x}) = \mathbf{W}\mathbf{x}$ [1, Sec. 9.5] with a matrix $\mathbf{W}$ that is determined by the correlations between private and non-private features. Figure 23 illustrates a toy dataset for which we can find a transformation that perfectly separates the private (gender) from the non-private feature which is used to predict the food preference of a person. One challenge for the use of learnt feature maps is that they need to be shared among all nodes. Indeed, we need use the same features for computing the GTV (75) (or its non-parametric extension (80)).

- the feature map $\mathbf{\Phi}(\mathbf{x})$ is piecewise constant such that each region contains at least a prescribed minimum number k of data points from $\mathcal{D}^{(i)}$. The resulting perturbed feature vectors $\mathbf{z}^{(r)} = \mathbf{\Phi}\big(\mathbf{x}^{(r)}\big)$ are said to provide k-anonymity [105, 106]. Figure 24 illustrates a piece-wise constant feature map that quantizes a feature that would uniquely identify a data point.

12.4 Notes

Q. Li, R. Heusdens and M. G. Christensen, "Privacy-Preserving Distributed Optimization via Subspace Perturbation: A General Framework," in IEEE Transactions on Signal Processing, vol. 68, pp. 5983-5996, 2020, doi: 10.1109/TSP.2020.3029887.

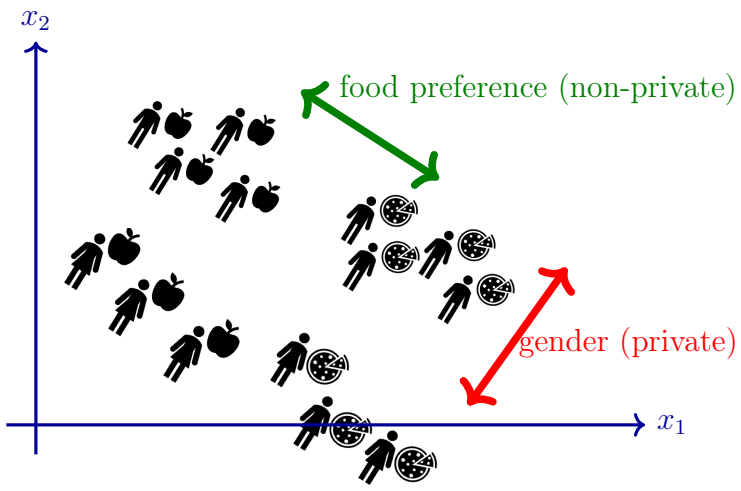

Figure 23: A toy dataset $\mathcal{D}^{(i)}$ whose data points represent persons that are characterized by two features x_1, x_2. These features carry sensitive information (gender) and non-sensitive information (food preference) about a person. The non-sensitive information allows to predicting the quantity of interest (e.g., if a person likes pizza).

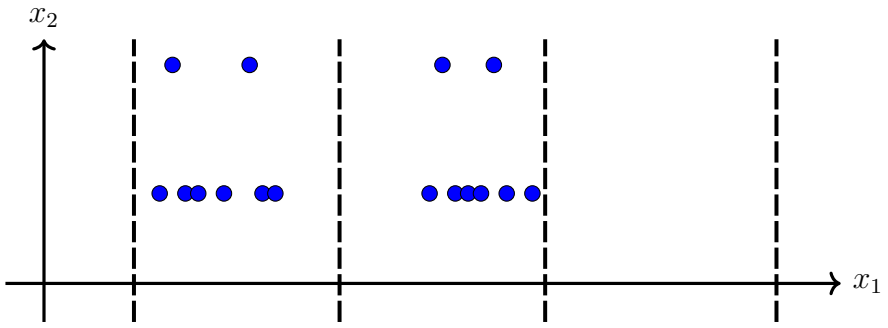

Figure 24: A toy dataset $\mathcal{D}^{(i)}$ whose data points are characterized by two features: x_1 and x_2. The feature x_2 is private and carries sensitive information. The other feature x_1 is non-private but uniquely identifies a data point: two different data points in $\mathcal{D}^{(i)}$ have different values of x_1. One simple approach to protect the private property x_1 is to quantize the feature x_1 resulting in a perturbed feature $\tilde{x}_1$ that is the same for at least k data points, thereby ensuring k-anonymity.

12.5 Exercises

Exercise 12.1. Privacy-Preserving Feature Learning (5 points) Consider the toy dataset `https://scikit-learn.org/stable/datasets/toy_dataset.html#diabetes-dataset` whose data points are patients. Each data point is characterized by $d = 10$ features $\mathbf{x} = \big(x_1, \ldots, x_d\big)^T$ and a label y which measures disease progression. The private feature x_1 characterizes the age of a person. We could simply remove this feature to obtain a privacy-preserving feature vector $\mathbf{x}'' = \big(x_2, \ldots, x_d\big)^T \in \mathbb{R}^d$. This exercise explores how much better we could do by learning a privacy-preserving feature map $\boldsymbol{\Phi}$. Construct a feature map such that the transformed features do not allow to accurately predict the age of the person. Choose on ML method from `scikit-learn` and compare the achievable performance when using the transformed features $\mathbf{x}' = \boldsymbol{\Phi}(\mathbf{x})$ with using the features $\mathbf{x}''$.

Exercise 12.2. Privacy-Preserving Feature Learning Using Decision Trees Consider the toy dataset $\mathcal{D}^{(\text{diabetes})}$, available at `https://scikit-learn.org/stable/datasets/toy_dataset.html#diabetes-dataset`, whose data points are patients. Each data point is characterized by $d = 10$ features and a label y which measures disease progression. This exercise only uses three features, denotes x_1, x_2, x_3 which are the `sex`, and blood serum measurements `s1`, `s2`, respectively.

We consider the feature x_1 (`sex`) as the private feature that should be protected as early (as close to the actual data storage location) as possible within the FL system. One approach to protect private information contained in raw features is to transform them into new features. The transformation should be such that the new features maximally conserve the relevant information for predicting the label y but carry only little information about the private feature x_1.

Your task is to construct a feature map ϕ that reads in the two features x_2, x_3 (`s1` and `s2` blood serum measurements of a patient) and delivers a new privacy-preserving feature $x' = \phi(x_2, x_3)$ such that:

- The new feature x' takes on values in a feature space $\mathcal{X}'$ of size 6.
- It is not possible to find a hypothesis map $g(x')$ that would allow to predict (solely from the new feature x') the private feature x_1 with classification error strictly smaller than 0.4. We measure the classification error by the fraction of data points in $\mathcal{D}^{(\text{diabetes})}$ for which $g(x') \neq x_1$.
- It is possible to find (or learn) a hypothesis map $h(x')$ that allows to predict the label y with relative average squared error loss smaller than

0.2. The relative average squared error loss of a hypothesis is defined as $\sum_{r=1}^{m} \left(h(\phi\big(x_2^{(r)}, x_3^{(r)}\big)\right) - y^{(r)}\big)^2 / \sum_{r=1}^{m} \left(y^{(r)}\right)^2$.

13 Data Poisoning

The FL algorithms of Section 10 pool the information contained in decentralized local datasets to train local (tailored) models (see Section 7.2). The benefit of information sharing, allowing to learn more accurate hypotheses, comes at the cost of increased vulnerability against data poisoning.

Data poisoning refers to the intentional manipulation (or fabrication) of local datasets to steer the training of a specific local model [107, 108]. We discuss two main flavours of data poisoning, known as a denial-of-service attack and a backdoor attack. Section 13.1 discusses how a denial-of-service attack manipulates local model training such that it performs poorly on its own local dataset [109]. Section 13.2 explains how backdoor attacks maliciously steer the training of a local model such that it performs well in general but anomalously for data points with specific features [110].

13.1 Denial-of-Service Attacks

Consider decentralized data that is represented by an empirical graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ whose nodes $i \in \mathcal{V}$ carry local datasets $\mathcal{D}^{(i)}$. We solve GTVMin to learn a hypothesis $h^{(i)}$ for some node $i \in \mathcal{V}$. Assume that some adversary controls the local datasets at a set $\mathcal{A} \subseteq \mathcal{V}$ of nodes.

13.2 Backdoor Attack

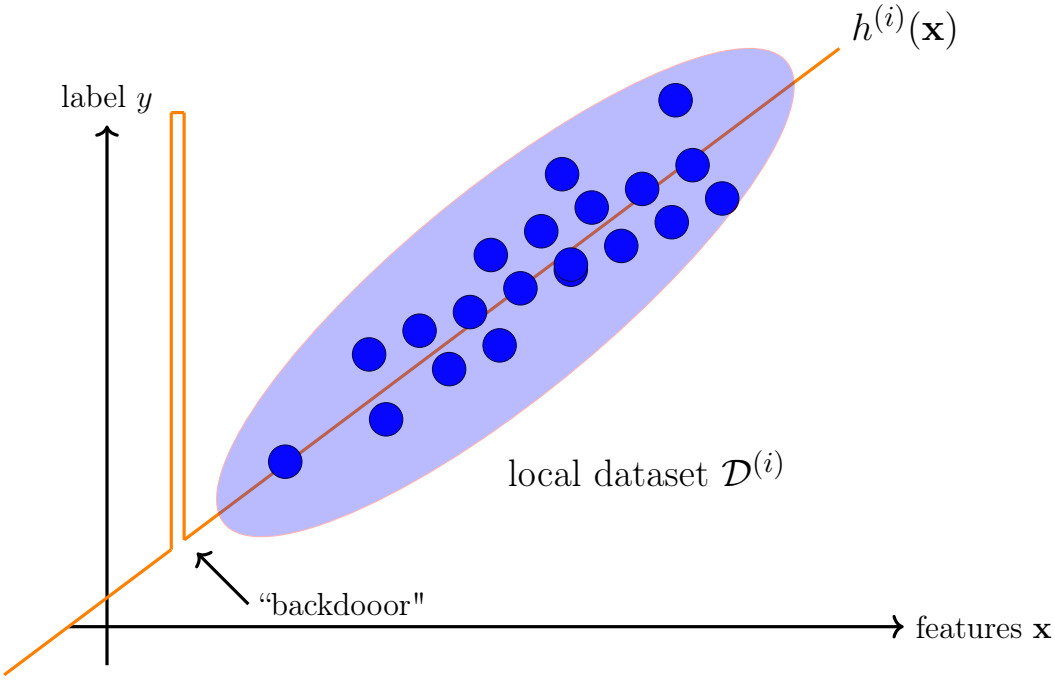

Figure 25: A backdoor attack aims at nudging the local hypothesis $h^{(i)}$ to behave in a specific way for a certain range of feature values. This range of feature values serves as the key or trigger that unlocks a "backdoor".

