Lecture Notes

CS-E4740 Federated Learning

spring 2023 at Aalto University and `https://fitech.io/en/`

ROUGH DRAFT - DO NOT DISTRIBUTE FURTHER!

Dipl.-Ing. Dr.techn. Alexander Jung[*†]

January 16, 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 train personalized models for each local dataset. These 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 generalized total variation minimization (GTVMin). It provides a flexible design principle for FL algorithms and includes some main flavours of FL (vertical FL, horizontal FL and clustered FL) as special cases. We obtain distributed FL algorithms by applying established distributed optimization methods to solve GTVMin.

*

†

Contents

Lists of Symbols

$\mathcal{G} = (\mathcal{V}, \mathcal{E})$	Empirical graph whose nodes $i \in \mathcal{V}$ carry local datasets and local models.
$i \in \mathcal{V}$	A node in the empirical graph that represents a local dataset and 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.
$\mathcal{X}^{(i)}$	The local dataset $\mathcal{X}^{(i)}$ at node $i \in \mathcal{V}$.
m_i	The number of data points (sample size) contained in the local dataset $\mathcal{X}^{(i)}$ at node $i \in \mathcal{V}$.
$\mathbf{x}^{(i,r)}$	The feature vector of the r-th data point in the local dataset $\mathcal{X}^{(i)}$
$\mathbf{v} \in \mathbb{R}^d$	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 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.
$\mathcal{H}$	A hypothesis space which consists of hypothesis maps h.

1 Introduction

Many important application domains generate collections of local datasets that are related via an intrinsic network structure [1]. Two timely application domains that generated such networked data are (i) the high-precision management of pandemics and (ii) the Internet of Things (IoT) [2, 3]. Here, local datasets are generated either by smartphones and wearables or by IoT devices [4, 5]. These local datasets are related via physical contact networks, social networks [6], co-morbidity networks [7], or communication networks [8].

Federated learning (FL) is an umbrella term for distributed optimization techniques to train ML models from collections of local datasets [9–13]. These methods carry out computations, such as gradient computations (see Section 5), for model training right at the location of data generation. This design philosophy is quite 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 location of data generation can be beneficial in several aspects [14].

- **Security.** 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 [11, 12]. By sharing only model updates, FL methods are considered privacy-friendly in the sense of not leaking sensitive information in local datasets [15]. Moreover, FL methods can offer robustness against malicious data perturbation due to its intrinsic averaging or aggregation over large collections of (mostly benign) datasets [16].

- **Parallel Computing.** One important application domain for FL is the personalization of smartphone functionalities (google keyboard) [17]. 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 computing gradients during the training of ML models [18].

- **Trading FLOPS with Bits.** 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 ML model) might be much higher than the computational cost incurred by using the low-complexity devices to (partially) train ML models [19].

- **Personalization.** One key challenge in FL applications is the heterogeneity of local datasets [20, 21]. 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.

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` [22]. We will combine these basic ML methods using

regularization techniques to couple a collection of local ML models and methods. This coupling enables the adaptive pooling of data to obtain sufficiently large training sets.

- **CS-C3240 - Machine Learning.** Teaches basic theory of ML models and methods [23]. This course extends basic ML models by considering networks of local (personalized) ML models. These models are coupled by a network structure that reflects statistical similarities within decentralized collections of local datasets.

- **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. We will use a special class of optimization problems and methods to formulate and solve FL problems.

- **ELEC-E5424 - Convex Optimization.** This course teaches advanced optimisation theory for the important class of convex optimization problems [24]. Convex optimization theory and methods can be used for the study and design of FL algorithms.

1.2 Outline of the Notes

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 GTV minimization 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 security of FL algorithms in the context of backdoor attacks.

1.3 Exercises

Exercise 1.1. Parallel Computing (see Sec. 1.2.3. of [18]). Assume that you have a single computer that is able to sum two number 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 $\mathcal{X} = \{x^{(1)}, \ldots, x^{(100)}\}$.

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 $\mathcal{X}$ using these two computers?

Part I

Plain Old Machine Learning

2 Three Components of ML

Machine learning revolves around computing accurate predictions for some 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 previous predictions. The space of possible hypotheses, from which a machine learning 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, machine learning methods use loss functions to obtain a quantitative measure for the prediction errors (see Section 2.3).

Different machine learning methods use different choices for data points (their features and label), the hypothesis space (or model) and loss function. 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

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 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 `https://www.amazon.de`. The feature space $\mathcal{X}$ collects all possible values that the features of a data point can take on.

Besides their features, data points are characterized by labels 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. ML applications involving a "numeric" $\mathcal{Y}$ (e.g., the real numbers $\mathbb{R}$) are referred to as regression problems. ML applications involving a "discrete" $\mathcal{Y}$ (e.g., a finite set or a countable set such as $\mathbb{N}$ or $\mathbb{Z}$) 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. However, we must clearly define or specific the meaning of data points. 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.

2.2 Design Choice Model

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 choice for the hypothesis space involves a trade-off between computational complexity and statistical properties of the resulting ML methods.

During the rest of this course, we will focus on FL methods that use linear models [23, Ch. 3]. Linear models can be used for data points characterized by d numeric features $x_1, \ldots, x_d \in \mathbb{R}$. It is convenient 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, the linear model is the hypothesis space constituted by all linear

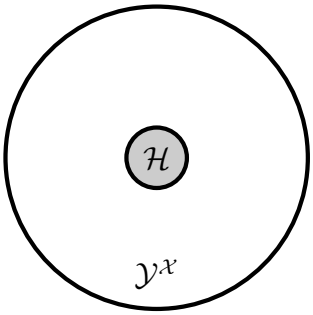

Figure 1: The hypothesis space $\mathcal{H}$ is a (typically very small) subset of the (typically very large) set $\mathcal{Y}^{\mathcal{X}}$ of all possible maps from feature space $\mathcal{X}$ into the label space $\mathcal{Y}$.

maps,

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

Upgrading a Hypothesis Space via Feature Maps. We can use the linear model $\mathcal{H}^{(d)}$ as a building block for constructing other models or hypothesis spaces. For example, we could apply the linear model to transformed features $\mathbf{x}' = \mathbf{\Phi}(\mathbf{x})$ (see Figure 2). The resulting model consists of all maps that are concatenations of linear maps with the feature transformation or map $\mathbf{\Phi}$. One widely used instance of this approach are kernel methods that use feature maps obtained from a kernel function. Figure 3 illustrates deep learning as another example for combining feature transformations with the linear model. Indeed, the input layers of an artificial neural network (ANN) (whose output layer is linear) are nothing but a feature map.

Non-Numeric Features. We can use the linear model (1) 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, there is no obvious

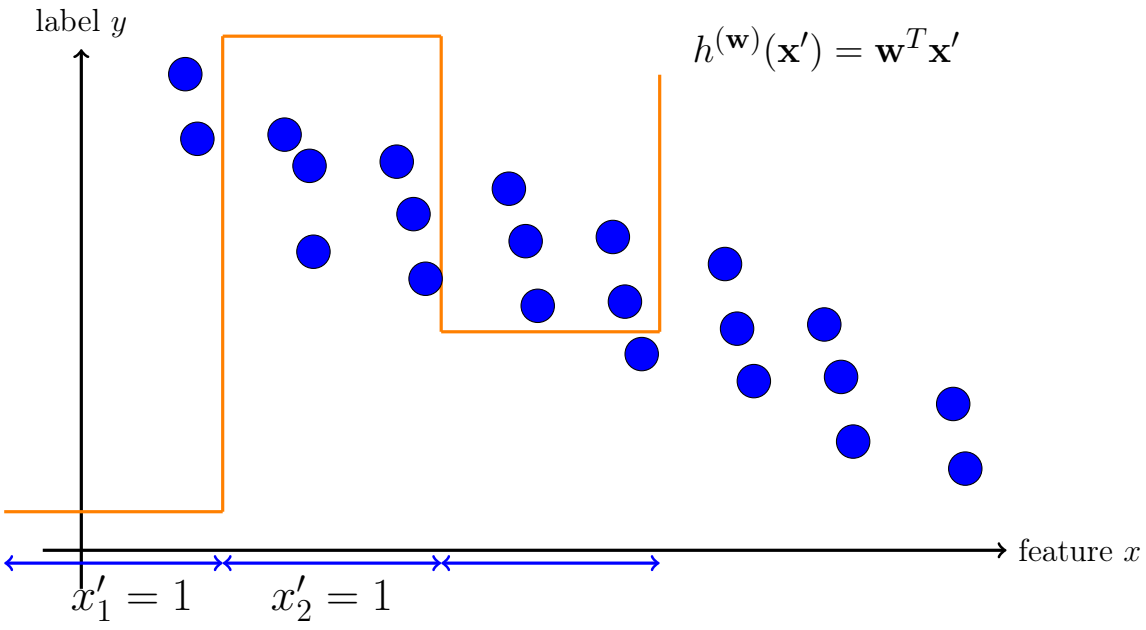

Figure 2: We can extend a linear model (1) by replacing the original feature x of a data point with transformed features $\mathbf{x}' = \mathbf{\Phi}(\mathbf{x})$. These transformed features are obtained by applying a feature map $\mathbf{\Phi}$ to the original features.

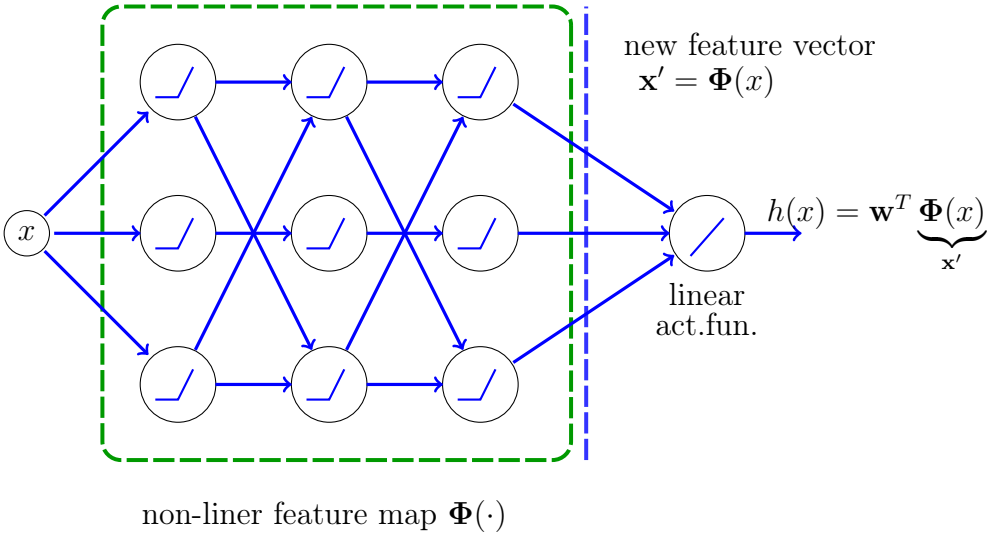

Figure 3: The hypothesis space spanned by an ANN (whose output layer is linear) coincides with the linear model applied to the activations $\mathbf{\Phi}(x)$ of the pen-ultimate layer. We can interpret these activations as transformed features that are obtained by applying the input layers to the original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

natural choice for numeric features. However, since ML methods based on the hypothesis space (1) 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 [25, Chap. 12] for more details). Moreover, [23, Sec. 9.3] shows how to construct numeric features for data points that that are related by some notion of similarity.

2.3 Design Choice Loss

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

ML methods find or learn the best hypothesis out of an entire hypothesis space $\mathcal{H}$ of model. To measure the quality of a hypothesis, ML methods use the concept of a loss function. Formally, 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, 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 low discrepancy between predicted label and true label of a data point. Figure 4 depicts a loss function for a given data point, with features $\mathbf{x}$

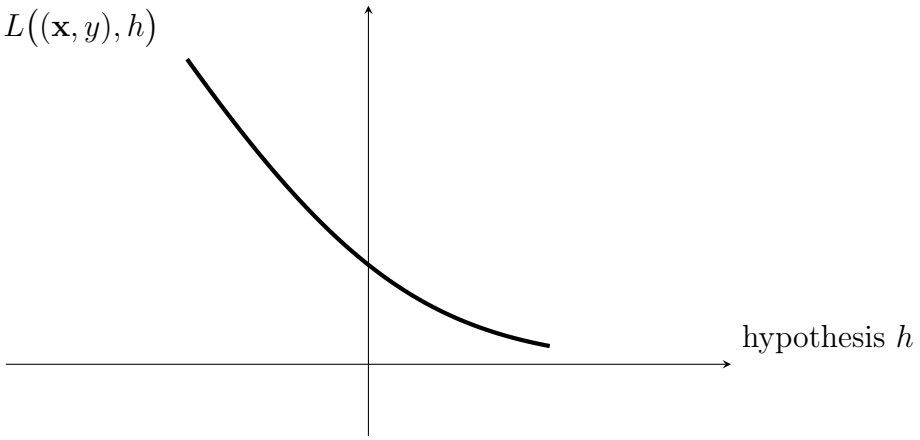

Figure 4: Some loss function $L\big((\mathbf{x},y),h\big)$ for a fixed data point, with features $\mathbf{x}$ and label y, and varying hypothesis h. ML methods try to find (learn) a hypothesis that incurs minimum loss.

and label y, as a function of the hypothesis $h \in \mathcal{H}$.

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 complexity,
- statistical properties,
- 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 [23, 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 [23, 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.

