5.7 Stochastic GD

Consider the GD steps (5.6) for minimizing the empirical risk (5.1). The gradient $\nabla f(\mathbf{w})$ of the objective function (5.1) has a particular structure. Indeed, this gradient is a sum

$$\nabla f(\mathbf{w}) = (1/m) \sum_{i=1}^{m} \nabla f_i(\mathbf{w}) \text{ with } f_i(\mathbf{w}) := L\left((\mathbf{x}^{(i)}, y^{(i)}), h^{(\mathbf{w})}\right). \tag{5.26}$$

Each component of the sum (5.26) corresponds to one particular data points $(\mathbf{x}^{(i)}, y^{(i)})$, for $i = 1, \ldots, m$. We need to compute a sum of the form (5.26) for each new GD step (5.6).

Computing the sum in (5.26) can be computationally challenging for at least two reasons. First, computing the sum is challenging for very large datasets with m in the order of billions. Second, for datasets which are stored in different data centres located all over the world, the summation would require a huge amount of network resources. Moreover, the finite transmission rate of communication networks limits the rate by which the GD steps (5.6) can be executed.

The idea of SGD is to replace the exact gradient $\nabla f(\mathbf{w})$ (5.26) by an approximation that is easier to compute than a direct evaluation of (5.26). The word "stochastic" in the name SGD hints already at the use of a stochastic approximation $g(\mathbf{w}) \approx \nabla f(\mathbf{w})$. It turns out that using a gradient approximation $g(\mathbf{w})$ can result in significant savings in computational complexity while incurring a graceful degradation in the overall optimization accuracy. The optimization accuracy (distance to minimum of $f(\mathbf{w})$) depends crucially on the "gradient noise"

$$\varepsilon := \nabla f(\mathbf{w}) - g(\mathbf{w}). \tag{5.27}$$

The elementary step of most SGD methods is obtained from the GD step (5.6) by replacing the exact gradient $\nabla f(\mathbf{w})$ with some stochastic approximation $g(\mathbf{w})$,

$$\mathbf{w}^{(r+1)} = \mathbf{w}^{(r)} - \alpha_r g\big(\mathbf{w}^{(r)}\big), \tag{5.28}$$

As the notation in (5.28) indicates, SGD methods use a learning rate α_r that varies between different iterations.

To avoid accumulation of the gradient noise (5.27) during the SGD updates (5.28), SGD methods typically decrease the learning rate α_r as the iterations proceed. The precise dependence of the learning rate α_r on the iteration index r is referred to as a learning rate schedule [48, Chapter 8]. One possible choice for the learning rate schedule is $\alpha_r := 1/r$ [101].

Exercise 5.3 discusses conditions on the learning rate schedule that guarantee convergence of the updates SGD to the minimum of $f(\mathbf{w})$.

The approximate ("noisy") gradient $g(\mathbf{w})$ can be obtained by different randomization strategies. The most basic form of SGD constructs the gradient approximation $g(\mathbf{w})$ by replacing the sum (5.26) with a randomly selected component,

$$g(\mathbf{w}) := \nabla f_{\hat{i}}(\mathbf{w}). \tag{5.29}$$

The index $\hat{i}$ is chosen randomly from the set $\{1, \ldots, m\}$. The resulting SGD method then repeats the update

$$\mathbf{w}^{(r+1)} = \mathbf{w}^{(r)} - \alpha \nabla f_{\hat{i}_r}(\mathbf{w}^{(r)}), \tag{5.30}$$

sufficiently often. Every update (5.30) uses a "fresh" randomly chosen (drawn) index $\hat{i}_r$. Formally, the indices $\hat{i}_r$ are realizations of i.i.d. RVs whose common probability distribution is the uniform distribution over the index set $\{1, \ldots, m\}$.

Note that (5.30) replaces the summation over the training set during the GD step (5.6) by randomly selecting a single component of this sum. The resulting savings in computational complexity can be significant when the training set consists of a large number of data points that might be stored in a distributed fashion (in the "cloud"). The saving in computational complexity of SGD comes at the cost of introducing a non-zero gradient noise

$$\begin{aligned} \varepsilon_r &\overset{(5.27)}{=} \nabla f(\mathbf{w}^{(r)}) - g\big(\mathbf{w}^{(r)}\big) \\ &= \nabla f(\mathbf{w}^{(r)}) - \nabla f_{\hat{i}_r}(\mathbf{w}). \end{aligned} \tag{5.31}$$

Mini-Batch SGD. Let us now discuss a variant of SGD that tries to reduce the approximation error (gradient noise) (5.31) arising in the SGD step (5.30). The idea behind this variant, referred to as mini-batch SGD, is quite simple. Instead of using only a single randomly selected component $\nabla f_i(\mathbf{w})$ (see (5.26)) for constructing a gradient approximation, mini-batch SGD uses several randomly chosen components.

We summarize mini-batch SGD in Algorithm 4 which requires an integer batch size B as input parameter. Algorithm 4 repeats the SGD step (5.28) using a gradient approximation that is constructed from a randomly selected subset $\mathcal{B} = \{i_1, \ldots, i_B\}$ (a "batch"),

$$g\big(\mathbf{w}\big) = (1/B) \sum_{i' \in \mathcal{B}} \nabla f_{i'}(\mathbf{w}). \tag{5.32}$$

For each new iteration of Algorithm 4, a new batch $\mathcal{B}$ is generated by a random generator. Note that Algorithm 4 includes the basic SGD variant (5.30) as a special case for the batch

Algorithm 4 Mini-Batch SGD

Input: components $f_i(\mathbf{w})$, for $i = 1, \ldots, m$ of objective function $f(\mathbf{w}) = \sum_{i=1}^{m} f_i(\mathbf{w})$; batch size B; learning rate schedule $\alpha_r > 0$.

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

1: **repeat**
2: randomly select a batch $\mathcal{B} = \{i_1, \ldots, i_B\} \subseteq \{1, \ldots, m\}$ of indices that select a subset of components f_i
3: compute an approximate gradient $g\big(\mathbf{w}^{(r)}\big)$ using (5.32)
4: $r := r + 1$ (increase iteration counter)
5: $\mathbf{w}^{(r)} := \mathbf{w}^{(r-1)} - \alpha_r g\big(\mathbf{w}^{(r-1)}\big)$
6: **until** stopping criterion met

Output: $\mathbf{w}^{(r)}$ (which approximates $\operatorname{argmin}_{\mathbf{w} \in \mathbb{R}^n} f(\mathbf{w})$))

size $B = 1$. Another special case is $B = m$, where the SGD step 5 in Algorithm 4 becomes an ordinary GD step (5.6).

Online Learning. A main motivation for the SGD step (5.30) is that a training set is already collected but so large that the sum in (5.26) is computationally intractable. Another variant of SGD is obtained for sequential (time-series) data. In particular, consider data points that are gathered sequentially, one new data point $\big(\mathbf{x}^{(t)}, y^{(t)}\big)$ at each time instant $t = 1, 2, \ldots$. With each new data point $\big(\mathbf{x}^{(t)}, y^{(t)}\big)$ we can access a new component $f_t(\mathbf{w}) = L\left((\mathbf{x}^{(t)}, y^{(t)}), h^{(\mathbf{w})}\right)$ (see (5.1)). For sequential data, we can use a slight modification of the SGD step (5.28) to obtain an online learning method (see Section 4.7). This online variant of SGD computes, at each time instant t,

$$\mathbf{w}^{(t)} := \mathbf{w}^{(t-1)} - \alpha_t \nabla f_t(\mathbf{w}^{(t-1)}). \tag{5.33}$$

5.8 Advanced Gradient-Based Methods

The main idea underlying GD and SGD is to approximate the objective function (5.1) locally around a current guess or approximation $\mathbf{w}^{(r)}$ for the optimal weights. This local approximation is a tangent hyperplane whose normal vector is determined by the gradient $\nabla f(\mathbf{w}^{(r)}$. We then obtain an updated (improved) approximation by minimizing this local approximation, i.e., by doing a GD step (5.6).

The idea of advanced gradient methods [48, Ch. 8] is to exploit the information provided

by the gradients $\nabla f\big(\mathbf{w}^{(r')}\big)$ at previous iterations $r' = 1, \ldots, r$, to build an improved local approximation of $f(\mathbf{w})$ around a current iterate $\mathbf{w}^{(r)}$. Figure 5.4 indicates such an improved local approximation of $f(\mathbf{w})$ which is non-linear (e.g., quadratic). These improved local approximations can be used to adapt the learning rate during the GD steps (5.6).

Advanced gradient-based methods use improved local approximations to modify the gradient $\nabla f\big(\mathbf{w}^{(r')}\big)$ to obtain an improved update direction. Figure 5.5 depicts the contours of an objective function $f(\mathbf{w})$ for which the gradient $\nabla f\big(\mathbf{w}^{(r)}\big)$ points only weakly towards the optimal parameter vector $\overline{\mathbf{w}}$ (minimizer of $f(\mathbf{w})$). The gradient history $\nabla f\big(\mathbf{w}^{(r')}\big)$, for$r' = 1, \ldots, r$, allows to detect such an unfavourable geometry of the objective function. Moreover, the gradient history can be used to "correct" the update direction $\nabla f\big(\mathbf{w}^{(r)}\big)$ to obtain an improved update direction towards the optimum parameter vector $\overline{\mathbf{w}}$.

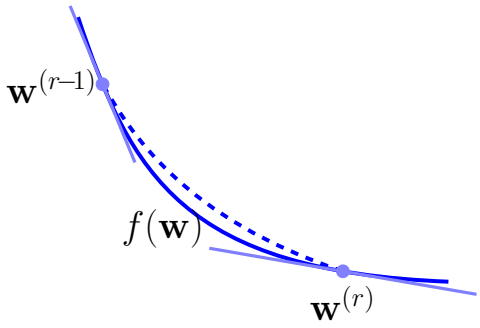

Figure 5.4: Advanced gradient-based methods use the gradients at previous iterations to construct an improved (non-linear) local approximation (dashed) of the objective function $f(\mathbf{w})$ (solid).

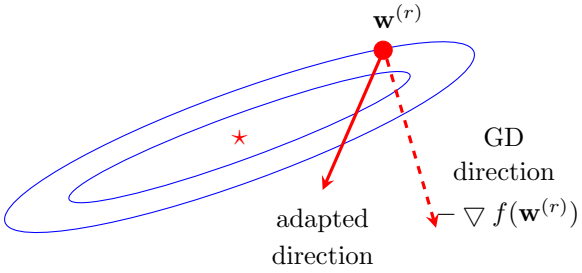

Figure 5.5: Advanced gradient-based methods use improved (non-linear) local approximations of the objective function $f(\mathbf{w})$ to "nudge" the update direction towards the optimal parameter vector $\overline{\mathbf{w}}$. The update direction of plain vanilla GD (5.6) is the negative gradient $-\nabla f\big(\mathbf{w}^{(r)}\big)$. For some objective functions the negative gradient might be only weakly correlated with the straight direction from $\mathbf{w}^{(r)}$ towards the optimal parameter vector ($\star$).

5.9 Exercises

Exercise 5.1. Use Knowledge About Problem Class. Consider the space $\mathcal{P}$ of sequences $f = (f[0], f[1], \ldots)$ that have the following properties

- for each sequence there is an index $i^{(f)}$ such that f is monotone increasing for indices $i' \geq i^{(f)}$ and monotone decreasing for indices $i' \geq i^{(f)}$
- any change point r of f, where $f[r] \neq f[r+1]$ can only occur at integer multiples of 100, e.g., $r = 100$ or $r = 300$.

Given a function $f \in \mathcal{P}$ and starting point r_0 our goal is to find the minimum value of $\min_r f[r] = f[r^{(f)}]$ as quickly as possible. Can you construct an iterative algorithm that can access a given function f only by querying the values $f[r], f[r-1]$ and $f[r+1]$ for any given index r.

Exercise 5.2. Maximum Eigenvalue of Flipped Product. Show that $\lambda_{\max}\big(\mathbf{A}\mathbf{A}^T\big) = \lambda_{\max}\big(\mathbf{A}^T\mathbf{A}\big)$ for any given matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$.

Exercise 5.3. Learning rate Schedule for SGD Let us learn a linear hypothesis $h(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$ using data points that arrive sequentially at discrete time instants $t = 0, 1, \ldots$. At time t, we gather a new data point $\big(\mathbf{x}^{(r)}, y^{(r)}\big)$. The data points can be modelled as realizations of i.i.d. copies of a RV $\big(\mathbf{x}, y\big)$. The probability distribution of the features $\mathbf{x}$ is

a standard multivariate normal distribution $\mathcal{N}(\mathbf{0}, \mathbf{I})$. The label of a random[2] data point is related to its features via $y = \overline{\mathbf{w}}^T \mathbf{x} + \varepsilon$ with some fixed but unknown true parameter vector $\overline{\mathbf{w}}$. The additive noise $\varepsilon \sim \mathcal{N}(0, 1)$ follows a standard normal distribution. We use SGD to learn the parameter vector $\mathbf{w}$ of a linear hypothesis,

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \alpha_t \big(\big(\mathbf{w}^{(t)}\big)^T \mathbf{x}^{(t)} - y^{(t)} \big) \mathbf{x}^{(t)}. \tag{5.34}$$

with learning rate schedule $\alpha_t = \beta / t^\gamma$. Note that we compute a new SGD iteration (5.34) for each new time instant t. What conditions on the hyper-parameters β, γ ensure that $\lim_{t \to \infty} \mathbf{w}^{(t)} = \overline{\mathbf{w}}$ in distribution?

Exercise 5.4. ImageNet. The "ImageNet" database contains more than 10^6 images [80]. These images are labeled according to their content (e.g., does the image show a dog?) and stored as a file of size at least 4 kilobytes. We want to learn a classifier that allows to predict if an image shows a dog or not. To learn this classifier we run GD for logistic regression on a small computer that has 32 kilobytes memory and is connected to the internet with bandwidth of 1 Mbit/s. Therefore, for each single GD update (5.6) it must essentially download all images in ImageNet. How long would such a single GD update take ?

Exercise 5.5. Apple or No Apple? Consider data points being images of size of 1024×1024 pixels. Each image is characterized by the RGB pixel color intensities (value range $0, \ldots, 255$ resuling in 24 bits for each pixel), which we stack into a feature vector $\mathbf{x} \in \mathbb{R}^n$. We assign each image the label $y = 1$ if it shows an apple and $y = -1$ if it does not show an apple. We use logistic regression to learn a linear hypothesis $h(\mathbf{x}) = \mathbf{w}^T \mathbf{x}$ to classify an image according to $\hat{y} = 1$ if $h(\mathbf{x}) \geq 0$. The training set consists of $m = 10^{10}$ labeled images which are stored in the cloud. We implement the ML method on our own laptop which is connected to the internet with a bandwidth of at most 100 Mbps. Unfortunately we can only store at most five images on our computer. How long does it take at least to complete one single GD step ?

Exercise 5.6. Feature Normalization To Speed Up GD Consider the dataset with feature vectors $\mathbf{x}^{(1)} = (100, 0)^T \in \mathbb{R}^2$ and $\mathbf{x}^{(2)} = (0, 1/10)^T \in \mathbb{R}^2$ which we stack into the matrix $\mathbf{X} = (\mathbf{x}^{(1)}, \mathbf{x}^{(2)})^T$. What is the condition number of $\mathbf{X}^T \mathbf{X}$? What is the condition number of $\big(\widehat{\mathbf{X}}\big)^T \widehat{\mathbf{X}}$ with the matrix $\widehat{\mathbf{X}} = (\widehat{\mathbf{x}}^{(1)}, \widehat{\mathbf{x}}^{(2)})^T$ constructed from the normalized feature vectors $\widehat{\mathbf{x}}^{(i)}$ delivered by Algorithm 3.

[2]More precisely, a data point that is obtained as the realization of RV.

Exercise 5.7. Convergence of GD Steps Consider a differentiable objective function $f(\mathbf{w})$ whose argument is a parameter vector $\mathbf{w} \in \mathbb{R}^n$. We make no assumption about smoothness or convexity. Thus, the function $f(\mathbf{w})$ might be non-convex and might also be not β smooth. However, the gradient $\nabla f(\mathbf{w})$ is uniformly upper bounded $\|\nabla f(\mathbf{w})\| \leq 100$ for every $\mathbf{w}$. Starting from some initial vector $\mathbf{w}^{(0)}$ we construct a sequence of parameter vectors using GD steps,

$$\mathbf{w}^{(r+1)} = \mathbf{w}^{(r)} - \alpha_r \nabla f\big(\mathbf{w}^{(r)}\big). \tag{5.35}$$

The learning rate α_r in (5.35) is allowed to vary between different iterations. Can you provide sufficient conditions on the evolution of the learning rate α_r, as iterations proceed, that ensure convergence of the sequence $\mathbf{w}^{(0)}, \mathbf{w}^{(1)}, \ldots$.

Chapter 6

Model Validation and Selection

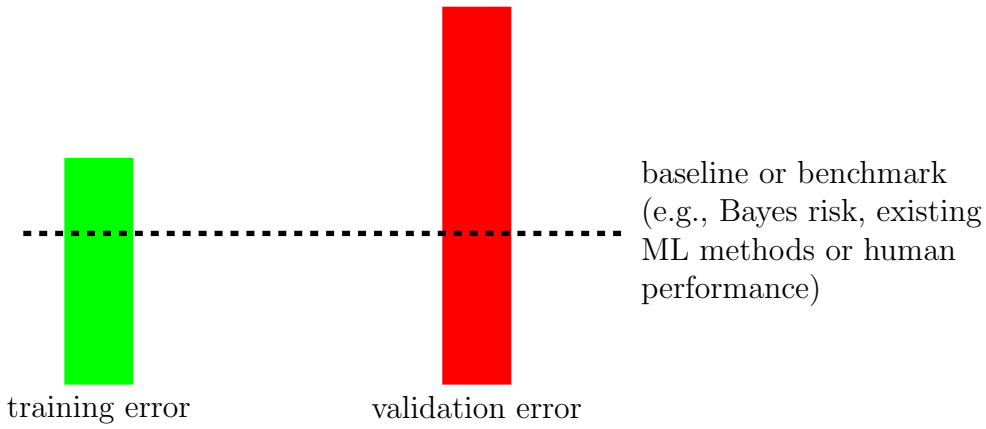

Figure 6.1: We can diagnose a ML method by comparing its training error with its validation error. Ideally both are on the same level as a baseline (or benchmark error level).

Chapter 4 discussed ERM as a principled approach to learning a useful hypothesis out of a hypothesis space or model. In particular, ERM-based ML methods learn a hypothesis $\widehat{h} \in \mathcal{H}$ that incurs minimum average loss on a set of labeled data points that serve as the training set. We refer to the average loss incurred by a hypothesis on the training set as the training error. The minimum average loss achieved by a hypothesis that solves the ERM might be referred to as the training error of the overall ML method. Note that a specific ML method consists of specific design choices for the hypothesis space (or model) and loss function (see Chapter 3).

ERM is sensible only if the training error of a hypothesis is an reliable approximation

for its (average) loss incurred on data points outside the training set. We say that a learnt hypothesis generalizes well if the loss incurred outside the training set is not significantly larger than the average loss on the training set. Whether the training error of a hypothesis is a reliable approximation for its loss on data points outside the training set depends on both, the statistical properties of the data points and on the hypothesis space used by the ML method.

Modern ML methods typically use hypothesis spaces with large effective dimension (see Section 2.2). As an example consider linear regression (see Section 3.1) with data points having a large number n of features (this setting is referred to as the high-dimensional regime). The effective dimension of the linear hypothesis space (3.1), which is used by linear regression, is equal to the number n of features. Modern technology allows to collect a huge number of features about individual data points which implies, in turn, that the effective dimension of (3.1) is large. Another example of a high-dimensional hypothesis space arises in deep learning methods using a hypothesis space are constituted by all maps represented by an ANN with billions of tunable parameters.

A high-dimensional hypothesis space is very likely to contain a hypothesis that perfectly fits any given training set. Such a hypothesis achieves a very small training error but might incur a large loss when predicting the labels of a data point that is not included in training set. Thus, the (minimum) training error achieved by a hypothesis learnt by ERM can be misleading. We say that a ML method, such as linear regression using too many features, overfits the training set when it learns a hypothesis (e.g., via ERM) that has small training error but incurs much larger loss outside the training set.

Section 6.1 shows that linear regression is likely to overfit a training set if the number of features of a data point exceeds the size of the training set. Section 6.2 demonstrates how to validate a learnt hypothesis by computing its average loss on data points outside the training set. We refer to the set of data points used to validate the learnt hypothesis as a validation set. If a ML method overfits the training set, it learns a hypothesis whose training error is much smaller than its validation error. We can detect if a ML method overfits by comparing its training error with its validation error (see Figure 6.1).

We can use the validation error not only to detect if a ML method overfits. The validation error can also be used as a quality measure for the hypothesis space or model used by the ML method. This is analogous to the concept of a loss function that allows us to evaluate the quality of a hypothesis $h \in \mathcal{H}$. Section 6.3 shows how to select between ML methods using different models by comparing their validation errors.

Section 6.4 uses a simple probabilistic model for the data to study the relation between the training error of a learnt hypothesis and its expected loss (see (4.1)). This probabilistic analysis reveals the interplay between the data, the hypothesis space and the resulting training error and validation error of a ML method.

Section 6.5 discusses the bootstrap as a simulation based alternative to the probabilistic analysis of Section 6.4. While Section 6.4 assumes a specific probability distribution of the data points, the bootstrap does not require the specification of a probability distribution underlying the data.

As indicated in Figure 6.1, for some ML applications, we might have a baseline (or benchmark) for the achievable performance of ML methods. Such a baseline might be obtained from existing ML methods, human performance levels or from a probabilistic model (see Section 6.4). Section 6.6 details how the comparison between training error, validation error and (if available) a baseline informs possible improvements for a ML method. These improvements might be obtained by collecting more data points, using more features of data points or by changing the hypothesis space (or model).

Having a baseline for the expected loss, such as the Bayes risk, allows to tell if a ML method already provides satisfactory results. If the training error and the validation error of a ML method are close to the baseline, there might be little point in trying to further improve the ML method.

6.1 Overfitting

We now have a closer look at the occurrence of overfitting in linear regression methods. As discussed in Section 3.1, linear regression methods learn a linear hypothesis $h(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$ which is parametrized by the parameter vector $\mathbf{w} \in \mathbb{R}^n$. The learnt hypothesis is then used to predict the numeric label $y \in \mathbb{R}$ of a data point based on its feature vector $\mathbf{x} \in \mathbb{R}^n$. Linear regression aims at finding a parameter vector $\widehat{\mathbf{w}}$ with minimum average squared error loss incurred on a training set

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

The training set $\mathcal{D}$ consists of m data points $\big(\mathbf{x}^{(i)}, y^{(i)}\big)$, for $i = 1, \ldots, m$, with known label values $y^{(i)}$. We stack the feature vectors $\mathbf{x}^{(i)}$ and labels $y^{(i)}$, respectively, of the data points in the training set into the feature matrix $\mathbf{X} = (\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)})^T$ and label vector $\mathbf{y} = (y^{(1)}, \ldots, y^{(m)})^T$.

The ERM (4.13) of linear regression is solved by any parameter vector $\widehat{\mathbf{w}}$ that solves (4.11). The (minimum) training error of the hypothesis $h^{(\widehat{\mathbf{w}})}$ is obtained as

$$\widehat{L}(h^{(\widehat{\mathbf{w}})} \mid \mathcal{D}) \stackrel{(4.4)}{=} \min_{\mathbf{w} \in \mathbb{R}^n} \widehat{L}(h^{(\mathbf{w})}|\mathcal{D}) \stackrel{(4.13)}{=} \|(\mathbf{I}-\mathbf{P})\mathbf{y}\|_2^2. \tag{6.1}$$

Here, we used the orthogonal projection matrix $\mathbf{P}$ on the linear span

$$\text{span}\{\mathbf{X}\} = \big\{\mathbf{X}\mathbf{a} : \mathbf{a} \in \mathbb{R}^n\big\} \subseteq \mathbb{R}^m,$$

of the feature matrix $\mathbf{X} = (\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)})^T \in \mathbb{R}^{m \times n}$.

In many ML applications we have access to a huge number of individual features to characterize a data point. As a point in case, consider a data point which is a snapshot obtained from a modern smartphone camera. These cameras have a resolution of several megapixels. Here, we can use millions of pixel colour intensities as its features. For such applications, it is common to have more features for data points than the size of the training set,

$$n \geq m. \tag{6.2}$$

Whenever (6.2) holds, the feature vectors $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)} \in \mathbb{R}^n$ of the data points in $\mathcal{D}$ are typically linearly independent. As a case in point, if the feature vectors $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)} \in \mathbb{R}^n$ are realizations of i.i.d. RVs with a continuous probability distribution, these vectors are linearly independent with probability one [100].

If the feature vectors $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)} \in \mathbb{R}^n$ are linearly independent, the span of the feature matrix $\mathbf{X} = (\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)})^T$ coincides with $\mathbb{R}^m$ which implies, in turn, $\mathbf{P} = \mathbf{I}$. Inserting $\mathbf{P} = \mathbf{I}$ into (4.13) yields

$$\widehat{L}(h^{(\widehat{\mathbf{w}})} \mid \mathcal{D}) = 0. \tag{6.3}$$

As soon as the number $m = |\mathcal{D}|$ of training data points does not exceed the number n of features that characterize data points, there is (with probability one) a linear predictor $h^{(\widehat{\mathbf{w}})}$ achieving zero training error(!).

While the hypothesis $h^{(\widehat{\mathbf{w}})}$ achieves zero training error, it will typically incur a non-zero average prediction error $y - h^{(\widehat{\mathbf{w}})}(\mathbf{x})$ on data points $(\mathbf{x}, y)$ outside the training set (see Figure 6.2). Section 6.4 will make this statement more precise by using a probabilistic model for the data points within and outside the training set.

Note that (6.3) also applies if the features $\mathbf{x}$ and labels y of data points are completely

unrelated. Consider an ML problem with data points whose labels y and features are realizations of a RV that are statistically independent. Thus, in a very strong sense, the features $\mathbf{x}$ contain no information about the label of a data point. Nevertheless, as soon as the number of features exceeds the size of the training set, such that (6.2) holds, linear regression methods will learn a hypothesis with zero training error.

We can easily extend the above discussion about the occurrence of overfitting in linear regression to other methods that combine linear regression with a feature map. Polynomial regression, using data points with a single feature z, combines linear regression with the feature map $z \mapsto \mathbf{\Phi}(z) := \big(z^0, \ldots, z^{n-1}\big)^T$ as discussed in Section 3.2.

It can be shown that whenever (6.2) holds and the features $z^{(1)}, \ldots, z^{(m)}$ of the training set are all different, the feature vectors $\mathbf{x}^{(1)} := \mathbf{\Phi}\big(z^{(1)}\big), \ldots, \mathbf{x}^{(m)} := \mathbf{\Phi}\big(z^{(m)}\big)$ are linearly independent. This implies, in turn, that polynomial regression is guaranteed to find a hypothesis with zero training error whenever $m \leq n$ and the data points in the training set have different feature values.

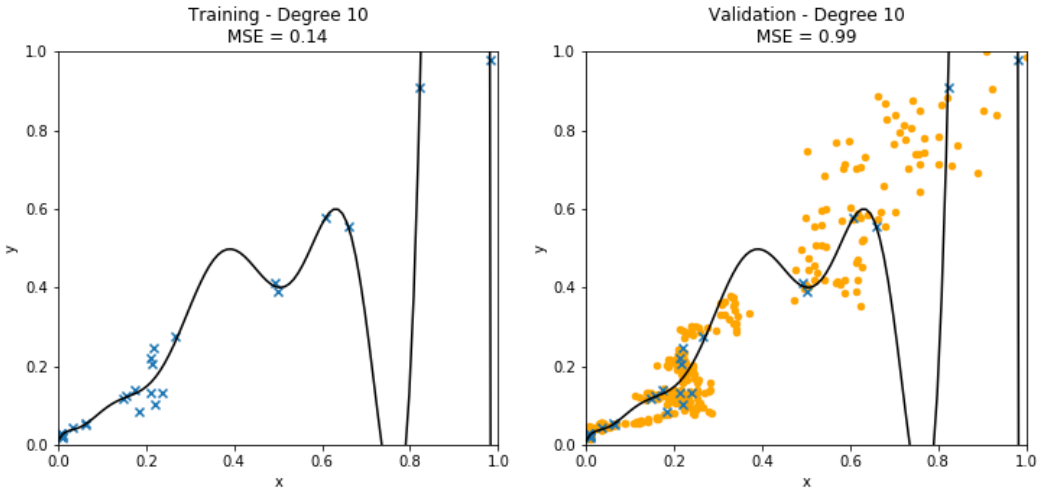

Figure 6.2: Polynomial regression learns a polynomial map with degree $n-1$ by minimizing its average loss on a training set (blue crosses). Using high-degree polynomials (large n) results in a small training error. However, the learnt high-degree polynomial performs poorly on data points outside the training set (orange dots).

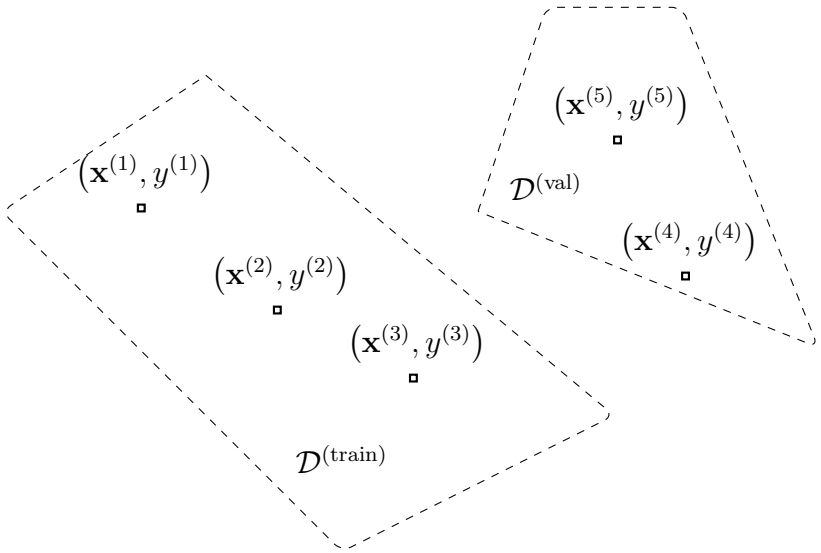

Figure 6.3: We split the dataset $\mathcal{D}$ into two subsets, a training set $\mathcal{D}^{(\text{train})}$ and a validation set $\mathcal{D}^{(\text{val})}$. We use the training set to learn (find) the hypothesis $\widehat{h}$ with minimum empirical risk $\widehat{L}(\widehat{h}|\mathcal{D}^{(\text{train})})$ on the training set (4.3). We then validate $\widehat{h}$ by computing its average loss $\widehat{L}(\widehat{h}|\mathcal{D}^{(\text{val})})$ on the validation set $\mathcal{D}^{(\text{val})}$. The average loss $\widehat{L}(\widehat{h}|\mathcal{D}^{(\text{val})})$ obtained on the validation set is the validation error. Note that $\widehat{h}$ depends on the training set $\mathcal{D}^{(\text{train})}$ but is completely independent of the validation set $\mathcal{D}^{(\text{val})}$.

6.2 Validation

Consider an ML method that uses ERM (4.3) to learn a hypothesis $\widehat{h} \in \mathcal{H}$ out of the hypothesis space $\mathcal{H}$. The discussion in Section 6.1 revealed that the training error of a learnt hypothesis $\widehat{h}$ can be a poor indicator for the performance of $\widehat{h}$ for data points outside the training set. The hypothesis $\widehat{h}$ tends to "look better" on the training set over which it has been tuned within ERM. The basic idea of validating the predictor $\widehat{h}$ is simple:

- first we learn a hypothesis $\widehat{h}$ using ERM on a training set and
- then we compute the average loss of $\widehat{h}$ on data points that do not belong to the training set.

Thus, validation means to compute the average loss of a hypothesis using data points that have not been used in ERM to learn that hypothesis.

Assume we have access to a dataset of m data points,

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

Each data point is characterized by a feature vector $\mathbf{x}^{(i)}$ and a label $y^{(i)}$. Algorithm 5 outlines how to learn and validate a hypothesis $h \in \mathcal{H}$ by splitting the dataset $\mathcal{D}$ into a training set and a validation set. The random shuffling in step 1 of Algorithm 5 ensures the i.i.d. assumption for the shuffled data. Section 6.2.1 shows next how the i.i.d. assumption ensures that the validation error (6.6) approximates the expected loss of the hypothesis $\widehat{h}$. The hypothesis $\widehat{h}$ is learnt via ERM on the training set during step 4 of Algorithm 5.

6.2.1 The Size of the Validation Set

The choice of the split ratio $\rho \approx m_t/m$ in Algorithm 5 is often based on trial and error. We try out different choices for the split ratio and pick the one with the smallest validation error. It is difficult to make a precise statement on how to choose the split ratio which applies broadly [83]. This difficulty stems from the fact that the optimal choice for ρ depends on the precise statistical properties of the data points.

One approach to determine the required size of the validation set is to use a probabilistic model for the data points. The i.i.d. assumption is maybe the most widely used probabilistic model within ML. Here, we interpret data points as the realizations of i.i.d. RVs. These i.i.d. RVs have a common (joint) probability distribution $p(\mathbf{x}, y)$ over possible features $\mathbf{x}$

Algorithm 5 Validated ERM

Input: model $\mathcal{H}$, loss function L, dataset $\mathcal{D} = \big\{\big(\mathbf{x}^{(1)}, y^{(1)}\big), \ldots, \big(\mathbf{x}^{(m)}, y^{(m)}\big)\big\}$; split ratio ρ

1: randomly shuffle the data points in $\mathcal{D}$

2: create the training set $\mathcal{D}^{(\text{train})}$ using the first $m_t = \lceil \rho m \rceil$ data points,

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

3: create the validation set $\mathcal{D}^{(\text{val})}$ by the $m_v = m - m_t$ remaining data points,

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

4: learn hypothesis $\widehat{h}$ via ERM on the training set,

$$\widehat{h} := \operatorname*{argmin}_{h \in \mathcal{H}} \widehat{L}\big(h|\mathcal{D}^{(\text{train})}\big) \tag{6.4}$$

5: compute the training error

$$E_t := \widehat{L}\big(\widehat{h}|\mathcal{D}^{(\text{train})}\big) = (1/m_t) \sum_{i=1}^{m_t} L\left((\mathbf{x}^{(i)}, y^{(i)}), \widehat{h}\right). \tag{6.5}$$

6: compute the validation error

$$E_v := \widehat{L}\big(\widehat{h}|\mathcal{D}^{(\text{val})}\big) = (1/m_v) \sum_{i=m_t+1}^{m} L\left((\mathbf{x}^{(i)}, y^{(i)}), \widehat{h}\right). \tag{6.6}$$

Output: learnt hypothesis $\widehat{h}$, training error E_t, validation error E_v

and labels y of a data point. Under the i.i.d. assumption, the validation error E_v (6.6) also becomes a realization of a RV. The expectation (or mean) $\mathbb{E}\{E_v\}$ of this RV is precisely the risk $\mathbb{E}\{L\left((\mathbf{x},y),\widehat{h}\right)\}$ of $\widehat{h}$ (see (4.1)).

Within the above i.i.d. assumption, the validation error E_v becomes a realization of a RV that fluctuates around its mean $\mathbb{E}\{E_v\}$. We can quantify this fluctuation using the variance

$$\sigma^2_{E_v} := \mathbb{E}\big\{\big(E_v - \mathbb{E}\{E_v\}\big)^2\big\}.$$

Note that the validation error is the average of the realizations $L\left((\mathbf{x}^{(i)},y^{(i)}),\widehat{h}\right)$ of i.i.d. RVs. The probability distribution of the RV $L\left((\mathbf{x},y),\widehat{h}\right)$ is determined by the probability distribution $p(\mathbf{x},y)$, the choice of loss function and the hypothesis $\widehat{h}$. In general, we do not know $p(\mathbf{x},y)$ and, in turn, also do not know the probability distribution of $L\left((\mathbf{x},y),\widehat{h}\right)$.

If we know an upper bound U on the variance of the (random) loss $L\left((\mathbf{x}^{(i)},y^{(i)}),\widehat{h}\right)$, we can bound the variance of E_v as

$$\sigma^2_{E_v} \leq U/m_v.$$

We can then, in turn, ensure that the variance $\sigma^2_{E_v}$ of the validation error E_v does not exceed a given threshold η, say $\eta = (1/100)E_t^2$, by using a validation set of size

$$m_v \geq U/\eta. \tag{6.7}$$

The lower bound (6.7) is only useful if we can determine an upper bound U on the variance of the RV $L\left((\mathbf{x},y),\widehat{h}\right)$ where $(\mathbf{x},y)$ is a RV with probability distribution $p(\mathbf{x},y)$. An upper bound on the variance of $L\left((\mathbf{x},y),\widehat{h}\right)$ can be derived using probability theory if we know an accurate probabilistic model $p(\mathbf{x},y)$ for the data points. Such a probabilistic model might be provided by application-specific scientific fields such as biology or psychology. Another option is to estimate the variance of $L\left((\mathbf{x},y),\widehat{h}\right)$ using the sample variance of the actual loss values $L\left((\mathbf{x}^{(1)},y^{(1)}),\widehat{h}\right),\ldots,L\left((\mathbf{x}^{(m)},y^{(m)}),\widehat{h}\right)$ obtained for the dataset $\mathcal{D}$.

6.2.2 k-Fold Cross Validation

Algorithm 5 uses the most basic form of splitting a given dataset $\mathcal{D}$ into a training set and a validation set. Many variations and extensions of this basic splitting approach have been proposed and studied (see [32] and Section 6.5). One very popular extension of the single split into training set and validation set is known as k-fold cross-validation (k-fold CV) [58,

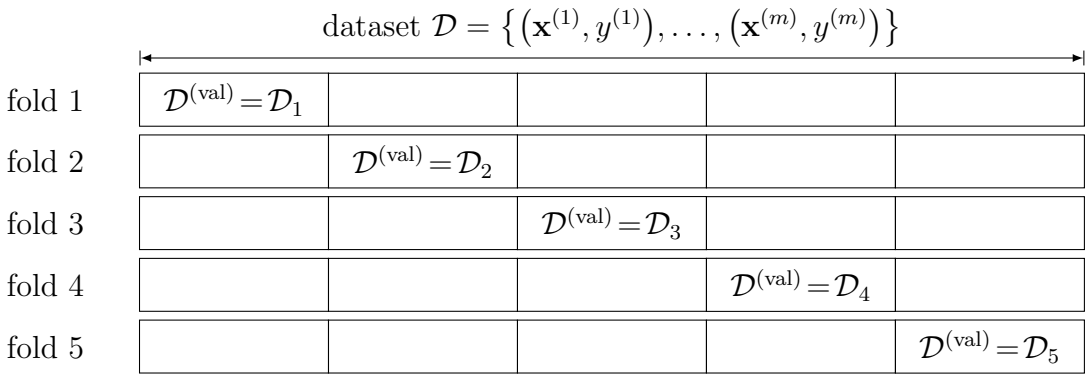

Figure 6.4: Illustration of k-fold CV for $k = 5$. We evenly partition the entire dataset $\mathcal{D}$ into $k = 5$ subsets (or folds) $\mathcal{D}_1, \ldots, \mathcal{D}_5$. We then repeat the validated ERM Algorithm 5 for $k = 5$ times. The bth repetition uses the bth fold $\mathcal{D}_b$ as the validation set and the remaining $k-1(= 4)$ folds as the training set for ERM (4.3).

Sec. 7.10]. We summarize k-fold CV in Algorithm 6 below.

Figure 6.4 illustrates the key principle behind k-fold CV. First, we divide the entire dataset evenly into k subsets which are referred to as "folds". The learning (via ERM) and validation of a hypothesis out of a given hypothesis space $\mathcal{H}$ is then repeated k times. During each repetition, we use one fold as the validation set and the remaining $k-1$ folds as a training set. We then average the values of the training error and validation error obtained for each repetition (fold).

The average (over all k folds) validation error delivered by k-fold CV tends to better estimate the expected loss or risk (4.1) compared to the validation error obtained from a single split in Algorithm 5. Consider a dataset that consists of a relatively small number of data points. If we use a single split of this small dataset into a training set and validation set, we might be very unlucky and choose data points for the validation set which are outliers and not representative for the statistical properties of most data points. The effect of such an unlucky split is typically averaged out when using k-fold CV.