13.3 Exercises

Exercise 13.1. Backdoor Attack (5 points) Consider the toy dataset `https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_digits.html#sklearn.datasets.load_digits` whose data points are images of hand-written digits. We use the first $m' = 100$ data points as a validation set. Construct a training set for a decision tree classifier such that the trained classifier achieves an accuracy at least 0.9 on the validation set but is also forced to classify the 104th data point as any prescribed digit.

Exercise 13.2. Poisoning Iris Dataset. Consider the toy dataset $\mathcal{D}^{(\text{iris})}$ described in `https://scikit-learn.org/stable/datasets/toy_dataset.html#iris-dataset`. Each data point $\left(\mathbf{x}^{(r)}, y^{(r)}\right)$ in $\mathcal{D}^{(\text{iris})}$ represents an iris plant and is characterized by four numeric features $\mathbf{x}^{(r)} = \left(x_1^{(r)}, \ldots, x_4^{(r)}\right)^T$ and label $y^{(r)} \in \{\text{“setosa”}, \text{“versicolor”}, \text{“virginica”}\}$. The features are the sepal length in cm (x_1), sepal width in cm (x_2), petal length in cm (x_3) and petal width in cm (x_4).

Your task is to poison $\mathcal{D}^{(\text{iris})}$ by augmenting it with carefully chosen data points. The resulting augmented dataset $\mathcal{D}'$ is then used as a training set for ERM using a decision tree. We want the learnt hypothesis $\hat{h}$ to have the following properties:

- $\hat{h}(x) =$ “virginica” for any data point with $x_2 = 3$ and $x_3 = 1.2$.
- $\hat{h}(x) =$ “setosa” for any data point with $x_2 = 2.7$ and $x_3 = 5.5$.
- the fraction of incorrectly classified data points, where $\hat{h}(\mathbf{x}^{(r)}) \neq y^{(r)}$, of the original dataset $\mathcal{D}^{(\text{iris})}$ is as small as possible.

Part IV

Appendix

14 Linear Algebra and Convex Analysis

The main data structure of the FL methods discussed in part II are numeric arrays such as vectors or matrices [111]. Numeric arrays can be combined via algebraic operations such as (element-wise) addition $\mathbf{x} + \mathbf{y}$ or multiplication $\mathbf{Zx}$ of compatible arrays (e.g., $\mathbf{Z} \in \mathbb{R}^{10\times 3}$ and $\mathbf{x} \in \mathbb{R}^3$). These operations are the most important building blocks for most FL methods discussed in part II.

14.1 Convex Sets

Linear spaces are somewhat impractical as they are infinitely large. Using finite computational resources it would by useful to have a modification of linear spaces with finite extension. It turns out that convex sets pretty much fit this bill. These sets have many (algebraic and topological) properties in common with linear spaces. Moreover, many useful convex sets have a finite extension.

14.2 Convex Functions

Consider some real-valued function f whose domain is s subset of $\mathbb{R}^d$. The function is fully specified by its epigraph. We can then define convex functions as those functions whose epigraph is a convex set [].

14.3 Proximal Operators

Given a closed proper convex function $f(\mathbf{x})$ with domain being a subset of $\mathbb{R}^d$, we define its associated convex conjugate function as [29]

$$f^*(\mathbf{x}) := \sup_{\mathbf{z}\in\mathbb{R}^d} \mathbf{x}^T\mathbf{z} - f(\mathbf{z}). \tag{117}$$

The proximity operator associated with a closed proper convex function $f(\mathbf{x})$ is defined as [112]

$$\mathbf{prox}_{f,\rho}(\mathbf{x}) := \operatorname*{argmin}_{\mathbf{x}'} f(\mathbf{x}') + (\rho/2)\|\mathbf{x}-\mathbf{x}'\|_2^2 \text{ for some } \rho > 0. \tag{118}$$

Note that the minimum in (118) exists and is unique since the objective function is strongly convex [29] .

15 Information Theory

It is often useful to interpret data points as realizations of RVs. The behaviour of ML algorithms that use these data points can then be studied using the relations between these RVs. A powerful tool to study relations between RVs is the concept of mutual information [113, 114]. The mutual information between two RVs $\mathbf{x}, y$ with joint probability distribution $p(\mathbf{x}, y)$ is defined as

$$I(\mathbf{x}; y) := \mathbb{E}\left\{\log p(\mathbf{x}, y)/(p(\mathbf{x})p(y))\right\} \tag{119}$$

16 Fixed Point Theory

Glossary

0/1 loss The 0/1 loss $L\left((\mathbf{x}, y), h\right)$ measures the quality of a classifier $h(\mathbf{x})$ that delivers a prediction $\hat{y}$ (e.g., via thresholding (120)) for a label y of a data point. It is equal to 0 if the prediction is correct, i.e., $L\left((\mathbf{x}, y), h\right) = 0$ when $\hat{y} = y$. It is equal to 1 if the prediction is wrong, $L\left((\mathbf{x}, y), h\right) = 1$ when $\hat{y} \neq y$. 36, 37, 42–44, 46, 47, 168, 190

k-means The k-means algorithm is a hard clustering method which assigns each data points to precisely one out of k different clusters. The method iteratively updates this assignment in order to minimize the average distance between data points in their nearest cluster mean (centre). 106, 171, 179, 200

accuracy Consider data points characterized by features $\mathbf{x} \in \mathcal{X}$ and a categorical label y which takes on values from a finite label space $\mathcal{Y}$. The accuracy of a hypothesis $h : \mathcal{X} \to \mathcal{Y}$, when applied to the data points in a dataset $\mathcal{D} = \left\{\left(\mathbf{x}^{(1)}, y^{(1)}\right), \ldots, \left(\mathbf{x}^{(m)}, y^{(m)}\right)\right\}$ is then defined as $1 - (1/m) \sum_{r=1}^{m} L\left(\left(\mathbf{x}^{(r)}, y^{(r)}\right), h\right)$ using the 0/1 loss. 36, 43, 164

activation function Each artificial neuron within an ANN consists of an activation function that maps the inputs of the neuron to a single output value. In general, an activation function is a non-linear map of the weighted sum of neuron inputs (this weighted sum is the activation of the neuron). 15

artificial intelligence Artificial intelligence aims to develop systems that behave rational in the sense of maximizing a long-term reward. 22, 145

artificial neural network An artificial neural network is a graphical (signal-flow) representation of a map from features of a data point at its input to a predicted label at its output. 15, 27, 28, 30–33, 58, 65, 74, 96, 123–125, 133, 168, 175, 182, 193, 200

backdoor A backdoor attack refers to the intentional manipulation of the training process underlying a ML method. The manipulation can be implemented by perturbing the training set (data poisoning) or optimization algorithm used by an ERM based method. The goal of a backdoor attack is to nudge the learnt hypothesis towards specific predictions for a certain range of feature values. This range of feature values serves as a key (or trigger) to unlock a "backdoor" in the sense of delivering anomolous predictions. These anomouls predictions are known to the attacker but cannot be inferred (easily) from the clean training set or optimization algorithm. 162, 163

batch A set of randomly selected data points out of a (typically very large) dataset. 90, 131

Bayes estimator A hypothesis h whose Bayes risk is minimal [44]. 50, 51, 53, 169

Bayes risk We use the term Bayes risk as a synonym for the risk or expected loss of a hypothesis. Some authors reserve the term Bayes risk for the risk of a hypothesis that achieves minimum risk, such a hypothesis being referred to as a Bayes estimator [44]. 169

bias Consider some unknown quantity $\bar{w}$, e.g., the true weight in a linear

model $y = \bar{w}x + e$ relating feature and label of a data point. We might use an ML method (e.g., based on ERM) to compute an estimate $\hat{w}$ for the $\bar{w}$ based on a set of data points that are realizations of RVs. The (squared) bias incurred by the estimate $\hat{w}$ is typically defined as $B^2 := \big(\mathbb{E}\{\hat{w}\} - \bar{w}\big)^2$. We extend this definition to vector-valued quantities using the squared Euclidean norm $B^2 := \big\|\mathbb{E}\{\widehat{\mathbf{w}}\} - \overline{\mathbf{w}}\big\|_2^2$. 14, 56, 57, 84

classification Classification is the task of determining a discrete-valued label y of a data point based solely on its features $\mathbf{x}$. The label y belongs to a finite set, such as $y \in \{-1, 1\}$, or $y \in \{1, \ldots, 19\}$ and represents a category to which the corresponding data point belongs to. 13, 27, 37, 40, 43–46, 48, 114, 124, 160

classifier A classifier is a hypothesis (map) $h(\mathbf{x})$ that is used to predict a discrete-valued label. Strictly speaking, a classifier is a hypothesis $h(\mathbf{x})$ that delivers values from a finite set $\mathcal{Y}$ (referred to as the label space). However, we are sometimes sloppy and use the term classifier also for a hypothesis that delivers a real number which is quantized, i.e., compared against a threshold, to obtain the predicted label value. For example, in a binary classification problem with label values $y \in \{-1, 1\}$, we refer to a hypothesis $h(\mathbf{x})$ as classifier if it is used to compute a predicted label according to

$$\hat{y} = 1 \text{ for } h(\mathbf{x}) \geq 0, \text{ and } \hat{y} = -1 \text{ otherwise.} \tag{120}$$

36, 37, 40, 41, 46, 47, 164, 168

cluster A cluster within an empirical graph is a subset of nodes that carry local datasets with (nearly) identical statistical distributions. Clustered FL aims at identying the clusters and use the pooled local darasets to train a separate model for each cluster. 103

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 notion and measure of similarity between data points is a design choice. If data points are characterized by numeric feature vectors in some Euclidean space we can define the similarity between two data points via the Euclidean distance between their feature vectors. 97, 98, 103, 104, 110, 112, 121, 123, 124

clustering Clustering methods decompose a given set of data points into 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 representation of clusters. The clustering method k-means uses the average feature vector ("cluster means") of a cluster as its representative. A popular soft-clustering method based on Gaussian mixture model (GMM) represents a cluster by a multivariate normal distribution. 100, 106, 112, 168

