Lectures Notes

CS-E4740 Federated Learning

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

April 24, 2024

Abstract

This course discusses theory and algorithms for federated learning (FL). FL exploits similarities between local datasets to train local (or personalized) models collaboratively. Two core mathematical objects of this course are empirical graphs and generalized total variation minimization (GTVMin). We use empirical graphs to store and process local datasets and parameters of local models. GTVMin formulates FL as an instance of regularized empirical risk minimization. As the regularizer, we use quantitative measures for the variation of local models across the edges of the empirical graph. We can obtain practical FL systems by applying distributed optimization methods to solve GTVMin.

*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, 349966) 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 [1].
$\mathbb{R}_+$	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.
$\mathbf{I}_d, \mathbf{I}$	A square identity matrix of size $d \times d$. If the size is clear from the context, we drop the subscript.
$\mathbb{R}^d$	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 with its jth entry being 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 $\mathbf{x} \in \mathbb{R}^d$ [**?**]. 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 $\mathbf{X} = \mathbf{X}^T$.
$\mathbf{0} = \big(0, \ldots, 0\big)^T$	The vector in $\mathbb{R}^d$ with each entry equal to zero.
$\mathbf{1} = \big(1, \ldots, 1\big)^T$	The vector in $\mathbb{R}^d$ with each entry equal to one.

$\left(\mathbf{v}^T, \mathbf{w}^T\right)^T$	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 $\mathbf{C}$.
$\mathbf{A} \otimes \mathbf{B}$	The Kronecker product of $\mathbf{A}$ and $\mathbf{B}$ [2].

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 [3, 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 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 [4, 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.
$\left(\mathbf{x}^{(r)}, y^{(r)}\right)$	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 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).
η	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 .
$d_{\text{eff}}(\mathcal{H})$	The effective dimension of a hypothesis space $\mathcal{H}$.

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.

α	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 $w_1, \ldots, w_d$, 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.
$\mathcal{G}^{(\mathcal{C})}$	The induced sub-graph of $\mathcal{G}$ using the nodes in $\mathcal{C} \subseteq \mathcal{V}$.
$\mathbf{L}^{(\mathcal{G})}$	The Laplacian matrix of a graph $\mathcal{G}$.
$\mathbf{L}^{(\mathcal{C})}$	The Laplacian matrix of the induced graph $\mathcal{G}^{(\mathcal{C})}$.
$\mathcal{D}^{(i)}$	The local dataset $\mathcal{D}^{(i)}$ at node $i \in \mathcal{V}$ of an empirical graph.
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.
$\mathbf{x}^{(i,r)}$	The features 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 $\mathcal{D}^{(i)}$.

$L^{(\mathrm{d})}\big(\mathbf{x}, h\big(\mathbf{x}\big), h'\big(\mathbf{x}\big)\big)$	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.
$\mathrm{stack}\big\{\mathbf{w}^{(i)}\big\}_{i=1}^{n}$	The vector $\Big(\big(\mathbf{w}^{(1)}\big)^{T}, \ldots, \big(\mathbf{w}^{(n)}\big)^{T}\Big)^{T} \in \mathbb{R}^{dn}$ that is obtained by vertically stacking the local model parameters $\mathbf{w}^{(i)} \in \mathbb{R}^{d}$.

1 Lecture - "Welcome and Intro"

Welcome to the course CS-E4740 Federated Learning. While the course can be completed fully remote, we strongly encourage you to attend the on-site events for your convenience. Any on-site event will be recorded and made available to students via this YouTube channel.

The basic variant (5 credits) of this course consists of lectures (schedule here) and corresponding coding assignments (schedule here). We test your completion of the coding assignments via quizzes (implemented on the MyCourses page). You can upgrade the course to an extended variant (10 credits) by completing a student project (see Section 1.8).

1.1 Learning Goals

This lecture offers

- an introduction of course topic and positioning in wider curricula,
- a discussion of learning goals, assignments and student project,
- an overview of course schedule.

1.2 Introduction

We are surrounded by devices, such as smartphones or wearables, generating decentralized collections of local datasets [6–10]. These local datasets typically have an intrinsic network structure that arises from functional constraints (connectivity between devices) or statistical similarities.

For example, the high-precision management of pandemics uses contact networks to relate local datasets generated by patients. Network medicine relates data about diseases via co-morbidity networks [11]. Social science uses notions of acquaintance to relate data collected from be-friended individuals [12]. Another example for network-structured are observations collected at Finnish Meteorological Institute (FMI) weather stations. Each FMI station generates a local dataset which tend have similar statistical properties for nearby stations.

FL is an umbrella term for distributed optimization techniques to train machine learning (ML) models from decentralized collections of local datasets [13–17]. The idea is to carry out the computations, such as gradient steps (see Lecture 4), arising during ML model training at the location of data generation (such as your smartphone or heart rate sensor). This design philosophy is different from the basic ML workflow, which is first to collect all local datasets at a single location (computer) and then train a single ML model on this pooled data.

It can be beneficial to train different ML models at the locations of actual data generation [18] for several reasons:

- **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 (updates of) model parameters [15, 16]. By exchanging only model parameters, FL methods are considered privacy-friendly in the sense of not leaking (too much) sensitive information that is contained in the local datasets (see Lecture 9).

- **Robustness.** By relying on decentralized data and computation, FL

methods offer robustness (to some extent) against hardware failures (such as "stragglers") and cyber attacks (such data poisoning discussed Lecture 10).

- **Parallel Computing.** Many ML systems are constituted by 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 computer allows to speed up computational tasks such as the computation of gradients required to train ML models (see Lecture 4).

- **Trading Computation against Communication.** Consider a FL application where local datasets are generated by low-complexity devices at remote locations (think of a wildlife camera) 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 [19].

- **Personalization.** FL can be used to train personalized ML models for collections of local datasets, which might be generated by smartphones (and their users) [20]. A key challenge for ensuring personalization is the heterogeneity of local datasets [21, 22]. 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 Lecture 6).

1.3 Prerequisites

The main mathematical structure used to study and design FL algorithms is the Euclidean space $\mathbb{R}^d$. We, therefore, expect some familiarity with the algebraic and geometric structure of $\mathbb{R}^d$. By algebraic structure, we mean the (real) vector space obtained from the elements ("vectors") in $\mathbb{R}^d$, along with the usual definitions of vector addition and multiplication by scalars in $\mathbb{R}$ [23, 24]. We will heavily use concepts from linear algebra to represent and manipulate data and ML models.

The metric structure of $\mathbb{R}^d$ is an excellent analysis tool for 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 $\mathbb{R}^d$. These operators are defined by the data (distribution) and ML models used within a FL system. A prime example of such a non-linear operator is the gradient step of gradient-based methods (see Lecture 4). The computational properties (such as convergence speed) of these FL algorithms can then be characterized via the contraction properties of the underlying operator [25].

A main tool for the design the FL algorithms are variants of gradient descent (GD). The common idea of these gradient-based methods is to approximate a function $f(\mathbf{x})$ locally by a linear function. This local linear approximation is determined by the gradient $\nabla f(\mathbf{x})$. We, therefore, expect some familiarity with multivariable calculus [5].

1.4 Related Courses

In what follows we briefly explain how CS-E4740 relates to selected courses at Aalto University and University of Helsinki.

- **CS-EJ3211 - Machine Learning with Python.** Teaches the application of basic ML methods using the Python package (library) `scikit-learn` [26]. 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 adaptively pool local datasets into sufficiently large training set for the personalized ML model.

- **Data Analysis with Python.** Can be considered as a substitute for CS-EJ3211.

- **CS-E4510 - Distributed Algorithms.** Teaches basic mathematical tools for the study and design of distributed algorithms that are implemented via distributed systems (computers) [27]. FL is enabled by distributed algorithms to train ML models from decentralized data (see Lecture 5).

- **CS-C3240 - Machine Learning (spring 2022 edition).** Teaches basic theory of ML models and methods [4]. 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.** 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.

- **MS-C2105 - Introduction to Optimization.** 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 Lecture 3) and design FL methods (see Lecture 5).

- **ELEC-E5424 - Convex Optimization.** Teaches advanced optimisation theory for the important class of convex optimization problems [28]. Convex optimization theory and methods can be used for the study and design of FL algorithms.

- **ELEC-E7120 - Wireless Systems.** Teaches the fundamentals of radio communications used in cellular and wireless systems. These systems provide important computational infrastructures for the implementation of FL algorithms (see Lecture 5).

1.5 Main Goal of the Course

The overarching goal of the course is to demonstrate how to apply concepts from graph theory and mathematical optimization to analyze and design FL algorithms. Students will learn to formulate a given FL application as an optimization problem over an undirected empirical graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ whose

nodes $i \in \mathcal{V}$ represent individual local datasets. We refer to this graph as the empirical graph of a collection of local datasets (see Lecture 3).

This course uses only undirected empirical graphs with a finite number n of nodes, which we identify with the first n positive integers:

$$\mathcal{V} := \{1, \ldots, n\}.$$

An edge $\{i, i'\} \in \mathcal{E}$ in the empirical graph $\mathcal{G}$ connects two different local datasets if they have similar statistical properties. We quantify the amount of similarity by the positive edge weight $A_{i,i'} > 0$.

We can formalize a FL application as an optimization problem associated with an empirical graph,

$$\min_{\mathbf{w}^{(i)}} \sum_{i \in \mathcal{V}} L_i \left(\mathbf{w}^{(i)} \right) + \alpha \sum_{\{i,i'\} \in \mathcal{E}} A_{i,i'} d^{(\mathbf{w}^{(i)}, \mathbf{w}^{(i')})}. \tag{1.1}$$

We refer to this problem as GTV minimization (GTVMin) and devote much of the course to its computational and statistical properties. The optimization variables $\mathbf{w}^{(i)}$ in (1.1) are local model parameters at the nodes $i \in \mathcal{V}$ of an empirical graph. The objective function in (1.1) 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 discrepancies $d^{(\mathbf{w}^{(i)}, \mathbf{w}^{(i')})}$ across the edges $\{i, i'\}$ of the empirical graph.

1.6 Outline of the Course

Our course is roughly divided into three parts:

- **Part I: ML Refresher.** Lecture 2 introduces data, models and loss functions as three main components of ML. This lecture also explains how these components are combined within empirical risk minimization (ERM). We also discuss how regularization of ERM can be achieved via manipulating its three main components. We then explain when and how to solve regularized ERM via simple GD methods in Lecture 4. Overall, this part 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 particularly relevant for the design and analysis of FL methods.

- **Part II: FL Theory and Methods.** Lecture 3 introduces the empirical graph as our main mathematical structure for representing collections of local datasets and corresponding tailored models. The undirected and weighted edges of the empirical graph represent statistical similarities between local datasets. Lecture 3 also 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. Lecture 4 discusses variations of gradient descent as our main algorithmic toolbox for solving GTVMin. Lecture 5 shows how FL algorithms can be obtained in a principled fashion by applying optimization methods, such as gradient-based methods, to GTVMin. We will obtain FL algorithms that can be implemented

as iterative message passing methods for the distributed training of tailored ("personalized") models. Lecture 6 derives some main flavours of FL as special cases of GTVMin. The usefulness of GTVMin crucially depends on the choice for the weighted edges in the empirical graph. Lecture 7 discusses graph learning methods that determine a useful empirical graph via different notions of statistical similarity between local datasets.

- **Part III: Trustworthy AI.** Lecture 8 enumerates seven key requirements for trustworthy artificial intelligence (AI) that have been put forward by the European Union. These key requirements include the protection of privacy as well as robustness against (intentional) perturbations of data or computation. We then discuss how FL algorithms can ensure privacy protection in Lecture 9. Lecture 10 discusses how to evaluate and ensure robustness of FL methods against intentional perturbations (poisoning) of local dataset.

1.7 Assignments and Grading

The course includes coding assignments that require you to implement concepts from the lecture in a Python notebook. We will use MyCourses quizzes to test your understanding of lecture contents and solutions to the coding assignments. For each quiz, you can earn around 10 points. We will sum up the points collected during the quizzes (no minimum requirements for any individual quiz) and determine your grade according to: **grade 1 for 50-59 points; grade 2 for 60-69; grade 3 for 70-79; grade 4 for 80-89 and top grade**

5 for at least 90 points. Students will be able to review their assignments' grading at the end of the course.

1.8 Student Project

You can extend the basic variant (which is worth 5 credits) to 10 credits by completing a student project and peer review. This project requires you to formulate an application of your choice as a FL problem using the concepts from this course. You then have to solve this FL problem using the FL algorithms taught in this course. The main deliverable will be a project report which must follow the structure indicated in the template. You will then peer-review the reports of your fellow students by answering a detailed questionnaire.

1.9 Schedule

The course lectures are held on Mo. and Wed. at 16.15, during 28-Feb-2024 until 30-Apr-2024. You can find the detailed schedule and lecture halls following this link. As the course can be completed fully remote, we will record each lecture and add the recording to the YouTube playlist here.

After each lecture, we release the corresponding assignment at this site. You have few days to work on the assignment before we open the corresponding quiz on the MyCourses page of the course (click me).

The assignments and quizzes are somewhat fast paced to encourage students to actively work on the course. We will also be strict about the deadlines for the quizzes. **However, students have the possibility to cover up for lost quiz points during a review meeting at the end of course.**

Moreover, active participation such as contribution to the discussion forum or providing feedback on the lecture notes will be taken into account.

1.10 Ground Rules

Note that as a student following this course, you must act according to the Code of Conduct of Aalto University (see here). Two main ground rules for this course are:

- **BE HONEST.** This course includes many tasks that require independent work, including the coding assignments, the working on student projects and the peer review of student projects. You must not use other's work inappropriately. For example, it is not allowed to copy other's solutions to coding assignments. We will randomly choose students who have to explain their solutions (and corresponding answers to quiz questions).

- **BE RESPECTFUL.** My personal wish is that this course provides a safe space for an enjoyable learning experience. Any form of disrespectful behaviour, including any course-related communication platforms, will be sanctioned rigorously (including reporting to university authorities).

2 Lecture - "ML Basics"

This lecture covers basic ML techniques that are instrumental for FL. This lecture is significantly more extensive content-wise compared to the following lectures. However, this lecture should be considerably easier to follow than the following lectures as it mainly refreshes pre-requisite knowledge.

2.1 Learning Goals

After this lecture, you should

- be familiar with the concept of data points (their features and labels), model and loss function,
- be familiar with ERM as a design principle for ML systems,
- know why and how validation is performed,
- be able to diagnose ML methods by comparing training error with validation error,
- be able to regularize ERM via modifying data, model and loss.

2.2 Three Components and a Design Principle

Machine Learning (ML) revolves around learning a hypothesis map h out of a hypothesis space $\mathcal{H}$ that allows to accurately predict the label of a data point solely from its features. One of the most crucial steps in applying ML methods to a given application domain is the definition or choice of what precisely a data point is. Coming up with a good choice or definition of data

points is not trivial as it influences the overall performance of a ML method in many different ways.

This course will focus mainly on one specific choice for the data points. In particular, we will consider data points that represent the daily weather conditions around FMI weather stations. We denote a specific data point by $\mathbf{z}$. It is characterized by the following features:

- name of the FMI weather station, e.g., "TurkuRajakari"
- latitude lat and longitude lon of the weather station, e.g., $\mathsf{lat} := 60.37788$, $\mathsf{lon} := 22.0964$,
- timestamp of the measurement in the format YYYY-MM-DD HH:MM:SS, e.g., 2023-12-31 18:00:00

It is convenient to stack the features into a feature vector $\mathbf{x}$. The label $y \in \mathbb{R}$ of such a data point is the maximum daytime temperature in degree Celsius, e.g., -20.

We predict the label of a data point with features $\mathbf{x}$ by the function value $h(\mathbf{x})$ of a hypothesis (map) $h(\cdot)$. The prediction will typically be not perfect, i.e., $h(\mathbf{x}) \neq y$. ML methods use a loss function $L\left(\mathbf{z}, h\right)$ to measure the prediction error. The choice of loss function crucially influences the statistical and computational properties of the resulting ML method. In what follows, unless stated otherwise, we use the squared error loss $L\left(\mathbf{z}, h\right) := \big(y - h(\mathbf{x})\big)^2$ to measure the prediction error.

It seems natural to choose (or learn) a hypothesis that incurs minimum average loss (or empirical risk) on a given set of data points $\mathcal{D} :=$

$\big\{\big(\mathbf{x}^{(1)}, y^{(1)}\big), \ldots, \big(\mathbf{x}^{m}, y^{(m)}\big)\big\}$. This is known as ERM,

$$\hat{h} \in \operatorname*{argmin}_{h \in \mathcal{H}} (1/m) \sum_{r=1}^{m} \big(y^{(r)} - h\big(\mathbf{x}^{(r)}\big)\big)^2 \tag{2.1}$$

As the notation in (2.1) indicates (using the symbol "$\in$" instead of ":="), there might be several different solutions to the optimization problem (2.1). Unless specified otherwise, $\hat{h}$ can be used to denote any hypothesis in $\mathcal{H}$ that has minimum average loss over $\mathcal{D}$.

Several important machine learning (ML) methods use a parametrized model $\mathcal{H}$: Each hypothesis $h \in \mathcal{H}$ is defined by parameters $\mathbf{w} \in \mathbb{R}^d$, often indicated by the notation $h^{(\mathbf{w})}$. A prominent instance of such a parametrized model is the linear model [4, Sec. 3.1],

$$\mathcal{H}^{(d)} := \big\{h(\mathbf{x}) := \mathbf{w}^T \mathbf{x}\big\}. \tag{2.2}$$

Linear regression methods, for example, learn the parameters of a linear model by minimizing the average squared error loss. For linear regression, ERM becomes an optimization over the parameter space $\mathbb{R}^d$,

$$\widehat{\mathbf{w}}^{(\mathrm{LR})} \in \operatorname*{argmin}_{\mathbf{w} \in \mathbb{R}^d} \underbrace{(1/m) \sum_{r=1}^{m} \big(y^{(r)} - \mathbf{w}^T \mathbf{x}^{(r)}\big)^2}_{:= f(\mathbf{w})}. \tag{2.3}$$

Note that (2.3) amounts to finding the minimum of a smooth and convex function

$$f(\mathbf{w}) = (1/m)\Big[\mathbf{w}^T \mathbf{X}^T \mathbf{X} \mathbf{w} - 2\mathbf{y}^T \mathbf{X} \mathbf{w} + \mathbf{y}^T \mathbf{y}\Big] \tag{2.4}$$

$$\text{with the feature matrix } \mathbf{X} := \big(\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)}\big)^T \tag{2.5}$$

$$\text{and the label vector } \mathbf{y} := \big(y^{(1)}, \ldots, y^{(m)}\big)^T \text{ of the training set } \mathcal{D}. \tag{2.6}$$

Inserting (2.4) into (2.3) allows to formulate linear regression as

$$\widehat{\mathbf{w}}^{(\mathrm{LR})} \in \operatorname*{argmin}_{\mathbf{w}\in\mathbb{R}^d} \mathbf{w}^T\mathbf{Q}\mathbf{w} + \mathbf{w}^T\mathbf{q} \tag{2.7}$$
$$\text{with } \mathbf{Q} := (1/m)\mathbf{X}^T\mathbf{X}, \mathbf{q} := -(2/m)\mathbf{X}^T\mathbf{y}.$$

The matrix $\mathbf{Q} \in \mathbb{R}^{d\times d}$ is psd with corresponding eigenvalue decomposition (EVD),

$$\mathbf{Q} = \sum_{j=1}^{d} \lambda_j\big(\mathbf{Q}\big)\mathbf{u}^{(j)}\big(\mathbf{u}^{(j)}\big)^T. \tag{2.8}$$

The EVD (2.8) involves non-negative eigenvalues

$$0 \leq \lambda_1\big(\mathbf{Q}\big) \leq \ldots \leq \lambda_d\big(\mathbf{Q}\big). \tag{2.9}$$

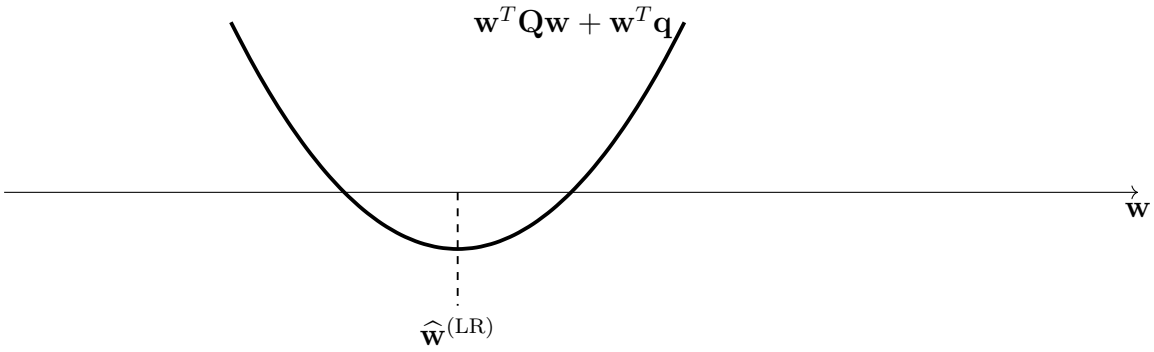


Figure 2.1: ERM (2.1) for linear regression minimizes a convex quadratic function $\mathbf{w}^T\mathbf{Q}\mathbf{w} + \mathbf{w}^T\mathbf{q}$.

To train a ML model $\mathcal{H}$ means to solve ERM (2.1) (or (2.3) for linear regression); the dataset $\mathcal{D}$ is therefore referred to as a training set. The trained model results in the learnt hypothesis $\hat{h}$. We obtain practical ML methods by applying optimization algorithms to solve (2.1). Two key questions studied in ML are:

- **computational aspects** How much computation do we need to solve (2.1) ?

- **statistical aspects** How useful is the solution $\hat{h}$ to (2.1) in general, i.e., how accurate is the prediction $\hat{h}(\mathbf{x})$ for the label y of an **arbitrary** data point with features $\mathbf{x}$?

2.3 Computational Aspects of ERM

ML methods use optimization algorithms to solve (2.1) in order to learn a hypothesis $\hat{h}$. Within this course, we use optimization algorithms that are iterative methods: Starting from an initial choice $h^{(0)}$, they construct a sequence

$$h^{(0)}, h^{(1)}, h^{(2)}, \ldots,$$

which are hopefully increasingly accurate approximations to a solution $\hat{h}$ of (2.1). The computational complexity of such a ML method can be measured by the number of iterations required to guarantee some prescribed level of approximation.

For a parametrized model and a smooth loss function, we can solve (2.3) by gradient-based methods: Starting from an initial parameters $\mathbf{w}^{(0)}$, we iterate the gradient step:

$$\begin{aligned} \mathbf{w}^{(k)} &:= \mathbf{w}^{(k-1)} - \eta \nabla f\big(\mathbf{w}^{(k-1)}\big) \\ &= \mathbf{w}^{(k-1)} + (2\eta/m) \sum_{r=1}^{m} \mathbf{x}^{(r)} \big(y^{(r)} - \big(\mathbf{w}^{(k-1)}\big)^T \mathbf{x}^{(r)}\big). \end{aligned} \tag{2.10}$$

How much computation do we need for one iteration of (2.10)? How many iterations do we need ? We will try to answer the latter question in Lecture

4. The first question can be answered more easily for typical computational infrastructure (e.g., "Python running on a commercial Laptop"). Indeed, a naive evaluation of (2.10) requires around m arithmetic operations (addition, multiplication).

It is instructive to consider the special case of a linear model which does not use any feature, i.e., $h(\mathbf{x}) = w$. For this extreme case, the ERM (2.3) has a simple closed-form solution:

$$\widehat{w} = (1/m) \sum_{r=1}^{m} x^{(r)}. \tag{2.11}$$

Thus, for this special case of the linear model, solving (2.11) amounts to summing m numbers $x^{(1)}, \ldots, x^{(m)}$. It seems reasonable to assume that the amount of computation required to compute (2.11) is proportional to m.

2.4 Statistical Aspects of ERM

We can train a linear model on a given training set as ERM (2.3). But how useful is the solution $\widehat{\mathbf{w}}$ of (2.3) for predicting the labels of data points outside the training set? Consider applying the learnt hypothesis $h^{(\widehat{\mathbf{w}})}$ to an arbitrary data point not contained in the training set. What can we say about the resulting prediction error $y - h^{(\widehat{\mathbf{w}})}(\mathbf{x})$ in general? In other words, how well does $h^{(\widehat{\mathbf{w}})}$ generalize beyond the training set.

Maybe the most widely used approach to study generalization of ML methods is via probabilistic models. Here, we interpret each data point as a realization of an i.i.d. RV with probability distribution $p(\mathbf{x}, y)$. Under this i.i.d. assumption, we can evaluate the overall performance of a hypothesis

$h \in \mathcal{H}$ via the expected loss (or risk)

$$\mathbb{E}\{L\left((\mathbf{x}, y), h\right)\}. \tag{2.12}$$

One example for a probability distribution $p(\mathbf{x}, y)$ relates the label y with the features $\mathbf{x}$ of a data point as

$$y = \overline{\mathbf{w}}^T \mathbf{x} + \varepsilon \text{ with } \mathbf{x} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), \varepsilon \sim \mathcal{N}(0, \sigma^2), \mathbb{E}\{\varepsilon \mathbf{x}\} = \mathbf{0}. \tag{2.13}$$

A simple calculation reveals the expected squared error loss of a given linear hypothesis $h(\mathbf{x}) = \mathbf{x}^T \widehat{\mathbf{w}}$ as[1]

$$\mathbb{E}\{(y - h(\mathbf{x}))^2\} = \|\overline{\mathbf{w}} - \widehat{\mathbf{w}}\|^2 + \sigma^2. \tag{2.14}$$

The component σ^2 can be interpreted as intrinsic noise level of the label y. We cannot hope to find a hypothesis with expected loss smaller than this level. The first component of the RHS in (2.14) is the estimation error $\|\overline{\mathbf{w}} - \widehat{\mathbf{w}}\|^2$ of a ML method that reads in the training set and delivers an estimate $\widehat{\mathbf{w}}$ (e.g., via (2.3)) for the parameters of a linear hypothesis.

We next study the estimation error $\overline{\mathbf{w}} - \widehat{\mathbf{w}}$ incurred by the specific estimate $\widehat{\mathbf{w}} = \widehat{\mathbf{w}}^{(\mathrm{LR})}$ (2.7) delivered by linear regression methods. To this end, we first use the probabilistic model (2.13) to decompose the label vector $\mathbf{y}$ in (2.6) as

$$\mathbf{y} = \mathbf{X}\overline{\mathbf{w}} + \mathbf{n} \text{ , with } \mathbf{n} := \left(\varepsilon^{(1)}, \ldots, \varepsilon^{(m)}\right)^T. \tag{2.15}$$

[1]Strictly speaking, the relation (2.14) only applies for constant (deterministic) parameters $\widehat{\mathbf{w}}$ that do not depend on the RVs whose realizations are the observed data points (see, e.g., (2.13)). However, the parameters $\widehat{\mathbf{w}}$ might be the output of a ML method (such as (2.3)) that is applied to a dataset $\mathcal{D}$ consisting of realizations i.i.d. RVs. In this case, we need to replace the expectation on the LHS of (2.14) with a conditional expectation $\mathbb{E}\big\{(y - h(\mathbf{x}))^2 \big| \mathcal{D}\big\}$.

Inserting (2.15) into (2.7) yields

$$\widehat{\mathbf{w}}^{(\mathrm{LR})} \in \operatorname*{argmin}_{\mathbf{w}\in\mathbb{R}^d} \mathbf{w}^T\mathbf{Q}\mathbf{w} + \mathbf{w}^T\mathbf{q}' + \mathbf{w}^T\mathbf{e} \tag{2.16}$$

$$\text{with } \mathbf{Q} := (1/m)\mathbf{X}^T\mathbf{X}, \mathbf{q}' := -(2/m)\mathbf{X}^T\mathbf{X}\overline{\mathbf{w}}, \text{ and } \mathbf{e} := -(2/m)\mathbf{X}^T\mathbf{n}. \tag{2.17}$$

Figure 2.2 depicts the objective function of (2.16). It is a perturbation of the convex quadratic function $\mathbf{w}^T\mathbf{Q}\mathbf{w} + \mathbf{w}^T\mathbf{q}'$, which is minimized at $\mathbf{w} = \overline{\mathbf{w}}$. In general, the minimizer $\widehat{\mathbf{w}}^{(\mathrm{LR})}$ delivered by linear regression is different from $\overline{\mathbf{w}}$ due the perturbation term $\mathbf{w}^T\mathbf{e}$ in (2.16).

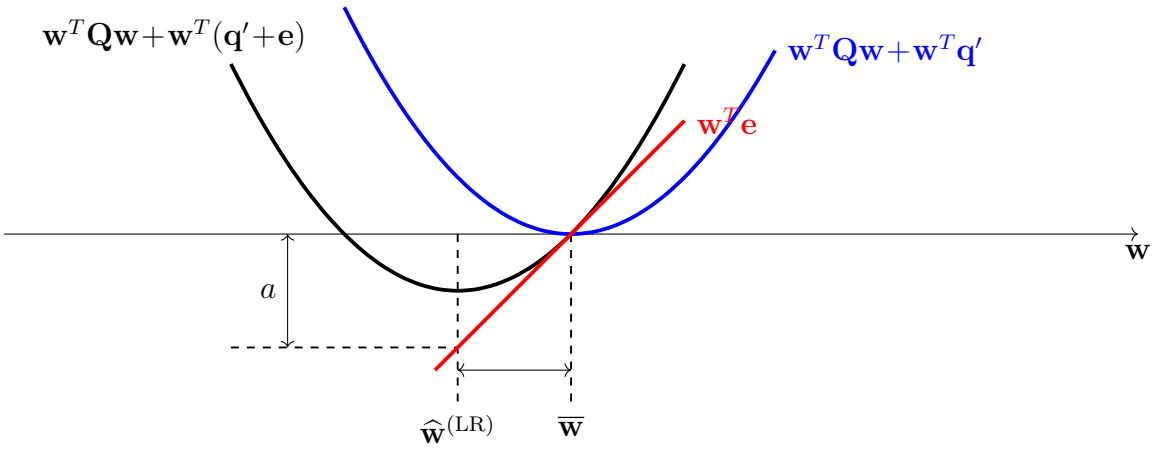


Figure 2.2: The estimation error of linear regression is determined by the effect of the perturbation term $\mathbf{w}^T\mathbf{e}$ on the minimizer of the convex quadratic function $\mathbf{w}^T\mathbf{Q}\mathbf{w} + \mathbf{w}^T\mathbf{q}'$.

The following result bounds the deviation between $\widehat{\mathbf{w}}^{(\mathrm{LR})}$ and $\overline{\mathbf{w}}$ under the assumption that the matrix $\mathbf{Q} = (1/m)\mathbf{X}^T\mathbf{X}$ is invertible.[2]

[2]Can you think of sufficient conditions on the feature matrix of the training set that ensure $\mathbf{Q} = (1/m)\mathbf{X}^T\mathbf{X}$ is invertible?

Proposition 2.1. *Consider a solution* $\widehat{\mathbf{w}}^{(\mathrm{LR})}$ *to the ERM instance* (2.16) *that is applied to the dataset* (2.15)*. If the matrix* $\mathbf{Q} = (1/m)\mathbf{X}^T\mathbf{X}$ *is invertible, with minimum eigenvalue* $\lambda_1(\mathbf{Q}) > 0$*,*

$$\big\|\widehat{\mathbf{w}}^{(\mathrm{LR})} - \overline{\mathbf{w}}\big\|_2^2 \leq \|\mathbf{e}\|_2^2/\lambda_1^2 \stackrel{(2.17)}{=} (4/m^2)\left\|\mathbf{X}^T\mathbf{n}\right\|_2^2/\lambda_1^2. \tag{2.18}$$

Proof. Let us rewrite (2.16) as

$$\widehat{\mathbf{w}}^{(\mathrm{LR})} \in \operatorname*{argmin}_{\mathbf{w}\in\mathbb{R}^d} f(\mathbf{w}) \text{ with } f(\mathbf{w}) := \big(\mathbf{w}-\overline{\mathbf{w}}\big)^T\mathbf{Q}\big(\mathbf{w}-\overline{\mathbf{w}}\big)+\mathbf{e}^T\big(\mathbf{w}-\overline{\mathbf{w}}\big). \tag{2.19}$$

Clearly $f\big(\overline{\mathbf{w}}\big) = 0$ and, in turn, $f(\widehat{\mathbf{w}}) = \min_{\mathbf{w}\in\mathbb{R}^d} f(\mathbf{w}) \leq 0$. On the other hand,

$$\begin{aligned} f(\mathbf{w}) &\stackrel{(2.19)}{=} \big(\mathbf{w}-\overline{\mathbf{w}}\big)^T\mathbf{Q}\big(\mathbf{w}-\overline{\mathbf{w}}\big) + \mathbf{e}^T\big(\mathbf{w}-\overline{\mathbf{w}}\big) \\ &\stackrel{(a)}{\geq} \big(\mathbf{w}-\overline{\mathbf{w}}\big)^T\mathbf{Q}\big(\mathbf{w}-\overline{\mathbf{w}}\big) - \|\mathbf{e}\|_2\,\|\mathbf{w}-\overline{\mathbf{w}}\|_2 \\ &\stackrel{(b)}{\geq} \lambda_1\,\|\mathbf{w}-\overline{\mathbf{w}}\|_2^2 - \|\mathbf{e}\|_2\,\|\mathbf{w}-\overline{\mathbf{w}}\|_2\,. \end{aligned} \tag{2.20}$$

Step (a) used Cauchy–Schwarz inequality and (b) used the EVD (2.8) of $\mathbf{Q}$. Evaluating (2.20) for $\mathbf{w}=\widehat{\mathbf{w}}$ and combining with $f\big(\widehat{\mathbf{w}}\big) \leq 0$ yields (2.18). □

The bound (2.18) suggests that the estimation error $\widehat{w}^{(\mathrm{LR})} - \overline{w}$ is small if $\lambda_1(\mathbf{Q})$ is large. This smallest eigenvalue of the matrix $\mathbf{Q} = (1/m)\mathbf{X}^T\mathbf{X}$ could be controlled by suitable choice (or transformation) of features $\mathbf{x}$ of a data point. Trivially, we can increase $\lambda_1(\mathbf{Q})$ by a factor 100 if we scale each feature by a factor 10. However, this approach would also scale (by factor 100) the error term $\left\|\mathbf{X}^T\mathbf{n}\right\|_2^2$ in (2.18). For some applications, we can find feature transformations ("whitening") that increases $\lambda_1(\mathbf{Q})$ but does not increase $\left\|\mathbf{X}^T\mathbf{n}\right\|_2^2$. We finally note that the error term $\left\|\mathbf{X}^T\mathbf{n}\right\|_2^2$ in (2.18) vanishes if the noise vector $\mathbf{n}$ is orthogonal to the columns of the feature matrix $\mathbf{X}$.

It is instructive to evaluate the bound (2.18) for the special case where each data point has the same feature $x = 1$. Here, the probabilistic model (2.15) reduces to a "signal in noise" model,

$$y^{(r)} = x^{(r)}\overline{w} + \varepsilon^{(r)} \text{ with } x^{(i)} = 1, \tag{2.21}$$

with some true underlying parameter $\overline{w}$. The noise terms $\varepsilon^{(r)}$, for $r = 1, \ldots, m$. are realizations of i.i.d. RVs with probability distribution $\mathcal{N}(0, \sigma^2)$. The feature matrix then becomes $\mathbf{X} = \mathbf{1}$ and, in turn, $\mathbf{Q} = 1$, $\lambda_1(\mathbf{Q}) = 1$. Inserting these values into (2.18), results in the bound

$$\left(\widehat{w}^{(\mathrm{LR})} - \overline{w}\right)^2 \leq 4 \left\|\mathbf{n}\right\|_2^2 / m^2. \tag{2.22}$$

Note that, for the labels and features in (2.21), the solution of (2.16) is given by

$$\widehat{w}^{(\mathrm{LR})} = (1/m) \sum_{r=1}^{m} y^{(r)} \overset{(2.21)}{=} \overline{w} + (1/m) \sum_{r=1}^{m} \varepsilon^{(r)}. \tag{2.23}$$

2.5 Validation and Diagnosis of ML

The above analysis of the generalization error started from postulating a probabilistic model for the generation of data points. However, this probabilistic model might be wrong and the bound (2.18) does not apply. Thus, we might want to use a more data-driven approach for assessing the usefulness of a learnt hypothesis.

Validation methods try to find out if a learnt hypothesis $\hat{h}$, does well outside the training set. In its most basic form, model validation amounts to computing the average loss of a learnt hypothesis $\hat{h}$ on some data points not included in the training set. We refer to these data points as the validation set.

Algorithm 2.1 summarizes a prototypical workflow of ML model training and validation. This workflow starts from choosing a dataset $\mathcal{D}$, model $\mathcal{H}$ and loss function $L\left(\cdot,\cdot\right)$. In general, we execute Algorithm 2.1 several times, each time using a (possibly) different choice for dataset, model and loss function. The adaptation of the design choices for Algorithm 2.1 can be based on some basic diagnosis.

Algorithm 2.1 One Iteration of ML Training and Validation

Input: dataset $\mathcal{D}$, model $\mathcal{H}$, loss function $L\left(\cdot,\cdot\right)$

1: split $\mathcal{D}$ into a training set $\mathcal{D}^{(\text{train})}$ and a validation set $\mathcal{D}^{(\text{val})}$

2: learn a hypothesis via solving ERM

$$\widehat{h} \in \operatorname*{argmin}_{h \in \mathcal{H}} \sum_{(\mathbf{x},y) \in \mathcal{D}^{(\text{train})}} L\left((\mathbf{x},y),h\right) \tag{2.24}$$

3: compute resulting training error

$$E_t := (1/|\mathcal{D}^{(\text{train})}|) \sum_{(\mathbf{x},y) \in \mathcal{D}^{(\text{train})}} L\left((\mathbf{x},y),\widehat{h}\right)$$

4: compute validation error

$$E_v := (1/|\mathcal{D}^{(\text{val})}|) \sum_{(\mathbf{x},y) \in \mathcal{D}^{(\text{val})}} L\left((\mathbf{x},y),\widehat{h}\right)$$

Output: learnt hypothesis (or trained model) $\widehat{h}$, training error E_t and validation error E_v

We can diagnose a ERM-based ML method, such as Algorithm 2.1, by comparing its training error with its validation error. This diagnosis is further enabled if we know a baseline $E^{(\text{ref})}$. One important source for a baseline

$E^{(\mathrm{ref})}$ are probabilistic models for the data points (see Section 2.4).

Given a probabilistic model $p(\mathbf{x}, y)$, we can compute the minimum achievable risk (2.12). Indeed, the minimum achievable risk is precisely the expected loss of the Bayes estimator $\widehat{h}(\mathbf{x})$ of the label y, given the features $\mathbf{x}$ of a data point. The Bayes estimator $\widehat{h}(\mathbf{x})$ is fully determined by the probability distribution $p(\mathbf{x}, y)$ [29, Chapter 4].

A further potential source for a baseline $E^{(\mathrm{ref})}$ is an existing, but for some reason unsuitable, ML method. This existing ML method might be computationally too expensive to be used for the ML application at hand. However, we might still use its statistical properties as a benchmark.

We can also use the performance of human experts as a baseline. If we want to develop a ML method that detects certain type of skin cancers from images of the skin, a benchmark might be the classification accuracy achieved by experienced dermatologists [30].

We can diagnose a ML method by comparing the training error E_t with the validation error E_v and (if available) the benchmark $E^{(\mathrm{ref})}$.

- $E_t \approx E_v \approx E^{(\mathrm{ref})}$: The training error is on the same level as the validation error error and the baseline. There is not much to improve here since the validation error is already close to the baseline. Moreover, the training error is not much smaller than the validation error which indicates that there is no overfitting.

- $E_v \gg E_t$: The validation error is significantly larger than the training error. This is an indicator for overfitting. We can address overfitting by reducing the effective dimension of the hypothesis space or by increasing the size of the training set. We can reduce the effective dimension

of the hypothesis space by using fewer features, a smaller maximum depth of decision trees or fewer layers in an ANN. An alternative to this discrete model selection, we can also reduce the effective dimension of a hypothesis space via regularization.

- $E_t \approx E_v \gg E^{(\text{ref})}$: The training error is on the same level as the validation error and both are significantly larger than the baseline. Thus, the learnt hypothesis seems to not overfit the training set. However, the training error achieved by the learnt hypothesis is significantly larger than the baseline. There can be several reasons for this to happen. First, it might be that the hypothesis space is too small, i.e., it does not include a hypothesis that provides a good approximation for the relation between features and label of a data point. One remedy to this situation is to use a larger hypothesis space, e.g., by including more features in a linear model, using higher polynomial degrees in polynomial regression, using deeper decision trees or ANNs (deep ANN (deep net)s). Second, besides the model being too small, another reason for a large training error could be that the optimization algorithm used to solve ERM (2.24) is not working properly (see Lecture 4).

- $E_t \gg E_v$: The training error is significantly larger than the validation error. The idea of ERM (2.24) is to approximate the risk (2.12) of a hypothesis by its average loss on a training set $\mathcal{D} = \{(\mathbf{x}^{(r)}, y^{(r)})\}_{r=1}^{m}$. The mathematical underpinning for this approximation is the law of large numbers which characterizes the average of (realizations of) i.i.d. RVs. The accuracy of this approximation depends on the validity of two

conditions: First, the data points used for computing the average loss "should behave" like realizations of i.i.d. RVs with a common probability distribution. Second, the number of data points used for computing the average loss must be sufficiently large.