6.2.3 Imbalanced Data

The simple validation approach discussed above requires the validation set to be a good representative for the overall statistical properties of the data. This might not be the case in applications with discrete valued labels and some of the label values being very rare. We might then be interested in having a good estimate of the conditional risks $\mathbb{E}\{L((\mathbf{x}, y), h) | y =$

Algorithm 6 k-fold CV ERM

Input: model $\mathcal{H}$, loss function L, dataset $\mathcal{D} = \big\{\big(\mathbf{x}^{(1)}, y^{(1)}\big), \ldots, \big(\mathbf{x}^{(m)}, y^{(m)}\big)\big\}$; number k of folds

randomly shuffle the data points in $\mathcal{D}$

divide the shuffled dataset $\mathcal{D}$ into k folds $\mathcal{D}_1, \ldots, \mathcal{D}_k$ of size $B = \lceil m/k \rceil$,

$$\mathcal{D}_1 = \big\{\big(\mathbf{x}^{(1)}, y^{(1)}\big), \ldots, \big(\mathbf{x}^{(B)}, y^{(B)}\big)\big\}, \ldots, \mathcal{D}_k = \big\{\big(\mathbf{x}^{((k-1)B+1)}, y^{((k-1)B+1)}\big), \ldots, \big(\mathbf{x}^{(m)}, y^{(m)}\big)\big\} \quad (6.8)$$

for fold index $b = 1, \ldots, k$ **do**

use bth fold as the validation set $\mathcal{D}^{(\text{val})} = \mathcal{D}_b$

use rest as the training set $\mathcal{D}^{(\text{train})} = \mathcal{D} \setminus \mathcal{D}_b$

learn hypothesis $\widehat{h}$ via ERM on the training set,

$$\widehat{h}^{(b)} := \operatorname*{argmin}_{h \in \mathcal{H}} \widehat{L}\big(h | \mathcal{D}^{(\text{train})}\big) \quad (6.9)$$

compute the training error

$$E_t^{(b)} := \widehat{L}\big(\widehat{h} | \mathcal{D}^{(\text{train})}\big) = (1/\big|\mathcal{D}^{(\text{train})}\big|) \sum_{i \in \mathcal{D}^{(\text{train})}} L\left((\mathbf{x}^{(i)}, y^{(i)}), h\right). \quad (6.10)$$

compute **validation error**

$$E_v^{(b)} := \widehat{L}\big(\widehat{h} | \mathcal{D}^{(\text{val})}\big) = (1/\big|\mathcal{D}^{(\text{val})}\big|) \sum_{i \in \mathcal{D}^{(\text{val})}} L\left((\mathbf{x}^{(i)}, y^{(i)}), \widehat{h}\right). \quad (6.11)$$

end for

compute average training and validation errors

$$E_t := (1/k) \sum_{b=1}^{k} E_t^{(b)}, \text{ and } E_v := (1/k) \sum_{b=1}^{k} E_v^{(b)}$$

pick a learnt hypothesis $\widehat{h} := \widehat{h}^{(b)}$ for some $b \in \{1, \ldots, k\}$

Output: learnt hypothesis $\widehat{h}$; average training error E_t; average validation error E_v

$y'\}$ where y' is one of the rare label values. This is more than requiring a good estimate for the risk $\mathbb{E}\{L((\mathbf{x}, y), h)\}$.

Consider data points characterized by a feature vector $\mathbf{x}$ and binary label $y \in \{-1, 1\}$. Assume we aim at learning a hypothesis $h(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$ to classify data points as $\hat{y} = 1$ if $h(\mathbf{x}) \geq 0$ while $\hat{y} = -1$ otherwise. The learning is based on a dataset $\mathcal{D}$ which contains only one single (!) data point with $y = -1$. If we then split the dataset into training and validation set, it is with high probability that the validation set does not include any data point with label value $y = -1$. This cannot happen when using k-fold CV since the single data point must be in one of the validation folds. However, even the applicability of k-fold CV for such an imbalanced dataset is limited since we evaluate the performance of a hypothesis $h(\mathbf{x})$ using only one single data point with $y = -1$. The resulting validation error will be dominated by the loss of $h(\mathbf{x})$ incurred on data points from the majority class (those with true label value $y = 1$).

To learn and validate a hypothesis with imbalanced data, it might be useful to to generate synthetic data points to enlarge the minority class. This can be done using data augmentation techniques which we discuss in Section 7.3. Another option is to choose a loss function that takes the different frequencies of label values into account. Let us illustrate this approach in what follows by an illustrative example.

Consider an imbalanced dataset of size $m = 100$, which contains 90 data points with label $y = 1$ but only 10 data points with label $y = -1$. We might want to put more weight on wrong predictions obtained for data points from the minority class (with true label value $y = -1$). This can be done by using a much larger value for the loss $L((\mathbf{x}, y = -1), h(\mathbf{x}) = 1)$ than for the loss $L((\mathbf{x}, y = 1), h(\mathbf{x}) = -1)$ incurred by incorrectly predicting the label of a data point from the majority class (with true label value $y = 1$).

6.3 Model Selection

Chapter 3 illustrated how many well-known ML methods are obtained by different combinations of a hypothesis space or model, loss function and data representation. While for many ML applications there is often a natural choice for the loss function and data representation, the right choice for the model is typically less obvious. We now discuss how to use the validation methods of Section 6.2 to choose between different candidate models.

Consider data points characterized by a single numeric feature $x \in \mathbb{R}$ and numeric label $y \in \mathbb{R}$. If we suspect that the relation between feature x and label y is non-linear, we might

use polynomial regression which is discussed in Section 3.2. Polynomial regression uses the hypothesis space $\mathcal{H}^{(n)}_{\rm poly}$ with some maximum degree n. Different choices for the maximum degree n yield a different hypothesis space: $\mathcal{H}^{(1)} = \mathcal{H}^{(0)}_{\rm poly}, \mathcal{H}^{(2)} = \mathcal{H}^{(1)}_{\rm poly}, \ldots, \mathcal{H}^{(M)} = \mathcal{H}^{(M)}_{\rm poly}$.

Another ML method that learns non-linear hypothesis map is Gaussian basis regression (see Section 3.5). Here, different choices for the variance σ and shifts μ of the Gaussian basis function (3.13) result in different hypothesis spaces. For example, $\mathcal{H}^{(1)} = \mathcal{H}^{(2)}_{\rm Gauss}$ with $\sigma = 1$ and $\mu_1 = 1$ and $\mu_2 = 2$, $\mathcal{H}^{(2)} = \mathcal{H}^{(2)}_{\rm Gauss}$ with $\sigma = 1/10$, $\mu_1 = 10$, $\mu_2 = 20$.

Algorithm 7 summarizes a simple method to choose between different candidate models $\mathcal{H}^{(1)}, \mathcal{H}^{(2)}, \ldots, \mathcal{H}^{(M)}$. The idea is to first learn and validate a hypothesis $\widehat{h}^{(l)}$ separately for each model $\mathcal{H}^{(l)}$ using Algorithm 6. For each model $\mathcal{H}^{(l)}$, we learn the hypothesis $\widehat{h}^{(l)}$ via ERM (6.4) and then compute its validation error $E_v^{(l)}$ (6.6). We then choose the hypothesis $\widehat{h}^{(\hat{l})}$ from those model $\mathcal{H}^{(\hat{l})}$ which resulted in the smallest validation error $E_v^{(\hat{l})} = \min_{l=1,\ldots,M} E_v^{(l)}$.

The workflow of Algorithm 7 is similar to the workflow of ERM. Remember that the idea of ERM is to learn a hypothesis out of a set of different candidates (the hypothesis space). The quality of a particular hypothesis h is measured using the (average) loss incurred on some training set. We use the same principle for model selection but on a higher level. Instead of learning a hypothesis within a hypothesis space, we choose (or learn) a hypothesis space within a set of candidate hypothesis spaces. The quality of a given hypothesis space is measured by the validation error (6.6). To determine the validation error of a hypothesis space, we first learn the hypothesis $\widehat{h} \in \mathcal{H}$ via ERM (6.4) on the training set. Then, we obtain the validation error as the average loss of $\widehat{h}$ on the validation set.

The final hypothesis $\widehat{h}$ delivered by the model selection Algorithm 7 not only depends on the training set used in ERM (see (6.9)). This hypothesis $\widehat{h}$ has also been chosen based on its validation error which is the average loss on the validation set in (6.11). Indeed, we compared this validation error with the validation errors of other models to pick the model $\mathcal{H}^{(\hat{l})}$ (see step 10) which contains $\widehat{h}$. Since we used the validation error (6.11) of $\widehat{h}$ to learn it, we cannot use this validation error as a good indicator for the general performance of $\widehat{h}$.

To estimate the general performance of the final hypothesis $\widehat{h}$ delivered by Algorithm 7 we must try it out on a test set. The test set, which is constructed in step 3 of Algorithm 7, consists of data points that are neither contained in the training set (6.9) nor the validation set (6.11) used for training and validating the candidate models $\mathcal{H}^{(1)}, \ldots, \mathcal{H}^{(M)}$. The average loss of the final hypothesis on the test set is referred to as the test error. The test error is computed in the step 12 of Algorithm 7.

Sometimes it is beneficial to use different loss functions for the training and the validation

Algorithm 7 Model Selection

Input: list of candidate models $\mathcal{H}^{(1)}, \ldots, \mathcal{H}^{(M)}$, loss function L, dataset $\mathcal{D} = \big\{\big(\mathbf{x}^{(1)}, y^{(1)}\big), \ldots, \big(\mathbf{x}^{(m)}, y^{(m)}\big)\big\}$; number k of folds, test set fraction ρ

1: randomly shuffle the data points in $\mathcal{D}$
2: determine size $m' := \lceil \rho m \rceil$ of test set
3: construct a test set

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

4: construct a training set and a validation set,

$$\mathcal{D}^{(\text{trainval})} = \big\{\big(\mathbf{x}^{(m'+1)}, y^{(m'+1)}\big), \ldots, \big(\mathbf{x}^{(m)}, y^{(m)}\big)\big\}$$

5: **for** model index $l = 1, \ldots, M$ **do**
6: run Algorithm 6 using $\mathcal{H} = \mathcal{H}^{(l)}$, dataset $\mathcal{D} = \mathcal{D}^{(\text{trainval})}$, loss function L and k folds
7: Algorithm 6 delivers hypothesis $\widehat{h}$ and validation error E_v
8: store learnt hypothesis $\widehat{h}^{(l)} := \widehat{h}$ and validation error $E_v^{(l)} := E_v$
9: **end for**
10: pick model $\mathcal{H}^{(\hat{l})}$ with minimum validation error $E_v^{(\hat{l})} = \min_{l=1,\ldots,M} E_v^{(l)}$
11: define optimal hypothesis $\widehat{h} = \widehat{h}^{(\hat{l})}$
12: compute **test error**

$$E^{(\text{test})} := \widehat{L}\big(\widehat{h} | \mathcal{D}^{(\text{test})}\big) = (1/\big|\mathcal{D}^{(\text{test})}\big|) \sum_{i \in \mathcal{D}^{(\text{test})}} L\left((\mathbf{x}^{(i)}, y^{(i)}), \widehat{h}\right). \tag{6.12}$$

Output: hypothesis $\widehat{h}$; training error $E_t^{(\hat{l})}$; validation error $E_v^{(\hat{l})}$, test error $E^{(\text{test})}$.

of a hypothesis. As an example, consider logistic regression and the SVM which have been discussed in Sections 3.6 and 3.7, respectively. Both methods use the same model which is the space of linear hypothesis maps $h(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$. The main difference between these two methods is in their choice for the loss function. Logistic regression minimizes the (average) logistic loss (2.12) on the training set to learn the hypothesis $h^{(1)}(\mathbf{x}) = \left(\mathbf{w}^{(1)}\right)^T\mathbf{x}$ with a parameter vector $\mathbf{w}^{(1)}$. The SVM instead minimizes the (average) hinge loss (2.11) on the training set to learn the hypothesis $h^{(2)}(\mathbf{x}) = \left(\mathbf{w}^{(2)}\right)^T\mathbf{x}$ with a parameter vector $\mathbf{w}^{(2)}$. It is inconvenient to compare the usefulness of the two hypotheses $h^{(1)}(\mathbf{x})$ and $h^{(2)}(\mathbf{x})$ using different loss functions to compute their validation errors. This comparison is more convenient if we instead compute the validation errors for $h^{(1)}(\mathbf{x})$ and $h^{(2)}(\mathbf{x})$ using the average 0/1 loss (2.9).

Algorithm 7 requires as one of its inputs a given list of candidate models. The longer this list, the more computation is required from Algorithm 7. Sometimes it is possible to prune the list of candidate models by removing models that are very unlikely to have minimum validation error.

Consider polynomial regression which uses as the model the space $\mathcal{H}^{(r)}_{\rm poly}$ of polynomials with maximum degree r (see (3.4)). For $r = 1$, $\mathcal{H}^{(r)}_{\rm poly}$ is the space of polynomials with maximum degree one (which are linear maps), $h(x) = w_2x + w_1$. For $r = 2$, $\mathcal{H}^{(r)}_{\rm poly}$ is the space of polynomials with maximum degree two, $h(x) = w_3x^2 + w_2x + w_1$.

The polynomial degree r parametrizes a nested set of models,

$$\mathcal{H}^{(r)}_{\rm poly} \subset \mathcal{H}^{(r)}_{\rm poly} \subset \ldots.$$

For each degree r, we learn a hypothesis $h^{(r)} \in \mathcal{H}^{(r)}_{\rm poly}$ with minimum average loss (training error) $E^{(r)}_t$ on a training set (see (6.5)). To validate the learnt hypothesis $h^{(r)}$, we compute its average loss (validation error) $E^{(r)}_v$ on a validation set (see (6.6)).

Figure 6.5 depicts the typical dependency of the training and validation errors on the polynomial degree r. The training error $E^{(r)}_t$ decreases monotonically with increasing polynomial degree r. To illustrate this monotonic decrease, we consider the two specific choices $r = 3$ and $r = 5$ with corresponding models $\mathcal{H}^{(r)}_{\rm poly}$ and $\mathcal{H}^{(r)}_{\rm poly}$. Note that $\mathcal{H}^{(3)}_{\rm poly} \subset \mathcal{H}^{(5)}_{\rm poly}$ since any polynomial with degree not exceeding 3 is also a polynomial with degree not exceeding 5. Therefore, the training error (6.5) obtained when minimizing over the larger model $\mathcal{H}^{(5)}_{\rm poly}$ can only decrease but never increase compared to (6.5) using the smaller model $\mathcal{H}^{(3)}_{\rm poly}$

Figure 6.5 indicates that the validation error $E^{(r)}_v$ (see (6.6)) behaves different compared to the training error $E^{(r)}_t$. Starting with degree $r = 0$, the validation error first decreases with increasing degree r. As soon as the degree r is increased beyond a critical value, the

validation error starts to increase with increasing r. For very large values of r, the training error becomes almost negligible while the validation error becomes very large. In this regime, polynomial regression overfits the training set.

Figure 6.6 illustrates the overfitting of polynomial regression when using a maximum degree that is too large. In particular, Figure 6.6 depicts a learnt hypothesis which is a degree 9 polynomial that fits quite well the training set, resulting in a small training error. To achieve such a low training error, requires the learnt polynomial to have an unreasonable high rate of change for feature values $x \approx 0$. This results in large prediction errors for data points with feature values $x \approx 0$.

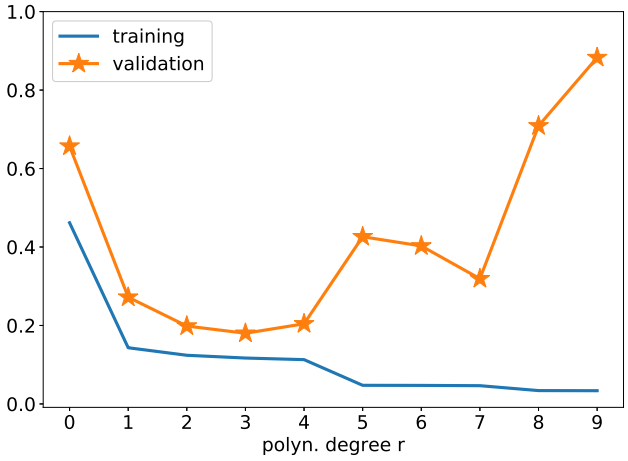

Figure 6.5: The training error and validation error obtained from polynomial regression using different values r for the maximum polynomial degree.

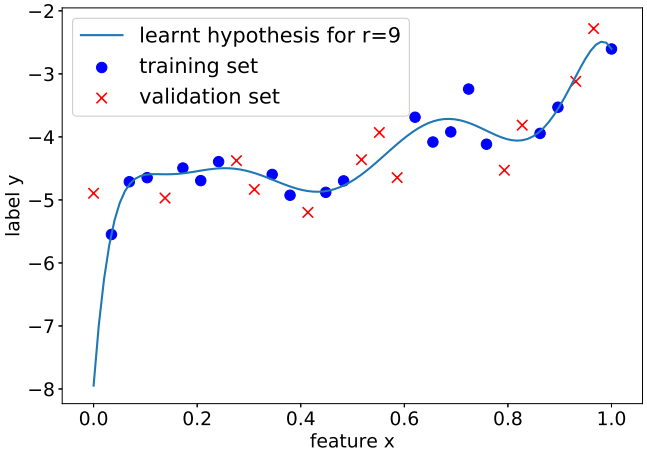

Figure 6.6: A hypothesis $\widehat{h}$ which is a polynomial with degree not larger than $r = 9$. The hypothesis has been learnt by minimizing the average loss on the training set. Note the rapid change of $\widehat{h}$ for feature values $x \approx 0$.

6.4 A Probabilistic Analysis of Generalization

More Data Beats Clever Algorithms ?; More Data Beats Clever Feature Selection?

A key challenge in ML is to ensure that a hypothesis that predicts well the labels on a training set (which has been used to learn that hypothesis) will also predict well the labels of data points outside the training set. We say that a ML method generalizes well if it learns a hypothesis $\widehat{h}$ that performs not significantly worse on data points outside the training set. In other words, the loss incurred by $\widehat{h}$ for data points outside the training set is not much larger than the average loss of $\widehat{h}$ incurred on the training set.

We now study the generalization of linear regression methods (see Section 3.1) using an i.i.d. assumption. In particular, we interpret data points as i.i.d. realizations of RVs that have the same distribution as a random data point $\mathbf{z} = (\mathbf{x}, y)$. The feature vector $\mathbf{x}$ is then a realization of a standard Gaussian RV with zero mean and covariance being the identity matrix, i.e., $\mathbf{x} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$.

The label y of a random data point is related to its features $\mathbf{x}$ via a linear Gaussian model

$$y = \overline{\mathbf{w}}^{T}\mathbf{x} + \varepsilon, \text{ with noise } \varepsilon \sim \mathcal{N}(0, \sigma^2). \tag{6.13}$$

We assume the noise variance σ^2 fixed and known. This is a simplifying assumption and in

practice we would need to estimate the noise variance from data [24]. Note that, within our probabilistic model, the error component ε in (6.13) is intrinsic to the data and cannot be overcome by any ML method. We highlight that the probabilistic model for the observed data points is just a modelling assumption. This assumption allows us to study some fundamental behaviour of ML methods. There are principled methods ("statistical tests") that allow to determine if a given dataset can be accurately modelled using (6.13) [62].

We predict the label y from the features $\mathbf{x}$ using a linear hypothesis $h(\mathbf{x})$ that depends only on the first l features $x_1, \ldots, x_l$. Thus, we use the hypothesis space

$$\mathcal{H}^{(l)} = \{h^{(\mathbf{w})}(\mathbf{x}) = (\mathbf{w}^T, \mathbf{0}^T)\mathbf{x} \text{ with } \mathbf{w} \in \mathbb{R}^l\}. \tag{6.14}$$

Note that each element $h^{(\mathbf{w})} \in \mathcal{H}^{(l)}$ corresponds to a particular choice of the parameter vector $\mathbf{w} \in \mathbb{R}^l$.

The model parameter $l \in \{0, \ldots, n\}$ coincides with the effective dimension of the hypothesis space $\mathcal{H}^{(l)}$. For $l < n$, the hypothesis space $\mathcal{H}^{(l)}$ is a proper (strict) subset of the space of linear hypothesis maps (2.4) used within linear regression (see Section 3.1). Moreover, the parameter l indexes a nested sequence of models,

$$\mathcal{H}^{(0)} \subseteq \mathcal{H}^{(1)} \subseteq \ldots \subseteq \mathcal{H}^{(n)}.$$

The quality of a particular predictor $h^{(\mathbf{w})} \in \mathcal{H}^{(l)}$ is measured via the average squared error $\widehat{L}(h^{(\mathbf{w})} \mid \mathcal{D}^{(\text{train})})$ incurred on the labeled training set

$$\mathcal{D}^{(\text{train})} = \{\big(\mathbf{x}^{(1)}, y^{(1)}\big), \ldots, \big(\mathbf{x}^{(m_t)}, y^{(m_t)}\big)\}. \tag{6.15}$$

We interpret data points in the training set $\mathcal{D}^{(\text{train})}$ as well as any other data point outside the training set as realizations of i.i.d. RVs with a common probability distribution. This common probability distribution is a multivariate normal (Gaussian) distribution,

$$\mathbf{x}, \mathbf{x}^{(i)} \text{ i.i.d. with } \mathbf{x}, \mathbf{x}^{(i)} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}). \tag{6.16}$$

The labels $y^{(i)}, y$ are related to the features of data points via (see (6.13))

$$y^{(i)} = \overline{\mathbf{w}}^T \mathbf{x}^{(i)} + \varepsilon^{(i)}, \text{ and } y = \overline{\mathbf{w}}^T \mathbf{x} + \varepsilon. \tag{6.17}$$

Here, the noise terms $\varepsilon, \varepsilon^{(i)} \sim \mathcal{N}(0, \sigma^2)$ are realizations of i.i.d. Gaussian RVs with zero mean and variance σ^2.

Chapter 4 showed that the training error $\widehat{L}(h^{(\mathbf{w})} \mid \mathcal{D}^{(\text{train})})$ is minimized by the predictor $h^{(\widehat{\mathbf{w}})}(\mathbf{x}) = \widehat{\mathbf{w}}^T \mathbf{I}_{l\times n}\mathbf{x}$, that uses the parameter vector

$$\widehat{\mathbf{w}} = \big(\big(\mathbf{X}^{(l)}\big)^T \mathbf{X}^{(l)}\big)^{-1}\big(\mathbf{X}^{(l)}\big)^T \mathbf{y}. \tag{6.18}$$

Here we used the (restricted) feature matrix $\mathbf{X}^{(l)}$ and the label vector $\mathbf{y}$ defined as, respectively,

$$\begin{aligned} \mathbf{X}^{(l)} &= (\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m_{\rm t})})^T \mathbf{I}_{n\times l} \in \mathbb{R}^{m_{\rm t}\times l}, \text{ and} \\ \mathbf{y} &= \big(y^{(1)}, \ldots, y^{(m_{\rm t})}\big)^T \in \mathbb{R}^{m_{\rm t}}. \end{aligned} \tag{6.19}$$

It will be convenient to tolerate a slight abuse of notation and denote both, the length-l vector (6.18) as well as the zero-padded parameter vector $\big(\widehat{\mathbf{w}}^T, \mathbf{0}^T\big)^T \in \mathbb{R}^n$, by $\widehat{\mathbf{w}}$. This allows us to write

$$h^{(\widehat{\mathbf{w}})}(\mathbf{x}) = \widehat{\mathbf{w}}^T \mathbf{x}. \tag{6.20}$$

We highlight that the formula (6.18) for the optimal weight vector $\widehat{\mathbf{w}}$ is only valid if the matrix $\big(\mathbf{X}^{(l)}\big)^T \mathbf{X}^{(l)}$ is invertible. Within our toy model (see (6.16)), this is true with probability one whenever $m_{\rm t} \geq l$. Indeed, for $m_{\rm t} \geq l$ the truncated feature vectors $\mathbf{I}_{l\times n}\mathbf{x}^{(1)}, \ldots, \mathbf{I}_{l\times n}\mathbf{x}^{(m_t)}$, which are i.i.d. realizations of a Gaussian RV, are linearly independent with probability one [9, 41].

In what follows, we consider the case $m_{\rm t} > l$ such that the formula (6.18) is valid (with probability one). The more challenging high-dimensional regime $m_{\rm t} \leq l$ will be studied in Chapter 7.

The optimal parameter vector $\widehat{\mathbf{w}}$ (see (6.18)) depends on the training set $\mathcal{D}^{(\text{train})}$ via the feature matrix $\mathbf{X}^{(l)}$ and label vector $\mathbf{y}$ (see (6.19)). Therefore, since we model the data points in the training set as realizations of RVs, the parameter vector $\widehat{\mathbf{w}}$ (6.18) is the realization of a RV. For each specific realization of the training set $\mathcal{D}^{(\text{train})}$, we obtain a specific realization of the optimal parameter vector $\widehat{\mathbf{w}}$.

The probabilistic model (6.13) relates the features $\mathbf{x}$ of a data point to its label y via some (unknown) true parameter vector $\overline{\mathbf{w}}$. Intuitively, the best linear hypothesis would be $h(\mathbf{x}) = \widehat{\mathbf{w}}^T\mathbf{x}$ with parameter vector $\widehat{\mathbf{w}} = \overline{\mathbf{w}}$. However, in general this will not be achievable since we have to compute $\widehat{\mathbf{w}}$ based on the features $\mathbf{x}^{(i)}$ and noisy labels $y^{(i)}$ of the data points in the training set $\mathcal{D}$.

The parameter vector $\widehat{\mathbf{w}}$ delivered by ERM (4.5) typically results in a non-zero estimation

error

$$\Delta \mathbf{w} := \widehat{\mathbf{w}} - \overline{\mathbf{w}}. \tag{6.21}$$

The estimation error (6.21) is the realization of a RV since the learnt parameter vector $\widehat{\mathbf{w}}$ (see (6.18)) is itself a realization of a RV.

The Bias and Variance Decomposition. The prediction accuracy of $h^{(\widehat{\mathbf{w}})}$, using the learnt parameter vector (6.18), depends crucially on the mean squared estimation error (MSEE)

$$E_{\rm est} := \mathbb{E}\{\|\Delta \mathbf{w}\|_2^2\} \stackrel{(6.21)}{=} \mathbb{E}\big\{ \|\widehat{\mathbf{w}} - \overline{\mathbf{w}}\|_2^2 \big\}. \tag{6.22}$$

We will next decompose the MSEE $E_{\rm est}$ into two components, which are referred to as a variance term and a bias term. The variance term quantifies the fluctuation of the parameter vector obtained from ERM on the training set (6.15). The bias term characterizes the systematic (or average) deviation between the true parameter vector $\overline{\mathbf{w}}$ (see (6.13)) and the expectation of the learnt parameter vector $\widehat{\mathbf{w}}$.

Let us start with rewriting (6.22) using elementary manipulations as

$$\begin{aligned} E_{\rm est} &\stackrel{(6.22)}{=} \mathbb{E}\big\{ \|\widehat{\mathbf{w}} - \overline{\mathbf{w}}\|_2^2 \big\}\big\} \\ &= \mathbb{E}\Big\{ \big\| \big(\widehat{\mathbf{w}} - \mathbb{E}\{\widehat{\mathbf{w}}\}\big) - \big(\overline{\mathbf{w}} - \mathbb{E}\{\widehat{\mathbf{w}}\}\big) \big\|_2^2 \Big\}. \end{aligned}$$

We can develop the last expression further by expanding the squared Euclidean norm,

$$\begin{aligned} E_{\rm est} &= \mathbb{E}\big\{ \|\widehat{\mathbf{w}} - \mathbb{E}\{\widehat{\mathbf{w}}\}\|_2^2 \big\} - 2\mathbb{E}\big\{ \big(\widehat{\mathbf{w}} - \mathbb{E}\{\widehat{\mathbf{w}}\}\big)^T \big(\overline{\mathbf{w}} - \mathbb{E}\{\widehat{\mathbf{w}}\}\big)\big\} + \mathbb{E}\big\{ \big\|\overline{\mathbf{w}} - \mathbb{E}\{\widehat{\mathbf{w}}\}\big\|_2^2 \big\} \\ &= \mathbb{E}\big\{ \|\widehat{\mathbf{w}} - \mathbb{E}\{\widehat{\mathbf{w}}\}\|_2^2 \big\} - 2\big(\underbrace{\mathbb{E}\{\widehat{\mathbf{w}}\} - \mathbb{E}\{\widehat{\mathbf{w}}\}}_{=\mathbf{0}}\big)^T \big(\overline{\mathbf{w}} - \mathbb{E}\{\widehat{\mathbf{w}}\}\big)\big\} + \mathbb{E}\big\{ \big\|\overline{\mathbf{w}} - \mathbb{E}\{\widehat{\mathbf{w}}\}\big\|_2^2 \big\} \\ &= \underbrace{\mathbb{E}\big\{ \|\widehat{\mathbf{w}} - \mathbb{E}\{\widehat{\mathbf{w}}\}\|_2^2 \big\}}_{\text{variance } V} + \underbrace{\mathbb{E}\big\{ \|\overline{\mathbf{w}} - \mathbb{E}\{\widehat{\mathbf{w}}\}\|_2^2 \big\}}_{\text{bias } B^2}. \end{aligned} \tag{6.23}$$

The first component in (6.23) represents the (expected) variance of the learnt parameter vector $\widehat{\mathbf{w}}$ (6.18). Note that, within our probabilistic model, the training set (6.15) is the realization of a RV since it is constituted by data points that are i.i.d. realizations of RVs (see (6.16) and (6.13)).

The second component in (6.23) is referred to as a bias term. The parameter vector $\widehat{\mathbf{w}}$ is computed from a randomly fluctuating training set via (6.18) and is therefore itself

fluctuating around its expectation $\mathbb{E}\{\widehat{\mathbf{w}}\}$. The bias term is the Euclidean distance between this expectation $\mathbb{E}\{\widehat{\mathbf{w}}\}$ and the true parameter vector $\overline{\mathbf{w}}$ relating features and label of a data point via (6.13).

The bias term B^2 and the variance V in (6.23) both depend on the model complexity parameter l but in a fundamentally different manner. The bias term B^2 typically decreases with increasing l while the variance V increases with increasing l. In particular, the bias term is given as

$$B^2 = \|\overline{\mathbf{w}} - \mathbb{E}\{\widehat{\mathbf{w}}\}\|_2^2 = \sum_{j=l+1}^{n} \overline{w}_j^2, \tag{6.24}$$

The bias term (6.24) is zero if and only if

$$\overline{w}_j = 0 \text{ for any index } j = l+1, \ldots, n. \tag{6.25}$$

The necessary and sufficient condition (6.25) for zero bias is equivalent to $h^{(\overline{\mathbf{w}})} \in \mathcal{H}^{(l)}$. Note that the condition (6.25) depends on both, the model parameter l and the true parameter vector $\overline{\mathbf{w}}$. While the model parameter l is under control, the true parameter vector $\overline{\mathbf{w}}$ is not under our control but determined by the underlying data generation process. The only way to ensure (6.25) for every possible parameter vector $\overline{\mathbf{w}}$ in (6.13) is to use $l = n$, i.e., to use all available features $x_1, \ldots, x_n$ of a data point.

When using the model $\mathcal{H}^{(l)}$ with $l < n$, we cannot guarantee a zero bias term since we have no control over the true underlying parameter vector $\overline{\mathbf{w}}$ in (6.13). In general, the bias term decreases with an increasing model size l (see Figure 6.7). We highlight that the bias term does not depend on the variance σ^2 of the noise ε in our toy model (6.13).

Let us now consider the variance term in (6.23). Using the statistical independence of the features and labels of data points (see (6.13), (6.16) and (6.17)), one can show that[1]

$$V = \mathbb{E}\{\, \|\widehat{\mathbf{w}} - \mathbb{E}\{\widehat{\mathbf{w}}\}\|_2^2 \,\} = \big(B^2 + \sigma^2\big) \mathrm{tr}\left\{ \mathbb{E}\left\{ \left(\big(\mathbf{X}^{(l)}\big)^T \mathbf{X}^{(l)} \right)^{-1} \right\} \right\}. \tag{6.26}$$

By (6.16), the matrix $\left(\big(\mathbf{X}^{(l)}\big)^T \mathbf{X}^{(l)} \right)^{-1}$ is a realization of a (matrix-valued) RV with an inverse Wishart distribution [91]. For $m_{\mathrm{t}} > l+1$, its expectation is given as

$$\mathbb{E}\{ \left(\big(\mathbf{X}^{(l)}\big)^T \mathbf{X}^{(l)} \right)^{-1} \} = 1/(m_{\mathrm{t}} - l - 1)\mathbf{I}. \tag{6.27}$$

[1]This derivation is not very difficult but rather lengthy. For more details about the derivation of (6.26) we refer to the literature [9, 88].

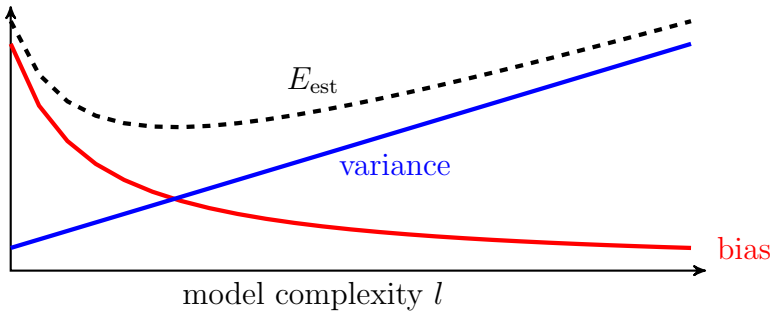

Figure 6.7: The MSEE E_{est} incurred by linear regression can be decomposed into a bias term B^2 and a variance term V (see (6.23)). These two components depend on the model complexity l in an opposite manner which results in a bias-variance trade-off.

By inserting (6.27) and $\text{tr}\{\mathbf{I}\} = l$ into (6.26),

$$V = \mathbb{E}\left\{\|\widehat{\mathbf{w}} - \mathbb{E}\left\{\widehat{\mathbf{w}}\right\}\|_2^2\right\} = \left(B^2 + \sigma^2\right) l/(m_{\text{t}} - l - 1). \tag{6.28}$$

The variance (6.28) typically increases with increasing model complexity l (see Figure 6.7). In contrast, the bias term (6.24) decreases with increasing l.

The opposite dependence of variance and bias on the model complexity results in a bias-variance trade-off. Choosing a model (hypothesis space) with small bias will typically result in large variance and vice versa. In general, the choice of model must balance between a small variance and a small bias.

Generalization. Consider a linear regression method that learns the linear hypothesis $h(\mathbf{x}) = \widehat{\mathbf{w}}^T\mathbf{x}$ using the parameter vector (6.18). The parameter vector $\widehat{\mathbf{w}}^T$ (6.18) results in a linear hypothesis with minimum training error, i.e., minimum average loss on the training set. However, the ultimate goal of ML is to find a hypothesis that predicts well the label of any data point. In particular, we want the hypothesis $h(\mathbf{x}) = \widehat{\mathbf{w}}^T\mathbf{x}$ to generalize well to data points outside the training set.

We quantify the generalization capability of $h(\mathbf{x}) = \widehat{\mathbf{w}}^T\mathbf{x}$ by its expected prediction loss

$$E_{\text{pred}} = \mathbb{E}\Big\{\big(y - \underbrace{\widehat{\mathbf{w}}^T\mathbf{x}}_{=\hat{y}}\big)^2\Big\}. \tag{6.29}$$

Note that E_{pred} is a measure for the performance of a ML method and not of a specific hypothesis. Indeed, the learnt parameter vector $\widehat{\mathbf{w}}$ is not fixed but depends on the data points in the training set. These data points are modelled as realizations of i.i.d. RVs

and, in turn, the learnt parameter vector $\widehat{\mathbf{w}}$ becomes a realization of a RV. Thus, in some sense, the expected prediction loss (6.29) characterizes the overall ML method that reads in a training set and delivers (learn) a linear hypothesis with parameter vector $\widehat{\mathbf{w}}$ (6.18). In contrast, the risk (4.1) introduced in Chapter 4 characterizes the performance of a specific (fixed) hypothesis h without taking into account a learning process that delivered h based on data.

Let us now relate the expected prediction loss (6.29) of the linear hypothesis $h(\mathbf{x}) = \widehat{\mathbf{w}}^T\mathbf{x}$ to the bias and variance of (6.18),

$$\begin{aligned} E_{\rm pred} &\overset{(6.13)}{=} \mathbb{E}\{\Delta\mathbf{w}^T\mathbf{x}\mathbf{x}^T\Delta\mathbf{w}\} + \sigma^2 \\ &\overset{(a)}{=} \mathbb{E}\{\mathbb{E}\{\Delta\mathbf{w}^T\mathbf{x}\mathbf{x}^T\Delta\mathbf{w} \mid \mathcal{D}^{(\rm train)}\}\} + \sigma^2 \\ &\overset{(b)}{=} \mathbb{E}\{\Delta\mathbf{w}^T\Delta\mathbf{w}\} + \sigma^2 \\ &\overset{(6.21),(6.22)}{=} E_{\rm est} + \sigma^2 \\ &\overset{(6.23)}{=} B^2 + V + \sigma^2. \end{aligned} \tag{6.30}$$

Here, step (a) uses the law of iterated expectation (see, e.g., [9]). Step (b) uses that the feature vector $\mathbf{x}$ of a "new" data point is a realization of a RV which is statistically independent of the data points in the training set $\mathcal{D}^{(\rm train)}$. We also used our assumption that $\mathbf{x}$ is the realization of a RV with zero mean and covariance matrix $\mathbb{E}\{\mathbf{x}\mathbf{x}^T\} = \mathbf{I}$ (see (6.16)).

According to (6.30), the average (expected) prediction error $E_{\rm pred}$ is the sum of three components: (i) the bias B^2, (ii) the variance V and (iii) the noise variance σ^2. Figure 6.7 illustrates the typical dependency of the bias and variance on the model (6.14), which is parametrized by the model complexity l. Note that the model complexity parameter l in (6.14) coincides with the effective model dimension $d_{\rm eff}\left(\mathcal{H}^{(l)}\right)$ (see Section 2.2).

The bias and variance, whose sum is the estimation error $E_{\rm est}$, can be influenced by varying the model complexity l which is a design parameter. The noise variance σ^2 is the intrinsic accuracy limit of our toy model (6.13) and is not under the control of the ML engineer. It is impossible for any ML method (no matter how computationally expensive) to achieve, on average, a prediction error smaller than the noise variance σ^2. Carefully note that this statement only applies if the data points arising in a ML application can be (reasonably well) modelled as realizations of i.i.d. RVs.

We highlight that our statistical analysis, resulting the formulas for bias (6.24), variance (6.28) and the average prediction error (6.30), applies only if the observed data points can

be well modelled using the probabilistic model specified by (6.13), (6.16) and (6.17). The validity of this probabilistic model can to be verified by principled statistical model validation techniques [159, 146]. Section 6.5 discusses a fundamentally different approach to analyzing the statistical properties of a ML method. Instead of a probabilistic model, this approach uses random sampling techniques to synthesize i.i.d. copies of given (small) data points. We can approximate the expectation of some relevant quantity, such as the loss $L\big(\big(\mathbf{x},y\big),h\big)$, using an average over synthetic data [58].

The qualitative behaviour of estimation error in Figure 6.7 depends on the definition for the model complexity. Our concept of effective dimension (see Section 2.2) coincides with most other notions of model complexity for the linear hypothesis space (6.14). However, for more complicated models such as deep nets it is often not obvious how effective dimension is related to more tangible quantities such as the total number of tunable weights or the number of artificial neurons. Indeed, the effective dimension might also depend on the specific learning algorithm such as SGD. Therefore, for deep nets, if we would plot estimation error against the number of tunable weights we might observe a behaviour of estimation error fundamentally different from the shape in Figure 6.7. One example for such un-intuitive behaviour is known as "double descent phenomena" [8].

6.5 The Bootstrap

basic idea of bootstrap: use histogram of dataset as the underlying probability distribution; generate new data points by random sampling (with replacement) from that distribution.

Consider learning a hypothesis $\widehat{h} \in \mathcal{H}$ by minimizing the average loss incurred on a dataset $\mathcal{D} = \{\big(\mathbf{x}^{(1)},y^{(1)}\big),\ldots,\big(\mathbf{x}^{(m)},y^{(m)}\big)\}$. The data points $\big(\mathbf{x}^{(i)},y^{(i)}\big)$ are modelled as realizations of i.i.d. RVs. Let use denote the (common) probability distribution of these RVs by $p(\mathbf{x},y)$.

If we interpret the data points $\big(\mathbf{x}^{(i)},y^{(i)}\big)$ as realizations of RVs, also the learnt hypothesis $\widehat{h}$ is a realization of a RV. Indeed, the hypothesis $\widehat{h}$ is obtained by solving an optimization problem (4.3) that involves realizations of RVs. The bootstrap is a method for estimating (parameters of) the probability distribution $p(\widehat{h})$ [58].

Section 6.4 used a probabilistic model for data points to derive (the parameters of) the probability distribution $p(\widehat{h})$. Note that the analysis in Section 6.4 only applies to the specific probabilistic model (6.16), (6.17). In contrast, the bootstrap can be used for data points drawn from an arbitrary probability distribution.

The core idea behind the bootstrap is to use the histogram $\hat{p}(\mathbf{z})$ of the data points in $\mathcal{D}$ to generate B new datasets $\mathcal{D}^{(1)}, \ldots, \mathcal{D}^{(B)}$. Each dataset is constructed such that is has the same size as the original dataset $\mathcal{D}$. For each dataset $\mathcal{D}^{(b)}$, we solve a separate ERM (4.3) to obtain the hypothesis $\widehat{h}^{(b)}$. The hypothesis $\widehat{h}^{(b)}$ is a realization of a RV whose distribution is determined by the histogram $\hat{p}(\mathbf{z})$ as well as the hypothesis space and the loss function used in the ERM (4.3).

6.6 Diagnosing ML

diagnose ML methods by comparing training error with validation error and (if available) some baseline; baseline can be obtained via the Bayes risk when using a probabilistic model (such as the i.i.d. assumption) or human performance or the performance of existing ML methods ("experts" in regret framework)

In what follows, we tacitly assume that data points can (to a good approximation) be interpreted as realizations of i.i.d. RVs (see Section 2.1.4). This "i.i.d. assumption" underlies ERM (4.3) as the guiding principle for learning a hypothesis with small risk (4.1). This assumption also motivates to use the average loss (6.6) on a validation set as an estimate for the risk. More fundamentally, we need the i.i.d. assumption to define the concept of risk as a measure for how well a hypothesis predicts the labels of arbitrary data points.

Consider a ML method which uses Algorithm 5 (or Algorithm 6) to learn and validate the hypothesis $\widehat{h} \in \mathcal{H}$. Besides the learnt hypothesis $\widehat{h}$, these algorithms also deliver the training error E_t and the validation error E_v. As we will see shortly, we can diagnose ML methods to some extent just by comparing training with validation errors. This diagnosis is further enabled if we know a baseline $E^{(\mathrm{ref})}$.

One important source for a baseline $E^{(\mathrm{ref})}$ are probabilistic models for the data points (see Section 6.4). Given a probabilistic model, which specifies the probability distribution $p(\mathbf{x}, y)$ of the features and label of data points, we can compute the minimum achievable risk (4.1). Indeed, the minimum achievable risk is precisely the expected loss of the Bayes estimator $\widehat{h}(\mathbf{x})$ of the label y, given the features $\mathbf{x}$ of a data point. The Bayes estimator $\widehat{h}(\mathbf{x})$ is fully determined by the probability distribution $p(\mathbf{x}, y)$ of the features and label of a (random) data point [85, Chapter 4]. When using the squared error loss (2.8) to define the risk (4.1), the Bayes estimator is given by the posterior mean $\widehat{h}(\mathbf{x}) = \mathbb{E}\{y|\mathbf{x}\}$.