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 with each other than with those outside the cluster [94]. We obtain different clustering methods

by using different notions of similarity between data points. 96, 123

computational aspects By computational aspects of a ML method, we mainly refer to the computational resources required for its implementation. For example, if a ML method uses iterative optimization technniques to solve ERM, then its computational aspects include (i) how many arithmetic operations are needed to implement a single iteration (gradient step) and (ii) how many iterations are needed to obtain useful model parameters. One important example for an iterative optimization technique is GD. 26, 29–31, 35–37, 52, 53, 84, 99–101, 109, 114, 115, 187, 202

convex A set $\mathcal{C} \subseteq \mathbb{R}^d$ is convex if it contains the line segment between any two points of that set. We define a function as convex if its epigraph is a convex set [29]. 36, 37, 39, 40, 43–45, 109, 167, 196

covariance matrix The covariance matrix of a RV $\mathbf{x} \in \mathbb{R}^d$ is defined as $\mathbb{E}\Big\{\big(\mathbf{x} - \mathbb{E}\{\mathbf{x}\}\big)\big(\mathbf{x} - \mathbb{E}\{\mathbf{x}\}\big)^T\Big\}$. 11, 68, 83, 116, 179, 191

data A (indexed) set of data points. 18, 21, 26, 62, 195

data augmentation Data augmentation methods add synthetic data points to an existing set of data points. These synthetic data points might be obtained by perturbations (adding noise) or transformations (rotations of images) of the original data points. 66–70, 72

data point A data point is any object that conveys information [113]. Data points might be students, radio signals, trees, forests, images, RVs, real

numbers or proteins. We characterize data points using two types of properties. One type of property is referred to as a feature. Features are properties of a data point that can be measured or computed in an automated fashion. Another type of property is referred to as labels. The label of a data point represents some higher-level fact (or quantity of interest). In contrast to features, determining the label of a data point typically requires human experts (domain experts). Roughly speaking, ML aims at predicting the label of a data point based solely on its features. 7, 11–14, 16, 26–32, 34–36, 38–43, 46–57, 59–69, 72–74, 81, 82, 86, 87, 89–91, 96, 97, 102, 106, 112, 113, 115, 117, 121, 124, 126, 127, 130, 136, 143, 147–149, 151, 154, 156–160, 162, 164, 167–179, 181–190, 192–194, 197–204

data poisoning FL methods allow to leverage the information contained in local datasets generated by other parties to improve the training of a tailored model. Depending on how much we trust the other parties, FL can be compromised by data poisoning. Data poisoning refers to the intentional manipulation (or fabrication) of local datasets to steer the training of a specific local model [107, 108]. 18, 162, 175

dataset With a slight abuse of notation we use the terms "dataset" or "set of data points" to refer to an indexed list of data points $\mathbf{z}^{(1)}, \mathbf{z}^{(2)}, \ldots$. Thus, there is a first data point $\mathbf{z}^{(1)}$, a second data point $\mathbf{z}^{(2)}$ and so on. Strictly speaking a dataset is a list and not a set [115]. By using indexed lists of data points we avoid some of the challenges arising in concept of an abstract set. 12, 16, 22, 27, 41, 47, 49, 51, 52, 59, 62, 63,

100, 108, 113, 124, 164

denial-of-service attack A denial-of-service attack uses data poisoning to steer the training of a local (client) model such that it performs poorly for typical data points 162

differentiable A function $f : \mathbb{R}^d \to \mathbb{R}$ is differentiable if it has a gradient $\nabla f(\mathbf{x})$ everywhere (for every $\mathbf{x} \in \mathbb{R}^d$) [5]. 15, 36, 37, 39, 40, 43–45, 74, 76, 93, 101, 129, 137, 180, 181, 199, 201

dimensionality reduction Dimensionality reduction methods map (typically many) raw features to a (relatively small) set of new features. These methods can be used to visualize data points by learning two features that can be used as the coordinates of a depication in a scatterplot. 100, 124

discrepancy Consider a FL application with networked data represented by an empirical graph. FL methods use a discrepancy measure to compare hypothesis maps from local models at nodes i, i' connected by an edge in the empirical graph. 26, 34, 62, 108, 113–115, 124, 134–136, 146

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 number of "independent" tunable parameters of the model. These parameters might be the coefficients used in a linear map or the weights and bias terms of an ANN. 56, 58, 64–66, 181

eigenvalue We refer to a number $\lambda \in \mathbb{R}$ as eigenvalue of a square matrix

$\mathbf{A} \in \mathbb{R}^{d \times d}$ if there is a non-zero vector $\mathbf{x} \in \mathbb{R}^d \setminus \{\mathbf{0}\}$ such that $\mathbf{A}\mathbf{x} = \lambda \mathbf{x}$. 15, 53, 79, 80, 82, 84, 86, 176

eigenvector An eigenvector of a matrix $\mathbf{A}$ is a non-zero vector $\mathbf{x} \in \mathbb{R}^d \setminus \{\mathbf{0}\}$ such that $\mathbf{A}\mathbf{x} = \lambda \mathbf{x}$ with some eigenvalue λ. 200

empirical graph Empirical graphs represent collections of local datasets and corresponding local models [94]. An empirical graph is an undirected weighted empirical graph whose nodes carry local datasets and models. FL methods learn a local hypothesis $h^{(i)}$, for each node $i \in \mathcal{V}$, such that it incurs small loss on the local datasets. 16, 21–23, 70, 96–99, 103–108, 111, 112, 114–122, 128, 129, 131, 133, 135, 137, 141, 153, 162, 171, 175, 185, 188, 192

empirical risk The empirical risk of a given hypothesis on a given set of data points is the average loss of the hypothesis computed over all data points in that set. 14, 51–55, 62, 68, 70, 71, 74, 75, 94, 108, 176, 192, 193, 201–203

empirical risk minimization Empirical risk minimization is the optimization problem of finding the hypothesis with minimum average loss (or empirical risk) on a given set of data points (the training set). Many ML methods are special cases of empirical risk. 21, 49, 52–55, 57, 59, 60, 62, 64–66, 69, 72, 74, 75, 81–87, 92–94, 96, 102, 107, 134, 136, 154, 164, 169, 170, 172, 181, 190, 193, 197, 202, 203

Euclidean space The Euclidean space $\mathbb{R}^d$ of dimension d refers to the space of all vectors $\mathbf{x} = \big(x_1, \ldots, x_d\big)$, with real-valued entries $x_1, \ldots, x_d \in \mathbb{R}$,

whose geometry is defined by the inner product $\mathbf{x}^T\mathbf{x}' = \sum_{j=1}^{d} x_j x'_j$ between any two vectors $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^d$ [5]. 19, 27, 100, 171, 178, 187, 197

expectation Consider a numeric RV $\mathbf{x}$ with probability distribution $p(\mathbf{x})$. The expectation of $\mathbf{x}$ is defined as the integral $\mathbb{E}\{\mathbf{x}\} := \int \mathbf{x} p(\mathbf{x})$ (if this integral is well-defined!) [5, 40, 116]. 56

explainability We can define the (subjective) explainability of a ML method as the level of predictability (or lack of uncertainty) in its predictions delivered to a specific human user. Quantitative measures for the (subjective) explainability can be obtained via probabilistic models for the data fed into the ML method. In particular, we can then identify explainability of a ML method via the uncertainty or entropy of the predictions it delivers [37, 117, 118]. 29, 30, 36, 37

feature A feature of a data point is one of its properties that can be measured or computed in an automated fashion. For example, if a data point is a bitmap image, then we could use the red-green-blue intensities of its pixels as features. Some widely used synonyms for the term feature are "covariate","explanatory variable", "independent variable", "input (variable)", "predictor (variable)" or "regressor" [119–121]. However, this book makes consequent use of the term features for low-level properties of data points that can be measured easily. 7, 11–16, 26–35, 38, 39, 43, 47, 48, 51, 53, 55–57, 59, 60, 64, 65, 67, 69, 72–74, 80–83, 86, 87, 96, 97, 106, 107, 112, 115, 121, 124, 126, 127, 130, 131, 136, 143, 149, 151, 154, 156, 157, 159, 160, 162–164, 168, 169, 173, 175, 178, 179, 181, 182, 184, 186–190, 193, 194, 197–201, 204

feature map A map that transforms the original features of a data point into new features. The so-obtained new features might be preferable over the original features for several reasons. For example, the shape of datasets might become simpler in the new feature space, allowing to use linear models in the new features. Another reason could be that the number of new features is much smaller which is preferable in terms of avoiding overfitting. The special case of feature maps that deliver two numeric features are particulary useful for data visualization. Indeed, we can then depict data points in a scatterplot by using these two features as the coordinates of a data point. 15, 32, 33, 48, 82, 83, 154–156, 159

feature matrix Consider a dataset $\mathcal{D}$ of m data points that are characterized by feature vectors $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)}$. It is convenient to collect these feature vectors into a feature matrix $\mathbf{X} := \big(\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)}\big)$. 79

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. Within this book the most frequently used choice for the feature space is the Euclidean space $\mathbb{R}^d$ with dimension d being the number of individual features of a data point. 12, 15, 27, 31, 115, 116, 160, 183, 184

federated averaging (FedAvg) Federated averaging is an iterative FL algorithm that alternates between local model trainings and averaging the resulting local model parameters. Different variants of this algorithm are obtained by different techniques for the model training. The authors

of [14] consider federated averaging methods where the local model training is implemented by running several GD steps 138

federated learning (FL) Federated learning is an umbrella term for ML methods that train models in a collaborative fashion using decentralized data and computation. 1, 17–23, 27, 30, 62, 69–71, 84, 99–101, 104, 108, 109, 111, 115, 118, 119, 121–128, 134, 137, 142, 145, 146, 148, 149, 160, 162, 166, 171, 173, 175, 176, 178, 182, 194, 204

Finnish Meteorological Institute The Finnish Meteorological Institute is a government agency responsible for gathering and reporting weather data in Finland. 106

flow-based clustering Flow-basted clustering groups the nodes of an undirected graph by applying k-means clustering to node-wise feature vectors. These feature vectors are built from flows between carefully selected source and destination nodes [82]. 112

Gaussian mixture model Gaussian mixture models (GMM) are a family of probabilistic models for data points characterized by a numeric feature vector $\mathbf{x}$. A GMM interprets $\mathbf{x}$ as being drawn from one out of k different multivariate normal distributions $p^{(c)} = \mathcal{N}\left(\boldsymbol{\mu}^{(c)}, \mathbf{C}^{(c)}\right)$, indexed by $c = 1, \ldots, k$. The probability that $\mathbf{x}$ is drawn from the c-th multivariate normal distribution is denoted p_c. Thus, a GMM is parametrized by the probability p_c, the mean vector $\boldsymbol{\mu}^{(c)}$ and covariance matrix $\boldsymbol{\Sigma}^{(c)}$ for each $c = 1, \ldots, k$. 171

Gaussian random variable A Gaussian RV $\mathbf{x} \in \mathbb{R}^d$ with a multivariate

128–130, 137, 138, 172, 196

gradient-based method Gradient-based methods are iterative algorithms for finding the minimum (or maximum) of a differentiable objective function of the model parameters. These algorithms construct a sequence of approximations to an optimal choice for model parameters that results in a minimum objective function value. As their name indicates, gradient-based methods use the gradients of the objective function evaluated during previous iterations to construct new (hopefully) improved model parameters. 13, 19, 36, 37, 45, 74, 101, 128

graph A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is a pair that consists of a node set $\mathcal{V}$ and an edge set $\mathcal{E}$. In general, a graph is specified by a map that assigns to each edge $e \in \mathcal{E}$ a pair of nodes [122]. One important family of graphs (simple undirected graphs) is obtained by identifying each edge $e \in \mathcal{E}$ with two different nodes $\{i, i'\}$. Weighted graphs also specify numeric weights A_e for each edge $e \in \mathcal{E}$. 118, 176