The choice for the loss function used to evaluate the quality of a hypothesis might also be influenced by its interpretability. Section 2.3.2 discusses loss functions for hypotheses that are used to classify data points into two categories. It seems natural to measure the quality of such a hypothesis by the average number of wrongly classified data points, which is precisely the average 0/1 loss (3) (see Section 2.3.2).

In contrast to its appealing interpretation as error-rate, the computational aspects of the average 0/1 loss 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 computationally challenging. Section

2.3.2 introduces the logistic loss as a computationally attractive alternative choice for the loss function in binary classification problems. Learning a hypothesis that minimizes a (average) logistic loss amounts to a smooth convex optimization problem. Section 5 discusses computationally cheap gradient-based methods for solving smooth convex optimization problems.

The above aspects (computation, statistic, interpretability) result typically in conflicting goals for the choice of 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 (what does it mean intuitively if the logistic loss of a hypothesis in a binary classification problem 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.

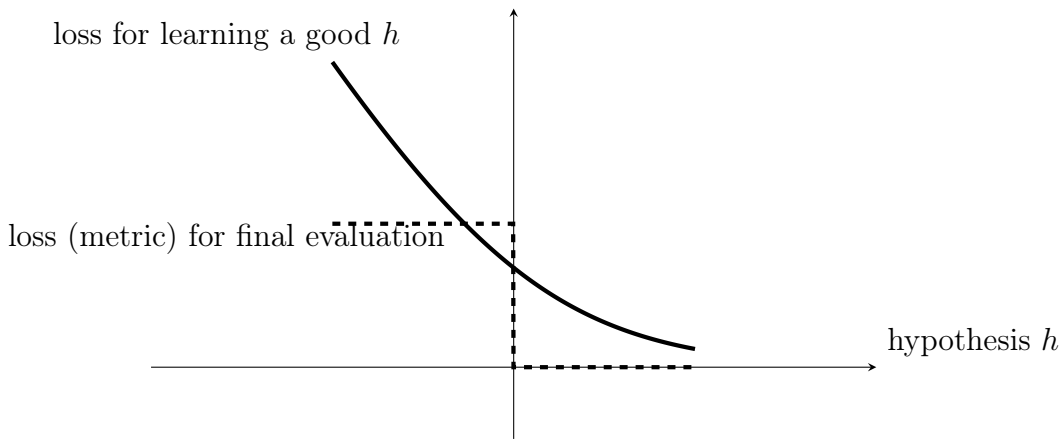

Figure 5: Two different loss functions for a given data point and varying hypothesis h. One of these loss functions (solid curve) is used to learn a good hypothesis by minimizing the loss. The other loss function (dashed curve) is used to evaluate the performance of the learnt hypothesis. The loss function used for this final performance evaluation is sometimes referred to as a metric.

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 $y - \hat{y}$.

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 [26, Sec. 2.6.3].

Another loss function used in regression problems is the absolute error loss $L\big(\big(\mathbf{x}, y\big), h\big) := \big|h(\mathbf{x}) - y\big|$. 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 [23, Sec. 3.3.]). This improved

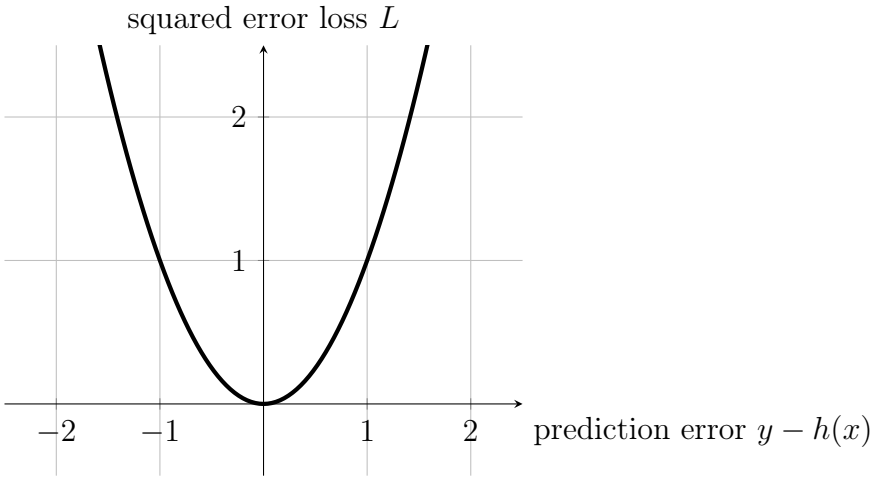

Figure 6: A widely used choice for the loss function in regression problems (with data points having numeric labels) is the squared error loss (2). Note that, for a given hypothesis h, we can evaluate the squared error loss only if we know the features $\mathbf{x}$ and the label y of the data point.

robustness comes at the expense of increased computational complexity of minimizing the non-differentiable absolute error loss 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 aim at learning a hypothesis or classifier that maps the features $\mathbf{x}$ of a data point to a predicted label $\hat{y} \in \mathcal{Y}$.

It is often convenient to implement a classifier by simple post-processing the value $h(\mathbf{x}) \in \mathbb{R}$ of a real-valued hypothesis map. For example, we classify a data point as $\hat{y} = 1$ if $h(\mathbf{x}) > 0$ and $\hat{y} = -1$ otherwise. Thus, the predicted label is obtained from the sign of the value $h(\mathbf{x})$. While the sign of $h(\mathbf{x})$ determines the predicted label $\hat{y}$, we can interpret the absolute value $|h(\mathbf{x})|$ as a measure for the confidence we have about this prediction. We will abuse notation and refer to both, the final classification rule $\mathbf{x} \mapsto \hat{y}$ and the hypothesis $h(\mathbf{x})$ (whose output is compared against a threshold) 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 is typically a poor measure for the quality of a hypothesis $h(\mathbf{x})$ that is used to classify a data point with binary label $y \in \{-1, 1\}$. Figure 7 illustrates how the squared error loss of a hypothesis can be (highly) misleading for binary classification.

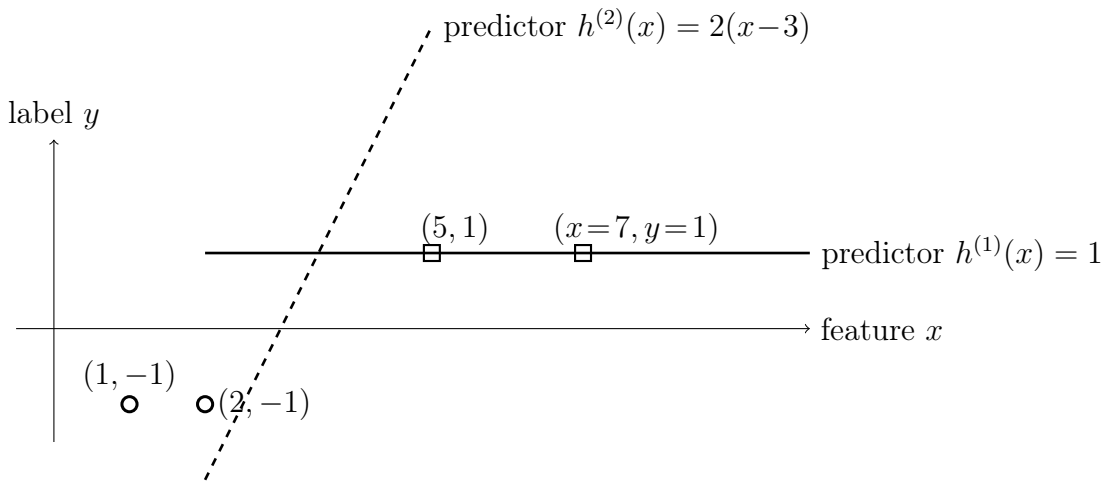

Figure 7: A training set consisting of four data points with binary labels $\hat{y}^{(r)} \in \{-1, 1\}$. Minimizing the squared error loss (2) would prefer the (poor) classifier $h^{(1)}$ over the (reasonable) classifier $h^{(2)}$.

Figure 7 depicts a dataset $\mathcal{X}$ consisting of $m = 4$ data points with binary labels $y^{(r)} \in \{-1, 1\}$, for $r = 1, \ldots, m$. The figure also depicts two candidate hypotheses $h^{(1)}(x)$ and $h^{(2)}(x)$ that can be used for classifying data points. The classifications $\hat{y}$ obtained with the hypothesis $h^{(2)}(x)$ would perfectly match the labels of the four training data points 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 wrong for data points with $y = -1$.

Looking at $\mathcal{X}$, we might prefer using $h^{(2)}(x)$ over $h^{(1)}$ to classify data points. However, the squared error loss 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 is very confident ($|h(\mathbf{x})|$ is large) in a wrong classification ($\hat{y} \neq y$). Moreover, a useful loss function loss function should not punish (deliver small values for) a hypothesis is very confident ($|h(\mathbf{x})|$ is large) in a correct classification ($\hat{y} = y$). However, by its very definition, the squared 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 noted otherwise, the formulas for these loss functions are valid only if the label values are the real numbers -1 and 1 (the label space is $\mathcal{Y} = \{-1, 1\}$). 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 a correct classifications, $\hat{y} = y$ for any data point. This suggests to learn a hypothesis $h(\mathbf{x})$ by minimizing the 0/1 loss

$$L\big((\mathbf{x}, y), h\big) := \begin{cases} 1 & \text{if } y \neq \hat{y} \\ 0 & \text{else,} \end{cases} \quad \text{with } \hat{y} = \begin{cases} 1 & \text{if } h(\mathbf{x}) \geq 0 \\ 0 & \text{else.} \end{cases} \tag{3}$$

Figure 8 illustrates the 0/1 loss (3) for a data point with features $\mathbf{x}$ and label $y{=}1$ as a function of the hypothesis 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 yields the value one.

The 0/1 loss (3) is conceptually appealing when data points are interpreted as realizations of i.i.d. random variable (RV)s with the same probability distribution $p(\mathbf{x}, y)$. 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

$$(1/m) \sum_{r=1}^{m} L\big((\mathbf{x}^{(r)}, y^{(r)}), h\big) \approx p(y \neq h(\mathbf{x})) \tag{4}$$

for sufficiently large sample size m. A precise formulation of the approximation (4) can be obtained from the law of large numbers [27, 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 (4), the 0/1 loss (3) 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 property 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 (3) 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 [23, Sec. 3.8.]).

To avoid the non-convexity of the 0/1 loss (3) we might approximate it by a convex "surrogate" loss function. One such approximation of the 0/1 loss is the hinge loss

$$L\big((\mathbf{x}, y), h\big) := \max\{0, 1 - y h(\mathbf{x})\}. \tag{5}$$

Figure 8 depicts the hinge loss (5) as a function of the hypothesis $h(\mathbf{x})$. The hinge loss (5) 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 $h(\mathbf{x})$. A non-differentiable loss function cannot be minimized by simple gradient-based methods but requires more advanced optimization methods.

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

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

The logistic loss (6) is used in logistic regression [23, Ch. 3] to measure the usefulness of a linear hypothesis map $h(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$. Figure 8 depicts the logistic loss (6) as a function of the hypothesis $h(\mathbf{x})$. The logistic loss (6) is a convex and differentiable function of $h(\mathbf{x})$. For a correct classification ($\hat{y}=y$), the logistic loss (6) 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 (5) and the logistic loss (6) are convex functions of the weights $\mathbf{w} \in \mathbb{R}^d$ in a linear hypothesis map $h(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$. However, in contrast to the hinge loss, the logistic loss (6) is also a differentiable function of the $\mathbf{w}$.

The convex and differentiable logistic loss function can be minimized using simple gradient-based methods such as gradient descent (GD) (see Section 5). In contrast, we cannot use basic 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 GD which is known as subgradient

descent [28]. Subgradient descent is obtained from GD by generalizing the concept of a gradient to that of a subgradient.

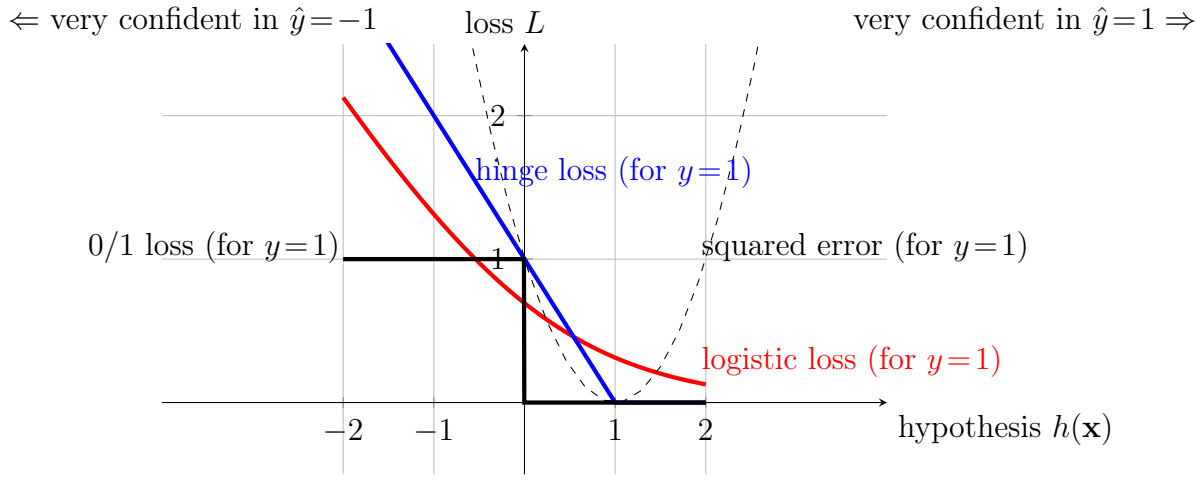

Figure 8: The solid curves depict three loss functions for a binary classification problem with label space $\mathcal{Y} = \{-1, 1\}$. A data point with features $\mathbf{x}$ and label $y = 1$ is classified as $\hat{y} = 1$ if $h(\mathbf{x}) \geq 0$ and classified as $\hat{y} = -1$ if $h(\mathbf{x}) < 0$. 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 $h(\mathbf{x})$.