Whenever the training set or validation set differs significantly from realizations of i.i.d. RVs, the interpretation (and comparison) of the training error and the validation error of a learnt hypothesis becomes more difficult. As an extreme case, the validation set might consist of data points for which every hypothesis incurs small average loss (see Figure 2.3). Here, we might try to increase the size of the validation set by collecting more labeled data points or by using data augmentation (see Section 2.6). If the size of both training set and validation set is large but we still obtain $E_t \gg E_v$, one should verify if data points in these sets conform to the i.i.d. assumption. There are principled statistical test for the validity of the i.i.d. assumption for a given dataset (see [31] and references therein).

2.6 Regularization

Consider a ERM-based ML method using a hypothesis space $\mathcal{H}$ and dataset $\mathcal{D}$ (we assume all data points are used for training). A key parameter for such a ML method is the ratio $d_{\text{eff}}(\mathcal{H})/|\mathcal{D}|$ between the model size $d_{\text{eff}}(\mathcal{H})$ and the number $|\mathcal{D}|$ of data points. The tendency of the ML method to overfit increases with the ratio $d_{\text{eff}}(\mathcal{H})/|\mathcal{D}|$.

Regularization techniques reduce the ratio $d_{\text{eff}}(\mathcal{H})/|\mathcal{D}|$ via three (essen-

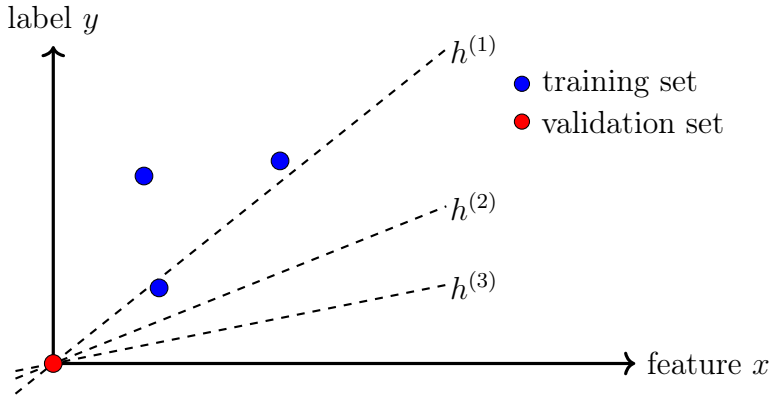


Figure 2.3: Example for an unlucky split into training set and validation set for the model $\mathcal{H} := \{h^{(1)}, h^{(2)}, h^{(3)}\}$.

tially equivalent) approaches:

- collect more data points, possibly via data augmentation (see Fig. 2.4),
- add penalty term $\alpha \mathcal{R}\big\{h\big\}$ to average loss in ERM (2.1) (see Fig. 2.4),
- shrink the hypothesis space, e.g., by adding constraints on the model parameters such as $\|\mathbf{w}\|_2 \leq 10$.

It can be shown that these three perspectives (corresponding to the three components data, model and loss) on regularization are closely related [4, Ch. 7]. For example, adding a penalty term $\alpha \mathcal{R}\big\{h\big\}$ in ERM (2.1) is equivalent to ERM (2.1) with a pruned hypothesis space $\mathcal{H}^{(\alpha)} \subseteq \mathcal{H}$. Using a larger α typically results in a smaller $\mathcal{H}^{(\alpha)}$.

One important example for regularization via adding a penalty term to the average loss is ridge regression. In particular, ridge regression uses the regularizer $\mathcal{R}\big\{h\big\} := \|\mathbf{w}\|_2^2$ for a linear hypothesis $h(\mathbf{x}) := \mathbf{w}^T\mathbf{x}$. Thus, ridge

regression learns the parameters of a linear hypothesis via solving

$$\widehat{\mathbf{w}}^{(\alpha)} \in \operatorname*{argmin}_{\mathbf{w}\in\mathbb{R}^d} \left[(1/m) \sum_{r=1}^{m} \left(y^{(r)} - \mathbf{w}^T \mathbf{x}^{(r)} \right)^2 + \alpha \left\| \mathbf{w} \right\|_2^2 \right]. \tag{2.25}$$

The objective function in (2.25) can be interpreted as the objective function of linear regression applied to a modification of the training set $\mathcal{D}$: We replace each data point $(\mathbf{x}, y) \in \mathcal{D}$ by a sufficient large number of i.i.d. realizations of

$$(\mathbf{x} + \mathbf{n}, y) \text{ with } \mathbf{n} \sim \mathcal{N}(\mathbf{0}, \alpha \mathbf{I}). \tag{2.26}$$

Thus, ridge regression (2.25) is equivalent to linear regression applied to an augmented variant $\mathcal{D}'$ of $\mathcal{D}$. The augmentation $\mathcal{D}'$ is obtained by replacing each data point $(\mathbf{x}, y) \in \mathcal{D}$ with a sufficiently large number of noisy copies. Each copy is obtained by adding a i.i.d. realization $\mathbf{n}$ of a zero-mean Gaussian noise with covariance matrix $\alpha \mathbf{I}$ to the features $\mathbf{x}$ (see (2.26)). The label of each copy is equal to y, i.e., the label is not perturbed.

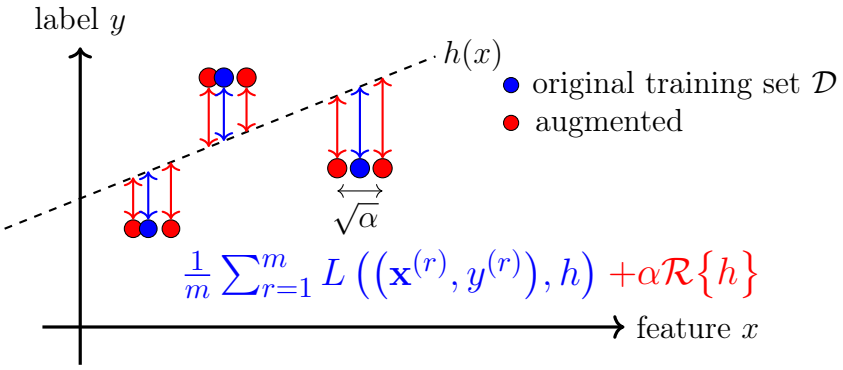


Figure 2.4: Equivalence between data augmentation and loss penalization.

To study the computational aspects of ridge regression, let us rewrite

(2.25) as

$$\widehat{\mathbf{w}}^{(\alpha)} \in \operatorname*{argmin}_{\mathbf{w}\in\mathbb{R}^d} \mathbf{w}^T\mathbf{Q}\mathbf{w} + \mathbf{w}^T\mathbf{q}$$
$$\text{with } \mathbf{Q} := (1/m)\mathbf{X}^T\mathbf{X} + \alpha\mathbf{I},\ \mathbf{q} := (-2/m)\mathbf{X}^T\mathbf{y}. \tag{2.27}$$

Thus, like linear regression (2.7), also ridge regression minimizes a convex quadratic function. A main difference between linear regression (2.7) and ridge regression (for $\alpha > 0$) is that the matrix $\mathbf{Q}$ in (2.27) is guaranteed to be invertible for any training set $\mathcal{D}$. In contrast, the matrix $\mathbf{Q}$ in (2.7) for linear regression might be singular for some training sets.[3]

The statistical aspects of the solutions to (2.27) (i.e., the parameters learnt by ridge regression) crucially depends on the value of α. This choice can be guided by an error analysis using a probabilistic model for the data (see Proposition 2.1). Instead of using a probabilistic model, we can also compare the training error and validation error of the hypothesis $h(\mathbf{x}) = \big(\widehat{\mathbf{w}}^{(\alpha)}\big)^T\mathbf{x}$ learnt by ridge regression with different values of α.

2.7 Overview of Coding Assignment

Python Notebook. `MLBasics_CodingAssignment.ipynb`

Data File. `Assignment_MLBasicsData.csv`

Description. The coding assignment revolves around weather data collected by the FMI and stored in a csv file. This file contains temperature measurements at different locations in Finland.

[3]Consider the extreme case where all features of each data point in the training set $\mathcal{D}$ are zero.

Each temperature measurement is a data point, characterized by $d = 7$ features $\mathbf{x} = \big(x_1, \ldots, x_7\big)^T$ and a label y which is the temperature measurement itself. The features are (normalized) values of the latitude and longitude of the FMI station as well as the (normalized) year, month, day, hour, minute during which this measurements has been taken. Your tasks include:

- Generate `numpy` arrays $\mathbf{X}, \mathbf{y}$, whose r-th row hold the features $\mathbf{x}^{(r)}$ and label $y^{(r)}$, respectively, of the r-th data point in the csv file.

- Split the dataset into a training set and validation set. The size of the training set should be 100.

- Train a linear model, using the `LinearRegression` class of the `scikit-learn` package, on the training set and determine the resulting training error and validation error

- Train and validate a linear model using feature augmentation via polynomial combinations (see `PolynomialFeatures` class). Try out different different choices of the maximal degree of these polynomial combinations.

- Using a fixed value for the polynomial degree for the feature augmentation step, train and validate a linear model using ridge regression (2.25) via the `Ridge` class.

- **CAUTION!** The input parameter `alpha` of the `Ridge` class has a different meaning than α in (2.25) In particular, the parameter α in (2.25) corresponds to using the value $m\alpha$ for `alpha`.

- Determine the resulting training error and validation error or the model parameters obtained for different values of α in (2.25).

3 Lecture - "FL Design Principle"

Lecture 2 reviewed ML methods that use numeric arrays to store data points (their features and labels) and model parameters. We have also discussed ERM (and its regularization) as a main design principle for practical ML systems. This lecture extends the basic ML concepts from a single-dataset single-model setting to FL applications involving distributed collections of data and models.

Section 3.2 introduces empirical graphs to store collections of local datasets and corresponding parameters of local models. Section 3.3 presents our main design principle for FL systems. This principle is a special case of ERM using a particular choice of loss function and model. In particular, we represent data and models via the nodes the "empirical graph". We penalize the loss incurred by individual local models by the variation of their parameters across the edges of an empirical graph. This penalization is an instance of regularization and couples the training of the individual local models..

3.1 Learning Goals

After this lecture, you should

- be familiar with the concept of an empirical graph,
- know how connectivity is related to the spectrum of the Laplacian matrix,
- know some measures for the variation of local models,
- be familiar with GTVMin as a formulation FL.

3.2 Empirical Graphs and Their Laplacian

Consider a collection of local datasets $\mathcal{D}^{(1)}, \ldots, \mathcal{D}^{(2)}$. Our goal is to train a personalized model $\mathcal{H}^{(i)}$ for each local dataset $\mathcal{D}^{(i)}$, with $i = 1, \ldots, n$. We represent such a collection of local datasets and (personal) local models, along with their relations, by an empirical graph. Figure 3.1 depicts an example for an empirical graph.

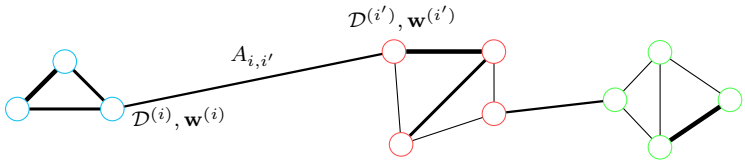


Figure 3.1: Example of an empirical graph whose nodes $i \in \mathcal{V}$ carry local datasets $\mathcal{D}^{(i)}$ and local models that are parametrized by local model parameters $\mathbf{w}^{(i)}$.

The empirical graph $\mathcal{G}$ is an undirected weighted graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with nodes $\mathcal{V} := \{1, \ldots, n\}$. Each node $i \in \mathcal{V}$ of the empirical graph $\mathcal{G}$ carries a local dataset

$$\mathcal{D}^{(i)} := \left\{ \left(\mathbf{x}^{(i,1)}, y^{(i,1)}\right), \ldots, \left(\mathbf{x}^{(i,m_i)}, y^{(i,m_i)}\right) \right\}. \tag{3.1}$$

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 $i \in \mathcal{V}$.

It is convenient to collect the feature vectors $\mathbf{x}^{(i,r)}$ and labels $y^{(i,r)}$ into a feature matrix $\mathbf{X}^{(i)}$ and label vector $\mathbf{y}^{(i)}$, respectively,

$$\mathbf{X}^{(i)} := \left(\mathbf{x}^{(i,1)}, \ldots, \mathbf{x}^{(i,m_i)}\right)^T, \text{ and } \mathbf{y} := \left(y^{(1)}, \ldots, y^{(m_i)}\right)^T. \tag{3.2}$$

The local dataset $\mathcal{D}^{(i)}$ can then be represented compactly by the feature matrix $\mathbf{X}^{(i)} \in \mathbb{R}^{m_i \times d}$ and the vector $\mathbf{y}^{(i)} \in \mathbb{R}^{m_i}$.

Besides the local dataset $\mathcal{D}^{(i)}$, each node $i \in \mathcal{G}$ also carries a local model $\mathcal{H}^{(i)}$. Our focus is on local models that can be parametrized by local model parameters $\mathbf{w}^{(i)} \in \mathbb{R}^d$, for $i = 1, \ldots, n$. The usefulness of a specific choice for the local model parameter $\mathbf{w}^{(i)}$ is measured by a local loss function $L_i\left(\mathbf{w}^{(i)}\right)$, for $i = 1, \ldots, n$. In principle, we can use different local loss functions $L_{i'}(\cdot) \neq L_{i'}(\cdot)$ at different nodes $i, i' \in \mathcal{V}$.

The empirical graph also contains undirected edges $\{i, i'\} \in \mathcal{E}$ between pairs of (different) nodes $i, i' \in \mathcal{V}$. We use an undirected edge $\{i, i'\} \in \mathcal{E}$ to couple the training of the corresponding local models $\mathcal{H}^{(i)}, \mathcal{H}^{(i')}$. The strength of this coupling is determined by a positive edge weight $A_{i,i'} > 0$. The coupling is implemented by penalizing the discrepancy between local model parameters $\mathbf{w}^{(i)}$ and $\mathbf{w}^{(i')}$ (see Section 3.3).

There are different ways to measure the discrepancy between the local model parameters $\mathbf{w}^{(i)}, \mathbf{w}^{(i')}$ at connected nodes $\{i, i'\} \in \mathcal{E}$. For example, we can use some cost or penalty function $\phi\big(\mathbf{w}^{(i)} - \mathbf{w}^{(i')}\big)$ that satisfies basic requirements such as being a (semi-)norm [32].

The choice of penalty $\phi(\cdot)$ has crucial impact on the computational and statistical properties of the FL methods arising from the design principle introduced in Section 3.3. Unless stated otherwise, we use the choice $\phi(\cdot) := \|\cdot\|_2^2$, i.e., we measure the discrepancy between local model parameters across an edge $\{i, i'\} \in \mathcal{E}$ by the squared Euclidean distance $\left\|\mathbf{w}^{(i)} - \mathbf{w}^{(i')}\right\|_2^2$. Summing the discrepancies (weighted by the edge weights) over all edges in the empirical

graph yields the total variation of local model parameters

$$\sum_{\{i,i'\}\in\mathcal{E}} A_{i,i'} \left\|\mathbf{w}^{(i)} - \mathbf{w}^{(i')}\right\|_2^2. \tag{3.3}$$

The connectivity of an empirical graph $\mathcal{G}$ can be characterized locally around a node $i \in \mathcal{V}$ by its weighted node degree

$$d^{(i)} := \sum_{i'\in\mathcal{N}^{(i)}} A_{i,i'}. \tag{3.4}$$

Here, we used the neighbourhood $\mathcal{N}^{(i)} := \{i' \in \mathcal{V} : \{i,i'\} \in \mathcal{E}\}$ of node $i \in \mathcal{V}$. A global characterization for the connectivity of $\mathcal{G}$ is the maximum weighted node degree

$$d_{\max} := \max_{i\in\mathcal{V}} d^{(i)} \overset{(3.4)}{=} \max_{i\in\mathcal{V}} \sum_{i'\in\mathcal{N}^{(i)}} A_{i,i'}. \tag{3.5}$$

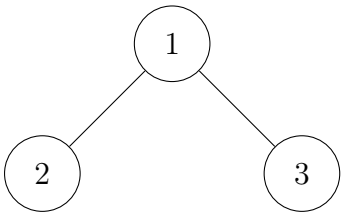
Besides the (maximum) node degrees, we can also analyze the connectivity of $\mathcal{G}$ via the eigenvalues and eigenvectors of its Laplacian matrix $\mathbf{L}^{(\mathcal{G})} \in \mathbb{R}^{n\times n}$. The Laplacian matrix is defined element-wise as (see Fig. 3.2)

$$L^{(\mathcal{G})}_{i,i'} := \begin{cases} -A_{i,i'} & \text{for } i \neq i', \{i,i'\} \in \mathcal{E} \\ \sum_{i''\neq i} A_{i,i''} & \text{for } i = i' \\ 0 & \text{else.} \end{cases} \tag{3.6}$$

The Laplacian matrix is symmetric and psd which follows from the identity

$$\mathbf{w}^T(\mathbf{L}^{(\mathcal{G})} \otimes \mathbf{I})\mathbf{w} = \sum_{\{i,i'\}\in\mathcal{E}} A_{i,i'} \left\|\mathbf{w}^{(i)} - \mathbf{w}^{(i')}\right\|_2^2$$

$$\text{for any } \mathbf{w} := \underbrace{\left(\left(\mathbf{w}^{(1)}\right)^T, \ldots, \left(\mathbf{w}^{(n)}\right)^T\right)^T}_{=:\text{stack}\left\{\mathbf{w}^{(i)}\right\}_{i=1}^n}. \tag{3.8}$$



$$\mathbf{L}^{(\mathcal{G})} = \begin{pmatrix} 2 & -1 & -1 \\ -1 & 1 & 0 \\ -1 & 0 & 1 \end{pmatrix} \tag{3.7}$$

Figure 3.2: Left: Example for an empirical graph $\mathcal{G}$ with three nodes $i = 1, 2, 3$. Right: Laplacian matrix $\mathbf{L}^{(\mathcal{G})} \in \mathbb{R}^{3\times 3}$ of $\mathcal{G}$.

As a psd matrix, $\mathbf{L}^{(\mathcal{G})}$ possesses an EVD

$$\mathbf{L}^{(\mathcal{G})} = \sum_{j=1}^{n} \lambda_j \mathbf{u}^{(j)} \big(\mathbf{u}^{(j)}\big)^T, \tag{3.9}$$

with increasingly ordered eigenvalues

$$0 = \lambda_1\big(\mathbf{L}^{(\mathcal{G})}\big) \leq \lambda_2\big(\mathbf{L}^{(\mathcal{G})}\big) \leq \ldots \leq \lambda_n\big(\mathbf{L}^{(\mathcal{G})}\big). \tag{3.10}$$

According to (3.8), we can measure the total variation of local model parameters by stacking them into a single vector $\mathbf{w} \in \mathbb{R}^{nd}$ and computing the quadratic form $\mathbf{w}^T\big(\mathbf{L}^{(\mathcal{G})} \otimes \mathbf{I}_{d\times d}\big)\mathbf{w}$.

One immediate consequence of (3.8) is that any collection of identical local model parameters

$$\mathbf{w} = \text{stack}\{\mathbf{c}\} = \big(\mathbf{c}^T, \ldots, \mathbf{c}^T\big)^T, \text{ with some } \mathbf{c} \in \mathbb{R}^d, \tag{3.11}$$

is an eigenvector of $\mathbf{L}^{(\mathcal{G})} \otimes \mathbf{I}$ with corresponding eigenvalue $\lambda_1 = 0$ (see (3.10)). Thus, the Laplacian matrix of any empirical graph is singular (non-invertible).

The second eigenvalue λ_2 of the Laplacian matrix provides a great deal of information about the connectivity structure of $\mathcal{G}$.[4]

[4]Much of spectral graph theory is devoted to the analysis of λ_2 for different graph constructions [33, 34].

- Consider the case $\lambda_2 = 0$: Here, we can find (beside (3.11)) another eigenvector

$$\widetilde{\mathbf{w}} = \mathrm{stack}\big\{\mathbf{w}^{(i)}\big\}_{i=1}^{n} \text{ with } \mathbf{w}^{(i)} \neq \mathbf{w}^{(i')} \text{ for some } i, i' \in \mathcal{V}, \tag{3.12}$$

  of $\mathbf{L}^{(\mathcal{G})} \otimes \mathbf{I}$ with eigenvalue equal to 0. In this case, the graph $\mathcal{G}$ is not connected, i.e., we can find two subsets (components) of nodes that do not have any edge between them. We can then obtain the eigenvector by assigning the same (non-zero) vector $\mathbf{c} \in \mathbb{R}^d \setminus \{\mathbf{0}\}$ to all nodes i belonging to the same connected component $\mathcal{C}$, and the zero vector $\mathbf{0}$ to other nodes $i \in \mathcal{V} \setminus \mathcal{C}$.

- On the other hand, if $\lambda_2 > 0$ then $\mathcal{G}$ is connected. Moreover, the larger the value of λ_2, the stronger the connectivity between the nodes in $\mathcal{G}$. We next show how to make this vague statement more precise via the identity (3.8).

The total variation on the RHS (3.8) is a measure for the connectivity: If the local model parameters $\mathbf{w}^{(i)}$ are different, then adding an edge will increase the total variation (3.8). Moreover, we can lower bound the total variation as [35]

$$\sum_{\{i,i'\} \in \mathcal{E}} A_{i,i'} \left\| \mathbf{w}^{(i)} - \mathbf{w}^{(i')} \right\|_2^2 \geq \lambda_2 \sum_{i=1}^{n} \left\| \mathbf{w}^{(i)} - \mathrm{avg}\{\mathbf{w}^{(i)}\} \right\|_2^2 . \tag{3.13}$$

Here, $\mathrm{avg}\{\mathbf{w}^{(i)}\} := (1/n) \sum_{i=1}^{n} \mathbf{w}^{(i)}$ is the average of all local model parameters. The bound (3.13) follows from (3.8) and the Courant–Fischer–Weyl min-max characterization [36, Thm. 8.1.2.] for the eigenvalues of the matrix $\mathbf{L}^{(\mathcal{G})} \otimes \mathbf{I}$.

The quantity $\sum_{i=1}^{n} \left\| \mathbf{w}^{(i)} - \mathrm{avg}\{\mathbf{w}^{(i)}\}_{i=1}^{n} \right\|_2^2$ on the RHS of (3.13) has an interesting geometric interpretation: It is the squared Euclidean norm of the pro-

jection of the stacked local model parameters $\mathbf{w} := \left(\left(\mathbf{w}^{(1)}\right)^T, \ldots, \left(\mathbf{w}^{(n)}\right)^T \right)^T$ on the orthogonal complement of the subspace

$$\left\{ \mathbf{1} \otimes \mathbf{a} : \mathbf{a} \in \mathbb{R}^d \right\} = \left\{ \left(\mathbf{a}^T, \ldots, \mathbf{a}^T\right)^T, \text{ for some } \mathbf{a} \in \mathbb{R}^d \right\} \subseteq \mathbb{R}^{dn}. \tag{3.14}$$

It might be convenient to replace a given empirical graph $\mathcal{G}$ with an equivalent fully connected empirical graph $\mathcal{G}'$ (see Figure 3.3). The graph $\mathcal{G}'$ has an edge between each pair of different nodes i, i',

$$\mathcal{E}' = \big\{ \{i, i'\} \text{ , with some } i, i' \in \mathcal{V}, i \neq i' \big\}.$$

The edge weights are chosen $A'_{i,i'} = A_{i,i'}$ for any edge $\{i, i'\} \in \mathcal{E}$ and $A'_{i,i'} = 0$ if the original empirical graph $\mathcal{G}$ does not contain an edge between nodes i, i'.

Figure 3.3: Left: Some empirical graph $\mathcal{G}$ consisting of $n = 4$ nodes. Right: Equivalent fully connected empirical graph $\mathcal{G}'$ with the same nodes and non-zero edge weights $A'_{i,i'} = A_{i,i'}$ for $\{i, i'\} \in \mathcal{E}$ and $A'_{i,i'} = 0$ for $\{i, i'\} \notin \mathcal{E}$.

Note that the undirected edges $\mathcal{E}$ 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', i.e., $\{i, i'\} \in \mathcal{E}$, then also the local dataset $\mathcal{D}^{(i')}$ is similar to the local dataset $\mathcal{D}^{(i)}$.

3.3 Generalized Total Variation Minimization

Consider some empirical graph $\mathcal{G}$ whose nodes $i \in \mathcal{V}$ carry local datasets $\mathcal{D}^{(i)}$ and local model parametrized by the vector $\mathbf{w}^{(i)} \in \mathbb{R}^d$. We learn these local model parameters by minimizing their local loss and at the same time enforce a small total variation. Requiring a small total variation of the learnt local model parameters enforces them to be (approximately) constant over well-connected nodes ("clusters").

To optimally balance local loss and total variation, we solve generalized total variation (GTV) minimization,

$$\big\{\widehat{\mathbf{w}}^{(i)}\big\}_{i=1}^{n} \in \operatorname*{argmin}_{\mathrm{stack}\left\{\mathbf{w}^{(i)}\right\}_{i=1}^{n}} \sum_{i\in\mathcal{V}} L_i\left(\mathbf{w}^{(i)}\right) + \alpha \sum_{\{i,i'\}\in\mathcal{E}} A_{i,i'} \left\|\mathbf{w}^{(i)} - \mathbf{w}^{(i')}\right\|_2^2 \quad \text{(GTVMin)}. \tag{3.15}$$

Note that GTVMin is an instance of RERM: The regularizer is the total variation of local model parameters over the weighted edges $A_{i,i'}$ of the empirical graph. Clearly, the empirical graph is an important design choice for GTVMin-based methods. This choice can be guided by computational aspects and statistical aspects of GTVMin-based FL systems.

Some application domains allow to leverage domain expertise to guess a useful choice for the empirical graph. If local datasets are generated at different geographic locations, we might use nearest-neighbour graphs based on geodesic distances between data generators (e.g., FMI weather stations). Lecture 7 will also discuss empirical graph learning methods that determine the edge weights $A_{i,i'}$ in a data-driven fashion, i.e., directly from the local datasets $\mathcal{D}^{(i)}, \mathcal{D}^{(i')}$.

Let us now consider the special case of GTVMin with local models being

a linear model. For each node $i \in \mathcal{V}$ of the empirical graph, we want to learn the parameters $\mathbf{w}^{(i)}$ of a linear hypothesis $h^{(i)}(\mathbf{x}) := \big(\mathbf{w}^{(i)}\big)^T \mathbf{x}$. We measure the quality of the parameters via the average squared error loss

$$L_i\left(\mathbf{w}^{(i)}\right) := (1/m_i) \sum_{r=1}^{m_i} \left(y^{(i,r)} - \big(\mathbf{w}^{(i)}\big)^T \mathbf{x}^{(i,r)}\right)^2$$
$$\stackrel{(3.2)}{=} (1/m_i) \left\|\mathbf{y}^{(i)} - \mathbf{X}^{(i)}\mathbf{w}^{(i)}\right\|_2^2. \tag{3.16}$$

Inserting (3.16) into (3.15), yields the following instance of GTVMin to train local linear models,

$$\big\{\widehat{\mathbf{w}}^{(i)}\big\} \in \operatorname*{argmin}_{\{\mathbf{w}^{(i)}\}_{i=1}^n} \sum_{i\in\mathcal{V}} (1/m_i) \left\|\mathbf{y}^{(i)} - \mathbf{X}^{(i)}\mathbf{w}^{(i)}\right\|_2^2 + \alpha \sum_{\{i,i'\}\in\mathcal{E}} A_{i,i'} \left\|\mathbf{w}^{(i)} - \mathbf{w}^{(i')}\right\|_2^2. \tag{3.17}$$

The identity (3.8) allows to rewrite (3.17) using the Laplacian matrix $\mathbf{L}^{(\mathcal{G})}$ as

$$\widehat{\mathbf{w}}^{(i)} \in \operatorname*{argmin}_{\mathbf{w}=\mathrm{stack}\left\{\mathbf{w}^{(i)}\right\}} \sum_{i\in\mathcal{V}} (1/m_i) \left\|\mathbf{y}^{(i)} - \mathbf{X}^{(i)}\mathbf{w}^{(i)}\right\|_2^2 + \alpha \mathbf{w}^T \big(\mathbf{L}^{(\mathcal{G})} \otimes \mathbf{I}_d\big)\mathbf{w}. \tag{3.18}$$

Let us rewrite the objective function in (3.18) as

$$\mathbf{w}^T \left(\begin{pmatrix} \mathbf{Q}^{(1)} & 0 & \cdots & 0 \\ 0 & \mathbf{Q}^{(2)} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \mathbf{Q}^{(n)} \end{pmatrix} + \alpha \mathbf{L}^{(\mathcal{G})} \otimes \mathbf{I} \right) \mathbf{w} + \big(\big(\mathbf{q}^{(1)}\big)^T, \ldots, \big(\mathbf{q}^{(n)}\big)^T\big)\mathbf{w} \tag{3.19}$$

with $\mathbf{Q}^{(i)} = (1/m_i)\big(\mathbf{X}^{(i)}\big)^T \mathbf{X}^{(i)}$, and $\mathbf{q}^{(i)} := (-2/m_i)\big(\mathbf{X}^{(i)}\big)^T \mathbf{y}^{(i)}$.

Thus, like linear regression (2.7) and ridge regression (2.27), also GTVMin (3.18) (for local linear models $\mathcal{H}^{(i)}$) minimizes a convex quadratic function,

$$\big\{\widehat{\mathbf{w}}^{(i)}\big\}_{i=1}^n \in \operatorname*{argmin}_{\mathbf{w}=\mathrm{stack}\left\{\mathbf{w}^{(i)}\right\}_{i=1}^n} \mathbf{w}^T\mathbf{Q}\mathbf{w} + \mathbf{q}^T\mathbf{w}. \tag{3.20}$$

Here, we used the psd matrix

$$\mathbf{Q} := \begin{pmatrix} \mathbf{Q}^{(1)} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{Q}^{(2)} & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{Q}^{(n)} \end{pmatrix} + \alpha \mathbf{L}^{(\mathcal{G})} \otimes \mathbf{I} \text{ with } \mathbf{Q}^{(i)} := (1/m_i)\big(\mathbf{X}^{(i)}\big)^T \mathbf{X}^{(i)} \tag{3.21}$$

and the vector

$$\mathbf{q} := \big(\big(\mathbf{q}^{(1)}\big)^T, \ldots, \big(\mathbf{q}^{(n)}\big)^T\big)^T, \text{ with } \mathbf{q}^{(i)} := (-2/m_i)\big(\mathbf{X}^{(i)}\big)^T \mathbf{y}^{(i)}. \tag{3.22}$$

3.3.1 Computational Aspects of GTVMin

Lecture 5 will apply optimization methods to solve GTVMin, resulting in practical FL algorithms. Different instances of GTVMin favour different classes of optimization methods. For example, using a differentiable loss function allows to apply gradient-based methods (see Lecture 4) to solve GTVMin.

Another important class of loss functions are those for which we can efficiently compute the proximity operator

$$\mathbf{prox}_{L,\rho}(\mathbf{w}) := \operatorname*{argmin}_{\mathbf{w}' \in \mathbb{R}^d} L(\mathbf{w}') + (\rho/2)\, \|\mathbf{w} - \mathbf{w}'\|_2^2 \text{ for some } \rho > 0.$$

Some authors refer to functions L for which $\mathbf{prox}_{L,\rho}(\mathbf{w})$ can be computed easily as *simple* or proximable [37]. GTVMin with proximable loss functions can be solved quite efficiently via proximal algorithms [38].

Besides influencing the choice of optimization method, the design choices underlying GTVMin also determine the amount of computation that is required by a given optimization method. For example, using an empirical

graph with relatively few edges ("sparse graphs") typically results in a smaller computational complexity. Indeed, Lecture 5 discusses GTVMin-based algorithms requiring an amount of computation that is proportional to the number of edges in the empirical graph.

Let us now consider the computational aspects of GTVMin (3.17) to train local linear models. As discussed above, this instance is equivalent to solving (3.20). Any solution $\widehat{\mathbf{w}}$ of (3.20) (and, in turn, (3.17)) is characterized by the zero-gradient condition

$$\mathbf{Q}\widehat{\mathbf{w}} = -(1/2)\mathbf{q}, \tag{3.23}$$

with $\mathbf{Q}, \mathbf{q}$ as defined in (3.21) and (3.22). If the matrix $\mathbf{Q}$ in (3.23) is invertible, the solution to (3.23) and, in turn, to the GTVMin instance (3.17) is unique and given by $\widehat{\mathbf{w}} = (-1/2)\mathbf{Q}^{-1}\mathbf{q}$.

The size of the matrix $\mathbf{Q}$ (see (3.21)) is proportional to the number of nodes in the empirical graph $\mathcal{G}$, which might be on the order of millions (or even billions) for internet-scale applications. For such large systems, we typically cannot use direct matrix inversion methods (e.g., based on Gaussian elimination) to compute $\mathbf{Q}^{-1}$.[5] Instead, we typically need to resort to iterative methods [39, 40].

One important family of such iterative methods are the gradient-based methods which we will discuss in Lecture 4. Starting from an initial choice for the local model parameters $\widehat{\mathbf{w}}_0 = \big(\widehat{\mathbf{w}}_0^{(1)}, \ldots, \widehat{\mathbf{w}}_0^{(n)}\big)$, these methods repeat (variants of) the gradient step,

$$\widehat{\mathbf{w}}_{k+1} := \widehat{\mathbf{w}}_k - \eta\big(2\mathbf{Q}\widehat{\mathbf{w}}_k + \mathbf{q}\big) \text{ for } k = 0, 1, \ldots.$$

[5] How many arithmetic operations (addition, multiplication) do you think are required to invert an arbitrary matrix $\mathbf{Q} \in \mathbb{R}^{d \times d}$?

The gradient step results in the updated local model parameters $\widehat{\mathbf{w}}^{(i)}$ which we stacked into

$$\widehat{\mathbf{w}}_{k+1} := \left(\left(\widehat{\mathbf{w}}^{(1)}\right)^T, \ldots, \left(\widehat{\mathbf{w}}^{(n)}\right)^T \right)^T.$$

We repeat the gradient step for a sufficient number of times, according to some stopping criterion (see Lecture 4).

3.3.2 Statistical Aspects of GTVMin

What can we say about FL system that use the solutions of GTVMin (3.15) as local model parameters? To answer such a question, we use - as for the statistical analysis of ERM in Lecture 2 - a probabilistic model for the local datasets. In particular, we use a variant of an i.i.d. assumption: Each local dataset $\mathcal{D}^{(i)}$, consists of data points whose features and labels are realizations of i.i.d. RVs

$$\mathbf{y}^{(i)} = \underbrace{\left(\mathbf{x}^{(i,1)}, \ldots, \mathbf{x}^{(i,m_i)}\right)^T}_{\text{local feature matrix } \mathbf{X}^{(i)}} \overline{\mathbf{w}}^{(i)} + \boldsymbol{\varepsilon}^{(i)} \text{ with } \mathbf{x}^{(i,r)} \overset{\text{i.i.d.}}{\sim} \mathcal{N}(\mathbf{0}, \mathbf{I}), \boldsymbol{\varepsilon}^{(i)} \sim \mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I}). \tag{3.24}$$

In contrast to the probabilistic model (2.13) (which we used for the analysis of ERM), the probabilistic model (3.24) allows for different node-specific parameters $\overline{\mathbf{w}}^{(i)}$, for $i \in \mathcal{V}$. In particular, the entire dataset obtained from pooling all local datasets does not conform to an i.i.d. assumption.

In what follows, we focus on the GTVMin instance (3.17) to learn the parameters $\mathbf{w}^{(i)}$ of a local linear model for each node $i \in \mathcal{V}$. For a reasonable choice for empirical graph, the parameters $\overline{\mathbf{w}}^{(i)}, \overline{\mathbf{w}}^{(i')}$ at connected nodes $\{i, i'\} \in \mathcal{E}$ should be similar. However, we cannot choose the edge weights based on parameters $\overline{\mathbf{w}}^{(i)}$ as they are unknown. We can only use (noisy)

estimates for $\overline{\mathbf{w}}^{(i)}$ the features and labels of the data points in the local datasets (see Lecture 7).

Consider a given (design) choice for the empirical graph (and its edge weights $A_{i,i'}$) for local datasets that conform to the probabilistic model (3.24) with true underlying parameters $\overline{\mathbf{w}}^{(i)}$. For ease of exposition, we assume that

$$\overline{\mathbf{w}}^{(i)} = \overline{\mathbf{c}}, \text{ for some } \overline{\mathbf{c}} \in \mathbb{R}^d \text{ and all } i \in \mathcal{V}. \tag{3.25}$$

To study the deviation between the solutions $\widehat{\mathbf{w}}^{(i)}$ of (3.17) and the true underlying parameters $\overline{\mathbf{w}}^{(i)}$, we decompose it as

$$\widehat{\mathbf{w}}^{(i)} =: \widetilde{\mathbf{w}}^{(i)} + \widehat{\mathbf{c}}, \text{ with } \widehat{\mathbf{c}} := (1/n) \sum_{i'=1}^{n} \widehat{\mathbf{w}}^{(i')}. \tag{3.26}$$

The component $\widehat{\mathbf{c}}$ is identical at all nodes $i \in \mathcal{V}$ and obtained as the orthogonal projection of $\widehat{\mathbf{w}} = \text{stack}\big\{\widehat{\mathbf{w}}^{(i)}\big\}_{i=1}^{n}$ on the subspace (3.14). The component $\widetilde{\mathbf{w}}^{(i)} := \widehat{\mathbf{w}}^{(i)} - (1/n)\sum_{i'=1}^{n} \widehat{\mathbf{w}}^{(i')}$ consists of the deviations, for each node i, between the GTVMin solution $\widehat{\mathbf{w}}^{(i)}$ and their average over all nodes. Trivially, the average of the deviations $\widetilde{\mathbf{w}}^{(i)}$ across all nodes is the zero vector, $(1/n)\sum_{i=1}^{n} \widetilde{\mathbf{w}}^{(i)} = \mathbf{0}$.

The decomposition (3.26) entails an analogous (orthogonal) decomposition of the error $\widehat{\mathbf{w}}^{(i)} - \overline{\mathbf{w}}^{(i)}$. Indeed, for identical true underlying model parameters (3.25) (which makes $\overline{\mathbf{w}}$ an element of the subspace (3.14)), we have

$$\sum_{i=1}^{n} \big\|\widehat{\mathbf{w}}^{(i)} - \overline{\mathbf{w}}^{(i)}\big\|_2^2 \overset{(3.25),(3.26)}{=} \underbrace{\sum_{i=1}^{n} \|\mathbf{c} - \widehat{\mathbf{c}}\|_2^2}_{n\|\mathbf{c}-\widehat{\mathbf{c}}\|_2^2} + \sum_{i=1}^{n} \big\|\widetilde{\mathbf{w}}^{(i)}\big\|_2^2. \tag{3.27}$$

The following proposition provides an upper bound on the second error component in (3.27).

Proposition 3.1. *Consider a connected empirical graph, i.e., $\lambda_2 > 0$ (see (3.10)), and the solution (3.26) to GTVMin (3.17) for the local datasets (3.24). If the true local model parameters in (3.24) are identical (see (3.25)), we can upper bound the deviation $\widetilde{\mathbf{w}}^{(i)} := \widehat{\mathbf{w}}^{(i)} - (1/n)\sum_{i=1}^{n} \widehat{\mathbf{w}}^{(i)}$ of learnt parameters $\widehat{\mathbf{w}}^{(i)}$ from their average, as*

$$\sum_{i=1}^{n} \big\|\widetilde{\mathbf{w}}^{(i)}\big\|_2^2 \leq \frac{1}{\lambda_2 \alpha} \sum_{i=1}^{n} (1/m_i) \big\|\boldsymbol{\varepsilon}^{(i)}\big\|_2^2 . \tag{3.28}$$

Proof. See Section 3.5.1. □

The upper bound (3.28) involves three components:

- properties of the local datasets, via the noise term $\boldsymbol{\varepsilon}^{(i)}$ in (3.24),
- the empirical graph via the eigenvalue $\lambda_2\big(\mathbf{L}^{(\mathcal{G})}\big)$ (see (3.10)),
- the GTVMin parameter α.

According to (3.28), we can ensure a small error component $\widetilde{\mathbf{w}}^{(i)}$ of the GTVMin solution by choosing a large value α. In other words, for sufficiently large α and a connected empirical graph (where $\lambda_2 > 0$), the solution of GTVMin $\widehat{\mathbf{w}}^{(i)}$ are approximately identical for all nodes $i \in \mathcal{V}$. This might be desirable for some FL applications where the goal is to train a common model for all nodes [13]. However, some FL applications involve heterogeneous local datasets for which it is detrimental to force all nodes to agree on a common model (see Lecture 6).

3.4 Overview of Coding Assignment

Python Notebook. `FLDesignPrinciple_CodingAssignment.ipynb`

Data File. `Assignment_MLBasicsData.csv`

This assignment revolves around a collection of temperature measurements that we store in the empirical graph $\mathcal{G}^{(\mathrm{FMI})}$. Each node $i \in \mathcal{V}$ represents a FMI weather station at latitude $\mathrm{lat}^{(i)}$ and longitude $\mathrm{lon}^{(i)}$, which we stack into the vector $\mathbf{z}^{(i)} := \big(\mathrm{lat}^{(i)}, \mathrm{lon}^{(i)}\big)^T \in \mathbb{R}^2$. The local dataset $\mathcal{D}^{(i)}$ contains m_i temperature measurements $y^{(i,1)}, \ldots, y^{(i,m_i)}$ at the station i.

The edges of $\mathcal{G}^{(\mathrm{FMI})}$ are obtained using the `Python` function `add_edges()`. Each FMI station i is connected to its nearest neighbours i', using the Euclidean distance between the corresponding vectors $\mathbf{z}^{(i)}, \mathbf{z}^{(i')}$. The number of neighbours is controlled by the input parameter `numneighbors`. All edges $\{i, i'\} \in \mathcal{E}$ have the same edge weight $A_{i,i'} = 1$.