A further potential source for a baseline $E^{(\mathrm{ref})}$ is an existing, but for some reason unsuitable, ML method. This existing ML method might be computationally too expensive to be

used for the ML application at end. However, we might still use its statistical properties as a benchmark. The

We might also use the performance of human experts as a baseline. If we want to develop a ML method that detects certain type of skin cancers from images of the skin, a benchmark might be the current classification accuracy achieved by experienced dermatologists [36].

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

- $E_t \approx E_v \approx E^{(\text{ref})}$: The training error is on the same level as the validation error and the benchmark error. There is not much to improve here since the validation error is already on the desired error level. Moreover, the training error is not much smaller than the validation error which indicates that there is no overfitting. It seems we have obtained a ML method that achieves the benchmark error level.

- $E_v \gg E_t$: The validation error is significantly larger than the training error. It seems that the ERM (4.3) results in a hypothesis $\widehat{h}$ that overfits the training set. The loss incurred by $\widehat{h}$ on data points outside the training set, such as those in the validation set, is significantly worse. This is an indicator for overfitting which can be addressed either by reducing the effective dimension of the hypothesis space or by increasing the size of the training set. To reduce the effective dimension of the hypothesis space we have different options depending on the used model. We might use a small number of features in a linear model (3.1), a smaller maximum depth of decision trees (Section 3.10) or a fewer layers in an ANN (Section 3.11). One very elegant means for reducing the effective dimension of a hypothesis space is by limiting the number of GD steps used in gradient-based methods. This optimization based shrinking of a hypothesis space is referred to as early stopping. More generally, we can reduce the effective dimension of a hypothesis space via regularization techniques (see Chapter 7).

- $E_t \approx E_v \gg E^{(\text{ref})}$: The training error is on the same level as the validation error and both are significantly larger than the baseline. Since the training error is not much smaller than the validation error, the learnt hypothesis seems to not overfit the training set. However, the training error achieved by the learnt hypothesis is significantly larger than the benchmark error level. There can be several reasons for this to happen. First, it might be that the hypothesis space used by the ML method is too small, i.e., it does not include a hypothesis that provides a good approximation for the relation between features and label of a data point. The remedy for this situation is to use a

larger hypothesis space, e.g., by including more features in a linear model, using higher polynomial degrees in polynomial regression, using deeper decision trees or having larger ANNs (deep nets). Another reason for the training error being too large is that the optimization algorithm used to solve ERM (4.3) is not working properly.

When using gradient-based methods (see Section 5.4) to solve ERM, one reason for $E_t \gg E^{(\text{ref})}$ could be that the learning rate α in the GD step (5.6) is chosen too small or too large (see Figure 5.3-(b)). This can be solved by adjusting the learning rate by trying out several different values and using the one resulting in the smallest training error. Another option is derive optimal values for the learning rate based on a probabilistic model for how the data points are generated. One example for such a probabilistic model is the i.i.d. assumption that has been used in Section 6.4 to analyze linear regression methods.

- $E_t \gg E_v$: The training error is significantly larger than the validation error (see Exercise 6.2). The idea of ERM (4.3) is to approximate the risk (4.1) of a hypothesis by its average loss on a training set $\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^{m}$. The mathematical underpinning for this approximation is the law of large numbers which characterizes the average of (realizations of) i.i.d. RVs. The quality and usefulness of this approximation depends on the validity of two conditions. First, the data points used for computing the average loss should be such that they would be typically obtained as realizations of i.i.d. RVs with a common probability distribution. Second, the number of data points used for computing the average loss must be sufficiently large.

  Whenever the data points behave different than the the realizations of i.i.d. RVs or if the size of the training set or validation set is too small, the interpretation (and comparison) of the training error and the validation error of a learnt hypothesis becomes more difficult. As an extreme case, it might then be that the validation error consists of data points for which every hypothesis incurs small average loss. Here, we might try to increase the size of the validation set by collecting more labeled data points or by using data augmentation (see Section 7.3). If the size of training set and validation set are large but we still obtain $E_t \gg E_v$, one should verify if data points in these sets conform to the i.i.d. assumption. There are principled statistical test for the validity of the i.i.d. assumption for a given dataset (see [88] and references therein).

6.7 Exercises

Exercise 6.1. Validation Set Size. Consider a linear regression problem with data points (x, y) characterized by a scalar feature x and a numeric label y. Assume data points are realizations of i.i.d. RVs whose common probability distribution is multivariate normal with zero-mean and covariance matrix $\mathbf{C} = \begin{pmatrix} \sigma_x^2 & \sigma_{x,y} \\ \sigma_{x,y} & \sigma_y^2 \end{pmatrix}$. The entries of this covariance matrix are the variance σ_x^2 of the (zero-mean) feature, the variance σ_x^2 of the (zero-mean) label and the covariance between feature and label of a random data point. How many data points do we need to include in a validation set such that with probability of at least 0.8 the validation error of a given hypothesis h does not deviate by more than 20 percent from its expected loss?

Exercise 6.2. Validation Error Smaller Than Training Error? Linear regression learns a linear hypothesis map $\widehat{h}$ having minimal average squared error on a training set. The learnt hypothesis $\widehat{h}$ is then validated on a validation set which is different from the training set. Can you construct a training set and validation set such that the validation error of $\widehat{h}$ is strictly smaller than the training error of $\widehat{h}$?

Exercise 6.3. When is Validation Set Too Small? The usefulness of the validation error as an indicator for the performance of a hypothesis depends on the size of the validation set. Experiment with different ML methods and datasets to find out the minimum required size for the validation set.

Exercise 6.4. Too many Features? Consider data points that are characterized by $n = 1000$ numeric features $x_1, \ldots, x_n \in \mathbb{R}$ and a numeric label $y \in \mathbb{R}$. We want to learn a linear hypothesis map $h(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$ for predicting the label of a data point based on its features. Could it be beneficial to constrain the learnt hypothesis by requiring it to only depend on the first 5 features of a data point?

Exercise 6.5. Benchmark via Probability Theory. Consider data points that are characterized with single numeric feature x and label y. We model the feature and label of a data point as i.i.d. realizations of a Gaussian random vector $\mathbf{z} \sim \mathcal{N}(0, \mathbf{C})$ with zero mean and covariance matrix $\mathbf{C}$. The optimal hypothesis $\widehat{h}(x)$ to predict the label y given the feature x is the conditional expectation of the (unobserved) label y given the (observed)

feature x. How is the expected squared error loss of this optimal hypothesis (which is the Bayes estimator) related to the covariance matrix $\mathbf{C}$ of the Gaussian random vector $\mathbf{z}$.

Chapter 7

Regularization

Keywords: Data Augmentation. Robustness. Semi-Supervised Learning. Transfer Learning. Multitask Learning.

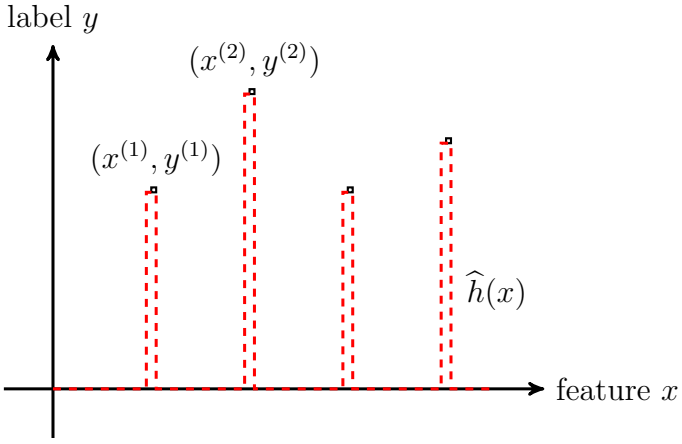

Figure 7.1: The non-linear hypothesis map $\widehat{h}$ perfectly predicts the labels of four data points in a training set and therefore has vanishing training error. Despite perfectly fitting the training set, the hypothesis $\widehat{h}$ delivers the trivial (and useless) prediction $\hat{y} = \widehat{h}(x) = 0$ for data points outside the training set. Regularization techniques help to prevent ML methods from learning such a map $\widehat{h}$.

Many ML methods use the principle of ERM (see Chapter 4) to learn a hypothesis out of a hypothesis space by minimizing the average loss (training error) on a set of labeled data points (which constitute a training set). Using ERM as a guiding principle for ML methods makes sense only if the training error is a good indicator for its loss incurred outside the training set.

Figure 7.1 illustrates a typical scenario for a modern ML method which uses a large hypothesis space. This large hypothesis space includes highly non-linear maps which can perfectly resemble any dataset of modest size. However, there might be non-linear maps for which a small training error does not guarantee accurate predictions for the labels of data points outside the training set.

Chapter 6 discussed validation techniques to verify if a hypothesis with small training error will predict also well the labels of data points outside the training set. These validation techniques, including Algorithm 5 and Algorithm 6, probe the hypothesis $\widehat{h} \in \mathcal{H}$ delivered by ERM on a validation set. The validation set consists of data points which have not been used in the training set of ERM (4.3). The validation error, which is the average loss of the hypothesis on the data points in the validation set, serves as an estimate for the average error or risk (4.1) of the hypothesis $\widehat{h}$.

This chapter discusses regularization as an alternative to validation techniques. In contrast to validation, regularization techniques do not require having a separate validation set which is not used for the ERM (4.3). This makes regularization attractive for applications where obtaining a separate validation set is difficult or costly (where labelled data is scarce).

Instead of probing a hypothesis $\widehat{h}$ on a validation set, regularization techniques estimate (or approximate) the loss increase when applying $\widehat{h}$ to data points outside the training set. The loss increase is estimated by adding a regularization term to the training error in ERM (4.3).

Section 7.1 discusses the resulting regularized ERM, which we will refer to as SRM. It turns out that the SRM is equivalent to ERM using a smaller (pruned) hypothesis space. The amount of pruning depends on the weight of the regularization term relative to the training error. For an increasing weight of the regularization term, we obtain a stronger pruning resulting in a smaller effective hypothesis space.

Section 7.2 constructs regularization terms by requiring the resulting ML method to be robust against (small) random perturbations of the data points in a training set. Here, we replace each data point of a training set by the realization of a RV that fluctuates around this data point. This construction allows to interpret regularization as a (implicit) form of data augmentation.

Section 7.3 discusses data augmentation methods as a simulation-based implementation of regularization. Data augmentation adds a certain number of perturbed copies to each data point in the training set. One way to construct perturbed copies of a data point is to add the realization of a RV to its features.

Section 7.4 analyzes the effect of regularization for linear regression using a simple probabilistic model for data points. This analysis parallels our previous study of the validation error of linear regression in Section 6.4. Similar to Section 6.4, we reveal a trade-off between the bias and variance of the hypothesis learnt by regularized linear regression. This trade- off was traced out by a discrete model parameter (the effective dimension) in Section 6.4. In contrast, regularization offers a continuous trade-off between bias and variance via a continuous regularization parameter.

Semi-supervised learning (SSL) uses (large amounts of) unlabeled data points to support the learning of a hypothesis from (a small number of) labeled data points [21]. Section 7.5 discusses SSL methods that use the statistical properties of unlabeled data points to construct useful regularization terms. These regularization terms are then used in SRM with a (typically small) set of labeled data points.

Multitask learning exploits similarities between different but similar learning tasks [19]. We can formally define a learning task by a particular choice for the loss function (see Section 2.3) . The primary role of a loss function is to score the quality of a hypothesis map. However, the loss function also encapsulates the choice for the label of a data point. For learning tasks defined for a single underlying data generation process it is reasonable to assume that the same subset of features is relevant for those learning tasks. One example for a ML application involving several similar learning tasks is multi-label classification (see Section 2.1.2). Each individual label of a data point represents a separate learning task. Section 7.6 shows how multitask learning can be implemented using regularization methods. The loss incurred in different learning tasks serves mutual regularization terms in a joint SRM for all learning tasks.

Section 7.7 shows how regularization can be used for transfer learning. Like multitask learning also transfer learning exploits relations between different but similar learning tasks. In contrast to multitask learning, which jointly solves the individual learning tasks, transfer learning solves the learning tasks sequentially. One example of transfer learning is to fine tune a pre-trained model. A pre-trained model can be obtained via ERM (4.3) in a ("source") learning task for which we have a large amount of labeled training data. The fine-tuning is then obtained via ERM (4.3) in the ("target") learning task of interest for which we might have only a small amount of labeled training data.

7.1 Structural Risk Minimization

Section 2.2 defined the effective dimension $d_{\rm eff}(\mathcal{H})$ of a hypothesis space $\mathcal{H}$ as the maximum number of data points that can be perfectly fit by some hypothesis $h \in \mathcal{H}$. As soon as the effective dimension of the hypothesis space in (4.3) exceeds the number m of training data points, we can find a hypothesis that perfectly fits the training data. However, a hypothesis that perfectly fits the training data might deliver poor predictions for data points outside the training set (see Figure 7.1).

Modern ML methods typically use a hypothesis space with large effective dimension [151, 17]. Two well-known examples for such methods is linear regression (see Section 3.1) using a large number of features and deep learning with ANNs using a large number (billions) of artificial neurons (see Section 3.11). The effective dimension of these methods can be easily on the order of billions (10^9) if not larger [126]. To avoid overfitting during the naive use of ERM (4.3) we would require a training set containing at least as many data points as the effective dimension of the hypothesis space. However, in practice we often do not have access to a training set consisting of billions of labeled data points. The challenge is typically in the labelling process which often requires human labour.

It seems natural to combat overfitting of a ML method by pruning its hypothesis space $\mathcal{H}$. We prune $\mathcal{H}$ by removing some of the hypothesis in $\mathcal{H}$ to obtain the smaller hypothesis space $\mathcal{H}' \subset \mathcal{H}$. We then replace ERM (4.3) with the restricted (or pruned) ERM

$$\widehat{h} = \operatorname*{argmin}_{h \in \mathcal{H}'} \widehat{L}(h|\mathcal{D}) \text{ with pruned hypothesis space } \mathcal{H}' \subset \mathcal{H}. \tag{7.1}$$

The effective dimension of the pruned hypothesis space $\mathcal{H}'$ is typically much smaller than the effective dimension of the original (large) hypothesis space $\mathcal{H}$, $d_{\rm eff}(\mathcal{H}') \ll d_{\rm eff}(\mathcal{H})$. For a given size m of the training set, the risk of overfitting in (7.1) is much smaller than the risk of overfitting in (4.3).

Let us illustrate the idea of pruning for linear regression using the hypothesis space (3.1) constituted by linear maps $h(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$. The effective dimension of (3.1) is equal to the number of features, $d_{\rm eff}(\mathcal{H}) = n$. The hypothesis space $\mathcal{H}$ might be too large if we use a large number n of features, leading to overfitting. We prune (3.1) by retaining only linear hypothesis maps $h(\mathbf{x}) = \big(\mathbf{w}'\big)^T\mathbf{x}$ whose weight vector $\mathbf{w}'$ satisfies $w'_3 = w'_4 = \ldots = w'_n = 0$. Thus, the hypothesis space $\mathcal{H}'$ is constituted by all linear maps that only depend on the first two features x_1, x_2 of a data point. The effective dimension of $\mathcal{H}'$ is $d_{\rm eff}(\mathcal{H}') = 2$ instead of

$d_{\rm eff}\left(\mathcal{H}\right) = n$.

Pruning the hypothesis space is a special case of a more general strategy which some authors refer to as SRM [145]. The idea behind SRM is to modify the training error in ERM (4.3) to favour a hypothesis that is more smooth or regular. By steering ML methods towards learning a smooth hypothesis, we make it less sensitive, or more robust, to small perturbations of data points in the training set. Section 7.2 discusses the intimate relation between the robustness (against perturbations of the data points in the training set) of a ML method and its ability to generalize to data points outside the training set.

We measure the smoothness of a hypothesis using a regularizer $\mathcal{R}(h) \in \mathbb{R}_+$. The value $\mathcal{R}(h)$ is a quantitative measure for the non-smoothness or irregularity of the hypothesis h. The (design) choice for the regularizer depends on the precise definition of what is meant by regularity or variation of a hypothesis. Section 7.3 discusses how a particular choice for the regularizer $\mathcal{R}(h)$ arises naturally from a probabilistic model for data points.

We obtain SRM by adding the scaled regularizer $\lambda \mathcal{R}(h)$ to the ERM (4.3) ,

$$\begin{aligned} \widehat{h} &= \operatorname*{argmin}_{h \in \mathcal{H}} \big[\widehat{L}(h|\mathcal{D}) + \lambda \mathcal{R}(h)\big] \\ &\stackrel{(2.16)}{=} \operatorname*{argmin}_{h \in \mathcal{H}} \big[(1/m) \sum_{i=1}^{m} L\left((\mathbf{x}^{(i)}, y^{(i)}), h\right) + \lambda \mathcal{R}(h)\big]. \end{aligned} \tag{7.2}$$

We can interpret the penalty term $\lambda \mathcal{R}(h)$ in (7.2) as an estimate (or approximation) for the increase, relative to the training error on $\mathcal{D}$, of the average loss of a hypothesis $\widehat{h}$ when applied to data points outside $\mathcal{D}$. Another interpretation of the term $\lambda \mathcal{R}(h)$ will be discussed in Section 7.3.

The regularization parameter λ allows us to trade between a small training error $\widehat{L}(h^{(\mathbf{w})}|\mathcal{D})$ and small regularization term $\mathcal{R}(h)$, which enforces smoothness or regularity of h. If we choose a large value for λ, we heavily punish any "irregular" hypothesis h with large $\mathcal{R}(h)$ (see (7.2)). Thus, increasing the value of λ results in the solution (minimizer) of (7.2) having smaller $\mathcal{R}(h)$. On the other hand, choosing a small value for λ in (7.2) puts more emphasis on obtaining a hypothesis h incurring a small training error. For the extreme case $\lambda = 0$, the SRM (7.2) reduces to ERM (4.3).

The pruning approach (7.1) is intimately related to the SRM (7.2). They are, in a certain sense, **dual** to each other. First, note that (7.2) reduces to the pruning approach (7.1) when using the regularizer $\mathcal{R}(h) = 0$ for all $h \in \mathcal{H}'$, and $\mathcal{R}(h) = \infty$ otherwise, in (7.2). In the other direction, for many important choices for the regularizer $\mathcal{R}(h)$, there is a restriction

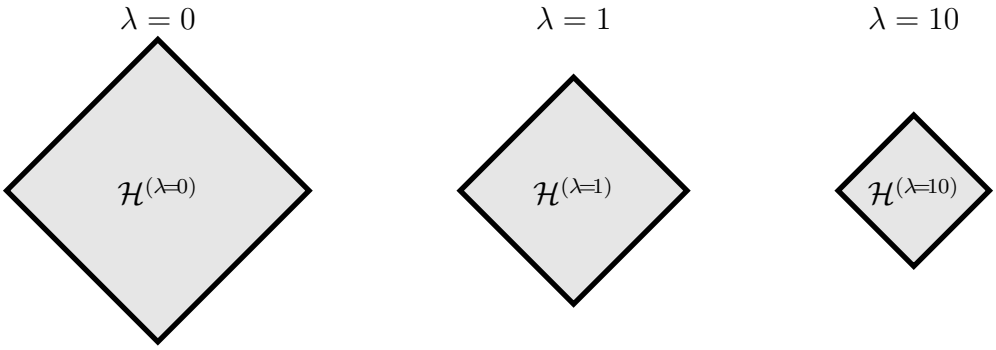

Figure 7.2: Adding the scaled regularizer $\lambda\mathcal{R}(h)$ to the training error in the objective function of SRM (7.2) is equivalent to solving ERM (7.1) with a pruned hypothesis space $\mathcal{H}^{(\lambda)}$.

$\mathcal{H}^{(\lambda)} \subset \mathcal{H}$ such that the solutions of (7.1) and (7.2) coincide (see Figure 7.2). The relation between the optimization problems (7.1) and (7.2) can be made precise using the theory of convex duality (see [14, Ch. 5] and [10]).

For a hypothesis space $\mathcal{H}$ whose elements $h \in \mathcal{H}$ are parametrized by a parameter vector $\mathbf{w} \in \mathbb{R}^n$, we can rewrite SRM (7.2) as

$$\begin{aligned}\widehat{\mathbf{w}}^{(\lambda)} &= \operatorname*{argmin}_{\mathbf{w}\in\mathbb{R}^n} \big[\widehat{L}(h^{(\mathbf{w})}|\mathcal{D}) + \lambda\mathcal{R}(\mathbf{w})\big] \\ &= \operatorname*{argmin}_{\mathbf{w}\in\mathbb{R}^n} \big[(1/m)\sum_{i=1}^{m} L\left((\mathbf{x}^{(i)}, y^{(i)}), h^{(\mathbf{w})}\right) + \lambda\mathcal{R}(\mathbf{w})\big]. \end{aligned} \tag{7.3}$$

For the particular choice of squared error loss (2.8), linear hypothesis space (3.1) and regularizer $\mathcal{R}(\mathbf{w}) = \|\mathbf{w}\|_2^2$, SRM (7.3) specializes to

$$\widehat{\mathbf{w}}^{(\lambda)} = \operatorname*{argmin}_{\mathbf{w}\in\mathbb{R}^n} \big[(1/m)\sum_{i=1}^{m} \left(y^{(i)} - \mathbf{w}^T\mathbf{x}^{(i)}\right)^2 + \lambda\|\mathbf{w}\|_2^2\big]. \tag{7.4}$$

The special case (7.4) of SRM (7.3) is known as ridge regression [58].

Ridge regression (7.4) is equivalent to (see [10, Ch. 5])

$$\widehat{\mathbf{w}}^{(\lambda)} = \operatorname*{argmin}_{h^{(\mathbf{w})}\in\mathcal{H}^{(\lambda)}} (1/m)\sum_{i=1}^{m} \left(y^{(i)} - h^{(\mathbf{w})}(\mathbf{x}^{(i)})\right)^2 \tag{7.5}$$

with the restricted hypothesis space

$$\mathcal{H}^{(\lambda)} := \{h^{(\mathbf{w})} : \mathbb{R}^n \to \mathbb{R} : h^{(\mathbf{w})}(\mathbf{x}) = \mathbf{w}^T\mathbf{x}, \text{ with some } \mathbf{w} \in \mathbb{R}^n, \|\mathbf{w}\|_2^2 \leq C(\lambda)\} \subset \mathcal{H}^{(n)}. \quad (7.6)$$

For any given value λ of the regularization parameter in (7.4), there is a number $C(\lambda)$ such that solutions of (7.4) coincide with the solutions of (7.5). Thus, ridge regression (7.4) is equivalent to linear regression with a pruned version $\mathcal{H}^{(\lambda)}$ of the linear hypothesis space (3.1). The size of the pruned hypothesis space $\mathcal{H}^{(\lambda)}$ (7.6) varies continuously with λ.

Another popular special case of ERM (7.3) is obtained for the regularizer $\mathcal{R}(\mathbf{w}) = \|\mathbf{w}\|_1$ and known as the Lasso [59]

$$\widehat{\mathbf{w}}^{(\lambda)} = \operatorname*{argmin}_{\mathbf{w}\in\mathbb{R}^n} \big[(1/m)\sum_{i=1}^{m} \big(y^{(i)} - \mathbf{w}^T\mathbf{x}^{(i)}\big)^2 + \lambda\|\mathbf{w}\|_1\big]. \quad (7.7)$$

Ridge regression (7.4) and the Lasso (7.7) have fundamentally different computational and statistical properties. Ridge regression (7.4) uses a smooth and convex objective function that can be minimized using efficient GD methods. The objective function of Lasso (7.7) is also convex but non-smooth and therefore requires more advanced optimization methods. The increased computational complexity of Lasso (7.7) comes at the benefit of typically delivering a hypothesis with a smaller expected loss than those obtained from ridge regression [17, 59].

7.2 Robustness

Section 7.1 motivates regularization as a soft variant of model selection. Indeed, the regularization term in SRM (7.2) is equivalent to ERM (7.1) using a pruned hypothesis space. We now discuss an alternative view on regularization as a means to make ML methods robust.

The ML methods discussed in Chapter 4 rest on the idealizing assumption that we have access to the true label values and feature values of labeled data points (that form a training set). These methods learn a hypothesis $h \in \mathcal{H}$ with minimum average loss (training error) incurred for data points in the training set. In practice, the acquisition of label and feature values might be prone to errors. These errors might stem from the measurement device itself (hardware failures or thermal noise in electronic devices) or might be due to human mistakes such as labelling errors.

Let us assume for the sake of exposition that the label values $y^{(i)}$ in the training set are accurate but that the features $\mathbf{x}^{(i)}$ are a perturbed version of the true features of the ith

data point. Thus, instead of having observed the data point $\big(\mathbf{x}^{(i)}, y^{(i)}\big)$ we could have equally well observed the data point $\big(\mathbf{x}^{(i)} + \boldsymbol{\varepsilon}, y^{(i)}\big)$ in the training set. Here, we have modelled the perturbations in the features using a RV $\boldsymbol{\varepsilon}$. The probability distribution of the perturbation $\boldsymbol{\varepsilon}$ is a design parameter that controls robustness properties of the overall ML method. We will study a particular choice for this distribution in Section 7.3.

A robust ML method should learn a hypothesis that incurs a small loss not only for a specific data point $\big(\mathbf{x}^{(i)}, y^{(i)}\big)$ but also for perturbed data points $\big(\mathbf{x}^{(i)} + \boldsymbol{\varepsilon}, y^{(i)}\big)$. Therefore, it seems natural to replace the loss $L\big(\big(\mathbf{x}^{(i)}, y^{(i)}\big), h\big)$, incurred on the ith data point in the training set, with the expectation

$$\mathbb{E}\big\{ L\big(\big(\mathbf{x}^{(i)} + \boldsymbol{\varepsilon}, y^{(i)}\big), h\big) \big\}. \tag{7.8}$$

The expectation (7.8) is computed using the probability distribution of the perturbation $\boldsymbol{\varepsilon}$. We will show in Section 7.3 that minimizing the average of the expectation (7.8), for $i = 1, \ldots, m$, is equivalent to the SRM (7.2).

Using the expected loss (7.8) is not the only possible approach to make a ML method robust. Another approach to make a ML method robust is known as bootstrap aggregation (bagging). The idea of bagging is to use the bootstrap method (see Section 6.5 and [58, Ch. 8]) to construct a finite number of perturbed copies $\mathcal{D}^{(1)}, \ldots, \mathcal{D}^{(B)}$ of the original training set $\mathcal{D}$.

We then learn (e.g, using ERM) a separate hypothesis $h^{(b)}$ for each perturbed copy $\mathcal{D}^{(b)}$, $b = 1, \ldots, B$. This results in a whole ensemble of different hypotheses $h^{(b)}$ which might even belong to different hypothesis spaces. For example, one the hypothesis $h^{(1)}$ could be a linear map (see Section 3.1) and the hypothesis $h^{(2)}$ could be obtained from an ANN (see Section 3.11).

The final hypothesis delivered by bagging is obtained by combining or aggregating (e.g., using the average) the predictions $h^{(b)}\big(\mathbf{x}\big)$ delivered by each hypothesis $h^{(b)}$, for $b = 1, \ldots, B$ in the ensemble. The ML method referred to as random forest uses bagging to learn an ensemble of decision trees (see Chapter 3.10). The individual predictions obtained from the different decision trees forming a random forest are then combined (e.g., using an average for numeric labels or a majority vote for finite-valued labels), to obtain a final prediction [58].

7.3 Data Augmentation

ML methods using ERM (4.3) are prone to overfitting as soon as the effective dimension of the hypothesis space $\mathcal{H}$ exceeds the number m of data points in the training set. Section 6.3 and Section 7.1 approached this by modifying either the model or the loss function by adding a regularization term. Both approaches prune the hypothesis space $\mathcal{H}$ underlying a ML method to reduce the effective dimension $d_{\text{eff}}(\mathcal{H})$. Model selection does this reduction in a discrete fashion while regularization implements a soft "shrinking" of the hypothesis space.

Instead of trying to reduce the effective dimension we could also try to increase the number m of data points in the training set used for ERM (4.3). We now discuss how to synthetically generate new labeled data points by exploiting statistical symmetries of data.

The data arising in many ML applications exhibit intrinsic symmetries and invariances at least in some approximation. The rotated image of a cat still shows a cat. The temperature measurement taken at a given location will be similar to another measurement taken 10 milliseconds later. Data augmentation exploits such symmetries and invariances to augment the raw data with additional synthetic data.

Let us illustrate data augmentation using an application that involves data points characterized by features $\mathbf{x} \in \mathbb{R}^n$ and number labels $y \in \mathbb{R}$. We assume that the data generating process is such that data points with close feature values have the same label. Equivalently, this assumption is requiring the resulting ML method to be robust against small perturbations of the feature values (see Section 7.2). This suggests to augment a data point $\big(\mathbf{x}, y\big)$ by several synthetic data points

$$\big(\mathbf{x} + \boldsymbol{\varepsilon}^{(1)}, y\big), \ldots, \big(\mathbf{x} + \boldsymbol{\varepsilon}^{(B)}, y\big), \tag{7.9}$$

with $\boldsymbol{\varepsilon}^{(1)}, \ldots, \boldsymbol{\varepsilon}^{(B)}$ being realizations of i.i.d. random vectors with the same probability distribution $p(\boldsymbol{\varepsilon})$.

Given a (raw) dataset $\mathcal{D} = \big\{\big(\mathbf{x}^{(1)}, y^{(1)}\big), \ldots, \big(\mathbf{x}^{(m)}, y^{(m)}\big)\big\}$ we denote the associated aug-

mented dataset by

$$\begin{aligned}\mathcal{D}' = \big\{ &\big(\mathbf{x}^{(1,1)}, y^{(1)}\big), \ldots, \big(\mathbf{x}^{(1,B)}, y^{(1)}\big), \\ &\big(\mathbf{x}^{(2,1)}, y^{(2)}\big), \ldots, \big(\mathbf{x}^{(2,B)}, y^{(2)}\big), \\ &\ldots \\ &\big(\mathbf{x}^{(m,1)}, y^{(m)}\big), \ldots, \big(\mathbf{x}^{(m,B)}, y^{(m)}\big)\big\}. \end{aligned} \tag{7.10}$$

The size of the augmented dataset $\mathcal{D}'$ is $m' = B \times m$. For a sufficiently large augmentation parameter B, the augmented sample size m' is larger than the effective dimension n of the hypothesis space $\mathcal{H}$. We then learn a hypothesis via ERM on the augmented dataset,

$$\begin{aligned}\widehat{h} &= \operatorname*{argmin}_{h \in \mathcal{H}} \widehat{L}(h|\mathcal{D}') \\ &\overset{(7.10)}{=} \operatorname*{argmin}_{h \in \mathcal{H}} (1/m') \sum_{i=1}^{m} \sum_{b=1}^{B} L\big((\mathbf{x}^{(i,b)}, y^{(i,b)}), h\big) \\ &\overset{(7.9)}{=} \operatorname*{argmin}_{h \in \mathcal{H}} (1/m) \sum_{i=1}^{m} (1/B) \sum_{b=1}^{B} L\big((\mathbf{x}^{(i)} + \boldsymbol{\varepsilon}^{(b)}, y^{(i)}), h\big). \end{aligned} \tag{7.11}$$

We can interpret data-augmented ERM (7.11) as a data-driven form of regularization (see Section 7.1): For each data point $\big(\mathbf{x}^{(i)}, y^{(i)}\big) \in \mathcal{D}$ in $\mathcal{D}$, we replace the corresponding loss $L\big((\mathbf{x}^{(i)}, y^{(i)}), h\big)$ with the average loss $(1/B)\sum_{b=1}^{B} L\big((\mathbf{x}^{(i)} + \boldsymbol{\varepsilon}^{(b)}, y^{(i)}), h\big)$. This average loss is computed over the augmented data points arising from $\big(\mathbf{x}^{(i)}, y^{(i)}\big) \in \mathcal{D}$.

The actual implementation of (7.11) require to first generate B realizations $\boldsymbol{\varepsilon}^{(b)} \in \mathbb{R}^n$ of i.i.d. RVs with common probability distribution $p(\boldsymbol{\varepsilon})$. This generation might be computationally costly for large values of B or n. However, when using a large augmentation parameter B, we might use the approximation

$$(1/B) \sum_{b=1}^{B} L\big((\mathbf{x}^{(i)} + \boldsymbol{\varepsilon}^{(b)}, y^{(i)}), h\big) \approx \mathbb{E}\big\{ L\big((\mathbf{x}^{(i)} + \boldsymbol{\varepsilon}, y^{(i)}), h\big) \big\}. \tag{7.12}$$

This approximation is made precise by a key result of probability theory, known as the law of large numbers. We obtain an instance of ERM by inserting (7.12) into (7.11),

$$\widehat{h} = \operatorname*{argmin}_{h \in \mathcal{H}} (1/m) \sum_{i=1}^{m} \mathbb{E}\big\{ L\big((\mathbf{x}^{(i)} + \boldsymbol{\varepsilon}, y^{(i)}), h\big) \big\}. \tag{7.13}$$

The usefulness of (7.13) as an approximation to the augmented ERM (7.11) depends on the difficulty of computing the expectation $\mathbb{E}\big\{ L\left((\mathbf{x}^{(i)}+\boldsymbol{\varepsilon}, y^{(i)}), h\right)\big\}$. The complexity of computing this expectation depends on the choice of loss function and the choice for the probability distribution $p(\boldsymbol{\varepsilon})$.

Let us study (7.13) for the special case linear regression with squared error loss (2.8) and linear hypothesis space (3.1),

$$\widehat{h} = \underset{h(\mathbf{w})\in\mathcal{H}^{(n)}}{\operatorname{argmin}} (1/m) \sum_{i=1}^{m} \mathbb{E}\big\{ \big(y^{(i)} - \mathbf{w}^T\big(\mathbf{x}^{(i)}+\boldsymbol{\varepsilon}\big)\big)^2\big\}. \tag{7.14}$$

We use perturbations $\boldsymbol{\varepsilon}$ drawn from a multivariate normal distribution with zero mean and covariance matrix $\sigma^2\mathbf{I}$,

$$\boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0}, \sigma^2\mathbf{I}). \tag{7.15}$$

We develop (7.14) further by using

$$\mathbb{E}\{\big(y^{(i)} - \mathbf{w}^T\mathbf{x}^{(i)}\big)\boldsymbol{\varepsilon}\} = \mathbf{0}. \tag{7.16}$$

The identity (7.16) uses that the data points $\big(\mathbf{x}^{(i)}, y^{(i)}\big)$ are fixed and known (deterministic) while $\boldsymbol{\varepsilon}$ is a zero-mean random vector. Combining (7.16) with (7.14),

$$\begin{aligned}\mathbb{E}\big\{ \big(y^{(i)} - \mathbf{w}^T\big(\mathbf{x}^{(i)}+\boldsymbol{\varepsilon}\big)\big)^2\big\} &= \big(y^{(i)} - \mathbf{w}^T\mathbf{x}^{(i)}\big)^2 + \|\mathbf{w}\|_2^2\, \mathbb{E}\big\{\, \|\boldsymbol{\varepsilon}\|_2^2\,\big\} \\ &= \big(y^{(i)} - \mathbf{w}^T\mathbf{x}^{(i)}\big)^2 + n\,\|\mathbf{w}\|_2^2\,\sigma^2. \end{aligned} \tag{7.17}$$

where the last step used $\mathbb{E}\big\{\, \|\boldsymbol{\varepsilon}\|_2^2\,\big\} \overset{(7.15)}{=} n\sigma^2$. Inserting (7.17) into (7.14),

$$\widehat{h} = \underset{h(\mathbf{w})\in\mathcal{H}^{(n)}}{\operatorname{argmin}} (1/m) \sum_{i=1}^{m} \big(y^{(i)} - \mathbf{w}^T\mathbf{x}^{(i)}\big)^2 + n\,\|\mathbf{w}\|_2^2\,\sigma^2. \tag{7.18}$$

We have obtained (7.18) as an approximation of the augmented ERM (7.11) for the special case of squared error loss (2.8) and the linear hypothesis space (3.1). This approximation uses the law of large numbers (7.12) and becomes more accurate for increasing augmentation parameter B.

Note that (7.18) is nothing but ridge regression (7.4) using the regularization parameter $\lambda = n\sigma^2$. Thus, we can interpret ridge regression as implicit data augmentation (7.10) by applying random perturbations (7.9) to the feature vectors in the original training set $\mathcal{D}$.

The regularizer $\mathcal{R}(\mathbf{w}) = \|\mathbf{w}\|_2^2$ in (7.18) arose naturally from the specific choice for the probability distribution (7.15) of the random perturbation $\boldsymbol{\varepsilon}^{(i)}$ in (7.9) and using the squared error loss. Other choices for this probability distribution or the loss function result in different regularizers.

Augmenting data points with random perturbations distributed according (7.15) treat the features of a data point independently. For application domains that generate data points with highly correlated features it might be useful to augment data points using random perturbations $\boldsymbol{\varepsilon}$ (see (7.9)) distributed as

$$\boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{C}). \tag{7.19}$$

The covariance matrix $\mathbf{C}$ of the perturbation $\boldsymbol{\varepsilon}$ can be chosen using domain expertise or estimated (see Section 7.5). Inserting the distribution (7.19) into (7.13),

$$\widehat{h} = \operatorname*{argmin}_{h^{(\mathbf{w})} \in \mathcal{H}^{(n)}} \Big[(1/m) \sum_{i=1}^{m} \big(y^{(i)} - \mathbf{w}^T \mathbf{x}^{(i)}\big)^2 + \mathbf{w}^T \mathbf{C} \mathbf{w} \Big]. \tag{7.20}$$

Note that (7.20) reduces to ordinary ridge regression (7.18) for the choice $\mathbf{C} = \sigma^2 \mathbf{I}$.

7.4 Statistical and Computational Aspects of Regularization

The goal of this section is to develop a better understanding for the effect of the regularization term in SRM (7.3). We will analyze the solutions of ridge regression (7.4) which is the special case of SRM using the linear hypothesis space (3.1) and squared error loss (2.8). Using the feature matrix $\mathbf{X} = \big(\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)}\big)^T$ and label vector $\mathbf{y} = (y^{(1)}, \ldots, y^{(m)})^T$, we can rewrite (7.4) more compactly as

$$\widehat{\mathbf{w}}^{(\lambda)} = \operatorname*{argmin}_{\mathbf{w} \in \mathbb{R}^n} \big[(1/m) \, \|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2 + \lambda \, \|\mathbf{w}\|_2^2 \big]. \tag{7.21}$$

The solution of (7.21) is given by [134, Sec. 3.3.]

$$\widehat{\mathbf{w}}^{(\lambda)} = (1/m)\big((1/m)\mathbf{X}^T\mathbf{X} + \lambda \mathbf{I}\big)^{-1} \mathbf{X}^T \mathbf{y}. \tag{7.22}$$

For $\lambda = 0$, (7.22) reduces to the formula (6.18) for the optimal weights in linear regression (see (7.4) and (4.5)). Note that for $\lambda > 0$, the formula (7.22) is always valid, even when $\mathbf{X}^T\mathbf{X}$ is singular (not invertible). For $\lambda > 0$ the optimization problem (7.21) (and (7.4)) has the unique solution (7.22).

To study the statistical properties of the predictor $h^{(\widehat{\mathbf{w}}^{(\lambda)})}(\mathbf{x}) = \big(\widehat{\mathbf{w}}^{(\lambda)}\big)^T\mathbf{x}$ (see (7.22)) we use the probabilistic toy model (6.13), (6.16) and (6.17) that we used already in Section 6.4. We interpret the training data $\mathcal{D}^{(\text{train})} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^m$ as realizations of i.i.d. RVs whose distribution is defined by (6.13), (6.16) and (6.17).

We can then define the average prediction error of ridge regression as

$$E_{\text{pred}}^{(\lambda)} := \mathbb{E}\bigg\{\bigg(y - h^{(\widehat{\mathbf{w}}^{(\lambda)})}(\mathbf{x})\bigg)^2\bigg\}. \tag{7.23}$$

As shown in Section 6.4, the error $E_{\text{pred}}^{(\lambda)}$ is the sum of three components: the bias, the variance and the noise variance σ^2 (see (6.30)). The bias of $\widehat{\mathbf{w}}^{(\lambda)}$ is

$$B^2 = \mathbb{E}\bigg\{\ \big\|(\mathbf{I} - (\mathbf{X}^T\mathbf{X} + m\lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{X})\overline{\mathbf{w}}\big\|_2^2\ \bigg\}. \tag{7.24}$$

For sufficiently large size m of the training set, we can use the approximation

$$\mathbf{X}^T\mathbf{X} \approx m\mathbf{I} \tag{7.25}$$

such that (7.24) can be approximated as

$$\begin{aligned} B^2 &\approx \big\|(\mathbf{I} - (\mathbf{I} + \lambda\mathbf{I})^{-1})\overline{\mathbf{w}}\big\|_2^2 \\ &= \sum_{j=1}^n \frac{\lambda}{1+\lambda}\overline{w}_j^2. \end{aligned} \tag{7.26}$$

Let us compare the (approximate) bias term (7.26) of ridge regression with the bias term (6.24) of ordinary linear regression (which is the extreme case of ridge regression with $\lambda = 0$). The bias term (7.26) increases with increasing regularization parameter λ in ridge regression (7.4). Sometimes the increase in bias is outweighed by the reduction in variance. The variance typically decreases with increasing λ as shown next.

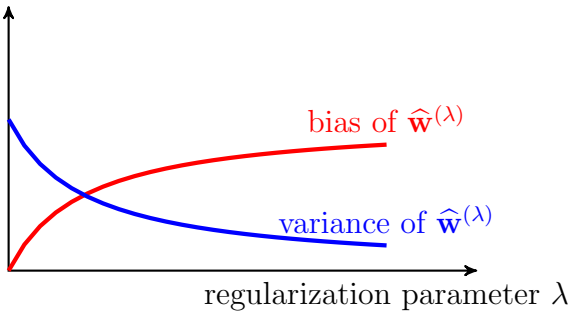

Figure 7.3: The bias and variance of ridge regression (7.4) depend on the regularization parameter λ in an opposite manner resulting in a bias-variance trade-off.