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

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

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

distribution $p(\mathbf{x}, y)$. We can then measure the quality of a hypothesis h by the expected loss or risk [29]

$$\mathbb{E}\big\{L\big((\mathbf{x}, y), h\big)\big\} := \int_{\mathbf{x},y} L\big((\mathbf{x}, y), h\big) dp(\mathbf{x}, y). \tag{7}$$

The risk (7) 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 (7) 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,

$$h^* \in \operatorname*{argmin}_{h \in \mathcal{H}} \mathbb{E}\big\{L\big((\mathbf{x}, y), h\big)\big\}. \tag{8}$$

Any hypothesis that solves (8), i.e., that incurs minimal risk, is referred to as a Bayes estimator [29, 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 (7). Efficient methods to solve the optimization problem (7) include variational methods and random sampling (Monte Carlo) methods [29, 30].

If we do not know the probability distribution underlying the data points, we cannot compute the risk and, in tun, cannot use (8) 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 $\mathcal{X}$ as

$$\widehat{L}\big(h|\mathcal{X}\big) = (1/m) \sum_{r=1}^{m} L\big((\mathbf{x}^{(r)}, y^{(r)}), h\big). \tag{9}$$

The empirical risk of the hypothesis $h \in \mathcal{H}$ is the average loss on the data points in $\mathcal{X}$. 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 $\mathcal{X}$.

If the data points used to compute the empirical risk (9) 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

$$\mathbb{E}\big\{L\big((\mathbf{x},y),h\big)\big\} \approx \underbrace{(1/m)\sum_{r=1}^{m} L\big((\mathbf{x}^{(r)},y^{(r)}),h\big)}_{\stackrel{(9)}{=}\widehat{L}\big(h|\mathcal{X}\big)} \text{ for sufficiently large } m. \quad (10)$$

The approximation error in (10) can be quantified precisely by different variants of the law of large numbers [27, 31, 32] or large-deviation bounds [33].

3.2 Empirical Risk Minimization

Many ML methods are motivated by (10) which suggests that a hypothesis with small empirical risk (9) 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

$$\widehat{h} := \operatorname*{argmin}_{h \in \mathcal{H}} \underbrace{(1/m) \sum_{r=1}^{m} L\big(\big(\mathbf{x}^{(r)}, y^{(r)}\big), h\big)}_{\widehat{L}\big(h|\mathcal{X}\big)} . \tag{11}$$

The objective function of ERM is the empirical risk (9) which, in turn, approximates the risk $\mathbb{E}\big\{L\big((\mathbf{x}, y), h\big)\big\}$ via the law of large numbers (10). Note that ERM (11) is parametrized by three components: the dataset $\mathcal{X}$, the hypothesis space $\mathcal{H}$ and loss function $L\big(\cdot, \cdot\big)$. The Python library `scikit-learn` provides implementations of (11) for different choices for the hypothesis space $\mathcal{H}$ and loss function $L\big(\cdot, \cdot\big)$ [22].

The objective function of ERM depends on the dataset

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

The statistical properties of $\mathcal{X}$ affect the computational and statistical aspects of ERM. By computational aspects of ERM, we mainly refer to the computational complexity of finding (approximate) solutions to (11). The statistical aspects of (11) include basic properties (such as its probability distribution) of the solutions to (11).

In general, we do not have (much) control over the statistical properties of the observed data. 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 and statistical properties of ERM.

For linear regression, which uses linear model $h^{(\mathbf{w})}(\mathbf{x}) = \mathbf{w}^T \mathbf{x}$ and squared

error loss (2), generic ERM (11) specializes to

$$\widehat{\mathbf{w}} := \operatorname*{argmin}_{\mathbf{w}\in\mathbb{R}^d} \underbrace{(1/m)\sum_{r=1}^{m}\big(\mathbf{w}^T\mathbf{x}^{(r)} - y^{(r)}\big)^2}_{\widehat{L}\big(h^{(\mathbf{w})}|\mathcal{X}\big)}. \tag{12}$$

3.3 How Much Training is Needed?

Beside the computational aspects of solving (approximately) (11), we must also ensure that the solutions of (11) provide a hypothesis with small risk. As illustrated in Figure 9, the empirical risk is typically different from the risk of a hypothesis. We can expect ERM to deliver a useful hypothesis only if this deviation is not too large.

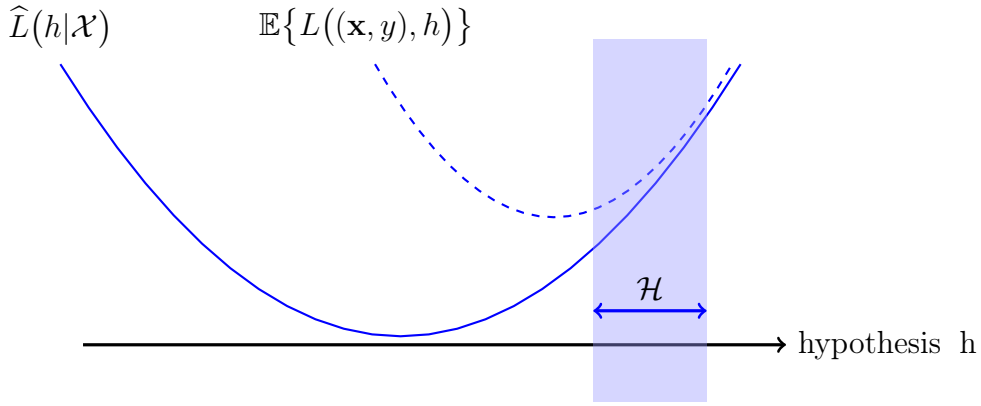

Figure 9: ERM (11) learns a hypothesis, out of a hypothesis space $\mathcal{H}$, by minimizing the empirical risk $\widehat{L}\big(h|\mathcal{X}\big)$ over a training set $\mathcal{X}$. However, the ultimate goal is to learn a hypothesis which predicts well the label of any data point and, in turn, has a small risk $\mathbb{E}\big\{L\big((\mathbf{x},y),h\big)\big\}$. Thus, ERM can be expected to deliver a useful hypothesis if the deviation between the empirical risk and the risk is small (uniformly over $h \in \mathcal{H}$).

3.4 Exercises

Exercise 3.1. Trivial Objective in ERM Consider some arbitrary dataset which is used to learn a hypothesis out of $\mathcal{H}$ via ERM (11). For which combinations of loss function and hypothesis space does ERM not depend at all on the dataset.

Exercise 3.2. Data Leakage via ERM Consider a ML method that uses ERM (11) to learn a hypothesis to predict a binary label $y \in \{-1, 1\}$ of a data point form its numeric features $\mathbf{x} \in \mathbb{R}^d$. Can you think of a hypothesis space $\mathcal{H}$ that would allow to perfectly recover the data points (their features and label!) in $\mathcal{X}$ solely from inspecting the empirical risk $\widehat{L}\big(h|\mathcal{X}\big)$ (viewed as a function of $h \in \mathcal{H}$).

4 Regularization

i.i.d. assumption. Structure of ERM for different choices of model and loss (smooth convex; non-smooth; non-convex); computational (how fast can we solve ERM) and statistical (are solutions any good?) aspects or ERM; regularization as implicit data augmentation; regularization as modified loss functions; regularization as model pruning;

The idea of ERM is to approximate the expected loss or risk of a hypothesis by the empirical risk $\widehat{L}\big(h|\mathcal{X}\big)$ (11) over a training set $\mathcal{X}$. Whenever this approximation fails to be accurate, the solution of ERM might be a hypothesis that performs poorly outside the training set used in (11). 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 (11) might deliver poor predictions outside the training set $\mathcal{X}$ used in (11). Indeed, if the size of the training set is small compared to the size (dimension) of the hypothesis space, solving (11) will likely result in overfitting. The ML method will then learn a hypothesis that perfectly fits the training set $\mathcal{X}$ but does a poor job in predicted the labels of data points outside the training set [23, Ch. 6].

Regularization techniques modify ERM (11) 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 $\mathcal{X}$ in (11) by adding new data points obtained by perturbing features or labels.
- model pruning: we modify (11) by reducing the optimization domain, which is the hypothesis space $\mathcal{H}$, to a (tiny) subset $\mathcal{H}' \subseteq \mathcal{H}$
- loss penalization: modify the loss function in (11) by adding a scaled regularization term,

$$\widehat{h} := \operatorname*{argmin}_{h \in \mathcal{H}} \underbrace{(1/m) \sum_{r=1}^{m} L\big(\big(\mathbf{x}^{(r)}, y^{(r)}\big), h\big)}_{\widehat{L}\big(h|\mathcal{X}\big)} + \lambda \mathcal{R}\big\{h\big\} \tag{13}$$

It turns out that these three approaches to regularization are essentially equivalent. Figure 11 illustrates the equivalence between data augmentation and loss penalization. Indeed, the empirical risk incurred over an enlarged dataset obtained by data augmentation coincides with the empirical risk on the original dataset using a penalized loss function. This penalty term corresponds to the average loss on the augmented data points.

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 $\mathcal{X}$ 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 $\mathcal{X}'$ which contains each data point of $\mathcal{X}$ along with its B perturbations. It can then be shown that, for $B \to \infty$, the average squared error loss $\widehat{L}\big(h^{(\mathbf{w})}|\mathcal{X}'\big)$ on the augmented dataset $\mathcal{X}'$ tends towards $\widehat{L}\big(h^{(\mathbf{w})}|\mathcal{X}\big) + \sigma^2 \left\|\mathbf{w}\right\|_2^2$. Note that $\widehat{L}\big(h^{(\mathbf{w})}|\mathcal{X}\big) + \sigma^2 \left\|\mathbf{w}\right\|_2^2$ is an instance of regularized empirical risk minimization (RERM) (13) using the regularizer $\mathcal{R}\big\{h^{(\mathbf{w})}\big\} = \left\|\mathbf{w}\right\|_2^2$.

4.3 FL via Regularization

We will see how FL methods are obtained by combining different instances of RERM (13). In particular, we will study FL methods whose main computational building block is the RERM instance,

$$\widehat{\mathbf{w}} := \operatorname*{argmin}_{\mathbf{w}\in\mathbb{R}^d} \underbrace{\widehat{L}\big(h^{(\mathbf{w})}|\mathcal{X}\big)}_{\text{training error}} + (\lambda/2) \underbrace{\left\|\mathbf{w}' - \mathbf{w}\right\|^2}_{\text{deviation from } \mathbf{w}'} . \tag{14}$$

The parameter vector $\widehat{\mathbf{w}}$ obtained from (14) balances between two (in general conflicting) goals. On one hand, we want to minimize the empirical risk

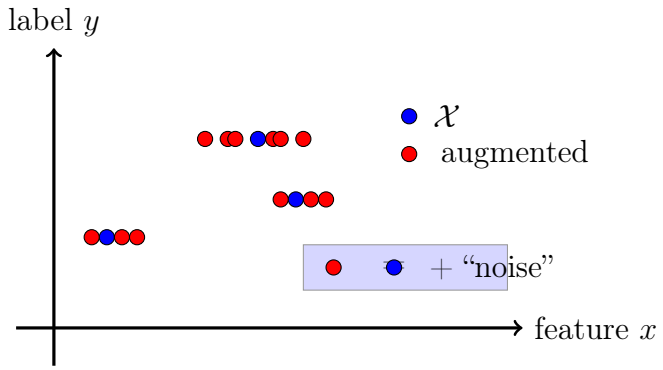

Figure 10: Data augmentation adds new data points to a given dataset $\mathcal{X}$. These new data points are obtained by perturbing features and labels of the data points in $\mathcal{X}$.

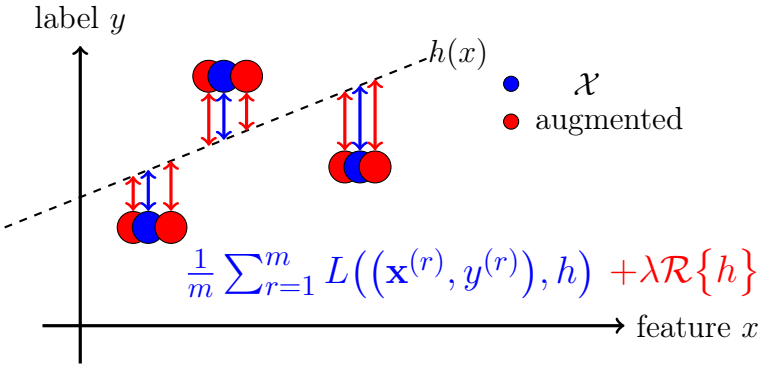

Figure 11: Equivalence of data augmentation and loss penalization techniques for regularization.

$\widehat{L}\big(h^{(\mathbf{w})}|\mathcal{X}\big)$. 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 (14) will be our main building block for the design of FL algorithms (see Section 9). In these algorithms, the vector $\mathbf{w}'$ in (14) 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 the associated learning tasks.

