Lecture Notes

CS-E4740 Federated Learning

offered in spring 2023 at Aalto University,

`https://fitech.io/en/`

ROUGH DRAFT - DO NOT DISTRIBUTE FURTHER!

Comments are warmly welcome at `alex.jung@aalto.fi` and will be acknowledged properly.

Dipl.-Ing. Dr.techn. Alexander Jung*†

February 3, 2023

Abstract

The course **CS-E4740 Federated Learning** discusses theory and algorithms for the federated learning from decentralized collections of local datasets. Federated learning (FL) methods collaboratively train tailored (" personalized") models for each local dataset. These tailored models are coupled via a network structure that reflects statistical similarities between local datasets. We use this network structure to

*

†

construct a regularization term which we refer to as GTV minimization. It provides a flexible design principle for FL algorithms and includes as important special cases vertical FL, horizontal FL and clustered FL. We obtain FL algorithms by applying established distributed optimization methods to solve GTVMin.

Contents

Lists of Symbols

$\mathcal{G} = \big(\mathcal{V}, \mathcal{E}\big)$	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 local model. It might also be useful to think of node i as a small computer that can collect data and execute computations for model training.
$\mathcal{D}^{(i)}$	The local dataset $\mathcal{D}^{(i)}$ at node $i \in \mathcal{V}$.
m_i	The number of data points (sample size) contained in the local dataset $\mathcal{D}^{(i)}$ at node $i \in \mathcal{V}$.
$\mathbf{x}$	The feature vector $\mathbf{x} = \big(x_1, \ldots, x_d\big)^T$ of a generic 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.
$\mathbf{x}^{(i,r)}$	The feature vector of the r-th data point in the local dataset $\mathcal{D}^{(i)}$
$\mathbf{v} \in \mathbb{R}^d$	Some numeric vector (one-dimensional array) of length d.

$\|\mathbf{w}\|_2$	The Euclidean norm $\|\mathbf{w}\|_2 := \sqrt{\sum_{j=1}^{d} w_j^2}$ of the d-dimensional vector $\mathbf{w}$.
h	A hypothesis map $h : \mathcal{X} \to \mathcal{Y}$ that reads in the features $\mathbf{x}$ of a data point and delivers a prediction $h(\mathbf{x})$ 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 maps $h : \mathcal{X} \to \mathcal{Y}$.
$\mathcal{H}$	A hypothesis space which consists of hypothesis maps h.

1 Introduction

Many important application domains generate decentralized collections of local datasets that are related via an intrinsic network structure [2]. Two timely application domains that generate such networked data are (i) the high-precision management of pandemics and (ii) the Internet of Things (IoT) [3, 4]. Here, local datasets are generated by smartphones, wearables or other IoT devices [5, 6]. These local datasets are related via physical contact networks, social networks [7], co-morbidity networks [8], or communication networks [9].

FL is an umbrella term for distributed optimization techniques to train machine learning (ML) models from decentralized collections of local datasets [10–14]. These methods implement computations, such as gradient computations (see Section 5) for ML model training, at the location of data generation. This design philosophy is different from a naive approach of first collecting local datasets at a single location at which a ML model is trained. Distributed training of (several different) ML models at the spatial location of data generation can be beneficial in several aspects [15]:

- **Privacy.** FL methods are appealing for applications involving sensitive data (such as healthcare) as they do not require the exchange of raw data but only model (parameter) updates [12, 13]. By sharing only model updates, FL methods are considered privacy-friendly in the sense of not leaking (too much) sensitive information in local datasets (see Section 11).

- **Robustness.** By relying on decentralized data and and computa-

tion, FL methods offer robustness against hardware failures (such as "stragglers") and data poisonings (see Section 12).

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

- **Trading Computation against Communication.** Consider a FL application where local datasets are generated by low-complexity devices at remote locations that cannot be easily accessed. The cost of communicating raw local datasets to some central unit (which then trains a single global ML model) might be much higher than the computational cost incurred by using the low-complexity devices to (partially) train ML models [16].

- **Personalization.** FL can be used to train personalized ML models for collections of local datasets, which might be generated by smartphones (and their users) [17]. A key challenge for ensuring personalization is the heterogeneity of local datasets [18, 19]. 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 parameters for a local model. This course discusses FL methods to train personalized models via combining the information carried in

decentralized and heterogeneous data (see Section 8.3 and Section 8.4).

1.1 Related Courses

In what follows we list some related courses offered at Aalto University.

- **CS-EJ3211 - Machine Learning with Python.** Teaches the application of basic ML methods using the Python package (library) `scikit-learn` [20]. CS-E4740 couples a network of basic ML methods using regularization techniques to obtain tailored (personalized) ML models for local datasets. This coupling is required to adaptive pool local datasets obtain sufficiently large training sets for the personalized ML models.

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

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

- **MS-C2105 - Introduction to Optimization.** This course teaches basic optimisation theory and how to model applications as (linear, integer, and non-linear) optimization problems. CS-E4740 uses optimization

theory and methods to formulate FL problems (see Section 7) and design FL methods (see Section 9).

- **ELEC-E5424 - Convex Optimization.** This course teaches advanced optimisation theory for the important class of convex optimization problems [21]. Convex optimization theory and methods can be used for the study and design of FL algorithms.

1.2 Outline of the Course

This notes are divided into three parts:

- Part I discusses basic component and principles of (non-federated) ML: Section 2 introduces data, models and loss functions as three main components of ML. Section 3 explains how these components can be combined via empirical risk minimization (ERM). The regularization of ERM, via each of its three main components, is then discussed in Section 4. We then explain when and how to solve regularized ERM via simple gradient descent methods in Section 5.

- Part II shows how to couple (the training of) tailored (personalized) ML models for decentralized data with an intrinsic network structure: Section 6 introduces an empirical graph as a representation for collections of local datasets and corresponding tailored models. The undirected and weighted edges of the empirical graph represent statistical similarities between local datasets. Section 7 uses the similarity structure to formulate GTVMin as a main design principle for FL. We then show how main flavours of FL are obtained as special cases of GTVMin

in Section 8. Section 9 applies well-known optimization methods to GTVMin, resulting in FL algorithms for distributed training of tailored ("personalized") models.

- Part III discusses requirements and design aspects for trustworthy FL: Section 10 enumerates seven key requirements for trustworthy artificial intelligence (AI) put forward by the European Union. We then discuss privacy and its protection by FL algorithms in Section 11. Section 12.2 discusses the vulnerability of FL methods against backdoor attacks.

1.3 Acknowledgements

The author is grateful for feedback on the lecture notes received from: Dr. Shamsiiat Abdurakhmanova, Alexander Pavlyuk and Ariana Marta. Some of the figures have been prepared with the help of Dr. Yu Tian.

1.4 Exercises

Exercise 1.1. Parallel Computing (see Sec. 1.2.3. of [22]). Assume that you have a single computer that is able to sum two numbers per second. You can assume that the computer has sufficient storage memory which can be read from and written to with high speed. How long does the computer take to compute the sum of 100 numbers $\mathcal{D} = \{x^{(1)}, \ldots, x^{(100)}\}$.

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

Exercise 1.2. Communication Bottleneck. Consider two laptops, each having a webcam taking pictures with 1080×1080 pixels. These two laptops are connected via a wireless `Bluetooth` connection with average bitrate of $1 \cdot 10^6$ bits/s [23]. What is the maximum rate at which one laptop can record images and transmit it to the other laptop? You can ignore any compression techniques and assume that images are stored as bitmaps with each pixel encoded by three bytes (red, green and blue intensities on scale $0, \ldots, 255$)

Part I

Plain Old Machine Learning

2 Three Components of ML

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

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

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

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

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

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

The features $\mathbf{x}$ are low-level properties that we can measure or compute easily in an automated fashion. Examples for such low-level properties are physical quantities that can be measured with low-cost hardware, such as temperature or humidity. The feature space $\mathcal{X}$ collects all possible values that the features $\mathbf{x} \in \mathcal{X}$ of a data point can take on. This course focuses on ML methods that use an Euclidean space $\mathbb{R}^d$ as the feature space.

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

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

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

Our definition of data points as elementary information-carrying objects is quite abstract and therefore flexible. Indeed, the definition of what we

consider as data points is an important design choice. As an example, we could use data points to represent time periods such as days. One data point would then represent the day 12-Mar-2020. Another data point represents the day 12-Apr-2019. Instead of entire days, we could also define data points as shorter time-periods such as hours or 10-minute intervals.

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

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

- **(learning task)** What is the ultimate quantity of interest, i.e., which properties should we use the label of data point? For the above example, we could define the label as the average temperature during the next two days. Another choice for the label y would be the average temperature during the next ten days.

The design choices for the data points, their features and label must take into account statistical aspects, computational aspects and explainability of the resulting ML method. For example, using a large number of features will typically result in a higher computational complexity of the resulting ML method. On the other hand, a large number of features might be required to learn an accurate hypothesis. To ensure explainability of the resulting ML method, we might prefer using few interpretable features [24].

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

The overall goal of ML is to find or learn a hypothesis map h that allows to predict the label of a data point from its features. Practical ML methods can search and evaluate only a (tiny) subset of the set $\mathcal{Y}^{\mathcal{X}}$ that contains all possible hypothesis maps. This subset of computationally feasible ("affordable") hypothesis maps is referred to as the hypothesis space or model underlying a ML method.

As depicted in Figure 1, ML methods typically use a hypothesis space $\mathcal{H}$ that is a tiny subset of $\mathcal{Y}^{\mathcal{X}}$. Similar to the features and labels used to characterize data points, also the hypothesis space underlying a ML method is a design choice. The design choice for the hypothesis space involves a trade-off between computational complexity and accuracy of the resulting ML methods [1, Ch. 2].

The hypothesis space should be large enough to contain at least one hypothesis that allows to accurately predict the label from the features of a data point. However, it should not be too large such that ML methods can efficiently search it for a useful hypothesis. Another limitation for the the size or dimension of the hypothesis space is the amount of available data (see Section 3.3). Beside computational aspects and statistical aspects, a third design aspect for the model is the explainability of the resulting ML method [25, 26].

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

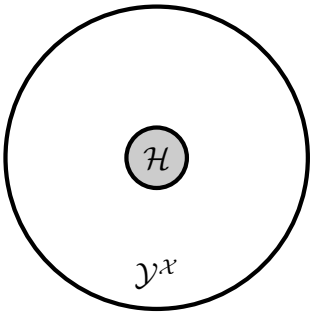

Figure 1: The hypothesis space $\mathcal{H}$ is a (typically very small) subset of the (typically very large) set $\mathcal{Y}^{\mathcal{X}}$ of all possible maps from the feature space space $\mathcal{X}$ to label space $\mathcal{Y}$ of a ML problem or method.

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

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

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

$$\mathcal{H}^{(d)} := \left\{ h(\mathbf{x}) = \mathbf{w}^T \mathbf{x} : \mathbf{w} \in \mathbb{R}^d \right\}. \tag{1}$$

Note that (79) defines an entire family of hypothesis spaces, which is indexed by the number d of features that are linearly combined to form the prediction $h(\mathbf{x})$. The design choice of d is guided by computational aspects (smaller d means less computation), statistical aspects (larger d might reduce prediction error) and interpretability (favouring a linear model with few features).

Each individual hypothesis in the linear model $\mathcal{H}^{(d)}$ is fully specified by a weight vector $\mathbf{w} \in \mathbb{R}^d$, which we indicate by the notation $h^{(\mathbf{w})}(\mathbf{x}) := \mathbf{w}^T \mathbf{x}$. Thus, the linear model (79) is an example for a parametrized model. Another

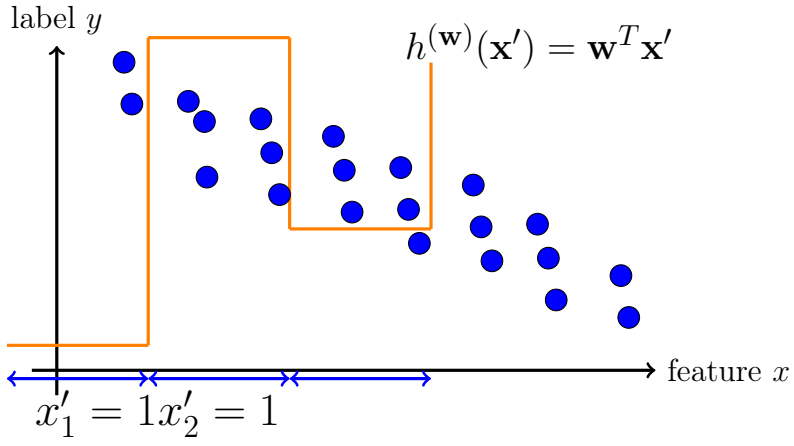

Figure 2: We can extend a linear model (79) by replacing the original feature x of a data point with transformed features $\mathbf{x}' = \boldsymbol{\Phi}(\mathbf{x})$. These transformed features are obtained by applying a feature map $\boldsymbol{\Phi}$ to the original features.

example for a parametrized model is an artificial neural network (ANN) [1, Ch. 2, 3].

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

Non-Numeric Features. We can use the linear model (79) only for data points whose features are numeric vectors $\mathbf{x} = (x_1, \ldots, x_d)^T \in \mathbb{R}^d$. In some application domains, such as natural language processing [27], there is

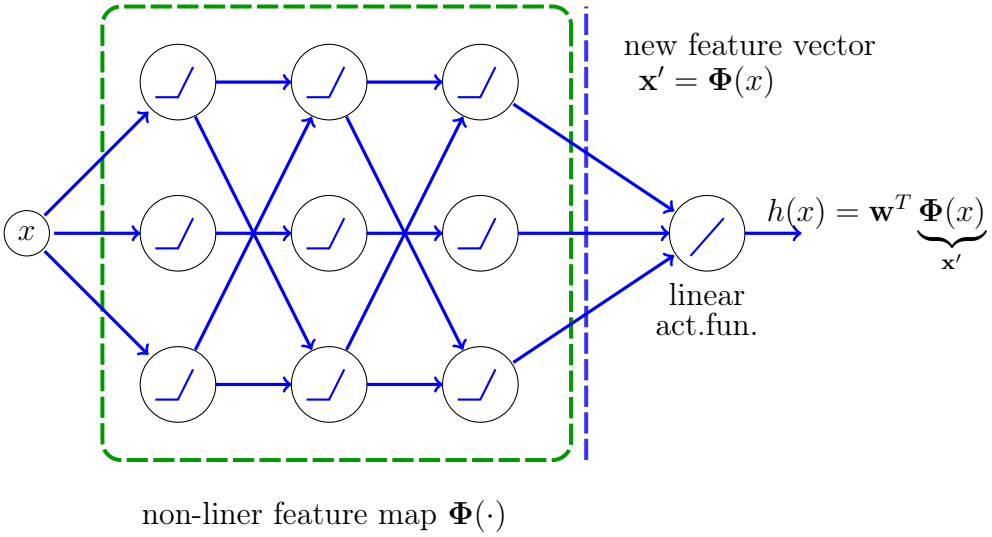

Figure 3: The hypothesis space spanned by an ANN (whose output layer is linear) coincides with the linear model applied to the activations $\mathbf{\Phi}(x)$ of the penultimate layer. We can interpret these activations as transformed features that are obtained by applying the input layers to the original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

no obvious natural choice for numeric features. However, since ML methods based on the hypothesis space (79) can be implemented efficiently using widely available soft and hardware, it might be useful to construct numerical features even for non-numeric data. For text data, there has been significant progress recently on methods that map a human-generated text into sequences of vectors (see [27, Chap. 12] for more details). Moreover, [1, Sec. 9.3] shows how to construct numeric features for data points that are related by some notion of similarity.

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

discuss effect of using Huber loss instead of squared error loss; illustrate robustness of Huber regression against outliers; these outliers might be planted intentionally as a form of data poisoning attacks;

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

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

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

The loss value $L\big((\mathbf{x}, y), h\big)$ quantifies the discrepancy between the true label y and the predicted label $h(\mathbf{x})$. A small (close to zero) value $L\big((\mathbf{x}, y), h\big)$ indicates a small discrepancy between predicted label and true label of a

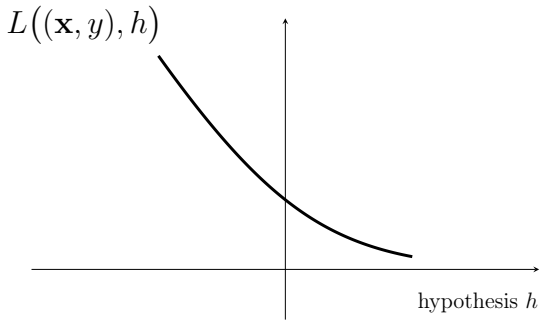

Figure 4: Some loss function $L\big((\mathbf{x},y),h\big)$ for a fixed data point, with features $\mathbf{x}$ and label y, and varying hypothesis h. ML methods try to find (learn) a hypothesis that incurs minimum loss.

data point. Figure 4 depicts a loss function for a given data point, with features $\mathbf{x}$ and label y, as a function of the hypothesis $h \in \mathcal{H}$. Note that Figure 4 needs to be interpreted as a cartoon as we use the horizontal axis to represent a hypothesis space (whose elements or hypothesis maps). We could make Figure 4 more rigorous, by using the horizontal axis to represent single numeric parameters of a hypothesis map such as its value $h(\mathbf{x})$ for the features of the considered data point. Another option would be to use the single weight of a linear model $\mathcal{H}^{(1)}$ as the horizontal axis.

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

- computational aspects,
- statistical aspects,

- trustworthiness of the resulting ML method.

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

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

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

A third aspect for the choice of loss function is its explainability. Section 2.3.2 discusses loss functions to measure the quality of binary classifier. It seems natural to measure the quality of a binary classifier by the average number of wrongly classified data points, which is precisely the average 0/1 loss (4) (see Section 2.3.2).

In contrast to its appealing interpretation as error-rate, the computational aspects of the average 0/1 loss (which is equivalent to accuracy) are less pleasant. Minimizing the average 0/1 loss to learn an accurate hypothesis amounts to a non-convex and non-smooth optimization problem which is typically computationally more challenging compared to minimizing a convex function.

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

The summarize, computational aspects, statistical aspects and explainability of ML methods typically result in conflicting design goals for a loss function. A loss function that has favourable statistical properties might incur a high computational complexity of the resulting ML method. Loss functions that result in computationally efficient ML methods might not allow for an easy interpretation (e.g., what does it mean intuitively if the logistic loss of a binary classifier is 10^{-1}?). It might therefore be useful to use different loss functions for the search of a good hypothesis and for its final evaluation (see Figure 5).

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

rate. The loss function used for the final performance evaluation of a learnt hypothesis is sometimes referred to as metric.

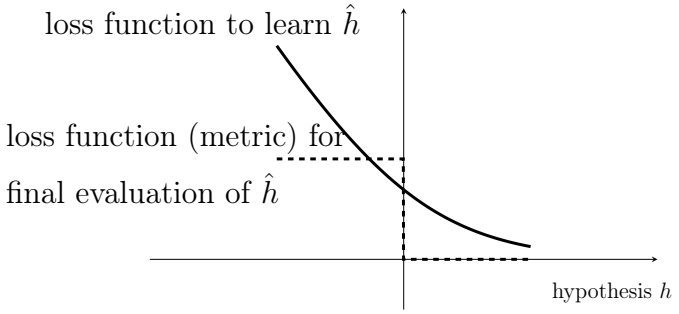

Figure 5: Two different loss functions for a given data point and varying hypothesis h. One of these loss functions (solid curve) is used to learn a good hypothesis by minimizing the loss. We use another loss function (dashed curve) to evaluate the performance of the learnt hypothesis $\hat{h}$.

2.3.1 Loss Functions for Numeric Labels

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

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

The squared error loss (2) depends on the features $\mathbf{x}$ only via the predicted label value $\hat{y} = h(\mathbf{x})$. We can evaluate the squared error loss solely using the prediction $h(\mathbf{x})$ and the true label value y. Besides the prediction $h(\mathbf{x})$, no other properties of the features $\mathbf{x}$ influence the value of the squared error loss.

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

$$L\big(\big(\mathbf{x},y\big),h\big) := \big|h(\mathbf{x}) - y\big|. \tag{3}$$

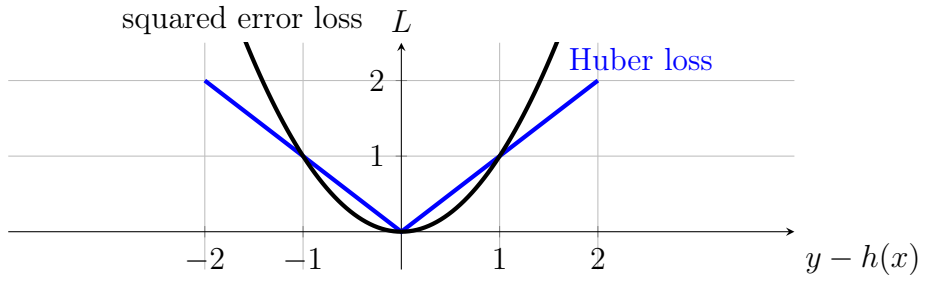

Figure 6: Two widely used loss functions to measure how well a hypothesis predicts a numeric label are the squared error loss (2) and the Huber loss (3). Note that, for a given hypothesis h, we can evaluate the loss function only if we know the features $\mathbf{x}$ and the label y of the data point.

The squared error loss (2) has appealing computational and statistical properties. For linear predictor maps $h(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$, the squared error loss is a convex and differentiable function of the parameter vector $\mathbf{w}$. This allows, in turn, to efficiently search for the optimal linear predictor using efficient iterative optimization methods (see Section 5). The squared error loss also has a useful interpretation in terms of a probabilistic model for the features and labels: Minimizing the squared error loss is equivalent to maximum likelihood estimation within a linear Gaussian model [28, Sec. 2.6.3].

Another loss function used in regression problems is the absolute error loss (3), which is a special case of the Huber loss [1, Sec. 3.3]. It can be shown that ML methods using absolute error loss are more robust against a few outliers in the training set compared to methods using squared error loss (see [1, Sec. 3.3.]). The improved robustness comes at the expense of increased computational complexity of minimizing the non-differentiable loss (3) compared to the convex and differentiable squared error loss (2).

2.3.2 Loss Functions for Categorical Labels

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

It seems that we cannot use linear hypothesis maps $h^{(\mathbf{w})}(\mathbf{x}) := \mathbf{w}^T\mathbf{x}$ (see (79)) for a classification problem. Indeed, the function value $h(\mathbf{x})$ is a real-number and (most likely) does not belong to the label space $\mathcal{Y} = \{-1, 1\}$. However, a simple post-processing step turns a linear hypothesis $h^{(\mathbf{w})}$ into a binary classifier: We classify a data point as $\hat{y} = 1$ if $h^{(\mathbf{w})}(\mathbf{x}) > 0$ and $\hat{y} = -1$ otherwise. While the sign of $h^{(\mathbf{w})}(\mathbf{x})$ determines the predicted label $\hat{y} \in \{-1, 1\}$, the absolute value $|h^{(\mathbf{w})}(\mathbf{x})|$ severs as a measure for the confidence about this prediction. We will abuse notation and also refer to the linear map $h^{(\mathbf{w})}(\mathbf{x})$ as a binary classifier.