The variance of ridge regression (7.4) satisfies

$$V = (\sigma^2/m^2)\times \\ \mathrm{tr}\big\{\mathbb{E}\{((1/m)\mathbf{X}^T\mathbf{X}+\lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{X}((1/m)\mathbf{X}^T\mathbf{X}+\lambda\mathbf{I})^{-1}\}\big\}. \quad (7.27)$$

Inserting the approximation (7.25) into (7.27),

$$V \approx (\sigma^2/m^2)\mathrm{tr}\big\{\mathbb{E}\{(\mathbf{I}+\lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{X}(\mathbf{I}+\lambda\mathbf{I})^{-1}\}\big\} = \sigma^2(1/m)(n/(1+\lambda)^2). \quad (7.28)$$

According to (7.28), the variance of $\widehat{\mathbf{w}}^{(\lambda)}$ decreases with increasing regularization parameter λ of ridge regression (7.4). This is the opposite behaviour as observed for the bias (7.26), which increases with increasing λ. By comparing the variance approximation (7.28) with the variance (6.28) of linear regression suggests to interpret the ratio $n/(1+\lambda)^2$ as an effective number of features used by ridge regression. Increasing the regularization parameter λ decreases the effective number of features.

Figure 7.3 illustrates the trade-off between the bias B^2 (7.26) of ridge regression, which increases for increasing λ, and the variance V (7.28) which decreases with increasing λ. Note that we have seen another example for a bias-variance trade-off in Section 6.4. This trade-off was traced out by a discrete (model complexity) parameter $l \in \{1, 2, \ldots\}$ (see (6.14)). In stark contrast to discrete model selection, the bias-variance trade-off for ridge regression is traced out by the continuous regularization parameter $\lambda \in \mathbb{R}_+$.

The main statistical effect of the regularization term in ridge regression is to balance the bias with the variance to minimize the average prediction error of the learnt hypothesis. There is also a computational effect or adding a regularization term. Roughly speaking, the

regularization term serves as a pre-conditioning of the optimization problem and, in turn, reduces the computational complexity of solving ridge regression (7.21).

The objective function in (7.21) is a smooth (infinitely often differentiable) convex function. We can therefore use GD to solve (7.21) efficiently (see Chapter 5). Algorithm 8 summarizes the application of GD to (7.21). The computational complexity of Algorithm 8 depends crucially on the number of GD iterations required to reach a sufficiently small neighbourhood of the solutions to (7.21). Adding the regularization term $\lambda\|\mathbf{w}\|_2^2$ to the objective function of linear regression speeds up GD. To verify this claim, we first rewrite (7.21) as the quadratic problem

$$\min_{\mathbf{w}\in\mathbb{R}^n} \underbrace{(1/2)\mathbf{w}^T\mathbf{Q}\mathbf{w} - \mathbf{q}^T\mathbf{w}}_{=f(\mathbf{w})} \text{ with } \mathbf{Q} = (1/m)\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I}, \mathbf{q} = (1/m)\mathbf{X}^T\mathbf{y}. \tag{7.29}$$

This is similar to the quadratic optimization problem (4.9) underlying linear regression but with a different matrix $\mathbf{Q}$. The computational complexity (number of iterations) required by GD (see (5.6)) to solve (7.29) up to a prescribed accuracy depends crucially on the condition number $\kappa(\mathbf{Q}) \geq 1$ of the psd matrix $\mathbf{Q}$ [68]. The smaller the condition number $\kappa(\mathbf{Q})$, the fewer iterations are required by GD. We refer to a matrix with a small condition number as being "well-conditioned".

The condition number of the matrix $\mathbf{Q}$ in (7.29) is given by

$$\kappa(\mathbf{Q}) = \frac{\lambda_{\max}((1/m)\mathbf{X}^T\mathbf{X}) + \lambda}{\lambda_{\min}((1/m)\mathbf{X}^T\mathbf{X}) + \lambda}. \tag{7.30}$$

According to (7.30), the condition number $\kappa(\mathbf{Q})$ tends to one for increasing regularization parameter λ,

$$\lim_{\lambda\to\infty} \frac{\lambda_{\max}((1/m)\mathbf{X}^T\mathbf{X}) + \lambda}{\lambda_{\min}((1/m)\mathbf{X}^T\mathbf{X}) + \lambda} = 1. \tag{7.31}$$

Thus, the number of required GD iterations in Algorithm 8 decreases with increasing regularization parameter λ.

7.5 Semi-Supervised Learning

Consider the task of predicting the numeric label y of a data point $\mathbf{z} = \big(\mathbf{x}, y\big)$ based on its feature vector $\mathbf{x} = \big(x_1, \ldots, x_n\big)^T \in \mathbb{R}^n$. At our disposal are two datasets $\mathcal{D}^{(u)}$ and $\mathcal{D}^{(l)}$.

Algorithm 8 Regularized Linear regression via GD

Input: dataset $\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^{m}$; GD learning rate $\alpha > 0$.
Initialize:set $\mathbf{w}^{(0)} := \mathbf{0}$; set iteration counter $r := 0$
1: **repeat**
2: $r := r + 1$ (increase iteration counter)
3: $\mathbf{w}^{(r)} := (1 - \alpha\lambda)\mathbf{w}^{(r-1)} + \alpha(2/m)\sum_{i=1}^{m}(y^{(i)} - \big(\mathbf{w}^{(r-1)}\big)^{T}\mathbf{x}^{(i)})\mathbf{x}^{(i)}$ (do a GD step (5.6))
4: **until** stopping criterion met
Output: $\mathbf{w}^{(r)}$ (which approximates $\widehat{\mathbf{w}}^{(\lambda)}$ in (7.21))

For each datapoint in $\mathcal{D}^{(u)}$ we only know the feature vector. We therefore refer to $\mathcal{D}^{(u)}$ as "unlabelled data". For each datapoint in $\mathcal{D}^{(l)}$ we know both, the feature vector $\mathbf{x}$ and the label y. We therefore refer to $\mathcal{D}^{(l)}$ as "labeled data".

SSL methods exploit the information provided by unlabelled data $\mathcal{D}^{(u)}$ to support the learning of a hypothesis based on minimizing its empirical risk on the labelled (training) data $\mathcal{D}^{(l)}$. The success of SSL methods depends on the statistical properties of the data generated within a given application domain. Loosely speaking, the information provided by the probability distribution of the features must be relevant for the ultimate task of predicting the label y from the the features $\mathbf{x}$ [21].

Let us design a SSL method, summarized in Algorithm 9 below, using the data augmentation perspective from Section 7.3. The idea is to augment the (small) labeled dataset $\mathcal{D}^{(l)}$ by adding random perturbations fo the features of each data point in $\mathcal{D}^{(l)}$. This is reasonable for applications where features are subject to inherent measurement or modelling errors. Given a data point with vector $\mathbf{x}$ we could have equally well observed a feature vector $\mathbf{x} + \boldsymbol{\varepsilon}$ with some small random perturbation $\boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{C})$. To estimate the covariance matrix matrix $\mathbf{C}$, we use the sample covariance matrix of the feature vectors in the (large) unlabelled dataset $\mathcal{D}^{(u)}$. We then learn a hypothesis using the augmented (regularized) ERM (7.20).

7.6 Multitask Learning

Consider a specific learning task of finding a hypothesis h with minimum (expected) loss $L\left((\mathbf{x}, y), h\right)$. Note that the loss incurred by h for a specific data point depends on the definition for the label of a data point. We can obtain different learning tasks for the same data points by using different choices or definitions for the label of a data point. Multitask learning exploits the similarities between different learning tasks to jointly solve them. Let us next discuss a simple example of a multitask learning problem.

Algorithm 9 A Semi-Supervised Learning Algorithm

Input: labeled dataset $\mathcal{D}^{(l)} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^{m}$; unlabeled dataset $\mathcal{D}^{(u)} = \{\widetilde{\mathbf{x}}^{(i)}\}_{i=1}^{m'}$

1: compute $\mathbf{C}$ via sample covariance on $\mathcal{D}^{(u)}$,

$$\mathbf{C} := (1/m') \sum_{i=1}^{m'} \big(\widetilde{\mathbf{x}}^{(i)} - \widehat{\mathbf{x}}\big)\big(\widetilde{\mathbf{x}}^{(i)} - \widehat{\mathbf{x}}\big)^T \text{ with } \widehat{\mathbf{x}} := (1/m') \sum_{i=1}^{m'} \widetilde{\mathbf{x}}^{(i)}. \tag{7.32}$$

2: compute (e.g. using GD)

$$\widehat{\mathbf{w}} := \operatorname*{argmin}_{\mathbf{w} \in \mathbb{R}^n} \Big[(1/m) \sum_{i=1}^{m} \big(y^{(i)} - \mathbf{w}^T \mathbf{x}^{(i)}\big)^2 + \mathbf{w}^T \mathbf{C} \mathbf{w} \Big]. \tag{7.33}$$

Output: hypothesis $\widehat{h}(\mathbf{x}) = \big(\widehat{\mathbf{w}}\big)^T \mathbf{x}$

Consider a data point $\mathbf{z}$ representing a hand-drawing that is collected via the online game `https://quickdraw.withgoogle.com/`. The features of a data point are the pixel intensities of the bitmap which is used to store the hand-drawing. As label we could use the fact if a hand-drawing shows an apple or not. This results in the learning task $\mathcal{T}^{(1)}$. Another choice for the label of a hand-drawing could be the fact if a hand-drawing shows a fruit at all or not. This results in another learning task $\mathcal{T}^{(2)}$ which is similar but different from the task $\mathcal{T}^{(1)}$.

The idea of multitask learning is that a reasonable hypothesis h for some learning task should also do well for a similar learning task. Thus, we can use the loss incurred on similar learning tasks as a regularization term for learning a hypothesis for the learning task at hand. Algorithm 10 is a straightforward implementation of this idea for a given dataset that gives rise to T related learning tasks $\mathcal{T}^{(1)}, \ldots, \mathcal{T}^{(T)}$. For each individual learning task $\mathcal{T}^{(t')}$ it uses the loss on the remaining learning tasks $\mathcal{T}^{(t)}$, with $t \neq t'$, as regularization term in (7.34).

The applicability of Algorithm 10 is somewhat limited as it aims at finding a single hypothesis that does well for all T learning tasks simultaneously. For certain application domains it might be more reasonable to not learn a single hypothesis for all learning tasks but to learn a separate hypothesis $h^{(t)}$ for each learning task $t = 1, \ldots, T$. However, these separate hypothesis maps typically might still share some structural similarities.[1] We can

[1] One important example for such a structural similarity in the case of linear predictors $h^{(t)}(\mathbf{x}) = \big(\mathbf{w}^{(t)}\big)^T \mathbf{x}$ is that the parameter vectors $\mathbf{w}^{(T)}$ have a small joint support. Requiring the parameter vectors to have a small joint support is equivalent to requiring the stacked vector $\widetilde{\mathbf{w}} = \big(\mathbf{w}^{(1)}, \ldots, \mathbf{w}^{(T)}\big)$ to be block (group)

Algorithm 10 A Multitask Learning Algorithm

Input: dataset $\mathcal{D} = \{\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(m)}\}$; T learning tasks with loss functions $L^{(1)}, \ldots, L^{(T)}$, hypothesis space $\mathcal{H}$

1: learn a hypothesis $\widehat{h}$ via

$$\widehat{h} := \operatorname*{argmin}_{h \in \mathcal{H}} \sum_{t=1}^{T} \sum_{i=1}^{m} L^{(t)}\big(\mathbf{z}^{(i)}, h\big). \tag{7.34}$$

Output: hypothesis $\widehat{h}$

enforce different notion of similarity between hypothesis maps $h^{(t)}$ by adding a regularization term to the individual loss functions of the learning tasks.

Algorithm 11 generalizes Algorithms 10 by learning a separate hypothesis for each task t while requiring these hypotheses to be structurally similar. The structural (dis-)similarity between the hypothesis maps is measured by a regularization term $\mathcal{R}$ in (7.35).

Algorithm 11 A Multitask Learning Algorithm

Input: dataset $\mathcal{D} = \{\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(m)}\}$ with T associated learning tasks with loss functions $L^{(1)}, \ldots, L^{(T)}$, hypothesis space $\mathcal{H}$

1: learn a hypothesis $\widehat{h}$ via

$$\widehat{h}^{(1)}, \ldots, \widehat{h}^{(T)} := \operatorname*{argmin}_{h^{(1)}, \ldots, h^{(T)} \in \mathcal{H}} \sum_{t=1}^{T} \sum_{i=1}^{m} L^{(t)}\big(\mathbf{z}^{(i)}, h^{(t)}\big) + \lambda \mathcal{R}\big(h^{(1)}, \ldots, h^{(T)}\big). \tag{7.35}$$

Output: hypotheses $\widehat{h}^{(1)}, \ldots, \widehat{h}^{(T)}$

7.7 Transfer Learning

Regularization is also instrumental for transfer learning to capitalize on synergies between different related learning tasks [108, 61]. Transfer learning is enabled by constructing regularization terms for a learning task by using the result of a previous learning task. While multitask learning methods solve many related learning tasks simultaneously, transfer learning methods operate in a more sequential fashion.

Let us illustrate the idea of transfer learning using two learning tasks which differ significantly in their intrinsic difficulty. Informally, we consider a learning task to be easy if

sparse [34].

we can easily gather large amounts of labeled (training) data for that task. Consider the learning task $\mathcal{T}^{(1)}$ of predicting whether an image shows a cat or not. For this learning task we can easily gather a large training set $\mathcal{D}^{(1)}$ using via image collections of animals. Another (related) learning task $\mathcal{T}^{(2)}$ is to predict whether an image shows a cat of a particular breed, with a particular body height and with a specific age. The learning task $\mathcal{T}^{(2)}$ is more dificult than $\mathcal{T}^{(1)}$ since we have only a very limited amount of cat images for which we know the particular breed, body height and precise age of the depicted cat.

7.8 Exercises

Exercise 7.1. Ridge Regression is a Quadratic Problem. Consider the linear hypothesis space consisting of linear maps parameterized by weights $\mathbf{w}$. We try to find the best linear map by minimizing the regularized average squared error loss (empirical risk) incurred on a training set

$$\mathcal{D} := \big\{(\mathbf{x}^{(1)}, y^{(1)}), (\mathbf{x}^{(2)}, y^{(2)}), \ldots, (\mathbf{x}^{(m)}, y^{(m)})\big\}.$$

Ridge reression augments the average squared error loss on $\mathcal{D}$ by the regularizer $\|\mathbf{w}\|^2$, yielding the following learning problem

$$\min_{\mathbf{w}\in\mathbb{R}^n} f(\mathbf{w}) = (1/m)\sum_{i=1}^{m}\big(y^{(i)} - \mathbf{w}^T\mathbf{x}^{(i)}\big) + \lambda\|\mathbf{w}\|_2^2.$$

Is it possible to rewrite the objective function $f(\mathbf{w})$ as a convex quadratic function $f(\mathbf{w}) = \mathbf{w}^T\mathbf{C}\mathbf{w} + \mathbf{b}^T\mathbf{w} + c$? If this is possible, how are the matrix $\mathbf{C}$, vector $\mathbf{b}$ and constant c related to the feature vectors and labels of the training data ?

Exercise 7.2. Regularization or Model Selection. Consider data points, each characterized by $n = 10$ features $\mathbf{x} \in \mathbb{R}^n$ and a single numeric label y. We want to learn a linear hypothesis $h(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$ by minimizing the average squared error on the training set $\mathcal{D}$ of size $m = 4$. We could learn such a hypothesis by two approaches. The first approach is to split the dataset into a training set and a validation set. Then we consider all models that consists of linear hypotheses with weight vectors having at most two non-zero weights. Each of these models corresponds to a different subset of two weights that might be non-zero. Find the model resulting in the smallest validation errors (see Algorithm 5). Compute the

average loss of the resulting optimal linear hypothesis on some data points that have neither been used in the training set nor the validation set. Compare this average loss ("test error") with the average loss obtained on the same data points by the hypothesis learnt by ridge regression (7.4).

Chapter 8

Clustering

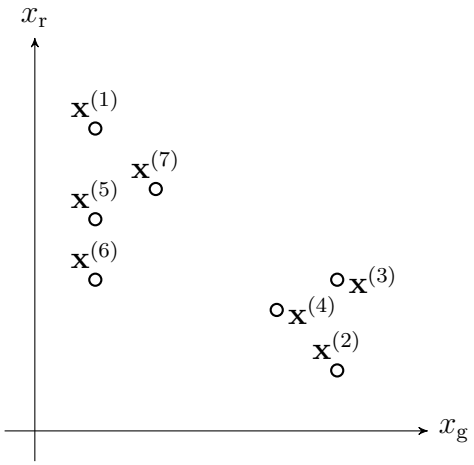

Figure 8.1: Each circle represents an image which is characterized by its average redness $x_{\rm r}$ and average greenness $x_{\rm g}$. The i-th image is depicted by a circle located at the point $\mathbf{x}^{(i)} = \big(x_{\rm r}^{(i)}, x_{\rm g}^{(i)}\big)^T \in \mathbb{R}^2$. It seems that the images can be grouped into two clusters.

So far we focused on ML methods that use the ERM principle and lean a hypothesis by minimizing the discrepancy between its predictions and the true labels on a training set. These methods are referred to as supervised methods as they require labeled data points for which the true label values have been determined by some human (who serves as a "supervisor"). This and the following chapter discus ML methods which do not require to know the label of any data point. These methods are often referred to as "unsupervised" since they do not require a "supervisor" to provide the label values for any data point.

One important family of unsupervised ML methods aim at clustering a given set of data

points such as those depicted in Figure 8.1. The basic idea of clustering is to decompose a set of data points into few subsets or clusters that consist of similar data points. For the dataset in Figure 8.1 it seems reasonable to define two clusters, one cluster $\{\mathbf{x}^{(1)}, \mathbf{x}^{(5)}, \mathbf{x}^{(6)}, \mathbf{x}^{(7)}\}$ and a second cluster $\{\mathbf{x}^{(2)}, \mathbf{x}^{(3)}, \mathbf{x}^{(4)}, \mathbf{x}^{(8)}\}$.

Formally, clustering methods learn a hypothesis that assign each data point either to precisely one cluster (see Section 8.1) or several clusters with different degrees of belonging (see Section 8.2). Different clustering methods use different measures for the similarity between data points. For data points characterized by (numeric) Euclidean feature vectors, the similarity between data points can be naturally defined in terms of the Euclidean distance between feature vectors. Section 8.3 discusses clustering methods that use notions of similarity that are not based on a Euclidean space.

There is a strong conceptual link between clustering methods and the classification methods discussed in Chapter 3. Both type of methods learn a hypothesis that reads in the features of a data point and delivers a prediction for some quantity of interest. In classification methods, this quantity of interest is some generic label of a data point. For clustering methods, this quantity of interest for a data point is the cluster assignment (for hard clustering) of the degree of belonging (for soft clustering). A main difference between clustering and classification is that clustering methods do not require the true label (cluster assignment or degree of belonging) of a single data point.

Classification methods learn a good hypothesis via minimizing their average loss incurred on a training set of labeled data points. In contrast, clustering methods do not have access to a single labeled data point. To find the correct labels (cluster assignments), clustering methods must rely solely on the intrinsic geometry of the data points. We will see that clustering methods use this intrinsic geometry to determine an empirical risk incurred by a candidate hypothesis (which maps data points to cluster assignments or degree of belonging). Thus, much like classification methods, also clustering implement the generic ERM principle (see Chapter 4) to find a good hypothesis (clustering).

This chapter discusses two main flavours of clustering methods:

- hard clustering (see Section 8.1)
- and soft clustering methods (see Section 8.2).

Hard clustering methods learn a hypothesis h that reads in the feature vector $\mathbf{x}$ of a data point and delivers a predicted cluster assignment $\hat{y} = h(\mathbf{x}) \in \{1, \ldots, k\}$. Thus, assigns each

data point to one single cluster. Section 8.1 will discuss one of the most widely-used hard clustering algorithms which is known as k-means.

In contrast to hard clustering methods, soft clustering methods assign each data point to several clusters with varying degree of belonging. These methods learn a hypothesis that delivers a vector $\hat{\mathbf{y}} = \big(\hat{y}_1, \ldots, \hat{y}_k\big)^T$ with entry $\hat{y}_c \in [0,1]$ being the predicted degree by which the data point belongs to the c-th cluster. Hard clustering is an extreme case of soft clustering where we enforce each degree of belonging to take only values in $\{0,1\}$. Moreover, hard clustering requires that for each data point only of the corresponding degree of belonging (one for each cluster) is non-zero.

The main focus of this chapter is on methods that require data points being represented by numeric feature vectors (see Sections 8.1 and 8.2). These methods define the similarity between data points using the Euclidean distance between their feature vectors. Some applications generate data points for which it is not obvious how to obtain numeric feature vectors such that their Euclidean distances reflect the similarity between data points. It is then desirable to use a more flexible notion of similarity which does not require to determine (useful) numeric feature vectors of data points.

Maybe the most fundamental concept to represent similarities between data points is a similarity graph. The nodes of the similarity graph are the individual data points of a dataset. Similar data points are connected by edges (links) that might be assigned some weight that quantities the amount of similarity. Section 8.3 discusses clustering methods that use a graph to represent similarities between data points.

8.1 Hard Clustering with k-means

Consider a dataset $\mathcal{D}$ which consists of m data points that are indexed by $i = 1, \ldots, m$. The data points are characterized via their numeric feature vectors $\mathbf{x}^{(i)} \in \mathbb{R}^n$, for $i = 1, \ldots, m$. It will be convenient for the following discussion if we identify a data point with its feature vector. In particular, we refer by $\mathbf{x}^{(i)}$ to the i-th data point. Hard clustering methods decompose (or cluster) the dataset into a given number k of different clusters $\mathcal{C}^{(1)}, \ldots, \mathcal{C}^{(k)}$. These methods assign each data point $\mathbf{x}^{(i)}$ to one and only one cluster $\mathcal{C}^{(c)}$ with the cluster index $c \in \{1, \ldots, k\}$.

Let us define for each data point its label $y^{(i)} \in \{1, \ldots, k\}$ as the index of the cluster to which the ith data point actually belongs to. The c-th cluster consists of all data points with $y^{(i)} = c$,

$$\mathcal{C}^{(c)} := \big\{ i \in \{1, \ldots, m\} : y^{(i)} = c \big\}. \tag{8.1}$$

We can interpret hard clustering methods as ML methods that compute predictions $\hat{y}^{(i)}$ for the ("correct") cluster assignments $y^{(i)}$. The predicted cluster assignments result in the predicted clusters

$$\widehat{\mathcal{C}}^{(c)} := \big\{ i \in \{1, \ldots, m\} : \hat{y}^{(i)} = c \big\}, \text{ for } c = 1, \ldots, k. \tag{8.2}$$

We now discuss a hard clustering method which is known as k-means. This method does not require the knowledge of the label or (true) cluster assignment $y^{(i)}$ for any data point in $\mathcal{D}$. This method computes predicted cluster assignments $\hat{y}^{(i)}$ based solely from the intrinsic geometry of the feature vectors $\mathbf{x}^{(i)} \in \mathbb{R}^n$ for all $i = 1, \ldots, m$. Since it does not require any labeled data points, k-means is often referred to as being an unsupervised method. However, note that k-means requires the number k of clusters to be given as an input (or hyper-) parameter.

The k-means method represents the c-th cluster $\widehat{\mathcal{C}}^{(c)}$ by a representative feature vector $\boldsymbol{\mu}^{(c)} \in \mathbb{R}^n$. It seems reasonable to assign data points in $\mathcal{D}$ to clusters $\widehat{\mathcal{C}}^{(c)}$ such that they are well concentrated around the cluster representatives $\boldsymbol{\mu}^{(c)}$. We make this informal requirement precise by defining the clustering error

$$\widehat{L}\big(\{\boldsymbol{\mu}^{(c)}\}_{c=1}^{k}, \{\hat{y}^{(i)}\}_{i=1}^{m} \mid \mathcal{D}\big) = (1/m) \sum_{i=1}^{m} \Big\| \mathbf{x}^{(i)} - \boldsymbol{\mu}^{(\hat{y}^{(i)})} \Big\|^2. \tag{8.3}$$

Note that the clustering error $\widehat{L}$ (8.3) depends on both, the cluster assignments $\hat{y}^{(i)}$, which

define the cluster (8.2), and the cluster representatives $\boldsymbol{\mu}^{(c)}$, for $c = 1, \ldots, k$.

Finding the optimal cluster means $\{\boldsymbol{\mu}^{(c)}\}_{c=1}^{k}$ and cluster assignments $\{\hat{y}^{(i)}\}_{i=1}^{m}$ that minimize the clustering error (8.3) is computationally challenging. The difficulty stems from the fact that the clustering error is a non-convex function of the cluster means and assignments. While jointly optimizing the cluster means and assignments is hard, separately optimizing either the cluster means for given assignments or vice-versa is easy. In what follows, we present simple closed-form solutions for these sub-problems. The k-means method simply combines these solutions in an alternating fashion.

It can be shown that for given predictions (cluster assignments) $\hat{y}^{(i)}$, the clustering error (8.3) is minimized by setting the cluster representatives equal to the **cluster means** [12]

$$\boldsymbol{\mu}^{(c)} := \big(1/|\widehat{\mathcal{C}}^{(c)}|\big) \sum_{\hat{y}^{(i)}=c} \mathbf{x}^{(i)}. \tag{8.4}$$

To evaluate (8.4) we need to know the predicted cluster assignments $\hat{y}^{(i)}$. The crux is that the optimal predictions $\hat{y}^{(i)}$, in the sense of minimizing clustering error (8.3), depend themselves on the choice for the cluster representatives $\boldsymbol{\mu}^{(c)}$. In particular, for given cluster representative $\boldsymbol{\mu}^{(c)}$ with $c = 1, \ldots, k$, the clustering error is minimized by the cluster assignments

$$\hat{y}^{(i)} \in \operatorname*{argmin}_{c \in \{1,\ldots,k\}} \big\|\mathbf{x}^{(i)} - \boldsymbol{\mu}^{(c)}\big\|. \tag{8.5}$$

Here, we denote by $\operatorname*{argmin}_{c' \in \{1,\ldots,k\}} \|\mathbf{x}^{(i)} - \boldsymbol{\mu}^{(c')}\|$ the set of all cluster indices $c \in \{1, \ldots, k\}$ such that $\|\mathbf{x}^{(i)} - \boldsymbol{\mu}^{(c)}\| = \min_{c' \in \{1,\ldots,k\}} \|\mathbf{x}^{(i)} - \boldsymbol{\mu}^{(c')}\|$.

Note that (8.5) assigns the ith datapoint to those cluster $\mathcal{C}^{(c)}$ whose cluster mean $\boldsymbol{\mu}^{(c)}$ is nearest (in Euclidean distance) to $\mathbf{x}^{(i)}$. Thus, if we knew the optimal cluster representatives, we could predict the cluster assignments using (8.5). However, we do not know the optimal cluster representatives unless we have found good predictions for the cluster assignments $\hat{y}^{(i)}$ (see (8.4)).

To recap: We have characterized the optimal choice (8.4) for the cluster representatives for given cluster assignments and the optimal choice (8.5) for the cluster assignments for given cluster representatives. It seems natural, starting from some initial guess for the cluster representatives, to alternate between the cluster assignment update (8.5) and the update (8.4) for the cluster means. This alternating optimization strategy is illustrated in Figure 8.2 and summarized in Algorithm 12. Note that Algorithm 12, which is maybe the most basic variant of k-means, simply alternates between the two updates (8.4) and (8.5)

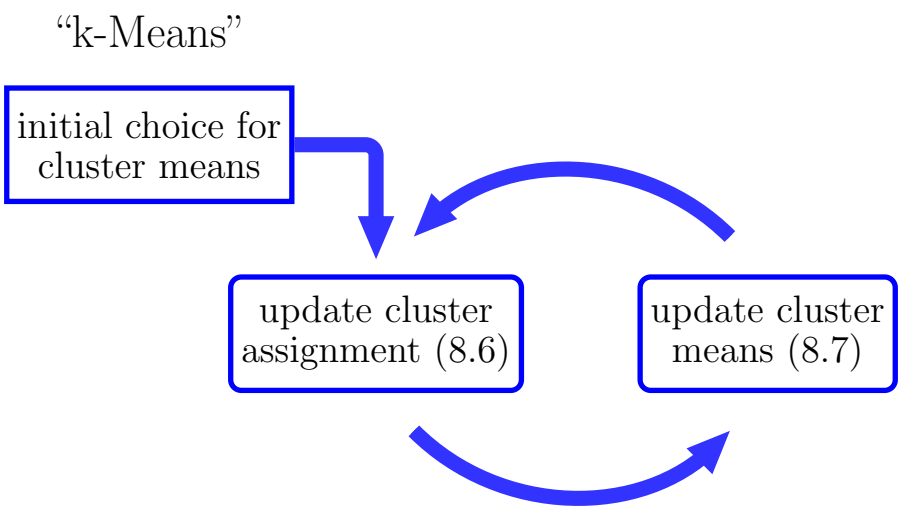

Figure 8.2: The workflow of k-means. Starting from an initial guess or estimate for the cluster means, the cluster assignments and cluster means are updated (improved) in an alternating fashion.

until some stopping criterion is satisfied.

Algorithm 12 requires the specification of the number k of clusters and initial choices for the cluster means $\boldsymbol{\mu}^{(c)}$, for $c = 1, \ldots, k$. Those quantities are hyper-parameters that must be tuned to the specific geometry of the given dataset $\mathcal{D}$. This tuning can be based on probabilistic models for the dataset and its cluster structure (see Section 2.1.4 and [81, 150]). Alternatively, if Algorithm 12 is used as pre-processing within an overall supervised ML method (see Chapter 3), the validation error (see Section 6.3) of the overall method might guide the choice of the number k of clusters.

Choosing Number of Clusters. The choice for the number k of clusters typically depends on the role of the clustering method within an overall ML application. If the clustering method serves as a pre-processing for a supervised ML problem, we could try out different values of the number k and determine, for each choice k, the corresponding validation error. We then pick the value of k which results in the smallest validation error. If the clustering method is mainly used as a tool for data visualization, we might prefer a small number of clusters. The choice for the number k of clusters can also be guided by the so-called "elbow-method". Here, we run the k-means Algorithm 12 for several different choices of k. For each value of k, Algorithm 12 delivers a clustering with clustering error

$$E^{(k)} = \widehat{L}\big(\{\boldsymbol{\mu}^{(c)}\}_{c=1}^{k}, \{\hat{y}^{(i)}\}_{i=1}^{m} \mid \mathcal{D}\big).$$

Algorithm 12 "k-means"

Input: dataset $\mathcal{D} = \{\mathbf{x}^{(i)}\}_{i=1}^{m}$; number k of clusters; initial cluster means $\boldsymbol{\mu}^{(c)}$ for $c = 1, \ldots, k$.

1: **repeat**

2: for each datapoint $\mathbf{x}^{(i)}$, $i=1, \ldots, m$, do

$$\hat{y}^{(i)} := \operatorname*{argmin}_{c' \in \{1,\ldots,k\}} \|\mathbf{x}^{(i)} - \boldsymbol{\mu}^{(c')}\| \quad \text{(update cluster assignments)} \tag{8.6}$$

3: for each cluster $c=1, \ldots, k$ do

$$\boldsymbol{\mu}^{(c)} := \frac{1}{|\{i : \hat{y}^{(i)} = c\}|} \sum_{i:\hat{y}^{(i)}=c} \mathbf{x}^{(i)} \quad \text{(update cluster means)} \tag{8.7}$$

4: **until** stopping criterion is met

5: compute final clustering error $E^{(k)} := (1/m) \sum_{i=1}^{m} \left\| \mathbf{x}^{(i)} - \boldsymbol{\mu}^{(\hat{y}^{(i)})} \right\|^2$

Output: cluster means $\boldsymbol{\mu}^{(c)}$, for $c = 1, \ldots, k$, cluster assignments $\hat{y}^{(i)} \in \{1, \ldots, k\}$, for $i = 1, \ldots, m$, final clustering error $E^{(k)}$

We then plot the minimum empirical error $E^{(k)}$ as a function of the number k of clusters. Figure 8.3 depicts an example for such a plot which typically starts with a steep decrease for increasing k and then flattening out for larger values of k. Note that for $k \geq m$ we can achieve zero clustering error since each datapoint $\mathbf{x}^{(i)}$ can be assigned to a separate cluster $\mathcal{C}^{(c)}$ whose mean coincides with that datapoint, $\mathbf{x}^{(i)} = \boldsymbol{\mu}^{(c)}$.

Cluster-Means Initialization. We briefly mention some popular strategies for choosing the initial cluster means in Algorithm 12. One option is to initialize the cluster means with realizations of i.i.d. random vectors whose probability distribution is matched to the dataset $\mathcal{D} = \{\mathbf{x}^{(i)}\}_{i=1}^{m}$ (see Section 3.12). For example, we could use a multivariate normal distribution $\mathcal{N}(\mathbf{x}; \widehat{\boldsymbol{\mu}}, \widehat{\boldsymbol{\Sigma}})$ with the sample mean $\widehat{\boldsymbol{\mu}} = (1/m) \sum_{i=1}^{m} \mathbf{x}^{(i)}$ and the sample covariance $\widehat{\boldsymbol{\Sigma}} = (1/m) \sum_{i=1}^{m} (\mathbf{x}^{(i)} - \widehat{\boldsymbol{\mu}})(\mathbf{x}^{(i)} - \widehat{\boldsymbol{\mu}})^T$. Alternatively, we could choose the initial cluster means $\boldsymbol{\mu}^{(c)}$ by selecting k different data points $\mathbf{x}^{(i)}$ from $\mathcal{D}$. This selection process might combine random choices with an optimization of the distances between cluster means [3]. Finally, the cluster means might also be chosen by evenly partitioning the principal component of the dataset (see Chapter 9).

Interpretation as ERM. For a practical implementation of Algorithm 12 we need to decide when to stop updating the cluster means and assignments (see (8.6) and (8.7)). To this end it is useful to interpret Algorithm 12 as a method for iteratively minimizing the

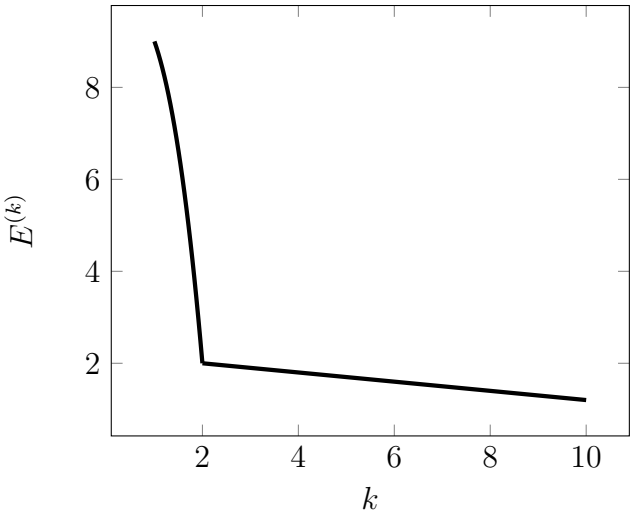

Figure 8.3: The clustering error $E^{(k)}$ achieved by k-means for increasing number k of clusters.

clustering error (8.3). As can be verified easily, the updates (8.6) and (8.7) always modify (update) the cluster means or assignments in such a way that the clustering error (8.3) is never increased. Thus, each new iteration of Algorithm 12 results in cluster means and assignments with a smaller (or the same) clustering error compared to the cluster means and assignments obtained after the previous iteration. Algorithm 12 implements a form of ERM (see Chapter 4) using the clustering error (8.3) as the empirical risk incurred by the predicted cluster assignments $\hat{y}^{(i)}$. Note that after completing a full iteration of Algorithm 12, the cluster means $\{\boldsymbol{\mu}^{(c)}\}_{c=1}^{k}$ are fully determined by the cluster assignments $\{\hat{y}^{(i)}\}_{i=1}^{m}$ via (8.7). It seems natural to terminate Algorithm 12 if the decrease in the clustering error achieved by the most recent iteration is below a prescribed (small) threshold.

Clustering and Classification. There is a strong conceptual link between Algorithm 12 and classification methods (see e.g. Section 3.13). Both methods essentially learn a hypothesis $h(\mathbf{x})$ that maps the feature vector $\mathbf{x}$ to a predicted label $\hat{y} = h(\mathbf{x})$ from a finite set. The practical meaning of the label values is different for Algorithm 12 and classification methods. For classification methods, the meaning of the label values is essentially defined by the training set (of labeled data points) used for ERM (4.3). On the other hand, clustering methods use the predicted label $\hat{y} = h(\mathbf{x})$ as a cluster index.

Another main difference between Algorithm 12 and most classification methods is the choice for the empirical risk used to evaluate the quality or usefulness of a given hypothesis $h(\cdot)$. Classification methods typically use an average loss over labeled data points in a

training set as empirical risk. In contrast, Algorithm 12 uses the clustering error (8.3) as a form of empirical risk. Consider a hypothesis that resembles the cluster assignments $\hat{y}^{(i)}$ obtained after completing an iteration in Algorithm 12, $\hat{y}^{(i)} = h\big(\mathbf{x}^{(i)}\big)$. Then we can rewrite the resulting clustering error achieved after this iteration as

$$\widehat{L}\big(h|\mathcal{D}\big) = (1/m)\sum_{i=1}^{m}\left\|\mathbf{x}^{(i)} - \frac{\sum_{i' \in \mathcal{D}^{(i)}} \mathbf{x}^{(i')}}{\big|\mathcal{D}^{(i)}\big|}\right\|^2 . \text{ with } \mathcal{D}^{(i)} := \big\{i' : h\big(\mathbf{x}^{(i)}\big) = h\big(\mathbf{x}^{(i')}\big)\big\}. \quad (8.8)$$

Note that the i-th summand in (8.8) depends on the entire dataset $\mathcal{D}$ and not only on (the features of) the i-th data point $\mathbf{x}^{(i)}$.

Some Practicalities. For a practical implementation of Algorithm 12 we need to fix three issues.

- Issue 1 ("tie-breaking"): We need to specify what to do if several different cluster indices $c \in \{1, \ldots, k\}$ achieve the minimum value in the cluster assignment update (8.6) during step 2.

- Issue 2 ("empty cluster"): The cluster assignment update (8.6) in step 3 of Algorithm 12 might result in a cluster c with no datapoints associated with it, $|\{i : \hat{y}^{(i)} = c\}| = 0$. For such a cluster c, the update (8.7) is not well-defined.

- Issue 3 ("stopping criterion"): We need to specify a criterion used in step 4 of Algorithm 12 to decide when to stop iterating.

Algorithm 13 is obtained from Algorithm 12 by fixing those three issues [51]. Step 3 of Algorithm 13 solves the first issue mentioned above ("tie breaking"), arising when there are several cluster clusters whose means have minimum distance to a data point $\mathbf{x}^{(i)}$, by assigning $\mathbf{x}^{(i)}$ to the cluster with smallest cluster index (see (8.9)). Step 4 of Algorithm 13 resolves the "empty cluster" issue by computing the variables $b^{(c)} \in \{0, 1\}$ for $c = 1, \ldots, k$. The variable $b^{(c)}$ indicates if the cluster with index c is active ($b^{(c)} = 1$) or the cluster c is inactive ($b^{(c)} = 0$). The cluster c is defined to be inactive if there are no data points assigned to it during the preceding cluster assignment step (8.9). The cluster activity indicators $b^{(c)}$ allows to restrict the cluster mean updates (8.10) only to the clusters c with at least one data point $\mathbf{x}^{(i)}$. To obtain a stopping criterion, step 7 Algorithm 13 monitors the clustering error E_r incurred by the cluster means and assignments obtained after r iterations. Algorithm 13 continues updating cluster assignments (8.9) and cluster means (8.10) as long as the decrease is above a given threshold $\varepsilon \geq 0$.

Algorithm 13 "k-Means II" (slight variation of "Fixed Point Algorithm" in [51])

Input: dataset $\mathcal{D} = \{\mathbf{x}^{(i)}\}_{i=1}^m$; number k of clusters; tolerance $\varepsilon \geq 0$; initial cluster means $\{\boldsymbol{\mu}^{(c)}\}_{c=1}^k$

1: **Initialize.** set iteration counter $r := 0$; $E_0 := 0$

2: **repeat**

3: for all datapoints $i = 1, \ldots, m$,

$$\hat{y}^{(i)} := \min\{ \operatorname*{argmin}_{c' \in \{1,\ldots,k\}} \|\mathbf{x}^{(i)} - \boldsymbol{\mu}^{(c')}\| \} \quad \text{(update cluster assignments)} \tag{8.9}$$

4: for all clusters $c = 1, \ldots, k$, update the activity indicator

$$b^{(c)} := \begin{cases} 1 & \text{if } |\{i : \hat{y}^{(i)} = c\}| > 0 \\ 0 & \text{else.} \end{cases}$$

5: for all $c = 1, \ldots, k$ with $b^{(c)} = 1$,

$$\boldsymbol{\mu}^{(c)} := \frac{1}{|\{i : \hat{y}^{(i)} = c\}|} \sum_{\{i : \hat{y}^{(i)} = c\}} \mathbf{x}^{(i)} \quad \text{(update cluster means)} \tag{8.10}$$

6: $r := r + 1$ (increment iteration counter)

7: $E_r := \widehat{L}\big(\{\boldsymbol{\mu}^{(c)}\}_{c=1}^k, \{\hat{y}^{(i)}\}_{i=1}^m \mid \mathcal{D}\big)$ (evaluate clustering error (8.3))

8: **until** $r > 1$ and $E_{r-1} - E_r \leq \varepsilon$ (check for sufficient decrease in clustering error)

9: $E^{(k)} := (1/m) \sum_{i=1}^m \left\| \mathbf{x}^{(i)} - \boldsymbol{\mu}^{(\hat{y}^{(i)})} \right\|^2$ (compute final clustering error)

Output: cluster assignments $\hat{y}^{(i)} \in \{1, \ldots, k\}$, cluster means $\boldsymbol{\mu}^{(c)}$, clustering error $E^{(k)}$.