GTV minimization GTV minimization is an instance of RERM using the GTV of local model parameters as a regularizer. 21–23, 108, 110–114, 119, 121–123, 125, 128, 129, 132, 133, 143, 146, 147, 162

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 for ML methods that operate in the high-dimensional regime is deep learning with large ANNs, having more tunable weights (and bias terms) than the 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 [47, 123]. 102

hinge loss Consider a data point that is 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 specific hypothesis h is defined as

$$L\left((\mathbf{x}, y), h\right) := \max\{0, 1 - yh(\mathbf{x})\}. \tag{121}$$

A regularized variant of the hinge loss is used by the support vector machine (SVM) [124] to learn a linear classifier with maximum margin between the two classes (see Figure 26). 44, 45, 183, 202

horizontal FL Horizontal FL refers to applications with local datasets that are constituted by different data points but using the same features to characterize them [**?**]. One example for horizontal FL is numerical weather prediction using a network of 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 spatio-temporal regions (each such region being a separate data point). 121, 126, 127

Huber loss The Huber loss is a mixture of the squared error loss and the absolute value of the prediction error. 39

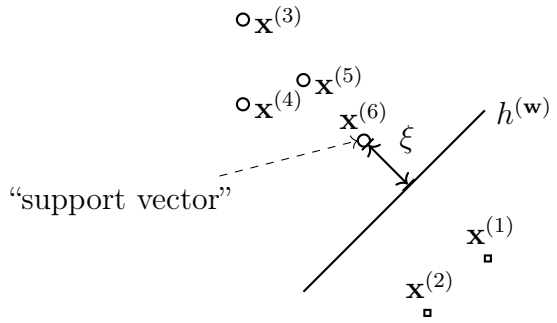

Figure 26: The SVM learns a hypothesis (or classifier) $h^{(\mathbf{w})}$ with minimum average soft-margin hinge loss. Minimizing this loss is equivalent to maximizing the margin ξ between the decision boundary of $h^{(\mathbf{w})}$ and each class of the training set.

hypothesis A map (or function) $h : \mathcal{X} \to \mathcal{Y}$ 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 predicted label $\hat{y} = h(\mathbf{x})$. ML is about learning (or finding) a hypothesis map h such that $y \approx h(\mathbf{x})$ for any data point. 7, 13–16, 26, 29–31, 34–45, 47–51, 53–57, 59–68, 73, 74, 83, 84, 94, 99–104, 113, 114, 119, 124, 129, 131, 134–136, 143, 149, 152, 160–164, 168–170, 174–176, 182, 186–194, 197, 198, 200–203

hypothesis space Every practical ML method uses a specific hypothesis space (or model) $\mathcal{H}$. The hypothesis space of a ML method is a subset of all possible maps from the feature space to label space. The design choice of the hypothesis space should take into account available computational resources and statistical aspects. If the computational

infrastructure allows for efficient matrix operations, and there is a (approximately) linear relation between features and label, a useful choice for the hypothesis space might be the linear model. 13, 26, 29–36, 47–49, 52–55, 59, 60, 63, 64, 67, 74, 93, 96, 99, 113, 114, 175, 187, 188, 191–193, 198, 202, 203

i.i.d. It can be useful to interpret data points $\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(m)}$ as realizations of independent and identically distributed RVs with a common probability distribution. If these RVs are continuous, their joint probability density function (pdf) is $p\big(\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(m)}\big) = \prod_{r=1}^{m} p\big(\mathbf{z}^{(r)}\big)$ with $p(\mathbf{z})$ being the common marginal pdf of the underlying RVs. 18, 43, 49, 51, 55, 56, 60, 63, 64, 68, 73, 83, 89, 97, 115, 152, 184, 186, 190, 192, 198, 199

i.i.d. assumption The i.i.d. assumption interprets data points of a dataset as the realizations of i.i.d. RVs. 49, 50, 52, 53, 55, 91, 103, 115, 192, 198

interpretability A ML method is interpretable for a specific user if she can well anticiapte the predictions delivered by the method. The notion of interpretability can be made precise using quantitative measures of the uncertainty about the predictions [37]. 31, 187

kernel Consider data points characterized by features $\mathbf{x} \in \mathcal{X}$ with values in some arbirtrary feature space. A kernel is a map that assigns each pair of feature values $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$ a real number. This number measures the similarity between $\mathbf{x}$ and $\mathbf{x}'$. For more details about kernels and the resulting kernel methods, we refer to the literature [124, 125]. 32

Kullback-Leibler divergence The Kullback–Leibler divergence is a quantitive measure for how much one probability distribution is different from another probability distribution [113]. 116, 117

label A higher level fact or quantity of interest associated with a data point. If a data point is an image, its label might be the fact that it shows a cat (or not). Some widely used synonyms for the term label are "response variable", "output variable" or "target" [119–121]. 7, 11, 13, 16, 26–30, 34, 38, 39, 41, 42, 47, 51–57, 60, 65, 67, 69, 72, 73, 77, 87, 97, 115, 126, 130, 131, 136, 149, 154, 159, 160, 168, 170, 173, 182, 184–190, 193, 194, 197, 200, 202

label space Consider a ML application that involves data points characterized by features and labels. The label space is constituted by all potential values that the label of a data point can take on. Regression methods, aiming at predicting numeric labels, often use the label space $\mathcal{Y} = \mathbb{R}$. Binary classification methods use a label space that consists of two different elements, e.g., $\mathcal{Y} = \{-1, 1\}$, $\mathcal{Y} = \{0, 1\}$ or $\mathcal{Y} = \{$“cat image”,”no cat image”$\}$ 27, 31, 40, 42, 46–48, 115, 116, 168, 170, 183, 189

Laplacian matrix The geometry or structure of a graph $\mathcal{G}$ can be analyzed using the properties of special matrices that are associated with $\mathcal{G}$. One such matrix is the graph Laplacian matrix $\mathbf{L}$ which is defined for an undirected and weighted graph (e.g., the empirical graph of networked data) [81, 87]. 119

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 (or expectation) of their common probability distribution. Different instances of the law of large numbers are obtained using different notions of convergence. 43, 51, 52

learning rate Consider an iterative method for finding or learning a good choice for a hypothesis. Such an iterative method repeats similar computational (update) steps that adjust or modify the current choice for the hypothesis to obtain an improved hypothesis. A prime example for such an iterative learning method is GD and its variants. We refer by learning rate to any parameter of an iterative learning method that controls the extent by which the current hypothesis might be modified or improved in each iteration. A prime example for such a parameter is the step size used in GD. Some authors use the term learning rate mostly as a synonym for the step size of (a variant of) GD 13, 77–82, 85, 86, 90, 92, 94, 129, 131

learning task A learning tasks consists of a specific choice for a collection of data points (e.g., all images stored in a particular database), their features and labels. 14, 18, 27, 28, 70, 111, 123, 124

least absolute shrinkage and selection operator (Lasso) The least absolute shrinkage and selection operator (Lasso) is an instance of structural risk minimization (SRM) for learning the weights $\mathbf{w}$ of a linear map $h(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$. The Lasso minimizes the sum consisting of an average squared error loss (as in linear regression) and the scaled ℓ_1 norm of the

weight vector $\mathbf{w}$. 67, 102, 198

linear classifier A classifier $h(\mathbf{x})$ maps the feature vector $\mathbf{x} \in \mathbb{R}^d$ of a data point to a predicted label $\hat{y} \in \mathcal{Y}$ out of a finite set of label values $\mathcal{Y}$. We can characterize such a classifier equivalently by the decision regions $\mathcal{R}_a$, for every possible label value $a \in \mathcal{Y}$. Linear classifiers are such that the boundaries between the regions $\mathcal{R}_a$ are hyperplanes in the Euclidean space $\mathbb{R}^d$. 182

linear model This book uses the term linear model in a very specific sense. In particular, a linear model is a hypothesis space which consists of all linear maps,

$$\mathcal{H}^{(d)} := \left\{ h(\mathbf{x}) = \mathbf{w}^T \mathbf{x} : \mathbf{w} \in \mathbb{R}^d \right\}. \tag{122}$$

Note that (122) defines an entire family of hypothesis spaces, which is parametrized by the number d of features that are linearly combined to form the prediction $h(\mathbf{x})$. The design choice of d is guided by computational aspects (smaller d means less computation), statistical aspects (increasing d might reduce prediction error) and interpretability (a linear model using few carefully chosen features might be considered interpretable). 30–33, 35, 53, 55–58, 64, 69, 72, 73, 96, 99, 100, 102, 129, 138, 169, 184

linear regression Linear regression aims at learning a linear hypothesis map to predict a numeric label based on numeric features of a data point. The quality of a linear hypothesis map is measured using the average squared error loss incurred on a set of labeled data points (which we

refer to as training set). 17, 53, 59, 68, 72, 74, 75, 83, 86, 87, 102, 133, 138, 140, 181, 186

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 which are 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 similarities. These similarities might arise from probabilistic models or communication infrastructure and are encoded in the edges of an empirical graph. 1, 16–18, 20, 22, 70, 96–104, 106–108, 111, 112, 115, 117–124, 126–129, 135–138, 143, 147–149, 151, 152, 154, 162, 173, 176, 182, 188, 192, 194, 204

local model Consider a collections of local datasets that are assigned to the nodes of an empirical graph. A local model $\mathcal{H}^{(i)}$ is a hypothesis space that is assigned to a node $i \in \mathcal{V}$. Different nodes might be assigned different hypothesis spaces, i.e., in general $\mathcal{H}^{(i)} \neq \mathcal{H}^{(i')}$ for different nodes $i, i' \in \mathcal{V}$. 1, 16, 19, 20, 62, 69, 84, 96–99, 101, 103, 104, 107, 108, 110–114, 119, 121, 123, 124, 128, 133–135, 137, 147–149, 173, 176, 192

logistic loss Consider a data point that is characterized by the features $\mathbf{x}$ and a binary label $y \in \{-1, 1\}$. We use a real-valued hypothesis h to predict the label y solely from the features $\mathbf{x}$. The logistic loss incurred by a specific hypothesis h is defined as

$$L\left((\mathbf{x}, y), h\right) := \log(1 + \exp(-yh(\mathbf{x}))). \tag{123}$$