In principle, we can measure the quality of a hypothesis when used to classify data points using the squared error loss (2). However, the squared

error loss is a poor quality measure for a binary classifier. Figure 7 illustrates how the squared error loss of a linear hypothesis can be (highly) misleading for evaluating the performance of a binary classifier.

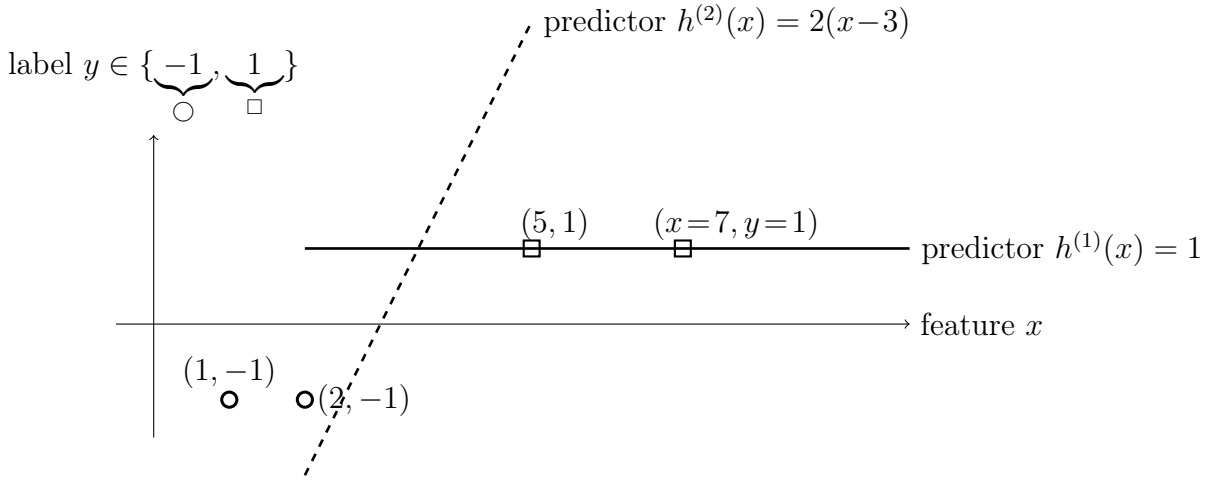

Figure 7: A training set consisting of four data points with binary labels $\hat{y}^{(r)} \in \{-1, 1\}$. Minimizing the squared error loss (2) would prefer the (poor) classifier $h^{(1)}$ over the (reasonable) classifier $h^{(2)}$.

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

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

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

classifier $h^{(2)}$ is much larger than the squared error loss incurred by the (poor) classifier $h^{(1)}$. The squared error loss is typically a bad choice for assessing the quality of a hypothesis map that is used for classifying data points into different categories.

Generally speaking, we want the loss function to punish (deliver large values for) a hypothesis that results in a wrong classification ($\hat{y} \neq y$) with high confidence ($|h(\mathbf{x})|$ is large) On the others hand, a useful loss function should not punish (deliver small values for) a hypothesis that results a correct classification ($\hat{y} = y$) with high confidence ($|h(\mathbf{x})|$ is large). However, by its very definition, the squared error loss (2) yields large values if the confidence $|h(\mathbf{x})|$ is large, no matter if the resulting classification (obtained after thresholding) is correct or wrong.

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

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

$$L\big((\mathbf{x}, y), h\big) := \begin{cases} 1 & \text{if } y \neq \hat{y} \\ 0 & \text{else,} \end{cases} \quad \text{with } \hat{y} = \begin{cases} 1 & \text{if } h(\mathbf{x}) \geq 0 \\ 0 & \text{else.} \end{cases} \tag{4}$$

Figure 8 illustrates the 0/1 loss (4) for a data point with features $\mathbf{x}$ and label

$y=1$ as a function of the value $h(\mathbf{x})$. The 0/1 loss is equal to zero if the hypothesis yields a correct classification $\hat{y} = y$. For a wrong classification $\hat{y} \neq y$, the 0/1 loss is equal to one.

The 0/1 loss (4) is conceptually appealing when data points are interpreted as realizations of i.i.d. random variables (RVs) with common probability distribution $p(\mathbf{x}, y)$ (see Section 3.1). Given m realizations $\big(\mathbf{x}^{(1)}, y^{(1)}\big), \ldots, \big(\mathbf{x}^{(m)}, y^{(m)}\big)$ of these i.i.d. RVs, with high probability

$$(1/m) \sum_{r=1}^{m} L\big((\mathbf{x}^{(r)}, y^{(r)}), h\big) \approx p(y \neq h(\mathbf{x})) \tag{5}$$

for sufficiently large sample size m.

A precise formulation of the approximation (5) can be obtained from the law of large numbers [29, Section 1]. We can apply the law of large numbers since the loss values $L\big((\mathbf{x}^{(r)}, y^{(r)}), h\big)$ are realizations of i.i.d. RVs. It is customary to indicate the average 0/1 loss of a hypothesis h indirectly via the accuracy $1 - (1/m) \sum_{r=1}^{m} L\big((\mathbf{x}^{(r)}, y^{(r)}), h\big)$.

In view of (5), the 0/1 loss (4) seems a very natural choice for assessing the quality of a classifier if our goal is to enforce correct classifications $\hat{y} = y$. This appealing statistical aspects of the 0/1 loss comes at the cost of a high computational complexity. Indeed, for a given data point $(\mathbf{x}, y)$, the 0/1 loss (4) is non-convex and non-differentiable when viewed as a function of the hypothesis h. Thus, ML methods that use the 0/1 loss to learn a hypothesis require advanced optimization methods to solve the resulting learning problem (see [1, Sec. 3.8.]).

To avoid the non-convexity of the 0/1 loss (4) we can approximate it by a convex "surrogate" loss function. One such approximation of the 0/1 loss is

the hinge loss

$$L\big((\mathbf{x},y),h\big) := \max\{0, 1 - yh(\mathbf{x})\}. \tag{6}$$

Figure 8 depicts the hinge loss (6) as a function of the hypothesis $h(\mathbf{x})$. The hinge loss (6) becomes minimal (zero) for a correct classification ($\hat{y} = y$) with sufficient confidence $h(\mathbf{x}) \geq 1$. For a wrong classification ($\hat{y} \neq y$), the hinge loss increases monotonically with the confidence $|h(x)|$ in the wrong classification. While the hinge loss avoids the non-convexity of the 0/1 loss, it still is a non-differentiable function of $h(\mathbf{x})$. Finding the optimizing of a non-differentiable loss function is typically harder, requiring more iterations of iterative methods, than optimizing a differentiable loss function [30, 31].

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

$$L\big((\mathbf{x},y),h\big) := \log(1 + \exp(-yh(\mathbf{x}))). \tag{7}$$

The logistic loss (7) is used in logistic regression [1, Ch. 3] to measure the usefulness of a linear hypothesis map $h^{(\mathbf{w})}(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$.

Figure 8 depicts the logistic loss (7) as a function of the hypothesis $h(\mathbf{x})$. The logistic loss (7) is a convex and differentiable function of $h(\mathbf{x})$. For a correct classification ($\hat{y}=y$), the logistic loss (7) decreases monotonically with increasing confidence $h(\mathbf{x})$. For a wrong classification ($\hat{y} \neq y$), the logistic loss increases monotonically with increasing confidence $|h(\mathbf{x})|$ in the wrong classification.

Both, the hinge loss (6) as well as the logistic loss (7) are convex functions of the weigh vector $\mathbf{w} \in \mathbb{R}^d$ for a linear hypothesis map $h^{(\mathbf{w})}(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$. In contrast to the hinge loss, the logistic loss (7) is also a differentiable function of the $\mathbf{w}$.

The convex and differentiable logistic loss function can be minimized using gradient-based methods (see Section 5). In contrast, we cannot use gradient-based methods to minimize the hinge loss since it is not differentiable (it does not have a gradient everywhere). However, we can apply a generalization of gradient descent (GD) which is known as subgradient descent [31]. Subgradient descent is obtained from GD by generalizing the concept of a gradient to that of a subgradient.

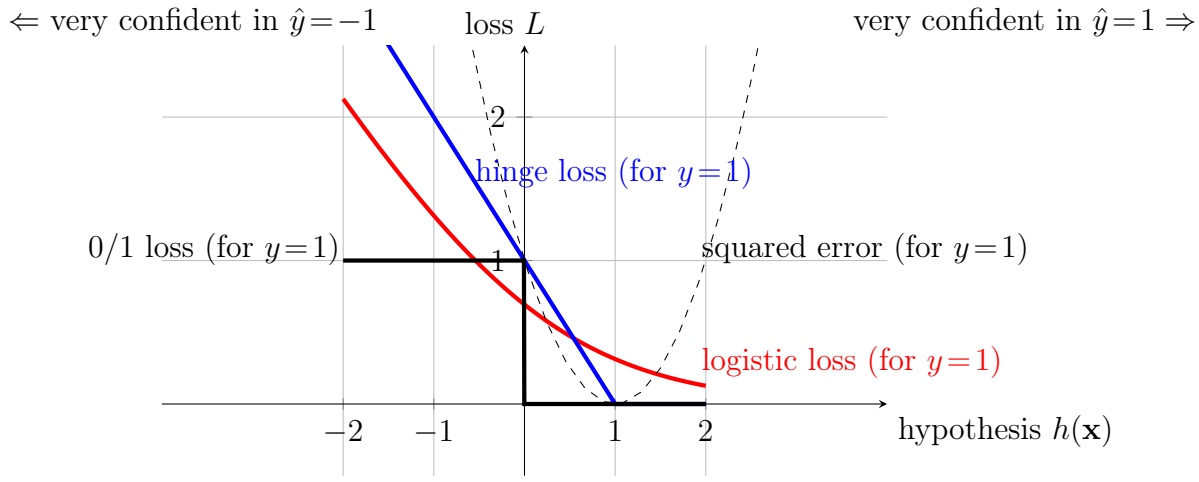

Figure 8: The solid curves depict three widely-used loss functions for measuring the performance of a binary classifier $h^{(\mathbf{w})}(\mathbf{x})$ with label space $\mathcal{Y} = \{-1, 1\}$. A data point with features $\mathbf{x}$ and label $y = 1$ is classified as $\hat{y} = 1$ if $h(\mathbf{x}) \geq 0$ and classified as $\hat{y} = -1$ if $h(\mathbf{x}) < 0$. We can interpret the absolute value $|h(\mathbf{x})|$ as the confidence in the classification $\hat{y}$. The more confident we are in a correct classification ($\hat{y}=y=1$), i.e, the more positive $h(\mathbf{x})$, the smaller the loss. Note that each of the three loss functions for binary classification tends monotonically towards 0 for increasing $h(\mathbf{x})$. The dashed curve depicts the squared error loss (2), which increases for increasing $h(\mathbf{x})$.

2.4 Exercises

Exercise 2.1. Data Imputation. (10 points) Consider the dataset available under `https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_wine.html#sklearn.datasets.load_wine`. This dataset consists of data points that represent wine samples. Each data point is characterized by $d = 13$ features $x_1, \ldots, x_d$. Let us assume that the feature value x_1 is missing for several data points. We could then define feature x_1 as the label of a data point and try to predict it from the remaining features. Try to find a coefficients $w_2, \ldots, w_{13}$ such that $\underbrace{y}_{x_1} \approx \sum_{j=2}^{d} w_j x_j$.

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

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

Exercise 2.4. Perfect Fit. For a prescribed sample size m, can you craft m data points characterized by feature vectors $\mathbf{x}^{(r)} \in \mathbb{R}^d$ and binary label $y^{(r)} \in \{-1, 1\}$ such that it is possible to find a linear hypothesis $h^{(\mathbf{w})}(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$ that correctly predicts the label of each data point in $\mathcal{D}$, i.e.,

$$\begin{aligned} &h^{(\mathbf{w})}(\mathbf{x}^{(r)}) \geq 0 \text{ for each data point with } y^{(r)} = 1, \text{ and} \\ &h^{(\mathbf{w})}(\mathbf{x}^{(r)}) < 0 \text{ for each data point with } y^{(r)} = -1. \end{aligned} \tag{8}$$

Exercise 2.5. Formulas for Logistic loss. Modify the formula (7) for the logistic loss, which is valid only for label space $\mathcal{Y} = \{-1, 1\}$, to measure

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

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

3 A Design Principle for ML

???? ERM for simple linreg; closed-form solution and GD iterations; ????

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

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

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

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

3.1 The i.i.d. Assumption

Consider a ML application that involves a dataset

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

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

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

$$\big(\mathbf{x}^{(r)}, y^{(r)}\big) \overset{\text{i.i.d.}}{\sim} p(\mathbf{x}, y). \tag{9}$$

The probabilistic model (9) enables us to measure the quality of a hypothesis $h \in \mathcal{H}$ by the expected loss or risk [32]

$$\bar{L}\big(h\big) := \mathbb{E}\big\{L\big((\mathbf{x}, y), h\big)\big\} = \int_{\mathbf{x},y} L\big((\mathbf{x}, y), h\big) dp(\mathbf{x}, y). \tag{10}$$

The risk (10) of h is the expected value of the loss $L\big((\mathbf{x}, y), h\big)$ incurred when applying the hypothesis h to (the realization of) a random data point with features $\mathbf{x}$ and label y. Note that the computation of the risk (10) requires the joint probability distribution $p(\mathbf{x}, y)$ of the (random) features and label of data points. It seems reasonable to learn a hypothesis h^* that incurs minimum risk,

$$h^* \in \operatorname*{argmin}_{h \in \mathcal{H}} \underbrace{\bar{L}\big(h\big)}_{=\mathbb{E}\left\{L\left((\mathbf{x},y),h\right)\right\}}. \tag{11}$$

Any hypothesis that solves (11), i.e., that incurs minimal risk, is referred to as a Bayes estimator [32, Chapter 4].

Thus, once we know the underlying probability distribution, the only challenge for learning the optimal hypothesis is the efficient (numerical) solution of the optimization problem (10). Efficient methods to solve the optimization problem (10) include variational methods and random sampling (Monte Carlo) methods [32, 33].

If we do not know the probability distribution underlying the data points, we cannot compute the risk and, in turn, cannot use (11) as a learning

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

$$\widehat{L}\big(h|\mathcal{D}\big) = (1/m) \sum_{r=1}^{m} L\big((\mathbf{x}^{(r)}, y^{(r)}), h\big). \tag{12}$$

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

If the data points used to compute the empirical risk (12) are modelled as realizations of i.i.d. RVs whose common probability distribution is $p(\mathbf{x}, y)$, basic results of probability theory tell us that

$$\mathbb{E}\big\{L\big((\mathbf{x}, y), h\big)\big\} \approx \underbrace{(1/m) \sum_{r=1}^{m} L\big((\mathbf{x}^{(r)}, y^{(r)}), h\big)}_{\stackrel{(12)}{=} \widehat{L}\big(h|\mathcal{D}\big)} \text{ for sufficiently large } m. \tag{13}$$

The approximation error in (13) can be quantified precisely by different variants of the law of large numbers [29, 34, 35] or large-deviation bounds [36].

3.2 Empirical Risk Minimization

Many ML methods are motivated by (13) which suggests that a hypothesis with small empirical risk (12) will also result in a small expected loss. In theory, the minimum expected loss is achieved by the Bayes estimator of the label y, given the features $\mathbf{x}$. However, to actually compute the optimal estimator we need the (joint) probability distribution $p(\mathbf{x}, y)$ of features $\mathbf{x}$

and label y. This joint probability distribution is typically unknown and must be estimated or approximated from observed data points.

Many practical ML methods are numerical optimization algorithms to solve ERM

$$\widehat{h} := \operatorname*{argmin}_{h \in \mathcal{H}} \underbrace{(1/m) \sum_{r=1}^{m} L\big(\big(\mathbf{x}^{(r)}, y^{(r)}\big), h\big)}_{\widehat{L}\big(h|\mathcal{D}\big)} . \tag{14}$$

The objective function of ERM is the empirical risk (12) which, in turn, approximates the risk $\bar{L}\big(h\big)$ via the law of large numbers (13). Note that ERM (14) is parametrized by three components: the dataset $\mathcal{D}$, the hypothesis space $\mathcal{H}$ and loss function $L\big(\cdot, \cdot\big)$. The Python library `scikit-learn` provides implementations of (14) for different choices for the hypothesis space $\mathcal{H}$ and loss function $L\big(\cdot, \cdot\big)$ [20].

The objective function of ERM depends on the dataset

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

In general, $\widehat{L}\big(h|\mathcal{D}\big)$ (viewed as a function of $h \in \mathcal{H}$) depends on each of the feature vectors $\mathbf{x}^{(r)}$ and labels $y^{(r)}$ in $\mathcal{D}$. By the i.i.d. assumption (9), each data point $\big(\mathbf{x}^{(r)}, y^{(r)}\big)$ in $\mathcal{D}$ is a realization of a RV with (unknown) probability distribution $p(\mathbf{x}, y)$. This probability distribution crucially affects the computational aspects and statistical aspects of ML methods that use ERM (14).

In general, we do not have (much) control over the validity of the i.i.d. assumption, let alone the underlying probability distribution $p(\mathbf{x}, y)$, We can only test if the i.i.d. assumption is likely to be valid [37]. However, we can make different choices for the hypothesis space $\mathcal{H}$ and the loss function that

define ERM. Different ML methods use different choices for these components which, in turn, result in different computational aspects and statistical aspects of ERM (see [1, Ch. 3]).

By computational aspects of ERM, we mainly refer to the amount of computation required to find (approximate) solutions to (14). This amount is related to the shape of the empirical risk, viewed as a function of the hypothesis (parameters) (see Figure 9).

The statistical aspectss of (14) include the probability distribution of the solutions to (14) and their deviation from the Bayes estimator (11). Another important statistical aspects of (14) is its robustness against violations of the i.i.d. assumption. Different choices for the model and loss function used in (14) yield different levels of robustness for the resulting ML method [38].

For linear regression, which uses a linear model $h^{(\mathbf{w})}(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$ and squared error loss (2), the generic ERM (14) specializes to

$$\widehat{\mathbf{w}} := \operatorname*{argmin}_{\mathbf{w}\in\mathbb{R}^d} \underbrace{(1/m)\sum_{r=1}^{m}\left(\mathbf{w}^T\mathbf{x}^{(r)} - y^{(r)}\right)^2}_{\widehat{L}\left(h^{(\mathbf{w})}|\mathcal{D}\right)}. \tag{15}$$

3.3 How Much Training is Needed?

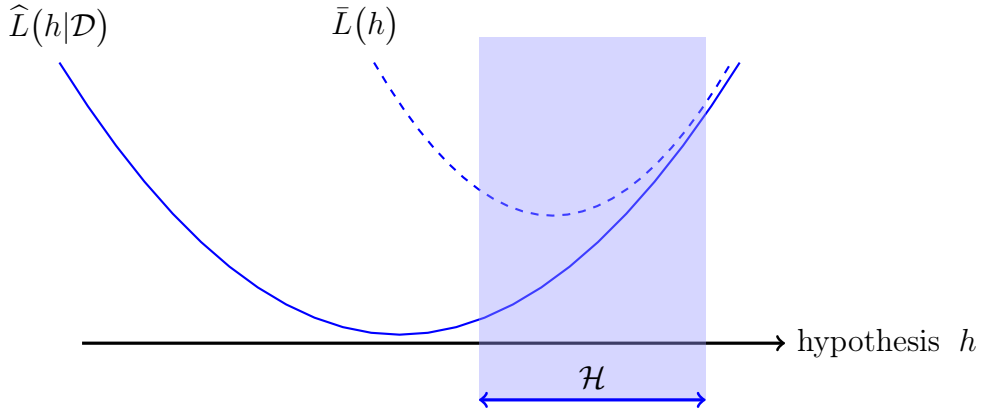

Figure 9: ERM (14) learns a hypothesis, out of a hypothesis space $\mathcal{H}$, by minimizing the empirical risk $\widehat{L}\big(h|\mathcal{D}\big)$ over a training set $\mathcal{D}$. However, the ultimate goal is to learn a hypothesis which predicts well the label of any data point and, in turn, has a small risk $\bar{L}\big(h\big)$. Thus, ERM can be expected to deliver a useful hypothesis if the deviation between the empirical risk and the risk is small (uniformly over $h \in \mathcal{H}$).

We have obtained ERM by approximating the risk $\bar{L}\big(h\big)$ with the empirical risk $\widehat{L}\big(h|\mathcal{D}\big)$ (see Figure 9). The risk is the ultimate performance criterion but cannot be computed without knowing the underlying probability distribution (see Section 3.1). Instead, we minimize the empirical risk on a given training set constituted by m data points. Thus, we learn a hypothesis $\hat{h}$ by solving ERM (14). Any solution of (14) can be used as the learnt hypothesis $\hat{h}$.

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

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

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

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

In what follows we interpret and data point, including those in the training set $\mathcal{D}$ of ERM (14), as realizations of i.i.d. RVs with a multivariate normal probability distribution. In particular, the features $\mathbf{x} \in \mathbb{R}^d$ of data points are realizations of a multivariate normal distribution, $\mathbf{x} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. The label of any data point is related to the features via a linear Gaussian model,

$$y = \overline{\mathbf{w}}^T \mathbf{x} + \varepsilon, \text{ with } \varepsilon \sim \mathcal{N}(0, \sigma^2), \mathbf{x} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}). \tag{16}$$

The linear Gaussian model (16) postulates that the label of data point is obtained as a noisy linear combination of its features. We form this linear combination using the entries of a true underlying (but unknown to the ML method) weight vector $\overline{\mathbf{w}} \in \mathbb{R}^d$. The additive noise ε in (16) is a realization of a zero-mean Gaussian RV with variance σ^2.

We learn a linear hypothesis $\hat{h}$ via the following instance of ERM (14),

$$\hat{h} := \operatorname*{argmin}_{h^{(\mathbf{w})} \in \mathcal{H}^{(d')}} (1/m) \sum_{r=1}^{m} \big(y^{(r)} - h^{(\mathbf{w})}\big(\mathbf{x}^{(r)}\big)\big)^2. \tag{17}$$

The linear model $\mathcal{H}^{(d')}$ uses the first $d' \in \{1, \ldots, d\}$ features $x_1, \ldots, x_{d'}$ of a

data point. Thus, we allow the ERM instance (17) to use only a subset of the d available features (which are modelled as $\mathbf{x} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$).