4.4 Proximal Operator and Iterations

The RERM instance (14) is closely related to the concept of a proximal operator [34, 35]. Given a function $f(\mathbf{w})$ with domain $\mathbb{R}^d$, the associated proximal operator is defined as

$$\mathbf{prox}_{f,\rho}(\mathbf{w}') := \operatorname*{argmin}_{\mathbf{w}\in\mathbb{R}^d} f(\mathbf{w}) + (\rho/2)\,\|\mathbf{w}-\mathbf{w}'\|_2^2 \text{ with } \rho>0. \tag{15}$$

Note that the proximal operator is parametrized by the quantity $\rho > 0$. Solving RERM (14) is to evaluate the proximal operator $\mathbf{prox}_{f,\lambda}(\mathbf{w}')$ of the empirical risk $f(\mathbf{w}) = \widehat{L}\big(h^{(\mathbf{w})}|\mathcal{X}\big)$ (viewed as a function of the model parameters $\mathbf{w}$).

*****

Discuss contraction properties of proximal operators of strongly convex functions.

****

****

Introduce (deterministic) proximal iterations to solve ERM.

****

Since we will use the proximal operator (52) as the main building block for FL methods, it is interesting to study its computational aspects. To this end, we consider an online learning setting where the training set $\mathcal{X}$ is build up sequentially over time. [36] problem of learning the weights of a linear hypothesis map from a training set $\mathcal{X}$ that is build up sequentially. We use

4.5 Exercises

Exercise 4.1. Regularization via Data Augmentation Consider the RERM (14) using some labeled dataset

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

linear hypothesis space and squared error loss (2). Show that (14) is equivalent to plain ERM (11) for linear regression applied to another dataset $\mathcal{X}'$ which is obtained by adding data points, with carefully chosen features and labels, to the original $\mathcal{X}$.

5 Gradient Based 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 the family of gradient-based methods for solving ERM arising in many widely-used ML methods [23, 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 minimizing a local linear approximation of the empirical risk.

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 GD steps due to different types of errors in the gradients. 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.

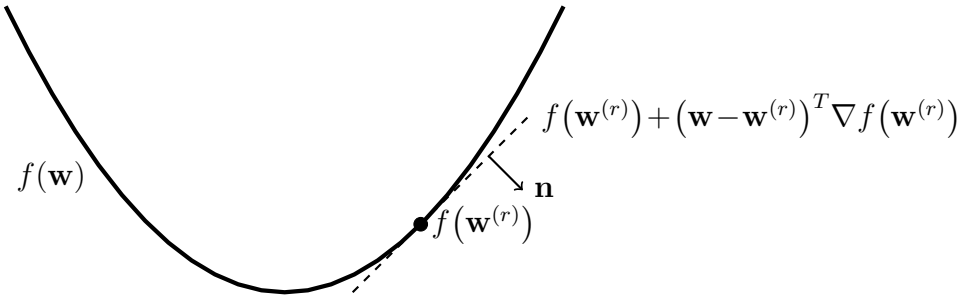

Figure 12: We can approximate a differentiable function $f(\mathbf{w})$ locally around a point $\mathbf{w}^{(r)}$ using a hyperplane. The normal vector $\mathbf{n} = (\nabla f\big(\mathbf{w}^{(r)}\big), -1)$ of this approximating hyperplane is determined by the gradient $\nabla f\big(\mathbf{w}^{(r)}\big)$ [37].

5.1 Gradient Descent

Let us rewrite ERM (12) as the optimization problem

$$\min_{\mathbf{w}\in\mathbb{R}^d} f(\mathbf{w}) := (1/m)\sum_{r=1}^{m} L\big((\mathbf{x}^{(r)}, y^{(r)}), h^{(\mathbf{w})}\big). \tag{16}$$

From now on we tacitly assume that each individual loss

$$f_r(\mathbf{w}) := L\big((\mathbf{x}^{(r)}, y^{(r)}), h^{(\mathbf{w})}\big) \tag{17}$$

arising in (16) represents a differentiable function of the parameter vector $\mathbf{w}$. Trivially, differentiability of the components (17) implies differentiability of the overall objective function $f(\mathbf{w})$ (16).

Two important examples of ERM involving such differentiable loss functions are linear regression and logistic regression. In contrast, the hinge loss (5) used by the support vector machine (SVM) results in a non-differentiable objective function $f(\mathbf{w})$ (16). However, it is possible to (significantly) extend the scope of gradient-based methods to non-differentiable functions by replacing the concept of a gradient with that of a subgradient.

5.2 Projected Gradient Descent

demonstrate how FedAvg is an instance of Projected GD

5.3 Perturbed Gradient Descent

5.4 Stochastic Gradient Descent

Part II

Federated Learning

6 Networked Data and Models

Many important application domains such as healthcare generate collections of local datasets. Indeed, each hospital maintains its own local database about their patients. These local datasets often share some statistical similarities, e.g., the frequency of flu patients at nearby hospitals might have a similar temporal pattern. Section 6.1 introduces the concept of an empirical graph to represent a collection of local datasets along with their similarities.

The network structure encoded by the edges of the empirical graph might arise not only from statistical similarities but also from the computational infrastructure. Consider the extreme case of local datasets having identical statistical properties (they are all sampled from the same probability distribution). Here, it would be natural to use a fully connected graph as the empirical graph. However, from a computational perspective it might be useful to use a sparser graph with a smaller number of edges.

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

By definition of the empirical graph, two nodes connected by a an edge with large weight carry local datasets with similar probability distribution. It seems therefore natural to pool local datasets from well-connected nodes to obtain a larger dataset to train a ML 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 model parameters at clusters or well-connected nodes.

Most of our course assumes that the empirical graph of decentralized data is given. The empirical graph might be obtained by domain-expertise which 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. Section 6.4 glances over different approaches for learning a graph from data.

6.1 The Empirical Graph

We find it useful to represent networked data by an undirected weighted *empirical graph* $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. The nodes $\mathcal{V} = \{1, \ldots, n\}$ represent collections of local datasets. In particular, each node $i \in \mathcal{V}$ of the empirical graph $\mathcal{G}$ carries a separate local dataset $\mathcal{X}^{(i)}$. To build intuition, think of a local dataset $\mathcal{X}^{(i)}$ as a labeled dataset

$$\mathcal{X}^{(i)} := \big\{ \big(\mathbf{x}^{(1)}, y^{(1)}\big), \ldots, \big(\mathbf{x}^{(m_i)}, y^{(m_i)}\big) \big\} . \tag{18}$$

Here, $\mathbf{x}^{(r)}$ and $y^{(r)}$ denote, respectively, the feature vector and true label of the r-th data point in the local dataset $\mathcal{X}^{(i)}$. Note that the size m_i of the local dataset might vary across 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 $\mathcal{X}^{(i)}$ and $\mathcal{X}^{(i')}$ have similar statistical properties. The strength

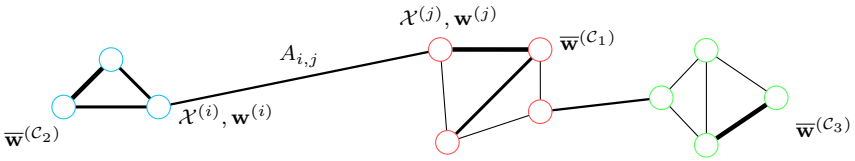

Figure 13: We represent networked data and corresponding models using an undirected empirical graph $\mathcal{G} = \big(\mathcal{V}, \mathcal{E}\big)$. Each node $i \in \mathcal{V}$ of the graph carries a local dataset $\mathcal{X}^{(i)}$ and model weights $\mathbf{w}^{(i)}$ which are scored using a local loss function $L_i\left(\mathbf{w}^{(i)}\right)$ (that encapsulates the local dataset $\mathcal{X}^{(i)}$). Two different nodes $i, i' \in \mathcal{V}$ are connected by a weighted edge $\{i, i'\}$ if they carry datasets with similar statistical properties. The amount of similarity is encoded in an edge weight $A_{i,i'} > 0$ (indicated by varying thickness). We consider FL methods that rely on a clustering assumption that the optimal local parameters for nodes in the same cluster $\mathcal{C} \subseteq \mathcal{V}$ are in the proximity of a cluster-wide optimal weight vector $\mathbf{w}^{(\mathcal{C})}$. In this example, the empirical graph is partitioned into three disjoint clusters $\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3$. Our FL method do not require the (typically unknown) partition but rather learns the partition based on the local datasets and network structure of $\mathcal{G}$.

of this similarity is quantified by the 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 $\mathcal{X}^{(i)}$ at node i is (statistically) similar to the local dataset $\mathcal{X}^{(i')}$ at node i' then also the local dataset $\mathcal{X}^{(i)}$ is (statistically) similar to the local dataset $\mathcal{X}^{(i)}$.

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 entire set of directed edges for an empirical graph is obtained as

$$\big\{ (i, i') : i, i' \in \mathcal{V}, i < i' \text{ and } \{i, i'\} \in \mathcal{E} \big\}. \tag{19}$$

We abuse notation and use $\mathcal{E}$ to denote both, the set of undirected edges and i its oriented version (19).

6.2 Networked Models

Consider data with empirical graph $\mathcal{G}$ whose nodes represent local datasets. We use the local dataset $\mathcal{X}^{(i)}$ to learn a hypothesis h from a local model or hypothesis space $\mathcal{H}^{(i)}$. A networked model $\mathcal{H}^{(\mathcal{G})}$ for empirical graph $\mathcal{G}$ is the collection of local models $\mathcal{H}^{(i)}$ for each node $i \in \mathcal{V}$. The local models might be different, e.g., $\mathcal{H}^{(i)}$ might be a linear model for some node i 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 $L_i(h)$. If node $i \in \mathcal{V}$ carries a local dataset of the form

(18), we might define a local loss function via the average loss

$$L_i\left(h\right) := \left(1/m_i\right) \sum_{r=1}^{m_i} L\big(\big(\mathbf{x}^{(i,r)}, y^{(i,r)}\big), h\big). \tag{20}$$

Our focus will be on parametrized networked models where local models are 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 [38–42]. Parametrized networked models consist of maps $\mathbf{w} : \mathcal{V} \mapsto \mathbb{R}^d$ that assigns each node $i \in \mathcal{V}$ the local model parameters $\mathbf{w}^{(i)} \in \mathbb{R}^d$. The collection of all such maps constitutes the "node space"

$$\mathcal{W} := \left\{\mathbf{w} : \mathcal{V} \to \mathbb{R}^d : i \mapsto \mathbf{w}^{(i)}\right\}.$$

We will abuse notation and refer by $\mathbf{w}^{(i)}$ also to the entire collection map $\mathbf{w}$.

We measure the usefulness of a particular choice for the local model parameters $\mathbf{w}^{(i)}$ by some local loss function $L_i\left(\mathbf{w}^{(i)}\right)$. Unless otherwise state, we consider local loss functions that are convex and differentiable. The NFL method proposed in Section **??** allows for different choices for the local loss functions. These different choices might be obtained, in turn, from different combinations of ML models and performance metrics [23, Ch. 3].

From a computational perspective, our main requirement on the choice for the local loss function $L_i\left(\mathbf{w}^{(i)}\right)$ is that it allows for efficient solution of the regularized problem,

$$\min_{\mathbf{w}' \in \mathbb{R}^d} L_i\left(\mathbf{w}'\right) + \lambda \|\mathbf{w}' - \mathbf{w}''\|_2^2. \tag{21}$$

The computational complexity of our approach depends on the ability to efficiently solve (21) for any given $\lambda \in \mathbb{R}_+$ and $\mathbf{w}'' \in \mathbb{R}^d$. Note that solving (21) is equivalent to evaluating the proximity operator $\mathbf{prox}_{L_i(\cdot),2\lambda}(\mathbf{w}'')$.

The NFL method in Section **??** applies to parametric models that can be trained by minimizing a loss function $L_i\left(\cdot\right)$ whose proximity operator can be evaluated efficiently. Convex functions for which the proximity operator can be computed efficiently are sometimes referred to as "proximable" or "simple" [43]. Note that the shape of the loss function typically depends on both, the choice for the local model and the metric used to measure prediction errors [23, Ch. 4].

6.3 Clustering Assumption

Many applications generate local datasets which do not carry sufficient statistical power to guide learning of model parameters $\mathbf{w}^{(i)}$. As a case in point, consider a local dataset $\mathcal{X}^{(i)}$ of the form (18), 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 [23, 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 weight vector with minimum average loss in $\mathcal{X}^{(i)}$ might incur large prediction errors on data points outside $\mathcal{X}^{(i)}$ [23, 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) [26]. 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.

Chapter 7 proposes different measures for the variation of local hypotheses $h^{(i)}$ 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. We now make the notion of datasets in the same cluster"having similar statistical properties" precise. In particular, we require the local loss functions at nodes $i \in \mathcal{C}$ in the same cluster $\mathcal{C}$ to have nearby minimizers. For differentiable and convex loss functions this is equivalent to require local loss functions have a small gradient at the cluster-wise minimizer (24).