For each station $i \in \mathcal{V}$, you need to learn the single parameter $w^{(i)} \in \mathbb{R}$ of a hypothesis $h(x) = w^{(i)}$ that predicts the temperature. We measure the quality of a hypothesis by the average squared error loss $L_i\left(w^{(i)}\right) = (1/m_i)\sum_{r=1}^{m_i}\left(y^{(i,r)} - w^{(i)}\right)^2$. You should learn the parameters $w^{(i)}$ via balancing the local loss with the total variation of $w^{(i)}$,

$$\big(\widehat{w}^{(1)}, \ldots, \widehat{w}^{(n)}\big)^T \in \operatorname*{argmin}_{\left(w^{(1)},\ldots,w^{(n)}\right)^T} \sum_{i=1}^{n} L_i\left(w^{(i)}\right) + \alpha \sum_{\{i,i'\}\in\mathcal{E}} \left(w^{(i)} - w^{(i')}\right)^2. \quad (3.29)$$

Your tasks involve

- Reformulate (3.29) as (3.17) using a suitable choice for features $\mathbf{x}^{(i,r)}$. Carefully note that this choice needs to be different from the raw features (longitude, latitude, year, month, day, hour, minute) used in the assignment "ML Basics".

- The computation of solutions $\widehat{w}^{(i)}$ to (3.29) via (3.23) and using the Python function `numpy.linalg.inv`. To this end, you should determine the matrix $\mathbf{Q}$ and vector $\mathbf{q}$ in terms of local datasets $\mathcal{D}^{(i)}$ and the Laplacian matrix $\mathbf{L}^{(\mathrm{FMI})}$ of the empirical graph $\mathcal{G}^{(\mathrm{FMI})}$.

- Studying the effect of varying values of α in (3.29) on the local loss and total variation of the corresponding solutions $\widehat{w}^{(i)}$.

3.5 Proofs

3.5.1 Proof of Proposition 3.1

Let us introduce the shorthand $f\big(\mathbf{w}^{(i)}\big)$ for the objective function of the GTVMin instance (3.17). We verify the bound (3.28) by showing that if it does not hold, the choice for the local model parameters $\mathbf{w}^{(i)} := \overline{\mathbf{w}}^{(i)}$ (see (3.24)) results in a smaller objective function value, $f\big(\overline{\mathbf{w}}^{(i)}\big) < f\big(\widehat{\mathbf{w}}^{(i)}\big)$. This would contradict the fact that $\widehat{\mathbf{w}}^{(i)}$ is a solution to (3.17).

First, note that

$$\begin{aligned} f\big(\overline{\mathbf{w}}^{(i)}\big) &= \sum_{i\in\mathcal{V}} (1/m_i) \left\|\mathbf{y}^{(i)} - \mathbf{X}^{(i)}\overline{\mathbf{w}}^{(i)}\right\|_2^2 + \alpha \sum_{\{i,i'\}\in\mathcal{E}} A_{i,i'} \left\|\overline{\mathbf{w}}^{(i)} - \overline{\mathbf{w}}^{(i')}\right\|_2^2 \\ &\overset{(3.25)}{=} \sum_{i\in\mathcal{V}} (1/m_i) \left\|\mathbf{y}^{(i)} - \mathbf{X}^{(i)}\overline{\mathbf{w}}^{(i)}\right\|_2^2 \\ &\overset{(3.24)}{=} \sum_{i\in\mathcal{V}} (1/m_i) \left\|\mathbf{X}^{(i)}\overline{\mathbf{w}}^{(i)} + \boldsymbol{\varepsilon}^{(i)} - \mathbf{X}^{(i)}\overline{\mathbf{w}}^{(i)}\right\|_2^2 \\ &= \sum_{i\in\mathcal{V}} (1/m_i) \left\|\boldsymbol{\varepsilon}^{(i)}\right\|_2^2 . \end{aligned} \tag{3.30}$$

Inserting (3.26) into (3.17),

$$\begin{aligned} f\big(\widehat{\mathbf{w}}^{(i)}\big) &= \underbrace{\sum_{i\in\mathcal{V}} (1/m_i) \left\|\mathbf{y}^{(i)} - \mathbf{X}^{(i)}\widehat{\mathbf{w}}^{(i)}\right\|_2^2}_{\geq 0} + \alpha \sum_{\{i,i'\}\in\mathcal{E}} A_{i,i'} \underbrace{\left\|\widehat{\mathbf{w}}^{(i)} - \widehat{\mathbf{w}}^{(i')}\right\|_2^2}_{\overset{(3.26)}{=}\left\|\widetilde{\mathbf{w}}^{(i)} - \widetilde{\mathbf{w}}^{(i')}\right\|_2^2} \\ &\geq \alpha \sum_{\{i,i'\}\in\mathcal{E}} A_{i,i'} \left\|\widetilde{\mathbf{w}}^{(i)} - \widetilde{\mathbf{w}}^{(i')}\right\|_2^2 \\ &\overset{(3.13)}{\geq} \alpha\lambda_2 \sum_{i=1}^{n} \left\|\widetilde{\mathbf{w}}^{(i)}\right\|_2^2 . \end{aligned} \tag{3.31}$$

If the bound (3.28) would not hold, then by (3.31) and (3.30) we would obtain $f\big(\widehat{\mathbf{w}}^{(i)}\big) > f\big(\overline{\mathbf{w}}^{(i)}\big)$. This is a contradiction to the fact that $\widehat{\mathbf{w}}^{(i)}$ solves (3.17).

.

4 Lecture - "Gradient Methods"

Lecture 3 introduced GTVMin as a central design principle for FL methods. Many important instances of GTVMin require to minimize a smooth objective function $f(\mathbf{w})$ over the parameter space $\mathbb{R}^d$. This lecture explores gradient-based methods, a widely used family of iterative algorithms to find the minimum of a smooth function. These methods approximate the objective function locally using its gradient at the current model parameters. Lecture 5 focuses on FL algorithms obtained from the application of gradient-based methods to solve GTVMin.

4.1 Learning Goals

After this lecture, you should

- have some intuition about the effect of a gradient step,
- understand the role of the step size (or learning rate),
- know some examples for a stopping criterion,
- be able to analyze the effect of perturbations in the gradient step,
- know about projected GD to cope with constraints on model parameters.

4.2 Gradient Descent

Gradient-based methods are iterative algorithms for finding the minimum of a differentiable objective function $f(\mathbf{w})$ of vector$\mathbf{w}$ (which could be model parameters in a ML method). Unless stated otherwise, we consider an objective function of form

$$f(\mathbf{w}) = \mathbf{w}^T \mathbf{Q} \mathbf{w} + \mathbf{q}^T \mathbf{w}. \tag{4.1}$$

Note that (4.1) defines an entire family of convex quadratic functions $f(\mathbf{w})$. Each member of this family is specified by a psd matrix $\mathbf{Q} \in \mathbb{R}^{d \times d}$ and a vector $\mathbf{q} \in \mathbb{R}^d$.

We have already encountered some ML and FL methods that minimize an objective function of the form (4.1): Linear regression (2.3) and ridge regression (2.27) in Lecture 2 as well as GTVMin (3.17) for local linear models in Lecture 3. Moreover, (4.1) might serve as a useful approximation for the objective functions arising for larger classes of ML methods [41–43].

Given model parameters $\mathbf{w}^{(k)}$, we want to update them towards a minimum of (4.1). To this end, we use the gradient $\nabla f\big(\mathbf{w}^{(k)}\big)$ to locally approximate $f(\mathbf{w})$ (see Figure 4.1). The gradient $\nabla f\big(\mathbf{w}^{(k)}\big)$ indicates the direction in which the function $f(\mathbf{w})$ maximally increases. Therefore, it seems reasonable to update $\mathbf{w}^{(k)}$ in the opposite direction of $\nabla f\big(\mathbf{w}^{(k)}\big)$,

$$\begin{aligned} \mathbf{w}^{(k+1)} &:= \mathbf{w}^{(k)} - \eta \nabla f\big(\mathbf{w}^{(k)}\big) \\ &\overset{(4.1)}{=} \mathbf{w}^{(k)} - \eta\big(2\mathbf{Q}\mathbf{w}^{(k)} + \mathbf{q}\big). \end{aligned} \tag{4.2}$$

The gradient step (4.2) involves the factor η which we refer to as step-size or learning rate. Algorithm 4.1 summarizes the most basic instance of gradient-

based methods which simply repeats (iterates) (4.2) until some stopping criterion is met.

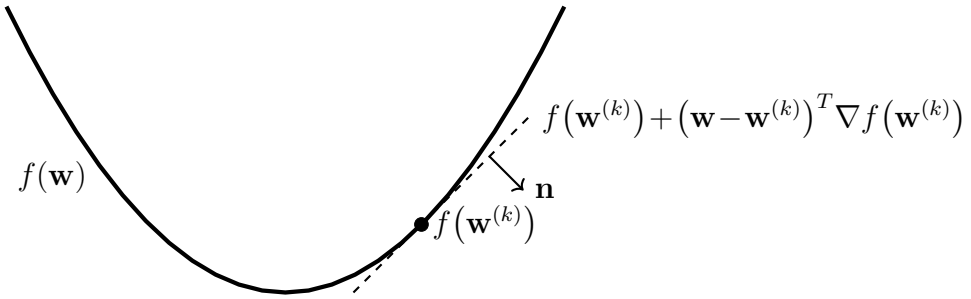


Figure 4.1: We can approximate a differentiable function $f(\mathbf{w})$ locally around a point $\mathbf{w}^{(k)} \in \mathbb{R}^d$ using the linear function $f\big(\mathbf{w}^{(k)}\big)+\big(\mathbf{w}-\mathbf{w}^{(k)}\big)^T\nabla f\big(\mathbf{w}^{(k)}\big)$. Geometrically, we approximate the graph of $f(\mathbf{w})$ by a hyperplane with normal vector $\mathbf{n} = (\nabla f\big(\mathbf{w}^{(k)}\big), -1)^T \in \mathbb{R}^{d+1}$ of this approximating hyperplane is determined by the gradient $\nabla f\big(\mathbf{w}^{(k)}\big)$ [5].

The usefulness of gradient-based methods depends on how difficult it is to evaluate the gradient $\nabla f(\mathbf{w})$. Modern software libraries for automatic differentiation allow to evaluate the gradient of objective functions functions arising in ERM-based methods [44]. However, besides the actual computation of the gradient, it might be already challenging to gather the required data points which define the objective function $f(\mathbf{w})$ (e.g., being the average loss over a large training set). Indeed, the matrix $\mathbf{Q}$ and vector $\mathbf{q}$ in (4.1) are constructed from the features and labels of data points in the training set. For example, the gradient of the objective function in ridge regression (2.27) is

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

Evaluating this gradient requires roughly $d \times m$ arithmetic operations (summations and multiplications).

Algorithm 4.1 A blueprint for gradient-based methods

Input: some objective function $f(\mathbf{w})$ (e.g., the average loss of a hypothesis $h^{(\mathbf{w})}$ on a training set); learning rate $\eta > 0$; some stopping criterion;

Initialize: set $\mathbf{w}^{(0)} := \mathbf{0}$; set iteration counter $k := 0$

1: **repeat**

2: $k := k + 1$ (increase iteration counter)

3: $\mathbf{w}^{(k)} := \mathbf{w}^{(k-1)} - \eta \nabla f\big(\mathbf{w}^{(k-1)}\big)$ (do a gradient step (4.2))

4: **until** stopping criterion is met

Output: learnt model parameters $\widehat{\mathbf{w}} := \mathbf{w}^{(k)}$ (hopefully $f\big(\widehat{\mathbf{w}}\big) \approx \min_{\mathbf{w}} f(\mathbf{w})$)

Note that Algorithm 4.1, as most other gradient-based methods, involves two hyper-parameters: (i) the learning rate η used for the gradient step and (ii) a stopping criterion to decide when to stop repeating the gradient step. We next discuss how to choose these hyper-parameters.

4.3 Learning Rate

The learning rate must not be too large to avoid overshooting or moving away from the optimum (see Figure 4.2-(b)). On the other hand, if the learning rate is chosen too small, the gradient step makes too little progress towards the solutions of (4.1) (see Figure 4.2-(a)). Note that in practice we can only afford to repeat the gradient step for a finite number of iterations.

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

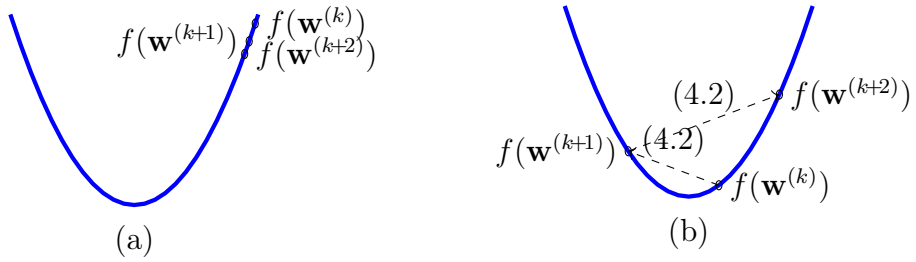


Figure 4.2: Effect of inadequate learning rates η in the gradient step (4.2). (a) If η is too small, the gradient steps make very little progress towards the optimum or even fail to reach the optimum at all. (b) If η is too large, the gradient steps might "overshoot" such that the iterates $\mathbf{w}^{(k)}$ might diverge from the optimum, i.e., $f(\mathbf{w}^{(k+1)}) > f(\mathbf{w}^{(k)})$!

this decrease does not agree with the decrease predicted by the (local linear approximation using 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 [45, Sec 6.1].

Alternatively, we can use a prescribed sequence (schedule) η_k, for $k = 1, 2, \ldots$, of learning rates that vary across successive gradient steps [46]. We can guarantee convergence of (4.2) (under mild assumptions on the objective function $f(\mathbf{w})$) by using a learning rate schedule that satisfies [45, Sec. 6.1]

$$\lim_{k\to\infty} \eta_k = 0, \ \sum_{k=1}^{\infty} \eta_k = \infty \ , \text{ and } \ \sum_{k=1}^{\infty} \eta_k^2 < \infty. \tag{4.3}$$

The first condition (4.3) requires that the learning rate eventually become sufficiently small to avoid overshooting. The third condition (4.3) ensures that this required decay of the learning rate does not take "forever". Note

that the first and third condition in (4.3) could be satisfied by the trivial learning rate schedule $\eta_k = 0$ which is clearly not useful as the gradient step has no effect.

The trivial schedule $\eta_k = 0$ is ruled out by the middle condition of (4.3). This middle condition ensures that the learning rate η_k is large enough such that the gradient steps make sufficient progress towards a minimizer of the objective function.

We emphasize that the conditions (4.3) do not involve any properties of the matrix $\mathbf{Q}$ in (4.1). This might be convenient if the matrix $\mathbf{Q}$ is determined by raw data (see, e.g., (2.3)) whose statistical properties can be controlled only to a limited extend.

4.4 When to Stop?

For the stopping criterion we might use a fixed number $k_{\max}$ of iterations. This hyper-parameter $k_{\max}$ might be dictated by limited resources (such as computational time) for implementing GD or tuned via validation techniques.

We can obtain another stopping criterion from monitoring the decrease in the objective function $f\big(\mathbf{w}^{(k)}\big)$: we stop repeating the gradient step (4.2) whenever $\big|f\big(\mathbf{w}^{(k)}\big) - f\big(\mathbf{w}^{(k+1)}\big)\big| \leq \varepsilon^{(\mathrm{tol})}$ for a given tolerance $\varepsilon^{(\mathrm{tol})}$. The tolerance level $\varepsilon^{(\mathrm{tol})}$ is a hyper-parameter of the resulting ML method which could be chosen via validation techniques (see Lecture 2).

The above technique for deciding when to stop the gradient steps are convenient as they do not require any information about the psd matrix $\mathbf{Q}$ in (4.1). However, they might result in sub-optimal use of computational resources by implementing "useless" additional gradient steps. We can avoid

such a waste of computation whenever we have some information about the psd matrix $\mathbf{Q}$ in (4.1).[6] Indeed, the choice for the learning rate η and the stopping criterion can be guided by the eigenvalues

$$0 \leq \lambda_1(\mathbf{Q}) \leq \ldots \leq \lambda_d(\mathbf{Q}). \tag{4.4}$$

Even if we do not know these eigenvalues precisely, we might know (or be able to ensure via feature learning) some upper and lower bounds,

$$0 \leq L \leq \lambda_1(\mathbf{Q}) \leq \ldots \leq \lambda_d(\mathbf{Q}) \leq U. \tag{4.5}$$

In what follows, we assume that $\mathbf{Q}$ is invertible and we know some positive lower bound $L > 0$ (see (4.5)).[7] The objective function (4.1) has then a unique solution $\widehat{\mathbf{w}}$. It turns out that a gradient step (4.2) reduces the distance $\left\|\mathbf{w}^{(k)} - \widehat{\mathbf{w}}\right\|_2$ to $\widehat{\mathbf{w}}$ by a constant factor [45, Ch. 6],

$$\left\|\mathbf{w}^{(k+1)} - \widehat{\mathbf{w}}\right\|_2 \leq \kappa^{(\eta_k)}(\mathbf{Q}) \left\|\mathbf{w}^{(k)} - \widehat{\mathbf{w}}\right\|_2. \tag{4.6}$$

Here, we used the contraction factor

$$\kappa^{(\eta)}(\mathbf{Q}) := \max\big\{|1 - \eta 2\lambda_1|, |1 - \eta 2\lambda_d|\big\}. \tag{4.7}$$

The contraction factor depends on the learning rate η which is a hyper-parameter of gradient-based methods that we can control. However, the

[6] For linear regression (2.7), the matrix $\mathbf{Q}$ is determined by the features of the data points in the training set. We can influence the properties of $\mathbf{Q}$ to some extent by feature transformation methods. One important example for such a transformation is the normalization of features.

[7] What are sufficient conditions for the local datasets and the edge weights used in GTVMin such that $\mathbf{Q}$ in (3.21) is invertible?

contraction factor also depends on the eigenvalues of the matrix $\mathbf{Q}$ in (4.1). In ML and FL applications, this matrix typically depends on data and can be controlled only to some extend, e.g., using feature transformation [4, Ch. 5]. We can ensure $\kappa^{(\eta)}(\mathbf{Q}) < 1$ if we use a positive learning rate $\eta_k < 1/U$.

For a contraction factor $\kappa^{(\eta)}(\mathbf{Q}) < 1$, a sufficient condition on the number $k^{(\varepsilon^{(\mathrm{tol})})}$ of gradient steps required to ensure an optimization error $\left\|\mathbf{w}^{(k+1)} - \widehat{\mathbf{w}}\right\|_2 \leq \varepsilon^{(\mathrm{tol})}$ is (see (4.6))

$$k^{(\varepsilon^{(\mathrm{tol})})} \geq \log\left(\left\|\mathbf{w}^{(0)} - \widehat{\mathbf{w}}\right\|_2 / \varepsilon^{(\mathrm{tol})}\right) / \log\left(1/\kappa^{(\eta)}(\mathbf{Q})\right). \tag{4.8}$$

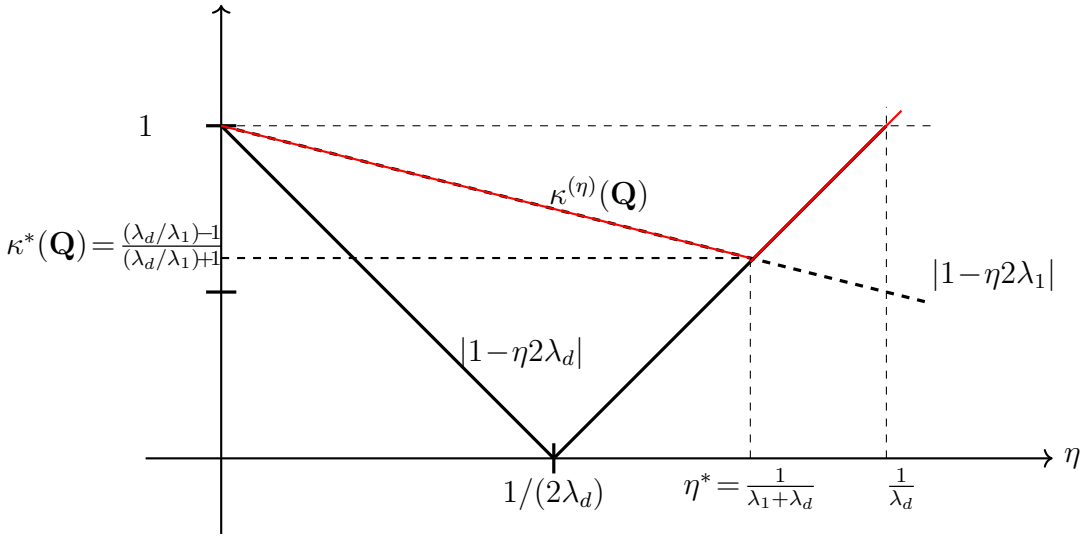


Figure 4.3: The contraction factor $\kappa^{(\eta)}(\mathbf{Q})$ (4.7), used in the upper bound (4.6), as a function of the learning rate η. Note that $\kappa^{(\eta)}(\mathbf{Q})$ also depends on the eigenvalues of the matrix $\mathbf{Q}$ in (4.1).

According to (4.6), smaller values of the contraction factor $\kappa^{(\eta)}(\mathbf{Q})$ guarantee a faster convergence of (4.2) towards the solution of (4.1). Figure 4.3 illustrates the dependence of $\kappa^{(\eta)}(\mathbf{Q})$ on the learning rate η. Thus, choosing a small η (close to 0) will typically result in a larger $\kappa^{(\eta)}(\mathbf{Q})$ and, in turn, require more iterations to ensure optimization error level $\varepsilon^{(\mathrm{tol})}$ via (4.6).

We can minimize this contraction factor by choosing the learning rate (see Figure 4.3)

$$\eta^{(*)} := \frac{1}{\lambda_1 + \lambda_d}. \tag{4.9}$$

[Note that evaluating (4.9) requires to know the extremal eigenvalues λ_1, λ_d of $\mathbf{Q}$.] Inserting the optimal learning rate (4.9) into (4.6),

$$\left\|\mathbf{w}^{(k+1)} - \widehat{\mathbf{w}}\right\|_2 \leq \underbrace{\frac{(\lambda_d/\lambda_1) - 1}{(\lambda_d/\lambda_1) + 1}}_{:=\kappa^*(\mathbf{Q})} \left\|\mathbf{w}^{(k)} - \widehat{\mathbf{w}}\right\|_2 . \tag{4.10}$$

Carefully note that the formula (4.10) is valid only if the matrix $\mathbf{Q}$ in (4.1) is invertible, i.e., if $\lambda_1 > 0$. If the matrix $\mathbf{Q}$ is singular ($\lambda_1 = 0$), the convergence of (4.2) towards a solution of (4.1) is much slower than the decrease of the bound (4.10). However, we can still ensure converge of gradient steps by using a fixed learning rate $\eta_k = \eta$ that satisfies [47, Thm. 2.1.14]

$$0 < \eta < 1/\lambda_d(\mathbf{Q}). \tag{4.11}$$

It is interesting to note that for linear regression, the matrix $\mathbf{Q}$ depends only on the features $\mathbf{x}^{(r)}$ of the data points in the training set (see (2.17)) but not on their labels $y^{(r)}$. Thus, the convergence of gradient steps is only affected by the features, whereas the labels are irrelevant. The same is true for ridge regression and GTVMin (using local linear models).

Note that both, the optimal learning rate (4.9) and the optimal contraction factor

$$\kappa^{*}(\mathbf{Q}) := \frac{(\lambda_d/\lambda_1) - 1}{(\lambda_d/\lambda_1) + 1} \tag{4.12}$$

depend on the eigenvalues of the matrix $\mathbf{Q}$ in (4.1). According to (4.10), the ideal case is when all eigenvalues are identical which leads, in turn, to a contraction factor $\kappa^{*}(\mathbf{Q}) = 0$. Here, a single gradient step arrives at the unique solution of (4.1).

In general, we do not have full control over the matrix $\mathbf{Q}$ and its eigenvalues. For example, the matrix $\mathbf{Q}$ arising in linear regression (2.7) is determined by the features of data points in the training set. These features might be obtained from sensing devices and therefore beyond our control. However, other applications might allow for some design freedom in the choice of feature vectors. We might also use feature transformations that nudge the resulting $\mathbf{Q}$ in (2.7) more towards a scaled identity matrix.

4.5 Perturbed Gradient Step

Consider the gradient step (4.2) used to find a minimum of (4.1). We again assume that the matrix $\mathbf{Q}$ in (4.1) is invertible ($\lambda_1(\mathbf{Q}) > 0$) and, in turn, (4.1) has a unique solution $\widehat{\mathbf{w}}$.

In some applications, it might be difficult to evaluate the gradient $\nabla f(\mathbf{w}) = 2\mathbf{Q}\mathbf{w} + \mathbf{q}$ of (4.1) exactly. For example, this evaluation might require to gather data points from distributed storage locations. These storage locations might become unavailable during the computation of $\nabla f(\mathbf{w})$ due to software or hardware failures (e.g., limited connectivity).

We can model imperfections during the computation of (4.2) as the

perturbed gradient step

$$\mathbf{w}^{(k+1)} := \mathbf{w}^{(k)} - \eta \nabla f\big(\mathbf{w}^{(k)}\big) + \boldsymbol{\varepsilon}^{(k)} \\ \stackrel{(4.1)}{=} \mathbf{w}^{(k)} - \eta\big(2\mathbf{Q}\mathbf{w}^{(k)} + \mathbf{q}\big) + \boldsymbol{\varepsilon}^{(k)}. \tag{4.13}$$

We can use the contraction factor (4.7) to upper bound the deviation between $\mathbf{w}^{(k)}$ and the optimum $\widehat{\mathbf{w}}$ as (see (4.6))

$$\big\|\mathbf{w}^{(k)} - \widehat{\mathbf{w}}\big\|_2 \leq \big(\kappa^{(\eta)}\left(\mathbf{Q}\right)\big)^k \big\|\mathbf{w}^{(0)} - \widehat{\mathbf{w}}\big\|_2 + \sum_{k'=1}^{k} \big(\kappa^{(\eta)}\left(\mathbf{Q}\right)\big)^{k'} \Big\|\boldsymbol{\varepsilon}^{(k-k')}\Big\|_2. \tag{4.14}$$

This bound applies for any number of iterations $k = 1, 2, \ldots$ of the perturbed gradient step (4.13).

Finally, note that the perturbed gradient step (4.13) could also be used as a tool to analyze the (exact) gradient step for an objective function $\tilde{f}(\mathbf{w})$ which does not belong to the family (4.1) of convex quadratic functions. Indeed, we can write the gradient step for minimizing $\tilde{f}(\mathbf{w})$ as

$$\mathbf{w}^{(k+1)} := \mathbf{w}^{(k)} - \eta \nabla \tilde{f}(\mathbf{w}) \\ = \mathbf{w}^{(k)} - \eta \nabla f(\mathbf{w}) + \underbrace{\eta\big(\nabla f(\mathbf{w}) - \nabla \tilde{f}(\mathbf{w})\big)}_{:=\boldsymbol{\varepsilon}^{(k)}}.$$

The last identity is valid for any choice of surrogate function $f(\mathbf{w})$. In particular, we can choose $f(\mathbf{w})$ as a convex quadratic function (4.1) that approximates $\tilde{f}(\mathbf{w})$. Note that the perturbation term $\boldsymbol{\varepsilon}^{(k)}$ is scaled by the learning rate η.

4.6 Handling Constraints - Projected Gradient Descent

Many important ML and FL methods amount to the minimization of an objective function of the form (4.1). The optimization variable $\mathbf{w}$ in (4.1)

represents some model parameters.

Sometimes we might require the parameters $\mathbf{w}$ to belong to a subset $\mathcal{C} \subset \mathbb{R}^d$. One example is regularization via model pruning (see Lecture 2). Another example are FL methods that learn identical local model parameters $\mathbf{w}^{(i)}$ at all nodes $i \in \mathcal{V}$ of an empirical graph. This can be implemented by requiring the stacked local model parameters $\mathbf{w} = \big(\mathbf{w}^{(1)}, \ldots, \mathbf{w}^{(n)}\big)^T$ 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 gradient step (4.2) to solve the constrained problem

$$f^* = \min_{\mathbf{w} \in \mathcal{C}} \mathbf{w}^T \mathbf{Q} \mathbf{w} + \mathbf{q}^T \mathbf{w}. \tag{4.15}$$

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

$$P_{\mathcal{C}}\big(\mathbf{w}\big) = \operatorname*{argmin}_{\mathbf{w}' \in \mathcal{C}} \|\mathbf{w} - \mathbf{w}'\|_2 \text{ for any } \mathbf{w} \in \mathbb{R}^d. \tag{4.16}$$

A suitable modification of the gradient step (4.2) to solve the constrained variant (4.15) is [45]

$$\begin{aligned} \mathbf{w}^{(k+1)} &:= P_{\mathcal{C}}\big(\mathbf{w}^{(k)} - \eta \nabla f\big(\mathbf{w}^{(k)}\big)\big) \\ &\overset{(4.1)}{=} P_{\mathcal{C}}\big(\mathbf{w}^{(k)} - \eta\big(2\mathbf{Q}\mathbf{w}^{(k)} + \mathbf{q}\big)\big). \end{aligned} \tag{4.17}$$

The projected GD step (4.17) amounts to

1. compute ordinary gradient step $\mathbf{w}^{(k)} \mapsto \mathbf{w}^{(k)} - \eta \nabla f\big(\mathbf{w}^{(k)}\big)$ and
2. project back to the constraint set $\mathcal{C}$.

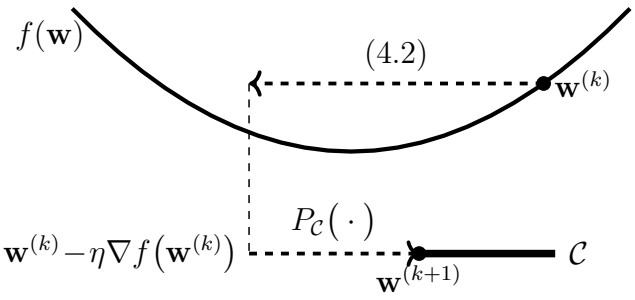


Figure 4.4: Projected GD augments a basic gradient step with a projection back onto the constraint set $\mathcal{C}$.

Note that we re-obtain the basic gradient step (4.2) from the projected gradient step (4.17) for the specific constraint set $\mathcal{C} = \mathbb{R}^d$.

The approaches for choosing the learning rate η and stopping criterion for basic gradient step (4.2) explained in Sections 4.3 and 4.4 work also for the projected gradient step (4.17). In particular, the convergence speed of the projected gradient step is also characterized by (4.6) [45, Ch. 6]. This follows from the fact that the concatenation of a contraction (such as the gradient step (4.2) for sufficiently small η) and a projection (such as $P_{\mathcal{C}}(\,\cdot\,)$) results again in a contraction with the same contraction factor.

Thus, the convergence speed of projected GD, in terms of number of iterations required to ensure a given level of optimization error, is essentially the same as that of basic GD. However, the bound (4.6) is only telling about the number of projected gradient steps (4.17) required to achieve a guaranteed level of sub-optimality $\big|f\big(\mathbf{w}^{(k)}\big) - f^*\big|$. The iteration (4.17) of projected GD might require significantly more computation than the basic gradient step, as it requires to compute the projection (4.16).

4.7 Generalizing the Gradient Step

The gradient-based methods discussed so far can be used to learn a hypothesis from a parametrized model. Let us now sketch one possible generalization of the gradient step (4.2) for a model $\mathcal{H}$ without a parametrization.

We start with rewriting the gradient step (4.2) as the optimization

$$\mathbf{w}^{(k+1)} = \operatorname*{argmin}_{\mathbf{w}\in\mathbb{R}^d} (1/(2\eta)) \left\|\mathbf{w}-\mathbf{w}^{(k)}\right\|_2^2 + \underbrace{f\left(\mathbf{w}^{(k)}\right) + \left(\mathbf{w}-\mathbf{w}^{(k)}\right)^T \nabla f\left(\mathbf{w}^{(k)}\right)}_{\approx f(\mathbf{w})}. \tag{4.18}$$

The objective function in (4.18) includes the first-order approximation

$$f(\mathbf{w}) \approx f\left(\mathbf{w}^{(k)}\right) + \left(\mathbf{w}-\mathbf{w}^{(k)}\right)^T \nabla f\left(\mathbf{w}^{(k)}\right)$$

of the function $f(\mathbf{w})$ around the location $\mathbf{w}=\mathbf{w}^{(k)}$ (see Figure 4.1).

Let us modify (4.18) by using $f(\mathbf{w})$ itself (instead of an approximation),

$$\mathbf{w}^{(k+1)} = \operatorname*{argmin}_{\mathbf{w}\in\mathbb{R}^d} (1/(2\eta)) \left\|\mathbf{w}-\mathbf{w}^{(k)}\right\|_2^2 + f(\mathbf{w}). \tag{4.19}$$

Like the gradient step, also (4.19) maps a given vector $\mathbf{w}^{(k)}$ to an updated vector $\mathbf{w}^{(k+1)}$. Note that (4.19) is nothing but the proximity operator of the function $f(\mathbf{w})$ [38]. Similar to the role of the gradient step as the main building block of gradient-based methods, the proximity operator (4.19) is the main building block of proximal algorithms [38].

To obtain a version of (4.19) for a non-parametric model, we need to be able to evaluate its objective function directly in terms of a hypothesis h instead of its parameters $\mathbf{w}$. The objective function (4.19) consists of two components. The second component $f(\cdot)$, which is the function we want to minimize, is obtained from a training error incurred by a hypothesis, which

might be parametrized $h^{(\mathbf{w})}$. Thus, we can evaluate the function $f(h)$ by computing the training error for a given hypothesis.

The first component of the objective function in (4.19) uses $\left\|\mathbf{w}-\mathbf{w}^{(k)}\right\|_2^2$ to measure the difference between the hypothesis maps $h^{(\mathbf{w})}$ and $h^{(\mathbf{w}^{(k)})}$. Another measure for the difference between two hypothesis maps can be obtained by using some test dataset $\mathcal{D}'=\left\{\mathbf{x}^{(1)},\ldots,\mathbf{x}^{(m')}\right\}$: The average squared difference between their predictions,

$$(1/m')\sum_{r=1}^{m'}\left(h\big(\mathbf{x}^{(r)}\big)-h^{(k)}\big(\mathbf{x}^{(r)}\big)\right)^2, \tag{4.20}$$

is a measure for the difference between h and $h^{(k)}$. Note that the measure (4.20) does require any model parameters but only the predictions delivered by the hypothesis maps on $\mathcal{D}'$

It is interesting to note that (4.20) coincides with $\left\|\mathbf{w}-\mathbf{w}^{(k)}\right\|_2^2$ for the linear model $h^{(\mathbf{w})}(\mathbf{x}):=\mathbf{w}^T\mathbf{x}$ and a specific construction of the dataset $\mathcal{D}'$. This construction uses the realizations $\mathbf{x}^{(1)},\mathbf{x}^{((2)},\ldots$ of i.i.d. RVs with common probability distribution $\mathbf{x}\sim\mathcal{N}(\mathbf{0},\mathbf{I})$. Indeed, by the law of large numbers

$$\begin{aligned}
&\lim_{m'\to\infty}(1/m')\sum_{r=1}^{m'}\left(h^{(\mathbf{w})}\big(\mathbf{x}^{(r)}\big)-h^{(\mathbf{w}^{(k)})}\big(\mathbf{x}^{(r)}\big)\right)^2\\
&=\lim_{m'\to\infty}(1/m')\sum_{r=1}^{m'}\left(\big(\mathbf{w}-\mathbf{w}^{(k)}\big)^T\mathbf{x}^{(r)}\right)^2\\
&=\lim_{m'\to\infty}(1/m')\sum_{r=1}^{m'}\big(\mathbf{w}-\mathbf{w}^{(k)}\big)^T\mathbf{x}^{(r)}\big(\mathbf{x}^{(r)}\big)^T\big(\mathbf{w}-\mathbf{w}^{(k)}\big)\\
&=\big(\mathbf{w}-\mathbf{w}^{(k)}\big)^T\underbrace{\left[\lim_{m'\to\infty}(1/m')\sum_{r=1}^{m'}\mathbf{x}^{(r)}\big(\mathbf{x}^{(r)}\big)^T\right]}_{=\mathbf{I}}\big(\mathbf{w}-\mathbf{w}^{(k)}\big)\\
&=\left\|\mathbf{w}-\mathbf{w}^{(k)}\right\|_2^2.
\end{aligned}\tag{4.21}$$

To arrive at our generalization of the gradient step, we replace $\left\|\mathbf{w}-\mathbf{w}^{(k)}\right\|_2^2$ in (4.19) with (4.20),

$$h^{(k+1)} = \underset{h\in\mathcal{H}}{\operatorname{argmin}} \left[(1/(2\eta m')) \sum_{r=1}^{m'} \left(h\big(\mathbf{x}^{(r)}\big) - h^{(k)}\big(\mathbf{x}^{(r)}\big)\right)^2 + f(h)\right]. \quad (4.22)$$

We can modify gradient-based methods (such as Algorithm 4.1), by replacing the gradient step with the update (4.22), to obtain training algorithms for a non-parametric model (or hypothesis space) $\mathcal{H}$.

4.8 Overview of Coding Assignment

Python Notebook. `GradientMethods_CodingAssignment.ipynb`

Data File. `Assignment_MLBasicsData.csv`

Description. This assignment builds on the assignment for Lecture 2 which used ready-made `Python` implementations of linear regression and ridge regression. Now you have to implement ridge regression "from scratch" by applying Algorithm 4.1 to solve the ridge regression problem (2.27).

The features are (normalized) values of the latitude and longitude of the FMI station as well as the (normalized) year, month, day, hour and minute during which the measurement has been taken. Your tasks include

- Generate `numpy` arrays $\mathbf{X}, \mathbf{y}$, whose r-th row hold the features $\mathbf{x}^{(r)}$ and label $y^{(r)}$, respectively, of the r-th data point in the csv file.

- Split the dataset into a training set and validation set. The size of the training set should be 100.

- Train a linear model, using the `RidgeRegression` class of the `scikit-learn` package, on the training set and determine the resulting training error E_t and validation error E_v.

- Implement the GD Algorithm 4.1 for the objective function (2.27). Unless stated otherwise, use the initialization $\mathbf{w}^{(0)} = \mathbf{0}$. Determine the training error E_t and validation error E_v using the model parameters delivered by Algorithm 4.1.

- To develop some intuition for the behaviour of Algorithm 4.1, you could try out difference choices for the learning rate η and maximum number

$k_{\max}$ of gradient steps. For each choice, you could monitor and plot the objective function value $f\left(\mathbf{w}^{(k)}\right)$ for the model parameters as a function of the iteration counter k in Algorithm 4.1.

5 Lecture - "FL Algorithms"

Lecture 3 introduced GTVMin as a flexible design principle for FL methods that arise from different design choices for the local models and edge weights of the empirical graph. The solutions of GTVMin are local model parameters that strike a balance between the loss incurred on local datasets and the total variation. This lecture applies the gradient-based methods from Lecture 4 to solve GTVMin. The resulting FL algorithms can be implemented by message passing over the edges of the empirical graph. The details of how this message passing is implemented physically (e.g., via short range wireless technology) is beyond the scope if this course.

Section 5.2 studies the gradient step for the GTVMin instance obtained for training local linear models. In particular, we show how the convergence rate of the gradient step can be characterized by properties of the local datasets and their empirical graph. Section (5.3) spells out the gradient step in the form of message passing over the empirical graph. Section 5.4 presents a FL algorithm that is obtained by replacing the exact GD with a stochastic approximation. Section 5.5 discusses FL algorithms for the single-model setting where we want to train a single global model in a distributed fashion.

5.1 Learning Goals

After this lecture, you

- can apply gradient-based methods to GTVMin for local linear models,
- can implement a gradient step via message passing over empirical graphs,

- know some motivation for stochastic gradient descent (SGD),
- know how federated averaging (FedAvg) is obtained from projected GD.

5.2 Gradient Step for GTVMin

Consider a collection of n local datasets represented by the nodes $\mathcal{V} = \{1,\ldots,n\}$ of an empirical graph $\mathcal{G} = (\mathcal{V},\mathcal{E})$. Each undirected edge $\{i,i'\} \in \mathcal{E}$ in empirical graph $\mathcal{G}$ has a known edge weight $A_{i,i'}$. We want to learn local model parameters $\mathbf{w}^{(i)}$ of a personalized linear model for each node $i = 1,\ldots,n$. To this end, we solve the GTVMin instance

$$\big\{\widehat{\mathbf{w}}^{(i)}\big\}_{i=1}^{n} \in \operatorname*{argmin}_{\{\mathbf{w}^{(i)}\}} \underbrace{\sum_{i\in\mathcal{V}} \overbrace{(1/m_i)\left\|\mathbf{y}^{(i)}-\mathbf{X}^{(i)}\mathbf{w}^{(i)}\right\|_2^2}^{\text{local loss } L_i\left(\mathbf{w}^{(i)}\right)} + \alpha \sum_{\{i,i'\}\in\mathcal{E}} A_{i,i'}\left\|\mathbf{w}^{(i)}-\mathbf{w}^{(i')}\right\|_2^2}_{=:f(\mathbf{w})}. \tag{5.1}$$

As discussed in Lecture 3, the objective function in (5.1) - viewed as a function of the stacked local model parameters $\mathbf{w} := \text{stack}\{\mathbf{w}^{(i)}\}_{i=1}^{n}$ - is a quadratic function of the form (4.1),

$$\mathbf{w}^T \underbrace{\left(\begin{pmatrix} \mathbf{Q}^{(1)} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{Q}^{(2)} & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{Q}^{(n)} \end{pmatrix} + \alpha \mathbf{L}^{(\mathcal{G})} \otimes \mathbf{I}\right)}_{:=\mathbf{Q}} \mathbf{w} + \underbrace{\left(\left(\mathbf{q}^{(1)}\right)^T,\ldots,\left(\mathbf{q}^{(n)}\right)^T\right)}_{:=\mathbf{q}^T}\mathbf{w} \tag{5.2}$$

with $\mathbf{Q}^{(i)} = (1/m_i)\big(\mathbf{X}^{(i)}\big)^T\mathbf{X}^{(i)}$, and $\mathbf{q}^{(i)} := (-2/m_i)\big(\mathbf{X}^{(i)}\big)^T\mathbf{y}^{(i)}$.