Note that the objective function in (17) involves the features and labels of data points in $\mathcal{D}$. Since we interpret those as realizations of RVs, also the learnt hypothesis $\hat{h}$ and its risk $\bar{L}\big(\hat{h}\big)$ become realizations of RVs. For a sufficiently large training set, such that $m > d' + 1$, the expectation of the risk $\bar{L}\big(\hat{h}\big)$ is [1, Ch. 6.3],

$$\mathbb{E}\{\bar{L}\big(\hat{h}\big)\} = B^2 + \underbrace{\big(B^2 + \sigma^2\big)d'/(m - d' - 1)}_{\text{variance } v} + \sigma^2. \tag{18}$$

Here, we used the bias term

$$B^2 = \sum_{j=d'+1}^{d} \overline{w}_j^2. \tag{19}$$

According to (18), the expected risk $\mathbb{E}\{\bar{L}\big(\hat{h}\big)\}$ depends on three components:

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

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

$$m \geq 10 \times d'. \tag{20}$$

- the noise floor σ^2 which is due to the intrinsic noise in the (assumed) relation (16) between features and label of a data point. Under our probabilistic model (16), no ML method can learn a hypothesis with (expected) risk smaller than σ^2.

We can interpret the component $B^2 + V$ in (18) as a measure for the estimation error incurred by using (17) (via its equivalent form (21)) as an estimator for the true underlying $\overline{\mathbf{w}}$ in (16). If we use the linear model $\mathcal{H}^{(d')}$ with $d' = d$, which implies that the bias vanishes $B^2 = 0$, the estimation error becomes $\sigma^2 d'/(m - d' - 1)$ [39, 40].

The third component σ^2 in (18) arises from the noise contained in the true label y (according to our probabilistic model (16)), which we predict by the learnt hypothesis $h^{(\widehat{\mathbf{w}})}(\mathbf{x})$.

Note that (18) is the expected risk of a specific ML method, i.e., using ERM (17) to learn the hypothesis $\hat{h}$. One might wonder if there is a better ML method that could learn a better linear hypothesis $h'(\mathbf{x}) = \big(\mathbf{w}'\big)^T \mathbf{x}$, having smaller expected risk: $\mathbb{E}\{\bar{L}\big(h'\big)\} < \mathbb{E}\{\bar{L}\big(\hat{h}\big)\}$? It turns out that this is not the case and that the hypothesis learnt via (17) achieves minimum expected risk. Indeed, the expected risk of any linear hypothesis learnt from m data points that follow the probabilistic model (16) is lower bounded by (18) [41].

To summarize, using a specific probabilistic model (16) we analyzed the expected risk $\mathbb{E}\bar{L}\big(\hat{h}\big)$ of the linear hypothesis $\hat{h}$ learnt via ERM (17). We decomposed $\mathbb{E}\bar{L}\big(\hat{h}\big)$ into different components, one of them being proportional to the ratio $d'/(m - d' - 1)$.

To ensure a small $\mathbb{E}\bar{L}\big(\hat{h}\big)$, the size m of the training set used by ERM (17) should be significantly larger (say, ten times) than the dimension d' of the

linear model used in (17). While our analysis only applies to linear models, their results provide useful intuition for non-linear models such as ANN. For example, we can (locally) approximate non-linear models by linear models to determine their effective dimension d' [1, Ch. 2].

3.4 Exercises

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

Exercise 3.2. Trivial Objective in ERM (5 points) Consider some arbitrary dataset which is used to learn a hypothesis out of $\mathcal{H}$ via ERM (14). For which combinations of loss function and hypothesis space does ERM not depend at all on the dataset.

Exercise 3.3. Data Leakage via ERM (10 points) Consider a dataset $\mathcal{D}$ with m data points, each characterized by a single integer-valued feature $x \in \mathbb{Z}$ and a real-valued label $y \in \mathbb{R}$. We try to learn a hypothesis which assigns each integer x a predicated label value $h(x)$. The learning is based on minimizing the average squared error loss $\widehat{L}\big(h|\mathcal{D}\big)$ over all hypothesis maps. Is it possible to perfectly recover the dataset just from knowing the values of $\widehat{L}\big(h|\mathcal{D}\big)$ for all $h \in \mathcal{H}$?

Exercise 3.4. ERM for linear regression (5 points) Show that the solution of (17) is the linear hypothesis $h^{(\widehat{\mathbf{w}})}(\mathbf{x}) = \big(\widehat{\mathbf{w}}\big)^T \mathbf{x}$ with weight vector $\widehat{\mathbf{w}} \in \mathbb{R}^{d'}$ being a solution to

$$\widehat{\mathbf{w}} \in \operatorname*{argmin}_{\mathbf{w} \in \mathbb{R}^{d'}} \sum_{r=1}^{m} \Big(y^{(r)} - \sum_{j=1}^{d'} w_j x_j^{(r)} \Big)^2 . \tag{21}$$

Exercise 3.5. Uniqueness in ERM for linear regression (10 points) Discuss sufficient conditions on the training set such that there is a single unique solution to (21).

Exercise 3.6. Estimation Error and Expected Risk (10 points) Consider a set of m data points whose features and label are realizations of RVs with a probability distribution defined via (16). We use ERM (21), with $d' = d$ to learn the weights of a linear hypothesis map. We can interpret the solution $\widehat{\mathbf{w}}$ of (21) as an estimator for the true underlying weights $\overline{\mathbf{w}}$ in (16). The performance of this estimator can be measured via the estimation error $\mathbb{E}\big\{ \|\widehat{\mathbf{w}} - \overline{\mathbf{w}}\|_2^2 \big\}$. Discuss the relation between this estimation error and the expected risk $\mathbb{E}\{\bar{L}\big(h^{(\widehat{\mathbf{w}})}\big)\}$.

Exercise 3.7. Fundamental Limit (10 points) Consider data points characterized by a feature vector $\mathbf{x} = \big(x_1, \ldots, x_5\big)^T$ and a numeric label $y \in \mathbb{R}$. Assume that the label of a data point is related to its features via a linear Gaussian model (see (16)),

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

Here, $\overline{w}_j$ are fixed weights and $\varepsilon \sim \mathcal{N}(0, 5)$ is a realization of a zero-mean Gaussian RV with variance 5.

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

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

If we use a sufficiently large hypothesis space $\mathcal{H}$, is it possible to find a hypothesis map $\hat{h} \in \mathcal{H}$ that perfectly fits the data points in $\mathcal{D}$ in the sense of

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

4 Regularization

The idea of ERM is to approximate the expected loss or risk of a hypothesis by the empirical risk $\widehat{L}\big(h|\mathcal{D}\big)$ (14) over a training set $\mathcal{D}$. Whenever this approximation fails to be accurate, the solution of ERM might be a hypothesis that performs poorly outside the training set used in (14). Figure 9 illustrates the difference between the risk and the empirical risk as its approximation.

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

4.1 Overfitting

Modern ML methods use a high-dimensional hypothesis space, such as a linear model with many features or an ANN with many neurons. These methods often use training sets with much fewer data points than dictated by the effective dimension of the hypothesis space. As a result, the training error of a learnt hypothesis (via ERM) is almost negligible while the validation error is excessively large. The ML method learns a hypothesis that overfits the training set but performs poorly outside the training set.

4.2 Regularization via Data, Model and Loss

The hypothesis delivered by ERM (14) might deliver poor predictions outside the training set $\mathcal{D}$ used in (14). Indeed, if the size of the training set is small compared to the size (dimension) of the hypothesis space, solving (14) will likely result in overfitting. The ML method will then learn a hypothesis that perfectly fits the training set $\mathcal{D}$ but does a poor job in predicted the labels of data points outside the training set [1, Ch. 6].

Regularization techniques modify ERM (14) to favour a hypothesis that does also well outside the training set. We can implement regularization via each of the three main ML components, either as

- data augmentation (see Figure 10): we enlarge the training set $\mathcal{D}$ in (14) by adding new data points obtained by perturbing features or labels.
- model pruning: we modify (14) by reducing the optimization domain, which is the hypothesis space $\mathcal{H}$, to a (tiny) subset $\mathcal{H}' \subseteq \mathcal{H}$,

$$\widehat{h} := \operatorname*{argmin}_{h \in \mathcal{H}'} \underbrace{(1/m) \sum_{r=1}^{m} L\big(\big(\mathbf{x}^{(r)}, y^{(r)}\big), h\big)}_{\widehat{L}\big(h|\mathcal{D}\big)} . \tag{22}$$

- loss penalization: modify the loss function in (14) by adding a scaled regularization term,

$$\widehat{h} := \operatorname*{argmin}_{h \in \mathcal{H}} \underbrace{(1/m) \sum_{r=1}^{m} L\big(\big(\mathbf{x}^{(r)}, y^{(r)}\big), h\big)}_{\widehat{L}\big(h|\mathcal{D}\big)} + \lambda \mathcal{R}\big\{h\big\} \tag{23}$$

It turns out that these three approaches to regularization are essentially equivalent. Figure 11 illustrates the equivalence between data augmentation

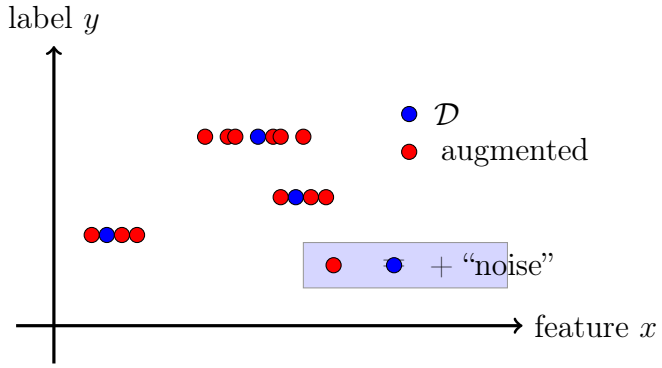

Figure 10: Data augmentation adds new data points to a given dataset $\mathcal{D}$. These new data points are obtained by perturbing features and labels of the data points in $\mathcal{D}$.

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

Consider linear regression methods that learn the weight vector of a linear hypothesis map $h^{(\mathbf{w})}(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$ by minimizing the average squared error loss on a training set. We augment $\mathcal{D}$ by adding B realizations of i.i.d. Gaussian random vectors with zero mean and covariance matrix $\sigma^2\mathbf{I}$. This results in the augmented dataset $\mathcal{D}'$ which contains each data point of $\mathcal{D}$ along with its B perturbations. It can then be shown that, for $B \to \infty$, the average squared error loss $\widehat{L}\big(h^{(\mathbf{w})}|\mathcal{D}'\big)$ on the augmented dataset $\mathcal{D}'$ tends towards $\widehat{L}\big(h^{(\mathbf{w})}|\mathcal{D}\big) + \sigma^2 \|\mathbf{w}\|_2^2$. Note that $\widehat{L}\big(h^{(\mathbf{w})}|\mathcal{D}\big) + \sigma^2 \|\mathbf{w}\|_2^2$ is an instance of regularized empirical risk minimization (RERM) (23) using the regularizer $\mathcal{R}\big\{h^{(\mathbf{w})}\big\} = \|\mathbf{w}\|_2^2$.

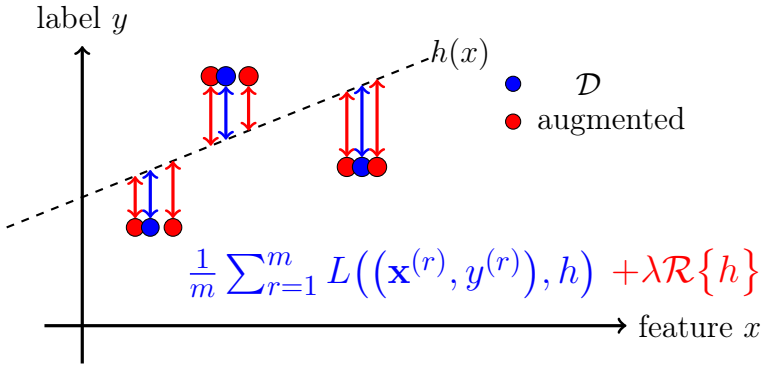

Figure 11: Equivalence of data augmentation and loss penalization techniques for regularization.

4.3 Federated Learning via Regularization

We will see how FL methods are obtained by combining different instances of RERM (23). In particular, we will study FL methods whose main building block is the RERM instance

$$\widehat{\mathbf{w}} := \operatorname*{argmin}_{\mathbf{w}\in\mathbb{R}^d} \underbrace{\widehat{L}\big(h^{(\mathbf{w})}|\mathcal{D}\big)}_{\text{training error}} + (\lambda/2) \underbrace{\|\mathbf{w}' - \mathbf{w}\|^2}_{\text{deviation from } \mathbf{w}'} . \tag{24}$$

The parameter vector $\widehat{\mathbf{w}}$ obtained from (24) balances between two (conflicting) goals. On one hand, we want to minimize the empirical risk $\widehat{L}\big(h^{(\mathbf{w})}|\mathcal{D}\big)$. On the other hand, the learnt parameter vector $\widehat{\mathbf{w}}$ should not deviate too much from a given reference parameter vector $\mathbf{w}'$.

The RERM (24) will be our main building block for the design of FL algorithms (see Section 9). In these algorithms, the vector $\mathbf{w}'$ in (24) might represent the current model parameters for similar learning tasks. Section 6 introduces empirical graphs to represent different but similar learning tasks. The nodes of an empirical graph represent local datasets and associated

learning tasks. The weighted edges of the empirical graph represent similarities between local datasets and corresponding learning tasks.

Note that solving RERM (24) is to evaluate the proximity operator $\mathbf{prox}_{f,\lambda}(\mathbf{w}')$ of the empirical risk $f(\mathbf{w}) = \widehat{L}\big(h^{(\mathbf{w})}|\mathcal{D}\big)$, viewed as a function of the model parameters $\mathbf{w}$. This interpretation provides a strong conceptual link between the FL methods developed in Section 7.2 and proximal algorithms [42].

4.4 Exercises

Exercise 4.1. Regularization via data augmentation (20 points). Consider the RERM (24) using some labeled dataset

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

linear hypothesis space and squared error loss (2). Show that (24) is equivalent to plain ERM (14) for linear regression applied to another dataset $\mathcal{D}'$ which is obtained by adding data points, with carefully chosen features and labels, to the original $\mathcal{D}$.

Exercise 4.2. Proximity operator for linear regression (10 points). Consider the update (24) for the specific choice of squared error loss and linear model. Show that this special case of (24) has always a unique solution.

5 Gradient Methods

plain vanilla GD; projected GD; stochastic GD; perturbed GD;

Section 3.2 formulated ML as ERM, which is a special case of an optimization problem. The properties of this optimization problem crucially depend on the design choices for the hypothesis space and loss function.

This section explains gradient-based methods for solving ERM arising in many widely-used ML methods [1, Ch. 3]. We will illustrate the key principles of gradient-based methods for linear regression methods. These methods aim at learning a linear hypothesis map to predict the numeric label of a data point from its numeric features. The restriction to linear regression allows for a rather complete analysis of gradient-based methods which also provides intuition for the behaviour of gradient-based methods in non-linear methods (e.g., using ANNs).

Gradient-based methods iterate the gradient step (see Section 5.1) which updates the current parameters by the scaled negative gradient of the empirical risk. We can interpret the gradient step as the constrained minimization of a local linear approximation of the empirical risk. Another interpretation of the gradient step is that of minimizing the approximation of the objective function by a quadratic function.

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

Section 5.3 discusses the effects of inexact gradient steps due to different

types of errors in the gradient computation. Numerical errors might arise from computational resource constraints (finite bitrate). Approximation errors might arise when using different objective functions for the implementation and the analysis of GD methods. This might seem strange but conceptually it might be useful to allow errors in order to analyze GD, e.g., smooth and strongly convex objective functions.

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

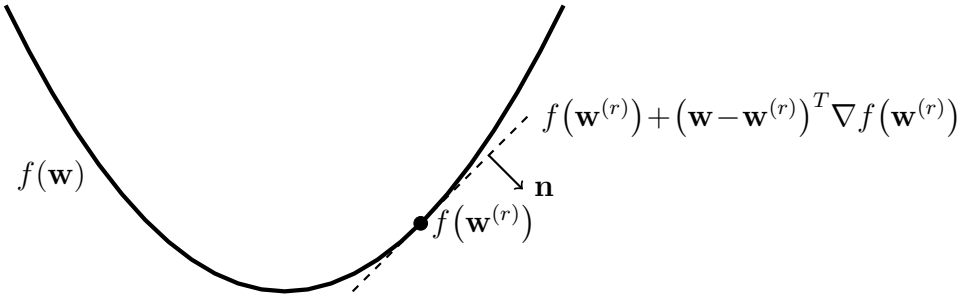

Figure 12: We can approximate a differentiable function $f(\mathbf{w})$ locally around a point $\mathbf{w}^{(r)}$ using a hyperplane. The normal vector $\mathbf{n} = (\nabla f\big(\mathbf{w}^{(r)}\big), -1)$ of this approximating hyperplane is determined by the gradient $\nabla f\big(\mathbf{w}^{(r)}\big)$ [43].

5.1 Gradient Descent

Let us rewrite ERM (15) for linear regression as

$$\min_{\mathbf{w}\in\mathbb{R}^d} f(\mathbf{w}) := (1/m)\sum_{r=1}^{m}\big(y^{(r)} - \mathbf{w}^T\mathbf{x}^{(r)}\big)^2. \tag{25}$$

Our goal is to compute (or approximate) an optimal parameter vector $\widehat{\mathbf{w}}$ that achieves the minimum objective value in (25). Note that the objective function $f(\mathbf{w})$ in (25) is a differentiable function of the parameter vector $\mathbf{w}$. In particular, we can compute a gradient $\nabla f(\mathbf{w})$ for any parameter vector $\mathbf{w} \in \mathbb{R}^d$ [44],

$$\nabla f(\mathbf{w}) = -(2/m)\sum_{r=1}^{m}\mathbf{x}^{(r)}\big(y^{(r)} - \mathbf{w}^T\mathbf{x}^{(r)}\big). \tag{26}$$

We use the gradient (26) to construct a sequence of parameter vectors $\mathbf{w}^{(0)}, \mathbf{w}^{(1)}, \ldots$. To this end, we start form an initial choice $\mathbf{w}^{(0)} \in \mathbb{R}^d$ and repeat the GD step

$$\begin{aligned}
\mathbf{w}^{(k+1)} &:= \mathbf{w}^{(k)} - \alpha\nabla f\big(\mathbf{w}^{(k)}\big) \\
&\overset{(26)}{=} \mathbf{w}^{(k)} + (2\alpha/m)\sum_{r=1}^{m}\mathbf{x}^{(r)}\big(y^{(r)} - \big(\mathbf{w}^{(k)}\big)^T\mathbf{x}^{(r)}\big) \\
&= \mathbf{w}^{(k)} + \mathbf{q} - \mathbf{Q}\mathbf{w}^{(k)} \text{ with } \mathbf{q} := (2\alpha/m)\sum_{r=1}^{m}\mathbf{x}^{(r)}y^{(r)}, \\
&\qquad\qquad \mathbf{Q} := (2\alpha/m)\sum_{r=1}^{m}\mathbf{x}^{(r)}\big(\mathbf{x}^{(r)}\big)^T.
\end{aligned} \tag{27}$$

Note that the GD step (27) only involves matrix/vector multiplications and additions. Thus, it can be implemented using numerical linear algebra soft- and hardware [45]. To obtain a practical method for learning a parameter vector, we need to

- choose a value for the step-size or learning rate α in (27) and
- define a stopping criterion to decide when to stop iterating (27).

One approach to choose the learning rate is to start with some initial value (first guess) and monitor the decrease of the objective value. If this decrease is too small (or even negative, which might happen when we overshoot), we gradually 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 is detected [30, Sec 6.1]. Another, even simpler approach is to use a prescribed sequence α_k of learning rates which satisfies the following conditions:

$$\lim_{k\to\infty} \alpha_k = 0, \ \sum_{k=1}^{\infty} \alpha_k = \infty \ , \text{and} \ \sum_{k=1}^{\infty} \alpha_k^2 = \infty. \tag{28}$$

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. Another choice for the stopping criterion could be obtained from monitoring the decrease in the objective value $f\big(\mathbf{w}^{(k)}\big)$, i.e., we stop repeating the GD step (27) whenever $\big|f\big(\mathbf{w}^{(k)}\big) - f\big(\mathbf{w}^{(k+1)}\big)\big| \leq \eta$ for a given tolerance η (which then becomes a hyper-parameter of the resulting ML method).

The above techniques for choosing the learning rate α and deciding when to stop the GD steps are quite useful in practice. However, they might result in sub-optimal use of computational resources for situations where we have more detailed knowledge about the objective function $f(\mathbf{w})$ in (25) and its gradient (see (27)). Indeed, the choice for the learning rate α and the stopping

criterion can be guided by the eigenvalues of the matrix (see (27))

$$\mathbf{Q} = (2\alpha/m) \sum_{r=1}^{m} \mathbf{x}^{(r)} \big(\mathbf{x}^{(r)}\big)^T = (2\alpha/m) \mathbf{X}\mathbf{X}^T. \tag{29}$$

Here we used the feature matrix $\mathbf{X} := \big(\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)}\big) \in \mathbb{R}^{d \times m}$. It can be shown that the matrix $\mathbf{Q}$ is positive semi-definite (psd) which implies than each of its eigenvalues is real-valued and non-negative,

$$\lambda_1\big(\mathbf{Q}\big) \leq \ldots \leq \lambda_d\big(\mathbf{Q}\big). \tag{30}$$

It turns out that a GD step (27) reduces the distance $\big\|\mathbf{w}^{(k)} - \widehat{\mathbf{w}}\big\|_2$ to $\widehat{\mathbf{w}}$ (the solution of (25)) by a constant factor [30, Ch. 6],

$$\big\|\mathbf{w}^{(k+1)} - \widehat{\mathbf{w}}\big\|_2 \leq \kappa^{(\alpha)}(\mathbf{Q}) \big\|\mathbf{w}^{(k)} - \widehat{\mathbf{w}}\big\|_2 . \tag{31}$$

Here we used contraction factor $\kappa^{(\alpha)}(\mathbf{Q}) := \max\big\{|1-\alpha 2\lambda_1|, |1-\alpha 2\lambda_d|\big\}$ which depends on the learning rate and the matrix $\mathbf{Q}$ (29) (via its eigenvalues (30)). According to (31), a sufficient condition on the number $k^{(\eta)}$ of GD steps required to ensure an optimization error $\big\|\mathbf{w}^{(k+1)} - \widehat{\mathbf{w}}\big\|_2 \leq \eta$ is

$$k^{(\eta)} \geq \log\big(\big\|\mathbf{w}^{(0)} - \widehat{\mathbf{w}}\big\|_2 / \eta\big) / \log \kappa^{(\alpha)}(\mathbf{Q}). \tag{32}$$

We can minimize the contraction factor in (31) by using the learning rate

$$\alpha^{(*)} := \frac{1}{\lambda_1 + \lambda_d}. \tag{33}$$

Inserting the optimal learning rate (33) into (31),

$$\big\|\mathbf{w}^{(k+1)} - \widehat{\mathbf{w}}\big\|_2 \leq \underbrace{\frac{(\lambda_1/\lambda_d) - 1}{(\lambda_1/\lambda_d) + 1}}_{:=\kappa^*(\mathbf{Q})} \big\|\mathbf{w}^{(k)} - \widehat{\mathbf{w}}\big\|_2 . \tag{34}$$

Note that both, the optimal learning rate (33) and the optimal contraction factor $\kappa^{*}(\mathbf{Q})$ depend on the eigenvalues of the matrix $\mathbf{Q}$ (29) which, in turn, is determined by the features of the data points used in ERM (15).