. Carefully note that the expression (123) for the logistic loss applies only for the label space $\mathcal{Y} = \{-1, 1\}$ and using the thresholding rule (120). 37, 44–47, 74, 189, 191

logistic regression Logistic regression learns a linear hypothesis map (classifier) $h(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$ to predict a binary label y based on numeric features $\mathbf{x}$ of a data point. The quality of a linear hypothesis map is measured using its average logistic loss on some labeled data points (the training set). 44

loss With a slight abuse of language, we use the term loss either for the loss function itself or for its value for a specific pair of a data point and a hypothesis. 14, 16, 34–40, 42–44, 46, 47, 49–51, 62, 67, 68, 70, 83, 99, 100, 102, 107, 110, 124, 168, 169, 176, 183, 190, 193, 197, 198, 200–203

loss function A loss function is a map

$$L : \mathcal{X} \times \mathcal{Y} \times \mathcal{H} \to \mathbb{R}_+ : \big(\big(\mathbf{x}, y\big), h\big) \mapsto L\left((\mathbf{x}, y), h\right)$$

which assigns a pair consisting of a data point, with features $\mathbf{x}$ and label y, and a hypothesis $h \in \mathcal{H}$ the non-negative real number $L\left((\mathbf{x}, y), h\right)$. The loss value $L\left((\mathbf{x}, y), h\right)$ quantifies the discrepancy between the true label y and the predicted label $h(\mathbf{x})$. Smaller (closer to zero) values $L\left((\mathbf{x}, y), h\right)$ mean a smaller discrepancy between predicted label and true label of a data point. Figure 27 depicts a loss function for a given data point, with features $\mathbf{x}$ and label y, as a function of the hypothesis $h \in \mathcal{H}$. 21, 26, 34–39, 42, 44–46, 51–53, 59, 62, 65, 67, 68, 74, 94, 100, 101, 103, 104, 110–112, 114, 121, 123, 124, 129, 130, 133, 134, 137, 138, 143, 146–148, 153, 154, 189, 190, 198, 199

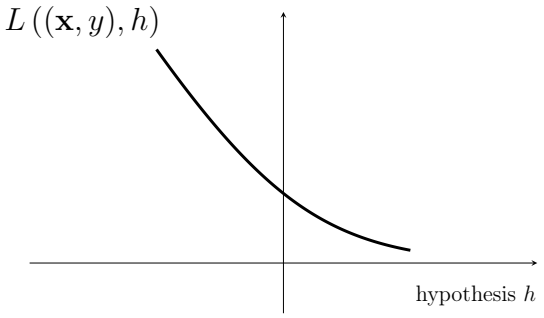

Figure 27: Some loss function $L\left((\mathbf{x}, y), h\right)$ for a fixed data point, with feature vector $\mathbf{x}$ and label y, and varying hypothesis h. ML methods try to find (learn) a hypothesis that incurs minimum loss.

maximum likelihood Consider data points $\mathcal{D} = \left\{\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(m)}\right\}$ that are interpreted as realizations of i.i.d. RVs with a common probability distribution $p(\mathbf{z}; \mathbf{w})$ which depends on a parameter vector $\mathbf{w} \in \mathcal{W} \subseteq \mathbb{R}^n$. Maximum likelihood methods aim at finding a parameter vector $\mathbf{w}$ such that the probability (density) $p(\mathcal{D}; \mathbf{w}) = \prod_{r=1}^{m} p(\mathbf{z}^{(r)}; \mathbf{w})$ of observing the data is maximized. Thus, the maximum likelihood estimator is obtained as a solution to the optimization problem $\max_{\mathbf{w} \in \mathcal{W}} p(\mathcal{D}; \mathbf{w})$. 36, 39

mean The expectation $\mathbb{E}\{\mathbf{x}\}$ of a numeric RV $\mathbf{x}$. 93

metric A metric refers to a loss function that is used solely for the final performance evaluation of a learnt hypothesis. The metric is typically a loss function that has a "natural" interpretation (such as the 0/1 loss) but is not a good choice to guide the learning process, e.g., via ERM. For ERM, we typically prefer loss functions that depend smoothly on the (parameters of the) hypothesis. Examples for such smooth loss

networked data Networked data consists of local datasets that are related by some notion of pair-wise similarity. We represent networked data using an empirical graph whose nodes carry local datasets and an edge indicates a similarity between the connected nodes. 1, 17, 99, 111

networked federated learning Networked federated learning refers to methods that learn personalized models in a distributed fashion from local datasets that are related by an intrinsic network structure. 113

networked model A networked model over an empirical graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ assigns a local model (hypothesis space) to each node $i \in \mathcal{V}$ of the empirical graph $\mathcal{G}$. 99, 100, 108

non-smooth We refer to a function as non-smooth if it is not smooth [59]. 37

objective function An objective function is a map that assigns each possible value of an optimization variable, such as the parameters $\mathbf{w}$ of a hypothesis $h^{(\mathbf{w})}$, to an objective value $f(\mathbf{w})$. The objective value $f(\mathbf{w})$ could by the the risk or the empirical risk of a hypothesis $h^{(\mathbf{w})}$. 52, 53, 56, 74–79, 83, 91, 92, 129, 133

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 (time-invariant). However, in some applications the data consists of a majority of "regular" data points that conform with an i.i.d.

assumption and a small number of data points that have fundamentally different statistical properties compared to the regular data points. We refer to a data point that substantially deviates from the statistical properties of the majority of data points as an outlier. Different methods for outlier detection use different measures for this deviation. 36, 39, 100

overfitting Consider a ML method that uses ERM to learn a hypothesis with minimum empirical risk on a given training set. Such a method is "overfitting" the training set if it learns hypothesis with small empirical risk on the training set but unacceptably large loss outside the training set. 62, 64, 65, 84, 178

parameters The parameters of a ML model are tunable (learnable or adjustable) quantities that allow to choose between different hypothesis maps. For example, the linear model $\mathcal{H} := \{h : h(x) = w_1 x + w_2\}$ consists of all hypothesis maps $h(x) = w_1 x + w_2$ with a particular choice for the parameters w_1, w_2. Another example of parameters are the weights assigned to the connections of an ANN. 75, 149

polynomial regression Polynomial regression aims at learning a polynomial hypothesis map to predict a numeric label based on numeric features of a data point. For data points characterized by a single numeric features, 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 training set). 64

positive semi-definite A symmetric matrix $\mathbf{Q} = \mathbf{Q}^T \in \mathbb{R}^{d \times d}$ is referred to as positive semi-definite if $\mathbf{x}^T \mathbf{Q} \mathbf{x} \geq 0$ for every vector $\mathbf{x} \in \mathbb{R}^d$. 10, 15, 73, 79, 92, 143

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 $\widehat{y} := h(\mathbf{x})$ for its label y. 7, 14, 26, 27, 29, 31, 34, 38, 40, 65, 77, 108, 113, 114, 135, 136, 168, 169, 187, 191, 198

privacy leakage Consider a (ML or FL) system that processes a local dataset $\mathcal{D}^{(i)}$ and shares data, such as the predictions obtained for new data points, with other parties. Privacy leakage arises if the shared data carries information about a private (sensitive) feature of a data point (which might be a human) of $\mathcal{D}^{(i)}$. The amount of privacy leakage can be measured via mutual information using a probabilistic model for the local dataset. 151, 194

privacy protection Privacy protection aims at avoiding (or minimizing) the privacy leakage occuring within data processing systems (such as ML or FL methods). 151–153

probabilistic model A probabilistic model interprets data points as realizations of RVs with a joint probability distribution.This joint probability distribution typically involves parameters which have to be manually chosen (=design choice) or learnt via statistical inference methods [44]. 36, 49, 50, 55, 57, 59, 89, 96, 98, 151, 177, 179, 188, 194, 200

probability We assign a probability value, typically chosen in the interval $[0,1]$, to each event that might occur in a random experiment [4, 40, 116, 127]. 73

probability density function (pdf) The probability density function (pdf) $p(x)$ of a real-valued RV $x \in \mathbb{R}$ is a particular representation of its probability distribution. If the pdf exists, it can be used to compute the probability that x takes on a value from a (measurable) set $\mathcal{B} \subseteq \mathbb{R}$ via $p(x \in \mathcal{B}) = \int_{\mathcal{B}} p(x')dx'$ [4, Ch. 3]. The pdf of a vector-valued RV $\mathbf{x} \in \mathbb{R}^d$ (if it exists) allows to compute the probability that $\mathbf{x}$ falls into a (measurable) region $\mathcal{R}$ via $p(\mathbf{x} \in \mathcal{R}) = \int_{\mathcal{R}} p(\mathbf{x}')dx'_1 \ldots dx'_d$ [4, Ch. 3]. 184

probability distribution The data generated in some ML applications can be reasonably well modelled as realizations of a RV. The overall statistical properties (or intrinsic structure) of such data are then governed by the probability distribution of this RV. We use the term probability distribution in a highly informal manner and mean the collection of probabilities assigned to different values or value ranges of a RV. The probability distribution of a binary RV $y \in \{0,1\}$ is fully specified by the probabilities $p(y=0)$ and $p(y=1)\big(=1\text{–}p(y=0)\big)$. The probability distribution of a real-valued RV $x \in \mathbb{R}$ might be specified by a probability density function $p(x)$ such that $p(x \in [a,b]) \approx p(a)|b-a|$. In the most general case, a probability distribution is defined by a probability measure [40, 46]. 11, 36, 43, 49–54, 60, 64, 73, 83, 89, 91, 97, 98, 103, 115–117, 119, 152, 167, 177, 184–186, 190–192, 194, 195, 199,

72, 111, 114, 138, 140, 181

regularizer A regularizer assigns each hypothesis h from a hypothesis space $\mathcal{H}$ a quantitative measure $\mathcal{R}\big\{h\big\}$ for how much its prediction error on a training set might differ from its prediction errors on data points outside the training set. Ridge regression uses the regularizer $\mathcal{R}\big\{h\big\} := \|\mathbf{w}\|_2^2$ for linear hypothesis maps $h^{(\mathbf{w})}(\mathbf{x}) := \mathbf{w}^T\mathbf{x}$ [1, Ch. 3]. The Lasso uses the regularizer $\mathcal{R}\big\{h\big\} := \|\mathbf{w}\|_1$ for linear hypothesis maps $h^{(\mathbf{w})}(\mathbf{x}) := \mathbf{w}^T\mathbf{x}$ [1, Ch. 3]. 22, 67, 68, 104, 111, 181

ridge regression Ridge regression learns the parameter (or weight) vector $\mathbf{w}$ of a linear hypothesis map $h^{(\mathbf{w})}(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$. The quality of a particular choice for the parameter vector $\mathbf{w}$ is measured by the sum of two components. The first components is the average squared error loss incurred by $h^{(\mathbf{w})}$ on a set of labeled data points (the training set). The second component is the scaled squared Euclidean norm $\lambda\|\mathbf{w}\|_2^2$ with a regularization parameter $\lambda > 0$. It can be shown that the effect of adding to $\lambda\|\mathbf{w}\|_2^2$ to the average squared error loss is equivalent to replacing the original data points by an ensemble of realizations of a RV centered around these data points. 67, 68, 72, 83, 84, 102, 198

risk Consider a hypothesis h that is 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 $L\left((\mathbf{x}, y), h\right)$. If we interpret data points as the realizations of i.i.d. RVs, also the $L\left((\mathbf{x}, y), h\right)$ becomes the realization of a RV. Using such an i.i.d. assumption allows to define the risk of a hypothesis as the expected loss $\mathbb{E}\big\{L\left((\mathbf{x}, y), h\right)\big\}$. Note that

the risk of h depends on both, the specific choice for the loss function and the probability distribution of the data points. 49–52, 54–57, 60, 62, 65, 66, 75, 91, 169, 192, 197, 203

sample covariance matrix The sample covariance matrix $\widehat{\mathbf{\Sigma}} \in \mathbb{R}^{d \times d}$ for a given set of feature vectors $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)} \in \mathbb{R}^d$ is defined as