Assumption 1 (Clustering)**.** *Consider some networked data represented by an empirical graph $\mathcal{G}$ whose nodes carry local loss functions $L_i(\mathbf{v})$, for $i \in \mathcal{V}$. There is a partition*

$$\mathcal{P} = \{\mathcal{C}_1, \ldots, \mathcal{C}_{|\mathcal{P}|}\} \text{ with } \mathcal{C}_c \cap \mathcal{C}_{c'} = \emptyset \text{ and } \mathcal{V} = \mathcal{C}_1 \cup \ldots \cup \mathcal{C}_{|\mathcal{P}|}, \tag{22}$$

of the nodes $\mathcal{V}$ into disjoint clusters such that for each cluster $\mathcal{C}_c \in \mathcal{P}$,

$$\|\nabla L_i\left(\overline{\mathbf{w}}^{(\mathcal{C}_c)}\right)\| \leq \varepsilon^{(i)} \text{ for all } i \in \mathcal{C}_c. \tag{23}$$

Here, the vector $\overline{\mathbf{w}}^{(\mathcal{C}_c)}$ *denotes the solution of the cluster-wide minimization* (24) *for the specific cluster* $\mathcal{C}_c$.

The clustering assumption requires the norm $\|\nabla L_i\big(\ \overline{\mathbf{w}}^{(\mathcal{C})}\big)\|$ to be bounded by a constant $\varepsilon^{(i)}$ for each nodes $i \in \mathcal{V}$ in the empirical graph. We can interpret this norm as a measure for the discrepancy between the cluster-wide minimizer $\overline{\mathbf{w}}^{(\mathcal{C})}$ (see (24)) and the minimizers of the local loss functions $L_i\big(\ \overline{\mathbf{w}}^{(\mathcal{C})}\big)$ for each node $i \in \mathcal{C}$.

The main theme of this paper is to use the empirical graph $\mathcal{G}$ to regularize the learning of local model parameters by requiring them to not vary too much over edges with large weights. Section 7 introduces the concept of generalized total variation (GTV) as a quantitative measure for the variation of local parameter vectors. Regularization by requiring a small GTV is an instance of the smoothness assumption used in semi-supervised learning [44].

Our analysis of GTVMin in Section **??** relates its underlying smoothness assumption to a clustering assumption (see Assumption 1) that requires local model parameters to be constant over subsets (clusters) of nodes in the empirical graph. Theorem **??** offers a precise characterization of the cluster structure delivered by GTVMin. Note that the clustering of local model parameters (being constant over well-connected nodes) is essentially an adaptive pooling of local datasets. The pooling of local datasets is driven jointly by the geometry (connectivity) of the empirical graph $\mathcal{G}$ and the geometry (shape) of local loss functions.

Consider networked data with empirical graph $\mathcal{G} = \big(\mathcal{V}, \mathcal{E}\big)$. Each node i in the graph carries a local dataset $\mathcal{X}^{(i)}$ and a local model with parameters $\mathbf{w}^{(i)}$. Our goal is to learn the local model parameters $\mathbf{w}^{(i)}$ for each node $i \in \mathcal{V}$. The

key assumption of clustered FL is that the local datasets form clusters with local datasets in the same cluster having similar statistical properties [21, 45]. Given a cluster $\mathcal{C}$ of nodes it seems natural to pool their local datasets or, equivalently, add their local functions to learn a cluster-specific parameter vector

$$\overline{\mathbf{w}}^{(\mathcal{C})} = \operatorname*{argmin}_{\mathbf{v} \in \mathbb{R}^d} f^{(\mathcal{C})}(\mathbf{v}) \text{ with } f^{(\mathcal{C})}(\mathbf{v}) := \sum_{i \in \mathcal{C}} L_i(\mathbf{v}). \tag{24}$$

Note that (24) cannot be implemented in practice since we typically do not know the cluster $\mathcal{C}$. The main analytical contribution of this paper is an upper bound for the deviation between solutions of GTVMin and the cluster-wise (but impractical) learning problem (24). This bound characterizes the statistical properties of FL algorithms that are obtained by applying optimization techniques for solving GTVMin (see Section **??**).

The solution $\overline{\mathbf{w}}^{(\mathcal{C})}$ of (24) minimizes the aggregation (sum) of all local loss functions that belong to the same cluster $\mathcal{C} \subseteq \mathcal{V}$. Thus, $\overline{\mathbf{w}}^{(\mathcal{C})}$ is the optimal model parameter for a training set obtained by pooling all local datasets that belong ot the cluster $\mathcal{C}$. As indicated by the the notation, we tacitly assume that the solution to (24) is unique. The uniqueness of the solution in (24) will be ensured by Assumption 2 below.

We use the cluster-wide minimization (24) only as a tool to analyze the NFL methods in Section **??**. The cluster-wide minimization (24) does not provide a practical approach as it requires knowledge of the clusters in the partition (22). It might be unrealistic to assume perfect knowledge of the partition (22) postulated by Assumption 1.

Section 7 formulates NFL as GTV minimization which is an instance of RERM. GTV minimization is enforce "clusteredness" of local model parameters

by requiring a small variation across edges in the empirical graph. Under Assumption 1, this regularization strategy will be useful if many (large weight) edges connect nodes in the same cluster but only few (small weight) edges connect nodes in different clusters. Section **??** presents a precise condition on the network structure such that GTV minimization succeeds in capturing the true underlying cluster structure of the local loss functions.

The analysis of the NFL method proposed in Section **??** requires the local loss functions to be convex and smooth. Moreover, we require their (partial) sums in the cluster-wide objective $f^{(\mathcal{C})}(\cdot)$ (24) to be strongly convex [46, Exercise 1.9].

Assumption 2 (Convexity and Smoothness.)**.** *For each node $i \in \mathcal{V}$, the local loss function $L_i\left(\mathbf{w}^{(i)}\right)$ is convex and differentiable with gradient $\nabla L_i\left(\mathbf{w}^{(i)}\right)$ satisfying*

$$\|\nabla L_i(\mathbf{w}) - \nabla L_i(\mathbf{v})\| \leq \beta^{(i)} \|\mathbf{w} - \mathbf{v}\|. \tag{25}$$

For each cluster in the partition (22)*, the objective function $f^{(c)}(\cdot)$ (see* (24)*) is strongly convex,*

$$f^{(c)}(\mathbf{v}') \geq f^{(c)}(\mathbf{v}) + \left(\mathbf{v}'-\mathbf{v}\right)^{T} \partial f^{(c)}(\mathbf{v}') + (\alpha^{(c)}/2)\|\mathbf{v}'-\mathbf{v}\|^2 \text{ for any } \mathbf{v}', \mathbf{v} \in \mathbb{R}^d. \tag{26}$$

Here, $\alpha^{(c)} > 0$ is a positive constants that might be different for different clusters $\mathcal{C}_c$. The norm $\|\cdot\|$ in (26) *will be specified later.*

Assumption 2 is rather standard in FL literature [20, 47, 48]. In particular, Assumption 2 is satisfied by many important ML models [38–41]. Assumption 2 does not hold for many deep learning models that result in non-convex

loss functions [9]. Nevertheless, we expect our theoretical analysis to provide useful insight also for settings where Assumption 2 is violated.

Assumption 2 does not require strong convexity for each local loss function. Rather, we only require their cluster-wide sums to be strongly convex. We also allow for trivial local loss functions that are constant and might arise from lack of local data (due to privacy constraints) or computational resources (stragglers).

The NFL method in Section **??** can tolerate the presence of non-informative local loss functions by exploiting the similarities between local datasets as reflected by the edges in the empirical graph $\mathcal{G}$. Appendix **??** presents examples for local loss functions, obtained from some widely-used ML models, that conform to Assumption 2.

6.4 Graph Learning Methods

simple tests for statistical similarity (t-test, Kolmogorov Smirnov); link prediction methods from deep learning over graphs; post-process first estimate of network by fitting a stochastic model (such as SBM)

Section 7 introduced GTVMin as a versatile design principle that includes some main flavours of FL as special cases (see Section 8). We then obtained practical FL algorithms in Section 9 by applying optimization methods to solve GTVMin. It is important to remember that the "GTVMin design paradigm" requires a useful choice for the empirical graph of networked data. The connectivity of nodes (that represent local datasets) in the empirical graph must reflect the clustering of local datasets sharing similar statistical properties. Some application domains offer a natural choice for the empirical graph, e.g.,

induced by spatio-temporal proximity, functional relations, physical models, or the underlying distributed ("edge") computing infrastructure [6, 67–70].

If a natural choice for the empirical graph is not available, we could try to learn the network structure by methods that measure statistical similarity between two datasets [71–75]. We demonstrate some of these methods in the numerical experiments of Section **??**. However, the detailed study of methods for learning networks from raw data is beyond the scope of this paper (see Section **??** for future research directions).

6.4.1 Testing Similarity

basic statistical tests; deep learning methods for data sets [76]

6.4.2 Total Variation Minimization

[73, 77]

6.4.3 Probabilistic Graphical Models

6.4.4 Stochastic Block Model

6.5 Exercises

Exercise 6.1. Get Weather Data. I added my code to flcourse/FMIData repository. The code now creates nodes for each observation and connects these nodes to five nearest nodes using edges. Nodes have names such as 'Helsinki Malmi lentokenttä' and 'Oulu Oulunsalo Pellonpää' and using these names it is possible to access weather data. This code also adds in weights to edges that connect two nodes together. The weights are right now

distances between these two nodes. Parsing of data from FMIopendata is explained in fmiopendata repository quite well (at the bottom of the page). `https://github.com/pnuu/fmiopendata`

7 A Design Principle for FL

Section 6.2 introduced the notion of an empirical graph to represent collections of local datasets and corresponding local models. This section develops a flexible design principle for FL algorithms that (jointly) train local models by combining the information contained in the local datasets as well as their network structure (encoded in the weighted edges of the empirical graph.

The training of local models will be coupled by requiring them to behave similar for similar local datasets. We make this requirement precise using the notation of GTV which will be defined in Section 7.1. Section 7.2 then introduces GTVMin whose solutions are local model parameters that optimally balance between the empirical risk incurred in local datasets and their GTV. Section 7.3 discusses several useful interpretations of GTVMin. The extension of GTV and GTVMin to non-parametric local models is then discussed in Section 7.4.

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

$$\mathbf{D} : \mathcal{W} \to \mathcal{U} : \mathbf{w} \mapsto \mathbf{u} \text{ with } \mathbf{u}^{(e)} = \mathbf{w}^{(e_+)} - \mathbf{w}^{(e_-)}. \tag{27}$$

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

$$\mathbf{D}^T : \mathcal{U} \to \mathcal{W} : \mathbf{u} \mapsto \mathbf{w} \text{ with } \mathbf{w}^{(i)} = \sum_{e \in \mathcal{E}} \sum_{i=e_+} \mathbf{u}^{(e)} - \sum_{i=e_-} \mathbf{u}^{(e)}. \tag{28}$$

Using the block-incidence matrix (27) we can write more compactly $\mathbf{u} = \mathbf{D}\mathbf{w}$.

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

$$\|\mathbf{w}\|_{\mathrm{GTV}} := \sum_{(i,i') \in \mathcal{E}} A_{i,i'} \phi\big(\mathbf{w}^{(i')} - \mathbf{w}^{(i)}\big) \text{ with penalty function } \phi(\cdot) : \mathbb{R}^d \to \mathbb{R}. \tag{29}$$

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

$$\|\mathbf{w}\|_{\mathcal{S}} := \sum_{(i,i') \in \mathcal{S}} A_{i,i'} \phi\big(\mathbf{w}^{(i')} - \mathbf{w}^{(i)}\big). \tag{30}$$

Strictly speaking, the GTV (29) 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 GTV minimization problem (see Section **??** and Section **??**). Two popular choices are $\phi(\mathbf{v}) := \|\mathbf{v}\|_2$, which is used by network Lasso [49], and $\phi(\mathbf{v}) := (1/2)\|\mathbf{v}\|_2^2$ which is used by "MOCHA" [13]. Another recent FL method for networked data uses the choice $\phi(\mathbf{v}) := \|\mathbf{v}\|_1$ [50].

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 $\phi(\mathbf{v}) := (1/2)\|\mathbf{v}\|_2^2$ [51, 52].

Section **??** designs NFL methods whose main computational steps includes computing the proximity operator $\mathbf{prox}_{\phi^*,\rho}(\cdot)$ of the convex conjugate ϕ^* of the penalty function $\phi(\cdot)$. Thus, for these methods to be computationally tractable we must choose $\phi(\cdot)$ such that this proximity operator can be computed (evaluated) efficiently.[1] Appendix **??** discusses some important choices for $\phi(\cdot)$ for which this is the case.

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 $\mathcal{X}^{(i)}$ 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,

[1] The difficulty of computing the proximity operator $\mathbf{prox}_{\phi^*,\rho}(\mathbf{u})$ is essentially the same as that of computing the proximity operator $\mathbf{prox}_{\phi,\rho}(\mathbf{u})$ as these two are related via $\mathbf{u} = \mathbf{prox}_{\phi,1}(\mathbf{u}) + \mathbf{prox}_{\phi^*,1}(\mathbf{u})$ [35].

we learn the local parameter vectors $\mathbf{w}^{(i)}$ by balancing between (the sum of) local loss functions and GTV (29),

$$\widehat{\mathbf{w}} \in \arg\min_{\mathbf{w}\in\mathcal{W}} \sum_{i\in\mathcal{V}} L_i\left(\mathbf{w}^{(i)}\right) + \lambda\|\mathbf{w}\|_{\mathrm{GTV}}. \tag{31}$$

Note that GTV minimization (31) is an instance of RERM that is obtained from using the GTV 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 $\sum_{i\in\mathcal{V}} L_i\left(\mathbf{w}^{(i)}\right)$. We will solve (31) using an established primal-dual method for non-smooth convex problems [53]. Section **??** offers precise conditions on the local loss functions and empirical graph such that the solutions of (31) capture the (unknown) underlying partition (22).