In general we have little control over the statistical properties of the features $\mathbf{x}^{(r)}$. However, we might be able to control the eigenvalues (30) of the matrix (29) via using a feature map as a pre-processing step. Two important example for such a feature map are the centering (removing sample mean) and normalization (enforcing unit norm) of features [1, Ch. 5]. Such a pre-processing of features might allow to ensure that the eigenvalues are contained between lower and upper bounds L and U, respectively. We can then use these bounds instead of the precise values for the (smallest and largest) eigenvalues.

According to (32), the ideal case is when all eigenvalues are identical which leads, in turn, to a contraction factor $\kappa^{*}(\mathbf{Q}) = 1$. In this regard, we should use a feature map that results in a matrix $\mathbf{Q}$ (obtained from (29) using the transformed feature vectors) being a scaled identity matrix.

Note that the matrix (29) can be interpreted as an approximates for the covariance matrix of a probability distribution from which the features are drawn i.i.d. (see 3.1). Some authors refer to a RV with covariance matrix being a scaled identity as "white" and, in turn, a feature map that aims at making the covariance matrix a scaled identity as a "whitening" of "decorrelation" transformation [46].

So far we have discussed the properties of GD for solving the ERM (25) arising in linear regression. Let us now study how the behaviour of GD changes by adding a regularization term to the average loss in (25). Thus,

we use GD to learn the parameters of a linear hypothesis by solving ridge regression

$$\min_{\mathbf{w}\in\mathbb{R}^d} f^{(\lambda)}(\mathbf{w}) := (1/m)\sum_{r=1}^{m}\big(y^{(r)} - \mathbf{w}^T\mathbf{x}^{(r)}\big)^2 + \lambda\,\|\mathbf{w}\|_2^2\,. \tag{35}$$

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

$$\nabla f^{(\lambda)}(\mathbf{w}) = -(2/m)\sum_{r=1}^{m}\mathbf{x}^{(r)}\big(y^{(r)} - \mathbf{w}^T\mathbf{x}^{(r)}\big) + 2\lambda\mathbf{w}. \tag{36}$$

Note that (36) and (35), respectively reduce to (36) and (25) for the specific choice $\lambda = 0$. The GD step for minimizing $f^{(\lambda)}$ becomes

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} + \mathbf{q} - \mathbf{Q}^{(\lambda)}\mathbf{w}^{(k)} \text{ with } \mathbf{q} := (2\alpha/m)\sum_{r=1}^{m}\mathbf{x}^{(r)}y^{(r)},$$
$$\mathbf{Q}^{(\lambda)} := 2\lambda\mathbf{I} + (2\alpha/m)\mathbf{X}\mathbf{X}^T. \tag{37}$$

Note that the GD step (37) for ridge regression has exactly the same form as the GD step (27) for linear regression. The convergence analysis (31) also applies to (37). In particular, the convergence speed of (37) is crucially influenced by the eigenvalues

$$0 \leq \tilde{\lambda}_1 \leq \ldots \leq \tilde{\lambda}_d \tag{38}$$

of the matrix $\mathbf{Q}^{(\lambda)}$. Basic results from linear algebra tell us that the eigenvalues (38) of the matrix $\mathbf{Q}^{(\lambda)} = \mathbf{Q} + \lambda\mathbf{I}$ and the eigenvalues (30) of the matrix $\mathbf{Q} = (2\alpha/m)\mathbf{X}\mathbf{X}^T$ are related via $\tilde{\lambda}_j = \lambda_j + \lambda$. This implies, in turn, that $\kappa^{(*)}\big(\mathbf{Q}^{(\lambda)}\big) \leq \kappa^{(*)}\big(\mathbf{Q}\big)$ and $\lim_{\lambda\to\infty}\kappa^{(*)}\big(\mathbf{Q}^{(\lambda)}\big) = 0$.

Thus, the stronger the regularization in (35) the faster the GD steps converge to a solution (35). However, this beneficial computational aspects

of using large λ might result in a poor hypothesis $\hat{h}(\mathbf{x}) = \big(\widehat{\mathbf{w}}^{(\lambda)}\big)^{T}\mathbf{x}$, with $\widehat{\mathbf{w}}^{(\lambda)}$ denoting the solution of (35). Indeed, using excessive regularization (too large λ) might incur a large bias (see Section 3.3 and [1, Ch. 6])

5.2 Projected Gradient Descent

Section 5.1 discussed methods for learning the parameter vector $\mathbf{w}$ of a linear hypothesis by ERM (25). For some applications it might be useful to constrain the parameter vector $\mathbf{w}$ to belong to a subset $\mathcal{C} \subset \mathbb{R}^{d}$. Indeed, restricting the parameter vectors to $\mathcal{C}$ is a form of model pruning which can help to avoid overfitting (see Section 4).

Another use case for constraining the model parameter are distributed FL methods. In particular, Section 9.3 discusses a FL method for coupling the training of local model parameters $\mathbf{w}^{(i)}$, for $i \in \{1,\ldots,n\}$ such that they eventually coincide. This coupling can be implemented by requiring the stacked parameter vector $\mathbf{w} = \big(\mathbf{w}^{(1)},\ldots,\mathbf{w}^{(n)}\big)^{T}$ to belong to the subset

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

Let us now show how to adapt the basic GD step (27) to solve the ERM

$$f^{*} = \min_{\mathbf{w}\in\mathcal{C}} f(\mathbf{w}) := (1/m) \sum_{r=1}^{m} \big(y^{(r)} - \mathbf{w}^{T}\mathbf{x}^{(r)}\big)^{2}. \tag{39}$$

We assume that the constraint set $\mathcal{C}$ is such that we can efficiently compute the projection

$$P_{\mathcal{C}}\big(\mathbf{w}\big) = \operatorname*{argmin}_{\mathbf{w}'\in\mathcal{C}} \|\mathbf{w} - \mathbf{w}'\|_{2}. \tag{40}$$

A suitable modification of (27) to solve the constrained problem (39) is

$$\begin{aligned}\mathbf{w}^{(k+1)} &:= P_{\mathcal{C}}\big(\mathbf{w}^{(k)} - \alpha \nabla f\big(\mathbf{w}^{(k)}\big)\big) \\ &\stackrel{(26)}{=} P_{\mathcal{C}}\big(\mathbf{w}^{(k)} + (2\alpha/m)\sum_{r=1}^{m} \mathbf{x}^{(r)}\big(y^{(r)} - \big(\mathbf{w}^{(k)}\big)^{T}\mathbf{x}^{(r)}\big)\big) \\ &= P_{\mathcal{C}}\big(\mathbf{w}^{(k)} + \mathbf{q} - \mathbf{Q}\mathbf{w}^{(k)}\big) \text{ with } \mathbf{q} := (2\alpha/m)\sum_{r=1}^{m} \mathbf{x}^{(r)} y^{(r)}, \\ &\qquad\qquad \mathbf{Q} := (2\alpha/m)\sum_{r=1}^{m} \mathbf{x}^{(r)}\big(\mathbf{x}^{(r)}\big)^{T}. \end{aligned} \tag{41}$$

The projected GD step (41) amounts to first computing an ordinary GD step $\mathbf{w}^{(k)} \mapsto \mathbf{w}^{(k)} - \alpha \nabla f\big(\mathbf{w}^{(k)}\big)$ and then projecting back to the constraint set $\mathcal{C}$. Note that we re-obtain the basic GD step (27) from the projected GD step (41) for the extreme case where $\mathcal{C} = \mathbb{R}^{d}$.

The approaches for choosing the learning rate α in - and defining a stopping criterion for - the basic GD step (27) explained in Section 5.1 can also be used for the projected GD step (41). Moreover, somewhat surprisingly, the convergence speed of the projected GD is characterized by the very same bound (31) as we derived for the unconstrained GD in Section 5.1 [30, Ch. 6]. However, the bound (31) is only telling about the number of GD steps required to achieve a guaranteed level of sub-optimality $\big|f\big(\mathbf{w}^{(k)}\big) - f^{*}\big|$. Each projected GD step (41) might require significantly more computation than the basic GD step, as it requires to compute the projection (40).

5.3 Perturbed Gradient Descent

5.4 Stochastic Gradient Descent

5.5 Exercises

Exercise 5.1. Positive Semidefinite (5 points). Develop a rigorous proof that the matrix $\mathbf{Q}$ defined in (27) is psd.

Exercise 5.2. Matrix Calculus (5 points). Provide a rigorous proof for the second equality in (29).

Part II

Federated Learning

6 Networked Data and Models

Many important application domains such as healthcare generate collections of local datasets. Indeed, healthcare providers, such hospitals or blood labs, maintain their own local patient databases. These local datasets are related via statistical similarities, e.g., the frequency of flu patients at nearby hospitals might have a similar temporal pattern. Section 6.1 introduces the empirical graph as a useful representation of collections of local datasets along with their similarities.

Section 6.2 augments the empirical graph by assigning a separate local hypothesis space (or model) to each node. We could train the local model for each node separately, using the corresponding local dataset. However, the local dataset might be too small to train a (high-dimensional) model such as a large ANN or a linear model with many features (see Section 3.3). We use the network structure of the empirical graph to pool local datasets to obtain a sufficiently large training set to train each local model.

This pooling can be motivated by a clustering assumption as discussed in Section 6.3. As an alternative to actually pooling local datasets at well-connected nodes, Section 7 puts forward regularization techniques to enforce similar local model parameters at well-connected nodes.

The empirical graph might be designed manually by using domain-expertise that provides measures for the similarity between local datasets. However, for some applications it might be useful to learn the empirical graph in a data-driven fashion (see Section 6.4).

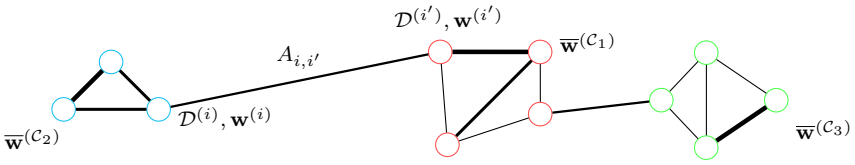

Figure 13: Example of an empirical graph whose nodes $i \in \mathcal{V}$ carry local datasets $\mathcal{D}^{(i)}$ and local models (parameters) $\mathbf{w}^{(i)}$. In this example, the empirical graph is partitioned into three disjoint clusters $\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3$, with cluster-wide optimal model parameter $\overline{\mathbf{w}}^{(\mathcal{C})}$.

6.1 The Empirical Graph

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

$$\mathcal{D}^{(i)} := \left\{ \left(\mathbf{x}^{(i,1)}, y^{(i,1)}\right), \ldots, \left(\mathbf{x}^{(i,m_i)}, y^{(i,m_i)}\right) \right\}. \tag{42}$$

Here, $\mathbf{x}^{(i,r)}$ and $y^{(i,r)}$ denote, respectively, the features and the label of the rth data point in the local dataset $\mathcal{D}^{(i)}$. Note that the size m_i of the local dataset might vary between different nodes $i \in \mathcal{V}$. Figure 13 depicts an example for an empirical graph.

An undirected edge $\{i, i'\} \in \mathcal{E}$ in the empirical graph indicates that the local datasets $\mathcal{D}^{(i)}$ and $\mathcal{D}^{(i')}$ have similar statistical properties. We quantify the level of similarity by a positive edge weight $A_{i,i'} > 0$. The neighbourhood of a node $i \in \mathcal{V}$ is $\mathcal{N}^{(i)} := \{i' \in \mathcal{V} : \{i, i'\} \in \mathcal{E}\}$.

Note that the undirected edges $\{i, i'\}$ of an empirical graph encode a

symmetric notion of similarity between local datasets. If the local dataset $\mathcal{D}^{(i)}$ at node i is (statistically) similar to the local dataset $\mathcal{D}^{(i')}$ at node i', then also the local dataset $\mathcal{D}^{(i')}$ is (statistically) similar to the local dataset $\mathcal{D}^{(i)}$.

Despite the symmetric notion of similarity represented by the edges $\mathcal{E}$ of an empirical graph, it will be convenient for the formulation and analysis of FL algorithms to orient the edges in $\mathcal{E}$. In particular, we define the head and tail of an undirected edge $e = \{i, i'\}$ as $e_+ := \min\{i, i'\}$ and $e_- := \max\{i, i'\}$, respectively. The directed edge of an empirical graph are

$$\big\{ (i, i') : i, i' \in \mathcal{V}, i < i' \text{ and } \{i, i'\} \in \mathcal{E} \big\}. \tag{43}$$

We abuse notation and use $\mathcal{E}$ to denote both, the set of undirected edges and its oriented version (43).

The empirical graph of networked data is a design choice which is guided by computational aspects and statistical aspects of the resulting ML method. For example, using an empirical graph with a small number of edges ("sparse graphs") typically results in a smaller computational complexity. Indeed, the amount of computation required by the FL methods developed in Section 9 is proportional to the number of edges in the empirical graph. On the other hand, the empirical graph should contain sufficient number of edges between nodes that carry statistically similar local datasets. This allows GTVMin techniques to adaptively pool local datasets into clusters of (approximately) homogeneous data (see Section 6.3).

6.2 Networked Models

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

The collection of local models $\mathcal{H}^{(i)}$, for each node $i \in \mathcal{V}$, constitutes a networked model $\mathcal{H}^{(\mathcal{G})}$ over an empirical graph $\mathcal{G}$. We also allow for heterogeneous networked models where different nodes carry different local models $\mathcal{H}^{(i)}$. For example, $\mathcal{H}^{(i)}$ might be a linear model $\mathcal{H}^{(d)}$, while $\mathcal{H}^{(i')}$ might be a decision tree for some other node $i' \neq i$.

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

$$L_i(\cdot) : \mathcal{H}^{(i)} \to \mathbb{R}_+ : h \mapsto L_i(h). \tag{44}$$

If node $i \in \mathcal{V}$ carries a local dataset of the form (42), we might define a local loss function via the average loss

$$L_i(h) := (1/m_i) \sum_{r=1}^{m_i} L\big(\big(\mathbf{x}^{(i,r)}, y^{(i,r)}\big), h\big). \tag{45}$$

However, the FL methods developed in Section 7 and Section 9 can be applied also to loss functions that are not of the form (45).

Our focus will be on parametrized networked models where each $\mathcal{H}^{(i)}$ is parametrized by a common finite-dimensional Euclidean space $\mathbb{R}^d$. This setting covers some widely-used ML models such as (regularized) generalized linear models or linear time series models [47–51]. For a parametrized networked model, the local hypothesis $h^{(i)}$ at any node $i \in \mathcal{V}$ is fully determined

by a local parameter vector $\mathbf{w}^{(i)} \in \mathbb{R}^d$. We find it convenient to collect local parameters $\mathbf{w}^{(i)}$ as the values of a map $\mathbf{w} : i \mapsto \mathbf{w}^{(i)}$. The set of all such maps is denoted

$$\mathcal{W} := \left\{ \mathbf{w} : i \mapsto \mathbf{w}^{(i)} \right\}. \tag{46}$$

The usefulness of a specific choice for the local model parameter $\mathbf{w}^{(i)}$ is measured by the local loss function $L_i\left(\mathbf{w}^{(i)}\right) := L_i\left(h^{(\mathbf{w}^{(i)})}\right)$. The local loss functions $L_i\left(\mathbf{w}^{(i)}\right)$ are an important design choice within FL systems. This design choice is related to the design choice for the local datasets and (the parametrization of) local models $\mathcal{H}^{(i)}$ (see (45)).

As for other ML components (see Section 2), there are statistical aspects and computational aspects guiding the choice for the local loss functions. Statistically, we want the loss function to favour local model parameters that result in a robust and accurate hypothesis for each node. Computationally, we want a local loss function that can be minimized efficiently. However, the difficulty of minimizing a local loss function depends on the type of optimization method that is used (see Section 5 and Section 9).

The FL methods in Section 9.3 and 9.2 require local loss functions that are differentiable and allow for efficient computation of their gradient at arbitrary points. Indeed, these FL methods use GD steps as their elementary computational unit.

The FL method in Section 9.1 requires that the local loss function $L_i\left(\mathbf{w}^{(i)}\right)$ allows for efficient solution of the regularized problem,

$$\min_{\mathbf{w}' \in \mathbb{R}^d} L_i\left(\mathbf{w}'\right) + \lambda \|\mathbf{w}' - \mathbf{w}''\|_2^2. \tag{47}$$

Note hat solving (47) is to evaluate the proximity operator $\mathbf{prox}_{L_i(\cdot),2\lambda}(\mathbf{w}'')$.

The basic computation (47) used by the FL method in Section 9.1 is itself an optimization problem which might be solved by the gradient-based methods in Section 5.

6.3 Clustering Assumption

Many applications generate local datasets which do not carry sufficient statistical power to guide learning of model parameters $\mathbf{w}^{(i)}$ (see Section 3.3).

As a case in point, consider a local dataset $\mathcal{D}^{(i)}$ of the form (42), with feature vectors $\mathbf{x}^{(r)} \in \mathbb{R}^d$ with $m_i \ll d$. We would like to learn the parameter vector $\mathbf{w}^{(i)}$ of a linear hypothesis $h(\mathbf{x}) = \mathbf{x}^T \mathbf{w}^{(i)}$. Linear regression methods [1, Sec. 3.1] learn the parameter vector by minimizing the average squared error loss

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

However, for $m_i \ll d$ (the high-dimensional regime) the above minimum is not unique and might also provide a poor hypothesis. In particular, a parameter vector with minimum average loss over $\mathcal{D}^{(i)}$ might incur unacceptably large prediction errors for data points outside $\mathcal{D}^{(i)}$ [1, Ch. 6].

Training linear models in the high-dimensional regime requires regularization techniques such as those used by ridge regression or least absolute shrinkage and selection operator (Lasso) [28]. These methods implement regularization by adding a penalty term to the average loss of a hypothesis incurred over a training set. This penalty term is an approximation or estimate for the increase in average loss when the hypothesis is applied to data points outside the training set.

Section 7 proposes different measures for the variation of local hypotheses $h^{(i)}$, for $i \in \mathcal{V}$, over edges in the empirical graph. The resulting measures for the total variation of a networked hypothesis are then used as a penalty term to regularize the training of local models. We will see that adding this penalty term results in an adaptive pooling of local datasets into clusters for which a common cluster-wise hypothesis is learnt.

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

Assumption 1 (Clustering (informal))**.** *Consider local datasets $\mathcal{D}^{(i)}$ represented by an empirical graph $\mathcal{G}$ whose nodes carry local loss functions $L_i(\mathbf{v})$ of the form (45). There is a partition*

$$\mathcal{P} = \{\mathcal{C}_1, \ldots, \mathcal{C}_{|\mathcal{P}|}\} \text{ with } \mathcal{C}_c \cap \mathcal{C}_{c'} = \emptyset \text{ and } \mathcal{V} = \mathcal{C}_1 \cup \ldots \cup \mathcal{C}_{|\mathcal{P}|}, \tag{48}$$

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

Assume we would know which nodes belong to the same cluster $\mathcal{C}$. It would then be sensible to pool the local datasets, or equivalently add the corresponding local loss functions (45), and learn model parameters by minimizing the resulting cluster-wise loss function.

$$\overline{\mathbf{w}}^{(\mathcal{C})} = \operatorname*{argmin}_{\mathbf{v} \in \mathbb{R}^d} f^{(\mathcal{C})}(\mathbf{v}) \text{ with } f^{(\mathcal{C})}(\mathbf{v}) := \sum_{i \in \mathcal{C}} L_i(\mathbf{v}). \tag{49}$$

Note that (49) cannot be implemented in practice since we typically do not know the cluster $\mathcal{C}$ for the i.i.d. assumption is valid (approximately). Instead, we will use regularization to couple the minimizing of individual local loss functions. This regularization will be implemented via measures for the variation of local model parameters across the weighted edges of the empirical graph.

Section 7 defines generalized total variation (GTV) as a measure for the variation of local model parameters. This results then in GTVMin as a flexible design principle for FL methods. These methods use the edges of the empirical graph to couple the training of local models by enforcing small variations of local model parameters across edges. It turns out that if nodes in the same cluster are connected by many edged (wight large weight) and only few edges (with small weights) connect nodes in different clusters, then GTVMin captures the underlying cluster structure in Assumption 1 and results in local model parameters that approximately solve (49).

6.4 Graph Learning Methods

We have introduced the empirical graph as a representation of collections of local datasets and their similarity structure. The empirical graph is a design choice that is guided by computational aspects and statistical aspects.

Computationally, we prefer a sparse empirical graph with few edges. Indeed, the FL methods presented in Section 9 can be implemented as message passing over the empirical graph which means that the amount of computation is proportional to the number of edges. Statistically, we want the empirical graph to reflect an underlying cluster structure (see Assumption

1). This might require to have a sufficiently large number of edges connecting nodes in the same cluster.

6.4.1 Testing Similarity

An edge between nodes $i, i' \in \mathcal{V}$ should represent a statistical similarity between local datasets $\mathcal{D}^{(i)}$ and $\mathcal{D}^{(i')}$. To make this notion precise we can use a probabilistic model (see Section 3.1) for (the generation of) local datasets. In particular, we interpret the data points in $\mathcal{D}^{(i)}$ as realizations of i.i.d. RVs with common probability distribution $p^{(i)}\big(\mathbf{x}, y\big)$. Similarly, we interpret the data points in $\mathcal{D}^{(i')}$ as realizations of i.i.d. RVs with common probability distribution $p^{(i')}\big(\mathbf{x}, y\big)$.

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

The KL divergence is between two probability distribution $p^{(i)}, p^{(i')}$ being multivariate normal with means $\boldsymbol{\mu}^{(i)}, \boldsymbol{\mu}^{(i')} \in \mathbb{R}^d$ and non-singular covariance matrices $\mathbf{C}^{(i)}, \mathbf{C}^{(i')}$, respectively,

$$D\big(p^{(i)}, p^{(i')}\big) = (1/2)\Big(\mathrm{tr}\big\{\big(\mathbf{C}^{(i')}\big)^{-1}\mathbf{C}^{(i)}\big\} - d \\ + \big(\boldsymbol{\mu}^{(i')} - \boldsymbol{\mu}^{(i)}\big)^T\big(\mathbf{C}^{(i')}\big)^{-1}\big(\boldsymbol{\mu}^{(i')} - \boldsymbol{\mu}^{(i)}\big) \\ + \log\big[\det(\mathbf{C}^{(i')})/\det(\mathbf{C}^{(i)})\big]\Big). \tag{50}$$

Note that in practice we do not know the underlying probability distribution, including the mean vector and covariance matrices, of a local dataset $\mathcal{D}^{(i)}$.