Therefore, the discussion and analysis of gradient-based methods from Lecture 4 also apply to GTVMin (5.1). In particular, we use the gradient step

$$\begin{aligned}\mathbf{w}^{(k+1)} &:= \mathbf{w}^{(k)} - \eta \nabla f\big(\mathbf{w}^{(k)}\big) \\ &\stackrel{(4.1)}{=} \mathbf{w}^{(k)} - \eta\big(2\mathbf{Q}\mathbf{w}^{(k)} + \mathbf{q}\big) \end{aligned} \tag{5.3}$$

to iteratively compute an approximate solution $\widehat{\mathbf{w}}$ to (5.1). This solution consists of learnt local model parameters $\widehat{\mathbf{w}}^{(i)}$, i.e., $\widehat{\mathbf{w}} = \text{stack}\{\widehat{\mathbf{w}}^{(i)}\}$.

According to the convergence analysis in Lecture 4, the convergence rate of the iterations (5.3) is determined by the eigenvalues of the matrix $\mathbf{Q}$ in (5.2). Clearly, the eigenvalues $\lambda_j(\mathbf{Q})$ of $\mathbf{Q}$ are related to the eigenvalues $\lambda_j\big(\mathbf{Q}^{(i)}\big)$ and to the eigenvalues $\lambda_j\big(\mathbf{L}^{(\mathcal{G})}\big)$ of the Laplacian matrix of the empirical graph $\mathcal{G}$. In particular, we will use the following two summary parameters

$$\lambda_{\max} := \max_{i=1,\ldots,n} \lambda_d\big(\mathbf{Q}^{(i)}\big), \text{ and } \bar{\lambda}_{\min} := \lambda_1\Big((1/n)\sum_{i=1}^{n} \mathbf{Q}^{(i)}\Big). \tag{5.4}$$

We first present an upper bound U (see (4.5)) on the eigenvalues of the matrix $\mathbf{Q}$ in (5.2).

Proposition 5.1. *The eigenvalues of* $\mathbf{Q}$ *in* (5.2) *are upper-bounded as*

$$\begin{aligned}\lambda_j(\mathbf{Q}) &\leq \lambda_{\max} + \alpha\lambda_n\big(\mathbf{L}^{(\mathcal{G})}\big) \\ &\leq \underbrace{\lambda_{\max} + \alpha 2 d_{\max}}_{:=U} \text{ , for } j = 1, \ldots, dn. \end{aligned} \tag{5.5}$$

Proof. See Section 5.7.1. □

The next result offers a lower bound on the eigenvalues $\lambda_j(\mathbf{Q})$.

Proposition 5.2. *Consider the matrix* $\mathbf{Q}$ *in* (5.2). *If* $\lambda_2\big(\mathbf{L}^{(\mathcal{G})}\big) > 0$ *(i.e., the empirical graph in* (5.1) *is connected) and* $\bar{\lambda}_{\min} > 0$ *(i.e., the average of the*

matrices $\mathbf{Q}^{(i)}$ *is non-singular), then the matrix* $\mathbf{Q}$ *is invertible and its smallest eigenvalue is lower bounded as*

$$\lambda_1(\mathbf{Q}) \geq \frac{1}{1+\rho^2}\min\{\lambda_2\big(\mathbf{L}^{(\mathcal{G})}\big)\alpha\rho^2, \bar{\lambda}_{\min}/2\}. \tag{5.6}$$

Here, we used the shorthand $\rho := \bar{\lambda}_{\min}/(4\lambda_{\max})$ *(see* (5.4)*).*

Proof. See Section 5.7.2. □

Prop. 5.2 and Prop. 5.1 can provide some guidance for the design choices of GTVMin. According to the convergence analysis of gradient-based methods in Lecture 4, the eigenvalue $\lambda_1\big(\mathbf{Q}\big)$ should be close to $\lambda_{dn}\big(\mathbf{Q}\big)$ to ensure fast convergence. This favours empirical graphs $\mathcal{G}$, with corresponding eigenvalue $\lambda_2\big(\mathbf{L}^{(\mathcal{G})}\big)$ and maximum node degree $d_{\max}$, having a small ratio between the upper bound (5.5) and the lower bound (5.6).

The bounds in (5.5) and (5.6) also depend on the GTVMin parameter α. While these bounds might provide some guidance for the choice of α, its effect on the convergence speed of the gradient step (5.3) is complicated. For a fixed value of learning rate in (5.3), using larger values for α might slow down the convergence of (5.3) for some collections of local datasets but speed up the convergence of (5.3) for another collection of local datasets.

5.3 Message Passing Implementation

We now discuss in more detail the implementation of gradient-based methods to solve the GTVMin instances with a differentiable objective function $f(\mathbf{w})$. One example for such an instance is (5.1). The core computational step of gradient-based methods is the gradient step

$$\mathbf{w}^{(k+1)} := \mathbf{w}^{(k)} - \eta\nabla f\big(\mathbf{w}^{(k)}\big). \tag{5.7}$$

The iterate $\mathbf{w}^{(k)}$ contains local model parameters $\mathbf{w}^{(i,k)}$,

$$\mathbf{w}^{(k)} =: \mathrm{stack}\big\{\mathbf{w}^{(i,k)}\big\}_{i=1}^{n}. \tag{5.8}$$

Inserting (5.1) into (5.7), we obtain the gradient step

$$\mathbf{w}^{(i,k+1)} := \mathbf{w}^{(i,k)} + \eta\Big[\underbrace{(2/m_i)\big(\mathbf{X}^{(i)}\big)^T\big(\mathbf{y}^{(i)} - \mathbf{X}^{(i)}\mathbf{w}^{(i,k)}\big)}_{\text{(I)}} + \underbrace{2\alpha\sum_{i' \in \mathcal{V}\setminus\{i\}} A_{i,i'}\big(\mathbf{w}^{(i',k)} - \mathbf{w}^{(i,k)}\big)}_{\text{(II)}}\Big]. \tag{5.9}$$

The update (5.9) consists of two components, denoted (I) and (II). The component (I) is nothing but the negative gradient $-\nabla L_i\left(\mathbf{w}^{(i,k)}\right)$ of the local loss $L_i\left(\mathbf{w}^{(i)}\right) := (1/m_i)\left\|\mathbf{y}^{(i)} - \mathbf{X}^{(i)}\mathbf{w}^{(i)}\right\|_2^2$. Component (I) drives the local model parameters $\mathbf{w}^{(i,k+1)}$ towards the minimum of $L_i\left(\cdot\right)$, i.e., having a small deviation between labels $y^{(i,r)}$ and the predictions $\big(\mathbf{w}^{(i,k+1)}\big)^T\mathbf{x}^{(i,r)}$. Note that we can rewrite the component (I) in (5.9), as

$$(2/m_i)\sum_{r=1}^{m_i}\mathbf{x}^{(i,r)}\big(y^{(i,r)} - \big(\mathbf{x}^{(i,r)}\big)^T\mathbf{w}^{(i,k)}\big). \tag{5.10}$$

The purpose of component (II) in (5.9) is to force the local model parameters to be similar across an edge $\{i, i'\}$ with large weight $A_{i,i'}$. We control the relative importance of (II) and (I) using the GTVMin parameter α: Choosing a large value for α puts more emphasis on enforcing similar local model parameters across the edges. Using a smaller α puts more emphasis on learning local model parameters delivering accurate predictions (incurring a small loss) on the local dataset.

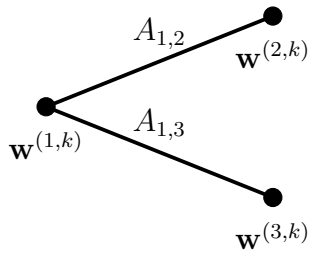


Figure 5.1: At the beginning of iteration k, node $i = 1$ collects the current local model parameters $\mathbf{w}^{(2,k)}$ and $\mathbf{w}^{(3,k)}$ from its neighbours. Then, it computes the gradient step (5.9) to obtain the new local model parameters $\mathbf{w}^{(1,k+1)}$. These updated parameters are then used in the next iteration for the local updates at the neighbours $i = 2, 3$.

The execution of the gradient step (5.9) requires only local information at node i. Indeed, the update (5.9) node i depends only on its current model parameters $\mathbf{w}^{(i,k)}$, the local loss function $L_i(\cdot)$, the neighbours' model parameters $\mathbf{w}^{(i',k)}$, for $i' \in \mathcal{N}^{(i)}$, and the corresponding edge weights $A_{i,i'}$ (see Figure 5.1). In particular, the update (5.9) does not depend on any properties (model parameters or edge weights) of the empirical graph beyond the neighbours $\mathcal{N}^{(i)}$.

We obtain Algorithm 5.1 by repeating the gradient step (5.9), simultaneously for each node $i \in \mathcal{V}$, until a stopping criterion is met. Algorithm 5.1 allows for potentially different learning rates $\eta_{k,i}$ at different nodes i and iterations k (see Section 4.3). It is important to note that Algorithm 5.1 requires a synchronous (simultaneous) execution of the updates (5.9) at all nodes $i \in \mathcal{V}$ [48]. Loosely speaking, all nodes i must the use the same "global clock" that maintains the current iteration counter k [49].

At the beginning of iteration k, each node $i \in \mathcal{V}$ sends their current model

Algorithm 5.1 FedGD for Local Linear Models

Input: empirical graph $\mathcal{G}$; GTV parameter α; learning rate $\eta_{k,i}$

local dataset $\mathcal{D}^{(i)} = \left\{\left(\mathbf{x}^{(i,1)}, y^{(i,1)}\right), \ldots, \left(\mathbf{x}^{(i,m_i)}, y^{(i,m_i)}\right)\right\}$ for each i; some stopping criterion.

Output: linear model parameters $\widehat{\mathbf{w}}^{(i)}$ for each node $i \in \mathcal{V}$

Initialize: $k{:=}0$; $\mathbf{w}^{(i,0)}{:=}\mathbf{0}$

1: **while** stopping criterion is not satisfied **do**
2: **for** all nodes $i \in \mathcal{V}$ (simultaneously) **do**
3: share local model parameters $\mathbf{w}^{(i,k)}$ with neighbours $i' \in \mathcal{N}^{(i)}$
4: update local model parameters via (5.9)
5: **end for**
6: increment iteration counter: $k{:=}k{+}1$
7: **end while**
8: $\widehat{\mathbf{w}}^{(i)} := \mathbf{w}^{(i,k)}$ for all nodes $i \in \mathcal{V}$

parameters $\mathbf{w}^{(i,k)}$ to their neighbours $i' \in \mathcal{N}^{(i)}$. Then, each node $i \in \mathcal{V}$ updates their model parameters according to (5.9) resulting in the updated model parameters $\mathbf{w}^{(i,k+1)}$. Right after these local updates are completed, the global clock increments the counter $k := k + 1$ and triggers the next iteration to be executed by all nodes.

??? MAYBE ADD A COMPUTATION GRAPH TO ILLUSTRATE ABOVE DISCUSSION ????

The implementation of Algorithm 5.1 in real-world computational infrastructures might incur deviations from the exact synchronous execution of (5.9). This deviation can be modelled as a perturbation of the gradient step (5.7) and therefore analyzed using the concepts of Section 4.5 on perturbed GD. Section 8.3 will also discuss the effect of imperfect computation in the context of key requirements for trustworthy FL.

5.4 FedSGD

For some applications, it might be infeasible to compute the sum (5.10) exactly for each gradient step. For example, local datasets might consist of a large number of data points which cannot be accessed quickly enough (stored in the cloud). It might then be useful to approximate the sum by

$$\underbrace{(2/B)\sum_{r\in\mathcal{B}} \mathbf{x}^{(i,r)}\big(y^{(i,r)} - \big(\mathbf{x}^{(i,r)}\big)^{T}\mathbf{w}^{(i,k)}\big)}_{\approx(5.10)}. \tag{5.11}$$

The approximation (5.11) uses a subset (so called "batch")

$$\mathcal{B} = \big\{\big(\mathbf{x}^{(r_1)}, y^{(r_1)}\big), \ldots, \big(\mathbf{x}^{(r_B)}, y^{(r_B)}\big)\big\}$$

of B randomly chosen data points from $\mathcal{D}^{(i)}$. While (5.10) requires summing over m data points, the approximation requires to to sum over B (typically $B \ll m$) data points.

Inserting the approximation (5.11) into the gradient step (5.9) yields the approximate gradient step

$$\mathbf{w}^{(i,k+1)} := \mathbf{w}^{(i,k)} + \eta\Bigg[\underbrace{(2/B)\sum_{r\in\mathcal{B}} \mathbf{x}^{(i,r)}\Big(y^{(i,r)} - \big(\mathbf{x}^{(i,r)}\big)^{T}\mathbf{w}^{(i,k)}\Big)}_{\approx(5.10)} + 2\alpha\sum_{i'\in\mathcal{V}\setminus\{i\}} A_{i,i'}\big(\mathbf{w}^{(i',k)} - \mathbf{w}^{(i,k)}\big)\Bigg]. \tag{5.12}$$

We obtain Algorithm 5.2 from Algorithm 5.1 by replacing the gradient step (5.9) with the approximation (5.12).

Algorithm 5.2 FedSGD for Local Linear Models

Input: empirical graph $\mathcal{G}$; GTV parameter α; learning rate $\eta_{k,i}$; local datasets $\mathcal{D}^{(i)} = \left\{\left(\mathbf{x}^{(i,1)}, y^{(i,1)}\right), \ldots, \left(\mathbf{x}^{(i,m_i)}, y^{(i,m_i)}\right)\right\}$ for each node i; batch size B; some stopping criterion.

Output: linear model parameters $\widehat{\mathbf{w}}^{(i)}$ at each node $i \in \mathcal{V}$

Initialize: $k{:=}0$; $\mathbf{w}^{(i,0)}{:=}\mathbf{0}$

1: **while** stopping criterion is not satisfied **do**
2: **for** all nodes $i \in \mathcal{V}$ (simultaneously) **do**
3: share local model parameters $\mathbf{w}^{(i,k)}$ with all neighbours $i' \in \mathcal{N}^{(i)}$
4: draw fresh batch $\mathcal{B}^{(i)} := \{r_1, \ldots, r_B\}$
5: update local model parameters via (5.12)
6: **end for**
7: increment iteration counter $k{:=}k{+}1$
8: **end while**
9: $\widehat{\mathbf{w}}^{(i)} := \mathbf{w}^{(i,k)}$ for all nodes $i \in \mathcal{V}$

5.5 FedAvg

Consider a FL use-case that requires to learn model parameters $\widehat{\mathbf{w}} \in \mathbb{R}^d$ for a single (global) linear model from local datasets $\mathcal{D}^{(i)}$, $i = 1, \ldots, n$. How can we learn $\widehat{\mathbf{w}}$ without exchanging local datasets but only some model parameters (updates)?

One approach could be to use GTVMin (5.1) and choose α such that its solutions $\widehat{\mathbf{w}}^{(i)}$ are identical for all nodes $i \in \mathcal{V}$. We can interpret the local model parameters delivered by GTVMin as a local copy of the global model parameters. According to our analysis of GTVMin in Lecture 3 (in particular, the upper bound in Prop. 3.1), we can enforce this agreement by a sufficiently large parameter α.

Note that the bound in Prop. 3.1 only applies if the empirical graph (used in GTVMin) is connected. One example for a connected empirical graph is the star as depicted in Figure 5.2. Here, we choose one node $i = 1$ as a centre node that is connected by an edge with weight $A_{1,i}$ to the remaining nodes $i = 2, \ldots, n$. The star graph is distinct in the sense of using the minimum number of edges required to connect n nodes [50].

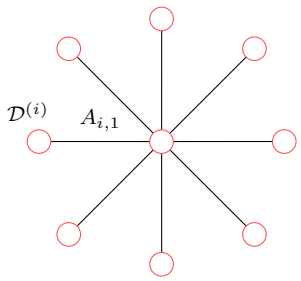


Figure 5.2: Star graph $\mathcal{G}^{(\text{star})}$ with centre node representing a server and peripheral nodes representing clients that generate local datasets.

Instead of using GTVMin with a connected empirical graph and a large value of α, we can also enforce identical local copies $\widehat{\mathbf{w}}^{(i)}$ via a constraint:

$$\widehat{\mathbf{w}} \in \arg\min_{\mathbf{w}\in\mathcal{C}} \sum_{i\in\mathcal{V}} (1/m_i) \left\|\mathbf{y}^{(i)} - \mathbf{X}^{(i)}\mathbf{w}^{(i)}\right\|_2^2$$
$$\text{with } \mathcal{C} = \big\{\mathbf{w} = \text{stack}\{\mathbf{w}^{(i)}\}_{i=1}^n : \mathbf{w}^{(i)} = \mathbf{w}^{(i')} \text{ for any } i, i' \in \mathcal{V}\big\}. \quad (5.13)$$

Note that the constraint set $\mathcal{C}$ is nothing but the subspace defined in (3.14). The projection of a given collection of local model parameters $\mathbf{w} = \text{stack}\{\mathbf{w}^{(i)}\}$ on $\mathcal{C}$ is given by

$$P_{\mathcal{C}}\big(\mathbf{w}\big) = \big(\mathbf{v}^T, \ldots, \mathbf{v}^T\big)^T \text{ with } \mathbf{v} := (1/n)\sum_{i\in\mathcal{V}} \mathbf{w}^{(i)}. \quad (5.14)$$

We can solve (5.13) using projected GD from Lecture 4 (see Section 4.6). The resulting projected gradient step for solving (5.13) is

$$\widehat{\mathbf{w}}_k^{(i)} := \underbrace{\mathbf{w}^{(i,k)} + \eta_{i,k}(2/m_i)\big(\mathbf{X}^{(i)}\big)^T\big(\mathbf{y}^{(i)} - \mathbf{X}^{(i)}\mathbf{w}^{(i,k)}\big)}_{\text{(local gradient step)}} \quad (5.15)$$

$$\mathbf{w}^{(i,k+1)} := (1/n)\sum_{i'\in\mathcal{V}} \widehat{\mathbf{w}}_k^{(i')} \quad \text{(projection)}. \quad (5.16)$$

We can implement this iteration of projected GD conveniently in a server-client system with each node i being a client. Indeed, the first step (5.15) implements, separately for each node i, a gradient step towards a minimum of the local loss $\left\|\mathbf{y}^{(i)} - \mathbf{X}^{(i)}\mathbf{w}\right\|_2^2$. Each node i sends the result $\widehat{\mathbf{w}}_k^{(i)}$ of its local gradient step to a server. After receiving these vectors from all nodes $i \in \mathcal{V}$, the server computes the projection step (5.16). This projection results in the new local model parameters $\mathbf{w}^{(i,k+1)}$ that are sent back to each client i.

The averaging step (5.16) might take much longer to execute than the local update step (5.15). Indeed, (5.16) 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 (5.15) it must wait until the server (i) has collected the updates $\widehat{\mathbf{w}}_k^{(i)}$ from all clients and (ii) sent back their average $\mathbf{w}^{(i,k+1)}$ to $i \in \mathcal{V}$.

Instead of using a computational cheap gradient step (5.15),[8] and then being forced to wait for receiving $\mathbf{w}^{(i,k+1)}$ back from the server, a client might "make better use of the time". For example, the client i could execute several local gradient steps (5.15) in order to make more progress towards the optimum. Another option is to use a local minimization of $L_i(\mathbf{v}) := (1/m_i)\left\|\mathbf{y}^{(i)} - \mathbf{X}^{(i)}\mathbf{v}\right\|_2^2$ around $\mathbf{w}^{(i,k)}$,

$$\widehat{\mathbf{w}}_k^{(i)} := \operatorname*{argmin}_{\mathbf{v}\in\mathbb{R}^d} \Big[\underbrace{(1/m_i)\left\|\mathbf{y}^{(i)} - \mathbf{X}^{(i)}\mathbf{v}\right\|_2^2}_{=L_i(\mathbf{v})} + (1/\eta)\left\|\mathbf{v} - \mathbf{w}^{(i,k)}\right\|_2^2 \Big]. \tag{5.17}$$

Note that (5.17) is nothing but the proximity operator of $L_i(\mathbf{v})$ (see (4.19)). We obtain Algorithm 5.3 from (5.15)-(5.16) by replacing the gradient step (5.15) with the local minimization (5.17).

The local minimization step (5.17) is closely related to ridge regression. Indeed, (5.17) is obtained from ridge regression (2.25) by replacing the regularizer $\mathcal{R}\{\mathbf{v}\} := \|\mathbf{v}\|_2^2$ with the regularizer $\mathcal{R}\{\mathbf{v}\} := \left\|\mathbf{v} - \widehat{\mathbf{w}}_k^{(i)}\right\|_2^2$.

As the notation in (5.17) indicates, the parameter η plays a role similar to the learning rate of a gradient step (4.2). It controls the size of the neighbourhood of $\mathbf{w}^{(i,k)}$ over which (5.17) optimizes the local loss function $L_i(\cdot)$. Choosing a small η forces the update (5.17) to not move too far from

[8] For a large local dataset, the local gradient step (5.15) might actually be computationally expensive and should be replaced by an approximation, e.g., based on the stochastic gradient approximation (5.11).

the given model parameters $\mathbf{w}^{(i,k)}$.

We can interpret the update (5.17) as a form of ERM (2.3) for linear regression. Indeed,

$$\min_{\mathbf{v}\in\mathbb{R}^d}\left[(1/m_i)\left\|\mathbf{y}^{(i)}-\mathbf{X}^{(i)}\mathbf{v}\right\|_2^2+(1/\eta)\left\|\mathbf{w}^{(i,k)}-\mathbf{v}\right\|_2^2\right]=$$
$$\min_{\mathbf{v}\in\mathbb{R}^d}\left[(1/m_i)\sum_{r=1}^{m_i}\left(y^{(i,r)}-\left(\mathbf{x}^{(i,r)}\right)^T\mathbf{v}\right)^2+(1/\eta)\left\|\mathbf{w}^{(i,k)}-\mathbf{v}\right\|_2^2\right]=$$
$$\min_{\mathbf{v}\in\mathbb{R}^d}(1/m_i)\left[\sum_{r=1}^{m_i}\left(y^{(i,r)}-\left(\mathbf{x}^{(i,r)}\right)^T\mathbf{v}\right)^2+\sum_{r=1}^{d}(m_i/\eta)\left(w_r^{(i,k)}-\left(\mathbf{e}^{(r)}\right)^T\mathbf{v}\right)^2\right]. \quad (5.18)$$

Here, $\mathbf{e}^{(r)}$ denotes the vector obtained from the rth column of the identity matrix $\mathbf{I}$. The only difference between (5.18) and (2.3) is the presence of the sample weights (m_i/η).[9]

[9]The "`.fit()`" method of the Python class `sklearn.linear_model.LinearRegression` allows to specify sample weights via the parameter `sample_weight` [26].

Algorithm 5.3 FedAvg to train a linear model

Input: client list $\mathcal{V}$

Server. Initialize. $k := 0$

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

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

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

5: $k := k+1$

6: **end while**

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

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

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

3: update local model parameters by RERM (see (5.17))

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

4: send $\mathbf{w}^{(i)}$ back to server

5: **end while**

5.6 Overview of Coding Assignment

Python Notebook. `FLAlgorithms_CodingAssignment.ipynb`

Data File. `Assignment_MLBasicsData.csv`

This coding assignment builds on the coding assignment "ML Basics" (see Section 2.7) and the coding assignment "FL Design Principle" (see Section 3.4). In particular, we consider an empirical graph $\mathcal{G}^{(\text{FMI})}$ with each node $i \in \mathcal{V}$ being a FMI weather station. The node $i \in \mathcal{V}$ holds a local dataset $\mathcal{D}^{(i)}$ that consists of m_i data points. Each data point is a temperature measurement, taken at station i, and characterized by $d = 7$ features $\mathbf{x} = \big(x_1, \ldots, x_7\big)^T$ and a label y which is the temperature measurement itself. The features are (normalized) values of the latitude and longitude of the FMI station as well as the (normalized) year, month, day, hour, minute when the temperature has been measured.

The edges of $\mathcal{G}^{(\text{FMI})}$ are obtained using the `Python` function `add_edges()`. Each FMI station i is connected to its nearest neighbours i', using the Euclidean distance between the corresponding vectors $\big(\text{lat}^{(i)}, \text{lon}^{(i)}\big)^T \in \mathbb{R}^2$. The number of neighbours is controlled by the input parameter `numneighbors`. All edges $\{i, i'\} \in \mathcal{E}$ have the same edge weight $A_{i,i'} = 1$.

Your tasks involve

- Construct the empirical graph $\mathcal{G}^{(\text{FMI})}$ as a `networkx.Graph()` object.
- For each node $i \in \mathcal{V}$ (FMI station), add node attributes that store the feature matrix (as `numpy` arrays) $\mathbf{X}^{(i)}$ and label vector $\mathbf{y}^{(i)}$. The r-th row of these holds the features $\mathbf{x}^{(i,r)}$ and label $y^{(i,r)}$, respectively, of the

r-th data point recorded for FMI station i in the csv file.

- For each node $i \in \mathcal{V}$ (FMI station), add a node attribute that stores local model parameters $\mathbf{w}^{(i)}$ (as a `numpy` array).
- Split each local dataset into a training set and a validation set.
- Learn local model parameters for a linear model by applying FedGD and FedSGD to the local training sets. Diagnose the resulting model parameters by computing the average (over all nodes) training error and validation error.
- Try to develop some intuition for the effect of choosing different values for the hyper-parameters such as GTVMin parameter α, learning rate η or batch size in Algorithms 5.1 and 5.2.

5.7 Proofs

5.7.1 Proof of Proposition 5.1

The first inequality in (5.5) follows from well-known results on the eigenvalues of a sum of symmetric matrices (see, e.g., [36, Thm 8.1.5]). In particular,

$$\lambda_{\max}\big(\mathbf{Q}\big) \leq \max \left\{ \underbrace{\max_{i=1,\ldots,n} \lambda_d\big(\mathbf{Q}^{(i)}\big)}_{\stackrel{(5.4)}{=}\lambda_{\max}}, \lambda_{\max}\big(\alpha \mathbf{L}^{(\mathcal{G})} \otimes \mathbf{I}\big) \right\}. \tag{5.19}$$

The second inequality in (5.5) uses the following upper bound on the maximum eigenvalue $\lambda_n\big(\mathbf{L}^{(\mathcal{G})}big\big)$ of the Laplacian matrix:

$$\begin{aligned}
\lambda_n\big(\mathbf{L}^{(\mathcal{G})}\big) &\stackrel{(a)}{=} \max_{\|\mathbf{w}\|_2^2=1} \mathbf{w}^T \mathbf{L}^{(\mathcal{G})} \mathbf{w} \\
&\stackrel{(3.8)}{=} \max_{\|\mathbf{w}\|_2^2=1} \sum_{\{i,i'\} \in \mathcal{E}} A_{i,i'} \big(w_i - w_{i'}\big)^2 \\
&\stackrel{(b)}{\leq} \max_{\|\mathbf{w}\|_2^2=1} \sum_{\{i,i'\} \in \mathcal{E}} A_{i,i'} 2\big(w_i^2 + w_{i'}^2\big) \\
&\stackrel{(c)}{=} \max_{\|\mathbf{w}\|_2^2=1} \sum_{i \in \mathcal{V}} 2 w_i^2 \sum_{i' \in \mathcal{N}^{(i)}} A_{i,i'} \\
&\stackrel{(3.5)}{\leq} \max_{\|\mathbf{w}\|_2^2=1} \sum_{i \in \mathcal{V}} 2 w_i^2 d_{\max} \\
&= 2 d_{\max}.
\end{aligned} \tag{5.20}$$

Here, step (a) uses Courant–Fischer–Weyl min-max characterization [36, Thm. 8.1.2.] and step (b)) uses the inequality $(u+v)^2 \leq 2(u^2+v^2)$, which is valid for any $u, v \in \mathbb{R}$. The bound (5.20) is essentially tight.[10]

[10] Consider an empirical graph being a chain (or path).

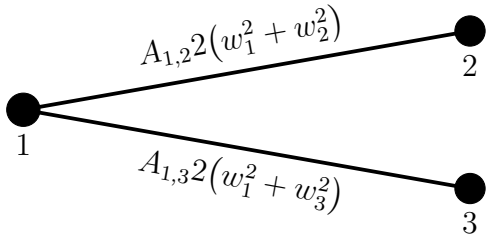


Figure 5.3: Illustration of step (c) in (5.20).

5.7.2 Proof of Proposition 5.2

Similar to the upper bound (5.20) we also start with the Courant–Fischer–Weyl min-max characterization for the eigenvalues of $\mathbf{Q}$ in (5.2). In particular,

$$\lambda_1 = \min_{\|\mathbf{w}\|_2^2=1} \mathbf{w}^T\mathbf{Q}\mathbf{w}. \tag{5.21}$$

We next analyze the RHS of (5.21) by partitioning the constraint set $\{\mathbf{w} : \|\mathbf{w}\|_2^2 = 1\}$ of (5.21) into two complementary regimes for the optimization variable $\mathbf{w} = \text{stack}\{\mathbf{w}^{(i)}\}$. To define these two regimes, we use the orthogonal decomposition

$$\mathbf{w} = \overline{\mathbf{w}} + \widetilde{\mathbf{w}}. \tag{5.22}$$

The component $\overline{\mathbf{w}} = \text{stack}\big\{\mathbf{c}\big\}_{i=1}^n$ consists of identical local model parameters $\mathbf{c} = (1/n)\sum_{i=1}^n \mathbf{w}^{(i)}$, for each node $i \in \mathcal{V}$, and is the orthogonal projection of $\mathbf{w}$ on the subspace (3.14). The second component $\widetilde{\mathbf{w}}$ is the orthogonal projection on the complement of (3.14). Note that

$$\|\mathbf{w}\|_2^2 = \|\overline{\mathbf{w}}\|_2^2 + \|\widetilde{\mathbf{w}}\|_2^2\,. \tag{5.23}$$

Regime I. This regime is obtained for $\|\widetilde{\mathbf{w}}\|_2 \geq \rho\,\|\overline{\mathbf{w}}\|_2$. Since $\|\mathbf{w}\|_2^2 = 1$, and due to (5.23), we have

$$\|\widetilde{\mathbf{w}}\|_2^2 \geq \rho^2/(1+\rho^2). \tag{5.24}$$

This implies, in turn, via (3.13) that

$$\mathbf{w}^T\mathbf{Q}\mathbf{w} \overset{(5.2)}{\geq} \alpha\mathbf{w}^T\big(\mathbf{L}^{(\mathcal{G})}\otimes\mathbf{I}\big)\mathbf{w}$$
$$\overset{(5.24)}{\geq} \lambda_2(\mathbf{L})\alpha\rho^2/(1+\rho^2). \tag{5.25}$$

Regime II. This regime is obtained for $\|\widetilde{\mathbf{w}}\|_2 < \rho\,\|\overline{\mathbf{w}}\|_2$. Here we have $\|\overline{\mathbf{w}}\|_2^2 > (1/\rho^2)\big(1-\|\overline{\mathbf{w}}\|_2^2\big)$ and, in turn,

$$n\,\|\mathbf{c}\|_2^2 = \|\overline{\mathbf{w}}\|_2^2 > 1/(1+\rho^2). \tag{5.26}$$

We next develop the RHS of (5.21) according to

$$\begin{aligned}
\mathbf{w}^T\mathbf{Q}\mathbf{w} &\overset{(5.2)}{\geq} \sum_{i=1}^n \big(\mathbf{w}^{(i)}\big)^T\mathbf{Q}^{(i)}\mathbf{w}^{(i)}\\
&\overset{(5.22)}{\geq} \sum_{i=1}^n \big(\mathbf{c}+\widetilde{\mathbf{w}}^{(i)}\big)^T\mathbf{Q}^{(i)}\big(\mathbf{c}+\widetilde{\mathbf{w}}^{(i)}\big)\\
&\overset{(5.26)}{\geq} \|\overline{\mathbf{w}}\|_2^2\,\underbrace{\lambda_1\Big((1/n)\sum_{i=1}^n\mathbf{Q}^{(i)}\Big)}_{\bar{\lambda}_{\min}} + \sum_{i=1}^n\big[2\big(\widetilde{\mathbf{w}}^{(i)}\big)^T\mathbf{Q}^{(i)}\mathbf{c} + \underbrace{\big(\widetilde{\mathbf{w}}^{(i)}\big)^T\mathbf{Q}^{(i)}\widetilde{\mathbf{w}}^{(i)}}_{\geq 0}\big]\\
&\geq \|\overline{\mathbf{w}}\|_2^2\,\bar{\lambda}_{\min} + \sum_{i=1}^n 2\big(\widetilde{\mathbf{w}}^{(i)}\big)^T\mathbf{Q}^{(i)}\mathbf{c}.
\end{aligned} \tag{5.27}$$

To develop (5.27) further, we note that

$$\Big|\sum_{i=1}^n 2\big(\widetilde{\mathbf{w}}^{(i)}\big)^T\mathbf{Q}^{(i)}\mathbf{c}\Big| \overset{(a)}{\leq} 2\lambda_{\max}\,\|\widetilde{\mathbf{w}}\|_2\,\|\overline{\mathbf{w}}\|_2$$
$$\overset{\|\widetilde{\mathbf{w}}\|_2<\rho\|\overline{\mathbf{w}}\|_2}{\leq} 2\lambda_{\max}\rho\,\|\overline{\mathbf{w}}\|_2^2\,. \tag{5.28}$$

Here, step (a) follows from $\max_{\|\mathbf{y}\|_2=1,\|\mathbf{x}\|_2=1}\mathbf{y}^T\mathbf{Q}\mathbf{x} = \lambda_{\max}$. Inserting (5.28) into (5.27) for $\rho = \bar{\lambda}_{\min}/(4\lambda_{\max})$,

$$\mathbf{w}^T\mathbf{Q}\mathbf{w} \geq \|\overline{\mathbf{w}}\|_2^2\,\bar{\lambda}_{\min}/2 \overset{(5.26)}{\geq} (1/(1+\rho^2))\bar{\lambda}_{\min}/2 \tag{5.29}$$

For each $\mathbf{w}$ with $\|\mathbf{w}\|_2^2 = 1$, either (5.25) or (5.29) must hold.

6 Lecture - "FL Main Flavours"

Lecture 3 discussed GTVMin as a main design principle for FL algorithms. GTVMin learns local model parameters that optimally balance the individual local loss with their variation across the edges of an empirical graph. Lecture 5 discussed how to obtain practical FL algorithms. These algorithms solve GTVMin using distributed optimization methods, such as those from Lecture 4.

This lecture discusses important special cases of GTVMin, known as "main flavours". These flavours arise for specific construction of local datasets, choices of local models, measures for their variation and, last but not least, the weighted edges in the empirical graph. We next briefly summarize the resulting main flavours of FL discussed in the following sections.

Section 6.2 discusses single-model FL that learns model parameters of a single (global) model from local datasets. This single-model flavour can be obtained from GTVMin using a connected empirical graph with large edge weights or, equivalently, a sufficient large value for the GTVMin parameter.

Section 6.3 discusses how clustered FL is obtained from GTVMin over empirical graphs with a clustering structure. CFL exploits the presence of clusters (subsets of local datasets) which can be approximated using an i.i.d. assumption. GTVMin captures these clusters if they are well-connected by many (large weight) edges in the empirical graph.

Section 6.4 discusses horizonal FL which is obtained from GTVMin over an empirical graph whose nodes carry different subsets of a single underlying global dataset. Loosely speaking, horizontal FL involves local datasets characterized by the same set of features but obtained from different data points

from an underlying dataset.

Section 6.5 discusses vertical FL which is obtained from GTVMin over an empirical graph whose nodes the same data points but using different features. As an example, consider the local datasets at different public institutions (tax authority, social insurance institute, supermarkets) which contain different informations about the same underlying population (anybody who has a Finnish social security number).

Section 6.6 shows how personalized FL can be obtained from GTVMin by using specific measures for the total variation of local model parameters. For example, using deep neural networks as local models, we might only use the model parameters corresponding to the first few input layers to define the total variation.

6.1 Learning Goals

After this lecture, you should know particular design choices for GTVMin corresponding to some main flavours of FL:

- single-model FL
- CFL (generalization of clustering methods)
- horizontal FL (relation to semi-supervised learning)
- personalized FL/multi-task learning
- vertical FL

6.2 Single-Model FL

Some FL use cases require to train a single (global) model $\mathcal{H}$ from a decentralized collection of local datasets $\mathcal{D}^{(i)}$, $i = 1, \ldots, n$ [14, 51]. In what follows we assume that the model $\mathcal{H}$ is parametrized by a vector $\mathbf{w} \in \mathbb{R}^d$. Figure 6.1 depicts a server-client architecture for an iterative FL algorithm that generates a sequence of (global) model parameters $\mathbf{w}^{(k)}$, $k = 1, \ldots..$ After computing the new model parameters $\mathbf{w}^{(k+1)}$, the server broadcasts it to the clients i. During the next iteration, each client i uses the current global model parameters $\mathbf{w}^{(k)}$ to compute a local update $\mathbf{w}^{(i,k)}$ based on its local dataset $\mathcal{D}^{(i)}$. The precise implementation of this local update step depends on the choice for the global model $\mathcal{H}$ (trained by the server). One example for such a local update has been discussed in Lecture 5 (see (5.17)).

Lecture 5 already hinted at an alternative to the server-based system in Figure 6.1. Indeed, we might learn local model parameters $\mathbf{w}^{(i)}$ for each client i using a distributed optimization of GTVMin. We can force the resulting model parameters $\mathbf{w}^{(i)}$ to be (approximately) identical by using a connected empirical graph and a sufficiently large GTVMin parameter α.

To minimize the computational complexity of the resulting single-model FL system, we prefer empirical graphs with small number of edges such as the star graph in Figure 5.2 [50]. However, to increase the robustness against node/link failures we might prefer using an empirical graph that has more edges. This "redundancy" helps to ensure that the empirical graph is connected even after removing some of its edges.

Much like the server-based system from Figure 6.1, GTVMin-based methods using a star graph offers a single point of failure (the server in Figure 6.1

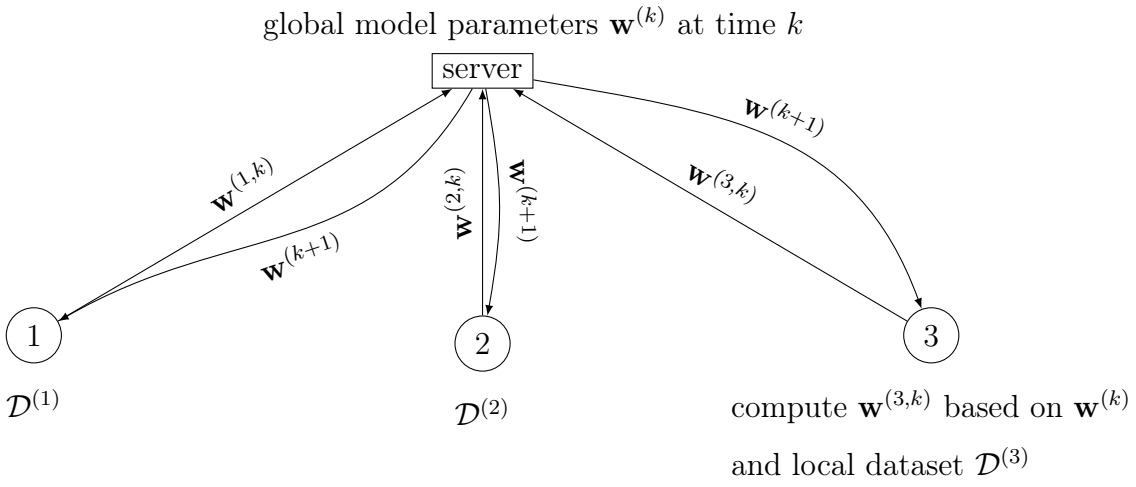


Figure 6.1: The operation of a server-based (or centralized) FL system during iteration k. First, the server broadcasts the current global model parameters $\mathbf{w}^{(k)}$ to each client $i \in \mathcal{V}$. Each client i then computes the update $\mathbf{w}^{(i,k)}$ by combining the previous model parameters $\mathbf{w}^{(k)}$ (received from the server) and its local dataset $\mathcal{D}^{(i)}$. The updates $\mathbf{w}^{(i,k)}$ are then sent back to the server who aggregates them to obtain the updated global model parameters $\mathbf{w}^{(k+1)}$.

or the centre node in Figure 5.2). Lecture 8 will discuss the robustness of GTVMin-based FL systems in slightly more detail (see Section 8.3).

6.3 Clustered FL

Single-model FL systems require the local datasets to be well approximated as i.i.d. realizations from a common underlying probability distribution. However, requiring homogeneous local datasets, generated from the same probability distribution, might be overly restrictive. Indeed, the local datasets might be heterogeneous and need to be modelled using different probability distribution [17, 32].

CFL relaxes the requirement of a common probability distribution underlying all local datasets. Instead, we approximate subsets of local datasets as i.i.d. realizations from a common probability distribution. In other words, CFL assumes that local datasets form clusters. Each cluster $\mathcal{C} \subseteq \mathcal{V}$ has a cluster-specific probability distribution $p^{(\mathcal{C})}$.