For Algorithm 13 to be useful we must ensure that the stopping criterion is met within a finite number of iterations. In other words, we must ensure that the clustering error decrease can be made arbitrarily small within a sufficiently large (but finite) number of iterations. To this end, it is useful to represent Algorithm 13 as a fixed-point iteration

$$\{\hat{y}^{(i)}\}_{i=1}^{m} \mapsto \mathcal{P}\{\hat{y}^{(i)}\}_{m=1}^{m}. \tag{8.11}$$

The operator $\mathcal{P}$, which depends on the dataset $\mathcal{D}$, reads in a list of cluster assignments and delivers an improved list of cluster assignments aiming at reducing the associated clustering error (8.3). Each iteration of Algorithm 13 updates the cluster assignments $\hat{y}^{(i)}$ by applying the operator $\mathcal{P}$. Representing Algorithm 13 as a fixed-point iteration (8.11) allows for an elegant proof of the convergence of Algorithm 13 within a finite number of iterations (even for $\varepsilon = 0$) [51, Thm. 2].

Figure 8.4 depicts the evolution of the cluster assignments and cluster means during the iterations Algorithm 13. Each subplot corresponds to one iteration of Algorithm 13 and depicts the cluster means before that iteration and the clustering assignments (via the marker symbols) after the corresponding iteration. In particular, the upper left subplot depicts the cluster means before the first iteration (which are the initial cluster means) and the cluster assignments obtained after the first iteration of Algorithm 13.

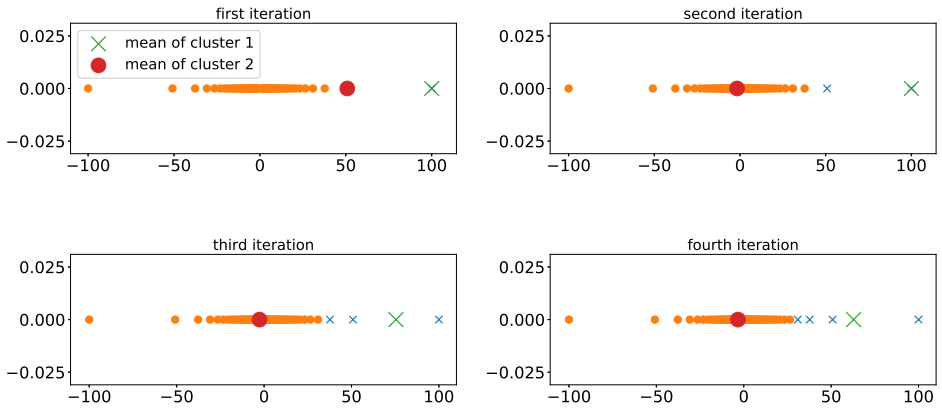

Figure 8.4: The evolution of cluster means (8.7) and cluster assignments (8.6) (depicted as large dot and large cross) during the first four iterations of k-means Algorithm 13.

Consider running Algorithm 13 with tolerance $\varepsilon = 0$ (see step 8) such that the iterations are continued until there is no decrease in the clustering error $E^{(r)}$ (see step 7 of Algorithm

13). As discussed above, Algorithm 13 will terminate after a finite number of iterations. Moreover, for $\varepsilon = 0$, the delivered cluster assignments $\big\{\hat{y}^{(i)}\big\}_{i=1}^{m}$ are fully determined by the delivered clustered means $\big\{\boldsymbol{\mu}^{(c)}\big\}_{c=1}^{k}$,

$$\hat{y}^{(i)} = \min\{ \operatorname*{argmin}_{c' \in \{1,\ldots,k\}} \|\mathbf{x}^{(i)} - \boldsymbol{\mu}^{(c')}\|\}. \tag{8.12}$$

Indeed, if (8.12) does not hold one can show the final iteration r would still decrease the clustering error and the stopping criterion in step 8 would not be met.

If cluster assignments and cluster means satisfy the condition (8.12), we can rewrite the clustering error (8.3) as a function of the cluster means solely,

$$\widehat{L}\big(\big\{\boldsymbol{\mu}^{(c)}\big\}_{c=1}^{k}|\mathcal{D}\big) := (1/m)\sum_{i=1}^{m} \min_{c' \in \{1,\ldots,k\}} \|\mathbf{x}^{(i)} - \boldsymbol{\mu}^{(c')}\|^2. \tag{8.13}$$

Even for cluster assignments and cluster means that do not satisfy (8.12), we can still use (8.13) to lower bound the clustering error (8.3),

$$\widehat{L}\big(\big\{\boldsymbol{\mu}^{(c)}\big\}_{c=1}^{k}|\mathcal{D}\big) \leq \widehat{L}\big(\{\boldsymbol{\mu}^{(c)}\}_{c=1}^{k}, \{\hat{y}^{(i)}\}_{i=1}^{m} \mid \mathcal{D}\big)$$

.

Algorithm 13 iteratively improves the cluster means in order to minimize (8.13). Ideally, we would like Algorithm 13 to deliver cluster means that achieve the global minimum of (8.13) (see Figure 8.5). However, for some combination of dataset $\mathcal{D}$ and initial cluster means, Algorithm 13 delivers cluster means that form only a local optimum of $\widehat{L}\big(\big\{\boldsymbol{\mu}^{(c)}\big\}_{c=1}^{k}|\mathcal{D}\big)$ which is strictly worse (larger) than its global optimum (see Figure 8.5).

The tendency of Algorithm 13 to get trapped around a local minimum of (8.13) depends on the initial choice for cluster means. It is therefore useful to repeat Algorithm 13 several times, with each repetition using a different initial choice for the cluster means. We then pick the cluster assignments $\{\hat{y}^{(i)}\}_{i=1}^{m}$ obtained for the repetition that resulted in the smallest clustering error $E^{(k)}$ (see step 9).

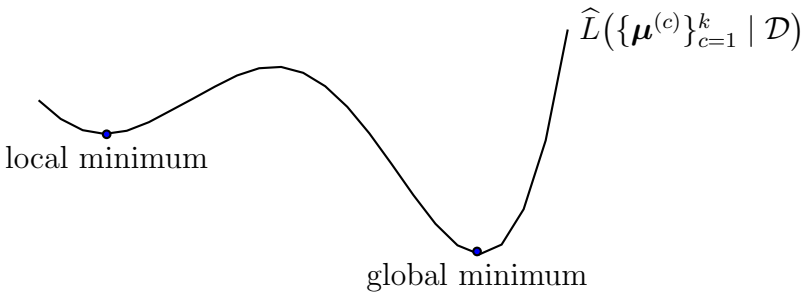

Figure 8.5: The clustering error (8.13) is a non-convex function of the cluster means $\{\boldsymbol{\mu}^{(c)}\}_{c=1}^{k}$. Algorithm 13 iteratively updates cluster means to minimize the clustering error but might get trapped around one of its local minimum.

8.2 Soft Clustering with Gaussian Mixture Models

Consider a dataset $\mathcal{D} = \{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)}\}$ that we wish to group into a given number of k different clusters. The hard clustering methods discussed in Section 8.1 deliver cluster assignments $\hat{y}^{(i)}$, for $i = 1, \ldots, m$. The cluster assignment $\hat{y}^{(i)}$ is the index of the cluster to which the ith data point $\mathbf{x}^{(i)}$ is assigned to. These cluster assignments $\hat{y}$ provide rather coarse-grained information. Two data points $\mathbf{x}^{(i)}, \mathbf{x}^{(i')}$ might be assigned to the same cluster c although their distances to the cluster mean $\boldsymbol{\mu}^{(c)}$ might differ significantly. Intuitively, these two data points have a different degree of belonging to the cluster c.

For some clustering applications it is desirable to quantify the degree by which a data point belongs to a cluster. Soft clustering methods use a continues range, such as the closed interval $[0, 1]$, of possible values for the degree of belonging. In contrast, hard clustering methods use only two possible values for the degree of belonging to a specific cluster, either "full belonging" or no "belonging at all". While hard clustering methods assign a given data point to precisely one cluster, soft clustering methods typically assign a data point to several different clusters with non-zero degree of belonging.

This chapter discusses soft clustering methods that compute, for each data point $\mathbf{x}^{(i)}$ in the dataset $\mathcal{D}$, a vector $\widehat{\mathbf{y}}^{(i)} = \big(\hat{y}_1^{(i)}, \ldots, \hat{y}_k^{(i)}\big)^T$. We can interpret the entry $\hat{y}_c^{(i)} \in [0, 1]$ as the degree by which the data point $\mathbf{x}^{(i)}$ belongs to the cluster $\mathcal{C}^{(c)}$. For $\hat{y}_c^{(i)} \approx 1$, we are quite confident in the data point $\mathbf{x}^{(i)}$ belonging to cluster $\mathcal{C}^{(c)}$. In contrast, for $\hat{y}_c^{(i)} \approx 0$, we are quite confident that the data point $\mathbf{x}^{(i)}$ is outside the cluster $\mathcal{C}^{(c)}$.

A widely used soft clustering method uses a probabilistic model for the data points $\mathcal{D} = \{\mathbf{x}^{(i)}\}_{i=1}^{m}$. Within this model, each cluster $\mathcal{C}^{(c)}$, for $c = 1, \ldots, k$, is represented by a

multivariate normal distribution [9]

$$\mathcal{N}\big(\mathbf{x};\boldsymbol{\mu}^{(c)},\boldsymbol{\Sigma}^{(c)}\big)=\frac{1}{\sqrt{\det\{2\pi\boldsymbol{\Sigma}\}}}\exp\big(-(1/2)\big(\mathbf{x}-\boldsymbol{\mu}^{(c)}\big)^{T}\big(\boldsymbol{\Sigma}^{(c)}\big)^{-1}\big(\mathbf{x}-\boldsymbol{\mu}^{(c)}\big)\big). \tag{8.14}$$

The probability distribution (8.14) is parametrized by a cluster-specific mean vector $\boldsymbol{\mu}^{(c)}$ and an (invertible) cluster-specific covariance matrix $\boldsymbol{\Sigma}^{(c)}$.[1] Let us interpret a specific data point $\mathbf{x}^{(i)}$ as a realization drawn from the probability distribution (8.14) of a specific cluster $c^{(i)}$,

$$\mathbf{x}^{(i)} \sim \mathcal{N}\big(\mathbf{x};\boldsymbol{\mu}^{(c)},\boldsymbol{\Sigma}^{(c)}\big) \text{ with cluster index } c=c^{(i)}. \tag{8.15}$$

We can think of $c^{(i)}$ as the true index of the cluster to which the data point $\mathbf{x}^{(i)}$ belongs to. The variable $c^{(i)}$ selects the cluster distributions (8.14) from which the feature vector $\mathbf{x}^{(i)}$ has been generated (drawn). We will therefore refer to the variable $c^{(i)}$ as the (true) cluster assignment for the ith data point. Similar to the feature vectors $\mathbf{x}^{(i)}$ we also interpret the cluster assignments $c^{(i)}$, for $i=1,\ldots,m$ as realizations of i.i.d. RVs.

In contrast to the feature vectors $\mathbf{x}^{(i)}$, we do not observe (know) the true cluster indices $c^{(i)}$. After all, the goal of soft clustering is to estimate the cluster indices $c^{(i)}$. We obtain a soft clustering method by estimating the cluster indices $c^{(i)}$ based solely on the data points in $\mathcal{D}$. To compute these estimates we assume that the (true) cluster indices $c^{(i)}$ are realizations of i.i.d. RVs with the common probability distribution (or probability mass function)

$$p_c := p(c^{(i)}=c) \text{ for } c=1,\ldots,k. \tag{8.16}$$

The (prior) probabilities p_c, for $c=1,\ldots,k$, are either assumed known or estimated from data [85, 9]. The choice for the probabilities p_c could reflect some prior knowledge about different sizes of the clusters. For example, if cluster $\mathcal{C}^{(1)}$ is known to be larger than cluster $\mathcal{C}^{(2)}$, we might choose the prior probabilities such that $p_1 > p_2$.

The probabilistic model given by (8.15), (8.16) is referred to as a a GMM. As its name suggests, within a GMM the common marginal distribution for the feature vectors $\mathbf{x}^{(i)}$, for $i=1,\ldots,m$, is a (additive) mixture of multivariate normal (Gaussian) distributions,

$$p(\mathbf{x}^{(i)}) = \sum_{c=1}^{k} \underbrace{p(c^{(i)}=c)}_{p_c} \underbrace{p(\mathbf{x}^{(i)}|c^{(i)}=c)}_{\mathcal{N}(\mathbf{x}^{(i)};\boldsymbol{\mu}^{(c)},\boldsymbol{\Sigma}^{(c)})}. \tag{8.17}$$

[1] Note that the expression (8.14) is only valid for an invertible (non-singular) covariance matrix $\boldsymbol{\Sigma}$.

As already mentioned, the cluster assignments $c^{(i)}$ are hidden (unobserved) RVs. We thus have to infer or estimate these variables from the observed data points $\mathbf{x}^{(i)}$ which realizations or i.i.d. RVs with the common distribution (8.17).

The GMM (see (8.15) and (8.16)) lends naturally to a rigorous definition for the degree $y_c^{(i)}$ by which data point $\mathbf{x}^{(i)}$ belongs to cluster c.[2] Let us define the label value $y_c^{(i)}$ as the "a-posteriori" probability of the cluster assignment $c^{(i)}$ being equal to $c \in \{1, \ldots, k\}$:

$$y_c^{(i)} := p(c^{(i)} = c|\mathcal{D}). \tag{8.18}$$

By their very definition (8.18), the degrees of belonging $y_c^{(i)}$ always sum to one,

$$\sum_{c=1}^{k} y_c^{(i)} = 1 \text{ for each } i = 1, \ldots, m. \tag{8.19}$$

We emphasize that we use the conditional cluster probability (8.18), conditioned on the dataset $\mathcal{D}$, for defining the degree of belonging $y_c^{(i)}$. This is reasonable since the degree of belonging $y_c^{(i)}$ depends on the overall (cluster) geometry of the dataset $\mathcal{D}$.

The definition (8.18) for the label values (degree of belongings) $y_c^{(i)}$ involves the GMM parameters $\{\boldsymbol{\mu}^{(c)}, \boldsymbol{\Sigma}^{(c)}, p_c\}_{c=1}^k$ (see (8.17)). Since we do not know these parameters beforehand we cannot evaluate the conditional probability in (8.18). A principled approach to solve this problem is to evaluate (8.18) with the true GMM parameters replaced by some estimates $\{\widehat{\boldsymbol{\mu}}^{(c)}, \widehat{\boldsymbol{\Sigma}}^{(c)}, \hat{p}_c\}_{c=1}^k$. Plugging in the GMM parameter estimates into (8.18) provides us with predictions $\hat{y}_c^{(i)}$ for the degrees of belonging. However, to compute the GMM parameter estimates we would have already needed the degrees of belonging $y_c^{(i)}$. This situation is similar to hard clustering where ultime goals is to jointly optimize cluster means and assignments (see Section 8.1).

Similar to the spirit of Algorithm 12 for hard clustering, we solve the above dilemma of soft clustering by an alternating optimization scheme. This scheme, which is illustrated in Figure 8.6, alternates between updating (optimizing) the predicted degrees of belonging (or soft cluster assignments) $\hat{y}_c^{(i)}$, for $i = 1, \ldots, m$ and $c = 1, \ldots, k$, given the current GMM parameter estimates $\{\widehat{\boldsymbol{\mu}}^{(c)}, \widehat{\boldsymbol{\Sigma}}^{(c)}, \hat{p}_c\}_{c=1}^k$ and then updating (optimizing) these GMM parameter estimates based on the updated predictions $\hat{y}_c^{(i)}$. We summarize the resulting soft clustering method in Algorithm 14. Each iteration of Algorithm 14 consists of an update

[2]Remember that the degree of belonging $y_c^{(i)}$ is considered as the (unknown) label value of a data point. The choice or definition for the labels of data points is a design choice. In particular, we can define the labels of data points using a hypothetical probabilistic model such as the GMM.

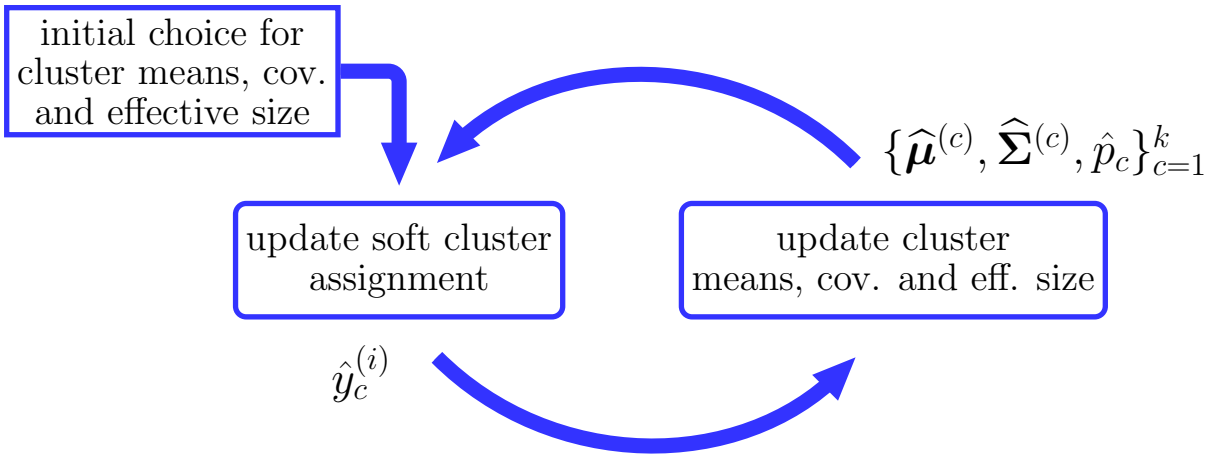

Figure 8.6: The workflow of the soft clustering Algorithm 14. Starting from an initial guess or estimate for the cluster parameters, the soft cluster assignments and cluster parameters are updated (improved) in an alternating fashion.

(8.22) for the degrees of belonging followed by an update (step 3) for the GMM parameters.

To analyze Algorithm 14 it is helpful to interpret (the features of) data points $\mathbf{x}^{(i)}$ as realizations of i.i.d. RVs distributed according to a GMM (8.15)-(8.16). We can then understand Algorithm 14 as a method for estimating the GMM parameters based on observing realizations drawn from the GMM (8.15)-(8.16). We can estimate the parameters of a probability distribution using the maximum likelihood method (see Section 3.12 and [76, 85]). As its name suggests, maximum likelihood methods estimate the GMM parameters by maximizing the probability (density)

$$p\big(\mathcal{D}; \{\boldsymbol{\mu}^{(c)}, \boldsymbol{\Sigma}^{(c)}, p_c\}_{c=1}^{k}\big) \tag{8.20}$$

of actually observing the data points in the dataset $\mathcal{D}$.

It can be shown that Algorithm 14 is an instance of a generic approximate maximum likelihood technique referred to as expectation maximization (EM) (see [58, Chap. 8.5] for more details). In particular, each iteration of Algorithm 14 updates the GMM parameter estimates such that the corresponding probability density (8.20) does not decrease [157]. If we denote the GMM parameter estimate obtained after r iterations of Algorithm 14 by $\boldsymbol{\theta}^{(r)}$ [58, Sec. 8.5.2],

$$p\big(\mathcal{D}; \boldsymbol{\theta}^{(r+1)}\big) \geq p\big(\mathcal{D}; \boldsymbol{\theta}^{(r)}\big) \tag{8.21}$$

As for Algorithm 12, we can also interpret Algorithm 14 as an instance of the ERM principle discussed in Chapter 4. Indeed, maximizing the probability density (8.20) is equivalent

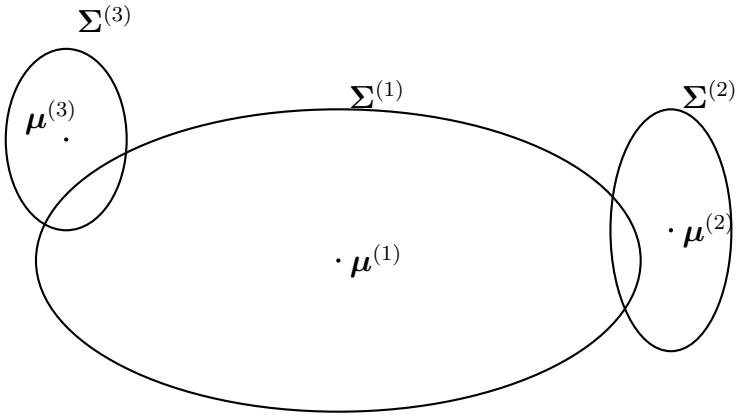

Figure 8.7: The GMM (8.15), (8.16) results in a probability distribution (8.17) for (feature vectors of) data points which is a weighted sum of multivariate normal distributions $\mathcal{N}(\boldsymbol{\mu}^{(c)}, \boldsymbol{\Sigma}^{(c)})$. The weight of the c-th component is the cluster probability $p(c^{(i)} = c)$.

Algorithm 14 "A Soft-Clustering Algorithm" [12]

Input: dataset $\mathcal{D} = \{\mathbf{x}^{(i)}\}_{i=1}^m$; number k of clusters, initial GMM parameter estimates $\{\widehat{\boldsymbol{\mu}}^{(c)}, \widehat{\boldsymbol{\Sigma}}^{(c)}, \hat{p}_c\}_{c=1}^k$

1: **repeat**

2: for each $i = 1, \ldots, m$ and $c = 1, \ldots, k$, update degrees of belonging

$$\hat{y}_c^{(i)} := \frac{\hat{p}_c \mathcal{N}(\mathbf{x}^{(i)}; \widehat{\boldsymbol{\mu}}^{(c)}, \widehat{\boldsymbol{\Sigma}}^{(c)})}{\sum_{c'=1}^k \hat{p}_{c'} \mathcal{N}(\mathbf{x}^{(i)}; \widehat{\boldsymbol{\mu}}^{(c')}, \widehat{\boldsymbol{\Sigma}}^{(c')})} \tag{8.22}$$

3: for each $c \in \{1, \ldots, k\}$, update GMM parameter estimates:

- $\hat{p}_c := m_c/m$ with effective cluster size $m_c := \sum_{i=1}^m \hat{y}_c^{(i)}$ (cluster probability)
- $\widehat{\boldsymbol{\mu}}^{(c)} := (1/m_c) \sum_{i=1}^m \hat{y}_c^{(i)} \mathbf{x}^{(i)}$ (cluster mean)
- $\widehat{\boldsymbol{\Sigma}}^{(c)} := (1/m_c) \sum_{i=1}^m \hat{y}_c^{(i)} \big(\mathbf{x}^{(i)} - \widehat{\boldsymbol{\mu}}^{(c)}\big)\big(\mathbf{x}^{(i)} - \widehat{\boldsymbol{\mu}}^{(c)}\big)^T$ (cluster covariance matrix)

4: **until** stopping criterion met

Output: predicted degrees of belonging $\widehat{\mathbf{y}}^{(i)} = (\hat{y}_1^{(i)}, \ldots, \hat{y}_k^{(i)})^T$ for $i = 1, \ldots, m$.

to minimizing the empirical risk

$$\widehat{L}\big(\boldsymbol{\theta} \mid \mathcal{D}\big) := -\log p\big(\mathcal{D}; \boldsymbol{\theta}\big) \text{ with GMM parameters } \boldsymbol{\theta} := \{\boldsymbol{\mu}^{(c)}, \boldsymbol{\Sigma}^{(c)}, p_c\}_{c=1}^{k} \tag{8.23}$$

The empirical risk (8.23) is the negative logarithm of the probability (density) (8.20) of observing the dataset $\mathcal{D}$ as i.i.d. realizations of the GMM (8.17). The monotone increase in the probability density (8.21) achieved by the iterations of Algorithm 14 translate into a monotone decrease of the empirical risk,

$$\widehat{L}\big(\boldsymbol{\theta}^{(r)} \mid \mathcal{D}\big) \leq \widehat{L}\big(\boldsymbol{\theta}^{(r-1)} \mid \mathcal{D}\big) \text{ with iteration counter } r. \tag{8.24}$$

The monotone decrease (8.24) in the empirical risk (8.23) achieved by the iterations of Algorithm 14 naturally lends to a stopping criterion. Let E_r denote the empirical risk (8.23) achieved by the GMM parameter estimates $\boldsymbol{\theta}^{(r)}$ obtained after r iterations in Algorithm 14. Algorithm 14 stops iterating as soon as the decrease $E_{r-1} - E_r$ achieved by the r-th iteration of Algorithm 14 falls below a given (positive) threshold $\varepsilon > 0$.

Similar to Algorithm 12, also Algorithm 14 might get trapped in local minima of the underlying empirical risk. The GMM parameters delivered by Algorithm 14 might only be a local minimum of (8.23) but not the global minimum (see Figure 8.5 for the analogous situation in hard clustering). As for hard clustering Algorithm 12, we typically repeat Algorithm 14 several times. During each repetition of Algorithm 14, we use a different (randomly chosen) initialization for the GMM parameter estimates $\boldsymbol{\theta} = \{\widehat{\boldsymbol{\mu}}^{(c)}, \widehat{\boldsymbol{\Sigma}}^{(c)}, \hat{p}_c\}_{c=1}^{k}$. Each repetition of Algorithm 14 results in a potentially different set of GMM parameter estimates and degrees of belongings $\hat{y}_c^{(i)}$. We then use the results for that repetition that achieves the smallest empirical risk (8.23).

Let us point out an interesting link between soft clustering methods based on GMM (see Algorithm 14) and hard clustering with k-means (see Algorithm 12). Consider the GMM (8.15) with prescribed cluster covariance matrices

$$\boldsymbol{\Sigma}^{(c)} = \sigma^2 \mathbf{I} \text{ for all } c \in \{1, \ldots, k\}, \tag{8.25}$$

with some given variance $\sigma^2 > 0$. We assume the cluster covariance matrices in the GMM to be given by (8.25) and therefore can replace the covariance matrix updates in Algorithm 14 with the assignment $\widehat{\boldsymbol{\Sigma}}^{(c)} := \sigma^2 \mathbf{I}$. It can be verified easily that for sufficiently small variance σ^2 in (8.25), the update (8.22) tends to enforce $\hat{y}_c^{(i)} \in \{0, 1\}$. In other words, each data point

$\mathbf{x}^{(i)}$ becomes then effectively associated with exactly one single cluster c whose cluster mean $\widehat{\boldsymbol{\mu}}^{(c)}$ is nearest to $\mathbf{x}^{(i)}$. For $\sigma^2 \to 0$, the soft clustering update (8.22) in Algorithm 14 reduces to the (hard) cluster assignment update (8.6) in k-means Algorithm 12. We can interpret Algorithm 12 as an extreme case of Algorithm 14 that is obtained by fixing the covariance matrices in the GMM to $\sigma^2\mathbf{I}$ with a sufficiently small σ^2.

Combining GMM with linear regression. Let us sketch how Algorithm 14 could be combined with linear regression methods (see Section 3.1). The idea is to first compute the degree of belongings to the clusters for each data point. We then learn separate linear predictors for each cluster using the degree of belongings as weights for the individual loss terms in the training error. To predict the label of a new data point, we first compute the predictions obtained for each cluster-specific linear hypothesis. These cluster-specific predictions are then averaged using the degree of belongings for the new data point as weights.

8.3 Connectivity-based Clustering

The clustering methods discussed in Sections 8.1 and 8.2 can only be applied to data points which are characterized by numeric feature vectors. These methods define the similarity between data points using the Euclidean distance between the feature vectors of these data points. As illustrated in Figure 8.8, these methods can only produce "Euclidean shaped" clusters that are contained either within hyper-spheres (Algorithm 12) or hyper-ellipsoids (Algorithm 14).

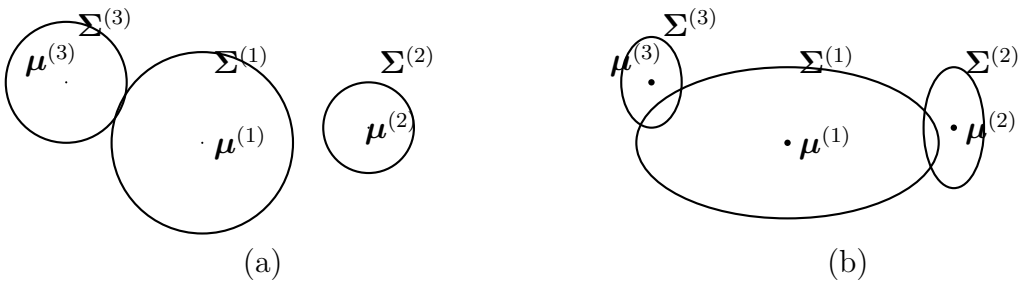

Figure 8.8: (a): Cartoon of typical cluster shapes delivered by k-means Algorithm 13. (b): Cartoon of typical cluster shapes delivered by soft clustering Algorithm 14.

Some applications generate data points for which the construction of useful numeric features is difficult. Even if we can easily obtain numeric features for data points, the

Euclidean distances between them might not reflect the actual similarities between data points. As a case in point, consider data points representing text documents. We could use the histogram of a pre-specified list (dictionary) of words as numeric features for a text document.

In general, a small Euclidean distance between histograms of text documents does not imply that the text documents have similar meanings. Moreover, clusters of similar text documents might have highly complicated shapes in the space of feature vectors that cannot be grouped within hyper-ellipsoids. For datasets with such "non-Euclidean" cluster shapes, k-means or GMM are not suitable as clustering methods. We should then replace the Euclidean distance between feature vectors with another concept to determine or measure the similarity between data points.

Connectivity-based clustering methods do not require any numeric features of data points. These methods cluster data points based on explicitly specifying for any two different data points if they are similar and to what extent. A convenient mathematical tool to represent similarities between the data points of a dataset $\mathcal{D}$ is a weighted undirected graph $\mathcal{G} = \big(\mathcal{V}, \mathcal{E}\big)$. We refer to this graph as the similarity graph of the dataset $\mathcal{D}$ (see Figure 8.9). The nodes $\mathcal{V}$ in this similarity graph $\mathcal{G}$ represent data points in $\mathcal{D}$ and the undirected edges connect nodes that represent similar data points. The extent of similarity is represented by the weights $W_{i,i'}$ for each edge $\{i, i'\} \in \mathcal{E}$.

Given a similarity graph $\mathcal{G}$ of a dataset, connectivity-based clustering methods determine clusters as subsets of nodes that are well connected within the cluster but weakly connected between different clusters. Different concepts for quantifying the connectivity between nodes in a graph yield different clustering methods:

- Spectral clustering methods use eigenvectors of a graph Laplacian matrix to measure the connectivity between nodes [148, 106].
- Flow-based clustering methods measure the connectivity between two nodes via the amount of flow that can be routed between them [73].

We can use connectivity measures between nodes of an empirical graph to construct meaningful numerical feature vectors ("embeddings") for these nodes. These feature vectors can then be fed into the hard-clustering Algorithm 13 or the soft clustering Algorithm 14 (see Figure 8.9).

The algorithm density-based spatial clustering of applications with noise (DBSCAN) considers two data points i, i' as connected if one of them (say i) is a core node and the other

node (i') can be reached via a sequence (path) of connected core nodes

$$i^{(1)}, \ldots, i^{(r)} \text{ , with } \{i, i^{(1)}\}, \{i^{(1)}, i^{(2)}\}, \ldots, \{i^{(r)}, i'\} \in \mathcal{E}.$$

DBSCAN considers a node to be a core node if it has a sufficiently large number of neighbours [35]. The minimum number of neighbours required for a node to be considered a core node is a hyper-parameter of DBSCAN. When DBSCAN is applied to data points with numeric feature vectors, it defines two data points as connected if the Euclidean distance between their feature vectors does not exceed a given threshold ε (see Figure 8.10).

In contrast to k-means and GMM, DBSCAN does not require the number of clusters to be specified. The number of clusters is determined automatically by DBSCAN and depends on its hyper-parameters. DBSCAN also performs an implicit outlier detection. The outliers delivered by DBSCAN are those data points which do not belong to the same cluster as any other data point.

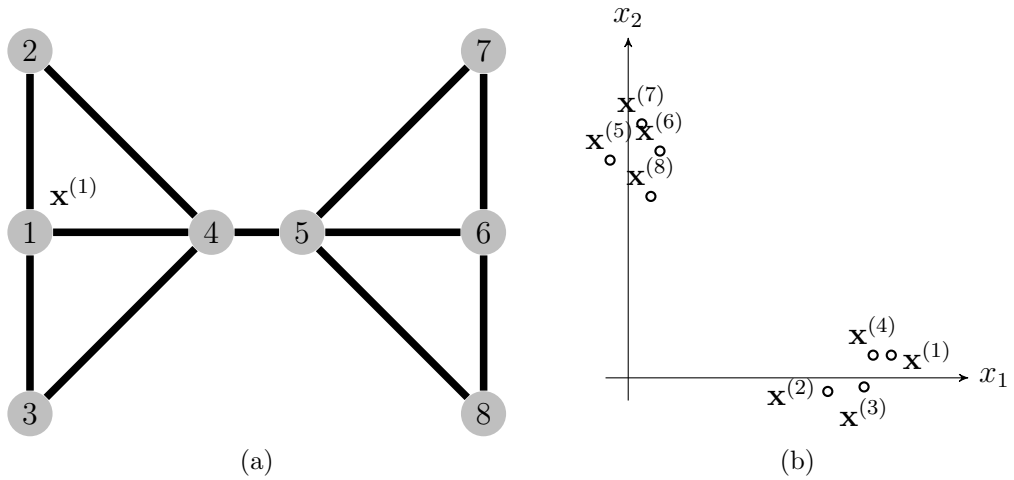

Figure 8.9: Connectivity-based clustering can be obtained by constructing features $\mathbf{x}^{(i)}$ that are (approximately) identical for well-connected data points. (a): A similarity graph for a dataset $\mathcal{D}$ consists of nodes representing individual data points and edges that connect similar data points. (b) Feature vectors of well-connected data points have small Euclidean distance.

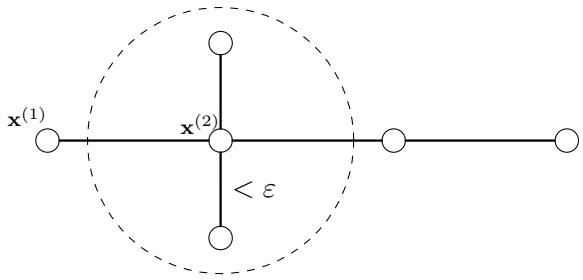

Figure 8.10: DBSCAN assigns two data points to the same cluster if they are reachable. Two data points $\mathbf{x}^{(i)}, \mathbf{x}^{(i')}$ are reachable if there is a path of data points from $\mathbf{x}^{(i')}$ to $\mathbf{x}^{(i)}$. This path consists of a sequence of data points that are within a distance of ε. Moreover, each data point on this path must be a core point which has at least a given number of neighbouring data points within the distance ε.

8.4 Clustering as Preprocessing

In applications it might be beneficial to combine clustering methods with supervised methods such as linear regression. As a point in case, consider a dataset that consists of data points obtained from two different data generation processes (such as two weather observation stations, one located in Helsinki and the other one in Lapland). Let us denote the data points generated by one process by $\mathcal{D}^{(1)}$ and the other one by $\mathcal{D}^{(2)}$. Each data point is characterized by several numeric features and a numeric label. While there would be an accurate linear hypothesis for predicting the label of data points in $\mathcal{D}^{(1)}$ and another linear hypothesis for $\mathcal{D}^{(2)}$ these two are very different. Formally, this approach amounts to using the cluster assignment of a data points as an additional feature. We could also use the degree of belonging delivered by soft clustering methods as additional features.

We could try to use clustering methods to assign any given data point to the corresponding data generation process. If we are lucky, the resulting clusters resemble (approximately) the two data generation processes $\mathcal{D}^{(1)}$ and $\mathcal{D}^{(2)}$. Once we have successfully clustered the data points, we can learn a separate (tailored) hypothesis for each cluster. More generally, we can use the predicted cluster assignments obtained from the methods of Section 8.1 - 8.3 as additional features for each data point.

Let us illustrate the above ideas by combining Algorithm 12 with linear regression. We first group data points into a given number k of clusters and then learn separate linear predictors $h^{(c)}(\mathbf{x}) = \left(\mathbf{w}^{(c)}\right)^{T}\mathbf{x}$ for each cluster $c = 1, \ldots, k$. To predict the label of a new data point with features $\mathbf{x}$, we first assign to the cluster c' with the nearest cluster mean.

We then use the linear predictor $h^{(c')}$ assigned to cluster c' to compute the predicted label $\hat{y} = h^{(c')}(\mathbf{x})$.

8.5 Exercises

Exercise 8.1. Monotonicity of k-means Updates. Show that the updates (8.7) and (8.6) for cluster means and cluster assignments, respectively, never result in an increase of the clustering error (8.3).

Exercise 8.2. How to choose k in k-means? Discuss and experiment with different strategies for choosing the number k of clusters in k-means Algorithm 13.

Exercise 8.3. Local Minima. Apply the hard clustering Algorithm 13 to the dataset $(-10, 1), (10, 1), (-10, -1), (10, -1)$ with initial cluster means $(0, 1), (0, -1)$ and tolerance $\varepsilon = 0$. For this initialization, will Algorithm 13 get trapped in a local minimum of the clustering error (8.13)?

Exercise 8.4. Image Compression with k-means. Apply k-means to image compression. Consider image pixels as data points whose features are RGB intensities. We obtain a simple image compression format by, instead of storing RGB pixel values, storing the cluster means (which are RGB triplets) and the cluster index for each pixel. Try out different values for the number k of clusters and discuss the resulting trade off between achievable reconstruction quality and storage size.

Exercise 8.5. Compression with k-means. Consider $m = 10000$ data points $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)}$ which are represented by two numeric features. We apply k-means to cluster the data set into $k = 5$ clusters. How many bits do we need to store the resulting cluster assignments?

Chapter 9

Feature Learning

"Solving Problems By Changing the Viewpoint."

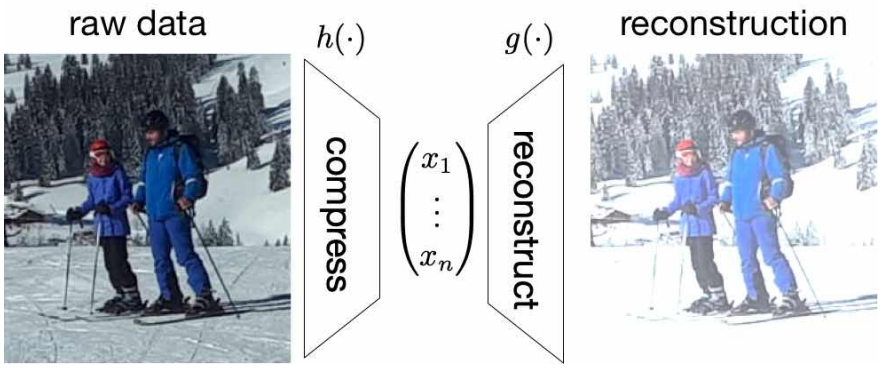

Figure 9.1: Dimensionality reduction methods aim at finding a map h which maximally compresses the raw data while still allowing to accurately reconstruct the original datapoint from a small number of features $x_1, \ldots, x_n$.

Chapter 2 defined features as those properties of a data point that can be measured or computed easily. Sometimes the choice of features follows naturally from the available hard- and software. For example, we might use the numeric measurement $z \in \mathbb{R}$ delivered by a sensing device as a feature. However, we could augment this single feature with new features such as the powers z^2 and z^3 or adding a constant $z+5$. Each of these computations produces a new feature. Which of these additional features are most useful?

Feature learning methods automate the choice of finding good features. These methods learn a hypothesis map that reads in some representation of a data point and transforms it to a set of features. Feature learning methods differ in the precise format of the original

data representation as well as the format of the delivered features. This chapter mainly discusses feature learning methods that require data points being represented by n' numeric raw features and deliver a set of n new numeric features. We will denote the raw features and the learnt new features by $\mathbf{z} = \big(z_1, \ldots, z_{n'}\big)^T \in \mathbb{R}^{n'}$ and $\mathbf{x} = \big(x_1, \ldots, x_n\big)^T \in \mathbb{R}^n$, respectively.

Many ML application domains generate data points for which can access a huge number of raw features. Consider data points being snapshots generated by a smartphone. It seems natural to use the pixel colour intensities as the raw features of the snapshot. Since modern smartphone have "Megapixel cameras", the pixel intensities would provide us with millions of raw features. It might seem a good idea to use as many raw features of a data point as possible since more features should offer more information about a data point and its label y. There are, however, two pitfalls in using an unnecessarily large number of features. The first one is a computational pitfall and the second one is a statistical pitfall.

Computationally, using very long feature vectors $\mathbf{x} \in \mathbb{R}^n$ (with n being billions), might result in prohibitive computational resource requirements (bandwidth, storage, time) of the resulting ML method. Statistically, using a large number of features makes the resulting ML methods more prone to overfitting. For example, linear regression will typically overfit when using feature vectors $\mathbf{x} \in \mathbb{R}^n$ whose length n exceeds the number m of labeled data points used for training (see Chapter 7).

Both from a computational and a statistical perspective, it is beneficial to use only the maximum necessary amount of features. The challenge is to select those features which carry most of the relevant information required for the prediction of the label y. Finding the most relevant features out of a huge number of raw features is the goal of dimensionality reduction methods. Dimensionality reduction methods form an important sub-class of feature learning methods. These methods learn a hypothesis $h(\mathbf{z})$ that maps a long raw feature vector $\mathbf{z} \in \mathbb{R}^{n'}$ to a new (short) feature vector $\mathbf{x} \in \mathbb{R}^n$ with $n \ll n'$.

Beside avoiding overfitting and coping with limited computational resources, dimensionality reduction can also be useful for data visualization. Indeed, if the resulting feature vector has length $n = 2$, we depict data points in the two-dimensional plane in form of a scatterplot.