However, we can estimate these parameters using sample mean and sample covariance,

$$\widehat{\boldsymbol{\mu}}^{(i)} = (1/m) \sum_{r=1}^{m} \mathbf{x}^{(i,r)} \text{ (sample mean)}$$
$$\widehat{\mathbf{C}}^{(i)} = (1/m) \sum_{r=1}^{m} (\mathbf{x}^{(r)} - \widehat{\boldsymbol{\mu}}^{(i)})(\mathbf{x}^{(r)} - \widehat{\boldsymbol{\mu}}^{(i)})^T \text{ (sample covariance).} \quad (51)$$

We can plug in the estimators (51) into (50) to obtain a distance measure $\widehat{D}\big(p^{(i)}, p^{(i')}\big)$ that can be computed directly from local datasets.

Having a measure $\widehat{D}\big(p^{(i)}, p^{(i')}\big)$ for how much (the statistics) of local dataset $\mathcal{D}^{(i)}$ differs from $\mathcal{D}^{(i')}$ allows to construct an empirical graph in different ways.

- **thresholding**: Add an edge $\{i, i'\}$ if $\widehat{D}\big(p^{(i)}, p^{(i')}\big)$ is below a prescribed threshold.

- **nearest neighbours:** Fix a number $k \in \{0, \ldots, n-1\}$ and iterate over all nodes $i \in \mathcal{V}$ (in some order). For each $i \in \mathcal{V}$, add k edges between i and those other k nodes $i' \in \mathcal{V} \setminus \{i\}$ with smallest distance $\widehat{D}\big(p^{(i)}, p^{(i')}\big)$.

- **fully connected weighted graph** Add edges between every pair $i, i'\mathcal{V}$ of nodes but use different edge weights. Choose edge weight $A_{i,i'}$ based on $\widehat{D}\big(p^{(i)}, p^{(i')}\big)$, e.g., $A_{i,i'} = \exp\big(-\widehat{D}\big(p^{(i)}, p^{(i')}\big)\big)$.

6.4.2 Total Variation Minimization

Section 7 defines GTV as a (family of) measure(s) for the variation of local model parameter across edges of the empirical graph. The FL methods in Section 9 are all based on optimizing the GTV, viewed as a function of the local model parameters. However, if we have sufficiently large local datasets

$\mathcal{D}^{(i)}$, we might be able to learn useful model parameters separately (using plain old ML methods). We can then use these model parameters to learn the edges of the empirical graph by minimizing the GTV, viewed as a function of the empirical graph. [52, 53]

6.5 Exercises

Exercise 6.1. Get Weather Data. Construct an empirical graph $\mathcal{G}$ for the data collected at Finnish Meteorological Institute (FMI) weather stations. The empirical graph should contain at least $n = 5$ nodes which represent different FMI weather stations. The local dataset $\mathcal{D}^{(i)}$ at node i is constituted by $m = 5$ data points that represent the daily minimum and maximum temperatures during five consecutive days starting with 31 January 2014. The empirical graph should be stored as a `networkx.Graph` containing the local datasets as node attributes (in the form of `numpy` arrays).

Exercise 6.2. Nearest Neighbour Graph. One approach to learn an empirical graph $\mathcal{G}^{(\mathrm{nn})}$ for a collection of n local datasets is to first compute a measure for the pairwise distances between local datasets and then iterating over all nodes i and connected it with the k nearest nodes (see Section 6.4). Here, $k \in \{0, \ldots, n-1\}$ is a hyper-parameter that controls the number of edges in the resulting empirical graph. However, can we be sure that each node i in $\mathcal{G}^{(\mathrm{nn})}$ has at most k neighbours ?

7 A Design Principle for FL

Section 6.2 introduced the empirical graph whose nodes carry local datasets and corresponding local models. This section develops a flexible design principle for FL methods to train local models at nodes of an empirical graph. These methods (see Section 9) combine information carried by local datasets with their network structure, which is encoded in the weighted edges of the empirical graph.

We couple the training of local models by enforcing approximately identical model parameters for similar local datasets. Section 7.1 defines the GTV as a measure for the discrepancy of local model parameters at similar local datasets. Section 7.2 then introduces GTVMin to learn local model parameters that optimally balance between their empirical risk incurred on local datasets and their GTV. Section 7.3 discusses several useful interpretations of GTVMin. Section 7.4 extends the concept behind GTV and GTVMin to non-parametric local models (such as decision trees).

7.1 Generalized Total Variation

Consider an empirical graph whose nodes carry local loss functions $L_i\left(\mathbf{w}^{(i)}\right)$ that form few clusters. The cluster structure of local loss functions is reflected by a high density of edges between nodes in the same cluster but few boundary edges between them. It then seems reasonable to require a small variation of local parameter vectors $\mathbf{w}^{(i)}$ across edges.

We can measure the variation of local parameter vectors $\mathbf{w} \in \mathcal{W}$ across the edges in $\mathcal{G}$ via their variation $\mathbf{u} : e \in \mathcal{E} \mapsto \mathbf{u}^{(e)} \in \mathbb{R}^d$. The variation

assigns each edge $e \in \mathcal{E}$ the difference $\mathbf{u}^{(e)} := \mathbf{w}^{(e_+)} - \mathbf{w}^{(e_-)}$. The "edge space" $\mathcal{U}$ associated with an empirical graph $\mathcal{G}$ and node space $\mathcal{W}$ is constituted by the variations of the networked parameters $\mathbf{w} \in \mathcal{W}$

Computing the variation of local model parameters amounts to applying the block-incidence matrix

$$\mathbf{D} : \mathcal{W} \to \mathcal{U} : \mathbf{w} \mapsto \mathbf{u} \text{ with } \mathbf{u}^{(e)} = \mathbf{w}^{(e_+)} - \mathbf{w}^{(e_-)}. \tag{52}$$

We will also use the adjoint (transpose) $\mathbf{D}^T$ of the block-incidence matrix. The transpose is a map

$$\mathbf{D}^T : \mathcal{U} \to \mathcal{W} : \mathbf{u} \mapsto \mathbf{w} \text{ with } \mathbf{w}^{(i)} = \sum_{e \in \mathcal{E}} \sum_{i=e_+} \mathbf{u}^{(e)} - \sum_{i=e_-} \mathbf{u}^{(e)}. \tag{53}$$

Using the block-incidence matrix (52) we can write more compactly $\mathbf{u} = \mathbf{D}\mathbf{w}$.

A quantitative measure for the variation of $\mathbf{w}$ is the GTV

$$\|\mathbf{w}\|_{\text{GTV}} := \sum_{(i,i') \in \mathcal{E}} A_{i,i'} \phi\big(\mathbf{w}^{(i')} - \mathbf{w}^{(i)}\big) \text{ with penalty function } \phi(\cdot) : \mathbb{R}^d \to \mathbb{R}. \tag{54}$$

We also define the GTV for a subset of edges $\mathcal{S} \subseteq \mathcal{E}$ as

$$\|\mathbf{w}\|_{\mathcal{S}} := \sum_{(i,i') \in \mathcal{S}} A_{i,i'} \phi\big(\mathbf{w}^{(i')} - \mathbf{w}^{(i)}\big). \tag{55}$$

Strictly speaking, the GTV (54) defines an entire ensemble of variation measures. This ensemble is parametrized by a penalty function $\phi(\mathbf{v}) \in \mathbb{R}$ which we tacitly assume to be convex. The penalty function $\phi(\cdot)$ is an important design choice that determines the computational and statistical properties of the resulting GTVMin problem (see Section **??** and Section **??**). Two popular choices are $\phi(\mathbf{v}) := \|\mathbf{v}\|_2$, which is used by network Lasso [54],

and $\phi(\mathbf{v}) := (1/2)\,\|\mathbf{v}\|_2^2$ which is used by "MOCHA" [14]. Another recent FL method based on GTV regularization uses the choice $\phi(\mathbf{v}) := \|\mathbf{v}\|_1$ [55].

Different choices for the penalty function offer different trade-offs between computational complexity and statistical properties of the resulting FL algorithms. As a case in point, the network Lasso (which uses $\phi(\mathbf{u}) = \|\mathbf{u}\|_2$) is computationally more challenging than MOCHA (which used penalty $\phi(\mathbf{u}) = (1/2)\|\mathbf{u}\|_2^2$). On the other hand, network Lasso is more accurate in learning models for data with specific network structures (such as chains) that are challenging for method that use the smooth penalty $\phi(\mathbf{v}) := (1/2)\|\mathbf{v}\|_2^2$ [56, 57].

7.2 Generalized Total Variation Minimization

We can enforce similar parameter vectors $\mathbf{w}^{(i)} \approx \mathbf{w}^{(i')}$ for well-connected nodes i, i' in the same cluster $\mathcal{C}_c$, by favouring local parameter vectors $\mathcal{D}^{(i)}$ with a small GTV $\|\mathbf{w}\|_{\rm GTV}$. The extreme case of vanishing GTV is achieved by using identical local parameter vectors $\mathbf{w}^{(i)} = \mathbf{a}$ for all nodes $i \in \mathcal{V}$. However, we also need to take into account the local loss functions $L_i\left(\mathbf{w}^{(i)}\right)$ whose minimizers differ for nodes in different clusters (see Assumption 1). Thus, we learn the local parameter vectors $\mathbf{w}^{(i)}$ by balancing between (the sum of) local loss functions and GTV (54),

$$\widehat{\mathbf{w}} \in \operatorname*{arg\,min}_{\mathbf{w} \in \mathcal{W}} \sum_{i \in \mathcal{V}} L_i\left(\mathbf{w}^{(i)}\right) + \lambda \|\mathbf{w}\|_{\rm GTV}. \tag{56}$$

The regularization parameter $\lambda > 0$ in (56) steers the preference for learning parameter vectors $\mathbf{w}^{(i)}$ with small GTV versus incurring small total loss $\sum_{i \in \mathcal{V}} L_i\left(\mathbf{w}^{(i)}\right)$. The choice of λ can be guided by cross validation [1, Ch.

8], [28] or by analysis of solutions to (56) under a clustering assumption (see Section 6.3 and Section 6.3).

By increasing the value of λ in (56) we can enforce its solutions to become clustered: the local parameter vectors $\widehat{\mathbf{w}}^{(i)}$ are constant over increasingly larger subsets of nodes. Choosing λ larger than some critical value, that depends on the shape of the local loss functions and the network structure of $\mathcal{G}$, results in $\widehat{\mathbf{w}}^{(i)}$ being the same for all nodes $i \in \mathcal{V}$. This is useful for learning a single global model for all nodes (see Sections 8.1 and 8.2).

GTVMin (56) unifies and considerably extends some well-known methods for distributed optimization and learning. In particular, the network Lasso [54] is obtained from (56) for the choice $\phi(\mathbf{v}) := \|\mathbf{v}\|_2$. The MOCHA method [14] is obtained from (56) for the choice $\phi(\mathbf{v}) := \|\mathbf{v}\|_2^2$. Another special case of (56), obtained for the choice $\phi(\mathbf{v}) := \|\mathbf{v}\|_1$, has been studied recently [55].

7.3 Interpretations

We next discuss some useful relations between GTVMin (56) and basic ML techniques.

7.3.1 Regularized Empirical Risk Minimization

Note that GTVMin (56) is an instance of RERM that uses GTV (54) as regularizer. The empirical risk incurred by the networked parameter vectors $\mathbf{w} \in \mathcal{W}$ is measured by the sum of the local loss functions $\sum_{i \in \mathcal{V}} L_i \left(\mathbf{w}^{(i)}\right)$. The interpretation of GTVMin as a form of RERM, lends naturally to consider optimization methods for structured optimization problems [58].

7.3.2 Multi-task Learning

Consider networked data with empirical graph $\mathcal{G}$ whose nodes $i \in \mathcal{V}$ carry local datasets $\mathcal{D}^{(i)}$. Each node also carries a local model $\mathcal{H}^{(i)}$ which is trained from $\mathcal{D}^{(i)}$. The training of the local models $\mathcal{H}^{(i)}$ is a separate learning task. GTVMin couples these learning tasks via requiring a small GTV of the resulting local model parameters $\mathbf{w}^{(i)}$. Thus, we can interpret GTVMin as a special case of multi-task learning [59, 60]

7.3.3 Clustering

It can be shown that the solutions of GTVMin are approximately piece-wise constant over well-connected subsets of nodes in the empirical graph [47,50,56]. Thus, we can interpret GTVMin as a generalization of graph clustering methods such as spectral clustering or flow-based clustering [61, 62].

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

Convex clustering [63, 64] is obtained from GTVMin (56) for the local loss functions

$$L_i\left(\mathbf{w}^{(i)}\right) = \left\|\mathbf{w}^{(i)} - \mathbf{a}^{(i)}\right\|_2^2, \text{ for all nodes } i \in \mathcal{V} \tag{57}$$

and GTV penalty $\phi(\mathbf{u}) = \|\mathbf{u}\|_p$ being a p-norm $\|\mathbf{u}\|_p := \left(\sum_{j=1}^{d} |u_j|^p\right)^{1/p}$ with some $p \geq 1$. The vectors $\mathbf{a}^{(i)}$ in (57) are the observations that we wish to cluster.

7.3.4 Locally Weighted Learning

Another interpretation of GTVMin is that of locally weighted learning [65]. It can be shown that the solutions of GTVMin are piece-wise constant over well-connected subsets (clusters) $\mathcal{C} \subset \mathcal{V}$ of nodes. Thus, for each node i in a given cluster $\mathcal{C}$, the solution $\widehat{\mathbf{w}}^{(i)}$ of GTV approximates the solution $\overline{\mathbf{w}}^{(\mathcal{C})}$ of (49), $\widehat{\mathbf{w}}^{(i)} \approx \overline{\mathbf{w}}^{(\mathcal{C})}$. The deviation between $\widehat{\mathbf{w}}^{(i)}$ and $\overline{\mathbf{w}}^{(\mathcal{C})}$ will be bounded by Theorem **??**. Note that the cluster-wide optimization (49) can be written as a locally weighted learning problem [65, Sec. 3.1.2]

$$\overline{\mathbf{w}}^{(\mathcal{C})} = \operatorname*{argmin}_{\mathbf{w} \in \mathbb{R}^d} \sum_{i' \in \mathcal{V}} \rho_{i'} L_{i'} \left(\mathbf{w} \right). \tag{58}$$

The locally-weighted learning problem (58) involves the weights $\rho_{i'}$ which are equal to 1 if $i' \in \mathcal{C}$ and 0 otherwise.

7.4 Non-Parametric Models

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

For parametrized local models, we can measure the variation of hypotheses $h^{(\mathbf{w}^{(i)})}$ via the parameter vector differences $\mathbf{w}^{(i)}$ and $\mathbf{w}^{(i')}$ over edges $\{i, i'\} \in \mathcal{E}$. However, we could also measure the variation between $h^{(i)}$ and $h^{(i')}$ (across

an edge $\{i, i'\}$) via the discrepancy between their predictions on a common test-set

$$\mathcal{D}' = \left\{ \mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m')} \right\}. \tag{59}$$

Each node $i \in \mathcal{V}$ maintains a copy of $\mathcal{D}'$ which allows to compute the discrepancy

$$d_h^{(i,i')} := (1/m') \sum_{r=1}^{m'} \left[L\big(\big(\mathbf{x}^{(r)}, h^{(i)}\big(\mathbf{x}^{(r)}\big)\big), h^{(i')}\big) + L\big(\big(\mathbf{x}^{(r)}, h^{(i')}\big(\mathbf{x}^{(r)}\big)\big), h^{(i)}\big) \right]. \tag{60}$$

We define a variant of GTV (54) for non-parametric models by summing the discrepancy (60) over all edges $\mathcal{E}$,

$$\mathrm{GTV}\,\{h\} := \sum_{\{i,i'\} \in \mathcal{E}} A_{i,i'} d_h^{(i,i')}. \tag{61}$$

Note that $\mathrm{GTV}\,\{h\}$ is parametrized by the choice for the loss function L used to compute the discrepancy $d_h^{(i,i')}$ (60).

We obtain GTVMin for non-parametric local models by using $\mathrm{GTV}\,\{h\}$ as regularization term in explainable empirical risk minimization (EERM),

$$\widehat{h} \in \operatorname*{argmin}_{h \in \mathcal{H}^{(\mathcal{G})}} \sum_{i \in \mathcal{V}} L_i\left(h^{(i)}\right) + \lambda \mathrm{GTV}\,\{h\}. \tag{62}$$

Here, we use a networked hypothesis space $\mathcal{H}^{(\mathcal{G})} = \mathcal{H}^{(1)} \times \ldots \times \mathcal{H}^{(n)}$ whose elements are maps that assign each node $i \in \mathcal{V}$ a hypothesis $h^{(i)} \in \mathcal{H}^{(i)}$. Note that (62) is parametrized by the choice for the loss function used to compute the discrepancy (60). Different choices for the loss function in (60) result in different computational and statistical properties of (62). If we use the

squared error loss to measure the discrepancy (60), (62) specializes to

$$\widehat{h} \in \operatorname*{argmin}_{h \in \mathcal{H}^{(\mathcal{G})}} \sum_{i \in \mathcal{V}} L_i \left(h^{(i)}\right) + (\lambda/m') \sum_{(i,i') \in \mathcal{E}} A_{i,i'} \sum_{r=1}^{m'} \left(h^{(i)}\big(\mathbf{x}^{(r)}\big) - h^{(i')}\big(\mathbf{x}^{(r)}\big)\right)^2. \tag{63}$$

7.5 Exercises

Exercise 7.1. Generalizing TV to Decision Trees Think about ways to modify GTVMin (56) to applications that use decision trees as the local models.

8 Main Flavours of Federated Learning

Let us now discuss how different choices for the empirical graph and local loss functions used in GTVMin result in main flavours of FL [66]. The most basic FL setup is obtained when the empirical graph is a star with a server at its centre (see Section 8.1). Section 8.2 explains distributed FL where an arbitrary empirical graph is used for message passing implementations of FL algorithms to collaboratively train a single global model.

Note that GTVMin uses a regularization parameter λ to control the (strength of the) coupling between the local models at connected nodes. Section 8.3 discusses clustered FL which uses this parameter to steer the clustering structure of the learnt local models. One extreme case of clustered FL is distributed FL, where the empirical graph becomes one single cluster. Another extreme case is personalized FL, which is discussed in Section 8.4, where each node in the empirical graph becomes a separate cluster.

Section 8.5 discusses horizontal FL from local datasets that are obtained for a common underlying dataset but using different features for data points. Section 8.6 discusses horizontal FL from local datasets that are subsets of a common underlying dataset.

8.1 Centralized Federated Learning

Probably the most basic FL setup uses a server which is connected by separate links to each client. The clients generate local datasets which are used to train a single global model $\mathcal{H}$. FL methods use the server to maintain the current model parameters [10]. These global model parameters are broadcasted to

the clients which compute parameter updates based on their local datasets (e.g., using gradient steps). These local updates are sent back to the server which aggregates them to obtain new global model parameters.

The above server-based (centralized) FL setup correspond to GTVMin for an empirical graph being a star graph (see Figure 14).

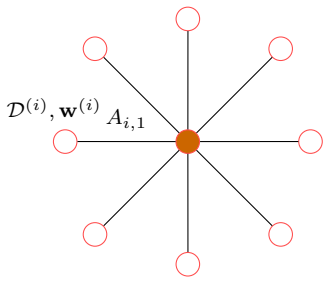

Figure 14: Star graph $\mathcal{G}^{(\text{star})}$ with centre node representing a server and peripheral nodes representing clients that generate local datasets.

8.2 Decentralized Federated Learning

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

8.3 Clustered Federated Learning

Clustered FL uses different forms of a clustering assumption that requires local datasets, and their associated learning tasks, to form clusters (see Assumption 1) [18, 19, 59, 68, 69]. Local datasets belonging to the same cluster have similar statistical properties and, in turn, similar optimal parameter values for the corresponding local models. Our recent work offers a precise characterization

of the cluster structure and local loss functions that allow FL methods from Section 9 to pool local datasets that form a cluster of statistically homogeneous local datasets [70]. This characterization also provides some guidance for choosing the regularization parameter λ in GTVMin (56).

8.4 Personalized Federated Learning

Using a sufficiently small regularization parameter in GTVMin allows local models (their parameters) to be different for each node i. However, these personalized trained models might still be coupled partially, e.g., sharing a subset ot its parameters such as the first (input) layers of an deep ANN. The partial parameter sharing can be implemented in many different ways [71, Sec. 4.3.].

We can use GTVMin (56) for partial parameter sharing by using a specific choice for the GTV penalty function. In particular, we could use a combination of two terms, each term measuring the variation of different parts (subsets) of parameters. These two terms might use different coupling strengths for the corresponding subsets of parameters (enforcing the low-level layers to have same parameters while deeper (closer to the output) layers can have different parameters). Another technique for partial parameter sharing is to train a hyper-model which, in turn, is used to initialize the training of personal local models [72].

8.5 Vertical Federated Learning

[73]

8.6 Horizontal Federated Learning

[74]

9 Federated Learning Algorithms

Section 7 introduced GTVMin as a design principle for FL methods that couple the training of local models on local datasets. We obtain FL algorithms by the application of optimization methods to solve GTVMin. For ease of exposition, and without essential loss of generality, we focus on the special case of GTVMin (56) with penalty $\phi(\mathbf{u}) = \|\mathbf{u}\|_2^2$,

$$\widehat{\mathbf{w}} \in \arg\min_{\mathbf{w}\in\mathcal{W}} \sum_{i\in\mathcal{V}} L_i\left(\mathbf{w}^{(i)}\right) + \lambda \sum_{\{i,i'\}\in\mathcal{E}} A_{i,i'} \left\|\mathbf{w}^{(i)} - \mathbf{w}^{(i')}\right\|_2^2. \tag{64}$$

We obtain FL methods by applying iterative optimization methods to solve GTVMin (64). These optimization methods construct a sequence

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

of local model parameters that are increasingly accurate approximations of the solutions $\widehat{\mathbf{w}}$ of (64).

9.1 FedRelax

We now apply block-coordinate minimization [22, 30] to solve GTVMin (64). To this end, we rewrite(64) as

$$\widehat{\mathbf{w}} \in \arg\min_{\mathbf{w}\in\mathcal{W}} \underbrace{\sum_{i\in\mathcal{V}} f^{(i)}(\mathbf{w})}_{:=f^{(\mathrm{GTV})}(\mathbf{w})}$$

$$\text{with } f^{(i)}(\mathbf{w}) := L_i\left(\mathbf{w}^{(i)}\right) + (\lambda/2) \sum_{i'\in\mathcal{N}^{(i)}} A_{i,i'} \left\|\mathbf{w}^{(i)} - \mathbf{w}^{(i')}\right\|_2^2. \tag{65}$$

The objective function in (65) decomposes into node-wise functions $f^{(i)}(\mathbf{w})$, for each $i \in \mathcal{V}$. Block-coordinate minimization exploits this structure to decouple the optimization of local model parameters $\widehat{\mathbf{w}}^{(i)}$.