The idea of CFL is to pool the local datasets $\mathcal{D}^{(i)}$ in the same cluster $\mathcal{C}$ to obtain a training set to learn cluster-specific $\widehat{\mathbf{w}}^{(\mathcal{C})}$. Each node $i \in \mathcal{C}$ then uses these learnt model parameters $\widehat{\mathbf{w}}^{(\mathcal{C})}$. A main challenge in CFL is that the cluster assignments of the local datasets are unknown in general.

To determine a cluster $\mathcal{C}$, we could apply basic clustering methods, such as k-means or Gaussian mixture model (GMM) to vector representations for local datasets [4, Ch. 5]. We can obtain a vector representation for local dataset $\mathcal{D}^{(i)}$ via the learnt model parameters $\widehat{\mathbf{w}}$ of some parametric ML model that is trained on $\mathcal{D}^{(i)}$.

We can also implement CFL via GTVMin with a suitably chosen empirical

graph. In particular, the empirical graph should contain many edges (with large weight) between nodes in the same cluster and few edges (with small weight) between nodes in different clusters. To fix ideas, consider the empirical graph in Figure 6.3, which contains a cluster $\mathcal{C} = \{1, 2, 3\}$.

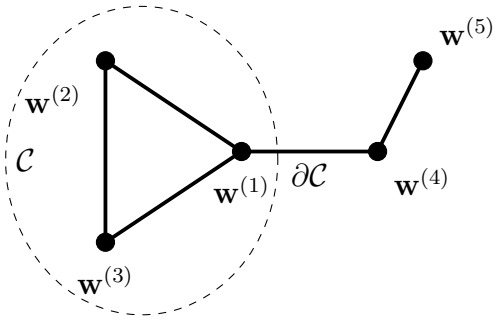


Figure 6.2: The solutions of GTVMin (3.15) are local model parameters that are approximately identical for all nodes in a tight-knit cluster $\mathcal{C}$.

Lecture 3 discussed how the eigenvalues of the Laplacian matrix can be used to measure the connectivity of $\mathcal{G}$. In a similar spirit, we measure the connectivity of a cluster $\mathcal{C}$ via the eigenvalue $\lambda_2\big(\mathbf{L}^{(\mathcal{C})}\big)$ of the Laplacian matrix $\mathbf{L}^{(\mathcal{C})}$ of the induced sub-graph $\mathcal{G}^{(\mathcal{C})}$.[11]

The larger $\lambda_2\big(\mathbf{L}^{(\mathcal{C})}\big)$, the better the connectivity among the nodes in $\mathcal{C}$. While $\lambda_2\big(\mathbf{L}^{(\mathcal{C})}\big)$ describes the intrinsic connectivity of a cluster $\mathcal{C}$, we also need to characterize its connectivity with the other nodes in the empirical graph. To this end, we use the cluster boundary

$$|\partial\mathcal{C}| := \sum_{\{i,i'\}\in\partial\mathcal{C}} A_{i,i'} \text{ with } \partial\mathcal{C} := \big\{\{i,i'\} \in \mathcal{E} : i \in \mathcal{C}, i' \notin \mathcal{C}\big\}. \tag{6.1}$$

[11]The graph $\mathcal{G}^{(\mathcal{C})}$ consists of the nodes in $\mathcal{C}$ and the edges $\{i,i'\} \in \mathcal{E}$ for $i, i' \in \mathcal{C}$.

Note that for a single-node cluster $\mathcal{C} = \{i\}$, the cluster boundary coincides with the node degree, $|\partial\mathcal{C}| = d^{(i)}$ (see (3.4)).

Intuitively, GTVMin tends to deliver (approximately) identical model parameters $\mathbf{w}^{(i)}$ for nodes $i \in \mathcal{C}$ if $\lambda_2\big(\mathbf{L}^{(\mathcal{C})}\big)$ is large and the cluster boundary $|\partial\mathcal{C}|$ is small. The following result makes this intuition more precise for the special case of GTVMin (5.1) for local linear models.

Proposition 6.1. *Consider an empirical graph $\mathcal{G}$ which contains a cluster $\mathcal{C}$ of local datasets with labels $\mathbf{y}^{(i)}$ and feature matrix $\mathbf{X}^{(i)}$ related via*

$$\mathbf{y}^{(i)} = \mathbf{X}^{(i)} \overline{\mathbf{w}}^{(\mathcal{C})} + \boldsymbol{\varepsilon}^{(i)}, \text{ for all } i \in \mathcal{C}. \tag{6.2}$$

We learn local model parameters $\widehat{\mathbf{w}}^{(i)}$ via solving GTVMin (5.1). If the cluster is connected, the error component

$$\widetilde{\mathbf{w}}^{(i)} := \widehat{\mathbf{w}}^{(i)} - (1/|\mathcal{C}|) \sum_{i \in \mathcal{C}} \widehat{\mathbf{w}}^{(i)} \tag{6.3}$$

is upper bounded as

$$\sum_{i \in \mathcal{C}} \big\|\widetilde{\mathbf{w}}^{(i)}\big\|_2^2 \leq \frac{1}{\alpha \lambda_2\big(\mathbf{L}^{(\mathcal{C})}\big)} \Big[\sum_{i \in \mathcal{C}} (1/m_i) \big\|\boldsymbol{\varepsilon}^{(i)}\big\|_2^2 + \alpha \, |\partial\mathcal{C}| \, 2 \Big(\big\|\overline{\mathbf{w}}^{(\mathcal{C})}\big\|_2^2 + R^2 \Big) \Big]. \tag{6.4}$$

Here, we used $R := \max_{i \in \mathcal{V} \setminus \mathcal{C}} \big\|\widehat{\mathbf{w}}^{(i)}\big\|_2$.

Proof. See Sec. 6.9.1. □

The bound (6.4) depends on the cluster $\mathcal{C}$ (via the eigenvalue $\lambda_2\big(\mathbf{L}^{(\mathcal{C})}\big)$ and the boundary $|\partial\mathcal{C}|$) and the GTVMin parameter α. Using a larger $\mathcal{C}$ might result in a decreased eigenvalue $\lambda_2\big(\mathbf{L}^{(\mathcal{C})}\big)$.[12] According to (6.4), we

[12] Consider an empirical graph (with uniform edge weights) that contains a fully connected cluster $\mathcal{C}$ which is connected via a single edge with another node $i' \in \mathcal{V} \setminus \mathcal{C}$. Compare the corresponding eigenvalues $\lambda_2\big(\mathbf{L}^{(\mathcal{C})}\big)$ and $\lambda_2\big(\mathbf{L}^{(\mathcal{C}')}\big)$ of $\mathcal{C}$ and the enlarged cluster $\mathcal{C}' := \mathcal{C} \cup \{i'\}$.

should then increase α to maintain a small deviation $\widetilde{\mathbf{w}}^{(i)}$ of the learnt local model parameters from their cluster-wise average. Thus, increasing α in (3.15) enforces its solutions to be approximately constant over increasingly larger subsets (clusters) of nodes (see Figure 6.3).

Using sufficiently large α for GTVMin over a connected graph enforces its solutions to be approximately identical for any two nodes i, i'. The resulting approximation error is quantified by Prop. 6.1 for the extreme case where the entire empirical graph forms a single cluster, i.e., $\mathcal{C} = \mathcal{V}$. Trivially, the cluster boundary is then equal to 0 and the bound (6.4) specializes to (3.28).

We hasten to add that the bound (6.4) only applies for local datasets that conform with the probabilistic model (6.2). In particular, it assumes that all cluster nodes $i \in \mathcal{C}$ have identical model parameters $\overline{\mathbf{w}}^{(\mathcal{C})}$. Trivially, this is no restriction if we allow for arbitrary error terms $\boldsymbol{\varepsilon}^{(i)}$ in the probabilistic model (6.4). However, as soon as we place additional assumptions on these error terms (such as being realizations of i.i.d. Gaussian RVs) we should verify their validity using principled statistical tests [31, 52]. Finally, we might replace $\left\|\overline{\mathbf{w}}^{(\mathcal{C})}\right\|_2^2$ in (6.4) with an upper bound for this quantity.

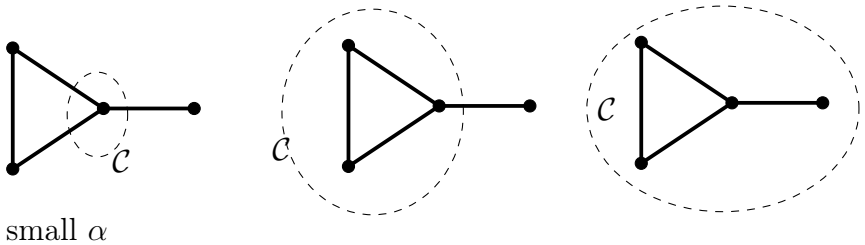


Figure 6.3: The solutions of GTVMin (3.15) become increasingly clustered for increasing α.

6.4 Horizontal FL

Horizontal FL uses local datasets $\mathcal{D}^{(i)}$, for $i \in \mathcal{V}$, that contain data points characterized by the same features [53]. We can think of each local dataset $\mathcal{D}^{(i)}$ as being a subset (or batch) of an underlying global dataset

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

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

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

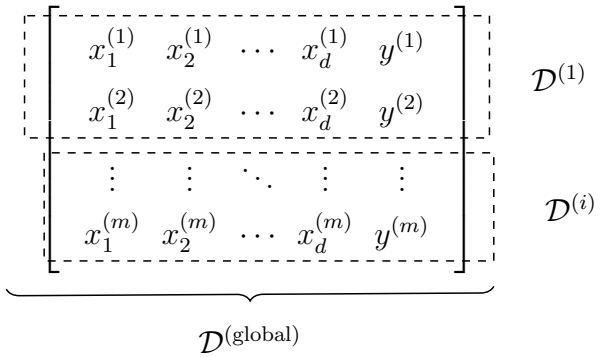


Figure 6.4: Horizontal FL uses the same features to characterize data points in different local datasets. Different local datasets are constituted by different subsets of an underlying global dataset.

We can interpret horizontal FL as a generalization of semi-supervised learning (SSL) [54]: For some local datasets $i \in \mathcal{U}$ we might not have access to the label values of data points. Still, we can use the features of the data points to construct (the weighted edges of) the empirical graph. To implement

SSL, we can solve GTVMin using trivial loss function $L_i\left(\mathbf{w}^{(i)}\right) = 0$ for each "unlabelled" node $i \in \mathcal{U}$.

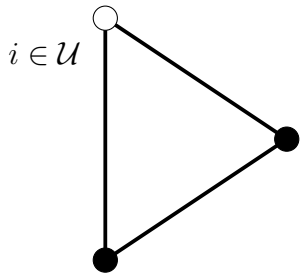


Figure 6.5: Horizontal FL includes SSL as a special case: Some local datasets $i \in \mathcal{U}$ do not contain labels and therefore we must use a trivial loss function $L_i\left(\cdot\right) = 0$. However, we can still use their features to construct an empirical graph for GTVMin based methods. These methods can learn model parameters also for $i \in \mathcal{U}$ by combining the information in local datasets $i \notin \mathcal{U}$ and their connections with nodes in $\mathcal{U}$.

6.5 Vertical FL

Vertical FL uses local datasets that are constituted by the same (identical!) data points. However, each local dataset uses a different choice of features to characterize these data points [55]. Formally, 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 $\mathbf{x}^{(r)}$,

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

6.6 Personalized Federated Learning

Consider GTVMin (3.15) for learning local model parameters $\widehat{\mathbf{w}}^{(i)}$ for each local dataset $\mathcal{D}^{(i)}$. If the value of α in (3.15) is not too large, the local model parameters $\widehat{\mathbf{w}}^{(i)}$ can be different for each $i \in \mathcal{V}$. However, the local model parameters are still coupled via the GTV term in (3.15).

For some FL use-cases we should use different coupling strengths for different components of the local model parameters. For example, if local models are deep ANNs we might enforce the parameters of input layers to be

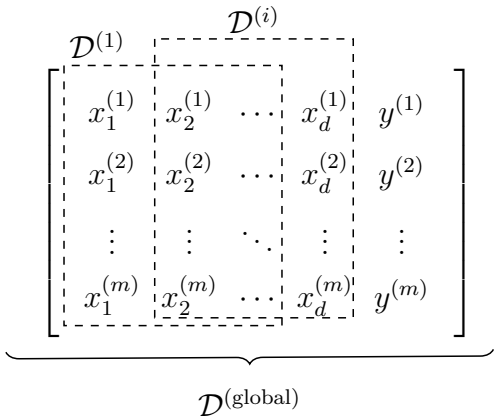


Figure 6.6: Vertical FL uses local datasets that consist of the same (identical!) data points. However, local datasets use different features to characterize data points.

identical while the parameters of the deeper layers might be different for each local dataset.

The partial parameter sharing for local models can be implemented in many different ways [56, Sec. 4.3.]. One way is to use a choice for the GTV penalty function that is different from $\phi = \big\|\mathbf{w}^{(i)} - \mathbf{w}^{(i')}\big\|_2^2$, which is the main choice in our course. In particular, we could construct the penalty function as a combination of two terms,

$$\phi\big(\mathbf{w}^{(i)} - \mathbf{w}^{(i')}\big) := \alpha^{(1)}\phi^{(1)}\big(\mathbf{w}^{(i)} - \mathbf{w}^{(i')}\big) + \alpha^{(2)}\phi^{(2)}\big(\mathbf{w}^{(i)} - \mathbf{w}^{(i')}\big). \tag{6.5}$$

Each component $\phi^{(1)}$, $\phi^{(2)}$ measures different components 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 $\alpha^{(1)}$ and $\alpha^{(2)}$ for different penalty components in (6.5) 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 6.7 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 [57].

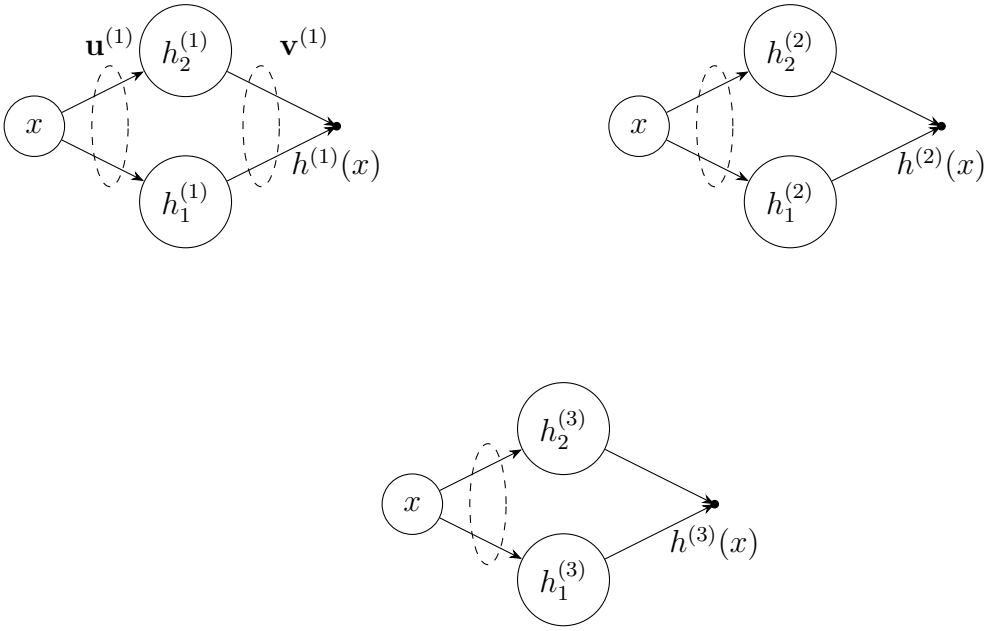


Figure 6.7: Personalized FL with local models being ANNs with one hidden layer. The ANN $h^{(i)}$ is parametrized by the vector $\mathbf{w}^{(i)} = \left(\left(\mathbf{u}^{(i)}\right)^T, \left(\mathbf{v}^{(i)}\right)^T \right)^T$, with parameters $\mathbf{u}^{(i)}$ of hidden layer and the parameters $\mathbf{v}^{(i)}$ of the output layer. We couple the training of $\mathbf{u}^{(i)}$ via GTVMin using the discrepancy measure $\phi = \left\| \mathbf{u}^{(i)} - \mathbf{u}^{(i')} \right\|_2^2$.

6.7 Few-Shot Learning

6.8 Overview of Coding Assignment

Python Notebook. `FLFlavors_CodingAssignment.ipynb`

Data File. `Assignment_MLBasicsData.csv`

This coding assignment builds on the coding assignment "ML Basics" (see Section 2.7) and the coding assignment "FL Design Principle" (see Section 3.4). In particular, we consider an empirical graph $\mathcal{G}^{(\mathrm{FMI})}$ with each node $i \in \mathcal{V}$ being a FMI weather station. The node $i \in \mathcal{V}$ holds a local dataset

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

that consists of m_i data points. Each data point is a temperature measurement, taken at station i, and characterized by $d = 7$ features $\mathbf{x} = \left(x_1, \ldots, x_7\right)^T$ and a label y which is the temperature measurement itself. The features are (normalized) values of the latitude and longitude of the FMI station as well as the (normalized) year, month, day, hour, minute during which this measurements has been taken.

The edges of $\mathcal{G}^{(\mathrm{FMI})}$ are obtained using the `Python` function `add_edges()`. Each FMI station i is connected to its nearest neighbours i', using the Euclidean distance between the corresponding vectors $\left(\mathrm{lat}^{(i)}, \mathrm{lon}^{(i)}\right)^T \in \mathbb{R}^2$. The number of neighbours is controlled by the input parameter `numneighbors`. All edges $\{i, i'\} \in \mathcal{E}$ have the same edge weight $A_{i,i'} = 1$.

Your tasks involve

- Construct the empirical graph $\mathcal{G}^{(\mathrm{FMI})}$ as a `networkx.Graph()` object.
- For each node $i \in \mathcal{V}$ (FMI station), add node attributes that store the feature matrix (as `numpy` arrays) $\mathbf{X}^{(i)}$ and label vector $\mathbf{y}^{(i)}$. The r-th

row of these holds the features $\mathbf{x}^{(i,r)}$ and label $y^{(i,r)}$, respectively, of the r-th data point recorded for FMI station i in the csv file. Add another node attribute that stores the index of the cluster to which this node belongs to.

- Cluster the nodes of $\mathcal{G}^{(\mathrm{FMI})}$ using the Python class `sklearn.cluster.KMeans` which implements the k-means clustering method (see [4, Ch. 8]). This clustering method requires a vector representation $\mathbf{z}^{(i)}$ for each local dataset $\mathcal{D}^{(i)}$. You should try out different choices for the vector representation $\mathbf{z}^{(i)}$ of a local dataset $\mathcal{D}^{(i)}$:
  - $\mathbf{z}^{(i)} \in \mathbb{R}^2$ with entries being the latitude and longitude of the FMI station $i \in \mathcal{V}$.
  - $\mathbf{z}^{(i)}$ is obtained from stacking the parameters of a GMM fitted to the feature vectors $\mathbf{x}^{(i,1)}, \ldots, \mathbf{x}^{(i,m_i)}$.
  - $\mathbf{z}^{(i)} = \big(u_i^{(1)}, u_i^{(2)}, \ldots, u_i^{(k)}\big)^T$ given by the i-th entries of the eigenvectors $\mathbf{u}^{(1)}, \mathbf{u}^{(2)}, \ldots, \mathbf{u}^{(k)}$ of the Laplacian matrix $\mathbf{L}^{(\mathrm{FMI})}$. Note that $\mathbf{u}^{(j)}$ is an eigenvector corresponding to some j-th smallest eigenvalue λ_j of $\mathbf{L}^{(\mathrm{FMI})}$ (see (3.9)).
- For each clustering obtained in the previous task, compute the cluster-wise average temperature $\hat{y}^{(\mathcal{C})} = \frac{1}{\sum_{i' \in \mathcal{C}} m_{i'}} \sum_{i' \in \mathcal{C}} \sum_{r=1}^{m_{i'}} y^{(i',r)}$ for each cluster $\mathcal{C}$. Let $\mathcal{C}^{(i)}$ denote the cluster to which node i belongs, then $\hat{y}^{(i,r)} := \hat{y}^{(\mathcal{C}^{(i)})}$ is a prediction for the actual temperature $y^{(i,r)}$. Compute the average squared error loss $\frac{1}{\sum_{i=1}^{n} m_i} \sum_{i \in \mathcal{V}} \sum_{r=1}^{m_i} \big(y^{(i,r)} - \hat{y}^{(i,r)}\big)^2$.

6.9 Proofs

6.9.1 Proof of Proposition 6.1

We will verify (6.4) via a similar argument as used in the proof (see Section 3.5.1) of Prop. 3.1.

First we decompose the objective function $f\big(\mathbf{w}\big)$ in (5.1) as follows:

$$f(\mathbf{w}) = \underbrace{\sum_{i\in\mathcal{C}}(1/m_i)\left\|\mathbf{y}^{(i)}-\mathbf{X}^{(i)}\mathbf{w}^{(i)}\right\|_2^2+\alpha\Bigg[\sum_{i,i'\in\mathcal{C}}A_{i,i'}\left\|\mathbf{w}^{(i)}-\mathbf{w}^{(i')}\right\|_2^2+\sum_{\{i,i'\}\in\partial\mathcal{C}}A_{i,i'}\left\|\mathbf{w}^{(i)}-\mathbf{w}^{(i')}\right\|_2^2\Bigg]}_{=:f'\left(\mathbf{w}\right)} + f''\big(\mathbf{w}\big). \tag{6.6}$$

Note that only the first component f' depends on the local model parameters $\mathbf{w}^{(i)}$ of cluster nodes $i \in \mathcal{C}$.

Let us introduce the shorthand $f'\big(\mathbf{w}^{(i)}\big)$ for the function obtained from $f'(\mathbf{w})$ for varying $\mathbf{w}^{(i)}$, $i \in \mathcal{C}$, but fixing $\mathbf{w}^{(i)} := \widehat{\mathbf{w}}^{(i)}$ for $i \notin \mathcal{C}$. We verify the bound (6.4) by showing that if it does not hold, the local model parameters $\overline{\mathbf{w}}^{(i)} := \overline{\mathbf{w}}^{(\mathcal{C})}$, for $i \in \mathcal{C}$, results in a smaller value $f'\big(\overline{\mathbf{w}}^{(i)}\big) < f'\big(\widehat{\mathbf{w}}^{(i)}\big)$ than the choice $\widehat{\mathbf{w}}^{(i)}$, for $i \in \mathcal{C}$. This would contradict the fact that $\widehat{\mathbf{w}}^{(i)}$ is a solution to (5.1).

First, note that

$$
\begin{aligned}
f'\big(\overline{\mathbf{w}}^{(i)}\big) &= \sum_{i\in\mathcal{C}}(1/m_i)\left\|\mathbf{y}^{(i)}-\mathbf{X}^{(i)}\overline{\mathbf{w}}^{(\mathcal{C})}\right\|_2^2 \\
&\qquad +\alpha\Bigg[\sum_{\substack{\{i,i'\}\in\mathcal{E}\\ i,i'\in\mathcal{C}}} A_{i,i'}\left\|\overline{\mathbf{w}}^{(\mathcal{C})}-\overline{\mathbf{w}}^{(\mathcal{C})}\right\|_2^2+\sum_{\substack{\{i,i'\}\in\mathcal{E}\\ i\in\mathcal{C},i'\notin\mathcal{C}}} A_{i,i'}\left\|\overline{\mathbf{w}}^{(\mathcal{C})}-\widehat{\mathbf{w}}^{(i')}\right\|_2^2\Bigg] \\
&\overset{(6.2)}{=} \sum_{i\in\mathcal{C}}(1/m_i)\left\|\boldsymbol{\varepsilon}^{(i)}\right\|_2^2+\alpha\sum_{\substack{\{i,i'\}\in\mathcal{E}\\ i\in\mathcal{C},i'\notin\mathcal{C}}} A_{i,i'}\left\|\overline{\mathbf{w}}^{(\mathcal{C})}-\widehat{\mathbf{w}}^{(i')}\right\|_2^2 \\
&\overset{(a)}{\leq} \sum_{i\in\mathcal{C}}(1/m_i)\left\|\boldsymbol{\varepsilon}^{(i)}\right\|_2^2+\alpha\sum_{\substack{\{i,i'\}\in\mathcal{E}\\ i\in\mathcal{C},i'\notin\mathcal{C}}} A_{i,i'}2\Bigg(\left\|\overline{\mathbf{w}}^{(\mathcal{C})}\right\|_2^2+\left\|\widehat{\mathbf{w}}^{(i')}\right\|_2^2\Bigg) \\
&\leq \sum_{i\in\mathcal{C}}(1/m_i)\left\|\boldsymbol{\varepsilon}^{(i)}\right\|_2^2+\alpha\,|\partial\mathcal{C}|\,2\Bigg(\left\|\overline{\mathbf{w}}^{(\mathcal{C})}\right\|_2^2+R^2\Bigg). \qquad (6.7)
\end{aligned}
$$

Step (a) uses the inequality $\|\mathbf{u}+\mathbf{v}\|_2^2 \leq 2\big(\|\mathbf{u}\|_2^2+\|\mathbf{v}\|_2^2\big)$ which is valid for any two vectors $\mathbf{u},\mathbf{v}\in\mathbb{R}^d$.

On the other hand,

$$
\begin{aligned}
f'\big(\widehat{\mathbf{w}}^{(i)}\big) &\geq \alpha\sum_{i,i'\in\mathcal{C}} A_{i,i'}\underbrace{\left\|\widehat{\mathbf{w}}^{(i)}-\widehat{\mathbf{w}}^{(i')}\right\|_2^2}_{\overset{(6.3)}{=}\left\|\widetilde{\mathbf{w}}^{(i)}-\widetilde{\mathbf{w}}^{(i')}\right\|_2^2} \\
&\overset{(3.13)}{\geq} \alpha\lambda_2\big(\mathbf{L}^{(\mathcal{C})}\big)\sum_{i\in\mathcal{C}}\left\|\widetilde{\mathbf{w}}^{(i)}\right\|_2^2. \qquad (6.8)
\end{aligned}
$$

If the bound (6.4) would not hold, then by (6.8) and (6.7) we would obtain $f'\big(\widehat{\mathbf{w}}^{(i)}\big) > f'\big(\overline{\mathbf{w}}^{(i)}\big)$, which contradicts the fact that $\widehat{\mathbf{w}}^{(i)}$ solves (5.1).

7 Lecture - "Graph Learning"

Lecture 3 discussed GTVMin as a main design principle for FL algorithms. In particular, Lecture 5 discusses FL algorithms that arise from the application of optimization methods, such as the gradient-based methods from Lecture 4, to solve GTVMin.

The computational and statistical properties of these algorithms depend crucially on the properties of the underlying empirical graph. For example, the number of edges in the empirical graph is typically proportional to the amount of computation and communication required by FL systems. Moreover, the connectivity of the empirical graph steers the pooling of local datasets into clusters that share common model parameters.

In some applications, domain expertise can guide the choice for the empirical graph. However, it might also be useful to learn the empirical graph in a more data-driven fashion. This lecture discusses methods that learn an empirical graph solely from a given collection of local datasets and corresponding local loss functions.

The outline of this lecture is as follows: Section 7.2 discusses how the computational and statistical properties of Algorithm 5.1 from Lecture 5 can guide the construction of the empirical graph. Section 7.3 presents some ideas for how to measure the discrepancy (lack of similarity) between two local datasets. The measure for the discrepancy is an important design choice for the graph learning methods discussed in Section 7.4. We formulate these methods as the optimization of edge weights given the discrepancy measure for any pair of local datasets. The formulation as an optimization problem allows to include connectivity constraints such as a minimum value for each

node degree.

7.1 Learning Goals

After this lecture, you should

- know how the computational and statistical properties of GTVMin-based methods depend on the structure of the empirical graph,
- know some measures for (dis-)similarity between local datasets,
- be able to learn a graph from given pairwise similarities and structural constraints (e.g., prescribed node degree).

7.2 Empirical Graph is a Design Choice

Consider the GTVMin instance (3.17) for learning the model parameters of a local linear model for each local dataset $\mathcal{D}^{(i)}$. To solve (3.17), we use Algorithm 5.1 as a message passing implementation of the basic gradient step (5.3). Note that GTVMin (3.17) is defined for a given empirical graph $\mathcal{G}$. Therefore, the choice for $\mathcal{G}$ is critical for the statistical and computational properties of Algorithm 5.1.

Statistical Properties. The statistical properties of Algorithm 5.1 can be assessed via a probabilistic model for the local datasets. One important example for such a probabilistic model is the clustering assumption (6.2) of CFL (see Section 3.3.1). For CFL, we would like to learn similar model parameters for nodes in the same cluster.

According to Prop. 6.1, the GTVMin solutions will be approximately constant over $\mathcal{C}$, if $\lambda_2\big(\mathbf{L}^{(\mathcal{C})}\big)$ is large and the cluster boundary $|\partial\mathcal{C}|$ is small.

Here, $\lambda_2\big(\mathbf{L}^{(\mathcal{C})}\big)$ denotes the smallest non-zero eigenvalue of the Laplacian matrix associated with the induced sub-graph $\mathcal{G}^{(\mathcal{C})}$.

Roughly speaking, $\lambda_2\big(\mathbf{L}^{(\mathcal{C})}\big)$ will be larger if there are more edges connecting the nodes in $\mathcal{C}$. This informal statement can be made precise using a celebrated result from spectral graph theory, known as Cheeger's inequality [33, Ch. 21]. Alternatively, we can analyse $\lambda_2\big(\mathbf{L}^{(\mathcal{C})}\big)$ by interpreting (or approximating) the induced sub-graph $\mathcal{G}^{(\mathcal{C})}$ as a typical realization of an Erdős-Rényi (ER) random graph.[13]

The ER model for $\mathcal{G}^{(\mathcal{C})}$ postulates that two nodes $i, i' \in \mathcal{C}$ are connected by an edge with probability p_e. The presence of an edge between a given pair of nodes does not depend on the presence of an edge between any other pair of nodes ("edges are i.i.d.").

Since we model the presence of edges as realizations of RVs also the node degrees $d^{(i)}$ as well as the eigenvalue $\lambda_2\big(\mathbf{L}^{(\mathcal{C})}\big)$ become realizations of RVs. The expected node degree is given as $\mathbb{E}\big\{d^{(i)}\big\} = p_e(|\mathcal{C}|-1)$. With high probability, a realization of an ER graph has maximum node degree

$$d_{\max} \approx \mathbb{E}\big\{d^{(i)}\big\} = p_e(|\mathcal{C}|-1). \tag{7.1}$$

Thus, increasing the ER parameter p_e results in a larger node degree (i.e., a higher connectivity) of $\mathcal{G}^{(\mathcal{C})}$.

We can approximate $\lambda_2\big(\mathbf{L}^{(\mathcal{C})}\big)$ with the second smallest eigenvalue $\lambda_2\big(\overline{\mathbf{L}}\big)$ of the expected Laplacian matrix $\overline{\mathbf{L}} := \mathbb{E}\big\{\mathbf{L}^{(\mathcal{C})}\big\} = |\mathcal{C}|p_e\mathbf{I} - p_e\mathbf{1}\mathbf{1}^T$. A simple

[13] This approximation is particularly useful if the empirical graph $\mathcal{G}$ itself is (close to) a typical realization of an ER graph.

calculation reveals that $\lambda_2\left(\overline{\mathbf{L}}\right) = |\mathcal{C}|p_e$. Thus, we have the approximation

$$\lambda_2\big(\mathbf{L}^{(\mathcal{C})}\big) \approx \lambda_2\big(\overline{\mathbf{L}}\big) = |\mathcal{C}|p_e \overset{(7.1)}{\approx} d_{\max}. \tag{7.2}$$

The precise quantification of the approximation error in (7.2) is beyond the scope of this course. We refer the reader to relevant literature on the theory of random graphs [58, 59].

Computational Properties. The computational complexity of Algorithm 5.1 depends on the amount of computation required by a single iteration of its steps (3) - (4). Clearly, this "per-iteration" complexity of Algorithm 5.1 increases with increasing node degrees $d^{(i)}$. Indeed, the step (3) requires to communicate local model parameters across each edge of the empirical graph. This communication can be implemented by different communication channels such as a short-range wireless link or an optical fibre cable [60, 61].

To summarize, using an empirical graph with smaller $d^{(i)}$ translates into a smaller amount of computation and communication needed during a single iteration of Algorithm 5.1. Trivially, the per-iteration complexity of Algorithm 5.1 is minimized by $d^{(i)} = 0$, i.e., an empty empirical graph without any edges ($\mathcal{E} = \emptyset$). However, the overall computational complexity of Algorithm 5.1 also depends on the number of iterations required to achieve an approximate solution to GTVMin (3.17).

According to (4.6), the convergence speed of the gradient steps (5.9) used in Algorithm 5.1 depends on the condition number $\lambda_{nd}(\mathbf{Q})/\lambda_1(\mathbf{Q})$ of the matrix $\mathbf{Q}$ in (5.2). Algorithm 5.1 tends to require fewer iterations when the ratio between the largest and smallest eigenvalues of $\mathbf{Q}$ is small (closer to 1). The condition number of $\mathbf{Q}$ tends to be smaller for a smaller ratio between

the maximum node degree $d_{\max}$ and eigenvalue $\lambda_2\big(\mathbf{L}^{(\mathcal{G})}\big)$ (see (5.5) and (5.6)).

To summarize, the per-iteration complexity of Algorithm 5.1 increases with the node degrees $d^{(i)}$ of the empirical graph $\mathcal{G}$. On the other hand, the number of iterations required by Algorithm 5.1 decrease with increasing $\lambda_2\big(\mathbf{L}^{(\mathcal{G})}\big)$. Some recent work studies graph constructions that maximize $\lambda_2\big(\mathbf{L}^{(\mathcal{G})}\big)$ for a given (prescribed) maximum node degree $d_{\max} = \max_{i \in \mathcal{V}} d^{(i)}$ [62, 63].

Spectral graph theory also provides upper bounds on $\lambda_2\big(\mathbf{L}^{(\mathcal{G})}\big)$ in terms of the node degrees [33, 35, 64]. These upper bounds can be used as a benchmark for practical constructions of the empirical graph: If some construction results in a value $\lambda_2\big(\mathbf{L}^{(\mathcal{G})}\big)$ close to the upper bound, there is little benefit in trying to further improve the construction (in the sense of achieving higher $\lambda_2\big(\mathbf{L}^{(\mathcal{G})}\big)$. The next result provides one example for such an upper bound.

Proposition 7.1. *Consider an empirical graph $\mathcal{G}$ with n nodes and associated Laplacian matrix $\mathbf{L}^{(\mathcal{G})}$. Then, $\lambda_2\big(\mathbf{L}^{(\mathcal{G})}\big)$ cannot exceed the node degree $d^{(i)}$ of any node by more than a factor $n/(n-1)$. In other words,*

$$\lambda_2\big(\mathbf{L}^{(\mathcal{G})}\big) \leq \frac{n}{n-1} d^{(i)}, \text{ for every } i = 1, \ldots, n. \tag{7.3}$$

Proof. Combine Courant–Fischer–Weyl min-max characterization [36, Thm. 8.1.2.] with the evaluation of the quadratic form $\mathbf{w}^T \mathbf{L}^{(\mathcal{G})} \mathbf{w}$ for the specific vector $\widetilde{\mathbf{w}} = \Big(\tilde{w}^{(1)} = -(1/n), \ldots, \tilde{w}^{(i)} = 1 - (1/n), \ldots, \tilde{w}^{(n)} = -(1/n)\Big)^T$. Note that $\widetilde{\mathbf{w}}^T \mathbf{1} = 0$, i.e., this vector is orthogonal to the subspace (3.14) (for the special case of local models with dimension $d = 1$). □

Alternative (and potentially tighter) upper bounds on $\lambda_2\big(\mathbf{L}^{(\mathcal{G})}\big)$ can be found in the graph theory literature [33, 34, 59, 65].

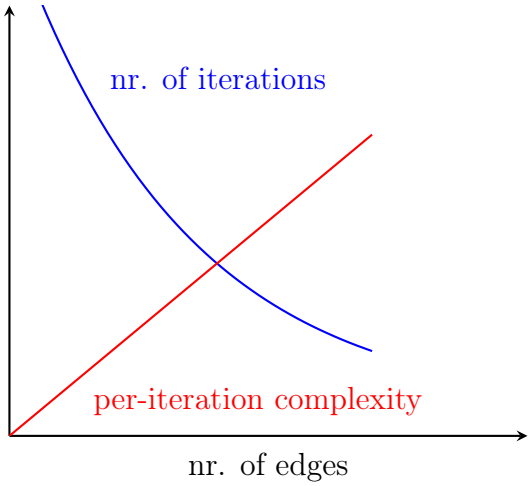


Figure 7.1: The per-iteration complexity and number of iterations required by Algorithm 5.1 depends on the number of edges in the underlying empirical graph in different manners.

7.3 Measuring (Dis-)Similarity Between Datasets

The whole idea of GTVMin is to enforce similar model parameters at two nodes i, i' that are connected by an edge $\{i, i'\}$ with (relatively) large edge weight $A_{i,i'}$. In general, the edges (and their weights) of the empirical graph are a design choice. However, the edges should somehow reflect the actual similarities between local datasets. We next discuss different approaches to measuring the similarity or, equivalently, the discrepancy (the lack of similarity) between two local datasets.

The first approach is based on a probabilistic model, i.e., we interpret the local dataset $\mathcal{D}^{(i)}$ as realizations of RVs with some parametrized probability distribution $p^{(i)}\big(\mathcal{D}^{(i)}; \mathbf{w}^{(i)}\big)$. We can then measure the discrepancy between $\mathcal{D}^{(i)}$ and $\mathcal{D}^{(i')}$ via the Euclidean distance $\big\|\mathbf{w}^{(i)} - \mathbf{w}^{(i')}\big\|_2$ between the parameters $\mathbf{w}^{(i)}, \mathbf{w}^{(i')}$ of the probability distributions.

In general, we do not know the parameters of the probability distribution $p^{(i)}\big(\mathcal{D}^{(i)}; \mathbf{w}^{(i)}\big)$ underlying a local dataset.[14] We might still be able to estimate these parameters, e.g., using variants of maximum likelihood [4, Ch. 3]. Given the estimates $\widehat{\mathbf{w}}^{(i)}, \widehat{\mathbf{w}}^{(i')}$ for the model parameters, we can then compute the discrepancy measure $d^{(i,i')} := \big\|\widehat{\mathbf{w}}^{(i)} - \widehat{\mathbf{w}}^{(i')}\big\|_2$.

Example. Consider local datasets being a single number $y^{(i)}$ which is modelled as a noisy observation $y^{(i)} = w^{(i)} + n^{(i)}$ with $n^{(i)} \sim \mathcal{N}(0,1)$. The maximum likelihood estimator for $w^{(i)}$ is then obtained as $\hat{w}^{(i)} = y^{(i)}$ [29, 66] and, in turn, the resulting discrepancy measure $d^{(i,i')} := \big|y^{(i)} - y^{(i')}\big|$ [67].

Example. Consider local datasets $\mathcal{D}^{(i)}$ constituted by images of handwritten digits $0, 1, \ldots, 9$. We model a local dataset using a hierarchical probabilistic model: Each node $i \in \mathcal{V}$ is assigned a deterministic but unknown distribution $\boldsymbol{\alpha}^{(i)} = \big(\alpha_1^{(i)}, \ldots, \alpha_9^{(i)}\big)$. The entry $\alpha_j^{(i)}$ is the fraction of images at node i that show digit j. We interpret the labels $y^{(i,1)}, \ldots, y^{(i,m_i)}$ as realizations of i.i.d. RVs, with values in $\{0, 1, \ldots, 9\}$ and distributed according to $\boldsymbol{\alpha}^{(i)}$. The features are interpreted as realizations of RVs with conditional distribution $p(\mathbf{x}|y)$ which is the same for all nodes $i \in \mathcal{V}$. A natural estimator for the dis-similarity between $\mathcal{D}^{(i)}$ and $\mathcal{D}^{(i')}$ is then the distance between estimators for the parameters $\boldsymbol{\alpha}^{(i)}$ and $\boldsymbol{\alpha}^{(i')}$.

The above discrepancy measure construction (using estimates for the parameters of a probabilistic model) is a special case of a more generic two-step approach:

- First, we determine a vector representation $\mathbf{z}^{(i)} \in \mathbb{R}^{m'}$ for each local

[14] One exception is when we generate the local dataset by drawing i.i.d. realizations from $p^{(i)}\big(\mathcal{D}^{(i)}; \mathbf{w}^{(i)}\big)$.

dataset $\mathcal{D}^{(i)}$ [4, 68].

- Second, we measure the discrepancy between local datasets $\mathcal{D}^{(i)}, \mathcal{D}^{(i')}$ via the distance between the corresponding representation vectors $\mathbf{z}^{(i)}, \mathbf{z}^{(i')} \in \mathbb{R}^{m'}$.

One example for constructing a vector representation $\mathbf{z}^{(i)} \in \mathbb{R}^{m'}$ for a local dataset $\mathcal{D}^{(i)}$ is, under an i.i.d. assumption, to use the maximum likelihood estimate $\widehat{\mathbf{w}}^{(i)}$ for the parameters of a probabilistic model $p(\mathbf{x}, y; \mathbf{w}^{(i)})$.

Let us next discuss a construction for the vector representation $\mathbf{z}^{(i)} \in \mathbb{R}^{m'}$ that is motivated by SGD (see Section 5.4). In particular, we could define the discrepancy between two local datasets by interpreting them as two possible batches used by SGD to train a model. If these two batches have similar statistical properties (in the sense of being useful for the model training), then their corresponding gradient approximations (5.11) should be aligned. This suggests to use the gradient $\nabla f(\mathbf{w}')$ of the average loss $f(\mathbf{w}) := (1/|\mathcal{D}^{(i)}|) \sum_{(\mathbf{x},y) \in \mathcal{D}^{(i)}} L\left((\mathbf{x}, y), h^{(\mathbf{w})}\right)$ as a vector representation $\mathbf{z}^{(i)}$ for $\mathcal{D}^{(i)}$. The properties of this representation depend on the choice of parametrized model, i.e., how $h^{(\mathbf{w})}$ depends precisely on the parameters $\mathbf{w}$ and on the model parameters $\mathbf{w}'$ at which the gradient is evaluated.