The regularization parameter $\lambda > 0$ in (31) 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 [26] or by our analysis of the solutions of (31) in Section **??**.

Loosely speaking, increasing the value of λ results in the solutions of (31) becoming increasingly clustered, i.e., the local parameter vectors $\widehat{\mathbf{w}}^{(i)}$ become constant over increasingly larger subsets of nodes. Choosing λ larger than some critical value, that depends on the shape of the local loss functions and the network structure of $\mathcal{G}$, results in $\widehat{\mathbf{w}}^{(i)}$ being constant over all nodes $i \in \mathcal{V}$.

GTV minimization (31) unifies and considerably extends some well-known methods for distributed optimization and learning. In particular, the network Lasso [49] is obtained from (31) for the choice $\phi(\mathbf{v}) := \|\mathbf{v}\|_2$. The MOCHA method [13] is obtained from (31) for the choice $\phi(\mathbf{v}) := \|\mathbf{v}\|_2^2$. Another special case of (31), obtained for the choice $\phi(\mathbf{v}) := \|\mathbf{v}\|_1$, has been studied recently [50].

7.3 Interpretations

We next discuss some useful relations between GTVMin (31) and basic ML techniques.

7.3.1 Multi-task Learning

Consider networked data with empirical graph $\mathcal{G}$ whose nodes $i \in \mathcal{V}$ carry local datasets $\mathcal{X}^{(i)}$. Each node also carries a local model $\mathcal{H}^{(i)}$ which is trained from $\mathcal{X}^{(i)}$. 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 [45, 54]

7.3.2 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 [38,41,51]. Thus, we can interpret GTVMin as a generalization of graph clustering methods such as spectral clustering or flow-based clustering [55, 56].

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

Convex clustering [57, 58] is obtained from GTVMin (31) for the local loss functions

$$L_i\left(\mathbf{w}^{(i)}\right) = \left\|\mathbf{w}^{(i)} - \mathbf{a}^{(i)}\right\|_2^2, \text{ for all nodes } i \in \mathcal{V} \tag{32}$$

and GTV penalty $\phi(\mathbf{u}) = \|\mathbf{u}\|_p$ being a p-norm $\|\mathbf{u}\|_p := \big(\sum_{j=1}^{d} |u_j|^p\big)^{1/p}$ with some $p \geq 1$. The vectors $\mathbf{a}^{(i)}$ in (32) are the observations that we wish to cluster.

7.3.3 Locally Weighted Learning

Another interpretation of GTVMin is that of locally weighted learning [59]. 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 (24), $\widehat{\mathbf{w}}^{(i)} \approx \overline{\mathbf{w}}^{(\mathcal{C})}$. The deviation between $\widehat{\mathbf{w}}^{(i)}$ and $\overline{\mathbf{w}}^{(\mathcal{C})}$ will be bounded by Theorem **??**. Note that the cluster-wide optimization (24) can be written as a locally weighted learning problem [59, Sec. 3.1.2]

$$\overline{\mathbf{w}}^{(\mathcal{C})} = \operatorname*{argmin}_{\mathbf{w} \in \mathbb{R}^d} \sum_{i' \in \mathcal{V}} \rho_{i'} L_{i'}\left(\mathbf{w}\right). \tag{33}$$

The locally-weighted learning problem (33) 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 of the local hypotheses $h^{(i)}$ and $h^{(i')}$ via the discrepancy between their predictions obtained for the same test-set

$$\mathcal{X}' = \left\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m')}\right\}. \tag{34}$$

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

$$d_h^{(i,i')} := (1/m') \sum_{r=1}^{m'} \left[L\big(\big(\mathbf{x}^{(r)}, h^{(i)}\big(\mathbf{x}^{(r)}\big)\big), h^{(i')}\big) + L\big(\big(\mathbf{x}^{(r)}, h^{(i')}\big(\mathbf{x}^{(r)}\big)\big), h^{(i)}\big) \right]. \tag{35}$$

We define a variant of GTV (29) for non-parametric models by summing the discrepancy (35) over all edges $\mathcal{E}$,

$$\mathrm{GTV}\,\{h\} := \sum_{\{i,i'\} \in \mathcal{E}} A_{i,i'} d_{\cdot}^{(i,i')} \tag{36}$$

Note that $\mathrm{GTV}\,\{h\}$ is parametrized by the choice for the loss function L used to compute the discrepancy $d_h^{(i,i')}$ (35).

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

$$\widehat{h} \in \operatorname*{argmin}_{h \in \mathcal{H}^{(\mathcal{G})}} \sum_{i \in \mathcal{V}} L_i\left(h^{(i)}\right) + \lambda \mathrm{GTV}\,\{h\}. \tag{37}$$

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 (37) is parametrized by the choice for the loss function used to compute the discrepancy (35). Different choices for the loss function in (35) result in different computational and statistical properties of (37). If we use the squared error loss to measure the discrepancy (35), (37) specializes to

$$\widehat{h} \in \operatorname*{argmin}_{h \in \mathcal{H}^{(\mathcal{G})}} \sum_{i \in \mathcal{V}} L_i \left(h^{(i)} \right) + (\lambda / m') \sum_{(i,i') \in \mathcal{E}} A_{i,i'} \sum_{r=1}^{m'} \left(h^{(i)}\big(\mathbf{x}^{(r)}\big) - h^{(i')}\big(\mathbf{x}^{(r)}\big) \right)^2 . \tag{38}$$

7.5 Exercises

Exercise 7.1. Generalizing TV to Decision Trees Think about ways to modify GTVMin (31) 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 [60]. 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 [9]. These global model parameters are broadcasted

to the clients which compute parameter updates based on their local datasets (e.g., using GD 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).

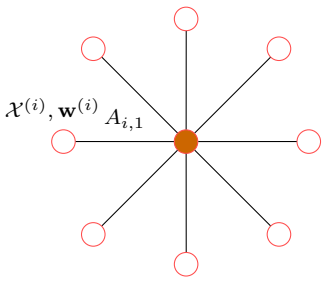

Figure 14: Star graph $\mathcal{G}^{(\text{star})}$ with centre node representing a server and peripheral nodes representing clients that generate local datasets.

8.2 Decentralized Federated Learning

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

8.3 Clustered Federated Learning

[62] empirical graph is clustered; GTVMin parameter chosen suitably to enforce common local model parameters over clusters

Clustered FL addresses the heterogeneity of local datasets using various forms of a clustering assumption [20, 21, 44, 45]. Informally, our clustering assumption requires local datasets, and their associated learning tasks, to form

few disjoint subsets or clusters. Local datasets belonging to the same cluster have similar statistical properties and, in turn, similar optimal parameter values for the corresponding local models. We make this clustering assumption precise in Section **??** (see Assumption 1). A main contribution of this paper is a precise characterization of the cluster structure and local model geometry for local datasets that allow our methods to pool local datasets that form statistically homogeneous clusters of datasets (see Section **??**).

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 (low-level layers in deep nets). This partial parameter sharing can be implemented by specific choice for the GTV penalty function. In particular, we could use a combination of two terms, each term measuring the variation of different 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 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 [63]

8.5 Vertical Federated Learning

[64]

8.6 Horizontal Federated Learning

[65]

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 (31) with penalty $\phi(\mathbf{u}) = \|\mathbf{u}\|_2^2$,

$$\widehat{\mathbf{w}} \in \arg\min_{\mathbf{w}\in\mathcal{W}} \sum_{i\in\mathcal{V}} L_i\left(\mathbf{w}^{(i)}\right) + \lambda \sum_{\{i,i'\}\in\mathcal{E}} A_{i,i'} \left\|\mathbf{w}^{(i)} - \mathbf{w}^{(i')}\right\|_2^2. \tag{39}$$

We obtain FL methods by applying iterative optimization methods to solve GTVMin (39). 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 (39).

9.1 FedRelax

We now apply block-coordinate minimization [18, 66] to solve GTVMin (39). To this end, we rewrite(39) as

$$\widehat{\mathbf{w}} \in \arg\min_{\mathbf{w}\in\mathcal{W}} \underbrace{\sum_{i\in\mathcal{V}} f^{(i)}(\mathbf{w})}_{:=f^{(\mathrm{GTV})}(\mathbf{w})}$$
$$\text{with } f^{(i)}(\mathbf{w}) := L_i\left(\mathbf{w}^{(i)}\right) + (\lambda/2) \sum_{i'\in\mathcal{N}^{(i)}} A_{i,i'} \left\|\mathbf{w}^{(i)} - \mathbf{w}^{(i')}\right\|_2^2. \tag{40}$$

Note that the objective function in (**??**) is a sum of 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 (40). 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 $\widehat{\mathbf{w}}_k$,

$$\begin{aligned}\widehat{\mathbf{w}}_{k+1}^{(i)} &\in \operatorname*{argmin}_{\mathbf{w}^{(i)} \in \mathbb{R}^d} f^{(\mathrm{GTV})}\left(\widehat{\mathbf{w}}_k^{(1)}, \ldots, \widehat{\mathbf{w}}_k^{(i-1)}, \mathbf{w}^{(i)}, \widehat{\mathbf{w}}_k^{(i+1)}, \ldots\right) \\ &\overset{(40)}{=} \operatorname*{argmin}_{\mathbf{w}^{(i)} \in \mathbb{R}^d} f^{(i)}\left(\widehat{\mathbf{w}}_k^{(1)}, \ldots, \widehat{\mathbf{w}}_k^{(i-1)}, \mathbf{w}^{(i)}, \widehat{\mathbf{w}}_k^{(i+1)}, \ldots\right) \\ &\overset{(40)}{=} \operatorname*{argmin}_{\mathbf{w}^{(i)} \in \mathbb{R}^d} L_i\left(\mathbf{w}^{(i)}\right) + (\lambda/2) \sum_{i' \in \mathcal{N}^{(i)}} A_{i,i'} \left\|\mathbf{w}^{(i)} - \widehat{\mathbf{w}}_k^{(i')}\right\|_2^2. \end{aligned} \tag{41}$$

Algorithm 1 simply repeats the update (41) 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 (37) instead of (39). We obtain the update

$$\begin{aligned}\widehat{h}_{k+1}^{(i)} \in \operatorname*{argmin}_{h \in \mathcal{H}^{(\mathcal{G})}} & L_i\left(h^{(i)}\right) + (\lambda/2) \sum_{i' \in \mathcal{N}^{(i)}} A_{i,i'} \underbrace{d_h^{(i,i')}}_{\text{see (35)}} \\ & \text{such that } h^{(i')} = \widehat{h}_k^{(i')} \text{ for } i' \in \mathcal{V} \setminus \{i\}. \end{aligned} \tag{43}$$

Replacing the update (41) with (43) in step 4 of Algorithm 1 makes it applicable to non-parametric local models. The update (43) is obtained by using the discrepancy (35) between local models within the GTV measure (36).

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 (41))

$$\widehat{\mathbf{w}}_{k+1}^{(i)} := \operatorname*{argmin}_{\mathbf{w}^{(i)} \in \mathbb{R}^d} L_i\left(\mathbf{w}^{(i)}\right) + (\lambda/2) \sum_{i' \in \mathcal{N}^{(i)}} A_{i,i'} \left\| \mathbf{w}^{(i)} - \widehat{\mathbf{w}}_k^{(i')} \right\|_2^2 \tag{42}$$

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 (35) between local hypotheses $h^{(i)}$ and $h^{(i')}$ (see (38)).

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 $\mathcal{X}' = \{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m')}\}$; 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 $\left\{\widehat{h}_k^{(i)}(\mathbf{x})\right\}_{\mathbf{x}\in\mathcal{X}^{(\text{test})}}$, with neighbours $i' \in \mathcal{N}^{(i)}$

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

$$\widehat{h}_{k+1}^{(i)} \in \operatorname*{argmin}_{h^{(i)}\in\mathcal{H}^{(i)}} \Bigg[L_i\left(h^{(i)}\right) + (\lambda/2) \sum_{i'\in\mathcal{N}^{(i)}} (A_{i,i'}/m') \sum_{r=1}^{m'} \left(h^{(i)}(\mathbf{x}^{(r)}) - \widehat{h}_k^{(i')}(\mathbf{x}^{(r)})\right)^2 \Bigg]. \quad (44)$$

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 (39) for $\lambda \to \infty$. Indeed, for sufficiently large λ, the penalty term in (39) dominates, enforcing the learnt local parameter vectors to be nearly identical. Thus, for

sufficiently large λ, we can approximate (39) by

$$\widehat{\mathbf{w}} \in \arg\min_{\mathbf{w} \in \mathcal{C}} \sum_{i \in \mathcal{V}} L_i\left(\mathbf{w}^{(i)}\right) \\ \text{with } \mathcal{C} = \left\{\mathbf{w} : \mathbf{w}^{(i)} = \mathbf{w}^{(i')} \text{ for any edge } \{i, i'\} \in \mathcal{E}\right\}. \tag{45}$$

We can solve (45) 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

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

The resulting Algorithm 4 performs regularized tailored linear regression for each local dataset. The regularization is implemented by the GTV (29) 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 (14)

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

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

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

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 (14))

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

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 (41) 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 [78]:

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 [79]. 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 $\mathbf{w}^{(i)}$.