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

Here, we used the sample mean $\widehat{\mathbf{m}}$. 117

sample mean The sample mean $\mathbf{m} \in \mathbb{R}^{d \times d}$ for a given set of feature vectors $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)} \in \mathbb{R}^d$ is defined as

$$\mathbf{m} = (1/m) \sum_{r=1}^{m} \mathbf{x}^{(r)}.$$

116, 199

sample size The number of individual data points contained in a dataset that is obtained from realizations of i.i.d. RVs. 43, 47, 56, 181

scatterplot A visualization technique that depicts data points by markers in a two-dimensional plane. 175, 178

smooth We refer to a real-valued function as smooth if it is differentiable and its gradient is continuous [42, 59]. In particular, a differentiable function $f(\mathbf{w})$ is refered to as β-smooth if the gradient $\nabla f(\mathbf{w})$ is Lipschitz continuous with Lipschitz constant β, i.e.,

$$\|\nabla f(\mathbf{w}) - \nabla f(\mathbf{w}')\| \leq \beta \|\mathbf{w} - \mathbf{w}'\|.$$

192

spectral clustering Spectral clustering groups the nodes of an undirected graph by applying k-means clustering to node-wise feature vectors. These feature vectors are built from the eigenvectors of the graph Laplacian matrix [81, 82]. 112

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

$$L\left((\mathbf{x}, y), h\right) := \big(y - \underbrace{h(\mathbf{x})}_{=\hat{y}}\big)^2. \tag{124}$$

38–42, 59, 67, 68, 72, 114, 115, 129, 134, 135, 138, 143, 160, 161, 187, 191, 193, 198

statistical aspects By statistical aspects of a ML method, we refer to (properties of) the probability distribution of its ouput given a probabilistic model for the data fed into the method. 26, 29–31, 35, 37, 43, 52, 53, 84, 99–101, 109, 114, 115, 187, 202

stochastic gradient descent Stochastic GD is obtained from GD by replacing the gradient of the objective function with a noisy (or stochastic) estimate. 75, 88–91, 93, 130

stopping criterion Many ML methods use iterative algorithms that construct a sequence of model parameters (such as the weights of a linear map or the weights of an ANN) that (hopefully) converge to an optimal choice for the model parameters. In practice, given finite computational resources, we need to stop iterating after a finite number of times. A

stopping criterion is any well-defined condition required for stopping iterating. 77, 79, 85, 94, 131–135, 139, 140, 153, 155

strongly convex A continuously differentiable real-valued function $f(\mathbf{x})$ is strongly convex with coefficient σ if $f(\mathbf{y}) \geq f(\mathbf{x}) + \nabla f(\mathbf{x})^T(\mathbf{y}-\mathbf{x}) + (\sigma/2)\left\|\mathbf{y}-\mathbf{x}\right\|_2^2$ [59], [41, Sec. B.1.1.]. 75

structural risk minimization Structural risk minimization is the problem of finding the hypothesis that optimally balances the average loss (or empirical risk) on a training set with a regularization term. The regularization term penalizes a hypothesis that is not robust against (small) perturbations of the data points in the training set. 186, 197

subgradient For a real-valued function $f : \mathbb{R}^d \to \mathbb{R} : \mathbf{w} \mapsto f(\mathbf{w})$, a vector $\mathbf{a}$ such that $f(\mathbf{w}) \geq f(\mathbf{w}') + \left(\mathbf{w}-\mathbf{w}'\right)^T\mathbf{a}$ is referred to as a subgradient of f at $\mathbf{w}'$ [56, 129]. 46

subgradient descent Subgradient descent is a generalization of GD that does not require differentiablity of the function to be minimized. This generalization is obtained by replacing the concept of a gradient with that of a sub-gradient. Similar to gradients, also sub-gradients allow to construct local approximations of an objective function. The objective function might be the empirical risk $\widehat{L}\left(h^{(\mathbf{w})}\middle|\mathcal{D}\right)$ viewed as a function of the model parameters $\mathbf{w}$ that select a hypothesis $h^{(\mathbf{w})} \in \mathcal{H}$. 45

support vector machine A binary classification method for learning a linear hypothesis map that maximally separates data points from the two different classes in the feature space ("maximum margin"). Maximizing

this separation is equivalent to minimizing a regularized variant of the hinge loss (121). 182, 183

training error The average loss of a hypothesis when predicting the labels of data points in a training set. We sometimes refer by training error also the minimum average loss incurred on the training set by any hypothesis out of a hypothesis space. 14, 54, 62, 64–66, 107, 202, 203

training set A set of data points that is 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 between training error and validation error of $\hat{h}$ allows to diagnose ML methods and informs how to improve them (e.g., using a different hypothesis space or collecting more data points). 13, 14, 39, 41, 49, 54–57, 59, 60, 62, 64–68, 70, 80, 88, 89, 91, 96, 97, 100, 102, 164, 169, 176, 181–183, 188, 189, 193, 197, 198, 201–203

trustworthiness Beside the computational aspects and statistical aspects, a third main design aspect for ML methods is their trustworthiness. The European Union has put forward seven key requirements for trustworthy AI systems (that typically build on ML methods) [100]: **human agency and oversight**, **technical robustness and safety**, **privacy and data governance**, **transparency**, **diversity non-discrimination and fairness**, **societal and environmental well-being**, **accountability**. 29

underfitting Consider a ML method that uses ERM to learn a hypothesis with minimum empirical risk on a given training set. Such a method

is "underfitting" the training set if it is not able to learn a hypothesis with sufficiently small empirical risk on the training set. If a method is underfitting it will typically also not be able to learn a hypothesis with a small risk. 65

validation Consider a hypothesis $\hat{h}$ that has been learn via ERM on some training set $\mathcal{D}$. Validation refers to the practice of trying out a hypothesis $\hat{h}$ on a validation set that consists of data points that are not contained in the training set $\mathcal{D}$. 79, 110

validation error Consider a hypothesis $\widehat{h}$ which is obtained by ERM on a training set. The average loss of $\widehat{h}$ on a validation set, which is different from the training set, is referred to as the validation error. 14, 119, 202, 203

validation set A set of data points that have not been used as training set in ERM to learn a hypothesis $\widehat{h}$. The average loss of $\widehat{h}$ on the validation set is referred to as the validation error and used to diagnose the ML method (see [1, Sec. 6.6.]). The comparison between training error and validation error can inform directions for improvements of the ML method (such as using a different hypothesis space). 14, 164, 203

Vapnik–Chervonenkis (VC) dimension The VC dimension of an infinite hypothesis space is a widely-used measure for its size. We refer to [130] for a precise definition of VC dimension as well as a discussion of its basic properties and use in ML. 64

variance The variance of a real-valued RV x is defined as the expectation

$\mathbb{E}\{\big(x - \mathbb{E}\{x\}\big)^2\}$ of the squared difference x and its expectation $\mathbb{E}\{x\}$. We extend this definition to vector-valued RVs $\mathbf{x}$ as $\mathbb{E}\{\big\|\mathbf{x} - \mathbb{E}\{\mathbf{x}\}\big\|_2^2\}$. 14, 55, 56, 60, 90

vertical FL Vertical FL refers to applications with local datasets that are constituted by the same data points but characterizing them with different features [98]. For example, different healthcare providers might all contain information about the same population of patients. However, different healthcare providers collect different measurements (blood values, electrocardiography, lung x-ray) for the same patients. 121, 124, 126

References

[1] A. Jung, *Machine Learning: The Basics*, 1st ed. Springer Singapore, Feb. 2022.

[2] W. Rudin, *Real and Complex Analysis*, 3rd ed. New York: McGraw-Hill, 1987.

[3] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 3rd ed. Baltimore, MD: Johns Hopkins University Press, 1996.

[4] D. Bertsekas and J. Tsitsiklis, *Introduction to Probability*, 2nd ed. Athena Scientific, 2008.

[5] W. Rudin, *Principles of Mathematical Analysis*, 3rd ed. New York: McGraw-Hill, 1976.

[6] S. Cui, A. Hero, Z.-Q. Luo, and J. Moura, Eds., *Big Data over Networks*. Cambridge Univ. Press, 2016.

[7] M. Wollschlaeger, T. Sauter, and J. Jasperneite, "The future of industrial communication: Automation networks in the era of the internet of things and industry 4.0," *IEEE Industrial Electronics Magazine*, vol. 11, no. 1, pp. 17–27, 2017.

[8] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, Jan. 2017. [Online]. Available: https://doi.org/10.1109/MC.2017.9

[9] H. Ates, A. Yetisen, F. Güder, and C. Dincer, "Wearable devices for the detection of covid-19," *Nature Electronics*, vol. 4, no. 1, pp. 13–14, 2021. [Online]. Available: https://doi.org/10.1038/s41928-020-00533-1

[10] H. Boyes, B. Hallaq, J. Cunningham, and T. Watson, "The industrial internet of things (iiot): An analysis framework," *Computers in Industry*, vol. 101, pp. 1–12, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0166361517307285

[11] M. E. J. Newman, *Networks: An Introduction.* Oxford Univ. Press, 2010.

[12] A. Barabási, N. Gulbahce, and J. Loscalzo, "Network medicine: a network-based approach to human disease," *Nature Reviews Genetics*, vol. 12, no. 56, 2011.

[13] K. Grantz, H. Meredith, D. Cummings, C. Metcalf, B. Grenfell, J. Giles, S. Mehta, S. Solomon, A. Labrique, N. Kishore, C. Buckee, and A. Wesolowski, "The use of mobile phone data to inform analysis of COVID-19 pandemic epidemiology," *Nature Communications*, vol. 11, no. 1, p. 4961, 2020. [Online]. Available: https://doi.org/10.1038/s41467-020-18190-5

[14] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, A. Singh and J. Zhu, Eds., vol. 54. Fort Lauderdale, FL,

USA: PMLR, 20–22 Apr 2017, pp. 1273–1282. [Online]. Available: http://proceedings.mlr.press/v54/mcmahan17a.html

[15] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated learning: Challenges, methods, and future directions," *IEEE Signal Processing Magazine*, vol. 37, no. 3, pp. 50–60, May 2020.