We will discuss the basic idea underlying dimensionality reduction methods in Section 9.1. Section 9.2 presents one particular example of a dimensionality reduction method that computes relevant features by a linear transformation of the raw feature vector. Section 9.4 discusses a method for dimensionality reduction that exploits the availability of labelled data points. Section 9.6 shows how randomness can be used to obtain computationally cheap

dimensionality reduction .

Most of this chapter discusses dimensionality reduction methods that determine a small number of relevant features from a large set of raw features. However, sometimes it might be useful to go the opposite direction. There are applications where it might be beneficial to construct a large (even infinite) number of new features from a small set of raw features. Section 9.7 will showcase how computing additional features can help to improve the prediction accuracy of ML methods.

9.1 Basic Principle of Dimensionality Reduction

The efficiency of ML methods depends crucially on the choice of features that are used to characterize data points. Ideally we would like to have a small number of highly relevant features to characterize data points. If we use too many features we risk to waste computations on exploring irrelevant features. If we use too few features we might not have enough information to predict the label of a data point. For a given number n of features, dimensionality reduction methods aim at learning an (in a certain sense) optimal map from the data point to a feature vector of length n.

Figure 9.1 illustrates the basic idea of dimensionality reduction methods. Their goal is to learn (or find) a "compression" map $h(\cdot) : \mathbb{R}^{n'} \rightarrow \mathbb{R}^{n}$ that transforms a (long) raw feature vector $\mathbf{z} \in \mathbb{R}^{n'}$ to a (short) feature vector $\mathbf{x} = (x_1, \ldots, x_n)^T := h(\mathbf{z})$ (typically $n \ll n'$).

The new feature vector $\mathbf{x} = h(\mathbf{z})$ serves as a compressed representation (or code) for the original feature vector $\mathbf{z}$. We can reconstruct the raw feature vector using a reconstruction map $r(\cdot) : \mathbb{R}^{n} \rightarrow \mathbb{R}^{n'}$. The reconstructed raw features $\widehat{\mathbf{z}} := r(\mathbf{x}) = r(h(\mathbf{z}))$ will typically differ from the original raw feature vector $\mathbf{z}$. In general, we obtain a non-zero reconstruction error

$$\underbrace{\widehat{\mathbf{z}}}_{=r(h(\mathbf{z})))} - \mathbf{z}. \tag{9.1}$$

Dimensionality reduction methods learn a compression map $h(\cdot)$ such that the reconstruction error (9.1) is minimized. In particular, for a dataset $\mathcal{D} = \big\{\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(m)}\big\}$, we measure the quality of a pair of compression map h and reconstruction map r by the average reconstruction error

$$\widehat{L}\big(h, r|\mathcal{D}\big) := (1/m) \sum_{i=1}^{m} L\left(\mathbf{z}^{(i)}, r\big(h\big(\mathbf{z}^{(i)}\big)\big)\right). \tag{9.2}$$

Here, $L\left(\mathbf{z}, r\big(h\big(\mathbf{z}^{(i)}\big)\big)\right)$ denotes a loss function that is used to measure the reconstruction error $\underbrace{r\big(h\big(\mathbf{z}^{(i)}\big)\big)}_{\widehat{\mathbf{z}}} - \mathbf{z}$. Different choices for the loss function in (9.2) result in different dimensionality reduction methods. One widely-used choice for the loss is the squared Euclidean norm

$$L\left(\mathbf{z}, g\big(h\big(\mathbf{z}\big)\big)\right) := \big\|\mathbf{z} - g\big(h\big(\mathbf{z}\big)\big)\big\|_2^2. \tag{9.3}$$

Practical dimensionality reduction methods have only finite computational resources. Any practical method must therefore restrict the set of possible compression and reconstruction maps to small subsets $\mathcal{H}$ and $\mathcal{H}^*$, respectively. These subsets are the hypothesis spaces for the compression map $h \in \mathcal{H}$ and the reconstruction map $r \in \mathcal{H}^*$. Feature learning methods differ in their choice for these hypothesis spaces.

Dimensionality reduction methods learn a compression map by solving

$$\begin{aligned} \widehat{h} &= \operatorname*{argmin}_{h \in \mathcal{H}} \min_{r \in \mathcal{H}^*} \widehat{L}\big(h, r|\mathcal{D}\big) \\ &\overset{(9.2)}{=} \operatorname*{argmin}_{h \in \mathcal{H}} \min_{r \in \mathcal{H}^*} (1/m) \sum_{i=1}^{m} L\left(\mathbf{z}^{(i)}, r\big(h\big(\mathbf{z}^{(i)}\big)\big)\right). \end{aligned} \tag{9.4}$$

We can interpret (9.4) as a (typically non-linear) approximation problem. The optimal compression map $\widehat{h}$ is such that the reconstruction$r(\widehat{h}(\mathbf{z}))$, with a suitably chosen reconstruction map r, approximates the original raw feature vector $\mathbf{z}$ with optimal accuracy. Note that we use a single compression map $h(\cdot)$ and a single reconstruction map $r(\cdot)$ for all data points in the dataset $\mathcal{D}$.

We obtain variety of dimensionality methods by using different choices for the hypothesis spaces $\mathcal{H}, \mathcal{H}^*$ and loss function in (9.4). Section 9.2 discusses a method that solves (9.4) for $\mathcal{H}, \mathcal{H}^*$ constituted by linear maps and the loss (9.3). Deep autoencoders are another family of dimensionality reduction methods that solve (9.4) with $\mathcal{H}, \mathcal{H}^*$ constituted by non-linear maps that are represented by deep neural networks [48, Ch. 14].

9.2 Principal Component Analysis

We now consider the special case of dimensionality reduction where the compression and reconstruction map are required to be linear maps. Consider a data point which is characterized by a (typically very long) raw feature vector $\mathbf{z} = \big(z_1, \ldots, z_{n'}\big)^T \in \mathbb{R}^{n'}$ of length n'. The length n' of the raw feature vector might be easily of the order of millions. To obtain a small

set of relevant features $\mathbf{x} = \big(x_1, \ldots, x_n\big)^T \in \mathbb{R}^n$, we apply a linear transformation to the raw feature vector,

$$\mathbf{x} = \mathbf{W}\mathbf{z}. \tag{9.5}$$

Here, the "compression" matrix $\mathbf{W} \in \mathbb{R}^{n \times n'}$ maps (in a linear fashion) the (long) raw feature vector $\mathbf{z} \in \mathbb{R}^{n'}$ to the (shorter) feature vector $\mathbf{x} \in \mathbb{R}^n$.

It is reasonable to choose the compression matrix $\mathbf{W} \in \mathbb{R}^{n \times n'}$ in (9.5) such that the resulting features $\mathbf{x} \in \mathbb{R}^n$ allow to approximate the original data point $\mathbf{z} \in \mathbb{R}^{n'}$ as accurate as possible. We can approximate (or recover) the data point $\mathbf{z} \in \mathbb{R}^{n'}$ back from the features $\mathbf{x}$ by applying a reconstruction operator $\mathbf{R} \in \mathbb{R}^{n' \times n}$, which is chosen such that

$$\mathbf{z} \approx \mathbf{R}\mathbf{x} \overset{(9.5)}{=} \mathbf{R}\mathbf{W}\mathbf{z}. \tag{9.6}$$

The approximation error $\widehat{L}\big(\mathbf{W}, \mathbf{R} \mid \mathcal{D}\big)$ resulting when (9.6) is applied to each data point in a dataset $\mathcal{D} = \{\mathbf{z}^{(i)}\}_{i=1}^m$ is then

$$\widehat{L}\big(\mathbf{W}, \mathbf{R} \mid \mathcal{D}\big) = (1/m) \sum_{i=1}^{m} \|\mathbf{z}^{(i)} - \mathbf{R}\mathbf{W}\mathbf{z}^{(i)}\|^2. \tag{9.7}$$

One can verify that the approximation error $\widehat{L}\big(\mathbf{W}, \mathbf{R} \mid \mathcal{D}\big)$ can only by minimal if the compression matrix $\mathbf{W}$ is of the form

$$\mathbf{W} = \mathbf{W}_{\rm PCA} := \big(\mathbf{u}^{(1)}, \ldots, \mathbf{u}^{(n)}\big)^T \in \mathbb{R}^{n \times n'}, \tag{9.8}$$

with n orthonormal vectors $\mathbf{u}^{(j)}$, for $j = 1, \ldots, n$. The vectors $\mathbf{u}^{(j)}$ are the eigenvectors corresponding to the n largest eigenvalues of the sample covariance matrix

$$\mathbf{Q} := (1/m)\mathbf{Z}^T\mathbf{Z} \in \mathbb{R}^{n' \times n'}. \tag{9.9}$$

Here we used the data matrix $\mathbf{Z} = \big(\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(m)}\big)^T \in \mathbb{R}^{m \times n'}$.[1] It can be verified easily, using the definition (9.9), that the matrix $\mathbf{Q}$ is psd. As a psd matrix, $\mathbf{Q}$ has an eigenvalue

[1] Some authors define the data matrix as $\mathbf{Z} = \big(\widetilde{\mathbf{z}}^{(1)}, \ldots, \widetilde{\mathbf{z}}^{(m)}\big)^T \in \mathbb{R}^{m \times D}$ using "centered" data points $\mathbf{z}^{(i)} - \widehat{\mathbf{m}}$ obtained by subtracting the average $\widehat{\mathbf{m}} = (1/m)\sum_{i=1}^m \mathbf{z}^{(i)}$.

decomposition (EVD) of the form [135]

$$\mathbf{Q} = \big(\mathbf{u}^{(1)}, \ldots, \mathbf{u}^{(n')}\big) \begin{pmatrix} \lambda_1 & \ldots & 0 \\ 0 & \ddots & 0 \\ 0 & \ldots & \lambda_{n'} \end{pmatrix} \big(\mathbf{u}^{(1)}, \ldots, \mathbf{u}^{(n')}\big)^T \tag{9.10}$$

with real-valued eigenvalues $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_{n'} \geq 0$ and orthonormal eigenvectors $\{\mathbf{u}^{(j)}\}_{j=1}^{n'}$.

The feature vectors $\mathbf{x}^{(i)}$ are obtained by applying the compression matrix $\mathbf{W}_{\rm PCA}$ (9.8) to the raw feature vectors $\mathbf{z}^{(i)}$. We refer to the entries of the learnt feature vector $\mathbf{x}^{(i)} = \mathbf{W}_{\rm PCA}\mathbf{z}^{(i)}$ (see (9.8)) as the principal components (PC) of the raw feature vectors $\mathbf{z}^{(i)}$. Algorithm 15 summarizes the overall procedure of determining the compression matrix (9.8) and computing the learnt feature vectors $\mathbf{x}^{(i)}$. This procedure is known as PCA. The length

Algorithm 15 PCA

Input: dataset $\mathcal{D} = \{\mathbf{z}^{(i)} \in \mathbb{R}^{n'}\}_{i=1}^{m}$; number n of PCs.

1: compute the EVD (9.10) to obtain orthonormal eigenvectors $\big(\mathbf{u}^{(1)}, \ldots, \mathbf{u}^{(n')}\big)$ corresponding to (decreasingly ordered) eigenvalues $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_{n'} \geq 0$

2: construct compression matrix $\mathbf{W}_{\rm PCA} := \big(\mathbf{u}^{(1)}, \ldots, \mathbf{u}^{(n)}\big)^T \in \mathbb{R}^{n \times n'}$

3: compute feature vector $\mathbf{x}^{(i)} = \mathbf{W}_{\rm PCA}\mathbf{z}^{(i)}$ whose entries are PC of $\mathbf{z}^{(i)}$

4: compute approximation error $\widehat{L}^{(\rm PCA)} = \sum_{j=n+1}^{n'} \lambda_j$ (see (9.11)).

Output: $\mathbf{x}^{(i)}$, for $i = 1, \ldots, m$, and the approximation error $\widehat{L}^{(\rm PCA)}$.

$n \in \{0, \ldots, n')$ of the delivered feature vectors $\mathbf{x}^{(i)}$, for $i = 1, \ldots, m$, is an input (or hyper-) parameter of Algorithm 15. Two extreme cases are $n = 0$ (maximum compression) and $n = n'$ (no compression). We finally note that the choice for the orthonormal eigenvectors in (9.8) might not be unique. Depending on the sample covariance matrix $\mathbf{Q}$, there might be different sets of orthonormal vectors that correspond to the same eigenvalue of $\mathbf{Q}$. Thus, for a given length n of the new feature vectors, there might be several different matrices $\mathbf{W}$ that achieve the same (optimal) reconstruction error $\widehat{L}^{(\rm PCA)}$.

Computationally, Algorithm 15 essentially amounts to an EVD of the sample covariance matrix $\mathbf{Q}$ (9.9). The EVD of $\mathbf{Q}$ provides not only the optimal compression matrix $\mathbf{W}_{\rm PCA}$ but also the measure $\widehat{L}^{(\rm PCA)}$ for the information loss incurred by replacing the original data points $\mathbf{z}^{(i)} \in \mathbb{R}^{n'}$ with the shorter feature vector $\mathbf{x}^{(i)} \in \mathbb{R}^{n}$. We quantify this information loss by the approximation error obtained when using the compression matrix $\mathbf{W}_{\rm PCA}$ (and

corresponding reconstruction matrix $\mathbf{R}_{\rm opt} = \mathbf{W}_{\rm PCA}^T$),

$$\widehat{L}^{(\rm PCA)} := \widehat{L}\big(\mathbf{W}_{\rm PCA}, \underbrace{\mathbf{R}_{\rm opt}}_{=\mathbf{W}_{\rm PCA}^T} \mid \mathcal{D}\big) = \sum_{j=n+1}^{n'} \lambda_j. \tag{9.11}$$

As depicted in Figure 9.2, the approximation error $\widehat{L}^{(\rm PCA)}$ decreases with increasing number n of PCs used for the new features (9.5). For the extreme case $n=0$, where we completely ignore the raw feature vectors $\mathbf{z}^{(i)}$, the optimal reconstruction error is $\widehat{L}^{(\rm PCA)} = (1/m)\sum_{i=1}^m \big\|\mathbf{z}^{(i)}\big\|^2$. The other extreme case $n = n'$ allows to use the raw features directly as the new features $\mathbf{x}^{(i)} = \mathbf{z}^{(i)}$. This extreme case means no compression at all, and trivially results in a zero reconstruction error $\widehat{L}^{(\rm PCA)} = 0$.

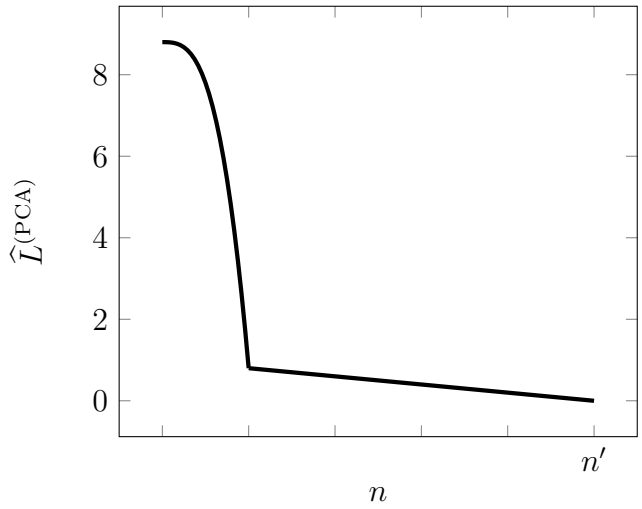

Figure 9.2: Reconstruction error $\widehat{L}^{(\rm PCA)}$ (see (9.11)) of PCA for varying number n of PCs.

9.2.1 Combining PCA with linear regression

One important use case of PCA is as a pre-processing step within an overall ML problem such as linear regression (see Section 3.1). As discussed in Chapter 7, linear regression methods are prone to overfitting whenever the data points are characterized by raw feature vectors $\mathbf{z}$ whose length n' exceeds the number m of labeled data points used in ERM.

One simple but powerful strategy to avoid overfitting is to preprocess the raw features $\mathbf{z}^{(i)} \in \mathbb{R}^{n'}$, for $i = 1, \ldots, m$ by applying PCA. Indeed, PCA Algorithm 15 delivers feature

vectors $\mathbf{x}^{(i)} \in \mathbb{R}^n$ of prescribed length n. Thus, choosing the parameter n such that $n < m$ will typically prevent the follow-up linear regression method from overfitting.

9.2.2 How To Choose Number of PC?

There are several aspects which can guide the choice for the number n of PCs to be used as features.

- To generate data visualizations we might use either $n = 2$ or $n = 3$.
- We should choose n sufficiently small such that the overall ML method fits the available computational resources.
- Consider using PCA as a pre-processing for linear regression (see Section 3.1). In particular, we can use the learnt feature vectors $\mathbf{x}^{(i)}$ delivered by PCA as the feature vectors of data points in plain linear regression methods. To avoid overfitting, we should choose $n < m$ (see Chapter 7).
- Choose n large enough such that the resulting approximation error $\widehat{L}^{(\mathrm{PCA})}$ is reasonably small (see Figure 9.2).

9.2.3 Data Visualisation

If we use PCA with $n = 2$, we obtain feature vectors $\mathbf{x}^{(i)} = \mathbf{W}_{\mathrm{PCA}}\mathbf{z}^{(i)}$ (see (9.5)) which can be depicted as points in a scatterplot (see Section 2.1.3). As an example, consider data points $\mathbf{z}^{(i)}$ obtained from historic recordings of Bitcoin statistics. Each data point $\mathbf{z}^{(i)} \in \mathbb{R}^{n'}$ is a vector of length $n' = 6$. It is difficult to visualise points in an Euclidean space $\mathbb{R}^{n'}$ of dimension $n' > 2$. Therefore, we apply PCA with $n = 2$ which results in feature vectors $\mathbf{x}^{(i)} \in \mathbb{R}^2$. These new feature vectors (of length 2) can be depicted conveniently as the scatterplot in Figure 9.3.

9.2.4 Extensions of PCA

We now briefly discuss variants and extensions of the basic PCA method.

- **Kernel PCA [58, Ch.14.5.4]:** The PCA method is most effective if the raw feature vectors of data points are concentrated around a n-dimensional linear subspace of $\mathbb{R}^{n'}$. Kernel PCA extends PCA to data points that are located near a low-dimensional

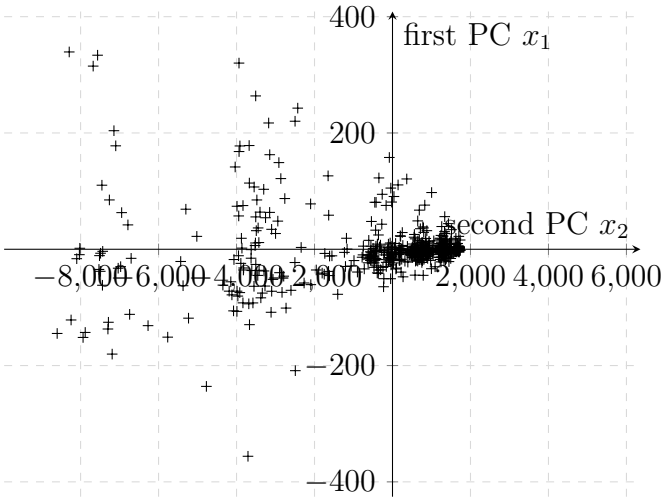

Figure 9.3: A scatterplot of data points with feature vectors $\mathbf{x}^{(i)} = \big(x_1^{(i)}, x_2^{(i)}\big)^T$ whose entries are the first two PCs of the Bitcoin statistics $\mathbf{z}^{(i)}$ of the i-th day.

manifold which might be highly non-linear. This is achieved by applying PCA to transformed feature vectors instead of the original raw feature vectors. Kernel PCA first applies a (typically non-linear) feature map ϕ to the raw feature vectors $\mathbf{z}^{(i)}$ (see Section 3.9) and applies PCA to the transformed feature vectors $\phi\big(\mathbf{z}^{(i)}\big)$, for $i = 1, \ldots, m$.

- **Robust PCA [156]:** The basic PCA Algorithm 15 is sensitive to outliers, i.e., a small number of data points with significantly different statistical properties than the bulk of data points. This sensitivity might be attributed to the properties of the squared Euclidean norm (9.3) which is used in PCA to measure the reconstruction error (9.1). We have seen in Chapter 3 that linear regression (see Section 3.1 and 3.3) can be made robust against outliers by replacing the squared error loss with another loss function. In a similar spirit, robust PCA replaces the squared Euclidean norm with another norm that is less sensitive to having very large reconstruction errors (9.1) for a small number of data points (which are outliers).

- **Sparse PCA [58, Ch.14.5.5]:** The basic PCA method transforms the raw feature vector $\mathbf{z}^{(i)}$ of a data point to a new (shorter) feature vector $\mathbf{x}^{(i)}$. In general each entry $x_j^{(i)}$ of the new feature vector will depend on each entry of the raw feature vector $\mathbf{z}^{(i)}$. More precisely, the new feature $x_j^{(i)}$ depends on all raw features $z_{j'}^{(i)}$ for which the corresponding entry $W_{j,j'}$ of the matrix $\mathbf{W} = \mathbf{W}_{\rm PCA}$ (9.8) is non-zero. For most

datasets, all entries of the matrix $\mathbf{W}_{\rm PCA}$ will typically be non-zero.

In some applications of linear dimensionality reduction we would like to construct new features that depend only on a small subset of raw features. Equivalently we would like to learn a linear compression map $\mathbf{W}$ (9.5) such that each row of $\mathbf{W}$ contains only few non-zero entries. To this end, sparse PCA enforces the rows of the compression matrix $\mathbf{W}$ to contain only a small number of non-zero entries. This enforcement can be implement either using additional constraints on $\mathbf{W}$ or by adding a penalty term to the reconstruction error (9.7).

- **Probabilistic PCA [118, 138]:** We have motivated PCA as a method for learning an optimal linear compression map (matrix) (9.5) such that the compressed feature vectors allows to linearly reconstruct the original raw feature vector with minimum reconstruction error (9.7). Another interpretation of PCA is that of a method that learns a subspace of $\mathbb{R}^{n'}$ that best fits the distribution of the raw feature vectors $\mathbf{z}^{(i)}$, for $i = 1, \ldots, m$. This optimal subspace is precisely the subspace spanned by the rows of $\mathbf{W}_{\rm PCA}$ (9.8).

  Probabilistic PCA (PPCA) interprets the raw feature vectors $\mathbf{z}^{(i)}$ as realizations of i.i.d. RVs. These realizations are modelled as

$$\mathbf{z}^{(i)} = \mathbf{W}^T \mathbf{x}^{(i)} + \boldsymbol{\varepsilon}^{(i)}, \text{ for } i = 1, \ldots, m. \tag{9.12}$$

  Here, $\mathbf{W} \in \mathbb{R}^{n \times n'}$ is some unknown matrix with orthonormal rows. The rows of $\mathbf{W}$ span the subspace around which the raw features are concentrated. The vectors $\mathbf{x}^{(i)}$ in (9.12) are realizations of i.i.d. RVs whose common probability distribution is $\mathcal{N}(\mathbf{0}, \mathbf{I})$. The vectors $\boldsymbol{\varepsilon}^{(i)}$ are realizations of i.i.d. RVs whose common probability distribution is $\mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I})$ with some fixed but unknown variance σ^2. Note that $\mathbf{W}$ and σ^2 parametrize the joint probability distribution of the feature vectors $\mathbf{z}^{(i)}$ via (9.12). PPCA amounts to maximum likelihood estimation (see Section 3.12) of the parameters $\mathbf{W}$ and σ^2. This maximum likelihood estimation problem can be solved using computationally efficient estimation techniques such as EM [138, Appendix B]. The implementation of PPCA via EM also offers a principled approach to handle missing data. Roughly speaking, the EM method allows to use the probabilistic model (9.12) to estimate missing raw features [138, Sec. 4.1].

9.3 Feature Learning for Non-Numeric Data

We have motivated dimensionality reduction methods as transformations of (very long) raw feature vectors to a new (shorter) feature vector $\mathbf{x}$ such that it allows to reconstruct the raw features $\mathbf{z}$ with minimum reconstruction error (9.1). To make this requirement precise we need to define a measure for the size of the reconstruction error and specify the class of possible reconstruction maps. PCA uses the squared Euclidean norm (9.7) to measure the reconstruction error and only allows for linear reconstruction maps (9.6).

Alternatively, we can view dimensionality reduction as the generation of new feature vectors $\mathbf{x}^{(i)}$ that maintain the intrinsic geometry of the data points with their raw feature vectors $\mathbf{z}^{(i)}$. Different dimensionality reduction methods use different concepts for characterizing the "intrinsic geometry" of data points. PCA defines the intrinsic geometry of data points using the squared Euclidean distances between feature vectors. Indeed, PCA produces feature vectors $\mathbf{x}^{(i)}$ such that for data points whose raw feature vectors have small squared Euclidean distance, also the new feature vectors $\mathbf{x}^{(i)}$ will have small squared Euclidean distance.

Some application domains generate data points for which the Euclidean distances between raw feature vectors does not reflect the intrinsic geometry of data points. As a point in case, consider data points representing scientific articles which can be characterized by the relative frequencies of words from some given set of relevant words (dictionary). A small Euclidean distance between the resulting raw feature vectors typically does not imply that the corresponding text documents are similar. Instead, the similarity between two articles might depend on the number of authors that are contained in author lists of both papers. We can represent the similarities between all articles using a similarity graph whose nodes represent data points which are connected by an edge (link) if they are similar (see Figure 8.9).

Consider a dataset $\mathcal{D} = \big(\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(m)}\big)$ whose intrinsic geometry is characterized by an unweighted similarity graph $\mathcal{G} = \big(\mathcal{V} := \{1, \ldots, m\}, \mathcal{E}\big)$. The node $i \in \mathcal{V}$ represents the i-th data point $\mathbf{z}^{(i)}$. Two nodes are connected by an undirected edge if the corresponding data points are similar.

We would like to find short feature vectors $\mathbf{x}^{(i)}$, for $i = 1, \ldots, m$, such that two data points i, i', whose feature vectors $\mathbf{x}^{(i)}, \mathbf{x}^{(i')}$ have small Euclidean distance, are well-connected to each other. This informal requirement must be made precise by a measure for how well two nodes of an undirected graph are connected. We refer the reader to literature on network theory for an overview and details of various connectivity measures [103].

Let us discuss a simple but powerful technique to map the nodes $i \in \mathcal{V}$ of an undirected graph $\mathcal{G}$ to (short) feature vectors $\mathbf{x}^{(i)} \in \mathbb{R}^n$. This map is such that the Euclidean distances between the feature vectors of two nodes reflect their connectivity within $\mathcal{G}$. This technique uses the Laplacian matrix $\mathbf{L} \in \mathbb{R}^{m \times m}$ which is defined for an undirected graph $\mathcal{G}$ (with node set $\mathcal{V} = \{1, \ldots, m\}$) element-wise

$$L_{i,i'} := \begin{cases} -1 & \text{, if } \{i, i'\} \in \mathcal{E} \\ d^{(i)} & \text{, if } i = i' \\ 0 & \text{otherwise.} \end{cases} . \tag{9.13}$$

Here, $d^{(i)} := \big|\{i' : \{i, i'\} \in \mathcal{E}\}\big|$ denotes the number of neighbours (the degree) of node $i \in \mathcal{V}$. It can be shown that the Laplacian matrix $\mathbf{L}$ is psd [148, Proposition 1]. Therefore we can find a set of orthonormal eigenvectors

$$\mathbf{u}^{(1)}, \ldots, \mathbf{u}^{(m)} \in \mathbb{R}^m \tag{9.14}$$

with corresponding (ordered in a non-decreasing fashion) eigenvalues $\lambda_1 \leq \ldots \leq \lambda_m$ of $\mathbf{L}$.

For a given number n, we construct the feature vector

$$\mathbf{x}^{(i)} := \big(u_i^{(1)}, \ldots, u_i^{(n)}\big)^T$$

for the ith data point. Here, we used the entries of of the first n eigenvectors (9.14). It can be shown that the Euclidean distances between the feature vectors $\mathbf{x}^{(i)}$, for $i = 1, \ldots, m$, reflect the connectivities between data points $i = 1, \ldots, m$ in the similarity graph $\mathcal{G}$. For a more precise statement of this informal claim we refer to the excellent tutorial [148].

To summarize, we can construct numeric feature vectors for (non-numeric) data points via the eigenvectors of the Laplacian matrix of a similarity graph for the data points. Algorithm 16 summarizes this feature learning method which requires as its input a similarity graph for the data points and the desired number n of numeric features. Note that Algorithm 16 does not make any use of the Euclidean distances between raw feature vectors and uses solely the similarity graph $\mathcal{G}$ to determine the intrinsic geometry of $\mathcal{D}$.

Algorithm 16 Feature Learning for Non-Numeric Data

Input: dataset $\mathcal{D} = \{\mathbf{z}^{(i)} \in \mathbb{R}^{n'}\}_{i=1}^{m}$; similarity graph $\mathcal{G}$; number n of features to be constructed for each data point.

1: construct the Laplacian matrix $\mathbf{L}$ of the similarity graph (see (9.13))

2: compute EVD of $\mathbf{L}$ to obtain n orthonormal eigenvectors (9.14) corresponding to the smallest eigenvalues of $\mathbf{L}$

3: for each data point i, construct feature vector

$$\mathbf{x}^{(i)} := \big(u_i^{(1)}, \ldots, u_i^{(n)}\big)^T \in \mathbb{R}^n \tag{9.15}$$

Output: $\mathbf{x}^{(i)}$, for $i = 1, \ldots, m$

9.4 Feature Learning for Labeled Data

We have discussed PCA as a linear dimensionality reduction method. PCA learns a compression matrix that maps raw features $\mathbf{z}^{(i)}$ of data points to new (much shorter) feature vectors $\mathbf{x}^{(i)}$. The feature vectors $\mathbf{x}^{(i)}$ determined by PCA depend solely on the raw feature vectors $\mathbf{z}^{(i)}$ of a given dataset $\mathcal{D}$. In particular, PCA determines the compression matrix such that the new features allow for a linear reconstruction (9.6) with minimum reconstruction error (9.7).

For some application domains we might not only have access to raw feature vectors but also to the label values $y^{(i)}$ of the data points in $\mathcal{D}$. Indeed, dimensionality reduction methods might be used as pre-processing step within a regression or classification problem that involves a labeled training set. However, in its basic form, PCA (see Algorithm 15) does not allow to exploit the information provided by available labels $y^{(i)}$ of data points $\mathbf{z}^{(i)}$. For some datasets, PCA might deliver feature vectors that are not very relevant for the overall task of predicting the label of a data point.

Let us now discuss a modification of PCA that exploits the information provided by available labels of the data points. The idea is to learn a linear construction map (matrix) $\mathbf{W}$ such that the new feature vectors $\mathbf{x}^{(i)} = \mathbf{W}\mathbf{z}^{(i)}$ allow to predict the label $y^{(i)}$ as good as possible. We restrict the prediction to be linear,

$$\hat{y}^{(i)} := \mathbf{r}^T\mathbf{x}^{(i)} = \mathbf{r}^T\mathbf{W}\mathbf{z}^{(i)}, \tag{9.16}$$

with some weight vector $\mathbf{r} \in \mathbb{R}^n$.

While PCA is motivated by minimizing the reconstruction error (9.1), we now aim at minimizing the prediction error $\hat{y}^{(i)} - y^{(i)}$. In particular, we assess the usefulness of a given pair of construction map $\mathbf{W}$ and predictor $\mathbf{r}$ (see (9.16)), using the empirical risk

$$\widehat{L}\big(\mathbf{W}, \mathbf{r} \mid \mathcal{D}\big) := (1/m) \sum_{i=1}^{m} \big(y^{(i)} - \hat{y}^{(i)}\big)^2 \\ \stackrel{(9.16)}{=} (1/m) \sum_{i=1}^{m} \big(y^{(i)} - \mathbf{r}^T \mathbf{W} \mathbf{z}^{(i)}\big)^2. \tag{9.17}$$

to guide the learning of a compressing matrix $\mathbf{W}$ and corresponding linear predictor weights $\mathbf{r}$ ((9.16)).

The optimal matrix $\mathbf{W}$ that minimizes the empirical risk (9.17) can be obtained via the EVD (9.10) of the sample covariance matrix $\mathbf{Q}$ (9.9). Note that we have used the EVD of $\mathbf{Q}$ already for PCA in Section 9.2 (see (9.8)). Remember that PCA uses the n eigenvectors $\mathbf{u}^{(1)}, \ldots, \mathbf{u}^{(n)}$ corresponding to the n largest eigenvalues of $\mathbf{Q}$. In contrast, to minimize (9.17), we need to use a different set of eigenvectors in the rows of $\mathbf{W}$ in general. To find the right set of n eigenvectors, we need the sample cross-correlation vector

$$\mathbf{q} := (1/m) \sum_{i=1}^{m} y^{(i)} \mathbf{z}^{(i)}. \tag{9.18}$$

The entry q_j of the vector $\mathbf{q}$ estimates the correlation between the raw feature $z_j^{(i)}$ and the label $y^{(i)}$. We then define the index set

$$\mathcal{S} := \{j_1, \ldots, j_n\} \text{ such that } \big(q_j\big)^2/\lambda_j \geq \big(q_{j'}\big)^2/\lambda_{j'} \text{ for any } j \in \mathcal{S}, j' \in \{1, \ldots, n'\} \notin \mathcal{S}. \tag{9.19}$$

The set $\mathcal{S}$ is constituted by the indices $j \in \{1, \ldots, n'\}$ of n largest numbers $\big(q_j\big)^2/\lambda_j$. It can then be shown that the rows of the optimal compression matrix $\mathbf{W}$ (minimizing (9.17)) are the eigenvectors $\mathbf{u}^{(j)}$ with $j \in \mathcal{S}$. We summarize the overall feature learning method in Algorithm 17.

The main focus of this section is on regression problems that involve data points with numeric labels (e.g., from the label space $\mathcal{Y} = \mathbb{R}$). Given the raw features and labels of the data points in $\mathcal{D}$, Algorithm 17 determines new feature vectors $\mathbf{x}^{(i)}$ that allow to linearly predict a numeric label with minimum squared error. A similar approach can be used for classification problems involving data points with a finite label space $\mathcal{Y}$. Linear (or Fisher) discriminant analysis aims at constructing a compression matrix $\mathbf{W}$ such that the learnt

Algorithm 17 Linear Feature Learning for Labeled Data

Input: dataset $\big(\mathbf{z}^{(1)}, y^{(1)}\big), \ldots, \big(\mathbf{z}^{(m)}, y^{(m)}\big)$ with raw features $\mathbf{z}^{(i)} \in \mathbb{R}^{n'}$ and numeric labels $y^{(i)} \in \mathbb{R}$; length n of new feature vectors.

1: compute EVD (9.10) of the sample covariance matrix (9.9) to obtain orthonormal eigenvectors $\big(\mathbf{u}^{(1)}, \ldots, \mathbf{u}^{(n')}\big)$ corresponding to (decreasingly ordered) eigenvalues $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_{n'} \geq 0$

2: compute the sample cross-correlation vector (9.18) and, in turn, the sequence

$$\big(q_1\big)^2/\lambda_1, \ldots, \big(q_{n'}\big)^2/\lambda_{n'} \tag{9.20}$$

3: determine indices $j_1, \ldots, j_n$ of n largest elements in (9.20)

4: construct compression matrix $\mathbf{W} := \big(\mathbf{u}^{(j_1)}, \ldots, \mathbf{u}^{(j_n)}\big)^T \in \mathbb{R}^{n \times n'}$

5: compute feature vector $\mathbf{x}^{(i)} = \mathbf{W}\mathbf{z}^{(i)}$

Output: $\mathbf{x}^{(i)}$, for $i = 1, \ldots, m$, and compression matrix $\mathbf{W}$.

features $\mathbf{x} = \mathbf{W}\mathbf{z}$ of a data point allow to predict its label y as accurately as possible [58].

9.5 Privacy-Preserving Feature Learning

Many important application domains of ML involve sensitive data that is subject to data protection law [149]. Consider a health-care provider (such as a hospital) holding a large database of patient records. From a ML perspective this databases is nothing but a (typically large) set of data points representing individual patients. The data points are characterized by many features including personal identifiers (name, social security number), bio-physical parameters as well as examination results . We could apply ML to learn a predictor for the risk of particular disease given the features of a data point.

Given large patient databases, the ML methods might not be implemented locally at the hospital but using cloud computing. However, data protection requirements might prohibit the transfer of raw patient records that allow to match individuals with bio-physical properties. In this case we might apply feature learning methods to construct new features for each patient such that they allow to learn an accurate hypothesis for predicting a disease but do not allow to identify sensitive properties of the patient such as its name or a social security number.

Let us formalize the above application by characterizing each data point (patient in the

hospital database) using raw feature vector $\mathbf{z}^{(i)} \in \mathbb{R}^{n'}$ and a sensitive numeric property $\pi^{(i)}$. We would like to find a compression map $\mathbf{W}$ such that the resulting features $\mathbf{x}^{(i)} = \mathbf{W}\mathbf{z}^{(i)}$ do not allow to accurately predict the sensitive property $\pi^{(i)}$. The prediction of the sensitive property is restricted to be a linear $\hat{\pi}^{(i)} := \mathbf{r}^T\mathbf{x}^{(i)}$ with some weight vector $\mathbf{r}$.

Similar to Section 9.4 we want to find a compression matrix $\mathbf{W}$ that transforms, in a linear fashion, the raw feature vector $\mathbf{z} \in \mathbb{R}^{n'}$ to a new feature vector $\mathbf{x} \in \mathbb{R}^n$. However the design criterion for the optimal compression matrix $\mathbf{W}$ was different in Section 9.4 where the new feature vectors should allow for an accurate linear prediction of the label. In contrast, here we want to construct feature vectors such that there is no accurate linear predictor of the sensitive property $\pi^{(i)}$.

As in Section 9.4, the optimal compression matrix $\mathbf{W}$ is given row-wise by the eigenvectors of the sample covariance matrix (9.9). However, the choice of which eigenvectors to use is different and based on the entries of the sample cross-correlation vector

$$\mathbf{c} := (1/m)\sum_{i=1}^{m} \pi^{(i)}\mathbf{z}^{(i)}. \tag{9.21}$$

We summarize the construction of the optimal privacy-preserving compression matrix and corresponding new feature vectors in Algorithm 18.

Algorithm 18 Privacy Preserving Feature Learning

Input: dataset $\big(\mathbf{z}^{(1)}, y^{(1)}\big), \ldots, \big(\mathbf{z}^{(m)}, y^{(m)}\big)$; each data point characterized by raw features $\mathbf{z}^{(i)} \in \mathbb{R}^{n'}$ and (numeric) sensitive property $\pi^{(i)} \in \mathbb{R}$; number n of new features.

1: compute the EVD (9.10) of the sample covariance matrix (9.9) to obtain orthonormal eigenvectors $\big(\mathbf{u}^{(1)}, \ldots, \mathbf{u}^{(n')}\big)$ corresponding to (decreasingly ordered) eigenvalues $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_{n'} \geq 0$

2: compute the sample cross-correlation vector (9.21) and, in turn, the sequence

$$\big(c_1\big)^2/\lambda_1, \ldots, \big(c_{n'}\big)^2/\lambda_{n'} \tag{9.22}$$

3: determine indices $j_1, \ldots, j_n$ of n smallest elements in (9.22)

4: construct compression matrix $\mathbf{W} := \big(\mathbf{u}^{(j_1)}, \ldots, \mathbf{u}^{(j_n)}\big)^T \in \mathbb{R}^{n \times n'}$

5: compute feature vector $\mathbf{x}^{(i)} = \mathbf{W}\mathbf{z}^{(i)}$

Output: feature vectors $\mathbf{x}^{(i)}$, for $i = 1, \ldots, m$, and compression matrix $\mathbf{W}$.

Algorithm 18 learns a map $\mathbf{W}$ to extract privacy-preserving features out of the raw

feature vector of a data point. These new features are privacy-preserving as they do not allow to accurately predict (in a linear fashion) a sensitive property π of the data point. Another formalization for the preservation of privacy can be obtained using information-theoretic concepts. This information-theoretic approach interprets data points, their feature vector and sensitive property, as realizations of RVs. It is then possible to use the mutual information between new features $\mathbf{x}$ and the sensitive (private) property π as an optimization criterion for learning a compression map h (Section 9.1). The resulting feature learning method (referred to as privacy-funnel) differs from Algorithm 18 not only in the optimization criterion for the compression map but also in that it allows it to be non-linear [89, 128].

9.6 Random Projections

Note that PCA uses an EVD of the sample covariance matrix $\mathbf{Q}$ (9.9). The computational complexity (e.g., measured by number of multiplications and additions) for computing this EVD is lower bounded by $n \min\{n'^2, m^2\}$ [46, 139]. This computational complexity can be prohibitive for ML applications with n' and m being of the order of millions or even billions.

There is a computationally cheap alternative to PCA (Algorithm 15) for finding a useful compression matrix $\mathbf{W}$ in (9.5). This alternative is to construct the compression matrix $\mathbf{W}$ entry-wise

$$W_{j,j'} := a_{j,j'} \text{ with } a_{j,j'} \sim p(a). \tag{9.23}$$

The matrix entries (9.23) are realizations $a_{i,j}$ of i.i.d. RVs with some common probability distribution $p(a)$. Different choices for the probability distribution $p(a)$ have been studied in the literature [39]. The Bernoulli distribution is used to obtain a compression matrix with binary entries. Another popular choice for $p(a)$ is the multivariate normal (Gaussian) distribution.

Consider data points whose raw feature vectors $\mathbf{z}$ are located near a s-dimensional subspace of $\mathbb{R}^{n'}$. The feature vectors $\mathbf{x}$ obtained via (9.5) using a random matrix (9.23) allows to reconstruct the raw feature vectors $\mathbf{z}$ with high probability whenever

$$n \geq Cs \log n'. \tag{9.24}$$

The constant C depends on the maximum tolerated reconstruction error η (such that $\|\widehat{\mathbf{z}} - \mathbf{z}\|_2^2 \leq \eta$ for any data point) and the probability that the features $\mathbf{x}$ (see (9.23)) allow for a maximum reconstruction error η [39, Theorem 9.27.].