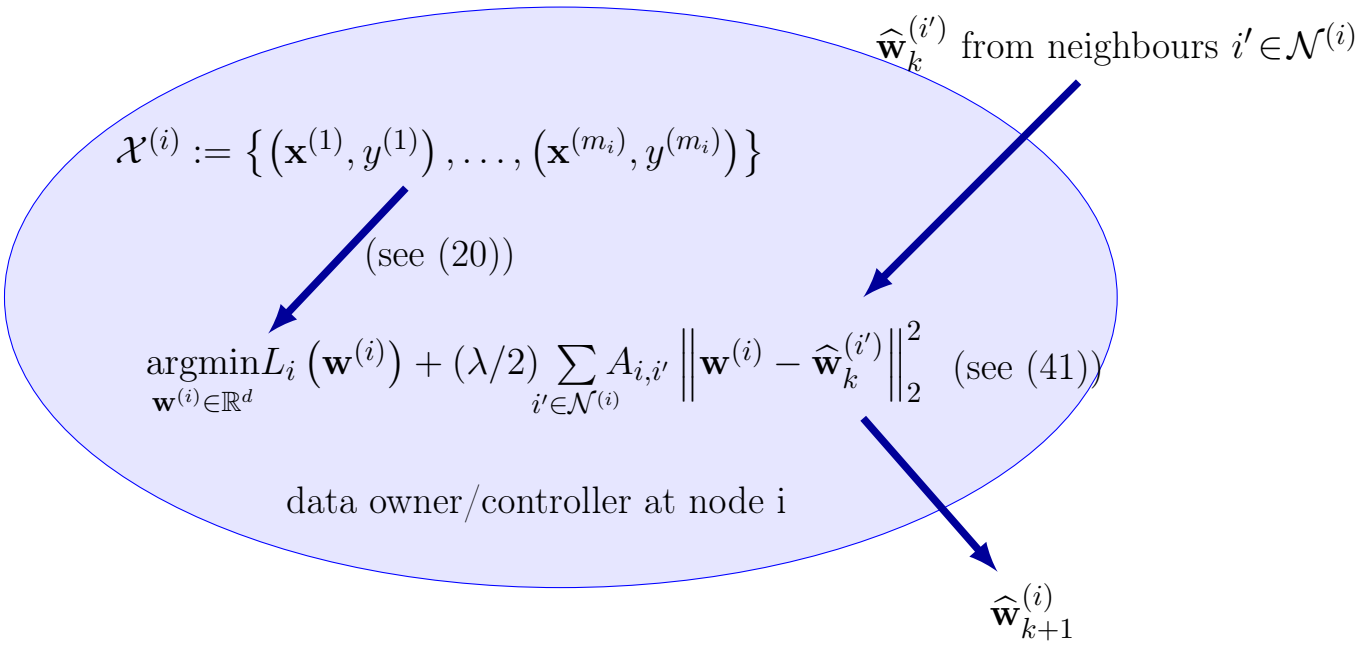

Figure 15: Data flow at node $i \in \mathcal{V}$ during a single iteration of Algorithm 1.

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 (41) 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 $\mathcal{X}^{(i)}$.

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 $\mathcal{X}^{(i)}$ 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 $\mathcal{X}^{(i)}$, 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 [80, 81]

$$\widetilde{\mathbf{w}}_k^{(i)} := \widehat{\mathbf{w}}_k^{(i)} + \sigma \boldsymbol{\varepsilon}^{(i,k)}. \tag{46}$$

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 (46) controls the amount of privacy protection offered by (46). 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

$$\widetilde{\mathbf{w}}_{k+1}^{(i)} := \underset{\mathbf{w}^{(i)} \in \mathbb{R}^d}{\operatorname{argmin}} \left[L_i\left(\mathbf{w}^{(i)}\right) + (\lambda/2) \sum_{i' \in \mathcal{N}^{(i)}} A_{i,i'} \left\| \mathbf{w}^{(i)} - \widetilde{\mathbf{w}}_k^{(i')} \right\|_2^2 \right] + \sigma \boldsymbol{\varepsilon}^{(i,k)}. \quad (47)$$

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 $\mathcal{X}^{(i))}$. Instead of perturbing the result of the updates 4 in Algorithm 1, we directly perturb the features of the data points in $\mathcal{X}^{(i)}$. 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 $\mathcal{X}^{(i)}$, we apply a feature map [23, Sec. 9.5]

$$\mathbf{\Phi} : \mathbf{x} \mapsto \mathbf{z} = \mathbf{\Phi}(\mathbf{x}). \tag{48}$$

A privacy-preserving feature map (48) delivers transformed feature vectors $\mathbf{z}$ that do not allow to estimate (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

$$\tilde{L}_i\left(\mathbf{w}^{(i)}\right) := (1/m_i) \sum_{r=1}^{m_i} L\big(\big(\mathbf{\Phi}\big(\mathbf{x}^{(r)}\big), y^{(r)}\big), h^{(\mathbf{w}^{(i)})}\big). \tag{49}$$

We obtain Algorithm 6 from Algorithm 1 by replacing the local loss function $L_i(\cdot)$ with the perturbed version $\tilde{L}_i(\cdot)$ (49).

Examples for privacy-preserving feature maps (48) that could be used for Algorithm 6 include

- "information obfuscation" by adding noise $\mathbf{\Phi}(\mathbf{x}) = \mathbf{x} + \text{noise}$ (see [81, Fig. 3])

Algorithm 6 FedRelax with Perturbed Features

Input: empirical graph $\mathcal{G}$; feature map $\mathbf{\Phi}$, local dataset $\mathcal{X}^{(i)}$ 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

$$\widehat{\mathbf{w}}_{k+1}^{(i)} := \operatorname*{argmin}_{\mathbf{w}^{(i)} \in \mathbb{R}^d} \Bigg[(1/m_i) \sum_{r=1}^{m_i} L\Big(\big(\mathbf{\Phi}\big(\mathbf{x}^{(r)}\big), y^{(r)}\big), h^{(\mathbf{w}^{(i)})}\Big) + (\lambda/2) \sum_{i' \in \mathcal{N}^{(i)}} A_{i,i'} \left\| \mathbf{w}^{(i)} - \widehat{\mathbf{w}}_k^{(i')} \right\|_2^2 \Bigg] \tag{50}$$

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 $\mathbf{\Phi}(\mathbf{x}) = \mathbf{W}\mathbf{x}$ [23, 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.

- 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 $\mathcal{X}^{(i)}$. The resulting perturbed feature vectors $\mathbf{z}^{(r)} = \mathbf{\Phi}\big(\mathbf{x}^{(r)}\big)$ are said to provide k-anonymity [82, 83]. Figure 17 illustrates a feature map that is piece-wise constant along the values of a single feature that uniquely identifies a data point.

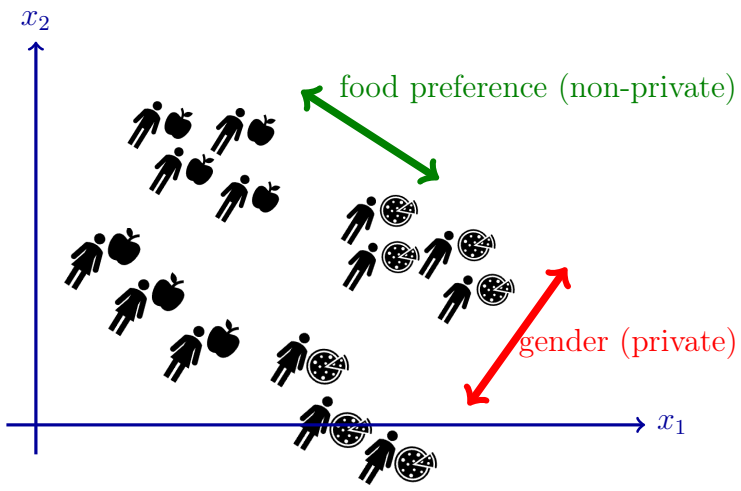

Figure 16: A toy dataset $\mathcal{X}^{(i)}$ whose data points represent persons that are characterized by two features x_1, x_2. These features carry sensitive information (gender) and non-sensitive information (food preference) about a person. The non-sensitive information allows to predicting the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

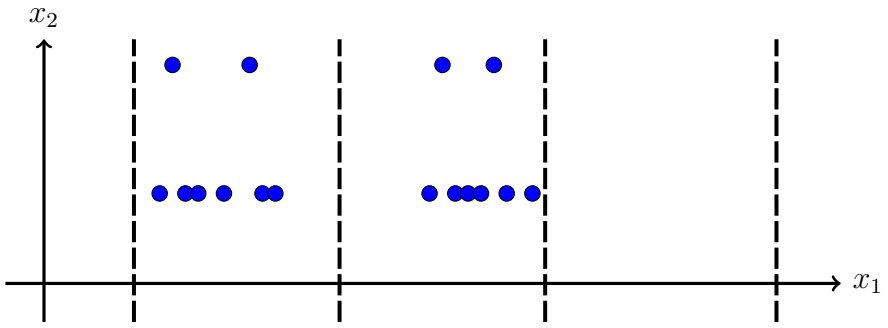

Figure 17: A toy dataset $\mathcal{X}^{(i)}$ whose data points are characterized by two features: x_1 and x_2. The feature x_2 is private and carries sensitive information. The other feature x_1 is non-private but uniquely identifies a data point: two different data points in $\mathcal{X}^{(i)}$ have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

12 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) for them. 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 [84, 85]. 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 [86]. 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 [87].

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 $\mathcal{X}^{(i)}$. 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

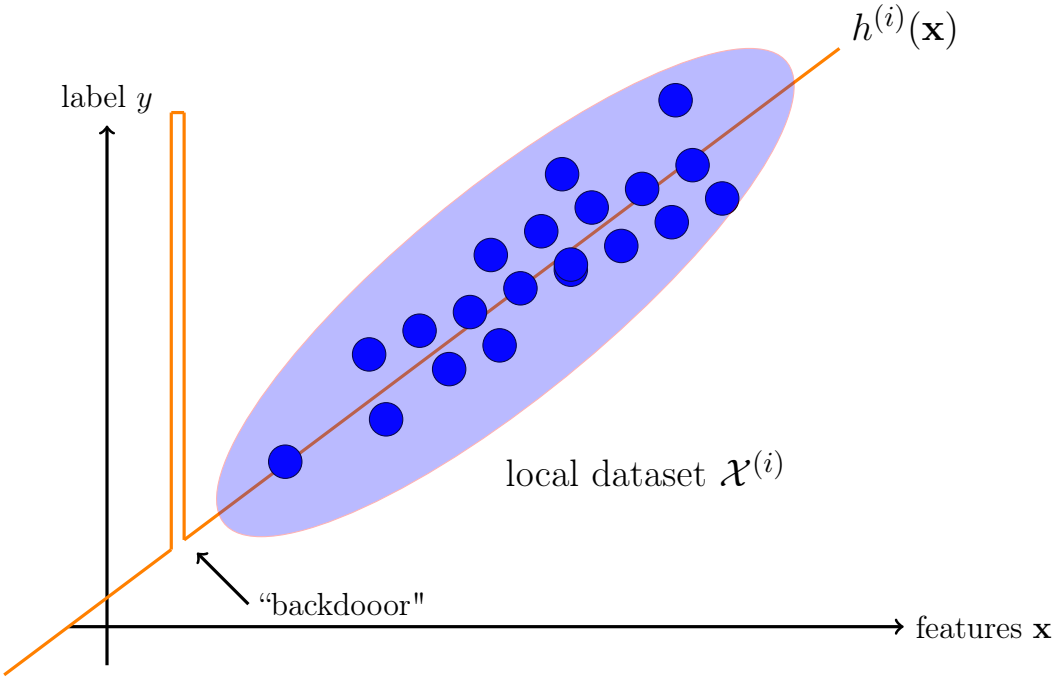

Figure 18: A backdoor attack aims at nudging the local hypothesis $h^{(i)}$ to behave in a specific way for certain feature values. These feature values can be interpreted as the key that unlocks a "backdoor".

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 [88]. 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 [24]

$$f^*(\mathbf{x}) := \sup_{\mathbf{z}\in\mathbb{R}^d} \mathbf{x}^T\mathbf{z} - f(\mathbf{z}). \tag{51}$$

We also need the proximity operator associated with a closed proper convex function convex function $f(\mathbf{x})$, defined as [34]

$$\mathbf{prox}_{f,\rho}(\mathbf{x}) := \operatorname*{argmin}_{\mathbf{x}'} f(\mathbf{x}') + (\rho/2)\|\mathbf{x}-\mathbf{x}'\|_2^2 \text{ for some } \rho > 0. \tag{52}$$

Note that the minimum exists and is unique since the objective function is strongly convex [24] .