We next discuss a basic variant of block-coordinate minimization to solve (65). Given the current local model parameters $\widehat{\mathbf{w}}_k$, we compute (hopefully improved) updated local model parameters $\widehat{\mathbf{w}}_{k+1}^{(i)}$ by minimizing $f^{(\mathrm{GTV})}(\cdot)$ along $\mathbf{w}^{(i)}$ starting from $\widehat{\mathbf{w}}_k$,

$$\begin{aligned}\widehat{\mathbf{w}}_{k+1}^{(i)} &\in \operatorname*{argmin}_{\mathbf{w}^{(i)} \in \mathbb{R}^d} f^{(\mathrm{GTV})}\left(\widehat{\mathbf{w}}_k^{(1)}, \ldots, \widehat{\mathbf{w}}_k^{(i-1)}, \mathbf{w}^{(i)}, \widehat{\mathbf{w}}_k^{(i+1)}, \ldots\right) \\ &\overset{(65)}{=} \operatorname*{argmin}_{\mathbf{w}^{(i)} \in \mathbb{R}^d} f^{(i)}\left(\widehat{\mathbf{w}}_k^{(1)}, \ldots, \widehat{\mathbf{w}}_k^{(i-1)}, \mathbf{w}^{(i)}, \widehat{\mathbf{w}}_k^{(i+1)}, \ldots\right) \\ &\overset{(65)}{=} \operatorname*{argmin}_{\mathbf{w}^{(i)} \in \mathbb{R}^d} L_i\left(\mathbf{w}^{(i)}\right) + (\lambda/2) \sum_{i' \in \mathcal{N}^{(i)}} A_{i,i'} \left\|\mathbf{w}^{(i)} - \widehat{\mathbf{w}}_k^{(i')}\right\|_2^2. \end{aligned} \tag{66}$$

Algorithm 1 simply repeats the update (66) for a sufficient number of iterations, i.e., until a stopping criterion is met.

Non-Parametric Models. Algorithm 1 can only be used for parametric local models $\mathcal{H}^{(i)}$ such as linear regression or ANNs with a fixed number d of parameters (which is the same for all nodes $i \in \mathcal{V}$). However, we can naturally extend the scope of Algorithm 1 to non-parametric models by applying block-coordinate minimization to the non-parametric GTVMin variant (62) instead of (64). We obtain the update

$$\begin{aligned}\widehat{h}_{k+1}^{(i)} \in \operatorname*{argmin}_{h \in \mathcal{H}^{(\mathcal{G})}} & L_i\left(h^{(i)}\right) + (\lambda/2) \sum_{i' \in \mathcal{N}^{(i)}} A_{i,i'} \underbrace{d_h^{(i,i')}}_{\text{see (60)}} \\ & \text{such that } h^{(i')} = \widehat{h}_k^{(i')} \text{ for } i' \in \mathcal{V} \setminus \{i\}. \end{aligned} \tag{68}$$

Replacing the update (66) with (68) in step 4 of Algorithm 1 makes it applicable to non-parametric local models. The update (68) is obtained by using the discrepancy (60) between local models within the GTV measure (61).

Algorithm 1 FedRelax for Parametric Local Models

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

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

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

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

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

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

4: update local parameters via (see (66))

$$\widehat{\mathbf{w}}_{k+1}^{(i)} := \operatorname*{argmin}_{\mathbf{w}^{(i)} \in \mathbb{R}^d} L_i\left(\mathbf{w}^{(i)}\right) + (\lambda/2) \sum_{i' \in \mathcal{N}^{(i)}} A_{i,i'} \left\| \mathbf{w}^{(i)} - \widehat{\mathbf{w}}_k^{(i')} \right\|_2^2 \tag{67}$$

5: **end for**

6: $k{:=}k{+}1$

7: **end while**

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

We obtain Algorithm 2 when using the squared error loss to measure the discrepancy (60) between local hypotheses $h^{(i)}$ and $h^{(i')}$ (see (63)).

Algorithm 2 FedRelax for Non-Parametric Models

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

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

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

2: **for** all nodes $i \in \mathcal{V}$ in parallel **do**

3: share test-set labels $\left\{\widehat{h}_k^{(i)}(\mathbf{x})\right\}_{\mathbf{x}\in\mathcal{D}^{(\text{test})}}$, with neighbours $i' \in \mathcal{N}^{(i)}$

4: update hypothesis $\widehat{h}_k^{(i)}$ as follows:

$$\widehat{h}_{k+1}^{(i)} \in \operatorname*{argmin}_{h^{(i)}\in\mathcal{H}^{(i)}} \Bigg[L_i\left(h^{(i)}\right) + (\lambda/2) \sum_{i'\in\mathcal{N}^{(i)}} (A_{i,i'}/m') \sum_{r=1}^{m'} \left(h^{(i)}(\mathbf{x}^{(r)}) - \widehat{h}_k^{(i')}(\mathbf{x}^{(r)})\right)^2 \Bigg]. \quad (69)$$

5: **end for**

6: $k{:=}k{+}1$

7: **end while**

9.2 FedSGD

9.3 FedAvg

The popular federated averaging (FedAvg) method is obtained from (64) for $\lambda \to \infty$. Indeed, for sufficiently large λ, the penalty term in (64) dominates, enforcing the learnt local parameter vectors to be nearly identical. Thus, for

sufficiently large λ, we can approximate (64) by

$$\widehat{\mathbf{w}} \in \arg\min_{\mathbf{w} \in \mathcal{C}} \sum_{i \in \mathcal{V}} L_i\left(\mathbf{w}^{(i)}\right) \\ \text{with } \mathcal{C} = \big\{\mathbf{w} : \mathbf{w}^{(i)} = \mathbf{w}^{(i')} \text{ for any edge } \{i, i'\} \in \mathcal{E}\big\}. \tag{70}$$

We can solve (70) using projected GD.

Let us spell out the special case of Algorithm 3 obtained when learning local linear models by minimizing the average squared error loss (2) incurred on the local dataset

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

The resulting Algorithm 4 performs regularized tailored linear regression for each local dataset. The regularization is implemented by the GTV (54) of the parameters $\mathbf{w}^{(i)}$ for the node-wise linear models.

Algorithm 3 FedAvg

Input: client list $\mathcal{V}$

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

Initialize. $k := 0$

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

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

3: update global parameter vector

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

4: send new global parameter vector $\overline{\mathbf{w}}\,[k]$ to all clients $i \in \mathcal{V}$

5: $k := k+1$

6: **end while**

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

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

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

3: update local parameters by RERM (24)

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

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

5: **end while**

Algorithm 4 FedAvg for linear regression

Input: client list $\mathcal{V}$

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

Initialize. $k := 0$

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

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

3: update global parameter vector

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

4: send new global parameter vector $\overline{\mathbf{w}}\,[k]$ 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 parameter vector $\overline{\mathbf{w}}\,[k]$ from server

3: update local parameters by RERM (see (24))

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

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

5: **end while**

9.4 Exercises

Exercise 9.1. FedRelax uses Proximity Operator Show that the coordinate minimization update (66) is equivalent to the evaluation of the proximity operator $\mathbf{prox}_{L_i(\cdot,,)\rho}(\mathbf{w}')$ for a specific vector $\mathbf{w}' \in \mathbb{R}^d$ and $\rho > 0$.

Exercise 9.2. FedAvg Linear Regression Discuss how step 3 of Algorithm 4 could be computed using the `LinearRegression.fit()` method provided by the Python package `scikit-learn`.

Part III

Trustworthy Federated Learning

10 Requirements for Trustworthy AI

FL is an important subfield of AI. As part of their AI strategy, the European Commission set up the High-Level Expert Group on Artificial Intelligence(AI HLEG) in 2018. This group put forward seven key requirements for trustworthy AI [75]:

1. **human agency and oversight.** *AI systems should empower human beings, allowing them to make informed decisions and fostering their fundamental rights. At the same time, proper oversight mechanisms need to be ensured, which can be achieved through human-in-the-loop, human-on-the-loop, and human-in-command approaches*

2. **technical robustness and safety.** *AI systems need to be resilient and secure. They need to be safe, ensuring a fall back plan in case something goes wrong, as well as being accurate, reliable and reproducible. That is the only way to ensure that also unintentional harm can be minimized and prevented.*

3. **privacy and data governance.** *besides ensuring full respect for privacy and data protection, adequate data governance mechanisms must also be ensured, taking into account the quality and integrity of the data, and ensuring legitimised access to data.*

4. **transparency.** *the data, system and AI business models should be transparent. Traceability mechanisms can help achieving this. Moreover, AI systems and their decisions should be explained in a manner adapted to the stakeholder concerned. Humans need to be aware that they are*

interacting with an AI system, and must be informed of the system's capabilities and limitations.

5. diversity non-discrimination and fairness,
6. societal and environmental well-being
7. accountability.

And I will now try to integrate these key requirements using the tools that we have built up in this course and these tools mainly revolve around design choices. Specifically, it´s a design choice for data, for the model, the set of hypothesis spaces and for the loss function. So, I will show you. How these design choices could be made to satisfy these key requirements.

11 Privacy Protection

A main application domain of FL is healthcare where local datasets are collected at different healthcare providers such as hospitals or laboratories [76]. Let us assume that local datasets contain data points that represent human beings. The features of data points are different bio-physical measurements and diagnosis reports. The label might the occurrence of some disease such as diabetes.

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

The FL methods discussed in Section 9 allow to share the information contained in local datasets to train local models in a privacy-friendly fashion. By privacy-friendly, we mean that no information about private properties (either a feature of a label) of a data point is revealed to anybody beyond the owner (or controller) of the local dataset. As a case in point, we will study the information flow during the iterations of Algorithm 1. Sections 11.1 and 11.2 will then explain modifications of Algorithm 1 to avoid the leakage of sensitive information. Note that Algorithm 1 only applies to parametrized hypothesis spaces which contain hypothesis maps $h^{(\mathbf{w}^{(i)})}$ that are fully determined by a parameter vector $\mathbf{w}^{(i)}$.

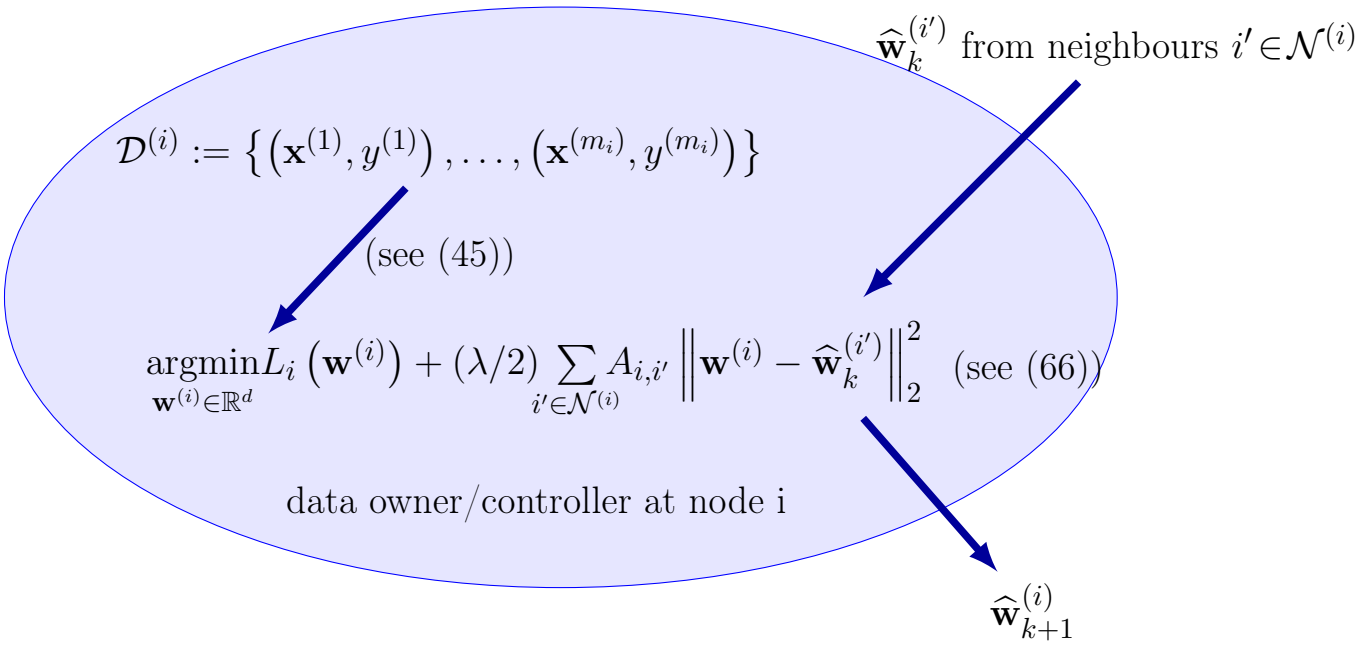

Figure 15: Data flow at node $i \in \mathcal{V}$ during a single iteration of Algorithm 1.

Figure 15 illustrates the data (and information) flow arising at node i during a single iteration of Algorithm 1. During each iteration of Algorithm 1, each node $i \in \mathcal{V}$ computes the update (66) and shares the updated local parameters $\widehat{\mathbf{w}}_{k+1}^{(i)}$ with its neighbours $\mathcal{N}^{(i)}$. It is the sharing of the updated parameters in step 3 of Algorithm 1 that might cause leakage of sensitive information.

We next discuss two different approaches to avoid this leakage of sensitive information. The first approach in Section 11.1 is to perturb (e.g., by adding noise) the update $\widehat{\mathbf{w}}_{k+1}^{(i)}$ before sharing it with the neighbours $\mathcal{N}^{(i)}$. Section 11.2 presents a complementary approach which is to directly perturb the features and labels of the data points in the local dataset $\mathcal{D}^{(i)}$.

11.1 Adding Noise

We can formalize the notion of privacy leakage and privacy protection with a probabilistic model (see Section 3.1). To this end, we interpret data points in the local dataset $\mathcal{D}^{(i)}$ as realizations of RVs. This implies, in turn, that the updates $\widehat{\mathbf{w}}_k^{(i)}$ are realizations of RVs. Moreover, a private feature $x_j^{(r)}$ becomes a realization of a RV. If the updates $\widehat{\mathbf{w}}_k^{(i)}$ could be used by an adversary to estimate or guess the value of $x_j^{(r)}$, they would leak sensitive information. A quantitative measure for how much observing one RV helps to estimate another RV is mutual information (see Section 14).

To avoid leakage of sensitive information, which is carried by the private feature $x_j^{(r)}$, we need to ensure a small mutual information $I\left(x_j^{(r)}, \widehat{\mathbf{w}}_k^{(i)}\right)$. This mutual information is determined by the joint probability distribution of the local datasets $\mathcal{D}^{(i)}$, for $i \in \mathcal{V}$, and the parameters of Algorithm 1. If $I\left(x_j^{(r)}, \widehat{\mathbf{w}}_k^{(i)}\right)$ is too large, we replace the updates $\widehat{\mathbf{w}}_k^{(i)}$ in steps 3 and 4 of Algorithm 1 by the perturbed updates [77, 78]

$$\widetilde{\mathbf{w}}_k^{(i)} := \widehat{\mathbf{w}}_k^{(i)} + \sigma \boldsymbol{\varepsilon}^{(i,k)}. \tag{71}$$

Here, $\boldsymbol{\varepsilon}^{(i,k)}$ are realizations of i.i.d. RVs with common probability distribution $\mathcal{N}(\mathbf{0}, \mathbf{I})$. The parameter σ in (71) controls the amount of privacy protection offered by (71). By increasing the value of σ, we obtain an increased protection of private features, however, at the cost of deteriorating accuracy of the hypothesis obtained from $\widetilde{\mathbf{w}}_k^{(i)}$. We summarize the resulting modification of Algorithm 1 in Algorithm 5.

Algorithm 5 FedRelax with Perturbed Updates

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

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

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

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

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

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

4: update local parameters and add noise

$$\widetilde{\mathbf{w}}_{k+1}^{(i)} := \underset{\mathbf{w}^{(i)} \in \mathbb{R}^d}{\operatorname{argmin}} \left[L_i\left(\mathbf{w}^{(i)}\right) + (\lambda/2) \sum_{i' \in \mathcal{N}^{(i)}} A_{i,i'} \left\| \mathbf{w}^{(i)} - \widetilde{\mathbf{w}}_k^{(i')} \right\|_2^2 \right] + \sigma \boldsymbol{\varepsilon}^{(i,k)}. \quad (72)$$

5: **end for**

6: $k:=k+1$

7: **end while**

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

11.2 Feature Perturbation

Let us now describe another modification of Algorithm 1 that protects a private feature $x_j^{(r)}$ of data points in the local dataset $\mathcal{D}^{(i))}$. Instead of perturbing the result of the updates 4 in Algorithm 1, we directly perturb the features of the data points in $\mathcal{D}^{(i)}$. The perturbed features result, in turn, in a perturbed local loss function $\tilde{L}_i(\cdot)$.

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

$$\mathbf{\Phi} : \mathbf{x} \mapsto \mathbf{z} = \mathbf{\Phi}(\mathbf{x}). \tag{73}$$

A privacy-preserving feature map (73) delivers transformed features $\mathbf{z}$ that do not allow to predict (infer) the private feature (too well). However, the new features $\mathbf{z}$ should still allow to learn a hypothesis $h^{(\mathbf{w}^{(i)})}$ that accurately predicts the label of a data point from its privacy-preserving features $\mathbf{z}$. Such a hypothesis can be found via ERM (see Section 3.2), i.e., by minimizing the average loss

$$\tilde{L}_i\left(\mathbf{w}^{(i)}\right) := (1/m_i) \sum_{r=1}^{m_i} L\big(\big(\mathbf{\Phi}\big(\mathbf{x}^{(r)}\big), y^{(r)}\big), h^{(\mathbf{w}^{(i)})}\big). \tag{74}$$

We obtain Algorithm 6 from Algorithm 1 by replacing the local loss function $L_i(\cdot)$ with the perturbed version $\tilde{L}_i(\cdot)$ (74).

Examples for privacy-preserving feature maps (73) that could be used for Algorithm 6 include

- "information obfuscation" by adding noise $\mathbf{\Phi}(\mathbf{x}) = \mathbf{x} + \text{noise}$ (see [78, Fig. 3])

Algorithm 6 FedRelax with Perturbed Features

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

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

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

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

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

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

4: update local parameters

$$\widehat{\mathbf{w}}_{k+1}^{(i)} := \operatorname*{argmin}_{\mathbf{w}^{(i)} \in \mathbb{R}^d} \Bigg[(1/m_i) \sum_{r=1}^{m_i} L\big(\big(\mathbf{\Phi}\big(\mathbf{x}^{(r)}\big), y^{(r)}\big), h^{(\mathbf{w}^{(i)})}\big) + (\lambda/2) \sum_{i' \in \mathcal{N}^{(i)}} A_{i,i'} \Big\| \mathbf{w}^{(i)} - \widehat{\mathbf{w}}_k^{(i')} \Big\|_2^2 \Bigg] \tag{75}$$

5: **end for**

6: $k{:=}k{+}1$

7: **end while**

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

- linear map $\mathbf{\Phi}(\mathbf{x}) = \mathbf{W}\mathbf{x}$ [1, Sec. 9.5] with a matrix $\mathbf{W}$ that is determined by the correlations between private and non-private features. Figure 16 illustrates a toy dataset for which we can find a transformation that perfectly separates the private (gender) from the non-private feature which is used to predict the food preference of a person. One challenge for the use of learnt feature maps is that they need to be shared among all nodes. Indeed, we need use the same features for computing the GTV (54) (or its non-parametric extension (60)).

- the feature map $\mathbf{\Phi}(\mathbf{x})$ is piecewise constant such that each region contains at least a prescribed minimum number k of data points from $\mathcal{D}^{(i)}$. The resulting perturbed feature vectors $\mathbf{z}^{(r)} = \mathbf{\Phi}\big(\mathbf{x}^{(r)}\big)$ are said to provide k-anonymity [79, 80]. Figure 17 illustrates a piece-wise constant feature map that quantizes a feature that uniquely identifies a data point.

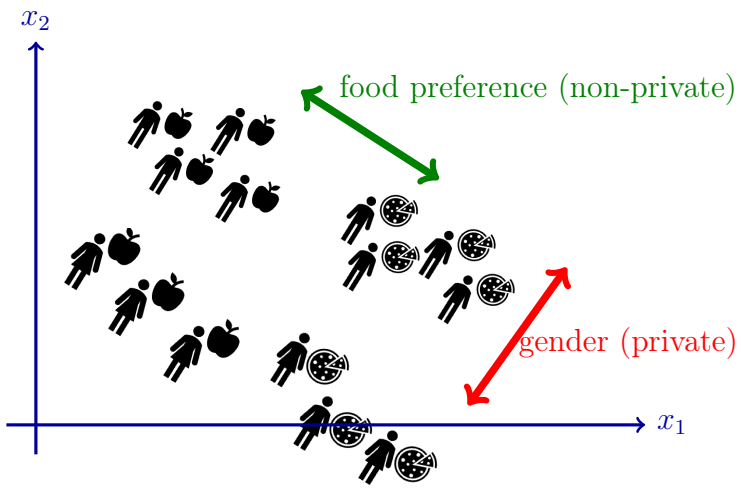

Figure 16: A toy dataset $\mathcal{D}^{(i)}$ whose data points represent persons that are characterized by two features x_1, x_2. These features carry sensitive information (gender) and non-sensitive information (food preference) about a person. The non-sensitive information allows to predicting the quantity of interest (e.g., if a person likes pizza).

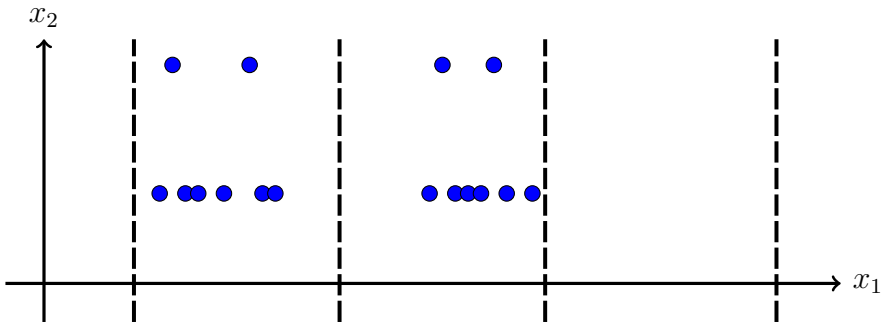

Figure 17: A toy dataset $\mathcal{D}^{(i)}$ whose data points are characterized by two features: x_1 and x_2. The feature x_2 is private and carries sensitive information. The other feature x_1 is non-private but uniquely identifies a data point: two different data points in $\mathcal{D}^{(i)}$ have different values of x_1. One simple approach to protect the private property x_1 is to quantize the feature x_1 resulting in a perturbed feature $\tilde{x}_1$ that is the same for at least k data points, thereby ensuring k-anonymity.