We can also use an autoencoder [68, Ch. 14] to learn a vector representation for a local dataset. In particular, we fed it into an encoder network which has been trained jointly with a decoder network on some learning task. Figure 7.2 illustrates a generic autoencoder setup.

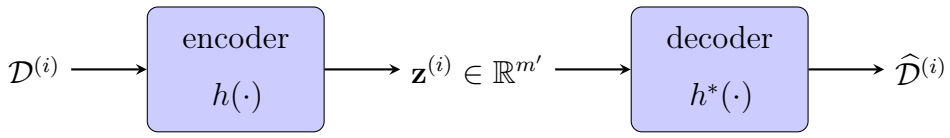


Figure 7.2: An autoencoder consists of an encoder, mapping the input to a latent vector, and a decoder which tries to reconstruct the input as accurately as possible. The encoder and the decoder are trained jointly by minimizing some measure (loss) of the reconstruction error (see [4, Ch. 9]). When using a local dataset as input, we can use the latent vector as its vector representation.

7.4 Graph Learning Methods

Assume we have constructed a useful measure $d^{(i,i')} \in \mathbb{R}_+$ for the discrepancy between any two local datasets $\mathcal{D}^{(i)}, \mathcal{D}^{(i')}$. We could then construct an empirical graph by connecting each node i with its nearest neighbours. The nearest neighbours of i are those other nodes $i' \in \mathcal{V} \setminus \{i\}$ with smallest discrepancy $d^{(i,i')}$. We next discuss an alternative to the nearest-neighbour graph construction. This alternative approach formulates graph learning as a constrained linear optimization problem.

Let us measure the usefulness of a given choice for the edge weights $A_{i,i'} \in \mathbb{R}_+$ via

$$\sum_{i,i' \in \mathcal{V}} A_{i,i'} d^{(i,i')}. \tag{7.4}$$

The objective function (7.4) penalizes having a large edge weight $A_{i,i'}$ between two nodes i, i' with a large discrepancy $d^{(i,i')}$. Note that the objective function (7.4) is minimized by the trivial choice $A_{i,i'} = 0$, i.e., the empty empirical graph without any edges $\mathcal{E} = \emptyset$.

As discussed in Section 7.2, the empirical graph should have a sufficient amount of edges in order to ensure that the GTVMin solutions are useful model parameters. Indeed, the desired pooling effect of GTVMin requires the eigenvalue $\lambda_2\big(\mathbf{L}^{(\mathcal{G})}\big)$ to be sufficiently large. Ensuring a large $\lambda_2\big(\mathbf{L}^{(\mathcal{G})}\big)$ requires, in turn, that the empirical graph $\mathcal{G}$ contains a sufficiently large number of edges and corresponding node degrees (see (7.2)).

We can enforce the presence of edges (with positive weight) by adding constraints to (7.4). For example, we might require

$$A_{i,i} = 0 \ , \ \sum_{i' \neq i} A_{i,i'} = d_{\max} \text{ for all } i \in \mathcal{V}, \ A_{i,i'} \in [0,1] \text{ for all } i, i' \in \mathcal{V}. \quad (7.5)$$

The constraints (7.5) require that each node i is connected with other nodes using total edge weight (weighted node degree) $\sum_{i' \neq i} A_{i,i'} = d_{\max}$.

Combining the constraints (7.5) with the objective function (7.4) results in the following graph learning principle,

$$\begin{aligned} \widehat{A}_{i,i'} \in & \operatorname*{argmin}_{A_{i,i'}=A_{i',i}} \sum_{i,i' \in \mathcal{V}} A_{i,i'} d^{(i,i')} \quad (7.6) \\ & A_{i,i'} \in [0,1] \text{ for all } i, i' \in \mathcal{V}, \\ & A_{i,i} = 0 \text{ for all } i \in \mathcal{V}, \\ & \sum_{i' \neq i} A_{i,i'} = d_{\max} \text{ for all } i \in \mathcal{V}. \end{aligned}$$

Note that (7.6) is an instance of the constrained quadratic minimization problem (4.15). We can therefore use (variants of) projected GD from Section 4.6 to compute (approximate) solutions to (7.6).

The first constraint in (7.6) requires each edge weight to belong to the interval $[0,1]$. The second constraint prohibits any self-loops in the resulting

empirical graph. Indeed, adding self-loops to an empirical graph has no effect on the resulting GTVMin-based method (see (3.15)). The last constraint of the learning principle (7.6) enforces a regular empirical graph: Each node i having the same weighted node degree $d^{(i)} = \sum_{i' \neq i} A_{i,i'} = d_{\max}$.

For some FL applications it might be detrimental to insist on identical node degrees of the empirical graph. Instead, we might prefer other structural properties such as a small total number of edges or presence of few hub nodes (with exceptionally large node degree) [12, 67].

We can enforce an upper bound on the total number $E_{\max}$ of edges by modifying the last constraint in (7.6),

$$\begin{aligned} \widehat{A}_{i,i'} \in & \operatorname*{argmin}_{A_{i,i'}=A_{i',i}} \sum_{i,i' \in \mathcal{V}} A_{i,i'} d^{(i,i')} \\ & A_{i,i'} \in [0,1] \text{ for all } i, i' \in \mathcal{V}, \\ & A_{i,i} = 0 \text{ for all } i \in \mathcal{V}, \\ & \sum_{i',i \in \mathcal{V}} A_{i,i'} = E_{\max}. \end{aligned} \tag{7.7}$$

The problem has a closed-form solution as explained in [67]: It is obtained by placing the edges between those pairs $i, i' \in \mathcal{V}$ that result in the smallest discrepancy $d^{(i,i')}$. However, it might still be useful to implement (7.7) via optimization methods (see Lecture 4) as they can be implemented in a fully distributed fashion as message passing over an underlying communication network which might be different from the learnt empirical graph [69].

7.5 Overview of Coding Assignment

Python Notebook. `GraphLearning_CodingAssignment.ipynb`

Data File. `Assignment_MLBasicsData.csv`

This coding assignment builds on the previous coding assignments "FL Algorithms" (see Section 5.6) and "FL Main Flavors" (see Section 6.8). In particular, we again construct an empirical graph $\mathcal{G}$ with nodes $i \in \mathcal{V}$ representing FMI stations. The node $i \in \mathcal{V}$ holds a local dataset $\mathcal{D}^{(i)}$ that consists of m_i data points. Each of these data points is a temperature measurement, taken at station i, and characterized by $d = 7$ features $\mathbf{x} = \big(x_1, \ldots, x_7\big)^T$ and a label y which is the temperature measurement itself. The features are (normalized) values of the latitude and longitude of the FMI station as well as the (normalized) year, month, day, hour, minute when the temperature has been measured. We then apply some of the graph learning methods from this lecture to construct the edge set of $\mathcal{G}$.

Similar to the coding assignment "FL Main Flavours", you should connect each node i to its nearest neighbours using a Python function `add_edges()`. This function has two input parameters: (i) a `networkx` graph `graph_FMI` that stores an empirical graph whose node attributes will be used to determine its edges and (ii) a parameter `node_degree` that determines the number of nearest neighbours connected to each node.

The nearest neighbours are determined based on the discrepancy measure $d^{(i,i')} := \big\|\mathbf{z}^{(i)} - \mathbf{z}^{(i')}\big\|_2$ between two nodes $i, i' \in \mathcal{V}$. This measure requires, for each node $i \in \mathcal{V}$, some representation vector $\mathbf{z}^{(i)}$ which is stored as a node attribute in the input parameter `graph_FMI`. Your tasks include to try out

the following constructions for the representation vectors:

- **I:** $z^{(i)} = (1/m_i) \sum_{r=1}^{(m_i)} y^{(i,r)}$, (average temperature at FMI station i)
- **II:** the vector $\mathbf{z}^{(i)}$ consisting of the parameters of a GMM that is fit to the feature vectors $\mathbf{x}^{(i,1)}, \ldots, \mathbf{x}^{(i,m_i)}$ in the local dataset $\mathcal{D}^{(i)}$.
- **III:**the vector $\mathbf{z}^{(i)} := \nabla L_i \left(\widehat{\mathbf{w}} \right)$ being the gradient of the average squared error loss $L_i \left(\mathbf{w} \right) = \frac{1}{m_i} \sum_{r=1}^{m_i} \left(y^{(i,r)} - \left(\mathbf{x}^{(i,r)} \right)^T \mathbf{w} \right)^2$. Carefully note that the function $L_i \left(\mathbf{w} \right)$ is determined by the local dataset $\mathcal{D}^{(i)}$. The gradient $\nabla L_i \left(\widehat{\mathbf{w}} \right)$ is evaluated for the model parameters $\widehat{\mathbf{w}}$ (of a linear model) obtained from minimizing the average squared error loss, over all local datasets,

$$\widehat{\mathbf{w}} = \operatorname*{argmin}_{\mathbf{w} \in \mathbb{R}^7} \frac{1}{\sum_{i=1}^{n} m_i} \sum_{i=1}^{n} \sum_{r=1}^{m_i} \left(y^{(i,r)} - \left(\mathbf{x}^{(i,r)} \right)^T \mathbf{w} \right)^2 .$$

Note that for each of the above discrepancy measures, we obtain (via connecting nearest neighbours) a potentially different empirical graph $\mathcal{G}^{(\mathrm{I})}, \mathcal{G}^{(\mathrm{II})}, \mathcal{G}^{(\mathrm{III})}$. Carefully note that these empirical graphs also depend on the choice for the input parameter `node_degree`.

Using each of the above constructions for the empirical graph, learn local model parameters for a linear model by applying FedGD Algorithm 5.1. To this end, you have to split each local dataset into a training set and a validation set (Algorithm 5.1 is applied to the training sets). Diagnose the resulting model parameters by computing the average (over all nodes) training error and validation error.

8 Lecture - "Trustworthy FL"

The Story So Far. We have introduced GTVMin as a main design principle for FL in Lecture 3. Lecture 5 applied the gradient-based methods from Lecture 4 to solve GTVMin, resulting in practical FL systems. Our focus has been on computational and statistical properties of these FL systems. In this and the following lectures, we shift the focus from technical properties to the trustworthiness of FL systems.

Section 8.2 reviews key requirements for trustworthy AI, which includes FL systems, that have been put forward by the European Union [70, 71]. We will also discuss how these requirements guide the design choices for GTVMin-based methods. Our focus will be on the three design criteria for trustworthy FL: privacy, robustness and explainability. This lecture covers robustness and explainability, the leakage and protection of privacy in FL systems is the subject of Lecture 9.

Section 8.3 discusses the robustness of FL systems against perturbations of local datasets and computations. A special type of perturbation is the intentional modification or poisoning of local datasets (see Lecture 10). Section 8.4 introduces a measure for the (subjective) explainability of the personalized models trained by FL systems.

8.1 Learning Goals

After this lecture, you should

- know some key requirements for trustworthy AI
- be familiar with quantitative measures of robustness and explainability

- have some intuition about how robustness, privacy and transparency guides design choices for local models, loss functions and empirical graph in GTVMin

8.2 Seven Key Requirements by the EU

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 [70, 71]. We next discuss in step-by-step fashion how these requirements might guide the design choices for GTVMin-based FL systems.

8.2.1 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...*" [71, p.15]

Human Dignity. Learning personalized model parameters for recommender systems allows to boost addiction or in widespread emotional manipulation resulting in genocide [72–74]. KR1 might rule out certain design choices for the labels of data points. In particular, we might not use the mental and psychological characteristics of a user as the label. We might also avoid loss functions that can be used to train predictors of psychological characteristics. Using personalized ML models to predict user preferences for products or susceptibility towards propaganda is also referred to as "micro-targeting" [75].

Simple is Good. Human oversight can be facilitated by relying on simple local models. Examples include linear models with few features or decision trees with small depth. However, we are unaware of a widely accepted definition of when a model is simple. Loosely speaking, a simple model results in a learn hypothesis that allows humans to understand how features of a data point relate to the prediction $h(\mathbf{x})$. This notion of simplicity is closely related to the concept of explainability which we discuss in more detail in Section 8.4.

Continuous Monitoring. In its simplest form, GTVMin-based methods involve a single training phase, i.e., learning local model parameters by solving GTVMin. However, this approach is only useful if the data can be well approximated by an i.i.d. assumption. In particular, this approach works only if the statistical properties of local datasets do not change over time. For many FL applications, this assumption is unrealistic (consider a social network which is exposed to constant change of memberships and user behaviour). It is then important to continuously compute a validation error on a timely validation set which is then used, in turn, to diagnose the overall FL system (see [4, Sec. 6.6]).

8.2.2 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. ...*' [71, p.16].

Practical FL systems are obtained by implementing FL algorithms in

physical computers (e.g., a wireless network of devices with limited computational capability). These computers typically incur imperfections, such as a temporary lack of connectivity or a device shutting down due to running out of battery. Moreover, also the data generation processes might be subject to perturbations such as statistical anomalies (outliers) or intentional modifications (see Lecture 10). Section 8.3 studies in some detail the robustness of GTVMin-based systems against different perturbations of data sources and imperfections of computational infrastructure.

8.2.3 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...*" [71, p.17].

So far, our discussion of FL system used a high level of abstraction using mathematical concepts such as GTVMin and its building blocks. However, to obtain actual FL systems we need implement these mathematical concepts in a given physical hardware. These implementations might incur deviations from the (idealized) GTVMin formulation (3.15) and the gradient-based methods (such as Algorithm 5.1) used to solve it. For example, we might have access only to quantized label values resulting in a quantization error. Moreover, the local datasets might deviate significantly from a typical realizations of i.i.d. RVs (such a situation is referred to as statistical bias [76, Sec. 3.3.])

Also, we might not be able to access all features of a data point due to data processing regulations [77–79]. For example, the GDPR includes a data minimization principle which requires to only access those features of data

points (being users) that are relevant for predicting the label.

Data Governance. FL systems might use local datasets that are generated by human users, i.e., personal data. Whenever personal data is used, special care must be dedicated towards data protection regulations [79]. It might then be useful (or even compulsory) to designate a data protection officer and conduct a data protection impact assessment [71].

Privacy. The operation of a FL system must not violate the fundamental human right for privacy [80]. Lecture 9 discusses quantitive measures and methods for privacy protection in GTVMin-based FL systems.

8.2.4 KR4 - Transparency.

Traceability. This key requirement includes the documentation of design choices (and underlying business models) for a GTVMin-based FL system. This includes the source for the local datasets, the local models, the local loss function as well as the construction of the empirical graph. Moreover, the documentation should also cover the details of the implemented optimization method used to solve GTVMin. This documentation might also require to periodically store the current model parameters along with a time-stamp ("logging").

Communication. Depending on the use case, FL systems need to communicate the capabilities and limitations to their end users (e.g., of a digital health app running on a smartphone). For example, we can indicate a measure of uncertainty about the predictions delivered by the trained local models. Such an uncertainty measure can be obtained naturally from probabilistic model for the data, e.g., the conditional variance of the label

y, given the features $\mathbf{x}$ of a random data point. Another example for an uncertainty measure is the validation error of a trained local model.

Explainability. The transparency of a GTVMin-based FL system also includes the explainability of the trained local models. Section 8.4 discusses quantitative measures for the subjective explainability of a learnt hypothesis. We will also use this measure as a regularizer to obtain GTVMin-based systems that guarantee subjective explainability "by design".

8.2.5 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.*" [71, p.18].

The local datasets used for the training of local models should be carefully selected to not enforce existing discrimination. In a health-care application, there might be significantly more training data for patients of a specific gender, resulting in models that perform best for that specific gender at the cost of worse performance for the minority [76, Sec. 3.3.]. Fairness is also important for ML methods used to determine credit score and, in turn, if a loan should be granted or not [81]. Here, we must ensure that ML methods do not discriminate customers based on ethnicity or race. To this end, we could augment data points via modifying any features that mainly reflect the ethnicity or race of a customer (see Figure 8.1).

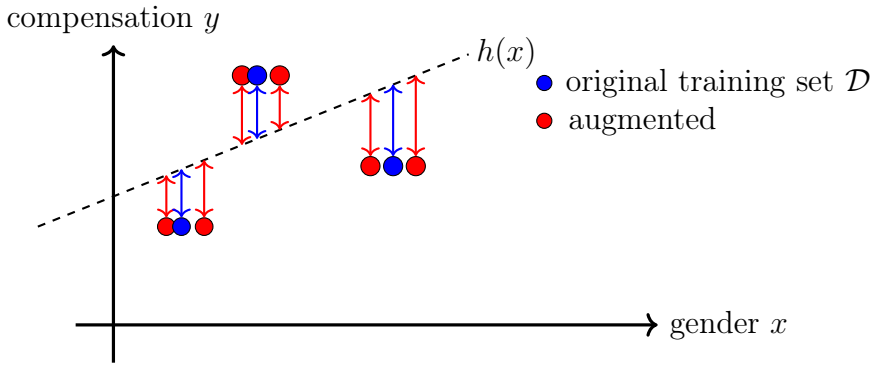


Figure 8.1: We can improve fairness of a ML method by augmenting the training set using perturbations of an irrelevant feature such as the gender of a person for which we want to predict the adequate compensation as the label.

8.2.6 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.*" [71, p.19].

Society. FL systems might be used to deliver personalized recommendations to users within a social media application (social network). These recommendations might be (fake) news used to boost polarization and, in the extreme case, social unrest [82].

Environment. Lecture 5 discussed FL algorithms that were obtained by applying gradient-based methods to solve GTVMin. These methods require computational resources to compute local updates for model parameters and

to share them across the edges of the empirical graph. Computation and communication always requires energy should be generated in an environmental-friendly fashion [83].

8.2.7 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.*" [71, p. 19].

8.3 Technical Robustness of FL Systems

To ensure **KR2** we need to understand the effect of perturbations on a GTVMin-based FL system. These perturbations might be intentional or non-intentional and affect the local datasets used to evaluate the loss of local model parameters or the computational infrastructure used to implement a GTVMin-based method (see Lecture 5). We next explain how to use some of the theoretic tools from previous lectures to quantify the robustness of GTVMin-based FL systems.

Consider a GTVMin-based FL system that aims at training a single (global) linear model in a distributed fashion from a collection of local datasets $\mathcal{D}^{(i)}$, for $i = 1, \ldots, n$. As discussed Section 6.2, this single-model FL setting amounts to using GTVMin (3.17) over a connected empirical graph with a sufficiently large choice for α.

8.3.1 Sensitivity Analysis

As pointed out in Lecture 3, GTVMin (3.17) can be rewritten as the minimization of a quadratic function,

$$\min_{\mathbf{w}=\text{stack}\{\mathbf{w}^{(i)}\}} \mathbf{w}^T\mathbf{Q}\mathbf{w} + \mathbf{q}^T\mathbf{w}. \tag{8.1}$$

The matrix $\mathbf{Q}$ and vector $\mathbf{q}$ are determined by the feature matrices $\mathbf{X}^{(i)}$ and label vector $\mathbf{y}^{(i)}$ at nodes $i \in \mathcal{V}$ (see (2.27)). We next study the sensitivity of (the solutions of) (8.1) towards external perturbations of the label vector.[15]

Consider an additive perturbation $\widetilde{\mathbf{y}}^{(i)} := \mathbf{y}^{(i)} + \boldsymbol{\varepsilon}^{(i)}$ of the label vector $\mathbf{y}^{(i)}$. Using the perturbed label vector $\widetilde{\mathbf{y}}^{(i)}$ results also in a "perturbation" of GTVMin (8.1),

$$\min_{\mathbf{w}=\text{stack}\{\mathbf{w}^{(i)}\}} \mathbf{w}^T\mathbf{Q}\mathbf{w} + \mathbf{q}^T\mathbf{w} + \mathbf{n}^T\mathbf{w} + c. \tag{8.2}$$

An inspection of (2.27) yields that $\mathbf{n} = \Big(\big(\boldsymbol{\varepsilon}^{(1)}\big)^T\mathbf{X}^{(1)}, \ldots, \big(\boldsymbol{\varepsilon}^{(n)}\big)^T\mathbf{X}^{(n)} \Big)^T$. The next result provides an upper bound on the deviation between the solutions of (8.1) and (8.2).

Proposition 8.1. *Consider the GTVMin instance* (8.1) *for learning local model parameters of a linear model for each node $i \in \mathcal{V}$ of an empirical graph $\mathcal{G}$. We assume that the empirical graph is connected, i.e., $\lambda_2\big(\mathbf{L}^{(\mathcal{G})}\big) > 0$ and the local datasets are such that $\bar{\lambda}_{\min} > 0$ (see* (5.4)*). Then, the deviation between $\widehat{\mathbf{w}}^{(i)}$ and the solution $\widetilde{\mathbf{w}}^{(i)}$ to the perturbed problem* (8.2) *is upper bounded as*

$$\sum_{i=1}^{n} \big\|\widehat{\mathbf{w}}^{(i)} - \widetilde{\mathbf{w}}^{(i)}\big\|_2^2 \leq \frac{\lambda_{\max}(1+\rho^2)^2}{\big[\min\{\lambda_2\big(\mathbf{L}^{(\mathcal{G})}\big)\alpha\rho^2, \bar{\lambda}_{\min}/2\}\big]^2} \sum_{i=1}^{n} \big\|\boldsymbol{\varepsilon}^{(n)}\big\|_2^2. \tag{8.3}$$

[15] Our study can be generalized to also take into account perturbations of the feature matrices $\mathbf{X}^{(i)}$, for $i = 1, \ldots, n$.

Here, we used the shorthand $\rho := \bar{\lambda}_{\min}/(4\lambda_{\max})$ *(see* (5.4)*).*

Proof. Left as an exercise to the reader. □

8.3.2 Estimation Error Analysis

Prop. 8.1 characterizes the sensitivity of GTVMin solutions against "external" perturbations of the local datasets. While this form of robustness is important, it might not suffice for a comprehensive assessment of a FL system. For example, we can trivially achieve perfect robustness (in the sense of minimum sensitivity) by delivering constant model parameters, e.g., $\widehat{\mathbf{w}}^{(i)} = \mathbf{0}$.

Another form of robustness is to ensure a small estimation error of (3.17). To study this form of robustness, we use a variant of the probabilistic model (3.24): We assume that the labels and features of data points of each local dataset $\mathcal{D}^{(i)}$, for $i = 1, \ldots, n$, are related via

$$\mathbf{y}^{(i)} = \mathbf{X}^{(i)}\overline{\mathbf{w}} + \boldsymbol{\varepsilon}^{(i)}. \tag{8.4}$$

In contrast to Lecture 3, we assume that all components of (8.4) are deterministic. In particular, the noise term $\boldsymbol{\varepsilon}^{(i)}$ is a deterministic but unknown quantity. This term accommodates any perturbation that might arise from technical imperfections or intrinsic label noise due to random fluctuations in the labelling process.

In the (unlikely) ideal case of no perturbation, we would have $\boldsymbol{\varepsilon}^{(i)} = \mathbf{0}$. However, in general might only know some upper bound measure for the size of the perturbation, e.g., $\left\|\boldsymbol{\varepsilon}^{(i)}\right\|_2^2$. We next present upper bounds on the estimation error $\widehat{\mathbf{w}}^{(i)} - \overline{\mathbf{w}}$ incurred by the GTVMin solutions $\widehat{\mathbf{w}}^{(i)}$.

This estimation error consists of two components, the first component being $\mathrm{avg}\big\{\widehat{\mathbf{w}}^{(i)}\big\} - \overline{\mathbf{w}}$ for each node $i \in \mathcal{V}$. Note that this error component is constant across the nodes $i \in \mathcal{V}$. The second component of the estimation error is the deviation $\widetilde{\mathbf{w}}^{(i)} := \widehat{\mathbf{w}}^{(i)} - \mathrm{avg}\big\{\widehat{\mathbf{w}}^{(i)}\big\}$ of the learnt local model parameters $\widehat{\mathbf{w}}^{(i)}$ from their average $\mathrm{avg}\big\{\widehat{\mathbf{w}}^{(i)}\big\} = (1/n)\sum_{i=1}^{n} \widehat{\mathbf{w}}^{(i)}$. As discussed in Section 3.3.2, these two components correspond to orthogonal subspaces.

According to Prop. 3.1, the second error component is upper bounded as

$$\sum_{i=1}^{n} \big\|\widetilde{\mathbf{w}}^{(i)}\big\|_2^2 \leq \frac{1}{\lambda_2 \alpha} \sum_{i=1}^{n} (1/m_i) \big\|\boldsymbol{\varepsilon}^{(i)}\big\|_2^2 . \tag{8.5}$$

To bound the first error component $\bar{\mathbf{c}} - \overline{\mathbf{w}}$, using the shorthand $\bar{\mathbf{c}} := \mathrm{avg}\big\{\widehat{\mathbf{w}}^{(i)}\big\}$, we first note that (see (3.17))

$$\bar{\mathbf{c}} = \operatorname*{argmin}_{\mathbf{w} \in \mathbb{R}^d} \sum_{i \in \mathcal{V}} (1/m_i) \big\|\mathbf{y}^{(i)} - \mathbf{X}^{(i)}\big(\mathbf{w} - \widetilde{\mathbf{w}}^{(i)}\big)\big\|_2^2 + \alpha \sum_{\{i,i'\} \in \mathcal{E}} A_{i,i'} \left\|\widetilde{\mathbf{w}}^{(i)} - \widetilde{\mathbf{w}}^{(i')}\right\|_2^2 . \tag{8.6}$$

Using a similar argument as in the proof for Prop. 2.1, we obtain

$$\|\bar{\mathbf{c}} - \overline{\mathbf{w}}\|_2^2 \leq \left\| \sum_{i=1}^{n} (1/m_i) \big(\mathbf{X}^{(i)}\big)^T \big(\boldsymbol{\varepsilon}^{(i)} + \mathbf{X}^{(i)} \widetilde{\mathbf{w}}^{(i)}\big) \right\|_2^2 / (n \bar{\lambda}_{\min})^2 . \tag{8.7}$$

Here, $\bar{\lambda}_{\min}$ is the smallest eigenvalue of $(1/n)\sum_{i=1}^{n} \mathbf{Q}^{(i)}$, i.e., the average of the matrices $\mathbf{Q}^{(i)} = (1/m_i)\big(\mathbf{X}^{(i)}\big)^T \mathbf{X}^{(i)}$ over all nodes $i \in \mathcal{V}$.[16] Note that the bound (8.7) is only valid if $\bar{\lambda}_{\min} > 0$ which, in turn, implies that the solution to (8.6) is unique.

[16] We encountered the quantity $\bar{\lambda}_{\min}$ already during our discussion of gradient-based methods for solving the GTVMin instance (3.17) (see (5.4)).

We can develop (8.7) further using

$$\begin{aligned}
&\left\|\sum_{i=1}^{n}(1/m_i)\big(\mathbf{X}^{(i)}\big)^T\big(\boldsymbol{\varepsilon}^{(i)}+\mathbf{X}^{(i)}\widetilde{\mathbf{w}}^{(i)}\big)\right\|_2 \\
&\overset{(a)}{\leq} \sum_{i=1}^{n}\left\|(1/m_i)\big(\mathbf{X}^{(i)}\big)^T\big(\boldsymbol{\varepsilon}^{(i)}+\mathbf{X}^{(i)}\widetilde{\mathbf{w}}^{(i)}\big)\right\|_2 \\
&\overset{(b)}{\leq} \sqrt{n}\sqrt{\sum_{i=1}^{n}\left\|(1/m_i)\big(\mathbf{X}^{(i)}\big)^T\big(\boldsymbol{\varepsilon}^{(i)}+\mathbf{X}^{(i)}\widetilde{\mathbf{w}}^{(i)}\big)\right\|_2^2} \\
&\overset{(c)}{\leq} \sqrt{n}\sqrt{\sum_{i=1}^{n}2\left\|(1/m_i)\big(\mathbf{X}^{(i)}\big)^T\boldsymbol{\varepsilon}^{(i)}\right\|_2^2+2\left\|(1/m_i)\big(\mathbf{X}^{(i)}\big)^T\mathbf{X}^{(i)}\widetilde{\mathbf{w}}^{(i)}\right\|_2^2} \\
&\overset{(d)}{\leq} \sqrt{n}\sqrt{\sum_{i=1}^{n}(2/m_i)\lambda_{\max}\left\|\boldsymbol{\varepsilon}^{(i)}\right\|_2^2+2\lambda_{\max}^2\left\|\widetilde{\mathbf{w}}^{(i)}\right\|_2^2}. \qquad (8.8)
\end{aligned}$$

Here, step (a) uses the triangle inequality form the norm $\|\cdot\|_2$, step (b) uses the Cauchy-Schwarz inequality, step (c) uses the inequality $\|\mathbf{a}+\mathbf{b}\|_2^2 \leq 2\Big(\|\mathbf{a}\|_2^2+\|\mathbf{b}\|_2^2\Big)$ and step (d) uses the maximum eigenvalue $\lambda_{\max}:=\max_{i\in\mathcal{V}}\lambda_d\big(\mathbf{Q}^{(i)}\big)$ of the matrices $\mathbf{Q}^{(i)}=(1/m_i)\big(\mathbf{X}^{(i)}\big)^T\mathbf{X}^{(i)}$ (see (5.4)).

Inserting (8.8) into (8.7) results in the upper bound

$$\begin{aligned}
\|\bar{\mathbf{c}}-\overline{\mathbf{w}}\|_2^2 &\leq 2\sum_{i=1}^{n}\Big[(1/m_i)\lambda_{\max}\left\|\boldsymbol{\varepsilon}^{(i)}\right\|_2^2+\lambda_{\max}^2\left\|\widetilde{\mathbf{w}}^{(i)}\right\|_2^2\Big]/(n\bar{\lambda}_{\min}^2) \\
&\overset{(8.5)}{\leq} 2\big(\lambda_{\max}+(\lambda_{\max}^2/(\lambda_2\alpha))\big)\sum_{i=1}^{n}(1/m_i)\left\|\boldsymbol{\varepsilon}^{(i)}\right\|_2^2/(n\bar{\lambda}_{\min}^2). \qquad (8.9)
\end{aligned}$$

The upper bound (8.9) on the estimation error of GTVMin-based methods depends on both, the empirical graph $\mathcal{G}$ via the eigenvalue λ_2 of $\mathbf{L}^{(\mathcal{G})}$, and the feature matrices $\mathbf{X}^{(i)}$ of the local datasets (via the quantities $\lambda_{\max}$ and $\bar{\lambda}_{\min}$ as defined in (5.4)). Let us next discuss how the upper bound (8.9) might

guide the choice for the empirical graph $\mathcal{G}$ and the features of data points in the local datasets.

According to (8.9), we should use an empirical graph $\mathcal{G}$ with large $\lambda_2\big(\mathbf{L}^{(\mathcal{G})}\big)$ to ensure a small estimation error for GTVMin-based methods. Note that we came across the same design criterion already when discussing graph learning methods in Lecture 7. In particular, using an empirical graph with large $\lambda_2\big(\mathbf{L}^{(\mathcal{G})}\big)$ tends also to speed up the convergence of gradient-based methods for solving GTVMin (such as Algorithm 5.1).

The upper bound (8.9) suggests to use features that result in a small ratio $\lambda_{\max}/\bar{\lambda}_{\min}$ between the quantities $\lambda_{\max}$ and $\bar{\lambda}_{\min}$ (see (5.4)). We might use feature learning methods to optimize this ratio.

8.3.3 Network Resilience

The previous sections studied the robustness of GTVMin-based methods against perturbations of local datasets (see Prop. 8.1) and in terms of ensuring a small estimation error (see (8.9)). We also need to ensure that FL systems are robust against imperfections of the computational infrastructure used to solve GTVMin. These imperfections include hardware failures, running out of battery or lack of wireless connectivity.

Lecture 5 showed how to design FL algorithms by applying gradient-based methods to solve GTVMin (3.17). We obtain practical FL systems by implementing these algorithms, such as Algorithm 5.1, in a particular computational infrastructure. Two important examples for such an infrastructure are mobile networks and wireless sensor networks.

The effect of imperfections in the implementation of the GD based Algo-

rithm 5.1 can be modelled as perturbed GD (4.13) from Lecture 4. We can then analyze the robustness of the resulting FL system via the convergence analysis of perturbed GD (see Section 4.5).

According to (4.14), the performance of the decentralized Algorithm 5.1 degrades gracefully in the presence of imperfections such as missing or faulty communication links. In contrast, the client-server based implementation of FedAvg Algorithm 5.3 offers a single point of failure (the server).

8.3.4 Stragglers

Using perturbed GD to model imperfections in the actual implementation of Algorithm 5.1 is quite flexible. The flexibility in allowing for a wide range of imperfections might come at the cost of a coarse-grained analysis. In particular, the upper bound (4.14) might be too loose (pessimistic) to be of any use. We might then need to take into account the specific nature of the imperfection and the resulting perturbation $\boldsymbol{\varepsilon}^{(i)}$. Let us next focus on a particular type of imperfection that arises from the asynchronous implementation of Algorithm 5.1 at different nodes $i \in \mathcal{V}$.

Note that Algorithm 5.1 requires the nodes to operate synchronous: at each iteration the nodes need to exchange their current model parameters $\mathbf{w}^{(i,k)}$ simultaneously with their neighbours in the empirical graph. This requires, in turn, that the update in step 4 of Algorithm 5.1 is completed at every node i before the global clock ticks (triggering the next iteration).

Some of the nodes i might have limited computational resources and therefore require much longer for the update step 4 of Algorithm 5.1. The literature refers to such slower nodes sometimes as stragglers [84]. Instead of

forcing the other (faster) nodes to wait until also the slower ones are ready, we could instead let them continue with their local updates. This results in an asynchronous variant of Algorithm 5.1 which we summarize in Algorithm 8.1.

Note that Algorithm 8.1 still uses a global iteration counter k. However, the role of this counter is fundamentally different to its role in the synchronous Algorithm 5.1. In particular, we merely need the counter k for notational convenience to denote the (arbitrary) time-instants during which the asynchronous local updates (8.10). These updates take place only for a subset of ("active") nodes $i \in \mathcal{A}^{(k)}$. All other ("inactive") nodes $i \notin \mathcal{A}^{(k)}$ leave their current model parameters unchanged, $\mathbf{w}^{(i,k+1)} := \mathbf{w}^{(i,k)}$.

The asynchronous local update (8.10) at some node $i \in \mathcal{A}^{(i)}$ uses the "outdated" model parameters $\mathbf{w}^{(i',k_{i,i'})}$ at its neighbours $i' \in \mathcal{N}^{(i)}$. In particular, some of these neighbours might have not been in the active sets $\mathcal{A}^{(k-1)}, \mathcal{A}^{(k-2)}, \ldots$ during the most recent iterations. In this case, we cannot use $\mathbf{w}^{(i',k)}$ as it is simply not available to node i at "time" k. Rather, we might need to use $\mathbf{w}^{(i',k_{i,i'})}$ obtained during some previous "time" $k_{i,i'}$.

We can interpret the iteration index $k_{i,i'}$ as the most recent time instant during which node i' has updated its local model parameters and shared it with node i. The difference $k - k_{i,i'}$, in turn, can be viewed as a measure for the communication delay from node i' to node i. The robustness of (the convergence of) Algorithm 8.1 against these communication delays is studied in-depth in [48, Ch. 6 and 7].

Algorithm 8.1 Asynchronous FedGD for Local Linear Models

Input: empirical graph $\mathcal{G}$; GTV parameter α; learning rate η

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 i; some stopping criterion.

Output: linear model parameters $\widehat{\mathbf{w}}^{(i)}$ for each node $i \in \mathcal{V}$

Initialize: $k{:=}0$; $\mathbf{w}^{(i,0)}{:=}\mathbf{0}$ for all nodes $i \in \mathcal{V}$.

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

2: **for** all active nodes $i \in \mathcal{A}^{(k)}$: **do**

3: update local model parameters via

$$\mathbf{w}^{(i,k+1)} := \mathbf{w}^{(i,k)} + \eta \Bigg[(2/m_i) \big(\mathbf{X}^{(i)}\big)^T \big(\mathbf{y}^{(i)} - \mathbf{X}^{(i)} \mathbf{w}^{(i,k)}\big) + 2\alpha \sum_{i' \in \mathcal{N}^{(i)}} A_{i,i'} \big(\mathbf{w}^{(i',k_{i,i'})} - \mathbf{w}^{(i,k)}\big) \Bigg]. \tag{8.10}$$

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

5: **end for**

6: **for** all inactive nodes $i \notin \mathcal{A}^{(k)}$: **do**

7: keep model parameters unchanged $\mathbf{w}^{(i,k+1)} := \mathbf{w}^{(i,k)}$

8: **end for**

9: increment iteration counter: $k{:=}k{+}1$

10: **end while**

11: $\widehat{\mathbf{w}}^{(i)} := \mathbf{w}^{(i,k)}$ for all nodes $i \in \mathcal{V}$

8.4 Subjective Explainability of FL Systems

Let us now discuss how to ensure key requirement **KR4 - Transparency** in GTVMin-based FL systems. This key requirement includes also the explainability of a trained personalized model $\hat{h}^{(i)} \in \mathcal{H}^{(i)}$ (and their predictions). It is important to note that the explainability of $\hat{h}^{(i)}$ is subjective: A given learnt hypothesis $\hat{h}^{(i)}$ might offer high degree of explainability to one user (a graduate student at a university)but a low degree of explainability to another user (a high-school student). We must ensure explainability for the specific user, which we will also denote by i, that "consumes" the predictions at node $i \in \mathcal{V}$ of the empirical graph.

One particular useful approach to the explainability of trained ML models is based on its simulatability [85–87]: How well can a user anticipate (or guess) the prediction $\hat{y} = \hat{h}^{(i)}(\mathbf{x})$ delivered by $\hat{h}^{(i)}$ for a data point with features $\mathbf{x}$. We can then measure the explainability of $\hat{h}^{(i)}(\mathbf{x})$ to the user at node i by comparing the prediction $\hat{h}^{(i)}(\mathbf{x})$ with the corresponding "guess" (or "simulation") $u^{(i)}(\mathbf{x})$.

We can enforce (subjective) explainability of FL systems by modifying the local loss functions in GTVMin. For ease of exposition we will focus on the GTVMin instance (5.1) for training local (personalized) linear models. For each node $i \in \mathcal{V}$, we construct a test-set $\mathcal{D}_t^{(i)}$ and ask user i to deliver a guess $u^{(i)}(\mathbf{x})$ for each data point in $\mathcal{D}_t^{(i)}$.[17]

We measure the (subjective) explainability of a linear hypothesis with

[17] We only use the features of the data points in $\mathcal{D}_t^{(i)}$, i.e., this dataset can be constructed from unlabeled data.

model parameters $\mathbf{w}^{(i)}$ by

$$(1/|\mathcal{D}_t^{(i)}|) \sum_{\mathbf{x} \in \mathcal{D}_t^{(i)}} \left(u^{(i)}(\mathbf{x}) - \mathbf{x}^T \mathbf{w}^{(i)} \right)^2. \tag{8.11}$$

It seems natural to add this measure as a penalty term to the local loss function in (5.1), resulting in the new loss function

$$L_i\left(\mathbf{w}^{(i)}\right) := \underbrace{(1/m_i) \left\| \mathbf{y}^{(i)} - \mathbf{X}^{(i)} \mathbf{w}^{(i)} \right\|_2^2}_{\text{training error}} + \rho \underbrace{(1/|\mathcal{D}_t^{(i)}|) \sum_{\mathbf{x} \in \mathcal{D}_t^{(i)}} \left(u^{(i)}(\mathbf{x}) - \mathbf{x}^T \mathbf{w}^{(i)} \right)^2}_{\text{subjective explainability}}. \tag{8.12}$$

The regularization parameter ρ controls the preference for a high subjective explainability of the hypothesis $h^{(i)}(\mathbf{x}) = \left(\mathbf{w}^{(i)}\right)^T \mathbf{x}$ over a small training error [87]. It can be shown that (8.12) is the average weighted squared error loss of $h^{(i)}(\mathbf{x})$ on an augmented version of $\mathcal{D}^{(i)}$. This augmented version includes the data point $\left(\mathbf{x}, u^{(i)}(\mathbf{x})\right)$ for each data point $\mathbf{x}$ in the test-set $\mathcal{D}_t^{(i)}$.

So far, we have focused on the problem of explaining a trained personalized model to some user. The general idea is to provide partial information, in the form of some explanation, about the learnt hypothesis map $\hat{h}$. Explanations should help the user to anticipate the prediction $\hat{h}(\mathbf{x})$ for any given data point. Instead of explaining a trained model, i.e., a learnt hypothesis $\hat{h}$, we might also want to explain an entire FL algorithm.

Mathematically, we can interpret an FL algorithm as a map $\mathcal{A}$ that reads in local datasets and delivers learnt hypothesis maps $\hat{h}^{(i)}$. Explaining an FL algorithm amounts to providing partial information about this map $\mathcal{A}$. Thus, mathematically speaking, the problem of explaining a learnt hypothesis is essentially the same as explaining an entire FL algorithm: Provide partial

information (explanation) about a map such that user can anticipate the results of applying the map to arbitrary arguments. However, the map $\mathcal{A}$ might be much more complicated compared to a learnt hypothesis (which could be a linear map for linear models). The different level of complexity of these two families of maps requires to use different forms of explanation. For example, we might explain a FL algorithm using a pseudo-code such as Algorithm 5.1. Another form of explanation could be a Python code snippet that illustrates a potential implementation of the algorithm.

```python
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

# Load the Iris dataset
data = load_iris()
X = data.data
y = data.target

# Split the dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size=0.3, random_state=42)

# Create a Decision Tree classifier
clf = DecisionTreeClassifier(random_state=42)

# Train the classifier
clf.fit(X_train, y_train)

# Make predictions on the test data
y_pred = clf.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
accuracy
```

Figure 8.2: Python code for fitting a decision tree model on the Iris dataset.