[16] Y. Cheng, Y. Liu, T. Chen, and Q. Yang, "Federated learning for privacy-preserving ai," *Communications of the ACM*, vol. 63, no. 12, pp. 33–36, Dec. 2020.

[17] N. Agarwal, A. Suresh, F. Yu, S. Kumar, and H. McMahan, "cpSGD: Communication-efficient and differentially-private distributed sgd," in *Proc. Neural Inf. Proc. Syst. (NIPS)*, 2018.

[18] V. Smith, C.-K. Chiang, M. Sanjabi, and A. Talwalkar, "Federated Multi-Task Learning," in *Advances in Neural Information Processing Systems*, vol. 30, 2017. [Online]. Available: https://proceedings.neurips.cc/paper/2017/file/6211080fa89981f66b1a0c9d55c61d0f-Paper.pdf

[19] J. You, J. Wu, X. Jin, and M. Chowdhury, "Ship compute or ship data? why not both?" in *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*. USENIX Association, April 2021, pp. 633–651. [Online]. Available: https://www.usenix.org/conference/nsdi21/presentation/you

[20] D. Tse and P. Viswanath, *Fundamentals of Wireless Communication*. Cambridge University Press, 2005.

[21] T. Yang, G. Andrew, H. Eichner, H. Sun, W. Li, N. Kong, D. Ramage, and F. Beaufays, "Applied federated learning: Improving google keyboard query suggestions," 2018. [Online]. Available: https://arxiv.org/abs/1812.02903

[22] A. Ghosh, J. Chung, D. Yin, and K. Ramchandran, "An efficient framework for clustered federated learning," in *34th Conference on Neural Information Processing Systems (NeurIPS 2020)*, Vancouver, Canada, 2020.

[23] F. Sattler, K. Müller, and W. Samek, "Clustered federated learning: Model-agnostic distributed multitask optimization under privacy constraints," *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[24] G. Strang, *Computational Science and Engineering.* Wellesley-Cambridge Press, MA, 2007.

[25] ——, *Introduction to Linear Algebra*, 5th ed. Wellesley-Cambridge Press, MA, 2016.

[26] H. H. Bauschke and P. L. Combettes, *Convex Analysis and Monotone Operator Theory in Hilbert Spaces.* New York: Springer, 2011.

[27] F. Pedregosa, "Scikit-learn: Machine learning in python," *Journal of Machine Learning Research*, vol. 12, no. 85, pp. 2825–2830, 2011. [Online]. Available: http://jmlr.org/papers/v12/pedregosa11a.html

[28] J. Hirvonen and J. Suomela. (2023) Distributed algorithms 2020.

[29] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge, UK: Cambridge Univ. Press, 2004.

[30] D. P. Bertsekas and J. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods*. Athena Scientific, 2015.

[31] M. Conti and D. Moretti, "System level analysis of the bluetooth standard," in *Design, Automation and Test in Europe*, 2005, pp. 118–123 Vol. 3.

[32] C. Doersch and A. Zisserman, "Multi-task self-supervised visual learning," in *2017 IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 2070–2079.

[33] M. Ribeiro, S. Singh, and C. Guestrin, ""Why should i trust you?": Explaining the predictions of any classifier," in *Proc. 22nd ACM SIGKDD*, Aug. 2016, pp. 1135–1144.

[34] R. Raina, A. Madhavan, and A. Ng, "Large-scale deep unsupervised learning using graphics processors," in *Proceedings of the 26th Annual International Conference on Machine Learning*, ser. ICML '09. New York, NY, USA: Association for Computing Machinery, 2009, pp. 873–880. [Online]. Available: https://doi.org/10.1145/1553374.1553486

[35] X. Lin, Y. Rivenson, N. Yardimci, M. Veli, Y. Luo, M. Jarrahi, and A. Ozcan, "All-optical machine learning using diffractive deep neural networks," *Science*, vol. 361, no. 6406, pp. 1004–1008, 2018. [Online]. Available: https://www.science.org/doi/abs/10.1126/science.aat8084

[36] C. Molnar, *Interpretable Machine Learning - A Guide for Making Black Box Models Explainable.* [online] Available: https://christophm.github.io/interpretable-ml-book/., 2019.

[37] A. Jung and P. Nardelli, "An information-theoretic approach to personalized explainable machine learning," *IEEE Sig. Proc. Lett.*, vol. 27, pp. 825–829, 2020.

[38] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning.* MIT Press, 2016.

[39] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*, ser. Springer Series in Statistics. New York, NY, USA: Springer, 2001.

[40] P. Billingsley, *Probability and Measure*, 3rd ed. New York: Wiley, 1995.

[41] D. P. Bertsekas, *Convex Optimization Algorithms.* Athena Scientific, 2015.

[42] S. Bubeck, "Convex optimization. algorithms and complexity." in *Foundations and Trends in Machine Learning.* Now Publishers, 2015, vol. 8.

[43] A. Nedic and A. Ozdaglar, "Distributed subgradient methods for multi-agent optimization," *IEEE Transactions on Automatic Control*, vol. 54, no. 1, pp. 48–61, 2009.

[44] E. L. Lehmann and G. Casella, *Theory of Point Estimation*, 2nd ed. New York: Springer, 1998.

[45] M. J. Wainwright and M. I. Jordan, *Graphical Models, Exponential Families, and Variational Inference*, ser. Foundations and Trends in Machine Learning. Hanover, MA: Now Publishers, 2008, vol. 1, no. 1–2.

[46] R. Gray, *Probability, Random Processes, and Ergodic Properties*, 2nd ed. New York: Springer, 2009.

[47] M. Wainwright, *High-Dimensional Statistics: A Non-Asymptotic Viewpoint*. Cambridge: Cambridge University Press, 2019.

[48] J. Cho and H. White, "Generalized runs tests for the iid hypothesis," *Journal of Econometrics*, vol. 162, no. 2, pp. 326–344, 2011. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0304407611000285

[49] P. J. Huber, *Robust Statistics*. New York: Wiley, 1981.

[50] T. W. Anderson, *An Introduction to Multivariate Statistical Analysis*. Wiley, 2003.

[51] L. Breiman and D. Freedman, "How many variables should be entered in a regression equation?" *Journal of the American Statistical Association*, vol. 78, no. 381, pp. 131–136, 1983.

[52] J. Mourtada, "Exact minimax risk for linear least squares, and the lower tail of sample covariance matrices," *The Annals of Statistics*, vol. 50, no. 4, pp. 2157 – 2178, 2022. [Online]. Available: https://doi.org/10.1214/22-AOS2181

[53] A. V. Oppenheim, R. W. Schafer, and J. R. Buck, *Discrete-Time Signal Processing*, 2nd ed. Englewood Cliffs, NJ: Prentice Hall, 1998.

[54] N. Tran, O. Abramenko, and A. Jung, "On the sample complexity of graphical model selection from non-stationary samples," *IEEE Transactions on Signal Processing*, vol. 68, pp. 17–32, 2020.

[55] N. Parikh and S. Boyd, "Proximal algorithms," *Foundations and Trends in Optimization*, vol. 1, no. 3, pp. 123–231, 2013.

[56] D. P. Bertsekas, *Nonlinear Programming*, 2nd ed. Belmont, MA: Athena Scientific, June 1999.

[57] G. Golub and C. van Loan, "An analysis of the total least squares problem," *SIAM J. Numerical Analysis*, vol. 17, no. 6, pp. 883–893, Dec. 1980.

[58] Y. L. T. Schaul, X. Zhang, "No more pesky learning rates," in *Proc. of the 30th International Conference on Machine Learning, PMLR 28(3)*, vol. 28, Atlanta, Georgia, June 2013, pp. 343–351.

[59] Y. Nesterov, *Introductory lectures on convex optimization*, ser. Applied Optimization. Kluwer Academic Publishers, Boston, MA, 2004, vol. 87, a basic course. [Online]. Available: http://dx.doi.org/10.1007/978-1-4419-8853-9

[60] A. Papoulis and S. U. Pillai, *Probability, Random Variables, and Stochastic Processes*, 4th ed. New York: Mc-Graw Hill, 2002.

[61] R. A. Horn and C. R. Johnson, *Matrix Analysis*, 2nd ed. Cambridge, UK: Cambridge Univ. Press, 2013.

[62] A. Jung, "A fixed-point of view on gradient methods for big data," *Frontiers in Applied Mathematics and Statistics*, vol. 3, 2017. [Online]. Available: https://www.frontiersin.org/article/10.3389/fams.2017.00018

[63] A. Rakhlin, O. Shamir, and K. Sridharan, "Making gradient descent optimal for strongly convex stochastic optimization," in *Proceedings of the 29 th International Conference on Machine Learning*, Edinburgh, Scotland, UK,, 2012.

[64] R. Diestel, *Graph Theory*. Springer Berlin Heidelberg, 2005.

[65] A. Jung, "Networked exponential families for big data over networks," *IEEE Access*, vol. 8, pp. 202 897–202 909, 2020.

[66] H. Ambos, N. Tran, and A. Jung, "The logistic network lasso," *arXiv*, 2018.

[67] ——, "Classifying big data over networks via the logistic network lasso," in *Proc. 52nd Asilomar Conf. Signals, Systems, Computers*, Oct./Nov. 2018.

[68] A. Jung and N. Tran, "Localized linear regression in networked data," *IEEE Sig. Proc. Lett.*, vol. 26, no. 7, Jul. 2019.

[69] P. J. Brockwell and R. A. Davis, *Time Series: Theory and Methods*. Springer New York, 1991.

[70] L. Z. Y. SarcheshmehPour, Y. Tian and A. Jung, "Networked federated learning," *arXiv e-prints*, 2022.

[71] A. Jung, "On the duality between network flows and network lasso," *IEEE Sig. Proc. Lett.*, vol. 27, pp. 940 – 944, 2020.

[72] A. Jung and Y. SarcheshmehPour, "Local graph clustering with network lasso," *IEEE Signal Processing Letters*, vol. 28, pp. 106–110, 2021.

[73] D. Hallac, J. Leskovec, and S. Boyd, "Network lasso: Clustering and optimization in large graphs," in *Proc. SIGKDD*, 2015, pp. 387–396.

[74] Y. SarcheshmehPour, M. Leinonen, and A. Jung, "Federated learning from big data over networks," in *Proc. IEEE Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP), preprint: https://arxiv.org/pdf/2010.14159.pdf*, 2021.

[75] A. Jung, N. Tran, and A. Mara, "When is Network Lasso Accurate?" *Front. Appl. Math. Stat.*, vol. 3, Jan. 2018.

[76] B. Nadler, N. Srebro, and X. Zhou, "Statistical analysis of semi-supervised learning: The limit of infinite unlabelled data," in *Advances in Neural Information Processing Systems 22*, 2009, pp. 1330–1338.