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 [89, 90]. The mutual information between two RVs $\mathbf{x}, y$ with joint probability distribution $p(\mathbf{x}, y)$ is defined as

$$I(\mathbf{x}, y) := \mathbb{E}\left\{\log p(\mathbf{x}, y)/(p(\mathbf{x})p(y))\right\} \tag{53}$$

15 Fixed Point Theory

Glossary

k-means The k-means algorithm is a hard clustering method. It aims at assigning data points to clusters such that they have minimum average distance from the cluster centre. 104, 115

accuracy Consider data points characterized by features $\mathbf{x}$ and a categorical label y which takes on values from a finite label space. The accuracy of a hypothesis h, when applied to the data points in a finite dataset $\mathcal{X} = \big\{\big(\mathbf{x}^{(1)}, y^{(1)}\big), \ldots, \big(\mathbf{x}^{(m)}, y^{(m)}\big)\big\}$ is the defined as $1 - (1/m)\sum_{r=1}^{m} L\big(\big(\mathbf{x}^{(r)}, y^{(r)}\big), h\big)$ with the 0/1 loss (3) 29

artificial intelligence Artificial intelligence aims at developing systems that behave rational in the sense of maximizing a long-term reward. 10, 82

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. 16, 18, 38, 44, 48, 74, 102, 106, 112, 116

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). 93

Bayes estimator A hypothesis whose Bayes risk is minimal [29]. 33, 34, 99

Bayes risk We use the term Bayes risk as a synonym for the risk or expected loss of a hypothesis. Some authors reserve the term Bayes risk for the

risk of a hypothesis that achieves minimum risk, such a hypothesis being referred to as a Bayes estimator [29]. 98

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. 14, 22, 26, 30, 31

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. 26

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. 54, 61

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 69

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 [24]. 30

data A set of data points. 13, 38, 113

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. 39–41

data point A data point is any object that conveys information [89]. 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. 6, 13–17, 19–21, 23–29, 31–34, 36–41, 43, 44, 54, 69, 84–86, 88, 90–93, 97–104, 106–112, 114–118

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 [84, 85]. 93, 98, 102

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 [91]. By using indexed lists of data points we avoid some of the challenges arising in concept of an abstract set. 6, 10, 32, 33, 35, 37, 38, 40, 41, 48, 49, 51, 61, 69, 88, 89, 91–93, 107

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 $h(\mathbf{x})$. 51, 66, 68

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 93

differentiable A function $f : \mathbb{R}^d \to \mathbb{R}$ is differentiable if it has a gradient $\nabla f(\mathbf{x})$ everywhere (for every $\mathbf{x} \in \mathbb{R}^d$) [37]. 46, 116

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. 39, 106

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. 10, 48–51, 54, 61, 65, 69, 70, 93, 99

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. 33, 34, 36–38, 40, 42, 44, 45, 61, 111, 112, 117

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. 10, 32, 35–39, 43–46, 88, 103, 111, 112, 114, 117, 118

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 $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^d$ [37]. 103, 114

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. 67

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. 16, 17, 88–90

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. 14, 107

features Features are those properties of a data point 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" [92–94]. However, this book makes consequent use of the term features for low-level properties of data points that can be measured easily. 6, 13–15, 17, 18, 25, 34, 37, 39, 41, 43, 44, 69, 85, 91, 93, 100, 103, 109, 115

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 [9] consider federated averaging methods where the local model training is implemented by running several GD steps 76, 78, 79

federated learning (FL) Federated learning is an umbrella term for ML methods that train models in a collaborative fashion using decentralized data and computation. 7–11, 14, 15, 38, 40, 42, 43, 50, 51, 58, 61, 69, 70, 73, 82, 84, 93, 96, 99, 101, 104

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 [56]. 65

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 or collect data related to people in the EU. 9

generalized total variation Generalized total variation measures the changes of vector-valued node attributes of a graph. 55, 61, 65–67, 71, 74–77, 87, 89, 105

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 $\nabla f(\mathbf{w})\big|_{\mathbf{w}'}$. 7, 10, 30, 31, 45, 46, 102, 115, 116

gradient descent (GD) Gradient descent is an iterative method for finding the minimum of a differentiable function $f(\mathbf{w})$. 30, 31, 44, 45, 70, 104, 113, 116, 117

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. 21, 22, 30, 44, 46

GTV minimization GTV minimization is an instance of RERM using the GTV of local paramter vectors as regularization term. 10, 55, 56, 58,

61, 65–71, 73, 74, 93

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 [33, 95]. 53

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 (5). A regularized variant of the hinge loss is used by the SVM to learn a linear classifier with maximum margin between the two classes (see Figure **??**). 29, 30, 117

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. 13, 15, 20–23, 26, 29, 30, 32–34, 36–39, 44, 53, 54, 67, 84, 86, 93, 98, 99, 109, 111, 112, 115

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 (**??**). 13, 15, 16, 18–20, 32, 35–39, 44, 48, 51, 66, 67, 102, 111, 117, 118

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. 8, 29, 32, 34, 38, 40, 86, 107, 108, 110, 111, 115

i.i.d. assumption The i.i.d. assumption interprets data points of a dataset as the realizations of i.i.d. RVs. 32, 111, 115

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 [96,97]. 16

label A higher level fact or quantity of interest associated with a data point. If a data point is an image, its label might be the fact that it shows a cat (or not). Some widely used synonyms for the term label are "response variable", "output variable" or "target" [92–94]. 13–15, 19, 25, 34, 36, 37, 39, 41, 43, 85, 88, 100, 109, 115

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"$\}$ 14, 26, 28, 31, 98, 107

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. 29, 34, 35

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 $\mathbf{w}$. 53

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 $\mathbb{R}^d$. 106

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). 35, 40, 43, 44, 46, 74, 77, 79, 108

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 (6). 22, 30, 31, 111

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). 30, 46

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. 20, 21, 23, 24, 29–34, 39–41, 46, 54, 75, 87, 98, 112, 114, 115

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 $h \in \mathcal{H}$. 10, 13, 19–25, 28–31, 34, 35, 37–40, 44, 46, 50–52, 65, 67–69, 76, 88, 109–111, 115

maximum likelihood Consider data points $\mathcal{X} = \big\{\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(m)}\big\}$ 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) $p(\mathcal{X}; \mathbf{w}) = \prod_{r=1}^{m} p(\mathbf{z}^{(r)}; \mathbf{w})$ of observing the data is maximized. Thus, the maximum likelihood estimator is obtained as a solution to the optimization problem $\max_{\mathbf{w} \in \mathcal{W}} p(\mathcal{X}; \mathbf{w})$. 21, 24

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 (3)) 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 (6). 22, 23

model We use the term model as a synonym for hypothesis space 15, 38, 39, 48, 49, 106, 107, 112

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. 66

non-smooth We refer to a function as non-smooth if it is not smooth [98]. 21

objective function An objective function is a map that assigns each possible value of an optimization variable, such as the parameters $\mathbf{w}$ of a hypothesis $h^{(\mathbf{w})}$, to an objective value $f(\mathbf{w})$. The objective value $f(\mathbf{w})$ could by the the risk or the empirical risk of a hypothesis $h^{(\mathbf{w})}$. 46

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. 21, 24

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. 38, 39

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. 44, 50, 75, 85, 87, 89

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 [29]. 32, 86

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 $p(x \in \mathcal{B}) = \int_{\mathcal{B}} p(x')dx'$ [32, 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 $p(\mathbf{x} \in \mathcal{R}) = \int_{\mathcal{R}} p(\mathbf{x}')dx'_1 \ldots dx'_d$ [32, Ch. 3]. 107

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 [27,31]. 21, 29, 32–35, 48, 86, 97, 107, 108, 110–112, 115

projected GD Projected GD is obtained from basic GD by projecting the result of a gradient step into a given constrain set. 77

random variable (RV) A random variable is a mapping for function from a set of elementary events to a set of values. The set of elementary events is equipped with a probability measure that assigns subsets of elemtary events a probability. 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 [27, 31]. 29, 32, 34, 86, 97, 100, 107, 108, 110–113, 115

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. 38–41, 49, 53, 54, 67, 69, 71, 105

regularized empirical risk minimization Regularized empirical 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. 40, 42, 43, 78, 79, 105

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 $\lambda > 0$.

53

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. 32, 33, 35, 36, 38, 45, 98, 99, 111

scatterplot A visualization technique that depicts data points by markers in a two-dimensional plane. 103

smooth We refer to a real-valued function as smooth if it is differentiable and its gradient is continuous [28, 98]. 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., $\|\nabla f(\mathbf{w}) - \nabla f(\mathbf{w}')\| \leq \beta \|\mathbf{w} - \mathbf{w}'\|$. 111

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 [55, 56]. 65

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. 24–26, 43, 68, 76

stochastic gradient descent Stochastic GD is obtained from GD by replacing the gradient of the objective function by a noisy (or stochastic) estimate. 45

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. 74, 75, 89

strongly convex A continuously differentiable real-valued function $f(\mathbf{x})$ is strongly convex with coefficient σ if $f(\mathbf{y}) \geq f(\mathbf{x}) + \nabla f(\mathbf{x})^T(\mathbf{y}-\mathbf{x}) + (\sigma/2)\left\|\mathbf{y}-\mathbf{x}\right\|_2^2$ [98], [66, Sec. B.1.1.]. 45

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. 108

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 $\mathbf{w}'$. 31, 46

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 $\widehat{L}\big(h^{(\mathbf{w})}\big|\mathcal{X}\big)$ as a function of the parameters $\mathbf{w}$ of a hypothesis $h^{(\mathbf{w})}$. 30, 31

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 (5). 46, 106

training error Consider a ML method that aims at learning a hypothesis $h \in \mathcal{H}$ out of a hypothesis space. We refer to the average loss or empirical risk of a hypothesis $h \in \mathcal{H}$ on a dataset as training error if it is used to choose between different hypotheses. The principle of ERM is find the hypothesis $h^* \in \mathcal{H}$ with smallest training error. Overloading the notation a bit, we might refer by training error also to the minimum empirical risk achieved by the optimal hypothesis $h^* \in \mathcal{H}$. 38, 39

training set A set of data points that is used in ERM to train a hypothesis $\widehat{h}$. The average loss of $\widehat{h}$ on the training set is referred to as the training error. The comparison between training and validation error informs adaptations of the ML method (such as using a different hypothesis space). 24, 27, 32, 36, 38–40, 43, 48, 54, 102, 106, 109, 112, 114, 116, 118

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. 39

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). 118

References

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

[2] 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.

[3] 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

[4] 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

[5] 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

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

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

[8] 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

[9] 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

[10] 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.

[11] 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.

[12] 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.

[13] 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

[14] 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

[15] F. Shang, T. Xu, Y. Liu, H. Liu, L. Shen, and M. Gong, "Differentially private admm algorithms for machine learning," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 4733–4745, 2021.

[16] F. Sattler, K. Müller, T. Wiegand, and W. Samek, "On the byzantine robustness of clustered federated learning," in *2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Barcelona, Spain, 2020, pp. 8861–8865.

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

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

[20] 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.

[21] 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.

[22] 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

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

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

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

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

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

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

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

[30] 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.

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

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

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

[34] 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.

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

[36] E. Ryu and S. Boyd, "Stochastic proximal iteration: A non-asymptotic improvement upon stochastic gradient descent," Stanford, Tech. Rep., 2017.

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

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

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

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

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

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

[43] 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.

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

[45] 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

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

[47] R. Nassif, S. Vlaski, C. Richard, J. Chen, and A. Sayed, "Multitask learning over graphs: An approach for distributed, streaming machine learning," *IEEE Signal Processing Magazine*, vol. 37, no. 3, pp. 14–25, 2020.

[48] O. Shamir and N. Srebro, "Distributed stochastic optimization and learning," in *2014 52nd Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, 2014, pp. 850–857.

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

[50] 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.

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

[52] 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.

[53] A. Chambolle and T. Pock, "An introduction to continuous optimization for imaging," *Acta Numer.*, vol. 25, pp. 161–319, 2016. [Online]. Available: http://dx.doi.org/10.1017/S096249291600009X

[54] 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

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

[56] 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.

[57] 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

[58] 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.

[59] 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

[60] 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

[61] 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.

[62] 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.

[63] 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

[64] 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

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

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

[67] D. Koller and N. Friedman, *Probabilistic Graphical Models: Principles and Techniques*, ser. Adaptive computation and machine learning. MIT Press, 2009.

[68] S. Banerjee, B. Carlin, and A. Gelfand, *Hierarchical Modeling and Analysis for Spatial Data*. Chapman and Hall/CRC, 2015.

[69] M. Straathof, M. Sinke, T. Roelofs, E. Blezer, R. Sarabdjitsingh, A. van der Toorn, O. Schmitt, W. Otte, and R. Dijkhuizen, "Distinct structure-function relationships across cortical regions and connectivity scales in the rat brain," *Scientific Reports*, vol. 10, no. 1, p. 56, 2020. [Online]. Available: https://doi.org/10.1038/s41598-019-56834-9

[70] F. Zhao and L. J. Guibas, *Wireless Sensor Networks: An Information Processing Approach*. Amsterdam, The Netherlands: Morgan Kaufmann, 2004.

[71] D. Alvarez-Melis and N. Fusi, "Geometric dataset distances via optimal transport," *Advances in Neural Information Processing Systems*, vol. 33, pp. 21 428–21 439, 2020.

[72] A. Jung, G. Hannak, and N. Görtz, "Graphical LASSO Based Model Selection for Time Series," *IEEE Sig. Proc. Letters*, vol. 22, no. 10, Oct. 2015.

[73] 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.

[74] A. Jung, "Learning the conditional independence structure of stationary time series: A multitask learning approach," *IEEE Trans. Signal Processing*, vol. 63, no. 21, Nov. 2015.

[75] N. Tran, O. Abramenko, and A. Jung, "On the sample complexity of graphical model selection from non-stationary samples," *IEEE Transactions on Signal Processing*, vol. 68, pp. 17–32, 2020.

[76] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. R. Salakhutdinov, and A. J. Smola, "Deep sets," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper/2017/file/f22e4747da1aa27e363d86d40ff442fe-Paper.pdf

[77] 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.

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

[79] 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

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

[81] 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.

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

[83] 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.

[84] 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.

[85] J. Zhang, B. Chen, X. Cheng, H. T. T. Binh, and S. Yu, "Poisongan: Generative poisoning attacks against federated learning in edge com-

puting systems," *IEEE Internet of Things Journal*, vol. 8, no. 5, pp. 3310–3322, 2021.

[86] 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.

[87] 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

[88] 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

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

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

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

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

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

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

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

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

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

[98] 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