8.5 Overview of Coding Assignment

Python Notebook. `TrustworthyFL_CodingAssignment.ipynb`

Data File. `Assignment_MLBasicsData.csv`

This coding assignment continues from the coding assignment "FL Algorithms" (see Section 5.6). This previous assignment required you to implement Algorithm 5.1 and apply it to an empirical graph whose nodes are FMI weather stations. We now study the effect of adding perturbations $\boldsymbol{\varepsilon}^{(i)}$ to the label vector $\mathbf{y}^{(i)}$, consisting of temperature measurements taken at the FMI station i. In particular, we replace the label vector $\mathbf{y}^{(i)}$ with the perturbed label vector $\mathbf{y}^{(i)} + \boldsymbol{\varepsilon}^{(i)}$. How much do the local model parameters delivered by Algorithm 5.1 change when adding perturbations of different strength $\left\|\boldsymbol{\varepsilon}^{(i)}\right\|_2$.

9 Lecture - "Privacy-Protection in FL"

The core idea of FL is to share information contained in local datasets in order to improve the training of ML models. Lecture 5 discussed FL algorithms that share information in the form of model parameters that are computed from the local loss function. Each node $i \in \mathcal{V}$ receives current model parameters from other nodes and, after executing a local update, shares its new model parameters with other nodes.

Depending on the design choices for GTVMin-based methods, sharing model parameters allows to reconstruct local loss functions and, in turn, to estimate private information about individual data points such as health-care customers ("patients") [88]. Thus, the bad news is that FL systems will almost inevitably incur some leakage of private information.The good news is, however, that the extent of privacy leakage can be controlled by (i) careful design choices for GTVMin and (ii) applying slight modifications of basic FL algorithms (such as those from Lecture 5).

This lecture revolves around two main questions:

- **Q1.** How can we measure privacy leakage of a FL system?
- **Q2.** How can we control (minimize) privacy leakage of a FL system?

Section 9.2 addresses Q1 while Sections 9.3 and 9.4 address Q2.

9.1 Learning Goals

After this lecture, you should

- be aware of threats to privacy and the need to protect it,

- know some quantitative measures for privacy leakage,
- understand the effect of GTVMin design choices on user privacy,
- be able to implement FL algorithms with privacy guarantees.

9.2 Measuring Privacy Leakage

Consider a FL system that trains a personalized model for the users, indexed by $i = 1, \ldots, n$, of heart rate sensors. Each user i generates a local dataset $\mathcal{D}^{(i)}$ that consists of time-stamped heart rate measurements. We define a single data point as a single continuous activity, e.g. as a 50-minute long run. The features of such a data point (activity) might include the trajectory in the form of a time-series of GPS coordinates (e.g., measured every 30 seconds). The label of a data point (activity) could be the average heart rate during the activity. Let us assume that this average heart rate is private information that should not be shared with anybody.[18]

Our FL system also exploits the information provided by a fitness expert that determines pair-wise similarities $A_{i,i'}$ between users i, i' (e.g., due to body weight and height). We then use Algorithm 5.1 to learn, for each user i, the model parameters $\mathbf{w}^{(i)}$ for some AI health-care assistant [89]. In what follows, we interpret Algorithm 5.1 as a map $\mathcal{A}(\cdot)$ (see Figure 9.1). The map $\mathcal{A}$ reads in the dataset $\mathcal{D} := \big\{\mathcal{D}^{(i)}\big\}_{i=1}^{n}$ (constituted by the local datasets $\mathcal{D}^{(i)}$ for $i = 1, \ldots, n$) and delivers the local model parameters $\mathcal{A}\big(\mathcal{D}\big) := \underbrace{\text{stack}\big\{\widehat{\mathbf{w}}^{(i)}\big\}_{i=1}^{n}}_{\widehat{\mathbf{w}}}$.

[18] In particular, we might not want to share our heart rate profiles with a potential future employer who prefers candidates with a long life expectation.

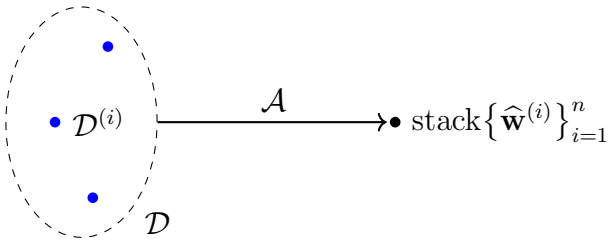


Figure 9.1: Algorithm 5.1 maps the collection $\mathcal{D}$ of local datasets $\mathcal{D}^{(i)}$ to the learnt model parameters $\widehat{\mathbf{w}}^{(i)}$, for each node $i = 1, \ldots, n$. These learnt model parameters are (approximate) solutions to the GTVMin instance (3.17).

A privacy-preserving FL system should not allow to infer, solely from the learned model parameters, the average heart rate $y^{(i,r)}$ during a specific single activity r of a specific user i. Mathematically, we must ensure that the map $\mathcal{A}$ is not invertible: The learnt model parameters should not change if we would apply the FL algorithm to a perturbed dataset that includes a different value for the average heart rate $y^{(i,r)}$. Figure 9.2 illustrates an algorithm that is partially invertible in the sense of allowing to infer the label of some data points used in the training set.

The sole requirement for a FL algorithm $\mathcal{A}$ to be not invertible is not useful in general. Indeed, we can easily make any algorithm $\mathcal{A}$ by simple pre- or post-processing techniques whose effect is limited to irrelevant regions of the input space (which is the space of all possible datasets). The level of privacy-protection offered by $\mathcal{A}$ can be characterized by a measure of its "non-invertibility".

A simple measure of non-invertibility is the sensitivity of the output $\mathcal{A}\big(\mathcal{D}\big)$

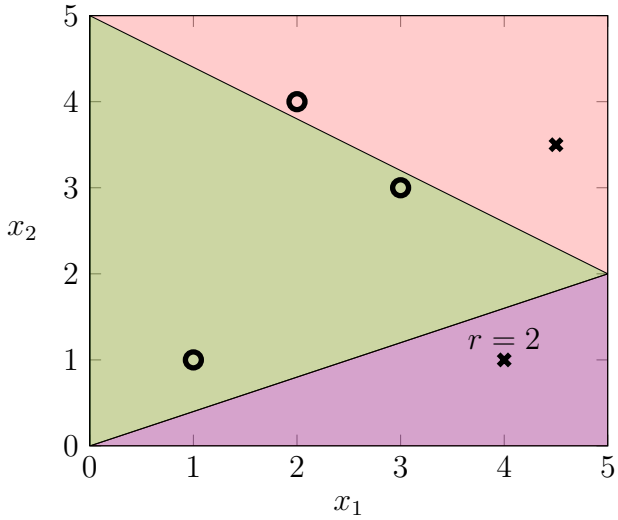


Figure 9.2: Scatterplot of data points $r = 1, 2, \ldots$, each characterized by features $\mathbf{x}^{(r)} = \left(x_1^{(r)}, x_2^{(r)}\right)^2$ and a binary label $y^{(r)} \in \{\circ, \times\}$. The plot also indicates the decision regions of a hypothesis $\widehat{h}$ that has been learned via ERM. Would you be able to infer the label of data point $r = 2$ if you knew the decision regions?

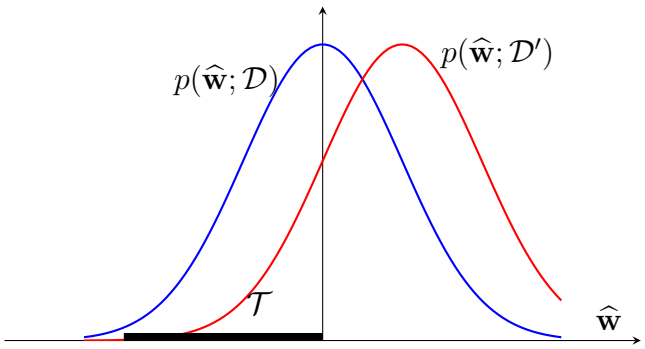


Figure 9.3: Two probability distributions of the learnt model parameters $\widehat{\mathbf{w}} = \text{stack}\big\{\widehat{\mathbf{w}}^{(i)}\big\}_{i=1}^{n}$ delivered by some FL algorithm (such as Algorithm 5.1). These two probability distributions correspond to two different choices for the input dataset, denoted by $\mathcal{D}'$ and $\mathcal{D}$. For example $\mathcal{D}'$ might be obtained from $\mathcal{D}$ by changing the value of a private feature of some data point in $\mathcal{D}$. We also indicate an "acceptance region" $\mathcal{T}$ that is used to detect if $\mathcal{D}$ (or a neighbouring dataset $\mathcal{D}'$) has been fed into the algorithm.

against varying the heart rate value $y^{(i,r)}$,

$$\frac{\left\|\mathcal{A}(\mathcal{D}) - \mathcal{A}(\mathcal{D}')\right\|_2}{\varepsilon}. \tag{9.1}$$

Here, $\mathcal{D}$ denotes some given collection of local datasets and $\mathcal{D}'$ is a modified dataset. In particular, $\mathcal{D}'$ is obtained by replacing the actual average heart rate $y^{(i,r)}$ with the modified value $y^{(i,r)} + \varepsilon$. The privacy-protection of $\mathcal{A}$ is higher for smaller values (9.1), i.e., the output changes only little when varying the value of the average heart rate.

Another measure for the non-invertibility of $\mathcal{A}$ is referred to as differential privacy (DP). This measure is particularly useful for stochastic algorithms that use some random mechanism. One example for such a random mechanism is the selection of a random subset of data points (a batch) within Algorithm 5.1. Section 9.3 discusses another example of a random mechanism: add the realization of a RV to the (intermediate) results of an algorithm.

A stochastic algorithm $\mathcal{A}$ can be described by a probability distribution $p(\widehat{\mathbf{w}}; \mathcal{D})$ over a measurable space that is constituted by the possible values of the learnt model parameters $\widehat{\mathbf{w}}$ (see Figure 9.3).[19] This probability distribution is parametrized by the dataset $\mathcal{D}$ that is fed as input to the algorithm $\mathcal{A}$.

DP measures the non-invertibility of a stochastic algorithm $\mathcal{A}$ via the similarity of the probability distributions obtained for two datasets $\mathcal{D}, \mathcal{D}'$ that are considered neighbouring or adjacent [76, 93]. Typically, we consider $\mathcal{D}'$ as adjacent to $\mathcal{D}$ if it is obtained by modifying the features or label of a single data point in $\mathcal{D}$. As a case in point, consider data points representing physical activities which are characterized by a binary feature $x_j \in \{0, 1\}$

[19] For more details about the concept of a measurable space, we refer to the literature on probability and measure theory [90–92].

that indicates an excessively high average heart rate during the activity. We could then define neighbouring datasets via flipping the feature x_j of a single data point. In general, the notion of neighbouring datasets is a design choice used in the formal definitions of DP.

Definition 1. *(from [93]) A stochastic algorithm $\mathcal{A}$ is (ε, δ)-DP if for any measurable set $\mathcal{S}$ and any two neighbouring datasets $\mathcal{D}, \mathcal{D}'$,*

$$\text{Prob}\big\{\mathcal{A}(\mathcal{D}) \in \mathcal{S}\big\} \leq \exp(\varepsilon)\text{Prob}\big\{\mathcal{A}(\mathcal{D}') \in \mathcal{S}\big\} + \delta. \tag{9.2}$$

It appears that Definition 1 is the current de-facto standard for measuring the (lack of) privacy-protection in FL systems [76, 93]. Nevertheless, there are also other measures for the similarity between probability distributions $p(\widehat{\mathbf{w}}; \mathcal{D})$ and $p(\widehat{\mathbf{w}}; \mathcal{D}')$ that might be more useful for practical or theoretical reasons [94]. One such alternative measure is the Rényi divergence of order $\alpha > 1$,

$$D_\alpha\Big(p(\widehat{\mathbf{w}}; \mathcal{D})\big\|p(\widehat{\mathbf{w}}; \mathcal{D}')\Big) := \frac{1}{\alpha - 1}\mathbb{E}_{p(\widehat{\mathbf{w}}; \mathcal{D}')}\left[\left(\frac{dp(\widehat{\mathbf{w}}; \mathcal{D})}{dp(\widehat{\mathbf{w}}; \mathcal{D}')}\right)^\alpha\right]. \tag{9.3}$$

The Rényi divergence allows to define the following variant of DP [94, 95].

Definition 2. *(from [93]) A stochastic algorithm $\mathcal{A}$ is (α, γ)-RDP if, for any two neighbouring datasets $\mathcal{D}, \mathcal{D}'$,*

$$D_\alpha\Big(p(\widehat{\mathbf{w}}; \mathcal{D})\big\|p(\widehat{\mathbf{w}}; \mathcal{D}')\Big) \leq \gamma. \tag{9.4}$$

One recent use-case of (α, γ)-RDP is the analysis of the DP guarantees offered by variants of SGD [94]. This analysis also exploited the fact that (α, γ)-RDP implies (ε, δ)-DP for suitable choices of ε, δ [94].

A very important property of the DP notions in Definition 1 and Definition 2 is that they are preserved by post-processing:

Proposition 9.1. *Consider a FL system $\mathcal{A}$ that is applied to some dataset $\mathcal{D}$ and some (possibly stochastic) map $\mathcal{B}$ that does not depend on $\mathcal{D}$. If $\mathcal{A}$ is (ε, δ)-DP (or (α, γ)-RDP), then so is also the composition $\mathcal{B} \circ \mathcal{A}$.*

Proof. See, e.g., [93, Sec. 2.3]. □

By Prop. (9.1), we cannot reduce the (level of) DP of $\mathcal{A}$ by any post-processing method $\mathcal{B}$ that has no access to the raw data itself. It seems almost natural to make this "post-processing immunity" a defining property of any useful notion of DP [95]. However, due to Prop. (9.1), this property is already "built-in" into the Definition 1 (and the Definition 2).

What does it mean in "reality"? The above (mathematically precise) definitions of DP are somewhat abstract. It is instructive to interpret them from the perspective of statistical testing: We could use the output of an algorithm $\mathcal{A}$ to test (or detect) if the underlying dataset fed into $\mathcal{A}$ was $\mathcal{D}$ or if it actually was a neighbouring dataset $\mathcal{D}'$ [96]. Such a statistical test amounts to specifying a region $\mathcal{T}$ and to declare either

- "dataset $\mathcal{D}$ seems to be used", if $\mathcal{A} \in \mathcal{T}$, or
- "dataset $\mathcal{D}'$ seems to be used", if $\mathcal{A} \notin \mathcal{T}$.

The performance of a test $\mathcal{T}$ is characterized by two error probabilities:

- The probability of declaring $\mathcal{D}'$ but actually $\mathcal{D}$ was fed into $\mathcal{A}$, which is $P_{\mathcal{D}\to\mathcal{D}'} := 1 - \int_{\mathcal{T}} p(\widehat{\mathbf{w}}; \mathcal{D})$.
- The probability of declaring $\mathcal{D}$ but actually $\mathcal{D}'$ was fed into $\mathcal{A}$, which is $P_{\mathcal{D}'\to\mathcal{D}} := \int_{\mathcal{T}} p(\widehat{\mathbf{w}}; \mathcal{D}')$.

For a privacy-preserving algorithm $\mathcal{A}$, there should be no test $\mathcal{T}$ for which both $P_{\mathcal{D}\to\mathcal{D}'}$ and $P_{\mathcal{D}'\to\mathcal{D}}$ are small (close to 0). This intuition can be made precise as follows (see, e.g., [97, Thm. 2.1.] or [98]): If an algorithm $\mathcal{A}$ is (ε,δ)-DP, then

$$\exp(\varepsilon)P_{\mathcal{D}\to\mathcal{D}'} + P_{\mathcal{D}'\to\mathcal{D}} \geq 1-\delta. \tag{9.5}$$

Thus, if $\mathcal{A}$ is (ε,δ)-DP with small ε,δ (close to 0), then (9.5) implies $P_{\mathcal{D}\to\mathcal{D}'} + P_{\mathcal{D}'\to\mathcal{D}} \approx 1$.

9.3 Ensuring Differential Privacy

Depending on the underlying design choices (for data, model and optimization method), a GTVMin-based method $\mathcal{A}$ might already ensure DP by design. However, for some design choices the resulting GTVMin-based method $\mathcal{A}$ might not ensure DP. However, according to Prop. 9.1, we might then still be able to ensure DP by applying pre- and/or post-processing techniques to the input (local datasets) and output (learnt model parameters) of $\mathcal{A}$. Formally, this means to compose the map $\mathcal{A}$ with two (possibly stochastic) maps $\mathcal{I}$ and $\mathcal{O}$, resulting in a new algorithm with map $\mathcal{A}' := \mathcal{O}\circ\mathcal{A}\circ\mathcal{I}$. The output of $\mathcal{A}'$ for a given dataset $\mathcal{D}$ is obtained by

- first applying the pre-processing $\mathcal{I}(\mathcal{D})$,
- then the original algorithm $\mathcal{A}\big(\mathcal{I}(\mathcal{D})\big)$,
- and the final post-processing $\mathcal{O}\big(\mathcal{A}\big(\mathcal{I}(\mathcal{D})\big)\big) =: \mathcal{A}'(\mathcal{D})$.

Post-Processing. Maybe the most widely used post-processing for

ensuring DP is to "add some noise" [93]:

$$\mathcal{O}(\mathcal{A}) := \mathcal{A} + \mathbf{n}, \text{ with noise } \mathbf{n} = \big(n_1, \ldots, n_{nd}\big)^T, \; n_1, \ldots, n_{nd} \overset{i.i.d.}{\sim} p(n). \quad (9.6)$$

Note that this post-processing technique is parametrized by the choice for the probability distribution $p(n)$ of the noise entries. Two important choices are the Laplacian distribution $p(n) := \frac{1}{2b} \exp\big(-\frac{|n|}{b}\big)$ and the Gaussian distribution $p(n) := \frac{1}{\sqrt{2\pi\sigma^2}} \exp\big(-\frac{n^2}{2\sigma^2}\big)$. The noise variance σ^2 can be chosen based on the sensitivity

$$\Delta_2\big(\mathcal{A}\big) := \max_{\mathcal{D},\mathcal{D}'} \left\| \mathcal{A}(\mathcal{D}) - \mathcal{A}(\mathcal{D}') \right\|_2 . \quad (9.7)$$

Here, the maximum is over all pairs of neighbouring datasets $\mathcal{D}, \mathcal{D}'$. Adding Gaussian noise with variance $\sigma^2 > \sqrt{2\ln(1.25/\delta)}\Delta_2(\mathcal{A})/\varepsilon$ ensures that $\mathcal{A}$ is (ε, δ)-DP [93, Thm. 3.22]. It might be difficult to evaluate the sensitivity (9.7) for a given FL algorithm $\mathcal{A}$ [99]. For a GTVMin-based method, i.e., $\mathcal{A}(\mathcal{D})$ is a solution to (3.15), we might obtain upper bounds on $\Delta_2\big(\mathcal{A}\big)$ by a perturbation analysis similar in spirit to the proof of Prop. 8.1.

Pre-Processing. Instead of ensuring DP via post-processing the output of a FL algorithm $\mathcal{A}$, we can ensure DP by applying a pre-processing map $\mathcal{I}(\mathcal{D})$ to the dataset $\mathcal{D}$. The result of the pre-processing is a new dataset $\widehat{\mathcal{D}} = \mathcal{I}(\mathcal{D})$ which can be made available (publicly!) to any algorithm $\mathcal{A}$ that has no direct access to $\mathcal{D}$. According to Prop. 9.1, as long as the pre-processing map $\mathcal{I}$ is (ε, δ)-DP (see Definition 1), so will be the composition $\mathcal{A} \circ \mathcal{I}$.

As for post-processing, one important approach to pre-processing is to "add" or "inject" noise. This results in a stochastic pre-processing map $\widehat{\mathcal{D}} = \mathcal{I}(\mathcal{D})$ that is characterized by a probability distribution. The noise mechanisms used for pre-processing might be different from just adding the realization of

a RV (see (9.6)):[20]

- For a classification method with discrete label space $\mathcal{Y} = \{1, \ldots, K\}$, we can inject noise by replacing the true label of a data point with a randomly selected element of $\mathcal{Y}$ [100, Mechanism 1]. The noise injection might also include the replacement of the features of a data point by a realization of a RV whose probability distribution is somehow matched to the dataset $\mathcal{D}$ [100, Mechanism 2].

- Another form of noise injection is to construct $\mathcal{I}(\mathcal{D})$ by randomly selecting data points from the original (private) dataset $\mathcal{D}$ [101]. Note that such a form of noise injection is naturally provided by SGD methods (see, e.g., step 4 of Algorithm 5.2).

How To Be Sure? Consider some algorithm $\mathcal{A}$, possibly obtained by pre- and post-processing techniques, that is claimed to be (ε, δ)-DP. In practice, we might not know the detailed implementation of the algorithm. For example, we might not have access to the noise generation mechanism used in the pre- or post-processing steps. How can we verify a claim about DP of an algorithm $\mathcal{A}$ without having access to the detailed implementation of $\mathcal{A}$? One approach could be to apply the algorithm to synthetic datasets $\mathcal{D}^{(1)}_{\rm syn}, \ldots, \mathcal{D}^{(L)}_{\rm syn}$ that differ only in some private attribute of a single data point. We can then try to predict the private attribute $s^{(r)}$ of the dataset $\mathcal{D}^{(r)}_{\rm syn}$ by applying a learnt hypothesis $\widehat{h}$ to the output $\mathcal{A}\big(\mathcal{D}^{(r)}_{\rm syn}\big)$ delivered by the "algorithm under test" $\mathcal{A}$. The hypothesis $\widehat{h}$ might be learnt by a ERM-based method (see

[20] Can you think of a simple pre-processing map that is deterministic and guarantees maximum DP?

Algorithm 2.1) using a training set consisting of pairs $\left(\mathcal{A}\big(\mathcal{D}_{\rm syn}^{(r)}\big), s^{(r)}\right)$ for some $r \in \{1, \ldots, L\}$.

9.4 Private Feature Learning

Section 9.3 discussed pre-processing techniques that ensure DP of a FL algorithm. We next discuss pre-processing techniques that are not directly motivated from a DP perspective. Instead, we cast privacy-friendly pre-processing of a dataset as a feature learning problem [4, Ch. 9].

Consider a data point characterized by a feature vector $\mathbf{x} \in \mathbb{R}^d$ and a label $y \in \mathbb{R}$. Moreover, each data point is characterized by a private attribute s. We want to learn a (potentially stochastic) feature map $\mathbf{\Phi} : \mathbb{R}^d \to \mathbb{R}^{d'}$ such that the new features $\mathbf{z} = \mathbf{\Phi}(\mathbf{x}) \in \mathbb{R}^{d'}$ do not allow to accurately predict the private attribute s. Trivially, we can make prediction of s from $\mathbf{\Phi}(\mathbf{x})$ impossible by using a constant map, e.g., $\mathbf{\Phi}(\mathbf{x}) = 0$. However, we still want the new features $\mathbf{z} = \mathbf{\Phi}(\mathbf{x})$ to allow an accurate prediction (using a suitable hypothesis) for the label y of a data point.

To quantify the predictability of the private attribute s solely from the transformed features $\mathbf{z} = \phi(\mathbf{x})$ we use an i.i.d. assumption. We can then use the mutual information (MI) $I\left(s; \mathbf{\Phi}(\mathbf{x})\right)$ as a measure for the predicability of s from $\mathbf{\Phi}(\mathbf{x})$. A small value of $I\left(s; \mathbf{\Phi}(\mathbf{x})\right)$ indicates that it is difficult to predict the private attribute s solely from $\mathbf{\Phi}(\mathbf{x})$, i.e., a high level of privacy protection.[21] Similarly, we can use the MI $I\left(y; \mathbf{\Phi}(\mathbf{x})\right)$ to measure the predicability of the label y from $\mathbf{\Phi}(\mathbf{x})$. A large value $I\left(y; \mathbf{\Phi}(\mathbf{x})\right)$ indicates

[21] The relation between MI-based privacy measures and DP has been studied in some detail recently [102].

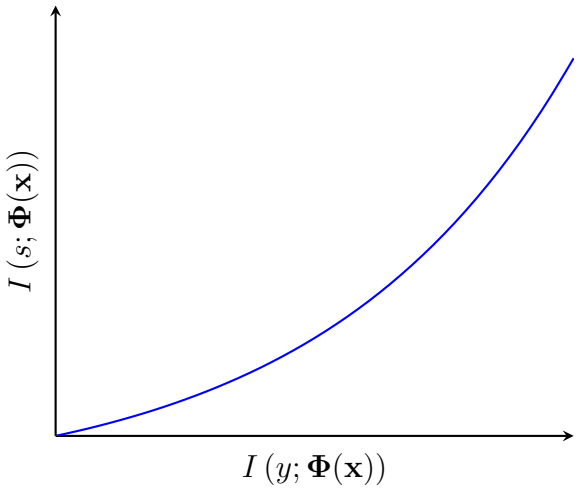


Figure 9.4: The solutions of the privacy funnel (9.8) trace out (for varying constraint R) a curve in the plane spanned by the values of $I\left(s;\mathbf{\Phi}(\mathbf{x})\right)$ (measuring the privacy leakage) and $I\left(y;\mathbf{\Phi}(\mathbf{x})\right)$ (measuring the usefulness of the transformed features for predicting the label).

that $\mathbf{\Phi}(\mathbf{x})$ allows to accurately predict y (which is of course preferable).

It seems natural to use a feature map $\mathbf{\Phi}(\mathbf{x})$ that optimally balances a small $I\left(s;\mathbf{\Phi}(\mathbf{x})\right)$ (privacy protection) with a sufficiently large $I\left(y;\mathbf{\Phi}(\mathbf{x})\right)$ (allowing to accurately predict y). The mathematically precise formulation of this plan is known as the privacy funnel [103, Eq. (2)],

$$\min_{\mathbf{\Phi}(\cdot)} I\left(s;\mathbf{\Phi}(\mathbf{x})\right) \text{ such that } I\left(y;\mathbf{\Phi}(\mathbf{x})\right) \geq R. \tag{9.8}$$

Figure 9.4 illustrates the solution of (9.8) for varying R, i.e., minimum value of $I\left(y;\mathbf{\Phi}(\mathbf{x})\right)$.

The privacy funnel (9.8) uses the MI $I\left(s;\mathbf{\Phi}(\mathbf{x})\right)$ to quantify the privacy leakage of a feature map $\mathbf{\Phi}(\mathbf{x})$. An alternative measure for the privacy leakage is the minimum reconstruction error $s-\hat{s}$. The reconstruction $\hat{s}$ is obtained

by applying a reconstruction map $r(\cdot)$ to the transformed features $\mathbf{\Phi}(\mathbf{x})$. If the joint probability distribution $p(s, \mathbf{x})$ is a multivariate normal and the $\mathbf{\Phi}(\cdot)$ is a linear map (of the form $\mathbf{\Phi}(\mathbf{x}) := \mathbf{F}\mathbf{x}$ with some matrix $\mathbf{F}$), then the optimal reconstruction map is again linear [29].

We would like to find the linear feature map $\mathbf{\Phi}(\mathbf{x}) := \mathbf{F}\mathbf{x}$ such that for any linear reconstruction map $\mathbf{r}$ (resulting in $\hat{s} := \mathbf{r}^T\mathbf{F}\mathbf{x}$) the expected squared error $\mathbb{E}\{(s - \hat{s})^2\}$ is large. The smallest possible expected squared error loss

$$\varepsilon(\mathbf{F}) := \min_{\mathbf{r}\in\mathbb{R}^{d'}} \mathbb{E}\{(s - \mathbf{r}^T\mathbf{F}\mathbf{x})^2\} \tag{9.9}$$

measures the the level of privacy protection offered by the new features $\mathbf{z} = \mathbf{F}\mathbf{x}$. The larger the value $\varepsilon(\mathbf{F})$, the more privacy protection is offered. It can be shown that $\varepsilon(\mathbf{F})$ is maximized by any $\mathbf{F}$ that is orthogonal to the cross-covariance vector $\mathbf{c}_{\mathbf{x},s} := \mathbb{E}\{\mathbf{x}s\}$, i.e., whenever $\mathbf{F}\mathbf{c}_{\mathbf{x},s} = \mathbf{0}$. One specific choice for $\mathbf{F}$ that satisfies this orthogonality condition is

$$\mathbf{F} = \mathbf{I} - (1/\left\|\mathbf{c}_{\mathbf{x},s}\right\|_2^2)\mathbf{c}_{\mathbf{x},s}\mathbf{c}_{\mathbf{x},s}^T. \tag{9.10}$$

Figure 9.5 illustrates a dataset for which we want to find a linear feature map $\mathbf{F}$ such that the new features $\mathbf{z} = \mathbf{F}\mathbf{x}$ do not allow to accurately predict a private attribute.

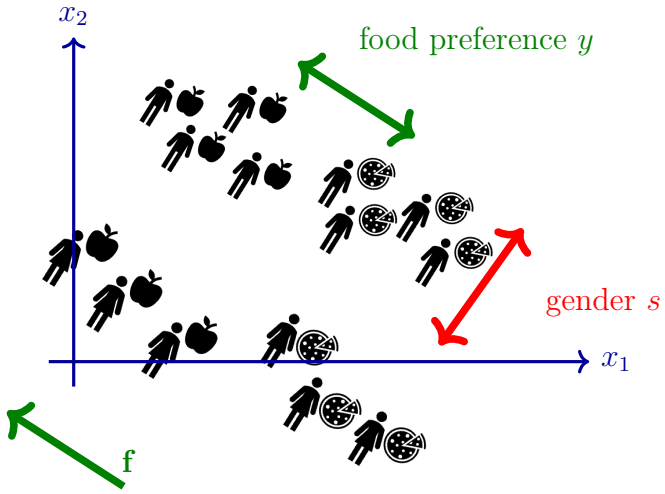


Figure 9.5: A toy dataset $\mathcal{D}$ whose data points represent costumers, each characterized by features $\mathbf{x} = \big(x_1, x_2\big)^T$. These raw features carry information about a private attribute s (gender) and the label y (food preference) of a person. The scatter-plot suggests that we can find a linear feature transformation $\mathbf{F} := \mathbf{f}^T \in \mathbb{R}^{1\times 2}$ resulting in a new feature $z := \mathbf{F}\mathbf{x}$ that does not allow to predict s, while still allowing to predict y.

9.5 Overview of Coding Assignment

Python Notebook. `PrivacyProtectionFL_CodingAssignment.ipynb`

Data File. `Assignment_MLBasicsData.csv`

This coding assignment builds on the coding assignment "ML Basics" (see Section 2.7). Similar to this previous coding assignment, we consider data points, indexed by $r = 1, \ldots, m$, that represent weather measurements recorded by the FMI. The r-th data point is characterized by (normalized) coordinates, a time stamp and a temperature measurement.

The overall idea of the assignment is to illustrate a model inversion attack: An attacker tries to determine the private attribute of an individual by evaluating a trained model (learnt hypothesis). Here, the private attribute is the latitude and longitude of a person that shares a snapshot of a thermometer during a Finland journey. The attacker might then use a trained ML model to predict the latitude and longitude from the disclosed (public) attributes $\mathbf{x}$ consisting of the time-stamp and temperature value.

9.5.1 Where Are You?

Consider a social media post of a friend that is travelling across Finland. This post includes a snapshot of a temperature measurement and a clock. Can you guess the latitude and longitude of the location where your friend took this snapshot? We can use ERM to do this: Use Algorithm 2.1 to learn a vector-valued hypothesis $\widehat{h}$ for predicting latitude and longitude from the time and value of a temperature measurement. For the training set and validation set, we use the FMI recordings stored in the data file.

9.5.2 Ensuring Privacy with Pre-Processing

Repeat the privacy attack described in Section 9.5.1 but this time using a pre-processed version of the raw data. In particular, try out combinations of randomly selecting a subset of the data points in the data file and also adding noise to their features and label. How well can you predict the latitude and longitude from the time and value of a temperature measurement using a hypothesis $\widehat{\mathbf{h}}$ learnt from the perturbed data.

9.5.3 Ensuring Privacy with Post-Processing

Repeat the privacy attack described in Section (9.5.1) but this time using a post-processing of the learnt hypothesis $\widehat{\mathbf{h}}$ (obtained from Algorithm 2.1 applied to the data file). In particular, study how well you can predict the latitude and longitude from the time and value of a temperature measurement using a noisy prediction hypothesis $\widehat{\mathbf{h}}(\mathbf{x}) + \mathbf{n}$. Here, $\mathbf{n}$ is a realization drawn from a multivariate normal distribution $\mathcal{N}(\mathbf{0}, \sigma^2\mathbf{I})$.

9.5.4 Private Feature Learning

We now use the original features $\mathbf{x} \in \mathbb{R}^7$ of a FMI weather recording as we used in the "ML Basics" assignment (see Section 2.7) The goal of this task is to learn a linear feature transformation $\mathbf{z} = \mathbf{F}\mathbf{x}$ such that the new features do not allow to recover the latitude x_1 (which is considered a private attribute s of the data point). In particular, we construct the matrix $\mathbf{F}$ according to (9.10) by replacing the exact cross-covariance vector $\mathbf{c}_{\mathbf{x},s}$ with an estimate (or approximation) $\hat{\mathbf{c}}_{\mathbf{x},s}$. This estimate is computed as follows:

1. read in all data points stored in the data file and construct a feature matrix $\mathbf{X} \in \mathbb{R}^{m \times 7}$ with m being the total number of data points

2. remove the sample means from each feature, resulting in the centered feature matrix

$$\widehat{\mathbf{X}} := \mathbf{X} - (1/m)\mathbf{1}\mathbf{1}^T\mathbf{X} \ , \ \mathbf{1} := \big(1, \ldots, 1\big)^T \in \mathbb{R}^m. \tag{9.11}$$

3. extract the private attribute (centred normalized latitude) for each data point and store in the vector

$$\mathbf{s} := \big(\hat{x}_1^{(1)}, \hat{x}_1^{(2)}, \ldots, \hat{x}_1^{(m)}\big)^T. \tag{9.12}$$

4. compute the approximate cross-covariance vector

$$\hat{\mathbf{c}}_{\mathbf{x},s} := (1/m)\big(\widehat{\mathbf{X}}\big)^T\mathbf{s} \tag{9.13}$$

The matrix $\mathbf{F}$ obtained from (9.10) by replacing $\mathbf{c}_{\mathbf{x},s}$ with $\hat{\mathbf{c}}_{\mathbf{x},s}$, is then used to compute the privacy preserving features $\mathbf{z}^{(r)} = \mathbf{F}\mathbf{x}^{(r)}$ for $r = 1, \ldots, m$. To verify if these new features are indeed privacy preserving, we use linear regression (as implemented by the `LinearRegression` class of the Python package `scikit-learn`) to learn the model parameters of a linear model to predict the private attribute $s^{(r)} = x_1^{(r)}$ (the latitude of the FMI station at which the r-th measurement has been taken) from the features $\mathbf{z}^{(r)}$. (Note that you have already trained and validated a linear model within the "ML Basics" assignment.) We can use the validation error of the trained linear model as a measure for the privacy protection (larger validation error means higher privacy protection). Beside the privacy protection, the new features

$\mathbf{z}^{(r)}$ should still allow to accurately predict the label $y^{(r)}$ of a data point. To this end, we learn the model parameters of a linear model to predict the private property $y^{(r)}$ solely from the features $\mathbf{z}^{(r)}$.

10 Lecture - "Data and Model Poisoning in FL"

Every ML method, including ERM or GTVMin, is to some extent at the mercy of the data generator. Indeed, the model parameters learnt via ERM (for basic ML) or via GTVMin (for FL) are determined by the statistical properties of the training set. We must hope (e.g., via an i.i.d. assumption) that the data points in the training set truthfully reflect the statistical properties of the underlying data generation process. However, these data points might have been intentionally perturbed (poisoned).

In general, it is impossible to perfectly detect if data points have been poisoned. The perfect detection of those perturbed data points requires complete knowledge of the underlying probability distribution. However, we typically do not know this probability distribution but can only estimate it from (possibly perturbed) data points. We can then use this estimate to detect perturbed data points via outlier detection techniques.

Instead of trying to identify and remove poisoned data points we can also try to make GTVMin-based FL systems more robust against data poisoning. We have already discussed the robustness of GTVMin-based methods in Section 8.3. The level of robustness crucially depends on the design choices for the local models, local loss functions and empirical graph in GTVMin.

Besides data poisoning, FL systems can also be subject to model poisoning attacks. FL systems are implemented with distributed computers such as a collection of smartphones (clients) that are inter-connected by wireless links (see Section 5). For some applications it is not possible (or desirable) to enforce sophisticated authentication techniques that determine which clients can participate in the FL system. The distributed computations might then

be compromised (poisoned) in order to disturb the training of the local model at a dedicated ("target") client.

10.1 Learning Goals

This lecture discusses the robustness of FL systems against data and model poisoning attacks. After this lecture, you should

1. be aware of data and model poisoning attacks at FL systems
2. have some intuition for how the design choices for GTVMin-based methods affect the vulnerability of FL systems against these attacks.

10.2 Attack Types

Consider a FL system that implements Algorithm 5.1 over a computer network. Each computer corresponds to one node of the empirical graph, and implements the local update in step 4 of Algorithm 5.1. The model parameters sharing in step 3 of Algorithm 5.1 is implemented over some communication channel (e.g., short-range wireless links).

An attack on such a FL system could be carried out in different forms, depending on the level of control that the attacker has over the implementation of Algorithm 5.1. If the attacker has some control over the communication links between the nodes, it can directly manipulate the model parameters shared between nodes (model poisoning). The attacker might instead have only access to the local datasets of some (vulnerable) nodes $\mathcal{W} \subset \mathcal{V}$. It could then manipulate (poison) the local datasets at these vulnerable nodes to perturb the corresponding local updates (see step 4 of Algorithm 5.1).

The perturbations at the nodes $i' \in \mathcal{W}$ then propagate over the edges of the empirical graph (via the model parameters sharing step 3 in Algorithm 5.1) and result in perturbed model parameters at other nodes (whose local datasets have not been poisoned).

Based on the objective of a attack on a FL system, we distinguish

- Denial-of-Service Attacks,
- Backdoor Attacks, and
- Privacy Attacks (see Lecture 9).

The goal of denial-of-service attacks is to render the learnt hypothesis $\bar{h}^{(i)}$, at some target node i, useless in the sense of having unacceptable large prediction errors. Denial-of-service attacks on GTVMin-based FL systems can be launched by manipulating some of the local datasets at few nodes in the empirical graph. These manipulated (poisoned) local datasets influence the model parameters at the target node i indirectly via the sharing of model parameters across the edges of the empirical graph.

Another type of denial-of-service attacks manipulate directly this sharing (communication) of model parameters across the edges in the empirical graph. This is possible when the sharing of model parameters is carried out over in-secure communication links. We can detect a denial-of-service attack by continuously monitoring the performance of the learnt model parameters $\mathbf{w}^{(i)}$.

A backdoor attack tries to make a target node i learn a hypothesis $\tilde{h}^{(i)}$ that behaves well on the local dataset $\mathcal{D}^{(i)}$ but highly irregular for a certain range of feature values. The goal of the attacker is to exploit this irregular behaviour by preparing a data point with feature values falling in this range

such that the hypothesis $\tilde{h}^{(i)}$ delivers a specific prediction (e.g., a prediction that results in granting access to a restricted area within a building). Figure 10.1 illustrates, for some (target) node i, the result of a denial-of-service and a backdoor attack on a FL system.

The goal of a privacy attack is to determine private attributes of the data points in the local dataset at the target node i. Here, the attacker could pretend to be a benign client corresponding to some node i'. This node uses a trivial loss function, e.g., equal to zero, and aims to learning similar model parameters as some target node i. To this end the attacker i' could try to establish many connections (which become edges of the empirical graph) to nodes that are in the same cluster as node i. After the attacker as ensured that their node i' has obtained similar model parameters as node i, they can try to probe the resulting hypothesis in order to infer private attributes of the data points in $\mathcal{D}^{(i)}$ (see Figure 9.2).

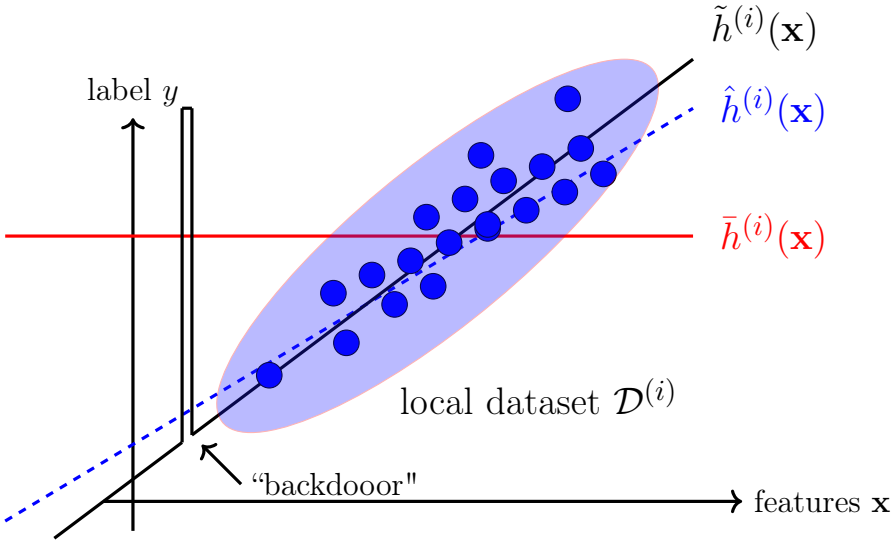


Figure 10.1: A local dataset $\mathcal{D}^{(i)}$ along with three hypothesis maps learnt via some GTVMin-based method such as Algorithm 5.1. These three maps are obtained for different attacks to the FL system: The map $\hat{h}^{(i)}$ is obtained when no manipulation is applied (no attacks). A backdoor attack aims at nudging the learnt hypothesis $\tilde{h}^{(i)}$ to behave similar to $\hat{h}^{(i)}$ when applied to data points in $\mathcal{D}^{(i)}$. However, it behaves very different for a certain range of feature values outside of $\mathcal{D}^{(i)}$. This value range can be interpreted as a trigger that opens a "backdoor".