9.7 Dimensionality Increase

The focus of this chapter is on dimensionality reduction methods that learn a feature map delivering new feature vectors which are (significantly) shorter than the raw feature vectors. However, it might sometimes be beneficial to learn a feature map that delivers new feature vectors which are longer than the raw feature vectors. We have already discussed two examples for such feature learning methods in Sections 3.2 and 3.9. Polynomial regression maps a single raw feature z to a feature vector containing the powers of the raw feature z. This allows to use apply linear predictor maps to the new feature vectors to obtain predictions that depend non-linearly on the raw feature z. Kernel methods might even use a feature map that delivers feature vectors belonging to an infinite-dimensional Hilbert space [125].

Mapping raw feature vectors into higher-dimensional (or even infinite-dimensional) spaces might be useful if the intrinsic geometry of the data points is simpler when looked at in the higher-dimensional space. Consider a binary classification problem where data points are highly inter-winded in the original feature space (see Figure 3.7). Loosely speaking, mapping into higher-dimensional feature space might "flatten-out" a non-linear decision boundary between data points. We can then apply linear classifiers to the higher-dimensional features to achieve accurate predictions.

9.8 Exercises

Exercise 9.1. Computational Burden of Many Features. Discuss the computational complexity of linear regression. How much computation do we need to compute the linear predictor that minimizes the average squared error on a training set?

Exercise 9.2. Power Iteration. The key computational step of PCA amounts to an EVD of the psd matrix (9.9). Consider an arbitrary initial vector $\mathbf{u}^{(r)}$ and the sequence obtained by iterating

$$\mathbf{u}^{(r+1)} := \mathbf{Q}\mathbf{u}^{(r)} / \big\|\mathbf{Q}\mathbf{u}^{(r)}\big\|. \tag{9.25}$$

What (if any) conditions on the initialization $\mathbf{u}^{(r)}$ ensure that the sequence $\mathbf{u}^{(r)}$ converges to the eigenvector $\mathbf{u}^{(1)}$ of $\mathbf{Q}$ that corresponds to its largest eigenvalue λ_1?

Exercise 9.3. Linear Classifiers with High-Dimensional Features. Consider a training set $\mathcal{D}$ consisting of $m = 10^{10}$ labeled data points $\big(\mathbf{z}^{(1)}, y^{(1)}\big), \ldots, \big(\mathbf{z}^{(m)}, y^{(m)}\big)$ with

raw feature vectors $\mathbf{z}^{(i)} \in \mathbb{R}^{4000}$ and binary labels $y^{(i)} \in \{-1, 1\}$. Assume we have used a feature learning method to obtain the new features $\mathbf{x}^{(i)} \in \{0, 1\}^n$ with $n = m$ and such that the only non-zero entry of $\mathbf{x}^{(i)}$ is $x_i^{(i)} = 1$, for $i = 1, \ldots, m$. Can you find a linear classifier that perfectly classifies the training set?

Chapter 10

Transparent and Explainable ML

The successful deployment of ML methods depends on their transparency or explainability. We formalize the notion of an explanation and its effect using a simple probabilistic model in Section 10.1. Roughly speaking, an explanation is any artefact. such as a list of relevant features or a reference data point from a training set, that conveys information about a ML method and its predictions. Put differently, explaining a ML method should reduce the uncertainty (of a human end-user) about its predictions.

Explainable ML is umbrella term for techniques that make ML method transparent or explainable. Providing explanations for the predictions of a ML method is particularly important when these predictions inform decision making [23]. It is increasingly becoming a legal requirement to provide explanations for automated decision making systems [53].

Even for applications where predictions are not directly used to inform far-reaching decisions, providing explanations is important. The human end users have an intrinsic desire for explanations that resolve the uncertainty about the prediction. This is known as the "need for closure" in psychology [29, 75]. Beside legal and psychological requirements, providing explanations for predictions might also be useful for validating and verifying ML methods. Indeed, the explanations of ML methods (and its predictions) can point the user (which might be a "domain expert") to incorrect modelling assumptions used by the ML method [37].

Explainable ML is challenging since explanations must be tailored (personalized) to human end-users with varying backgrounds and in different contexts [87]. The user background includes the formal education as well as the individual digital literacy. Some users might have received university-level education in ML, while other users might have no relevant formal training (such as an undergraduate course in linear algebra). Linear regression with

few features might be perfectly interpretable for the first group but be considered a "black box" for the latter. To enable tailored explanations we need to model the user background as relevant for understanding the ML predictions.

This chapter discusses explainable ML methods that have access to some user signal or feedback for some data points. Such a user signal might be obtained in various ways, including answers to surveys or bio-physical measurements collected via wearables or medical diagnostics. The user signal is used to determine (to some extent) the end-user background and, in turn, to tailor the delivered explanations for this end-user.

Existing explainable ML methods can be roughly divided into two categories. The first category is referred to as "model-agnostic" [23]). Model-agnostic methods do not require knowledge of the detailed work principles of a ML method. These methods do not require knowledge of the hypothesis space used by a ML method but learn how to explain its predictions by observing them on a training set [22].

A second category of explainable ML methods, sometimes referred to as "white-box" methods [23], uses ML methods that are considered as intrinsically explainable. The intrinsic explainability of a ML method depends crucially on its choice for the hypothesis space (see Section 2.2). This chapter discusses one recent method from each of the two explainable ML categories [72, 69]. The common theme of both methods is the use of information-theoretic concepts to measure the usefulness of explanations [27].

Section 10.1 discusses a recently proposed model-agnostic approach to explainable ML that constructs tailored explanations for the predictions of a given ML method [72]. This approach does not require any details about the internal mechanism of a ML method whose predictions are to be explained. Rather, this approach only requires a (sufficiently large) training set of data points for which the predictions of the ML method are known.

To tailor the explanations to a particular user, we use the values of a user (feedback) signal provided for the data points in the training set. Roughly speaking, the explanations are chosen such that they maximally reduce the "surprise" or uncertainty that the user has about the predictions of the ML method.

Section 10.2 discusses an example for a ML method that uses a hypothesis space that is intrinsically explainable [69]. We construct an explainable hypothesis space by appropriate pruning of a given hypothesis space such as linear maps (see Section 3.1) or non-linear maps represented by either an ANN (see Section 3.11) or decision trees (see Section 3.10). This pruning is implemented via adding a regularization term to ERM (4.3), resulting in an instance of SRM (7.2) which we refer to as explainable empirical risk minimization (EERM).

The regularization term favours hypotheses that are explainable to a user. Similar to the method in Section 10.1, the explainability of a map is quantified by information theoretic quantities. For example, if the original hypothesis space is the set of linear maps using a large number of features, the regularization term might favour maps that depend only on few features that are interpretable. Hence, we can interpret EERM as a feature learning method that aims at learning relevant and interpretable features (see Chapter 9).

10.1 Personalized Explanations for ML Methods

Consider a ML application involving data points with features $\mathbf{x} = \big(x_1, \ldots, x_n\big)^T \in \mathbb{R}^n$ and label $y \in \mathbb{R}$. We use a ML method that reads in some labelled data points

$$\big(\mathbf{x}^{(1)}, y^{(1)}\big), \big(\mathbf{x}^{(2)}, y^{(2)}\big), \ldots, \big(\mathbf{x}^{(m)}, y^{(m)}\big), \tag{10.1}$$

and learns a hypothesis

$$h(\cdot) : \mathbb{R}^n \to \mathbb{R} : \mathbf{x} \mapsto \hat{y} = h(\mathbf{x}). \tag{10.2}$$

The precise working principle of this ML method for how to learn this hypothesis h is not relevant in what follows.

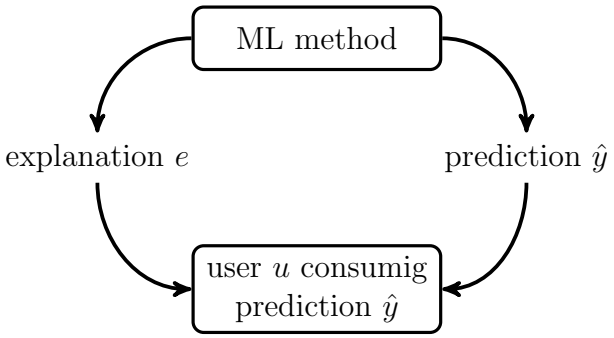

Figure 10.1: An explanation e provides additional information $I(\hat{y}, e|u)$ to a user u about the prediction $\hat{y}$.

The learnt predictor $h(\mathbf{x})$ is applied to the features of a data point to obtain the predicted label $\hat{y} := h(\mathbf{x})$. The prediction $\hat{y}$ is then delivered to a human end-user (see Figure 10.1). Depending on the ML application, this end-user might be a streaming service subscriber [47], a dermatologist [36] or a city planner [158].

Human users of ML methods often have some conception or model for the relation between features $\mathbf{x}$ and label y of a data point. This intrinsic model might vary significantly between users with different (social or educational) background. We will model the user understanding of a data point by a "user summary" $u \in \mathbb{R}$. The summary is obtained by a (possibly stochastic) map from the features $\mathbf{x}$ of a data point. For ease of exposition, we focus on summaries obtained by a deterministic map

$$u(\cdot) : \mathbb{R}^n \to \mathbb{R} : \mathbf{x} \mapsto u := u(\mathbf{x}). \tag{10.3}$$

However, the resulting explainable ML method can be extended to user feedback u modelled as a stochastic maps. In this case, the user feedback u is characterized by a probability distribution $p(u|\mathbf{x})$.

The user feedback u is determined by the features $\mathbf{x}$ of a data point. We might think of the value u for a specific data point as a signal that reflects how the human end-user interprets (or perceives) the data point, given her knowledge (including formal education) and the context of the ML application. We do not assume any knowledge about the details for how the signal value u is formed for a specific data point. In particular, we do not know any properties of the map $u(\cdot) : \mathbf{x} \mapsto u$.

The above approach is quite flexible as it allows for very different forms of user summaries. The user summary could be the prediction obtained from a simplified model, such as linear regression using few features that the user anticipates as being relevant. Another example for a user summary u could be a higher-level feature, such as eye spacing in facial pictures, that the user considers relevant [67].

Note that, since we allow for an arbitrary map in (10.3), the user summary $u(\mathbf{x})$ obtained for a random data point with features $\mathbf{x}$ might be correlated with the prediction $\hat{y} = h(\mathbf{x})$. As an extreme case, consider a very knowledgable user that is able to predict the label of any data point from its features as well as the ML method itself. In this case, the maps (10.2) and (10.3) might be nearly identical. However, in general the predictions delivered by the learnt hypothesis (10.2) will be different from the user summary $u(\mathbf{x})$.

We formalize the act of explaining a prediction $\hat{y} = h(\mathbf{x})$ as presenting some additional quantity e to the user (see Figure 10.1). This explanation e can be any artefact that helps the user to understand the prediction $\hat{y}$, given her understanding u of the data point. Loosely speaking, the aim of providing explanation e is to reduce the uncertainty of the user u about the prediction $\hat{y}$ [75].

For the sake of exposition, we construct explanations e that are obtained via a determin-

istic map

$$e(\cdot) : \mathbb{R}^n \to \mathbb{R} : \mathbf{x} \mapsto e := e(\mathbf{x}), \tag{10.4}$$

from the features $\mathbf{x}$ of a data point. However, the explainable ML methods in this chapter can be generalized without difficulty to handle explanations obtained from a stochastic map. In the end, we only require the specification of the conditional probability distribution $p(e|\mathbf{x})$.

The explanation e (10.4) depends only on the features $\mathbf{x}$ but not explicitly on the prediction $\hat{y}$. However, our method for constructing the map (10.4) takes into account the properties of the predictor map $h(\mathbf{x})$ (10.2). In particular, Algorithm 19 below requires as input the predicted labels $\hat{y}^{(i)}$ for a set of data points (that serve as a training set for our method).

To obtain comprehensible explanations that can be computed efficiently, we must typically restrict the space of possible explanations to a small subset $\mathcal{F}$ of maps (10.4). This is conceptually similar to the restriction of the space of possible predictor functions in a ML method to a small subset of maps which is known as the hypothesis space.

10.1.1 Probabilistic Data Model for XML

In what follows, we model data points as realizations of i.i.d. RVs with common (joint) probability distribution $p(\mathbf{x}, y)$ of features and label (see Section 2.1.4). Modelling the data points as realizations of RVs implies that the user summary u, prediction $\hat{y}$ and explanation e are also realizations of RVs. The joint distribution $p(u, \hat{y}, e, \mathbf{x}, y)$ conforms with the Bayesian network [111] depicted in Figure 10.2. Indeed,

$$p(u, \hat{y}, e, \mathbf{x}, y) = p(u|\mathbf{x}) \cdot p(e|\mathbf{x}) \cdot p(\hat{y}|\mathbf{x}) \cdot p(\mathbf{x}, y). \tag{10.5}$$

We measure the amount of additional information provided by an explanation e for a prediction $\hat{y}$ to some user u via the conditional mutual information (MI) [27, Ch. 2 and 8]

$$I(e; \hat{y}|u) := \mathbb{E}\left\{ \log \frac{p(\hat{y}, e|u)}{p(\hat{y}|u)p(e|u)} \right\}. \tag{10.6}$$

The conditional MI $I(e; \hat{y}|u)$ can also be interpreted as a measure for the amount by which the explanation e reduces the uncertainty about the prediction $\hat{y}$ which is delivered to some user u. Providing the explanation e serves the apparent human need to understand observed phenomena, such as the predictions from a ML method [75].

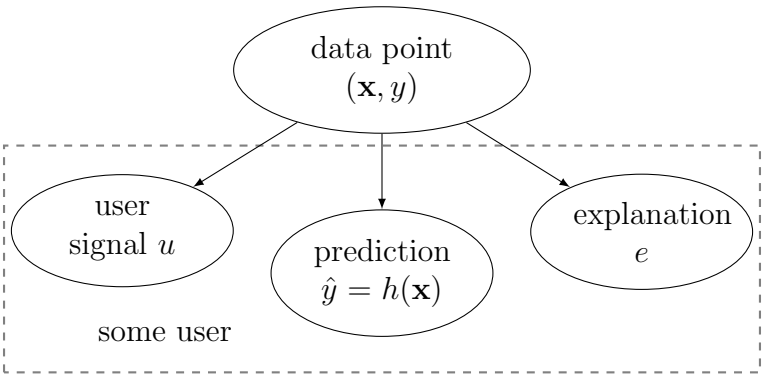

Figure 10.2: A simple probabilistic graphical model (a Bayesian network [84, 79]) for explainable ML. We interpret data points (with features $\mathbf{x}$ and label y) along with the user summary u, e and predicted label $\hat{y}$ as realizations of RVs. These RVs satisfy conditional independence relations encoded by the directed links of the graph [79]. Given the data point, the predicted label $\hat{y}$, the explanation e and the user summary u are conditionally independent. This conditional independence is trivial if all these quantities are obtained from deterministic maps applied to the features $\mathbf{x}$ of the data point.

10.1.2 Computing Optimal Explanations

Capturing the effect of an explanation using the probabilistic model (10.6) offers a principled approach to computing an optimal explanation e. We require the optimal explanation e^* to maximize the conditional MI (10.6) between the explanation e and the prediction $\hat{y}$ conditioned on the user summary u of the data point.

Formally, an optimal explanation e^* solves

$$I(e^*; \hat{y}|u) = \sup_{e \in \mathcal{F}} I(e; \hat{y}|u). \tag{10.7}$$

The choice for the subset $\mathcal{F}$ of valid explanations offers a trade-off between comprehensibility, informativeness and computational cost incurred by an explanation e^* (solving (10.7)).

The maximization problem (10.7) for obtaining optimal explanations is similar to the approach in [22]. However, while [22] uses the unconditional MI between explanation and prediction, (10.7) uses the conditional MI given the user summary u. Therefore, (10.7) delivers personalized explanations that are tailored to the user who is characterized by the summary u.

It is important to note that the construction (10.7) allows for many different forms of

explanations. An explanation could be a subset of features of a data point (see [116] and Section 10.1.2). More generally, explanations could be obtained from simple local statistics (averages) of features that are considered related, such as nearby pixels in an image or consecutive amplitude values of an audio signal. Instead of individual features, carefully chosen data points from a training set can also serve as an explanation [92, 117].

Let us illustrate the concept of optimal explanations (10.7) using linear regression. We model the features $\mathbf{x}$ as a realization of a multivariate normal random vector with zero mean and covariance matrix $\mathbf{C}_x$,

$$\mathbf{x} \sim \mathcal{N}(\mathbf{0}, \mathbf{C}_x). \tag{10.8}$$

The predictor and the user summary are linear functions

$$\hat{y} := \mathbf{w}^T \mathbf{x}, \text{ and } u := \mathbf{v}^T \mathbf{x}. \tag{10.9}$$

We construct explanations via subsets of individual features x_j that are considered most relevant for a user to understand the prediction $\hat{y}$ (see [98, Definition 2] and [97]). Thus, we consider explanations of the form

$$e := \{x_j\}_{j \in \mathcal{E}} \text{ with some subset } \mathcal{E} \subseteq \{1, \ldots, n\}. \tag{10.10}$$

The complexity of an explanation e is measured by the number $|\mathcal{E}|$ of features that contribute to it. We limit the complexity of explanations by a fixed (small) sparsity level,

$$|\mathcal{E}| \leq s (\ll n). \tag{10.11}$$

Modelling the feature vector $\mathbf{x}$ as Gaussian (10.8) implies that the prediction $\hat{y}$ and user summary u obtained from (10.9) is jointly Gaussian for a given $\mathcal{E}$ (10.4). Basic properties of multivariate normal distributions [27, Ch. 8], allow to develop (10.7) as

$$\begin{aligned} &\max_{\substack{\mathcal{E} \subseteq \{1,\ldots,n\} \\ |\mathcal{E}| \leq s}} I(e; \hat{y}|u) \\ &\qquad = h(\hat{y}|u) - h(\hat{y}|u, \mathcal{E}) \\ &\qquad = (1/2) \log \mathbf{C}_{\hat{y}|u} - (1/2) \log \det \mathbf{C}_{\hat{y}|u, \mathcal{D}^{(\text{train})}} \\ &\qquad = (1/2) \log \sigma^2_{\hat{y}|u} - (1/2) \log \sigma^2_{\hat{y}|u, \mathcal{D}^{(\text{train})}}. \end{aligned} \tag{10.12}$$

Here, $\sigma^2_{\hat{y}|u}$ denotes the conditional variance of the prediction $\hat{y}$, conditioned on the user summary u. Similarly, $\sigma^2_{\hat{y}|u,\mathcal{E}}$ denotes the conditional variance of $\hat{y}$, conditioned on the user summary u and the subset $\{x_j\}_{j\in\mathcal{E}}$ of features. The last step in (10.12) follows from the fact that $\hat{y}$ is a scalar random variable.

The first component of the final expression of (10.12) does not depend on the index set $\mathcal{E}$ used to construct the explanation e (see (10.10)). Therefore, the optimal choice for $\mathcal{E}$ solves

$$\sup_{|\mathcal{E}|\leq s} -(1/2)\log\sigma^2_{\hat{y}|u,\mathcal{E}}. \tag{10.13}$$

The maximization (10.13) is equivalent to

$$\inf_{|\mathcal{E}|\leq s} \sigma^2_{\hat{y}|u,\mathcal{E}}. \tag{10.14}$$

In order to solve (10.14), we relate the conditional variance $\sigma^2_{\hat{y}|u,\mathcal{E}}$ to a particular decomposition

$$\hat{y} = \eta u + \sum_{j\in\mathcal{E}} \beta_j x_j + \varepsilon. \tag{10.15}$$

For an optimal choice of the coefficients η and β_j, the variance of the error term in (10.15) is given by $\sigma^2_{\hat{y}|u,\mathcal{E}}$. Indeed,

$$\min_{\eta,\beta_j\in\mathbb{R}} \mathbb{E}\big\{\big(\hat{y} - \eta u - \sum_{j\in\mathcal{E}} \beta_j x_j\big)^2\big\} = \sigma^2_{\hat{y}|u,e}. \tag{10.16}$$

Inserting (10.29) into (10.14), an optimal choice $\mathcal{E}$ (of feature) for the explanation of prediction $\hat{y}$ to user u is obtained from

$$\inf_{|\mathcal{E}|\leq s} \min_{\eta,\beta_j\in\mathbb{R}} \mathbb{E}\big\{\big(\hat{y} - \eta u - \sum_{j\in\mathcal{E}} \beta_j x_j\big)^2\big\} \tag{10.17}$$

$$= \min_{\|\boldsymbol{\beta}\|_0\leq s} \mathbb{E}\big\{\big(\hat{y} - \eta u - \boldsymbol{\beta}^T\mathbf{x}\big)^2\big\}. \tag{10.18}$$

An optimal subset $\mathcal{E}_{\rm opt}$ of features defining the explanation e (10.10) is obtained from any solution $\boldsymbol{\beta}_{\rm opt}$ of (10.18) via

$$\mathcal{E}_{\rm opt} = \operatorname{supp}\boldsymbol{\beta}_{\rm opt}. \tag{10.19}$$

Section 10.1.2 uses the probabilistic model (10.8) to construct optimal explanations via the (support of the) solutions $\boldsymbol{\beta}_{\rm opt}$ of the sparse linear regression problem (10.18). To obtain

a practical algorithm for computing (approximately) optimal explanations (10.19), we approximate the expectation in (10.18) using an average over the training set $\big(\mathbf{x}^{(i)}, \hat{y}^{(i)}, u^{(i)}\big)$, for $i = 1, \ldots, m$. This resulting method for computing personalized explanations is summarized in Algorithm 19.

Algorithm 19 XML Algorithm

Input: explanation complexity s, training set $\big(\mathbf{x}^{(i)}, \hat{y}^{(i)}, u^{(i)}\big)$ for $i = 1, \ldots, m$

1: compute $\widehat{\boldsymbol{\beta}}$ by solving

$$\widehat{\boldsymbol{\beta}} \in \operatorname*{argmin}_{\eta \in \mathbb{R}, \|\boldsymbol{\beta}\|_0 \leq s} (1/m) \sum_{i=1}^{m} \big(\hat{y}^{(i)} - \eta u^{(i)} - \boldsymbol{\beta}^T \mathbf{x}^{(i)}\big)^2 \tag{10.20}$$

Output: feature set $\widehat{\mathcal{E}} := \operatorname{supp} \widehat{\boldsymbol{\beta}}$

Algorithm 19 is interactive in the sense that the user has to provide a feedback signal $u^{(i)}$ for the data points with features $\mathbf{x}^{(i)}$. Based on the user feedback $u^{(i)}$, for $i = 1, \ldots, m$, Algorithm 19 learns an optimal subset $\mathcal{E}$ of features (10.10) that are used for the explanation of predictions.

The sparse regression problem (10.20) becomes intractable for large feature length n. However, if the features are weakly correlated with each other and the user summary u, the solutions of (10.20) can be found by efficient convex optimization methods. One popular method to (approximately) solve sparse regression (10.20) is the Lasso (see Section 3.4),

$$\widehat{\boldsymbol{\beta}} \in \operatorname*{argmin}_{\eta \in \mathbb{R}, \boldsymbol{\beta} \in \mathbb{R}^n} (1/m) \sum_{i=1}^{m} \big(\hat{y}^{(i)} - \eta u^{(i)} - \boldsymbol{\beta}^T \mathbf{x}^{(i)}\big)^2 + \lambda \|\boldsymbol{\beta}\|_1. \tag{10.21}$$

There is large body of work that studies the choice of Lasso parameter λ in (10.21) such that solutions (10.21) coincide with the solutions of (10.20) (see [59, 144] and references therein). The proper choice for λ typically requires knowledge of statistical properties of data. If such a probabilistic model is not available, the choice of λ can be guided by simple validation techniques (see Section 6.2).

10.2 Explainable Empirical Risk Minimization

Section 7.1 discussed SRM (7.1) as a method for pruning the hypothesis space $\mathcal{H}$ used in ERM (4.3). This pruning is implemented either via a (hard) constraint as in (7.1) or by

adding a regularization term to the training error as in (7.2). The idea of SRM is to avoid (prune away) hypothesis maps that perform good on the training set but poorly outside, i.e., they do not generalize well. Here, we will use another criterion for steering the pruning and construction of regularization terms. In particular, we use the (intrinsic) explainability of a hypotheses map as a regularization term.

To make the notion of explainability precise we use again the probabilistic model of Section 10.1.1. We interpret data points as realizations of i.i.d. RVs with common (joint) probability distribution $p(\mathbf{x}, y)$ of features $\mathbf{x}$ and label y. A quantitative measure the intrinsic explainability of a hypothesis $h \in \mathcal{H}$ is the conditional (differential) entropy [27, Ch. 2 and 8]

$$H(\hat{y}|u) := -\mathbb{E}\Big\{ \log p(\hat{y}|u) \Big\}. \tag{10.22}$$

The conditional entropy (10.22) indicates the uncertainty about the prediction $\hat{y}$, given the user summary $\hat{u} = u(\mathbf{x})$. Smaller values $H(\hat{y}; u)$ correspond to smaller levels of uncertainty in the predictions $\hat{y}$ that is experienced by user u.

We obtain EERM by requiring a sufficiently small conditional entropy (10.22) of a hypothesis,

$$\hat{h} \in \operatorname*{argmin}_{h \in \mathcal{H}} \widehat{L}\big(h\big) \quad \text{s.t.} \quad H(\hat{y}|\hat{u}) \leq \eta. \tag{10.23}$$

The random variable $\hat{y} = h(\mathbf{x})$ in the constraint of (10.23) is obtained by applying the predictor map $h \in \mathcal{H}$ to the features. The constraint $H(\hat{y}|\hat{u}) \leq \eta$ in (10.23) enforces the learnt hypothesis $\hat{h}$ to be sufficiently explainable in the sense that the conditional entropy $H(\hat{h}|\hat{u}) \leq \eta$ does not exceed a prescribed level η.

Let us now consider the special case of EERM (10.23) for the linear hypothesis space

$$h^{(\mathbf{w})}(\mathbf{x}) := \mathbf{w}^T \mathbf{x} \text{ with some parameter vector } \mathbf{w} \in \mathbb{R}^n. \tag{10.24}$$

Moreover, we assume that the features $\mathbf{x}$ of a data point and its user summary u are jointly Gaussian with mean zero and covariance matrix $\mathbf{C}$,

$$\big(\mathbf{x}^T, \hat{u}\big)^T \sim \mathcal{N}(\mathbf{0}, \mathbf{C}). \tag{10.25}$$

Under the assumptions (10.24) and (10.25) (see [27, Ch. 8]),

$$H(\hat{u}|\hat{y}) = (1/2) \log \sigma^2_{\hat{y}|\hat{u}}. \tag{10.26}$$

Here, we used the conditional variance $\sigma^2_{\hat{y}|\hat{u}}$ of $\hat{y}$ given the random user summary $u = u(\mathbf{x})$. Inserting (10.26) into the generic form of EERM (10.23),

$$\hat{h} \in \operatorname*{argmin}_{h \in \mathcal{H}} \widehat{L}(h) \quad \text{s.t.} \quad \log \sigma^2_{\hat{y}|\hat{u}} \leq \eta. \tag{10.27}$$

By the monotonicity of the logarithm, (10.27) is equivalent to

$$\hat{h} \in \operatorname*{argmin}_{h \in \mathcal{H}} \widehat{L}(h) \quad \text{s.t.} \quad \sigma^2_{\hat{y}|\hat{u}} \leq e^{(\eta)}. \tag{10.28}$$

To further develop (10.29), we use the identity

$$\min_{\eta \in \mathbb{R}} \mathbb{E}\big\{\big(\hat{y} - \eta u\big)^2\big\} = \sigma^2_{\hat{y}|\hat{u}}. \tag{10.29}$$

The identity (10.29) relates the conditional variance $\sigma^2_{\hat{y}|\hat{u}}$ to the minimum mean squared error that can be achieved by estimating $\hat{y}$ using a linear estimator $\eta\hat{u}$ with some $\eta \in \mathbb{R}$. Inserting (10.29) and (10.24) into (10.28),

$$\hat{h} \in \operatorname*{argmin}_{\mathbf{w} \in \mathbb{R}^n, \eta \in \mathbb{R}} \widehat{L}(h^{(\mathbf{w})}) \quad \text{s.t.} \quad \mathbb{E}\big\{\big(\underbrace{\mathbf{w}^T\mathbf{x}}_{\overset{(10.24)}{=}\hat{y}} - \eta\hat{u}\big)^2\big\} \leq e^{(\eta)}. \tag{10.30}$$

The inequality constraint in (10.30) is convex [14, Ch. 4.2.]. For squared error loss, the objective function $\widehat{L}(h^{(\mathbf{w})})$ is also convex. Thus, for linear least squares regression, we can reformulate (10.30) as an equivalent (dual) unconstrained problem [14, Ch. 5]

$$\hat{h} \in \operatorname*{argmin}_{\mathbf{w} \in \mathbb{R}^n, \eta \in \mathbb{R}} \mathcal{E}(h^{(\mathbf{w})}) + \lambda \mathbb{E}\big\{\big(\mathbf{w}^T\mathbf{x} - \eta\hat{u}\big)^2\big\}. \tag{10.31}$$

By convex duality, for a given threshold $e^{(\eta)}$ in (10.30), we can find a value for λ in (10.31) such that (10.30) and (10.31) have the same solutions [14, Ch. 5]. Algorithm 20 below is obtained from (10.31) by approximating the expectation $\mathbb{E}\big\{\big(\mathbf{w}^T\mathbf{x} - \eta\hat{u}\big)^2\big\}$ with an average over the data points $\big(\mathbf{x}^{(i)}, \hat{y}^{(i)}, \hat{u}^{(i)}\big)$ for $i = 1, \ldots, m$.

10.3 Exercises

Exercise 10.1. Convexity of Explainable Linear regression Rewrite the optimization problem (10.32) as an equivalent quadratic optimization problem $\min_{\mathbf{v} \in \mathbb{R}^n} \mathbf{v}^T\mathbf{Q}\mathbf{v} + \mathbf{v}^T\mathbf{q}$.

Algorithm 20 Explainable Linear Least Squares Regression

Input: explainability parameter λ, training set $\left(\mathbf{x}^{(i)}, \hat{y}^{(i)}, \hat{u}^{i)}\right)$ for $i = 1, \ldots, m$

1: solve

$$\widehat{\mathbf{w}} \in \underset{\eta \in \mathbb{R}, \mathbf{w} \in \mathbb{R}^n}{\operatorname{argmin}} (1/m) \sum_{i=1}^{m} \underbrace{\left(\hat{y}^{(i)} - \mathbf{w}^T \mathbf{x}^{(i)}\right)^2}_{\text{empirical risk}} + \lambda \underbrace{(\mathbf{w}^T \mathbf{x}^{(i)} - \eta \hat{u}^{(i)})^2}_{\text{explainability}} \tag{10.32}$$

Output: weights $\widehat{\mathbf{w}}$ of explainable linear hypothesis

Identify the matrix $\mathbf{Q} \in \mathbb{R}^{n \times n}$ and the vector $\mathbf{q} \in \mathbb{R}^n$.

Glossary

k-fold cross-validation (k-fold CV) k-fold cross-validation is a method for learning and validating a hypothesis using a given dataset. This method first divides the dataset evenly into k subsets or "folds" and then executes k repetitions of training and validation. Each repetition uses a different fold as the validation set and the remaining $k-1$ folds as a training set. The final output is the average of the validation errors obtained from the k repetitions. 159–162

k-means The k-means algorithm is a hard clustering method which assigns each data points to pecisely one out of k different clusters. The method iteratively updates this assignment in order to minimize the average distance between data points in their nearest cluster mean (centre). 14, 202, 203, 206, 207, 210, 217–220, 222, 256, 258

accuracy Consider data points characterized by features $\mathbf{x} \in \mathbf{x}$ and a categorical label y which takes on values from a finite label space $\mathcal{Y}$. The accuracy of a hypothesis $h : \mathcal{X} \to \mathcal{Y}$, when applied to the data points in a dataset $\mathcal{D} = \left\{\left(\mathbf{x}^{(1)}, y^{(1)}\right), \ldots, \left(\mathbf{x}^{(m)}, y^{(m)}\right)\right\}$ is then defined as $1 - (1/m) \sum_{i=1}^{m} L\left(\left(\mathbf{x}^{(i)}, y^{(i)}\right), h\right)$ using the 0/1 loss (2.9) 63, 80, 107

activation function Each artificial neuron within an ANN consists of an activation function that maps the inputs of the neuron to a single output value. In general, an activation function is a non-linear map of the weighted sum of neuron inputs (this weighted sum is the activation of the neuron). 10, 100, 101, 107, 108, 270

artificial intelligence Artificial intelligence aims at developing systems that behave rational in the sense of maximizing a long-term reward. 26–29, 31

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. 10, 11, 26, 34, 50, 53–55, 74, 81, 100–102, 107, 108, 114, 115, 135, 152, 176, 177, 183, 187, 243, 254, 258, 259, 263, 268, 270

bagging bagging (or "bootstrap aggregation") is a generic technique to improve (the robustness of) a given ML method. The idea is to use the bootstrap to generate perturbed copy of a given training set and then apply the original ML method to learn a separate hypothesis for each perturbed copy of the training set. The resulting set of hypotheses is then used to predict the label of a data point by combining or aggregating the individual predictions of each individual hypothesis. For hypotheses that deliver numeric label values (regression methods) this aggregation could be implemented by computing the average of individual predictions. 187

baseline A reference value or benchmark for the average loss incurred by a hypothesis when applied to the data points generated in a specific ML application. Such a reference value might be obtained from human performance (e.g., error rate of dermatologists diagnosing cancer from visual inspection of skin areas) or other ML methods ("competitors") 111, 151, 153, 175, 176

Bayes estimator A hypothesis whose Bayes risk is minimal [85]. 67, 68, 70, 95, 104, 109, 111–114, 122, 124, 125, 175, 179, 255

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 [85]. 66, 67, 70, 111, 151, 153, 175, 255, 260

bias Consider some unknown quantity $\bar{w}$, e.g., the true weight in a linear model $y = \bar{w}x + e$ relating feature and label of a data point. We might use an ML method (e.g., based on ERM) to compute an estimate $\hat{w}$ for the $\bar{w}$ based on a set of data points that are realizations of RVs. The (squared) bias incurred by the estimate $\hat{w}$ is typically defined as $B^2 := \big(\mathbb{E}\{\hat{w}\} - \bar{w}\big)^2$. We extend this definition to vector-valued quantities using the squared Euclidean norm $B^2 := \big\|\mathbb{E}\{\widehat{\mathbf{w}}\} - \overline{\mathbf{w}}\big\|_2^2$. 10, 170–173, 192, 193

bootstrap Consider a probabilistic model that interprets a given set of data points $\mathcal{D} = \big\{\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(m)}\big\}$ as realizations of i.i.d. RVs with a common probability distribution $p(\mathbf{z})$. The bootstrap uses the histogram of $\mathcal{D}$ as the underlying proability distribution $p(\mathbf{z})$. 153, 174, 175

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. 44, 59, 64, 65, 80, 201

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. 49, 62, 80, 95

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 30, 31, 34, 40, 201, 202, 209, 212, 258, 271

clustering Clustering means to decompose a given set of data points into few subsets, which are referred to as clusters, that consist of similar data points. Different clustering methods use different measures for the similarity between data points and different representation of clusters. The clustering method k-means uses the average feature vector ("cluster means") of a cluster as its representative (see Section 8.1). A popular soft clustering method based on GMM represents a cluster by a Gaussian (multivariate normal) probability distribution (see Section 8.2). 30, 32, 34, 54, 201, 219, 221

condition number The condition number $\kappa(\mathbf{Q})$ of a psd matrix $\mathbf{Q}$ is the ratio of the largest to the smallest eigenvalue of $\mathbf{Q}$. 119, 140, 143, 149, 194

confusion matrix Consider data points characterized by features $\mathbf{x}$ and label y having value $c \in \{1, \ldots, k\}$. The confusion matrix is $k \times k$ matrix with rows representing different values c of the true label of a data point. The columns of a confusion matrix correspond to different values c' delivered by a hypothesis $h(\mathbf{x})$. The (c, c')-th entry of the confusion matrix is the fraction of data points with label $y{=}c$ and predicted label $\hat{y}{=}c'$. 68, 69

convex A set $\mathcal{C}$ in $\mathbb{R}^n$ 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 [14]. 60, 64, 72, 94, 107, 115, 116, 134, 136, 138, 198

data A set of data points. 33, 35, 95, 105, 114, 173, 269

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. 42, 162, 177, 181, 188, 190, 195

data point A data point is any object that conveys information [27]. 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. 2, 3, 5, 7–11, 17, 20–26, 28–32, 34–51, 53–73, 76–80, 82–97, 101–115, 118, 119, 122, 123, 125–130, 133, 137, 138, 140–142, 144–146, 148, 149, 151–155, 157–164, 166–178, 180–184, 186–191, 194–196, 198–203, 206–208, 212–231, 233–240, 242–248, 250–252, 254–274

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 [56]. By using indexed lists of data points we avoid some of the challenges arising in concept of an abstract set. 8, 9, 38, 40, 43, 57, 73, 76, 78, 79, 107, 130, 174, 195, 201, 208, 214, 232, 235, 254, 261, 264

decision boundary Consider a hypothesis map h that reads in a feature vector $\mathbf{x} \in \mathbb{R}^n$ and delivers a value from a finite set $\mathcal{Y}$. The decision boundary induced by h is the set of vectors $\mathbf{x} \in \mathbb{R}^n$ that lie between different decision regions. More precisely, a vector $\mathbf{x}$ belongs to the decision boundary if and only if each neighbourhood $\{\mathbf{x}' : \|\mathbf{x}-\mathbf{x}'\| \leq \varepsilon\}$, for any $\varepsilon > 0$, contains at least two vectors with different function values. 53, 54, 93, 94

decision region Consider a hypothesis map h that can only take values from a finite set $\mathcal{Y}$. We refer to the set of features $\mathbf{x} \in \mathcal{X}$ that result in the same output $h(\mathbf{x}) = a$ as a decision region of the hypothesis h. 49, 53, 97, 98, 257

decision tree A decision tree is a flow-chart like representation of a hypothesis map h. More formally, a decision tree is a directed graph which reads in the feature vector $\mathbf{x}$ of a data point at its root node. The root node then forwards the data point to one of its children nodes based on some elementary test on the features $\mathbf{x}$. If the receiving children node is not a leaf node, i.e., it has itself children nodes, it represents another test. Based on the test result, the data point is further pushed to one of its neighbours. This testing and forwarding of the data point is repeated until the data point ends up in a leaf node (having no children nodes). The leaf nodes represent sets (decision regions) constituted by feature vectors $\mathbf{x}$ that are mapped to the same function value $h(\mathbf{x})$. 11, 97–99, 112, 115, 120–122, 176, 177, 187, 243, 269

deep net We refer to an ANN with a (relatively) large number of hidden layers as a deep ANN or "deep net". Deep nets are used to represent the hypothesis spaces of deep learning methods [48]. 102, 105, 131, 177

degree of belonging A number that indicates the extend by which a data point belongs to a cluster. The degree of belonging can be interpreted as a soft cluster assignment. Soft clustering methods typically represent the degree of belonging by a real number in the interval $[0, 1]$. The boundary values 0 and 1 correspond to hard cluster assignments. 201, 202, 212, 214, 218, 221, 271

density-based spatial clustering of applications with noise (DBSCAN) A clustering algorithm for data points that are characterized by numeric feature vectors. Similar to k-means and soft clustering via GMM also DBSCAN uses the Euclidean distances between feature vectors to determine the clusters. However, in contrast to these other clustering methods, DBSCAN uses a different notion of similarity between data points. In particular, DBSCAN considers two data points as similar if they are "connected" via a sequence (path) of close-by intermediate data points. Thus, DBSCAN might consider two data points as similar (and therefore belonging to the same cluster) even if their feature vectors have a large Euclidean distance. 219–221

differentiable A function $f : \mathbb{R}^n \to \mathbb{R}$ is differentiable if it has a gradient $\nabla f(\mathbf{x})$ everywhere (for every $\mathbf{x} \in \mathbb{R}^n$) [119]. 58, 60, 61, 64, 72, 94, 132

effective dimension The effective dimension $d_{\text{eff}}(\mathcal{H})$ of an infinite hypothesis space $\mathcal{H}$ is a measure of its size. Loosely speaking, the effective dimension is equal to the number of "independent" tunable parameters of the model. These parameters might be the coefficients used in a linear map or the weights and bias terms of an ANN. 56, 57, 108, 152, 168, 174, 176, 182, 183, 188, 189, 262

eigenvalue We refer to a number $\lambda \in \mathbb{R}$ as eigenvalue of a square matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ if there is a non-zero vector $\mathbf{x} \in \mathbb{R}^n \setminus \{\mathbf{0}\}$ such that $\mathbf{A}\mathbf{x} = \lambda \mathbf{x}$. 136, 137, 140, 143, 227, 228, 234, 235, 240, 259

eigenvalue decomposition The task of computing the eigenvalues and corresponding eigenvectors of a matrix. 227, 228, 235–240

eigenvector An eigenvector of a matrix $\mathbf{A}$ is a non-zero vector $\mathbf{x} \in \mathbb{R}^n \setminus \{\mathbf{0}\}$ such that $\mathbf{A}\mathbf{x} = \lambda \mathbf{x}$ with some eigenvalue λ. 219, 227, 228, 234–236, 238, 240

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. 10, 67, 68, 76, 90, 110, 111, 113, 115, 118, 120, 122, 126, 144, 156, 195, 198, 201, 207, 208, 217, 236, 253, 268, 272