11.3 Exercises

Exercise 11.1. Privacy-Preserving Feature Learning (10 points) Consider the toy dataset `https://scikit-learn.org/stable/datasets/toy_dataset.html#diabetes-dataset` whose data points consider patients. Each data point is characterized by $d = 10$ features $\mathbf{x} = \big(x_1, \ldots, x_d\big)^T$ and a label y which measures a disease progression. The private feature x_1 characterizes the age of a person. We could simply remove this feature to obtain a privacy-preserving feature vector $\mathbf{x}'' = \big(x_2, \ldots, x_d\big)^T \in \mathbb{R}^d$. This exercise explores how much better we could do by learning a privacy-preserving feature map $\boldsymbol{\Phi}$. Construct a feature map such that the transformed features do not allow to predict the age of the person. Choose on ML method from `scikit-learn` and compare the achievable performance when using the transformed features $\mathbf{x}' = \boldsymbol{\Phi}(\mathbf{x})$ with using the features $\mathbf{x}''$.

12 Data Poisoning

The FL algorithms of Section 9 pool the information contained in decentralized local datasets to train local (tailored) models (see Section 6.2). The benefit of information sharing, allowing to learn more accurate hypotheses, comes at the cost of increased vulnerability against data poisoning.

Data poisoning refers to the intentional manipulation (or fabrication) of local datasets to steer the training of a specific local model [81, 82]. We discuss two main flavours of data poisoning, known as a denial-of-service attack and a backdoor attack. Section 12.1 discusses how a denial-of-service attack manipulates local model training such that it performs poorly on its own local dataset [83]. Section 12.2 explains how backdoor attacks maliciously steer the training of a local model such that it performs well in general but anomalously for data points with specific features [84].

12.1 Denial-of-Service Attacks

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

12.2 Backdoor Attack

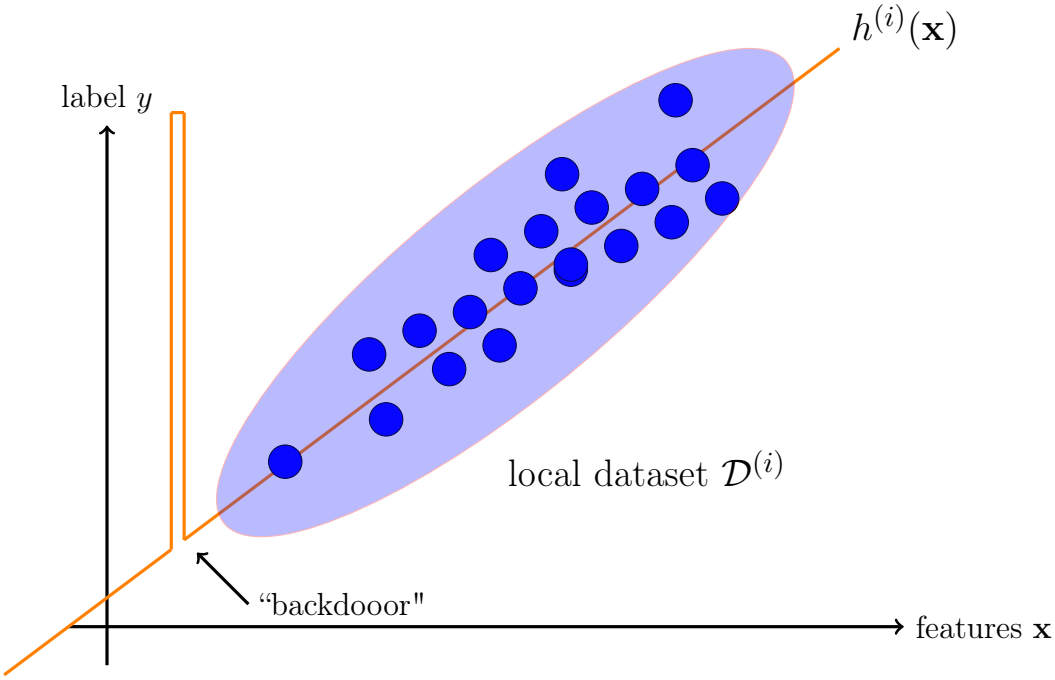

Figure 18: A backdoor attack aims at nudging the local hypothesis $h^{(i)}$ to behave in a specific way for certain feature values. These feature values can be interpreted as the key that unlocks a "backdoor".

12.3 Exercises

Exercise 12.1. Backdoor Attack (10 points) Consider the toy dataset `https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_digits.html#sklearn.datasets.load_digits` whose data points are images of hand-written digits. We use the first $m' = 100$ data points as a validation set. Construct a training set for a decision tree classifier such that the trained classifier achieves at accuracy at least 0.9 on the validation set and classifies the 104th data point as any prescribed digit.

Part IV

Appendix

13 Linear Algebra and Convex Analysis

The main data structure of the FL methods discussed in part II are numeric arrays such as vectors or matrices [85]. Numeric arrays can be combined via algebraic operations such as (element-wise) addition $\mathbf{x} + \mathbf{y}$ or multiplication $\mathbf{Zx}$ of compatible arrays (e.g., $\mathbf{Z} \in \mathbb{R}^{10\times 3}$ and $\mathbf{x} \in \mathbb{R}^3$). These operations are the most important building blocks for most FL methods discussed in part II.

13.1 Convex Sets

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

13.2 Convex Functions

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

13.3 Proximal Operators

Given a closed proper convex function $f(\mathbf{x})$ with domain being a subset of $\mathbb{R}^d$, we define its associated convex conjugate function as [21]

$$f^*(\mathbf{x}) := \sup_{\mathbf{z}\in\mathbb{R}^d} \mathbf{x}^T\mathbf{z} - f(\mathbf{z}). \tag{76}$$

The proximity operator associated with a closed proper convex function $f(\mathbf{x})$ is defined as [86]

$$\mathbf{prox}_{f,\rho}(\mathbf{x}) := \operatorname*{argmin}_{\mathbf{x}'} f(\mathbf{x}') + (\rho/2)\|\mathbf{x}-\mathbf{x}'\|_2^2 \text{ for some } \rho > 0. \tag{77}$$

Note that the minimum in (77) exists and is unique since the objective function is strongly convex [21] .

14 Information Theory

It is often useful to interpret data points as realizations of RVs. The behaviour of ML algorithms that use these data points can then be studied using the relations between these RVs. A powerful tool to study relations between RVs is the concept of mutual information [87, 88]. The mutual information between two RVs $\mathbf{x}, y$ with joint probability distribution $p(\mathbf{x}, y)$ is defined as

$$I(\mathbf{x}, y) := \mathbb{E}\left\{\log p(\mathbf{x}, y)/(p(\mathbf{x})p(y))\right\} \tag{78}$$

15 Fixed Point Theory

Glossary

0/1 loss The 0/1 loss $L\big(\big(\mathbf{x},y\big),h\big)$ measures the quality of a classifier $h(\mathbf{x})$ that delivers a prediction $\hat{y}$ for a label y of a data point. It is equal to 0 if the prediction is correct, i.e., $L\big(\big(\mathbf{x},y\big),h\big)=0$ when $\hat{y}=y$. It is equal to 1 if the prediction is wrong, $L\big(\big(\mathbf{x},y\big),h\big)=0$ when $\hat{y}\neq y$. 25, 26, 31–33, 35, 36

k-means The k-means algorithm is a hard clustering method which assigns each data points to pecisely 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). 126, 140

accuracy Consider data points characterized by features $\mathbf{x}\in\mathbf{x}$ and a categorical label y which takes on values from a finite label space $\mathcal{Y}$. The accuracy of a hypothesis $h:\mathcal{X}\rightarrow\mathcal{Y}$, when applied to the data points in a dataset $\mathcal{D}=\big\{\big(\mathbf{x}^{(1)},y^{(1)}\big),\ldots,\big(\mathbf{x}^{(m)},y^{(m)}\big)\big\}$ is then defined as $1-(1/m)\sum_{r=1}^{m}L\big(\big(\mathbf{x}^{(r)},y^{(r)}\big),h\big)$ using the 0/1 loss (4) 26, 32

artificial intelligence Artificial intelligence aims at developing systems that behave rational in the sense of maximizing a long-term reward. 13, 100

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. 21, 22, 47, 50, 56, 67, 89, 92, 123, 128, 136, 141

backdoor A backdoor (atttack) is obtained by data poisoning such that a trained local model predicts well most data points but anomalously for specific feature values (which unlock the backdoor). 112

Bayes estimator A hypothesis whose Bayes risk is minimal [32]. 39, 40, 42, 119

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 [32]. 119

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 $B^2 := \big\|\mathbb{E}\{\widehat{\mathbf{w}}\} - \overline{\mathbf{w}}\big\|_2^2$. 45, 46, 63

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. 17, 26, 29, 33, 35, 37

classifier A classifier is a hypothesis (map) $h(\mathbf{x})$ that is used to predict a

discrete-valued label. Strictly speaking, a classifier is a hypothesis $h(\mathbf{x})$ that can take only a finite number of different values. However, we are sometimes sloppy and use the term classifier also for a hypothesis that delivers a real number which is quantized (compared against a threshold) to obtain the predicted label value. For example, in a binary classification problem with label values $y \in \{-1, 1\}$, we refer to a linear hypothesis $h(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$ as classifier if it is used to predict the label value according to $\hat{y} = 1$ when $h(\mathbf{x}) \geq 0$ and $\hat{y} = -1$ otherwise. 25, 26, 29, 30, 35, 36, 114, 118

cluster A cluster within an empirical graph is a subset of nodes that carry local datasets with (nearly) identical statistical distributions. Clustered FL aims at identying the clusters and use the pooled local darasets to train a separate model for each cluster. 73, 79

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 Euclidean feature vectors it might be reasonable to define similarity between two data points using the (inverse of th) Euclidean distance between the corresponding feature vectors 87–89

clustering assumption The clustering assumption postulates that data points form a (small) number of groups or clusters, within which the data points behave similar [68]. 67, 82, 88

computational aspects By computational aspects of a ML method, we

refer to resources required to implement the method. 16, 18–20, 24, 26, 41, 42, 62, 69, 71, 74, 132

convex A set $\mathcal{C}$ in $\mathbb{R}^d$ is convex if it contains the line segment between any two points of that set. A function is called convex if its epigraph is a convex set [21]. 26, 28, 29, 32–34, 137

data A set of data points. 9, 16, 50, 137

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. 51–53, 55

data point A data point is any object that conveys information [87]. 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 a label. The label of a data point represents a higher-level facts or quantities of interest. In contrast to features, determining the label of a data point typically requires human experts (domain experts). Roughly speaking, ML aims at predicting the label of a data point based solely on its features. 7, 8, 16–21, 23–25, 27–32, 35–41, 43–46, 48–52, 55, 56, 61, 68, 72, 75, 78, 87, 102–104, 106, 108–112, 114, 117–125, 128–135, 139–142

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 [81, 82]. 10, 112, 119, 123

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 $\mathbf{z}^{(1)}, \ldots,$. 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 [89]. By using indexed lists of data points we avoid some of the challenges arising in concept of an abstract set. 7, 12, 36, 38, 40, 41, 48, 50, 52, 68, 87, 106, 107, 109–112, 114, 118, 125, 129, 132

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 $h(\mathbf{x})$. 11, 37, 70, 79, 84, 86, 114

denial-of-service attack A denial-of-service attack uses data poisoning to steer the training of a local (client) model such that it performs poorly for typical data points 112

differentiable A function $f : \mathbb{R}^d \to \mathbb{R}$ is differentiable if it has a gradient $\nabla f(\mathbf{x})$ everywhere (for every $\mathbf{x} \in \mathbb{R}^d$) [43]. 26, 28, 29, 32–34, 58, 71, 127, 141

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. 47, 50, 127

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 $\mathbf{A}\mathbf{x} = \lambda \mathbf{x}$. 60–62

empirical graph An empirical graph is an undirected weighted graph whose nodes represent local datasets and models for them. Each node holds a local parameter vector that parametrizes a hypothesis. 12, 53, 54, 67–70, 73, 74, 76–80, 83, 87, 88, 112, 120, 133, 135

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. 40, 42–44, 50, 52–54, 56, 57, 79, 135, 141

empirical risk minimization Empirical risk minimization is the optimization problem of finding the hypothesis with minimum average loss

(empirical risk) on a given set of data points (the training set). Many ML methods are special cases of empirical risk minimization. 12, 38, 41–46, 48–51, 55–58, 61, 63, 106, 119, 124, 134, 135, 138, 142

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 $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^d$ [43]. 70, 125, 138

explainability We define the (subjective) explainability of a ML method as the level of predictability (or lack of uncertainty) in its predictions delivered to a specific human user. Quantitative measures for the (subjective) explainability can be obtained via probabilistic models for the data fed into the ML method [26, 90]. 18, 19, 25, 26, 129

explainable empirical risk minimization An instance of structural risk minimization that adds a regularization term to the training error in ERM. The regularization term is chosen to favour hypotheses that are intrinsically explainable for a user. 85

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” [91–93]. However, this book makes consequent use of the term features for low-level properties

of data points that can be measured easily. 7, 8, 16–22, 24, 28, 31, 37, 40, 44–46, 49, 51, 52, 55, 56, 61, 68, 87, 103, 106, 108, 109, 111, 112, 118, 121, 125, 131–133, 140

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 oginal 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. If the feature map delivers two new feautres, we can depict data points in a scatterplot. 21, 22, 37, 61, 106–108, 111

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 $\mathbf{X} := \big(\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)}\big)$. 60

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. 7, 17, 20, 129

federated averagng (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 [10] consider federated averaging methods where the local model training is implemented by running several GD steps 94, 96, 97

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, 2, 9–13, 17, 19, 50, 53, 54, 63, 69–72, 74, 76, 79, 87–89, 91, 100, 102, 112, 116, 120–122, 125

Finnish Meteorological Institute The Finnish Meteorological Institute is a government agency responsible for gathering and reporting weather data in Finland. 78

flow-based clustering Flow-basted clustering groups the nodes of an undirected graph by applying k-means clustering to node-wise feature vectors. These feature vectors are built from flows between carefully selected source and destination nodes [62]. 83

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. 11

generalized total variation Generalized total variation measures the changes of vector-valued node attributes of a graph. 74, 76, 77, 79–85, 89, 92–95, 105, 107, 108, 127

gradient For a real-valued function $f : \mathbb{R}^d \to \mathbb{R} : \mathbf{w} \mapsto f(\mathbf{w})$, a vector $\mathbf{a}$ such that $\lim_{\mathbf{w} \to \mathbf{w}'} \frac{f(\mathbf{w}) - \left(f(\mathbf{w}') + \mathbf{a}^T(\mathbf{w} - \mathbf{w}')\right)}{\|\mathbf{w} - \mathbf{w}'\|} = 0$ is referred to as the gradient of f at $\mathbf{w}'$. If such a vector exists it is denoted $\nabla f(\mathbf{w}')$ or $\nabla f(\mathbf{w})\big|_{\mathbf{w}'}$. 9, 10, 12, 34, 57–59, 62, 71, 123, 127, 140

gradient descent (GD) Gradient descent is an iterative method for finding the minimum of a differentiable function $f(\mathbf{w})$. 34, 56–64, 71, 126, 131, 137, 140, 141

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 $\nabla f(\mathbf{w}')$, $\mathbf{w}' \mapsto \mathbf{w}' - \alpha \nabla f(\mathbf{w}')$. 56, 88

gradient-based method Gradient-based methods are iterative algorithms for finding the minimum (or maximum) of a differentiable objective function of a parameter vector. These algorithms construct a sequence of approximations to an optimal parameter vector whose function value is minimal (or maximal). As their name indicates, gradient-based methods use the gradients of the objective function evaluated during previous iterations to construct a new (hopefully) improved approximation of an optimal parameter vector. 25, 26, 34, 56, 72

GTV minimization GTV minimization is an instance of RERM using the GTV of local paramter vectors as regularization term. 2, 12, 13, 69, 74, 79, 80, 82–89, 91, 92, 112

high-dimensional regime A ML method or problem belongs to the high-dimensional regime if the effective dimension of the model is larger

than the number of available (labeled) data points. For example, linear regression belongs to the high-dimensional regime whenever the number d of features used to characterize data points is larger than the number of data points in the training set. Another example for the high-dimensional regime are deep learning methods that use a hypothesis space generated by a ANN with much more tunable weights than the number of data points in the training set. The recent field of high-dimensional statistics uses probability theory to analyze ML methods in the high-dimensional regime [36, 94]. 72

hinge loss Consider a data point that is characterized by a feature vector $\mathbf{x} \in \mathbb{R}^d$ and a binary label $y \in \{-1, 1\}$. The hinge loss incurred by a specific hypothesis h is defined as (6). A regularized variant of the hinge loss is used by the support vector machine (SVM) to learn a linear classifier with maximum margin between the two classes (see Figure **??**). 33, 34, 142

Huber loss The Huber loss is a mixture of the squared error loss and the absolute value of the prediction error. 28, 29

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. 8, 16, 18–20, 23–33, 36–40, 42–46, 48–51, 56, 62, 63, 70–73, 85, 102, 104, 112, 118–120, 133, 135, 140, 142

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 we expect a linear relation between feature values and label, a resonable first candidate for the hypothesis space is the space of linear maps (**??**). 8, 16, 19, 20, 22–25, 36–38, 41, 43, 44, 48–51, 56, 67, 70, 84, 85, 123, 132, 134, 142, 143

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. 10, 32, 38, 40, 44, 52, 61, 75, 104, 129, 130, 134, 135, 139

i.i.d. assumption The i.i.d. assumption interprets data points of a dataset as the realizations of i.i.d. RVs. 38, 39, 41, 42, 44, 73, 74, 135, 139

interpretability Within this book, we use the term interpretability as a synomym for explainability. 20

kernel Consider data points characterized by features $\mathbf{x} \in \mathcal{X}$ with values in some arbirtrary feature space. A kernel is a map that assignes each pair of features $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$ a real number. This number is interpretred as a measure for the similarity between $\mathbf{x}$ and $\mathbf{x}'$. For more details about

kernels and the resulting kernel methods, we refer to the literature [95,96]. 21

Kullback-Leibler divergence The Kullback–Leibler divergence is a quantitive measure for how much one probability distribution is different from another probability distribution [87]. 75

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" [91–93]. 16–19, 23, 28, 40, 41, 43–46, 49, 51, 52, 55, 68, 103, 106, 111, 118, 121, 131–133, 140

label space Consider a ML application that involves data points characterized by features and labels. The label space of a given ML application or method is constituted by all potential values that the label of a data point can take on. A popular choice for the label space in regression problems (or methods) is $\mathcal{Y} = \mathbb{R}$. Binary classification problems (or 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”$\}$ 17, 20, 29, 31, 35–37, 118, 129

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. 32, 40, 41

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 (see **??**). 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. Within this book we use the term learning rate mostly as a synonym for the step size of (a variant of) GD 59–61, 64

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. 53, 54, 88

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 $\mathbf{w}$. 72

linear classifier A classifier $h(\mathbf{x})$ maps the feature vector $\mathbf{x} \in \mathbb{R}^d$ of a data point to a predicted label $\hat{y} \in \mathcal{Y}$ out of a finite set of label values $\mathcal{Y}$. We can characterize such a classifier equivalently by the decision regions $\mathcal{R}_a$, for every possible label value $a \in \mathcal{Y}$. Linear classifiers are such that

the boundaries between the regions $\mathcal{R}_a$ are hyperplanes in $\mathbb{R}^d$. 128

linear model This book uses the term linear model in a very specific sense. In particular, a linear model is the hypothesis space which consists of all linear maps,

$$\mathcal{H}^{(d)} := \left\{ h(\mathbf{x}) = \mathbf{w}^T \mathbf{x} : \mathbf{w} \in \mathbb{R}^d \right\}. \tag{79}$$

Note that (79) defines an entire family of hypothesis spaces, which is indexed by the number d of features that are linearly combined to form the prediction $h(\mathbf{x})$. The design choice of d is guided by computational aspects (smaller d means less computation), statistical aspects (larger d might reduce prediction error) and interpretability (linear models with few features are considered interpretable). 19–22, 24, 42, 44–47, 50, 55, 70, 72, 95

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 typically measured using the average squared error loss incurred on a set of labeled data points (the training set). 42, 48, 52, 55, 56, 58, 61, 62, 72, 92, 95, 97, 131

local dataset The concept of a local dataset is somewhat half-way between the concept of a data point and a dataset. Similar to the basic notion of a dataset, also a local dataset consists of several individual data points which are characterized by features and label. In constrast 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 arising from probabilistic models or communcation infrastructure and are encoded in the edges of an empirical graph. 1, 9–11, 53, 54, 67–76, 78, 79, 83, 88, 89, 134, 135

logistic loss Consider a data point that is characterized by the features $\mathbf{x}$ and a binary label $y \in \{-1, 1\}$. We use a hypothesis h to predict the label y solely from the features $\mathbf{x}$. The logistic loss incurred by a specific hypothesis h is defined as (7). 26, 33–36, 134

logistic regression Logistic regression aims at learning a linear hypothesis map to predict a binary label based on numeric features of a data point. The quality of a linear hypothesis map (classifier) is measured using its average logistic loss on some labeled data points (the training set). 33

loss With a slight abuse of language, we use the term loss either for loss function itself or for its value for a specific pair of data point and hypothesis. 23–29, 31–33, 35, 36, 38–40, 51–53, 61, 70, 72, 93, 105, 118, 119, 136, 139, 140, 142

loss function A loss function is a map

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

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 $L\big((\mathbf{x}, y), h\big)$. The loss value $L\big((\mathbf{x}, y), h\big)$ quantifies the discrepancy between the true label y and the predicted label $h(\mathbf{x})$. Smaller (closer to zero) values $L\big((\mathbf{x}, y), h\big)$ mean a smaller discrepancy between predicted label and

true label of a data point. Figure 4 depicts a loss function for a given data point, with features $\mathbf{x}$ and label y, as a function of the hypothesis $h \in \mathcal{H}$. 12, 16, 23–28, 31–33, 35, 40–42, 48, 50–52, 56, 70, 71, 73, 74, 79, 81–83, 85, 87, 89, 94, 106, 133, 134, 139

maximum likelihood Consider data points $\mathcal{D} = \{\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(m)}\}$ that are interpreted as realizations of i.i.d. RVs with a common probability distribution $p(\mathbf{z}; \mathbf{w})$ which depends on a parameter vector $\mathbf{w} \in \mathcal{W} \subseteq \mathbb{R}^n$. Maximum likelihood methods aim at finding a parameter vector $\mathbf{w}$ such that the probability (density) $p(\mathcal{D}; \mathbf{w}) = \prod_{r=1}^{m} p(\mathbf{z}^{(r)}; \mathbf{w})$ of observing the data is maximized. Thus, the maximum likelihood estimator is obtained as a solution to the optimization problem $\max_{\mathbf{w} \in \mathcal{W}} p(\mathcal{D}; \mathbf{w})$. 25, 28

metric A metric refers to a loss function that is used solely for the final performance evaluation of a learnt hypothesis. The metric is typically a loss function that has a "natural" interpretation (such as the 0/1 loss (4)) but is not a good choice to guide the learning process, e.g., via ERM. For ERM, we typically prefer loss functions that depend smoothly on the (parameters of the) hypothesis. Examples for such smooth loss functions include the squared error loss (2) and the logistic loss (7). 27

model We use the term model as a synonym for hypothesis space 16, 19, 21, 23, 50, 51, 67, 127, 129, 136

networked data Networked data refers to a collection of local datasets that are related by some notion of pair-wise similarities. We present net-

worked data using an empirical graph whose nodes carry local datasets and edges represent pair-wise similarities betweem them. 9, 69, 83

networked federated learning Networked federated learning refers to methods that learn personalized models in a distributed fashion from local datasets that are related by an intrinsic network structure. 84

networked model A networked model is a collection of local models or hypothesis spaces assigned to the nodes of an empirical graph. 70

non-smooth We refer to a function as non-smooth if it is not smooth [97]. 26

outlier Many ML methods are motivated by the i.i.d. assumption which interprets data points as realizations of i.i.d. RVs with a common probability distribution. The i.i.d. assumption is useful for applications where the statistical properties of the data generation process are stationary (time-invariant). However, in some applications the data consists of a majority of "regular" data points that conform with an i.i.d. assumption and a small number of data points that have fundamentally different statistical properties compared to the regular data points. We refer to a data point that substantially deviates from the statistical properties of the majority of data points as an outlier. Different methods for outlier detection use different measures for this deviation. 25, 29

overfitting Consider a ML methods that uses ERM to learn a hypothesis with minimum empirical risk on a given training set. Such a method is called overfitting if it learns hypothesis with small empirical risk on the

training set but unacceptly large loss outside the training set. 50, 51, 63, 125

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. 56, 93, 103, 105, 107

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 $\mathbf{x} \in \mathbb{R}^d$. 60, 65

probabilistic model A probabilistic model interprets data as realizations of RVs with some joint probability distribution.This joint probability distribution typically involves parameters which have to be manually chosen or learnt via staitstical inference methods [32]. 38, 39, 44, 46, 48, 75, 104, 124, 140