10.3 Data Poisoning

Consider a GTVMin-based FL system that learns local model parameters for a local (personal) model for each local dataset $\mathcal{D}^{(i)}$, $i = 1, \ldots, n$. A data poisoning attack on such a FL system is to manipulate (to poison) data points in some of the local datasets.

The manipulation (poisoning) of data points can consist of adding the realization of RVs to the features and label of a data point: We poison a data point by replacing its features $\mathbf{x}$ and label y with $\widetilde{\mathbf{x}} := \mathbf{x} + \Delta\mathbf{x}$ and $\tilde{y} = y + \Delta y$.

For classification problems, with discrete label spaces, we distinguish between the following data poisoning strategies [104]:

- Label Poisoning: The attacker manipulates the labels of data points in the training set.
- Clean-Label Attack: The attacker leaves the labels untouched and only manipulates the features of data points in the training set.

From a GTVMin-perspective, the effect of a data poisoning attack is that the original local loss functions $L_i(\cdot)$ in GTVMin (3.17) are replaced by perturbed local loss functions $\tilde{L}_i(\cdot)$. The extend of perturbation depends on the fraction of data points that are poisoned as well as on the loss function used to measure the prediction errors.

Different choices for the underlying loss function offer different levels of robustness against poisoning. For example, using the absolute error loss yields higher robustness against perturbations of the label values of few data points

compared to the squared error loss. Another class of robust loss functions is obtained by including a penalty term (as in regularization).

10.4 Model Poisoning

A model poisoning attack on Algorithm 5.1 manipulates the model parameters sharing step such that a target node i receives perturbed model parameters of its neighbours. Note that Algorithm 5.1 is nothing but a message passing implementation of plain GD (see Section 4.2). Thus, the effect of a model poisoning attack on Algorithm 5.1 is that it becomes an instance of perturbed GD. We can use the analysis of perturbed GD (see Section 4.5) to study the impact of model poisoning on the learnt model parameters $\mathbf{w}^{(i)}$.

10.5 Assignment

Python Notebook. `PoisoningFL_CodingAssignment.ipynb`
Data File. `Assignment_MLBasicsData.csv`

This coding assignment builds on the coding assignments "ML Basics" (see Section 2.7) and "FL Algorithm" (see Section 5.6). As in these previous assignments, we consider data points, indexed by $r = 1, \ldots, m$, that represent weather measurements recorded by the FMI. The r-th data point is characterized by (normalized) coordinates, a time stamp and a temperature measurement.

The overall idea of the assignment is to illustrate two different types of data poisoning attacks: a denial-of-service attack and a backdoor attack.

10.5.1 Denial-of-Service Attack

Construct an empirical graph of FMI stations and store it as a `networkx.Graph()` object. Implement Algorithm 5.1 to learn, for each node $i = 1, \ldots, n$, the model parameters of local linear model. Then, implement a denial-of-service attack by poisoning the local datasets as increasingly many nodes $i' \neq 1$. The goal of the attack is to increase the validation error of the learnt model parameters $\mathbf{w}^{(1)}$ (at target node $i = 1$) by 20 %.

10.5.2 Backdoor Attack

We now use a different collection of features for a data point (= temperature recording). In particular, we replace the numeric feature representing the hour of the measurement with 24 new features. These new features are the one-hot encoding of the hour. For example if the hour was 0 then $x'_1 = 1, x'_2 = 0, \ldots$.

Glossary

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). 3, 5, 16

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. 2, 9

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. 1, 6, 7, 11, 13, 15, 20, 26

autoencoder An autoencoder is a ML method that jointly learns an encoder map $h(\cdot) \in \mathcal{H}$ and a decoder map $h^{*}(\cdot) \in \mathcal{H}^{*}$. It is an instance of ERM using a loss computed from the reconstruction error $\mathbf{x} - h^{*}\big(h\big(\mathbf{x}\big)\big)$. 8, 9

baseline A reference value or benchmark for the average loss incurred by a hypothesis when applied to the data points generated in a specific ML application. Such a reference value might be obtained from human

performance (e.g., error rate of dermatologists diagnosing cancer from visual inspection of skin areas) or other ML methods ("competitors") 11–13

batch A set of randomly selected data points out of a (typically very large) dataset. 6, 9, 10, 17

Bayes estimator A hypothesis h whose Bayes risk is minimal [29]. 2, 12

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 [29]. 2

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

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. 11, 13

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. 5–8, 13, 16

clustered federated learning (CFL) Clustered FL assumes that local datasets clusters. The local datasets belonging to the same cluster have similar statistical properties. 1, 2, 5

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 GMM represents a cluster by a multivariate normal distribution. 1, 2, 5, 16

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 [54]. We obtain different clustering methods by using different notions of similarity between data points. 2

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

techniques 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. 5, 8, 11, 16

condition number The condition number $\kappa(\mathbf{Q}) \geq 1$ of a psd matrix $\mathbf{Q}$ is the ratio $\lambda_{\max}/\lambda_{\min}$ between the largest $\lambda_{\max}$ and the smallest $\lambda_{\min}$ eigenvalue of $\mathbf{Q}$. One example for such a matrix $\mathbf{Q}$ is obtained from the feature matrix $\mathbf{X}$ of a dataset, $\mathbf{Q} = \mathbf{X}\mathbf{X}^T$. From a computational perspective, we prefer $\mathbf{Q}$ to have a condition number close to 1. 4

connected graph An undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is connected if we cannot find a (non-empty) subset $\mathcal{V}' \subset \mathcal{V}$ with no edges leaving $\mathcal{V}'$. 6

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 [28]. 2–4, 8, 9, 17, 22, 28

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, 16, 18

data A (indexed) set of data points. 1, 2, 4, 8, 9, 15, 17, 21

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. 14–16

data point A data point is any object that conveys information [105]. 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. 1–21, 23–25, 27, 28

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 [106, 107]. 3

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 [108]. By using indexed lists of data points we avoid some of the challenges arising in concept of an abstract set. 1–12, 14–18

decision region Consider a hypothesis map h that delivers values from a

finite set $\mathcal{Y}$. We refer to the set of features $\mathbf{x} \in \mathcal{X}$ that result in the same output $h(\mathbf{x}) = a$ as a decision region of the hypothesis h. 4, 15

decision tree A decision tree is a flow-chart like representation of a hypothesis map h. More formally, a decision tree is a directed graph which reads in the feature vector $\mathbf{x}$ of a data point at its root node. The root node then forwards the data point to one of its children nodes based on some elementary test on the features $\mathbf{x}$. If the receiving children node is not a leaf node, i.e., it has itself children nodes, it represents another test. Based on the test result, the data point is further pushed to one of its neighbours. This testing and forwarding of the data point is repeated until the data point ends up in a leaf node (having no children nodes). The leaf nodes represent sets (decision regions) constituted by feature vectors $\mathbf{x}$ that are mapped to the same function value $h(\mathbf{x})$. 3, 5, 13, 15

deep net We refer to an ANN with a (relatively) large number of hidden layers as a deep ANN or "deep net". Deep nets are used to represent the hypothesis spaces of deep learning methods [68]. 13

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]. 2, 3, 10, 12, 15, 25

differential privacy Consider an algorithm used to compute some output based on a dataset. Differential privacy is a precise measure of this algorithm's privacy leakage. Roughly speaking, the algorithm is differentially private if the probability distribution of its output does not allow it to infer if the dataset contains a specific data point. 6–12

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. 1, 3, 6–9, 11–13

edge weight Each edge $\{i, i'\}$ of an empirical graph is assigned a non-negative edge weight $A_{i,i'} \geq 0$. A zero weight $A_{i,i'} = 0$ indicates the absence of an edge between nodes $i, i' \in \mathcal{V}$. 1–3, 6–10, 12, 13, 15, 16

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. 12, 13

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 $\mathbf{A}\mathbf{x} = \lambda \mathbf{x}$. 3–12, 14–16, 18, 19

eigenvalue decomposition The eigenvalue decomposition for a square matrix $\mathbf{A} \in \mathbb{R}^{d \times d}$ is a factorization of the form

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

The columns of the matrix $\mathbf{V} = \big(\mathbf{v}^{(1)}, \ldots, \mathbf{v}^{(d)}\big)$ are the eigenvectors of the matrix $\mathbf{V}$. The diagonal matrix $\mathbf{\Lambda} = \mathrm{diag}\big\{\lambda_1, \ldots, \lambda_d\big\}$ contains the eigenvalues λ_j corresponding to the eigenvectors $\mathbf{v}^{(j)}$. Note that the above decomposition exists only if the matrix $\mathbf{A}$ is diagonalizable. 4, 5, 9

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. 4–7, 16

empirical graph Empirical graphs represent collections of local datasets and corresponding local models [54]. 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. 1–19, 21, 27

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. 2, 8, 14, 19, 20, 27

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. 1–4, 6, 8, 9, 11–16, 19, 20, 23, 27, 28

estimation error Consider data points with feature vectors $\mathbf{x}$ and label y. In some applications we can model the relation between features and label of a data point as $y = \bar{h}(\mathbf{x}) + \varepsilon$. Here we used some true hypothesis $\bar{h}$ and a noise term ε which might represent modelling or labelling errors. The estimation error incurred by a ML method that learns a hypothesis $\widehat{h}$, e.g., using ERM, is defined as $\widehat{h} - \bar{h}$. For a parametrized hypothesis space, consisting of hypothesis maps that are determined by a parameter vector $\mathbf{w}$, we define the estimation error in terms of parameter vectors as $\Delta\mathbf{w} = \widehat{\mathbf{w}} - \overline{\mathbf{w}}$. first 7–10, 12, 13

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]. 3, 4, 10, 23

explainability We define the (subjective) explainability of a ML method as the level of simulatability [85] of the predictions delivered to a specific human user. Quantitative measures for the (subjective) explainability of a trained model can be constructed by comparing its predictions with the corresponding guesses provided by a user on a test-set [87]. Alternatively, we can use probabilistic models for data and measure explainability of a trained ML model via the (differential) conditional entropy of its predictions [86, 109]. 1, 3, 6, 17, 18

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" [110–112]. However, this book makes consequent use of the term features for low-level properties of data points that can be measured easily. 1–21, 23, 25, 28

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. 13–15

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 $\mathbf{X} := \big(\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)}\big)$. 2–4, 7–10, 12, 15, 16, 18

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, 13, 15

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 [13] consider federated averaging methods where the local model training is implemented by running several GD steps 2, 14

federated learning (FL) Federated learning is an umbrella term for ML methods that train models in a collaborative fashion using decentralized data and computation. 1–19, 28

Finnish Meteorological Institute The Finnish Meteorological Institute is a government agency responsible for gathering and reporting weather data in Finland. 2, 7, 8, 12, 13, 15–18, 21

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 $c = 1, \ldots, k$. 3, 5, 13, 16

General Data Protection Regulation The General Data Protection Regulation (GDPR) is a law that has been passed by the European Union (EU) and put into effect on May 25, 2018 `https://gdpr.eu/tag/gdpr/`. The GDPR imposes obligations onto organizations anywhere, so long as they target, collect or in any other way process data related to people (i.e., personal data) in the EU. 4, 6

generalized total variation Generalized total variation measures the changes of vector-valued node attributes of a graph. 7, 8, 10–12, 16

gradient For a real-valued function $f : \mathbb{R}^d \to \mathbb{R} : \mathbf{w} \mapsto f(\mathbf{w})$, a vector $\mathbf{g}$ such that $\lim_{\mathbf{w}\to\mathbf{w}'} \frac{f(\mathbf{w}) - \left(f(\mathbf{w}') + \mathbf{g}^T(\mathbf{w}-\mathbf{w}')\right)}{\|\mathbf{w}-\mathbf{w}'\|} = 0$ is referred to as the gradient of f at $\mathbf{w}'$. If such a vector exists it is denoted $\nabla f(\mathbf{w}')$ or $\nabla f(\mathbf{w})\big|_{\mathbf{w}'}$ [5]. 1–6, 8, 10, 12, 13, 15, 25, 26, 29

gradient descent (GD) Gradient descent is an iterative method for finding the minimum of a differentiable function $f(\mathbf{w})$. 1, 2, 4, 6–8, 10, 12–15, 17, 22, 26

gradient step Given a differentiable real-valued function $f(\mathbf{w})$ and a vector $\mathbf{w}'$, the gradient step updates $\mathbf{w}'$ by adding the scaled negative gradient $\nabla f(\mathbf{w}')$, $\mathbf{w}' \mapsto \mathbf{w}' - \eta \nabla f(\mathbf{w}')$. 1–14, 16, 18, 22

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. 1–5, 7, 8, 10, 11, 13, 14, 16, 26

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 [113]. 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 $e \in \mathcal{E}$. 6, 8, 9, 19

GTV minimization GTV minimization is an instance of RERM using the GTV of local model parameters as a regularizer. 1–14, 17

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). 1, 2, 9, 10

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. 1–20, 23–25, 27, 28

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. 1, 6, 7, 12–18, 20, 24, 27, 28

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. 3–8, 10, 12–17, 25

i.i.d. assumption The i.i.d. assumption interprets data points of a dataset as the realizations of i.i.d. RVs. 1, 3, 6, 8, 12, 14, 25

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 [86]. 16

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" [110–112]. 1–21, 23, 25, 27

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"$\}$ 11, 13

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) [114, 115]. 1, 3–6, 9, 16, 18

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. 13, 15

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 1, 2, 4–11, 13, 16, 17, 26

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. 3, 14

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 $\mathbf{w}$. 24

linear model We use the term linear model in a very specific sense. In particular, a linear model is a hypothesis space which consists of all linear maps,

$$\mathcal{H}^{(d)} := \left\{h(\mathbf{x}) = \mathbf{w}^T\mathbf{x} : \mathbf{w} \in \mathbb{R}^d\right\}. \tag{10.1}$$

Note that (10.1) 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). 1–3, 6–13, 15–19

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). 2–4, 7–10, 14, 16–18

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. 1–18, 28

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 $i, i' \in \mathcal{V}$. 1–8, 11–13, 16, 17, 19, 27

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. 1–5, 7–10, 12, 14–18, 20, 24, 25, 27, 28

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 10.2 depicts a loss function for a given data point, with features $\mathbf{x}$ and label y, as a function of the hypothesis $h \in \mathcal{H}$. 1–8, 10, 11, 13, 17, 18, 25

maximum likelihood Consider data points $\mathcal{D} = \big\{\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(m)}\big\}$ that are interpreted as realizations of i.i.d. RVs with a common probability

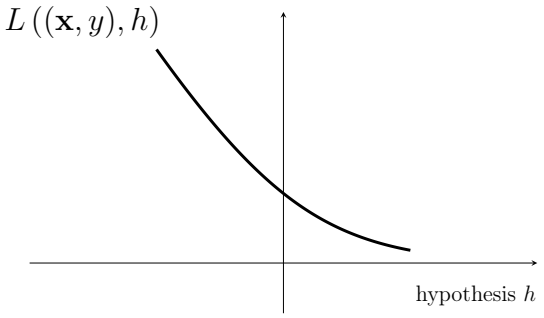


Figure 10.2: Some loss function $L\left((\mathbf{x}, y), h\right)$ for a fixed data point, with feature vector $\mathbf{x}$ and label y, and varying hypothesis h. ML methods try to find (learn) a hypothesis that incurs minimum loss.

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 $\max_{\mathbf{w} \in \mathcal{W}} p(\mathcal{D}; \mathbf{w})$. 7, 8

model We use the term model as a synonym for hypothesis space 1–5, 8, 9, 11–16, 18, 20

model parameters Model parameters are numbers that select a hypothesis map out of a hypothesis space. 1–19, 21, 22

multivariate normal distribution The multivariate normal distribution $\mathcal{N}(\mathbf{m}, \mathbf{C})$ is an important family ofprobability distributions for a continuous RV $\mathbf{x} \in \mathbb{R}^d$ [3, 116, 117]. This family is paramtrized by the mean $\mathbf{m}$ and covariance matrix $\mathbf{C}$ of $\mathbf{x}$. If the covariance matrix is invertible,

the probability distribution of $\mathbf{x}$ is

$$p(\mathbf{x}) \propto \exp\Big(-(1/2)\big(\mathbf{x}-\mathbf{m}\big)^T \mathbf{C}^{-1}\big(\mathbf{x}-\mathbf{m}\big)\Big).$$

3, 11, 17

mutual information The mutual information $I(\mathbf{x};y)$ between two RVs $\mathbf{x}$, y defined on the same probability space is given by [105]

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

It is a measure for how well we can estimate y based solely from $\mathbf{x}$. A large value of $I(\mathbf{x};y)$ indicates that y can be well predicted solely from $\mathbf{x}$. This prediction could be obtained by a hypothesis learnt by a ERM-based ML method. 12, 13

neighbourhood The neighbourhood of a node $i \in \mathcal{V}$ within an empirical graph are those nodes $i' \in \mathcal{V} \setminus \{i\}$ that are connected to i by an edge with i. 4

neighbours The neighbours of a node $i \in \mathcal{V}$ within an empirical graph are those nodes $i' \in \mathcal{V}$ that are connected by an edge with i. 6–8, 10, 16

node degree The degree of a node i in an undirected empirical graph is the number of its neighbours. 2–5, 7, 10, 11

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 $h^{(\mathbf{w})}$. 1–11, 14–18, 28

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. 10, 12

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. 1–19, 27

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). 13

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 $\mathbf{x} \in \mathbb{R}^d$. 2, 4–7, 10, 15

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. 2–7, 9, 14–19, 25

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 [29]. 2, 5–13, 17, 26

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 $p(x \in \mathcal{B}) = \int_{\mathcal{B}} p(x')dx'$ [3, 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 $p(\mathbf{x} \in \mathcal{R}) = \int_{\mathcal{R}} p(\mathbf{x}')dx'_1 \ldots dx'_d$ [3, Ch. 3]. 13, 14

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 $p(y = 1)\big(=1–p(y = 0)\big)$. 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 [92, 116]. 1, 5–7, 10–15, 17–19, 21, 23, 25, 26

projected GD Projected GD extends basic GD for unconstrained optimization to handle constraints on the optimization variable (model parameters). A single iteration of projected GD consists of first taking a gradient step and then projecting the result back into a constrain set. 1, 2, 10, 12, 13

proximable A Convex function for which the proximity operator can be computed efficiently are sometimes referred to as "proximable" or "simple" [37]. 10

proximal operator Given a convex function and a vector $\mathbf{x}$, we define its proximity operator as

$$\mathbf{prox}_{L_i(\cdot),2\alpha}(\mathbf{w}'') := \operatorname*{argmin}_{\mathbf{w}\in\mathbb{R}^d} f(\mathbf{w}) + (\rho/2)\left\|\mathbf{w} - \mathbf{w}'\right\|_2^2 \text{ with } \rho > 0.$$

Convex functions for which the proximity operator can be computed efficiently are sometimes referred to as "proximable" or "simple" [37]. 10, 13, 14, 22

quadratic function A quadratic function $f(\mathbf{w})$, reading in a vector $\mathbf{w} \in \mathbb{R}^d$ as its argument, is such that

$$f(\mathbf{w}) = \mathbf{w}^T\mathbf{Q}\mathbf{w} + \mathbf{q}^T\mathbf{w} + a,$$

with some matrix $\mathbf{Q} \in \mathbb{R}^{d\times d}$, vector $\mathbf{q} \in \mathbb{R}^d$ and scalar $a \in \mathbb{R}$. 2, 4, 8, 9, 17

Rényi divergence The Rényi divergence measures the (dis-)similarity between two probability distributions. 7

random variable (RV) A random variable is a mapping from a probability space $\mathcal{P}$ to a value space [92]. The probability space, whose elements are elementary events, is equipped with a probability measure that assigns a probability to subsets of $\mathcal{P}$. A binary random variable maps elementary events to a set containing two different values, e.g., $\{-1, 1\}$ or $\{\text{cat}, \text{no cat}\}$. A real-valued random variable maps elementary events to real numbers $\mathbb{R}$. A vector-valued random variable maps elementary events to the Euclidean space $\mathbb{R}^d$. Probability theory uses the concept of measurable spaces to rigorously define and study the properties of (large) collections of random variables [92, 116]. 2–8, 10–15, 17–19, 21, 23–25, 28

realization Consider a RV x which maps each element (outcome, or elementary event) $\omega \in \mathcal{P}$ of a probability space $\mathcal{P}$ to an element a of a measurable space $\mathcal{N}$ [5, 91, 92]. A realization of x is any element $a' \in \mathcal{N}$ such that there is an element $\omega' \in \mathcal{P}$ with $x(\omega') = a'$. 3, 5–8, 10–12, 14, 16, 21, 25

regression Regresion problems revolve around the problem of predicting a numeric label solely from the features of a data point. 13

regularization Regularization techniques modify the ERM principle such that the learnt hypothesis performs well (generalizes) beyond the training set. One specific implementation of regularization is to add a penalty or regularization term to the objective function of ERM (which is the

average loss on the training set). This regularization term can be interpreted as an estimate for the increase in the expected loss (risk) compared to the average loss on the training set. 1, 5, 7, 8, 12–15, 18, 23, 24, 27

regularized empirical risk minimization Synoym for SRM. 8, 12, 15

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 $h^{(\mathbf{w})}(\mathbf{x}) := \mathbf{w}^T\mathbf{x}$ [4, Ch. 3]. The least absolute shrinkage and selection operator (Lasso) uses the regularizer $\mathcal{R}\big\{h\big\} := \|\mathbf{w}\|_1$ for linear hypothesis maps $h^{(\mathbf{w})}(\mathbf{x}) := \mathbf{w}^T\mathbf{x}$ [4, Ch. 3]. 6, 8, 12, 13, 15

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 $\alpha\|\mathbf{w}\|_2^2$ with a regularization parameter $\alpha > 0$. It can be shown that the effect of adding to $\alpha\|\mathbf{w}\|_2^2$ 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. 2, 3, 9, 13, 15–18, 24

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. 2, 7, 12, 19, 24

scatterplot A visualization technique that depicts data points by markers in a two-dimensional plane. 4, 10

semi-supervised learning Semi-supervised learning methods use (large amounts of) unlabeled data points to support the learning of a hypothesis from (a small number of) labeled data points [54]. 9, 10

smooth We refer to a real-valued function as smooth if it is differentiable and its gradient is continuous [47, 118]. In particular, a differentiable function $f(\mathbf{w})$ is referred to as β-smooth if the gradient $\nabla f(\mathbf{w})$ is Lipschitz continuous with Lipschitz constant β, i.e.,

$$\|\nabla f(\mathbf{w}) - \nabla f(\mathbf{w}')\| \leq \beta \|\mathbf{w} - \mathbf{w}'\|.$$

1, 3, 5, 28

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

$$L\left((\mathbf{x}, y), h\right) := \big(y - \underbrace{h(\mathbf{x})}_{=\hat{y}}\big)^2. \tag{10.2}$$

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. 15, 24

total variation Consider some empirical graph whose nodes carry (personalized) local model parameters. The total variation measures the discrepancies between across edges $\{i, i'\}$. 1, 2, 4, 6, 8, 15, 16

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. 1, 11–15, 17–19, 27, 28

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). 1, 3–18, 20, 23, 24, 27, 28

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 $\mathcal{D}$. 1, 6, 10, 11

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. 1, 3, 6, 8, 11–14, 17–19, 27, 28

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 [4, 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). 3, 10, 11, 13–18, 27, 28

variance The variance of a real-valued RV x is defined as the expectation $\mathbb{E}\big\{\big(x - \mathbb{E}\{x\}\big)^2\big\}$ of the squared difference x and its expectation $\mathbb{E}\{x\}$. We extend this definition to vector-valued RVs $\mathbf{x}$ as $\mathbb{E}\big\{\big\|\mathbf{x} - \mathbb{E}\{\mathbf{x}\}\big\|_2^2\big\}$. 14

vertical FL Vertical FL refers to applications with local datasets that are constituted by the same data points but characterizing them with different features [55]. 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. 2, 11, 12

zero-gradient condition Consider the unconstrained optimization problem $\min_{\mathbf{w}\in\mathbb{R}^d} f(\mathbf{w})$ with a smooth and convex objective function $f(\mathbf{w})$. A

necessary and sufficient condition for a vector $\widehat{\mathbf{w}} \in \mathbb{R}^d$ to solve this problem is that the gradient $\nabla f\big(\widehat{\mathbf{w}}\big)$ is the zero-vector,

$$\nabla f\big(\widehat{\mathbf{w}}\big) = \mathbf{0} \Leftrightarrow f\big(\widehat{\mathbf{w}}\big) = \min_{\mathbf{w} \in \mathbb{R}^d} f(\mathbf{w}).$$

11

References

[1] W. Rudin, *Real and Complex Analysis*, 3rd ed. New York: McGraw-Hill, 1987.

[2] 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.

[3] D. Bertsekas and J. Tsitsiklis, *Introduction to Probability*, 2nd ed. Athena Scientific, 2008.

[4] A. Jung, *Machine Learning: The Basics*, 1st ed. Springer Singapore, Feb. 2022.

[5] W. Rudin, *Principles of Mathematical Analysis*, 3rd ed. New York: McGraw-Hill, 1976.

[6] 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.

[7] 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

[8] 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

[9] 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

[10] S. Cui, A. Hero, Z.-Q. Luo, and J. Moura, Eds., *Big Data over Networks*. Cambridge Univ. Press, 2016.

[11] A. Barabási, N. Gulbahce, and J. Loscalzo, "Network medicine: a network-based approach to human disease," *Nature Reviews Genetics*, vol. 12, no. 56, 2011.

[12] M. E. J. Newman, *Networks: An Introduction*. Oxford Univ. Press, 2010.

[13] 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. PMLR, 20–22 Apr 2017, pp. 1273–1282. [Online]. Available: https://proceedings.mlr.press/v54/mcmahan17a.html

[14] 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.

[15] 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.

[16] 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.

[17] 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

[18] 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

[19] D. Tse and P. Viswanath, *Fundamentals of Wireless Communication*. Cambridge University Press, 2005.

[20] 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

[21] 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.

[22] 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.

[23] G. Strang, *Computational Science and Engineering.* Wellesley-Cambridge Press, MA, 2007.

[24] ——, *Introduction to Linear Algebra*, 5th ed. Wellesley-Cambridge Press, MA, 2016.

[25] H. H. Bauschke and P. L. Combettes, *Convex Analysis and Monotone Operator Theory in Hilbert Spaces.* New York: Springer, 2011.

[26] 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

[27] J. Hirvonen and J. Suomela. (2023) Distributed algorithms 2020.

[28] S. Boyd and L. Vandenberghe, *Convex Optimization.* Cambridge, UK: Cambridge Univ. Press, 2004.

[29] E. L. Lehmann and G. Casella, *Theory of Point Estimation*, 2nd ed. New York: Springer, 1998.

[30] A. Esteva, B. Kuprel, R. A. Novoa, J. Ko, S. M. Swetter, H. M. Blau, and S. Thrun, "Dermatologist-level classification of skin cancer with deep neural networks," *Nature*, vol. 542, 2017.

[31] H. Lütkepohl, *New Introduction to Multiple Time Series Analysis*. New York: Springer, 2005.

[32] Y. SarcheshmehPour, Y. Tian, L. Zhang, and A. Jung, "Clustered federated learning via generalized total variation minimization," *IEEE Transactions on Signal Processing*, vol. 71, pp. 4240–4256, 2023.

[33] D. Spielman, "Spectral and algebraic graph theory," 2019.

[34] F. Chung, "Spectral graph theory," in *Regional Conference Series in Mathematics*, 1997, no. 92.

[35] D. Spielman, "Spectral graph theory," in *Combinatorial Scientific Computing*, U. Naumann and O. Schenk, Eds. Chapman and Hall/CRC, 2012.

[36] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 4th ed. Baltimore, MD: Johns Hopkins University Press, 2013.

[37] 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.

[38] N. Parikh and S. Boyd, "Proximal algorithms," *Foundations and Trends in Optimization*, vol. 1, no. 3, pp. 123–231, 2013.

[39] R. Peng and D. A. Spielman, "An efficient parallel solver for SDD linear systems," in *Proc. ACM Symposium on Theory of Computing*, New York, NY, 2014, pp. 333–342.

[40] N. K. Vishnoi, "$Lx = b$ — Laplacian solvers and their algorithmic applications," *Foundations and Trends in Theoretical Computer Science*, vol. 8, no. 1–2, pp. 1–141, 2012. [Online]. Available: http://dx.doi.org/10.1561/0400000054

[41] A. Jacot, F. Gabriel, and C. Hongler, "Neural tangent kernel: Convergence and generalization in neural networks," in *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., vol. 31. Curran Associates, Inc., 2018. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2018/file/5a4be1fa34e62bb8a6ec6b91d2462f5a-Paper.pdf

[42] S. S. Du, X. Zhai, B. Póczos, and A. Singh, "Gradient descent provably optimizes over-parameterized neural networks," in *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. [Online]. Available: https://openreview.net/forum?id=S1eK3i09YQ

[43] W. E, C. Ma, and L. Wu, "A comparative analysis of optimization and generalization properties of two-layer neural network and random feature models under gradient descent dynamics," *Science China Mathematics*, vol. 63, no. 7, pp. 1235–1258, 2020. [Online]. Available: https://doi.org/10.1007/s11425-019-1628-5

[44] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. Red Hook, NY, USA: Curran Associates Inc., 2019.

[45] D. P. Bertsekas, *Convex Optimization Algorithms*. Athena Scientific, 2015.

[46] 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.

[47] 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

[48] D. P. Bertsekas and J. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods*. Athena Scientific, 2015.

[49] D. Mills, "Internet time synchronization: the network time protocol," *IEEE Transactions on Communications*, vol. 39, no. 10, pp. 1482–1493, 1991.

[50] R. Diestel, *Graph Theory*. Springer Berlin Heidelberg, 2005.

[51] Q. Yang, Y. Liu, Y. Cheng, Y. Kang, T. Chen, and H. Yu, *Federated Learning*, 1st ed. Springer, 2022.

[52] D. J. Spiegelhalter, "An omnibus test for normality for small samples," *Biometrika*, vol. 67, no. 2, pp. 493–496, 2024/03/25/ 1980. [Online]. Available: http://www.jstor.org/stable/2335498

[53] Q. Yang, Y. Liu, Y. Cheng, Y. Kang, T. Chen, and H. Yu, *Horizontal Federated Learning*. Cham: Springer International Publishing, 2020, pp. 49–67. [Online]. Available: https://doi.org/10.1007/978-3-031-01585-4_4

[54] O. Chapelle, B. Schölkopf, and A. Zien, Eds., *Semi-Supervised Learning*. Cambridge, Massachusetts: The MIT Press, 2006.

[55] 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

[56] H. Ludwig and N. Baracaldo, Eds., *Federated Learning: A Comprehensive Overview of Methods and Applications*. Springer, 2022.

[57] 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

[58] J. Tropp, "An introduction to matrix concentration inequalities," *Found. Trends Mach. Learn.*, May 2015.

[59] B. Bollobas, W. Fulton, A. Katok, F. Kirwan, and P. Sarnak, *Random graphs*. Cambridge studies in advanced mathematics., 2001, vol. 73.

[60] G. Keiser, *Optical Fiber Communication*, 4th ed. New Delhi: Mc-Graw Hill, 2011.

[61] D. Tse and P. Viswanath, *Fundamentals of wireless communication*. USA: Cambridge University Press, 2005.

[62] B. Ying, K. Yuan, Y. Chen, H. Hu, P. PAN, and W. Yin, "Exponential graph is provably efficient for decentralized deep training," in *Advances in Neural Information Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, Eds., vol. 34. Curran Associates, Inc., 2021, pp. 13 975–13 987. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2021/file/74e1ed8b55ea44fd7dbb685c412568a4-Paper.pdf

[63] Y.-T. Chow, W. Shi, T. Wu, and W. Yin, "Expander graph and communication-efficient decentralized optimization," in *2016 50th Asilomar Conference on Signals, Systems and Computers*, 2016, pp. 1715–1720.

[64] M. Fiedler, "Algebraic connectivity of graphs," *Czechoslovak Mathematical Journal,*, vol. 23, no. 2, pp. 298–305, 1973.

[65] S. Hoory, N. Linial, and A. Wigderson, "Expander graphs and their applications," *Bull. Amer. Math. Soc.*, vol. 43, no. 04, pp. 439–562, Aug. 2006.

[66] S. M. Kay, *Fundamentals of Statistical Signal Processing: Estimation Theory*. Englewood Cliffs, NJ: Prentice Hall, 1993.

[67] S. Chepuri, S. Liu, G. Leus, and A. Hero, "Learning sparse graphs under smoothness prior," in *Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 2017, pp. 6508–6512.

[68] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[69] 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.

[70] E. Commission, C. Directorate-General for Communications Networks, and Technology, *The Assessment List for Trustworthy Artificial Intelligence (ALTAI) for self assessment*. Publications Office, 2020.

[71] H.-L. E. G. on Artificial Intelligence, "Ethics guidelines for trustworthy ai," European Commission, Tech. Rep., April 2019.

[72] D. J. Kuss and O. Lopez-Fernandez, "Internet addiction and problematic internet use: A systematic review of clinical research." *World J Psychiatry*, vol. 6, no. 1, pp. 143–176, Mar 2016.

[73] L. Munn, "Angry by design: toxic communication and technical architectures," *Humanities and Social Sciences Communications*, vol. 7, no. 1, p. 53, 2020. [Online]. Available: https://doi.org/10.1057/s41599-020-00550-7

[74] P. Mozur, "A genocide incited on facebook, with posts from myanmar's military," *The New York Times*, 2018.

[75] A. Simchon, M. Edwards, and S. Lewandowsky, "The persuasive effects of political microtargeting in the age of generative artificial intelligence." *PNAS Nexus*, vol. 3, no. 2, p. pgae035, Feb 2024.

[76] J. Near and D. Darais, "Guidelines for evaluating differential privacy guarantees," National Institute of Standards and Technology, Gaithersburg, MD, Tech. Rep., 2023.

[77] S. Wachter, "Data protection in the age of big data," *Nature Electronics*, vol. 2, no. 1, pp. 6–7, 2019. [Online]. Available: https://doi.org/10.1038/s41928-018-0193-y

[78] P. Samarati, "Protecting respondents identities in microdata release," *IEEE Transactions on Knowledge and Data Engineering*, vol. 13, no. 6, pp. 1010–1027, 2001.

[79] E. Comission, "Regulation (eu) 2016/679 of the european parliament and of the council of 27 april 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing directive 95/46/ec (general data protection regulation) (text with eea relevance)," no. 119, pp. 1–88, May 2016.

[80] U. N. G. Assembly, *The Universal Declaration of Human Rights (UDHR)*, New York, 1948.

[81] N. Kozodoi, J. Jacob, and S. Lessmann, "Fairness in credit scoring: Assessment, implementation and profit implications," *European Journal*

of Operational Research, vol. 297, no. 3, pp. 1083–1094, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0377221721005385

[82] J. Gonçalves-Sá and F. Pinheiro, *Societal Implications of Recommendation Systems: A Technical Perspective*. Cham: Springer International Publishing, 2024, pp. 47–63. [Online]. Available: https://doi.org/10.1007/978-3-031-41264-6_3

[83] A. Abrol and R. K. Jha, "Power optimization in 5g networks: A step towards green communication," *IEEE Access*, vol. 4, pp. 1355–1374, 2016.

[84] C. Wang, Y. Yang, and P. Zhou, "Towards efficient scheduling of federated mobile devices under computational and statistical heterogeneity," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 2, pp. 394–410, 2021.

[85] J. Colin, T. Fel, R. Cadène, and T. Serre, "What i cannot predict, i do not understand: A human-centered evaluation framework for explainability methods." *Adv Neural Inf Process Syst*, vol. 35, pp. 2832–2845, 2022.

[86] A. Jung and P. Nardelli, "An information-theoretic approach to personalized explainable machine learning," *IEEE Sig. Proc. Lett.*, vol. 27, pp. 825–829, 2020.

[87] L. Zhang, G. Karakasidis, A. Odnoblyudova, L. Dogruel, Y. Tian, and A. Jung, "Explainable empirical risk minimization," *Neural Computing*

and Applications, vol. 36, no. 8, pp. 3983–3996, 2024. [Online]. Available: https://doi.org/10.1007/s00521-023-09269-3

[88] 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

[89] P. Amin, N. R. Anikireddypally, S. Khurana, S. Vadakkemadathil, and W. Wu, "Personalized health monitoring using predictive analytics," in *2019 IEEE Fifth International Conference on Big Data Computing Service and Applications (BigDataService)*, 2019, pp. 271–278.

[90] R. B. Ash, *Probability and Measure Theory*, 2nd ed. New York: Academic Press, 2000.

[91] P. R. Halmos, *Measure Theory*. New York: Springer, 1974.

[92] P. Billingsley, *Probability and Measure*, 3rd ed. New York: Wiley, 1995.

[93] C. Dwork and A. Roth, "The algorithmic foundations of differential privacy," *Foundations and Trends® in Theoretical Computer Science*, vol. 9, no. 3–4, pp. 211–407, 2014. [Online]. Available: http://dx.doi.org/10.1561/0400000042

[94] S. Asoodeh, J. Liao, F. P. Calmon, O. Kosut, and L. Sankar, "A Better Bound Gives a Hundred Rounds: Enhanced Privacy Guarantees via f-Divergences," *arXiv e-prints*, p. arXiv:2001.05990, Jan. 2020.

[95] I. Mironov, "Rényi differential privacy," in *2017 IEEE 30th Computer Security Foundations Symposium (CSF)*, 2017, pp. 263–275.

[96] S. M. Kay, *Fundamentals of statistical signal processing. Vol. 2., Detection theory*, ser. Prentice-Hall signal processing series. Upper Saddle River, NJ: Prentice-Hall PTR, 1998.

[97] P. Kairouz, S. Oh, and P. Viswanath, "The composition theorem for differential privacy," in *Proceedings of the 32nd International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, F. Bach and D. Blei, Eds., vol. 37. Lille, France: PMLR, 07–09 Jul 2015, pp. 1376–1385. [Online]. Available: https://proceedings.mlr.press/v37/kairouz15.html

[98] Q. Geng and P. Viswanath, "The optimal noise-adding mechanism in differential privacy," *IEEE Transactions on Information Theory*, vol. 62, no. 2, pp. 925–951, 2016.

[99] H. Shu and H. Zhu, "Sensitivity analysis of deep neural networks," in *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence*, ser. AAAI'19/IAAI'19/EAAI'19. AAAI Press, 2019. [Online]. Available: https://doi.org/10.1609/aaai.v33i01.33014943

[100] R. I. Busa-Fekete, A. Munoz Medina, U. Syed, and S. Vassilvitskii, "Label differential privacy and private training data release," in *Proceedings of the 40th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, A. Krause, E. Brunskill, K. Cho, B. Engelhardt, S. Sabato, and J. Scarlett, Eds., vol.

202. PMLR, 23–29 Jul 2023, pp. 3233–3251. [Online]. Available: https://proceedings.mlr.press/v202/busa-fekete23a.html

[101] B. Balle, G. Barthe, and M. Gaboardi, "Privacy amplification by subsampling: tight analyses via couplings and divergences," in *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, ser. NIPS'18. Red Hook, NY, USA: Curran Associates Inc., 2018, pp. 6280–6290.

[102] P. Cuff and L. Yu, "Differential privacy as a mutual information constraint," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16. New York, NY, USA: Association for Computing Machinery, 2016, pp. 43–54. [Online]. Available: https://doi.org/10.1145/2976749.2978308

[103] A. Makhdoumi, S. Salamatian, N. Fawaz, and M. Médard, "From the information bottleneck to the privacy funnel," in *2014 IEEE Information Theory Workshop (ITW 2014)*, 2014, pp. 501–505.

[104] A. Turner, D. Tsipras, and A. Madry, "Clean-label backdoor attacks," 2019. [Online]. Available: https://openreview.net/forum?id=HJg6e2CcK7

[105] T. M. Cover and J. A. Thomas, *Elements of Information Theory*, 2nd ed. New Jersey: Wiley, 2006.

[106] 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.

[107] 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.

[108] P. Halmos, *Naive set theory*. Springer-Verlag, 1974.

[109] 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.

[110] D. Gujarati and D. Porter, *Basic Econometrics*. Mc-Graw Hill, 2009.

[111] Y. Dodge, *The Oxford Dictionary of Statistical Terms*. Oxford University Press, 2003.

[112] B. Everitt, *Cambridge Dictionary of Statistics*. Cambridge University Press, 2002.

[113] R. T. Rockafellar, *Network Flows and Monotropic Optimization*. Athena Scientific, Jul. 1998.

[114] U. von Luxburg, "A tutorial on spectral clustering," *Statistics and Computing*, vol. 17, no. 4, pp. 395–416, Dec. 2007.

[115] A. Y. Ng, M. I. Jordan, and Y. Weiss, "On spectral clustering: Analysis and an algorithm," in *Adv. Neur. Inf. Proc. Syst.*, 2001.

[116] R. Gray, *Probability, Random Processes, and Ergodic Properties*, 2nd ed. New York: Springer, 2009.

[117] A. Lapidoth, *A Foundation in Digital Communication*. New York: Cambridge University Press, 2009.

[118] S. Bubeck, "Convex optimization. algorithms and complexity." in *Foundations and Trends in Machine Learning*. Now Publishers, 2015, vol. 8.