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. 25, 33–36, 95, 103, 110–118, 120, 122, 124–132, 138, 140, 141, 151, 152, 154, 157, 158, 160, 161, 163, 169, 170, 175–177, 180–190, 200, 201, 206, 207, 215, 229, 243, 250, 255, 259, 260, 267, 270, 273

estimation error Consider data points with feature vectors $\mathbf{x}$ and label y. In some applications we can model the relation between features and label of a data point as $y = \bar{h}(\mathbf{x}) + \varepsilon$. Here we used some true hypothesis $\bar{h}$ and a noise term ε which might represent modelling or labelling errors. The estimation error incurred by a ML method that learns a hypothesis $\widehat{h}$, e.g., using ERM, is defined as $\widehat{h} - \bar{h}$. For a parametrized hypothesis space, consisting of hypothesis maps that are determined by a parameter vector $\mathbf{w}$, we define the estimation error in terms of parameter vectors as $\Delta\mathbf{w} = \widehat{\mathbf{w}} - \overline{\mathbf{w}}$. first 169, 170, 267

Euclidean space The Euclidean space $\mathbb{R}^n$ of dimension n refers to the space of all vectors $\mathbf{x} = \big(x_1, \ldots, x_n\big)$, with real-valued entries $x_1, \ldots, x_n \in \mathbb{R}$, whose geometry is defined by the inner product $\mathbf{x}^T\mathbf{x}' = \sum_{j=1}^{n} x_j x'_j$ between any two vectors $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^n$ [119]. 20, 21, 41, 43, 89, 96, 103, 201, 230, 261, 270

expectation maximization Expectation maximization is a generic technique for estimating the parameters of a probabilistic model (a parametrized probability distribution) $p(\mathbf{z}; \mathbf{w})$ from data [12, 58, 153]. Expectation maximization delivers an approximation to the maximum likelihood estimate for the model parameters $\mathbf{w}$. 215, 232, 269

expert ML aims at learning a hypothesis h that accurately predicts the label of a data point based on its features. We measure the prediction error using some loss function. Ideally we want to find a hypothesis that incurs minimum loss. One approach to make this goal precise is to use the i.i.d. assumption and use the resulting Bayes risk as the benchmark level for the (average) loss of a hypothesis. Alternatively we might know a reference or benchmark hypothesis h' which might be obtained by some existing ML method. We can then compare the loss incurred by h with the loss incurred by h'. Such a reference or baseline hypothesis h' is referred to as an expert. Note that an expert might deliver very poor predictions. We typically compare against many different experts and aim at incurring not much more loss than the best among those experts (this is known as regret minimization) [20, 60]. first 69, 70, 260, 270

explainability We define the (subjective) explainability of a ML method as the level of predictability (or lack of uncertainty) in its predictions delivered to a specific human user. Quantitative measures for the (subjective) explainability can be obtained via probabilistic models for the data fed into the ML method [72, 69]. 34, 242–244, 251, 253

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. 243, 244, 251, 252

explainable machine learning Explainable ML methods aim at complementing its predictions with some form of explanation for how each prediction has been obtained. 3, 23, 34, 242, 243, 245–247

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. 54, 55, 84, 88, 155, 240

feature matrix Conisder a dataset $\mathcal{D}$ with m data points, each of them characterized by the features $\mathbf{x}^{(i)}$, $i = 1, \ldots, m$. The feature matrix $\mathbf{X}$ of $\mathcal{D}$ is constructed by stacking, colum-wise, the features of the data points into a matrix $\mathbf{X} = \left(\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)}\right)$ of size $m \times n$. first 119

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 $\mathbb{R}^n$ with dimension n being the number of individual features of a data point. 41–43, 55, 94, 98, 103, 263

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" [52, 30, 38]. However, this book makes consequent use of the term features for low-level properties of data points that can be measured easily. 8, 35, 37, 38, 43, 55, 60, 61, 82, 87, 90, 95, 96, 101, 119, 129, 130, 140, 195, 222, 254, 257, 261, 265, 266, 268, 272

federated learning (FL) Federated learning is an umbrella term for ML methods that train models in a collaborative fashion using decentralized data and computation. 25, 114

Finnish Meteorological Institute The Finnish Meteorological Institute is a government agency responsible for gathering and reporting weather data in Finland. 16, 17, 25, 73, 126

Gaussian mixture model Gaussian mixture models (GMM) are a family of probabilistic

models for data points. Within a GMM, the feature vector $\mathbf{x}$ of a data point is interpreted as being drawn from one out of k different multivariate normal (Gaussian) distributions indexed by $c = 1, \ldots, k$. The probability that the feature vector $\mathbf{x}$ is drawn from the c-th Gaussian distribution is denoted p_c. The GMM is parametrized by the probability p_c of $\mathbf{x}$ being drawn from the c-th Gaussian distribution as well as the mean vectors $\boldsymbol{\mu}^{(c)}$ and covariance matrices $\mathbf{\Sigma}^{(c)}$ for $c = 1, \ldots, k$. 123, 213–220, 256, 258

gradient For a real-valued function $f : \mathbb{R}^n \to \mathbb{R} : \mathbf{w} \mapsto f(\mathbf{w})$, 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 $\nabla f(\mathbf{w})\big|_{\mathbf{w}'}$. 11, 33, 53, 65, 131, 132, 134, 138, 141, 144, 147, 148, 258, 271, 272

gradient descent (GD) Gradient descent is an iterative method for finding the minimum of a differentiable function $f(\mathbf{w})$. 64, 65, 119, 120, 131, 133–150, 176, 177, 186, 194–196, 265, 272

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. 3, 9, 33, 58, 59, 64, 94, 95, 112, 115, 116, 122, 131, 132, 134, 135, 147, 148, 176, 177, 272

hard clustering Hard clustering refers to the task of partitioning a given set of data points into (few) non-overlapping clusters. Each data point is assigned to one specific cluster. 201–203, 212, 217, 222

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 n 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 [151, 17]. 87, 152, 169

Hilbert space A Hilbert space is a linear vector space that is equipped with an inner product between pairs of vectors. One important example for a Hilbert space is the Euclidean spaces $\mathbb{R}^n$, for some dimension n, which consists of Euclidean vectors $\mathbf{u} = \big(u_1, \ldots, u_n\big)^T$ along with the inner product $\mathbf{u}^T\mathbf{v}$. 240

hinge loss Consider a data point that is characterized by a feature vector $\mathbf{x} \in \mathbb{R}^n$ and a binary label $y \in \{-1, 1\}$. The hinge loss incurred by a specific hypothesis h is defined as (2.11). 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 3.6). 64, 65, 93, 94, 165, 272

histogram Consider a dataset $\mathcal{D}$ consisting of data points $\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(m)}$ that belong to some box in $\mathbb{R}^n$. We partition this hyper-rectangle into smaller elementary boxes. The histogram of $\mathcal{D}$ is the assignment of each elementary box to the corresponding fractions of data points in $\mathcal{D}$ that belong to this elementary box. 174, 175

Huber loss The Huber loss is a mixture of the squared error loss and the absolute value of the prediction error. 85, 86

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. 18, 19, 31, 48, 49, 58, 60–63, 78, 80, 87, 90, 93, 94, 98, 107, 119, 128–130, 138, 151, 152, 167, 177, 181, 183, 184, 196, 197, 201, 254, 256, 259, 266, 268, 272, 273

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 (2.4). 2, 3, 12, 17, 18, 20, 31, 35, 36, 49–58, 66, 71, 74,

learning rate Consider an iterative method for finding or learning a good choice for a hypothesis. Such an iterative method repeats similar computational (update) steps that adjust or modify the current choice for the hypothesis to obtain an improved hypothesis. A prime example for such an iterative learning method is GD and its variants (see 5). We refer by learning rate to any parameter of an iterative learning method that controls the extent by which the current hypothesis might be modified or improved in each iteration. A prime example for such a parameter is the step size used in GD. Within this book we use the term learning rate mostly as a synonym for the step size of (a variant of) GD 9, 134–140, 142, 144–150, 177, 195, 272

learning task A learning tasks consists of a specific choice for a collection of data points (e.g., all images stored in a particular database), their features and labels. 182, 196, 197, 273

least absolute deviation regression Least absolute deviation regression uses the average of the absolute precondition errors to find a linear hypothesis. 115

least absolute shrinkage and selection operator (Lasso) The least absolute shrinkage and selection operator (Lasso) is an instance of 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 $\mathbf{w}$. 87, 250

linear classifier A classifier $h(\mathbf{x})$ maps the feature vector $\mathbf{x} \in \mathbb{R}^n$ 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 $\mathbb{R}^n$. 21, 53, 54, 94–97, 100, 141, 263

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). 2, 15, 33, 36, 45, 56, 78, 82, 84–89, 92, 96, 99, 104, 105, 107, 108, 111, 114–117, 119, 120, 126, 127, 129–132, 134, 137–139, 142, 152–155, 167, 168, 172, 177, 178, 182, 183, 186, 192–195, 218, 221, 224, 229–231, 240, 242, 245, 248, 249, 252, 265

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 (2.12). 59, 64, 65, 79, 80, 90, 94, 95, 107, 141, 165, 267

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). 45, 53, 64, 89, 90, 92, 94–96, 99, 100, 104, 105, 107, 108, 120, 124, 125, 131, 132, 134, 137, 141, 142, 149, 165

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. 2, 10, 19, 22, 32, 33, 35, 58–60, 64, 65, 79, 82, 87, 93, 95, 111, 115, 122, 127, 128, 130, 132, 138, 151–153, 157, 162, 167, 177, 181, 184, 189, 254, 255, 270–273

loss function A loss function is a map

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

which assigns a pair consisting of a data point, with features $\mathbf{x}$ and label y, and a hypothesis $h \in \mathcal{H}$ the non-negative real number $L\left((\mathbf{x}, y), h\right)$. The loss value $L\left((\mathbf{x}, y), h\right)$ quantifies the discrepancy between the true label y and the predicted label $h(\mathbf{x})$. Smaller (closer to zero) values $L\left((\mathbf{x}, y), h\right)$ mean a smaller discrepancy between predicted label and true label of a data point. Figure 2.12 depicts a loss function for a given data point, with features $\mathbf{x}$ and label y, as a function of the hypothesis $h \in \mathcal{H}$. 2, 3, 18–20, 25, 30–33, 35, 36, 45, 53, 57–61, 63–65, 68, 78, 81, 82, 85, 92–96, 102, 105, 111, 113, 115, 116, 127, 128, 132, 133, 151, 152, 162, 164, 165, 175, 182, 197, 231, 260, 266, 267, 271

maximum Given a set of real numbers, the maximum is the largest of those numbers. 74

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

as a solution to the optimization problem $\max_{\mathbf{w} \in \mathcal{W}} p(\mathcal{D}; \mathbf{w})$. 48, 58, 91, 92, 102, 103, 123, 215, 260

mean The expectation of a real-valued random variable. 47

mean squared estimation error Consider a ML method that uses a parametrized hypothesis space. For a given training set, whose data points are interpreted as realizations of RVs, the ML method learns the parameters incurring the estimation error $\Delta \mathbf{w}$. The mean squared estimation error is defined as the expectation $\mathbb{E}\{\|\Delta \mathbf{w}\|^2\}$ of the squared Euclidean norm of the estimation error. 170, 172

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 (2.9)) 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 (2.8) and the logistic loss (2.12). 59, 60

minimum Given a set of real numbers, the minimum is the smallest of those numbers. 74

missing data By missing data, we refer to a situation where some feature values of a subset of data points are unknown. Data imputation techniaues aim at estimating (predicting) these missing feature values [1]. 38

model We use the term model as a synonym for hypothesis space 2, 18, 33, 35, 45, 49, 81, 82, 151–153, 162, 176, 262, 263, 268, 274

multi-label classification Multi-label classification problems and methods involve data points that are characterized by several individual labels. 43, 45, 76, 182

nearest neighbour Nearest neighbour methods learn a hypothesis $h : \mathcal{X} \rightarrow \mathcal{Y}$ whose function value $h(\mathbf{x})$ is solely determined by the nearest neighbours in the feature space $\mathcal{X}$ 103–105, 108, 109

non i.i.d. data A dataset that cannot be well modelled as realizations of i.i.d. RVs. 267

non-i.i.d. See non-i.i.d. data. 114

non-smooth We refer to a function as non-smooth if it is not smooth [102]. 59, 116

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 $h^{(\mathbf{w})}$. 113, 115, 116, 122, 130–134, 136–139, 141–144, 146–148, 150

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. 58, 61, 85, 86, 128, 160, 231

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. 17, 152

positive semi-definite A symmetric matrix $\mathbf{Q} = \mathbf{Q}^T \in \mathbb{R}^{n \times n}$ is referred to as positive semi-definite if $\mathbf{x}^T \mathbf{Q} \mathbf{x} \geq 0$ for every vector $\mathbf{x} \in \mathbb{R}^n$. 6, 10, 103, 136, 140, 194, 227, 234, 240, 256

prediction A prediction is an estimate or approximation for some quantity of interest. ML revolves around learning or finding a hypothesis map h that reads in the features $\mathbf{x}$ of a data point and delivers a prediction $\widehat{y} := h(\mathbf{x})$ for its label y. 260, 272

predictor A predictor is a hypothesis whose function values are numeric, such as real numbers. Given a data point with features $\mathbf{x}$, the predictor value $h(\mathbf{x}) \in \mathbb{R}$ is used as a prediction (estimate/guess/approximation) for the true numeric label $y \in \mathbb{R}$ of the data point. 96

principal component analysis (PCA) Principal component analysis determines a given number of new features that are obtained by a linear transformation (map) of the raw

features. 15, 21, 228–233, 235, 236, 239, 240, 269

probabilistic model A probabilistic model interprets data points as realizations of RVs with a joint probability distribution.This joint probability distribution typically involves parameters which have to be manually chosen (=design choice) or learnt via statistical inference methods [85]. 260

probabilistic PCA Probabilistic PCA extends basic PCA by using a probabilistic model for data points. Within this probabilistic model, the task of dimensionality reduction becomes an estimation problem that can be solved using EM methods. 232

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 $p(x \in \mathcal{B}) = \int_{\mathcal{B}} p(x')dx'$ [9, Ch. 3]. The pdf of a vector-valued RV $\mathbf{x} \in \mathbb{R}^n$ (if it exists) allows to compute the probability that $\mathbf{x}$ falls into a (measurable) region $\mathcal{R}$ via $p(\mathbf{x} \in \mathcal{R}) = \int_{\mathcal{R}} p(\mathbf{x}')dx'_1 \ldots dx'_n$ [9, Ch. 3]. 123, 264

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 [50, 11]. 22, 25, 26, 47, 56, 58, 63, 66–68, 95, 110–113, 122, 123, 125, 128, 129, 145, 148, 153, 154, 157, 159, 168, 174, 175, 177, 178, 189, 255, 256, 260, 264, 266, 268, 269, 271

random forest A random forest is a set (ensemble) of different decision trees. Each of these decision trees is obtained by fitting a perturbed copy of the original dataset. 187

random variable (RV) A random variable is a mapping from a probability space $\mathcal{P}$ to a value space [11]. The probability space, whose element are elementary events, is equipped with a probability measure that assigns a proability to subsets of $\mathcal{P}$. A binary

risk Consider a hypothesis h that is used to predict the label y of a data point based on its features $\mathbf{x}$. We measure the quality of a particular prediction using a loss function $L\left((\mathbf{x}, y), h\right)$. If we interpret data points as realizations of i.i.d. RVs, also the $L\left((\mathbf{x}, y), h\right)$ becomes the realization of a RV. Using such an i.i.d. assumption allows to define the risk of a hypothesis as the expected loss $\mathbb{E}\big\{L\left((\mathbf{x}, y), h\right)\big\}$. Note that the risk of h depends on both, the specific choice for the loss function and the common probability distribution of the data points. 112, 159, 173, 175, 255, 268

sample size The number of individual data points contained in a dataset that is obtained from realizations of i.i.d. RVs. 37, 38, 56, 116

scatterplot A visualization technique that depicts data points by markers in a two-dimensional plane. 20, 25, 39, 46, 47, 73, 83, 224, 230, 231, 261

semi-supervised learning Semi-supervised learning methods use (large amounts of) unlabeled data points to support the learning of a hypothesis from (a small number of) labeled data points [21]. 182, 195

similarity graph Some applications generate data points that are related by a domain-specific notion of similarity. These similarities can be represented conveniently using a similarity graph $\mathcal{G} = \big(\mathcal{V} := \{1, \ldots, m\}, \mathcal{E}\big)$. The node $i \in \mathcal{V}$ represents the i-th data point. Two nodes are connected by an undirected edge if the corresponding data points are similar. 233–235, 264

smooth We refer to a real-valued function as smooth if it is differentiable and its gradient is continuous [102, 16]. 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., $\|\nabla f(\mathbf{w}) - \nabla f(\mathbf{w}')\| \leq \beta \|\mathbf{w} - \mathbf{w}'\|$. 107, 115, 116, 133, 134, 136, 138, 267

soft clustering Soft clustering refers to the task of partitioning a given set of data points into (few) overlapping clusters. Each data point is assigned to several different clusters with varying degree of belonging. Soft clustering amounts to determining such a degree of belonging (or soft cluster assignment) for each data point and each cluster. 123, 201, 202, 213, 214, 218, 221, 258

spectogram The spectogram of a time signal, e.g., an audio recording, characterizes the time-frequency distribution of the signal. Loosely speaking, the spectogram quantities the signal strength at a specific time and frequency. 39, 40

squared error loss The squared error loss is widely used to measure the quality of a hypothesis h when predicting a numeric label $y \in \mathbb{R}$ from the features $\mathbf{x}$ of a data point. It is defined as the square $\big(y - h(\mathbf{x})\big)^2$ of the prediction error $y - \underbrace{h(\mathbf{x})}_{=\hat{y}}$. 60–63, 71, 76, 78, 119, 129, 130

step size Many ML methods use iterative optimization methods (such as gradient-based methods) to construct a sequence of increasingly accurate hypothesis maps $h^{(1)}, h^{(2)}, \ldots$. The rth iteration of such an algorithm starts from the current hypothesis $h^{(r)}$ and tries to modify it to obtain an improved hypothesis $h^{(r+1)}$. Iterative algorithms often use a step size (hyper-) parameter. The step size controls the amount by which a single iteration can change or modify the current hypothesis. Since the overall goal of such iteration ML methods is to learn a (approximately) optimal hypothesis we refer to a step size parameter also as a learning rate. 131, 133, 134

stochastic gradient descent Stochastic GD is obtained from GD by replacing the gradient of the objective function by a noisy (or stochastic) estimate. 8, 144–146, 148, 149, 174

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. 10, 181, 182, 184–187, 191, 243, 250, 251, 265

subgradient For a real-valued function $f : \mathbb{R}^n \to \mathbb{R} : \mathbf{w} \mapsto f(\mathbf{w})$, a vector $\mathbf{a}$ such that $f(\mathbf{w}) \geq f(\mathbf{w}') + \big(\mathbf{w} - \mathbf{w}'\big)^T \mathbf{a}$ is referred to as a subgradient of f at $\mathbf{w}'$ [**?** 10]. 65, 132

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 $\widehat{L}\big(h^{(\mathbf{w})} \big| \mathcal{D}\big)$ as a function of the parameters $\mathbf{w}$ of a hypothesis $h^{(\mathbf{w})}$. 65

support vector machine A binary classification method for learning a linear map that maximally separates 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 (2.11). 33, 53, 92–95, 99, 100, 115, 124, 125, 132, 165, 263

test set A set of data points that have neither been used in a training set to learn parameters of a model nor in a validation set to choose between different models (by comparing validation errors). 164

training error The average loss of a hypothesis when predicting the labels of data points in a training set. We sometimes refer by training error also the minimum average loss incurred on the training set by any hypothesis out of a hypothesis space. 10, 31, 34, 71, 72, 76, 110, 113, 114, 130, 151–155, 157, 158, 160, 161, 164–166, 169, 172, 175–178, 180, 181, 184–186, 218, 251, 273

training set A set of data points that is used in ERM to learn a hypothesis $\hat{h}$. The average loss of $\hat{h}$ on the training set is referred to as the training error. The comparison between training error and validation error of $\hat{h}$ allows to diagnose ML methods and informs how to improve them (e.g., using a different hypothesis space or collecting more data points). 9, 10, 22, 24, 25, 30, 31, 34, 51, 61, 62, 68, 73, 76, 79, 86, 87, 90, 94, 95, 103, 104, 107, 109–114, 116, 118–120, 122, 125–130, 137, 138, 141, 145, 146, 149, 151–161, 163–170, 172, 173, 176–178, 180, 181, 183, 184, 186–188, 190, 192, 198–200, 235, 242, 243, 246, 248, 250, 251, 253, 254, 259, 262, 263, 265–267, 270, 272, 273

transfer learning Transfer learning aims at leveraging information obtained while solving an existing learning task to solve another learning task. 182, 197

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. 10, 151–153, 156–160, 162–166, 175–178, 181, 254, 273

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). 10, 87, 111, 152, 156–160, 163–165, 175–178, 181, 198, 199, 254, 273

Vapnik–Chervonenkis (VC) dimension The VC dimension of an infinite hypothesis space is a widely-used measure for its size. We refer to [126] for a precise definition of VC dimension as well as a discussion of its basic properties and use in ML. 56

variance The variance of a real-valued RV x is defined as the expectation $\mathbb{E}\{\big(x - \mathbb{E}\{x\}\big)^2\}$ of the squared difference x and its expectation $\mathbb{E}\{x\}$. We extend this definition to vector-valued RVs $\mathbf{x}$ as $\mathbb{E}\{\big\|\mathbf{x} - \mathbb{E}\{\mathbf{x}\}\big\|_2^2\}$. 10, 47, 130, 170–173, 192, 193

weights We use the term weights synonymously for a finite set of parameters within a model. For example, the linear model consists of all linear maps $h(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$ that read in a feature vector $\mathbf{x} = \big(x_1, \ldots, x_n\big)^T$ of a data point. Each specific linear map is characterized by specific choices for the parameters for weights $\mathbf{w} = \big(w_1, \ldots, w_n\big)^T$. 17, 18, 87, 126, 174, 192, 198, 218, 219, 236, 253

Bibliography

[1] K. Abayomi, A. Gelman, and M. A. Levy. Diagnostics for multivariate imputations. *Journal of The Royal Statistical Society Series C-applied Statistics*, 57:273–291, 2008.

[2] M. Abramowitz and I. A. Stegun, editors. *Handbook of Mathematical Functions*. Dover, New York, 1965.

[3] D. Arthur and S. Vassilvitskii. k-means++: the advantages of careful seeding". In *Proc. of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics Philadelphia, 2007.

[4] P. Austin, P. Kaski, and K. Kubjas. Tensor network complexity of multilinear maps. *arXiv*, 2018.

[5] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. addwes, 1999.

[6] F. Barata, K. Kipfer, M. Weber, P. Tinschert, E. Fleisch, and T. Kowatsch. Towards device-agnostic mobile cough detection with convolutional neural networks. In *2019 IEEE International Conference on Healthcare Informatics (ICHI)*, pages 1–11, 2019.

[7] M. S. Bartlett. An inverse matrix adjustment arising in discriminant analysis. *The Annals of Mathematical Statistics*, 22(1):107 – 111, 1951.

[8] M. Belkin, D. Hsu, S. Ma, and S. Mandal. Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, 116(32):15849–15854, 2019.

[9] D. Bertsekas and J. Tsitsiklis. *Introduction to Probability*. Athena Scientific, 2 edition, 2008.

[10] D. P. Bertsekas. *Nonlinear Programming*. Athena Scientific, Belmont, MA, 2nd edition, June 1999.

[11] P. Billingsley. *Probability and Measure*. Wiley, New York, 3 edition, 1995.

[12] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[13] B. Boashash, editor. *Time Frequency Signal Analysis and Processing: A Comprehensive Reference*. Elsevier, Amsterdam, The Netherlands, 2003.

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

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

[16] S. Bubeck. Convex optimization. algorithms and complexity. In *Foundations and Trends in Machine Learning*, volume 8. Now Publishers, 2015.

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

[18] S. Carrazza. Machine learning challenges in theoretical HEP. *arXiv*, 2018.

[19] R. Caruana. Multitask learning. *Machine Learning*, 28(1):41–75, 1997.

[20] N. Cesa-Bianchi and G. Lugosi. *Prediction, Learning, and Games*. Cambridge University Press, New York, NY, USA, 2006.

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

[22] J. Chen, L. Song, M. Wainwright, and M. Jordan. Learning to explain: An information-theoretic perspective on model interpretation. In *Proc. 35th Int. Conf. on Mach. Learning*, Stockholm, Sweden, 2018.

[23] H.-F. Cheng, R. Wang, Z. Zhang, F. O'Connell, T. Gray, F. M. Harper, and H. Zhu. Explaining decision-making algorithms through UI: Strategies to help non-expert stakeholders. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, CHI '19, pages 1–12, New York, NY, USA, 2019. Association for Computing Machinery.

[24] I. Cohen and B. Berdugo. Noise estimation by minima controlled recursive averaging for robust speech enhancement. *IEEE Sig. Proc. Lett.*, 9(1):12–15, Jan. 2002.

[25] D. Cohn, Z. Ghahramani, and M. Jordan. Active learning with statistical models. *J. Artif. Int. Res.*, 4(1):129–145, March 1996.

[26] T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, 1967.

[27] T. M. Cover and J. A. Thomas. *Elements of Information Theory*. Wiley, New Jersey, 2 edition, 2006.

[28] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems 2*, (4):303–314, 1989.

[29] T. K. DeBacker and H. M. Crowson. Influences on cognitive engagement: Epistemological beliefs and need for closure. *British Journal of Educational Psychology*, 76(3):535–551, 2006.

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

[31] O. Dürr, Y. Pauchard, D. Browarnik, R. Axthelm, and M. Loeser. Deep learning on a raspberry pi for real time face recognition. 01 2015.

[32] B. Efron and R. Tibshirani. Improvements on cross-validation: The 632+ bootstrap method. *Journal of the American Statistical Association*, 92(438):548–560, 1997.

[33] R. Eldan and O. Shamir. The power of depth for feedforward neural networks. *CoRR*, abs/1512.03965, 2015.

[34] Y. C. Eldar, P. Kuppinger, and H. Bölcskei. Block-sparse signals: Uncertainty relations and efficient recovery. *IEEE Trans. Signal Processing*, 58(6):3042–3054, June 2010.

[35] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters a density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, pages 226–231, Portland, Oregon, 1996.

[36] A. Esteva, B. Kuprel, R. A. Novoa, J. Ko, S. M. Swetter, H. M. Blau, and S. Thrun. Dermatologist-level classification of skin cancer with deep neural networks. *Nature*, 542, 2017.

[37] J. W. et.al. Association between surgical skin markings in dermoscopic images and diagnostic performance of a deep learning convolutional neural network for melanoma recognition. *JAMA Dermatol.*, 155(10):1135–1141, Oct. 2019.

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

[39] S. Foucart and H. Rauhut. *A Mathematical Introduction to Compressive Sensing*. Springer, New York, 2012.

[40] M. Friendly. A brief history of data visualization. In C. Chen, W. Härdle, and A. Unwin, editors, *Handbook of Computational Statistics: Data Visualization*, volume III. Springer-Verlag, 2006.

[41] R. G. Gallager. *Stochastic Processes: Theory for Applications*. Cambridge University Press, 2013.

[42] A. E. Gamal and Y.-H. Kim. *Network Information Theory*. Cambridge Univ. Press, 2012.

[43] M. Gao, H. Igata, A. Takeuchi, K. Sato, and Y. Ikegaya. Machine learning-based prediction of adverse drug effects: An example of seizure-inducing compounds. *Journal of Pharmacological Sciences*, 133(2):70 – 78, 2017.

[44] W. Gautschi and G. Inglese. Lower bounds for the condition number of vandermonde matrices. *Numer. Math.*, 52:241 – 250, 1988.

[45] G. Golub and C. van Loan. An analysis of the total least squares problem. *SIAM J. Numerical Analysis*, 17(6):883–893, Dec. 1980.

[46] G. H. Golub and C. F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, Baltimore, MD, 3rd edition, 1996.

[47] C. Gomez-Uribe and N. Hunt. The netflix recommender system: Algorithms, business value, and innovation. *Association for Computing Machinery*, 6(4), January 2016.

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

[49] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Proc. Neural Inf. Proc. Syst. (NIPS)*, 2014.

[50] R. Gray. *Probability, Random Processes, and Ergodic Properties*. Springer, New York, 2 edition, 2009.

[51] R. Gray, J. Kieffer, and Y. Linde. Locally optimal block quantizer design. *Information and Control*, 45:178 – 198, 1980.

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

[53] P. Hacker, R. Krestel, S. Grundmann, and F. Naumann. Explainable AI under contract and tort law: legal incentives and technical challenges. *Artificial Intelligence and Law*, 2020.

[54] H. Hagras. Toward human-understandable, explainable ai. *Computer*, 51(9):28–36, Sep. 2018.

[55] A. Halevy, P. Norvig, and F. Pereira. The unreasonable effectiveness of data. *IEEE Intelligent Systems*, March/April 2009.

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

[57] P. R. Halmos. *Measure Theory*. Springer, New York, 1974.

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

[59] T. Hastie, R. Tibshirani, and M. Wainwright. *Statistical Learning with Sparsity. The Lasso and its Generalizations*. CRC Press, 2015.

[60] E. Hazan. *Introduction to Online Convex Optimization*. Now Publishers Inc., 2016.

[61] J. Howard and S. Ruder. Universal language model fine-tuning for text classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 328–339, Melbourne, Australia, July 2018. Association for Computational Linguistics.

[62] P. Huber. Approximate models. In C. Huber-Carol, N. Balakrishnan, M. Nikulin, and M. Mesbah, editors, *Goodness-of-Fit Tests and Model Validity. Statistics for Industry and Technology*. Birkhäuser, Boston, MA, 2002.

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

[64] L. Hyafil and R. Rivest. Constructing optimal binary decision trees is np-complete. *Information Processing Letters*, 5(1):15–17, 1976.

[65] L. Hyafil and R. L. Rivest. Constructing optimal binary decision trees is np-complete. *Information Processing Letters*, 5(1):15–17, 1976.

[66] G. James, D. Witten, T. Hastie, and R. Tibshirani. *An Introduction to Statistical Learning with Applications in R*. Springer, 2013.

[67] K. Jeong, J. Choi, and G. Jang. Semi-local structure patterns for robust face detection. *IEEE Sig. Proc. Letters*, 22(9), 2015.

[68] A. Jung. A fixed-point of view on gradient methods for big data. *Frontiers in Applied Mathematics and Statistics*, 3, 2017.

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

[70] A. Jung. Networked exponential families for big data over networks. *IEEE Access*, 8:202897–202909, 2020.

[71] A. Jung, Y. Eldar, and N. Görtz. On the minimax risk of dictionary learning. *IEEE Trans. Inf. Theory*, 62(3):1501 – 1515, Mar. 2016.

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

[73] A. Jung and Y. SarcheshmehPour. Local graph clustering with network lasso. *IEEE Signal Processing Letters*, 28:106–110, 2021.

[74] A. Jung and N. Tran. Localized linear regression in networked data. *IEEE Sig. Proc. Lett.*, 26(7), Jul. 2019.

[75] J. Kagan. Motives and development. *Journal of Personality and Social Psychology*, 22(1):51–66, 1972.

[76] S. M. Kay. *Fundamentals of Statistical Signal Processing: Estimation Theory*. Prentice Hall, Englewood Cliffs, NJ, 1993.

[77] T. Kibble and F. Berkshire. *Classical Mechanics*. Imperical College Press, 5 edition, 2011.

[78] P. Koehn. Europarl: A parallel corpus for statistical machine translation. In *The 10th Machine Translation Summit, page 79–86., AAMT,*, Phuket, Thailand, 2005.

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

[80] A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. In *Neural Information Processing Systems, NIPS*, 2012.

[81] B. Kulis and M. I. Jordan. Revisiting k-means: New algorithms via bayesian nonparametrics. In *Proc. of the 29th International Conference on Machine Learning, ICML 2012, Edinburgh, Scotland, UK, June 26 - July 1, 2012*. icml.cc / Omnipress, 2012.

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

[83] J. Larsen and C. Goutte. On optimal data split for generalization estimation and model selection. In *IEEE Workshop on Neural Networks for Signal Process*, 1999.

[84] S. L. Lauritzen. *Graphical Models*. Clarendon Press, Oxford, UK, 1996.

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

[86] L. Li, W. Chu, J. Langford, and R. Schapire. A contextual-bandit approach to personalized news article recommendation. In *Proc. International World Wide Web Conference*, pages 661–670, Raleigh, North Carolina, USA, April 2010.

[87] Q. V. Liao, D. Gruen, and S. Miller. Questioning the ai: Informing design practices for explainable ai user experiences. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, CHI '20, pages 1–15, New York, NY, USA, 2020. Association for Computing Machinery.

[88] H. Lütkepohl. *New Introduction to Multiple Time Series Analysis*. Springer, New York, 2005.

[89] A. Makhdoumi, S. Salamatian, N. Fawaz, and M. Médard. From the information bottleneck to the privacy funnel. In *2014 IEEE Information Theory Workshop (ITW 2014)*, pages 501–505, 2014.

[90] S. G. Mallat. *A Wavelet Tour of Signal Processing – The Sparse Way*. Academic Press, San Diego, CA, 3 edition, 2009.

[91] K. V. Mardia, J. T. Kent, and J. M. Bibby. *Multivariate Analysis*. Academic Press, 1979.

[92] J. McInerney, B. Lacker, S. Hansen, K. Higley, H. Bouchard, A. Gruson, and R. Mehrotra. Explore, exploit, and explain: personalizing explainable recommendations with bandits. In *Proceedings of the 12th ACM Conference on Recommender Systems*, 2018.

[93] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas. Communication-efficient learning of deep networks from decentralized data. In A. Singh and J. Zhu, editors, *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54 of *Proceedings of Machine Learning Research*, pages 1273–1282, Fort Lauderdale, FL, USA, 20–22 Apr 2017. PMLR.

[94] C. Meyer. Generalized inversion of modified matrices. *SIAM J. Appied Mathmetmatics*, 24(3), 1973.

[95] C. Millard, editor. *Cloud Computing Law*. Oxford University Press, 2 edition, May 2021.

[96] T. Mitchell. The need for biases in learning generalizations. Technical Report CBM-TR 5-110,, Rutgers University, New Brunswick, New Jersey, USA, 1980.

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

[98] G. Montavon, W. Samek, and K. Müller. Methods for interpreting and understanding deep neural networks. *Digital Signal Processing*, 73:1–15, 2018.

[99] K. Mortensen and T. Hughes. Comparing amazon's mechanical turk platform to conventional data collection methods in the health and medical research literature. *J. Gen. Intern Med.*, 33(4):533–538, 2018.

[100] R. Muirhead. *Aspects of Multivariate Statistical Theory*. John Wiley & Sons Inc., 1982.

[101] N. Murata. A statistical study on on-line learning. In D. Saad, editor, *On-line Learning in Neural Networks*, pages 63–92. Cambridge University Press, New York, NY, USA, 1998.

[102] Y. Nesterov. *Introductory lectures on convex optimization*, volume 87 of *Applied Optimization*. Kluwer Academic Publishers, Boston, MA, 2004. A basic course.

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

[104] A. Ng. *Shaping and Policy search in Reinforcement Learning*. PhD thesis, University of California, Berkeley, 2003.

[105] A. Y. Ng and M. I. Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14*, pages 841–848. MIT Press, 2002.

[106] A. Y. Ng, M. I. Jordan, and Y. Weiss. On spectral clustering: Analysis and an algorithm. In *Adv. Neur. Inf. Proc. Syst.*, 2001.

[107] S. Oymak, B. Recht, and M. Soltanolkotabi. Sharp time–data tradeoffs for linear inverse problems. *IEEE Transactions on Information Theory*, 64(6):4129–4158, June 2018.

[108] S. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, 2010.

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

[110] N. Parikh and S. Boyd. Proximal algorithms. *Foundations and Trends in Optimization*, 1(3):123–231, 2013.

[111] J. Pearl. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann, 1988.

[112] F. Pedregosa. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12(85):2825–2830, 2011.

[113] L. Pitt and L. G. Valiant. Computational limitations on learning from examples. *J. ACM*, 35(4):965–984, Oct. 1988.

[114] T. Poggio, H. Mhaskar, L. Rosasco, B. Miranda, and Q. Liao. Why and when can deep-but not shallow-networks avoid the curse of dimensionality: A review. *International Journal of Automation and Computing*, 14(5):503–519, 2017.

[115] H. Poor. *An Introduction to Signal Detection and Estimation*. Springer, 2 edition, 1994.

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

[117] M. Ribeiro, S. Singh, and C. Guestrin. Anchors: High-precision model-agnostic explanations. In *Proc. AAAI Conference on Artificial Intelligence (AAAI)*, 2018.

[118] S. Roweis. EM Algorithms for PCA and SPCA. In *Advances in Neural Information Processing Systems*, pages 626–632. MIT Press, 1998.

[119] W. Rudin. *Principles of Mathematical Analysis*. McGraw-Hill, New York, 3 edition, 1976.

[120] W. Rudin. *Real and Complex Analysis*. McGraw-Hill, New York, 3rd edition, 1987.

[121] S. J. Russel and P. Norvig. *Artificial Intelligence - A Modern Approach*. Prentice Hall, New York, 3 edition, 2010.

[122] N. P. Santhanam and M. J. Wainwright. Information-theoretic limits of selecting binary graphical models in high dimensions. *IEEE Trans. Inf. Theory*, 58(7):4117–4134, Jul. 2012.

[123] 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.

[124] 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.

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

[126] S. Shalev-Shwartz and S. Ben-David. *Understanding Machine Learning – from Theory to Algorithms*. Cambridge University Press, 2014.

[127] C. E. Shannon. Communication in the presence of noise. 1948.

[128] Y. Y. Shkel, R. S. Blum, and H. V. Poor. Secrecy by design with applications to privacy and compression. *IEEE Transactions on Information Theory*, 67(2):824–843, 2021.

[129] S.Levine, C. Finn, T. Darrell, and P.Abbeel. End-to-end training of deep visuomotor policies. *J. Mach. Learn. Res.*, 17, 2016.

[130] V. Smith, C.-K. Chiang, M. Sanjabi, and A. Talwalkar. Federated multi-task learning. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

[131] S. Smoliski and K. Radtke. Spatial prediction of demersal fish diversity in the baltic sea: comparison of machine learning and regression-based techniques. *ICES Journal of Marine Science*, 74(1):102–111, 2017.

[132] A. Sorokin and D. Forsyth. Utility data annotation with amazon mechanical turk. In *2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, pages 1–8, 2008.

[133] S. Sra, S. Nowozin, and S. J. Wright, editors. *Optimization for Machine Learning*. MIT Press, 2012.

[134] G. Strang. *Linear Algebra and its Applications*. Cengage Learning, 4 edition, 2006.

[135] G. Strang. *Computational Science and Engineering*. Wellesley-Cambridge Press, MA, 2007.

[136] G. Strang. *Introduction to Linear Algebra*. Wellesley-Cambridge Press, MA, 5 edition, 2016.

[137] R. Sutton and A. Barto. *Reinforcement learning: An introduction*. MIT press, Cambridge, MA, 2 edition, 2018.

[138] M. E. Tipping and C. Bishop. Probabilistic principal component analysis. *Journal of the Royal Statistical Society, Series B*, 21/3:611–622, January 1999.

[139] M. E. Tipping and C. M. Bishop. Probabilistic principal component analysis. *J. Roy. Stat. Soc. Ser. B*, 61:611–622, 1999.

[140] N. Tishby and N. Zaslavsky. Deep learning and the information bottleneck principle. In *2015 IEEE Information Theory Workshop (ITW)*, pages 1–5, 2015.

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

[142] N. Tran, H. Ambos, and A. Jung. Classifying partially labeled networked data via logistic network lasso. In *Proc. IEEE Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3832–3836, Barcelona, Spain, May 2020.

[143] L. G. Valiant. A theory of the learnable. In *Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing*, STOC '84, pages 436–445, New York, NY, USA, 1984. Association for Computing Machinery.

[144] S. A. van de Geer and P. Bühlmann. On the conditions used to prove oracle results for the Lasso. *Electron. J. Statist.*, 3:1360 – 1392, 2009.

[145] V. N. Vapnik. *The Nature of Statistical Learning Theory*. Springer, 1999.

[146] O. Vasicek. A test for normality based on sample entropy. *Journal of the Royal Statistical Society. Series B (Methodological)*, 38(1):54–59, 1976.

[147] R. Vidal. Subspace clustering. *IEEE Signal Processing Magazine*, 52, March 2011.

[148] U. von Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4):395–416, Dec. 2007.

[149] S. Wachter. Data protection in the age of big data. *Nature Electronics*, 2(1):6–7, 2019.

[150] S. Wade and Z. Ghahramani. Bayesian Cluster Analysis: Point Estimation and Credible Balls (with Discussion). *Bayesian Analysis*, 13(2):559 – 626, 2018.

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

[152] M. J. Wainwright. Information-theoretic limits on sparsity recovery in the high-dimensional and noisy setting. *IEEE Trans. Inf. Theory*, 55(12):5728–5741, Dec. 2009.

[153] M. J. Wainwright and M. I. Jordan. *Graphical Models, Exponential Families, and Variational Inference*, volume 1 of *Foundations and Trends in Machine Learning*. Now Publishers, Hanover, MA, 2008.

[154] A. Wang. An industrial-strength audio search algorithm. In *International Symposium on Music Information Retrieval*, Baltimore, MD, 2003.

[155] W. Wang, M. J. Wainwright, and K. Ramchandran. Information-theoretic bounds on model selection for Gaussian Markov random fields. In *Proc. IEEE ISIT-2010*, pages 1373–1377, Austin, TX, Jun. 2010.

[156] J. Wright, Y. Peng, Y. Ma, A. Ganesh, and S. Rao. Robust principal component analysis: Exact recovery of corrupted low-rank matrices by convex optimization. In *Neural Information Processing Systems, NIPS 2009*, 2009.

[157] L. Xu and M. Jordan. On convergence properties of the EM algorithm for Gaussian mixtures. *Neural Computation*, 8(1):129–151, 1996.

[158] X. Yang and Q. Wang. Crowd hybrid model for pedestrian dynamic prediction in a corridor. *IEEE Access*, 7, 2019.

[159] K. Young. Bayesian diagnostics for checking assumptions of normality. *Journal of Statistical Computation and Simulation*, 47(3–4):167 – 180, 1993.