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 $p(x \in \mathcal{B}) = \int_{\mathcal{B}} p(x')dx'$ [35, 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 $p(\mathbf{x} \in \mathcal{R}) = \int_{\mathcal{R}} p(\mathbf{x}')dx'_1 \ldots dx'_d$ [35, Ch. 3]. 129

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 [29, 34]. 25, 32, 38–44, 49, 61, 73, 75, 104, 117, 129, 130, 134–136, 139, 140

projected GD Projected GD is obtained from basic GD by projecting the result of a gradient step into a given constrain set. 95

proximity operator Given a convex function and a vector $\mathbf{x}$, we define the proximity operator as

$$\mathbf{prox}_{L_i(\cdot),2\lambda}(\mathbf{w}'') := \operatorname*{argmin}_{\mathbf{w}\in\mathbb{R}^d} f(\mathbf{w}) + (\rho/2)\left\|\mathbf{w}-\mathbf{w}'\right\|_2^2 \text{ with } \rho > 0.$$

Convex functions for which the proximity operator can be computed efficiently are sometimes referred to as “proximable” or “simple” [98]. 54, 55, 71, 98, 117

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 $a \in \mathbb{R}$. 56

random variable (RV) A random variable is a mapping from a probability space $\mathcal{P}$ to a value space [29]. The probability space, whose element are elementary events, is equipped with a probability measure that assigns a proability to subsets of $\mathcal{P}$. A binary random variable maps elementary events to a set containing two different value, such as $\{-1, 1\}$ 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 [29,34]. 32, 38, 40, 41, 44, 45, 48, 49, 61, 75, 104, 117, 119, 121, 129, 130, 134–137, 139, 143

regularization Regularization techniques modify the ERM principle such that the learnt hypothesis performs well also outside the training set which is used in ERM. One specific approach to regularization is by adding a penalty or regularization term to the objective function of ERM (which is the average loss on the training set). This regulazation term can be interpreted as an estimate for the increase in the expected loss (risk) compared to the average loss on the training set. 11, 50, 51, 53, 55, 61–63, 67, 72, 74, 81, 85, 87, 89, 127

regularized empirical risk minimization Regularized empirical risk min-

imization is the problem of finding the hypothesis that optimally balances the average loss (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. 52–55, 82, 96, 97, 127

ridge regression Ridge regression aims at learning the weights $\mathbf{w}$ of linear hypothesis map $h^{(\mathbf{w})}(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$. The quality of a particular choice for the weights $\mathbf{w}$ is measured by the sum of two terms (see (**??**)). The first term is the average squared error loss incurred by $h^{(\mathbf{w})}$ on a set of labeled data points (the training set). The second term is the scaled squared Euclidean norm $\lambda\|\mathbf{w}\|_2^2$ with a regularization parameter $\lambda > 0$. 62, 72

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 $L\big((\mathbf{x}, y), h\big)$. If we interpret data points as realizations of i.i.d. RVs, also the $L\big((\mathbf{x}, y), h\big)$ 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 $\mathbb{E}\big\{L\big((\mathbf{x}, y), h\big)\big\}$. Note that the risk of h depends on both, the specific choice for the loss function and the common probability distribution of the data points. 38–41, 43–46, 49, 50, 57, 119

sample size The number of individual data points contained in a dataset that is obtained from realizations of i.i.d. RVs. 32, 36, 45

scatterplot A visualization technique that depicts data points by markers in a two-dimensional plane. 125

smooth We refer to a real-valued function as smooth if it is differentiable and its gradient is continuous [31,97]. In particular, a differentiable function $f(\mathbf{w})$ is refered to as β-smooth if the gradient $\nabla f(\mathbf{w})$ is Lipschitz continuous with Lipschitz constant β, i.e., $\|\nabla f(\mathbf{w}) - \nabla f(\mathbf{w}')\| \leq \beta \|\mathbf{w} - \mathbf{w}'\|$. 135

spectral clustering Spectral clustering groups the nodes of an undirected graph by applying k-means clustering to node-wise feature vectors. These feature vectors are built from the eigenvectors the graph Laplacian matrix [61,62]. 83

squared error loss The squared error loss is widely used to measure the quality of a hypothesis when predicting a numeric label from the features of a data point. 27–31, 48, 55, 86, 94, 95

statistical aspects By statistical aspects of a ML method, we refer to (properties of) the probability distribution of its ouput given a probabilistic model for the data fed into the method. 16, 18–20, 24, 26, 32, 41, 42, 69, 71, 74, 132

stochastic gradient descent Stochastic GD is obtained from GD by replacing the gradient of the objective function by a noisy (or stochastic) estimate. 57

stopping criterion Many ML methods use iterative algorithms that construct a sequence of model parameters (such as the weights of a linear

map or the weights of an ANN) that (hopefully) converge to an optimal choice for the model parameters. In practice, given finite computational resources, we need to stop iterating after a finite number of times. A stopping criterion is any well-defined condition required for stopping iterating. 59, 64, 92–94, 96, 97, 105, 107

strongly convex A continuously differentiable real-valued function $f(\mathbf{x})$ is strongly convex with coefficient σ if $f(\mathbf{y}) \geq f(\mathbf{x}) + \nabla f(\mathbf{x})^T(\mathbf{y}-\mathbf{x}) + (\sigma/2)\left\|\mathbf{y}-\mathbf{x}\right\|_2^2$ [97], [30, Sec. B.1.1.]. 57

structural risk minimization Structural risk minimization is the problem of finding the hypothesis that optimally balances the average loss (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. 131

subgradient For a real-valued function $f : \mathbb{R}^d \to \mathbb{R} : \mathbf{w} \mapsto f(\mathbf{w})$, a vector $\mathbf{a}$ such that $f(\mathbf{w}) \geq f(\mathbf{w}') + \left(\mathbf{w}-\mathbf{w}'\right)^T\mathbf{a}$ is referred to as a subgradient of f at $\mathbf{w}'$ [99, 100]. 34

subgradient descent Subgradient descent is a generalization of GD that is obtained by using sub-gradients (instead of gradients) to construct local approximations of an objective function such as the empirical risk $\widehat{L}\left(h^{(\mathbf{w})}\middle|\mathcal{D}\right)$ as a function of the parameters $\mathbf{w}$ of a hypothesis $h^{(\mathbf{w})}$. 34

support vector machine A binary classification method for learning a linear map that maximally seperates data points the two classes in the feature space ("maximum margin"). Maximizing this separation is

equivalent to minimizing a regularized variant of the hinge loss (6). 128

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. 43, 50, 142

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). 29, 30, 38, 43–46, 48, 50–52, 67, 72, 114, 124, 128, 132, 133, 135, 136, 138, 139, 141, 142

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 $\mathcal{D}$. 59

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. 50, 142

validation set A set of data points that has not been used as training set in ERM to train 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. The comparison between training and validation error informs

adaptations of the ML method (such as using a different hypothesis space). 114, 142

variance The variance of a real-valued RV x is defined as the expectation $\mathbb{E}\{(x - \mathbb{E}\{x\})^2\}$ of the squared difference x and its expectation $\mathbb{E}\{x\}$. We extend this definition to vector-valued RVs $\mathbf{x}$ as $\mathbb{E}\{\|\mathbf{x} - \mathbb{E}\{\mathbf{x}\}\|_2^2\}$. 44, 45

References

[1] A. Jung, *Machine Learning: The Basics*, 1st ed. Springer Singapore, Feb. 2022.

[2] S. Cui, A. Hero, Z.-Q. Luo, and J. Moura, Eds., *Big Data over Networks*. Cambridge Univ. Press, 2016.

[3] 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.

[4] 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

[5] 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

[6] 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

[7] M. E. J. Newman, *Networks: An Introduction*. Oxford Univ. Press, 2010.

[8] A. Barabási, N. Gulbahce, and J. Loscalzo, "Network medicine: a network-based approach to human disease," *Nature Reviews Genetics*, vol. 12, no. 56, 2011.

[9] K. Grantz, H. Meredith, D. Cummings, C. Metcalf, B. Grenfell, J. Giles, S. Mehta, S. Solomon, A. Labrique, N. Kishore, C. Buckee, and A. Wesolowski, "The use of mobile phone data to inform analysis of COVID-19 pandemic epidemiology," *Nature Communications*, vol. 11, no. 1, p. 4961, 2020. [Online]. Available: https://doi.org/10.1038/s41467-020-18190-5

[10] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, A. Singh and J. Zhu, Eds., vol. 54. Fort Lauderdale, FL, USA: PMLR, 20–22 Apr 2017, pp. 1273–1282. [Online]. Available: http://proceedings.mlr.press/v54/mcmahan17a.html

[11] 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.

[12] 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.

[13] 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.

[14] 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

[15] 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

[16] D. Tse and P. Viswanath, *Fundamentals of Wireless Communication*. Cambridge University Press, 2005.

[17] 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

[18] 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.

[19] 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.

[20] 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

[21] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge, UK: Cambridge Univ. Press, 2004.

[22] D. P. Bertsekas and J. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods*. Athena Scientific, 2015.

[23] M. Conti and D. Moretti, "System level analysis of the bluetooth standard," in *Design, Automation and Test in Europe*, 2005, pp. 118–123 Vol. 3.

[24] M. Ribeiro, S. Singh, and C. Guestrin, ""Why should i trust you?": Explaining the predictions of any classifier," in *Proc. 22nd ACM SIGKDD*, Aug. 2016, pp. 1135–1144.

[25] C. Molnar, *Interpretable Machine Learning - A Guide for Making Black Box Models Explainable*. [online] Available: https://christophm.github.io/interpretable-ml-book/., 2019.

[26] A. Jung and P. Nardelli, "An information-theoretic approach to personalized explainable machine learning," *IEEE Sig. Proc. Lett.*, vol. 27, pp. 825–829, 2020.

[27] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[28] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*, ser. Springer Series in Statistics. New York, NY, USA: Springer, 2001.

[29] P. Billingsley, *Probability and Measure*, 3rd ed. New York: Wiley, 1995.

[30] D. P. Bertsekas, *Convex Optimization Algorithms*. Athena Scientific, 2015.

[31] S. Bubeck, "Convex optimization. algorithms and complexity." in *Foundations and Trends in Machine Learning*. Now Publishers, 2015, vol. 8.

[32] E. L. Lehmann and G. Casella, *Theory of Point Estimation*, 2nd ed. New York: Springer, 1998.

[33] M. J. Wainwright and M. I. Jordan, *Graphical Models, Exponential Families, and Variational Inference*, ser. Foundations and Trends in Machine Learning. Hanover, MA: Now Publishers, 2008, vol. 1, no. 1–2.

[34] R. Gray, *Probability, Random Processes, and Ergodic Properties*, 2nd ed. New York: Springer, 2009.

[35] D. Bertsekas and J. Tsitsiklis, *Introduction to Probability*, 2nd ed. Athena Scientific, 2008.

[36] M. Wainwright, *High-Dimensional Statistics: A Non-Asymptotic Viewpoint*. Cambridge: Cambridge University Press, 2019.

[37] J. Cho and H. White, "Generalized runs tests for the iid hypothesis," *Journal of Econometrics*, vol. 162, no. 2, pp. 326–344, 2011. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0304407611000285

[38] P. J. Huber, *Robust Statistics*. New York: Wiley, 1981.

[39] T. W. Anderson, *An Introduction to Multivariate Statistical Analysis*. Wiley, 2003.

[40] L. Breiman and D. Freedman, "How many variables should be entered in a regression equation?" *Journal of the American Statistical Association*, vol. 78, no. 381, pp. 131–136, 1983.

[41] J. Mourtada, "Exact minimax risk for linear least squares, and the lower tail of sample covariance matrices," *The Annals of Statistics*, vol. 50, no. 4, pp. 2157 – 2178, 2022. [Online]. Available: https://doi.org/10.1214/22-AOS2181

[42] N. Parikh and S. Boyd, "Proximal algorithms," *Foundations and Trends in Optimization*, vol. 1, no. 3, pp. 123–231, 2013.

[43] W. Rudin, *Principles of Mathematical Analysis*, 3rd ed. New York: McGraw-Hill, 1976.

[44] G. Strang, *Introduction to Linear Algebra*, 5th ed. Wellesley-Cambridge Press, MA, 2016.

[45] 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.

[46] A. Papoulis and S. U. Pillai, *Probability, Random Variables, and Stochastic Processes*, 4th ed. New York: Mc-Graw Hill, 2002.

[47] A. Jung, "Networked exponential families for big data over networks," *IEEE Access*, vol. 8, pp. 202 897–202 909, 2020.

[48] H. Ambos, N. Tran, and A. Jung, "The logistic network lasso," *arXiv*, 2018.

[49] ——, "Classifying big data over networks via the logistic network lasso," in *Proc. 52nd Asilomar Conf. Signals, Systems, Computers*, Oct./Nov. 2018.

[50] A. Jung and N. Tran, "Localized linear regression in networked data," *IEEE Sig. Proc. Lett.*, vol. 26, no. 7, Jul. 2019.

[51] P. J. Brockwell and R. A. Davis, *Time Series: Theory and Methods*. Springer New York, 1991.

[52] V. Kalofolias, "How to learn a graph from smooth signals," in *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, A. Gretton and C. C. Robert, Eds., vol. 51. Cadiz, Spain: PMLR, 09–11 May 2016, pp. 920–929.

[53] X. Dong, D. Thanou, M. Rabbat, and P. Frossard, "Learning graphs from data: A signal representation perspective," *IEEE Signal Processing Magazine*, vol. 2019, no. 3, May 2019.

[54] D. Hallac, J. Leskovec, and S. Boyd, "Network lasso: Clustering and optimization in large graphs," in *Proc. SIGKDD*, 2015, pp. 387–396.

[55] Y. SarcheshmehPour, M. Leinonen, and A. Jung, "Federated learning from big data over networks," in *Proc. IEEE Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP), preprint: https://arxiv.org/pdf/2010.14159.pdf*, 2021.

[56] A. Jung, N. Tran, and A. Mara, "When is Network Lasso Accurate?" *Front. Appl. Math. Stat.*, vol. 3, Jan. 2018.

[57] B. Nadler, N. Srebro, and X. Zhou, "Statistical analysis of semi-supervised learning: The limit of infinite unlabelled data," in *Advances in Neural Information Processing Systems 22*, 2009, pp. 1330–1338.

[58] F. Bach, R. Jenatton, J. Mairal, and G. Obozinski, "Optimization with sparsity-inducing penalties," *Found. Trends Mach. Learn.*, vol. 4, no. 1, pp. 1–106, Jan. 2012.

[59] L. Jacob, J.-P. Vert, and F. Bach, "Clustered multi-task learning: A convex formulation," in *Advances in Neural Information Processing Systems*, D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, Eds., vol. 21. Curran Associates, Inc., 2009. [Online]. Available: https://proceedings.neurips.cc/paper/2008/file/fccb3cdc9acc14a6e70a12f74560c026-Paper.pdf

[60] T. Evgeniou, C. Micchelli, and M. Pontil, "Learning multiple tasks with kernel methods," *Journal of Machine Learning Research*, vol. 6, no. 21, pp. 615–637, 2005. [Online]. Available: http://jmlr.org/papers/v6/evgeniou05a.html

[61] U. von Luxburg, "A tutorial on spectral clustering," *Statistics and Computing*, vol. 17, no. 4, pp. 395–416, Dec. 2007.

[62] Y. SarcheshmehPour, Y. Tian, L. Zhang, and A. Jung, "Flow-based clustering and spectral clustering: A comparison," in *2021 55th Asilomar Conference on Signals, Systems, and Computers*, 2021, pp. 1292–1296.

[63] D. Sun, K.-C. Toh, and Y. Yuan, "Convex clustering: Model, theoretical guarantee and efficient algorithm," *Journal of Machine Learning Research*, vol. 22, no. 9, pp. 1–32, 2021. [Online]. Available: http://jmlr.org/papers/v22/18-694.html

[64] K. Pelckmans, J. D. Brabanter, J. Suykens, and B. D. Moor, "Convex clustering shrinkage," in *PASCAL Workshop on Statistics and Optimization of Clustering Workshop*, 2005.

[65] C. G. Atkeson, A. W. Moore, and S. Schaal, "Locally weighted learning," *Artif. Intell. Rev.*, vol. 11, no. 1–5, pp. 11–73, feb 1997. [Online]. Available: https://doi.org/10.1023/A:1006559212014

[66] V. Mothukuri, R. M. Parizi, S. Pouriyeh, Y. Huang, A. Dehghantanha, and G. Srivastava, "A survey on security and privacy of federated learning," *Future Generation Computer Systems*, vol. 115, pp. 619–640,

2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X20329848

[67] S. R. Pokhrel and J. Choi, "Federated learning with blockchain for autonomous vehicles: Analysis and design challenges," *IEEE Transactions on Communications*, vol. 68, no. 8, pp. 4734–4746, 2020.

[68] O. Chapelle, B. Schölkopf, and A. Zien, Eds., *Semi-Supervised Learning*. Cambridge, Massachusetts: The MIT Press, 2006.

[69] A. Ghosh, J. Chung, D. Yin, and K. Ramchandran, "An efficient framework for clustered federated learning," *IEEE Transactions on Information Theory*, vol. 68, no. 12, pp. 8076–8091, 2022.

[70] L. Z. Y. SarcheshmehPour, Y. Tian and A. Jung, "Networked federated learning," *arXiv e-prints*, 2022.

[71] H. Ludwig and N. Baracaldo, Eds., *Federated Learning: A Comprehensive Overview of Methods and Applications*. Springer, 2022.

[72] 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

[73] 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

[74] ——, *Horizontal Federated Learning*. Cham: Springer International Publishing, 2020, pp. 49–67. [Online]. Available: https://doi.org/10.1007/978-3-031-01585-4_4

[75] E. Commission, C. Directorate-General for Communications Networks, and Technology, *The Assessment List for Trustworthy Artificial Intelligence (ALTAI) for self assessment*. Publications Office, 2020.

[76] 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

[77] K. Chaudhuri, C. Monteleoni, and A. Sarwate, "Differentially private empirical risk minimization," *J. Mach. Learn. Res.*, vol. 12, pp. 1069–1109, Mar. 2011.

[78] H. Hsu, N. Martinez, M. Bertran, G. Sapiro, and F. P. Calmon, "A survey on statistical, information, and estimation—theoretic views on privacy," *IEEE BITS the Information Theory Magazine*, vol. 1, no. 1, pp. 45–56, 2021.

[79] P. Samarati, "Protecting respondents identities in microdata release," *IEEE Transactions on Knowledge and Data Engineering*, vol. 13, no. 6, pp. 1010–1027, 2001.

[80] L. Sweeney, "k-anonymity: a model for protecting privacy," *International Journal on Uncertainty, Fuzziness and Knowledge-based Systems*, vol. 10, no. 5, pp. 557–570, 2002.

[81] 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.

[82] 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.

[83] N. M. Müller, S. Roschmann, and K. Böttinger, "Defending Against Adversarial Denial-of-Service Data Poisoning Attacks," *arXiv e-prints*, p. arXiv:2104.06744, April 2021.

[84] H. Wang, K. Sreenivasan, S. Rajput, H. Vishwakarma, S. Agarwal, J.-y. Sohn, K. Lee, and D. Papailiopoulos, "Attack of the tails: Yes, you really can backdoor federated learning," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 16 070–16 084. [Online]. Available: https://proceedings.neurips.cc/paper/2020/file/b8ffa41d4e492f0fad2f13e29e1762eb-Paper.pdf

[85] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk,

M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with numpy," *Nature*, vol. 585, no. 7825, pp. 357–362, 2020. [Online]. Available: https://doi.org/10.1038/s41586-020-2649-2

[86] 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.

[87] T. M. Cover and J. A. Thomas, *Elements of Information Theory*, 2nd ed. New Jersey: Wiley, 2006.

[88] C. E. Shannon, "Communication in the presence of noise," 1948.

[89] P. Halmos, *Naive set theory*. Springer-Verlag, 1974.

[90] A. Jung, "Explainable empiricial risk minimization," *submitted to IEEE Sig. Proc. Letters (preprint: https://arxiv.org/pdf/2009.01492.pdf)*, 2020.

[91] D. Gujarati and D. Porter, *Basic Econometrics*. Mc-Graw Hill, 2009.

[92] Y. Dodge, *The Oxford Dictionary of Statistical Terms*. Oxford University Press, 2003.

[93] B. Everitt, *Cambridge Dictionary of Statistics*. Cambridge University Press, 2002.

[94] P. Bühlmann and S. van de Geer, *Statistics for High-Dimensional Data*. New York: Springer, 2011.

[95] C. Lampert, "Kernel methods in computer vision," *Foundations and Trends in Computer Graphics and Vision*, 2009.

[96] B. Schölkopf and A. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. Cambridge, MA, USA: MIT Press, Dec. 2002.

[97] 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

[98] 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.

[99] D. P. Bertsekas, A. Nedic, and A. E. Ozdaglar, *Convex Analysis and Optimization*. Athena Scientific, 2003.

[100] D. P. Bertsekas, *Nonlinear Programming*, 2nd ed. Belmont, MA: Athena Scientific, June 1999.