[77] F. Bach, R. Jenatton, J. Mairal, and G. Obozinski, "Optimization with sparsity-inducing penalties," *Found. Trends Mach. Learn.*, vol. 4, no. 1, pp. 1–106, Jan. 2012.

[78] L. Jacob, J.-P. Vert, and F. Bach, "Clustered multi-task learning: A convex formulation," in *Advances in Neural Information*

Processing Systems, D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, Eds., vol. 21. Curran Associates, Inc., 2009. [Online]. Available: https://proceedings.neurips.cc/paper/2008/file/fccb3cdc9acc14a6e70a12f74560c026-Paper.pdf

[79] T. Evgeniou, C. Micchelli, and M. Pontil, "Learning multiple tasks with kernel methods," *Journal of Machine Learning Research*, vol. 6, no. 21, pp. 615–637, 2005. [Online]. Available: http://jmlr.org/papers/v6/evgeniou05a.html

[80] F. Sung, Y. Yang, L. Zhang, T. Xiang, P. H. Torr, and T. M. Hospedales, "Learning to compare: Relation network for few-shot learning," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 1199–1208.

[81] U. von Luxburg, "A tutorial on spectral clustering," *Statistics and Computing*, vol. 17, no. 4, pp. 395–416, Dec. 2007.

[82] Y. SarcheshmehPour, Y. Tian, L. Zhang, and A. Jung, "Flow-based clustering and spectral clustering: A comparison," in *2021 55th Asilomar Conference on Signals, Systems, and Computers*, 2021, pp. 1292–1296.

[83] D. Sun, K.-C. Toh, and Y. Yuan, "Convex clustering: Model, theoretical guarantee and efficient algorithm," *Journal of Machine Learning Research*, vol. 22, no. 9, pp. 1–32, 2021. [Online]. Available: http://jmlr.org/papers/v22/18-694.html

[84] K. Pelckmans, J. D. Brabanter, J. Suykens, and B. D. Moor, "Convex clustering shrinkage," in *PASCAL Workshop on Statistics and Optimization of Clustering Workshop*, 2005.

[85] 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. [Online]. Available: https://doi.org/10.1023/A:1006559212014

[86] P. Clement and W. Desch, "An elementary proof of the triangle inequality for the wasserstein metric," *Proceedings of the American Mathematical Society*, vol. 136, no. 1, pp. 333–339, 2008. [Online]. Available: http://www.jstor.org/stable/20535091

[87] A. Y. Ng, M. I. Jordan, and Y. Weiss, "On spectral clustering: Analysis and an algorithm," in *Adv. Neur. Inf. Proc. Syst.*, 2001.

[88] V. Kalofolias, "How to learn a graph from smooth signals," in *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, A. Gretton and C. C. Robert, Eds., vol. 51. Cadiz, Spain: PMLR, 09–11 May 2016, pp. 920–929.

[89] X. Dong, D. Thanou, M. Rabbat, and P. Frossard, "Learning graphs from data: A signal representation perspective," *IEEE Signal Processing Magazine*, vol. 2019, no. 3, May 2019.

[90] L. Stanković, D. Mandic, M. Daković, M. Brajović, B. Scalzo, S. Li, and A. Constantinides, "Data analytics on graphs part III: Machine learning on graphs, from graph topology to applications," *Foundations*

and Trends® in Machine Learning, vol. 13, no. 4, pp. 332–530, 2020. [Online]. Available: http://dx.doi.org/10.1561/2200000078-3

[91] V. Mothukuri, R. M. Parizi, S. Pouriyeh, Y. Huang, A. Dehghantanha, and G. Srivastava, "A survey on security and privacy of federated learning," *Future Generation Computer Systems*, vol. 115, pp. 619–640, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X20329848

[92] Q. Yang, Y. Liu, Y. Cheng, Y. Kang, T. Chen, and H. Yu, *Federated Learning*, 1st ed. Springer, 2022.

[93] S. R. Pokhrel and J. Choi, "Federated learning with blockchain for autonomous vehicles: Analysis and design challenges," *IEEE Transactions on Communications*, vol. 68, no. 8, pp. 4734–4746, 2020.

[94] O. Chapelle, B. Schölkopf, and A. Zien, Eds., *Semi-Supervised Learning*. Cambridge, Massachusetts: The MIT Press, 2006.

[95] A. Ghosh, J. Chung, D. Yin, and K. Ramchandran, "An efficient framework for clustered federated learning," *IEEE Transactions on Information Theory*, vol. 68, no. 12, pp. 8076–8091, 2022.

[96] H. Ludwig and N. Baracaldo, Eds., *Federated Learning: A Comprehensive Overview of Methods and Applications*. Springer, 2022.

[97] A. Shamsian, A. Navon, E. Fetaya, and G. Chechik, "Personalized federated learning using hypernetworks," in *Proceedings of the 38th International Conference on Machine Learning*, ser. Proceedings of

Machine Learning Research, M. Meila and T. Zhang, Eds., vol. 139. PMLR, 18–24 Jul 2021, pp. 9489–9502. [Online]. Available: https://proceedings.mlr.press/v139/shamsian21a.html

[98] Q. Yang, Y. Liu, Y. Cheng, Y. Kang, T. Chen, and H. Yu, *Vertical Federated Learning*. Cham: Springer International Publishing, 2020, pp. 69–81. [Online]. Available: https://doi.org/10.1007/978-3-031-01585-4_5

[99] ——, *Horizontal Federated Learning*. Cham: Springer International Publishing, 2020, pp. 49–67. [Online]. Available: https://doi.org/10.1007/978-3-031-01585-4_4

[100] E. Commission, C. Directorate-General for Communications Networks, and Technology, *The Assessment List for Trustworthy Artificial Intelligence (ALTAI) for self assessment*. Publications Office, 2020.

[101] H.-L. E. G. on Artificial Intelligence, "Ethics guidelines for trustworthy ai," European Commission, Tech. Rep., April 2019.

[102] M. J. Sheller, B. Edwards, G. A. Reina, J. Martin, S. Pati, A. Kotrotsou, M. Milchenko, W. Xu, D. Marcus, R. R. Colen, and S. Bakas, "Federated learning in medicine: facilitating multi-institutional collaborations without sharing patient data," *Scientific Reports*, vol. 10, no. 1, p. 12598, 2020. [Online]. Available: https://doi.org/10.1038/s41598-020-69250-1

[103] K. Chaudhuri, C. Monteleoni, and A. Sarwate, "Differentially private empirical risk minimization," *J. Mach. Learn. Res.*, vol. 12, pp. 1069–1109, Mar. 2011.

[104] H. Hsu, N. Martinez, M. Bertran, G. Sapiro, and F. P. Calmon, "A survey on statistical, information, and estimation—theoretic views on privacy," *IEEE BITS the Information Theory Magazine*, vol. 1, no. 1, pp. 45–56, 2021.

[105] P. Samarati, "Protecting respondents identities in microdata release," *IEEE Transactions on Knowledge and Data Engineering*, vol. 13, no. 6, pp. 1010–1027, 2001.

[106] L. Sweeney, "k-anonymity: a model for protecting privacy," *International Journal on Uncertainty, Fuzziness and Knowledge-based Systems*, vol. 10, no. 5, pp. 557–570, 2002.

[107] X. Liu, H. Li, G. Xu, Z. Chen, X. Huang, and R. Lu, "Privacy-enhanced federated learning against poisoning adversaries," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 4574–4588, 2021.

[108] 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 of Things Journal*, vol. 8, no. 5, pp. 3310–3322, 2021.

[109] N. M. Müller, S. Roschmann, and K. Böttinger, "Defending Against Adversarial Denial-of-Service Data Poisoning Attacks," *arXiv e-prints*, p. arXiv:2104.06744, April 2021.

[110] H. Wang, K. Sreenivasan, S. Rajput, H. Vishwakarma, S. Agarwal, J.-y. Sohn, K. Lee, and D. Papailiopoulos, "Attack of the tails: Yes, you really can backdoor federated learning," in *Advances in Neural*

Information Processing Systems, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 16 070–16 084. [Online]. Available: https://proceedings.neurips.cc/paper/2020/file/b8ffa41d4e492f0fad2f13e29e1762eb-Paper.pdf

[111] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with numpy," *Nature*, vol. 585, no. 7825, pp. 357–362, 2020. [Online]. Available: https://doi.org/10.1038/s41586-020-2649-2

[112] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, *Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers*. Hanover, MA: Now Publishers, 2010, vol. 3, no. 1.

[113] T. M. Cover and J. A. Thomas, *Elements of Information Theory*, 2nd ed. New Jersey: Wiley, 2006.

[114] C. E. Shannon, "Communication in the presence of noise," 1948.

[115] P. Halmos, *Naive set theory*. Springer-Verlag, 1974.

[116] P. R. Halmos, *Measure Theory*. New York: Springer, 1974.

[117] A. Jung, "Explainable empiricial risk minimization," *submitted to IEEE Sig. Proc. Letters (preprint: https://arxiv.org/pdf/2009.01492.pdf)*, 2020.

[118] J. Chen, L. Song, M. Wainwright, and M. Jordan, "Learning to explain: An information-theoretic perspective on model interpretation," in *Proc. 35th Int. Conf. on Mach. Learning*, Stockholm, Sweden, 2018.

[119] D. Gujarati and D. Porter, *Basic Econometrics*. Mc-Graw Hill, 2009.

[120] Y. Dodge, *The Oxford Dictionary of Statistical Terms*. Oxford University Press, 2003.

[121] B. Everitt, *Cambridge Dictionary of Statistics*. Cambridge University Press, 2002.

[122] R. T. Rockafellar, *Network Flows and Monotropic Optimization*. Athena Scientific, Jul. 1998.

[123] P. Bühlmann and S. van de Geer, *Statistics for High-Dimensional Data*. New York: Springer, 2011.

[124] C. Lampert, "Kernel methods in computer vision," *Foundations and Trends in Computer Graphics and Vision*, 2009.

[125] B. Schölkopf and A. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. Cambridge, MA, USA: MIT Press, Dec. 2002.

[126] A. Lapidoth, *A Foundation in Digital Communication*. New York: Cambridge University Press, 2009.

[127] O. Kallenberg, *Foundations of modern probability.* New York: Springer, 1997.

[128] L. Condat, "A primal–dual splitting method for convex optimization involving lipschitzian, proximable and linear composite terms," *Journal of Opt. Th. and App.*, vol. 158, no. 2, pp. 460–479, Aug. 2013.

[129] D. P. Bertsekas, A. Nedic, and A. E. Ozdaglar, *Convex Analysis and Optimization.* Athena Scientific, 2003.

[130] S. Shalev-Shwartz and S. Ben-David, *Understanding Machine Learning – from Theory to Algorithms.* Cambridge University Press, 2014.